

**Oracle8i**

アプリケーション開発者ガイド アドバンスト・キューイング

リリース 8.1

2000 年 2 月

**ORACLE®**

原本名: Application Developer's Guide - Advanced Queuing, Release 2 (8.1.6)

原本著者: Kevin MacDowell, Den Raphaely

原本協力者: Neerja Bhat, Shelley Higgins, Krishnan Meiyyappan, Bhagat Nainani, James Rawles, Sashi Chandrasekaran, Dieter Gawlick, Mohan Kamath, Goran Olsson, Madhu Reddy, Mary Rhodes, Ashok Saxena, Ekrem Soylemez, Alvin To, Rahim Yaseen

Copyright © 1996, 1999, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム (ソフトウェアおよびドキュメントを含む) の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されています。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

\* オラクル社とは、Oracle Corporation (米国オラクル) または日本オラクル株式会社 (日本オラクル) を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation (米国オラクル) およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Notice が適用されます。

#### Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

---

# 目次

はじめに .....	xxxvii
このマニュアルについて .....	xxxviii
Oracle8i で導入された新機能 .....	xxxviii
関連マニュアル .....	xxxix
このマニュアルの構成 .....	xl
図の解釈方法 .....	xl
ユースケース図 .....	xl
状態図 .....	xli
このマニュアルの表記規則 .....	li
<b>1 Oracle アドバンスト・キューイングの概要</b>	
<b>キューイング: メッセージ操作の最適な手段</b> .....	1-2
ビジネス・イベントの核心としてのメッセージ .....	1-2
メッセージ交換の調整に対するキューの使用 .....	1-2
キューイング・システムによる効果 .....	1-3
<b>キューイング・アプリケーション開発のコンテキスト</b> .....	1-4
<b>キューイング・メッセージのモデル</b> .....	1-5
Point-to-Point モデル .....	1-5
パブリッシュ / サブスクライブ・モデル .....	1-6
<b>アドバンスト・キューイング (AQ) の機能</b> .....	1-7
一般機能 .....	1-7
ENQUEUE 機能 .....	1-10
DEQUEUE 機能 .....	1-13
伝播機能 .....	1-15
<b>アドバンスト・キューイング (ネイティブ AQ) の要素</b> .....	1-17

メッセージ .....	1-17
キュー .....	1-17
キュー・テーブル .....	1-17
エージェント .....	1-17
受信者 .....	1-18
受信者およびサブスクリプション・リスト .....	1-18
ルール .....	1-19
ルールベースのサブスクライバ .....	1-19
キュー・モニター .....	1-19
デモの参照 .....	1-20

## 2 基本的なコンポーネント

データ構造 .....	2-2
オブジェクト名 .....	2-2
型名 .....	2-2
エージェント .....	2-3
AQ 受信者リスト型 .....	2-4
AQ エージェント・リスト型 .....	2-4
AQ サブスクライバ・リスト型 .....	2-4
管理インタフェースの列挙定数 .....	2-5
操作インタフェースの列挙定数 .....	2-6
問題および考慮点 .....	2-7
INIT.ORA パラメータ .....	2-7
Java コンポーネント - oracle.AQ .....	2-8
Java AQ クラスの場所 .....	2-8

## 3 AQ プログラム環境

AQ にアクセスするためのプログラム環境 .....	3-2
PL/SQL (DBMS_AQADM および DBMS_AQ パッケージ) を使用した AQ へのアクセス .....	3-4
Visual Basic (OO4O) を使用した AQ へのアクセス .....	3-6
詳細情報 .....	3-6
OCI を使用した AQ へのアクセス .....	3-7
例 .....	3-7
AQ Java (oracle.AQ) クラスを使用した AQ へのアクセス .....	3-8
Java AQ クラスへのアクセス .....	3-8

アドバンスト・キューイングの例 .....	3-8
Java AQ API の管理 .....	3-9
<b>Oracle Java Message Service (JMS) を使用した AQ へのアクセス .....</b>	<b>3-10</b>
標準 JMS 機能 .....	3-10
Oracle JMS 拡張機能 .....	3-10
標準 JMS および Oracle JMS へのアクセス .....	3-11
詳細情報 .....	3-12
<b>AQ プログラム環境の比較 .....</b>	<b>3-13</b>
<b>AQ 管理インタフェース .....</b>	<b>3-13</b>
<b>AQ 操作インタフェース .....</b>	<b>3-16</b>

## 4 AQ の管理

<b>キュー・テーブルの移行（インポート / エクスポート） .....</b>	<b>4-2</b>
キュー・テーブル・データのエクスポート .....	4-2
キュー・テーブル・データのインポート .....	4-3
<b>セキュリティ .....</b>	<b>4-4</b>
8.0 および 8.1 互換のキューのセキュリティ .....	4-4
権限およびアクセス制御 .....	4-5
ロール .....	4-6
管理者ロール .....	4-6
ユーザー・ロール .....	4-7
AQ オブジェクト型へのアクセス .....	4-7
OCI アプリケーション .....	4-7
伝播 .....	4-7
使用上の注意 .....	4-8
例：8.0 互換のキュー・テーブルを 8.1 互換のキュー・テーブルにアップグレードする方法 .....	4-8
<b>Enterprise Manager サポート .....</b>	<b>4-9</b>
<b>プロトコル .....</b>	<b>4-9</b>
<b>AQ 操作を準備するための DBA のアクション例 .....</b>	<b>4-9</b>
<b>現在の制限事項 .....</b>	<b>4-11</b>
DBMS_AQADM パッケージの自動コミット機能 .....	4-11
メッセージ・ペイロード（実際に通信される情報）内のコレクション型 .....	4-11
AQ の Java API におけるオブジェクト型ペイロード（実際に通信される情報）のサポート .....	4-11
キュー・テーブルおよびキューにおけるシノニム .....	4-11
8.0 互換の複数コンシューマ・キューではトランスポータブル表領域が無効 .....	4-12

表領域の Point-in-Time リカバリ .....	4-12
オブジェクト・キューからの伝播 .....	4-12
非永続キュー .....	4-12
<b>互換性</b> .....	4-13

## 5 高度なトピック

<b>パフォーマンス</b> .....	5-2
表および索引構造 .....	5-2
スループット .....	5-2
可用性 .....	5-2
<b>スケーラビリティ</b> .....	5-3
<b>伝播の問題点</b> .....	5-3
<b>AQ 伝播問題のデバッグについてのガイドライン</b> .....	5-5

## 6 FAQ

## 7 モデリングおよび設計

<b>キュー・エンティティのモデリング</b> .....	7-2
基本キューイング .....	7-3
基本キューイングの説明 .....	7-3
AQ を使用したクライアント / サーバー通信の例 .....	7-5
複数コンシューマによる同一メッセージのデキュー .....	7-6
複数コンシューマによる同一メッセージのデキューの例 .....	7-7
指定された受信者による指定されたメッセージのデキュー例 .....	7-9
AQ を使用したワークフローの実装例 .....	7-11
AQ を使用したパブリッシュ / サブスクライブの実装例 .....	7-13
メッセージの伝播 .....	7-15
メッセージの伝播の例 .....	7-17

## 8 AQ を使用したサンプル・アプリケーション

<b>サンプル・アプリケーション</b> .....	8-3
<b>一般機能</b> .....	8-4
システム・レベルのアクセス制御 .....	8-5
構造化ペイロード .....	8-7

キュー・レベルのアクセス制御 .....	8-10
非永続キュー .....	8-12
保存およびメッセージ履歴 .....	8-23
パブリッシュ / サブスクライブ・サポート .....	8-25
Oracle Parallel Server (OPS) のサポート .....	8-28
統計ビューのサポート .....	8-33
<b>ENQUEUE 機能</b> .....	8-34
サブスクリプションおよび受信者リスト .....	8-35
メッセージの優先順位および順序付け .....	8-37
時間指定 : 遅延 .....	8-45
時間指定 : 期限切れ .....	8-48
メッセージのグループ化 .....	8-51
<b>DEQUEUE 機能</b> .....	8-54
デキューの方法 .....	8-55
複数の受信者 .....	8-60
ローカルおよびリモートの受信者 .....	8-62
デキューにおけるメッセージ・ナビゲーション .....	8-64
デキューのモード .....	8-68
メッセージ到着待機の最適化 .....	8-74
非同期通知 .....	8-76
遅延間隔をのりでの再試行 .....	8-83
例外処理 .....	8-87
ルールベースのサブスクリプション .....	8-93
Listen 機能 .....	8-97
<b>伝播機能</b> .....	8-101
伝播 .....	8-102
伝播スケジュール .....	8-103
LOB 属性を伴うメッセージの伝播 .....	8-107
拡張伝播スケジュール機能 .....	8-109
伝播中の例外処理 .....	8-112

## 9 管理インタフェース

ユースケース・モデル : 管理インタフェース - 基本操作 .....	9-2
キュー・テーブルの作成 .....	9-5
用途 .....	9-6

使用上の注意 .....	9-6
構文 .....	9-7
例 .....	9-8
PL/SQL (DBMS_AQADM パッケージ) : キュー・テーブルの作成 .....	9-8
VB (OO4O) : キュー・テーブルの作成 .....	9-9
Java (JDBC) : キュー・テーブルの作成 .....	9-10
<b>キュー・テーブルの作成 (STORAGE 句の設定) .....</b>	<b>9-13</b>
<b>キュー・テーブルの変更 .....</b>	<b>9-14</b>
用途 .....	9-14
使用上の注意 .....	9-14
構文 .....	9-15
例 .....	9-15
PL/SQL (DBMS_AQADM パッケージ) : キュー・テーブルの変更 .....	9-15
Java (JDBC) : キュー・テーブルの変更 .....	9-16
<b>キュー・テーブルの削除 .....</b>	<b>9-17</b>
用途 .....	9-18
使用上の注意 .....	9-18
構文 .....	9-18
例 .....	9-18
PL/SQL (DBMS_AQADM パッケージ) : キュー・テーブルの削除 .....	9-18
Java (JDBC) : キュー・テーブルの削除 .....	9-19
<b>キューの作成 .....</b>	<b>9-20</b>
用途 .....	9-21
使用上の注意 .....	9-22
構文 .....	9-22
例 .....	9-22
PL/SQL (DBMS_AQADM) : キューの作成 .....	9-23
Java (JDBC) : キューの作成 .....	9-25
<b>非永続キューの作成 .....</b>	<b>9-27</b>
用途 .....	9-27
使用上の注意 .....	9-28
構文 .....	9-28
例 .....	9-28
PL/SQL (DBMS_AQADM) : 非永続キューの作成 .....	9-29
Java (JDBC) : 非永続キューの作成 .....	9-29



<b>キューの変更</b> .....	9-30
用途 .....	9-31
使用上の注意 .....	9-31
構文 .....	9-31
例 .....	9-31
PL/SQL (DBMS_AQADM) : キューの変更 .....	9-32
Java (JDBC) : キューの変更 .....	9-32
<b>キューの削除</b> .....	9-33
用途 .....	9-34
使用上の注意 .....	9-34
構文 .....	9-34
例 .....	9-34
PL/SQL (DBMS_AQADM) : キューの削除 .....	9-35
Java (JDBC) : キューの削除 .....	9-35
<b>キューの開始</b> .....	9-36
用途 .....	9-36
使用上の注意 .....	9-37
構文 .....	9-37
例 .....	9-37
PL/SQL (DBMS_AQADM パッケージ) : キューの開始 .....	9-37
Java (JDBC) : キューの開始 .....	9-38
<b>キューの停止</b> .....	9-39
用途 .....	9-40
使用上の注意 .....	9-40
構文 .....	9-40
例 .....	9-40
PL/SQL (DBMS_AQADM) : キューの停止 .....	9-41
Java (JDBC) : キューの停止 .....	9-41
<b>システム権限の付与</b> .....	9-42
用途 .....	9-43
使用上の注意 .....	9-43
例 .....	9-43
PL/SQL (DBMS_AQADM) : システム権限の付与 .....	9-44
Java (JDBC) : システム権限の付与 .....	9-44
<b>システム権限の取消し</b> .....	9-45

用途 .....	9-45
使用上の注意 .....	9-46
構文 .....	9-46
例 .....	9-46
PL/SQL (DBMS_AQADM) の使用 : システム権限の取消し .....	9-46
<b>キュー権限の付与 .....</b>	<b>9-47</b>
用途 .....	9-47
使用上の注意 .....	9-48
構文 .....	9-48
例 .....	9-48
PL/SQL (DBMS_AQADM) : キュー権限の付与 .....	9-48
Java (JDBC) : キュー権限の付与 .....	9-49
<b>キュー権限の取消し .....</b>	<b>9-50</b>
用途 .....	9-51
使用上の注意 .....	9-51
構文 .....	9-51
例 .....	9-51
PL/SQL (DBMS_AQADM) : キュー権限の取消し .....	9-52
Java (JDBC) : キュー権限の取消し .....	9-52
<b>サブスクライバの追加 .....</b>	<b>9-53</b>
用途 .....	9-54
使用上の注意 .....	9-54
構文 .....	9-54
例 .....	9-54
PL/SQL (DBMS_AQADM) : サブスクライバの追加 .....	9-55
PL/SQL (DBMS_AQADM) : ルールベースのサブスクライバの追加 .....	9-55
Java (JDBC) : サブスクライバの追加 .....	9-56
<b>サブスクライバの変更 .....</b>	<b>9-58</b>
用途 .....	9-59
使用上の注意 .....	9-59
構文 .....	9-59
例 .....	9-59
PL/SQL (DBMS_AQADM) : サブスクライバの変更 .....	9-60
Java (JDBC) : サブスクライバの変更 .....	9-61
<b>サブスクライバの削除 .....</b>	<b>9-62</b>

用途 .....	9-63
使用上の注意 .....	9-63
構文 .....	9-63
例 .....	9-63
PL/SQL (DBMS_AQADM) : サブスクライバの削除 .....	9-64
Java (JDBC) : サブスクライバの削除 .....	9-64
<b>キューの伝播のスケジュール .....</b>	<b>9-65</b>
用途 .....	9-66
使用上の注意 .....	9-66
構文 .....	9-66
例 .....	9-66
PL/SQL (DBMS_AQADM) : キューの伝播のスケジュール .....	9-67
Java (JDBC) : キューの伝播のスケジュール .....	9-67
<b>キューの伝播スケジュールの解除 .....</b>	<b>9-69</b>
用途 .....	9-69
使用上の注意 .....	9-69
構文 .....	9-69
例 .....	9-70
PL/SQL (DBMS_AQADM) : 伝播スケジュールの解除 .....	9-70
Java (JDBC) : キューの伝播スケジュールの解除 .....	9-70
<b>キュー・タイプの検証 .....</b>	<b>9-72</b>
用途 .....	9-72
使用上の注意 .....	9-72
構文 .....	9-72
例 .....	9-73
PL/SQL (DBMS_AQADM) : キュー・タイプの検証 .....	9-73
Java (JDBC) : キュー・タイプの検証 .....	9-74
<b>伝播スケジュールの変更 .....</b>	<b>9-75</b>
用途 .....	9-76
使用上の注意 .....	9-76
構文 .....	9-76
例 .....	9-76
PL/SQL (DBMS_AQADM) : 伝播スケジュールの変更 .....	9-77
Java (JDBC) : 伝播スケジュールの変更 .....	9-77
<b>伝播スケジュールの使用可能化 .....</b>	<b>9-79</b>

用途 .....	9-79
使用上の注意 .....	9-79
構文 .....	9-79
例 .....	9-80
PL/SQL (DBMS_AQADM) : 伝播の使用可能化 .....	9-80
Java (JDBC) : 伝播スケジュールの使用可能化 .....	9-80
<b>伝播スケジュールの使用不可 .....</b>	<b>9-82</b>
用途 .....	9-82
使用上の注意 .....	9-82
構文 .....	9-82
例 .....	9-83
PL/SQL (DBMS_AQADM) : 伝播の使用不可 .....	9-83
Java (JDBC) : 伝播スケジュールの使用不可 .....	9-84

## 10 管理インタフェース：ビュー

ユースケース・モデル: 管理インタフェース - ビュー .....	10-2
データベース内のすべてのキュー・テーブルの選択 .....	10-4
ユーザーのキュー・テーブルの選択 .....	10-7
データベース内のすべてのキューの選択 .....	10-10
すべての伝播スケジュールの選択 .....	10-12
ユーザーが何らかの権限を持っているキューの選択 .....	10-17
ユーザーがキュー権限を持っているキューの選択 .....	10-19
キュー・テーブルのメッセージの選択 .....	10-21
ユーザー・スキーマのキュー・テーブルの選択 .....	10-25
ユーザー・スキーマのキューの選択 .....	10-28
ユーザー・スキーマの伝播スケジュールの選択 .....	10-30
キューのサブスクライバの選択 .....	10-35
キューのサブスクライバおよびそのルール of 選択 .....	10-37
データベース全体における状態ごとのメッセージ数の選択 .....	10-39
特定のインスタンスにおける状態ごとのメッセージ数の選択 .....	10-41

## 11 操作インターフェース：基本操作

ユースケース・モデル: 操作インタフェース - 基本操作 .....	11-2
メッセージのエンキュー .....	11-5
用途 .....	11-5
使用上の注意 .....	11-6

構文 .....	11-6
例 .....	11-6
メッセージのエンキュー（オプションの指定） .....	11-7
用途 .....	11-8
使用上の注意 .....	11-8
構文 .....	11-8
例 .....	11-9
メッセージのエンキュー（メッセージ・プロパティの指定） .....	11-10
用途 .....	11-11
使用上の注意 .....	11-11
構文 .....	11-11
例 .....	11-12
メッセージのエンキュー（メッセージ・プロパティの指定（送信者 ID の指定）） .....	11-13
用途 .....	11-13
使用上の注意 .....	11-13
構文 .....	11-14
例 .....	11-14
メッセージのエンキュー（ペイロードの追加） .....	11-15
用途 .....	11-15
使用上の注意 .....	11-15
構文 .....	11-16
例 .....	11-16
PL/SQL（DBMS_AQ パッケージ）：オブジェクト型メッセージのエンキュー .....	11-17
Java（JDBC）：メッセージのエンキュー（ペイロードの追加） .....	11-19
Visual Basic（OO4O）：メッセージのエンキュー .....	11-21
1 個（複数個）のキューのリスニング .....	11-23
用途 .....	11-23
使用上の注意 .....	11-23
構文 .....	11-24
例 .....	11-24
1 個（複数個）の単一コンシューマ・キューのリスニング .....	11-25
使用上の注意 .....	11-25
構文 .....	11-26
例 .....	11-26
PL/SQL（DBMS_AQ パッケージ）：キューのリスニング .....	11-26

C (OCI) : 単一コンシューマ・キューのリスニング .....	11-27
<b>1 個 (複数個) の複数コンシューマ・キューのリスニング .....</b>	<b>11-36</b>
使用上の注意 .....	11-37
構文 .....	11-37
例 .....	11-37
PL/SQL (DBMS_AQ パッケージ) : キューのリスニング .....	11-38
C (OCI) : 複数コンシューマ・キューのリスニング .....	11-39
<b>メッセージのデキュー .....</b>	<b>11-45</b>
用途 .....	11-45
使用上の注意 .....	11-46
構文 .....	11-47
例 .....	11-47
<b>単一コンシューマ・キューからのメッセージのデキュー (オプションの指定) .....</b>	<b>11-48</b>
用途 .....	11-49
使用上の注意 .....	11-49
構文 .....	11-49
例 .....	11-49
PL/SQL (DBMS_AQ パッケージ) : オブジェクト型メッセージのデキュー .....	11-50
Java (JDBC) : 単一コンシューマ・キューからのメッセージのデキュー (オプションの指定) .	11-50
Visual Basic (OO4O) : メッセージのデキュー .....	11-51
<b>複数コンシューマ・キューからのメッセージのデキュー (オプションの指定) .....</b>	<b>11-53</b>
用途 .....	11-54
使用上の注意 .....	11-54
構文 .....	11-54
例 .....	11-54
Java (JDBC) : 単一コンシューマ・キューからのメッセージのデキュー (オプションの指定) .	11-55
<b>通知登録 .....</b>	<b>11-56</b>
用途 .....	11-57
使用上の注意 .....	11-57
構文 .....	11-58
例 .....	11-58
<b>通知登録 (サブスクリプション名の指定 - 単一コンシューマ・キュー) .....</b>	<b>11-59</b>
<b>通知登録 (サブスクリプション名の指定 - 複数コンシューマ・キュー) .....</b>	<b>11-60</b>
使用上の注意 .....	11-60
構文 .....	11-61

例 .....	11-61
C (OCI) : 単一コンシューマおよび複数コンシューマ・キューへの通知登録 .....	11-61

## 12 JMS を使用したアプリケーションの作成

サンプル・アプリケーション .....	12-4
一般的な機能 .....	12-5
JMS 接続およびセッション .....	12-6
JMS 宛先 - キューおよびトピック .....	12-11
システム・レベルのアクセス制御 .....	12-13
宛先レベルのアクセス制御 .....	12-15
保存およびメッセージ履歴 .....	12-16
Oracle Parallel Server のサポート .....	12-17
統計ビューのサポート .....	12-19
構造化ペイロード / メッセージの型 .....	12-20
JMS サンプルで使したペイロード .....	12-31
Point-to-Point モデル機能 .....	12-38
キュー .....	12-39
キュー送信者 .....	12-40
キュー受信者 .....	12-40
キュー・ブラウザ .....	12-43
パブリッシュ / サブスクライブ・モデル機能 .....	12-45
トピック .....	12-46
永続サブスクライバ .....	12-48
トピック・パブリッシャ .....	12-51
受信者リスト .....	12-53
トピック受信者 .....	12-54
メッセージ・プロデューサ機能 .....	12-57
メッセージの優先順位および順序付け .....	12-58
時刻指定 : 遅延 .....	12-62
時刻指定 : 期限切れ .....	12-64
メッセージのグループ化 .....	12-66
メッセージ・コンシューマ機能 .....	12-70
メッセージの受信 .....	12-71
受信におけるメッセージのナビゲーション .....	12-74
メッセージ受信モード .....	12-77

遅延間隔をおいての再試行 .....	12-79
メッセージ・リスナーを使用したメッセージの非同期受信 .....	12-81
AQ の例外処理 .....	12-85
<b>伝播</b> .....	12-89
リモート・サブスクライバ .....	12-90
伝播スケジュール .....	12-95
拡張伝播スケジュール機能 .....	12-97
伝播中の例外処理 .....	12-99

## 13 JMS 管理インタフェース : 基本操作

ユースケース・モデル: JMS 管理インタフェース - 基本操作 .....	13-2
ユースケース図: JMS 管理インタフェース - 基本操作 .....	13-4
Point-to-Point: キュー・コネクション・ファクトリを作成する 2 つの方法 .....	13-5
JDBC URL でのキュー・コネクション・ファクトリの取得 .....	13-6
用途 .....	13-6
使用上の注意 .....	13-6
構文 .....	13-7
例 .....	13-7
JDBC 接続パラメータでのキュー・コネクション・ファクトリの取得 .....	13-8
用途 .....	13-9
使用上の注意 .....	13-9
構文 .....	13-9
例 .....	13-9
パブリッシュ/サブスクライブ: トピック・コネクション・ファクトリを作成する 2 つの方法 .....	13-10
JDBC URL でのトピック・コネクション・ファクトリの取得 .....	13-11
用途 .....	13-11
使用上の注意 .....	13-11
構文 .....	13-12
例 .....	13-12
JDBC 接続パラメータでのトピック・コネクション・ファクトリの取得 .....	13-13
使用上の注意 .....	13-14
用途 .....	13-14
構文 .....	13-14
例 .....	13-14
キュー・テーブルの作成 .....	13-15



用途 .....	13-15
使用上の注意 .....	13-16
構文 .....	13-16
例 .....	13-16
<b>キュー・テーブルの作成（キュー・テーブルのプロパティの指定） .....</b>	<b>13-17</b>
用途 .....	13-17
使用上の注意 .....	13-17
構文 .....	13-17
例 .....	13-18
<b>キュー・テーブルの取得 .....</b>	<b>13-19</b>
用途 .....	13-19
使用上の注意 .....	13-20
構文 .....	13-20
例 .....	13-20
<b>宛先プロパティの指定 .....</b>	<b>13-21</b>
用途 .....	13-22
使用上の注意 .....	13-22
構文 .....	13-22
例 .....	13-22
<b>Point-to-Point: キューの作成 .....</b>	<b>13-23</b>
用途 .....	13-23
使用上の注意 .....	13-24
構文 .....	13-24
例 .....	13-24
<b>パブリッシュ/サブスクライブ: トピックの作成 .....</b>	<b>13-25</b>
用途 .....	13-25
使用上の注意 .....	13-26
構文 .....	13-26
例 .....	13-26
<b>システム権限の付与 .....</b>	<b>13-27</b>
用途 .....	13-27
使用上の注意 .....	13-28
構文 .....	13-28
例 .....	13-28
<b>システム権限の取消し .....</b>	<b>13-29</b>

用途 .....	13-29
使用上の注意 .....	13-29
構文 .....	13-30
例 .....	13-30
<b>パブリッシュ / サブスクライブ: トピック権限の付与 .....</b>	<b>13-31</b>
用途 .....	13-32
使用上の注意 .....	13-32
構文 .....	13-32
例 .....	13-32
<b>パブリッシュ / サブスクライブ: トピック権限の取消し .....</b>	<b>13-33</b>
用途 .....	13-33
使用上の注意 .....	13-34
構文 .....	13-34
例 .....	13-34
<b>Point-to-Point: キュー権限の付与 .....</b>	<b>13-35</b>
用途 .....	13-36
使用上の注意 .....	13-36
構文 .....	13-36
例 .....	13-36
<b>Point-to-Point: キュー権限の取消し .....</b>	<b>13-37</b>
用途 .....	13-38
使用上の注意 .....	13-38
構文 .....	13-38
例 .....	13-38
<b>宛先の開始 .....</b>	<b>13-39</b>
用途 .....	13-40
使用上の注意 .....	13-40
構文 .....	13-40
例 .....	13-40
<b>宛先の停止 .....</b>	<b>13-41</b>
用途 .....	13-42
使用上の注意 .....	13-42
構文 .....	13-42
例 .....	13-42
<b>宛先の変更 .....</b>	<b>13-43</b>

用途 .....	13-43
使用上の注意 .....	13-43
構文 .....	13-44
例 .....	13-44
<b>宛先の削除 .....</b>	<b>13-45</b>
用途 .....	13-45
使用上の注意 .....	13-45
構文 .....	13-45
例 .....	13-46
<b>伝播のスケジュール .....</b>	<b>13-47</b>
用途 .....	13-48
使用上の注意 .....	13-48
構文 .....	13-48
例 .....	13-48
<b>伝播スケジュールの使用可能化 .....</b>	<b>13-49</b>
用途 .....	13-49
使用上の注意 .....	13-49
構文 .....	13-50
例 .....	13-50
<b>伝播スケジュールの変更 .....</b>	<b>13-51</b>
用途 .....	13-52
使用上の注意 .....	13-52
構文 .....	13-52
例 .....	13-52
<b>伝播スケジュールの使用不可 .....</b>	<b>13-53</b>
用途 .....	13-53
使用上の注意 .....	13-53
構文 .....	13-54
例 .....	13-54
<b>伝播スケジュールの解除 .....</b>	<b>13-55</b>
用途 .....	13-55
使用上の注意 .....	13-55
構文 .....	13-56
例 .....	13-56

## 14 JMS 操作インタフェース：基本操作（Point-to-Point）

ユースケース・モデル：JMS 操作インタフェース - 基本操作（Point-to-Point） .....	14-2
ユースケース図：JMS 操作インタフェース - 基本操作（Point-to-Point） .....	14-3
キュー接続を確立する 3 つの方法 .....	14-4
ユーザー名 / パスワードでのキュー接続の確立 .....	14-5
用途 .....	14-5
使用上の注意 .....	14-5
構文 .....	14-6
例 .....	14-6
オープン JDBC 接続でのキュー接続の確立 .....	14-7
用途 .....	14-7
使用上の注意 .....	14-7
構文 .....	14-7
例 .....	14-8
デフォルトのコネクション・ファクトリ・パラメータでのキュー接続の確立 .....	14-9
用途 .....	14-9
使用上の注意 .....	14-9
構文 .....	14-9
例 .....	14-10
キュー・セッションの作成 .....	14-11
用途 .....	14-11
使用上の注意 .....	14-11
構文 .....	14-12
例 .....	14-12
キュー送信者の作成 .....	14-13
用途 .....	14-13
使用上の注意 .....	14-13
構文 .....	14-13
例 .....	14-14
キュー送信者を使用してメッセージを送信する 2 つの方法 .....	14-15
デフォルトの送信オプションを持つキュー送信者を使用したメッセージの送信 .....	14-16
用途 .....	14-16
使用上の注意 .....	14-17
構文 .....	14-17
例 .....	14-17

送信オプションの指定によるキュー送信者を使用したメッセージの送信 .....	14-18
用途 .....	14-19
使用上の注意 .....	14-19
構文 .....	14-19
例 .....	14-20
JMS メッセージ・キュー用のキュー・ブラウザを作成する 2 つの方法 .....	14-21
Text、Stream、Object、Byte または Map メッセージを含むキュー用のキュー・ブラウザの作成 ..	14-22
用途 .....	14-22
使用上の注意 .....	14-23
構文 .....	14-23
例 .....	14-23
Text、Stream、Object、Byte または Map メッセージを含むキュー用で、ブラウズ中に メッセージをロックするキュー・ブラウザの作成 .....	14-24
用途 .....	14-24
使用上の注意 .....	14-25
構文 .....	14-25
例 .....	14-25
Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザを作成する 2 つの 方法 .....	14-26
Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザの作成 .....	14-27
用途 .....	14-28
使用上の注意 .....	14-28
構文 .....	14-28
例 .....	14-28
Oracle オブジェクト型 (ADT) メッセージ・キュー用で、ブラウズ中にメッセージをロックする キュー・ブラウザの作成 .....	14-29
用途 .....	14-30
使用上の注意 .....	14-30
構文 .....	14-30
例 .....	14-30
キュー・ブラウザを使用したメッセージのブラウズ .....	14-31
用途: .....	14-31
使用上の注意 .....	14-31
構文 .....	14-31
例 .....	14-32
キュー受信者を作成する 2 つの方法 .....	14-33

標準 JMS 型メッセージ・キューに対するキュー受信者の作成 .....	14-34
用途 .....	14-34
使用上の注意 .....	14-35
構文 .....	14-35
例 .....	14-35
Oracle オブジェクト型 (ADT) メッセージ・キューに対するキュー受信者の作成 .....	14-36
用途 .....	14-37
使用上の注意 .....	14-37
構文 .....	14-37
例 .....	14-37

## 15 JMS 操作インタフェース : 基本操作 (パブリッシュ / サブスクライブ)

ユースケース・モデル : JMS 操作インタフェース – 基本操作 (パブリッシュ / サブスクライブ) ....	15-2
ユースケース図 : JMS 操作インタフェース – 基本操作 (パブリッシュ / サブスクライブ) .....	15-4
トピック接続を確立する 3 つの方法 .....	15-5
ユーザー名 / パスワードでのトピック接続の確立 .....	15-6
用途 .....	15-6
使用上の注意 .....	15-6
構文 .....	15-7
例 .....	15-7
既存の JDBC 接続でのトピック接続の確立 .....	15-8
用途 .....	15-8
使用上の注意 .....	15-8
構文 .....	15-8
例 .....	15-9
デフォルトのコネクション・ファクトリ・パラメータでのトピック接続の確立 .....	15-10
用途 .....	15-10
使用上の注意 .....	15-10
構文 .....	15-10
例 .....	15-10
トピック・セッションの作成 .....	15-11
用途 .....	15-11
使用上の注意 .....	15-11
構文 .....	15-12
例 .....	15-12

トピック・パブリッシャの作成 .....	15-13
用途 .....	15-13
使用上の注意 .....	15-13
構文 .....	15-13
例 .....	15-14
トピック・パブリッシャを使用してメッセージを公開する 4 つの方法 .....	15-15
最小限の指定でのメッセージの公開 .....	15-16
用途 .....	15-16
使用上の注意 .....	15-17
構文 .....	15-17
例 .....	15-17
相関および遅延を指定したメッセージの公開 .....	15-19
用途 .....	15-20
使用上の注意 .....	15-20
構文 .....	15-20
例 .....	15-20
優先順位および Time-To-Live を指定したメッセージの公開 .....	15-22
用途 .....	15-23
使用上の注意 .....	15-23
構文 .....	15-23
例 .....	15-23
トピック・サブスクライバをオーバーライドする受信者リストを指定したメッセージの公開 .....	15-25
用途 .....	15-26
使用上の注意 .....	15-26
構文 .....	15-26
例 .....	15-26
標準 JMS 型メッセージのトピックに対して永続サブスクライバを作成する 2 つの方法 .....	15-28
セクタを指定しない JMS トピックに対する永続サブスクライバの作成 .....	15-29
用途 .....	15-29
使用上の注意 .....	15-30
構文 .....	15-30
例 .....	15-30
セクタを指定しての JMS トピックに対する永続サブスクライバの作成 .....	15-31
用途 .....	15-32
使用上の注意 .....	15-32
構文 .....	15-33

例 .....	15-33
<b>Oracle オブジェクト型 (ADT) メッセージのトピックに対する永続サブスクライバを作成する</b>	
2 つの方法 .....	15-34
<b>セレクトを指定しない JMS トピックに対する永続サブスクライバの作成</b> .....	15-35
用途 .....	15-36
使用上の注意 .....	15-36
構文 .....	15-36
例 .....	15-36
<b>セレクトを指定しての JMS トピックに対する永続サブスクライバの作成</b> .....	15-37
用途 .....	15-38
使用上の注意 .....	15-38
構文 .....	15-39
例 .....	15-39
<b>リモート・サブスクライバを作成する 2 つの方法</b> .....	15-40
<b>JMS メッセージのトピックに対するリモート・サブスクライバの作成</b> .....	15-41
用途 .....	15-42
使用上の注意 .....	15-42
構文 .....	15-42
例 .....	15-42
<b>Oracle オブジェクト型 (ADT) メッセージのトピックに対するリモート・サブスクライバの作成</b> .....	15-44
用途 .....	15-45
使用上の注意 .....	15-45
構文 .....	15-46
例 .....	15-46
<b>永続サブスクリプションのサブスクライブを解除する 2 つの方法</b> .....	15-47
<b>ローカル・サブスクライバに対する永続サブスクリプションのサブスクライブの解除</b> .....	15-48
用途 .....	15-48
使用上の注意 .....	15-49
構文 .....	15-49
例 .....	15-49
<b>リモート・サブスクライバに対する永続サブスクリプションのサブスクライブの解除</b> .....	15-50
用途 .....	15-50
使用上の注意 .....	15-51
構文 .....	15-51
例 .....	15-51



トピック受信者を作成する2つの方法 .....	15-52
標準 JMS 型メッセージのトピックに対するトピック受信者の作成 .....	15-53
用途 .....	15-54
使用上の注意 .....	15-54
構文 .....	15-54
例 .....	15-54
Oracle オブジェクト型 (ADT) メッセージのトピックに対するトピック受信者の作成 .....	15-55
用途 .....	15-56
使用上の注意 .....	15-56
構文 .....	15-56
例 .....	15-56

## 16 JMS 操作インタフェース: 基本操作 (共有インタフェース)

ユースケース・モデル: JMS 操作インタフェース - 基本操作 (共有インタフェース) .....	16-2
JMS 接続の開始 .....	16-6
用途 .....	16-6
使用上の注意 .....	16-6
構文 .....	16-7
例 .....	16-7
セッションからの JMS 接続の取得 .....	16-8
用途 .....	16-8
使用上の注意 .....	16-8
構文 .....	16-8
例 .....	16-9
トランザクションを実行したセッションにおけるすべての操作のコミット .....	16-10
用途 .....	16-10
使用上の注意 .....	16-10
構文 .....	16-10
例 .....	16-11
トランザクションを実行したセッションにおけるすべての操作のロールバック .....	16-12
用途 .....	16-12
使用上の注意 .....	16-12
構文 .....	16-12
例 .....	16-13
JMS セッションからの JDBC 接続の取得 .....	16-14

用途 .....	16-14
使用上の注意 .....	16-14
構文 .....	16-14
例 .....	16-15
<b>Byte メッセージの作成</b> .....	16-16
用途 .....	16-16
使用上の注意 .....	16-16
構文 .....	16-16
例 .....	16-17
<b>Map メッセージの作成</b> .....	16-18
用途 .....	16-18
使用上の注意 .....	16-18
構文 .....	16-18
例 .....	16-19
<b>Stream メッセージの作成</b> .....	16-20
用途 .....	16-20
使用上の注意 .....	16-20
構文 .....	16-20
例 .....	16-21
<b>Object メッセージの作成</b> .....	16-22
用途 .....	16-22
使用上の注意 .....	16-22
構文 .....	16-23
例 .....	16-23
<b>Text メッセージの作成</b> .....	16-24
用途 .....	16-24
使用上の注意 .....	16-24
構文 .....	16-25
例 .....	16-25
<b>ADT メッセージの作成</b> .....	16-26
用途 .....	16-26
使用上の注意 .....	16-26
構文 .....	16-27
例 .....	16-27
<b>メッセージの相関識別子の指定</b> .....	16-28

用途 .....	16-28
使用上の注意 .....	16-28
構文 .....	16-29
例 .....	16-29
<b>JMS メッセージ・プロパティの指定 .....</b>	<b>16-30</b>
使用上の注意 .....	16-31
<b>メッセージ・プロパティを Boolean として指定 .....</b>	<b>16-32</b>
用途 .....	16-32
使用上の注意 .....	16-32
構文 .....	16-33
例 .....	16-33
<b>メッセージ・プロパティを String として指定 .....</b>	<b>16-34</b>
用途 .....	16-34
使用上の注意 .....	16-34
構文 .....	16-35
例 .....	16-35
<b>メッセージ・プロパティを Int として指定 .....</b>	<b>16-36</b>
用途 .....	16-36
使用上の注意 .....	16-36
構文 .....	16-37
例 .....	16-37
<b>メッセージ・プロパティを Double として指定 .....</b>	<b>16-38</b>
用途 .....	16-38
使用上の注意 .....	16-38
構文 .....	16-39
例 .....	16-39
<b>メッセージ・プロパティを Float として指定 .....</b>	<b>16-40</b>
用途 .....	16-40
使用上の注意 .....	16-40
構文 .....	16-41
例 .....	16-41
<b>メッセージ・プロパティを Byte として指定 .....</b>	<b>16-42</b>
用途 .....	16-42
使用上の注意 .....	16-42
構文 .....	16-43

例 .....	16-43
メッセージ・プロパティを Long として指定 .....	16-44
用途 .....	16-44
使用上の注意 .....	16-44
構文 .....	16-45
例 .....	16-45
メッセージ・プロパティを Short として指定 .....	16-46
用途 .....	16-46
使用上の注意 .....	16-46
構文 .....	16-47
例 .....	16-47
メッセージ・プロパティを Object として指定 .....	16-48
用途 .....	16-48
使用上の注意 .....	16-48
構文 .....	16-49
例 .....	16-49
メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの Time-To-Live の 設定 .....	16-50
用途 .....	16-50
使用上の注意 .....	16-50
構文 .....	16-51
例 .....	16-51
メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの優先順位の設定 ...	16-52
用途 .....	16-52
使用上の注意 .....	16-52
構文 .....	16-53
例 .....	16-53
AQjms エージェントの作成 .....	16-54
用途 .....	16-54
使用上の注意 .....	16-55
構文 .....	16-55
例 .....	16-55
メッセージ・コンシューマを使用してメッセージを同期受信する 2 つの方法 .....	16-56
用途 .....	16-56
使用上の注意 .....	16-56
構文 .....	16-57

例 .....	16-57
タイムアウトを指定してのメッセージ・コンシューマを使用したメッセージの受信 .....	16-58
用途 .....	16-58
使用上の注意 .....	16-59
構文 .....	16-59
例 .....	16-59
待機なしでのメッセージ・コンシューマを使用したメッセージの受信 .....	16-60
用途 .....	16-60
使用上の注意 .....	16-60
構文 .....	16-60
例 .....	16-61
メッセージの受信に対するナビゲーション・モードの指定 .....	16-62
用途 .....	16-63
使用上の注意 .....	16-63
構文 .....	16-63
例 .....	16-63
メッセージを非同期受信するためにメッセージ・リスナーを指定する 2 つの方法 .....	16-65
メッセージ・コンシューマでのメッセージ・リスナーの指定 .....	16-66
用途 .....	16-66
使用上の注意 .....	16-66
構文 .....	16-67
例 .....	16-67
セッションでのメッセージ・リスナーの指定 .....	16-69
用途 .....	16-69
使用上の注意 .....	16-69
構文 .....	16-70
例 .....	16-70
メッセージの相関識別子の取得 .....	16-71
用途 .....	16-71
使用上の注意 .....	16-71
構文 .....	16-71
例 .....	16-72
メッセージのメッセージ ID を取得する 2 つの方法 .....	16-73
メッセージのメッセージ ID を Byte として取得 .....	16-74
用途 .....	16-74
使用上の注意 .....	16-74

構文 .....	16-74
例 .....	16-75
メッセージのメッセージ ID を String として取得 .....	16-76
用途 .....	16-76
使用上の注意 .....	16-76
構文 .....	16-76
例 .....	16-77
JMS メッセージ・プロパティの取得 .....	16-78
メッセージ・プロパティを Boolean として取得 .....	16-80
用途 .....	16-80
使用上の注意 .....	16-80
構文 .....	16-81
例 .....	16-81
メッセージ・プロパティを String として取得 .....	16-82
用途 .....	16-82
使用上の注意 .....	16-82
構文 .....	16-83
例 .....	16-83
メッセージ・プロパティを Int として取得 .....	16-84
用途 .....	16-84
使用上の注意 .....	16-84
構文 .....	16-85
例 .....	16-85
メッセージ・プロパティを Double として取得 .....	16-86
用途 .....	16-86
使用上の注意 .....	16-86
構文 .....	16-87
例 .....	16-87
メッセージ・プロパティを Float として取得 .....	16-88
用途 .....	16-88
使用上の注意 .....	16-88
構文 .....	16-89
例 .....	16-89
メッセージ・プロパティを Byte として取得 .....	16-90
用途 .....	16-90
使用上の注意 .....	16-90

構文 .....	16-91
例 .....	16-91
メッセージ・プロパティを Long として取得 .....	16-92
用途 .....	16-92
使用上の注意 .....	16-92
構文 .....	16-93
例 .....	16-93
メッセージ・プロパティを Short として取得 .....	16-94
用途 .....	16-94
使用上の注意 .....	16-94
構文 .....	16-95
例 .....	16-95
メッセージ・プロパティを Object として取得 .....	16-96
用途 .....	16-96
使用上の注意 .....	16-96
構文 .....	16-97
例 .....	16-97
メッセージ・プロデューサのクローズ .....	16-98
用途 .....	16-98
使用上の注意 .....	16-98
構文 .....	16-98
例 .....	16-99
メッセージ・コンシューマのクローズ .....	16-100
用途 .....	16-100
使用上の注意 .....	16-100
構文 .....	16-100
例 .....	16-101
JMS 接続の停止 .....	16-102
用途 .....	16-102
使用上の注意 .....	16-102
構文 .....	16-102
例 .....	16-103
JMS セッションのクローズ .....	16-104
用途 .....	16-104
使用上の注意 .....	16-104

構文 .....	16-104
例 .....	16-105
<b>JMS 接続のクローズ .....</b>	<b>16-106</b>
用途 .....	16-106
使用上の注意 .....	16-106
構文 .....	16-107
例 .....	16-107
<b>JMS 例外のエラー・コードの取得 .....</b>	<b>16-108</b>
用途 .....	16-108
使用上の注意 .....	16-108
構文 .....	16-108
例 .....	16-109
<b>JMS 例外のエラー番号の取得 .....</b>	<b>16-110</b>
用途 .....	16-110
使用上の注意 .....	16-110
構文 .....	16-110
例 .....	16-111
<b>JMS 例外のエラー・メッセージの取得 .....</b>	<b>16-112</b>
用途 .....	16-112
使用上の注意 .....	16-112
構文 .....	16-112
例 .....	16-113
<b>JMS 例外にリンクされた例外の取得 .....</b>	<b>16-114</b>
用途 .....	16-114
使用上の注意 .....	16-114
構文 .....	16-114
例 .....	16-115
<b>JMS 例外のスタック・トレースの出力 .....</b>	<b>16-116</b>
用途 .....	16-116
使用上の注意 .....	16-116
構文 .....	16-116
例 .....	16-117

## A Oracle アドバンスト・キューイングの使用例

キュー・テーブルおよびキューの作成 .....	A-4
-------------------------	-----



オブジェクト型のキュー・テーブルおよびキューの作成 .....	A-4
RAW 型のキュー・テーブルおよびキューの作成 .....	A-5
優先順位指定メッセージのキュー・テーブルおよびキューの作成 .....	A-5
複数コンシューマのキュー・テーブルおよびキューの作成 .....	A-5
伝播を実証するためのキューの作成 .....	A-6
Java AQ の例の設定 .....	A-6
Java AQ セッションの作成 .....	A-7
Java を使用したキュー・テーブルおよびキューの作成 .....	A-8
Java を使用したキューの作成およびエンキュー / デキューの開始 .....	A-9
Java を使用した複数コンシューマ・キューの作成およびサブスクライバの追加 .....	A-9
<b>メッセージのエンキューおよびデキュー .....</b>	<b>A-11</b>
PL/SQL を使用したオブジェクト型メッセージのエンキューおよびデキュー .....	A-11
Pro*C/C++ を使用したオブジェクト型メッセージのエンキューおよびデキュー .....	A-12
OCI を使用したオブジェクト型メッセージのエンキューおよびデキュー .....	A-14
Java を使用したオブジェクト型メッセージ (CustomDatum インタフェース) のエンキュー およびデキュー .....	A-16
Java を使用したオブジェクト型メッセージ (SQLData インタフェース使用) のエンキュー およびデキュー .....	A-18
PL/SQL を使用した RAW 型メッセージのエンキューおよびデキュー .....	A-21
Pro*C/C++ を使用した RAW 型メッセージのエンキューおよびデキュー .....	A-22
OCI を使用した RAW 型メッセージのエンキューおよびデキュー .....	A-25
Java を使用した RAW 型メッセージのエンキュー .....	A-26
Java を使用したメッセージのデキュー .....	A-27
Java を使用したブラウズ・モードでのメッセージのデキュー .....	A-28
PL/SQL を使用した優先順位によるメッセージのエンキューおよびデキュー .....	A-30
Java を使用した優先順位によるメッセージのエンキュー .....	A-32
PL/SQL を使用したプレビュー後のメッセージのデキュー .....	A-33
PL/SQL を使用した遅延および期限切れによるメッセージのエンキューおよびデキュー .....	A-36
Pro*C/C++ を使用した相関識別子およびメッセージ ID によるメッセージのエンキューおよび デキュー .....	A-37
OCI を使用した相関識別子およびメッセージ ID によるメッセージのエンキューおよび デキュー .....	A-42
PL/SQL を使用した複数コンシューマ・キューでのメッセージのエンキューおよびデキュー ....	A-44
OCI を使用した複数コンシューマ・キューでのメッセージのエンキューおよびデキュー .....	A-47
PL/SQL を使用したメッセージのグループ化によるメッセージのエンキューおよびデキュー ....	A-51
PL/SQL を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー ..	A-53

Java を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー .....	A-56
<b>伝播</b> .....	A-62
PL/SQL を使用したリモートのサブスクライバ / 受信者用のメッセージの複数コンシューマ・ キューへのエンキューおよび伝播のスケジュール .....	A-62
PL/SQL を使用した同一データベース内の 1 つのキューから他のキューへの伝播管理 .....	A-64
PL/SQL を使用した 1 つのキューから他のデータベース内の他のキューへの伝播管理 .....	A-64
PL/SQL を使用した伝播スケジュールの解除 .....	A-65
<b>AQ オブジェクトの削除</b> .....	A-66
<b>ロールおよび権限の取消し</b> .....	A-67
<b>AQ による XA の使用</b> .....	A-68
<b>AQ およびメモリーの使用</b> .....	A-73
Creat_types.sql: Scott のスキーマへのペイロード型およびキューの作成 .....	A-73
OCI を使用したメッセージのエンキュー（各コール後のメモリー解放） .....	A-73
OCI を使用したメッセージのエンキュー（メモリーの再使用） .....	A-77
OCI を使用したメッセージのデキュー（各コール後のメモリー解放） .....	A-81
OCI を使用したメッセージのデキュー（メモリーの再使用） .....	A-84

## B Oracle JMS インタフェース、クラスおよび例外

Oracle JMS クラス（パート 1） .....	B-5
Oracle JMS クラス（パート 2） .....	B-6
Oracle JMS クラス（パート 3） .....	B-7
Oracle JMS クラス（パート 4） .....	B-8
Oracle JMS クラス（パート 5） .....	B-9
Oracle JMS クラス（パート 6） .....	B-10
Oracle JMS クラス（パート 6 の続き） .....	B-11
Oracle JMS クラス（パート 7） .....	B-12
Oracle JMS クラス（パート 8） .....	B-13
Oracle JMS クラス（パート 9） .....	B-14
Oracle JMS クラス（パート 10） .....	B-15
Oracle JMS クラス（パート 10 の続き） .....	B-16
インタフェース - javax.jms.BytesMessage .....	B-17
インタフェース - javax.jms.Connection .....	B-18
インタフェース - javax.jms.ConnectionFactory .....	B-19
インタフェース - javax.jms.ConnectionMetaData .....	B-20
インタフェース - javax.jms.DeliveryMode .....	B-21
インタフェース - javax.jms.Destination .....	B-22
インタフェース - javax.jms.MapMessage .....	B-23

インタフェース - javax.jms.Message .....	B-24
インタフェース - javax.jms.MessageConsumer .....	B-25
インタフェース - javax.jms.MessageListener .....	B-26
インタフェース - javax.jms.MessageProducer .....	B-27
インタフェース - javax.jms.ObjectMessage .....	B-28
インタフェース - javax.jms.Queue .....	B-29
インタフェース - javax.jms.QueueBrowser .....	B-30
インタフェース - javax.jms.QueueConnection .....	B-31
インタフェース - javax.jms.QueueConnectionFactory .....	B-32
インタフェース - javax.jms.QueueReceiver .....	B-33
インタフェース - javax.jms.QueueSender .....	B-34
インタフェース - javax.jms.QueueSession .....	B-35
インタフェース - javax.jms.Session .....	B-36
インタフェース - javax.jms.StreamMessage .....	B-37
インタフェース - javax.jms.TextMessage .....	B-38
インタフェース - javax.jms.Topic .....	B-39
インタフェース - javax.jms.TopicConnection .....	B-40
インタフェース - javax.jms.TopicConnectionFactory .....	B-41
インタフェース - javax.jms.TopicPublisher .....	B-42
インタフェース - javax.jms.TopicSession .....	B-43
インタフェース - javax.jms.TopicSubscriber .....	B-44
例外 - javax.jms.InvalidDestinationException .....	B-45
例外 - javax.jms.InvalidSelectorException .....	B-46
例外 - javax.jms.JMSEException .....	B-47
例外 - javax.jms.MessageEOFException .....	B-48
例外 - javax.jms.MessageFormatException .....	B-49
例外 - javax.jms.MessageNotReadableException .....	B-50
例外 - javax.jms.MessageNotWriteableException .....	B-51
インタフェース - oracle.jms.AdtMessage .....	B-52
インタフェース - oracle.jms.AQjmsQueueReceiver .....	B-53
インタフェース - oracle.jms.AQjmsQueueSender .....	B-54
インタフェース - oracle.jms.AQjmsTopicPublisher .....	B-55
インタフェース - oracle.jms.TopicReceiver .....	B-56
インタフェース - oracle.jms.AQjmsTopicSubscriber .....	B-57
インタフェース - oracle.jms.AQjmsTopicReceiver .....	B-58
クラス - oracle.jms.AQjmsAdtMessage .....	B-59
クラス - oracle.jms.AQjmsAgent .....	B-60
クラス - oracle.jms.AQjmsBytesMessage .....	B-61

クラス - oracle.jms.AQjmsConnection .....	B-62
インタフェース - oracle.jms.AQjmsConnectionMetadata .....	B-63
クラス - oracle.jms.AQjmsConstants .....	B-64
インタフェース - oracle.jms.AQjmsConsumer .....	B-65
クラス - oracle.jms.AQjmsDestination .....	B-66
クラス - oracle.jms.AQjmsDestinationProperty .....	B-67
クラス - oracle.jms.AQjmsFactory .....	B-68
クラス - oracle.jms.AQjmsMapMessage .....	B-69
クラス - oracle.jms.AQjmsMessage .....	B-70
クラス - oracle.jms.AQjmsObjectMessage .....	B-71
クラス - oracle.jms.AQjmsOracleDebug .....	B-72
クラス - oracle.jms.AQjmsProducer .....	B-73
クラス - oracle.jms.AQjmsQueueBrowser .....	B-74
クラス - AQjmsQueueConnectionFactory .....	B-75
クラス - oracle.jms.AQjmsSession .....	B-76
クラス - oracle.jms.AQjmsStreamMessage .....	B-77
クラス - oracle.jms.AQjmsTextMessage .....	B-78
クラス - oracle.jms.AQjmsTopicConnectionFactory .....	B-79
例外 - oracle.jms.AQjmsException .....	B-80
例外 - oracle.jms.AQjmsInvalidDestinationException .....	B-81
例外 - oracle.jms.AQjmsInvalidSelectorException .....	B-82
例外 - oracle.jms.AQjmsMessageEOFException .....	B-83
例外 - oracle.jms.AQjmsMessageFormatException .....	B-84
例外 - oracle.jms.AQjmsMessageNotReadableException .....	B-85
例外 - oracle.jms.AQjmsMesssageNotWriteableException .....	B-86
インタフェース - oracle.AQ.AQQueueTable .....	B-87
クラス - oracle.AQ.AQQueueTableProperty .....	B-88

## C BooksOnLine 用スクリプト

tkaqdoca.sql: ユーザー、オブジェクト、キュー・テーブル、キューおよびサブスクライバ作成用の スクリプト .....	C-2
tkaqdocd.sql: 管理インタフェースおよび操作インタフェースの例 .....	C-16
tkaqdoce.sql: 操作例 .....	C-21
tkaqdocp.sql: 操作インタフェースの例 .....	C-22
tkaqdocc.sql: クリーンアップ・スクリプト .....	C-37

## D JMS エラー・メッセージ

### 索引



---

# はじめに

このマニュアルでは、Oracle Server におけるアプリケーション開発のうち、Oracle アドバンスド・キューイングに関連する機能について説明します。このマニュアルの情報は、すべてのプラットフォームで実行する Oracle Server の各バージョンに適用されますが、システム固有の情報は含まれていません。

内容は次のとおりです。

- [このマニュアルについて](#)
- [Oracle8i で導入された新機能](#)
- [関連マニュアル](#)
- [このマニュアルの構成](#)
- [図の解釈方法](#)
- [このマニュアルの表記規則](#)

## このマニュアルについて

Oracle アドバンスト・キューイング (Oracle AQ) では、Oracle Server に統合されたメッセージ・キューイングが提供されます。Oracle AQ は、キューイング・システムとデータベースを統合し、メッセージ対応のデータベースを作成することでこの機能を実現しています。Oracle AQ が提供する統合ソリューションによって、アプリケーション開発者は、メッセージ交換用のインフラストラクチャを組み立てる必要がなくなり、特定のビジネス・ロジックに力を注ぐことができます。

このマニュアルは、Oracle アドバンスト・キューイングを使用して新たにアプリケーションを開発するプログラムの他、すでにこのテクノロジーを実装して、さらに新機能を活用しようとしているプログラムを対象としています。

Oracle AQ の重要性が高まりつつあることを受けて、Oracle アプリケーション開発者用のドキュメント・セットの中で独立したマニュアルとして提供されるようになりました。

## Oracle8i で導入された新機能

- キュー・レベルのアクセス制御
- 非永続キュー
- OPS 環境のサポート
- パブリッシュ / サブスクライプ用のルールベース・サブスクライバ
- 非同期通知
- 送信元の識別
- リスニング機能 (複数のキューでの待機)
- LOB を伴うメッセージの伝播
- 伝播スケジューリング機能の拡張
- ペイロード (実際に通信される情報) を伴わないメッセージ・ヘッダーのみのデキュー
- 統計ビューのサポート
- Java API
- 履歴管理情報の記憶域の分離

---

---

### 参照:

- [第 8 章「AQ を使用したサンプル・アプリケーション」](#) を参照してください。
- 
-



## 関連マニュアル

PL/SQL は、SQL に対してオラクル社が開発したプロシージャ型拡張機能です。この高水準プログラミング言語の詳細は、『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

Oracle コール・インタフェース (OCI) については、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

OCI を使用すると、Oracle Server にアクセスする第 3 世代言語 (3GL) アプリケーションを作成できます。

オラクル社は、プリコンパイラの Pro\* シリーズも提供しています。これを使用することで、ご使用のアプリケーション・プログラムに SQL および PL/SQL を組み込むことができます。埋込み SQL を取り込む 3GL アプリケーション・プログラムを Ada、C、C++、COBOL または FORTRAN で作成する場合は、該当するプリコンパイラ・マニュアルを参照してください。たとえば、C または C++ でプログラミングする場合は、『Oracle8i Pro\*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

SQL については、『Oracle8i SQL リファレンス』および『Oracle8i 管理者ガイド』を参照してください。Oracle の基本概念については、『Oracle8i 概要』を参照してください。

# このマニュアルの構成

このマニュアルは、次の各章および付録で構成されています。

## 第1章「Oracle アドバンスト・キューイングの概要」

この章では、最適なメッセージ・システムの要件について説明します。Oracle AQ は比較的新しいテクノロジーであり、この目標がすべて実現されているわけではありませんが、基礎となる設計の概要を把握し、それが意図している方向を明確に理解できます。

## 第2章「基本的なコンポーネント」

この章では、Oracle AQ で提供されている機能について、「一般機能」、「エンキュー機能」および「デキュー機能」の項に沿って説明します。

## 第3章「AQ プログラム環境」

この章では、ユーザーの作業に必要な要素、および AQ アプリケーション環境を準備するときに考慮する必要がある問題点を説明します。

## 第4章「AQ の管理」

この章では、キュー・テーブル（インポート / エクスポート）の移行、セキュリティ、Enterprise Manager サポート、プロトコル、AQ の作業準備での dba サンプル・アクション、現行の制限事項など、アドバンスト・キューイングの管理について説明します。

## 第5章「高度なトピック」

この章では、高度なトピックについて説明します。

## 第6章「FAQ」

この章では、よくある質問事項について説明します。

## 第7章「モデリングおよび設計」

この章では、アドバンスト・キューイング・モデリングおよび設計の基礎について説明します。

## 第8章「AQ を使用したサンプル・アプリケーション」

この章では、サンプル・アプリケーションのコンテキストにおける Oracle アドバンスト・キューイングの機能について説明します。

## 第9章「管理インタフェース」

この章では、Oracle アドバンスト・キューイング用の管理インタフェースについて説明します。

## 第 10 章「管理インタフェース：ビュー」

この章では、ユースケースと状態図を組み合わせた複合図に沿って、ビューの管理インタフェースについて説明します。

## 第 11 章「操作インタフェース：基本操作」

この章では、ユースケースに沿って、Oracle アドバンスド・キューイングの操作インタフェースについて説明します。

## 第 12 章「JMS を使用したアプリケーションの作成」

この章では、使用例に基づいたサンプル・アプリケーションに沿って、AQ の機能について説明します。

## 第 13 章「JMS 管理インタフェース：基本操作」

この章では、ユースケースに沿って、Oracle アドバンスド・キューイングの管理インタフェースについて説明します。

## 第 14 章「JMS 操作インタフェース：基本操作（Point-to-Point）」

この章では、Point-to-Point の操作について説明します。

## 第 15 章「JMS 操作インタフェース：基本操作（パブリッシュ / サブスクライブ）」

この章では、パブリッシュ / サブスクライブの操作について説明します。

## 第 16 章「JMS 操作インタフェース：基本操作（共有インタフェース）」

この章では、共有インタフェースの操作について説明します。

## 付録 A「Oracle アドバンスド・キューイングの使用例」

この付録では、様々なプログラム環境での使用例を掲載しています。

## 付録 B「Oracle JMS インタフェース、クラスおよび例外」

この付録では、Oracle JMS インタフェース、クラスおよび例外リストを掲載しています。

## 付録 C「BooksOnLine 用スクリプト」

この付録では、サンプル・アプリケーション「BooksOnLine」で使用するスクリプトを掲載しています。

付録 D 「JMS エラー・メッセージ」

この付録では、トラブルシューティングのために、エラー・メッセージ、原因および処置のリストを掲載しています。

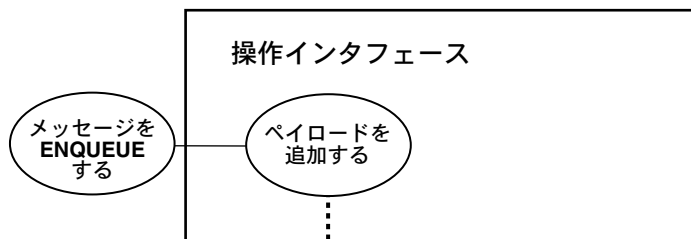
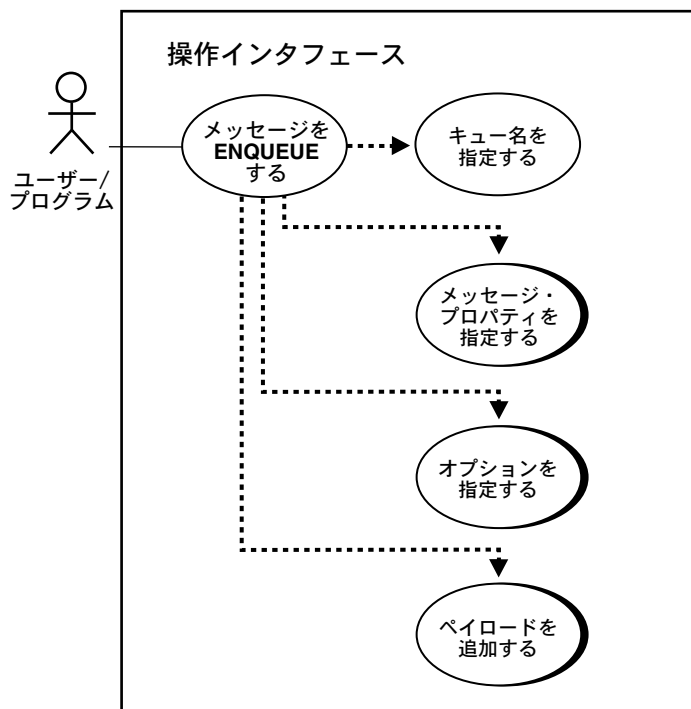
図の解釈方法

このマニュアルは、テクノロジーの説明方法として統一モデリング言語（UML）を使用しています。このマニュアルには UML の完全な説明はありませんが、『Oracle8i アプリケーション開発者ガイド ラージ・オブジェクト』の「はじめに」の章で、UML 表記について簡単に説明しています。次の説明は、UML 表記のうち、このマニュアルで使用する要素を抜き出したものです。

ユースケース図

図形要素	説明
	<p>このマニュアルでは、今回のリリースからユースケース図を採用して幅広く活用しています。1 次ユースケースは、それぞれアクター（人を表すマーク）によって開始します。アクターは、ユーザー、アプリケーションまたはサブプログラムのものでかまいません。アクターは、ユースケース・アクションを囲む楕円（吹出し）で示される 1 次ユースケースに接続されます。</p> <p>1 次ユースケース全体は、ユースケース・モデル図によって記述されます。</p>
	<p>1 次ユースケースを完了するには、他の操作が必要になる場合があります。この図では、</p> <ul style="list-style-type: none"><li>■ キュー名を指定する</li></ul> <p>が、次に示すオペレーションを完了するために必要なサブオペレーションまたは 2 次ユースケースの 1 つです。</p> <ul style="list-style-type: none"><li>■ メッセージを ENQUEUE する</li></ul> <p>1 次ユースケースから、必要な他の操作（ここでは省略されています）に向かって下向きの線が伸びています。</p>

## 図形要素



## 説明

影付きの2次ユースケースは、展開されます（自分自身のユースケース図によって記述されます）。これには次の2つの理由があります。

- (a) オペレーション・ロジックが理解しやすくなる。
- (b) すべてのオペレーションおよびサブオペレーションを同一ページ内に収めることができない。

この例では、

- メッセージ・プロパティを指定する
- オプションを指定する
- ペイロードを追加する

というアクションは、すべてさらに詳しいユースケース図に展開されます。

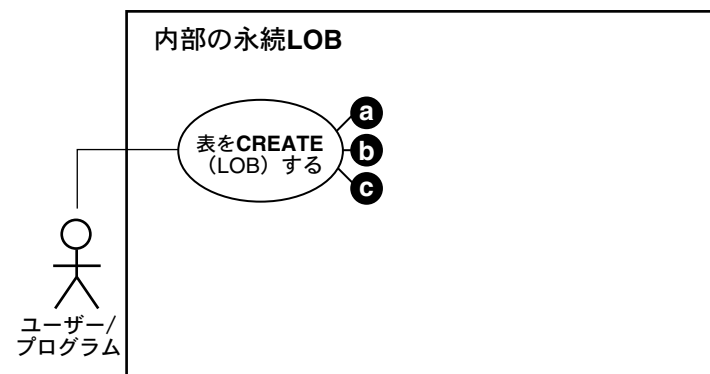
この図（抜粋）には、展開されたユースケース図が示されています。標準的な図は、通常、アクターから始まりますが、ここではユースケース自体がサブオペレーションへの出発点になっています。この例では、

- ペイロードを追加する

の展開ビューが、次のオペレーションの構成要素を表しています。

- メッセージを ENQUEUE する

## 図形要素



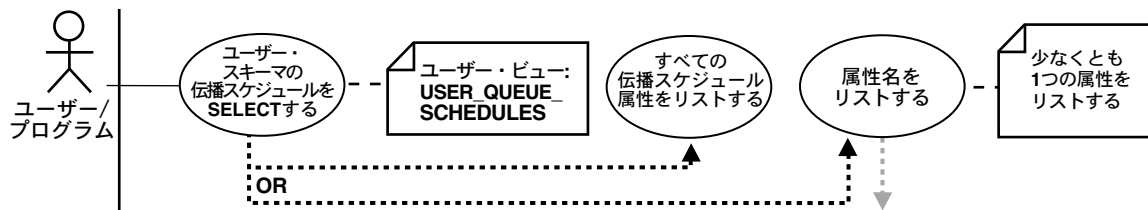
## 説明

この表記（a、b、c）は、LOBを含む表の作成に3つの異なる方法があることを示します。



これは注釈ボックスの使用方法の1つを示したもので、LOBを含む表の3つの作成方法のどれを表したものが区別できます。

## 図形要素



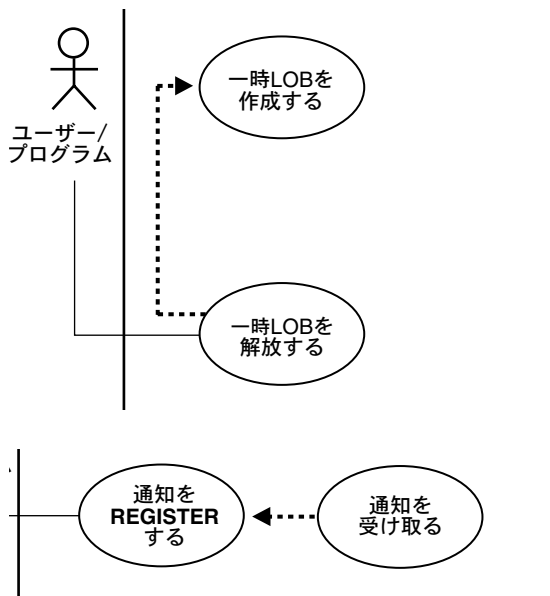
## 説明

この図では、注釈ボックスの表記でよく使用されるものを2つ示します。

(a) 代替名を表す場合、この例では、「伝播スケジュールを SELECT する」というアクションは、USER\_QUEUE\_SCHEDULES というビューによって表されます。

(b) 「属性名をリストする」というアクションには、ユーザーに対する注釈が付けられています。注釈には、伝播スケジュール属性をすべてリストしない場合は、属性を少なくとも1つリストする必要があることが示されています。

## 図形要素



## 説明

ユースケース図の破線の矢印は、依存性を示します。この例では、

- 一時LOBを解放する

するためには、まず、

- 一時LOBを作成する

必要があることを示しています。

矢印の宛先は、最初に実行する必要があるオペレーションを示すということを理解しておく必要があります。

ユースケースとそのサブオペレーションは複雑な関係でリンクする可能性があります。このコールバックの例では、まず

- 通知をREGISTERする

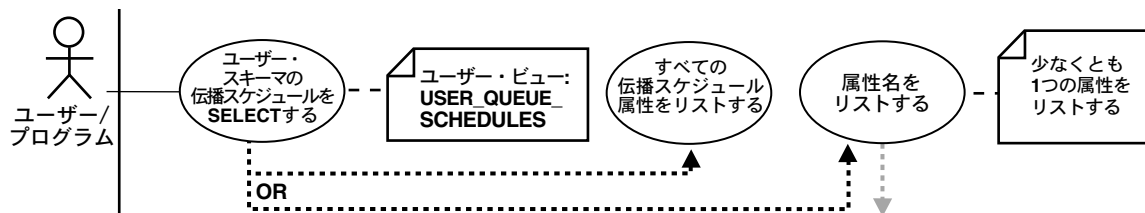
を実行した後で

- 通知を受け取る

必要があります。



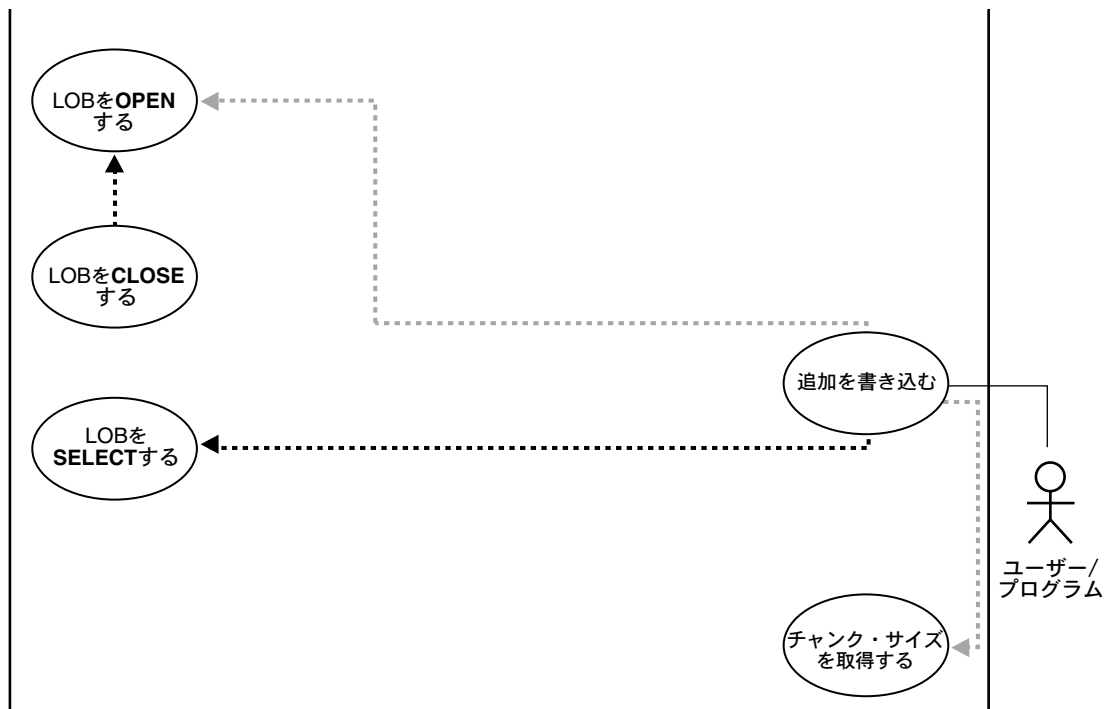
## 図形要素



## 説明

この例では、OR 条件の分岐パスが示されています。ビューを起動する際に、すべての属性のリストを選択するか、1つ以上の属性を表示するかを選択できます。ユーザーがどの属性を参照可能にするかを指定できるということが、灰色の矢印で示されています。

## 図形要素



## 説明

線の付いたオペレーションがすべて必須というわけではありません。黒の破線および矢印は、ユースケースを完了するにはその矢印の宛先のオペレーションを実行する必要があることを示しますが、オプションのアクションは灰色の破線および矢印で示されています。この例では、

- 追加を書き込む

を LOB に対して実行するには、まず

- LOB を SELECT する

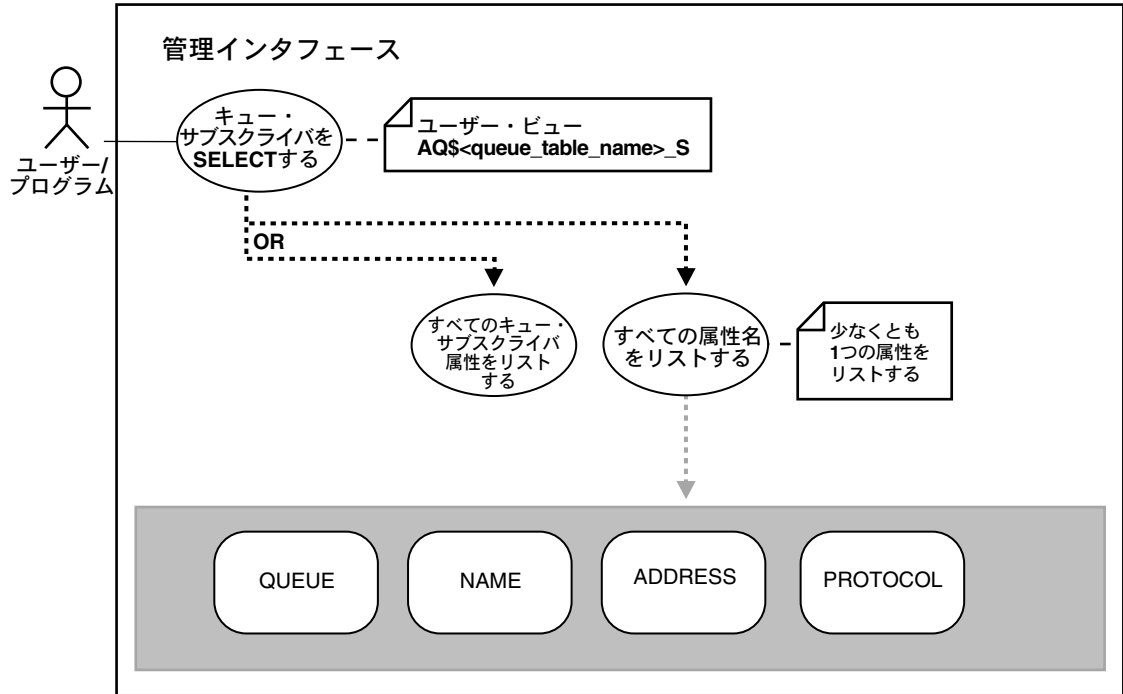
必要があります。これを支援する操作として、次を選択できます。

- LOB を OPEN する、またはチャンク・サイズを取得する（あるいはその両方）

ただし、「LOB を OPEN する」場合は、後で「LOB を CLOSE する」必要があります。

## 状態図

### 図形要素



### 説明

これまででは、すべてユースケース図に沿って説明しましたが、ここでは、ビューの属性を表すためにこのマニュアルで使用している状態図の基本的な使用方法を説明します。実際、ビューの属性には、可視または不可視という2つの状態しかありません。表示する必要があるのは状態の変化ではなく、ビューの起動時に何が見えるようにするかです。このため、UMLを拡張し、部分的な状態図をユースケース図に連結して属性の全体像を表示し、ビューのサブ状態をすべて表示できるようにしました。ユースケースとビューの状態は、状態図の背景を灰色にすることで区別しています。

この例では、キューのサブスクライバを問い合わせることができます。4つの属性は、単独で、いくつか組み合わせて、またはすべてを指定できます。

図形要素	説明
<div data-bbox="154 348 843 432">内部一時LOB（パート1/2）</div> <div data-bbox="141 569 404 616">次ページに続く</div>	<p data-bbox="855 309 1263 520">ユースケース・モデル図は、内部一時LOBなどの特定ドメイン内のユースケースをすべて要約したものです。この図は1ページに収まらないほど複雑になることがあります。その場合は、図を2つの部分に分割します。この分割は順序を意味するものではないため注意してください。</p> <p data-bbox="855 539 1263 638">場合によっては、単にページが長くなりすぎるという理由で図を分割することがあります。このようなときのために、このマーカーが用意されています。</p>

# このマニュアルの表記規則

このマニュアルで使用する表記およびフォーマットの規則は、次のとおりです。

**[ ]**

大カッコは、中に含まれている項目がオプションであることを示します。カッコは入力しないでください。

**{ }**

中カッコは、複数の項目を囲み、その中の 1 つのみが必要であることを示します。

**|**

縦棒は、中カッコ内の項目を分割します。また、複数の値を関数のパラメータに渡すことを示す場合にも使用します。

**...**

コード例における省略記号は、その説明内容に関係のないコードを省略していることを示します。

## 固定幅フォント

SQL または C のコーディング例は、固定幅フォントで示します。

## イタリック

イタリックは、OCI パラメータ、OCI ルーチン名、ファイル名およびデータ・フィールドを示します。

## 大文字

大文字は、SELECT、UPDATE などの SQL キーワードを示します。

このマニュアルでは、ある情報に対して読者の注意を促すために、特別なテキスト・フォーマットを使用しています。太字テキストの見出しで始まる段落は、特別な意味を持つ場合があります。次の各段落では、それぞれの見出しで示される各種の情報を説明します。

**注意：**共通の問題を避けたり概念の理解を深めるために、情報に特別な注意を払う必要があることを示します。

**警告：**アプリケーションが正しく動作するために注意して行う必要があること、および行ってはならないことを OCI プログラマに示します。

**参照：**説明する項目についての追加情報が記載されているこのマニュアルの他の項、または他のマニュアルを示します。



---

# Oracle アドバンスト・キューイングの概要

この章では、Oracle アドバンスト・キューイング（AQ）の概要を説明します。この章では、分散環境における複雑な情報処理の要件について考察します。内容は次のとおりです。

- [キューイング: メッセージ操作の最適な手段](#)
- [キューイング・アプリケーション開発のコンテキスト](#)
- [キューイング・メッセージのモデル](#)
- [アドバンスト・キューイング（AQ）の機能](#)
- [アドバンスト・キューイング（ネイティブ AQ）の要素](#)
- [デモの参照](#)

## キューイング：メッセージ操作の最適な手段

### ビジネス・イベントの核心としてのメッセージ

次のようなアプリケーション・シナリオを想定してみます。

大型の書籍販売店である BooksOnLine の運営は、販売プロセス全体に関わる様々な部署にまたがるアクティビティを自動化する、オンライン書籍発注システムが基礎となっています。システムのフロント・エンドは、新しい注文を入力するための注文入力アプリケーションです。入力された注文は、注文処理アプリケーションによって妥当性チェックおよび記録されます。地域倉庫に置かれた出荷部門は、発注された書籍をすぐに確実に発送する責任があります。地域倉庫は3箇所、それぞれが東部地域、西部地域、そして海外からの発注に対応しています。発注された書籍の発送完了後、発注情報は支払処理のために中央の請求部門に送られます。各地の顧客サービス部門は、発注情報の状況を管理し、発注に関する問合せを処理する責任があります。

このシナリオは、発生したメッセージを分散コンピューティング環境の複数のクライアント（ノード）に分配するアプリケーションを示しています。メッセージはクライアントとサーバーの間で受け渡されるのみでなく、異なるサーバーのプロセス間で複雑に分散配置されます。様々なコンポーネントを統合した大規模アプリケーションは、複数のステップからなるプロセスで構成されています。各ステップは1つ以上のメッセージによって引き起こされ、また、1つ以上のメッセージを発生させることがあります。

### メッセージ交換の調整に対するキューの使用

ビジネス・アプリケーションは、前述のメッセージを使用して相互に通信します。問題は、この通信がどのように調整されるかということです。BooksOnLine は1つの企業ですが、インターネットを介した商取引が広まるにつれて、異なる企業のアプリケーションが、相互に通信する必要性がますます高まっています。通常、これらのアプリケーション間のやりとりは、データを含むメッセージ、およびデータに埋め込まれているアクションに対する要求で構成されます。そのため、企業内および企業間のアプリケーションは自律的に機能する必要があり、同時に相互に依存して情報を処理する必要もあるという問題があります。

キューイング・システムでは、プロデューサ・アプリケーションはキューにメッセージを入れます（エンキュー）。通常、このメッセージは、コンシューマ・アプリケーションによって同じキューから取り出されます（デキュー）。このようなモデルでは、応答の待機によってブロックされることがないため、要求をキューに入れた後も処理を続けるアプリケーションに適しています。また、取り出すメッセージが発生するまで、アプリケーションが処理を継続することも可能になります。



## キューイング・システムによる効果

### パフォーマンス

「サービスに対する要求」と「サービスの供給」を分離することによって効率が向上し、複雑なスケジューリングに対するインフラストラクチャが提供されます。さらに、キューイング・システムは、次の基準で測定した際に、高いパフォーマンス特性を示す必要があります。

- 1秒間にエンキュー / デキューされるメッセージ数
- メッセージ・ウェアハウスに対する複雑な問合せの評価にかかる時間
- 障害後にメッセージ処理のリカバリ / 再起動にかかる時間

### スケーラビリティ

キューイング・システムは、高いスケーラビリティを示す必要があります。アプリケーションを使用するプログラム数が増加しても、メッセージ数が増加しても、またメッセージ・ウェアハウスのサイズが増加しても、システムは一貫して高いパフォーマンスを示し続ける必要があります。

### セキュリティのための永続性

メッセージの受渡しの複雑なスケジューリング操作は、単に難しいというだけではありません。残念なことに、ネットワーク、コンピュータ・ハードウェアおよびソフトウェア・アプリケーションのすべてのもので、障害が発生する場合があります。ネットワーク、マシンおよびアプリケーションに障害が発生したときに、遅延実行を正常に動作させるためには、サービスへの要求から構成されるメッセージが永続的に格納され、確実に1回のみ処理される必要があります。つまり、メッセージ機能には永続性が必要です。

メッセージを保存できることが、基本となります。アプリケーションは、外部クライアントまたは内部プログラムから同時に到着する、複数の未処理メッセージを処理することが必要な場合があり、そのようなときに必要なリソースが得られない可能性があります。同様に、データベース間の通信リンクは、常に使用可能なわけではなく、他の用途のために確保されていることもあります。システム容量が不足しているためにメッセージをすぐに処理できない場合、アプリケーションは、処理可能になるまでそのメッセージを保存しておく必要があります。同様に、外部クライアントまたは内部プログラムが、処理済メッセージを受信する準備ができていないこともあります。

### スケジューリングのための永続性

キューイング・システムには、優先順位に対処できるメッセージの永続性も必要です。後から到着したメッセージが先に到着したメッセージよりも高い優先順位を持つ場合、先に到着したメッセージが後のメッセージを待ってからアクションを実行する場合、同一のメッセージが異なるプロセスからアクセスされる場合などです。このような優先順位は、固定できません。メッセージの永続性の動的な側面を処理するときに重要なことの1つは、処理可能枠が伸びたり縮んだりすることに連動していることです。特定のキューにあるメッセージが他のキューにあるメッセージより重要になり、遅延や他のキューのメッセージからの介入が少なくなるような処理が必要な場合があるためです。同様に、ある受信者へのメッセージ送信が、他の受信者への送信よりも重要になる場合があります。

### メタデータのアクセスおよび分析のための永続性

最後に、メッセージの制御コンポーネントがペイロード・データと同様に重要になりうるため、メッセージの永続性はさきわめて重要です。たとえば、メッセージの受信時刻またはディスパッチ時刻がメッセージの重要な部分であることがあります。しばしばこの時刻がトランザクションに対する正当なインタフェースを構成します。正当性の問題は別として、メッセージは実行後もビジネス資産としての重要性を保持することがよくあります。たとえば、最大要求の周期の分析、または注文を受信してから処理を完了するまでのタイムラグの評価が重要になることがあります。これらの要件を考慮すると、追跡およびドキュメント化は、開発者ではなくメッセージ・システムによって処理される必要があります。

## キューイング・アプリケーション開発のコンテキスト

Oracle AQ では、次の2つの開発コンテキストを提供します。

- **ネイティブ AQ** 次の3つのプログラム環境を介してアクセスできます。
  - PL/SQL パッケージ DBMS\_AQ/AQADM を使用した PL/SQL (『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』を参照)
  - Oracle Object for OLE (OO4O) を使用した Visual Basic (Oracle Object for OLE に関するオンライン・ヘルプを参照)
  - Java パッケージ oracle.AQ を使用した Java (『Oracle8i Java パッケージ・プロシージャリファレンス』を参照)

このコンテキストでの作業については、第1章～第11章で説明します。

- **Java Messaging Service (JMS)**
  - Java パッケージ oracle.jms を使用した Java。この公開標準の実装によって、定義済みの W3C インタフェースが拡張され、JMS コンテキストで操作する開発者が、ネイティブ AQ と同じ手段を使用できるようになります。

このコンテキストでの作業については、第 12 章～第 16 章で説明します。oracle.jms パッケージの詳細は、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

次の項で説明するように、これらのコンテキストでは、同じ機能に対して異なる用語が使用されます。

## キューイング・メッセージのモデル

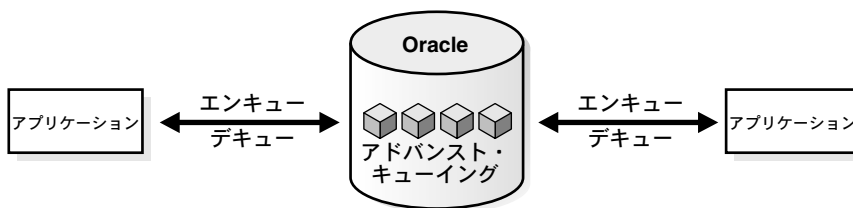
Oracle AQ には、メッセージの送受信に関する主要なモデルが 2 つあります。

- Point-to-Point
- パブリッシュ/サブスクライブ

### Point-to-Point モデル

このモデルのメッセージは、特定のターゲットを対象としています。つまり、送信者および受信者は、メッセージ交換に使用する共通キューを決定します。各メッセージは、1 人の受信者のみによって使用されます。図 1-1 「Point-to-Point モデル」に、各アプリケーションが独自のメッセージ・キューを持つことを示します。

図 1-1 Point-to-Point モデル



**ネイティブ AQ 用語** この図では、キューにメッセージを入れるプロセス、およびキューからメッセージを取り出すプロセスに対して、それぞれ**エンキュー**および**デキュー**というネイティブ AQ 用語を使用しています。より正確には、各メッセージは 1 回のみ使用されるため、ネイティブ AQ ではこのキューを**単一コンシューマ・キュー**といいます。これは、このようなキューに関連付けられたコンシューマが 1 つのみという意味ではなく、各メッセージが 1 回のみ使用されるという意味です。

**JMS 用語** JMS 用語では、エンキューを**送信**といいます。メッセージに対するメッセージの宛先は、単なる**キュー**です。

## パブリッシュ / サブスクライブ・モデル

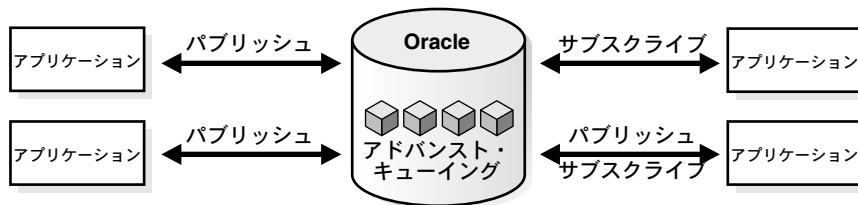
このモデルの各メッセージは、複数の受信者によって使用される場合があります。Point-to-Point メッセージ機能の単一ターゲット・モデルとは対照的に、パブリッシュ / サブスクライブ・モデルは、広範囲な送信を目標としています。ただし、パブリッシュ / サブスクライブ・モデルには、目標範囲がより狭いモード（マルチキャスト）および目標範囲がほとんどないモード（ブロードキャスト）もあります。

概念的に、ブロードキャストは、ある番組の聴取者を正確に知らないラジオ放送局と同じです。それとは対照的に、マルチキャストは、購読者を正確に把握している出版社と同じです。マルチキャストは、単一のパブリッシャが複数の受信者にメッセージを送信するため、「Point-to-Multipoint」ともいわれます。

**ネイティブ AQ 用語** このコンテキストでは、ブロードキャスト・メッセージは、**複数コンシューマ・キュー**に対する**サブスクライバ**として説明されます。メッセージのプロデューサは、メッセージをエンキューしますが、目標を絞ったモードのパブリック / サブスクライブは、**受信者のターゲット・リスト**として説明されます。この受信者は、交換メカニズムとして機能するキューのサブスクライバである場合と、そうでない場合があります。

**JMS 用語** JMS ではメッセージのコンテナは**トピック**であり、各アプリケーションが特定のトピックについて**パブリッシュ**、またはそのトピックを**サブスクライブ**できるようになっています。つまり、JMS のトピックは、ネイティブ AQ の**複数コンシューマ・キュー**へマップできます。oracle.jms は、定義済の受信者リストを使用するために、公開 JMS 標準を拡張します。

図 1-2 パブリッシュ / サブスクライブ・モデル



## アドバンスト・キューイング (AQ) の機能

トランザクション処理とキューイング・テクノロジーを統合することで、アドバンスト・キューイングという形の永続的なメッセージ機能が可能になりました。ここでは、次の 4 つの項に沿って、Oracle AQ の機能について説明します。

- 1-7 ページの「[一般機能](#)」
- 1-10 ページの「[ENQUEUE 機能](#)」
- 1-13 ページの「[DEQUEUE 機能](#)」
- 1-15 ページの「[伝播機能](#)」

### 一般機能

次の機能は、Oracle AQ 全体に当てはまります。

#### SQL アクセス

メッセージはデータベース表の標準の行に置かれるため、標準 SQL で問い合わせることができます。つまり、ユーザーは SQL を使用して、メッセージのプロパティ、履歴およびペイロードにアクセスできます。索引など、使用可能なすべての SQL テクノロジーを使用してメッセージへのアクセスを最適化できます。

#### 統合されたデータベース・レベルの操作サポート

リカバリ、再起動、Enterprise Manager などの標準データベース機能がサポートされます。Oracle AQ のキューは、データベース表内に実装されます。このため、高可用性、スケーラビリティおよび信頼性という操作上のすべての効果が、キュー・データにも適用されます。さらに、データベース開発および管理ツールでもキューを使用できます。たとえば、キュー・テーブルをインポートおよびエクスポートできます。

#### 構造化ペイロード

ユーザーは、オブジェクト型を使用してペイロードを構造化し管理できます。一般に RDBMS は、メッセージ・システムよりもはるかに豊富な型指定のシステムを備えています。Oracle8i はオブジェクト・リレーショナル DBMS であり、伝統的なリレーショナル型とユーザー定義型の両方をサポートします。強力な型指定を持つ内容（外部の型指定システムによって定義された形式を持つ内容）によって、次のような多くの高性能な機能が使用できます。

- 内容に基づくルーティング: 外部エージェントが内容を調査し、その内容に基づいてメッセージを別のキューにルーティングできます。

- 内容に基づくサブスクリプション: パブリッシュおよびサブスクライブ・システムはメッセージ・システム上に組み込まれており、サブスクリプションを基にした内容を提供できます。
- 問合せ: メッセージ内容の問合せ機能によって、メッセージ・ウェアハウスが可能です。

この機能を BooksOnLine のシナリオに適用した場合については、8-5 ページの「[構造化ベイロード](#)」を参照してください。

### 保存およびメッセージ履歴

AQ のユーザーは、処理済のメッセージを保存しておくように指定できます。システム管理者は、メッセージの保存期間を指定できます。Oracle AQ は、各メッセージの履歴情報を格納し、キューおよびメッセージのプロパティ（遅延、期限切れ、およびローカルまたはリモートの受信者に対するメッセージの保存）を保存します。この情報には、ENQUEUE/DEQUEUE 時刻、および各要求を実行したトランザクションの識別子が含まれます。これによって、ユーザーは関連するメッセージの履歴を保持できます。この履歴は、追跡、データ・ウェアハウスおよびデータ・マイニングの各操作で使用できます。

この機能を BooksOnLine のシナリオに適用した場合については、8-23 ページの「[保存およびメッセージ履歴](#)」を参照してください。

### 追跡およびイベント・ジャーナル

保存されているメッセージは、相互に関連付けられます。たとえば、メッセージ m1 を処理した結果、メッセージ m2 が生成された場合、m1 は m2 に関連付けられます。これによって、ユーザーは関連するメッセージの順序を追跡できます。これらの順序は「イベント・ジャーナル」を表しており、アプリケーションによって作成されることがよくあります。Oracle AQ では、アプリケーションが自動的にイベント・ジャーナルを作成できるように設計されています。

### 統合されたトランザクション

メッセージ内に制御情報と内容（データ・ペイロード）が統合されたことによって、アプリケーションの開発および管理が単純化されます。

### キュー・レベルのアクセス制御

Oracle8i では、8.1 形式のキューの所有者が、キュー・レベルの権限を付与したり取り消すことができます。DBA は、すべてのデータベース・ユーザーに、新しい AQ システム・レベル権限を付与したり取り消すことができます。また、DBA は、任意のデータベース・ユーザーを AQ 管理者にすることもできます。

この機能を BooksOnLine のシナリオに適用した場合については、8-10 ページの「[キュー・レベルのアクセス制御](#)」を参照してください。

## 非永続キュー

AQ では、サブスクライバに非永続性メッセージを非同期に配信できます。これらのメッセージはイベント駆動方式の可能性があり、システム（またはインスタンス）に障害が発生すると保持されません。AQ は、1 つの共通 API で永続および非永続メッセージをサポートします。

この機能を BooksOnLine のシナリオに適用した場合については、8-28 ページの「[Oracle Parallel Server \(OPS\) のサポート](#)」を参照してください。

## パブリッシュ / サブスクライブ・サポート

アプリケーション間でパブリッシュ / サブスクライブ・スタイルのメッセージ機能を実現するために、いくつかの機能が導入されています。この機能とは、ルールベースのサブスクライバ、メッセージ伝播、リスニング機能、通知機能などです。

## OPS 環境のサポート

Oracle8i では、アプリケーションでキュー・テーブルのインスタンス親和性を指定できます。AQ を Parallel Server および複数インスタンスで使用する場合、この情報を使用してキュー・モニター・スケジューリングのインスタンス間でキュー・テーブルがパーティション化されます。キュー・テーブルは、ユーザーが指定したインスタンスのキュー・モニターによって監視されます。インスタンス親和性が指定されないと、キュー・テーブルは、使用可能なインスタンス間で任意にパーティション化されることになります。キュー・テーブルにアクセスするアプリケーションとそれを監視するキュー・モニターの間で、「ping」が発生する可能性があります。インスタンス親和性を指定しても、そのアプリケーションが他のインスタンスから、キュー・テーブルとそのキューにアクセスできなくなることはありません。

この機能によって、キュー・モニターと別のインスタンスで実行している AQ 伝播ジョブの間で ping が発生しなくなります。Oracle8i では、キュー・テーブルに対してインスタンス親和性（プライマリおよびセカンダリ）を指定できます。AQ を Parallel Server および複数インスタンスで使用する場合、この情報を使用して、伝播用のインスタンスのみでなく、キュー・モニター・スケジューリング用のインスタンス間でもキュー・テーブルをパーティション化できます。キュー・テーブルは、常に 1 つのインスタンスと関連付けられています。明示的に指定された親和性がない場合、使用可能な任意のインスタンスがキュー・テーブルの所有者となる可能性があります。キュー・テーブルの所有者がなくなると、セカンダリ・インスタンスまたはいずれかの使用可能なインスタンスがキュー・テーブルの所有権を引き継ぎます。

この機能を BooksOnLine のシナリオに適用した場合については、8-28 ページの「[Oracle Parallel Server \(OPS\) のサポート](#)」を参照してください。

## 統計ビューのサポート

GV\$AQ ビューによって、データベース内のキューに関する基本的な統計が利用できます。

## 信頼性およびリカバリ可能性

キュー・データには、データベースの標準的な信頼性特性およびリカバリ可能性特性が適用されます。

## ENQUEUE 機能

次の機能は、メッセージをキューにエンキューすることでメッセージを生成するプロセスに適用されます。

### 関連識別子

ユーザーが各メッセージに識別子を割り当てることができるため、後で特定のメッセージを取り出せます。

### サブスクリプション・リストおよび受信者リスト

単一のメッセージを、複数のコンシューマによって使用するよう設計できます。キュー管理者は、キューからメッセージを取り出せるサブスクライバのリストを指定できます。異なるキューには異なるサブスクライバを指定でき、1つのコンシューマ・プログラムが複数のキューのサブスクライバになることもできます。さらに、キュー内の特定のメッセージに特定の受信者（そのキューのサブスクライバであってもなくても可）を指定することで、サブスクライバ・リストをオーバーライドできます。

単一のメッセージを複数のコンシューマが様々な方法で処理するように設計できます。メッセージの取出しを許可されたコンシューマが、メッセージをエンキューしたユーザーまたはアプリケーションによって、メッセージの明示的な受信者として指定されます。明示的に指定されたすべての受信者はエージェントで、名前、アドレスおよびプロトコルによって識別されます。

キュー管理者は、特定のキューのメッセージをすべて取り出せる受信者のデフォルト・リストを指定できます。これらの暗黙的な受信者は、デフォルト・リストに指定されていることでキューのサブスクライバになります。受信者を明示せずにエンキューされたメッセージは、指定されているすべてのサブスクライバに配信されます。

ルールベースのサブスクライバとは、デフォルトの受信者リストの中でルールを対応付けられている受信者をいいます。受信者が明示されていないメッセージは、対応付けられたルールの評価結果が TRUE のときにのみ、ルールベースのサブスクライバに送信されます。異なるキューには異なるサブスクライバを指定でき、同じ受信者が複数のキューのサブスクライバになることも可能です。さらに、キュー内の特定のメッセージに特定の受信者（そのキューのサブスクライバであってもなくても可）を指定することで、サブスクライバ・リストをオーバーライドできます。



受信者は名前のみで指定できますが、その場合、受信者はそのメッセージがエンキューされたキューからデキューする必要があります。プロトコルの値が0 (ゼロ) の場合、名前およびアドレスで受信者を指定できます。アドレスは、同じデータベースまたは (データベース・リンクによって識別された) 他の Oracle8i データベース中の別のキューの名前です。この場合、メッセージは指定されたキューに伝播され、指定された名前のコンシューマによってデキューできます。受信者の名前が NULL の場合、メッセージはアドレスに指定されたキューに伝播され、アドレスに指定されたキューのサブスクライバによってデキューされます。プロトコル・フィールドの値が0 (ゼロ) でない場合、名前およびアドレスのフィールドはシステムに無視され、メッセージは特定のコンシューマによってデキューできます (1-15 ページの「[伝播機能](#)」の「[サード・パーティ・サポート](#)」を参照)。

この機能を BooksOnLine のシナリオに適用した場合については、1-17 ページの「[アドバンスト・キューイング \(ネイティブ AQ\) の要素](#)」を参照してください。

## エンキューにおけるメッセージの優先順位および順序付け

エンキューされたメッセージの優先順位を指定できます。また、エンキューされたメッセージは、指定されたキュー内の正確な位置に置くことができます。つまり、ユーザーがメッセージの使用順序を指定するためのオプションが、3つあるということです。(a) ソート順序は、どのプロパティを使用してキューのすべてのメッセージを順序付けるかを指定します。(b) 優先順位は、各メッセージに割り当てられます。(c) 順序逸脱は、メッセージの位置を他のメッセージとの相対で指定します。さらに、複数のコンシューマが同一のキューを操作している場合、コンシューマは即時使用可能な最初のメッセージを取得します。別のコンシューマで処理中のメッセージはスキップされます。

この機能を BooksOnLine のシナリオに適用した場合については、8-37 ページの「[メッセージの優先順位および順序付け](#)」を参照してください。

## メッセージのグループ化

1つのキューに属しているメッセージをグループ化して1つのセットにし、一度に1ユーザーしか使用できないようにできます。このためには、メッセージのグループ化を使用可能にしたキュー・テーブルに、キューを作成する必要があります。1つのグループに属するメッセージは、すべて同一のトランザクションで作成される必要があります。また、1つのトランザクションで作成されるメッセージは、すべて同一のグループに属します。この機能によって、ユーザーは複雑なメッセージをセグメント化できます。たとえば、あるキュー宛てのメッセージに請求書情報が含まれている場合、そのメッセージは、ヘッダーのメッセージ、詳細情報のメッセージ、フッターのメッセージで構成されるグループとして作成できます。

この機能を BooksOnLine のシナリオに適用した場合については、8-51 ページの「[メッセージのグループ化](#)」を参照してください。

### 伝播

この機能によって、同じデータベースまたは同じキューに接続しなくても、アプリケーション同士が互いに通信できます。メッセージは、ローカルまたはリモートにかかわらず、ある Oracle AQ から別の Oracle AQ に伝播できます。伝播は、データベース・リンクおよび Net8 を使用して行われます。

この機能を BooksOnLine のシナリオに適用した場合については、8-76 ページの「[非同期通知](#)」を参照してください。

### 送信者識別

アプリケーションは、送信メッセージにユーザー定義の識別マークを付加できます。Oracle は、メッセージがデキューされたキューを自動的に識別することもできます。これによって、伝播されたメッセージや同じデータベース内の文字列メッセージを、アプリケーションが追跡できます。

### 時刻指定およびスケジューリング

エンキューされたメッセージに対して遅延間隔または期限切れ（あるいはその両方）を指定することで、実行枠を設定できます。メッセージをマークして、指定した時間（遅延時間）が経過しないと処理できないようにしたり、指定した期限が切れる前に必ず処理されるようにできます。

### ルールベースのサブスクライバ

メッセージは、そのプロパティや内容に基づいて、複数の受信者に配信できます。ユーザーは、所定のキューに対してルールベースのサブスクリプションを定義し、関心があるメッセージの受信希望を指定するメカニズムとして使用します。ルールは、メッセージ・プロパティおよび（オブジェクト型および RAW 型ペイロードの）メッセージ・データに基づいて指定できます。サブスクライバ・ルールを使用して、メッセージ配信の受信者を評価します。

この機能を BooksOnLine のシナリオに適用した場合については、8-55 ページの「[ルールベースのサブスクリプション](#)」を参照してください。

### 非同期式通知

OCI クライアントは、新しいコール `OCISubscriptionRegister` を使用して、メッセージ通知用のコールバックを登録できます。クライアントは、サブスクリプション名およびコールバックを指定する登録コールを発行します。そのサブスクリプションに対するメッセージを受信すると、コールバックが起動します。コールバックは、この後、メッセージを取り出す明示的なデキューを発行できます。

この機能を BooksOnLine のシナリオに適用した場合については、8-76 ページの「[非同期通知](#)」を参照してください。

## DEQUEUE 機能

### 複数の受信者

キューにある 1 つのメッセージを、複数のコピーを作成せずに複数の受信者が取り出せます。

この機能を BooksOnLine のシナリオに適用した場合については、8-60 ページの「[複数の受信者](#)」を参照してください。

### ローカルおよびリモートの受信者

受信者をローカルまたはリモート（あるいはその両方）のサイトに設定できます。

この機能を BooksOnLine のシナリオに適用した場合については、8-62 ページの「[ローカルおよびリモートの受信者](#)」を参照してください。

### デキューにおけるメッセージのナビゲーション

キューからメッセージを選択するにはいくつかの方法があります。最初のメッセージを選択することも、いったんメッセージを選択して位置を設定してから次のメッセージを選択することもできます。選択は順序付けに影響されたり、相関識別子の指定で限定されることがあります。また、メッセージ識別子を使用して特定のメッセージを取り出すこともできます。

この機能を BooksOnLine のシナリオに適用した場合については、8-64 ページの「[デキューにおけるメッセージ・ナビゲーション](#)」を参照してください。

### デキュー・モード

DEQUEUE 要求によって、メッセージの参照または取消しができます。メッセージは、参照後でも処理可能です。メッセージが取り消されると、そのメッセージに対する DEQUEUE 要求は無効になります。キューのプロパティによっては、取り消されたメッセージがキュー・テーブルに保持されることもあります。

この機能を BooksOnLine のシナリオに適用した場合については、8-68 ページの「[デキューのモード](#)」を参照してください。

## メッセージ到着待機の最適化

空のキューに対して DEQUEUE を発行できます。新しいメッセージの到着を確認するポーリングを回避するために、要求がメッセージ到着を待機できるようにするかどうか、およびその待機時間をユーザーが指定できます。

この機能を BooksOnLine のシナリオに適用した場合については、8-74 ページの「[メッセージ到着待機の最適化](#)」を参照してください。

## 遅延を伴う再試行

メッセージは、厳密に 1 回しか使用できません。メッセージをデキューしようとして失敗し、トランザクションがロールバックされた場合、ユーザーが指定した遅延が終了した後で、そのメッセージの再処理が可能になります。再処理は、ユーザーが指定した回数まで試行されます。

この機能を BooksOnLine のシナリオに適用した場合については、8-83 ページの「[遅延間隔をおいての再試行](#)」を参照してください。

## トランザクション保護のオプション

ENQUEUE/DEQUEUE 要求は、通常は複数の要求を含むトランザクションの一部として、必要なトランザクション動作を実現しています。ただし、特定の要求をそれ自身でトランザクションに指定して、その要求の結果をすぐに他のトランザクションから参照できるようにできます。つまり、ENQUEUE 文または DEQUEUE 文が発行されるとすぐに、またはそのトランザクションがコミットされた直後に、メッセージを外部から参照できるようにできます。

## 例外処理

メッセージは、たとえば実行環境の状態や再試行回数の制限など、所定の制約内では処理できない可能性があります。このような状況が発生すると、メッセージはユーザーが指定した例外キューに移されます。

この機能を BooksOnLine のシナリオに適用した場合については、8-87 ページの「[例外処理](#)」を参照してください。

## リスニング機能（複数キューの待機）

リスニング・コールは、複数キュー上でのメッセージの受取りを待機することができるブロック・コールです。このコールは、ゲートウェイ・アプリケーションによって一連のキューを監視するために使用できます。アプリケーションは、サブスクリプション・リスト上のメッセージを待機する目的でもリスニングを使用できます。リスニングが成功したら、デキューによってメッセージを取り出す必要があります。

この機能を BooksOnLine のシナリオに適用した場合については、8-97 ページの「[Listen 機能](#)」を参照してください。

## ペイロードを伴わないメッセージ・ヘッダーのデキュー

新しいデキュー・モードである REMOVE\_NODATA は、ペイロードを取り出さずにキューからメッセージを取り出すときに使用できます。このモードは、ペイロードが大きいメッセージを削除する場合、またはペイロードの内容に関心がないアプリケーションの場合に便利です。

## 伝播機能

### エンキューおよびデキューの自動調整

前述のように、受信者はローカルでもリモートでもかまいません。Oracle8i では、分散されたオブジェクト型がサポートされていないため、標準データベース・リンクを使用したりリモートのエンキューまたはデキューは正常に処理されません。ただし、AQ のメッセージ伝播を使用すれば、リモート・キューにエンキューできます。

たとえば、データベース X に接続して、そこにある DROPBOX キューにメッセージをエンキューできます。また、AQ を構成することで、DROPBOX キューにエンキューされたすべてのメッセージを、(ローカルまたはリモートにかかわらず) データベース Y の別のキューに自動的に伝播できます。AQ によって、データベース Y のリモート・キューのタイプが、データベース X のローカル・キューのタイプと構造的に同一であるかが自動的にチェックされ、メッセージが伝播されます。

伝播されたメッセージの受信者は、アプリケーションまたはキューのどちらかです。受信者がキューの場合、実際の受信者は、受信者キューのサブスクリプション・リストによって決まります。キューがリモートの場合、メッセージは指定したデータベース・リンクを使用して伝播されます。サポートされるのは、AQ から AQ へのメッセージ伝播のみです。

### LOB を伴うメッセージの伝播

伝播は LOB 属性を持つペイロードを処理します。

この機能を BooksOnLine のシナリオに適用した場合については、8-107 ページの「[サンプル・シナリオ](#)」を参照してください。

### 伝播スケジュール

メッセージは、キューからローカルまたはリモートの受信者に伝播するようにスケジュールできます。管理者は、開始時刻、伝播枠および次の伝播枠を決定する関数 (定期的なスケジュールの場合) を指定できます。

## 伝播スケジューリング機能の拡張

伝播に関する詳細なランタイム情報が収集され、伝播スケジュールごとに DBA\_QUEUE\_SCHEDULES ビューに格納されます。この情報はキューの設計者および管理者によって、問題の解決やパフォーマンス・チューニングのために使用できます。たとえば、伝播されたメッセージ数またはバイト数の合計および平均の詳細情報が、スケジュール調整に使用できます。同様に、ビューによって報告されたエラーは、問題の診断および解決に使用できます。ビューには、伝播処理をしたセッションの ID、ジョブ・キュー・プロセスの名前などの追加情報も示されます。

この機能を BooksOnLine のシナリオに適用した場合については、8-109 ページの「[拡張伝播スケジュール機能](#)」を参照してください。

## サード・パーティ・サポート

アドバンスト・キューイングでは、エンキューされたキューから、サード・パーティ製のプロパゲータによって異なるメッセージ機能システムにメッセージが伝播されることが可能です。受信者のプロトコル番号が 128 ～ 255 の範囲にある場合、AQ は受信者のアドレスを無視するため、メッセージがアドバンスト・キューイング・システムによって伝播されることはありません。かわりに、サード・パーティ製のプロパゲータが、コンシューマ名として確保されている名前をデキュー操作に指定して、メッセージをデキューできます。確保されているコンシューマ名は、AQ\$\_P#（# は 128 ～ 255 のプロトコル番号）形式です。たとえば、AQ\$\_P128 というコンシューマ名は、プロトコル番号 128 の受信者に対してメッセージをデキューするために使用されます。特定のプロトコル番号を伴うメッセージに対する受信者リストは、デキュー時に recipient\_list メッセージ・プロパティに戻されます。

## アドバンスト・キューイング（ネイティブ AQ）の要素

トランザクション処理とキューイング・テクノロジーを統合することで、アドバンスト・キューイングという形の永続的なメッセージ機能が可能になりました。

### メッセージ

メッセージは、キューに挿入され、そこから取り出される情報の最小単位です。メッセージは、次の2つの要素で構成されます。

- 制御情報（メタデータ）
- ペイロード（データ）

制御情報は、AQ がメッセージの管理に使用するメッセージ・プロパティを表します。ペイロード・データはキューに格納される情報で、Oracle AQ に対して透過的です。1つのメッセージは、1つのキューにのみ常駐できます。メッセージは、エンキュー・コールによって作成され、デキュー・コールによって処理されます。

### キュー

キューは、メッセージのリポジトリです。キューには、ユーザー・キュー（標準キュー）および例外キューの2種類があります。ユーザー・キューは、通常のメッセージ処理に使用されます。なんらかの理由で取り出して処理できないメッセージは、例外キューに転送されます。キューは、Oracle AQ 管理インタフェースを使用して作成、変更、起動、停止および削除できます（第9章「[管理インタフェース](#)」を参照）。

### キュー・テーブル

キューは、キュー・テーブルに格納されます。キュー・テーブルはデータベース表で、1つ以上のキューを含みます。各キュー・テーブルには、デフォルト例外キューがあります。7-2 ページの図 7-1「[基本キュー](#)」に、メッセージ、キューおよびキュー・テーブル間の関連を示します。

### エージェント

エージェントは、キューのユーザーです。エンド・ユーザーまたはアプリケーションがエージェントになります。エージェントには、次の2種類があります。

- メッセージをキューに入れる（エンキューする）プロデューサ
- メッセージを取り出す（デキューする）コンシューマ

任意の時点でキューにアクセスできるプロデューサおよびコンシューマの数に、制限はありません。エージェントは、Oracle AQ 操作インタフェースを使用して、メッセージをキューに挿入したり取り出したりできます（第 11 章「[操作インタフェース：基本操作](#)」を参照）。

エージェントは、名前、アドレスおよびプロトコルで識別されます（このデータ構造の詳細は、2-3 ページの「[エージェント](#)」を参照）。

- エージェントの名前には、アプリケーションの名前またはアプリケーションによって割り当てられた名前を使用できます。後半で説明するとおり、キュー自体がエージェントになって、別のキューにエンキューまたはデキューすることがあります。
- アドレス・フィールドは、最大 1024 バイトの文字フィールドで、プロトコルに関連して解釈されます。たとえば、プロトコルのデフォルト値は 0（ゼロ）で、データベース・リンクのアドレスを示します。この場合、このプロトコルのアドレスは次のようになります。

```
queue_name@dblink
```

ここで queue\_name の形式は [schema.]queue で、dblink は完全に修飾されたデータベース・リンク名、またはドメイン名がないデータベース・リンク名のどちらかです。

## 受信者

メッセージの受信者は名前のみで指定できますが、その場合、受信者はそのメッセージがエンキューされたキューからデキューする必要があります。プロトコルの値が 0（ゼロ）の場合、名前およびアドレスで受信者を指定できます。アドレスは、同じデータベースまたは（データベース・リンクによって識別された）他の Oracle8 データベース中の別のキューの名前です。この場合、メッセージは指定されたキューに伝播され、指定された名前のコンシューマによってデキューできます。受信者の名前が NULL の場合、メッセージはアドレスに指定されたキューに伝播され、アドレスに指定されたキューのサブスクライバによってデキューされます。プロトコル・フィールドの値が 0（ゼロ）でない場合、名前およびアドレスのフィールドはシステムに無視され、メッセージは特定のコンシューマによってデキューできます（1-15 ページの「[伝播機能](#)」の「[サード・パーティ・サポート](#)」を参照）。

## 受信者およびサブスクリプション・リスト

単一のメッセージを複数のコンシューマによって使用するように設計できます。これには 2 通りの方法があります。



- エンキュー元は、メッセージの受信者としてメッセージを取り出せるコンシューマを明示的に指定できます。受信者はエージェントで、名前、アドレスおよびプロトコルによって識別されます。
- キュー管理者は、キューからメッセージを取り出せる受信者のデフォルト・リストを指定できます。デフォルト・リストで指定される受信者を、サブスクライバといいます。受信者の指定がないメッセージがエンキューされると、そのメッセージはすべてのサブスクライバに送信されます。

異なるキューには異なるサブスクライバを指定でき、同じ受信者が複数のキューのサブスクライバになることも可能です。さらに、キュー内の特定のメッセージに特定の受信者（そのキューのサブスクライバであってもなくても可）を指定することで、サブスクライバ・リストをオーバーライドできます。

## ルール

ルールは、サブスクライバがメッセージをサブスクライブする際に、持っている1つ以上の関心事項を定義するために使用します。この基準に合ったメッセージが、関心を持っているサブスクライバに配信されます。言い換えると、ルールは、キューのメッセージに対して、サブスクライバが関心を持っているトピックでフィルタ処理を実行します。

ルールは、SQL 問合せの WHERE 句に似た構文を使用するブール式（TRUE または FALSE という値を持つ）で定義されます。このブール式には、次のような条件を組み込むことができます。

- メッセージ・プロパティ（現行では、priority および相関識別子）
- ユーザー・データ・プロパティ（オブジェクト・ペイロードのみ）
- 関数（SQL 問合せの WHERE 句において指定される）

## ルールベースのサブスクライバ

ルールベースのサブスクライバとは、デフォルトの受信者リストでルールが対応付けられているサブスクライバです。明示的な受信者が指定されていないメッセージは、対応付けられたルールの評価結果が TRUE のときに、ルールベースのサブスクライバに送信されます。

## キュー・モニター

キュー・モニター（QMn）は、キュー内のメッセージを監視するバックグラウンド・プロセスです。キュー・モニターでは、メッセージの遅延処理、期限切れおよび再試行遅延の機能を提供します。QMn は、キュー・テーブルおよびその索引と索引構成表のガベージ・コレクションも行います。同時に開始できるキュー・モニターは、最大で 10 個です。動的 `init.ora` パラメータ `aq_tm_processes` を設定して、必要な数のキュー・モニターを開始します。キュー・モニターは、毎分、またはメッセージに期限切れや処理準備完了というマークを付加するという作業があるときに動作します。

# デモの参照

次のデモが、\$ORACLE\_HOME/demo ディレクトリにあります。

表 1-1

デモおよびその場所	トピック
newaqdemo00.sql	ユーザー、メッセージ型、表などの作成
newaqdemo01.sql	キュー・テーブル、キュー、サブスクライバおよびセットアップの設定
newaqdemo02.sql	メッセージのエンキュー
newaqdemo03.sql	デキュー・プロシージャのインストール
newaqdemo04.sql	「ブロッキング・デキュー」の実行
newaqdemo05.sql	複数エージェントの「リスニング」の実行
newaqdemo06.sql	ユーザー、キュー・テーブル、キュー、サブスクライバなどのクリーン・アップ (cleanup スクリプト)
ociaqdemo00.c	メッセージのエンキュー
ociaqdemo01.c	「ブロッキング・デキュー」の実行
ociaqdemo02.c	複数エージェントの「リスニング」の実行

---

## 基本的なコンポーネント

この章では、次の基本的なコンポーネントについて説明します。

- [データ構造](#)
- [管理インタフェースの列挙定数](#)
- [操作インタフェースの列挙定数](#)
- [Java コンポーネント - oracle.AQ](#)

## データ構造

この章で説明するデータ構造は、次の操作インタフェースおよび管理インタフェースで 사용됩니다。

- [第 9 章「管理インタフェース」](#)
- [第 11 章「操作インタフェース: 基本操作」](#)

## オブジェクト名

### 用途

データベース・オブジェクトに名前を付けます。このネーミング規則は、キュー、キュー・テーブルおよびオブジェクト型に適用されます。

### 構文

```
object_name := VARCHAR2  
object_name := [<schema_name>.]<name>
```

### 使用方法

オブジェクト名は、オプションのスキーマ名および名前で指定します。スキーマ名を指定しない場合は、現行のスキーマが想定されます。名前は、『Oracle8i SQL リファレンス』の予約文字に関する説明のオブジェクト名のガイドラインに従う必要があります。スキーマ名、エージェント名およびオブジェクト型名は、それぞれ 30 バイトまで可能です。ただし、キュー名およびキュー・テーブル名は最大 24 バイトまで可能です。

## 型名

### 用途

キューの型を定義します。

### 構文

```
type_name := VARCHAR2  
type_name := <object_type> | "RAW"
```

使用方法

表 2-1 型名

パラメータ	説明
<object_types>	オブジェクト型作成の詳細は、サーバー概念のマニュアルを参照してください。オブジェクト型の属性数は最大で 900 です。
"RAW"	RAW 型のペイロードを格納するために、AQ ではペイロード・リポジトリとして LOB 列を持つキュー・テーブルを作成します。ペイロードのデータ・サイズは、最大で 32KB です。LOB 列は RAW ペイロードの格納に利用されるため、AQ 管理者は、キュー・テーブルの作成時に LOB 表領域を選択し、storage 句パラメータに LOB 記憶域文字列を記述することで、LOB 記憶域を構成できます。

エージェント

用途

メッセージのプロデューサまたはコンシューマを識別します。

構文

```
TYPE aq$_agent IS OBJECT (  
    name          VARCHAR2 (30) ,  
    address       VARCHAR2 (1024) ,  
    protocol      NUMBER)
```

使用方法

表 2-2 エージェント

パラメータ	説明
name (VARCHAR2 (30))	プロデューサまたはコンシューマの名前。名前は、『Oracle8i SQL リファレンス』の予約文字に関する説明のオブジェクト名のガイドラインに従う必要があります。
address (VARCHAR2 (1024))	受信者のプロトコル固有のアドレス。プロトコルが 0 (デフォルト) の場合、アドレスの形式は [schema.]queue [@dblink] です。
protocol (NUMBER)	アドレスを解析してメッセージを伝播するプロトコル。デフォルト値は 0 (ゼロ) です。

## 使用上の注意

複数コンシューマ・キューにサブスクライバとして追加されたすべてのコンシューマは、AQ\$\_AGENT パラメータに一意の値を持つ必要があります。すなわち、2つのサブスクライバがAQ\$\_AGENT 型の NAME、ADDRESS および PROTOCOL 属性に同じ値を持つことはできません。この3つの属性のうち少なくとも1つは、2つのサブスクライバに対して異なる値にする必要があります。

## AQ 受信者リスト型

### 用途

メッセージを受信するエージェントのリストを識別します。

### 構文

```
TYPE aq$_recipient_list_t IS TABLE OF aq$_agent  
    INDEX BY BINARY_INTEGER;
```

## AQ エージェント・リスト型

### 用途

DBMS\_AQ.LISTEN によってリスニングするエージェントのリストを識別します。

### 構文

```
TYPE aq$_agent_list_t IS TABLE OF aq$_agent  
    INDEX BY BINARY_INTEGER;
```

## AQ サブスクライバ・リスト型

### 用途

このキューにサブスクライブするサブスクライバのリストを識別します。

### 構文

```
TYPE aq$_subscriber_list_t IS TABLE OF aq$_agent  
    INDEX BY BINARY_INTEGER;
```

# 管理インタフェースの列挙定数

INFINITE、TRANSACTIONAL、NORMAL\_QUEUE などの列挙定数を使用する場合、定数はそれを定義するパッケージの有効範囲とともに指定する必要があります。管理インタフェースに関連付けられるすべての型には、前に DBMS\_AQADM を付ける必要があります。たとえば、次のとおりです。

```
DBMS_AQADM.NORMAL_QUEUE
```

表 2-3 管理インタフェースの列挙型

パラメータ	オプション
retention	0、1、2、...INFINITE
message_grouping	TRANSACTIONAL、NONE
queue_type	NORMAL_QUEUE、EXCEPTION_QUEUE、NON_PERSISTENT_QUEUE

# 操作インタフェースの列挙定数

BROWSE、LOCKED、REMOVE などの列挙定数を使用する場合、PL/SQL 定数はそれを定義するパッケージの有効範囲とともに指定する必要があります。管理インタフェースに関連付けられるすべての型には、前に DBMS\_AQ を付ける必要があります。たとえば、次のとおりです。

```
DBMS_AQ.BROWSE
```

表 2-4 操作インタフェースの列挙型

パラメータ	オプション
visibility	IMMEDIATE、ON_COMMIT
dequeue mode	BROWSE、LOCKED、REMOVE、REMOVE_NODATA
navigation	FIRST_MESSAGE、NEXT_MESSAGE、NEXT_TRANSACTION
state	WAITING、READY、PROCESSED、EXPIRED
sequence_deviation	BEFORE、TOP
wait	FOREVER、NO_WAIT
delay	NO_DELAY
expiration	NEVER



## 問題および考慮点

### INIT.ORA パラメータ

#### AQ\_TM\_PROCESSES

キュー・メッセージに対して時間監視を行う場合は、init.ora パラメータ・ファイルに AQ\_TM\_PROCESSES パラメータを指定する必要があります。これは、遅延および期限切れのプロパティを指定されたメッセージに対して使用されます。このパラメータは、0（ゼロ）～10 の範囲内で設定できます。それ以外の数値を設定すると、エラーになります。このパラメータを1にすると、メッセージを監視するためのバックグラウンド・プロセスとして、キュー・モニター・プロセス（QMN）が1つ作成されます。このパラメータが未指定の場合、または0（ゼロ）にした場合、キュー・モニター・プロセスは作成されません。

パラメータ名:	aq_tm_processes
パラメータ型:	整数
パラメータ・クラス:	動的
設定可能値:	0 ～ 10
構文:	aq_tm_processes = <0 to 10>
プロセス名:	ora_qmn<n>_<oracle sid>
例:	aq_tm_processes = 1

#### JOB\_QUEUE\_PROCESSES

伝播は、ジョブ・キュー（SNP）プロセスによって処理されます。あるインスタンスで開始されるジョブ・キュー・プロセスの数は、init.ora ファイルの JOB\_QUEUE\_PROCESSES パラメータで制御されます。このパラメータのデフォルト値は0（ゼロ）です。メッセージを伝播するには、このパラメータを1以上にする必要があります。次のような場合、DBAはこのパラメータをより高い値にできます。伝播が必要なメッセージを持っているキューが多数ある場合、伝播するメッセージの宛先が多数ある場合、またはジョブ・キューに他のジョブがある場合です。

**参照:** JOB\_QUEUE\_PROCESSES の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

Java AQ API は、Oracle AQ（アドバンスト・キューイング）の管理機能および操作機能の両方をサポートします。メッセージ機能アプリケーション用の Java プログラムを開発するときは、JDBC を使用してデータベースへの接続をオープンしてから、Java AQ API である oracle.AQ でメッセージ・キューイングを行います。これは、PL/SQL インタフェースを使用する必要がないことを意味します。

## Java コンポーネント - oracle.AQ

次の項では、現在の PL/SQL インタフェースに基づく Java の共通インタフェースおよびクラスを説明します。

- 共通インタフェースには「**AQ**」という接頭辞が付いています。これらのインタフェースは、Oracle8i および Oracle Lite では異なる実装になります。
- このマニュアルでは、この共通インタフェースおよびこれに対応する Oracle8i の実装を説明しますが、これには「**AQOracle**」という接頭辞が付きます。

## Java AQ クラスの場所

Java AQ クラスは、`$ORACLE_HOME/rdbms/jlib/aqapi.jar` にあります。これらのクラスは、Oracle8i のすべての JDBC ドライバで使用できます。

アプリケーションで OCI8 または Thin JDBC Driver を使用する場合は、JDK 1.2 では、CLASSPATH に `$ORACLE_HOME/rdbms/jlib/aqapi.jar` を指定する必要があります。また、JDK 1.1 では、CLASSPATH に `$ORACLE_HOME/rdbms/jlib/aqapi11.jar` を指定する必要があります。

アプリケーションで Oracle Server ドライバを使用して、Java ストアド・プロシージャから Java AQ API にアクセスする場合、Java ファイルは、通常、Java 対応のデータベースで自動的に事前にロードされます。ロードされていない場合は、まず `loadjava` ユーティリティを使用して、データベースに `aqapi.jar` および `jmscommon.jar` ファイルをロードする必要があります。

[付録 A「Oracle アドバンスト・キューイングの使用例」](#)には、次の例が含まれています。

- Java を使用したオブジェクト型メッセージ（CustomDatum インタフェース）のエンキューおよびデキュー
- Java を使用したオブジェクト型メッセージ（SQLData インタフェース）のエンキューおよびデキュー
- Java を使用したキュー・テーブルおよびキューの作成
- Java を使用したキュー作成およびエンキュー / デキューの開始
- Java を使用した複数コンシューマ・キューの作成およびサブスクライバの追加
- Java を使用した RAW メッセージのエンキュー
- Java を使用したメッセージのデキュー
- Java を使用した参照モードでのメッセージのデキュー
- Java を使用した優先順位によるメッセージのエンキュー

- Java を使用した、LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー

test\_aqjava クラスのセットアップは、『Oracle8i Java パッケージ・プロシージャ リファレンス』の「oracle.AQ の設定例」を参照してください。



---

## AQ プログラム環境

この章では、操作が必要な要素、およびアプリケーション環境を準備するときに考慮する必要がある問題点を説明します。内容は次のとおりです。

- AQ にアクセスするためのプログラム環境
- PL/SQL (DBMS\_AQADM および DBMS\_AQ パッケージ) を使用した AQ へのアクセス
- Visual Basic (OO4O) を使用した AQ へのアクセス
- OCI を使用した AQ へのアクセス
- AQ Java (oracle.AQ) クラスを使用した AQ へのアクセス
- Oracle Java Message Service (JMS) を使用した AQ へのアクセス
- AQ プログラム環境の比較

# AQ にアクセスするためのプログラム環境

Oracle8i のアドバンスト・キューイング機能へのアクセスには、次のプログラム環境が使用されます。

- **ネイティブ AQ インタフェース**
  - PL/SQL (DBMS\_AQADM および DBMS\_AQ パッケージ) : 管理機能および操作機能をサポートします。
  - C (OCI) : 操作機能をサポートします。
  - Visual Basic (OO4O) : 操作機能をサポートします。
  - Java (JDBC を使用する oracle.AQ パッケージ) : 管理および操作機能をサポートします。
- **AQ への JMS インタフェース**
  - Java (JDBC を使用する javax.jms および oracle.jms パッケージ) : 標準 JMS 管理機能と操作機能、および Oracle の JMS 拡張機能をサポートします。

これらのプログラム環境で使用可能な機能を比較するには、次の表を参照してください。

- [表 3-2 「AQ プログラム環境の比較: 管理インタフェース」](#)
- [表 3-3 「AQ プログラム環境の比較: 操作インタフェース」](#)

[表 3-1 「AQ プログラム環境」](#) に、AQ プログラム環境およびその構文の参照先を示します。

表 3-1 AQ プログラム環境

言語	プリコンパイラまたは インタフェース・プログラム	構文の参照先	この章にある参照先
PL/SQL	DBMS_AQADM および DBMS_AQ パッケージ	『Oracle8i PL/SQL パッケージ・プロ シージャ リファレンス』	3-4 ページの「 <a href="#">PL/SQL (DBMS_AQADM および DBMS_AQ パッケージ) を使用した AQ へのアクセス</a> 」
C	Oracle コール・インタ フェース (OCI)	『Oracle8i コール・インタフェ ース・プログラマーズ・ガイド』	3-7 ページの「 <a href="#">OCI を使用した AQ へのアクセス</a> 」

表 3-1 AQ プログラム環境 (続き)

言語	ブリコンバイラまたは インタフェース・プロ グラム	構文の参照先	この章にある参照先
Visual Basic	Oracle Objects for OLE (OO4O)	Oracle Objects for OLE (OO4O) は、Oracle8i Client for Windows NT に含まれている Windows ベー スの製品です。  この製品については、オンライン・ ヘルプのみが提供されています。オ ンライン・ヘルプは、Oracle8i イン ストール時の「Application Development」サブメニューから使 用可能です。	3-8 ページの「 <a href="#">AQ Java (oracle.AQ) クラスを使用した AQ へのアクセス</a> 」 3-6 ページの「 <a href="#">Visual Basic (OO4O) を使用した AQ へのアクセス</a> 」
Java (AQ)	JDBC API を介しての oracle.AQ パッケー ジ	『Oracle8i Java パッケージ・プロ シージャ リファレンス』	3-8 ページの「 <a href="#">AQ Java (oracle.AQ) クラスを使用した AQ へのアクセス</a> 」
Java (JMS)	JDBC API を介しての oracle.JMS パッケー ジ	『Oracle8i Java パッケージ・プロ シージャ リファレンス』	3-8 ページの「 <a href="#">AQ Java (oracle.AQ) クラスを使用した AQ へのアクセス</a> 」 3-10 ページの「 <a href="#">Oracle Java Message Service (JMS) を使用した AQ へのア クセス</a> 」

## PL/SQL (DBMS\_AQADM および DBMS\_AQ パッケージ) を使用した AQ へのアクセス

PL/SQL パッケージ (DBMS\_AQADM および DBMS\_AQ) は、ネイティブ AQ インタフェースを介した Oracle8i アドバンスト・キューイング管理機能および操作機能へのアクセスをサポートします。

次の機能が含まれます。

- キュー、キュー・テーブル、非永続キュー、複数コンシューマ・キュー / トピック、RAW メッセージ、構造化データを持つメッセージの作成
- キュー・テーブル、キュー、複数コンシューマ・キュー / トピックの取得
- キュー・テーブル、キュー / トピックの変更
- キュー / トピックの削除
- キュー / トピックの開始または停止
- 権限の付与および取消し
- サブスクライバの追加、削除、変更
- 伝播スケジュールの使用可能化、使用禁止化および変更
- 単一コンシューマ・キュー (Point-to-Point モデル) へのメッセージのエンキュー
- 複数コンシューマ・キュー / トピック (パブリッシュ / サブスクライブ・モデル) へのメッセージの公開
- 複数コンシューマ・キューのメッセージに対するサブスクライブ
- キューのメッセージのブラウズ
- キュー / トピックからのメッセージの受信
- 複数キュー / トピック上のメッセージのリスニング

### 参照:

パラメータ、パラメータ・タイプ、戻り値、例、DBMS\_AQADM、DBMS\_AQ 構文などの詳細は、『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』を参照してください。

PL/SQL の DBMS\_AQADM および DBMS\_AQ の使用可能な機能の詳細は、次の表を参照してください。



- 表 3-2「AQ プログラム環境の比較: 管理インタフェース」
- 表 3-3「AQ プログラム環境の比較: 操作インタフェース」

## Visual Basic (OO4O) を使用した AQ へのアクセス

Visual Basic (OO4O) は、ネイティブ AQ インタフェースを介した Oracle8i アドバンスト・キューイング操作機能へのアクセスをサポートします。

次の機能が含まれます。

- 接続、RAW メッセージ、構造化データを持つメッセージの作成
- 単一コンシューマ・キュー (Point-to-Point モデル) へのメッセージのエンキュー
- 複数コンシューマ・キュー / トピック (パブリッシュ / サブスクライブ・モデル) へのメッセージの公開
- キューのメッセージのブラウズ
- キュー / トピックからのメッセージの受信
- メッセージの非同期受信登録

使用可能な Visual Basic (OO4O) 機能の詳細は、次の表を参照してください。

- [表 3-2 「AQ プログラム環境の比較: 管理インタフェース」](#)
- [表 3-3 「AQ プログラム環境の比較: 操作インタフェース」](#)

## 詳細情報

OO4O の詳細は、次の Web サイトを参照してください。

- <http://technet.oracle.com>  
Products > Internet Tools > Programmer を順に選択し、Oracle Objects for OLE にスクロールします。ページ最下部には、インタフェースを使用する場合に有効な記事のリストがあります。
- <http://www.oracle.com/products>  
OO4O または Oracle Objects for OLE に関する記事を検索します。

## OCI を使用した AQ へのアクセス

Oracle コール・インタフェース (OCI) は、ネイティブ AQ インタフェースを介した Oracle8i アドバンスト・キューイング機能へのインタフェースを提供します。

OCI クライアントは、次のアクションを実行できます。

- メッセージのエンキュー
- メッセージのデキュー
- 複数キュー上のメッセージのリスニング

さらに、OCI クライアントは、OCISubscriptionRegister を使用して、キューの新しいメッセージの非同期通知を受信できます。

**参照：** 構文の詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の「OCI およびアドバンスト・キューイング」および「パブリッシュ / サブスクライブの通知」の項を参照してください。

ユーザー定義のパイロード型を持つキューでは、Oracle 型の OCI マッピングを生成するために、OTT を使用する必要があります。OCI クライアントは、AQ 記述子のメモリーおよびメッセージ・パイロードを解放する必要があります。

## 例

### OCI インタフェース

OCI アドバンスト・キューイング・インタフェースの例は、A-11 ページの「[メッセージのエンキューおよびデキュー](#)」を参照してください。

### OCI 記述子メモリーの管理

OCI 記述子のメモリー管理の例は、A-73 ページの「[AQ およびメモリーの使用](#)」を参照してください。

## AQ Java (oracle.AQ) クラスを使用した AQ へのアクセス

Java AQ API は、Oracle AQ (アドバンスト・キューイング) の管理機能および操作機能の両方をサポートします。メッセージ機能アプリケーション用の Java プログラムを開発するときは、JDBC を使用してデータベースに接続してから、Java AQ API である `oracle.AQ` でメッセージ・キューイングを行います。これは、PL/SQL インタフェースを使用する必要がないことを意味します。

現行の PL/SQL インタフェースに基づく共通インタフェースおよびクラスについては、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

- 共通インタフェースには「AQ」という接頭辞が付いています。これらのインタフェースは、Oracle8i および Oracle Lite では異なる実装になります。
- このマニュアルでは、この共通インタフェースおよびこれに対応する Oracle8i の実装を説明します。

## Java AQ クラスへのアクセス

Java AQ クラスは、`$ORACLE_HOME/rdbms/jlib/aqapi.jar` にあります。これらのクラスは、Oracle8i のすべての JDBC ドライバで使用できます。

- **OCI8 または Thin JDBC Driver の使用:** ご使用のアプリケーションが OCI8 または Thin JDBC Driver を使用する場合、次のように指定する必要があります。
  - JDK 1.2 の場合、CLASSPATH に `$ORACLE_HOME/rdbms/jlib/aqapi.jar`
  - JDK 1.1 の場合、CLASSPATH に `$ORACLE_HOME/rdbms/jlib/aqapi11.jar`
- **JServer での Oracle Server ドライバの使用:** このアプリケーションが、Oracle Server ドライバを使用して、Java ストアド・プロシージャから Java AQ API にアクセスする場合、Java ファイルは、通常、Java 対応のデータベースに自動的に事前にロードされます。Java ファイルがロードされない場合、まず `loadjava` ユーティリティを使用して、`jmscommon.jar` および `aqapi.jar` ファイルをデータベースにロードする必要があります。

## アドバンスト・キューイングの例

付録 A「Oracle アドバンスト・キューイングの使用例」には、次の例が含まれています。

- Java を使用したオブジェクト型メッセージ (CustomDatum インタフェース) のエンキューおよびデキュー

- Java を使用したオブジェクト型メッセージ (SQLData インタフェース) のエンキューおよびデキュー
- Java を使用したキュー・テーブルおよびキューの作成
- Java を使用したキュー作成およびエンキュー / デキューの開始
- Java を使用した複数コンシューマ・キューの作成およびサブスクライバの追加
- Java を使用した RAW メッセージのエンキュー
- Java を使用したメッセージのデキュー
- Java を使用したブラウズ・モードでのメッセージのデキュー
- Java を使用した優先順位によるメッセージのエンキュー
- Java を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー

## Java AQ API の管理

Java AQ API の様々な実装は、AQDriverManager を介して管理されます。Olite と Oracle8i の両方には、AQDriverManager に登録されている AQDriver があります。ドライバ・マネージャを使用して、メッセージ・タスクの実行に使用できる AQSession を作成します。

Oracle8i AQ ドライバは、Class.forName ("oracle.AQ.AQOracleDriver") コマンドによって登録されます。

AQDriverManager.createAQSession() が起動されると、このメソッドは、createAQSession() コールに渡されたパラメータに基づいて、(登録済ドライバの中から) 該当する AQDriver をコールします。

Oracle8i AQDriver では、AQSession を作成するパラメータとして、有効な JDBC 接続が渡されるものと想定します。ユーザーが AQ Java インタフェースを使用するには、DBMS\_AQIN パッケージに対する実行権限が必要です。この権限は、AQ\_USER\_ROLE または AQ\_ADMINISTRATOR\_ROLE を介しても取得できます。また、ユーザーには、8.1 形式のキュー・テーブルに対する適切なシステム権限およびキュー権限も必要です。

## Oracle Java Message Service (JMS) を使用した AQ へのアクセス

**Java Message Service (JMS)** : Java Message Service (JMS) は、オラクル社、IBM 社およびその他のベンダーとともにサン・マイクロシステムズ社が定義したメッセージ機能標準です。JMS は、JMS クライアントが企業のメッセージ関連製品の機能にアクセスする方法を定義する、インタフェースおよび対応するセマンティクスの集合です。

**Oracle Java Message Service (Oracle JMS)** : Oracle Java Message Service は、JMS 標準に基づいて、Oracle8i アドバンスト・キューイング用に Java API を提供します。Oracle JMS は、標準 JMS インタフェースをサポートし、また、AQ 管理操作およびその他の標準ではない AQ 機能をサポートする拡張機能を持ちます。

### 標準 JMS 機能

標準 JMS 機能には、次のものが含まれます。

- キューを使用した Point-to-Point 通信モデル
- トピックを使用したパブリッシュ / サブスクライブ通信モデル
- ObjectMessage、StreamMessage、TextMessage、BytesMessage、MapMessage という 5 つのメッセージ型
- 同期および非同期のメッセージ配信
- メッセージ・ヘッダー・フィールド / プロパティに基づくメッセージ選択

### Oracle JMS 拡張機能

Oracle JMS 拡張機能には、次のものが含まれます。

- キュー・テーブル、キューおよびトピック作成用の管理 API
- トピックの受信者リストを使用した Point-to-Multipoint 通信
- 宛先間のメッセージ伝播

アプリケーションによるリモート・サブスクライバの定義を許可します。

- 1 回のアトミック・トランザクションでの JMS および SQL 操作の実行を可能にする、トランザクション属性セッションのサポート
- メッセージがデキューされた後のメッセージ保存
- メッセージ遅延
- 例外処理

メッセージを正常に処理できない場合、メッセージは例外キューに移されます。

- 標準 JMS メッセージ型の他に、Oracle8i は `AdtMessages` もサポートします。これは、Oracle オブジェクトとしてデータベースに格納されるため、メッセージのペイロードをエンキュー後に問い合わせることができます。サブスクリプションは、メッセージ・プロパティのみでなく、これらのメッセージの内容にも定義できます。

## 標準 JMS および Oracle JMS へのアクセス

Oracle JMS は JDBC を使用してデータベースに接続するため、アプリケーションは次のように実行できます。

- データベース外では、OCI8 または Thin Driver を使用して実行します。
- Oracle8i Jserver 内では、Oracle Server ドライバを使用して実行します。

標準 JMS インタフェースは、`javax.jms` パッケージにあります。

Oracle JMS インタフェースは、`oracle.jms` パッケージにあります。

- **OCI8 または Thin JDBC Driver の使用** : データベース外で実行するクライアントで JMS を使用するには、CLASSPATH に適切な JDBC ドライバおよび次の jar ファイルを指定する必要があります。
  - JDK 1.1 の場合  
`$ORACLE_HOME/rdbms/jlib/jmscommon.jar`  
`$ORACLE_HOME/rdbms/jlib/aqapi11.jar`
  - JDK 1.2 の場合  
`$ORACLE_HOME/rdbms/jlib/jmscommon.jar`  
`$ORACLE_HOME/rdbms/jlib/aqapi.jar`
- **JServer での Oracle Server ドライバの使用** : ご使用のアプリケーションが JServer 内で実行している場合、JServer のインストール時に自動的にロードされた Oracle JMS クラスにアクセスする必要があります。これらのクラスが使用不可の場合、`loadjava` ユーティリティを使用して、`jmscommon.jar` の後に `aqapi.jar` をロードする必要があります。

## 権限

Oracle JMS インタフェースを使用する場合、ユーザーには、`DBMS_AQIN` および `DBMS_AQJMS` パッケージの実行権限が必要です。これらの権限は、`AQ_USER_ROLE` または `AQ_ADMINISTRATOR_ROLE` を介しても取得できます。

また、ユーザーがメッセージを送受信するには、適切なシステム、およびキュー権限またはトピック権限が必要です。

### 詳細情報

Oracle JMS インタフェースの詳細は、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。



# AQ プログラム環境の比較

ここでは、AQ プログラム環境で使用可能な機能を、ユースケース別に示します。

- 「AQ プログラム環境の比較 : 管理インタフェース」
- 「AQ プログラム環境の比較 : 操作インタフェース」

各ユースケースの詳細は、第 9 章～第 16 章を参照してください。

# AQ 管理インタフェース

表 3-2 に、PL/SQL、Java（ネイティブ AQ）および Java（JMS）のプログラム環境における同等の管理機能を示します。

表 3-2 AQ プログラム環境の比較 : 管理インタフェース

ユースケース	PL/SQL	Java（ネイティブ AQ）	Java（JMS）
接続ファクトリの作成	不可	不可	AQjmsFactory.getQueueConnectionFactory  AQjmsFactory.getTopicConnectionFactory
キュー・テーブルの作成	DBMS_AQADM.create_queue_table	AQQueueTableProperty の後に AQSession.createQueueTable	AQjmsSession.createQueueTable
キュー・テーブルの取得	<schema>.<queue_table_name> を使用	AQSession.getQueueTable	AQjmsSession.getQueueTable
キュー・テーブルの変更	DBMS_AQADM.alter_queue_table	AQQueueTable.alter	AQQueueTable.alter
キュー・テーブルの削除	DBMS_AQADM.drop_queue_table	AQQueueTable.drop	AQQueueTable.drop
キューの作成	DBMS_AQADM.create_queue	AQSession.createQueue	AQjmsSession.createQueue
キューの取得	<schema>.<queue_name> を使用	AQSession.getQueue	AQjmsSession.getQueue
非永続キューの作成	DBMS_AQADM.create_np_queue	未サポート	未サポート

表 3-2 AQ プログラム環境の比較 : 管理インタフェース (続き)

ユースケース	PL/SQL	Java (ネイティブ AQ)	Java (JMS)
複数コンシューマ・キュー / トピックの作成	DBMS_AQADM.create_queue 複数コンシューマが使用可能なキュー・テーブルで使用	AQSession.createQueue 複数コンシューマが使用可能なキュー・テーブルで使用	AQjmsSession.createTopic 複数コンシューマが使用可能なキュー・テーブルで使用
複数コンシューマ・キュー / トピックの取得	<schema>.<queue_name>を使用	AQSession.getQueue	AQjmsSession.getTopic
キューの変更	DBMS_AQADM.alter_queue	AQQueue.alterQueue	AQjmsDestination.alter
キュー / トピックの開始	DBMS_AQADM.start_queue	AQQueue.start AQQueue.startEnqueue AQQueue.startDequeue	AQjmsDestination.start
キュー / トピックの停止	DBMS_AQADM.stop_queue	AQQueue.stop AQQueue.stopEnqueue AQQueue.stopDequeue	AQjmsDestination.stop
キュー / トピックの削除	DBMS_AQADM.drop_queue	AQQueue.drop AQQueueTable.dropQueue	AQjmsDestination.drop
システム権限の付与	DBMS_AQADM.grant_system_privilege	未サポート	AQjmsSession.grantSystemPrivilege
システム権限の取消し	DBMS_AQADM.revoke_system_privilege	未サポート	AQjmsSession.revokeSystemPrivilege
キュー / トピック権限の付与	DBMS_AQADM.grant_queue_privilege	AQQueue.grantQueuePrivilege	AQjmsDestination.grantQueuePrivilege AQjmsDestination.grantTopicPrivilege
キュー / トピック権限の取消し	DBMS_AQADM.revoke_queue_privilege	AQQueue.revokeQueuePrivilege	AQjmsDestination.revokeQueuePrivilege AQjmsDestination.revokeTopicPrivilege
キュー・タイプの検証	DBMS_AQADM.verify_queue_types	未サポート	未サポート

表 3-2 AQ プログラム環境の比較：管理インタフェース（続き）

ユースケース	PL/SQL	Java（ネイティブ AQ）	Java（JMS）
サブスクライバ <sup>1</sup> の追加	DBMS_AQADM.add_subscriber	AQQueue.addSubscriber	表 3-3「AQ プログラム環境の比較：操作インタフェース」を参照
サブスクライバの変更	DBMS_AQADM.alter_subscriber	AQQueue.alterSubscriber	表 3-3「AQ プログラム環境の比較：操作インタフェース」を参照
サブスクライバの削除	DBMS_AQADM.remove_subscriber	AQQueue.removeSubscriber	表 3-3「AQ プログラム環境の比較：操作インタフェース」を参照
伝播のスケジュール	DBMS_AQADM.schedule_propagation	AQQueue.schedulePropagation	AQjmsDestination.schedulePropagation
伝播スケジュールの使用可能	DBMS_AQADM.enable_propagation_schedule	AQQueue.enablePropagationSchedule	AQjmsDestination.enablePropagationSchedule
伝播スケジュールの変更	DBMS_AQADM.alter_propagation_schedule	AQQueue.alterPropagationSchedule	AQjmsDestination.alterPropagationSchedule
伝播スケジュールの使用禁止	DBMS_AQADM.disable_propagation_schedule	AQQueue.disablePropagationSchedule	AQjmsDestination.disablePropagationSchedule
伝播スケジュールの解除	DBMS_AQADM.unschedule_propagation	AQQueue.unschedulePropagation	AQjmsDestination.unschedulePropagation

<sup>1</sup> サブスクライバおよび受信者の違いについては、第 1 章「Oracle アドバンスト・キューイングの概要」を参照してください。

# AQ 操作インタフェース

表 3-3 に、PL/SQL、Java（ネイティブ AQ）、OCI、Visual Basic（OO4O）および Java（JMS）のプログラム環境における同等の AQ 操作機能を示します。

表 3-3 AQ プログラム環境の比較：操作インタフェース

ユースケース	PL/SQL	Java (ネイティブ AQ)	OCI	Visual Basic	JMS
接続、セッション、メッセージの作成					
接続の作成	不可	JDBC 接続の 作成	OCIServerAttach	OpenDatabase	AQjmsQueueConnection Factory.createQueue Connection  AQjmsTopicConnection Factory.createTopicConnecion
セッションの作成	不可	AQDriver Manager.create AQSession	OCISessionBegin		QueueConnection.create QueueSession  TopicConnecion.createTopic Session
RAW メッセージの 作成	メッセージに SQL RAW 型を 使用	AQQueue. createMessage メッセージに AQRaw Payload を設定	メッセージに OCI RAW を使用	AQMsg (OraAQ) を メッセージ型 ORATYPE_RAW で使用	未サポート
構造化データを持 つメッセージの 作成	メッセージに SQL ADT 型を 使用	AQQueue. createMessage メッセージに AQObject Payload を設定	メッセージに SQL ADT 型を使 用	AQMsg (OraAQ) を ORATYPE_ OBJECT で、 ペイロードの SQL ADT 型を 型名で使用	Session.createTextMessage Session.createObjectMessage Session.createMapMessage Session.createBytesMessage Session.createStreamMessage AQjmsSession.createAdt Message
メッセージ・プロ デューサの作成	不可	不可	不可	不可	QueueSession.createSender TopicSession.createPublisher

表 3-3 AQ プログラム環境の比較：操作インタフェース（続き）

ユースケース	PL/SQL	Java (ネイティブ AQ)	OCI	Visual Basic	JMS
<b>単一コンシューマ・キューへのメッセージのエンキュー：Point-to-Point モデル</b>					
単一コンシューマ・キューへのメッセージのエンキュー	DBMS_AQ.enqueue	AQQueue.enqueue	OCIAQEnq	OraAQ.Enqueue	QueueSender.send
キューへのメッセージのエンキュー - 可視性オプションの指定	DBMS_AQ.enqueue ENQUEUE_OPTIONS の可視性を指定	AQQueue.enqueue AQEnqueue Option の可視性を指定	OCIAQEnq OCIAQEnq Options の OCI_ATTR_VISIBILITY を指定	OraAQ.Enqueue OraAQ の可視性プロパティを指定	未サポート
単一コンシューマ・キューへのメッセージのエンキュー - メッセージ・プロパティ（優先順位、期限切れ）の指定	DBMS_AQ.enqueue MESSAGE_PROPERTIES の優先順位、期限切れを指定	AQQueue.enqueue AQMessage Property の優先順位、期限切れを指定	OCIAQEnq OCIAQMsg Properties の OCI_ATTR_PRIORITY, OCI_ATTR_EXPIRATION を指定	OraAQ.Enqueue OraAQMsg の優先順位および期限切れを指定	QueueSender.send 時に、優先順位および TimeToLive を指定 または MessageProducer.setTimeToLive、MessageProducer.setPriority の後に、QueueSender.send
単一コンシューマ・キューへのメッセージのエンキュー - メッセージ・プロパティ（相関 ID、遅延、例外キュー）の指定	DBMS_AQ.enqueue MESSAGE_PROPERTIES の相関、遅延、exception_queue（例外キュー）を指定	AQQueue.enqueue AQMessage Property の相関、遅延、例外キューを指定	OCIAQEnq OCIAQMsg Properties の OCI_ATTR_CORRELATION, OCI_ATTR_DELAY, OCI_ATTR_EXCEPTION_QUEUE を指定	OraAQ.Enqueue OraAQMsg の相関、遅延、例外キュー・プロパティを指定	Message.setJMSCorrelationID プロバイダ固有のメッセージ・プロパティとして指定された遅延および例外キュー JMS_OracleDelay、JMS_OracleExcpQ などの後に、QueueSender.send
単一コンシューマ・キューへのメッセージのエンキュー - メッセージ・プロパティ（ユーザー定義）の指定	未サポート プロパティはペイロードの一部である必要がある	未サポート プロパティはペイロードの一部である必要がある	未サポート プロパティはペイロードの一部である必要がある	未サポート プロパティはペイロードの一部である必要がある	Message.setIntProperty、Message.setStringProperty、Message.setBooleanProperty などの後に、QueueSender.send

表 3-3 AQ プログラム環境の比較：操作インタフェース（続き）

ユースケース	PL/SQL	Java (ネイティブ AQ)	OCI	Visual Basic	JMS
<b>複数コンシューマ・キュー / トピックへのメッセージの公開 - パブリッシュ / サブスクライブ・モデル</b>					
複数コンシューマ・キュー / トピックへのメッセージの公開（デフォルト・サブスクリプション・リストを使用）	DBMS_AQ.enqueue  MESSAGE_PROPERTIES の受信者リストを NULL に設定	AQQueue.enqueue  AQMessageProperty の受信者リストを NULL に設定	OCIAQEnq  OCIAQMsgProperties の OCI_ATTR_RECIPIENT_LIST を NULL に設定	OraAQ.Enqueue	TopicPublisher.publish
複数コンシューマ・キュー / トピックへのメッセージの公開（特定の受信者リストを使用） <sup>1</sup>	DBMS_AQ.enqueue  MESSAGE_PROPERTIES の受信者リストを指定	AQQueue.enqueue  AQMessageProperty の受信者リストを指定	OCIAQEnq  OCIAQMsgProperties の OCI_ATTR_RECIPIENT_LIST を指定	OraAQ.Enqueue	AQjmsTopicPublisher.publish  AQjmsAgent の配列として受信者を指定
複数コンシューマ・キュー / トピックへのメッセージの公開 - メッセージ・プロパティ（優先順位、期限切れ）の指定	DBMS_AQ.enqueue  MESSAGE_PROPERTIES の優先順位、期限切れを指定	AQQueue.enqueue  AQMessageProperty の優先順位、期限切れを指定	OCIAQEnq  OCIAQMsgProperties の OCI_ATTR_PRIORITY, OCI_ATTR_EXPIRATION を指定	OraAQ.Enqueue  OraAQMsg の優先順位および期限切れプロパティを指定	QueueSender.send 時に、優先順位および TimeToLive を指定  または MessageProducer.setTimeToLive、MessageProducer.setPriority の後に、TopicPublisher.publish
複数コンシューマ・キュー / トピックへのメッセージの公開 - 送信オプション（相関 ID、遅延、例外キュー）の指定	DBMS_AQ.enqueue  MESSAGE_PROPERTIES の相関、遅延、exception_queue（例外キュー）を指定	AQQueue.enqueue  AQMessageProperty の相関、遅延、例外キューを指定	OCIAQEnq  OCIAQMsgProperties の OCI_ATTR_CORRELATION, OCI_ATTR_DELAY, OCI_ATTR_EXCEPTION_QUEUE を指定	OraAQ.Enqueue  OraAQMsg の相関、遅延、例外キュー・プロパティを指定	Message.setJMSCorrelationID  プロバイダ固有メッセージ・プロパティとして指定された遅延および例外キュー  JMS_OracleDelay、JMS_OracleExcpQ などの後に、TopicPublisher.publish

表 3-3 AQ プログラム環境の比較：操作インタフェース（続き）

ユースケース	PL/SQL	Java (ネイティブ AQ)	OCI	Visual Basic	JMS
トピックへのメッセージの公開 - メッセージ・プロパティ（ユーザー定義）の指定	未サポート  プロパティはペイロードの一部である必要がある	未サポート  プロパティはペイロードの一部である必要がある	未サポート  プロパティはペイロードの一部である必要がある	未サポート  プロパティはペイロードの一部である必要がある	Message.setIntProperty、 Message.setStringProperty、 Message.setBooleanProperty などの後に、 TopicPublisher.publish
<b>複数コンシューマ・キュー/トピック（パブリッシュ/サブスクライブ・モデル）のメッセージに対するサブスクライブ</b>					
サブスクライバの追加	管理インタフェースを参照	管理インタフェースを参照	未サポート	未サポート	TopicSession.createDurableSubscriber  AQjmsSession.createDurableSubscriber
サブスクライバの変更	管理インタフェースを参照	管理インタフェースを参照	未サポート	未サポート	TopicSession.createDurableSubscriber  AQjmsSession.createDurableSubscriber  新しいセレクトクを使用
サブスクライバの削除	管理インタフェースを参照	管理インタフェースを参照	未サポート	未サポート	AQjmsSession.unsubscribe

表 3-3 AQ プログラム環境の比較 : 操作インタフェース (続き)

ユースケース	PL/SQL	Java (ネイティブ AQ)	OCI	Visual Basic	JMS
<b>キューのメッセージのブラウズ</b>					
キュー / トピック のメッセージのブ ラウズ	DBMS_AQ. dequeue  DEQUEUE_ OPTIONS の dequeue_ mode を BROWSE に 設定	AQQueue. dequeue  AQDequeue Option の dequeue_ mode を BROWSE に設定	OCIAQDeq  OCIAQDeq Options の OCI_ATTR_ DEQ_MODE を BROWSE に設定	OraAQ. Dequeue  OraAQ のデ キュー・モー ドを ORAAQ_DQ_ BROWSE に設 定	QueueSession.createBrowser  QueueBrowser.getEnumeration  トピックでは未サポート
キュー / トピック のメッセージのブ ラウズ - ブラウズ 中のメッセージの ロック	DBMS_AQ. dequeue  DEQUEUE_ OPTIONS の dequeue_ mode を LOCKED に 設定	AQQueue. dequeue  AQDequeue Option の dequeue_mode を LOCKED に 設定	OCIAQDeq  OCIAQDeq Options の OCI_ATTR_ DEQ_MODE を LOCKED に設定	OraAQ. Dequeue  OraAQ の DequeueMode を ORAAQ_DQ_ LOCKED とし て指定	locked を TRUE に設定した AQjmsSession.createBrowser  QueueBrowser.getEnumeration  トピックでは未サポート
<b>キュー / トピックからのメッセージの受信</b>					
メッセージ受信用 の接続を起動	不可	不可	不可	不可	Connection.start
メッセージ・コン シューマの作成	不可	不可	不可	不可	QueueSession.createQueue Receiver  TopicSession.createDurable Subscriber  AQjmsSession.createTopic Receiver
キュー / トピック からのメッセージ のデキュー - 可視 性の指定	DBMS_AQ. dequeue  DEQUEUE_ OPTIONS の 可視性を指定	AQQueue. dequeue  AQDequeue Option の可視 性を指定	OCIAQDeq  OCIAQEnq Options の OCI_ATTR_ VISIBILITY を 指定	OraAQ. Dequeue  OraAQ の可視 プロパティを 指定	未サポート



表 3-3 AQ プログラム環境の比較：操作インタフェース（続き）

ユースケース	PL/SQL	Java (ネイティブ AQ)	OCI	Visual Basic	JMS
キュー / トピック からのメッセージ のデキュー - ナビ ゲーション・モー ドの指定	DBMS_AQ. dequeue  DEQUEUE_ OPTIONS の ナビゲーションを指定	DBMS_AQ. dequeue  AQDequeue Option のナビ ゲーションを 指定	OCIAQDeq  OCIAQDeq Options の OCI_ATTR_ NAVIGATION を指定	OraAQ. Dequeue  OraAQ のナビ ゲーションを 指定	AQjmsQueueReceiver.set NavigationMode  AQjmsTopicSubscriber.set NavigationMode  AQjmsTopicReceiver.set NavigationMode
単一コンシュー マ・キューからの メッセージのデ キュー	DBMS_AQ. dequeue  DEQUEUE_ OPTIONS の dequeue_ mode を REMOVE に 設定	AQQueue. dequeue  AQDequeue Option の dequeue_mode を REMOVE に 設定	OCIAQDeq  OCIAQDeq Options の OCI_ATTR_DEQ _MODE を REMOVE に設定	OraAQ. Dequeue  OraAQ の DequeueMode を ORAAQ_DQ_ REMOVE とし て指定	QueueReceiver.receive または QueueReceiver.receiveNo Wait または AQjmsQueueReceiver. receiveNoData
複数コンシュー マ・キュー / ト ピックからのメッ セージのデキュー (サブスクリプ ション名の使用) <sup>1</sup>	DBMS_AQ. dequeue  DEQUEUE_ OPTIONS の dequeue_ mode を REMOVE に、 また、 consumer_ name をサブ スクリプション名に設定	AQQueue. dequeue  AQDequeue Option の dequeue_mode を REMOVE に、 また、 consumer_ name をサブス クリプション名 に設定	OCIAQDeq  OCIAQDeq Options の OCI_ATTR_DEQ _MODE を REMOVE に、 また、 OCI_ATTR_ CONSUMER_ NAME をサブス クリプション名 に設定	OraAQ. Dequeue  OraAQ の DequeueMode を ORAAQ_DQ_ REMOVE とし て指定  OraAQ の Consumer を サブスクリプ ション名とし て設定	サブスクリプション名を使用 してトピック上に持続的 な TopicSubscriber を作成し た後 TopicSubscriber.receive または TopicSubscriber.receiveNo Wait または AQjmsTopicSubscriber. receiveNoData
複数コンシュー マ・キュー / ト ピックからのメッ セージのデキュー (受信者名の使 用) <sup>1</sup>	DBMS_AQ. dequeue  DEQUEUE_ OPTIONS の dequeue_ mode を REMOVE に、 また、 consumer_ name を受信 者名に設定	AQQueue. dequeue  AQDequeue Option の dequeue_mode を REMOVE に、 また、 consumer_ name を受信者 名に設定	OCIAQDeq  OCIAQDeq Options の OCI_ATTR_DEQ _MODE を REMOVE に、 また、 OCI_ATTR_ CONSUMER_ NAME を受信者 名に設定	OraAQ. Dequeue  OraAQ の DequeueMode を ORAAQ_DQ_ REMOVE とし て指定  OraAQ の Consumer を サブスクリプ ション名とし て設定	受信者名を使用してトピック 上に TopicReceiver を作 成した後 AQjmsSession.createTopic Receiver AQjmsTopicReceiver.receive または AQjmsTopicReceiver.receive NoWait または AQjmsTopicReceiver.receive NoData

表 3-3 AQ プログラム環境の比較 : 操作インタフェース (続き)

ユースケース	PL/SQL	Java (ネイティブ AQ)	OCI	Visual Basic	JMS
キュー / トピックからメッセージを非同期受信するための登録					
単一コンシューマ・キューからのメッセージの非同期受信	未サポート	未サポート	OCISubscription Register  queue_name をサブスクリプション名として指定  OCISubscription Enable	OraAQ. MonitorStart	キュー上に QueueReceiver を作成した後  QueueReceiver.setMessageListener
複数コンシューマ・キュー / トピックからのメッセージの非同期受信	未サポート	未サポート	OCISubscription Register  キュー OCI_ATTR_CONSUMER_NAME をサブスクリプション名として指定  OCISubscription Enable	OraAQ. MonitorStart  OraAQ の Consumer をサブスクリプション名として指定	トピック上に TopicSubscriber または TopicReceiver を作成した後  TopicSubscriber.setMessageListener  TopicReceiver.setMessageListener
複数キュー / トピック上でのメッセージのリスニング					
1 つ (または多数) の単一コンシューマ・キュー上でのメッセージのリスニング	DBMS_AQ. listen  agent_name を agent_list のすべてのエージェントで NULL として使用	未サポート	OCIAQListen  agent_name を agent_list のすべてのエージェントで NULL として使用	未サポート	QueueSession 上に複数の QueueReceiver を作成した後  QueueSession.setMessageListener
1 つ (または多数) の複数コンシューマ・キュー / トピック上でのメッセージのリスニング	DBMS_AQ. listen  agent_list のすべてのエージェントの agent_name を指定	未サポート	OCIAQListen  agent_list のすべてのエージェントの agent_name を指定	未サポート	TopicSession 上に複数の TopicSubscriber または TopicReceiver を作成した後  TopicSession.setMessageListener

<sup>1</sup> サブスクライバおよび受信者の違いについては、第 1 章「Oracle アドバンスト・キューイングの概要」を参照してください。

---

## AQ の管理

この章では、アドバンスト・キューイングの管理に関する次の項目について説明します。

- キュー・テーブルの移行（インポート / エクスポート）
- セキュリティ
- Enterprise Manager サポート
- プロトコル
- AQ 操作を準備するための DBA のアクション例
- 現在の制限事項

## キュー・テーブルの移行（インポート / エクスポート）

キュー・テーブルがエクスポートされると、キュー・テーブル・データおよび PL/SQL コードの無名ブロックがエクスポート・ダンプ・ファイルに書き込まれます。キュー・テーブルがインポートされると、インポート・ユーティリティがこれらの PL/SQL 無名ブロックを実行して、メタデータをデータ・ディクショナリに書き込みます。

## キュー・テーブル・データのエクスポート

キューは、表に実装されます。キューをエクスポートすると、必然的に基になるキュー・テーブルおよび関連するディクショナリ表もエクスポートされます。キューは、キュー・テーブル単位でのみエクスポートできます。

### 複数の受信者がいるキュー・テーブルのエクスポート

複数受信者をサポートするすべてのキュー・テーブルに対し、重要なキュー・メタデータを収めた索引構成表（IOT）および時間管理表が別にあります。8.1 互換のキュー・テーブルには、サブスクリバ表、履歴表およびルール表もあります。このメタデータはキューの操作に不可欠であるため、ユーザーはインポート後に作業するためには、そのキュー自体のキュー・テーブルの他に、メタデータの表もエクスポートする必要があります。データベース・モードおよびユーザー・モードのエクスポートでは、この表は自動的にエクスポートされます。

このメタデータ表には、キュー・テーブル内の一部の行の ROWID が含まれているため、メタデータ表をインポートするときに、廃止される ROWID についての注意情報が生成されます。廃止される ROWID は、キューイング・システムのインポート操作の一部として自動的に修正されるため、このメッセージは無視してもかまいません。ただし、インポート中に他の問題（ロールバック・セグメント領域の不足など）が発生した場合は、この問題を修正してインポートを繰り返す必要があります。

### ルールのエクスポート

ルールはキュー・テーブルと対応付けられます。キュー・テーブルがエクスポートされるとき、対応付けられたルールがある場合は、それらもすべて自動的にエクスポートされます。

### サポートされているエクスポート・モード

現在、エクスポートは3通りのモードで操作されます。全データベース・モード、ユーザー・モードおよび表モードです。次に、3通りのエクスポート・モードの操作について説明します。

### 全データベース・モード

このモードはサポートされています。キュー・テーブル、すべての関連表、システム・レベルの権限付与、および1次のおよび2次のオブジェクト権限付与が自動的にエクスポートされます。

### ユーザー・モード

このモードはサポートされています。キュー・テーブル、すべての関連表および1次のオブジェクト権限付与が自動的にエクスポートされます。

### 表モード

お薦めできません。キュー・テーブルを表モードでエクスポートする必要がある場合、ユーザーは、そのキュー・テーブルに属するすべての関連表をエクスポートする責任があります。たとえば、8.1 互換の複数コンシューマ・キュー・テーブル MCQ をエクスポートするとき、次の各表もエクスポートする必要があります。

```
AQ$_MCQ_I  
AQ$_MCQ_H  
AQ$_MCQ_S  
AQ$_MCQ_T
```

### 増分エクスポート

キュー・テーブルの増分エクスポートはサポートされていません。

## キュー・テーブル・データのインポート

エクスポートと同様に、キューをインポートすると、必ず基になるキュー・テーブルおよび関連するディクショナリ・データもインポートされます。キュー・テーブル・データがインポートされると、インポート・ユーティリティがダンプ・ファイルの PL/SQL 無名ブロックを実行して、メタデータをデータ・ディクショナリに書き込みます。

### 複数の受信者がいるキュー・テーブルのインポート

前述のように、複数の受信者をサポートするすべてのキュー・テーブルには、重要なキュー・メタデータを収めた索引構成表 (IOT)、サブスクライバ表、履歴表および時間管理表があります。インポート後に作業するためには、そのキュー自体のキュー・テーブルの他に、それらメタデータの表もエクスポートする必要があります。

このメタデータ表には、キュー・テーブル内の一部の行の ROWID が含まれているため、メタデータ表をインポートするときに、廃止される ROWID についての注意情報が発行されます。廃止される ROWID は、キューイング・システムのインポート操作の一部として自動的に修正されるため、このメッセージは無視してもかまいません。

ただし、インポート中に他の問題（ロールバック・セグメント領域の不足など）が発生した場合は、この問題を修正し、インポートを再実行する必要があります。

### インポート IGNORE パラメータ

すでにデータがあるキュー・テーブルには、キュー・データをインポートしないようにしてください。キュー・テーブルをインポートする DBA は、常にインポート・ユーティリティの IGNORE パラメータに NO を設定することをお勧めします。IGNORE パラメータに YES が設定され、すでに存在しているキュー・テーブルがダンプ・ファイル中の表定義と互換性があるとき、ダンプ・ファイルの各行は既存の表にロードされます。同時に、古いキュー・テーブル定義およびキュー定義は削除および再作成されます。そのため、インポート以前に作成されたキュー・テーブルおよびキュー定義は失われ、複製された行がキュー・テーブルの中に現れます。

## セキュリティ

構成情報は、DBMS\_AQADM パッケージ内のプロシージャを使用して管理できます。最初は、SYS および SYSTEM のみに、DBMS\_AQADM および DBMS\_AQ 内のプロシージャに対する実行権限が付与されています。この 2 つのパッケージに対する実行権限を付与されているユーザーは誰でも、自分自身のスキーマ内のキューを作成、管理および使用できます。他のスキーマのキューを作成および管理するには、MANAGE ANY QUEUE 権限が必要です。

### 8.0 および 8.1 互換のキューのセキュリティ

リリース 8.1 データベースの AQ 管理者は、8.0 または 8.1 互換のキューを作成できます。リリース 8.1 のセキュリティ機能はすべて 8.1 互換のキューで可能になります。ただし、AQ 8.1 のセキュリティ機能が作用するのは 8.1 互換のキューに限られ、8.0 互換のキューは、8.0 互換のセキュリティ機能によって保護されることに注意してください。

新しいセキュリティ機能を実現したリリース 8.1 でキューを作成するには、DBMS\_AQADM.CREATE\_QUEUE\_TABLE の互換パラメータに 8.1 以上の値を設定する必要があります。本来、リリース 8.0 のデータベース用に作成されたキューで新しいセキュリティ機能を使用するときは、DBMS\_AQADM.MIGRATE\_QUEUE\_TABLE を実行して、そのキュー・テーブルを 8.1 互換に変換する必要があります。

データベースのダウングレードが必要になった場合は、8.1 互換のすべてのキュー・テーブルを 8.0 互換に変換するか、ダウングレードを実行する前に削除する必要があります。変換によって、オブジェクト権限と同様に、それらのキューのリリース 8.1 のセキュリティ機能はすべて削除されます。あるキューが 8.0 互換に変換されると、そのキューにはリリース 8.0 のセキュリティ・モデルが適用され、リリース 8.0 のセキュリティ機能のみがサポートされます。

次の表に、Oracle8 データベースの各バージョンでサポートされている AQ セキュリティ機能、および異なるデータベース・バージョンにおける同等の権限を示します。

**表 4-1 8.0 および 8.1 互換のキューのセキュリティ**

権限	8.0.x データベース	8.1.x データベースの 8.0.x 互換のキュー	8.1.x データベースの 8.1.x 互換のキュー
AQ_USER_ROLE	サポート  権限受領者は、そのロールを介して DBMS_AQ の実行権限が付与されます。	サポート  権限受領者は、そのロールを介して DBMS_AQ の実行権限が付与されます。	未サポート  同等の権限は次のとおりです。  1. DBMS_AQ の実行権限  2. システム権限 ENQUEUE ANY QUEUE  3. システム権限 DEQUEUE ANY QUEUE
AQ_ADMINISTRATOR_ROLE	サポート	サポート	サポート
DBMS_AQ の実行権限	PL/SQL で AQ アプリケーションを作成する開発者には、DBMS_AQ の実行権限が付与される必要があります。	PL/SQL で AQ アプリケーションを作成する開発者には、DBMS_AQ の実行権限が付与される必要があります。	すべての AQ ユーザーに DBMS_AQ の実行権限が付与される必要があります。8.1 互換のキューにエンキューまたはデキューするには、ユーザーに次の権限が必要です。  1. DBMS_AQ の実行権限  2. ターゲット・キューに対するエンキュー権限 / デキュー権限、またはシステム権限 ENQUEUE ANY QUEUE/DEQUEUE ANY QUEUE のいずれか

## 権限およびアクセス制御

Oracle8i では、8.1 互換のキューにオブジェクト・レベルの権限を付与したり取り消すことができます。また、様々なシステム・レベル権限を付与したり取り消すことができます。

次の表にすべての共通 AQ 操作、およびそれらの操作を 8.1 互換のキューで実行するために必要な権限を示します。

表 4-2 8.1 セキュリティ・モデルにおける操作および必要な権限

操作	必要な権限
所有するキューに対する CREATE/DROP/MONITOR	DBMS_AQADM の実行権限が必要です。その他の権限は不要です。
すべてのキューに対する CREATE/DROP/MONITOR	DBMS_AQADM の実行権限が必要です。 また、AQ_ADMINISTRATOR_ROLE を付与されている他のユーザーによって、このロールを付与される必要があります（最初は、SYS および SYSTEM が AQ_ADMINISTRATOR_ROLE を付与します）。
所有するキューに対する ENQUEUE/DEQUEUE	DBMS_AQ の実行権限が必要です。その他の権限は不要です。
他のユーザーが所有する キューに対する ENQUEUE/DEQUEUE	DBMS_AQ の実行権限が必要です。また、所有者から、DBMS_AQADM.GRANT_QUEUE_PRIVILEGE を使用して権限を付与される必要があります。
すべてのキューに対する ENQUEUE/DEQUEUE	DBMS_AQ の実行権限が必要です。また、AQ 管理者から、DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE を使用してシステム権限 ENQUEUE ANY QUEUE または DEQUEUE ANY QUEUE を付与される必要があります。

ロール

Oracle リリース 8.0 の AQ 操作には、AQ プロシージャに実行権限を付与するロールを通じてアクセスできます。Oracle リリース 8.0 を使用するときデータベース・オブジェクト・レベルの制御がないということは、Oracle リリース 8.0 で AQ\_USER\_ROLE を持つユーザーは、システム内のすべてのキューに対してエンキューおよびデキューできるということを意味します。Oracle8i ではきめ細かいアクセス制御が提供されているため、リリース 8.1 のコンテキストでアプリケーション開発をすると、ロールの機能も変化します。

管理者ロール

Oracle8i でも、引き続き AQ\_AQMISTRATOR\_ROLE をサポートします。リリース 8.0 と同様に、AQ\_ADMINISTRATOR\_ROLE はキュー管理に必要な権限をすべて付与します。ロールに付与された権限によって、権限受領者は次のことができるようになります。

- データベースのすべてのスキーマに対するすべてのキュー管理操作（キューおよびキュー・テーブルの作成など）の実行
- データベースのすべてのキューに対するエンキューおよびデキュー操作の実行
- そのキューの作業負荷を監視するための統計ビューへのアクセス



## ユーザー・ロール

AQ\_USER\_ROLE は、リリース 8.0 との互換性を持って作成されたキューに対して引き続き有効です。ただし、このロールでは 8.1 互換のキューにエンキューまたはデキューするために必要な権限を付与することができないため、Oracle8i では、AQ\_USER\_ROLE を付与しないでください。

データベース管理者にはデータベース・ユーザーに直接システム権限 ENQUEUE ANY QUEUE および DEQUEUE ANY QUEUE を付与したり、DBMS\_AQADM.GRANT\_SYSTEM\_PRIVILEGE および DBMS\_AQADM.REVOKE\_SYSTEM\_PRIVILEGE を行使する権限があります。アプリケーション開発者としては DBMS\_AQADM.GRANT\_QUEUE\_PRIVILEGE および DBMS\_AQADM.REVOKE\_QUEUE\_PRIVILEGE を行使することで、ユーザーはオブジェクト・レベルの権限を付与または取り消して、キューに権限を付与します。

データベース・ユーザーとして、所有するスキーマのキューにエンキューまたはデキューするためには、DBMS\_AQ の実行権限の他にオブジェクト・レベルまたはシステム・レベルの明示的な権限は必要ありません。

## AQ オブジェクト型へのアクセス

grant\_type\_access プロシージャは、8.0 互換キューおよび 8.1 互換キューの両方に関して、リリース 8.1.5 で廃止されました。現在、内部 AQ オブジェクトはすべて、PUBLIC でアクセス可能です。

## OCI アプリケーション

OCI アプリケーションで 8.0 互換のキューにアクセスするには、そのセッション・ユーザーに DBMS\_AQ に対する実行権限が付与されている必要があります。OCI アプリケーションで 8.1 互換のキューにアクセスするには、そのセッション・ユーザーに、アクセス先のキューのオブジェクト権限、またはシステム権限 ENQUEUE ANY QUEUE または DEQUEUE ANY QUEUE（あるいはその両方）のいずれかが付与されている必要があります。アクセス先のキューが 8.1 互換の場合、DBMS\_AQ に対する実行権限がそのセッション・ユーザーの権限と照合してチェックされることはありません。

## 伝播

AQ は、データベース・リンクを介してメッセージを伝播します。伝播ドライバは、ソース・キューの所有者としてソース・キューからデキューします。そのため、ソース・キューに対する明示的なアクセス権限を付与される必要はありません。宛先では、そのデータベース・リンクにログインしているユーザーは ENQUEUE ANY QUEUE 権限またはその宛先キューにエンキューする権限のどちらかを付与されている必要があります。ただし、データベース・リンクのログイン・ユーザーが宛先のキュー・テーブルを所有している場合は、どのような明示的な AQ 権限も付与される必要はありません。

用途

8.0 互換のキュー・テーブルを 8.1 互換のキュー・テーブルにアップグレード、または 8.1 互換のキュー・テーブルを 8.0 互換のキュー・テーブルにダウングレードします。

構文

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE(  
    queue_table      IN      VARCHAR2,  
    compatible       IN      VARCHAR2)
```

使用方法

表 4-3 DBMS\_AQADM\_MIGRATE\_QUEUE\_TABLE

パラメータ	説明
queue_table (IN VARCHAR2)	移行されるキューの名前を指定します。
compatible	8.0 のキュー・テーブルを 8.1 互換のキュー・テーブルにアップグレードするときは、'8.1' に設定します。8.1 のキュー・テーブルを 8.0 互換のキュー・テーブルにダウングレードするときは、'8.0' に設定します。

使用上の注意

異なるリリース間の相互関係に関する最新情報は、4-13 ページの「[互換性](#)」を参照してください。

例：8.0 互換のキュー・テーブルを 8.1 互換のキュー・テーブルにアップグレードする方法

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (  
    queue_table      => 'qtable1',  
    multiple_consumers  => TRUE,  
    queue_payload_type  => 'aq.message_type',  
    compatible       => '8.0');
```

```
EXECUTE DBMS_AQADM.MIGRATE_QUEUE_TABLE(  
    queue_table => 'qtable1',  
    compatible  => '8.1');
```

## Enterprise Manager サポート

Enterprise Manager では、「管理インタフェース」の項に示した管理機能の大半で、GUI をサポートします。

次のものが含まれます。

1. キューのプロパティを参照するスキーマ・マネージャ
2. キューの作成、開始、停止および削除
3. 伝播のスケジュールおよびスケジュール解除
4. サブスクライバの追加および削除
5. 現在の伝播スケジュールの表示
6. 権限の付与および取消し

## プロトコル

AQ OCI インタフェースを使用する場合は、`xa_open` 文字列で「`Objects=T`」を指定する必要があります。これは、オブジェクト・モードで、XA にクライアント側キャッシュを初期化させます。OCI または Pro\*C から、PL/SQL ラッパーを介して AQ を使用する場合は、この処理は必要ありません。AQ は（LOB として格納されるにもかかわらず）単純な RAW バッファに抽象化されるため、Pro\*C シリーズのマニュアルに記載されている LOB メモリー管理の概念は、RAW 型のメッセージには当てはまりません。

XA から AQ を使用するときは、AQ ナビゲーション・オプションを慎重に使用する必要があります。XA は、`xa_end` が終了すると、カーソル・フェッチ状態を取り消します。したがって、サービス間（`xa_start` 境界と `xa_end` 境界の間）でデキュー処理を続ける場合は、`FIRST_MESSAGE` ナビゲーション・オプションを使用してデキュー位置をリセットする必要があります。これを実行しないと、ORA-25237（ナビゲーションの順序が正しくありません。）というエラーになります。

## AQ 操作を準備するための DBA のアクション例

### AQ 管理者としてのユーザー作成

ユーザーを AQ 管理者として設定するためには、次のステップが必要です。

```
CONNECT system/manager
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT AQ_ADMINISTRATOR_ROLE TO aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
```

さらに、AQ パッケージの実行権限を、次のように付与します。

```
GRANT EXECUTE ON DBMS_AQADM TO aqadm;  
GRANT EXECUTE ON DBMS_AQ TO aqadm;
```

これによって、ユーザー・プロシージャから AQ パッケージに含まれるプロシージャを実行できます。

### ユーザー AQUSER1 および AQUSER2 を、2 つの AQ ユーザーとして作成

自スキーマにキューを作成およびアクセスできる AQ ユーザーを作成する場合、AQ\_ADMINISTRATOR\_ROLE の付与を除いて、前の項で説明したステップに従います。

```
CONNECT system/manager  
CREATE USER aquser1 IDENTIFIED BY aquser1;  
GRANT CONNECT, RESOURCE TO aquser1;
```

さらに、AQ パッケージの実行権限を、次のように付与します。

```
GRANT EXECUTE ON DBMS_AQADM to aquser1;  
GRANT EXECUTE ON DBMS_AQ TO aquser1;
```

キューを作成しないで、別のスキーマのキューを使用する AQ ユーザーを作成する場合は、まず、前の項で説明したステップに従います。さらに、オブジェクト・レベル権限を付与します。ただし、これは 8.1 互換のキュー・テーブルを使用して定義されたキューにのみ適用されることに注意してください。

```
CONNECT system/manager  
CREATE USER aquser2 IDENTIFIED BY aquser2;  
GRANT CONNECT, RESOURCE TO aquser2;
```

さらに、AQ パッケージの実行権限を、次のように付与します。

```
GRANT EXECUTE ON DBMS_AQADM to aquser2;  
GRANT EXECUTE ON DBMS_AQ TO aquser2;
```

aquser2 が aquser1 スキーマの aquser1\_q1 キューにアクセスするために、aquser1 は次の文を実行する必要があります。

```
CONNECT aquser1/aquser1  
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(  
    'ENQUEUE', 'aquser1_q1', 'aquser2', FALSE);
```

## 現在の制限事項

現在、次の制限事項が適用されています。

### DBMS\_AQADM パッケージの自動コミット機能

DBMS\_AQADM パッケージの CREATE\_QUEUE\_TABLE、DROP\_QUEUE\_TABLE、CREATE\_QUEUE、DROP\_QUEUE および ALTER\_QUEUE コールの auto\_commit パラメータは、リリース 8.1.5 以降では使用しないでください。Oracle は、下位互換性を保つために、引き続きこのパラメータをサポートします。

### メッセージ・ペイロード（実際に通信される情報）内のコレクション型

オブジェクトに含まれていない VARRAY を使用して、メッセージ・ペイロードを組み立てることはできません。また、現時点では、メッセージ・ペイロード内で埋込みオブジェクトとして NESTED TABLE を使用できません。ただし、1 つ以上の VARRAY を含むオブジェクト型を作成して、このオブジェクト型に基づくキュー・テーブルを作成できます。

たとえば、次の操作は有効です。

```
CREATE TYPE number_varray AS VARRAY(32) OF NUMBER;
CREATE TYPE embedded_varray AS OBJECT (coll number_varray);
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'QT',
    queue_payload_type => 'embedded_varray');
```

### AQ の Java API におけるオブジェクト型ペイロード（実際に通信される情報）のサポート

リリース 8.1.5 に含まれる AQ の Java クラスでは、ペイロード内にオブジェクト型を利用した場合のエンキューおよびデキューはできず、RAW 型を利用した場合しかサポートされていません。

### キュー・テーブルおよびキューにおけるシノニム

どの AQ PL/SQL コールも、キューおよびキュー・テーブルにおけるシノニムを解釈しません。ユーザーはシノニムを作成できますが、そのシノニムを AQ インタフェースに適用しないでください。

## 8.0 互換の複数コンシューマ・キューではトランスポートابل表領域が無効

8.0 互換の複数コンシューマ・キュー・テーブルを含む表領域は、トランスポートابل表領域メカニズムでは転送されません。ただし、単一コンシューマ・キューや 8.1 互換の複数コンシューマ・キューを含む表領域では、そのメカニズムが有効です。トランスポートابل・モードで表領域をエクスポートするには、その前に、その表領域を読み専用モードに変更する必要があります。8.0 互換の複数コンシューマ・キュー・テーブルを含む読み専用表領域をインポートしようとする、キュー・テーブルの索引をインポート時に更新できないという Oracle エラーになります。

## 表領域の Point-in-Time リカバリ

AQ は、現時点で表領域の Point-in-Time リカバリをサポートしていません。表領域にキュー・テーブルを作成すると、Point-in-Time リカバリが使用できなくなります。

## オブジェクト・キューからの伝播

AQ では、ペイロードに BFILE または REF 属性があるオブジェクト・キューからの伝播はサポートされていません。

## 非永続キュー

現在、非永続キューが作成できるのは RAW 型のみです。また、それらのメッセージを送信できるのは、サブスクライバおよびローカルな明示的受信者に対してのみです。非永続キューからの伝播は、サポートされていません。さらに、メッセージを取り出すときはデキュー・コールではなく、OCISubscriptionRegister によって通知を登録する非同期通知メカニズムを使用する必要があります。

# 互換性

8.1 互換に設定すると、いくつかの機能は使用できなくなります。[表 4-4](#) に示すとおり、`init.ora` の `compatible` パラメータまたはキュー・テーブルの `compatible` パラメータ（あるいはその両方）を設定する必要がある場合があります。

**表 4-4 新機能を使用するために必須の互換性設定**

機能	Init.ora compatible = ' 8.1.x'	queue table compatible = ' 8.1'
キュー・レベルのアクセス制御	X	X
非永続キュー	X	自動作成される
OPS 環境のサポート	X	
パブリッシュ / サブスクライブにおけるルールベースのサブスクライバ	X	X
非同期通知	X	
送信者識別	X	X
分離した履歴管理情報の格納	X	X

**参照：**

- 詳細は、4-2 ページの「[キュー・テーブルの移行（インポート / エクスポート）](#)」を参照してください。
- 『Oracle8i 移行ガイド』も参照してください。





---

## 高度なトピック

この章では次の高度なトピックについて説明します。

- パフォーマンス
  - 表および索引構造
  - スループット
  - 可用性
- スケーラビリティ
- 伝播の問題点
- AQ 伝播問題のデバッグについてのガイドライン

## パフォーマンス

キューは、データベース表に格納されます。キュー操作のパフォーマンス特性は、基になるデータベース操作に非常によく似ています。

### 表および索引構造

キューのパフォーマンス特性を理解するには、AQ オブジェクト用の表および索引レイアウトを理解することが重要です。

キュー・テーブルを作成すると、約 25 列のデータベース表が作成されます。これらの列には、AQ のメタデータおよびユーザー定義のペイロードが格納されます。このペイロードの型は、オブジェクト型または RAW 型です。AQ メタデータには、オブジェクト型およびスカラー型が含まれています。キュー・テーブルには、1 つのビューおよび 2 つの索引が作成されます。このビューを使用することで、ユーザーはメッセージ・データを問い合わせることができます。索引は、メッセージ・データへのアクセス時間を短縮するために利用されます。作成されるオブジェクトの詳細は、9-5 ページの「[キュー・テーブルの作成](#)」を参照してください。

### スループット

エンキュー操作のコード・パスは、2 つの索引を持つ複数列表への挿入に似ています。デキュー操作のコード・パスは、同様の表に対する選択および削除操作に似ています。これらの操作は、PL/SQL ファンクションを使用して行われます。

### 可用性

Oracle Parallel Server (OPS) を使用することで、キュー・データに対する高可用性のアクセスを実現できます。キューは、データベース表を使用して実装されます。キューの先頭および末尾は、かなりのホット・スポット（混雑点）になる可能性があります。ホット・スポットが存在すると OPS は十分に性能を発揮できないため、キューに通常のアクセスをするインスタンスを 1 つに制限することをお勧めします。インスタンス障害の場合、障害が発生したインスタンスによって管理されていたメッセージは、障害がないインスタンスの 1 つによってすぐに処理されます。

## スケーラビリティ

キュー操作のスケーラビリティは、基になるデータベース操作のスケーラビリティに似ています。マルチスレッド・サーバー（MTS）環境で wait オプション付きのデキュー操作が行われると、共有サーバー・プロセスでは、待ち時間を含むコールの所要時間中、デキュー操作のみが行われます。このようなプロセスが多数存在すると、パフォーマンスおよび可用性に重大な問題が発生し、共有サーバー・プロセスのデッドロックを引き起こす可能性があります。そのため、wait オプション付きのデキュー要求は、専用サーバー・プロセスでのみ発行することをお勧めします。この制限は、強制されません。

## 伝播の問題点

---

---

**注意：** 伝播は、システム・キュー `aq$_prop_notify_X`（X は、スケジュールのソース・キューが常駐するインスタンスのインスタンス番号）を使用して、伝播ランタイム・イベントを操作します。このキューのメッセージは、システム表 `aq$_prop_table_X`（X は、スケジュールのソース・キューが常駐するインスタンスのインスタンス番号）に格納されます。伝播が正常に動作するために、キュー `aq$_prop_notify_X` が中止または削除されたり、表 `aq$_prop_table_X` が削除されることがないよう to してください。

---

---

### 伝播の最適化

`JOB_QUEUE_PROCESSES` 数を設定するとき、DBA は、この値が、伝播するメッセージの伝播元のキュー数、およびメッセージの伝播先の宛先（キューではない）の数によって決まることを理解しておく必要があります。

今回のリリースでは、伝播を処理するために、新しいスケーラブルなスケジューリング・アルゴリズムが組み込まれました。このアルゴリズムは使用可能なジョブ・キュー・プロセスを最適化し、メッセージがソース・キューにエンキューされてから宛先に現れるまでの時間を最小限にするように設計されているため、OLTP とほぼ同様の動作を実現します。このアルゴリズムが同時に操作できるスケジュールの数は、無制限です。また、様々な種類の障害に対しても強力に対処できます。伝播は、使用可能なジョブ・キュー・プロセスの最適使用を試みますが、開始するジョブ・キュー・プロセスの数はレプリケーション・ジョブのような非伝播関連ジョブの存在にも依存します。したがって、この新しいアルゴリズムによって最高の結果を得るためには、次のガイドラインを利用することが重要です。

新しいアルゴリズムは、次のようにジョブ・キュー・プロセスを使用します（ここでは、アクティブ・スケジュールが適切に設定されているとします）。

- アクティブ・スケジュール数がジョブ・キュー・プロセスの半数に満たない場合、アクティブ・スケジュール数と同数のジョブ・キュー・プロセスが取得されます。
- アクティブ・スケジュール数がジョブ・キュー・プロセス数の半数を超える場合、ジョブ・キュー・プロセスの半数を取得した後で、取得した各ジョブ・キュー・プロセスに複数のアクティブ・スケジュールを割り当てます。
- システムがオーバーロードになっている（すべてのスケジュールが伝播のためにビジーになっている）場合、可用性に応じて、ジョブ・キュー・プロセス総数 -1 になるまで追加のジョブ・キュー・プロセスが取得されます。
- あるプロセスによって操作されるアクティブ・スケジュールがどれも伝播の必要なメッセージを持っていない場合、そのジョブ・キュー・プロセスは解放されます。
- このアルゴリズムによって、負荷が大きいプロセスから小さいプロセスにスケジュールを転送することで自動ロード・バランスを行い、オーバーロードのプロセスがなくなるようにします。

このスケジューリング・アルゴリズムには、伝播できるジョブ・キュー・プロセスを少なくとも 2 つ使用可能にする必要があるという制限事項があります。非伝播の関連ジョブがあるときは、より多くのジョブ・キュー・プロセスが必要になります。負荷が大きくなる条件（多数のアクティブ・スケジュールがあって、そのすべてに伝播が必要なメッセージがあるとき）が予測されるときは、非伝播の関連ジョブも同時にジョブ・キュー・プロセスを使用することを認識して、より多くのジョブ・キュー・プロセスを開始することをお勧めします。伝播ジョブのみを持っているシステムでは、2 つのジョブ・キュー・プロセスによってすべてのスケジュールを操作できますが、ジョブ・キュー・プロセス数が増えると、メッセージの伝播も速くなります。1 つのジョブ・キュー・プロセスが複数のスケジュールによってメッセージを伝播できるため、ジョブ・キュー・プロセス数はスケジュール数と同じでなくてもよいことに注意してください。

### 伝播中の障害対策

新しいアルゴリズムは、様々な障害に対して強力に対処できます。様々な障害のために、メッセージが伝播されないことがあります。よくある原因としては、「データベース・リンクの障害」、「リモート・データベースが使用不可」、「リモート・キューが存在しない」、「リモート・キューが開始されていない」、「リモート・キューにエンキューしようとしたときのセキュリティ違反」などがあります。このような状況では、適切なエラー・メッセージが `DBA_QUEUE_SCHEDULES` ビューにレポートされます。あるスケジュールでエラーが発生すると、そのスケジュールによるメッセージ伝播は、指数バックオフ・アルゴリズムを使用して最大 16 回まで周期的に試行され、その後は使用不可になります。エラーの原因になっていた問題が解決してスケジュールが使用可能になった場合、最新のエラー日付、時刻およびメッセージ・フィールドにそのエラー情報が残ります。このフィールドがリセットされるのは、そのスケジュールによってメッセージが正常に伝播されたときです。指数バックオフの段階が進むと、伝播が試行されるタイム・スパンが数時間から数日になることがあります。

エラーが長時間見過ごされた場合に、このような問題が発生します。このような状況では、その伝播スケジュールを解除して、再度スケジューリングするようにしてください。

## AQ 伝播問題のデバッグについてのガイドライン

ここでは、ソースおよびターゲット・データベースにキュー・テーブルおよびキューを作成し、接続先データベースに対するデータベース・リンクを定義していると想定します。

このマニュアルに記載されている伝播の説明を参照してください。また、付録 D のトラブルシューティングに関する説明も参照してください。

1. イベント 24040、レベル 10 を使用して、伝播トレースを最高レベルで ON にします。伝播の発生時に、デバッグ情報がジョブ・キュー・トレース・ファイルにログされます。トレース・ファイルで、エラーがないか、またメッセージが送信されたことを示す文を確認できます。

2. データベース 2 に対するデータベース・リンクを確認します。これを行うには、次のように入力します。

```
select count(*) from emp@db2
```

3. 伝播スケジュールが作成され、ジョブ・キュー・プロセスが割り当てられたことを確認します。スケジュールに対する `dba_queue_schedules` および `aq$_schedules` のエントリを検索します。`aq$_schedules` に `jobno` があり、`job$` または `dbms_jobs` にその `jobno` のエントリがあることを確認します。

4. 1 つ以上のジョブ・キュー・プロセスが実行中であることを確認します。Oracle8i では、2 つ以上のジョブ・キュー・プロセスが必要です。

5. 次のように入力して、ソース・キューのメッセージを確認します。

```
select count(*) from where q_name = '';
```

6. 同じように入力して、宛先キューのメッセージを確認します。

7. 他にジョブ・キュー・プロセスを使用しているものがあるかどうかを確認します。他のジョブによって、伝播ジョブの処理時間が失われる可能性がないかどうかを確認します。

8. `sys.aq$_prop_table_x` が `dba_queue_tables` に存在し、また `aq$_prop_notify_x` キューが `dba_queues` ( $x$  は OPS インスタンス番号) に存在することを確認します。これらは、ジョブ・キュー・プロセス間の通信に使用されます。

9. 宛先キューからメッセージをデキューするコンシューマが、伝播されたメッセージの受信者であることを確認します。8.1 形式のキューでは、次のように入力します。

```
select consumer_name, deq_txn_id, deq_time, deq_user_id
propagated_msgid from aq$
where queue = ;
```

8.0 形式のキューでは、次のように入力して、キュー・テーブルの履歴列から同じ情報を取得できます。

```
select h.consumer, h.transaction_id, h.deq_time, h.deq_user
h.propagated_msgid from t, table(t.history) h
where t.q_name = '';
```

または、次のように入力します。

```
select consumer, transaction_id, deq_time, deq_user,
propagated_msgid from
the(select cast(history as sys.aq$_dequeue_history_t)
from where q_name = '');
```

この章では、アドバンスト・キューイングに関して、よくある質問およびそれに対する回答を示します。

### キュー・テーブルに残っているデキュー済メッセージにアクセスする方法は？

SQL を使用してアクセスします。キュー・テーブルのメッセージは、保存されているか、まだ処理されていません。キューごとにビューがあります（10-39 ページの「[データベース全体における状態ごとのメッセージ数の選択](#)」を参照）。

### メッセージの保存とはメッセージがそこにあることですが、これらのメッセージにサブスクライバがアクセスする方法は？

通常、サブスクライバは、デキュー・インタフェースを使用してメッセージにアクセスします。ただし、処理済または待機中のメッセージを参照する場合は、メッセージ ID でデキューするか、または SQL を使用します。

### キュー・テーブル作成後でもソート順序は変更できますか？

キュー・テーブルを作成した後は、メッセージのソート順序は変更できません。

### 例外キューからデキューする方法は？

複数コンシューマ・キューに対する例外キューも、複数コンシューマ・キューである必要があります。

複数コンシューマ・キュー内の期限切れメッセージは、メッセージの対象受信者によってデキューされません。ただし、デキュー・オプションのコンシューマ名を NULL に指定すると、REMOVE モードで 1 回のみデキューできます。メッセージ ID を指定すると、メッセージも例外キューからデキューできます。

---

複数コンシューマ例外キューを作成したキュー・テーブルが、互換パラメータを使用しないで作成されたか、または互換パラメータを 8.0 に設定して作成された場合、期限切れメッセージはメッセージ ID を指定する方法でしかデキューできません。

## 伝播のスケジューリングでは、待ち時間（latency）パラメータは何を意味しますか？

伝播スケジュールで待ち時間を 0（ゼロ）未満に指定すると、ジョブは指定された待ち時間の後に実行されるようにスケジュールが再設定されます。ジョブが実際に実行する時間は、準備ジョブの数や `job_queue_processes` の数など、他の要因に依存します。また、`job_queue_interval` の値にも影響される場合があります。ジョブ・キューおよび SNP バックグラウンド・プロセスの詳細は、『Oracle8i 管理者ガイド』の第 8 章「ジョブ・キューの管理」を参照してください。

## キュー・テーブルを作成する表領域の制御方法は？

DBMS\_AQADM.CREATE\_QUEUE\_TABLE の `storage_clause` パラメータを介して、キュー・テーブルおよびそのすべての補助オブジェクトを格納するための表領域を指定できます。ただし、一度表領域を指定すると、そのキュー・テーブルに対して作成されたすべての索引構成表および索引は、指定された表領域に作成されます。現在のところ、これらを異なる表領域間に分割することはできません。

## OPS インスタンス親和性をキュー・テーブルに対応付ける方法は？

Oracle8i では、OPS インスタンス親和性をキュー・テーブルに対応付けることができます。異なるインスタンスで `q1` および `q2` を使用している場合、キュー・テーブルで `ALTER_QUEUE_TABLE` を使用（`create_queue_table` も可）して `primary_instance` に適切なインスタンス ID を設定できます。

## メッセージ・プロパティ、メッセージ・データ・プロパティを含むサブスクライバ・ルールの例を教えてください。

メッセージ・プロパティを指定する場合の簡単なルールは、`rule = 'priority 1'` です。メッセージ・プロパティおよびデータ属性の組合せを指定するルールの例は、次のとおりです。

```
rule = 'priority 1 AND tab.userdata.sal 1000' rule = '((priority between 0 AND 3)
OR correlation = ''BACK_ORDERS'') AND tab.userdata.customer_name like ''JOHN DOE''')
```

ユーザー・データ・プロパティまたは属性は、オブジェクト・ペイロードのみに適用され、常に接頭辞として `tab.userdata` を付ける必要があります。



---

## 通知登録（OCI）とリスナーの開始は同じですか？

同じではありません。登録とは、非同期通知（プッシュ）に使用される OCI クライアント・コールです。登録では、基本的に、メッセージがデキュー可能になった場合に、サーバーからクライアントへの通知が提供されます。メッセージが使用可能な場合、クライアント側の関数（コールバック）が、サーバーによって起動されます。通知登録は、非ブロックで非ポーリングです。

## 非永続キューの使用目的は何ですか？

現在接続されているすべてのユーザーへの通知のメカニズムを提供します。非永続キューのメカニズムによって、非永続キューに対するメッセージのエンキューがサポートされます。また、OCI 通知は、通知に対して現在登録されているユーザーに、このようなメッセージを配信するために使用されます。

## 受信者リストの長さに制限はありますか？または、特別なキューに対するサブスクライバの数に制限はありますか？

あります。各キューに対するサブスクライバまたは受信者の数は、1024 に制限されています。

## UNDELIVERABLE メッセージが表示されたキューをクリーンアウトする方法は？

これらのメッセージは、msgid でデキューできます。キュー・テーブル・ビューを問い合わせることで msgid を検索できます。メッセージは、最終的に例外キューに移動されます（これを行うには、AQ バックグラウンド・プロセスが実行中である必要があります）。通常のデキュー方法で、例外キューからこれらのメッセージをデキューできます。

## メッセージ・ペイロードは、エンキューした後で更新できますか？

メッセージをデキューし、再度エンキューすることによってのみ更新できます。メッセージ・ペイロードを変更している場合は、これはまったく別のメッセージになります。

## 非同期通知を使用して、新しいメッセージを受信するたびに実行可能ファイルを起動できますか？

通知は、OCI クライアントに対してのみ可能です。クライアントは、通知を受信するためにデータベースに接続している必要はありません。クライアントは、各メッセージに対して実行されるコールバック関数を指定します。非同期通知を使用して実行可能ファイルを起動することはできませんが、コールバック関数でストアード・プロシージャを起動することはできます。

---

## 伝播は複数コンシューマ・キューから単一コンシューマへ（およびその逆で）動作しますか？

複数コンシューマ・キューから単一コンシューマ・キューへの伝播は可能です。その逆は不可能です（単一コンシューマ・キューからの伝播はできません）。

## デキュー時に ORA-01555 エラーが発生する場合があるのはなぜですか？

デキューに対して NEXT\_MESSAGE ナビゲーション・オプションを使用している可能性があります。このオプションでは、最初のデキュー・コール中に作成されたスナップショットが使用されます。その後、他のデキュー・コールによって、UNDO が生成されてロールバック・セグメントに書き込まれ、ORA-01555 エラーが戻されます。

これを解決するには、FIRST\_MESSAGE オプションを使用してメッセージをデキューします。これによってカーソルが再実行され、新しいスナップショットが取得されます。これは、正常に実行しない場合があるため、たとえば、1つのメッセージに FIRST\_MESSAGE を使用し、次の 1000 のメッセージに NEXT\_MESSAGE を使用し、再び FIRST\_MESSAGE を使用するというように、メッセージをバッチでデキューすることをお勧めします。

## パフォーマンスに影響を及ぼさずに、表に含めることができるキューの最大数はいくらですか？

表内のキューの数によって、パフォーマンスに影響を受けることはありません。

## 伝播を使用してメッセージをあるキューから別のキューに移動する場合、メッセージを1つずつ移動するのではなく、バッチで移動するように最適化されていますか？

はい。これは最適化されており、伝播はバッチで実行されます。

また、リモート・キューが異なるデータベースにある場合、2 フェーズ・コミットの必要性を回避するために順序アルゴリズムを使用します。

メッセージを同じ宛先の複数キューに送信する必要がある場合、メッセージは、複数回送信されます。メッセージがその宛先の同じキューにある複数のコンシューマに送信される必要がある場合、メッセージは 1 回で送信されます。

## サブスクライバ表に記録されているサブスクライバ・タイプの違いは何ですか？

サブスクライバ・タイプおよびその値は、次のとおりです。

1 - Current Subscriber。サブスクライバの名前、アドレスおよびプロトコルは同じ行にあります。

---

2 - Ex subscriber。サブスクライブしていないが、履歴 `aq$_queuetable_h` IOT にエージェント・エントリを持っていたサブスクライバです。

4 - Address。受信者のアドレスを格納するために使用されます。名前は常に NULL です。アドレスは常に NULL 以外です。

8 - Proxy for Propagation。名前は常に NULL です。

ローカル・キューへのデータベース・プロキシの場合、`address=NULL`、`protocol=0` です。

リモート・キューへのデータベース・プロキシの場合、`address=dblink` アドレス、`protocol=0` です。

サード・パーティ・プロキシの場合、`address = NULL`、`protocol = サード・パーティ・プロトコル`です。

### **メッセージが例外キューに移動した後、SQL またはその他を使用して、例外キューに移動する前にメッセージが常駐していたキューを識別する方法はありますか？**

ありません。AQ では、このような情報は提供されません。この問題を回避するために、アプリケーションでそのメッセージの情報を保存できます。

### **多くのメッセージが同時にエンキューされている場合、メッセージはどのような順序でデキューされますか？**

メッセージの `enq_time` が同じである場合、(同じ `enq_time` を持つ各メッセージに対して) 一定の単位で増加する `step_no` という別のフィールドがあります。これが、メッセージ順序のメンテナンスに有効です。同じセッションからエンキューされた 2 つ以上のメッセージに対して、`enq_time` および `step_no` の両方が同じであることはありません。

### **dbms\_aqadm.add\_subscriber および dbms\_aqadm.remove\_subscriber コールが発行された同じキューで、同時エンキューまたはデキューが実行された場合に、これらのコールがハングする場合がありますのはなぜですか？**

`add_subscriber` および `remove_subscriber` は、キューにおける管理操作です。AQ は、アプリケーションが管理コールおよび操作コールを同時に発行することを防止することはありませんが、これらの発行はシリアルに実行されます。メッセージをエンキューまたはデキューした保留トランザクションがコミットおよび保持するリソースを解放するまで、`add_subscriber` および `remove_subscriber` はブロックします。サブスクライバの追加および削除は、頻繁に発生しないと想定されています。これは、ほとんどの場合、アプリケーションの設定の一部として行われます。これを解決するには、キューで他の操作が行われていないときに、設定またはクリーン・アップ・フェーズで、`add_subscriber` および `remove_subscriber` へのコールを分離します。これによって、操作コールによるリソースの解放を、これらのコールがブロック状態のまま待機し続けることがなくなります。

---

## TopicSession.createDurableSubscriber および TopicSession.unsubscribe コールによって、「ORA - 04020: オブジェクトをロックしようとしてデッドロックを検出しました。」というメッセージで JMS 例外が発生するのはなぜですか？

createDurableSubscriber および unsubscribe コールには、Topic への排他的アクセスが必要です。そのため、これらのコールが発行される前に、同じ Topic に対する保留 JMS 操作（送信 / 公開 / 受信）がある場合、ORA - 04020 例外が発生します。

この問題を解決するには、次の 2 つの方法があります。

1. Topic に対して他の JMS 操作が行われていないときに、設定またはクリーン・アップ・フェーズで、createDurableSubscriber および unsubscribe に対するコールを分離します。これによって、必要なリソースが他の JMS 操作コールによって保持されないようになります。したがって、ORA - 04020 エラーは発生しません。
2. createDurableSubscriber および unsubscribe コールをコールする前に、TopicSession.commit コールを発行します。

## AQ\_ADMINISTRATOR\_ROLE または AQ\_USER\_ROLE が、Java/JMS API を使用する AQ アプリケーションに対して機能しない場合があるのはなぜですか？

ロールの付与に加えて、次のように、パッケージに対する実行権限もユーザーに付与する必要があります。

- `grant execute on sys.dbms_aqin to <userid>`
- `grant execute on sys.dbms_aqjms to <userid>`

## Oracle8i JServer 内の Java ストアド・プロシージャから JMS MessageListeners を使用すると、java.security.AccessControlException が発生するのはなぜですか？

Oracle8i JServer 内の MessageListeners を使用するには、次のいずれかを実行します。

1. `GRANT JAVA_SYSPRIV to <userid>`
2. `call dbms_java.grant_permission ('JAVASYSPRIV',  
 'SYS:java.net.SocketPermission',  
 '*', 'accept,connect,listen,resolve');`

---

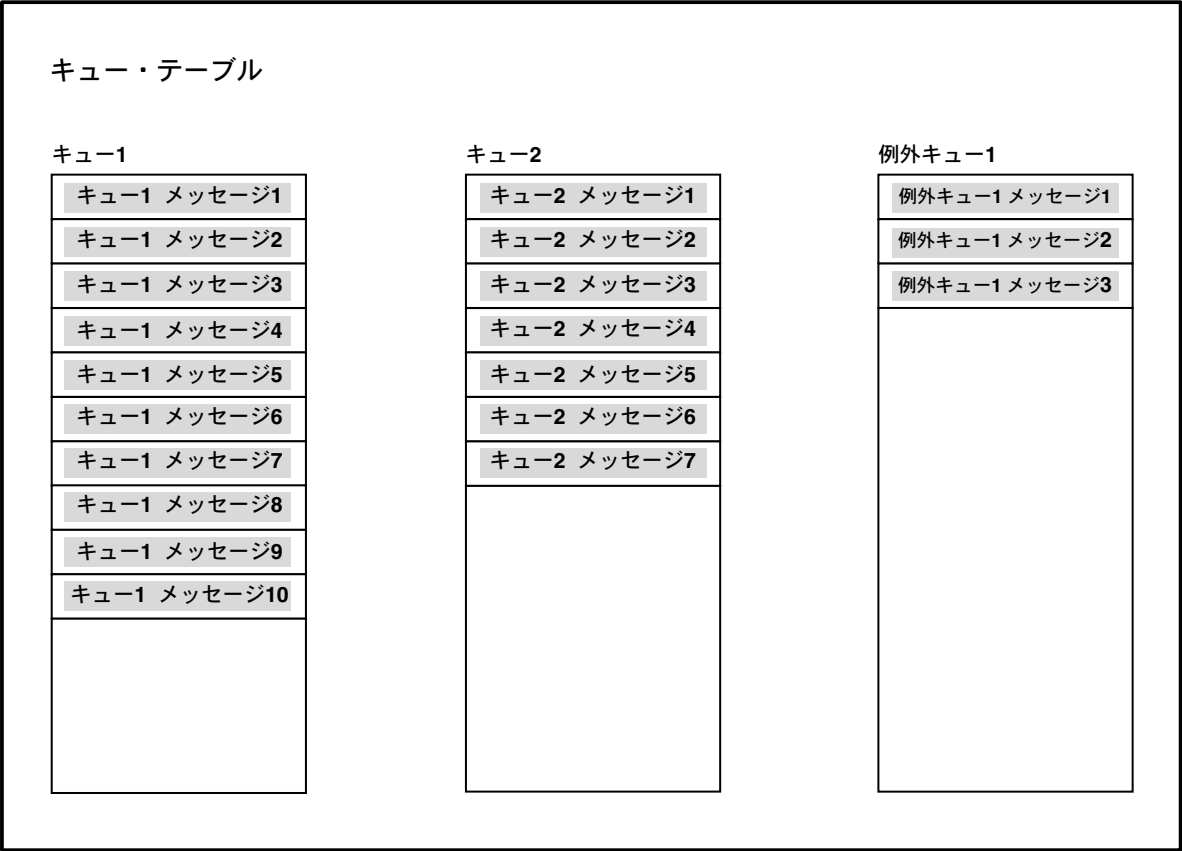
## モデリングおよび設計

この章では、アドバンスド・キューイングのモデリングおよび設計の基礎について説明します。内容は次のとおりです。

- 基本キューイング
- 基本キューイングの説明
- AQを使用したクライアント / サーバー通信の例
- 複数コンシューマによる同一メッセージのデキュー
- 複数コンシューマによる同一メッセージのデキューの例
- 指定された受信者による指定されたメッセージのデキュー例
- AQを使用したワークフローの実装例
- AQを使用したパブリッシュ / サブスクライブの実装例
- メッセージの伝播
- メッセージの伝播の例

# キュー・エンティティのモデリング

図 7-1 基本キュー



この図には、2つのキューおよび1つの例外キューを含むキュー・テーブルがあります。

- キュー 1 には、10 個のメッセージが含まれます。
- キュー 2 には、7 個のメッセージが含まれます。
- 例外キュー 1 には、3 個のメッセージが含まれます。

## 基本キューイング

### 基本キューイング・プロデューサ、コンシューマがともに1つの場合

最も基本的な場合、1つのプロデューサが、1つのキューに複数の異なるメッセージをエンキューします。各メッセージは、1つのコンシューマによって1回のみデキューされ、処理されます。メッセージは、コンシューマによってデキューされるか、または期限が切れるまでは、キュー内に格納されたままです。プロデューサは、メッセージが使用可能になるまでの遅延および期限切れを指定できます。同様に、デキュー時に使用可能なメッセージがない場合は、コンシューマは待機できます。エージェント・プログラムまたはアプリケーションは、プロデューサおよびコンシューマの両方として動作できる点に注意してください。

### 基本キューイング・プロデューサが複数で、コンシューマが1つの場合

やや複雑な場合、多数のプロデューサがメッセージをキューにエンキューし、そのすべてのメッセージが1つのコンシューマによって処理されることがあります。

### 基本キューイング・不連続なメッセージでプロデューサおよびコンシューマがともに複数の場合

次のステージでは、多数のプロデューサがメッセージをエンキューし、それぞれのメッセージを型および相関識別子によって異なるコンシューマが処理する場合を考えます。次の図で、このシナリオを説明します。

## 基本キューイングの説明

図 7-2「[基本キューイングのモデル](#)」に、メッセージがエンキューおよびデキューされる1つのキューを含むキュー・テーブルを示します。

### プロデューサ

この図では、メッセージのプロデューサが6つあり、そのうちの4つのみが示されています。つまり、他の2つのプロデューサ（P4 および P5）は、メッセージをエンキューする権利はあるが、図に示された時点ではメッセージをエンキューしていないと仮定しています。図では、次のことを示しています。

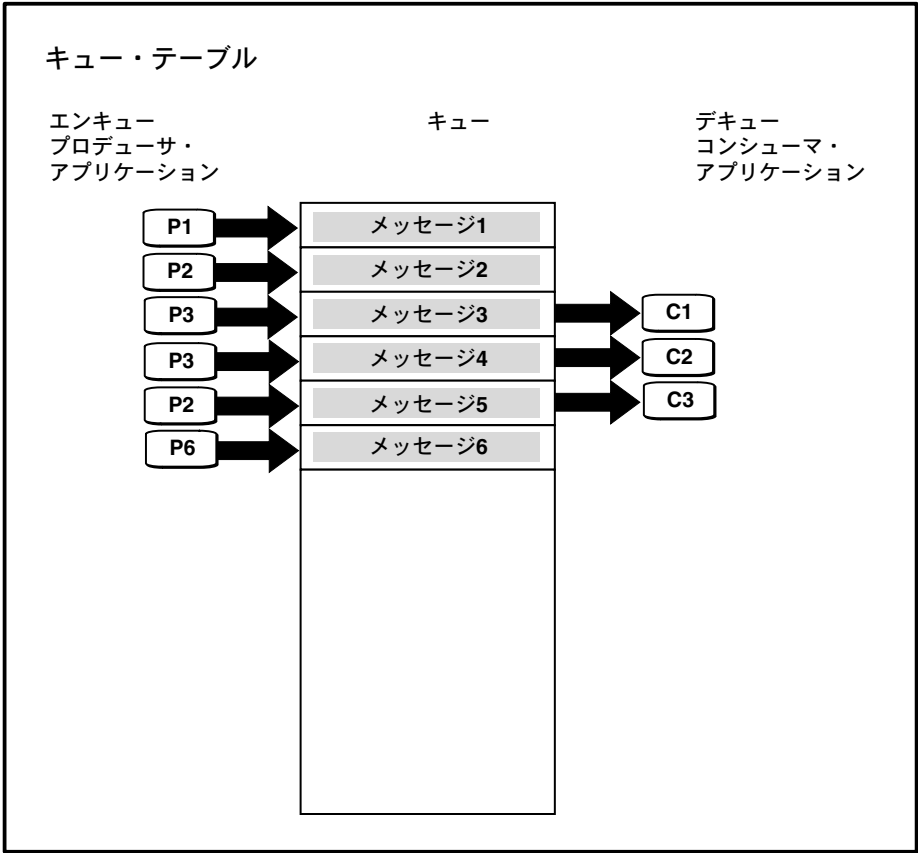
- 1つのプロデューサが1つ以上のメッセージをエンキューできます。
- プロデューサは、メッセージをどの順序でもエンキューできます。

コンシューマ

この図では、メッセージのコンシューマが3つあり、コンシューマはこれがすべてです。図では、次のことを示しています。

- メッセージは、エンキューされた順序でデキューする必要はありません。
- メッセージは、デキューされなくてもエンキューできます。

図 7-2 基本キューイングのモデル



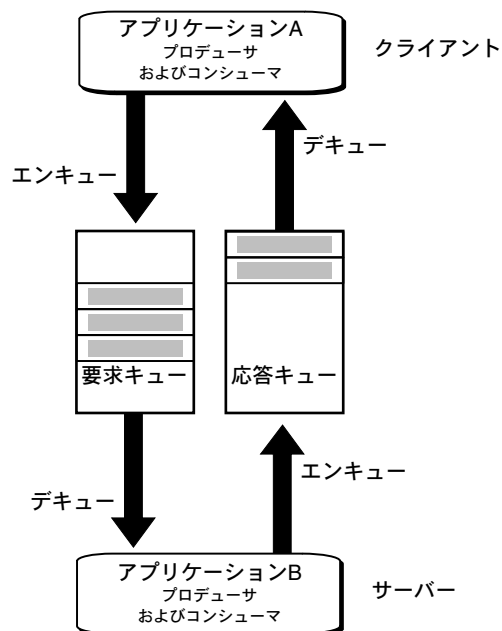


## AQ を使用したクライアント / サーバー通信の例

前述の図では、一連のプロデューサによって複数のメッセージがエンキューされ、一連のコンシューマによってメッセージがデキューされることを示しました。この図で明らかにされていないのは、時間の概念および Oracle AQ を使用するメリットです。

クライアント / サーバー・アプリケーションは、通常、同期方式で実行され、密結合によって発生するデメリットがあります。図 7-3 「AQ を使用したクライアント / サーバー通信」に、AQ を使用した非同期の代替案を示します。この例では、アプリケーション B（サーバー）は、要求 / 応答キューを使用して、アプリケーション A（クライアント）にサービスを提供しています。

図 7-3 AQ を使用したクライアント / サーバー通信



1. アプリケーション A は、要求を要求キューにエンキューします。
2. アプリケーション B は、要求をデキューします。
3. アプリケーション B は、要求を処理します。
4. アプリケーション B は、その結果を応答キューにエンキューします。

### 5. アプリケーション A は、その結果を応答キューからデキューします。

このように、クライアントはサーバーに対する接続の確立を待つ必要はなく、サーバーは独自の処理タイミングでメッセージをデキューします。サーバーによるメッセージ処理が終了したとき、クライアントは結果を受け取るために待つ必要はありません。このような二重遅延処理によって、クライアントおよびサーバーの両方が自由に処理できるようになります。

---

**注意：** 様々なエンキューおよびデキューの操作は、異なるトランザクションの一部です。

---

## 複数コンシューマによる同一メッセージのデキュー

メッセージは、一度に1つのキューにしか入れることができません。複数のコンシューマに届けるために、プロデューサから同じメッセージを複数のキューに挿入する必要がある場合には、非常に多くのキューを管理する必要があります。Oracle AQ では、複数のコンシューマから同じメッセージをデキューできるようにするため、キューのサブスクライバおよびメッセージの受信者という2つのメカニズムを提供しています。これを利用する場合、キューを常駐させるキュー・テーブルは、サブスクライバ・リストおよび受信者リストの余裕を取るために、複数コンシューマ・オプションを使用して作成する必要があります。各メッセージは、所定のすべてのコンシューマによって処理されるまで、キュー内にとどまります。

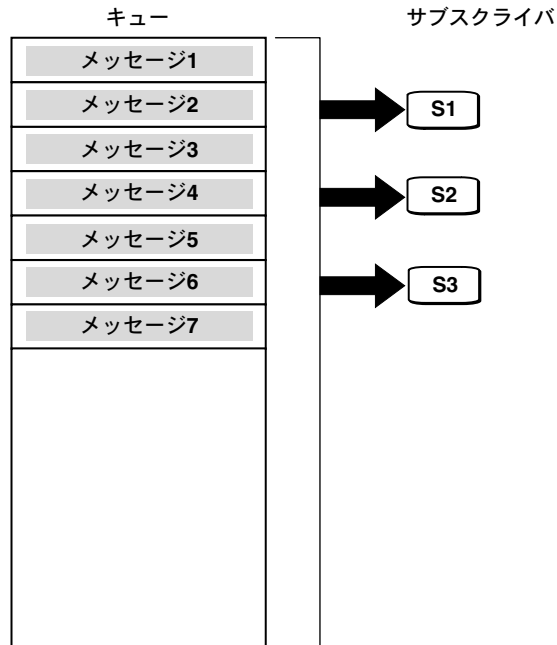
**キューのサブスクライバ** この方法を使用することで、複数のコンシューマ・サブスクライバを1つのキューに対応付けることができます。これによって、キューにエンキューされたすべてのメッセージを、それぞれのキューのサブスクライバから使用できるようになります。キューに対応するサブスクライバは、メッセージまたはメッセージ・プロデューサを一切変更することなく、動的に変更できます。Oracle AQ 管理パッケージを使用することで、キューに対応するサブスクライバを追加および削除できます。次の図で、メッセージが複数のプロデューサによってキューにエンキューされ、それぞれのメッセージが複数のコンシューマ・サブスクライバによって処理される様子を説明します。

**メッセージの受信者** メッセージ・プロデューサは、メッセージがエンキューされた時点で受信者リストを送ることができます。これによって、キュー内の各メッセージに、受信者の一意の集合を対応付けることができます。メッセージに対応付けられた受信者リストは、キューに対応付けられたサブスクライバ・リストが存在する場合には、それをオーバーライドします。受信者は、サブスクライバ・リストに含まれている必要はありません。ただし、受信者はサブスクライバの中から選択できます。

図 7-4 複数コンシューマによる同一メッセージのデキュー

キュー・テーブル

サブスクライバ・リスト: s1、s2、s3



## 複数コンシューマによる同一メッセージのデキューの例

図 7-4 では、3 つすべてのコンシューマがキューのサブスクライバとしてリストされている場合について説明します。これは、3 つのコンシューマが、そのキューにエンキューされるすべてのメッセージをサブスクライブしているということです。図には、いくつかの重要な点が表示されています。

- 3 つのコンシューマが、すでにエンキューされている 7 つのメッセージに対するサブスクライバであり、まだエンキューされていないメッセージのサブスクライバになる可能性があります。
- どのメッセージも、最終的にそれぞれのサブスクライバによってデキューされます。

- サブスクライバの間に優先順位はありません。これは、どのサブスクライバがどのメッセージをどのような順番でデキューするかはわからないということです。つまり、サブスクライバによってデキューされる順序は決定されていません。
- この図からは、メッセージがすでにデキューされたのか、そしてキューから削除されたのかについては判断できません。

図 7-5 複数コンシューマ・キューを使用した通信

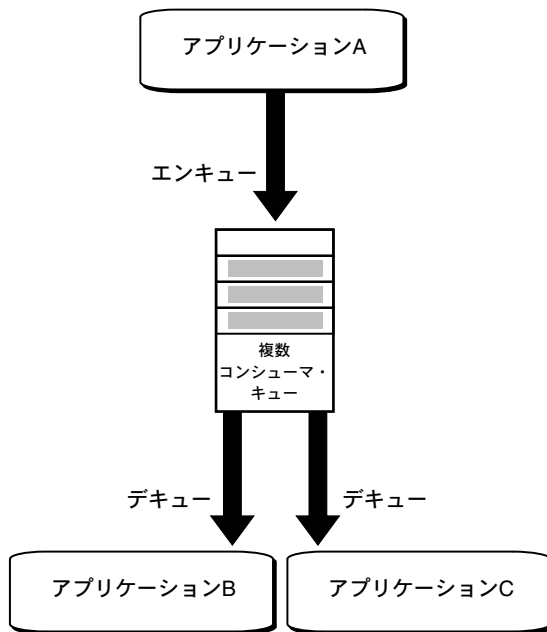
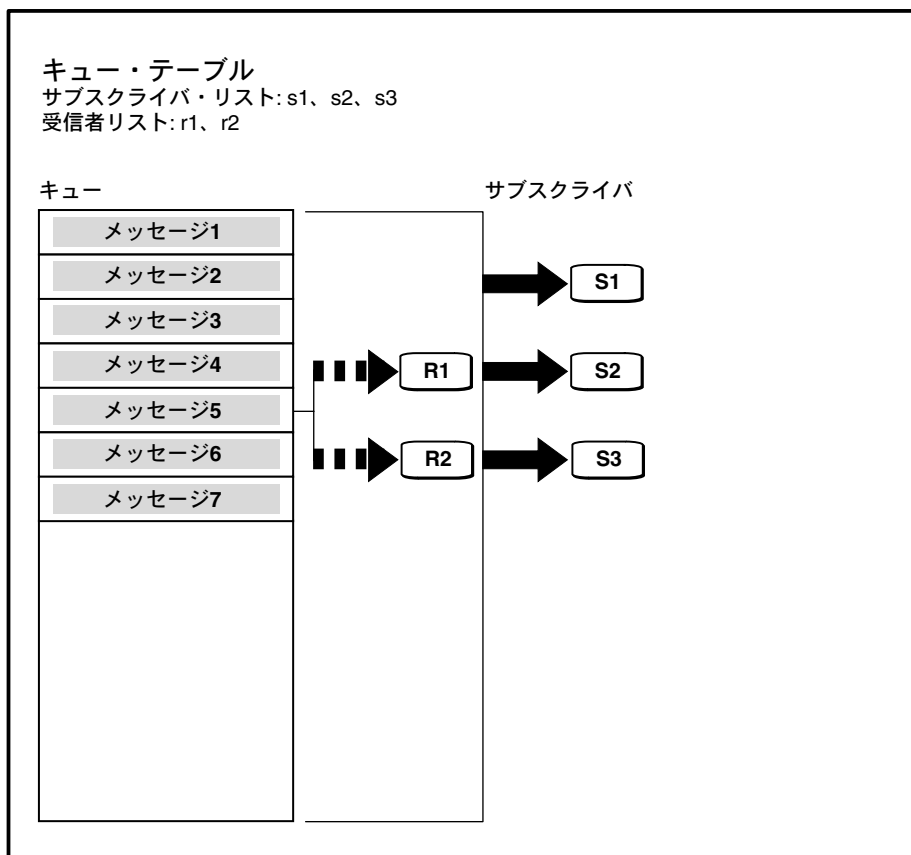


図 7-5 では、同じテクノロジーを動的な観点から示しています。この例では、あるアプリケーションの作成結果が複数のアプリケーションで必要になるシナリオを示しています。アプリケーション A によってエンキューされたすべてのメッセージは、アプリケーション B およびアプリケーション C によってデキューされます。これは、アプリケーション B およびアプリケーション C をキュー・サブスクライバとする複数のコンシューマ・キューが、特別に設定されて可能になります。結果的に、これらのアプリケーションが、暗黙的にキューに格納されるすべてのメッセージの受信者になります。

**注意：** アプリケーションまたは他のキューが、キュー・サブスクライバになることができます。

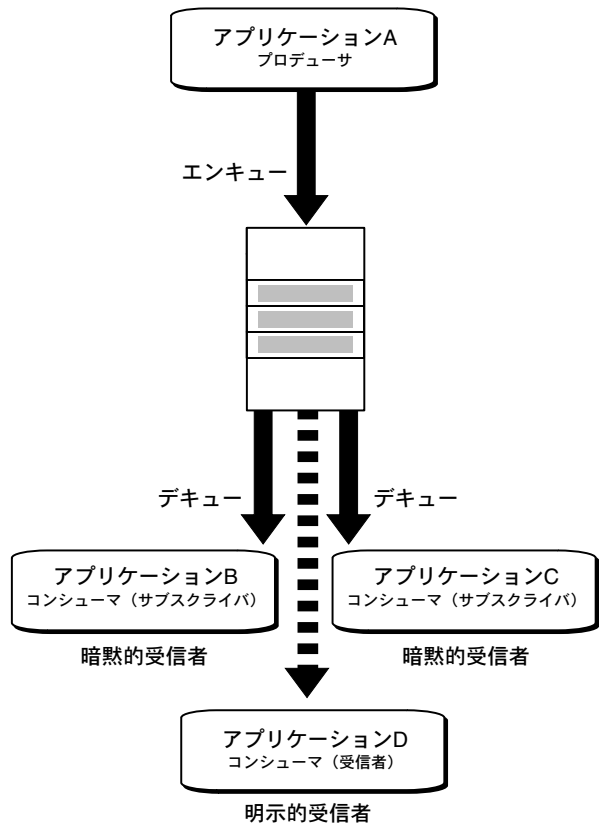
図 7-6 指定された受信者による指定されたメッセージのデキュー



## 指定された受信者による指定されたメッセージのデキュー例

図 7-6 は、どのようにしてメッセージを 1 人以上の受信者に指定できるかを示しています。この場合、メッセージ 5 は、受信者 1 および受信者 2 によってデキューされるように指定されています。この図が示すように、どの受信者も、キューに対する 3 つのサブスクライバの 1 つではありません。

図 7-7 メッセージの明示的および暗黙的な受信者



前述のように、サブスクライバは、特定のキューに格納されたすべてのメッセージをデキューできるという点から、「暗黙的な受信者」といわれます。これは、雑誌の購読契約をすることで暗黙的にすべての記事にアクセスできることと似ています。受信者といわれるコンシューマのカテゴリは、特定のメッセージのターゲットに指定されるという点から「明示的な受信者」といわれます。

図 7-7 に、Oracle AQ が動的にどのように調整され、両方の種類のコンシューマに対応するかを示します。このシナリオでは、アプリケーション B およびアプリケーション C は、暗黙的受信者（サブスクライバ）です。ただし、メッセージは、キューのサブスクライバであるかどうかにかかわらず、特定のコンシューマ（サブスクライバ）に明示的に転送することもできます。このような受信者リストは、そのメッセージのエンキュー・コールで指定され、そのキューのサブスクライバのリストをオーバーライドします。図では、アプリケーション D は、アプリケーション A によってエンキューされたメッセージの唯一の受信者に指定されています。

---

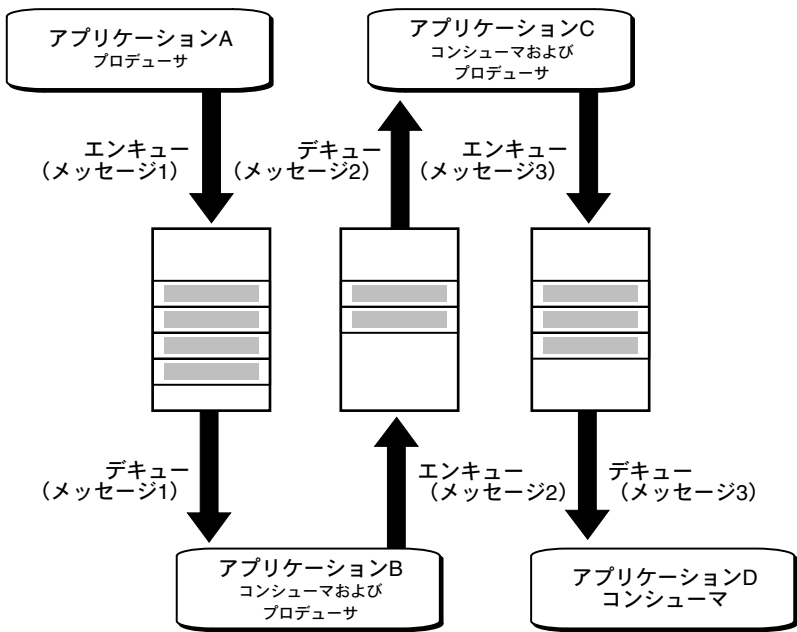
**注意：** 複数のプロデューサによって、異なる受信者に宛てられたメッセージが同時にエンキューされることがあります。

---

## AQ を使用したワークフローの実装例

図 7-8 では、ワークフロー（連鎖したアプリケーション・トランザクションともいいます）を実装するために AQ が使用されています。ワークフローは、アプリケーション A、B、C および D によって実行される 4 つのステップで構成されています。キューは、各業務処理の段階で相互に流れる情報をバッファするために使用されます。メッセージの遅延間隔および期限切れを指定することで、各アプリケーションに実行枠を設定できます。

図 7-8 AQ を使用したワークフローのインプリメント



ワークフローの観点からみると、メッセージの受渡しは、ペイロード・データの価値以上に業務資産となります。そのため、AQ では、過去の傾向を分析して将来の動向を予測するためにメッセージの保存がオプションとしてサポートされています。たとえば、この章のはじめに示した3つのアプリケーション・シナリオのうち、2つは実際のワークフロー分析の実装を基礎にしています。

**注意：** メッセージ1、2および3の内容は同一の場合も異なる場合もあります。異なる場合でも、メッセージに前のメッセージの内容が一部含まれることがあります。

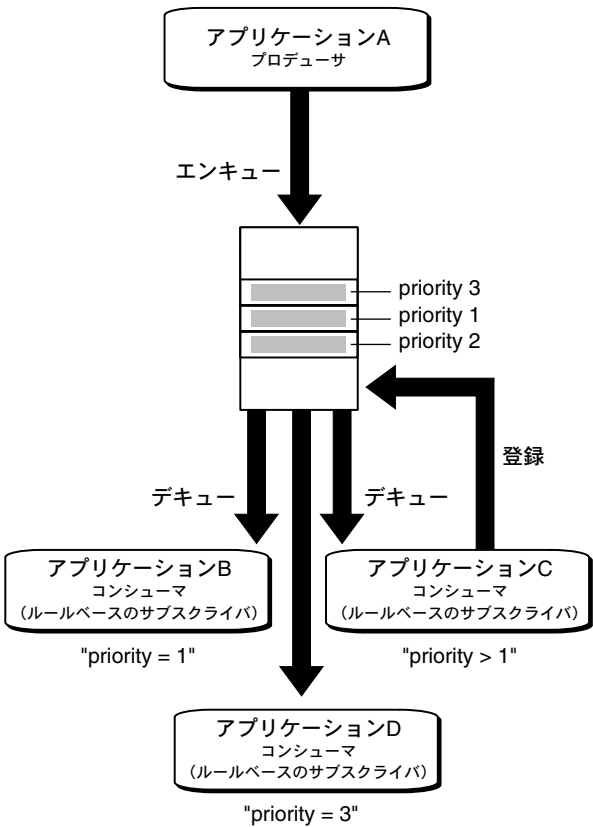


## AQ を使用したパブリッシュ / サブスクライブの実装例

図 7-9 では、アプリケーション間のメッセージのパブリッシュ / サブスクライブ・スキームの実装のために AQ が使用されています。アプリケーション A は、メッセージをキューに公開（パブリッシュ）するパブリッシャ・アプリケーションです。アプリケーション B、アプリケーション C およびアプリケーション D は、サブスクライバ・アプリケーションです。アプリケーション A は、匿名でメッセージをキューに公開します。すると、それらのメッセージは、アプリケーションごとに指定されたルールに従ってサブスクライバ・アプリケーションに配信されます。サブスクライバ・アプリケーションは、メッセージ・プロパティおよびメッセージ・データの内容に対してルールを定義して、関心のあるメッセージを指定できます。

この例では、アプリケーション B は「priority=1」ルールでサブスクライブし、アプリケーション C は「priority > 1」ルールでサブスクライブし、アプリケーション D は「priority = 3」ルールでサブスクライブしています。アプリケーション A は、3 個のメッセージ（優先順位はそれぞれ 3、1、2）をエンキューします。アプリケーション B は、1 個のメッセージ（優先順位が 1）を受信し、アプリケーション C は、2 個のメッセージ（優先順位が 2、3）を受信し、アプリケーション D は、1 個のメッセージ（優先順位が 3）を受信します。このように、メッセージ受信者はメッセージ・プロパティおよび内容に基づいて動的に計算されます。さらに、図にはアプリケーション C がどのようにメッセージ配信の非同期式通知を使用するかも示されています。アプリケーション C はキューのメッセージに登録しています。メッセージが届くと、そのことが通知されるため、アプリケーション C はメッセージをデキューできます。

図 7-9 AQ を使用したパブリッシュ / サブスクライブのインプリメント



ワークフローの観点からみると、メッセージの受渡しは、ペイロード・データの価値以上に業務資産となります。そのため、AQでは、過去の傾向を分析して将来の動向を予測するためにメッセージの保存がオプションとしてサポートされています。たとえば、この章のはじめに示した3つのアプリケーション・シナリオのうち、2つはワークフロー分析の実装を基礎にしています。

## メッセージの伝播

### メッセージのファンアウト

AQ では、メッセージの受信者はコンシューマまたは他のキューのどちらかです。キューの場合には、実際の受信者は、そのキューに対するサブスクライバ（これが他のキューの場合もある）によって決定されます。これによって、全員が1つのキューからメッセージのデキューをしなくても、多くの受信者にメッセージをファンアウトできます。

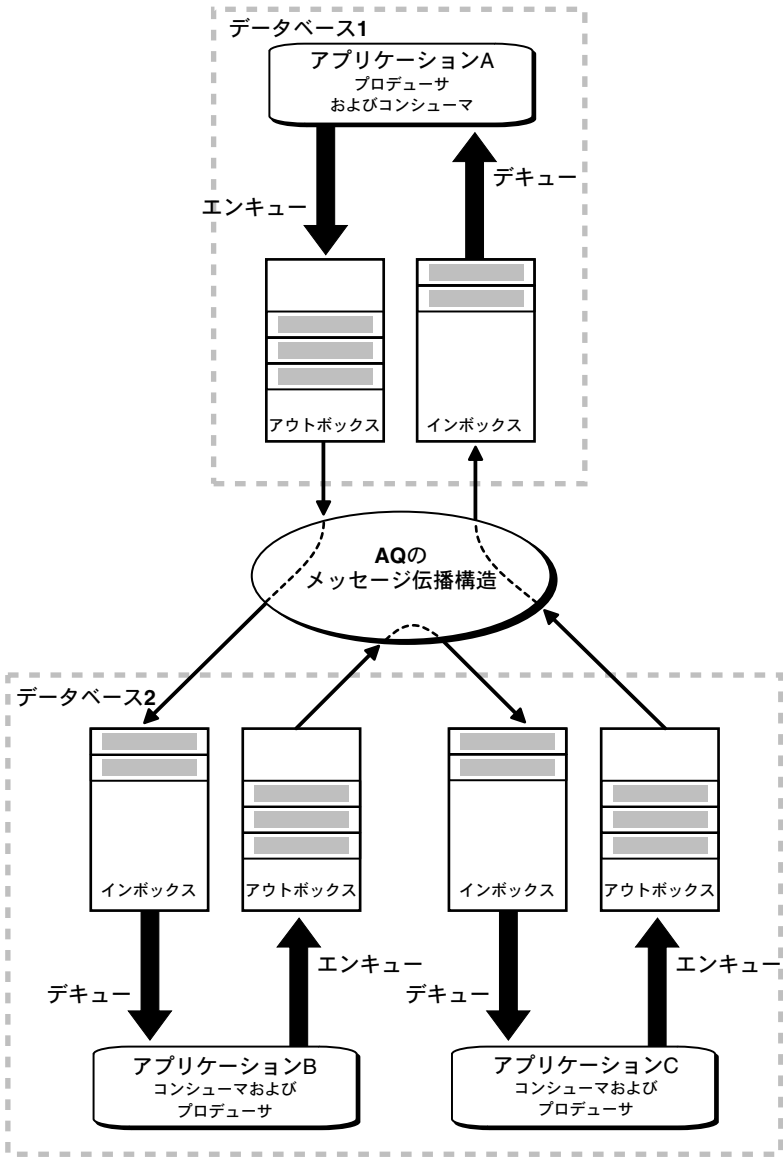
たとえば、ソース・キューが、サブスクライバ・キューとして `dispatch1@dest1` および `dispatch2@dest2` を持つことがあります。次にキュー `dispatch1@dest1` が、サブスクライバとしてキュー `outerreach1@dest3` および `outerreach2@dest4` を持ち、キュー `dispatch2@dest2` のサブスクライバとしてキュー `outerreach3@dest21` および `outerreach4@dest4` を持つことがあります。このようにして、ソース・キューにエンキューされたメッセージは、4つの異なるキューのサブスクライバに伝播します。

### メッセージのファネルイン

メッセージの受信者としてのキューの使用方法には、他に、異なるキューのメッセージを単一のキューに結合する機能があります。このプロセスは「コンポジット」といわれることもあります。

たとえば、キュー `composite@endpoint` がキュー `funnel1@source1` および `funnel2@source2` の両方に対するサブスクライバである場合には、キュー `composite@endpoint` に対応するサブスクライバは、そのキュー自身に直接エンキューされるメッセージのみでなく、残り2つのキューにエンキューされるすべてのメッセージを取得できます。

図 7-10 メッセージの伝播



## メッセージの伝播の例

図 7-10 は、異なるデータベース上のアプリケーションが AQ を介して通信している様子を示しています。各アプリケーションには、受信メッセージおよび送信メッセージを処理するためのインボックスおよびアウトボックスがあります。アプリケーションは、メッセージが送信されるアプリケーションが、ローカル（同一ノード上）かまたはリモート（異なるノード上）にかかわらず、メッセージをアウトボックスにエンキューします。

同様に、アプリケーションでは、メッセージがローカルまたはリモートで作成されたかどうかは問題になりません。すべての場合に、アプリケーションは、そのインボックスにあるメッセージをデキューします。

Oracle AQ はメッセージを同じ基礎で扱い、この交換を容易にします。



---

## AQ を使用したサンプル・アプリケーション

第1章では、架空の会社 BooksOnLine を基にメッセージ・システムを説明しました。この章では、そのシナリオに基づいたサンプル・アプリケーションに沿って、AQ の機能を検討します。

- サンプル・アプリケーション
- 一般機能
  - システム・レベルのアクセス制御
  - 構造化ペイロード
  - キュー・レベルのアクセス制御
  - 非永続キュー
  - 保存およびメッセージ履歴
  - パブリッシュ / サブスクライブ・サポート
  - Oracle Parallel Server (OPS) のサポート
  - 統計ビューのサポート
- ENQUEUE 機能
  - サブスクリプションおよび受信者リスト
  - メッセージの優先順位および順序付け
  - 時間指定 : 遅延
  - 時間指定 : 期限切れ
  - メッセージのグループ化
  - 非同期通知

- 
- DEQUEUE 機能
    - デキューの方法
    - 複数の受信者
    - ローカルおよびリモートの受信者
    - デキューにおけるメッセージ・ナビゲーション
    - デキューのモード
    - メッセージ到着待機の最適化
    - 遅延間隔をおいての再試行
    - 例外処理
    - ルールベースのサブスクリプション
    - Listen 機能
  - 伝播機能
    - 伝播
    - 伝播スケジュール
    - LOB 属性を伴うメッセージの伝播
    - 拡張伝播スケジュール機能
    - 伝播中の例外処理



## サンプル・アプリケーション

大型の書籍販売店である BooksOnLine の運営は、販売プロセス全体にかかわる様々な部門にまたがるアクティビティを自動化する、オンライン書籍発注システムが基礎となっています。システムのフロント・エンドは、新しい注文を入力するための注文入力アプリケーションです。入力された注文は、注文処理アプリケーションによって妥当性チェックおよび記録されます。地域倉庫に置かれた出荷部門は、発注された書籍をすぐに確実に発送する責任があります。地域倉庫は3箇所、それぞれが東部地域、西部地域、そして海外からの発注に対応しています。発注された書籍の発送完了後、発注情報は支払処理のために中央の請求部門に送られます。各地の顧客サービス部門は、発注情報の状況を管理し、発注に関する問合せを処理する責任があります。

第1章では、架空の会社 BooksOnLine を基にメッセージ・システムの概要を示しました。この章では、そのシナリオに基づいたサンプル・アプリケーションに沿って、AQ の機能を検討します。このサンプル・アプリケーションは、Oracle AQ の機能を説明する目的でのみ作成されたものです。この統合化シナリオを作成するにあたって、単一のコンテキストに絞った説明をすることで、このテクノロジーの可能性を理解しやすくするよう努めました。付録には、コードの完全なスクリプトも記載されています（付録 C「BooksOnLine 用スクリプト」を参照）。ただし、考えられる AQ のすべての応用例を、単一の比較的小さいコード・サンプルの範囲内で示すことは不可能であることを理解しておいてください。

## 一般機能

- システム・レベルのアクセス制御
- 構造化ペイロード
- キュー・レベルのアクセス制御
- 非永続キュー
- 保存およびメッセージ履歴
- パブリッシュ / サブスクライブ・サポート
- Oracle Parallel Server (OPS) のサポート
- 統計ビューのサポート

## システム・レベルのアクセス制御

Oracle8i は、すべてのキューイング操作に対してシステム・レベルのアクセス制御をサポートします。この機能によって、アプリケーション設計者または DBA は、ユーザーをキュー管理者にできます。キュー管理者は、データベース上のどのキューに対しても、すべての AQ インタフェース（管理および操作）を起動できます。これによって、データベース上のキュー全体に対するすべての管理スクリプトを 1 つのスキーマで管理できるため、管理作業が単純になります。詳細は、4-4 ページの「[セキュリティ](#)」を参照してください。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・シナリオおよびコード

BooksOnLine のアプリケーションでは、DBA がデータベースのキュー管理者として BOLADM (BooksOnLine の管理者アカウント) を作成します。これによって、BOLADM はデータベース上のすべてのキューを作成、削除、管理および監視できるようになります。どのアプリケーションからでもエンキューまたはデキューできるように BOLADM スキーマに PL/SQL パッケージを作成する場合は、BOLADM にシステム権限 ENQUEUE\_ANY および DEQUEUE\_ANY を付与する必要があります。

```
CREATE USER BOLADM IDENTIFIED BY BOLADM;
GRANT CONNECT, RESOURCE, aq_administrator_role TO BOLADM;
GRANT EXECUTE ON dbms_aq TO BOLADM;
GRANT EXECUTE ON dbms_aqadm TO BOLADM;
EXECUTE dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','BOLADM',FALSE);
EXECUTE dbms_aqadm.grant_system_privilege('DEQUEUE_ANY','BOLADM',FALSE);
```

Java AQ API を使用する場合、ユーザーには DBMS\_AQIN パッケージの実行権限が必要です。

```
GRANT EXECUTE ON DBMS_AQIN to BOLADM;
```

このアプリケーションでは、AQ のプロパゲータは OE (注文入力) スキーマから WS (西部売上)、ES (東部売上) および OS (海外売上) スキーマにメッセージを移入します。次に、WS、ES および OS スキーマから CB (顧客請求) および CS (顧客サービス) スキーマにメッセージを移入します。したがって、OE、WS、ES および OS スキーマのすべてに、プロパゲータのソース・キューとなるキューがあります。

メッセージが宛先キューに届くと、ソース・キューのスキーマ名に基づいたセッションによって、新しく届いたメッセージが宛先キューにエンキューされます。つまり、ソース・キューのスキーマに、宛先キューに対するエンキュー権限を付与する必要があります。

管理を単純化するために、BooksOnLine アプリケーションでソース・キューを持つすべてのスキーマに ENQUEUE\_ANY システム権限を付与します。

```
EXECUTE dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','OE',FALSE);
```

```
EXECUTE dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','WS',FALSE);  
EXECUTE dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','ES',FALSE);  
EXECUTE dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','OS',FALSE);
```

リモートの宛先キューに伝播するために、エージェント構造体のアドレス・フィールドのデータベース・リンクに指定されたログイン・ユーザーには、ENQUEUE ANY QUEUE 権限を付与するか、または宛先キューに対するエンキュー権限を付与する必要があります。ただし、データベース・リンクのログイン・ユーザーが宛先のキュー・テーブルを所有している場合は、どのような明示的な権限も付与する必要はありません。

### **Visual Basic (OO4O) : サンプル・コード**

この機能については、データベースの dbexecutesql インタフェースを使用します。

### **Java (JDBC) : サンプル・コード**

今回のリリースでは、例は記載していません。

## 構造化ペイロード

Oracle AQ では、メッセージのペイロードを構造化して管理するためにオブジェクト型を使用できます。オブジェクト・リレーショナル・データベース・システム (ORDBMS) は、一般的にメッセージ・システムよりも豊富な型指定ができます。Oracle8i のオブジェクト・リレーショナル機能は、伝統的なリレーショナル・データ型からユーザー定義型にいたる豊富なデータ型セットを備えています (A-53 ページの「[PL/SQL を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー](#)」を参照)。

強力な型指定を持つ内容 (外部の型指定システムによって定義された形式を持つ内容) によって、次のような多くの高性能な機能が使用できます。

- 内容ベースのルーティング: 外部エージェントが内容を調査し、その内容に基づいてメッセージを別のキューにルーティングできます。
- 内容ベースのサブスクリプション: パブリッシュ / サブスクライブ・システムがメッセージ・システム上に組み込まれており、内容ベースのサブスクリプションを提供できます。
- 問合せ: メッセージ内容の問合せ機能によって、メッセージ・ウェアハウジングを含む、様々なアプリケーションで必要な現行および処理済のメッセージを可能にします。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、書籍注文データをメッセージ内容としてモデル化するために豊富なデータ型セットが利用されます。

- 顧客は、customer\_typ というオブジェクト型にモデル化されます。

```
CREATE OR REPLACE TYPE customer_typ AS OBJECT (
    custno          NUMBER,
    name            VARCHAR2(100),
    street          VARCHAR2(100),
    city            VARCHAR2(30),
    state           VARCHAR2(2),
    zip             NUMBER,
    country         VARCHAR2(100));
```

- 書籍は、book\_typ というオブジェクト型にモデル化されます。

```
CREATE OR REPLACE TYPE book_typ AS OBJECT (
    title          VARCHAR2(100),
    authors        VARCHAR2(100),
    ISBN           NUMBER,
    price          NUMBER);
```

- 注文明細項目を表す注文項目は、`orderitem_typ` というオブジェクト型にモデル化されます。注文項目は、書籍型を含むネストされた型になります。

```
CREATE OR REPLACE TYPE orderitem_typ AS OBJECT (  
    quantity      NUMBER,  
    item          BOOK_TYP,  
    subtotal      NUMBER);
```

- 注文項目リストは、注文明細項目のリストとして使用され、注文項目の `VARRAY` としてモデル化されます。

```
create or replace type orderitemlist_vartyp AS VARRAY (20) OF orderitem_typ;
```

- 注文は、`order_typ` というオブジェクト型にモデル化されます。これはコンポジット型で、ネストされたオブジェクト型（定義済）を含んでいます。この型は注文、顧客情報および注文項目リストの詳細を格納します。

```
create or replace type order_typ as object (  
    orderno      NUMBER,  
    status       VARCHAR2(30),  
    ordertype    VARCHAR2(30),  
    orderregion  VARCHAR2(30),  
    customer     CUSTOMER_TYP,  
    paymentmethod VARCHAR2(30),  
    items        ORDERITEMLIST_VARTYP,  
    total        NUMBER);
```

## Visual Basic (OO4O) : サンプル・コード

この機能については、データベースの `dbexecutesql` インタフェースを使用します。

## Java (JDBC) : サンプル・コード

1. 型を作成した後、`JPublisher` を使用して、SQL 型にマップする Java クラスを生成する必要があります。
  - a. 次のように入力して、`JPublisher` に対する入力ファイル `jaqbol.typ` を作成します。

```
TYPE boladm.customer_typ as Customer  
TYPE boladm.book_typ as Book  
TYPE boladm.orderitem_typ AS OrderItem  
TYPE boladm.orderitemlist_vartyp AS OrderItemList  
TYPE boladm.order_typ AS Order
```

- b. 次の引数を使用して、JPublisher を実行します。

```
jpub -input=jaqbol.typ -user=boladm/boladm -case=mixed -methods=false
```

これによって、前述の項で作成された SQL オブジェクト型にマップする Java クラス (Customer、Book、OrderItem および OrderItemList) が作成されます。

- c. Java AQ ドライバをロードし、JDBC 接続を作成します。

```
public static Connection loadDriver(String user, String passwd)
{
    Connection db_conn = null;

    try
    {

        Class.forName("oracle.jdbc.driver.OracleDriver");

        /* 実際のホスト名、ポート番号およびSIDは、次のものとは異なります。
        ここでは、明示的に 'dlsun736' '5521' および 'test' を指定します。*/

        db_conn =
            DriverManager.getConnection(
                "jdbc:oracle:thin:@dlsun736:5521:test",
                user, passwd);

        System.out.println("JDBC Connection opened ");
        db_conn.setAutoCommit(false);

        /* Oracle8i AQ ドライバをロードします。*/
        Class.forName("oracle.AQ.AQOracleDriver");

        System.out.println("Successfully loaded AQ driver ");
    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
        ex.printStackTrace();
    }
    return db_conn;
}
```

## キュー・レベルのアクセス制御

Oracle8i は、エンキューおよびデキュー操作に対するキュー・レベルのアクセス制御をサポートします。この機能によって、アプリケーション設計者は、あるスキーマに作成されたキューを他のスキーマで実行中の他のアプリケーションから保護することができます。そのキューが属するスキーマの外で実行しているアプリケーションには、最小限度のアクセス権限を付与するのみで済みます。キューに対するアクセス権限でサポートされている ENQUEUE、DEQUEUE および ALL の詳細は、4-4 ページの「[セキュリティ](#)」を参照してください。

### サンプル・シナリオ

BooksOnLine アプリケーションでは、顧客請求情報を CB および CBADM スキーマで処理します。CB（顧客請求情報）スキーマには顧客請求情報アプリケーションがあり、CBADM スキーマには、関連する請求情報データがキュー・テーブルの形で格納されています。

請求情報データを保護するために、請求処理アプリケーションおよび請求情報データは別のスキーマに常駐しています。請求処理アプリケーションは、出荷済の注文情報キューである CBADM\_shippedorders\_queue からメッセージをデキューする以外の処理はできません。そこでメッセージを処理し、請求済の注文情報キューである CBADM\_billedorders\_queue に新しいメッセージをエンキューします。

他のアプリケーションの不正な操作からキューを保護するために、次の 2 つの grant コールを行います。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

/\* 顧客請求処理アプリケーションに、出荷済の注文情報キューに対するデキュー権限を付与します。CB アプリケーションは、出荷済ではあるが未請求の注文を出荷済の注文情報キューから受け取ります。\*/

```
EXECUTE dbms_aqadm.grant_queue_privilege(
    'DEQUEUE','CBADM_shippedorders_queue', 'CB', FALSE);
```

/\* 顧客請求処理アプリケーションに、請求済の注文情報キューに対するエンキュー権限を付与します。CB アプリケーションは、注文を処理した後、このキューに請求済の注文を置くことができます。\*/

```
EXECUTE dbms_aqadm.grant_queue_privilege(
    'ENQUEUE', 'CBADM_billedorders_queue', 'CB', FALSE);
```

### Visual Basic (OO4O) : サンプル・コード

この機能については、データベースの dbexecutesql インタフェースを使用します。



## Java (JDBC) : サンプル・コード

```
public static void grantQueuePrivileges(Connection db_conn)
{
    AQSession  aq_sess;
    AQQueue    sh_queue;
    AQQueue    bi_queue;

    try
    {

        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* 顧客請求処理アプリケーションに、出荷済の注文情報キューに対するデキュー権限
        を付与します。CB アプリケーションは、出荷済ではあるが未請求の注文を出荷済の注文情報
        キューから受け取ります。*/

        sh_queue = aq_sess.getQueue("CBADM", "CBADM_shippedorders_que");

        sh_queue.grantQueuePrivilege("DEQUEUE", "CB", false);

        /* 顧客請求処理アプリケーションに、請求済の注文情報キューに対するエンキュー権限を付
        与します。CB アプリケーションは、注文を処理した後、このキューに請求済の注文を置くこ
        とができます。*/

        bi_queue = aq_sess.getQueue("CBADM", "CBADM_billedorders_que");

        bi_queue.grantQueuePrivilege("ENQUEUE", "CB", false);
    }
    catch (AQException ex)
    {
        {
            System.out.println("AQ Exception: " + ex);
        }
    }
}
```

## 非永続キュー

非永続キューにあるメッセージは、データベース表に格納されていないため永続性がありません。

ユーザーは、単一コンシューマまたは複数コンシューマ・タイプの非永続の RAW 型キューを作成できます。このキューは、`create_np_queue` コマンドで指定されたスキーマ内の、システムによって作成されたキュー・テーブル（単一コンシューマ・キューは `AQ$_MEM_SC`、複数コンシューマ・キューは `AQ$_MEM_MC`）内に作成されます。複数コンシューマ・キューにはサブスクライバを追加できます（9-27 ページの「[非永続キューの作成](#)」を参照）。非永続キューは、伝播の宛先になることができます。

ユーザーは、エンキュー・インタフェースを使用して、通常の方法で非永続キューにメッセージをエンキューします。ユーザーは、関心があるキューに（`OCISubscriptionRegister` によって）通知登録を行い、非同期式通知メカニズムを通じて、非永続キューからメッセージを取り出します（11-56 ページの「[通知登録](#)」を参照）。

あるキューにメッセージがエンキューされると、そのキューに対してアクティブな登録を持っているクライアントに配信されます。さらにそのメッセージは、データベースへの格納というオーバーヘッドなしで、関心を持っているクライアントに公開されます。

---

---

### 参照：

- 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の `OCISubscriptionRegister` に関する説明を参照してください。
- 
- 

## サンプル・シナリオ

注文入力システムには、ユーザー要求に対処するアプリケーション・プロセスが3つあると想定します。接続ディスパッチャ・プロセスは、接続要求を複数のアプリケーション・プロセスに分配するもので、注文入力システムにログイン中のユーザー数、およびアプリケーション・プロセスあたりのユーザー数をメンテナンスします。アプリケーション・プロセスの名前は、`APP_1`、`APP_2`、`APP_3` です。単純化のために、アプリケーション・プロセス障害は想定から除外します。

この要件を解決する方法の1つは、非永続キューを使用することです。ユーザーがデータベースにログインするときに、アプリケーション・プロセスは複数コンシューマの非永続キューである `LOGIN_LOGOUT` にエンキューし、アプリケーション名をコンシューマ名にします。ユーザーがログアウトするときも、同じように処理されます。2つのイベントを区別するために、ログイン・メッセージの相関識別子は「`LOGIN`」、ログアウト・メッセージの相関識別子は「`LOGOUT`」にします。

コールバック関数は、アプリケーション・プロセスあたりのログイン / ログアウトのイベント件数を数えます。ディスパッチャ・プロセスがデータベースに接続する必要があるのは、サブスクリプションを登録する場合のみということに注意してください。通知そのものは、プロセスがデータベースから切断されている間にも受け取ることができます。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : 非永続キュー

```
CONNECT oe/oe;

/* OE スキーマに複数コンシューマ・タイプの非永続キューを作成します。*/
EXECUTE dbms_aqadm.create_np_queue(queue_name      => 'LOGON_LOGOFF',
                                   multiple_consumers => TRUE);

/* キューのエンキューおよびデキューを可能にします。*/
EXECUTE dbms_aqadm.start_queue(queue_name => 'LOGON_LOGOFF');

/* 非永続キューのシナリオ - ログイン時に実行するプロシージャ */
CREATE OR REPLACE PROCEDURE User_Logon(app_process IN VARCHAR2)
AS
    msgprop      dbms_aq.message_properties_t;
    enqopt       dbms_aq.enqueue_options_t;
    enq_msgid    RAW(16);
    payload      RAW(1);
BEGIN
    /* 可視性は、非永続キューに対して常に直接的である必要があります。*/
    enqopt.visibility:=dbms_aq.IMMEDIATE;
    msgprop.correlation:= 'LOGON';
    msgprop.recipient_list(0) := aq$_agent(app_process, NULL, NULL);
    /* payload は NULL です。*/
    dbms_aq.enqueue(
        queue_name      => 'LOGON_LOGOFF',
        enqueue_options => enqopt,
        message_properties => msgprop,
        payload         => payload,
        msgid           => enq_msgid);

END;
/

/* 非永続キューのシナリオ - ログオフ時に実行するプロシージャ */
CREATE OR REPLACE PROCEDURE User_Logoff(app_process IN VARCHAR2)
AS
```

```

msgprop      dbms_aq.message_properties_t;
engopt       dbms_aq.enqueue_options_t;
enq_msgid    RAW(16);
payload      RAW(1);
BEGIN
  /* 可視性は、非永続キューに対して常に直接的である必要があります。 */
  enqopt.visibility:=dbms_aq.IMMEDIATE;
  msgprop.correlation:= 'LOGOFF';
  msgprop.recipient_list(0) := aq$_agent(app_process, NULL, NULL);
  /* payload は NULL です。 */
  dbms_aq.enqueue(
    queue_name      => 'LOGON_LOGOFF',
    enqueue_options => enqopt,
    message_properties => msgprop,
    payload         => payload,
    msgid           => enq_msgid);
END;
/

/* APP1 でログインするときに、メッセージを 'login_logoff' にエンキューします。 相関識別子は
'LOGIN' です。 */
EXECUTE User_logon('APP1');

/* APP3 でログオフするときに、メッセージを 'login_logoff' にエンキューします。 相関識別子は
'LOGOFF' です。 */
EXECUTE User_logoff('App3');

/* 通知を待つ OCI プログラムです。 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>
#ifdef WIN32COMMON
#define sleep(x)    Sleep(1000*(x))
#endif

/* ログイン / パスワード */
static text *username = (text *) "OE";
static text *password = (text *) "OE";

/* メッセージの相関識別子を表す文字列です。 */
static char *logon = "LOGON";
static char *logoff = "LOGOFF";

```

```

/* キューが取り得るコンシューマ名です。*/
static char *applist[] = {"APP1", "APP2", "APP3"};

static OCIEnv *envhp;
static OCIServer *srvhp;
static OCLError *errhp;
static OCISvcCtx *svchp;

static void checkerr(/*_ OCLError *errhp, sword status _*/);

struct process_statistics
{
    ub4    logon;
    ub4    logoff;
};

typedef struct process_statistics process_statistics;

int main(/*_ int argc, char *argv[] _*/);

/* コールバックを通知します。*/
ub4 notifyCB(ctx, subscrhp, pay, payl, desc, mode)
dvoid *ctx;
OCISubscription *subscrhp;
dvoid *pay;
ub4    payl;
dvoid *desc;
ub4    mode;
{
    text          *subname;    /* サブスクリプション名 */
    ub4           lsub;        /* サブスクリプション名の長さ */
    text          *queue;      /* キュー名 */
    ub4           *lqueue;     /* キュー名 */
    text          *consumer;   /* コンシューマ名 */
    ub4           lconsumer;
    text          *correlation;
    ub4           lcorrelation;
    ub4           size;
    ub4           appno;
    OCIRaw        *msgid;
    OCIAQMsgProperties *msgprop; /* メッセージ・プロパティの記述子 */
    process_statistics *user_count = (process_statistics *)ctx;

```

```
OCIAttrGet((dvoid *)subscrhp, OCI_HTYPE_SUBSCRIPTION,
           (dvoid *)&subname, &lsub,
           OCI_ATTR_SUBSCR_NAME, errhp);

/* AQ 記述子から属性を取り出します。*/
/* キュー名 */
OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&queue, &size,
           OCI_ATTR_QUEUE_NAME, errhp);

/* コンシューマ名 */
OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&consumer, &lconsumer,
           OCI_ATTR_CONSUMER_NAME, errhp);

/* メッセージ・プロパティ */
OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgprop, &size,
           OCI_ATTR_MSG_PROP, errhp);

/* メッセージ・プロパティから相関識別子を取得します。*/
checkerr(errhp, OCIAttrGet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES,
                           (dvoid *)&correlation, &lcorrelation,
                           OCI_ATTR_CORRELATION, errhp));

if (lconsumer == strlen(applist[0]))
{
    if (!memcmp((dvoid *)consumer, (dvoid *)applist[0], strlen(applist[0])))
        appno = 0;
    else if (!memcmp((dvoid *)consumer, (dvoid *)applist[1],
strlen(applist[1])))
        appno = 1;
    else if (!memcmp((dvoid *)consumer, (dvoid *)applist[2],
strlen(applist[2])))
        appno = 2;
    else
    {
        printf("Wrong consumer in notification");
        return;
    }
}
else
{
    /* コンシューマ名は、"APP1"、"APP2" または "APP3" です。*/
    printf("Wrong consumer in notification");
    return;
}

if (lcorrelation == strlen(logon) &&                                /* ログイン・イベント */
```

```

        !memcmp((dvoid *)correlation, (dvoid *)logon, strlen(logon)))
    {
        user_count[appno].logon++;
        /* app プロセスに対するログイン回数が増加します。*/
        printf("Logon by APP%d \n", (appno+1));
    }
    else if (lcorrelation == strlen(logoff) && /* ログオフ・イベント */
        !memcmp((dvoid *)correlation, (dvoid *)logoff, strlen(logoff)))
    {
        user_count[appno].logoff++;
        /* app プロセスに対するログオフ回数が増加します。*/
        printf("Logoff by APP%d \n", (appno+1));
    }
    else
        /* 相関識別子は "LOGON" または "LOGOFF" です。*/
        printf("Wrong correlation in notification");

    printf("Total : \n");

    printf("App1 : %d \n", user_count[0].logon-user_count[0].logoff);
    printf("App2 : %d \n", user_count[1].logon-user_count[1].logoff);
    printf("App3 : %d \n", user_count[2].logon-user_count[2].logoff);
}

int main(argc, argv)
int argc;
char *argv[];
{
    OCISession *authp = (OCISession *) 0;
    OCISubscription *subscrhp[3];
    ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;
    process_statistics ctx[3] = {{0,0}, {0,0}, {0,0}};
    ub4 sleep_time = 0;

    printf("Initializing OCI Process\n");

    /* OCI_EVENTS フラグ・セットで OCI 環境を初期化します。*/
    (void) OCIInitialize((ub4) OCI_EVENTS|OCI_OBJECT, (dvoid *)0,
        (dvoid * (*) (dvoid *, size_t)) 0,
        (dvoid * (*) (dvoid *, dvoid *, size_t)) 0,
        (void *) (dvoid *, dvoid *) 0 );

    printf("Initialization successful\n");

    printf("Initializing OCI Env\n");

```

```
(void) OCIEnvInit( (OCIEnv **) &envhp, OCI_DEFAULT, (size_t) 0, (dvoid **) 0
);
printf("Initialization successful\n");

checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp,
OCI_HTYPE_ERROR,
    (size_t) 0, (dvoid **) 0));

checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp,
OCI_HTYPE_SERVER,
    (size_t) 0, (dvoid **) 0));

checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp,
OCI_HTYPE_SVCCTX,
    (size_t) 0, (dvoid **) 0));

printf("connecting to server\n");
checkerr(errhp, OCIServerAttach( srvhp, errhp, (text *)"inst1_alias",
    strlen("inst1_alias"), (ub4) OCI_DEFAULT));
printf("connect successful\n");

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
checkerr(errhp, OCIAttrSet( (dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid *)srvhp,
    (ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp));

checkerr(errhp, OCIHandleAlloc((dvoid *) envhp, (dvoid **)&authp,
    (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0));

/* セッション・ハンドルにユーザー名およびパスワードを設定します。*/
checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
    (dvoid *) username, (ub4) strlen((char *)username),
    (ub4) OCI_ATTR_USERNAME, errhp));

checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
    (dvoid *) password, (ub4) strlen((char *)password),
    (ub4) OCI_ATTR_PASSWORD, errhp));

/* セッションを開始します。*/
checkerr(errhp, OCISessionBegin ( svchp, errhp, authp, OCI_CRED_RDBMS,
    (ub4) OCI_DEFAULT));

(void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
    (dvoid *) authp, (ub4) 0,
    (ub4) OCI_ATTR_SESSION, errhp);
```



```
/* 通知を登録します。*/
printf("allocating subscription handle\n");
subscrhp[0] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[0],
                      (ub4) OCI_HTYPE_SUBSCRIPTION,
                      (size_t) 0, (dvoid **) 0);

/* アプリケーション・プロセス APP1 用です。*/
printf("setting subscription name\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) "OE.LOGON_LOGOFF:APP1",
                  (ub4) strlen("OE.LOGON_LOGOFF:APP1"),
                  (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

printf("setting subscription callback\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) notifyCB, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *)&ctx, (ub4) sizeof(ctx),
                  (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

printf("setting subscription namespace\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) &namespace, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

printf("allocating subscription handle\n");
subscrhp[1] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[1],
                      (ub4) OCI_HTYPE_SUBSCRIPTION,
                      (size_t) 0, (dvoid **) 0);

/* アプリケーション・プロセス APP2 用です。*/
printf("setting subscription name\n");
(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) "OE.LOGON_LOGOFF:APP2",
                  (ub4) strlen("OE.LOGON_LOGOFF:APP2"),
                  (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

printf("setting subscription callback\n");
(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) notifyCB, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);
```

```
(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *)&ctx, (ub4)sizeof(ctx),
                 (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

printf("setting subscription namespace\n");
(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) &namespace, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

printf("allocating subscription handle\n");
subscrhp[2] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[2],
                     (ub4) OCI_HTYPE_SUBSCRIPTION,
                     (size_t) 0, (dvoid **) 0);

/* アプリケーション・プロセス APP3 用です。*/
printf("setting subscription name\n");
(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) "OE.LOGON_LOGOFF:APP3",
                 (ub4) strlen("OE.LOGON_LOGOFF:APP3"),
                 (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

printf("setting subscription callback\n");
(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) notifyCB, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *)&ctx, (ub4)sizeof(ctx),
                 (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

printf("setting subscription namespace\n");
(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) &namespace, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

printf("Registering fornifications \n");
checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 3, errhp,
                                       OCI_DEFAULT));

sleep_time = (ub4)atoi(argv[1]);
printf ("waiting for %d s \n", sleep_time);
sleep(sleep_time);
```

```
    printf("Exiting");
    exit(0);
}

void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    sb4 errcode = 0;

    switch (status)
    {
        case OCI_SUCCESS:
            break;
        case OCI_SUCCESS_WITH_INFO:
            (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
            break;
        case OCI_NEED_DATA:
            (void) printf("Error - OCI_NEED_DATA\n");
            break;
        case OCI_NO_DATA:
            (void) printf("Error - OCI_NODATA\n");
            break;
        case OCI_ERROR:
            (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL, &errcode,
                               errbuf, (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
            (void) printf("Error - %.*s\n", 512, errbuf);
            break;
        case OCI_INVALID_HANDLE:
            (void) printf("Error - OCI_INVALID_HANDLE\n");
            break;
        case OCI_STILL_EXECUTING:
            (void) printf("Error - OCI_STILL_EXECUTE\n");
            break;
        case OCI_CONTINUE:
            (void) printf("Error - OCI_CONTINUE\n");
            break;
        default:
            break;
    }
}

/* ファイル tkagdocn.c の終わり */
```

### **Visual Basic (OO4O) : サンプル・コード**

この機能は、現在サポートされていません。

### **Java (JDBC) : サンプル・コード**

Java API ではサポートされません。

## 保存およびメッセージ履歴

AQ によって、ユーザーはメッセージをキュー・テーブルに保存することができます。これは、SQL 問合せを使用してこのメッセージを問い合わせて分析できることを意味します。メッセージは、相互に関連していることがよくあります。たとえば、あるメッセージを処理した結果、他のメッセージが生成された場合、両者は関連付けられています。アプリケーション設計者としては、そのような関連を追跡することが必要な場合があります。保存機能およびメッセージ識別子とともに、メッセージ・ジャーナルが AQ によって自動作成され、追跡ジャーナルまたはイベント・ジャーナルとして参照できます。保存、メッセージ識別子および SQL 問合せの協調によって、強力なメッセージ・ウェアハウスが構築できます。

### サンプル・シナリオ

出荷処理アプリケーションで、注文の平均処理時間を判断する必要があると仮定します。この時間には、backed\_order キューでの待機時間も含まれます。つまり、backed\_order キューでの待機時間の平均も調査することになります。出荷キューの保存を TRUE に指定し、メッセージの相関フィールドを注文番号に設定すると、SQL 問合せにより出荷処理アプリケーション中の注文情報の待機時間を判断できます。

単純化のために、すでに処理された注文情報のみを分析することにします。出荷処理アプリケーション中で注文の処理にかかる時間は、WS\_bookedorders\_que にエンキューされる時刻と WS\_shipped\_orders\_que にエンキューされる時刻の差です（C-2 ページの「[tkaqdoca.sql: ユーザー、オブジェクト、キュー・テーブル、キューおよびサブスクライバ作成用のスクリプト](#)」を参照）。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
SELECT SUM(SO.enq_time - BO.enq_time) / count (*) AVG_PRCS_TIME
FROM WS.AQ$WS_orders_pr_mqtab BO , WS.AQ$WS_orders_mqtab SO
WHERE SO.msg_state = 'PROCESSED' and BO.msg_state = 'PROCESSED'
AND SO.corr_id = BO.corr_id and SO.queue = 'WS_shippedorders_que';

/* backed_order キューでの平均待機時間 */
SELECT SUM(BACK.deq_time - BACK.enq_time)/count (*) AVG_BACK_TIME
FROM WS.AQ$WS_orders_mqtab BACK
WHERE BACK.msg_state = 'PROCESSED' AND BACK.queue = 'WS_backorders_que';
```

### Visual Basic (OO4O) : サンプル・コード

この機能については、データベースの dbexecutesql インタフェースを使用します。

### Java (JDBC) : サンプル・コード

今回のリリースでは、例は記載していません。

## パブリッシュ / サブスクライブ・サポート

Oracle AQ には様々な機能が追加され、ユーザーがパブリッシュ / サブスクライブ・モデルに基づいたアプリケーションを開発できるようになっています。このアプリケーション・モデルの目的は、パブリッシャ（公開者）の機能を持つアプリケーションとサブスクライバ（購読者）の役割を果たすアプリケーションとの間の柔軟で動的な通信を可能にすることです。具体的な設計のポイントは、そのように異なる役割を果たすアプリケーションを通信上では分離し、メッセージおよびメッセージ内容に基づいた相互作用をさせるという点です。

分散メッセージでは、パブリッシャ・アプリケーションが明示的にメッセージ受信者を処理または管理する必要はありません。このため、受信メッセージに新しいサブスクライバ・アプリケーションを動的に追加でき、パブリッシャ・アプリケーションの論理を変更する必要はありません。サブスクライバ・アプリケーションは、どのパブリッシャ・アプリケーションが送信したかにかかわらず、メッセージの内容に基づいてメッセージを受信します。このため、サブスクライバ・アプリケーションを動的に追加でき、どのサブスクライバ・アプリケーションの論理も変更する必要はありません。キューのメッセージ内容（ペイロード）およびメッセージ・ヘッダー・プロパティに対してルールベースのサブスクリプションを定義することで、サブスクライバ・アプリケーションは関心のあることを定義します。システムは、ルールベースのサブスクリプションを使用して、公開されたメッセージの受信者を確認し、自動的にルーティングします。

次の各ステップを実行することで、AQ を使用した通信のパブリッシュ / サブスクライブ・モデルを実装できます。

- メッセージを保持するために1つまたは複数のキューを設定します。このキューは、関心がある領域またはサブジェクトを表しています。たとえば、あるキューは請求済注文を表すために使用される可能性があります。
- ルールベースのサブスクライバを1組設定します。各サブスクライバは、受信を希望するメッセージ仕様を表すルールを指定できます。NULL ルールは、サブスクライバがすべてのメッセージの受信を希望することを意味します。
- パブリッシャ・アプリケーションが、エンキュー・コールをコールしてキューにメッセージを公開します。

- サブスクライバ・アプリケーションは、次のような方法でメッセージを受信できます。
  - DEQUEUE コールが、サブスクリプションの基準に合うメッセージを取り出します。
  - LISTEN コールを使用して、様々なキューに対するサブスクリプションを複数のキューで監視できます。これは、サブスクライバ・アプリケーションが多数のキューにサブスクライブしていて、どのキューに届いたメッセージでも受信する場合には、非常にスケーラブルなソリューションです。
  - OCI 通知メカニズムを使用します。これによって、プッシュ・モードのメッセージ配信が可能になります。このモードでは、メッセージを受信するキュー（およびサブスクライブ・エージェントとして指定されたサブスクリプション）のルールに一致するメッセージが届いたときにコールされるコールバックを、サブスクライバ・アプリケーションが登録します。

## サンプル・シナリオ

BooksOnLine アプリケーションは、アプリケーション間通信のパブリッシュ / サブスクライバ・モデルの使用法を示しています。次の項で、いくつか例を示します。

**キューの定義** 注文入力アプリケーションでは、様々なアプリケーションに入力済注文および通信するためのキュー (OE\_booked\_orders\_que) が定義されます。注文入力アプリケーションは、複数のサブスクライバ・アプリケーションを認識しないため、注文入力 (パブリッシャ) アプリケーションの設定または論理を混乱させずに新しいサブスクライバ・アプリケーションを追加できます。

**サブスクリプションの設定** 様々な出荷処理アプリケーションおよび顧客サービス・アプリケーション（東部向け出荷、西部向け出荷、海外向け出荷および顧客サービス）は、注文入力アプリケーションの booked\_orders キューに対するサブスクライバとして定義されます。ルールは、関心があるメッセージを様々なサブスクライバに送るために使用されます。たとえば、東部向け出荷は東海岸地区向けの注文および合衆国全域向けの至急注文を扱い、次のようなサブスクリプション・ルールの記述になります。

```
rule => 'tab.user_data.orderregion = ''EASTERN'' OR
(tab.user_data.ordertype = ''RUSH'' AND
tab.user_data.customer.country = ''USA'')
```

各サブスクライバは、ローカル・キューを指定して、そこにメッセージが配信されるようにできます。東部向け出荷処理アプリケーションでは、サブスクライバ・アドレスを次のように指定して、メッセージ配信用のローカル・キュー (ES\_booked\_orders\_que) を指定します。

```
subscriber := aq$_agent('East_Shipping', 'ES.ES_bookedorders_que', null);
```



**伝播の設定** 各パブリッシャ・アプリケーション・キューから伝播できます。サブスクライブされたメッセージをリモート・キューに配信できるようにするために、注文入力アプリケーションでは次のように指定して伝播を使用可能にします。

```
execute dbms_aqadm.schedule_propagation(queue_name => 'OE.OE_bookedorders_que');
```

**メッセージの公開** 注文入力アプリケーションが注文を妥当性チェックし、出荷準備完了後に (OE\_booked\_order\_que に) エンキューしたときに、入力済注文が公開されます。このメッセージは、次に、サブスクライブしている各アプリケーションに送られます。メッセージは、各サブスクライバ・アプリケーションのローカル・キュー（指定されている場合）に配信されます。

**メッセージの受信** 出荷処理アプリケーションおよび顧客サービス・アプリケーションは、それぞれローカル・キューのメッセージを受信します。たとえば、東部向け出荷処理アプリケーションは、東海岸向けまたは RUSH というマークのある合衆国内向けの入力済注文のみを受信します。その後、このアプリケーションはメッセージをデキューして、その注文を出荷のために処理します。

## Oracle Parallel Server (OPS) のサポート

Oracle Parallel Server を使用すると、異なるキューを別々のインスタンスによって管理できるようにして AQ パフォーマンスを改善できます。このためには、キューを格納するキュー・テーブルに様々なインスタンス親和性を指定します。これによって、様々なキューに対する操作（エンキュー / デキュー）をパラレルで行うことができるようになります。

AQ のキュー・モニター・プロセスは、キュー・テーブルのインスタンス親和性を継続的に監視します。キュー・モニターは指定されたプライマリ・インスタンスが使用可能な場合はそれにキュー・テーブルの所有権を割り当て、失敗した場合は指定されたセカンダリ・インスタンスに割り当てます。キュー・テーブルを所有しているインスタンスが停止した場合はいつでも、キュー・モニターはそのキュー・テーブルの所有権を適切なインスタンスに割り当てます。それがセカンダリ・インスタンスで、それも使用不可な場合は他の使用可能なインスタンスに変更します。

AQ の伝播は OPS でも使用可能になりますが、これはユーザーにとっては完全に透過的です。伝播スケジュールのジョブ親和性は、それぞれのキュー・テーブルの親和性と同じ値に設定されます。このように、キュー・テーブルを所有するインスタンスに対応付けられたジョブ・キュー・プロセスは、そのキュー・テーブルに格納されているキューからの伝播を処理し、ping 操作を最小限に抑えます。詳細は、9-65 ページの「[キューの伝播のスケジュール](#)」を参照してください。

---

### 参照：

- Oracle Parallel Server の詳細は、『Oracle8i Parallel Server セットアップおよび構成ガイド』を参照してください。
- 

## サンプル・シナリオ

BooksOnLine の例では、注文入力 (OE) サイトでの `new_orders_queue` および `booked_order_queue` の操作は、この 2 つのキューが異なるインスタンスと対応付けられている場合は高速化できます。そのために、2 つのキューを別々のキュー・テーブルに作成し、`create_queue_table()` コマンドでそれらのキュー・テーブルに別々の親和性を指定します。

この例では、キュー・テーブル `OE_orders_sqtab` にキュー `new_orders_queue` を格納し、プライマリおよびセカンダリ・インスタンスはそれぞれインスタンス 1 および 2 です。キュー・テーブル `OE_orders_mqtab` にはキュー `booked_order_queue` を格納し、プライマリおよびセカンダリ・インスタンスはそれぞれインスタンス 2 および 1 です。目的は、インスタンス 1 および 2 に 2 つのキューをパラレルで管理させることです。デフォルトでは、1 つのインスタンスのみが使用可能であり、この場合、両方のキュー・テーブルの所有者はインスタンス 1 に設定されます。ただし、OPS が正しく設定され、インスタンス 1 および 2 が両方とも使用可能な場合は、キュー・テーブル `OE_orders_sqtab` がインスタンス 1 によって所有され、他のキュー・テーブルはインスタンス 2 によって所有されます。キュー・テーブルに対するプライマリおよびセカンダリ・インスタンスの指定は、次の例に示すように、`alter_queue_table()` コマンドを使用して動的に変更できます。キュー・

テーブルのプライマリ、セカンダリおよび所有者インスタンスに関する情報は、[USER\\_QUEUE\\_TABLES](#) ビューを問い合わせて取得できます（10-25 ページの「[管理インタフェース:ビュー](#)」の「[ユーザー・スキーマのキュー・テーブルの選択](#)」を参照）。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```

/* OE にキュー・テーブル、キューを作成します。*/
CONNECT OE/OE;
EXECUTE dbms_aqadm.create_queue_table( \
    queue_table      => 'OE_orders_sqtab',\
    comment          => 'Order Entry Single-Consumer Orders queue table',\
    queue_payload_type => 'BOLADM.order_typ',\
    compatible       => '8.1',\
    primary_instance  => 1,\
    secondary_instance => 2);

EXECUTE dbms_aqadm.create_queue_table(\
    queue_table      => 'OE_orders_mqtab',\
    comment          => 'Order Entry Multi Consumer Orders queue table',\
    multiple_consumers => TRUE,\
    queue_payload_type => 'BOLADM.order_typ',\
    compatible       => '8.1',\
    primary_instance  => 2,\
    secondary_instance => 1);

EXECUTE dbms_aqadm.create_queue ( \
    queue_name       => 'OE_neworders_que',\
    queue_table      => 'OE_orders_sqtab');

EXECUTE dbms_aqadm.create_queue ( \
    queue_name       => 'OE_bookedorders_que',\
    queue_table      => 'OE_orders_mqtab');

/* AQ 管理ビューで、OE キュー・テーブルのインスタンス親和性を確認します。*/
SELECT queue_table, primary_instance, secondary_instance, owner_instance
FROM user_queue_tables;

/* OE キュー・テーブルのインスタンス親和性を変更します。*/
EXECUTE dbms_aqadm.alter_queue_table( \
    queue_table      => 'OE.OE_orders_sqtab',\

```

```
primary_instance => 2,\
secondary_instance => 1);

EXECUTE dbms_aqadm.alter_queue_table( \
queue_table      => 'OE.OE_orders_mqtab', \
primary_instance => 1,\
secondary_instance => 2);

/* AQ 管理ビューで、OE キュー・テーブルのインスタンス親和性を確認します。*/
SELECT queue_table, primary_instance, secondary_instance, owner_instance
FROM user_queue_tables;
```

## Visual Basic (OO4O) : サンプル・コード

この機能は、現在サポートされていません。

## Java (JDBC) : サンプル・コード

```
public static void createQueueTablesAndQueues(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueueTableProperty sqt_prop;
    AQQueueTableProperty mqt_prop;
    AQQueueTable        sq_table;
    AQQueueTable        mq_table;
    AQQueueProperty     q_prop;
    AQQueue              neworders_q;
    AQQueue              bookedorders_q;

    try
    {

        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* 単一コンシューマの注文キュー・テーブルを作成します。*/
        sqt_prop = new AQQueueTableProperty("BOLADM.order_ttyp");
        sqt_prop.setComment("Order Entry Single-Consumer Orders queue table");
        sqt_prop.setCompatible("8.1");
        sqt_prop.setPrimaryInstance(1);
        sqt_prop.setSecondaryInstance(2);

        sq_table = aq_sess.createQueueTable("OE", "OE_orders_sqtab", sqt_prop);
```

```
/* 複数コンシューマの注文キュー・テーブルを作成します。 */
mqt_prop = new AQQueueTableProperty("BOLADM.order_ttyp");
mqt_prop.setComment("Order Entry Multi Consumer Orders queue table");
mqt_prop.setCompatible("8.1");
mqt_prop.setMultiConsumer(true);
mqt_prop.setPrimaryInstance(2);
mqt_prop.setSecondaryInstance(1);

mq_table = aq_sess.createQueueTable("OE", "OE_orders_mqtab", mqt_prop);

/* これらのキュー・テーブルにキューを作成します。 */
q_prop = new AQQueueProperty();

neworders_q = aq_sess.createQueue(sq_table, "OE_neworders_que",
                                   q_prop);

bookedorders_q = aq_sess.createQueue(mq_table, "OE_bookedorders_que",
                                       q_prop);

}
catch (AQException ex)
{
    System.out.println("AQ Exception: " + ex);
}
}

public static void alterInstanceAffinity(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueueTableProperty sqt_prop;
    AQQueueTableProperty mqt_prop;
    AQQueueTable        sq_table;
    AQQueueTable        mq_table;
    AQQueueProperty      q_prop;

    try
    {

        /* AQ セッションを作成します。 */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* インスタンス親和性を確認します。 */
        sq_table = aq_sess.getQueueTable("OE", "OE_orders_sqtab");
```

```
sqt_prop = sq_table.getProperty();
System.out.println("Current primary instance for OE_orders_sqtab: " +
    sqt_prop.getPrimaryInstance());

mq_table = aq_sess.getQueueTable("OE", "OE_orders_mqtab");
mgt_prop = mq_table.getProperty();
System.out.println("Current primary instance for OE_orders_mqtab: " +
    mgt_prop.getPrimaryInstance());

/* キュー・テーブル親和性を変更します。*/
sq_table.alter(null, 2, 1);

mq_table.alter(null, 1, 2);

sqt_prop = sq_table.getProperty();
System.out.println("Current primary instance for OE_orders_sqtab: " +
    sqt_prop.getPrimaryInstance());

mq_table = aq_sess.getQueueTable("OE", "OE_orders_mqtab");
mgt_prop = mq_table.getProperty();
System.out.println("Current primary instance for OE_orders_mqtab: " +
    mgt_prop.getPrimaryInstance());

}
catch (AQException ex)
{
    System.out.println("AQ Exception: " + ex);
}
}
```

## 統計ビューのサポート

各インスタンスは、それぞれの AQ 統計情報を SGA に所有し、他のインスタンスによって収集された統計については認識しません。ただし、あるインスタンスで GV\$AQ ビューに対して問い合わせると、その他のすべてのインスタンスからそれぞれの AQ 統計情報が問合せ元のインスタンスに集計されます。

### サンプル・シナリオ

GV\$ ビューでは、待機中、準備完了または期限切れの各メッセージ数を必要なときにいつでも問い合わせることができます。また、メッセージが処理されるまでの平均待機秒数も表示します。注文処理アプリケーションは、これを使用して注文処理プロセスの数を動的にチューニングできます（10-39 ページの「[データベース全体における状態ごとのメッセージ数の選択](#)」を参照）。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
CONNECT oe/oe

/* メッセージ数およびメッセージが処理されるまでの平均待機時間をカウントします。*/
SELECT READY, AVERAGE_WAIT FROM gv$aq Stats, user_queues Qs
  WHERE Stats.qid = Qs.qid and Qs.Name = 'OE_neworders_que';
```

### Visual Basic (OO4O) : サンプル・コード

この機能については、データベースの dbxexecutesql インタフェースを使用します。

### Java (JDBC) : サンプル・コード

今回のリリースでは、例は記載していません。

## ENQUEUE 機能

- サブスクリプションおよび受信者リスト
- メッセージの優先順位および順序付け
- 時間指定 : 遅延
- 時間指定 : 期限切れ
- メッセージのグループ化
- 非同期通知



## サブスクリプションおよび受信者リスト

単一コンシューマ・キューでは、メッセージは一度に1つのコンシューマでしか処理できません。同じキューから、複数のプロセスまたはオペレーティング・システム・スレッドが同時にデキューしようとする、どうなるでしょうか。ロックされたメッセージはロックを作成したプロセスにしかデキューできないとすると、各プロセスはキューの先頭にあるロックされていない最初のメッセージをデキューします。処理が済むと、キューの `retention_time` が0（ゼロ）の場合はそのメッセージは削除され、そうでない場合は指定された期間保存されます。メッセージが保存されている間は、キュー・テーブル・ビューに対する SQL 問合せでそのメッセージを参照したり、BROWSE（ブラウズ）モードで処理済メッセージの ID を指定してデキューすることができます。

AQ によって、単一のメッセージを複数のコンシューマが処理できます。この機能を使用するには、複数コンシューマ・キューを作成し、それらの複数コンシューマ・キューにメッセージをエンキューする必要があります。AQ では、メッセージのコンシューマ・リストを識別するための、サブスクリプションおよび受信者リストという2つの方法があります。

### サブスクリプション

DBMS\_AQADM.ADD\_SUBSCRIBER という PL/SQL プロシージャを使用して、キューにサブスクリプションを追加できます（9-53 ページの「[サブスクライバの追加](#)」を参照）。これによって、エンキューされたメッセージに対する AQ\$\_AGENT パラメータでコンシューマを指定できます。多くのサブスクライバを追加するには、DBMS\_AQADM.ADD\_SUBSCRIBER プロシージャを繰り返し使用することで、1つの複数コンシューマ・キューあたり最大 1024 サブスクライバまで追加できます（Oracle 8.0.3 では、1つの複数コンシューマ・キューに最大でも 32 個のサブスクライバしか追加できないことに注意してください）。

ある複数コンシューマ・キューにサブスクライバとして追加されたすべてのコンシューマは、AQ\$\_AGENT パラメータに一意の値を持つ必要があります。すなわち、2つのサブスクライバが AQ\$\_AGENT 型の NAME、ADDRESS および PROTOCOL 属性に同じ値を持つことはできません。この3つの属性のうち少なくとも1つは、違う値にする必要があります（このデータ構造の詳細は、2-3 ページの「[エージェント](#)」を参照）。

単一コンシューマ・キューまたは例外キューには、サブスクリプションを追加できません。あるキューにサブスクライバとして追加されたコンシューマは、DBMS\_AQADM.ADD\_SUBSCRIBER プロシージャが完了した後にエンキューされたメッセージしかデキューできません。つまり、このプロシージャが実行される前にエンキューされたメッセージは、このコンシューマからはデキューできません。

サブスクリプションを削除するには、DBMS\_AQADM.REMOVE\_SUBSCRIBER プロシージャを使用します（9-62 ページの「[サブスクライバの削除](#)」を参照）。AQ は、AQ\$\_AGENT パラメータによって識別されるコンシューマに対応するすべてのメタデータを、自動的にキューから削除します。つまり、そのコンシューマがデキューできるが保留しているメッセージがあるときに、REMOVE\_SUBSCRIBER プロシージャを実行してもエラーにはなりません。このメッセージは、REMOVE\_SUBSCRIBER プロシージャが実行されると自動的にデキューに対して使用不可になります。8.1 以上に設定された互換パラメータで作成されたキュー・テーブルでは、コンシューマによってデキューされなかったメッセージが、

AQ\$<queue\_table> ビューに「UNDELIVERABLE」と表示されます。互換パラメータなしで作成された、または 8.0 に設定された互換パラメータで作成された複数コンシューマ・キュー・テーブルは、コンシューマごとのメッセージ状態を表示せず、全体的なメッセージの状態のみを表示します。

### 受信者リスト

メッセージをエンキューしたプロデューサがコンシューマの受信者リストを提供している場合、複数コンシューマ・キューにサブスクリプションを指定する必要はありません。状況によっては、デフォルトのサブスクライバ・リストのかわりに、特定の一連のコンシューマをターゲットに指定したメッセージをエンキューする方が望ましいこともあります。このような場合は、メッセージをエンキューするときに受信者リストを指定します。

- PL/SQL では、message\_properties レコードの recipient\_list フィールドに要素を追加して、受信者リストを指定します。
- OCI では、OCISetAttr プロシージャを使用して、OCI\_DTYPE\_AQAGENT 記述子の配列を OCI\_DTYPE\_AQMSG\_PROPERTIES メッセージ・プロパティ記述子の受信者リスト (OCI\_ATTR\_RECIPIENT\_LIST 属性) に指定することで、受信者リストを指定します。

エンキュー中に指定された受信者リストは、サブスクリプション・リストをオーバーライドします。つまり、受信者リストが指定されているメッセージは、そのキューのサブスクライバからはデキューできません。受信者リストで指定されたコンシューマは、そのキューのサブスクライバの場合もあれば、そうでない場合もあります。サブスクライバがないキューに受信者リストを指定しないでエンキューすると、エラーになります (11-5 ページの「[メッセージのエンキュー](#)」を参照)。

## メッセージの優先順位および順序付け

メッセージの順序付けとは、キューからメッセージがデキューされる順序を示します。キューに対する順序付けの方法は、キュー・テーブルが作成されるときに指定されます（9-5 ページの「[キュー・テーブルの作成](#)」を参照）。現在、AQ は 2 種類のメッセージ順序付けをサポートしています。

- メッセージの優先順位による順序付け。優先順位による順序付けを選択すると、各メッセージがエンキューされるときにエンキュー元によって優先順位が割り当てられます。デキューのときには、割り当てられた優先順位の順序でデキューされます。2 つのメッセージに同じ優先順位が割り当てられた場合の両者のデキュー順序は定義されせん。
- 先入れ先出し（FIFO）による順序付け。FIFO 優先順位のキューも、優先順位およびエンキュー時刻の両方でメッセージのソート順序を指定することで作成できます。FIFO 優先順位のキューは、優先順位による順序付けに似ていますが、同じ優先順位のメッセージが 2 つあるときにエンキュー時刻の順序でデキューされるところが異なります。

### サンプル・シナリオ

BooksOnLine アプリケーションでは、顧客は次の方法を要求できます。

- FedEx による出荷（優先順位 1）
- 優先扱い航空便による出荷（優先順位 2）
- 通常地上便による出荷（優先順位 3）

注文入力アプリケーションは、入力済注文を FIFO 優先順位のキューに格納します。入力済注文は、地区の入力済注文キューに伝播されます。各地区では、その地区の入力済注文キューにある注文が、出荷の優先順位に従って処理されます。

次のコールで、注文入力アプリケーションに FIFO 優先順位キューを作成します。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
/* OE に対して優先順位のキュー・テーブルを作成します。*/
EXECUTE dbms_aqadm.create_queue_table( \
  queue_table      => 'OE_orders_pr_mqtab', \
  sort_list        => 'priority,enq_time', \
  comment          => 'Order Entry Priority \
                      MultiConsumer Orders queue table',\
  multiple_consumers => TRUE, \
  queue_payload_type => 'BOLADM.order_typ', \
  compatible       => '8.1', \
  primary_instance  => 2, \
  secondary_instance => 1);
```

```
EXECUTE dbms_aqadm.create_queue ( \
    queue_name      => 'OE_bookedorders_que', \
    queue_table     => 'OE_orders_pr_mqtab');
```

/\* 注文が届くと、注文処理アプリケーションは、次のプロシージャでその注文を入力済注文キューにエンキューできます。出荷の優先順位は、注文ごとに指定されています。\*/

```
CREATE OR REPLACE procedure order_eng(book_title      IN VARCHAR2,
                                       book_qty       IN NUMBER,
                                       order_num       IN NUMBER,
                                       shipping_priority IN NUMBER,
                                       cust_state      IN VARCHAR2,
                                       cust_country     IN VARCHAR2,
                                       cust_region     IN VARCHAR2,
                                       cust_ord_typ    IN VARCHAR2) AS
```

```
OE_eng_order_data      BOLADM.order_typ;
OE_eng_cust_data       BOLADM.customer_typ;
OE_eng_book_data       BOLADM.book_typ;
OE_eng_item_data       BOLADM.orderitem_typ;
OE_eng_item_list       BOLADM.orderitemlist_vartyp;
engopt                dbms_aq.enqueue_options_t;
msgprop               dbms_aq.message_properties_t;
eng_msgid              RAW(16);

BEGIN
    msgprop.correlation := cust_ord_typ;
    OE_eng_cust_data    := BOLADM.customer_typ(NULL, NULL, NULL, NULL,
                                                cust_state, NULL, cust_country);
    OE_eng_book_data    := BOLADM.book_typ(book_title, NULL, NULL, NULL);
    OE_eng_item_data    := BOLADM.orderitem_typ(book_qty,
                                                OE_eng_book_data, NULL);
    OE_eng_item_list    := BOLADM.orderitemlist_vartyp(
        BOLADM.orderitem_typ(book_qty,
        OE_eng_book_data, NULL));
    OE_eng_order_data   := BOLADM.order_typ(order_num, NULL,
                                                cust_ord_typ, cust_region,
        OE_eng_cust_data, NULL,
        OE_eng_item_list, NULL);
```

/\* メッセージをエンキューする前に、出荷の優先順位をメッセージ・プロパティに指定します。\*/

```

msgprop.priority      := shipping_priority;
dbms_aq.enqueue('OE.OE_bookedorders_que', enqopt, msgprop,
               OE_enq_order_data, enq_msgid);

COMMIT;

END;

/

```

/\* 地区ごとに、同様の入力済注文キューが作成されます。注文は、注文処理アプリケーションの中央の入力済注文キューから地区の入力済注文キューに移されます。例として、西部地区での入力済注文キューを作成します。WS での出荷について、優先順位のキュー・テーブルを作成します。\*/

```

EXECUTE dbms_aqadm.create_queue_table( \
  queue_table      => 'WS_orders_pr_mqtab',
  sort_list        => ' priority,enq_time', \
  comment          => 'West Shipping Priority \
                     MultiConsumer Orders queue table',\
  multiple_consumers => TRUE, \
  queue_payload_type => 'BOLADM.order_typ', \
  compatible       => '8.1');

```

/\* 入力済の注文は、優先順位のキュー・テーブルに保存されます。\*/

```

EXECUTE dbms_aqadm.create_queue ( \
  queue_name       => 'WS_bookedorders_que', \
  queue_table      => 'WS_orders_pr_mqtab');

```

/\* 出荷処理アプリケーションは、地区ごとに、その地区の入力済注文キューからの注文を出荷の優先順位に従ってデキューし、注文を処理し、処理済注文を出荷済注文キューまたは受注残キューにエンキューします。\*/

## Visual Basic (OO4O) : サンプル・コード

```

Dim OraSession as object
Dim OraDatabase as object
Dim OraAq as object
Dim OraMsg as Object
Dim OraOrder,OraCust,OraBook,OraItem,OraItemList as Object
Dim Msgid as String

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.DbOpenDatabase("dbname", "user/pwd", 0&)
set oraaq = OraDatabase.CreateAQ("OE.OE_bookedorders_que")

```

```
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
Set OraOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")
Set OraCust = OraDatabase.CreateOraObject("BOLADM.Customer_typ")
Set OraBook = OraDatabase.CreateOraObject("BOLADM.book_typ")
Set OraItem = OraDatabase.CreateOraObject("BOLADM.orderitem_typ")
Set OraItemList = OraDatabase.CreateOraObject("BOLADM.orderitemlist_vartyp")

' Get the values of cust_state,cust_country etc from user(form_based
' input) and then a cmd_click event for Enqueue
' will execute the subroutine order_eng.
Private Sub Order_eng()

OraMsg.correlation = txt_correlation
'Initialize the customer details
    OraCust("state") = txt_cust_state
    OraCust("country") = txt_cust_country
    OraBook("title") = txt_book_title
    OraItem("quantity") = txt_book_qty
    OraItem("item") = OraBook
    OraItemList(1) = OraItem
    OraOrder("orderno") = txt_order_num
    OraOrder("ordertype") = txt_cust_order_typ
    OraOrder("orderregion") = cust_region
    OraOrder("customer") = OraCust
    OraOrder("items") = OraItemList

'Put the shipping priority into message property before enqueueing
' the message:
OraMsg.priority = priority
OraMsg = OraOrder
Msgid = OraAq.enqueue

'Release all allocations
End Sub
```

## Java (JDBC) : サンプル・コード

```
public static void createPriorityQueueTable(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueueTableProperty mqt_prop;
    AQQueueTable       pr_mq_table;
    AQQueueProperty     q_prop;
    AQQueue             bookedorders_q;
```

```
try
{
    /* AQ セッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    /* OE に対して優先順位のキュー・テーブルを作成します。*/
    mqt_prop = new AQQueueTableProperty("BOLADM.order_ttyp");
    mqt_prop.setComment("Order Entry Priority " +
                        "MultiConsumer Orders queue table");
    mqt_prop.setCompatible("8.1");
    mqt_prop.setMultiConsumer(true);

    mqt_prop.setSortOrder("priority,enq_time");

    pr_mq_table = aq_sess.createQueueTable("OE", "OE_orders_pr_mqtab",
                                           mqt_prop);

    /* このキュー・テーブルにキューを作成します。*/
    q_prop = new AQQueueProperty();

    bookedorders_q = aq_sess.createQueue(pr_mq_table,
                                           "OE_bookedorders_que", q_prop);

    /* キューに対するエンキューおよびデキューを可能にします。*/
    bookedorders_q.start(true, true);
}
catch (AQException ex)
{
    System.out.println("AQ Exception: " + ex);
}
}
```

/\* 注文が届くと、注文処理アプリケーションは、次のプロシージャでその注文を入力済注文キューにエンキューできます。出荷の優先順位は、注文ごとに指定されています。\*/

```
public static void order_enqueue(Connection db_conn, String book_title,
                                double book_qty, double order_num,
                                int ship_priority, String cust_state,
                                String cust_country, String cust_region,
                                String cust_order_type)
```

```
{
    AQSession          aq_sess;
    AQQueue             bookedorders_q;
    Order               enq_order;
    Customer             cust_data;
    Book                 book_data;
    OrderItem            item_data;
    OrderItem[]          items;
    OrderItemList         item_list;
    AQEnqueueOption      enq_option;
    AQMessageProperty    m_property;
    AQMessage             message;
    AQObjectPayload       obj_payload;
    byte[]                enq_msg_id;

    try
    {

        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);

        cust_data = new Customer();
        cust_data.setCountry(cust_country);
        cust_data.setState(cust_state);

        book_data = new Book();
        book_data.setTitle(book_title);

        item_data = new OrderItem();
        item_data.setQuantity(new BigDecimal(book_qty));
        item_data.setItem(book_data);

        items = new OrderItem[1];
        items[0] = item_data;

        item_list = new OrderItemList(items);

        enq_order = new Order();
        enq_order.setCustomer(cust_data);
        enq_order.setItems(item_list);
        enq_order.setOrdermo(new BigDecimal(order_num));
        enq_order.setOrdertype(cust_order_type);

        bookedorders_q = aq_sess.getQueue("OE", "OE_bookedorders_que");
    }
}
```



```

        message = bookedorders_q.createMessage();

        /* エンキューする前に、メッセージ・プロパティに出荷の優先順位を指定します。*/
        m_property = message.getMessageProperty();

        m_property.setPriority(ship_priority);

        obj_payload = message.getObjectPayload();

        obj_payload.setPayloadData(enq_order);

        enq_option = new AQEnqueueOption();

        /* メッセージをエンキューします。*/
        enq_msg_id = bookedorders_q.enqueue(enq_option, message);

        db_conn.commit();

    }
    catch (AQException aq_ex)
    {
        System.out.println("AQ Exception: " + aq_ex);
    }
    catch (SQLException sql_ex)
    {
        System.out.println("SQL Exception: " + sql_ex);
    }
}

```

/\* 地区ごとに、同様の入力済注文キューが作成されます。注文は、注文処理アプリケーションの中央の入力済注文キューから地区の入力済注文キューに移されます。例として、西部地区での入力済注文キューを作成します。ws での出荷について、優先順位のキュー・テーブルを作成します。\*/

```

public static void createWesternShippingQueueTable(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueueTableProperty mqt_prop;
    AQQueueTable        mq_table;
    AQQueueProperty     q_prop;
    AQQueue              bookedorders_q;

```

```
try
{
    /* AQ セッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    /* WS に対して優先順位のキュー・テーブルを作成します。*/
    mqt_prop = new AQQueueTableProperty("BOLADM.order_ttyp");
    mqt_prop.setComment("Western Shipping Priority " +
        "MultiConsumer Orders queue table");
    mqt_prop.setCompatible("8.1");
    mqt_prop.setMultiConsumer(true);
    mqt_prop.setSortOrder("priority,enq_time");

    mq_table = aq_sess.createQueueTable("WS", "WS_orders_pr_mqtab",
        mqt_prop);

    /* 入力済の注文は、優先順位のキュー・テーブルに保存されます。*/
    q_prop = new AQQueueProperty();

    bookedorders_q = aq_sess.createQueue(mq_table, "WS_bookedorders_que",
        q_prop);

    /* キューを開始します。*/
    bookedorders_q.start(true, true);
}
catch (AQException ex)
{
    System.out.println("AQ Exception: " + ex);
}

/* 出荷処理アプリケーションは、地区ごとに、その地区の入力済注文キューからの注文を出荷の優先
順位に従ってデキューし、注文を処理し、処理済注文を出荷済注文キューまたは受注残キューにエン
キューします。*/
}
```

## 時間指定 : 遅延

AQ は、メッセージをエンキューするエンキュー元に遅延間隔を指定させることで、メッセージの遅延配信をサポートします。つまり、デキュー・コールによってメッセージを取り出せるようになるまでの時間を指定できます (11-10 ページの「[メッセージのエンキュー \(メッセージ・プロパティの指定\)](#)」を参照)。エンキューされたメッセージに、デキュー元で使用可能というマークを付ける時機が、遅延間隔によって決定されます。またプロデューサは、メッセージが期限切れになる時刻も指定できます。期限切れになると、メッセージは例外キューに移動されます。

あるメッセージが遅延時間指定付きでエンキューされると、そのメッセージには WAIT 状態というマークが付きます。WAIT 状態のメッセージは、デフォルトのデキュー・コールからはマスクされます。

バックグラウンドのタイム・マネージャ・デーモンは定期的にアクティブになり、内部索引を WAIT 状態にあるすべてのメッセージでスキャンし、遅延時間が過ぎたメッセージには READY 状態というマークを付けます。次に、タイム・マネージャは、メッセージが使用可能になったキューで待機しているすべてのフォアグラウンド・プロセスに、そのことを通知します。

### サンプル・シナリオ

BooksOnLine アプリケーションでは、遅延は繰延べ請求処理を実装するために使用できます。請求処理アプリケーションはキューを定義して、その繰延べ請求処理キューの中には出荷済ですぐには請求されない注文が遅延とともに入力されるようにできます。たとえば、法人顧客のような顧客アカウントの中には、15 日間経過するまで請求が発行されないケースがあります。請求処理アプリケーションは、受け取った出荷済注文メッセージを (shippedorders キューから) デキューし、それが法人顧客からの注文である場合は、遅延とともに繰延べ請求処理キューにエンキューします。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

/\* 遅延が期限切れになるまで注文が参照可能にならないように、繰延べ請求処理を実装するために、注文をエンキューします。\*/

```
CREATE OR REPLACE PROCEDURE defer_billing(deferred_billing_order order_typ)
AS
    defer_bill_queue_name    VARCHAR2(62);
    enqopt                  dbms_aq.enqueue_options_t;
    msgprop                 dbms_aq.message_properties_t;
    enq_msgid               RAW(16);
BEGIN

/* 15 日間の遅延を指定して、繰延べ請求処理キューに注文をエンキューします。*/
    defer_bill_queue_name := 'CBADM.deferbilling_que';
    msgprop.delay := 15*60*60*24;
    dbms_aq.enqueue(defer_bill_queue_name, enqopt, msgprop,
                    deferred_billing_order, enq_msgid);

END;
/
```

## Visual Basic (OO4O) : サンプル・コード

```
set oraqaq = OraDatabase.CreateAQ("CBADM.deferbilling_que")
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
Set OraOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")

Private Sub defer_billing

    OraMsg = OraOrder
    OraMsg.delay = 15*60*60*24
    OraMsg = OraOrder 'OraOrder contains the order details
    Msgid = OraAq.enqueue

End Sub
```

## Java (JDBC) : サンプル・コード

```
public static void defer_billing(Connection db_conn, Order deferred_order)
{
    AQSession          aq_sess;
    AQQueue             def_bill_q;
    AQEnqueueOption     enq_option;
    AQMessageProperty m_property;
    AQMessage            message;
```

```
AQObjectPayload  obj_payload;
byte[]            enq_msg_id;

try
{
    /* AQセッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    def_bill_q = aq_sess.getQueue("CBADM", "deferbilling_que");

    message = def_bill_q.createMessage();

    /* 15 日間の遅延を指定して、繰延べ請求処理キューに注文をエンキューします。*/
    m_property = message.getMessageProperty();
    m_property.setDelay(15*60*60*24);

    obj_payload = message.getObjectPayload();
    obj_payload.setPayloadData(deferred_order);

    enq_option = new AQEnqueueOption();

    /* メッセージをエンキューします。*/
    enq_msg_id = def_bill_q.enqueue(enq_option, message);

    db_conn.commit();
}
catch (Exception ex)
{
    System.out.println("Exception " + ex);
}
}
```

## 時間指定 : 期限切れ

メッセージをエンキューするときに、そのメッセージがいつまで使用可能かという期限切れを指定できます。期限切れ処理では、キュー・モニターが起動されている必要があることに注意してください。

### サンプル・シナリオ

BooksOnLine アプリケーションでは、期限切れは受注残処理にあてられる時間を制御するために使用できます。出荷処理アプリケーションは、処理できない書籍注文を受注残キューに送ります。すべての受注残を1週間以内に必ず出荷するという方針であれば、1週間という期限切れとともにメッセージを受注残キューにエンキューできます。このケースでは、1週間以内に処理されなかった受注残は例外キューに移されて、EXPIRED というメッセージ状態になります。これは、受注残の出荷方針に従って未出荷になっている注文情報にフラグを付けるために使用できます。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
CONNECT BOLADM/BOLADM
/* 受注残を受注残キューに再度エンキューし、7日間の遅延を設定します。つまり、すべての受注残は、
7日以内に処理される必要があります。処理されない場合は、例外キューに移されます。*/
CREATE OR REPLACE PROCEDURE requeue_back_order(sale_region varchar2,
                                                backorder order_typ)
AS
    back_order_queue_name    VARCHAR2(62);
    enqopt                  dbms_aq.enqueue_options_t;
    msgprop                  dbms_aq.message_properties_t;
    enq_msgid                RAW(16);
BEGIN
    /* ディレクトリ・サービスを使用して地区ごとに受注残キューを検索します。*/
    IF sale_region = 'WEST' THEN
        back_order_queue_name := 'WS.WS_backorders_que';
    ELSIF sale_region = 'EAST' THEN
        back_order_queue_name := 'ES.ES_backorders_que';
    ELSE
        back_order_queue_name := 'OS.OS_backorders_que';
    END IF;
```

```

/* 期限切れを7日間に設定した注文をエンキューします。*/
msgprop.expiration := 7*60*60*24;
dbms_aq.enqueue(back_order_queue_name, enqopt, msgprop,
                backorder, enq_msgid);
END;
/

```

## Visual Basic (OO4O) : サンプル・コード

```

set oraaq1 = OraDatabase.CreateAQ("WS.WS_backorders_que")
set oraaq2 = OraDatabase.CreateAQ("ES.ES_backorders_que")
set oraaq3 = OraDatabase.CreateAQ("CBADM.deferbilling_que")
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
Set OraBackOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")

Private Sub Requeue_backorder
    Dim q as oraobject
    If sale_region = WEST then
        q = oraaq1
    else if sale_region = EAST then
        q = oraaq2
    else
        q = oraaq3
    end if

    OraMsg.delay = 7*60*60*24
    OraMsg = OraBackOrder 'OraOrder contains the order details
    Msgid = q.enqueue

End Sub

```

## Java (JDBC) : サンプル・コード

/\* 受注残を受注残キューに再度エンキューし、7日間の遅延を設定します。つまり、すべての受注残は、7日以内に処理される必要があります。処理されない場合は、例外キューに移されます。\*/

```

public static void requeue_back_order(Connection db_conn,
                                     String sale_region, Order back_order)
{
    AQSession      aq_sess;
    AQQueue        back_order_q;
    AQEnqueueOption enq_option;
    AQMessageProperty m_property;
    AQMessage      message;

```

```
AQObjectPayload    obj_payload;
byte[]             enq_msg_id;

try
{
    /* AQ セッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    /* 地区ごとに受注残キューを検索します。*/
    if(sale_region.equals("WEST"))
    {
        back_order_q = aq_sess.getQueue("WS", "WS_backorders_que");
    }
    else if(sale_region.equals("EAST"))
    {
        back_order_q = aq_sess.getQueue("ES", "ES_backorders_que");
    }
    else
    {
        back_order_q = aq_sess.getQueue("OS", "OS_backorders_que");
    }

    message = back_order_q.createMessage();

    m_property = message.getMessageProperty();

    /* 期限切れを7日間に設定した注文をエンキューします。*/
    m_property.setExpiration(7*60*60*24);

    obj_payload = message.getObjectPayload();
    obj_payload.setPayloadData(back_order);

    enq_option = new AQEnqueueOption();

    /* メッセージをエンキューします。*/
    enq_msg_id = back_order_q.enqueue(enq_option, message);

    db_conn.commit();
}
catch (Exception ex)
{
    System.out.println("Exception :" + ex);
}
}
```



## メッセージのグループ化

1つのキューに属しているメッセージをグループ化して1つのセットにし、一度に1ユーザーしか使用できないようにできます。そのためには、メッセージのグループのトランザクション処理化に対応したキュー・テーブルに、そのキューを作成する必要があります（9-5ページの「[キュー・テーブルの作成](#)」を参照）。1つのグループに属するメッセージは、すべて同一のトランザクションで作成される必要があります。また、1つのトランザクションで作成されるメッセージは、すべて同一のグループに属します。この機能によって、複合メッセージを単純メッセージにセグメント化できます。

たとえば、あるキュー宛てのメッセージに請求書情報が含まれている場合、そのメッセージは、ヘッダーのメッセージ、詳細情報のメッセージ、フッターのメッセージで構成されるグループとして作成できます。小さいオブジェクトにセグメント化できるイメージやビデオなどの複合ラージ・オブジェクトがメッセージ・ペイロードにある場合は、メッセージのグループ化が非常に有効です。

グループに含まれるメッセージの一般的なメッセージ・プロパティ（優先順位、遅延、期限切れ）は、単にグループの最初のメッセージ（ヘッダー）のプロパティによってのみ判断され、グループの他のメッセージのプロパティは無視されます。

グループ化メッセージのプロパティは、伝播されても保持されます。ただし、メッセージが伝播される宛先キューも、トランザクション処理でグループ化可能であることが必要な点に注意してください。トランザクション処理でグループ化可能なキューからデキューするときに、グループ化メッセージのプロパティを保持する場合、他にも認識しておく必要がある制限があります（詳細は、8-55ページの「[デキューの方法](#)」および8-68ページの「[デキューのモード](#)」を参照）。

### サンプル・シナリオ

BooksOnLine アプリケーションでは、メッセージのグループ化は新規の注文を扱うために使用できます。各注文には、注文された多数の書籍が1つずつ連続して入っています。Webを経由して注文された項目も同様です。

次の例では、各エンキューが注文の中の個々の書籍に対応し、グループ / トランザクションが1つの完全な注文を表します。最初のエンキューにのみ、顧客情報が含まれています。

OE\_neworders\_que は、グループをトランザクション処理化できる表 OE\_orders\_sqtanb に格納されることに注意してください。プロシージャ new\_order\_enq() および same\_order\_enq() の説明は、サンプル・コードを参照してください。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
connect OE/OE;

/* OE に対してキュー・テーブルを作成します。*/
EXECUTE dbms_aqadm.create_queue_table( \
    queue_table      => 'OE_orders_sqtan', \
    comment          => 'Order Entry Single-Consumer Orders queue table', \
    queue_payload_type => 'BOLADM.order_typ', \
    message_grouping => DBMS_AQADM.TRANSACTIONAL, \
    compatible       => '8.1', \
    primary_instance  => 1, \
    secondary_instance => 2);

/* OE に対して新規注文キューを作成します。*/
EXECUTE dbms_aqadm.create_queue ( \
    queue_name       => 'OE_neworders_que', \
    queue_table      => 'OE_orders_sqtan');

/* OE アカウントでログインします。*/
CONNECT OE/OE;
SET serveroutput on;
/* メッセージをグループ化することによって、いくつかの注文を OE_neworders_que (最初の注文グループ) にエンキューします。*/
EXECUTE BOLADM.new_order_enq('My First   Book', 1, 1001, 'CA');
EXECUTE BOLADM.same_order_enq('My Second Book', 2);
COMMIT;
/
/* 2 番目の注文グループ */
EXECUTE BOLADM.new_order_enq('My Third   Book', 1, 1002, 'WA');
COMMIT;
/
/* 3 番目の注文グループ */
EXECUTE BOLADM.new_order_enq('My Fourth  Book', 1, 1003, 'NV');
EXECUTE BOLADM.same_order_enq('My Fifth   Book', 3);
EXECUTE BOLADM.same_order_enq('My Sixth   Book', 2);
COMMIT;
/
/* 4 番目の注文グループ */
EXECUTE BOLADM.new_order_enq('My Seventh Book', 1, 1004, 'MA');
EXECUTE BOLADM.same_order_enq('My Eighth  Book', 3);
EXECUTE BOLADM.same_order_enq('My Ninth   Book', 2);
COMMIT;
/
```

## Visual Basic (OO4O) : サンプル・コード

この機能は、現在使用できません。

## Java (JDBC) : サンプル・コード

```
public static void createMsgGroupQueueTable(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueueTableProperty sqt_prop;
    AQQueueTable        sq_table;
    AQQueueProperty      q_prop;
    AQQueue              neworders_q;

    try
    {

        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* 単一コンシューマの注文キュー・テーブルを作成します。*/
        sqt_prop = new AQQueueTableProperty("BOLADM.order_ttyp");
        sqt_prop.setComment("Order Entry Single-Consumer Orders queue table");
        sqt_prop.setCompatible("8.1");
        sqt_prop.setMessageGrouping(AQQueueTableProperty.TRANSACTIONAL);

        sq_table = aq_sess.createQueueTable("OE", "OE_orders_sqtab", sqt_prop);

        /* OE に対して新規注文キューを作成します。*/
        q_prop = new AQQueueProperty();

        neworders_q = aq_sess.createQueue(sq_table, "OE_neworders_que",
                                           q_prop);

    }
    catch (AQException ex)
    {
        System.out.println("AQ Exception: " + ex);
    }
}
```

## DEQUEUE 機能

- [デキューの方法](#)
- [複数の受信者](#)
- [ローカルおよびリモートの受信者](#)
- [デキューにおけるメッセージ・ナビゲーション](#)
- [デキューのモード](#)
- [メッセージ到着待機の最適化](#)
- [遅延間隔をおいての再試行](#)
- [例外処理](#)
- [ルールベースのサブスクリプション](#)
- [Listen 機能](#)

## デキューの方法

キューからメッセージをデキューするには、相関識別子またはメッセージ識別子の 2 つの方法があります。

相関識別子はユーザー定義のメッセージ・プロパティ (VARCHAR2 データ型) で、メッセージ識別子はシステムによって割り当てられる値 (RAW データ型) です。1 つのキューに同じ相関識別子を持つメッセージが複数存在することがありますが、同じメッセージ識別子を持つメッセージが複数存在することはありません。相関識別子によるデキュー・コールは、まず内容を調べてから削除するという REMOVE (削除) モードと LOCK (ロック) モードの組合せではなく、特定の関心の対象になるメッセージを直接削除します。したがって、相関識別子には、通常、ペイロードで最も有効な属性が含まれます。同じ相関識別子を持つメッセージが複数ある場合、メッセージ同士の順序付け (エンキュー順序) がデキュー・コールでは変更されることがあります。一連のデキュー・コールの間は、first message ナビゲーション・オプションを指定しないと相関識別子を変更できません。

2 通りのデキュー方法のどちらでメッセージをデキューしても、グループ化メッセージのプロパティは変更されることがあります (詳細は、8-51 ページの「[メッセージのグループ化](#)」および 8-64 ページの「[デキューにおけるメッセージ・ナビゲーション](#)」を参照)。

### サンプル・シナリオ

次の BooksOnLine によるシナリオでは、東部向け出荷サイトに届いた至急 (RUSH) 注文が、最初に処理されています。そのために、注文タイプ (至急 / 普通) を含めて定義されている相関識別子を使用してメッセージをデキューします。メッセージ識別子を使用するデキューの詳細は、8-68 ページの「[デキューのモード](#)」にある例の get\_northamerican\_orders プロシージャを参照してください。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
CONNECT boladm/boladm;
```

```
/* 単一コンシューマ・キューにエンキューするためのプロシージャを作成します。*/
create or replace procedure get_rushtitles(consumer in varchar2) as
```

```

deq_cust_data      BOLADM.customer_typ;
deq_book_data      BOLADM.book_typ;
deq_item_data      BOLADM.orderitem_typ;
deq_msgid          RAW(16);
dopt               dbms_aq.dequeue_options_t;
mprop              dbms_aq.message_properties_t;
deq_order_data     BOLADM.order_typ;
qname              varchar2(30);
```

```
no_messages          exception;
pragma exception_init (no_messages, -25228);
new_orders           BOOLEAN := TRUE;

begin

    dopt.consumer_name := consumer;
    dopt.wait := 1;
    dopt.correlation := 'RUSH';

    IF (consumer = 'West_Shipping') THEN
        qname := 'WS.WS_bookedorders_que';
    ELSIF (consumer = 'East_Shipping') THEN
        qname := 'ES.ES_bookedorders_que';
    ELSE
        qname := 'OS.OS_bookedorders_que';
    END IF;

    WHILE (new_orders) LOOP
        BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_data,
                msgid => deq_msgid);
            commit;

            deq_item_data := deq_order_data.items(1);
            deq_book_data := deq_item_data.item;

            dbms_output.put_line(' rushorder book_title: ' ||
                                deq_book_data.title ||
                                ' quantity: ' || deq_item_data.quantity);
        EXCEPTION
            WHEN no_messages THEN
                dbms_output.put_line (' ---- NO MORE RUSH TITLES ---- ');
                new_orders := FALSE;
        END;
    END LOOP;

end;
/

CONNECT EXECUTE on get_rushtitles to ES;
```

```

/* 注文をデキューします。*/
CONNECT ES/ES;

/* 東部向け出荷のすべての至急注文タイトルをデキューします。*/
EXECUTE BOLADM.get_rushtitles('East_Shipping');

```

## Visual Basic (OO4O) : サンプル・コード

```

set oraaq1 = OraDatabase.CreateAQ("WS.WS_backorders_que")
set oraaq2 = OraDatabase.CreateAQ("ES.ES_backorders_que")
set oraaq3 = OraDatabase.CreateAQ("CBADM.deferbilling_que")
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
Set OraBackOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")

Private Sub Requeue_backorder
    Dim q as oraobject
    If sale_region = WEST then
        q = oraaq1
    else if sale_region = EAST then
        q = oraaq2
    else
        q = oraaq3
    end if

    OraMsg.delay = 7*60*60*24
    OraMsg = OraBackOrder 'OraOrder contains the order details
    Msgid = q.enqueue

End Sub

```

## Java (JDBC) : サンプル・コード

```

public static void getRushTitles(Connection db_conn, String consumer)
{
    AQSession      aq_sess;
    Order           deq_order;
    byte[]          deq_msgid;
    AQDequeueOption deq_option;
    AQMessageProperty msg_prop;
    AQQueue         bookedorders_q;
    AQMessage        message;
    AQObjectPayload  obj_payload;
    boolean          new_orders = true;

```

```
try
{
    /* AQ セッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    deq_option = new AQDequeueOption();

    deq_option.setConsumerName(consumer);
    deq_option.setWaitTime(1);
    deq_option.setCorrelation("RUSH");

    if(consumer.equals("West_Shipping"))
    {
        bookedorders_q = aq_sess.getQueue("WS", "WS_bookedorders_que");
    }
    else if(consumer.equals("East_Shipping"))
    {
        bookedorders_q = aq_sess.getQueue("ES", "ES_bookedorders_que");
    }
    else
    {
        bookedorders_q = aq_sess.getQueue("OS", "OS_bookedorders_que");
    }

    while(new_orders)
    {
        try
        {
            /* メッセージをデキューします。*/
            message = bookedorders_q.dequeue(deq_option, Order.getFactory());

            obj_payload = message.getObjectPayload();

            deq_order = (Order) obj_payload.getPayloadData();

            System.out.println("Order number " + deq_order.getOrderno() +
                               " is a rush order");

        }
        catch (AQException agex)
        {
            new_orders = false;
            System.out.println("No more rush titles");
            System.out.println("Exception-1: " + agex);
        }
    }
}
```



```
        }  
    }  
}  
catch (Exception ex)  
{  
    System.out.println("Exception-2: " + ex);  
}  
}
```

## 複数の受信者

コンシューマは、DBMS\_AQADM.ADD\_SUBSCRIBER プロシージャの AQ\$\_AGENT 型で使用されている名前、またはメッセージ・プロパティの受信者リストを指定することで、複数コンシューマの通常キューからメッセージをデキューできます（11-10 ページの「サブスクライバの追加」または 9-53 ページの「メッセージのエンキュー（メッセージ・プロパティの指定）」を参照）。

- PL/SQL では、コンシューマ名は dequeue\_options\_t レコードの consumer\_name フィールドを使用して提供されます。
- OCI では、コンシューマ名は OCISetAttr プロシージャを使用して、テキスト文字列を OCI\_DTYPE\_AQDEQ\_OPTIONS 記述子の OCI\_ATTR\_CONSUMER\_NAME に指定して提供されます。
- OO4O では、コンシューマ名は OraAQ オブジェクトのコンシューマ・プロパティを設定することで提供されます。

複数のプロセスまたはオペレーティング・システム・スレッドが、1つのキューから同じ consumer\_name を使用して同時にデキューすることがあります。そのような場合、AQ はキューの先頭にあるロックされていない最初のメッセージをコンシューマに提供します。デキュー中に特定のメッセージのメッセージ ID が指定されない限り、コンシューマは READY 状態のメッセージをデキューできます。

メッセージが PROCESSED（処理済）とみなされるのは、所定のコンシューマ全員がメッセージのデキューに成功したときのみです。メッセージが EXPIRED（期限切れ）とみなされるのは、1つまたは複数のコンシューマが EXPIRATION 時刻までにそのメッセージをデキューしなかったときです。期限切れになったメッセージは、例外キューに移動します。

例外キューも必ず複数コンシューマ・キューです。複数コンシューマ・キューから移された期限切れメッセージは、所定の受信者にはデキューできません。ただし、デキュー・オプションのコンシューマ名に NULL を指定することによって、REMOVE モードで1回のみデキューできます。したがって、デキューの観点からすると、複数コンシューマの例外キューは単一コンシューマ・キューのように動作します。これは、それぞれの期限切れメッセージは、NULL コンシューマ名を使用して1回しかデキューできないためです。複数コンシューマ例外キューを作成したキュー・テーブルが、互換パラメータを使用せずに作成された場合、または互換パラメータを 8.0 に設定して作成された場合は、期限切れメッセージはメッセージ ID を指定する方法でしかデキューできないことに注意してください。

リリース 8.0.x では、異なる consumer\_names を使用する 2 つ以上のプロセス / スレッドがキューからデキューする場合、指定されたメッセージを LOCKED モードまたは REMOVE モードでいつでもデキューできるプロセス / スレッドは、1 つしかありません。つまり、メッセージをロックしたコンシューマが、そのトランザクションをコミットまたは中断してメッセージのロックを解除するまで、そのメッセージをデキューする他のコンシューマは待機する必要があるということです。Oracle8 が同一メッセージに対する異なるコンシューマの同時実行をサポートしていなかったのに対し、Oracle8i では、すべてのコンシューマが同一メッセージに同時にアクセスできます。その結果、2 つのプロセス / スレッドが別々の consumer\_name で同一メッセージをデキューしても、互いにブロックしません。AQ は、

メッセージのデキュー作業とキューからの削除処理を分離することで、この改善を実現しています。Oracle8i では、複数コンシューマ・キューからメッセージを削除できるのはキュー・モニターのみです。これによって、デキュー者はキュー・テーブルのメッセージをロックせずにデキュー操作を完了できます。キュー・モニターは、すべてのコンシューマが処理を完了したメッセージを複数コンシューマ・キューから削除する作業を毎分約 1 回行うため、メッセージが完全に処理された後キューから物理的に削除されるまでの遅延があります。

## ローカルおよびリモートの受信者

複数コンシューマ・キュー内のメッセージのコンシューマ（そのキューのサブスクリバであるか、エンキュー元の受信者リストに含まれている受信者）は、ローカルでもリモートでもかまいません。

- ローカル・コンシューマは、プロデューサがそのメッセージをエンキューしたキューからデキューします。ローカル・コンシューマの場合は、AQ\$\_AGENT 型の NAME は NULL 以外で、ADDRESS および PROTOCOL フィールドは NULL です（2-3 ページの「[エージェント](#)」を参照）。
- リモート・コンシューマは、メッセージがエンキューされたものとは別のキュー（ただし、ペイロード型はソース・キューと同じ）からデキューします。ユーザーは、このリモート・コンシューマの特徴を理解し、AQ 伝播機能を使用してリモート・コンシューマを使用する必要があります。リモート・コンシューマは次の 3 つのカテゴリに分類されます。
  - a. ADDRESS フィールドが同一データベース内のキューを参照しているもの。この場合、コンシューマは同一データベースの別のキューからメッセージをデキューすることになります。アドレスの形式は [schema].queue\_name で、queue\_name（オプションで、スキーマ名によって修飾されることもある）がターゲット・キューです。スキーマが指定されないと、ADD\_SUBSCRIBER プロシージャを実行しているカレント・ユーザーのスキーマまたはエンキューされたスキーマが使用されます（9-53 ページの「[サブスクリバの追加](#)」または 11-5 ページの「[メッセージのエンキュー](#)」を参照）。このようリモート・コンシューマへの伝播のスケジュールには、宛先に NULL（デフォルト値）を指定した DBMS\_AQADM.SCHEDULE\_PROPAGATION コマンドを使用します（9-65 ページの「[キューの伝播のスケジュール](#)」を参照）。
  - b. ADDRESS フィールドが別のデータベース内のキューを参照しているもの。この場合の別のデータベースは、データベース・リンクによって到達可能で、PROTOCOL が NULL または 0（ゼロ）である必要があります。アドレスの形式は、[schema].queue\_name@dblink になります。スキーマが指定されないと、ADD\_SUBSCRIBER プロシージャを実行しているカレント・ユーザーのスキーマまたはエンキューされたスキーマが使用されます。データベース・リンクが完全な修飾名でない場合（ドメイン名が指定されない場合）、db\_domain init.ora パラメータで指定されたデフォルトのドメインが使用されます。データベース・リンクを宛先とした DBMS\_AQADM.SCHEDULE\_PROPAGATION プロシージャを使用して伝播をスケジュールします。AQ は、シノニムを使用してキューまたはデータベース・リンクを参照する方法はサポートしていません。
  - c. ADDRESS フィールドがサード・パーティのプロトコルによって到達できる宛先を参照しているもの。サード・パーティ・ソフトウェアのドキュメントを参照してデータベース・リンクの ADDRESS および PROTOCOL をどのように指定するか、またどのように伝播をスケジュールするかを決定する必要があります。

コンシューマがリモートの場合、伝播されたメッセージがリモート・キューからデキューされていないにもかかわらず、ソース・キューでは、伝播された直後に PROCESSED とマークされます。同様に、伝播されたメッセージがリモート・キューで期限切れになると、そのメッセージはローカル・キューの例外キューではなく、リモート・キューのキュー・テーブルの DEFAULT 例外キューに移動されます。この2つのケースからわかるように、現状では、AQ は例外をソース・キューに伝播しません。キュー・テーブル・ビュー (AQ\$<queue\_table>) の MSGID 列および ORIGINAL\_MSGID 列を使用して、伝播されたメッセージを連鎖させることができます。メッセージ ID が m1 のメッセージがリモート・キューに伝播されると、m1 はリモート・キューの ORIGINAL\_MSGID 列に格納されます。

DELAY、EXPIRATION および PRIORITY パラメータは、ローカル・コンシューマにもリモート・コンシューマにも同様に適用されます。AQ は伝播による遅延を見込んで、DELAY および EXPIRATION パラメータを調整します。たとえば、EXPIRATION が 1 時間で、メッセージが 15 分後に伝播された場合、リモート・キューでの期限切れは 45 分に設定されます。

データベースがメッセージ伝播を処理するため、OO4O はリモートおよびローカルの受信者を区別しません。メッセージをデキューするには、ローカルおよびリモートの受信者に対して、コール / ステップの順序が同じである必要があります。

## デキューにおけるメッセージ・ナビゲーション

キューからメッセージを選択するには、いくつかのオプションがあります。first message (最初のメッセージ) を選択できます。または、いったんメッセージを選択してキュー内での位置を (たとえば 4 番目のメッセージとして) 確立してから、next message (次のメッセージ) を取り出せます。

そのキューがトランザクション処理でグループ化できる場合は、前述の選択は少し違ったものになります。

- first message が要求されると、デキュー位置はキューの最初にリセットされます。
- next message が要求されると、位置は同一トランザクションの次のメッセージに設定されます。
- next transaction (次のトランザクション) が要求されると、位置は次のトランザクションの最初のメッセージに設定されます。

関連識別子を指定してデキューするか、メッセージ識別子を指定してデキューするか、または任意のトランザクション・メッセージをデキューおよびコミットすると、グループ化トランザクション・プロパティは否定されることに注意してください (8-55 ページの「[デキューの方法](#)」を参照)。

next message または next transaction オプションを使用したプログラムがキューの中をナビゲートしてキューの最後に到達し、待機時間を 0 (ゼロ) 以外に指定していた場合、ナビゲート位置は自動的にそのキューの先頭に変更されます。

### サンプル・シナリオ

次の BooksOnLine のシナリオでは、エンキューに関連してすでに説明したメッセージ・グループ化の例を続けます (8-55 ページの「[デキューの方法](#)」を参照)。

get\_orders() プロシージャは、OE\_neworders\_que から注文をデキューします。各トランザクションがそれぞれの注文を参照し、各メッセージがその注文に含まれる各書籍に対応していることを思い出してください。get\_orders() プロシージャは、メッセージをループして書籍注文をデキューします。最初のデキューの前に、first message オプションでキューの先頭にリセットします。それから、next message ナビゲーション・オプションで、注文 (トランザクション) から次の書籍 (メッセージ) を取り出します。現行のグループ / トランザクションのすべてのメッセージがフェッチされたというエラー・メッセージが戻されたら、ナビゲーション・オプションを next transaction に変更して、次の注文の最初の書籍を取り出します。次に、再度 next message オプションに戻して、同一トランザクションの次のメッセージをフェッチします。こうして、すべての注文 (トランザクション) がフェッチされるまで、繰り返します。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```

CONNECT boladm/boladm;

create or replace procedure get_new_orders as

    deq_cust_data          BOLADM.customer_typ;
    deq_book_data          BOLADM.book_typ;
    deq_item_data          BOLADM.orderitem_typ;
    deq_msgid              RAW(16);
    dopt                   dbms_aq.dequeue_options_t;
    mprop                  dbms_aq.message_properties_t;
    deq_order_data         BOLADM.order_typ;
    qname                  VARCHAR2(30);
    no_messages             exception;
    end_of_group            exception;
    pragma exception_init   (no_messages, -25228);
    pragma exception_init   (end_of_group, -25235);
    new_orders              BOOLEAN := TRUE;

BEGIN

    dopt.wait := 1;
    dopt.navigation := DBMS_AQ.FIRST_MESSAGE;
    qname := 'OE.OE_neworders_que';
    WHILE (new_orders) LOOP
        BEGIN
            LOOP
                BEGIN
                    dbms_aq.dequeue(
                        queue_name          => qname,
                        dequeue_options     => dopt,
                        message_properties  => mprop,
                        payload              => deq_order_data,
                        msgid                => deq_msgid);

                    deq_item_data := deq_order_data.items(1);
                    deq_book_data := deq_item_data.item;
                    deq_cust_data := deq_order_data.customer;

                    IF (deq_cust_data IS NOT NULL) THEN
                        dbms_output.put_line(' **** NEXT ORDER **** ');
                        dbms_output.put_line('order_num: ' ||
                                                deq_order_data.orderno);
                        dbms_output.put_line('ship_state: ' ||
                                                deq_cust_data.state);

```

```

        END IF;
        dbms_output.put_line(' ---- next book ---- ');
        dbms_output.put_line(' book_title: ' ||
                               deq_book_data.title ||
                               ' quantity: ' || deq_item_data.quantity);
    EXCEPTION
        WHEN end_of_group THEN
            dbms_output.put_line ('*** END OF ORDER ***');
            commit;
            dopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
    END;
END LOOP;
EXCEPTION
    WHEN no_messages THEN
        dbms_output.put_line (' ---- NO MORE NEW ORDERS ---- ');
        new_orders := FALSE;
    END;
END LOOP;

END;
/

CONNECT EXECUTE ON get_new_orders to OE;

/* 注文をデキューします。*/
CONNECT OE/OE;
EXECUTE BOLADM.get_new_orders;

```

## Visual Basic (OO4O) : サンプル・コード

```

Dim OraSession as object
Dim OraDatabase as object
Dim OraAq as object
Dim OraMsg as Object
Dim OraOrder,OraItemList,OraItem,OraBook,OraCustomer as Object
Dim Msgid as String

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.DbOpenDatabase("", "boladm/boladm", 0&)
set oraaq = OraDatabase.CreateAQ("OE.OE_neworders_que")
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
    OraAq.wait = 1
OraAq.Navigation = ORAAQ_DQ_FIRST_MESSAGE

```



```

private sub get_new_orders
    Dim MsgIsDequeued as Boolean
    On Error goto ErrHandler
    MsgIsDequeued = TRUE
    msgid = q.Dequeue
    if MsgIsDequeued then
        set OraOrder = OraMsg
        OraItemList = OraOrder("items")
        OraItem = OraItemList(1)
        OraBook = OraItem("item")
        OraCustomer = OraOrder("customer")

        ' Populate the textboxes with the values
        if( OraCustomer ) then
            if OraAq.Navigation <> ORAAQ_DQ_NEXT_MESSAGE then
                MsgBox " ***** NEXT ORDER *****"
            end if
            txt_book_orderno = OraOrder("orderno")
            txt_book_shipstate = OraCustomer("state")
        End if
        OraAq.Navigation = ORAAQ_DQ_NEXT_MESSAGE
        txt_book_title = OraBook("title")
        txt_book_qty = OraItem("quantity")
    Else
        MsgBox " ***** END OF ORDER *****"
    End if

ErrHandler :
    'Handle error case, like no message etc
    If OraDatabase.LastServerErr = 25228 then
        OraAq.Navigation = ORAAQ_DQ_NEXT_TRANSACTION
        MsgIsDequeued = FALSE
        Resume Next
    End If
    'Process other errors
end sub

```

## Java (JDBC) : サンプル・コード

今回のリリースでは、例は記載していません。

## デキューのモード

デキュー要求によって、メッセージを参照または削除できます（11-45 ページの「[メッセージのデキュー](#)」を参照）。

- メッセージを参照するときは、BROWSE（ブラウズ）モードまたは LOCKED（ロック）モードを使用します。
- メッセージを削除するときは、REMOVE（削除）モードまたは REMOVE WITH NO DATA（データ削除を伴わない）モードを使用します。

メッセージは、ブラウズ後でも処理可能です。同様に、ロック後でも、トランザクションのコミットまたはロールバックによってそのロックが解除されると、メッセージは処理可能です。どちらかの削除モードで一度メッセージが削除されると、デキュー要求は使用できません。

REMOVE\_NODATA モードでメッセージがデキューされた場合、メッセージのペイロードは取り出されません。このモードは、ユーザーが先に BROWSE モードでデキューしてペイロードを調べてあるときに有効です。このようにして、ペイロードの取出しというオーバーヘッドを回避できます。

キューに保存時間が指定されている場合、メッセージは削除された後でもキュー・テーブルに保存されます。例外キューのメッセージは保存できません（詳細は、8-87 ページの「[例外処理](#)」を参照）。一般に、データを伴わないメッセージの削除は、（あらかじめ BROWSE/LOCKED モードでデキューしていて）ペイロード内容がわかっていたり、メッセージが使用される見込みがないときに使用されます。

メッセージをブラウズした後は、そのメッセージを再度デキューできる保証がないことに注意してください。これは、同時ユーザーのデキュー・コールによってそのメッセージが削除される可能性があるためです。一度参照したメッセージが同時ユーザーによってデキューされないようにするには、LOCKED モードでメッセージを参照する必要があります。

BROWSE モードの使用に特に注意が必要な理由が他にもあります。待機時間が 0（ゼロ）以外に指定されていて、ナビゲート位置がキューの最後に到達した場合、デキュー位置は自動的にそのキューの先頭に変更されます。したがって、BROWSE モードで next message ナビゲーション・オプションおよび 0（ゼロ）以外の待機時間を指定してデキューを繰り返すと、何度も同一メッセージをデキューすることがあります。キューに対する最初のデキューに待機時間を 0（ゼロ）以外で指定して、それ以降のデキュー・コールには待機時間 0（ゼロ）の next message ナビゲーション・オプションを使用することをお勧めします。デキュー・コールで「end of queue」エラー・メッセージが戻された場合は、デキュー位置を first message ナビゲーション・オプションを使用して明示的にキューの先頭に設定できます。このようにすると、そのキューのメッセージを再度ブラウズできます。

## サンプル・シナリオ

次の BooksOnLine のシナリオでは、海外からの注文（メキシコおよびカナダ向け）が、通商政策と送料割引の理由で別々に処理されます。したがって、（他の同時ユーザーが削除できないように）メッセージをロック・モードで参照し、顧客の国（メッセージ・ペイロード）をチェックします。顧客の国がメキシコまたはカナダの場合、（ペイロードはすでにわかっているため）REMOVE WITH NO DATA モードでメッセージをキューから削除します。これを実行しないと、そのメッセージのロックはコミット・コールによって解除されるためです。削除モードのデキュー・コールには、ロック・モードのデキュー・コールによって取得したメッセージ識別子を使用することに注意してください。  
shipping\_bookedorder\_deq コールは、BROWSE モードの使用方を示します（このプロセスについては、サンプル・コードを参照）。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
CONNECT boladm/boladm;

create or replace procedure get_northamerican_orders as

deq_cust_data          BOLADM.customer_typ;
deq_book_data          BOLADM.book_typ;
deq_item_data          BOLADM.orderitem_typ;
deq_msgid              RAW(16);
dopt                   dbms_aq.dequeue_options_t;
mprop                  dbms_aq.message_properties_t;
deq_order_data         BOLADM.order_typ;
deq_order_nodata       BOLADM.order_typ;
qname                  VARCHAR2(30);
no_messages            exception;
pragma exception_init  (no_messages, -25228);
new_orders             BOOLEAN := TRUE;

begin

    dopt.consumer_name := consumer;
    dopt.wait := DBMS_AQ.NO_WAIT;
    dopt.navigation := dbms_aq.FIRST_MESSAGE;
    dopt.dequeue_mode := DBMS_AQ.LOCKED;

    qname := 'OS.OS_bookedorders_que';

    WHILE (new_orders) LOOP
        BEGIN
            dbms_aq.dequeue (
                queue_name => qname,
```

```
        dequeue_options => dopt,
        message_properties => mprop,
        payload => deq_order_data,
        msgid => deq_msgid);

deq_item_data := deq_order_data.items(1);
deq_book_data := deq_item_data.item;
deq_cust_data := deq_order_data.customer;

IF (deq_cust_data.country = 'Canada' OR
    deq_cust_data.country = 'Mexico' ) THEN

    dopt.dequeue_mode := dbms_aq.REMOVE_NODATA;
    dopt.msgid := deq_msgid;
    dbms_aq.dequeue(
        queue_name => qname,
        dequeue_options => dopt,
        message_properties => mprop,
        payload => deq_order_nodata,
        msgid => deq_msgid);
    commit;

    dbms_output.put_line(' **** next booked order **** ');
    dbms_output.put_line('order_no: ' || deq_order_data.orderno ||
        ' book_title: ' || deq_book_data.title ||
        ' quantity: ' || deq_item_data.quantity);
    dbms_output.put_line('ship_state: ' || deq_cust_data.state ||
        ' ship_country: ' || deq_cust_data.country ||
        ' ship_order_type: ' || deq_order_data.ordertype);

END IF;

commit;
dopt.dequeue_mode := DBMS_AQ.LOCKED;
dopt.msgid := NULL;
dopt.navigation := dbms_aq.NEXT_MESSAGE;
EXCEPTION
    WHEN no_messages THEN
        dbms_output.put_line (' ---- NO MORE BOOKED ORDERS ---- ');
        new_orders := FALSE;
END;
END LOOP;

end;
/
```

```

CONNECT EXECUTE on get_northamerican_orders to OS;

CONNECT ES/ES;

/* 東部向け出荷のすべての入力済注文をブラウズします。*/
EXECUTE BOLADM.shipping_bookedorder_deq('East_Shipping', DEMS_AQ.BROWSE);

CONNECT OS/OS;

/* 海外向け出荷のアメリカ合衆国内のすべての注文をデキューします。*/
EXECUTE BOLADM.get_northamerican_orders;

```

## Visual Basic (OO4O) : サンプル・コード

OO4O は、前述したデキューのすべてのモードをサポートします。可能な値は次のとおりです。

- ORAAQ\_DQ\_BROWSE (1) - デキュー時にロックしません。
- ORAAQ\_DQ\_LOCKED (2) - メッセージを読み込んで、書き込みロックを取得します。
- ORAAQ\_DQ\_REMOVE (3) (デフォルト) - メッセージを読み込んで、更新または削除します。

```

Dim OraSession as object
Dim OraDatabase as object
Dim OraAq as object
Dim OraMsg as Object
Dim OraOrder,OraItemList,OraItem,OraBook,OraCustomer as Object
Dim Msgid as String

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.DbOpenDatabase("", "boladm/boladm", 0&)
set oraaq = OraDatabase.CreateAQ("OE.OE_neworders_que")
OraAq.DequeueMode = ORAAQ_DQ_BROWSE

```

## Java (JDBC) : サンプル・コード

```

public static void get_northamerican_orders(Connection db_conn)
{

    AQSession        aq_sess;
    Order             deq_order;
    Customer          deq_cust;

```

```
String          cust_country;
byte[]          deq_msgid;
AQDequeueOption deq_option;
AQMessageProperty msg_prop;
AQQueue         bookedorders_q;
AQMessage       message;
AQObjectPayload obj_payload;
boolean         new_orders = true;

try
{
    /* AQセッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    deq_option = new AQDequeueOption();

    deq_option.setConsumerName("Overseas_Shipping");
    deq_option.setWaitTime(AQDequeueOption.WAIT_NONE);
    deq_option.setNavigationMode(AQDequeueOption.NAVIGATION_FIRST_MESSAGE);
    deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_LOCKED);

    bookedorders_q = aq_sess.getQueue("OS", "OS_bookedorders_que");

    while(new_orders)
    {
        try
        {
            /* メッセージをデキューします (ロック付きでブラウズします)。*/
            message = bookedorders_q.dequeue(deq_option, Order.getFactory());

            obj_payload = message.getObjectPayload();

            deq_msgid = message.getMessageId();
            deq_order = (Order) obj_payload.getPayloadData();

            deq_cust = deq_order.getCustomer();

            cust_country = deq_cust.getCountry();

            if(cust_country.equals("Canada") ||
               cust_country.equals("Mexico"))
            {
                deq_option.setDequeueMode(
                    AQDequeueOption.DEQUEUE_REMOVE_NODATA);
                deq_option.setMessageId(deq_msgid);
            }
        }
    }
}
```

```
/* メッセージを削除します。*/
bookedorders_q.dequeue(deq_option, Order.getFactory());

System.out.println("---- next booked order -----");
System.out.println("Order no: " + deq_order.getOrderno());
System.out.println("Ship state: " + deq_cust.getState());
System.out.println("Ship country: " + deq_cust.getCountry());
System.out.println("Order type: " + deq_order.getOrderType());

}

db_conn.commit();

deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_LOCKED);
deq_option.setMessageId(null);
deq_option.setNavigationMode(
    AQDequeueOption.NAVIGATION_NEXT_MESSAGE);
}
catch (AQException agex)
{
    new_orders = false;
    System.out.println("--- No more booked orders ---");
    System.out.println("Exception-1: " + agex);
}
}

}
catch (Exception ex)
{
    System.out.println("Exception-2: " + ex);
}
}
```

## メッセージ到着待機の最適化

AQ の最も重要な機能の 1 つは、新しくエンキューされるメッセージまたは READY 状態になるメッセージに、1 つまたは複数のキュー上で、アプリケーションを待機させることができます。DEQUEUE 操作によって、あるキューへのメッセージ到着を待機することができます (11-45 ページの「[メッセージのデキュー](#)」を参照)、または、LISTEN 操作によって複数のキューへのメッセージ到着を待機できます (11-23 ページの「[1 個 \(複数個\) のキューのリスニング](#)」を参照)。

ブロック (待機) している DEQUEUE コールが戻るとき、メッセージ・プロパティおよびメッセージ・ペイロードが戻されます。それに対して、ブロックしている LISTEN コールが戻るときは、メッセージが届いたキューの名前のみが戻ります。メッセージをデキューするためには、その後に DEQUEUE 操作をする必要があります。

アプリケーションは、必要に応じて AQ のメッセージ到着待機時間を示すタイムアウトを 0 (ゼロ) または任意の秒数に指定できます。デフォルトでは、そのキューにメッセージが到着するまで、待機することになっています。この最適化は、2 つの点で重要です。まず、アプリケーションからメッセージをポーリングし続けるという負担がなくなります。また、アプリケーションは、新しいメッセージがエンキューされるか、DELAY 時間が過ぎて READY 状態になるまでブロックし続けるため、CPU およびネットワーク・リソースの節約になります。リリース 8.1.5 では、アプリケーションは例外キューのブロッキング・デキューをして、EXPIRED メッセージを待機することもできます。

デキューによってブロックされたプロセスまたはスレッドは、新しいメッセージに DELAY が指定されていない場合はエンキュー元が直接アクティブにし、DELAY または EXPIRATION 時間が経過した場合はキュー・モニター・プロセスがアクティブにします。アプリケーションは、エンキュー元がエンキューするキューに届くメッセージを待機する他に、DBMS\_AQADM.SCHEDULE\_PROPAGATION による伝播がスケジュールされている場合にはリモート・キューに届くメッセージも待機できます。この場合、AQ のプロパゲータは、メッセージの伝播後に、ブロックされたデキュー元をアクティブにします。

### サンプル・シナリオ

BooksOnLine の例では、「[デキューの方法](#)」で説明した `get_rushtitles` プロシージャがデキュー・コールの引数 `dequeue_options` の待機時間を 1 秒に指定しています。待機時間は、次のサンプル・コードでも示すとおり、別の方法でも指定できます。

- 待機時間を 10 秒に指定すると、そのキューのメッセージが使用可能になるまで、デキュー・コールは 10 秒のタイムアウト指定でブロックされます。つまり、10 秒が過ぎてもそのキューにメッセージがない場合は、デキュー・コールはメッセージなしで戻ります。事前定義された定数をこの待機時間に割り当てすることもできます。



- 待機時間を DBMS\_AQ.NO\_WAIT に指定すると、待機時間には 0（ゼロ）秒が設定されます。この場合、そのキューにメッセージがない場合でも、デキュー・コールはすぐに戻ります。
- 待機時間を DBMS\_AQ.FOREVER に指定すると、そのキューのメッセージが使用可能になるまで、デキュー・コールはタイムアウト指定なしでブロックされます。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
/* dopt は、dbms_aq.dequeue_options_t 型の変数です。デキュー待機時間を 10 秒に設定します。*/  
dopt.wait := 10;
```

```
/* デキュー待機時間を 0 秒に設定します。*/  
dopt.wait := DBMS_AQ.NO_WAIT;
```

```
/* デキュー待機時間を無制限に設定します。*/  
dopt.wait := DBMS_AQ.FOREVER;
```

### Visual Basic (OO4O) : サンプル・コード

OO4O は、メッセージの非同期デキューをサポートします。まず、モニターが特定のキューに対して開始されます。ユーザー基準に合うメッセージがデキューされると、ユーザーのコールバック・オブジェクトが通知されます。

### Java (JDBC) : サンプル・コード

```
AQDequeueOption deq-opt;  
deq-opt = new AQDequeueOption ();
```

## 非同期通知

この機能によって、OCI クライアントは、関心があるキューにメッセージがあるときに通知を受け取ることができます。このクライアントは、この機能を使用して複数のサブスクリプションを監視できます。クライアントは、自分自身のサブスクリプションに関する通知を受け取るためにデータベースと接続されている必要はありません。

ユーザーは、OCI 関数 `OCISubscriptionRegister` を使用して、キュー内のどんなメッセージに関心があるかを登録します（11-56 ページの「[通知登録](#)」を参照）。

---

### 参照：

- OCI オペレーションの通知登録の詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。
- 

クライアントは、新しいメッセージがエンキューされるたびに起動されるコールバック関数を指定できます。非永続キューの場合、メッセージは通知の一部としてクライアントに配信されます。永続キューの場合は、通知の一部としてメッセージ・プロパティのみが配信されます。つまり、永続キューの場合にクライアントがメッセージ内容にアクセスするには、明示的にデキューする必要があります。

## サンプル・シナリオ

BooksOnLine アプリケーションでは、顧客は FedEx による出荷（優先順位 1）、優先扱い航空便による出荷（優先順位 2）または通常地上便による出荷（優先順位 3）のいずれかの方法を要求できます。

出荷処理アプリケーションは、注文された書籍をユーザーの要求に従って出荷します。BooksOnLine は、毎日、それぞれの出荷方法が何件要求されるかに関心があります。アプリケーションは、非同期通知機能を用いてこの目的に使用します。`WS.WS_bookedorders_queue` に対して通知を登録し、そのキューに新しいメッセージがあることを通知されると、アプリケーションはメッセージの優先順位に従って適切な出荷方法の件数を更新します。

## Visual Basic (OO4O) : サンプル・コード

OO4O のオンライン・ヘルプの「Monitoring Message」を参照してください。

## Java (JDBC) : サンプル・コード

この機能は、Java API ではサポートされません。

## C (OCI) : サンプル・コード

この例では OCIRegister の使用方法を示します。この OCI クライアント・プログラムは、出荷サイトで、FEDEX、AIR および GROUND の各出荷方法の注文が何件あるかを追跡します。メッセージの優先順位フィールドによって、希望の出荷方法を判断できます。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>
#ifdef WIN32COMMON
#define sleep(x)    Sleep(1000*(x))
#endif
static text *username = (text *) "WS";
static text *password = (text *) "WS";

static OCIEnv *envhp;
static OCIServer *srvhp;
static OCIError *errhp;
static OCISvcCtx *svchp;

static void checkerr(/*_ OCIError *errhp, sword status _*/);

struct ship_data
{
    ub4 fedex;
    ub4 air;
    ub4 ground;
};

typedef struct ship_data ship_data;

int main(/*_ int argc, char *argv[] _*/);

/* コールバックを通知します。*/
ub4 notifyCB(ctx, subscrhp, pay, payl, desc, mode)
dvoid *ctx;
OCISubscription *subscrhp;
dvoid *pay;
ub4 payl;
dvoid *desc;
ub4 mode;
{
```

```
text            *subname;
ub4             size;
ship_data       *ship_stats = (ship_data *)ctx;
text           *queue;
text           *consumer;
OCIRaw         *msgid;
ub4            priority;
OCIAQMsgProperties *msgprop;

OCIAttrGet((dvoid *)subscrhp, OCI_HTYPE_SUBSCRIPTION,
           (dvoid *)&subname, &size,
           OCI_ATTR_SUBSCR_NAME, errhp);

/* AQ 記述子から属性を取り出します。キュー名は、次のとおりです。*/
OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&queue, &size,
           OCI_ATTR_QUEUE_NAME, errhp);

/* コンシューマ名 */
OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&consumer, &size,
           OCI_ATTR_CONSUMER_NAME, errhp);

/* メッセージ ID */
OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgid, &size,
           OCI_ATTR_NFY_MSGID, errhp);

/* メッセージ・プロパティ */
OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgprop, &size,
           OCI_ATTR_MSG_PROP, errhp);

/* メッセージ・プロパティから優先順位を取得します。*/
checkerr(errhp, OCIAttrGet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES,
                           (dvoid *)&priority, 0,
                           OCI_ATTR_PRIORITY, errhp));

switch (priority)
{
case 1: ship_stats->fedex++;
        break;
case 2: ship_stats->air++;
        break;
case 3: ship_stats->ground++;
        break;
default:
        printf(" Error priority %d", priority);
}
```

```

    }
}

int main(argc, argv)
int argc;
char *argv[];
{
    OCISession *authp = (OCISession *) 0;
    OCISubscription *subscrhp[8];
    ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;
    ship_data ctx = {0,0,0};
    ub4 sleep_time = 0;

    printf("Initializing OCI Process\n");

    /* OCI_EVENTS フラグ・セットで OCI 環境を初期化します。*/
    (void) OCIInitialize((ub4) OCI_EVENTS|OCI_OBJECT, (dvoid *)0,
                        (dvoid * (*)(dvoid *, size_t)) 0,
                        (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                        (void (*)(dvoid *, dvoid *)) 0 );

    printf("Initialization successful\n");

    printf("Initializing OCI Env\n");
    (void) OCIEnvInit( (OCIEnv **) &envhp, OCI_DEFAULT, (size_t) 0, (dvoid **) 0 );
    printf("Initialization successful\n");

    checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp,
OCI_HTYPE_ERROR,
                        (size_t) 0, (dvoid **) 0));

    checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp,
OCI_HTYPE_SERVER,
                        (size_t) 0, (dvoid **) 0));

    checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp,
OCI_HTYPE_SVCCTX,
                        (size_t) 0, (dvoid **) 0));

    printf("connecting to server\n");
    checkerr(errhp, OCIServerAttach( srvhp, errhp, (text *) "inst1_alias",
                        strlen("inst1_alias"), (ub4) OCI_DEFAULT));

```

```
printf("connect successful\n");

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。 */
checkerr(errhp, OCIAttrSet((dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid *)svrhp,
                          (ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp));

checkerr(errhp, OCIHandleAlloc((dvoid *) envhp, (dvoid **)&authp,
                              (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0));

/* セッション・ハンドルにユーザー名およびパスワードを設定します。 */
checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                          (dvoid *) username, (ub4) strlen((char *)username),
                          (ub4) OCI_ATTR_USERNAME, errhp));

checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                          (dvoid *) password, (ub4) strlen((char *)password),
                          (ub4) OCI_ATTR_PASSWORD, errhp));

/* セッションを開始します。 */
checkerr(errhp, OCISessionBegin ( svchp, errhp, authp, OCI_CRED_RDBMS,
                              (ub4) OCI_DEFAULT));

(void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
                (dvoid *) authp, (ub4) 0,
                (ub4) OCI_ATTR_SESSION, errhp);

/* 通知を登録します。 */
printf("allocating subscription handle\n");
subscrhp[0] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[0],
                    (ub4) OCI_HTYPE_SUBSCRIPTION,
                    (size_t) 0, (dvoid **) 0);

printf("setting subscription name\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) "WS.WS_BOOKEDORDERS_QUEUE:BOOKED_ORDERS",
                (ub4) strlen("WS.WS_BOOKEDORDERS_QUEUE:BOOKED_ORDERS"),
                (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

printf("setting subscription callback\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) notifyCB, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);
```

```

(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *)&ctx, (ub4) sizeof(ctx),
                  (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

printf("setting subscription namespace\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) &namespace, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

printf("Registering \n");
checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 1, errhp,
                                         OCI_DEFAULT));

sleep_time = (ub4)atoi(argv[1]);
printf ("waiting for %d s", sleep_time);
sleep(sleep_time);

printf("Exiting");
exit(0);
}

void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    sb4 errcode = 0;

    switch (status)
    {
        case OCI_SUCCESS:
            break;
        case OCI_SUCCESS_WITH_INFO:
            (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
            break;
        case OCI_NEED_DATA:
            (void) printf("Error - OCI_NEED_DATA\n");
            break;
        case OCI_NO_DATA:
            (void) printf("Error - OCI_NODATA\n");
            break;
        case OCI_ERROR:
            (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL, &errcode,
                               errbuf, (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
            (void) printf("Error - %.*s\n", 512, errbuf);
    }
}

```

```
        break;
    case OCI_INVALID_HANDLE:
        (void) printf("Error - OCI_INVALID_HANDLE\n");
        break;
    case OCI_STILL_EXECUTING:
        (void) printf("Error - OCI_STILL_EXECUTE\n");
        break;
    case OCI_CONTINUE:
        (void) printf("Error - OCI_CONTINUE\n");
        break;
    default:
        break;
    }
}
```



## 遅延間隔をおいての再試行

キューからメッセージをデキューするトランザクションが失敗した場合、メッセージ削除が正常に行われなかったと考えられます。AQ は、メッセージ削除に失敗した数をメッセージ履歴に記録します。アプリケーションは、キュー・テーブル・ビューの `RETRY_COUNT` 列を問い合わせ、メッセージに対する試行の失敗回数を参照できます。さらに、AQ では、アプリケーションがキュー内のメッセージに対する再試行の最大回数を、キュー・レベルで指定できます。メッセージ削除がこの数より多く失敗した場合、メッセージは例外キューに移動されるか、またはアプリケーションで使用できなくなります。

### 再試行遅延

メッセージを受信するトランザクションが異常終了した場合、不適切な条件が原因であると考えられます。AQ では、事前に指定された間隔で、不適切なメッセージを隠すことができます。再試行の最大回数とともに再試行遅延を指定できます。つまり、試行が失敗したメッセージは、再試行遅延間隔後に、デキュー用にキューで参照できます。それまでは、`WAITING` 状態になります。AQ バックグラウンド・プロセスである `タイム・マネージャ` が再試行遅延を施行します。再試行の最大回数のデフォルトの値は 5 であり、再試行遅延のデフォルト値は 0（ゼロ）です。再試行の最大回数および再試行遅延は、8.0 互換の複数コンシューマ・キューでは使用できないことに注意してください。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
/* 遅延を 1 日に設定したエンキューを行うパッケージを作成します。*/
CONNECT BOLADM/BOLADM
>
/* キューの再試行の最大回数は 4 で、再試行の遅延は 12 時間です。*/
execute dbms_aqadm.alter_queue(queue_name = 'WS.WS_BOOKED_ORDERS_QUE',
                               max_retries = 4,
                               retry_delay = 3600*12);
>
/* booked_order_queue で有効な次の注文を処理します。*/
CREATE OR REPLACE PROCEDURE process_next_order()
AS
    dqgopt                                dbms_aq.dequeue_options_t;
    msgprop                                dbms_aq.message_properties_t;
    deq_msgid                              RAW(16);
    book                                  BOLADM.book_typ;
    item                                  BOLADM.orderitem_typ;
    BOLADM.order_typ                      order;
BEGIN
>
```

```

        dqgopt.dequeue_option := DBMS_AQ.FIRST_MESSAGE;
        dbms_aq.dequeue('WS.WS_BOOKED_ORDERS_QUEUE', dqgopt, msgprop, order,
        deq_msgid
    );
>
    /* 簡略化のため、単品の注文であると想定しています。*/
    item = order.items(1);
    book = the_orders.item;
>
    /* search_inventory が書籍の在庫を検索すると想定しています。*/
    /* 書庫にその書籍が見つからない場合は、トランザクションは強制終了されます。*/
    IF (search_inventory(book) != TRUE)
        rollback;
    ELSE
        process_order(order);
    END IF;
>
END;
/

```

## Visual Basic (OO4O) : サンプル・コード

この機能については、データベースの dbexecutesql インタフェースを使用します。

## Java (JDBC) : サンプル・コード

```

public static void setup_queue(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueue             bookedorders_q;
    AQQueueProperty     q_prop;

    try
    {
        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);

        bookedorders_q = aq_sess.getQueue("WS", "WS_bookedorders_que");

        /* キューの再試行の最大回数を 4、再試行の遅延を 12 時間に変更します。*/
        q_prop = new AQQueueProperty();
        q_prop.setMaxRetries(4);

        q_prop.setRetryInterval(3600*12); // specified in seconds
    }
    catch (Exception e)
    {
        // 例外処理
    }
}

```

```
        bookedorders_q.alterQueue(q_prop);

    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
    }
}

public static void process_next_order(Connection db_conn)
{
    AQSession      aq_sess;
    Order           deq_order;
    OrderItem       order_item;
    Book            book;
    AQDequeueOption deq_option;
    AQMessageProperty msg_prop;
    AQQueue         bookedorders_q;
    AQMessage        message;
    AQObjectPayload obj_payload;

    try
    {
        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);

        deq_option = new AQDequeueOption();

        deq_option.setNavigationMode(AQDequeueOption.NAVIGATION_FIRST_MESSAGE);

        bookedorders_q = aq_sess.getQueue("WS", "WS_bookedorders_que");

        /* メッセージをデキューします。*/
        message = bookedorders_q.dequeue(deq_option, Order.getFactory());

        obj_payload = message.getObjectPayload();

        deq_order = (Order) (obj_payload.getPayloadData());

        /* 簡略化のため、単品の注文であると想定しています。*/
        order_item = deq_order.getItems().getElement(0);
    }
}
```

```
book = order_item.getItem();

/* search_inventory が書籍の在庫を検索すると想定しています。
 * 書庫にその書籍が見つからない場合は、トランザクションは強制終了されます。
 */
if (search_inventory(book) != true)
    db_conn.rollback();
else
    process_order(deq_order);

}
catch (AQException aqex)
{
    System.out.println("Exception-1: " + aqex);
}
catch (Exception ex)
{
    System.out.println("Exception-2: " + ex);
}
}
```

## 例外処理

AQ は、EXCEPTION\_QUEUES、EXPIRATION、MAX\_RETRIES および RETRY\_DELAY の 4 つの統合化メカニズムによって、アプリケーションの例外処理をサポートします。

exception\_queue (例外キュー) は、期限切れまたは処理できないすべてのメッセージのリポジトリになります。アプリケーションから例外キューには直接エンキューできません。また、複数コンシューマの例外キューに、サブスクライバを対応付けることはできません。ただし、期限切れまたは処理できないメッセージを処理するアプリケーションは、例外キューからデキューできます。複数コンシューマ・キューのメッセージを受け入れるために作成された例外キューは、それ自体が複数コンシューマ・キューである必要があります。他のキューと同様に、例外キューも DBMS\_AQADM.START\_QUEUE プロシージャを使用してデキューする必要があります。例外キューをエンキュー可能に設定しようとすると、Oracle エラーになります。

期限切れになったメッセージは、例外キューに移動されます。複数コンシューマ・キューのメッセージを受け入れる例外キューも、複数コンシューマ・キューである必要があります。複数コンシューマ・キューから移動された期限切れメッセージは、受信者を指定してデキューできません。ただし、デキュー・オプションのコンシューマ名に NULL を指定することによって、REMOVE モードで 1 回のみデキューできます。したがって、デキューの観点からすると、複数コンシューマの例外キューは単一コンシューマ・キューのように動作します。これは、それぞれの期限切れメッセージは、NULL コンシューマ名を使用して 1 回しかデキューできないためです。メッセージ ID を指定しても、例外キューからメッセージをデキューできます。複数コンシューマ例外キューを作成したキュー・テーブルが、互換パラメータを使用せずに作成された場合、または互換パラメータを 8.0 に設定して作成された場合は、期限切れメッセージはメッセージ ID を指定する方法でしかデキューできないことに注意してください。

例外キューは、エンキュー時に指定できるメッセージ・プロパティです (11-10 ページの「[メッセージのエンキュー \(メッセージ・プロパティの指定\)](#)」を参照)。PL/SQL では、DBMS\_AQ.MESSAGE\_PROPERTIES\_T レコードの exception\_queue 属性を使用して、例外キューを指定できます。OCI では、OCISetAttr プロシージャを使用して、OCIAQMsgProperties 記述子の OCI\_ATTR\_EXCEPTION\_QUEUE 属性を設定します。

例外キューが指定されていないと、デフォルトの例外キューが使用されます。キューがキュー・テーブル (たとえば QTAB) の中に作成されている場合、デフォルトの例外キューを AQ\$\_QTAB\_E といいます。デフォルトの例外キューは、キュー・テーブルが作成されるときに自動的に作成されます。メッセージは、次の条件が成立するときに AQ によって例外キューに移されます。

- そのメッセージが、指定された期限内にデキューされません。複数の受信者を指定したメッセージの場合、指定されていながら指定された期限切れまでにそのメッセージをデキューできない受信者が1つでもあると、例外キューに移動されます。デフォルトの期限切れは `DBMS_AQ.NEVER` で、そのメッセージは期限切れになりません。
- そのメッセージは正常にデキューされているが、そのメッセージの処理中にエラーが発生したために、メッセージをデキューするアプリケーションがトランザクションのロールバックを選択しました。この場合、メッセージはキューに戻され、同一キューからのデキューを待機しているアプリケーションならどれでも使用可能になります。アプリケーションがトランザクション全体にわたって、またはデキュー以前のセーブポイントまでロールバックしたときは、デキュー処理がロールバックまたは取り消されたとみなされます。メッセージがデキューされてロールバックされることが再試行制限の指定を超えて繰り返されると、そのメッセージは例外キューに移されます。

複数の受信者を指定したメッセージの場合、各メッセージが受信者ごとにそれぞれの再試行回数を保持しています。すべての受信者の再試行回数が再試行制限の指定を超えたときのみ、そのメッセージは例外キューに移動されます。単一コンシューマ・キューと 8.1 互換の複数コンシューマ・キューの場合、デフォルトの再試行制限は5回です。8.0 互換の複数コンシューマ・キューでは、再試行制限はサポートされていません。

- クライアントによって実行された文に含まれているデキューは成功したが、文自体は後で例外処理のために取り消されました。これがどのような場合かを理解するために、`DBMS_AQ.DEQUEUE` コールを含む PL/SQL プロシージャを考えてみます。デキュー・プロシージャが成功しても、PL/SQL プロシージャが例外を呼び出した場合、AQ はデキュー・プロシージャから戻されたメッセージの `RETRY_COUNT` を増分します。
- クライアント・プログラムはメッセージのデキューに成功したが、トランザクションをコミットする前に終了しました。

8.1 互換の複数コンシューマ・キューを指定したメッセージは、一度例外キューに移動されると、指定された受信者にはデキューできません。ただし、`REMOVE` または `BROWSE` モードで、デキュー・オプションのコンシューマ名に `NULL` を指定することによって、1 回のみデキューできます。メッセージ ID を指定してデキューすることもできます。

単一コンシューマ・キューまたは 8.0 互換の複数コンシューマ・キューを指定したメッセージが例外キューに移された場合は、メッセージ ID を指定してデキューできます。

ユーザーは、キューに RETRY\_DELAY を対応付けることができます。このパラメータのデフォルト値は 0（ゼロ）で、RETRY\_COUNT が増分されるとすぐにそのメッセージがデキューできるようになるという意味です。それ以外の値の場合、そのメッセージは RETRY\_DELAY 秒間は使用できません。RETRY\_DELAY 秒が過ぎると、キュー・モニターがそのメッセージに READY マークを付けます。

## サンプル・シナリオ

BooksOnLine アプリケーションでは、各地域のビジネス・ルールによって、すぐに応じられない注文は受注残キューに移動されます。受注残アプリケーションは毎日 1 回その注文を出荷しようとします。5 日以内に出荷できない場合、その注文は例外キューに移動されて特別に処理されます。AQ の再試行および例外処理機能を使用して、この処理を実装できます。

次の例では、キューを作成するときに、どのようにして再試行回数の最大値および再試行の遅延間隔を設定するかを示します。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

/\* 最大 5 回の再試行および各再試行の間に 1 日の遅延が可能な、西部地区の受注残キューを作成する例です。\*/

```
CONNECT BOLADM/BOLADM
BEGIN
    dbms_aqadm.create_queue (
        queue_name          => 'WS.WS_backorders_que',
        queue_table          => 'WS.WS_orders_mqtab',
        max_retries          => 5,
        retry_delay          => 60*60*24);
END;
/
```

/\* 西部地区の受注残キューに対して例外キューを作成します。\*/

```
CONNECT BOLADM/BOLADM
BEGIN
    dbms_aqadm.create_queue (
        queue_name          => 'WS.WS_backorders_excpt_que',
        queue_table          => 'WS.WS_orders_mqtab',
        queue_type           => DBMS_AQADM.EXCEPTION_QUEUE);
end;
/
```

/\* WS\_backorders\_que にメッセージをエンキューし、そのメッセージの例外キューとして WS\_backorders\_excpt\_que を指定します。\*/

```
CONNECT BOLADM/BOLADM
```

```
CREATE OR REPLACE PROCEDURE enqueue_WS_unfilled_order(backorder order_typ)
AS
    back_order_queue_name    varchar2(62);
    enqopt                   dbms_aq.enqueue_options_t;
    msgprop                   dbms_aq.message_properties_t;
    enq_msgid                 raw(16);
BEGIN

    /* このメッセージの受注残キュー名を設定します。*/
    back_order_queue_name := 'WS.WS_backorders_que';

    /* このメッセージの例外キュー名を設定します。 */
    msgprop.exception_queue := 'WS.WS_backorders_excpt_que';

    dbms_aq.enqueue(back_order_queue_name, enqopt, msgprop,
                    backorder, enq_msgid);

END;
/
```

## Visual Basic (OO4O) : サンプル・コード

例外キューは、メッセージをエンキューするときに提供されるメッセージ・プロパティです。このプロパティが設定されていない場合、キューのデフォルト例外キューがすべてのエラー条件に対して使用されます。

```
set oraaq = OraDatabase.CreateAQ("CBADM.deferbilling_que")
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
Set OraOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")
OraMsg = OraOrder
    OraMsg.delay = 15*60*60*24
    OraMsg.ExceptionQueue = "WS.WS_backorders_que"
    'Fill up the order values
    OraMsg = OraOrder 'OraOrder contains the order details
    Msgid = OraAq.enqueue
```

## Java (JDBC) : サンプル・コード

```
public static void createBackOrderQueues(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueue             backorders_q;
    AQQueue             backorders_excp_q;
    AQQueueProperty     q_prop;
    AQQueueProperty     q_prop2;
```



```

AQueueTable      mq_table;

try
{
    /* AQセッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    mq_table = aq_sess.getQueueTable("WS", "WS_orders_mqtab");

    /* 最大5回の再試行および各再試行の間に1日の遅延が可能な、西部地区の受注残キューを
    作成します。*/

    q_prop = new AQueueProperty();
    q_prop.setMaxRetries(5);
    q_prop.setRetryInterval(60*24*24);

    backorders_q = aq_sess.createQueue(mq_table, "WS_backorders_que",
                                       q_prop);

    backorders_q.start(true, true);

    /* 西部地区の受注残キューに対して例外キューを作成します。*/
    q_prop2 = new AQueueProperty();
    q_prop2.setQueueType(AQueueProperty.EXCEPTION_QUEUE);

    backorders_excp_q = aq_sess.createQueue(mq_table,
                                             "WS_backorders_excpt_que", q_prop2);
}
catch (Exception ex)
{
    System.out.println("Exception " + ex);
}

}

/* WS_backorders_queにメッセージをエンキューし、そのメッセージの例外キューとして
WS_backorders_excpt_queを指定します。*/
public static void enqueue_WS_unfilled_order(Connection db_conn,
                                             Order back_order)
{
    AQSession      aq_sess;
    AQueue          back_order_q;

```

```
AQEnqueueOption    enq_option;
AQMessageProperty  m_property;
AQMessage           message;
AQObjectPayload    obj_payload;
byte[]             enq_msg_id;

try
{
    /* AQ セッションを作成します。*/
    aq_sess = AQDriverManager.createAQSession(db_conn);

    back_order_q = aq_sess.getQueue("WS", "WS_backorders_que");

    message = back_order_q.createMessage();

    /* このメッセージの例外キュー名を設定します。*/
    m_property = message.getMessageProperty();

    m_property.setExceptionQueue("WS.WS_backorders_excpt_que");

    obj_payload = message.getObjectPayload();
    obj_payload.setPayloadData(back_order);

    enq_option = new AQEnqueueOption();

    /* メッセージをエンキューします。*/
    enq_msg_id = back_order_q.enqueue(enq_option, message);

    db_conn.commit();
}
catch (Exception ex)
{
    System.out.println("Exception: " + ex);
}
}
```

## ルールベースのサブスクリプション

メッセージは、そのメッセージ・プロパティまたは内容に基づいて、様々な受信者に送られます。ユーザーは、与えられたキューに対して、特定の条件に合うメッセージ受信の関心を指定するルールベースのサブスクリプションを定義します。

ルールは、評価の結果 TRUE または FALSE となるブール式です。SQL 問合せの WHERE 句の構文と同様、ルールはメッセージ・プロパティまたはメッセージ内容を示す属性によって表現されます。このようなサブスクライバ・ルールが、着信メッセージに対して評価され、一致したルールを使用してメッセージ受信者が判断されます。この機能によって、メッセージの内容ベースのサブスクリプションおよびルーティングがサポートされます。

### サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、内容ベースのサブスクリプションおよびルーティングを利用するパブリッシュ / サブスクライブ・パラダイムを実装するために、どのようにルールベースのサブスクリプションが使用されているかを説明します。注文入力アプリケーションと各出荷処理アプリケーションの相互作用が、次のようにモデル化されています。

- 西部向け出荷は、アメリカ合衆国の西部地区の注文を扱います。
- 東部向け出荷は、アメリカ合衆国の東部地区の注文を扱います。
- 海外向け出荷は、合衆国以外からの注文を扱います。
- 東部向け出荷は、アメリカ合衆国全域の至急注文を扱います。

各出荷処理アプリケーションは、OE の入力済注文キューにサブスクライブします。次に示すルールベースのサブスクリプションは、注文入力アプリケーションから各出荷処理アプリケーションに入力済注文をルーティングするために、OE のユーザーによって定義されます。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
CONNECT OE/OE;
```

西部向け出荷は、エージェント・アドレス（メッセージが配信される宛先キュー）として WS 入力済注文キューを設定した「West\_Shipping」といわれるエージェントを定義します。このエージェントは OE の入力済注文キューに、orderregion および ordertype 属性を指定するルールを使用してサブスクライブします。

```
/* 西部向け出荷のルールベースのサブスクライバを追加します。 -
   西部向け出荷は、アメリカ合衆国の西部地区の注文を扱います。
   西部地区の至急注文は、東部向け出荷が扱います。 */
```

```
DECLARE
    subscriber      aq$_agent;
```

```

BEGIN
    subscriber := aq$_agent('West_Shipping', 'WS.WS_bookedorders_que', null);
    dbms_aqadm.add_subscriber(
        queue_name => 'OE.OE_bookedorders_que',
        subscriber => subscriber,
        rule       => 'tab.user_data.orderregion =
                    ''WESTERN'' AND tab.user_data.ordertype != ''RUSH''');
END;
/

```

東部向け出荷は、エージェント・アドレス（メッセージが配信される宛先キュー）として ES 入力済注文キューを設定した East\_Shipping といわれるエージェントを定義します。このエージェントは、orderregion、ordertype および customer 属性に指定されたルールを使用して OE の入力済注文キューを定義します。

/\* 東部向け出荷のルールベースのサブスクライバを追加します。 -  
 東部向け出荷は、アメリカ合衆国の東部地区の注文を扱います。  
 また、アメリカ合衆国の至急注文も扱います。\*/

```

DECLARE
    subscriber    aq$_agent;
BEGIN
    subscriber := aq$_agent('East_Shipping', 'ES.ES_bookedorders_que', null);
    dbms_aqadm.add_subscriber(
        queue_name => 'OE.OE_bookedorders_que',
        subscriber => subscriber,
        rule       => 'tab.user_data.orderregion = ''EASTERN'' OR
                    (tab.user_data.ordertype = ''RUSH'' AND
                     tab.user_data.customer.country = ''USA'') ');
END;
/

```

海外向け出荷は、エージェント・アドレス（メッセージが配信される宛先キュー）として OS 入力済注文キューを設定した Overseas\_Shipping といわれるエージェントを定義します。このエージェントは、orderregion 属性に指定したルールを使用して OE の入力済注文キューを定義します。

/\* 海外向け出荷のルールベースのサブスクライバを追加します。 -  
 海外向け出荷は、アメリカ合衆国以外からの注文を扱います。\*/

```

DECLARE
    subscriber    aq$_agent;
BEGIN
    subscriber := aq$_agent('Overseas_Shipping', 'OS.OS_bookedorders_que',
null);
    dbms_aqadm.add_subscriber(
        queue_name => 'OE.OE_bookedorders_que',
        subscriber => subscriber,
        rule       => 'tab.user_data.orderregion = ''INTERNATIONAL''');

```

```
END;
/
```

## Visual Basic (OO4O) : サンプル・コード

この機能は、現在使用できません。

## Java (JDBC) : サンプル・コード

```
public static void addRuleBasedSubscribers(Connection db_conn)
{

    AQSession      aq_sess;
    AQQueue         bookedorders_q;
    String          rule;
    AQAgent         agt1, agt2, agt3;

    try
    {
        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);

        bookedorders_q = aq_sess.getQueue("OE", "OE_booked_orders_que");

        /* 西部向け出荷のルールベースのサブスクライバを追加します。 -
        西部向け出荷は、アメリカ合衆国の西部地区の注文を扱います。
        西部地区の至急注文は、東部向け出荷が扱います。 */

        agt1 = new AQAgent("West_Shipping", "WS.WS_bookedorders_que");

        rule = "tab.user_data.orderregion = 'WESTERN' AND " +
               "tab.user_data.ordertype != 'RUSH'";

        bookedorders_q.addSubscriber(agt1, rule);

        /* 東部向け出荷のルールベースのサブスクライバを追加します。 -
        東部向け出荷は、アメリカ合衆国の東部地区の注文を扱います。
        また、アメリカ合衆国の至急注文も扱います。*/

        agt2 = new AQAgent("East_Shipping", "ES.ES_bookedorders_que");
        rule = "tab.user_data.orderregion = 'EASTERN' OR " +
               "(tab.user_data.ordertype = 'RUSH' AND " +
               "tab.user_data.customer.country = 'USA')";
```

```
        bookedorders_q.addSubscriber(agt2, rule);

        /* 海外向け出荷のルールベースのサブスクリバを追加します。 -
        海外向け出荷は、アメリカ合衆国以外からの注文を扱います。 */

        agt3 = new AQAgent("Overseas_Shipping", "OS.OS_bookedorders_queue");
        rule = "tab.user_data.orderregion = 'INTERNATIONAL'";

        bookedorders_q.addSubscriber(agt3, rule);
    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
    }
}
```

## Listen 機能

Oracle8i では、AQ は 1 つの listen コールで複数のキューのメッセージを監視する機能を持っています。アプリケーションは、listen を使用して複数のサブスクリプションに対するメッセージを待機できます。また、ゲートウェイ・アプリケーションでも、この機能を使用して複数のキューを監視できます。listen コールが成功して戻った場合は、メッセージを取り出すためにデキューする必要があります（11-23 ページの「**1 個（複数個）のキューのリスニング**」を参照）。

listen コールがない場合、一連のキューからデキューするアプリケーションは、それらのキューにメッセージがあるかどうかを判断するために継続的にポーリングする必要があります。または、各キューに対するデキュー処理が互いに独立するようにアプリケーションを設計することもできます。ただし、長期間そのキューに何もメッセージがない場合は、これらの方法によって、許容できないオーバーヘッドが引き起こされます。listen コールはそのようなアプリケーションに適しています。

エージェント・リストの複数のエージェントに向けたメッセージがいくつかある場合、listen はメッセージの宛先になっている最初のエージェントとともに戻ります。その意味で、listen のキュー監視は公正ではありません。アプリケーション設計者はこの仕様を認識してこのコールを使用する必要があります。あるエージェントが他のエージェントに対してメッセージの欠乏状態を引き起こさないように、アプリケーションはエージェント・リストのエージェント順序を変更できます。

## サンプル・シナリオ

BooksOnLine シナリオの顧客サービス・コンポーネントでは、異なるデータベースから顧客サービス・キューに届いたメッセージにはその状態が示されます。顧客サービス・アプリケーションはそのキューを監視し、顧客の注文に関するメッセージがあるときはいつでも order\_status\_table の注文状態を更新します。アプリケーションは listen コールを使用して様々なキューを監視します。それらのキューのどれかにメッセージがあるときはいつでも、メッセージをデキューし、それに応じて注文状態を更新します。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

CODE (in tkaqdocd.sql)

```
/* order_status_table の注文状態を更新します。*/
CREATE OR REPLACE PROCEDURE update_status(
                                new_status      IN VARCHAR2,
                                order_msg        IN BOLADM.ORDER_TYP)
IS
  old_status  VARCHAR2(30);
  dummy      NUMBER;
BEGIN
```

```
BEGIN
    /* 表から古い状態を問い合わせます。*/
    SELECT st.status INTO old_status FROM order_status_table st
        WHERE st.customer_order.orderno = order_msg.orderno;

    /* 状態は、'BOOKED_ORDER'、'SHIPPED_ORDER'、'BACK_ORDER'、'BILLED_ORDER' のいずれか
    です。*/

    IF new_status = 'SHIPPED_ORDER' THEN
        IF old_status = 'BILLED_ORDER' THEN
            return;          /* 以前の状態についてのメッセージ */
        END IF;
    ELSIF new_status = 'BACK_ORDER' THEN
        IF old_status = 'SHIPPED_ORDER' OR old_status = 'BILLED_ORDER' THEN
            return;          /* 以前の状態についてのメッセージ */
        END IF;
    END IF;

    /* 注文状態を更新します。*/
    UPDATE order_status_table st
        SET st.customer_order = order_msg, st.status = new_status;

    COMMIT;

    EXCEPTION
    WHEN OTHERS THEN        /* データが見つからない場合は変更します。*/
        /* まず、その注文用に更新します。*/
        INSERT INTO order_status_table(customer_order, status)
            VALUES (order_msg, new_status);
        COMMIT;

    END;
END;
/

/* 'QUEUE' から 'CONSUMER' にメッセージをデキューします。*/
CREATE OR REPLACE PROCEDURE DEQUEUE_MESSAGE (
    queue      IN   VARCHAR2,
    consumer   IN   VARCHAR2,
    message    OUT  BOLADM.order_typ)
IS

    dopt       dbms_aq.dequeue_options_t;
    mprop      dbms_aq.message_properties_t;
```



```

deq_msgid          RAW(16);
BEGIN
    dopt.dequeue_mode := dbms_aq.REMOVE;
    dopt.navigation := dbms_aq.FIRST_MESSAGE;
    dopt.consumer_name := consumer;

    dbms_aq.dequeue(
        queue_name => queue,
        dequeue_options => dopt,
        message_properties => mprop,
        payload => message,
        msgid => deq_msgid);

    commit;
END;
/

/* 顧客サービス・データベースのキューを 'time' 秒ごとに監視します。*/
CREATE OR REPLACE PROCEDURE MONITOR_STATUS_QUEUE(time IN NUMBER)
IS
    agent_w_message    aq$_agent;
    agent_list          dbms_aq.agent_list_t;
    wait_time           INTEGER := 120;
    no_message          EXCEPTION;
    pragma EXCEPTION_INIT(no_message, -25254);
    order_msg           boladm.order_typ;
    new_status          VARCHAR2(30);
    monitor             BOOLEAN := TRUE;
    begin_time          NUMBER;
    end_time            NUMBER;
BEGIN

    begin_time := dbms_utility.get_time;
    WHILE (monitor)
    LOOP
        BEGIN

            /* エージェント・リストを構成します。*/
            agent_list(1) := aq$_agent('BILLED_ORDER', 'CS_billedorders_que', NULL);
            agent_list(1) := aq$_agent('SHIPPED_ORDER', 'CS_shippedorders_que',
NULL);
            agent_list(2) := aq$_agent('BACK_ORDER', 'CS_backorders_que', NULL);
            agent_list(3) := aq$_agent('Booked_ORDER', 'CS_bookedorders_que', NULL);

            /* 注文状態メッセージを待ちます。*/
            dbms_aq.listen(agent_list, wait_time, agent_w_message);

```

```

        dbms_output.put_line('Agent' || agent_w_message.name || ' Address ' ||
agent_w_message.address);
    /* キューからメッセージをデキューします。*/
    dequeue_message(agent_w_message.address, agent_w_message.name, order_msg);

    /* メッセージの種類によって注文状態を更新します。エージェント名には新しい状態が含まれます。*/
    update_status(agent_w_message.name, order_msg);

    /* 終了します。*/
    end_time := dbms_utility.get_time;
    IF (end_time - begin_time > time) THEN
        EXIT;
    END IF;

EXCEPTION
WHEN no_message THEN
    dbms_output.put_line('No messages in the past 2 minutes');
    end_time := dbms_utility.get_time;
    /* 終了します。*/
    IF (end_time - begin_time > time) THEN
        EXIT;
    END IF;
END;

END LOOP;
END;
/

```

## Visual Basic (OO4O) : サンプル・コード

この機能は、現在使用できません。

## Java (JDBC) : サンプル・コード

この機能は、Java ではサポートされません。

## 伝播機能

- 伝播
- 伝播スケジュール
- サンプル・シナリオ
- 拡張伝播スケジュール機能
- 伝播中の例外処理

## 伝播

この機能によって、同じデータベースまたは同じキューに接続しなくても、アプリケーション同士が互いに通信できます。メッセージは、ローカルまたはリモートに関係なく、ある Oracle AQ から別の Oracle AQ に伝播できます。伝播は、スナップショット（ジョブ・キュー）・バックグラウンド・プロセスによって実行されます。リモート・キューへの伝播は、データベース・リンクおよび Net 8 を使用して行われます。

伝播機能は次のように使用されます。まず、あるキューに、そのキューからメッセージが伝播される相手先のサブスクライバが 1 つ以上定義されます（8-35 ページの「[サブスクリプションおよび受信者リスト](#)」を参照）。次に、そのキューからメッセージを伝播する宛先ごとに、スケジュールが定義されます。エンキューされたメッセージは伝播されて、自動的に宛先キューでデキューできるようになります。

伝播を使用するためには、複数のジョブ・キュー・バックグラウンド・プロセスを実行中にする必要があります。その他に、伝播しない関連ジョブを扱うために必要な数のジョブ・キュー・バックグラウンド・プロセスが必要です。また、リモート伝播を実行する場合は、そのスケジュールのために指定されたデータベース・リンクが有効で、宛先キューにエンキューするための適切な権限を持っていることを確認する必要があります。伝播スケジュールを管理するための管理コマンドの詳細は、8-76 ページの「[非同期通知](#)」を参照してください。

伝播には、障害に対処するためのメカニズムもあります。たとえば、指定されたデータベース・リンクが無効な場合、リモート・データベースが使用不可な場合、またはリモート・キューにエンキューできない場合には、適切なエラー・メッセージが戻されます。

最後に、伝播されたメッセージおよびそのスケジュールに関する詳細な統計を取得する機能があります。この情報は、最高のパフォーマンスが得られるように適切にスケジュールを調整するために使用できます。伝播の障害対処 / エラー・レポート機能および伝播統計については、8-109 ページの「[拡張伝播スケジュール機能](#)」を参照してください。

## 伝播スケジュール

伝播スケジュールは、1組のソース（伝播元のキュー）および宛先キューに対して定義されます。あるキューに複数のメッセージがあり、いくつかのキューに伝播されることになっている場合、宛先キューごとにスケジュールを定義する必要があります。スケジュールは時間の枠を示し、メッセージはその枠内にソース・キューから伝播されます。この時間枠は、ネットワーク・トラフィック、ソース・データベースの負荷、接続先データベースの負荷などの多くの要因によって左右されます。したがって、スケジュールはその特定のソースおよび宛先に合せて作成する必要があります。スケジュールが作成されると、ジョブは自動的にジョブ・キューに発行され、伝播が処理されます。

伝播スケジュールリングのための管理コールによって、スケジュール管理に大きな柔軟性が得られます（9-65 ページの「[キューの伝播のスケジュール](#)」を参照）。あるスケジュールの存続時間または伝播枠パラメータによって、伝播が開始される時間枠が指定されます。存続時間が指定されない場合、時間枠は無制限の単一枠になります。枠を定期的に繰り返す必要がある場合、連続する枠の間の周期的間隔を定義する `next_time` 関数を使用して有限の存続時間を指定します。

スケジュールの待ち時間パラメータが関係するのは、あるキューに伝播する必要があるメッセージが1つもないときのみです。このパラメータは、キューを再チェックする時間間隔を指定します。待ち時間パラメータが施行された場合、ジョブ・キュー・プロセス用の `job_queue_interval` パラメータは `latency` パラメータより小さい必要がある点に注意してください。

あるキューに定義された伝播スケジュールは、そのキューの有効期間中いつでも変更または削除できます。さらに、（スケジュールを削除するかわりに）一時的に使用不可にするコール、および使用不可のスケジュールを使用可能にするコールがあります。メッセージがスケジュール内で伝播されているとき、そのスケジュールはアクティブです。すべての管理コールは、スケジュールがアクティブかどうかに関係なく実行されます。アクティブなスケジュールでは、コールが実行されるまでに数秒かかります。

### サンプル・シナリオ

BooksOnLine の例では、`OE_bookedorders_que` にあるメッセージが別の出荷サイトに伝播されます。スケジュールを指定し管理するために利用できる様々な管理コールを、次のサンプル・コードで示します。また、ソース・キューにメッセージをエンキューしたり、宛先サイトでメッセージをデキューするためのコールも示します。カタログ・ビュー `USER_QUEUE_SCHEDULES` によって、任意のスケジュールに関連するすべての情報が得られます（10-30 ページの「[ユーザー・スキーマの伝播スケジュールの選択](#)」を参照）。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
CONNECT OE/OE;

/* bookedorders_que から出荷へのスケジュールの伝播 */
EXECUTE dbms_aqadm.schedule_propagation( \
    queue_name      => 'OE.OE_bookedorders_que');

/* スケジュールが作成されているかどうかを確認します。 */
SELECT * FROM user_queue_schedules;

/* いくつかの注文を OE_bookedorders_que にエンキューします。 */
EXECUTE BOLADM.order_enq('My First   Book', 1, 1001, 'CA', 'USA', \
    'WESTERN', 'NORMAL');
EXECUTE BOLADM.order_enq('My Second  Book', 2, 1002, 'NY', 'USA', \
    'EASTERN', 'NORMAL');
EXECUTE BOLADM.order_enq('My Third   Book', 3, 1003, '',   'Canada', \
    'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Fourth  Book', 4, 1004, 'NV', 'USA', \
    'WESTERN', 'RUSH');
EXECUTE BOLADM.order_enq('My Fifth   Book', 5, 1005, 'MA', 'USA', \
    'EASTERN', 'RUSH');
EXECUTE BOLADM.order_enq('My Sixth   Book', 6, 1006, '',   'UK', \
    'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Seventh Book', 7, 1007, '',   'Canada', \
    'INTERNATIONAL', 'RUSH');
EXECUTE BOLADM.order_enq('My Eighth  Book', 8, 1008, '',   'Mexico', \
    'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Ninth   Book', 9, 1009, 'CA', 'USA', \
    'WESTERN', 'RUSH');
EXECUTE BOLADM.order_enq('My Tenth   Book', 8, 1010, '',   'UK', \
    'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Last    Book', 7, 1011, '',   'Mexico', \
    'INTERNATIONAL', 'NORMAL');

/* 伝播が行われるのを待ちます。 */
EXECUTE dbms_lock.sleep(100);

/* 出荷サイトに接続し、伝播されたメッセージを確認します。 */
CONNECT WS/WS;
set serveroutput on;

/* 西部向け出荷のすべての入力済注文をデキューします。 */
EXECUTE BOLADM.shipping_bookedorder_deq('West_Shipping', DBMS_AQ.REMOVE);
```

```
CONNECT ES/ES;
SET SERVEROUTPUT ON;

/* 東部向け出荷の残りすべての入力済注文（普通注文）をデキューします。*/
EXECUTE BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.REMOVE);

CONNECT OS/OS;
SET SERVEROUTPUT ON;

/* 海外向け出荷のアメリカ合衆国内のすべての注文をデキューします。*/
EXECUTE BOLADM.get_northamerican_orders('Overseas_Shipping');

/* 海外向け出荷の残りの入力済注文をデキューします。*/
EXECUTE BOLADM.shipping_bookedorder_deq('Overseas_Shipping', DBMS_AQ.REMOVE);

/* 入力済注文の伝播スケジュールを使用不可にします。*/
EXECUTE dbms_aqadm.disable_propagation_schedule( \
    queue_name => 'OE_bookedorders_que');

/* コールの効果が出るまでしばらく待ちます。*/
EXECUTE dbms_lock.sleep(30);

/* スケジュールが使用不可になっているかどうかを確認します。*/
SELECT schedule_disabled FROM user_queue_schedules;

/* 入力済注文に対して、枠の存続期間が300秒で、15分（900秒）ごとに伝播スケジュールが実行されるように変更します。*/
EXECUTE dbms_aqadm.alter_propagation_schedule( \
    queue_name      => 'OE_bookedorders_que', \
    duration         => 300, \
    next_time        => 'SYSDATE + 900/86400', \
    latency          => 25);

/* コールの効果が出るまでしばらく待ちます。*/
EXECUTE dbms_lock.sleep(30);

/* スケジュール・パラメータが変更されているかどうかを確認します。*/
SELECT next_time, latency, propagation_window FROM user_queue_schedules;

/* 入力済注文の伝播スケジュールを使用可能にします。*/
EXECUTE dbms_aqadm.enable_propagation_schedule( \
    queue_name      => 'OE_bookedorders_que');

/* コールの効果が出るまでしばらく待ちます。*/
EXECUTE dbms_lock.sleep(30);
```

```
/* スケジュールが使用可能になっているかどうかを確認します。*/  
SELECT schedule_disabled FROM user_queue_schedules;  
  
/* 入力済注文の伝播スケジュールを解除します。*/  
EXECUTE dbms_aqadm.unschedule_propagation(  \  
    queue_name      => 'OE.OE_bookedorders_que');  
  
/* コールの効果が出るまでしばらく待ちます。*/  
EXECUTE dbms_lock.sleep(30);  
  
/* スケジュールが削除されたかどうかを確認します。*/  
SELECT * FROM user_queue_schedules;
```

### Visual Basic (OO4O) : サンプル・コード

この機能は、現在使用できません。

### Java (JDBC) : サンプル・コード

今回のリリースでは、例は記載していません。



## LOB 属性を伴うメッセージの伝播

AQ を使用したラージ・オブジェクトの伝播には、2 通りの方法があります。

- RAW キューからの伝播。RAW キューでは、メッセージ・ペイロードはバイナリ・ラージ・オブジェクト (BLOB) として保存されます。これによって、PL/SQL インタフェースを使用したときには 32KB までのデータを格納でき、OCI を使用したときには、クライアントが同じ容量のデータを連続して割り当てることができます。この方法は、リリース 8.0.4 以降でサポートされています。
- LOB 属性を伴うオブジェクト・キューからの伝播。ユーザーは、Oracle の LOB 操作ルーチンを使用して、LOB を移入することも LOB から読み込むこともできます。LOB 属性は、BLOB または CLOB (NCLOB ではなく) です。属性が CLOB の場合、AQ はソース・キューと宛先キューの間で必要なすべてのキャラクタ・セット変換を自動的に実行します。この方法は、リリース 8.1.3 以降でサポートされています。

---

### 参照：

- LOB の詳細は、『Oracle8i アプリケーション開発者ガイド ラージ・オブジェクト』を参照してください。
- 

AQ では、ペイロードに BFILE または REF 属性があるオブジェクト・キューからの伝播はサポートされていないことに注意してください。

## サンプル・シナリオ

BooksOnLine アプリケーションでは、書籍注文とともに販売促進用の割引券を送ることができます。割引券を送るかどうかは、注文内容およびその他の顧客情報によって決定されます。その割引券のイメージは、いくつかのマルチメディア・データベースで生成され、LOB として保存されます。

注文情報が出荷倉庫に送られるときに、割引券の内容も倉庫に送られます。次に示したコードでは、LOB 型の割引券属性を含むように `order_ttyp` が拡張されています。このコードは、注文が発生したときに `OE_bookedorders_que` にエンキューされるメッセージに LOB 内容を挿入する方法を示しています。メッセージ・ペイロードは、最初に空の LOB によって組み立てられます。プレース・ホルダ (LOB ロケータ) 情報はキュー・テーブルから取得され、`DBMS_LOB.WRITE()` などの LOB 操作ルーチンと組み合わせて LOB 内容を充填するために使用されます。さらに、ペイロードの一部に LOB があるメッセージのエンキューおよびデキューに関する例も示します。

COMMIT は、LOB 内容が適切なイメージ・データによって充填されてから発行されます。伝播は、LOB 内容をそれ以外のメッセージ内容とともに自動的に移動させる処理をします。次のコードには、宛先キューで伝播されたメッセージの LOB 内容を読み込むデキューについても示します。LOB 内容は、バッファに読み込まれ、そこから割引券を印刷するためにプリンタに送られることもあります。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```

/* 割引券フィールド (lob フィールド) を含むように、order_typ 型を拡張します。*/
CREATE OR REPLACE TYPE order_typ AS OBJECT (
    orderno          NUMBER,
    status            VARCHAR2(30),
    ordertype         VARCHAR2(30),
    orderregion       VARCHAR2(30),
    customer          customer_typ,
    paymentmethod     VARCHAR2(30),
    items             orderitemlist_vartyp,
    total             NUMBER,
    coupon            BLOB);

/

/* lob_loc は BLOB 型の変数です。
   buffer は RAW 型の変数です。
   length は NUMBER 型の変数です。*/

/* 注文データを書き込み、order_enq() プロシージャを使用してエンキューを実行します。*/
dbms_aq.enqueue('OE.OE_bookedorders_queue', enqopt, msgprop,
    OE_enq_order_data, enq_msgid);

/* エンキューの後、キュー・テーブルの lob ロケータを取得します。*/
SELECT t.user_data.coupon INTO lob_loc
FROM   OE.OE_orders_pr_mqtab t
WHERE  t.msgid = enq_msgid;

/* 100 バイトのサンプル LOB を生成します。*/
buffer := hextoraw(rpad('FF',100,'FF'));

/* dbms_lob パッケージの LOB ルーチンを使用して、lob を充填します。*/
dbms_lob.write(lob_loc, 90, 1, buffer);

/* lob の内容が充填されてからコミットを発行します。*/
COMMIT;

/* 伝播が完了するまで待ちます。*/

```

```

/* 西部向け出荷倉庫でデキューを実行します。*/
dbms_aq.dequeue(
    queue_name      => qname,
    dequeue_options => dopt,
    message_properties => mprop,
    payload         => deq_order_data,
    msgid           => deq_msgid);

/* デキューの後、LOB ロケータを取得します。*/
lob_loc := deq_order_data.coupon;

/* LOB の長さを求めます。*/
length := dbms_lob.getlength(lob_loc);

/* LOB の内容をバッファに読み込みます。*/
dbms_lob.read(lob_loc, length, 1, buffer);

```

## Visual Basic (OO4O) : サンプル・コード

この機能は、現在使用できません。

## Java (JDBC) : サンプル・コード

今回のリリースでは、例は記載していません。

## 拡張伝播スケジュール機能

スケジュールに関する詳細情報は、伝播のために定義されたカタログ・ビューから取得できます。アクティブ・スケジュールに関する情報、たとえば、そのスケジュールを処理しているバックグラウンド・プロセス名、伝播を処理するセッションのSID（セッションのシリアル番号）、スケジュールを処理する Oracle インスタンス（OPS が使用されているときは関連があります）などの情報は、そのカタログ・ビューから取得できます。同じカタログ・ビューによって、先行して正常に実行されたスケジュール（最後に正常に伝播されたメッセージ）および次に実行されるスケジュールに関する情報も取得できます。

各スケジュールには伝播の詳細情報が保持されます。これには、スケジュールの中で伝播されたメッセージの合計数とバイトの合計数、伝播枠の中で伝播されたメッセージの最大数と平均数、伝播されたメッセージの平均サイズと平均時間が含まれます。このような統計は、キュー管理者が最も効率的なスケジュール調整に役立てることができるように設計されています。

伝播機能には、障害対処およびエラー・レポートが組み込まれています。たとえば、指定されたデータベース・リンクが無効な場合、リモート・データベースが使用できない場合、またはリモート・キューにエンキューできない場合、適切なエラー・メッセージがレポートされます。伝播は指数バックオフ・スキームを使用して、障害が発生したスケジュールからの伝播を再試行します。あるスケジュールが続けて障害を発生したときは、最初の再試行は 30 秒後、次の再試行は 60 秒後、3 回目の再試行は 120 秒後、というように続きます。再試行時間が現行の伝播枠の期限切れ時刻を超える場合は、次の再試行は、次の伝播枠の開始時刻に行われます。最大 16 回の再試行が行われた後、そのスケジュールは自動的に使用不可になります。障害のためにスケジュールが自動的に使用不可になると、関連情報がアラート・ログに書き込まれます。どの場合でも、スケジュールに障害が発生しているか、発生している場合は継続的な障害がいくつあるかということが、障害の原因および直前の障害の発生時刻を示すエラー・メッセージによってチェックできます。この情報を調べることで、キュー管理者は障害を回復し、スケジュールを使用可能にできます。再試行の間に伝播が成功したときは、障害の数は 0 (ゼロ) にリセットされます。

伝播機能には OPS サポートが組み込まれていますが、ユーザーおよびキュー管理者には完全に透過的です。伝播を処理するジョブは、キューが常駐しているキュー・テーブルの所有者と同じインスタンスに送られます。あるインスタンスに障害があってキューを保存しているキュー・テーブルが他のインスタンスに移行される場合は、伝播ジョブも必ず自動的に新しいインスタンスに移されます。これによって、インスタンス間の ping 操作は最小限に抑えられ、パフォーマンスが向上します。伝播は、同時スケジュールをいくつでも処理できるように設計されています。ジョブ・キュー・プロセスの数は、最大 36 に制限され、その中のいくつかは伝播以外の関連ジョブを処理するために使用されることに注意してください。このように、伝播にはマルチタスキングおよびロード・バランスのサポートが組み込まれています。伝播アルゴリズムは、複数スケジュールが単一スナップショット (ジョブ・キュー)・プロセスによって処理できるように設計されています。ジョブ・キュー・プロセスに対する伝播の負荷は、異なるソース・キューからのメッセージ到着割合に基づいて偏りが発生する場合があります。あるプロセスが数個のアクティブ・スケジュールによって過負荷になっている一方で、別のプロセスは受動的なスケジュールが多いために余力があるというとき、伝播はプロセス間で負荷が均等になるようにスケジュールを自動的に再分散します。

## サンプル・シナリオ

BooksOnLine の例では、OE\_bookedorders\_que は、そこに含まれるメッセージが別の出荷サイトに伝播されるため、ビジー・キューになります。エラー・チェックおよびスケジュール監視のための拡張伝播スケジュールリングをサポートしているコールを、次のサンプル・コードで示します。

## PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・コード

```
CONNECT OE/OE;

/* 平均を求めます。*/
select avg_time, avg_number, avg_size from user_queue_schedules;

/* 合計を求めます。*/
select total_time, total_number, total_bytes from user_queue_schedules;

/* 枠の最大数を求めます。*/
select max_number, max_bytes from user_queue_schedules;

/* 現在のスケジュールに関する詳細情報を取得します。*/
select process_name, session_id, instance, schedule_disabled
       from user_queue_schedules;

/* 最後に実行した日時、次に実行する日時を求めます。*/
select last_run_date, last_run_time, next_run_date, next_run_time
       from user_queue_schedules;

/* 最新のエラー情報を取得します（ある場合）。*/
select failures, last_error_msg, last_error_date, last_error_time
       from user_queue_schedules;
```

## Visual Basic (OO4O) : サンプル・コード

この機能は、現在使用できません。

## Java (JDBC) : サンプル・コード

今回のリリースでは、例は記載していません。

## 伝播中の例外処理

ネットワーク障害のようなシステム・エラーが発生した場合、AQ は指数バックオフ・アルゴリズムを使用してメッセージを伝播する試みを継続します。状況がアプリケーション・エラーを示しているとき、メッセージの伝播でエラーが発生した場合は、AQ はそのメッセージに UNDELIVERABLE というマークを付けます。

このようなエラーは、リモート・キューが存在しない場合、またはソース・キューおよびリモート・キューの型が一致しない場合に発生します。このような状況では、ユーザーは DBA\_SCHEDULES ビューを問い合わせ、特定の宛先向けの伝播中に発生した最新のエラーを調べます。\$ORACLE\_HOME/log ディレクトリのトレース・ファイルには、そのエラーに関する追加情報があります。

### サンプル・シナリオ

BooksOnLine の例では、東部向け出荷の ES\_bookedorders\_que が stop\_queue() コールを使用して故意に中止されます。少し時間が経つと、OE\_bookedorders\_que の伝播スケジュールには、リモート・キュー ES\_bookedorders\_que にはエンキューできないというエラーが表示されます。start\_queue() コールを使用して ES\_bookedorders\_que を開始すると、そのキューへの伝播が再開され、OE\_bookedorders\_que のスケジュールに関連したエラー・メッセージはなくなります。

### PL/SQL (DBMS\_AQ/ADM パッケージ) : サンプル・シナリオ

```
/* 東部向け出荷キューを故意に中止します。*/
connect BOLADM/BOLADM
EXECUTE dbms_aqadm.stop_queue(queue_name => 'ES.ES_bookedorders_que');

/* dba_queue_schedules にエラーが表示されるまで待ちます。*/
EXECUTE dbms_lock.sleep(100);

/* この問合せによって、エンキューできないという ORA-25207 エラーが戻されます。*/
SELECT qname, last_error_msg from dba_queue_schedules;

/* 東部向け出荷キューを開始します。*/
EXECUTE dbms_aqadm.start_queue(queue_name => 'ES.ES_bookedorders_que');

/* 東部向け出荷キューに対して伝播が再開されるまで待ちます。*/
EXECUTE dbms_lock.sleep(100);

/* 次の問合せは、伝播に関するエラーがないことを示します。*/
SELECT qname, last_error_msg from dba_queue_schedules;
```

**Visual Basic (OO4O) : サンプル・コード**

この機能は、データベースでは処理されません。

**Java (JDBC) : サンプル・コード**

今回のリリースでは、例は記載していません。





---

## 管理インタフェース

### ユースケース・モデル

この章では、Oracle アドバンスド・キューイングの管理インタフェースについて説明します。それぞれの操作（[キュー・テーブルの作成](#)など）を、その操作名ごとにユースケースの点から説明します。[表 9-1「ユースケース・モデル:管理インタフェース – 基本操作」](#)に、すべてのユースケースをリストします。

### ユースケース・モデルの図形概要

[図 9-1](#) では、すべてのユースケースを 1 つの図にまとめています。

### 個々のユースケース

個々のユースケースは、次のように配置されています。

- **ユースケース図**: ユースケースを表す図
- **用途**: このユースケースの用途
- **使用上の注意**: 実装に有効なガイドライン
- **構文**: このアクティビティの実行に使用する主な構文
- **例**: 各プログラム環境でのユースケースの例

# ユースケース・モデル: 管理インタフェース – 基本操作

表 9-1「ユースケース・モデル: 管理インタフェース – 基本操作」の「+」は、そのユースケースで、プログラム環境の例が記載されていることを示します。

この表では、プログラム環境を次の略称で表しています。

- P – DBMS\_AQADM および DBMS\_AQ パッケージを使用した PL/SQL
- O – OCI (Oracle コール・インタフェース) を使用した C
- V – OO4O (Oracle Objects for OLE) を使用した Visual Basic
- J – JDBC (Java Database Connectivity) を使用した Java (ネイティブ AQ)
- JM – JDBC (Java Database Connectivity) を使用した Java (JMS 標準)

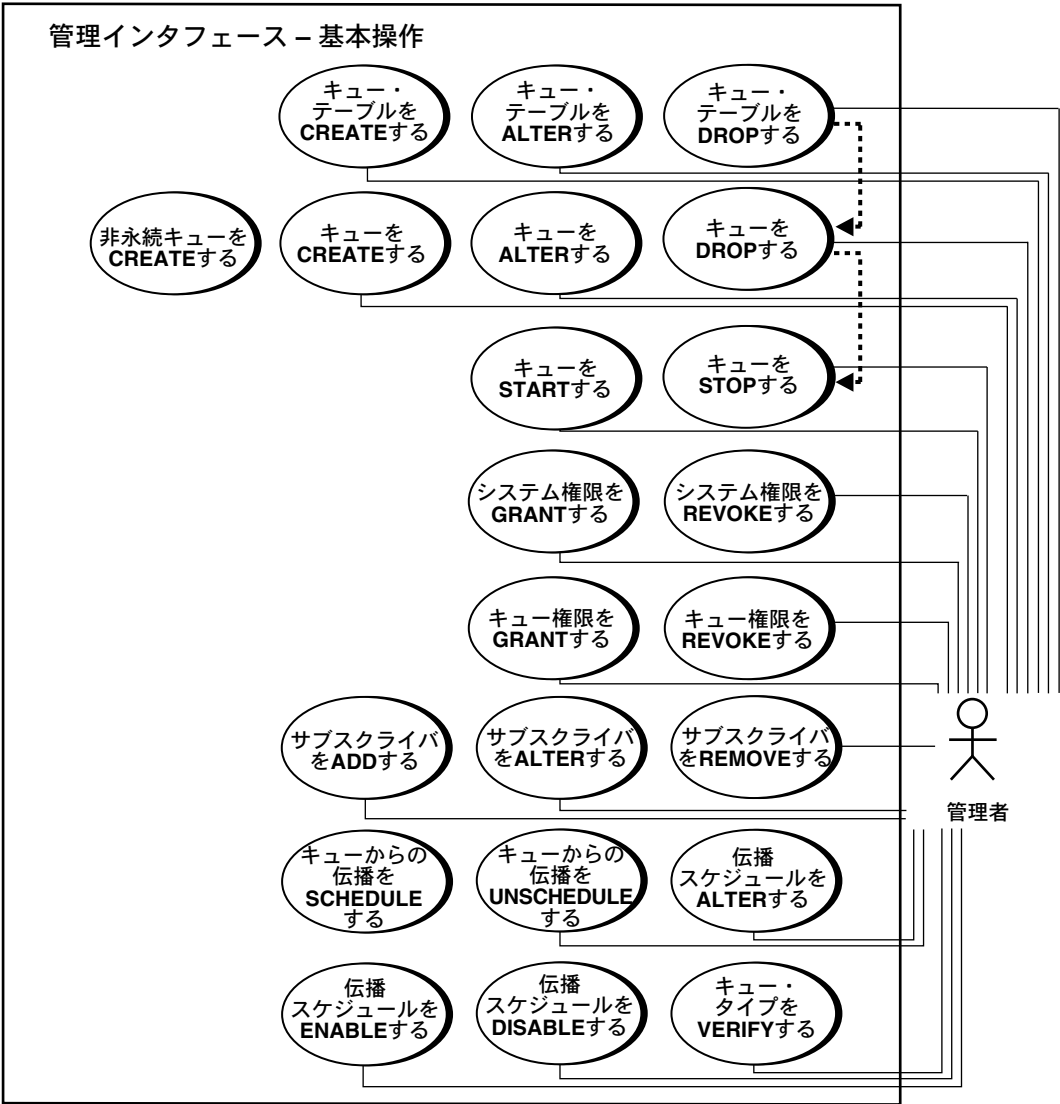
表 9-1 ユースケース・モデル: 管理インタフェース – 基本操作

ユースケース	プログラム環境の例				
	P	O	V	J	JM
9-5 ページの「キュー・テーブルの作成」	+		+	+	
9-13 ページの「キュー・テーブルの作成 (STORAGE 句の設定)」	+			+	
9-14 ページの「キュー・テーブルの変更」	+			+	
9-17 ページの「キュー・テーブルの削除」	+			+	
9-20 ページの「キューの作成」	+			+	
9-27 ページの「非永続キューの作成」	+				
9-30 ページの「キューの変更」	+			+	
9-33 ページの「キューの削除」	+			+	
9-36 ページの「キューの開始」	+			+	
9-39 ページの「キューの停止」	+			+	
9-42 ページの「システム権限の付与」	+			+	
9-45 ページの「システム権限の取消し」	+				
9-47 ページの「キュー権限の付与」	+			+	
9-50 ページの「キュー権限の取消し」	+			+	
9-53 ページの「サブスクライバの追加」	+			+	

表 9-1 ユースケース・モデル: 管理インタフェース – 基本操作 (続き)

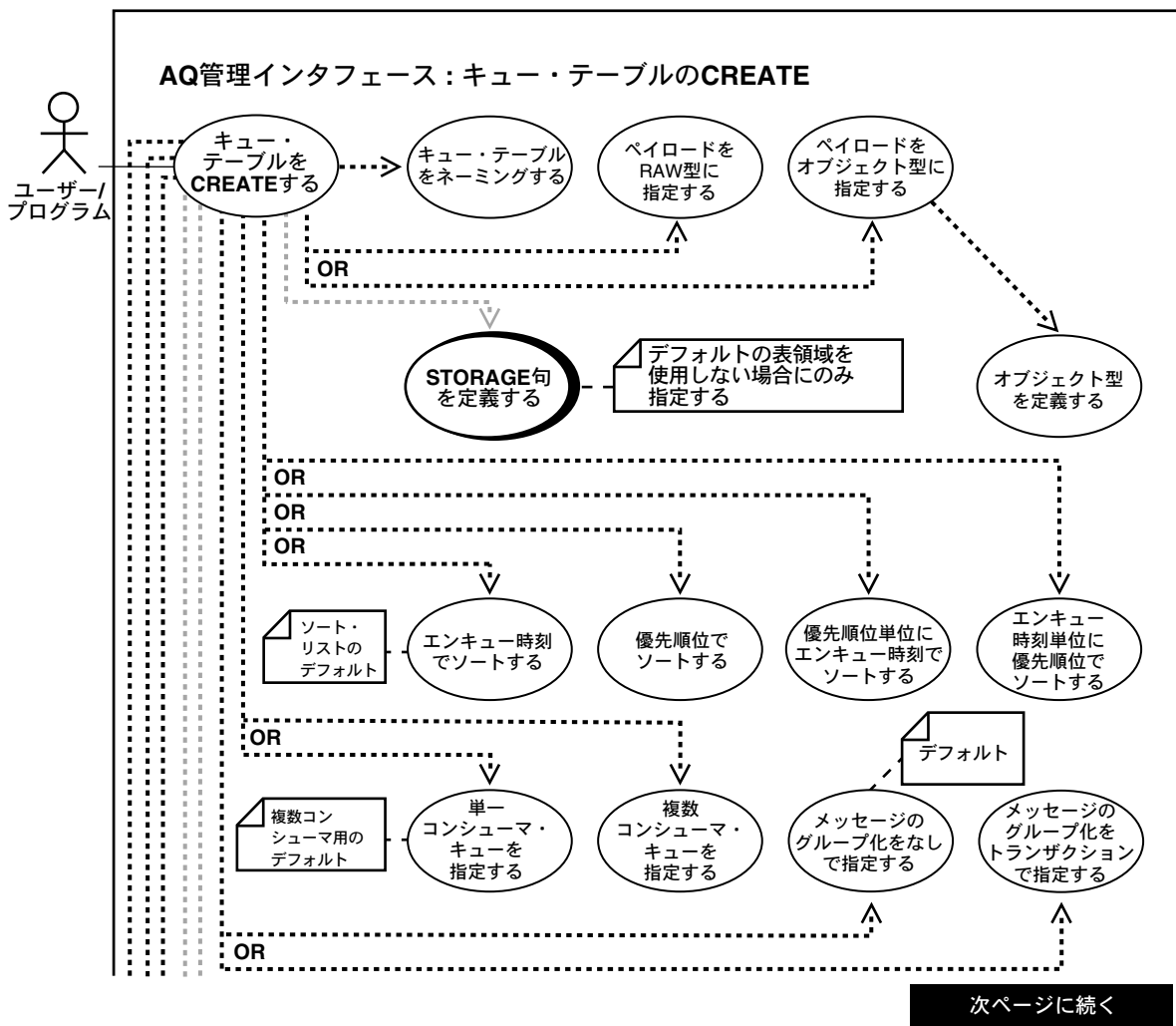
ユースケース	プログラム環境の例				
	P	O	V	J	JM
9-58 ページの「サブスクリイバの変更」	+			+	
9-62 ページの「サブスクリイバの削除」	+			+	
9-65 ページの「キューの伝播のスケジュール」	+			+	
9-69 ページの「キューの伝播スケジュールの解除」	+			+	
9-72 ページの「キュー・タイプの検証」	+				
9-75 ページの「伝播スケジュールの変更」	+			+	
9-79 ページの「伝播スケジュールの使用可能化」	+			+	
9-82 ページの「伝播スケジュールの使用不可」	+			+	

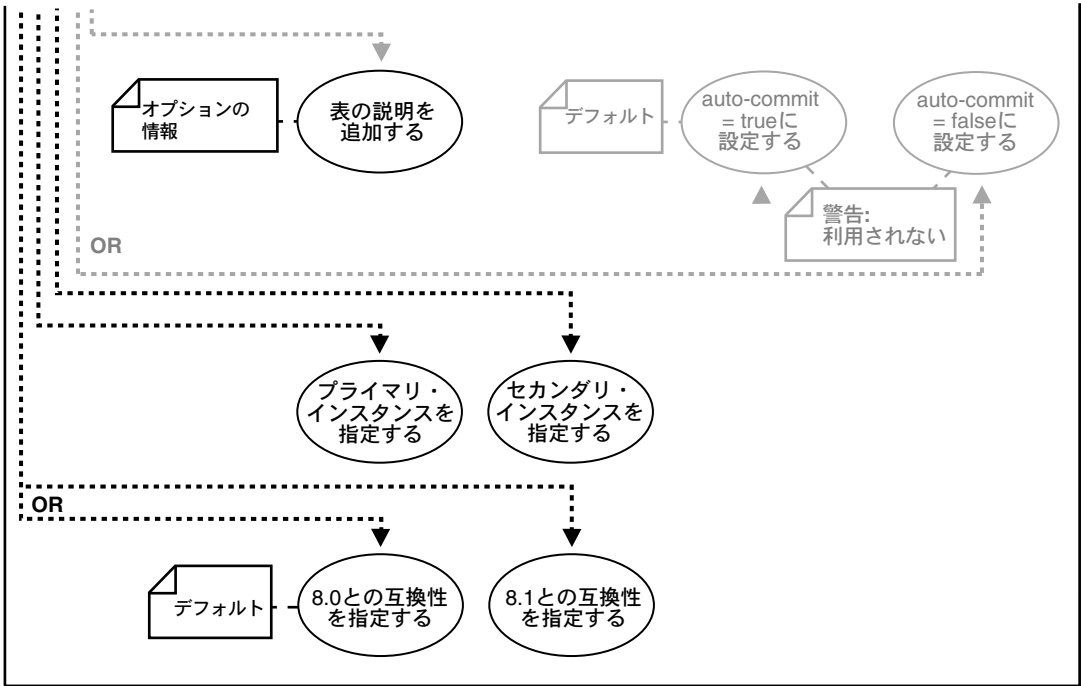
図 9-1 ユースケース図: 管理インタフェース - 基本操作



# キュー・テーブルの作成

図 9-2 ユースケース図：キュー・テーブルの作成





**参照:**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:管理インタフェース – 基本操作」を参照してください。

**用途**

事前定義された型のメッセージを持つキュー・テーブルを作成します。

**使用上の注意**

- デキュー順序付け用のソート・キーを使用する場合は、表の作成時に定義する必要があります。この時点では、次のオブジェクトが作成されます。

- キュー・テーブルに対応付けられた `aq$_<queue_table_name>_e` という名前のデフォルトの例外キュー
- AQ アプリケーションでキュー・データの問合せに使用される、`aq$_<queue_table_name>` という名前の読み専用ビュー
- `aq$_<queue_table_name>_t` という名前のキュー・モニター操作の索引
- 複数コンシューマ・キューの場合に使用される、`aq$_<queue_table_name>_i` という名前のデキュー操作の索引または索引構成表 (IOT)
- 8.1 互換の複数コンシューマ・キュー・テーブルの場合、次の追加オブジェクトが作成されます。
  - `aq$_<queue_table_name>_s` という名前の表。この表にはサブスクライバの情報が格納されます。
  - `aq$_<queue_table_name>_r` という名前の表。この表にはサブスクリプション・ルール情報が格納されます。
  - `aq$_<queue_table_name>_h` という名前の索引構成表 (IOT)。この表にはデキュー履歴データが格納されます。
- AQ メッセージでは、CLOB、BLOB または BFILE オブジェクトが有効です。Oracle8i では、AQ 伝播を使用してこれらのオブジェクト型を伝播できません。LOB を持つオブジェクト型をエンキューするには、まず、`LOB_attribute` を `EMPTY_BLOB()` に設定してからエンキューを実行する必要があります。これによって、キュー・テーブルのビューから生成された LOB ロケータを選択でき、標準の LOB 操作を使用できます。詳細は、『Oracle8i アプリケーション開発者ガイド ラージ・オブジェクト』を参照してください。
- 8.1 互換のモードの場合のみ、`primary_instance` および `secondary_instance` を指定および変更できます。
- プライマリ・インスタンスがない場合は、セカンダリ・インスタンスを指定できません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の `CREATE_QUEUE_TABLE` プロシージャ

- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQSession」の createQueueTable

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

- [PL/SQL \(DBMS\\_AQADM パッケージ\) : キュー・テーブルの作成](#) (9-8 ページ)
- [VB \(OO4O\) : キュー・テーブルの作成](#) (9-9 ページ)
- [Java \(JDBC\) : キュー・テーブルの作成](#) (9-10 ページ)

## PL/SQL (DBMS\_AQADM パッケージ) : キュー・テーブルの作成

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
GRANT CONNECT, RESOURCE TO aqadm;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
```

---

## オブジェクト型のメッセージを持つキューのキュー・テーブルを作成する

```
CREATE type aq.Message_typ as object (
    Subject          VARCHAR2(30),
    Text             VARCHAR2(80));
```

/\* 注意：スキーマを明示的に指定しない場合は、デフォルトでユーザーのスキーマに設定されます。\*/

```
EXECUTE dbms_aqadm.create_queue_table (
    Queue_table      => 'aq.ObjMsgs_qtab',
    Queue_payload_type => 'aq.Message_typ');
```



## RAW 型のメッセージを持つキューのキュー・テーブルを作成する

```
EXECUTE dbms_aqadm.create_queue_table (  
  Queue_table          => 'aq.RawMsgs_qtab',  
  Queue_payload_type   => 'RAW');
```

## 優先メッセージのキュー・テーブルを作成する

```
EXECUTE dbms_aqadm.create_queue_table (  
  Queue_table          => 'aq.PriorityMsgs_qtab',  
  Sort_list            => 'PRIORITY,ENQ_TIME',  
  Queue_payload_type   => 'aq.Message_typ');
```

## 複数コンシューマのキュー・テーブルを作成する

```
EXECUTE dbms_aqadm.create_queue_table (  
  Queue_table          => 'aq.MultiConsumerMsgs_qtab',  
  Multiple_consumers   => TRUE,  
  Queue_payload_type   => 'aq.Message_typ');
```

## 8.1 互換の複数コンシューマのキュー・テーブルを作成する

```
EXECUTE dbms_aqadm.create_queue_table (  
  Queue_table          => 'aq.Multiconsumermsgs8_1qtab',  
  Multiple_consumers   => TRUE,  
  Compatible           => '8.1',  
  Queue_payload_type   => 'aq.Message_typ');
```

## 指定された表領域にキュー・テーブルを作成する

```
EXECUTE dbms_aqadm.create_queue_table(  
  queue_table          => 'aq.aq_tbsMsg_qtab',  
  queue_payload_type   => 'aq.Message_typ',  
  storage_clause       => 'tablespace aq_tbs');
```

## VB (0040) : キュー・テーブルの作成

0040 uses database functionality for this operation.

## Java（JDBC）：キュー・テーブルの作成

Java を使用したキュー・テーブルの作成方法の 3 つの例を、次に示します。

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;

CREATE type aq.Message_typ as object (
    Subject          VARCHAR2(30),
    Text             VARCHAR2(80));
```

---

### オブジェクト型のメッセージを持つキューのキュー・テーブルを作成する

```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;

    /* AQQueueTableProperty オブジェクト（ペイロード型 Message_typ）を作成します。*/
    qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");

    /* aq スキーマにキュー・テーブルを作成します。*/
    q_table = aq_sess.createQueueTable ("aq", "ObjMsgs_qtab", qtable_prop);

    System.out.println("Successfully created ObjMsgs_qtab in aq schema");
}
```

## RAW 型のメッセージを持つキューのキュー・テーブルを作成する

```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;

    /* AQQueueTableProperty オブジェクト (ペイロード型 RAW) を作成します。*/
    qtable_prop = new AQQueueTableProperty("RAW");

    /* aq スキーマにキュー・テーブルを作成します。*/
    q_table = aq_sess.createQueueTable ("aq", "RawMsgs_qtab", qtable_prop);

    System.out.println("Successfully created RawMsgs_qtab in aq schema");
}
```

### 3. Create a queue table for multiple consumers and prioritized messages

```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;

    qtable_prop = new AQQueueTableProperty("RAW");

    /* 複数コンシューマを使用可能にします。*/
    qtable_prop.setMultiConsumer(true);
    qtable_prop.setCompatible("8.1");

    /* 優先順位および enqueue_time でソート順を指定します。*/
    qtable_prop.setSortOrder("PRIORITY,ENQ_TIME");

    /* aq スキーマにキュー・テーブルを作成します。*/
    q_table = aq_sess.createQueueTable ("aq", "PriorityMsgs_qtab",
    qtable_prop);

    System.out.println("Successfully created PriorityMsgs_qtab in aq schema");
}
```

## 指定された表領域にキュー・テーブルを作成する

```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;

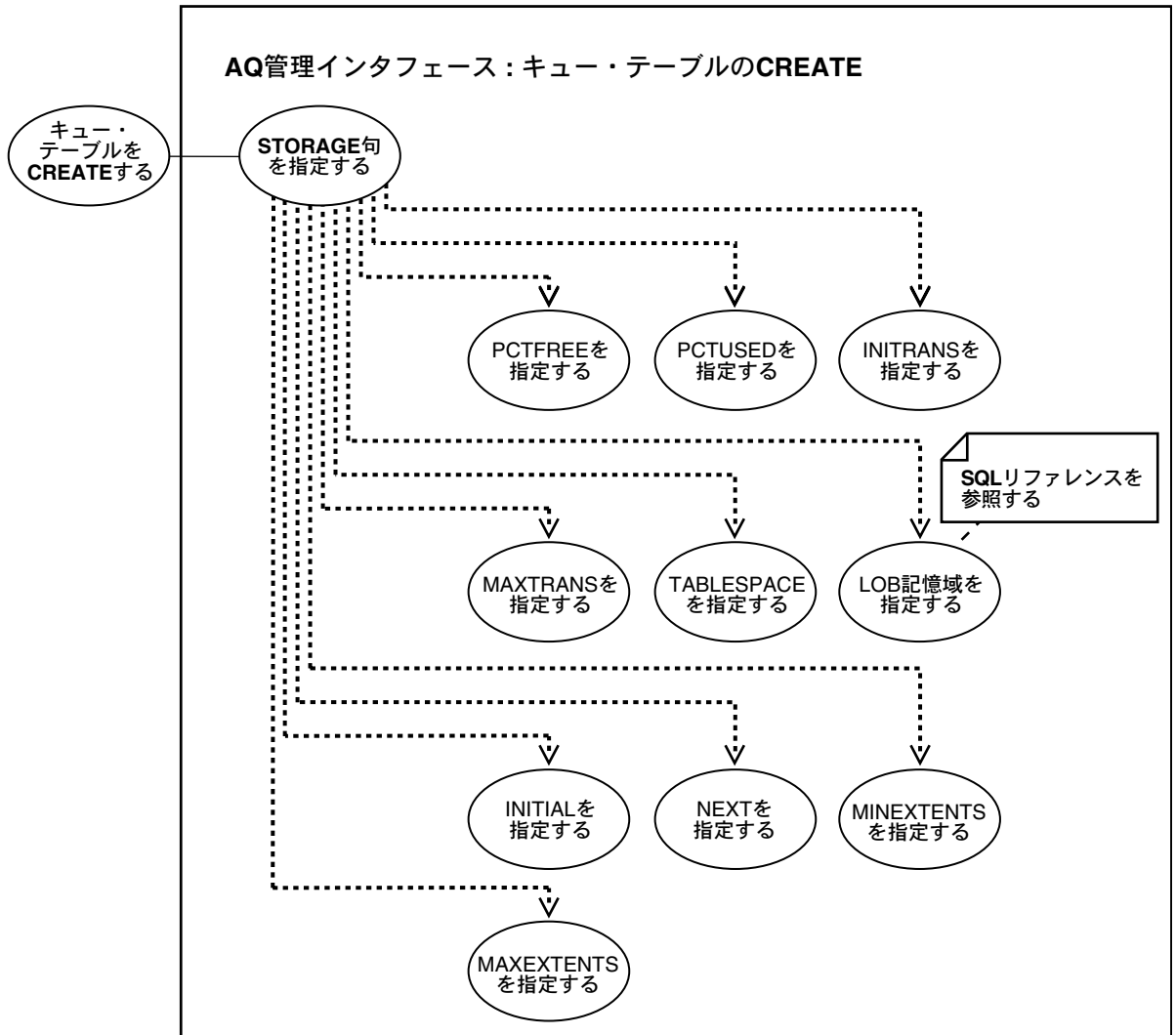
    /* AQQueueTableProperty オブジェクト (ペイロード型 Message_typ) を作成します。*/
    qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");

    /* キュー・テーブルに表領域を指定します。*/
    qtable_prop.setStorageClause("tablespace aq_tbs");

    /* aq スキーマにキュー・テーブルを作成します。*/
    q_table = aq_sess.createQueueTable ("aq", "aq_tbsMsg_qtab", qtable_prop);
}
```

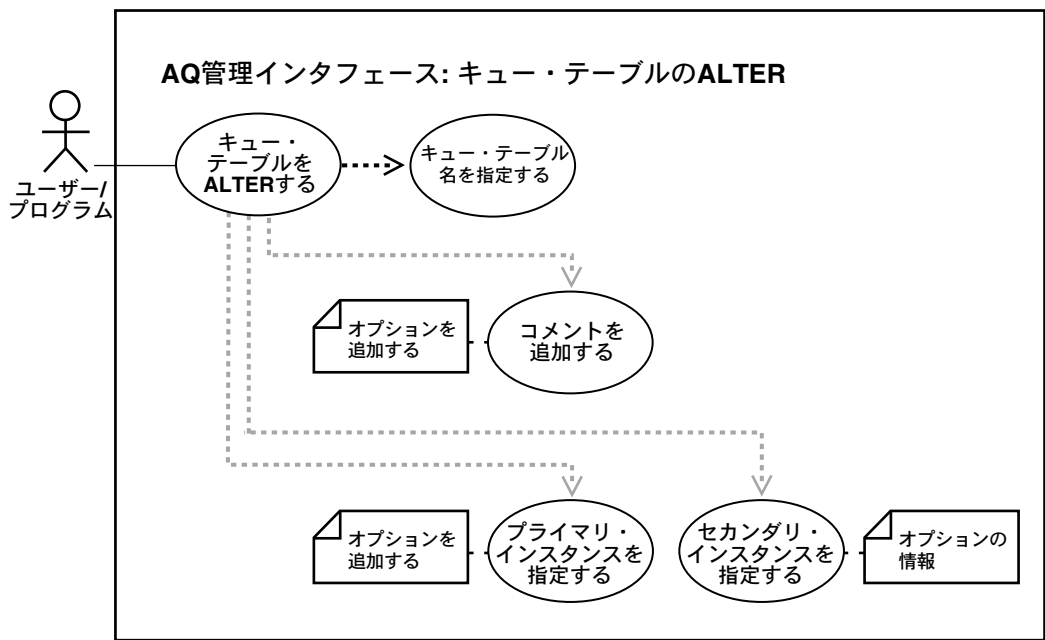
## キュー・テーブルの作成 (STORAGE 句の設定)

図 9-3 ユースケース図：キュー・テーブルの作成 (STORAGE 句の設定)



# キュー・テーブルの変更

図 9-4 ユースケース図：キュー・テーブルの変更



**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:管理インタフェース - 基本操作](#)」を参照してください。

## 用途

キュー・テーブルの既存のプロパティを変更します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第5章「DBMS\_AQADM」の ALTER\_QUEUE\_TABLE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の alterQueue

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM パッケージ\) : キュー・テーブルの変更](#) (9-15 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キュー・テーブルの変更](#) (9-16 ページ)

## PL/SQL (DBMS\_AQADM パッケージ) : キュー・テーブルの変更

/\* 表を変更して、キュー所有者のプライマリおよびセカンダリ・インスタンスを変更します (OPS 環境の場合のみ)。プライマリ・インスタンスは、キュー・テーブルの1次所有者のインスタンス番号です。セカンダリ・インスタンスは、キュー・テーブルの2次所有者のインスタンス番号です。\*/

```
EXECUTE dbms_aqadm.alter_queue_table (  
    Queue_table      => 'aq.ObjMsgs_qtab',  
    Primary_instance  => 3,  
    Secondary_instance => 2);
```

/\* 表を変更して、キュー・テーブルに対するコメントを変更します。\*/

```
EXECUTE dbms_aqadm.alter_queue_table (  
    Queue_table      => 'aq.ObjMsgs_qtab',  
    Comment          => 'revised usage for queue table');
```

## Java (JDBC) : キュー・テーブルの変更

```
/* キュー・テーブルを変更します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueTable            q_table;

    q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

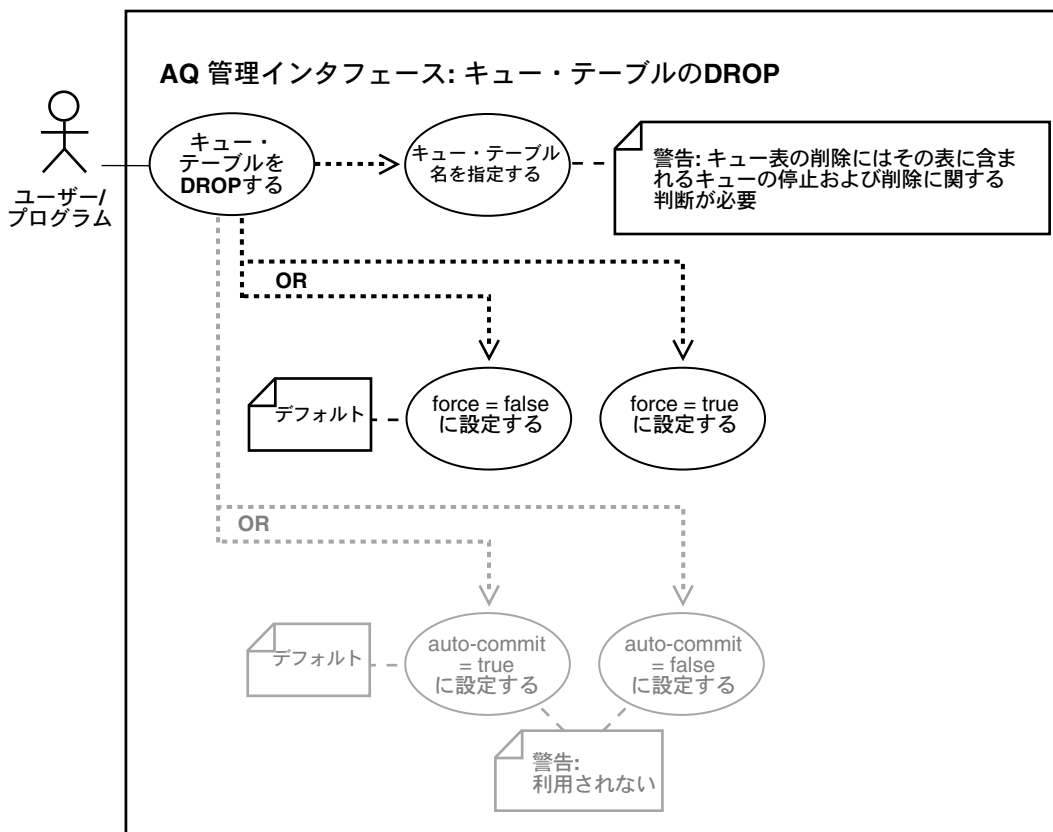
    /* キュー・テーブルのプロパティを取得します。*/
    qtable_prop = q_table.getProperty();

    /* キュー・テーブルのコメントおよびインスタンス親和性を変更します。*/
    q_table.alter("altered queue table", 3, 2);
}
```



## キュー・テーブルの削除

図 9-5 ユースケース図：キュー・テーブルの削除



### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル: 管理インタフェース – 基本操作」を参照してください。

### 用途

既存のキュー・テーブルを削除します。キュー・テーブルを削除する前に、そのキュー・テーブル内のすべてのキューを停止する必要があることに注意してください。force オプションを使用して自動的に行わない限り、この操作は明示的に行う必要があります。

### 使用上の注意

ありません。

### 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第5章「DBMS\_AQADM」の DROP\_QUEUE\_TABLE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueTable」の drop

### 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM パッケージ\) : キュー・テーブルの削除](#) (9-18 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キュー・テーブルの削除](#) (9-19 ページ)

### PL/SQL (DBMS\_AQADM パッケージ) : キュー・テーブルの削除

```
/* (すべてのキューがユーザーによって事前に削除されている) キュー・テーブルを削除します。*/  
EXECUTE dbms_aqadm.drop_queue_table (  
    queue_table      => 'aq.Objmsgs_qtab');
```

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

---

```
/* キュー・テーブルを削除し、すべてのキューがシステムによって停止および削除されるようにしま
す。*/
EXECUTE dbms_aqadm.drop_queue_table (
    queue_table      => 'aq.Objmsgs_qtab',
    force            => TRUE);
```

## Java (JDBC) : キュー・テーブルの削除

```
/* キュー・テーブルを削除します。 - すべてのキューは、ユーザーによってすでに削除されていま
す。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTable      q_table;

    q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

    /* キュー・テーブルを削除します。*/
    q_table.drop(false);
    System.out.println("Successful drop");
}

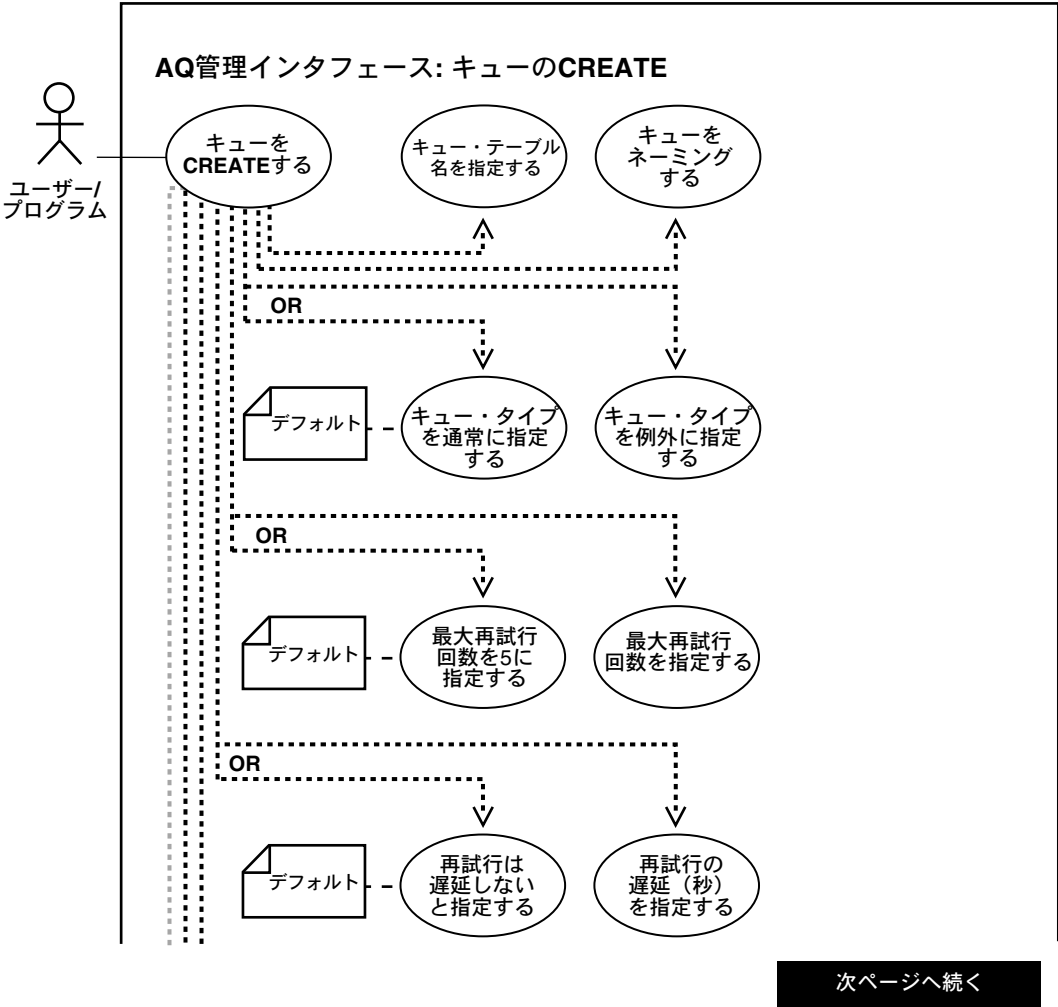
/* キュー・テーブルを削除します (また、すべてのキューがユーザーによって停止および削除されるよう
にします)。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTable      q_table;

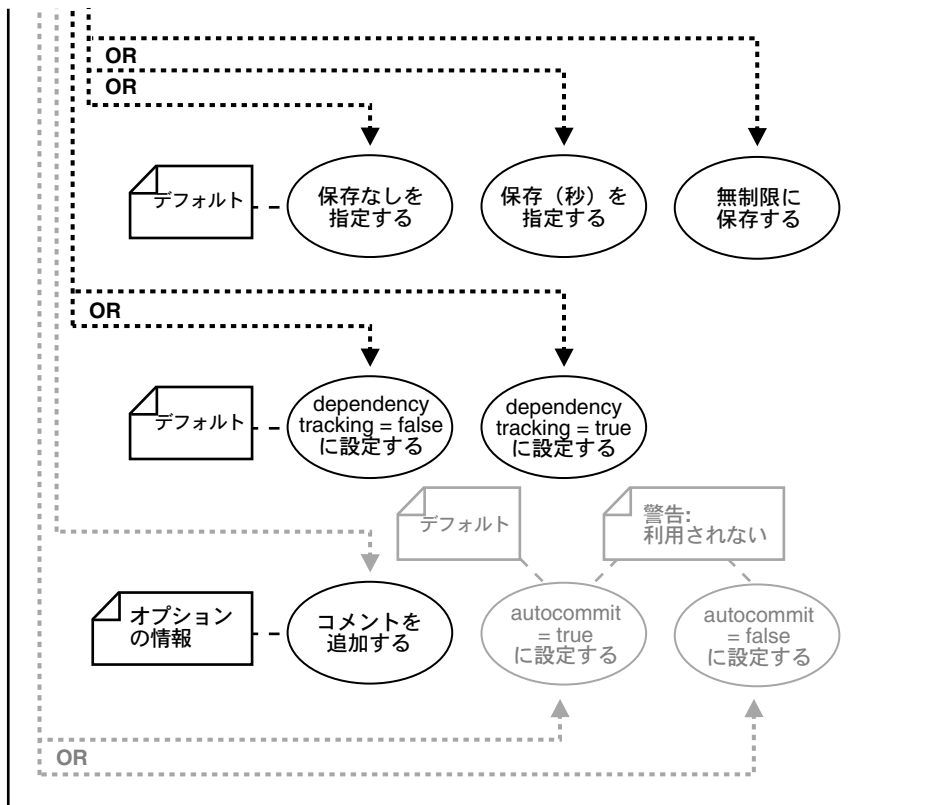
    q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

    /* キュー・テーブルを削除します (また、その中のすべてのキューが自動的に削除されるように
します)。*/
    q_table.drop(true);
    System.out.println("Successful drop");
}
```

# キューの作成

図 9-6 ユースケース図：キューの作成





### 参照:

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:管理インタフェース – 基本操作」を参照してください。

## 用途

指定したキュー・テーブルにキューを作成します。

### 使用上の注意

- すべてのキュー名は、スキーマ内において一意である必要があります。キューは、`CREATE_QUEUE` で作成した後、`START_QUEUE` をコールすると使用可能になります。デフォルトでは、キューはエンキューとデキューの両方を使用不可にして作成されます。
- 保存されているメッセージを参照するには、メッセージ ID によってデキューするか、または SQL を使用します。

### 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の `DROP_QUEUE_TABLE` プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャリファレンス』の第1章「パッケージ `oracle.AQ`」の「`AQSession`」の `CreateQueue`

### 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM パッケージ\) : キュー・テーブルの削除 \(9-18 ページ\)](#)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キュー・テーブルの削除 \(9-19 ページ\)](#)

## PL/SQL (DBMS\_AQADM) : キューの作成

### オブジェクト型のメッセージ用のキュー・テーブルにキューを作成する

```
/* メッセージ型を作成します。*/
CREATE type aq.Message_typ as object (
    Subject      VARCHAR2(30),
    Text         VARCHAR2(80));

/* オブジェクト型のキュー・テーブルおよびキューを作成します。*/
EXECUTE dbms_aqadm.create_queue_table (
    Queue_table      => 'aq.ObjMsgs_qtab',
    Queue_payload_type => 'aq.Message_typ');

EXECUTE dbms_aqadm.create_queue (
    Queue_name       => 'msg_queue',
    Queue_table      => 'aq.ObjMsgs_qtab');
```

### RAW 型のメッセージ用のキュー・テーブルにキューを作成する

```
/* RAW 型のキュー・テーブルおよびキューを作成します。*/
EXECUTE dbms_aqadm.create_queue_table (
    Queue_table      => 'aq.RawMsgs_qtab',
    Queue_payload_type => 'RAW');

/* キューを作成します。*/
EXECUTE dbms_aqadm.create_queue (
    Queue_name       => 'raw_msg_queue',
    Queue_table      => 'aq.RawMsgs_qtab');
```

### 優先メッセージのキュー・テーブルおよびキューを作成する

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

---

```
/* 優先メッセージのキュー・テーブルを作成します。*/
EXECUTE dbms_aqadm.create_queue_table (
    Queue_table      => 'aq.PriorityMsgs_qtab',
    Sort_list        => 'PRIORITY,ENQ_TIME',
    Queue_payload_type => 'aq.Message_typ');
/* キューを作成します。*/
```

```
EXECUTE dbms_aqadm.create_queue (  
    Queue_name      => 'priority_msg_queue',  
    Queue_table     => 'aq.PriorityMsgs_qtab');
```

### 複数コンシューマ用のキュー・テーブルおよびキューを作成する

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

---

```
/* 複数コンシューマ用のキュー・テーブルを作成します。*/  
EXECUTE dbms_aqadm.create_queue_table (  
    queue_table      => 'aq.MultiConsumerMsgs_qtab',  
    Multiple_consumers => TRUE,  
    Queue_payload_type => 'aq.Message_typ');  
  
/* キューを作成します。*/  
EXECUTE dbms_aqadm.create_queue (  
    Queue_name      => 'MultiConsumerMsg_queue',  
    Queue_table     => 'aq.MultiConsumerMsgs_qtab');
```

### 伝播を実証するためのキュー・テーブルおよびキューを作成する

```
/* キューを作成します。*/  
EXECUTE dbms_aqadm.create_queue (  
    Queue_name      => 'AnotherMsg_queue',  
    queue_table     => 'aq.MultiConsumerMsgs_qtab');
```

### 8.1 互換の複数コンシューマのキュー・テーブルおよびキューを作成する

```
/* リリース 8.1 互換の複数コンシューマのキュー・テーブルを作成します。*/  
EXECUTE dbms_aqadm.create_queue_table (  
    Queue_table      => 'aq.MultiConsumerMsgs81_qtab',  
    Multiple_consumers => TRUE,  
    Compatible       => '8.1',  
    Queue_payload_type => 'aq.Message_typ');  
  
EXECUTE dbms_aqadm.create_queue (  
    Queue_name      => 'MultiConsumerMsg81_queue',  
    Queue_table     => 'aq.MultiConsumerMsgs81_qtab');
```



## Java (JDBC) : キューの作成

### オブジェクト型のメッセージ用のキュー・テーブルにキューを作成する

```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueProperty      queue_prop;
    AQQueueTable         q_table;
    AQQueue              queue;

    q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

    /* 新しいAQQueueProperty オブジェクトを作成します。*/
    queue_prop = new AQQueueProperty();

    queue = aq_sess.createQueue (q_table, "msg_queue", queue_prop);
    System.out.println("Successful createQueue");
}
```

### RAW 型のメッセージ用のキュー・テーブルにキューを作成する

```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueProperty      queue_prop;
    AQQueueTable         q_table;
    AQQueue              queue;

    q_table = aq_sess.getQueueTable ("aq", "RawMsgs_qtab");

    /* 新しいAQQueueProperty オブジェクトを作成します。*/
    queue_prop = new AQQueueProperty();

    queue = aq_sess.createQueue (q_table, "msg_queue", queue_prop);
    System.out.println("Successful createQueue");
}
```

## 優先メッセージを持つ複数コンシューマ・キューを作成する

```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;
    AQAgent                 agent;

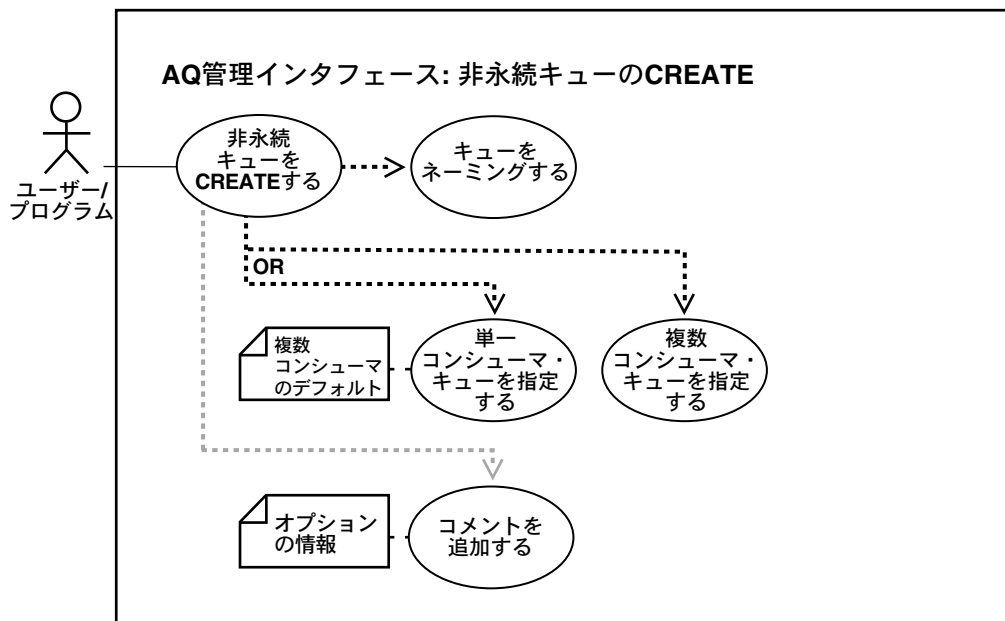
    qtable_prop = new AQQueueTableProperty("RAW");
    qtable_prop.setMultiConsumer(true);

    qtable_prop.setSortOrder("priority,enq_time");
    q_table = aq_sess.createQueueTable ("aq", "PriorityMsgs_qtab", qtable_prop);

    queue_prop = new AQQueueProperty();
    queue = aq_sess.createQueue (q_table, "priority_msg_queue", queue_prop);
}
```

## 非永続キューの作成

図 9-7 ユースケース図：非永続キューの作成



### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル：管理インタフェース – 基本操作」を参照してください。

## 用途

非永続の RAW 型キューを作成します。

## 使用上の注意

このキューは単一コンシューマまたは複数コンシューマ・キューのどちらかです。すべてのキュー名は、スキーマ内において一意である必要があります。このキューは、キュー名によって指定された同じスキーマ内にシステムが作成した 8.1 互換のキュー・テーブル (AQ\$\_MEM\_SC または AQ\$\_MEM\_MC) に作成されます。キュー名にスキーマ名が指定されていないときは、ログイン・ユーザーのスキーマに作成されます。キューは、CREATE\_NP\_QUEUE で作成した後、START\_QUEUE をコールすると使用可能になります。デフォルトでは、キューはエンキューとデキューの両方を使用不可にして作成されます。

ユーザーは、非永続キューからはデキューできません。非永続キューからメッセージを取り出すには、OCI 通知メカニズムを使用する方法しかありません (11-56 ページの「[通知登録](#)」を参照)。

非永続キューには、listen コールを起動できません (11-23 ページの「[1 個 \(複数個\) のキューのリスニング](#)」を参照)。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第 5 章「DBMS\_AQADM」の CREATE\_NP\_QUEUE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

## 例

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- PL/SQL (DBMS\_AQADM パッケージ) : キュー・テーブルの削除 (9-18 ページ)
- VB (OO4O) : 例はありません。
- Java (JDBC) : キュー・テーブルの削除 (9-19 ページ)

## PL/SQL (DBMS\_AQADM) : 非永続キューの作成

/\* 非永続単一コンシューマ・キューを作成します（注意：キュー・テーブルを作成する前に行います。） \*/

```
EXECUTE dbms_aqadm.create_np_queue(  
    Queue_name      => 'Singleconsumersmsg_npque',  
    Multiple_consumers => FALSE);
```

/\* 非永続複数コンシューマ・キューを作成します（注意：キュー・テーブルを作成する前に行います。） \*/

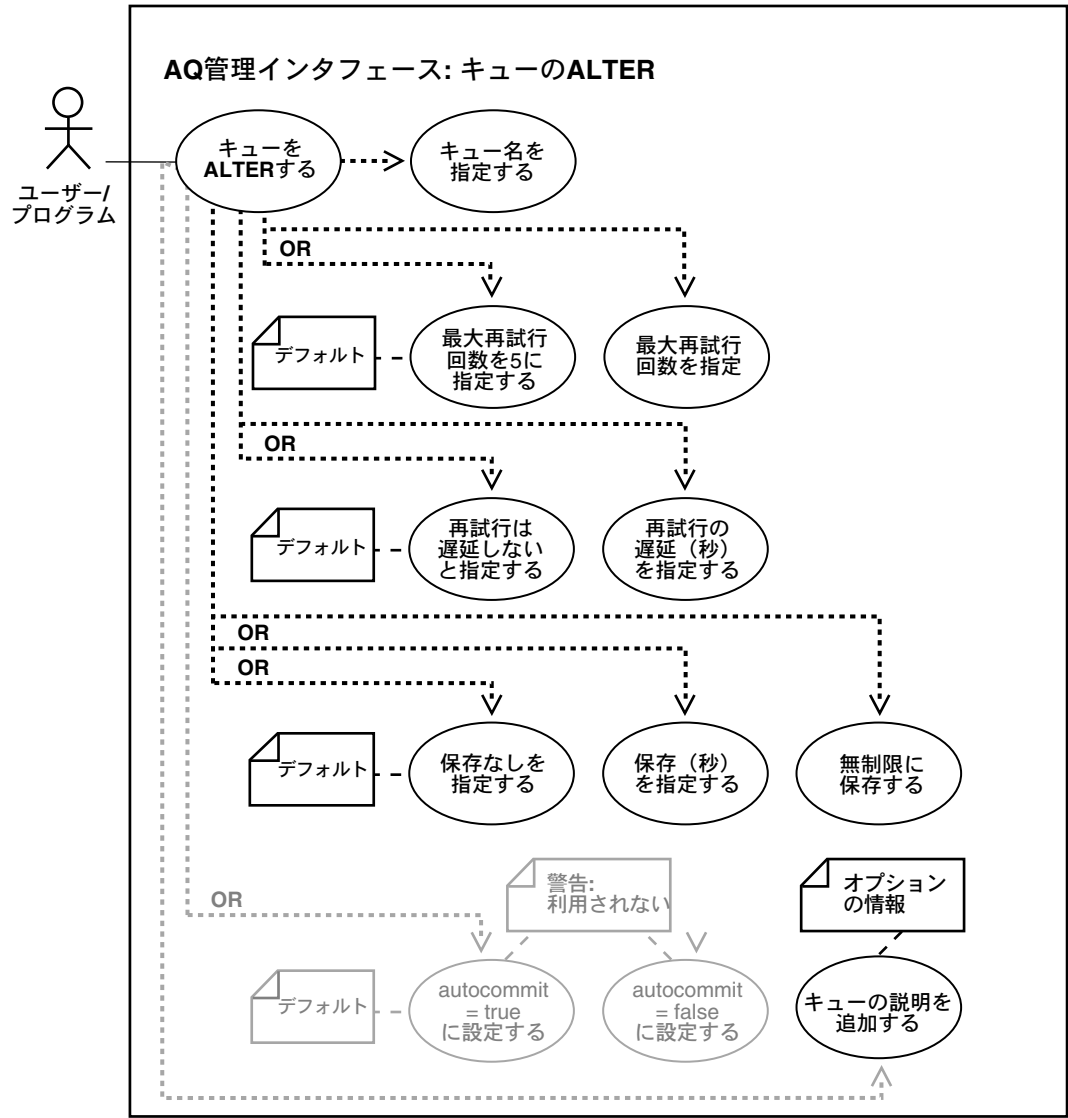
```
EXECUTE dbms_aqadm.create_np_queue(  
    Queue_name      => 'Multiconsumersmsg_npque',  
    Multiple_consumers => TRUE);
```

## Java (JDBC) : 非永続キューの作成

この機能は、Java API を介しては使用できません。

# キューの変更

図 9-8 ユースケース図：キューの変更



---

**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:管理インタフェース – 基本操作](#)」を参照してください。
- 

## 用途

キューの既存のプロパティを変更します。変更できるのは、max\_retries、comment、retry\_delay および retention\_time のみです。

## 使用上の注意

保存されているメッセージを参照するには、メッセージ ID によってデキューするか、または SQL を使用します。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第5章「DBMS\_AQADM」の ALTER\_QUEUE\_TABLE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueTable」の alter

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : キューの変更](#) (9-32 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キューの変更](#) (9-32 ページ)

## PL/SQL (DBMS\_AQADM) : キューの変更

```
/* キューを変更して保存時間を変更し、デキューの1日後にメッセージを保存します。*/  
EXECUTE dbms_aqadm.alter_queue (  
    queue_name      => 'aq.Anothermsg_queue',  
    retention_time   => 86400);
```

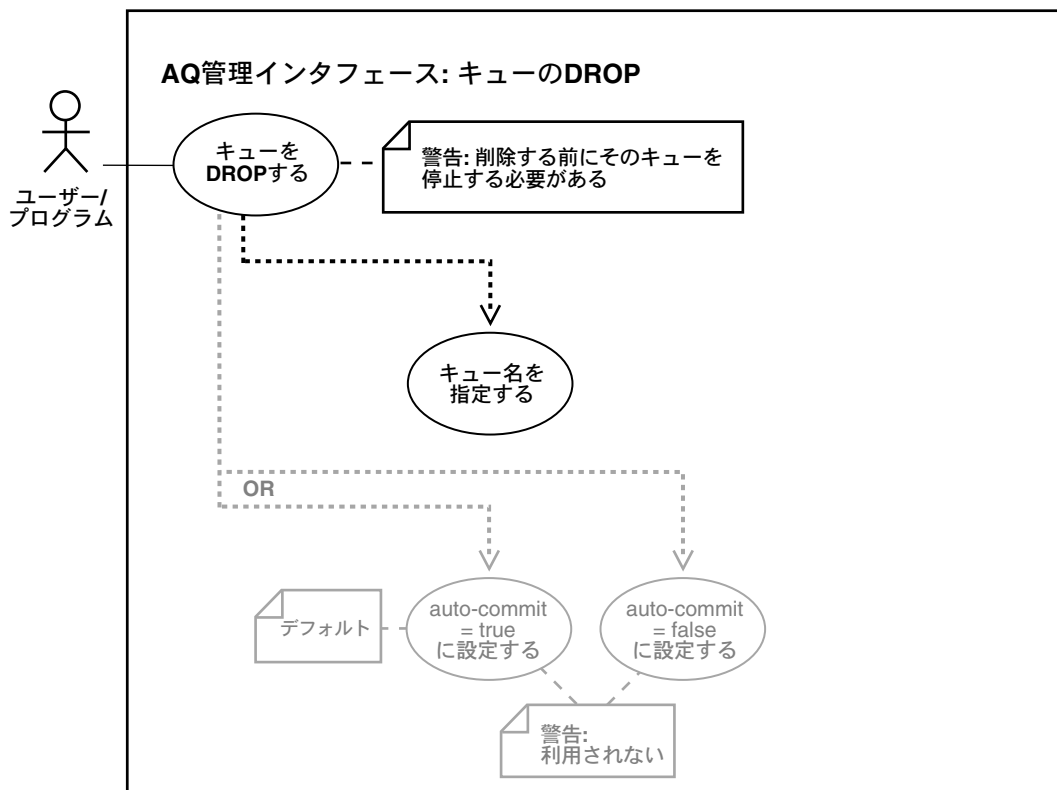
## Java (JDBC) : キューの変更

```
/* キューを変更して保存時間を変更し、デキューの1日後にメッセージを保存します。*/  
public static void example(AQSession aq_sess) throws AQException  
{  
    AQQueueProperty    queue_prop;  
    AQQueue             queue;  
  
    /* キュー・オブジェクトを取得します。*/  
    queue = aq_sess.getQueue("AQ", "Anothermsg_queue");  
  
    /* 新しいAQQueueProperty オブジェクトを作成します。*/  
    queue_prop = new AQQueueProperty();  
  
    /* 保存時間を1日に変更します。*/  
    queue_prop.setRetentionTime(new Double(86400));  
  
    /* キューを変更します。*/  
    queue.alterQueue(queue_prop);  
}
```



## キューの削除

図 9-9 ユースケース図：キューの削除



### 参照：

- 管理インターフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル: 管理インターフェース - 基本操作」を参照してください。

### 用途

既存のキューを削除します。あらかじめ `STOP_QUEUE` がコールされ、キューがエンキューおよびデキューの両方に対して使用不可にされていない限り、`DROP_QUEUE` は許可されません。すべてのキュー・データは、削除操作の一部として削除されます。

### 使用上の注意

ありません。

### 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の `DROP_QUEUE_TABLE` プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャリファレンス』の第1章「パッケージ `oracle.AQ`」の「`AQQueueTable`」の `dropQueue`

### 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : キューの削除](#) (9-35 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キューの削除](#) (9-35 ページ)

## PL/SQL (DBMS\_AQADM) : キューの削除

### 標準キューを削除する

/\* キューを削除する前にキューを停止します (キューをエンキューおよびデキューする前には、正常に停止しておく必要があります)。\*/

```
EXECUTE dbms_aqadm.stop_queue (  
    Queue_name      => 'aq.Msg_queue');
```

/\* キューを削除します。\*/

```
EXECUTE dbms_aqadm.drop_queue (  
    Queue_name      => 'aq.Msg_queue');
```

### 非永続キューを削除する

```
EXECUTE DBMS_AQADM.DROP_QUEUE( queue_name => 'Nonpersistent_singleconsumerq1');  
EXECUTE DBMS_AQADM.DROP_QUEUE( queue_name => 'Nonpersistent_multiconsumerq1');
```

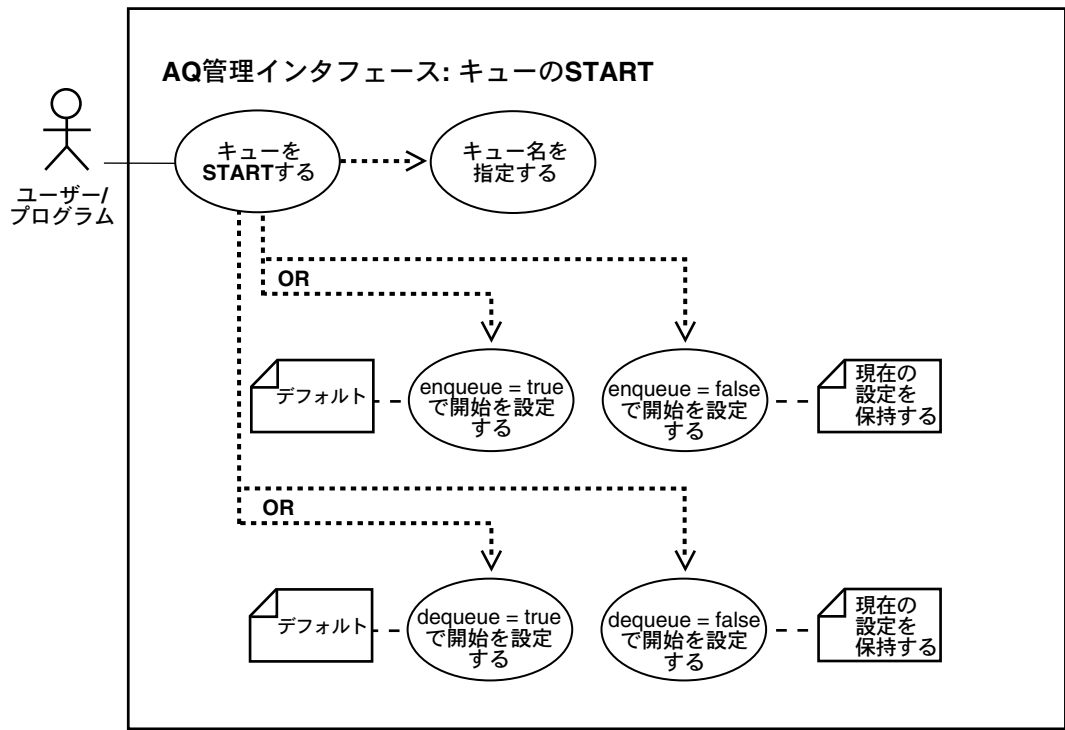
## Java (JDBC) : キューの削除

/\* キューを削除します。\*/

```
public static void example(AQSession aq_sess) throws AQException  
{  
    AQQueue queue;  
  
    /* キュー・オブジェクトを取得します。*/  
    queue = aq_sess.getQueue("AQ", "Msg_queue");  
  
    /* キューを停止します。*/  
    queue.stop(true);  
  
    /* キューを削除します。*/  
    queue.drop();  
}
```

# キューの開始

図 9-10 ユースケース図：キューの開始



**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:管理インタフェース – 基本操作」を参照してください。

## 用途

指定したキューに対するエンキューまたはデキュー（あるいはその両方）を使用可能にします。

## 使用上の注意

管理者は、キューを作成した後、START\_QUEUE を使用してそのキューを使用可能にする必要があります。デフォルトでは、ENQUEUE および DEQUEUE の両方を使用可能にします。例外キューに対しては、デキュー操作のみが可能です。この操作は、コールが完了し、コールにトランザクションの特性が一切ない場合にのみ有効になります。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第5章「DBMS\_AQADM」の START\_QUEUE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の start

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM パッケージ\) : キューの開始](#) (9-37 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キューの開始](#) (9-38 ページ)

## PL/SQL (DBMS\_AQADM パッケージ) : キューの開始

```
/* キューを開始して、エンキューおよびデキューを使用可能にします。*/  
EXECUTE dbms_aqadm.start_queue (  
    queue_name      => 'Msg_queue');  
  
/* 事前に停止しておいたキューを開始して、デキューのみ使用可能にします。*/  
EXECUTE dbms_aqadm.start_queue (  
    queue_name      => 'aq.msg_queue',  
    dequeue         => TRUE,  
    enqueue         => FALSE);
```

## Java (JDBC) : キューの開始

```
/* キューを開始します。 - エンキューおよびデキューを使用可能にします。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue                queue;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "Msg_queue");

    /* キューおよびデキューを使用可能にします。*/
    queue.start();
}

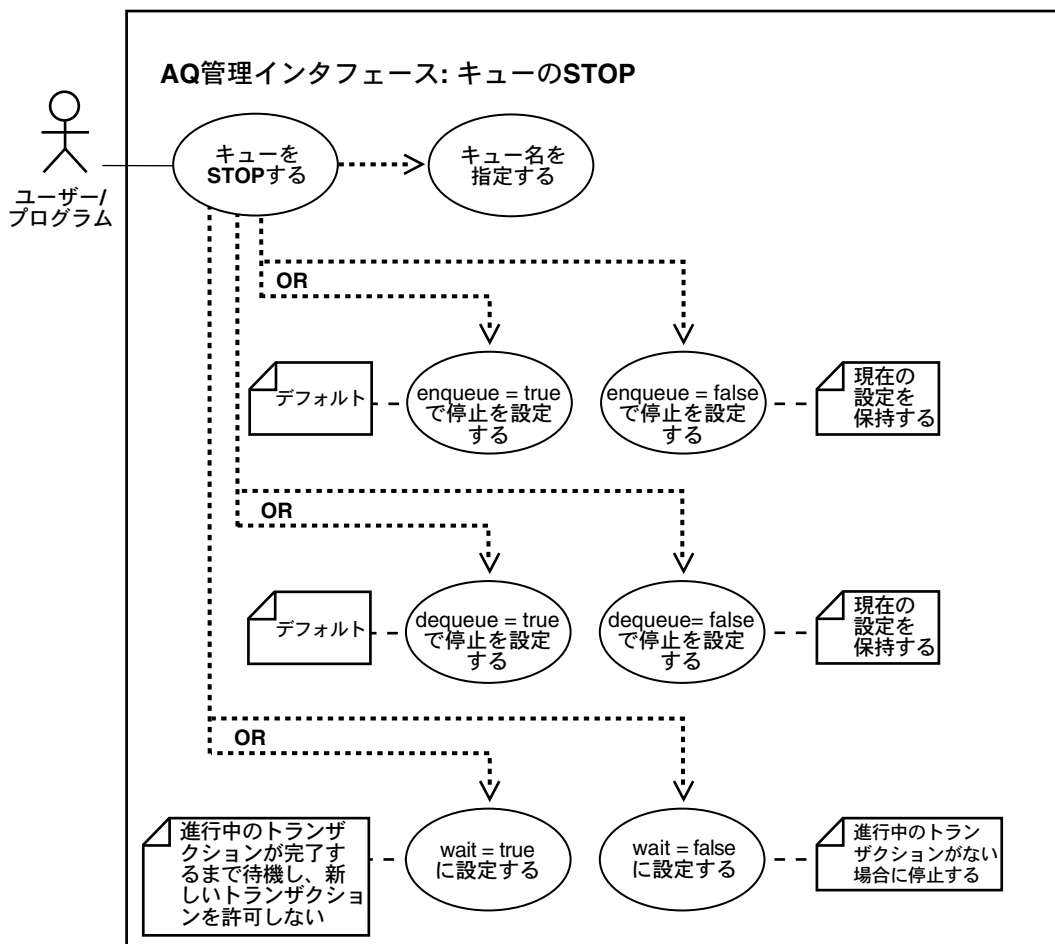
/* 事前に停止しておいたキューを開始して、デキューのみ使用可能にします。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue                queue;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "Msg_queue");

    /* キューおよびデキューを使用可能にします。*/
    queue.start(false, true);
}
```

## キューの停止

図 9-11 ユースケース図：キューの停止



---

---

### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル：管理インタフェース – 基本操作](#)」を参照してください。
- 
- 

## 用途

指定したキューに対するエンキューまたはデキュー（あるいはその両方）を使用不可にします。

## 使用上の注意

デフォルトでは、このコールによって ENQUEUE および DEQUEUE の両方が使用不可になります。キューは、未完了のトランザクションが存在する場合には停止できません。この操作は、コールが完了し、コールにトランザクションの特性が一切ない場合にのみ有効になります。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第5章「DBMS\_AQADM」の STOP\_QUEUE プロシージャ
- Visual Basic (OO4O)：参照マニュアルはありません。
- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の stop

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\)：キューの停止](#) (9-41 ページ)
- VB (OO4O)：例はありません。
- [Java \(JDBC\)：キューの停止](#) (9-41 ページ)



## PL/SQL (DBMS\_AQADM) : キューの停止

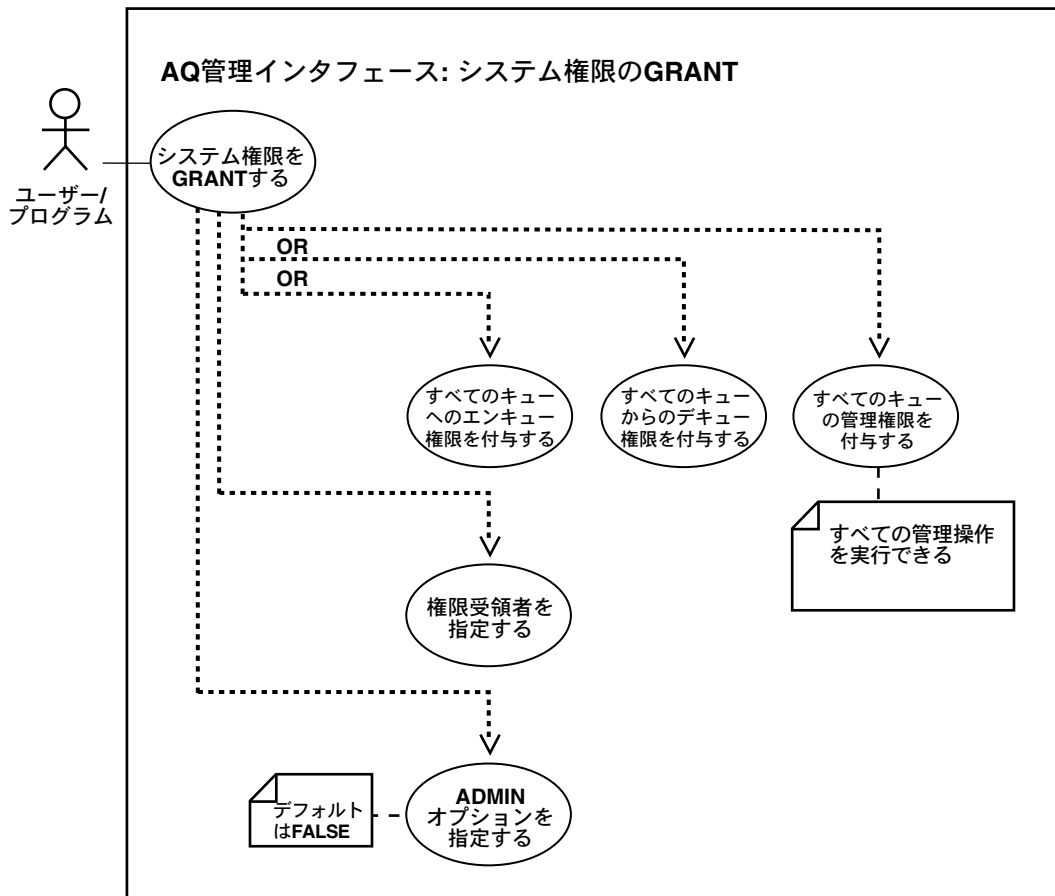
```
/* キューを停止します。*/  
EXECUTE dbms_aqadm.stop_queue (  
    queue_name      => 'aq.Msg_queue');
```

## Java (JDBC) : キューの停止

```
/* キューを停止します - 未完了のトランザクションを待ちます。*/  
public static void example(AQSession aq_sess) throws AQException  
{  
    AQQueue          queue;  
  
    /* キュー・オブジェクトを取得します。*/  
    queue = aq_sess.getQueue("AQ", "Msg_queue");  
  
    /* エンキューおよびデキューを使用可能にします。*/  
    queue.stop(true);  
}
```

## システム権限の付与

図 9-12 ユースケース図：システム権限の付与



### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル: 管理インタフェース – 基本操作」を参照してください。

## 用途

ユーザーおよびロールに AQ システム権限を付与します。この権限とは、ENQUEUE\_ANY、DEQUEUE\_ANY および MANAGE\_ANY です。最初は、SYS および SYSTEM のみがこのプロシージャを正常に使用できます。

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の GRANT\_SYSTEM\_PRIVILEGE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

## 使用上の注意

ありません。

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : システム権限の付与](#) (9-44 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : システム権限の付与](#) (9-44 ページ)

## PL/SQL (DBMS\_AQADM) : システム権限の付与

/\* ユーザー AQADM が、すべてのキューに対するエンキューおよびデキュー権限を付与します。 \*/

---

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

```
CONNECT system/manager;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
```

---

---

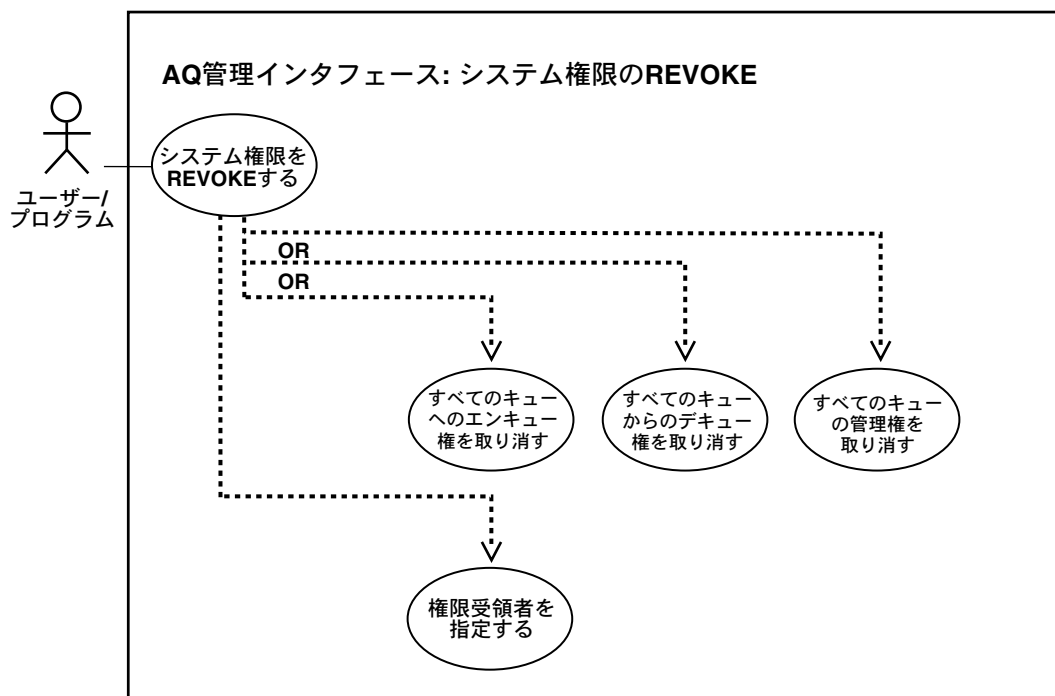
```
CONNECT aqadm/aqadm;
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
  privilege      =>    'ENQUEUE_ANY',
  grantee        =>    'Jones',
  admin_option   =>    FALSE);
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
  privilege      =>    'DEQUEUE_ANY',
  grantee        =>    'Jones',
  admin_option   =>    FALSE);
```

## Java (JDBC) : システム権限の付与

この機能は、Java API を介しては使用できません。

## システム権限の取消し

図 9-13 ユースケース図：システム権限の取消し

**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:管理インタフェース – 基本操作」を参照してください。

## 用途

ユーザーおよびロールの AQ システム権限を取り消します。この権限とは、ENQUEUE\_ANY、DEQUEUE\_ANY および MANAGE\_ANY です。システム権限の ADMIN オプションを選択的に取り消すことはできません。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の REVOKE\_SYSTEM\_PRIVILEGE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

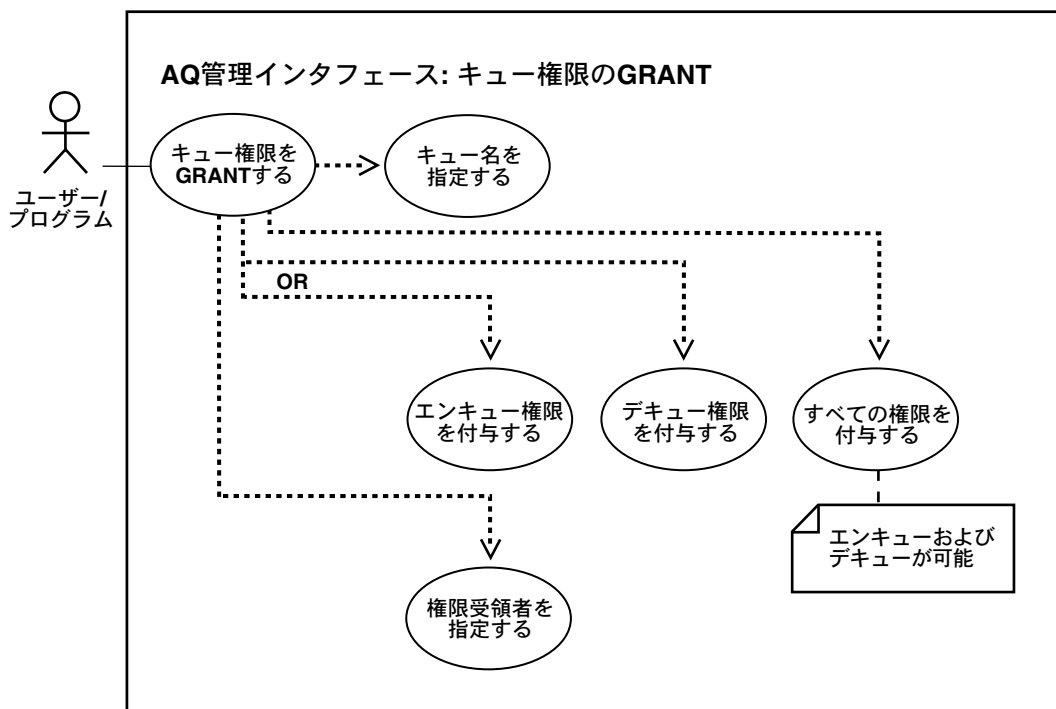
- [PL/SQL \(DBMS\\_AQADM\) の使用 : システム権限の取消し](#) (9-46 ページ)
- VB (OO4O) : 例はありません。
- Java (JDBC) : 例はありません。

## PL/SQL (DBMS\_AQADM) の使用 : システム権限の取消し

```
/* Jones から DEQUEUE_ANY システム権限を取り消します。*/  
CONNECT system/manager;  
execute DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(privilege=>'DEQUEUE_ANY',  
                                             grantee=>'Jones');
```

## キュー権限の付与

図 9-14 ユースケース図：キュー権限の付与

**参照：**

- 管理インターフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル: 管理インターフェース - 基本操作」を参照してください。

## 用途

ユーザーおよびロールにキュー権限を付与します。この権限とは、ENQUEUE または DEQUEUE です。最初は、キュー・テーブルの所有者のみがキュー権限を付与するプロシージャを使用できます。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の GRANT\_QUEUE\_PRIVILEGE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャリファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の grantQueuePrivilege

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : キュー権限の付与](#) (9-48 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キュー権限の付与](#) (9-49 ページ)

## PL/SQL (DBMS\_AQADM) : キュー権限の付与

/\* ユーザーに、DBMS\_AQADM.GRANT を使用してエンキューおよびデキューのためのアクセス権限を付与します。\*/

```
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (  
    privilege      =>    'ALL',  
    queue_name     =>    'aq.multiconsumermsg81_queue',  
    grantee        =>    'Jones',  
    grant_option    =>    TRUE);
```



## Java (JDBC) : キュー権限の付与

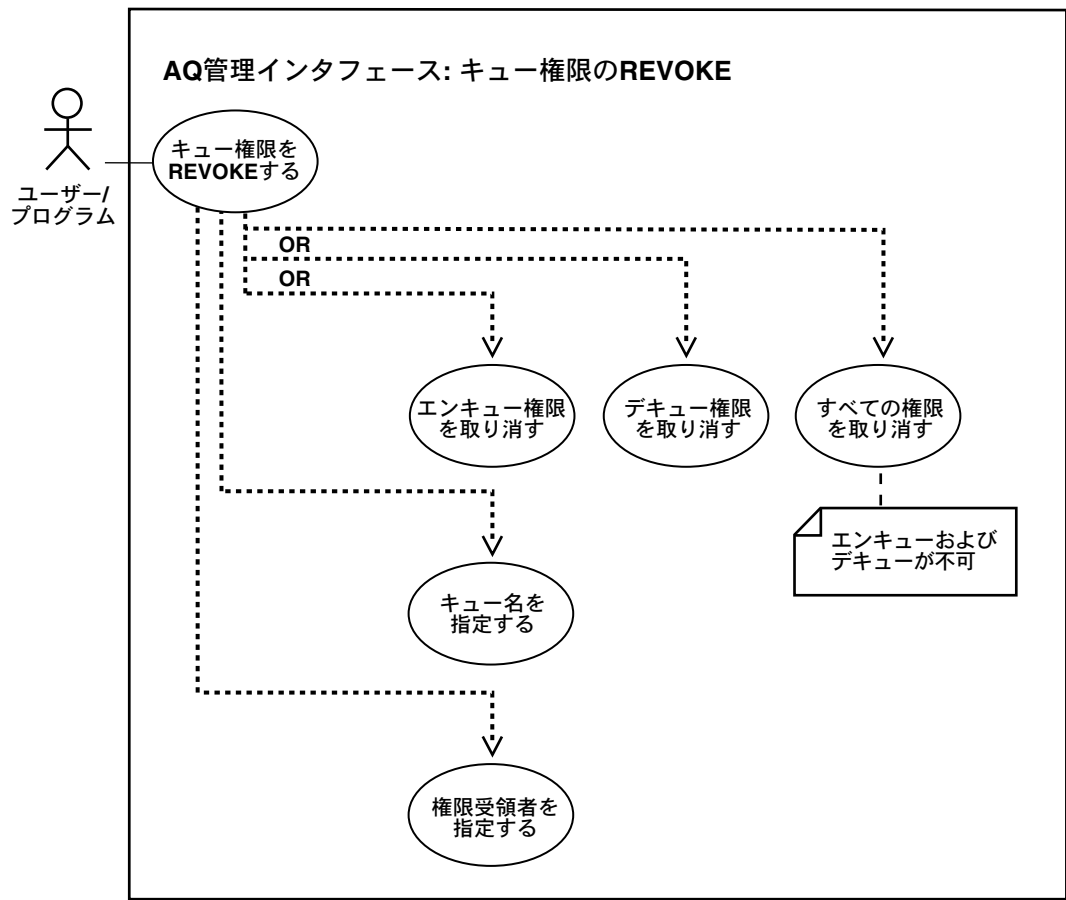
```
/* ユーザー 'Jones' に、キューに対するエンキューおよびデキュー権限を付与します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue queue;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "multiconsumermsg81_queue");

    /* エンキューおよびデキューを使用可能にします。*/
    queue.grantQueuePrivilege("ALL", "Jones", true);
}
```

# キュー権限の取消し

図 9-15 ユースケース図：キュー権限の取消し



**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:管理インタフェース－基本操作」を参照してください。

## 用途

ユーザーおよびロールのキュー権限を取り消します。この権限とは、ENQUEUE または DEQUEUE です。

## 使用上の注意

権限を取り消すユーザーは、取消しの対象となる権限の付与者である必要があります。GRANT オプションによって伝播された権限は、伝播させた付与者の権限が取り消されたときに取り消されます。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の REVOKE\_QUEUE\_PRIVILEGE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャリファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の revokeQueuePrivledge

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : キュー権限の取消し](#) (9-52 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キュー権限の取消し](#) (9-52 ページ)

## PL/SQL (DBMS\_AQADM) : キュー権限の取消し

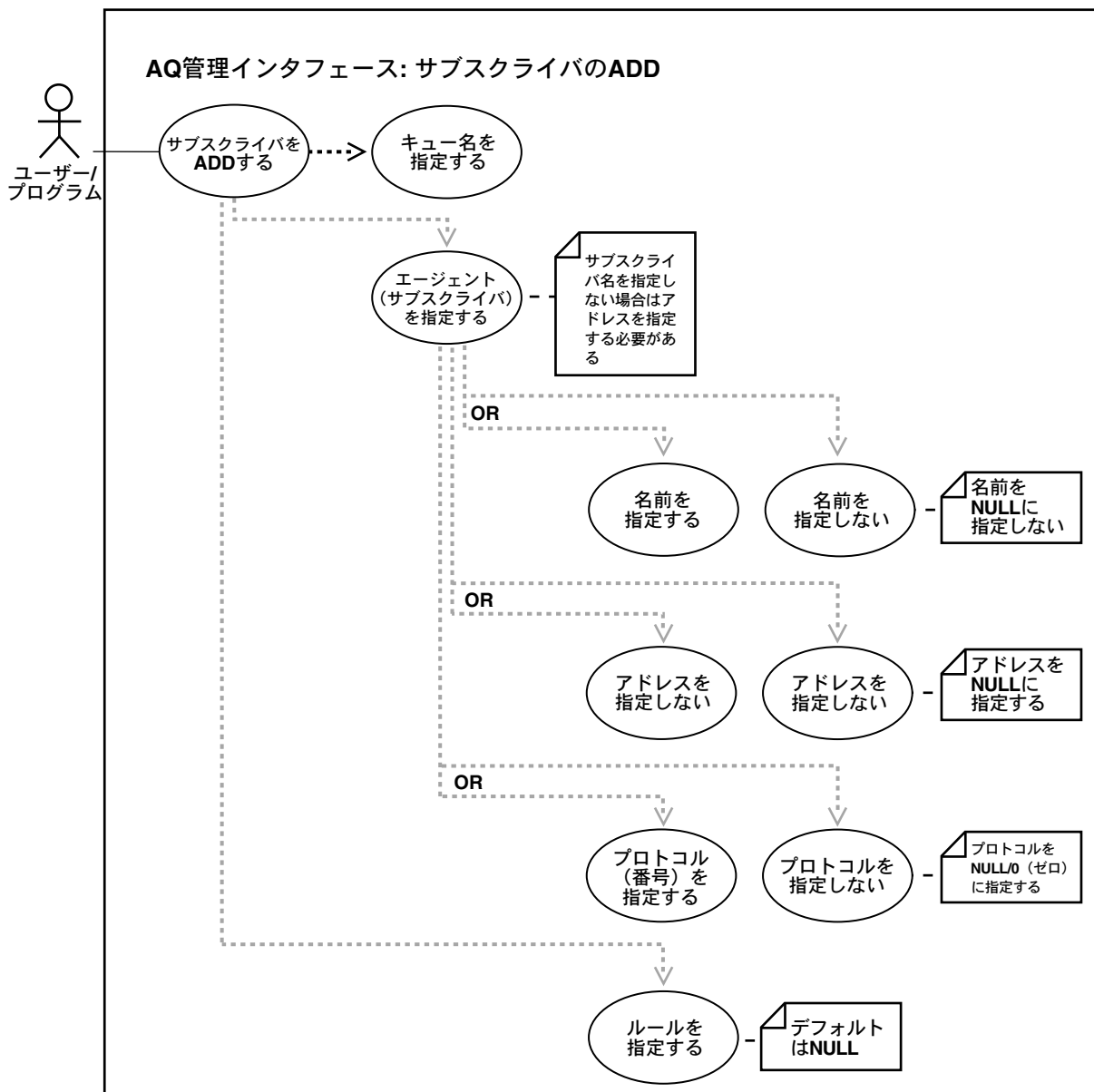
```
/* 権限付与者の特定のキューに対するエンキュー権限のみを残し、デキュー権限を取り消します。*/  
CONNECT scott/tiger;  
EXECUTE DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE(  
    privilege => 'DEQUEUE',  
    queue_name => 'scott.ScottMsgs_queue',  
    grantee => 'Jones');
```

## Java (JDBC) : キュー権限の取消し

```
/* 権限付与者の特定のキューに対するエンキュー権限のみを残し、デキュー権限を取り消します。*/  
public static void example(AQSession aq_sess) throws AQException  
{  
    AQQueue queue;  
  
    /* キュー・オブジェクトを取得します。*/  
    queue = aq_sess.getQueue("SCOTT", "ScottMsgs_queue");  
  
    /* エンキューおよびデキューを使用可能にします。*/  
    queue.revokeQueuePrivilege("DEQUEUE", "Jones");  
}
```

## サブスクライバの追加

図 9-16 ユースケース図: サブスクライバの追加



---

### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:管理インタフェース – 基本操作](#)」を参照してください。
- 

## 用途

デフォルトのサブスクライバをキューに追加します。

## 使用上の注意

- プログラムから特定の受信者リストまたはデフォルトのサブスクライバ・リストに、メッセージをエンキューできます。この操作は、複数コンシューマに対応したキューに対してのみ正常に実行できます。この操作はすぐに有効になり、この操作を含むトランザクションはコミットされます。このコールが完了した後に実行されるエンキュー要求には、新しい動作が反映されます。

- 次に示すように、ルール内のすべての文字列は引用符で囲む必要があります。

```
rule => 'PRIORITY <= 3 AND CORRID = ''FROM JAPAN'''
```

すべて一重引用符を使用することに注意してください。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第5章「DBMS\_AQADM」の ADD\_SUBSCRIBER プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の addSubscriber

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : サブスクライバの追加](#) (9-55 ページ)

- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : サブスクライバの追加](#) (9-55 ページ)

## PL/SQL (DBMS\_AQADM) : サブスクライバの追加

/\* データベース・リンクの設計済スキーマ内の設計済キューに、サブスクライバを追加するための無名 PL/SQL ブロック \*/

```
DECLARE
    subscriber          sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent ('subscriber1', 'aq2.msg_queue2@london', null);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name       => 'aq.multi_queue',
        subscriber       => subscriber);
END;
```

/\* サブスクライバをルール付きで追加します。\*/

```
DECLARE
    subscriber          sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent ('subscriber2', 'aq2.msg_queue2@london', null);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name       => 'aq.multi_queue',
        subscriber       => subscriber,
        rule             => 'priority < 2');
END;
```

## PL/SQL (DBMS\_AQADM) : ルールベースのサブスクライバの追加

```
DECLARE
    subscriber          sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent ('East_Shipping', 'ES.ES_bookedorders_que', null);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name       => 'OE.OE_bookedorders_que',
        subscriber       => subscriber,
        rule             => 'tab.user_data.orderregion = ''EASTERN'' OR
                           (tab.user_data.ordertype = ''RUSH'' AND
                            tab.user_data.customer.country = ''USA'') ');
END;
```

## Java (JDBC) : サブスクリイバの追加

```
/* 設定します。*/
public static void setup(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;

    /*AQQueueTableProperty オブジェクトを作成します。*/
    qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");
    qtable_prop.setMultiConsumer(true);

    q_table = aq_sess.createQueueTable ("aq", "multi_qtab", qtable_prop);

    /* 新しいAQQueueProperty オブジェクトを作成します。*/
    queue_prop = new AQQueueProperty();
    queue = aq_sess.createQueue (q_table, "multi_queue", queue_prop);
}

/* キューにサブスクリイバを追加します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue                queue;
    AQAgent                agent1;
    AQAgent                agent2;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "multi_queue");

    /* サブスクリイバを追加します。*/
    agent1 = new AQAgent("subscriber1", "aq2.msg_queue2@london");
    queue.addSubscriber(agent1, null);

    /* サブスクリイバをルール付きで追加します。*/
    agent2 = new AQAgent("subscriber2", "aq2.msg_queue2@london");

    queue.addSubscriber(agent2, "priority < 2");
}

/* サブスクリイバをルール付きで追加します。*/
public static void example(AQSession aq_sess) throws AQException
{

```



```
AQQueue      queue;
AQAgent      agent1;

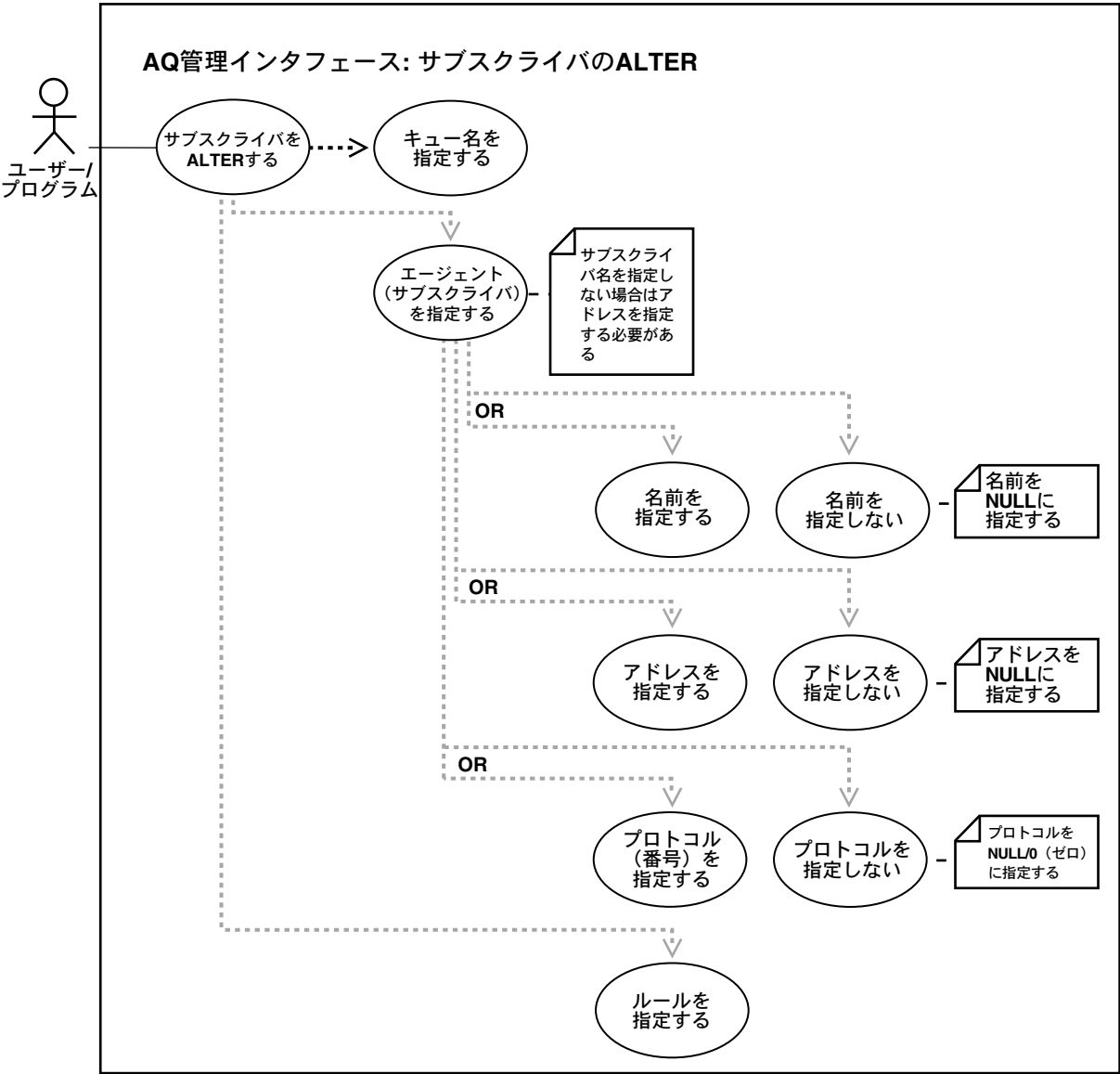
/* キュー・オブジェクトを取得します。*/
queue = aq_sess.getQueue("OE", "OE_bookedorders_que");

/* サブスクライバを追加します。*/
agent1 = new AQAgent("East_Shipping", "ES.ES_bookedorders_que");

queue.addSubscriber(agent1,
"tab.user_data.orderregion='EASTERN' OR " +
"(tab.user_data.ordertype='RUSH' AND " +
"tab.user_data.customer.country='USA')");
}
```

# サブスクリイバの変更

図 9-17 ユースケース図：サブスクリイバの変更



---

---

**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル：管理インタフェース – 基本操作](#)」を参照してください。
- 
- 

## 用途

指定されたキューのサブスクリイバの既存のプロパティを変更します。ルールのみを変更できます。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の ALTER\_SUBSCRIBER プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャリファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の alterSubscriber

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : サブスクリイバの変更](#) (9-60 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : サブスクリイバの変更](#) (9-61 ページ)

## PL/SQL (DBMS\_AQADM) : サブスクリイバの変更

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
    queue_table          => 'aq.multi_qtab',
    multiple_consumers   => TRUE,
    queue_payload_type   => 'aq.message_typ',
    compatible           => '8.1.5');
EXECUTE DBMS_AQADM.CREATE_QUEUE (
    queue_name           => 'multi_queue',
    queue_table          => 'aq.multi_qtab');
```

---

/\* サブスクリイバをルール付きで追加します。\*/

```
DECLARE
    subscriber          sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('SUBSCRIBER1', 'aq2.msg_queue2@london', null);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name       => 'aq.msg_queue',
        subscriber       => subscriber,
        rule              => 'priority < 2');
```

END;

/\* サブスクリイバのルールを変更します。\*/

```
DECLARE
    subscriber          sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('SUBSCRIBER1', 'aq2.msg_queue2@london', null);
    DBMS_AQADM.ALTER_SUBSCRIBER(
        queue_name       => 'aq.msg_queue',
        subscriber       => subscriber,
        rule              => 'priority = 1');
```

END;

## Java (JDBC) : サブスクライバの変更

```
/* サブスクライバのルールを変更します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue      queue;
    AQAgent      agent1;
    AQAgent      agent2;

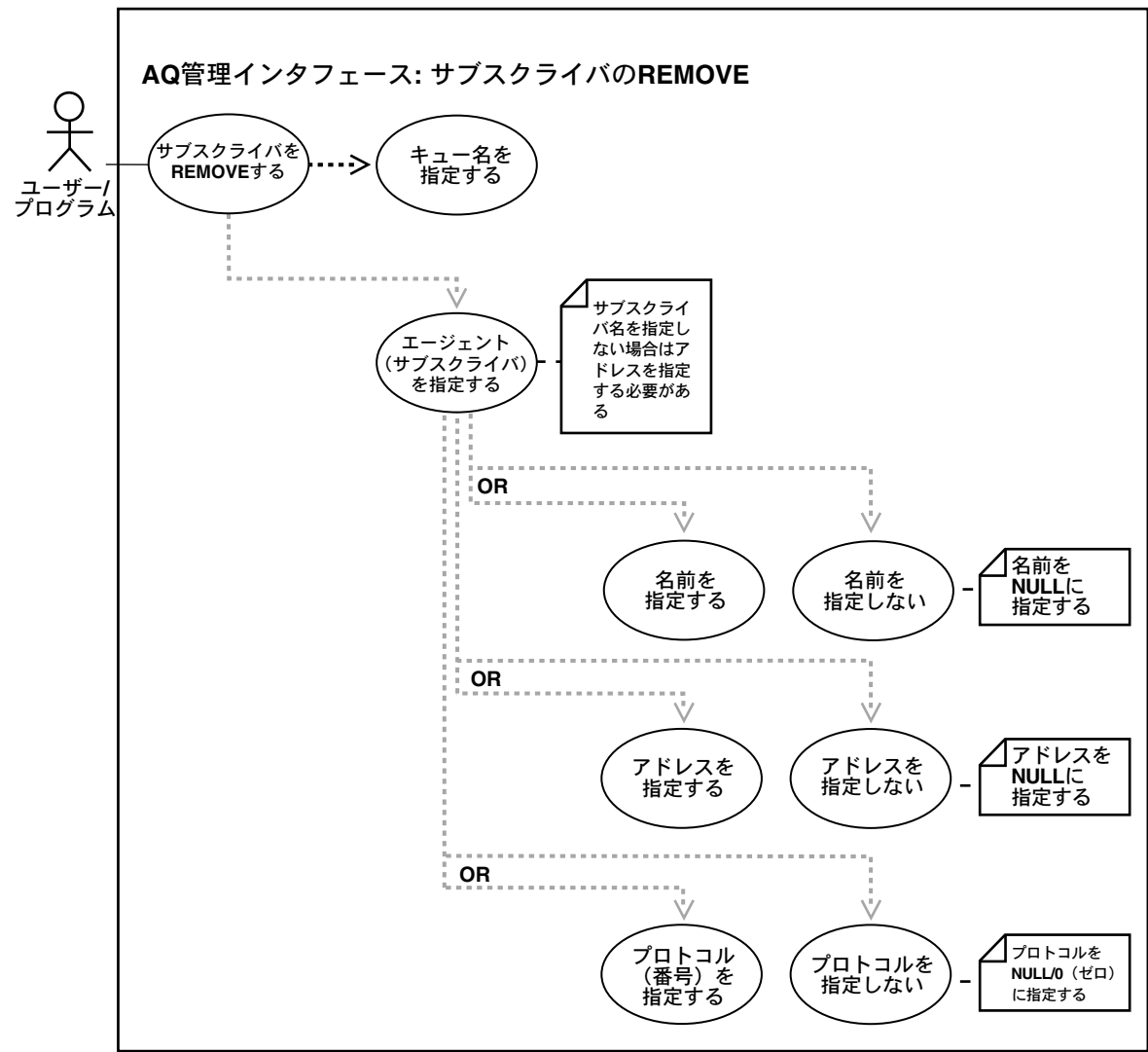
    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "multi_queue");

    /* サブスクライバを追加します。*/
    agent1 = new AQAgent("subscriber1", "aq2.msg_queue2@london");

    queue.alterSubscriber(agent1, "priority=1");
}
```

# サブスクライバの削除

図 9-18 ユースケース図：サブスクライバの削除



---

**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:管理インタフェース – 基本操作](#)」を参照してください。
- 

## 用途

デフォルトのサブスクライバをキューから削除します。

## 使用上の注意

この操作はすぐに有効になり、この操作を含むトランザクションはコミットされます。既存メッセージ内のこのサブスクライバに対するすべての参照は、操作の一部として削除されます。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第5章「DBMS\_AQADM」の REMOVE\_SUBSCRIBER プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の removeSubscriber

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : サブスクライバの削除](#) (9-64 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : サブスクライバの削除](#) (9-64 ページ)

## PL/SQL (DBMS\_AQADM) : サブスクリイバの削除

```
DECLARE
    subscriber      sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('subscriber1', 'aq2.msg_queue2', NULL);
    DBMS_AQADM.REMOVE_SUBSCRIBER(
        queue_name => 'aq.multi_queue',
        subscriber => subscriber);
END;
```

## Java (JDBC) : サブスクリイバの削除

```
/* サブスクリイバを削除します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue      queue;
    AQAgent      agent1;
    AQAgent      agent2;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "multi_queue");

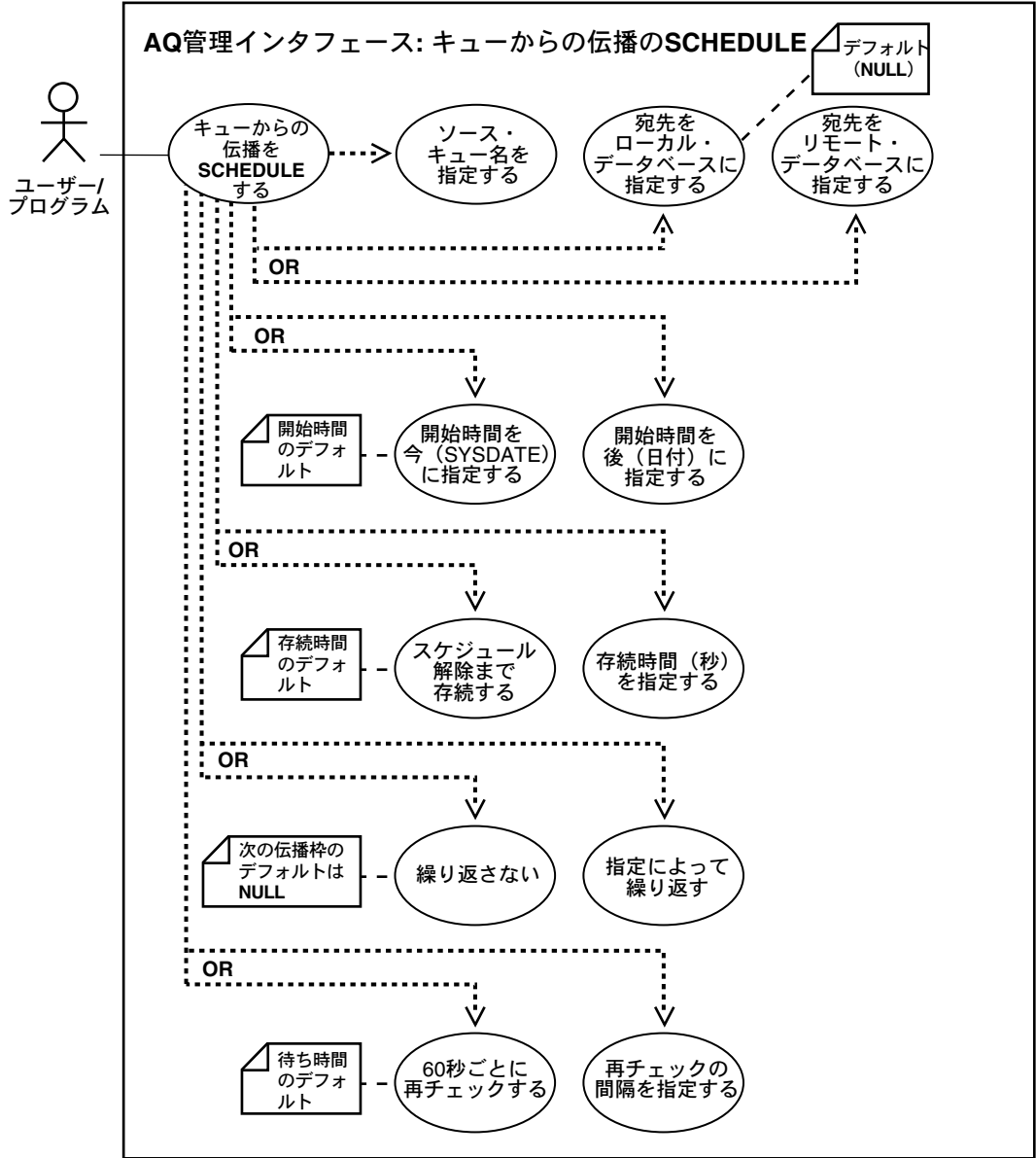
    /* サブスクリイバを追加します。*/
    agent1 = new AQAgent("subscriber1", "aq2.msg_queue2@london");

    queue.removeSubscriber(agent1);
}
```



# キューの伝播のスケジュール

図 9-19 ユースケース図：キューの伝播のスケジュール



---

---

### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル：管理インタフェース – 基本操作](#)」を参照してください。
- 
- 

## 用途

あるキューから特定の dblink で識別される宛先へのメッセージ伝播をスケジュールします。

## 使用上の注意

宛先に NULL を指定すると、メッセージは同じデータベース内の他のキューにも伝播されます。同じ宛先に複数の受信者を持つ場合、（キューが同じでも異なっても）メッセージは、すべての受信者に同時に伝播されます。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第 5 章「DBMS\_AQADM」の SCHEDULE\_PROPAGATION プロシージャ
- Visual Basic (OO4O)：参照マニュアルはありません。
- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャリファレンス』の第 1 章「パッケージ oracle.AQ」の「AQQueueAdmin」の schedulePropagation

## 例

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\)：キューの伝播のスケジュール](#) (9-67 ページ)
- VB (OO4O)：例はありません。
- [Java \(JDBC\)：キューの伝播のスケジュール](#) (9-67 ページ)

## PL/SQL (DBMS\_AQADM) : キューの伝播のスケジュール

**注意:** 次のようなデータ構造を設定しないと機能しない例もあります。

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
    queue_table           => 'aq.objmsgs_qtab',
    queue_payload_type    => 'aq.message_typ',
    multiple_consumers    => TRUE);
EXECUTE DBMS_AQADM.CREATE_QUEUE (
    queue_name            => 'aq.q1def',
    queue_table           => 'aq.objmsgs_qtab');
```

### あるキューから同じデータベース内の他のキューへの伝播をスケジュールする

/\* キュー aq.q1def 同じデータベース内の他のキューへ、伝播をスケジュールします。\*/  
EXECUTE DBMS\_AQADM.SCHEDULE\_PROPAGATION(  
 Queue\_name => 'aq.q1def');

### あるキューから別のデータベース内の他のキューへの伝播をスケジュールする

/\* キュー aq.q1def 同じデータベース内の他のキューへ、伝播をスケジュールします。\*/  
EXECUTE DBMS\_AQADM.SCHEDULE\_PROPAGATION(  
 Queue\_name => 'aq.q1def',  
 Destination => 'another\_db.world');

## Java (JDBC) : キューの伝播のスケジュール

```
/* 設定します。*/
public static void setup(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty  qtable_prop;
    AQQueueProperty       queue_prop;
    AQQueueTable          q_table;
    AQQueue                queue;

    qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");
    qtable_prop.setMultiConsumer(true);

    q_table = aq_sess.createQueueTable ("aq", "objmsgs_qtab", qtable_prop);

    /* 新しいAQQueueProperty オブジェクトを作成します。*/
```

```
        queue_prop = new AQQueueProperty();
        queue = aq_sess.createQueue (q_table, "q1def", queue_prop);
    }

    /* あるキューから同じデータベース内の他のキューに、伝播をスケジュールします。*/
    public static void example(AQSession aq_sess) throws AQException
    {
        AQQueue          queue;
        AQAgent          agent1;
        AQAgent          agent2;

        /* キュー・オブジェクトを取得します。*/
        queue = aq_sess.getQueue("AQ", "q1def");

        queue.schedulePropagation(null, null, null, null, null);
    }

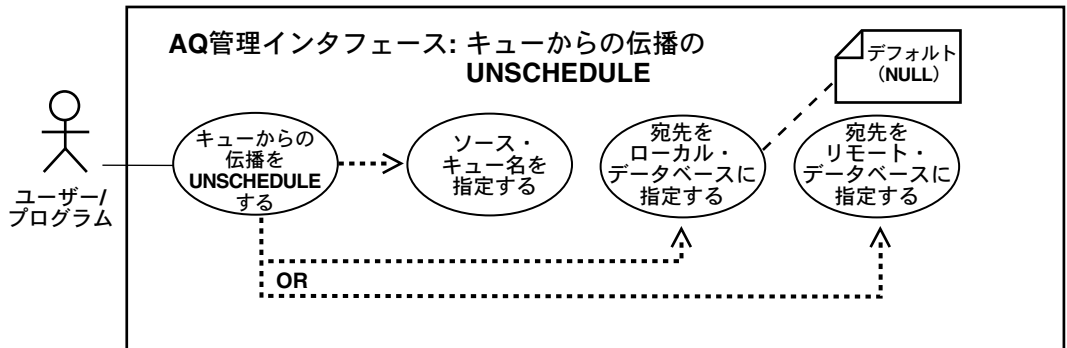
    /* Schedule propagation from a queue to other queues in another database */
    public static void example(AQSession aq_sess) throws AQException
    {
        AQQueue          queue;
        AQAgent          agent1;
        AQAgent          agent2;

        /* キュー・オブジェクトを取得します。*/
        queue = aq_sess.getQueue("AQ", "q1def");

        queue.schedulePropagation("another_db.world", null, null, null, null);
    }
}
```

## キューの伝播スケジュールの解除

図 9-20 ユースケース図：キューの伝播スケジュールの解除



### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル：管理インタフェース – 基本操作」を参照してください。

## 用途

あるキューから特定の dblink で識別される宛先に対して設定されていたメッセージ伝播スケジュールを解除します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第 5 章「DBMS\_AQADM」の UNSCHEDULE\_PROPAGATION プロシージャ
- Visual Basic (OO4O)：参照マニュアルはありません。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の schedulePropagation

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : 伝播スケジュールの解除](#) (9-70 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キューの伝播スケジュールの解除](#) (9-70 ページ)

## PL/SQL (DBMS\_AQADM) : 伝播スケジュールの解除

### あるキューから同じデータベース内の他のキューへの伝播スケジュールを解除する

```
/* キュー aq.q1def から同じデータベース内の他のキューへの伝播スケジュールを解除します。*/  
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(queue_name => 'aq.q1def');
```

### あるキューから別のデータベース内の他のキューへの伝播スケジュールを解除する

```
/* キュー aq.q1def から、データベース・リンク another_db.world の宛先である、別のデータベース内の他のキューへの伝播スケジュールを解除します。*/  
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(  
    Queue_name      => 'aq.q1def',  
    Destination     => 'another_db.world');
```

## Java (JDBC) : キューの伝播スケジュールの解除

```
/* あるキューから同じデータベース内の他のキューへの伝播スケジュールを解除します。*/  
public static void example(AQSession aq_sess) throws AQException  
{  
    AQQueue      queue;  
    AQAgent      agent1;  
    AQAgent      agent2;  
  
    /* キュー・オブジェクトを取得します。*/  
    queue = aq_sess.getQueue("AQ", "q1def");  
  
    queue.unschedulePropagation(null);  
}
```

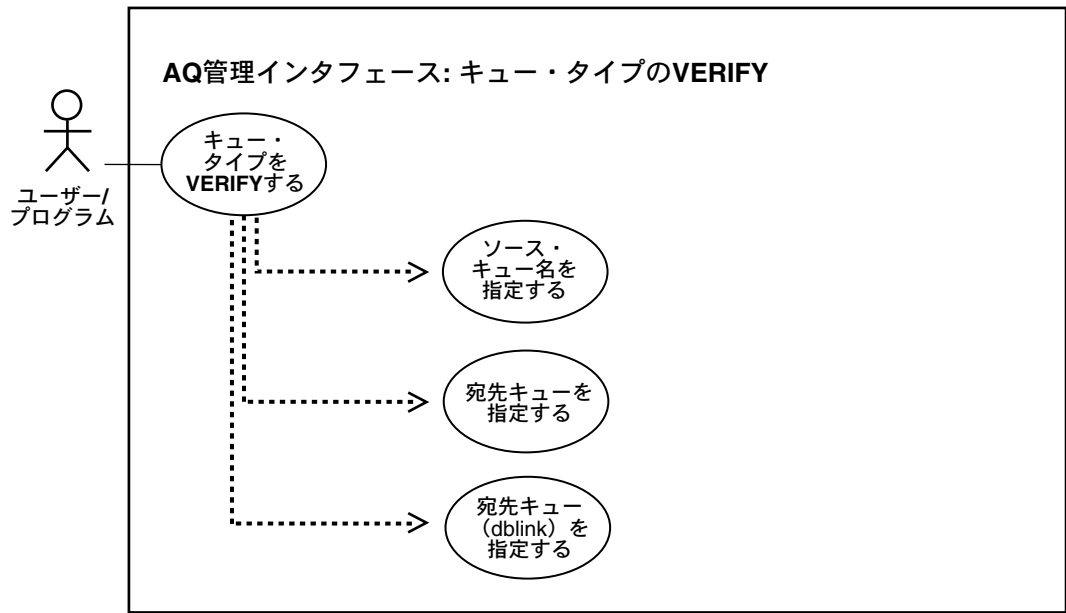
```
/* あるキューから別のデータベース内の他のキューへの伝播スケジュールを解除します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue      queue;
    AQAgent      agent1;
    AQAgent      agent2;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "qldef");

    queue.unschedulePropagation("another_db.world");
}
```

# キュー・タイプの検証

図 9-21 ユースケース図：キュー・タイプの検証



## 用途

ソースおよび宛先に同じ型のキューがないか検証します。検証の結果は、SYS.AQ\$\_MESSAGE\_TYPES 表に格納され、以前にこのコマンドから出力されたすべての結果は上書きされます。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。



- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の VERIFY\_QUEUE\_TYPES プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

## 例

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : キュー・タイプの検証](#) (9-73 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : キュー・タイプの検証](#) (9-74 ページ)

## PL/SQL (DBMS\_AQADM) : キュー・タイプの検証

---

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (
    queue_name      => 'aq.q2def',
    queue_table     => 'aq.objmsgs_qtab');
```

---

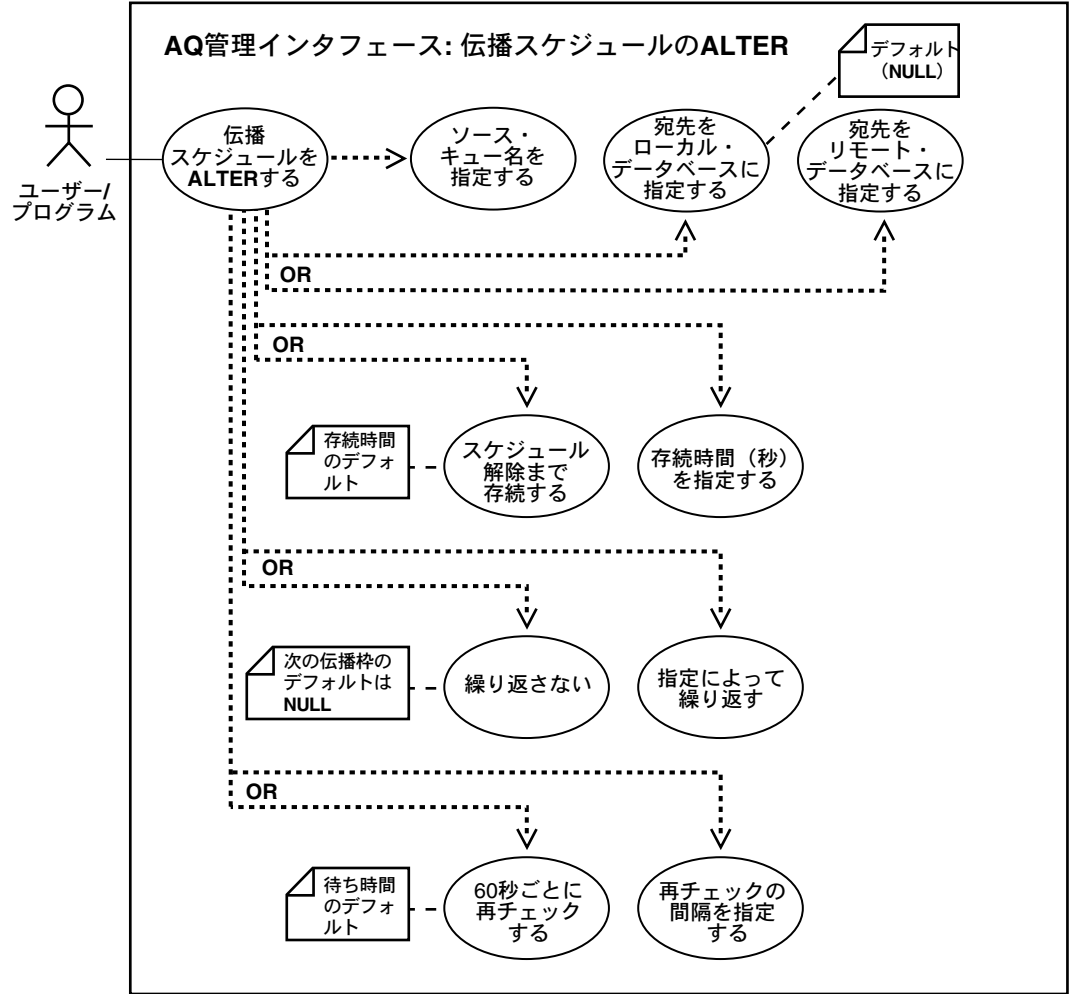
```
/* ソースおよび宛先キューの型が同じかどうかを検証します。検証の結果には、ディクショナリ表
AQ$_MESSAGE_TYPES のソースおよび宛先キューへのエントリの挿入 / 更新の副作用があります。*/
DECLARE
rc      BINARY_INTEGER;
BEGIN
/* ローカル・データベースのキュー aquser.q1def および aquser.q2def のペイロード型が同じかどうかを検証します。*/
    DBMS_AQADM.VERIFY_QUEUE_TYPES (
        src_queue_name => 'aq.q1def',
        dest_queue_name => 'aq.q2def',
        rc              => rc);
    DBMS_OUTPUT.PUT_LINE(rc);
END;
```

## Java（JDBC）：キュー・タイプの検証

この機能は、Java API を介しては使用できません。

# 伝播スケジュールの変更

図 9-22 ユースケース図：伝播スケジュールの変更



---

---

### 参照：

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:管理インタフェース – 基本操作](#)」を参照してください。
- 
- 

## 用途

伝播スケジュールのパラメータを変更します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の ALTER\_QUEUE\_TABLE プロシージャ
- Visual Basic (OO4O)：参照マニュアルはありません。
- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャリファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の alterPropagationSchedule

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\)：伝播スケジュールの変更](#) (9-77 ページ)
- VB (OO4O)：例はありません。
- [PL/SQL \(DBMS\\_AQADM\)：伝播スケジュールの変更](#) (9-77 ページ)

## PL/SQL (DBMS\_AQADM) : 伝播スケジュールの変更

### あるキューから同じデータベース内の他のキューへの伝播スケジュールを変更する

```
/* あるキューから同じデータベース内の他のキューへの伝播スケジュールを変更します。*/
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
    Queue_name    =>    'aq.q1def',
    Duration       =>    '2000',
    Next_time      =>    'SYSDATE + 3600/86400',
    Latency        =>    '32');
```

### あるキューから別のデータベース内の他のキューへの伝播スケジュールを変更する

```
/* キュー aq.q1def から、データベース・リンク another_db.world の宛先である、別のデータベース
内の他のキューへの伝播スケジュールを変更します。*/
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
    Queue_name    =>    'aq.q1def',
    Destination    =>    'another_db.world',
    Duration       =>    '2000',
    Next_time      =>    'SYSDATE + 3600/86400',
    Latency        =>    '32');
```

## Java (JDBC) : 伝播スケジュールの変更

```
/* あるキューから同じデータベース内の他のキューへの伝播スケジュールを変更します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue        queue;
    AQAgent        agent1;
    AQAgent        agent2;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "q1def");

    queue.alterPropagationSchedule(null, new Double(2000),
    "SYSDATE + 3600/86400", new Double(32));
}

/* あるキューから別のデータベース内の他のキューへの伝播スケジュールを解除します。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue        queue;
    AQAgent        agent1;
```

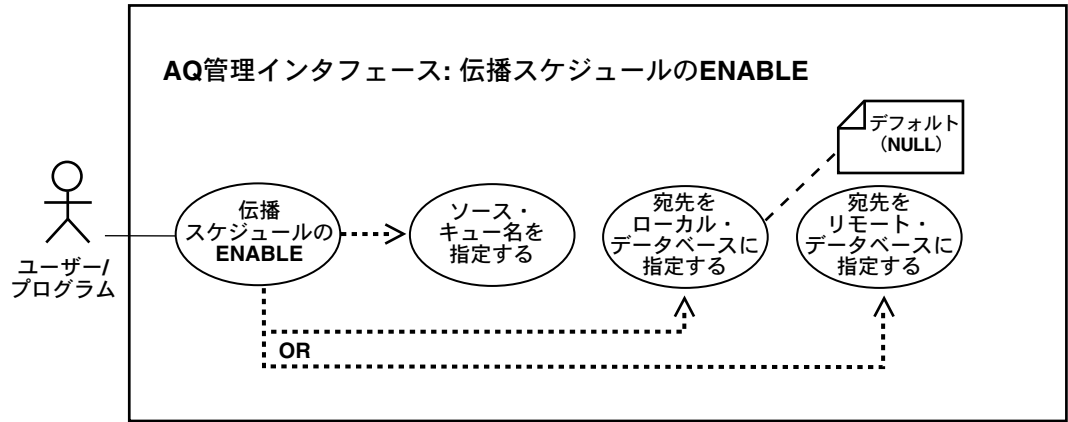
```
AQAgent          agent2;

/* キュー・オブジェクトを取得します。*/
queue = aq_sess.getQueue("AQ", "q1def");

queue.alterPropagationSchedule("another_db.world", new Double(2000),
"SYSDATE + 3600/86400", new Double(32));
}
```

# 伝播スケジュールの使用可能化

図 9-23 ユースケース図：伝播スケジュールの使用可能化



**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:管理インタフェース－基本操作」を参照してください。

## 用途

以前に使用不可にされた伝播スケジュールを使用可能にします。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャリファレンス』の第5章「DBMS\_AQADM」の ENABLE\_PROPAGATION\_SCHEDULE プロシージャ

- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueueAdmin」の enablePropagationSchedule

## 例

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。次のプログラム環境の例が示されています。

- PL/SQL (DBMS\_AQADM) : 伝播の使用可能化 (9-80 ページ)
- VB (OO4O) : 例はありません。
- Java (JDBC) : 伝播スケジュールの使用可能化 (9-80 ページ)

## PL/SQL (DBMS\_AQADM) : 伝播の使用可能化

### あるキューから同じデータベース内の他のキューへの伝播を使用可能にする

```
/* キュー aq.q1def から同じデータベース内の他のキューへの伝播を使用可能にします。*/  
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(  
    Queue_name => 'aq.q1def');
```

### あるキューから別のデータベース内の他のキューへの伝播を使用可能にする

```
/* キュー aq.q1def から、データベース・リンク another_db.world の宛先である、別のデータベース内の他のキューへの伝播を使用可能にします。*/  
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(  
    Queue_name => 'aq.q1def',  
    Destination => 'another_db.world');
```

## Java (JDBC) : 伝播スケジュールの使用可能化

```
/* あるキューから同じデータベース内の他のキューへの伝播を使用可能にします。*/  
public static void example(AQSession aq_sess) throws AQException  
{  
    AQQueue queue;  
    AQAgent agent1;  
    AQAgent agent2;  
  
    /* キュー・オブジェクトを取得します。*/  
    queue = aq_sess.getQueue("AQ", "q1def");
```



```
        queue.enablePropagationSchedule (null);
    }

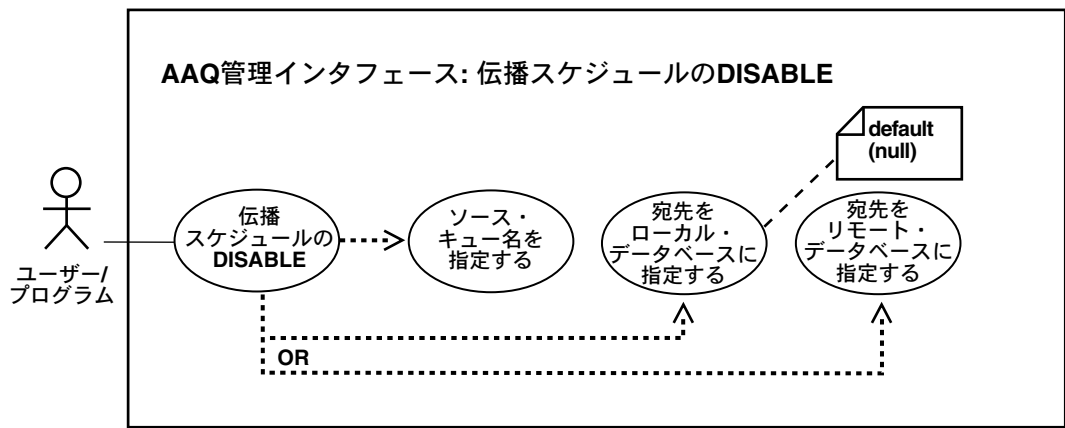
    /* あるキューから別のデータベース内の他のキューへの伝播を使用可能にします。*/
    public static void example(AQSession aq_sess) throws AQException
    {
        AQQueue      queue;
        AQAgent      agent1;
        AQAgent      agent2;

        /* キュー・オブジェクトを取得します。*/
        queue = aq_sess.getQueue("AQ", "q1def");

        queue.enablePropagationSchedule("another_db.world");
    }
```

# 伝播スケジュールの使用不可

図 9-24 ユースケース図：伝播スケジュールの使用不可



**参照：**

- 管理インタフェースに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:管理インタフェース – 基本操作」を参照してください。

## 用途

以前に使用可能にされた伝播スケジュールを使用不可にします。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQADM パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第 5 章「DBMS\_AQADM」の  
DISABLE\_PROPAGATION\_SCHEDULE プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 1 章  
「パッケージ oracle.AQ」の「AQQueueAdmin」の disablePropagationSchedule

## 例

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQADM\) : 伝播の使用可能化](#) (9-80 ページ)
- VB (OO4O) : 例はありません。
- [Java \(JDBC\) : 伝播スケジュールの使用可能化](#) (9-80 ページ)

## PL/SQL (DBMS\_AQADM) : 伝播の使用不可

### あるキューから同じデータベース内の他のキューへの伝播を使用不可にする

```
/* キュー aq.q1def から同じデータベース内の他のキューへの伝播を使用不可にします。*/
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
    Queue_name => 'aq.q1def');
```

### あるキューから別のデータベース内の他のキューへの伝播を使用不可にする

```
/* キュー aq.q1def から、データベース・リンク another_db.world の宛先である、別のデータベース
内の他のキューへの伝播を使用不可にします。*/
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
    Queue_name    => 'aq.q1def',
    Destination   => 'another_db.world');
```

## Java（JDBC）：伝播スケジュールの使用不可

```
/* あるキューから同じデータベース内の他のキューへの伝播を使用不可にします。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue      queue;
    AQAgent      agent1;
    AQAgent      agent2;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "q1def");

    queue.disablePropagationSchedule(null);
}

/* あるキューから別のデータベース内の他のキューへの伝播を使用不可にします。*/
public static void example(AQSession aq_sess) throws AQException
{
    AQQueue      queue;
    AQAgent      agent1;
    AQAgent      agent2;

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("AQ", "q1def");

    queue.disablePropagationSchedule("another_db.world");
}
```

---

## 管理インタフェース：ビュー

### ユースケース・モデル

この章では、それぞれの操作（データベース内のすべてのキュー・テーブルの選択など）を、その操作名ごとにユースケースの点から説明します。この章の先頭に、すべてのユースケースを示します（図 10-1「ユースケース・モデル：管理インタフェース－ビュー」を参照）。

### ユースケース・モデルの図形概要

図 10-1 では、すべてのユースケースを 1 つの図にまとめています。

### 個々のユースケース

個々のユースケースは、次のように配置されています。

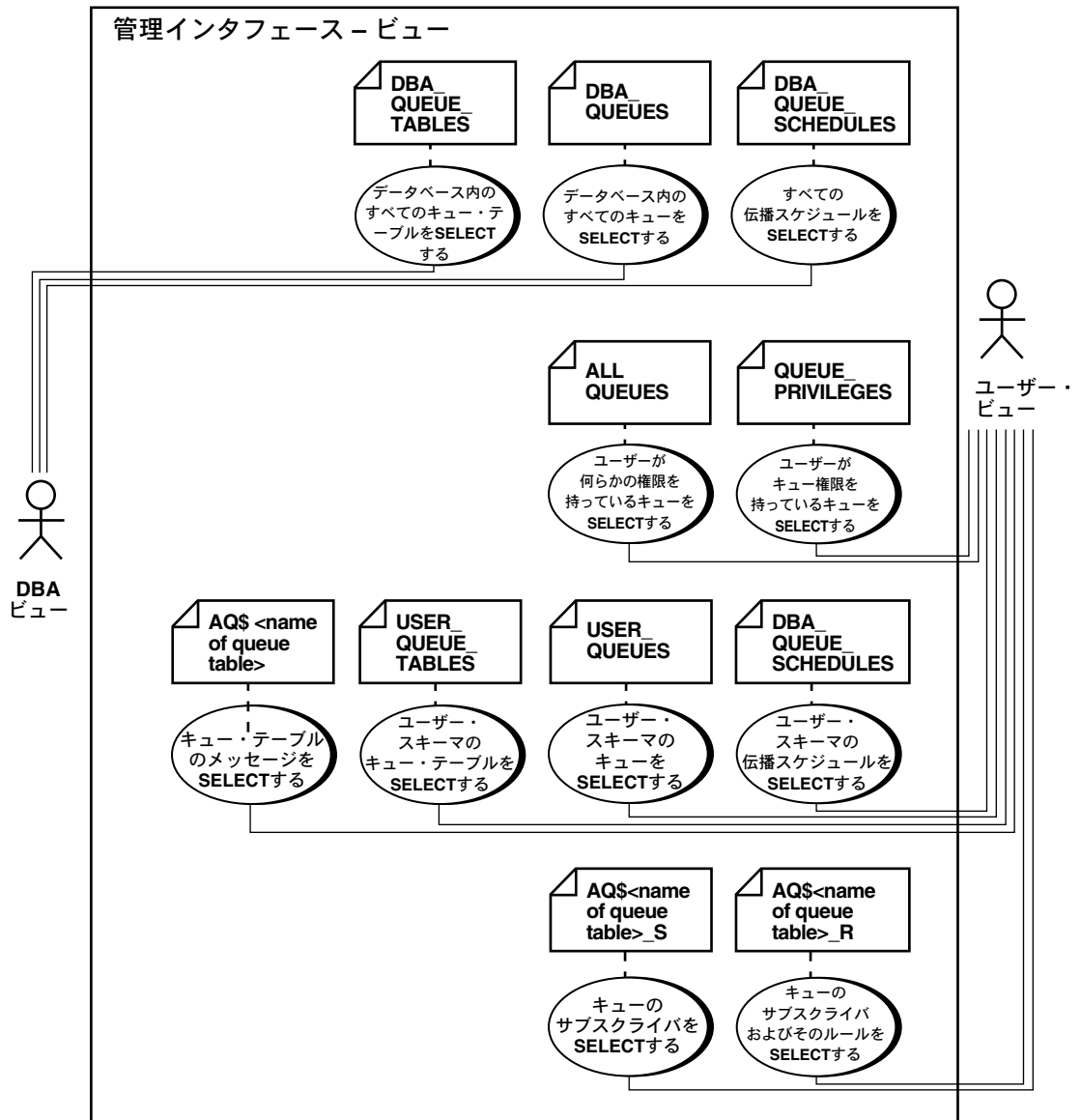
- **ユースケース図**：ユースケースを表す図（図の見方は、「はじめに」を参照）。ユースケースと状態図を組み合わせた複合図に沿ってビューの管理インタフェースを説明します。ユースケースでは、各ビューを代表的な操作（データベース内のすべてのキューの選択など）に沿って説明します。状態図では、各ビューの各属性の状態を表します。どの属性（列）も、状態は可視または不可視のいずれかです。
- **構文**：このアクティビティの実行に使用する構文。

# ユースケース・モデル: 管理インタフェース - ビュー

表 10-1 ユースケース・モデル: 管理インタフェース - ビュー

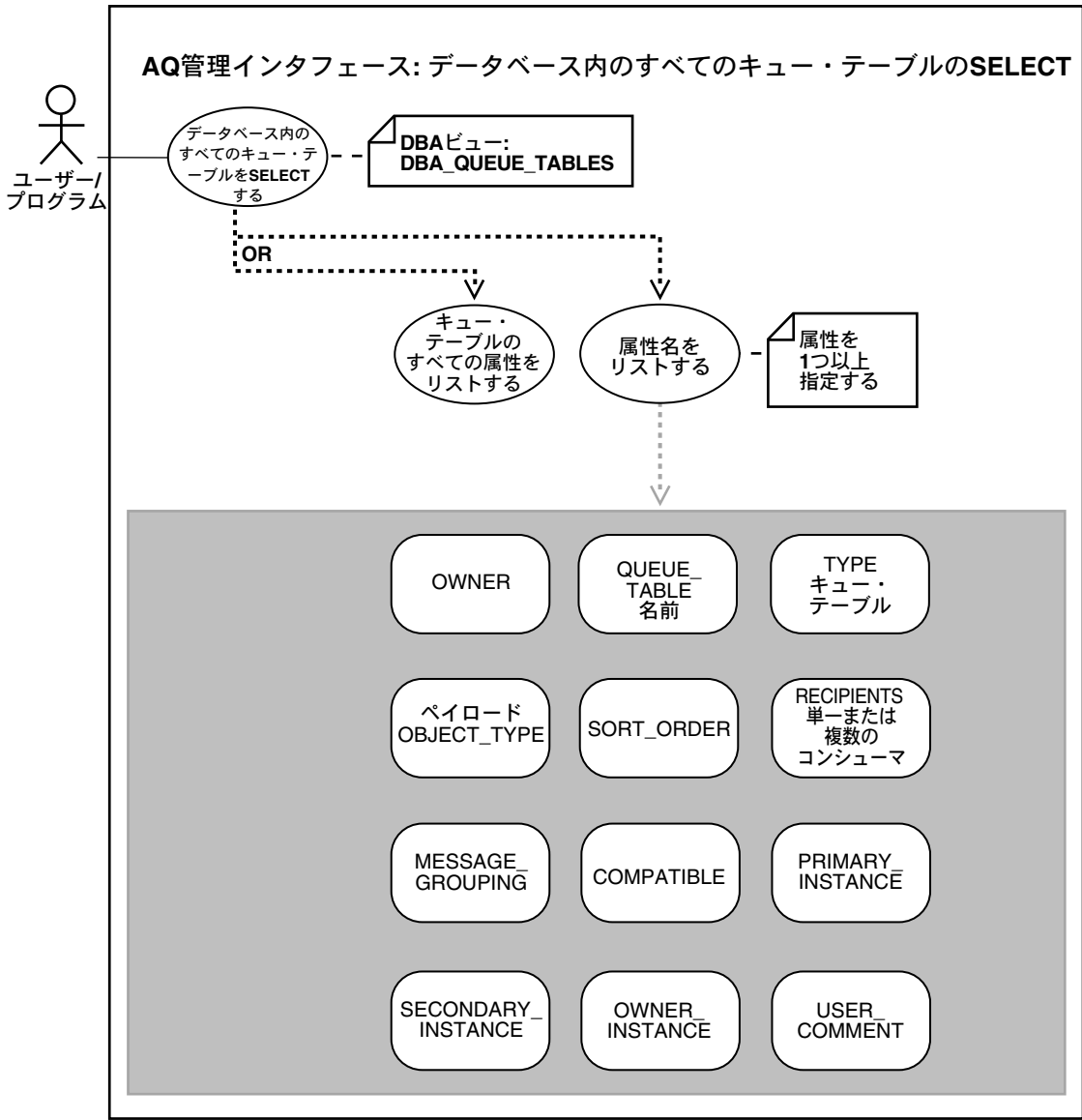
ユースケース	ビュー名
10-4 ページの「データベース内のすべてのキュー・テーブルの選択」	DBA_QUEUE_TABLES
10-7 ページの「ユーザーのキュー・テーブルの選択」	ALL_QUEUE_TABLES
10-10 ページの「データベース内のすべてのキューの選択」	DBA_QUEUES
10-12 ページの「すべての伝播スケジュールの選択」	DBA_QUEUE_SCHEDULES
10-17 ページの「ユーザーが何らかの権限を持っているキューの選択」	ALL_QUEUES
10-19 ページの「ユーザーがキュー権限を持っているキューの選択」	QUEUE_PRIVILEGES
10-21 ページの「キュー・テーブルのメッセージの選択」	AQ\$<name of queue table>
10-25 ページの「ユーザー・スキーマのキュー・テーブルの選択」	USER_QUEUE_TABLES
10-28 ページの「ユーザー・スキーマのキューの選択」	USER_QUEUES
10-30 ページの「ユーザー・スキーマの伝播スケジュールの選択」	USER_QUEUE_SCHEDULES
10-35 ページの「キューのサブスクライバの選択」	AQ\$<name of queue table>_S
10-37 ページの「キューのサブスクライバおよびそのルール of の選択」	AQ\$<name of queue table>_R
10-39 ページの「データベース全体における状態ごとのメッセージ数の選択」	GV\$AQ
10-41 ページの「特定のインスタンスにおける状態ごとのメッセージ数の選択」	V\$AQ

図 10-1 ユースケース・モデル：管理インタフェース - ビュー



# データベース内のすべてのキュー・テーブルの選択

図 10-2 ユースケース図：データベース内のすべてのキュー・テーブルの選択





参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース – ビュー」を参照してください。

ビュー名

DBA\_QUEUE\_TABLES

用途

このビューは、データベース内で作成されたすべてのキュー・テーブルの名前および型を示します。

表 10-2 DBA\_QUEUE\_TABLES

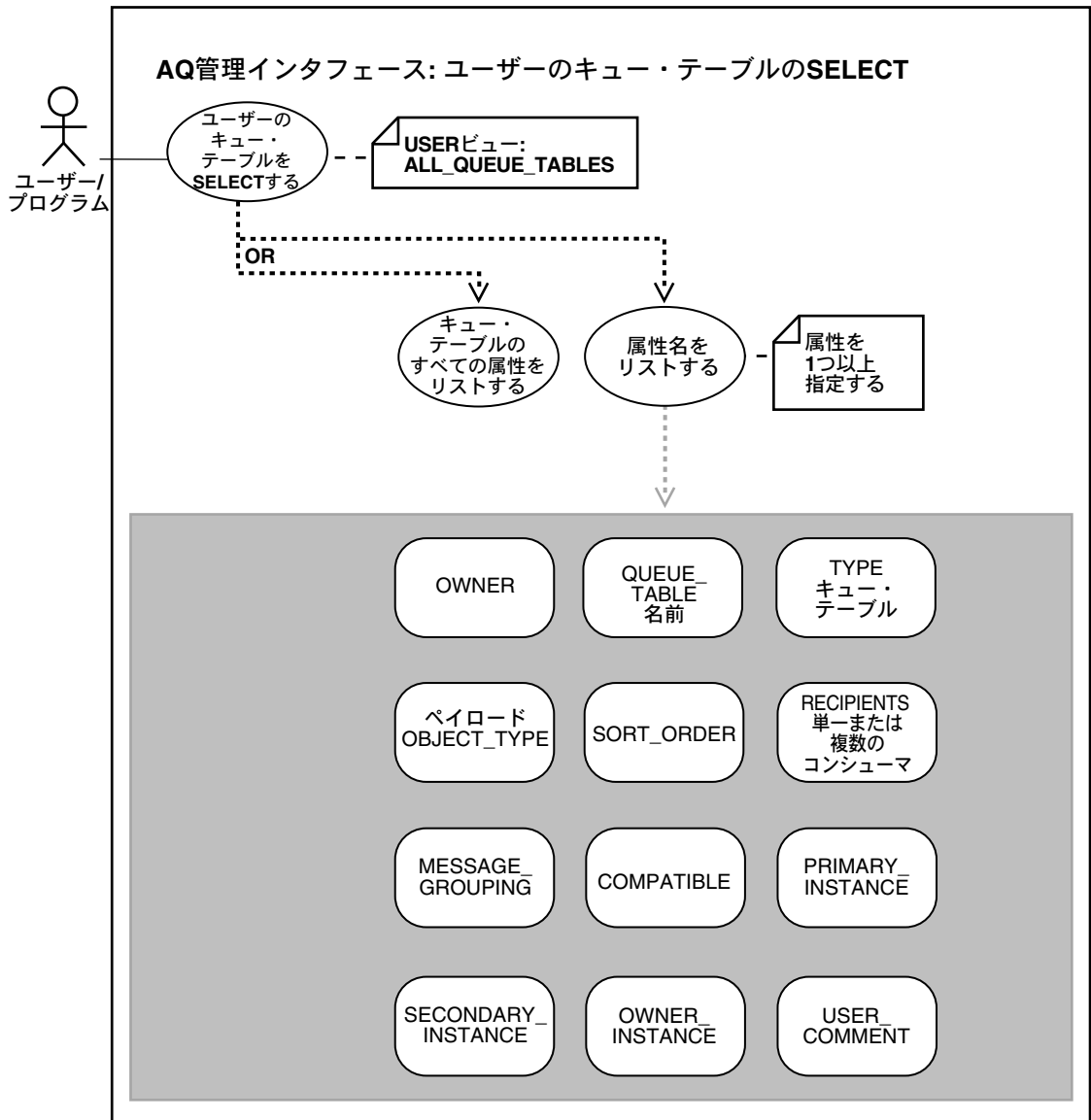
列名および説明	Null かどうか	型
OWNER - キュー・テーブル・スキーマ		VARCHAR2 (30)
QUEUE_TABLE - キュー・テーブル名		VARCHAR2 (30)
TYPE - ペイロードの型		VARCHAR2 (7)
OBJECT_TYPE - オブジェクト型が定義されている場合、その型名		VARCHAR2 (61)
SORT_ORDER - ユーザー指定のソート順序		VARCHAR2 (22)
RECIPIENTS - SINGLE または MULTIPLE		VARCHAR2 (8)
MESSAGE_GROUPING - NONE または TRANSACTIONAL		VARCHAR2 (13)
COMPATIBLE - そのキュー・テーブルが互換性を持つ最も低いバージョン		VARCHAR2 (5)
PRIMARY_INSTANCE - そのキュー・テーブルの 1 次所有者になっているインスタンス。値が 0 (ゼロ) のときは、1 次所有者が指定されていません。		NUMBER

表 10-2 DBA\_QUEUE\_TABLES

列名および説明	Null かどうか	型
SECONDARY_INSTANCE - そのキュー・テーブルの 2 次所有者になっているインスタンスで、1 次所有者がいないときにキュー・テーブルの所有者になります。値が 0（ゼロ）のときは、2 次所有者が指定されていません。		NUMBER
OWNER_INSTANCE - 現在のキュー表の所有者インスタンス		NUMBER
USER_COMMENT - キュー・テーブルに対するユーザー・コメント		VARCHAR2 (50)

# ユーザーのキュー・テーブルの選択

図 10-3 ユースケース図：ユーザーのキュー・テーブルの選択



参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「[ユースケース・モデル:管理インタフェース - ビュー](#)」を参照してください。

ビュー名

ALL\_QUEUE\_TABLES

用途

このビューは、ユーザーがアクセスできるキュー・テーブルを示します。

表 10-3 DBA\_QUEUE\_TABLES

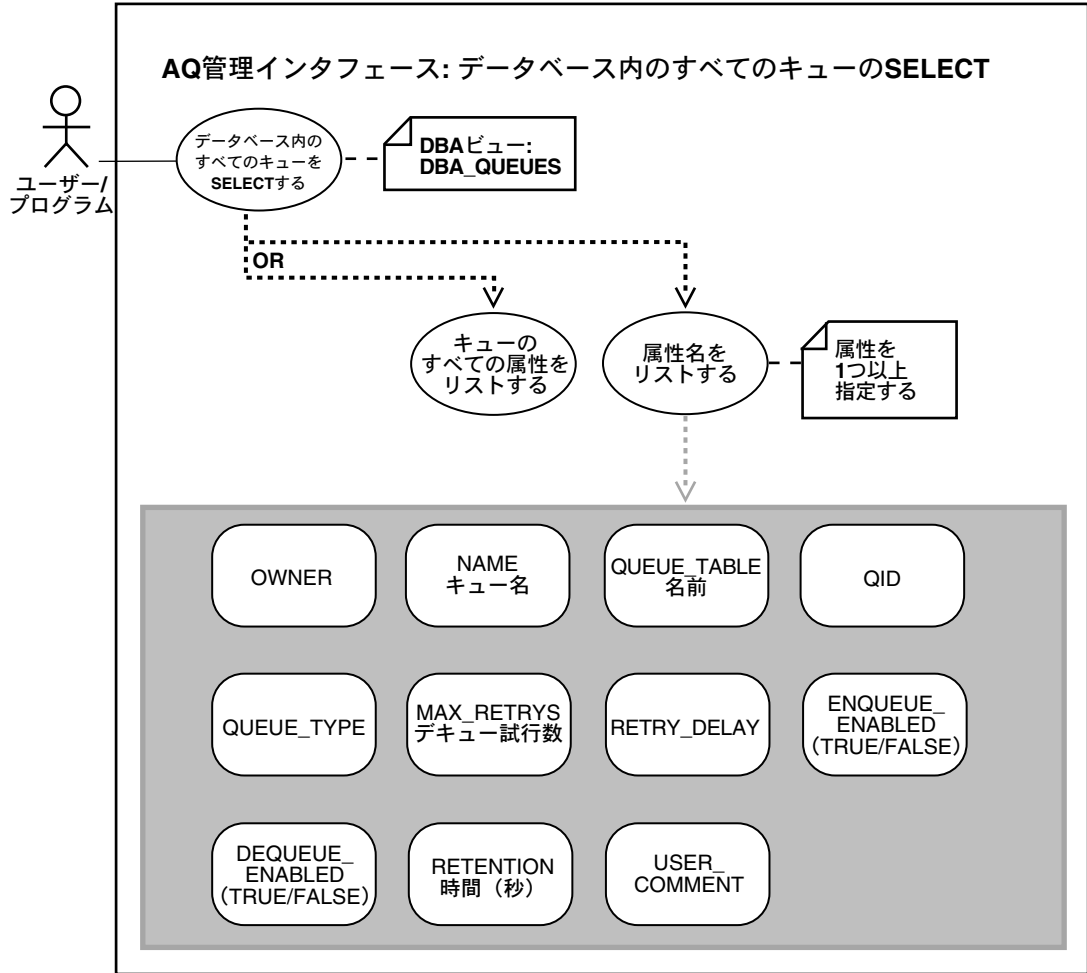
列名および説明	Null かどうか	型
OWNER - キュー・テーブルの所有者		VARCHAR2 (30)
QUEUE_TABLE - キュー・テーブル名		VARCHAR2 (30)
TYPE - ペイロードの型		VARCHAR2 (7)
OBJECT_TYPE - オブジェクト型が定義されている場合、その型名		VARCHAR2 (61)
SORT_ORDER - ユーザー指定のソート順序		VARCHAR2 (22)
RECIPIENTS - SINGLE または MULTIPLE		VARCHAR2 (8)
MESSAGE_GROUPING - NONE または TRANSACTIONAL		VARCHAR2 (13)
COMPATIBLE - そのキュー・テーブルが互換性を持つ最も低いバージョン		VARCHAR2 (5)
PRIMARY_INSTANCE - そのキュー・テーブルの 1 次所有者になっているインスタンス。値が 0 (ゼロ) のときは、1 次所有者が指定されていません。		NUMBER

表 10-3 DBA\_QUEUE\_TABLES

列名および説明	Null かどうか	型
SECONDARY_INSTANCE - そのキュー・テーブルの 2 次所有者になっているインスタンスで、1 次所有者がいないときにキュー・テーブルの所有者になります。値が 0（ゼロ）のときは、2 次所有者が指定されていません。		NUMBER
OWNER_INSTANCE - 現在のキュー・テーブルの所有者インスタンス		NUMBER
USER_COMMENT - キュー・テーブルに対するユーザー・コメント		VARCHAR2 (50)

# データベース内のすべてのキューの選択

図 10-4 ユースケース図：データベース内のすべてのキューの選択



参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース – ビュー」を参照してください。

ビュー名

DBA\_QUEUES

用途

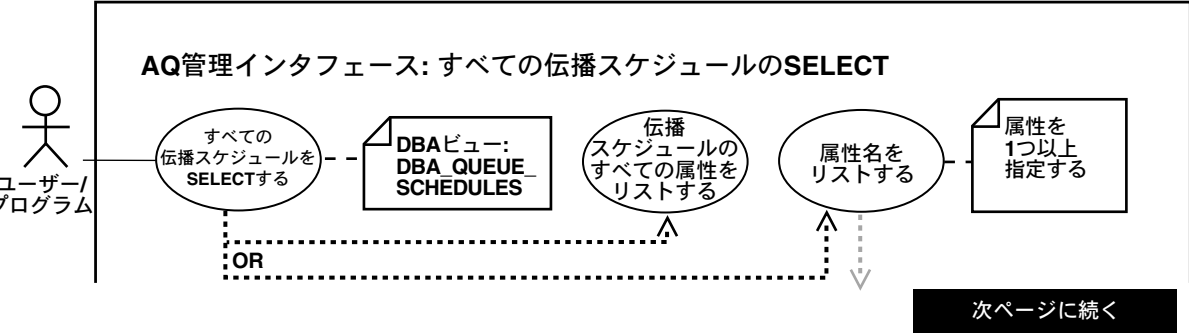
ユーザーは、個々のキューについて操作上の特性を指定できます。DBA\_QUEUES には、データベース内のすべてのキューについての関連情報を含むビューが含まれます。

表 10-4 DBA\_QUEUES

列名および説明	Null かどうか	型
OWNER - キューのスキーマ名	NOT NULL	VARCHAR2 (30)
NAME - キュー名	NOT NULL	VARCHAR2 (30)
QUEUE_TABLE - このキューが格納される キュー・テーブル	NOT NULL	VARCHAR2 (30)
QID - 一意のキュー識別子	NOT NULL	NUMBER
QUEUE_TYPE - キューの型		VARCHAR2 (15)
MAX_RETRIES - デキューの許容試行回数		NUMBER
RETRY_DELAY - 再試行までの秒数		NUMBER
ENQUEUE_ENABLED - YES/NO		VARCHAR2 (7)
ENQUEUE_ENABLED - YES/NO		VARCHAR2 (7)
RETENTION - メッセージがデキュー後に保存さ れる秒数		VARCHAR2 (40)
USER_COMMENT - キューに対するユーザー・コ メント		VARCHAR2 (50)

# すべての伝播スケジュールの選択

図 10-5 ユースケース図：すべての伝播スケジュールの選択







### 参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース - ビュー」を参照してください。

ビュー名

DBA\_QUEUE\_SCHEDULES

用途

このビューは、現在のメッセージ伝播スケジュールを示します。

表 10-5 DBA\_QUEUE\_SCHEDULES

列名および説明	Null かどうか	型
SCHEMA - ソース・キューのスキーマ名	NOT NULL	VARCHAR2 (30)
QNAME - ソース・キューの名前	NOT NULL	VARCHAR2 (30)
DESTINATION - 宛先名。現在は DBLINK 名に制限されています。	NOT NULL	VARCHAR2 (128)
START_DATE - デフォルトの日付書式による伝播の開始日付		DATE
START_TIME - HH:MI:SS 形式による伝播の開始時刻		VARCHAR2 (8)
PROPAGATION_WINDOW - 伝播枠の存続期間 (秒)		NUMBER
NEXT_TIME - 次の伝播枠の開始を計算する関数		VARCHAR2 (200)
LATENCY - 伝播枠内で、メッセージを伝播するまでの最大待機時間		NUMBER
SCHEDULE_DISABLED - 使用可能のときは N、使用不可でスケジュールが実行されていないときは Y		VARCHAR (1)
PROCESS_NAME - このスケジュールを実行する SNP バックグラウンド・プロセス。実行されていないときは NULL になります。		VARCHAR2 (8)
SESSION_ID - このスケジュールを実行しているジョブのセッション ID (SID, SERIAL#)。実行されていないときは NULL になります。		NUMBER

表 10-5 DBA\_QUEUE\_SCHEDULES (続き)

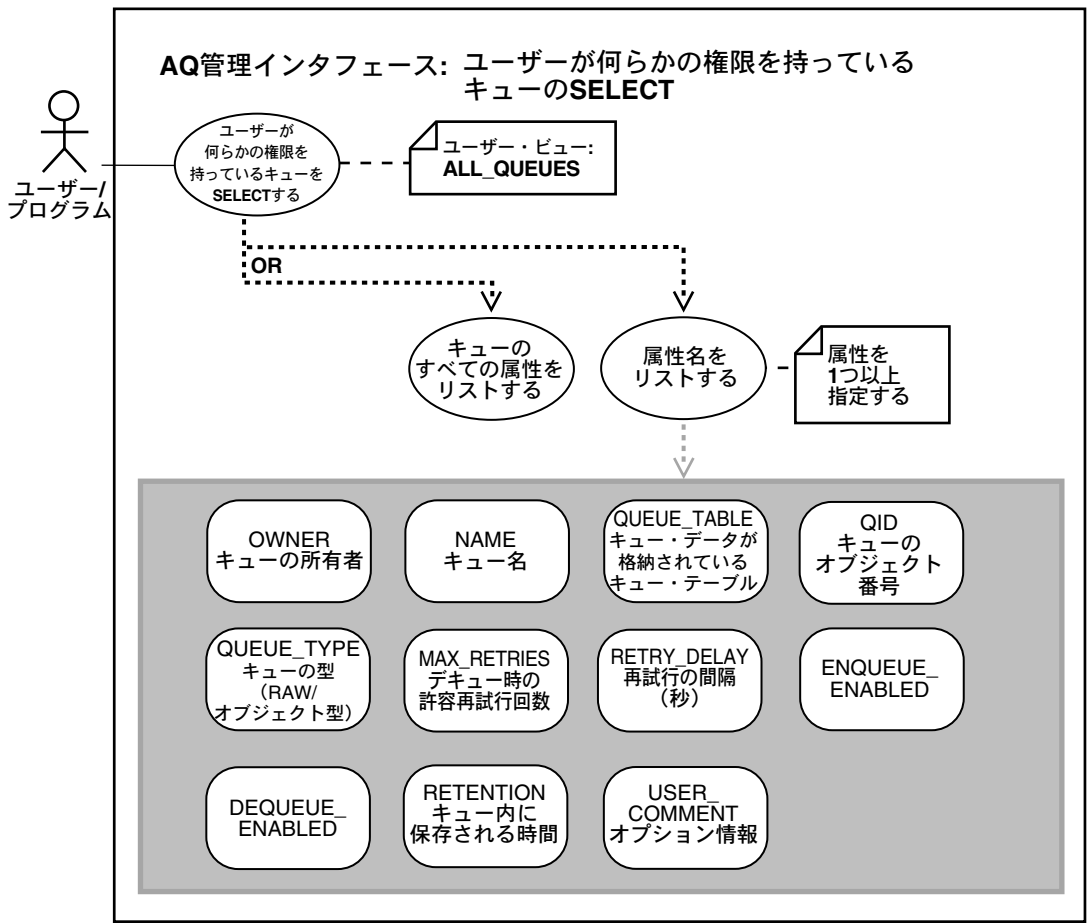
列名および説明	Null かどうか	型
INSTANCE - このスケジュールを実行している OPS インスタンス番号		NUMBER
LAST_RUN_DATE - 最後に正常に実行された日 付		DATE
LAST_RUN_TIME - 最後に正常に実行された時 刻 (HH:MI:SS 形式)		VARCHAR2 (8)
CURRENT_START_DATE - このスケジュール枠が 開始された日付		DATE
CURRENT_START_TIME - このスケジュール枠 が開始された時刻 (HH:MI:SS 形式)		VARCHAR2 (8)
NEXT_RUN_DATE - 次のスケジュール枠が開始 される日付		DATE
NEXT_RUN_TIME - 次のスケジュール枠が開始 される時刻 (HH:MI:SS 形式)		VARCHAR2 (8)
TOTAL_TIME - このスケジュールでメッセージ の伝播に費やされた総時間 (秒)		NUMBER
TOTAL_NUMBER - このスケジュールで伝播され た総メッセージ数		NUMBER
TOTAL_BYTES - このスケジュールで伝播された 総バイト数		NUMBER
MAX_NUMBER - 伝播枠で伝播された最大メッ セージ数		NUMBER
MAX_BYTES - 伝播枠で伝播された最大バイト数		NUMBER
AVG_NUMBER - 伝播枠で伝播された平均メッ セージ数		NUMBER

表 10-5 DBA\_QUEUE\_SCHEDULES (続き)

列名および説明	Null かどうか	型
AVG_SIZE - 伝播されたメッセージの平均サイズ (バイト)		NUMBER
AVG_TIME - 1つのメッセージ伝播にかかる平均時間 (秒)		NUMBER
FAILURES - 実行が失敗した回数。16 になった場合、そのスケジュールは使用禁止になります。		NUMBER
LAST_ERROR_DATE - 最後に実行に失敗した日付		DATE
LAST_ERROR_DATE - 最後に実行に失敗した時刻		VARCHAR2 (8)
LAST_ERROR_MSG - 最後に実行に失敗したときの、エラー番号およびエラー・メッセージ		VARCHAR2 (4000)

# ユーザーが何らかの権限を持っているキューの選択

図 10-6 ユースケース図：ユーザーが何らかの権限を持っているキューの選択



**参照：**

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「[ユースケース・モデル：管理インタフェース - ビュー](#)」を参照してください。

ビュー名

ALL\_QUEUES

用途

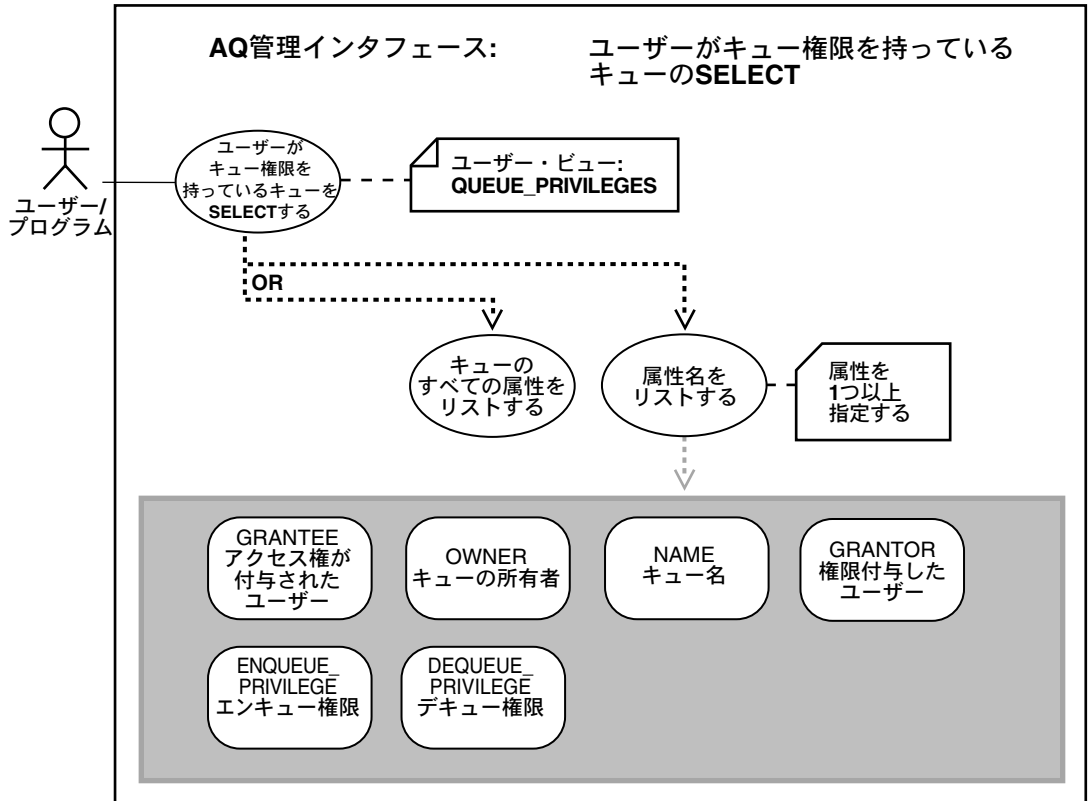
このビューは、ユーザーがアクセスできるすべてのキューを示します。

表 10-6 ALL\_QUEUES

列名および説明	Null かどうか	型
OWNER - キューの所有者	NOT NULL	VARCHAR2 (30)
NAME - キュー名	NOT NULL	VARCHAR2 (30)
QUEUE_TABLE - そのキュー・データが格納されているキュー・テーブルの名前	NOT NULL	VARCHAR2 (30)
QID - そのキューのオブジェクト番号	NOT NULL	NUMBER
QUEUE_TYPE - キューの型		VARCHAR2 (15)
MAX_RETRIES - そのキューからデキューするときの許容試行回数		NUMBER
RETRY_DELAY - 再試行の間隔		NUMBER
ENQUEUE_ENABLED - キューにエンキュー可能		VARCHAR2 (7)
DEQUEUE_ENABLED - キューからデキュー可能		VARCHAR2 (7)
RETENTION - 処理済メッセージがキュー内に保存される時間		VARCHAR2 (40)
USER_COMMENT - ユーザー指定のコメント		VARCHAR2 (50)

## ユーザーがキュー権限を持っているキューの選択

図 10-7 ユースケース図：ユーザーがキュー権限を持っているキューの選択



### 参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル：管理インタフェース - ビュー」を参照してください。

### ビュー名

QUEUE\_PRIVILEGES

用途

このビューは、ユーザーが権限付与者、権限受領者、所有者または使用可能なロールになっているキュー、あるいは PUBLIC に付与されているキューを示します。

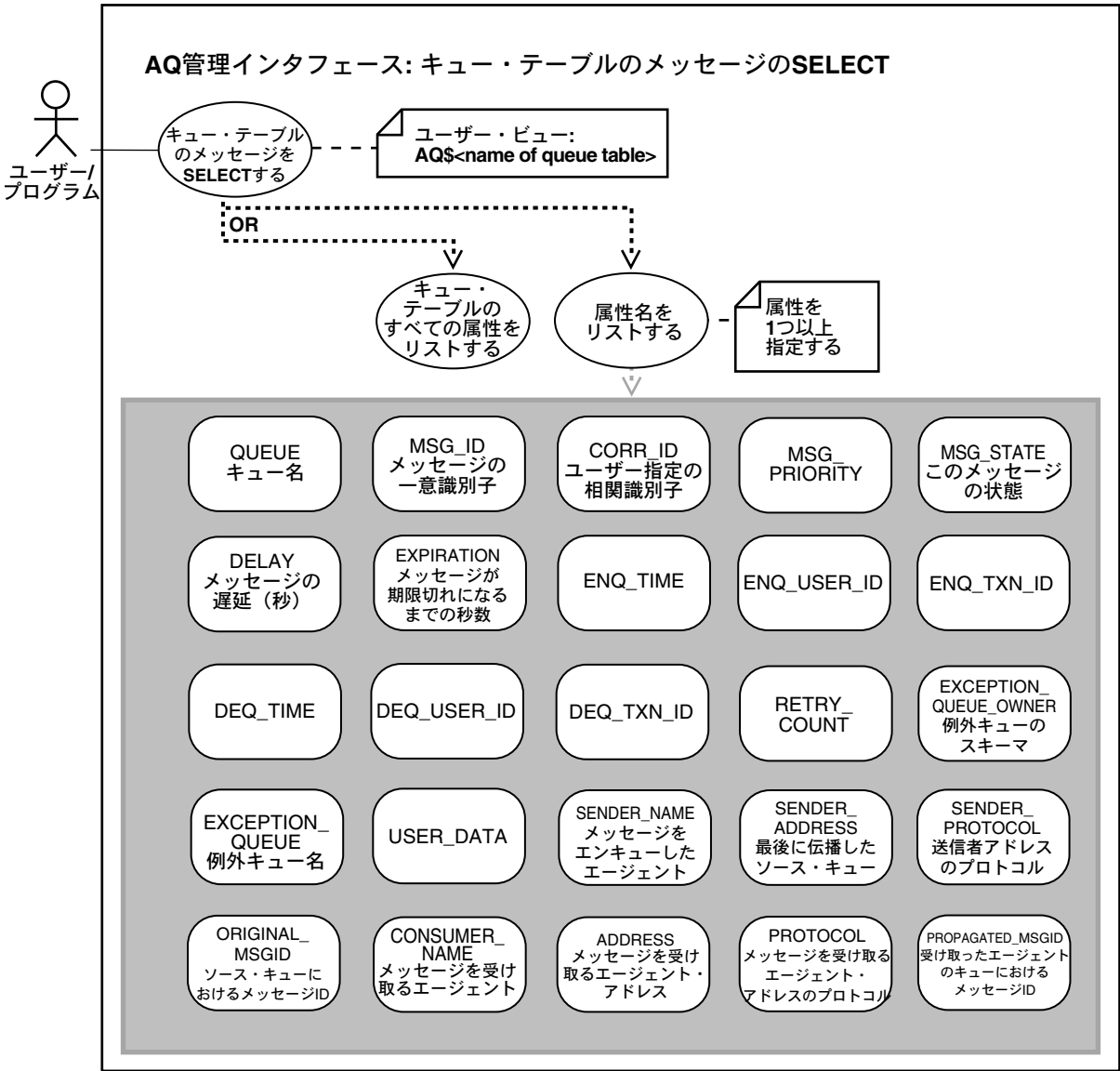
表 10-7 QUEUE\_PRIVILEGES

列名および説明	Null かどうか	型
GRANTEE - アクセス権限が付与されたユーザー	NOT NULL	VARCHAR2 (30)
OWNER - キューの所有者	NOT NULL	VARCHAR2 (30)
NAME - キュー名	NOT NULL	VARCHAR2 (30)
GRANTOR - 権限付与を実行したユーザー	NOT NULL	VARCHAR2 (30)
ENQUEUE_PRIVILEGE - そのキューに対する ENQUEUE 権限		NUMBER (付与されているときは 1、付与されていないときは 0 (ゼロ))
ENQUEUE_PRIVILEGE - そのキューに対する ENQUEUE 権限		NUMBER (付与されているときは 1、付与されていないときは 0 (ゼロ))



# キュー・テーブルのメッセージの選択

図 10-8 ユースケース図：キュー・テーブルのメッセージの選択



参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース - ビュー」を参照してください。

ビュー名

AQ\$<name of queue table>

用途

このビューは、メッセージ・データが格納されているキュー・テーブルを示します。このビューはキュー・テーブルごとに自動的に作成され、キュー・データの間合せに使用されます。デキュー履歴データ（時間、ユーザー ID およびトランザクション ID）は、単一コンシューマ・キューについてのみ有効です。

表 10-8 AQ\$<name of queue table>

列名および説明	Null かどうか	型
NAME - キュー名		VARCHAR2 (30)
MSG_ID - メッセージの一意識別子		RAW (16)
CORR_ID - ユーザー指定の関連識別子		VARCHAR2 (128)
MSG_PRIORITY - メッセージの優先順位		NUMBER
MSG_STATE - このメッセージの状態		VARCHAR2 (9)
DELAY - メッセージが遅延処理されるまでの秒数		DATE
EXPIRATION - メッセージが READY 状態になってから、期限切れになるまでの秒数		NUMBER
ENQ_TIME - エンキュー時刻		DATE
ENQ_USER_ID - エンキューしたユーザーのユーザー ID		NUMBER
ENQ_TXN_ID - エンキューしたトランザクションのトランザクション ID	NOT NULL	VARCHAR2 (30)
DEQ_TIME - デキュー時刻		DATE
DEQ_USER_ID - デキューしたユーザーのユーザー ID		NUMBER

表 10-8 AQ\$&lt;name of queue table&gt; (続き)

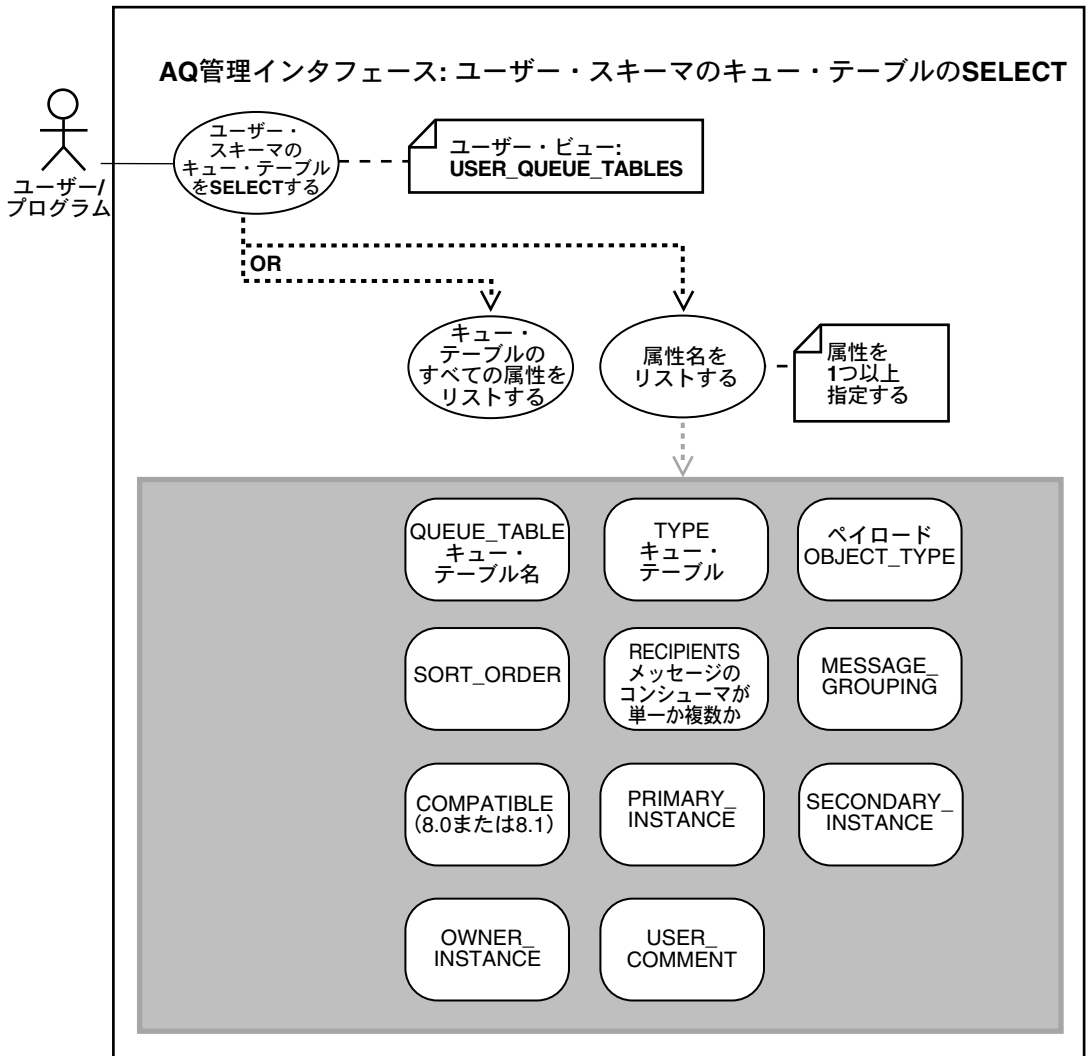
列名および説明	Null かどうか	型
DEQ_TXN_ID - デキューしたトランザクションのトランザクション ID		VARCHAR2 (30)
RETRY_COUNT - 再試行回数		NUMBER
EXCEPTION_QUEUE_OWNER - 例外キューのスキーマ		VARCHAR2 (30)
EXCEPTION_QUEUE - 例外キュー名		VARCHAR2 (30)
USER_DATA - ユーザー・データ		BLOB
SENDER_NAME - メッセージをエンキューしたエージェント名 (8.1 互換のキュー・テーブルでのみ有効)		VARCHAR2 (30)
SENDER_ADDRESS - 最後に伝播を行ったソース・キュー名およびデータベース名。ソース・キューがローカル・データベースにあるときは、データベース名は指定されません (8.1 互換のキュー・テーブルでのみ有効)。		VARCHAR2 (1024)
SENDER_PROTOCOL - 送信者アドレスのプロトコル。将来の使用のために確保されています (8.1 互換のキュー・テーブルでのみ有効)。		NUMBER
ORIGINAL_MSGID - ソース・キューにおけるメッセージ ID (8.1 互換のキュー・テーブルでのみ有効)		RAW (16)
CONSUMER_NAME - メッセージを受け取るエージェント名 (8.1 互換の複数コンシューマ・キュー表でのみ有効)		VARCHAR2 (30)
ADDRESS - メッセージを受け取るエージェントのアドレス (キュー名およびデータベース・リンク名)。ソース・キューがローカルデータベースにあるときは、データベース・リンク名は指定されません。そのキューのローカル・エージェントがメッセージを受け取る時、アドレスは NULL になります (8.1 互換の複数コンシューマ・キュー表でのみ有効)。		VARCHAR2 (1024)

表 10-8 AQ\$<name of queue table> (続き)

列名および説明	Null かどうか	型
PROTOCOL - メッセージを受け取るエージェント・アドレスのプロトコル (8.1 互換のキュー・テーブルでのみ有効)		NUMBER
PROPAGATED_MSGID - 受け取ったエージェントのキューにおけるメッセージ ID (8.1 互換のキュー・テーブルでのみ有効)	NULL	RAW (16)

## ユーザー・スキーマのキュー・テーブルの選択

図 10-9 ユースケース図：ユーザー・スキーマのキュー・テーブルの選択



参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース – ビュー」を参照してください。

ビュー名

USER\_QUEUE\_TABLES

構文

このビューは、ユーザー・スキーマ内のキュー・テーブルのみを表示する点を除いて、DBA\_QUEUE\_TABLES と同じです。このビューには、OWNER の列は含まれません。

表 10-9 USER\_QUEUE\_TABLES

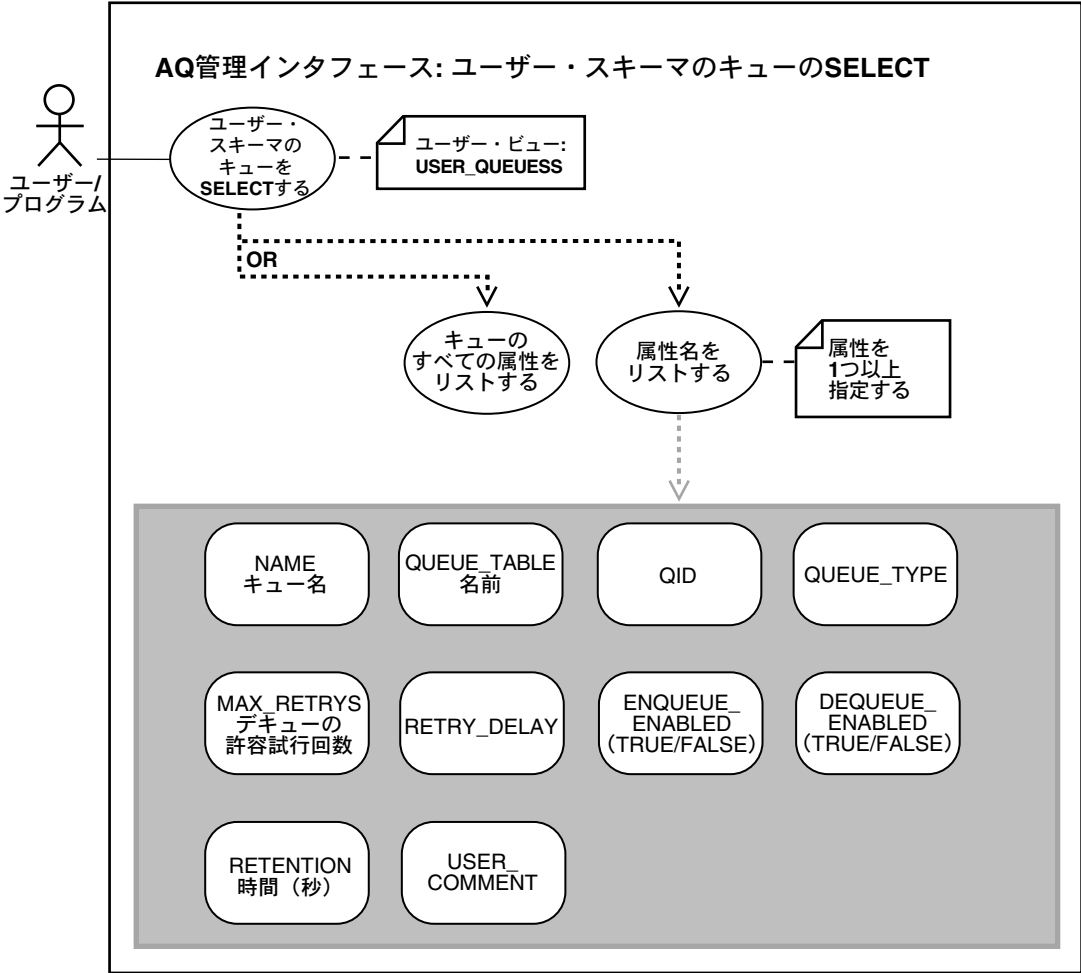
列名および説明	Null かどうか	型
QUEUE_TABLE - キュー・テーブル名		VARCHAR2 (30)
TYPE - ペイロードの型		VARCHAR2 (7)
OBJECT_TYPE - オブジェクト型が定義されている場合、その型名		VARCHAR2 (61)
SORT_ORDER - ユーザー指定のソート順		VARCHAR2 (22)
RECIPIENTS - SINGLE または MULTIPLE		VARCHAR2 (8)
MESSAGE_GROUPING - NONE または TRANSACTIONAL		VARCHAR2 (13)
COMPATIBLE - そのキュー・テーブルが互換性を持つ最も低いバージョン		VARCHAR2 (5)
PRIMARY_INSTANCE - そのキュー・テーブルの 1 次所有者になっているインスタンス。値が 0（ゼロ）のときは、1 次所有者が指定されていません。		NUMBER

表 10-9 USER\_QUEUE\_TABLES (続き)

列名および説明	Null かどうか	型
SECONDARY_INSTANCE - そのキュー・テーブルの 2 次所有者になっているインスタンスで、1 次所有者がいないときにキュー・テーブルの所有者になります。値が 0 (ゼロ) のときは、2 次所有者が指定されていません。		NUMBER
OWNER_INSTANCE - 現在のキュー・テーブルの所有者インスタンス		NUMBER
USER_COMMENT - キュー・テーブルに対するユーザー・コメント		VARCHAR2 (50)

# ユーザー・スキーマのキューの選択

図 10-10 ユースケース図：ユーザー・スキーマのキューの選択





参照:

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「[ユースケース・モデル:管理インタフェース - ビュー](#)」を参照してください。

ビュー名

USER\_QUEUES

用途

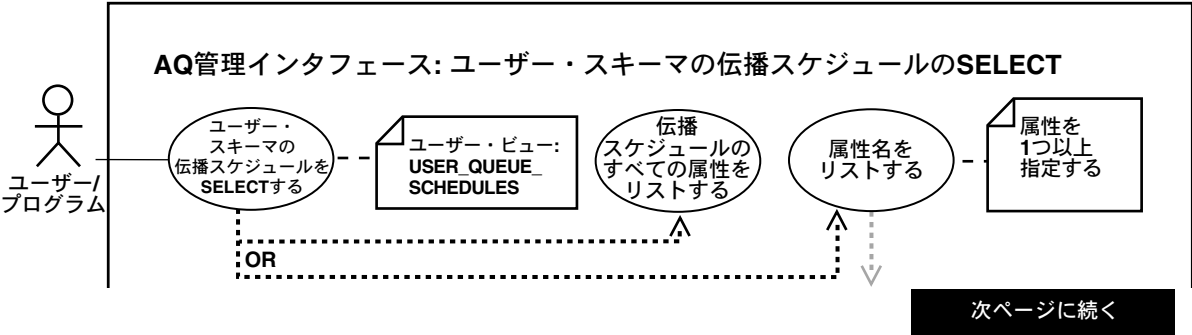
このビューは、ユーザー・スキーマ内のキューのみを表示する点を除いて、DBA\_QUEUES と同じです。

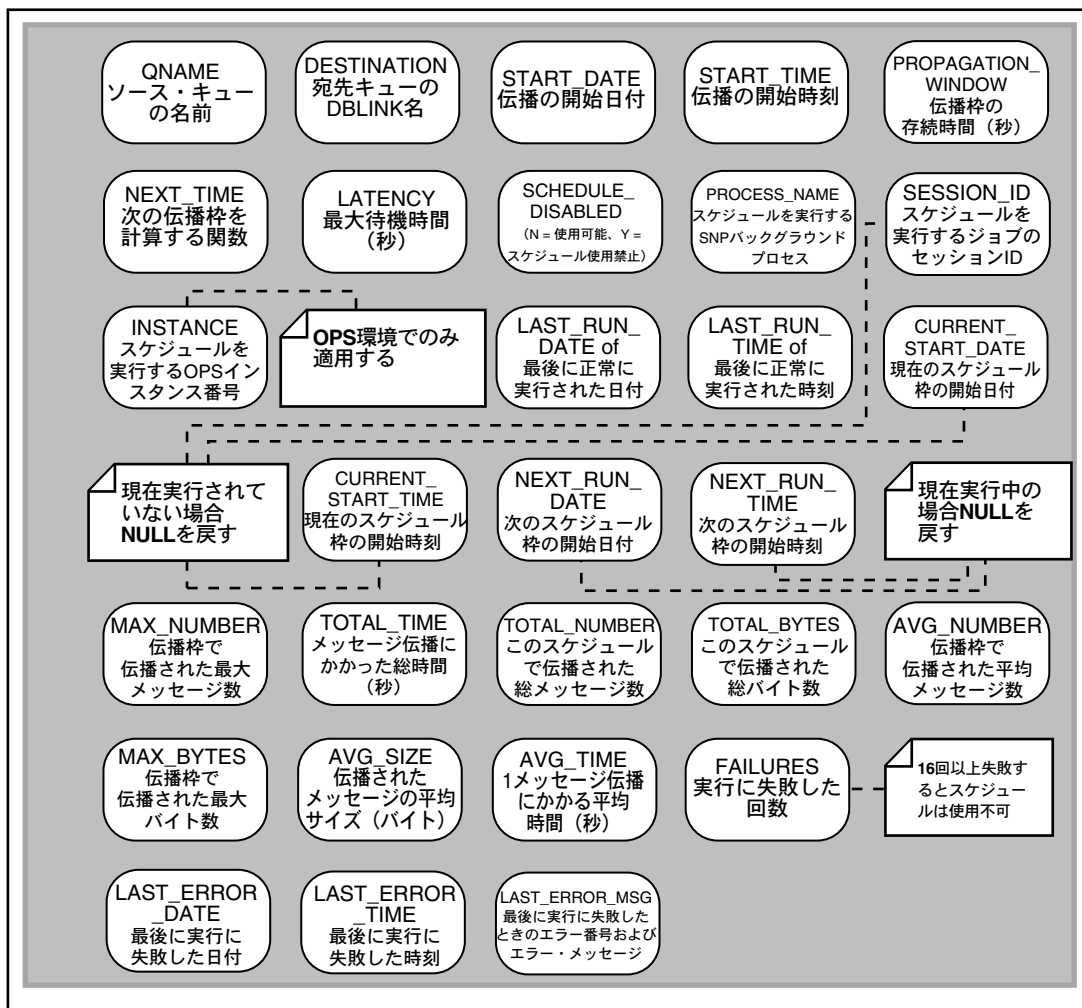
表 10-10 USER\_QUEUES

列名および説明	Null かどうか	型
NAME - キュー名	NOT NULL	VARCHAR2 (30)
QUEUE_TABLE - このキューが格納される キュー・テーブル	NOT NULL	VARCHAR2 (30)
QID - 一意のキュー識別子	NOT NULL	NUMBER
QUEUE_TYPE - キューの型		VARCHAR2 (15)
MAX_RETRIES - デキューの許容試行回数		NUMBER
RETRY_DELAY - 再試行までの秒数		NUMBER
ENQUEUE_ENABLED - YES/NO		VARCHAR2 (7)
ENQUEUE_ENABLED - YES/NO		VARCHAR2 (7)
RETENTION - メッセージがデキュー後に保存さ れる秒数		VARCHAR2 (40)
USER_COMMENT - キューに対するユーザー・コ メント		VARCHAR2 (50)

# ユーザー・スキーマの伝播スケジュールの選択

図 10-11 ユースケース図：ユーザー・スキーマの伝播スケジュールの選択





### 参照:

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース - ビュー」を参照してください。

ビュー名

USER\_QUEUE\_SCHEDULES

用途

表 10-11 USER\_QUEUE\_SCHEDULES

列名および説明	Null かどうか	型
QNAME - ソース・キューの名前	NOT NULL	VARCHAR2 (30)
DESTINATION - 宛先名。現在は DBLINK 名に制限されています。	NOT NULL	VARCHAR2 (128)
START_DATE - デフォルトの日付書式による伝播の開始日付		DATE
START_TIME - HH:MI:SS 形式による伝播の開始時刻		VARCHAR2 (8)
PROPAGATION_WINDOW - 伝播枠の存続期間 (秒)		NUMBER
NEXT_TIME - 次の伝播枠の開始を計算する関数		VARCHAR2 (200)
LATENCY - 伝播枠内で、メッセージを伝播するまでの最大待機時間		NUMBER
SCHEDULE_DISABLED - 使用可能のときは N、使用不可でスケジュールが実行されていないときは Y		VARCHAR (1)
PROCESS_NAME - このスケジュールを実行する SNP バックグラウンド・プロセス。実行されていないときは NULL になります。		VARCHAR2 (8)
SESSION_ID - このスケジュールを実行しているジョブのセッション ID (SID,SERIAL#)。実行されていないときは NULL になります。		VARCHAR2 (82)
INSTANCE - このスケジュールを実行している OPS インスタンス番号		NUMBER

表 10-11 USER\_QUEUE\_SCHEDULES (続き)

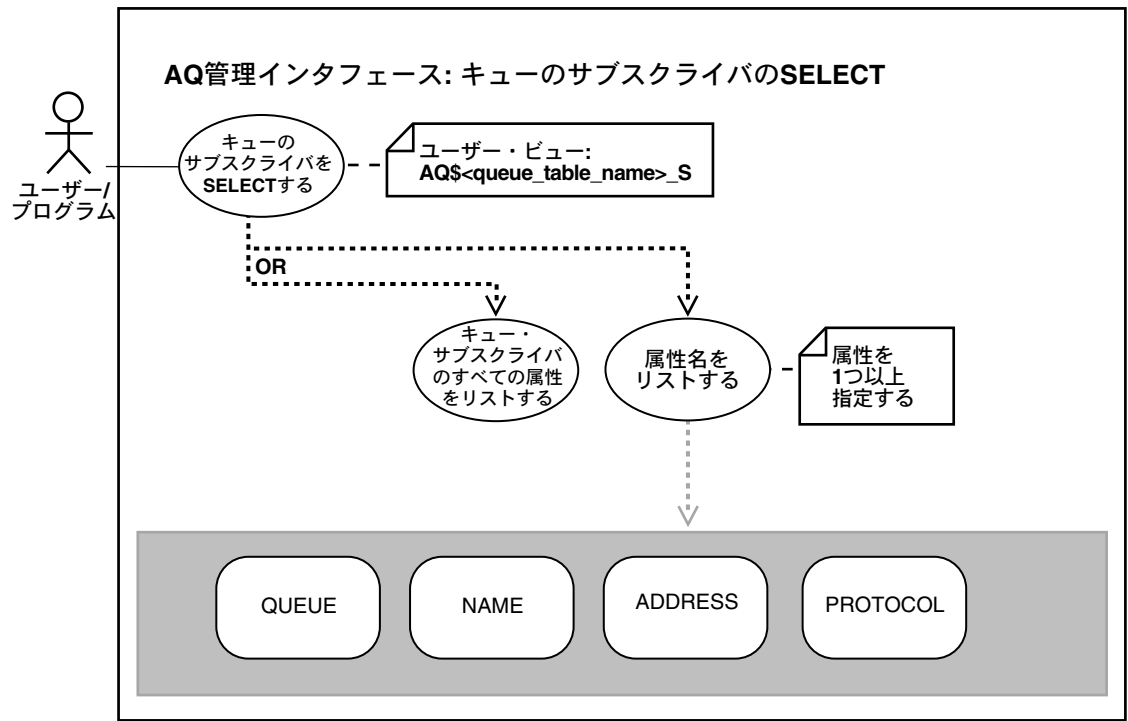
列名および説明	Null かどうか	型
LAST_RUN_DATE - 最後に正常に実行された日付		DATE
LAST_RUN_TIME - 最後に正常に実行された時刻 (HH:MI:SS 形式)		VARCHAR2 (8)
CURRENT_START_DATE - このスケジュール枠が開始された日付		DATE
CURRENT_START_TIME - このスケジュール枠が開始された時刻 (HH:MI:SS 形式)		VARCHAR2 (8)
NEXT_RUN_DATE - 次のスケジュール枠が開始される日付		DATE
NEXT_RUN_TIME - 次のスケジュール枠が開始される時刻 (HH:MI:SS 形式)		VARCHAR2 (8)
TOTAL_TIME - このスケジュールでメッセージの伝播に費やされた総時間 (秒)		NUMBER
TOTAL_NUMBER - このスケジュールで伝播された総メッセージ数		NUMBER
TOTAL_BYTES - このスケジュールで伝播された総バイト数		NUMBER
MAX_NUMBER - 伝播枠で伝播される最大メッセージ数		NUMBER
MAX_BYTES - 伝播枠で伝播される最大バイト数		NUMBER
AVG_NUMBER - 伝播枠で伝播される平均メッセージ数		NUMBER
AVG_SIZE - 伝播されたメッセージの平均サイズ (バイト)		NUMBER

表 10-11 USER\_QUEUE\_SCHEDULES (続き)

列名および説明	Null かどうか	型
AVG_TIME - 1 つのメッセージ伝播にかかる平均時間 (秒)		NUMBER
FAILURES - 実行が失敗した回数。16 になった場合、そのスケジュールは使用禁止になります。		NUMBER
LAST_ERROR_DATE - 最後に実行に失敗した日付		DATE
LAST_ERROR_TIME - 最後に実行に失敗した時刻		VARCHAR2 (8)
LAST_ERROR_MSG - 最後に実行に失敗したときの、エラー番号およびエラー・メッセージ		VARCHAR2 (4000)

# キューのサブスクライバの選択

図 10-12 ユースケース図：キューのサブスクライバの選択



**参照：**

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース - ビュー」を参照してください。

**ビュー名**

AQ\$<queue\_table\_name>\_S

用途

このビューは、指定されたキュー・テーブルの、すべてのキューに対するすべてのサブスクライバを示します。このビューは、キュー・テーブルが作成されたときに AQ\$<queue\_table\_name>\_S という名前で生成されます。キュー・テーブル内の一部またはすべてのキューに対するサブスクライバを問い合わせるために使用します。このビューは、8.1 互換のキュー・テーブルでのみ作成されることに注意してください。

表 10-12 AQ\$<queue\_table\_name>\_S

列名および説明	Null かどうか	型
QUEUE - サブスクライバが定義されたキューの名前	NOT NULL	VARCHAR2 (30)
NAME - エージェント名		VARCHAR2 (30)
ADDRESS - エージェントのアドレス		VARCHAR2 (1024)
PROTOCOL - エージェントのプロトコル		NUMBER

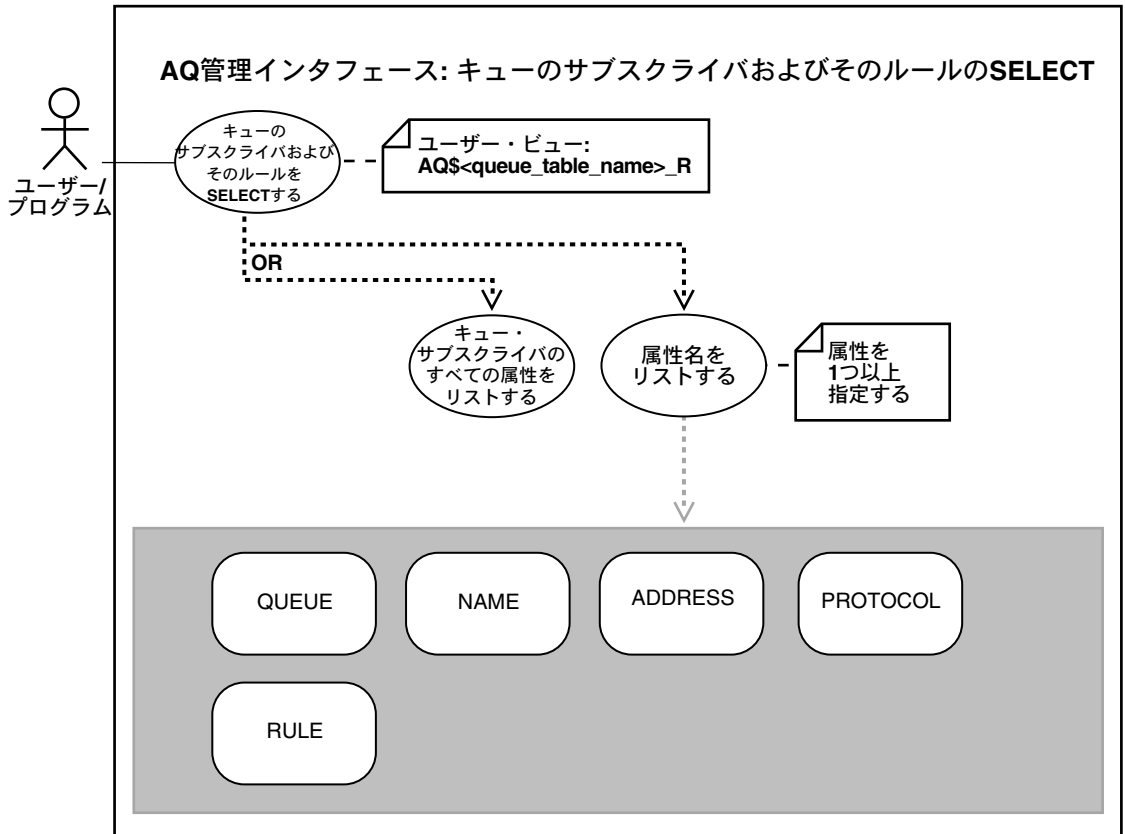
使用上の注意

このビューは、8.1 互換のキュー・テーブルに作成されたキューに対して、dbms\_aqadm.queue\_subscribers() プロシージャと等価の機能を提供します。これらのキューのサブスクライバを参照する場合、プロシージャではなくビューを使用することをお勧めします。



## キューのサブスクライバおよびそのルールを選択

図 10-13 ユースケース図：キューのサブスクライバおよびそのルールを選択



**参照：**

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル：管理インタフェース - ビュー」を参照してください。

ビュー名

AQ\$<queue\_table\_name>\_R

用途

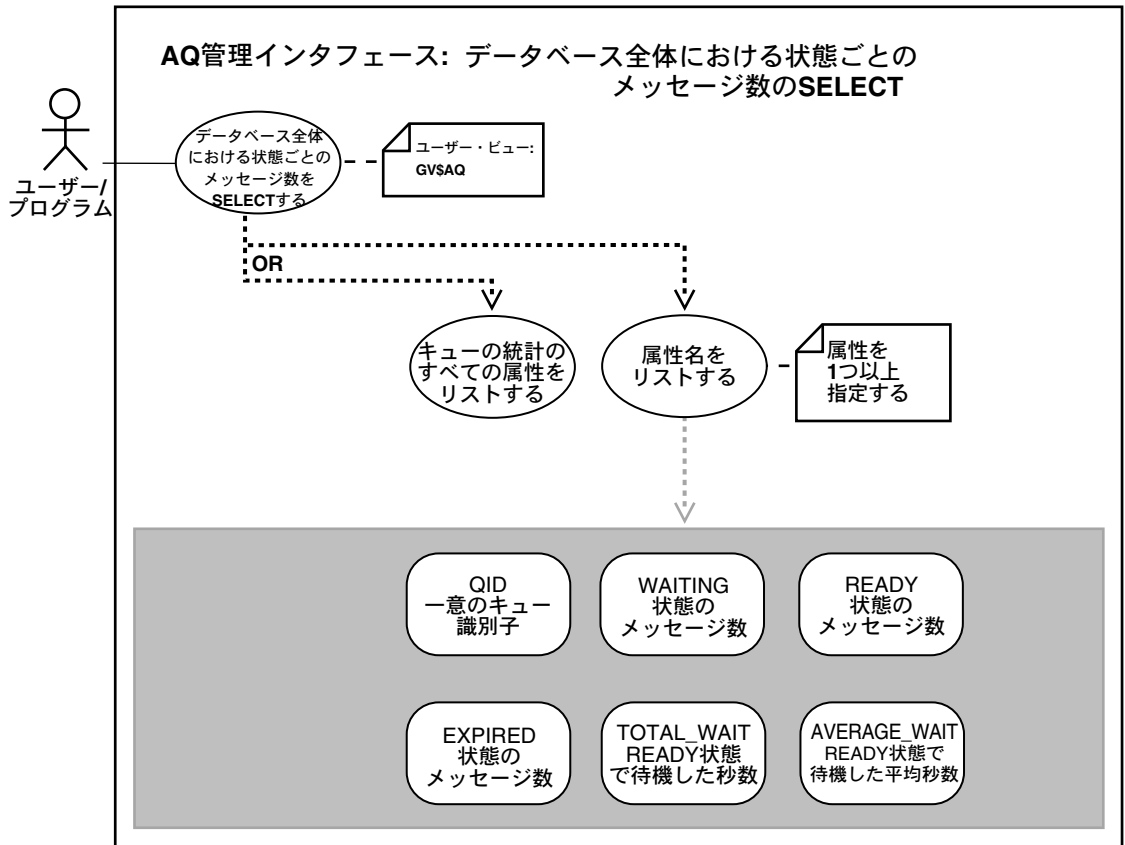
このビューは、指定されたキュー・テーブルのすべてのキューに対するルールベースのサブスクライバのみ（各サブスクライバによって定義されたルールのテキストを含む）を示します。このビューでは、指定されたキュー・テーブルのいずれかのキューに対して、ルールを定義されたサブスクライバを示します。このビューは、キュー・テーブルが作成されたときに AQ\$<queue\_table\_name>\_R という名前で生成されます。キュー・テーブル内の一部またはすべてのキューに対するサブスクライバを問い合わせるために使用します。このビューは、8.1 互換のキュー・テーブルでのみ作成されることに注意してください。

表 10-13 AQ\$<queue\_table\_name>\_R

列名および説明	Null かどうか	型
QUEUE - サブスクライバが定義されたキューの名前	NOT NULL	VARCHAR2 (30)
NAME - エージェント名		VARCHAR2 (30)
ADDRESS - エージェントのアドレス		VARCHAR2 (1024)
PROTOCOL - エージェントのプロトコル		NUMBER
RULE - 定義されたルールのテキスト		VARCHAR2 (30)

## データベース全体における状態ごとのメッセージ数の選択

図 10-14 データベース全体における状態ごとのメッセージ数の選択



参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース – ビュー」を参照してください。

ビュー名

GV\$AQ

用途

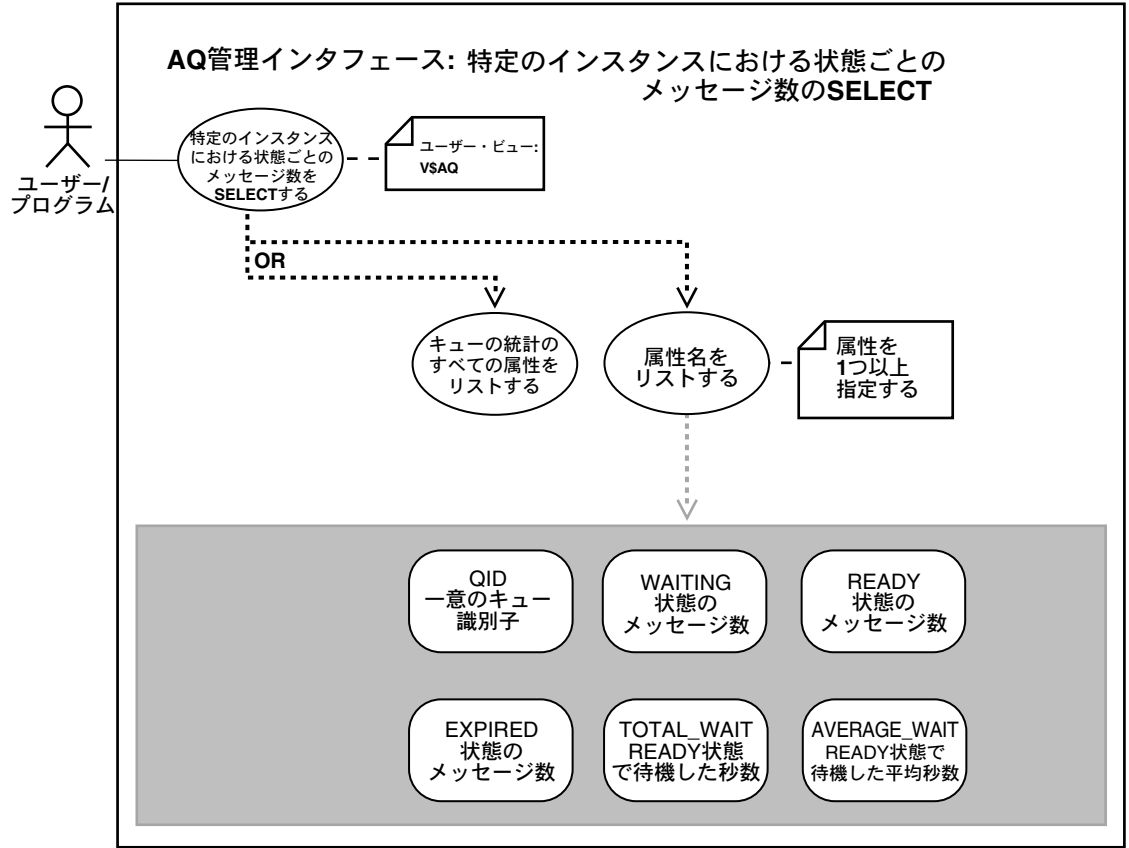
このビューは、データベース全体における状態ごとのメッセージ数の情報を示します。

表 10-14 AQ\$(queue\_table\_name)\_R

列名および説明	Null かどうか	型
QID - キューの ID。user_queues および dba_queues の qid と同じです。		NUMBER
WAITING - WAITING 状態にあるメッセージ数		NUMBER
READY - READY 状態にあるメッセージ数		NUMBER
EXPIRED - EXPIRED 状態にあるメッセージ数		NUMBER
TOTAL_WAIT - READY 状態でメッセージがキューに待機している秒数		NUMBER
AVERAGE_WAIT - READY 状態で、メッセージがデキューを待機している平均秒数。		NUMBER

# 特定のインスタンスにおける状態ごとのメッセージ数の選択

図 10-15 特定のインスタンスにおける状態ごとのメッセージ数の選択



参照：

- 管理インタフェースに関するすべての基本操作については、10-2 ページの「ユースケース・モデル:管理インタフェース – ビュー」を参照してください。

ビュー名

V\$AQ

用途

このビューは、特定のインスタンスにおける様々な状態にあるメッセージ数の情報を示します。

表 10-15 AQ\$<queue\_table\_name>\_R

列名および説明	Null かどうか	型
QID - キューの ID。user_queues および dba_queues の qid と同じです。		NUMBER
WAITING - WAITING 状態にあるメッセージ数		NUMBER
READY - READY 状態にあるメッセージ数		NUMBER
EXPIRED - EXPIRED 状態にあるメッセージ数		NUMBER
TOTAL_WAIT - READY 状態でメッセージがキューに待機している秒数		NUMBER
AVERAGE_WAIT - READY 状態で、メッセージがデキューを待機している平均秒数		NUMBER

---

## 操作インターフェース：基本操作

### ユースケース・モデル

この章では、Oracle アドバンスド・キューイングの操作インターフェースをユースケースに沿って説明します。それぞれの操作（「[メッセージのエンキュー](#)」など）を、その操作名ごとにユースケースの点から説明します。この章の先頭に、すべてのユースケースをリストします（[図 11-1「ユースケース・モデル図：操作インターフェース](#)」を参照）。

### ユースケース・モデルの図形概要

[図 11-1](#) に、すべてのユースケースを 1 つの図にまとめています。

### 個々のユースケース

個々のユースケースは、次のように配置されています。

- **ユースケース図**：ユースケースを表す図
- **用途**：このユースケースの用途
- **使用上の注意**：実装に有効なガイドライン
- **構文**：このアクティビティの実行に使用する主な構文
- **例**：各プログラム環境でのユースケースの例

# ユースケース・モデル：操作インタフェース – 基本操作

表 11-1「ユースケース・モデル：操作インタフェース」の「+」は、そのユースケースで、プログラム環境の例が記載されていることを示します。

この表では、プログラム環境を次の略称で表しています。

- P – DBMS\_AQADM および DBMS\_AQ パッケージを使用した PL/SQL
- O – OCI（Oracle コール・インタフェース）を使用した C
- V – OO4O（Oracle Objects for OLE）を使用した Visual Basic
- J – JDBC（Java Database Connectivity）を使用した Java（ネイティブ AQ）
- JM – JDBC（Java Database Connectivity）を使用した Java（JMS 標準）

表 11-1 ユースケース・モデル：操作インタフェース

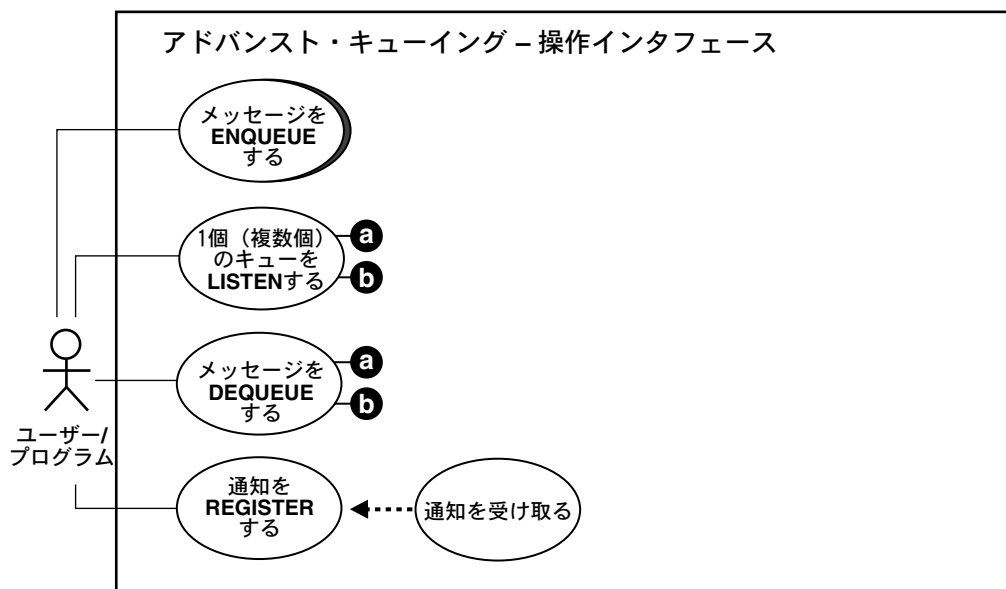
ユースケース	プログラム環境の例				
	P	O	V	J	JM
11-5 ページの「メッセージのエンキュー」					
11-7 ページの「メッセージのエンキュー（オプションの指定）」	+		+		+
11-10 ページの「メッセージのエンキュー（メッセージ・プロパティの指定）」	+		+		+
11-15 ページの「メッセージのエンキュー（ペイロードの追加）」	+		+		+
11-23 ページの「1 個（複数個）のキューのリスニング」					
11-25 ページの「1 個（複数個）の単一コンシューマ・キューのリスニング」	+	+	+		
11-36 ページの「1 個（複数個）の複数コンシューマ・キューのリスニング」	+	+	+		
11-45 ページの「メッセージのデキュー」					
11-48 ページの「単一コンシューマ・キューからのメッセージのデキュー（オプションの指定）」	+		+		+
11-53 ページの「複数コンシューマ・キューからのメッセージのデキュー（オプションの指定）」	+		+		+



表 11-1 ユースケース・モデル: 操作インタフェース

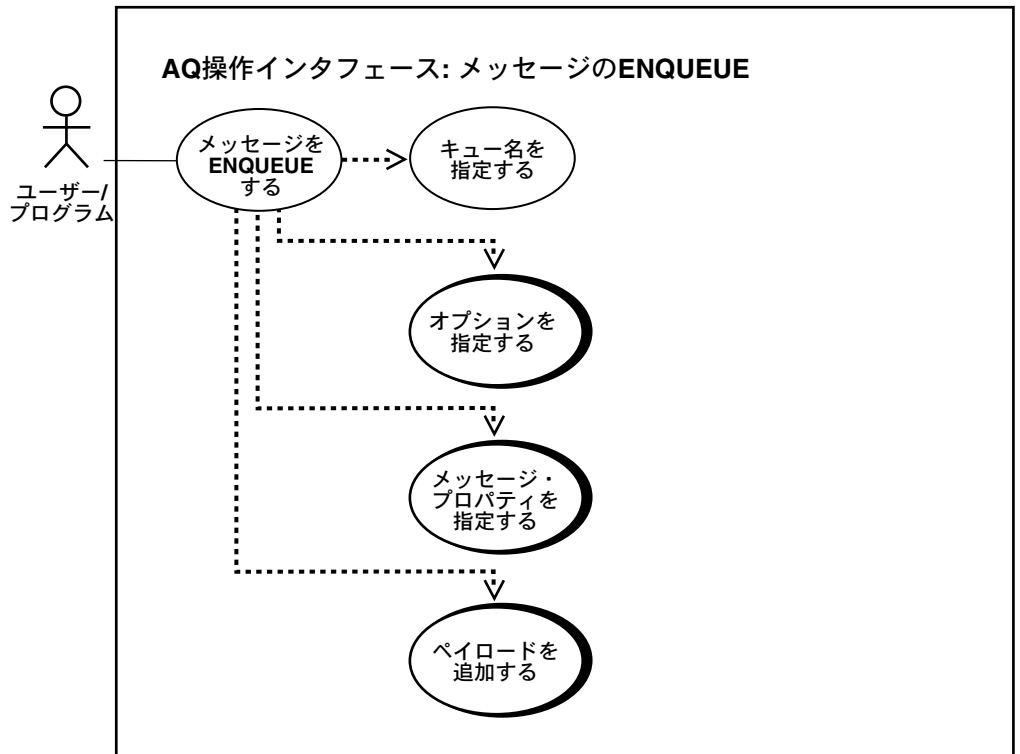
ユースケース	プログラム環境の例				
	P	O	V	J	JM
11-56 ページの「通知登録」					
11-59 ページの「通知登録（サブスクリプション名の指定 – 単一コンシューマ・キュー）」		+			
11-60 ページの「通知登録（サブスクリプション名の指定 – 複数コンシューマ・キュー）」		+			

図 11-1 ユースケース・モデル図：操作インタフェース



# メッセージのエンキュー

図 11-2 ユースケース図：メッセージのエンキュー

**参照：**

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インタフェース – 基本操作」を参照してください。

## 用途

メッセージを、指定したキューに追加します。

## 使用上の注意

- メッセージが受信者を指定しないで複数コンシューマ・キューにエンキューされ、そのキューにサブスクライバが指定されていない（またはそのメッセージに一致するルールベースのサブスクライバが存在しない）場合、ORA-24033 エラーになります。これは、配信先になる受信者またはサブスクライバが存在しないメッセージは廃棄されるという警告です。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第4章「DBMS\_AQ」の ENQUEUE プロシージャ
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ」を選択してください。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueue」の enqueue

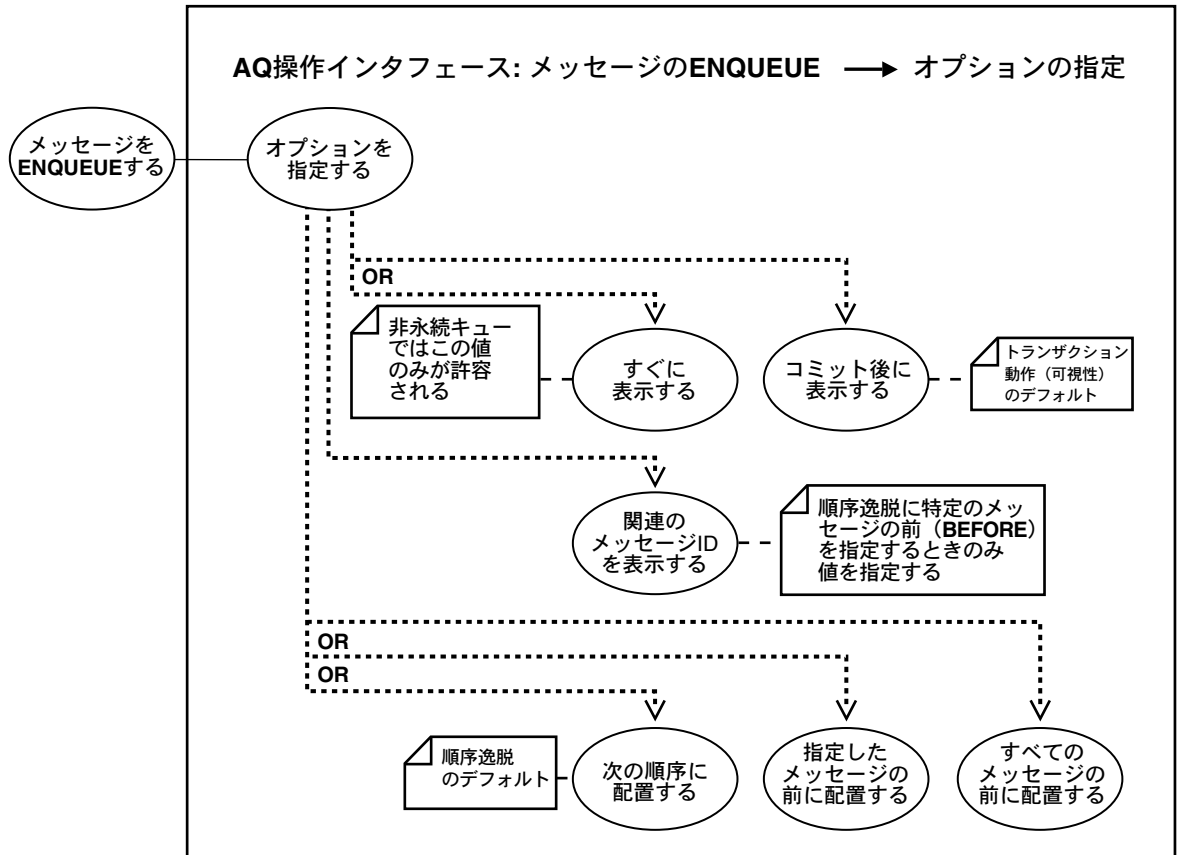
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : オブジェクト型メッセージのエンキュー](#) (11-17 ページ)
- [Java \(JDBC\) : メッセージのエンキュー \(ペイロードの追加\)](#) (11-19 ページ)
- [Visual Basic \(OO4O\) : メッセージのエンキュー](#) (11-21 ページ)

## メッセージのエンキュー（オプションの指定）

図 11-3 ユースケース図：メッセージのエンキュー（オプションの指定）



### 参照：

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インタフェース – 基本操作」を参照してください。

### 用途

エンキュー操作で利用できるオプションを指定します。

### 使用上の注意

LOB ロケータは、トランザクションの存続期間中のみ有効であるため、LOB ロケータを使用するときは、IMMEDIATE オプションを使用しないでください。IMMEDIATE オプションを使用するとトランザクションが自動的にコミットされるため、ロケータは有効になりません。

- エンキュー・オプションの `sequence deviation` パラメータを使用すると、2つのメッセージ間の処理順序を変更できます。他のメッセージが存在する場合は、その識別情報をエンキュー・オプションのパラメータ関連 `msgid` で指定します。関連が、`Sequence deviation` パラメータによって識別されます。

メッセージに `Sequence deviation` を指定すると、そのメッセージに指定できる遅延および優先順位の値が制限されます。このメッセージの遅延の値は、このメッセージより前にエンキューされるメッセージの遅延の値以下にする必要があります。このメッセージの優先順位の値は、このメッセージより前にエンキューされるメッセージの優先順位以上にする必要があります。

- 非永続キューでは、可視性オプションを IMMEDIATE にする必要があります。
- 非永続キューでは、ローカル受信者のみがサポートされています。

### 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第4章「DBMS\_AQ」の ENQUEUE プロシージャ
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ」を選択してください。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQEnqueueOption」

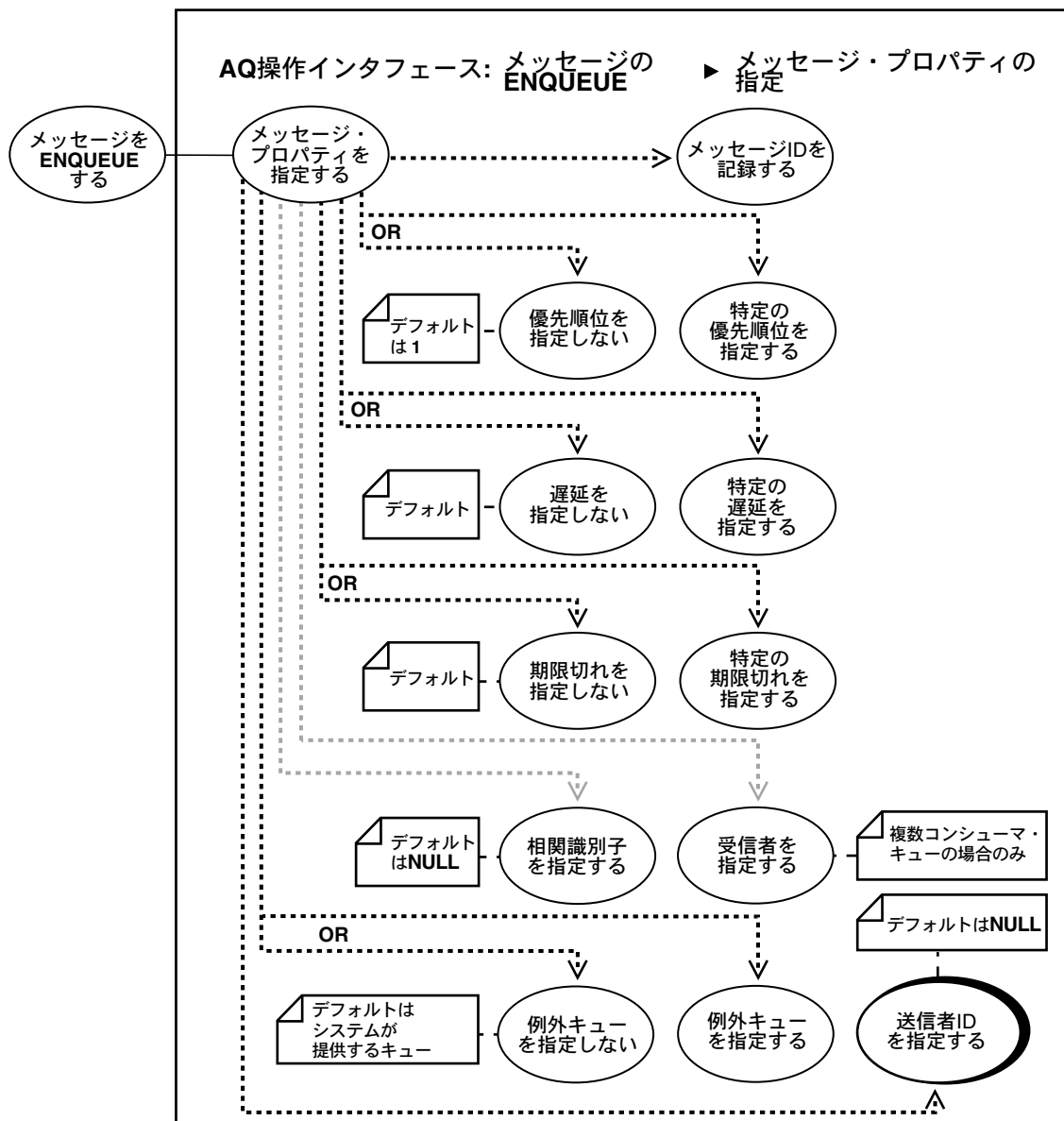
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\)](#) : オブジェクト型メッセージのエンキュー (11-17 ページ)
- [Java \(JDBC\)](#) : メッセージのエンキュー (ペイロードの追加) (11-19 ページ)
- [Visual Basic \(OO4O\)](#) : メッセージのエンキュー (11-21 ページ)

## メッセージのエンキュー（メッセージ・プロパティの指定）

図 11-4 ユースケース図：メッセージのエンキュー（メッセージ・プロパティの指定）





---

**参照：**

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「[ユースケース・モデル：操作インタフェース – 基本操作](#)」を参照してください。
- 

## 用途

メッセージ・プロパティには、AQ によって個々のメッセージを管理するために使用される情報が記述されています。これはエンキュー時に設定され、デキュー時にその値が戻されます。

## 使用上の注意

- 待機中または処理済状態のメッセージを参照するには、メッセージ ID によってデキューまたはブラウズするか、または SELECT 文を使用します。
- メッセージの遅延および期限切れの処理は、キュー・モニター（QMN）・バックグラウンド・プロセスによって行われます。AQ の遅延および期限切れの機能を使用するときは、データベースの QMN プロセスを開始する必要があります。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL（DBMS AQ パッケージ）：『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第 4 章「DBMS\_AQ」の ENQUEUE プロシージャ
- Visual Basic（Oracle Objects for OLE オンライン・ヘルプ）：「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ」を選択してください。
- Java（JDBC）：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 1 章「パッケージ oracle.AQ」の「AQMessageProperty」

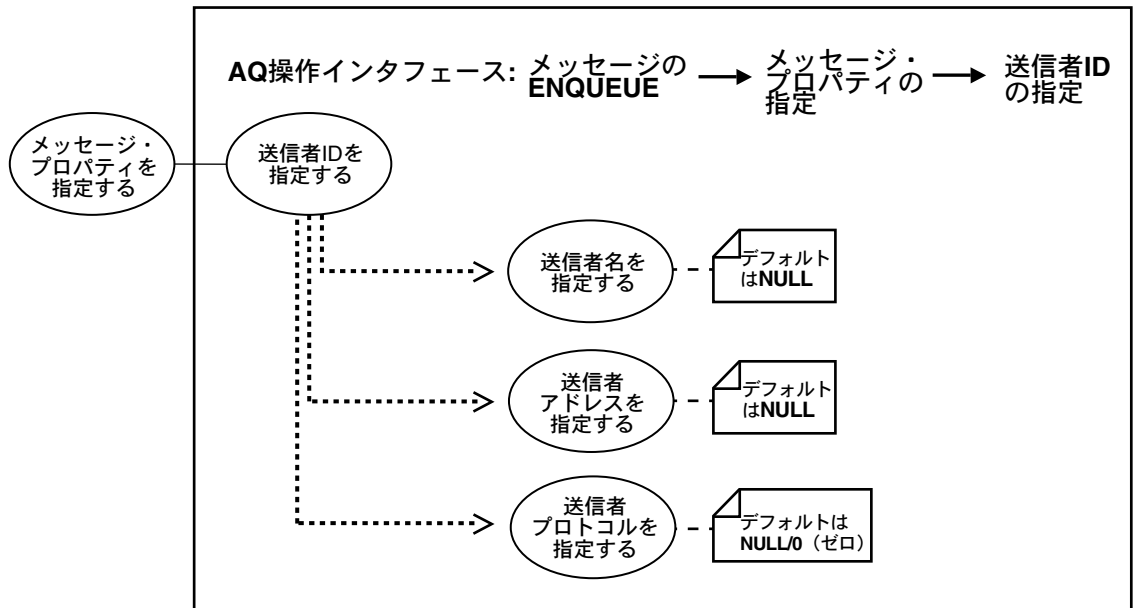
### 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : オブジェクト型メッセージのエンキュー](#) (11-17 ページ)
- [Java \(JDBC\) : メッセージのエンキュー \(ペイロードの追加\)](#) (11-19 ページ)
- [Visual Basic \(OO4O\) : メッセージのエンキュー](#) (11-21 ページ)

## メッセージのエンキュー（メッセージ・プロパティの指定（送信者 ID の指定））

図 11-5 ユースケース図：メッセージのエンキュー（メッセージ・プロパティの指定（送信者 ID の指定））



### 参照：

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 操作インタフェース – 基本操作」を参照してください。

## 用途

メッセージの送信者（プロデューサ）を識別します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第 4 章「DBMS\_AQ」の ENQUEUE プロシージャ
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ」を選択してください。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 1 章「パッケージ oracle.AQ」の「AQMessageProperty」の setsender

---

---

### 参照:

- エージェントの詳細は、2-3 ページの「[エージェント](#)」を参照してください。
- 
- 

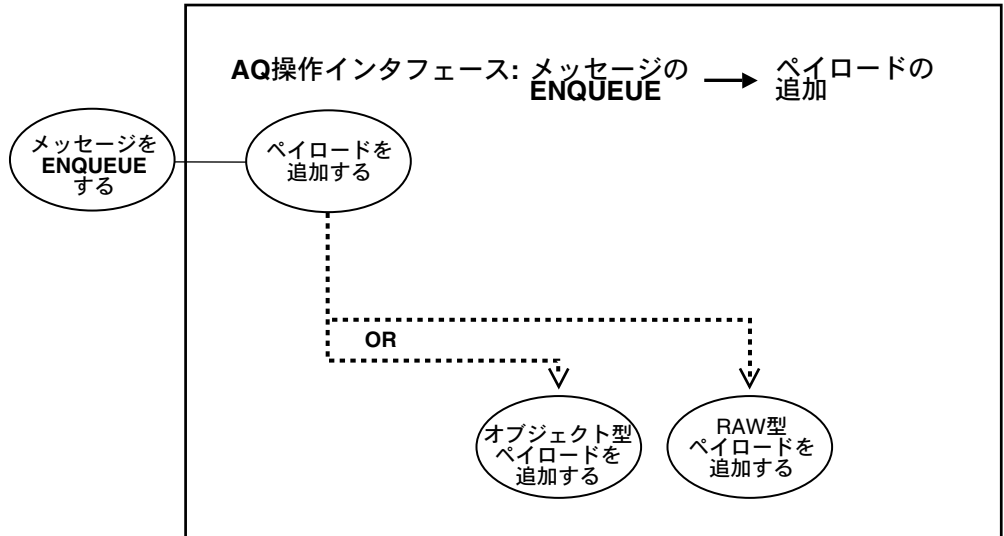
## 例

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : オブジェクト型メッセージのエンキュー](#) (11-17 ページ)
- [Java \(JDBC\) : メッセージのエンキュー \(ペイロードの追加\)](#) (11-19 ページ)
- [Visual Basic \(OO4O\) : メッセージのエンキュー](#) (11-21 ページ)

## メッセージのエンキュー（ペイロードの追加）

図 11-6 ユースケース図：メッセージのエンキュー（ペイロードの追加）



### 参照：

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「[ユースケース・モデル: 操作インタフェース - 基本操作](#)」を参照してください。

## 用途

## 使用上の注意

RAW 型のペイロードを格納するために、AQ ではペイロード・リポジトリとして LOB 列を持つキュー・テーブルを作成します。ペイロードの最大サイズは、AQ にアクセスするために使用しているプログラム環境によって決定されます。PL/SQL、Java およびプリコンパイラの場合は 32KB、OCI の場合は 4GB に制限されます。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第4章「DBMS\_AQ」の ENQUEUE プロシージャ
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ」を選択してください。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueue」の enqueue

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : オブジェクト型メッセージのエンキュー](#) (11-17 ページ)
- [Java \(JDBC\) : メッセージのエンキュー（ペイロードの追加）](#) (11-19 ページ)
- [Visual Basic \(OO4O\) : メッセージのエンキュー](#) (11-21 ページ)

## PL/SQL（DBMS\_AQ パッケージ）：オブジェクト型メッセージのエンキュー

**注意：** 次のようなデータ構造を設定しないと機能しない例もあります。

```
CONNECT system/manager
CREATE USER aq IDENTIFIED BY aq;
GRANT Aq_administrator_role TO aq;
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
    Queue_table           => 'aq.objmsgs_qtab',
    Queue_payload_type    => 'aq.message_typ');
EXECUTE DBMS_AQADM.CREATE_QUEUE (
    Queue_name           => 'aq.msg_queue',
    Queue_table          => 'aq.objmsgs_qtab');
EXECUTE DBMS_AQADM.START_QUEUE (
    Queue_name           => 'aq.msg_queue',
    Enqueue              => TRUE);
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
    Queue_table           => 'aq.prioritymsgs_qtab',
    Sort_list             => 'PRIORITY,ENQ_TIME',
    Queue_payload_type    => 'aq.message_typ');
EXECUTE DBMS_AQADM.CREATE_QUEUE (
    Queue_name           => 'aq.priority_msg_queue',
    Queue_table          => 'aq.prioritymsgs_qtab');
EXECUTE DBMS_AQADM.START_QUEUE (
    Queue_name           => 'aq.priority_msg_queue',
    Enqueue              => TRUE);
```

### 単一メッセージをエンキューし、キュー名およびペイロードを指定する

```
/* msg_queue にエンキューします。*/
DECLARE
    Enqueue_options      DBMS_AQ.enqueue_options_t;
    Message_properties   DBMS_AQ.message_properties_t;
    Message_handle       RAW(16);
    Message              aq.message_typ;

BEGIN
    Message := aq.message_typ('NORMAL MESSAGE',
        'enqueued to msg_queue first.');
```

```
    DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
        Enqueue_options      => enqueue_options,
```

```
Message_properties => message_properties,  
Payload            => message,  
Msgid              => message_handle);  
  
COMMIT;  
END;
```

### 単一メッセージをエンキューし、優先順位を指定する

/\* キュー名 `priority_msg_queue` は、オブジェクト型のキュー・テーブルとして定義されています。ペイロード・オブジェクト型はメッセージです。キューのスキーマは `aq` です。\*/

/\* 優先順位 30 でメッセージをエンキューします。\*/

```
DECLARE  
    Enqueue_options    dbms_aq.enqueue_options_t;  
    Message_properties  dbms_aq.message_properties_t;  
    Message_handle      RAW(16);  
    Message             aq.Message_typ;  
  
BEGIN  
    Message := Message_typ('PRIORITY MESSAGE', 'enqueued at priority 30.');
```

message\_properties.priority := 30;

```
    DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',  
enqueue_options            => enqueue_options,  
message_properties          => message_properties,  
payload                     => message,  
msgid                       => message_handle);  
  
    COMMIT;  
END;
```



## Java（JDBC）：メッセージのエンキュー（ペイロードの追加）

```
/* 設定します。*/
connect system/manager
create user aq identified by aq;
grant aq_administrator_role to aq;

public static void setup(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;
    AQAgent                 agent;

    qtable_prop = new AQQueueTableProperty("RAW");

    q_table = aq_sess.createQueueTable ("aq", "rawmsgs_qtab", qtable_prop);

    queue_prop = new AQQueueProperty();
    queue = aq_sess.createQueue (q_table, "msg_queue", queue_prop);

    queue.start();

    qtable_prop = new AQQueueTableProperty("RAW");
    qtable_prop.setMultiConsumer(true);

    qtable_prop.setSortOrder("priority,enq_time");
    q_table = aq_sess.createQueueTable ("aq", "rawmsgs_qtab2",
    qtable_prop);

    queue_prop = new AQQueueProperty();
    queue = aq_sess.createQueue (q_table, "priority_msg_queue", queue_prop);

    queue.start();

    agent = new AQAgent("subscriber1", null);

    queue.addSubscriber(agent, null);
}

/* メッセージをエンキューします。*/
public static void example(AQSession aq_sess) throws AQException, SQLException
{

```

```
AQQueue          queue;
AQMessage         message;
AQRawPayload      raw_payload;
AQEnqueueOption   enq_option;
String            test_data = "new message";
byte[]            b_array;
Connection         db_conn;

db_conn = ((AQOracleSession)aq_sess).getDBConnection();

/* キューへのハンドルを取得します。*/
queue = aq_sess.getQueue ("aq", "msg_queue");

/* RAW 型のペイロードを含むメッセージを作成します。*/
message = queue.createMessage();

/* Get handle to the AQRawPayload オブジェクトへのハンドルを取得し、RAW データ付きで
移入します。*/
b_array = test_data.getBytes();

raw_payload = message.getRawPayload();

raw_payload.setStream(b_array, b_array.length);

/* AQEnqueueOption オブジェクトをデフォルトのオプション付きで作成します。*/
enq_option = new AQEnqueueOption();

/* メッセージをエンキューします。*/
queue.enqueue(enq_option, message);

db_conn.commit();
}

/* 優先順位 5 でメッセージをエンキューします。*/
public static void example(AQSession aq_sess) throws AQException, SQLException
{
    AQQueue          queue;
    AQMessage         message;
    AQMessageProperty msg_prop;
    AQRawPayload      raw_payload;
    AQEnqueueOption   enq_option;
    String            test_data = "priority message";
    byte[]            b_array;
    Connection         db_conn;
```

```

db_conn = ((AQOracleSession)aq_sess).getDBConnection();

/* キューへのハンドルを取得します。*/
queue = aq_sess.getQueue ("aq", "msg_queue");

/* RAW 型のペイロードを含むメッセージを作成します。*/
message = queue.createMessage();

/* メッセージ・プロパティを取得します。*/
msg_prop = message.getMessageProperty();

/* 優先順位を指定します。*/
msg_prop.setPriority(5);

/* Get handle to the AQRawPayload オブジェクトへのハンドルを取得し、RAW データ付きで
移入します。*/
b_array = test_data.getBytes();

raw_payload = message.getRawPayload();

raw_payload.setStream(b_array, b_array.length);

/* AQEnqueueOption オブジェクトをデフォルトのオプション付きで作成します。*/
enq_option = new AQEnqueueOption();

/* メッセージをエンキューします。*/
queue.enqueue(enq_option, message);

db_conn.commit();
}

```

## Visual Basic（0040）：メッセージのエンキュー

### オブジェクト型メッセージのエンキュー

```

'Prepare the message. MESSAGE_TYPE is a user defined type
' in the "AQ" schema
Set OraMsg = Q.AQMsg(1, "MESSAGE_TYPE")
Set OraObj = DB.CreateOraObject("MESSAGE_TYPE")

OraObj("subject").Value = "Greetings from 0040"
OraObj("text").Value = "Text of a message originated from 0040"

Set OraMsg.Value = OraObj
Msgid = Q.Enqueue

```

### RAW 型メッセージのエンキュー

```
'Create an OraAQ object for the queue "DBQ"
Dim Q as object
Dim Msg as object
Dim OraSession as object
Dim DB as object

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.OpenDatabase(mydb, "scott/tiger" 0&)
Set Q = DB.CreateAQ("DBQ")

'Get a reference to the AQMsg object
Set Msg = Q.AQMsg
Msg.Value = "Enqueue the first message to a RAW queue."

'Enqueue the message
Q.Enqueue()

'Enqueue another message.

Msg.Value = "Another message"
Q.Enqueue()

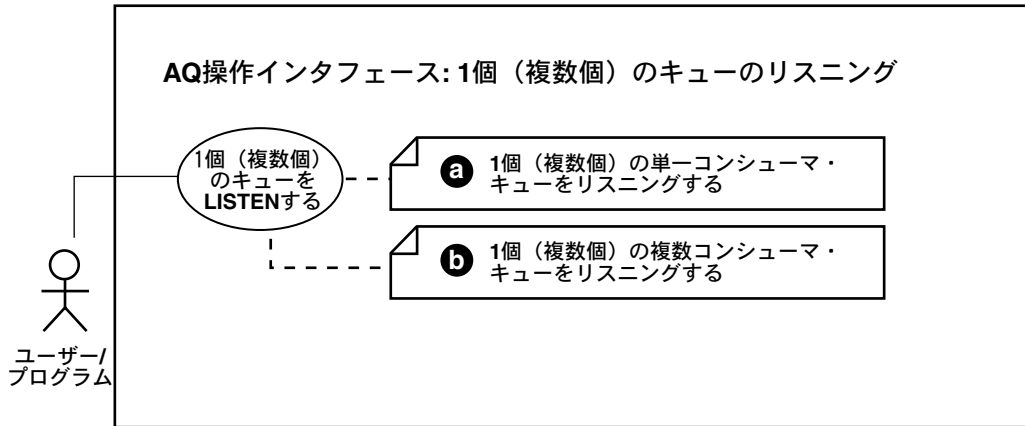
'Enqueue a message with non-default properties.
Msg.Priority = ORAQMSG_HIGH_PRIORITY
Msg.Delay = 5
Msg.Value = "Urgent message"
Q.Enqueue()
Msg.Value = "The visibility option used in the enqueue call is
             ORAAQ_ENQ_IMMEDIATE"
Q.Visible = ORAAQ_ENQ_IMMEDIATE
Msgid = Q.Enqueue

'Enqueue Ahead of message Msgid_1
Msg.Value = "First Message to test Relative Message id"
Msg.Correlation = "RELATIVE_MESSAGE_ID"

Msgid_1 = Q.Enqueue
Msg.Value = "Second message to test RELATIVE_MESSAGE_ID is queued
            ahead of the First Message "
OraAq.relmsgid = Msgid_1
Msgid = Q.Enqueue
```

## 1 個（複数個）のキューのリスニング

図 11-7 ユースケース図：1 個（複数個）のキューのリスニング



### 参照：

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インタフェース – 基本操作」を参照してください。

## 用途

エージェント・リストのかわりに、1 つまたは複数のキューを監視します。

## 使用上の注意

コールの引数に、エージェント・リストがあります。各エージェント・リストのアドレス・フィールドには、監視するキューを指定します。複数コンシューマ・キューを監視するときは、エージェント名も指定する必要があります。単一コンシューマ・キューの場合は、エージェント名を指定しないでください。アドレスでサポートされているのは、ローカル・キューのみです。プロトコルは、将来の使用に備えて確保されています。

このコールは、リストにあるエージェントによって処理可能なメッセージがある場合に返されるブロッキング・コールです。処理可能なエージェントが複数ある場合、最初にリストされたエージェントのみが返されます。待機期限切れになった時点で1 つもメッセージがない場合は、エラーになります。

Listen コールから正常に戻っても、指定されたキューの 1 つに指定されたエージェントの 1 つによって処理できるメッセージがあることを意味しているに過ぎません。処理するエージェントは、その関連メッセージをデキューする必要があります。

非永続キューには Listen をコールできないことに注意してください。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第 4 章「DBMS\_AQ」の LISTEN プロシージャ
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ Object」>「Monitoring Messages」を選択してください。
- この機能は、Java API を介しては使用できません。

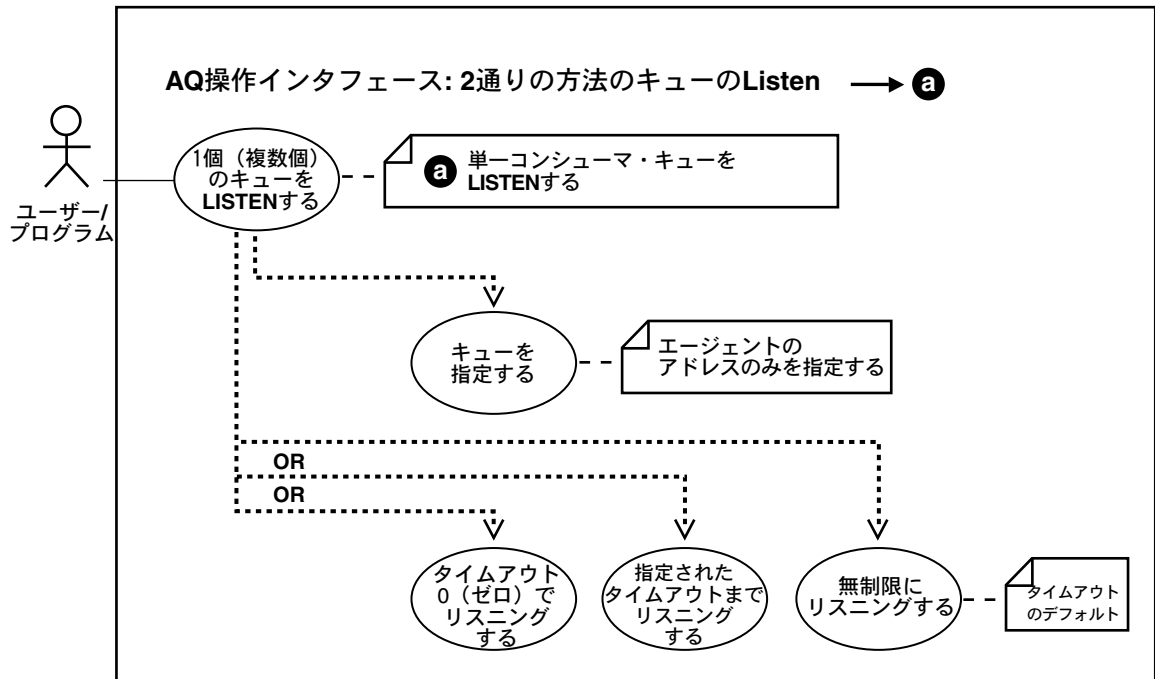
## 例

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : キューのリスニング](#) (11-26 ページ)
- [C \(OCI\) : 単一コンシューマ・キューのリスニング](#) (11-27 ページ)

# 1 個（複数個）の単一コンシューマ・キューのリスニング

図 11-8 ユースケース図：1 個（複数個）の単一コンシューマ・キューのリスニング



## 参照：

- 管理インターフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インターフェース – 基本操作」を参照してください。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第 4 章「DBMS\_AQ」の LISTEN プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の OCIAQListen
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ Object」>「Monitoring Messages」を選択してください。
- この機能は、Java API を介しては使用できません。

## 例

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : キューのリスニング](#) (11-26 ページ)
- [C \(OCI\) : 単一コンシューマ・キューのリスニング](#) (11-27 ページ)

## PL/SQL (DBMS\_AQ パッケージ) : キューのリスニング

/\* リスニング・コールによって、特定のエージェントに対するメッセージのリストを監視できます。監視するすべてのキューに対するデキュー権限が必要です。\*/

### タイムアウト 0 (ゼロ) で単一コンシューマ・キューをリスニングする

```
DECLARE
    Agent_w_msg      aq$_agent;
    My_agent_list     dbms_aq.agent_list_t;

BEGIN
    /* 注意 : MCQ1, MCQ2, MCQ3 は、SCOTT のスキーマの複数コンシューマ・キューです。
       SCQ1, SCQ2, SCQ3 は、SCOTT のスキーマの単一コンシューマ・キューです。*/ 1

    Qlist(1) := aq$_agent(NULL, 'scott.SCQ1', NULL);
```



```

Qlist(2) := aq$_agent(NULL, 'SCQ2', NULL);
Qlist(3) := aq$_agent(NULL, 'SCQ3', NULL);

/* タイムアウト 0 でリスニングします。*/
DBMS_AQ.LISTEN(
  Agent_list => My_agent_list,
  Wait       => 0,
  Agent      => agent_w_msg);
DBMS_OUTPUT.PUT_LINE('Message in Queue :- ' || agent_w_msg.address);
DBMS_OUTPUT.PUT_LINE('');
END;

```

## C (OCI) : 単一コンシューマ・キューのリスニング

### タイムアウト 0（ゼロ）で単一コンシューマ・キューをリスニングする

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
  text errbuf[512];
  ub4 buflen;
  sb4 errcode;

  switch (status)
  {
  case OCI_SUCCESS:
    break;
  case OCI_SUCCESS_WITH_INFO:
    printf("Error - OCI_SUCCESS_WITH_INFO\n");
    break;
  case OCI_NEED_DATA:
    printf("Error - OCI_NEED_DATA\n");
    break;
  case OCI_NO_DATA:
    printf("Error - OCI_NO_DATA\n");
    break;
  case OCI_ERROR:

```

```

        OCIErrGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
        printf("Error - %s\n", errbuf);
        break;
    case OCI_INVALID_HANDLE:
        printf("Error - OCI_INVALID_HANDLE\n");
        break;
    case OCI_STILL_EXECUTING:
        printf("Error - OCI_STILL_EXECUTE\n");
        break;
    case OCI_CONTINUE:
        printf("Error - OCI_CONTINUE\n");
        break;
    default:
        break;
    }
}

/* 記述子にエージェントを設定します。*/
void SetAgent(agent, appname, queue, errhp)

OCIAQAgent  *agent;
text        *appname;
text        *queue;
OCIErr      *errhp;
{

    OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
        appname ? (dvoid *)appname : (dvoid *)"",
        appname ? strlen((const char *)appname) : 0,
        OCI_ATTR_AGENT_NAME, errhp);

    OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
        queue ? (dvoid *)queue : (dvoid *)"",
        queue ? strlen((const char *)queue) : 0,
        OCI_ATTR_AGENT_ADDRESS, errhp);

    printf("Set agent name to %s\n", appname ? (char *)appname : "NULL");
    printf("Set agent address to %s\n", queue ? (char *)queue : "NULL");
}

/* 記述子からエージェントを取得します。*/
void GetAgent(agent, errhp)
OCIAQAgent *agent;
OCIErr      *errhp;

```

```

{
    text      *appname;
    text      *queue;
    ub4       appsz;
    ub4       queuesz;

    if (!agent )
    {
        printf("agent was NULL \n");
        return;
    }
    checkerr(errhp, OCIAAttrGet(agent, OCI_DTYPE_AQAGENT,
        (dvoid *)&appname, &appsz, OCI_ATTR_AGENT_NAME, errhp));
    checkerr(errhp, OCIAAttrGet(agent, OCI_DTYPE_AQAGENT,
        (dvoid *)&queue, &queuesz, OCI_ATTR_AGENT_ADDRESS, errhp));
    if (!appsz)
        printf("agent name: NULL\n");
    else printf("agent name: %.*s\n", appsz, (char *)appname);
    if (!queuesz)
        printf("agent address: NULL\n");
    else printf("agent address: %.*s\n", queuesz, (char *)queue);
}

int main()
{
    OCIEnv *envhp;
    OCIServer *srvhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCISession *usrhp;
    OCIAQAgent *agent_list[3];
    OCIAQAgent *agent = (OCIAQAgent *)0;
    /* 次の 2 121598 が追加されました。*/
    int i;

    /* 標準的な OCI の初期化*/

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
        (dvoid * (*)()) 0, (void (*)()) 0 );

    OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp,
        (ub4) OCI_HTYPE_ENV, 0, (dvoid **) 0);

```

```
OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 0, (dvoid **) 0);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
    0, (dvoid **) 0);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
    0, (dvoid **) 0);

OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
    0, (dvoid **) 0);

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) srvhp, (ub4) 0,
    (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* ユーザー・コンテキスト・ハンドルを割り当てます。*/
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
    (size_t) 0, (dvoid **) 0);

/* ユーザー・コンテキスト・ハンドルを割り当てます。*/
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
    (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
    (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION,
    (dvoid *) "tiger", (ub4) strlen("tiger"),
    (ub4) OCI_ATTR_PASSWORD, errhp);

OCISessionBegin( svchp, errhp, usrhp, OCI_CRED_RDBMS, OCI_DEFAULT);

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
    (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* AQ LISTEN の初期化 - エージェント・ハンドルを割り当てます。*/
for (i = 0; i < 3; i++)
{
    agent_list[i] = (OCIAQAgent *)0;
    OCIDescriptorAlloc(envhp, (dvoid **)&agent_list[i],
        OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
}

/*
 *   SCQ1、SCQ2、SCQ3 は、SCOTT のスキーマの単一コンシューマ・キューです。
 */
```

```

*/

SetAgent(agent_list[0], (text *)0, "SCOTT.SQ1", errhp);
SetAgent(agent_list[1], (text *)0, "SCOTT.SQ2", errhp);
SetAgent(agent_list[2], (text *)0, "SCOTT.SQ3", errhp);

checkerr(errhp, OCIAQListen(svchp, errhp, agent_list, 3, 0, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");
}

```

## タイムアウト 120 秒で単一コンシューマ・キューをリスニングする

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    switch (status)
    {
    case OCI_SUCCESS:
        break;
    case OCI_SUCCESS_WITH_INFO:
        printf("Error - OCI_SUCCESS_WITH_INFO\n");
        break;
    case OCI_NEED_DATA:
        printf("Error - OCI_NEED_DATA\n");
        break;
    case OCI_NO_DATA:
        printf("Error - OCI_NO_DATA\n");
        break;
    case OCI_ERROR:
        OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,

```

```

        errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
case OCI_STILL_EXECUTING:
    printf("Error - OCI_STILL_EXECUTE\n");
    break;
case OCI_CONTINUE:
    printf("Error - OCI_CONTINUE\n");
    break;
default:
    break;
}
}

/* 記述子にエージェントを設定します。*/
/* SetAgent(agent, appname, queue) を無効にします。*/
void SetAgent(agent, appname, queue, errhp)

OCIAQAgent  *agent;
text        *appname;
text        *queue;
OCIError    *errhp;
{

    OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
        appname ? (dvoid *)appname : (dvoid *)"",
        appname ? strlen((const char *)appname) : 0,
        OCI_ATTR_AGENT_NAME, errhp);

    OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
        queue ? (dvoid *)queue : (dvoid *)"",
        queue ? strlen((const char *)queue) : 0,
        OCI_ATTR_AGENT_ADDRESS, errhp);

    printf("Set agent name to %s\n", appname ? (char *)appname : "NULL");
    printf("Set agent address to %s\n", queue ? (char *)queue : "NULL");
}

/* 記述子からエージェントを取得します。*/
void GetAgent(agent, errhp)
OCIAQAgent *agent;
OCIError   *errhp;

```

```

{
    text      *appname;
    text      *queue;
    ub4       appsz;
    ub4       queuesz;

    if (!agent )
    {
        printf("agent was NULL \n");
        return;
    }
    checkerr(errhp, OCIAAttrGet(agent, OCI_DTYPE_AQAGENT,
        (dvoid *)&appname, &appsz, OCI_ATTR_AGENT_NAME, errhp));
    checkerr(errhp, OCIAAttrGet(agent, OCI_DTYPE_AQAGENT,
        (dvoid *)&queue, &queuesz, OCI_ATTR_AGENT_ADDRESS, errhp));
    if (!appsz)
        printf("agent name: NULL\n");
    else printf("agent name: %.*s\n", appsz, (char *)appname);
    if (!queuesz)
        printf("agent address: NULL\n");
    else printf("agent address: %.*s\n", queuesz, (char *)queue);
}

int main()
{
    OCIEnv *envhp;
    OCIServer *srvhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCISession *usrhp;
    OCIAQAgent *agent_list[3];
    OCIAQAgent *agent = (OCIAQAgent *)0;
    /* 次の 2 121598 が追加されました。*/
    int i;

    /* 標準的な OCI の初期化*/

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
        (dvoid * (*)()) 0, (void (*)()) 0 );

    OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp,
        (ub4) OCI_HTYPE_ENV, 0, (dvoid **) 0);

    OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 0, (dvoid **) 0);

```

```
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
    0, (dvoid **) 0);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
    0, (dvoid **) 0);

OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
    0, (dvoid **) 0);

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) srvhp, (ub4) 0,
    (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* ユーザー・コンテキスト・ハンドルを割り当てます。*/
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
    (size_t) 0, (dvoid **) 0);

/* ユーザー・コンテキスト・ハンドルを割り当てます。*/
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
    (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
    (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION,
    (dvoid *) "tiger", (ub4) strlen("tiger"),
    (ub4) OCI_ATTR_PASSWORD, errhp);

OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS, OCI_DEFAULT);

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
    (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* AQ LISTEN の初期化 - エージェント・ハンドルを割り当てます。*/
for (i = 0; i < 3; i++)
{
    agent_list[i] = (OCAQAgent *)0;
    OCIDescriptorAlloc(envhp, (dvoid **)&agent_list[i],
        OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
}

/* SCQ1、SCQ2、SCQ3 は、SCOTT のスキーマの単一コンシューマ・キューです。*/

SetAgent(agent_list[0], (text *)0, "SCOTT.SCQ1", errhp);
SetAgent(agent_list[1], (text *)0, "SCOTT.SCQ2", errhp);
```



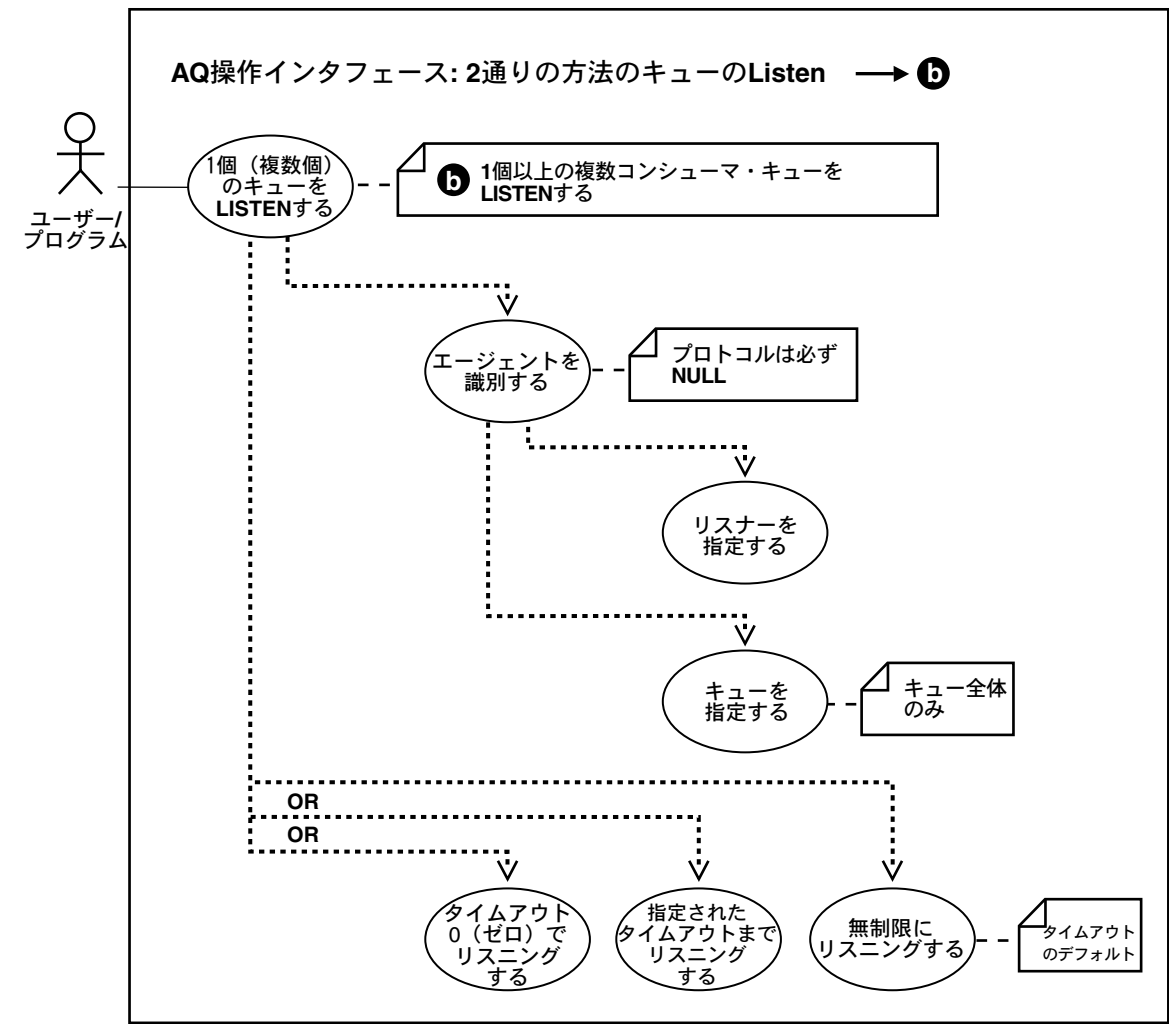
```
SetAgent(agent_list[2], (text *)0, "SCOTT.SQ3", errhp);

checkerr(errhp,OCIAQListen(svchp, errhp, agent_list, 3, 120, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");
}
```

# 1 個（複数個）の複数コンシューマ・キューのリスニング

図 11-9 ユースケース図: 1 個（複数個）の複数コンシューマ・キューのリスニング



---

**参照：**

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「[ユースケース・モデル：操作インタフェース – 基本操作](#)」を参照してください。
- 

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第 4 章「DBMS\_AQ」の LISTEN プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の「リレーショナル機能」の OCIAQListen
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ Object」>「Monitoring Messages」を選択してください。
- この機能は、Java API を介しては使用できません。

## 例

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境](#)」を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : キューのリスニング](#) (11-38 ページ)
- [C \(OCI\) : 複数コンシューマ・キューのリスニング](#) (11-39 ページ)

## PL/SQL (DBMS\_AQ パッケージ) : キューのリスニング

/\* リスニング・コールによって、特定のエージェントに対するメッセージのリストを監視できます。監視するすべてのキューに対するデキュー権限が必要です。\*/

### タイムアウト 0 (ゼロ) で複数コンシューマ・キューをリスニングする

```
DECLARE
    Agent_w_msg      aq$_agent;
    My_agent_list     dbms_aq.agent_list_t;

BEGIN
    /* 注意 : MCQ1、MCQ2、MCQ3 は、SCOTT のスキーマの複数コンシューマ・キューです。
     *      SCQ1、SCQ2、SCQ3 は、SCOTT のスキーマの単一コンシューマ・キューです。 */
    Qlist(1) := aq$_agent('agent1', 'MCQ1', NULL);
    Qlist(2) := aq$_agent('agent2', 'scott.MCQ2', NULL);
    Qlist(3) := aq$_agent('agent3', 'scott.MCQ3', NULL);

    /* タイムアウト 0 でリスニングします。*/
    DBMS_AQ.LISTEN(
        agent_list => My_agent_list,
        wait       => 0,
        agent       => agent_w_msg);
    DBMS_OUTPUT.PUT_LINE('Message in Queue :- ' || agent_w_msg.address);
    DBMS_OUTPUT.PUT_LINE('');
END;
/
```

### タイムアウト 100 秒で複数コンシューマ・キューのグループをリスニングする

```
DECLARE
    Agent_w_msg      aq$_agent;
    My_agent_list     dbms_aq.agent_list_t;

BEGIN
    /* 注意 : MCQ1、MCQ2、MCQ3 は、SCOTT のスキーマの複数コンシューマ・キューです。
     *      SCQ1、SCQ2、SCQ3 は、SCOTT のスキーマの単一コンシューマ・キューです。*/
    Qlist(1) := aq$_agent('agent1', 'MCQ1', NULL);
    Qlist(2) := aq$_agent(NULL, 'scott.SQ1', NULL);
    Qlist(3) := aq$_agent('agent3', 'scott.MCQ3', NULL);
    /* タイムアウト 100 秒でリスニングします。*/
    DBMS_AQ.LISTEN(
        Agent_list => My_agent_list,
        Wait       => 100,
        Agent       => agent_w_msg);
```

```

DBMS_OUTPUT.PUT_LINE('Message in Queue :- ' || agent_w_msg.address
                      || 'for agent' || agent_w_msg.name);
DBMS_OUTPUT.PUT_LINE('');
END;
/

```

## C (OCI) : 複数コンシューマ・キューのリスニング

### タイムアウト 0（ゼロ）、120 秒および 100 秒で複数コンシューマ・キューをリスニングする

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    switch (status)
    {
    case OCI_SUCCESS:
        break;
    case OCI_SUCCESS_WITH_INFO:
        printf("Error - OCI_SUCCESS_WITH_INFO\n");
        break;
    case OCI_NEED_DATA:
        printf("Error - OCI_NEED_DATA\n");
        break;
    case OCI_NO_DATA:
        printf("Error - OCI_NO_DATA\n");
        break;
    case OCI_ERROR:
        OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                     errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    }
}

```

```

        printf("Error - %s\n", errbuf);
        break;
    case OCI_INVALID_HANDLE:
        printf("Error - OCI_INVALID_HANDLE\n");
        break;
    case OCI_STILL_EXECUTING:
        printf("Error - OCI_STILL_EXECUTE\n");
        break;
    case OCI_CONTINUE:
        printf("Error - OCI_CONTINUE\n");
        break;
    default:
        break;
    }
}

void SetAgent(OCIAQAgent *agent,
              text      *appname,
              text      *queue,
              OCIError   *errhp,
              OCIEnv     *envhp);

void GetAgent(OCIAQAgent *agent,
              OCIError   *errhp);

/*-----*/
/* OCI での複数コンシューマのリスニングの例 */
/*-----*/
void SetAgent(agent, appname, queue, errhp)
OCIAQAgent   *agent;
text         *appname;
text         *queue;
OCIError     *errhp;
{
    OCIAttrSet(agent,
                OCI_DTYPE_AQAGENT,
                appname ? (dvoid *)appname : (dvoid *)"",
                appname ? strlen((const char *)appname) : 0,
                OCI_ATTR_AGENT_NAME,
                errhp);

    OCIAttrSet(agent,
                OCI_DTYPE_AQAGENT,
                queue ? (dvoid *)queue : (dvoid *)"",
                queue ? strlen((const char *)queue) : 0,

```

```

        OCI_ATTR_AGENT_ADDRESS,
        errhp);

    printf("Set agent name to %s\n", appname ? (char *)appname : "NULL");
    printf("Set agent address to %s\n", queue ? (char *)queue : "NULL");
}

/* 記述子からエージェントを取得します。*/
void GetAgent(agent, errhp)
OCIAQAgent *agent;
OCIError *errhp;
{
    text      *appname;
    text      *queue;
    ub4       appsz;
    ub4       queuesz;

    if (!agent)
    {
        printf("agent was NULL \n");
        return;
    }
    checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
        (dvoid *)&appname, &appsz, OCI_ATTR_AGENT_NAME, errhp));
    checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
        (dvoid *)&queue, &queuesz, OCI_ATTR_AGENT_ADDRESS, errhp));
    if (!appsz)
        printf("agent name: NULL\n");
    else printf("agent name: %.*s\n", appsz, (char *)appname);
    if (!queuesz)
        printf("agent address: NULL\n");
    else printf("agent address: %.*s\n", queuesz, (char *)queue);
}

/* from AQ Listenから複数コンシューマ・キューへのメイン */

/* int main() */
int main(char *argv, int argc)
{
    OCIEnv      *envhp;
    OCIServer   *srvhp;
    OCIError    *errhp;
    OCISvcCtx   *svchp;
    OCISession  *usrhp;
    OCIAQAgent  *agent_list[3];

```

```
OCIQAgent *agent;
int        i;

/* 標準的な OCI の初期化 */
OCIInitialize((ub4) OCI_OBJECT,
              (dvoid *)0,
              (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,
              (void (*)()) 0 );

OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                0, (dvoid **) 0);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 0, (dvoid **)0);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                0, (dvoid **) 0);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                0, (dvoid **) 0);

OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                0, (dvoid **) 0);

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。 */
OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
            (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* ユーザー・コンテキスト・ハンドルを割り当てます。 */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
              (size_t) 0, (dvoid **) 0);

/* ユーザー・コンテキスト・ハンドルを割り当てます。 */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
              (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
           (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION,
```



```

        (dvoid *) "tiger", (ub4) strlen("tiger"),
        (ub4) OCI_ATTR_PASSWORD, errhp);

OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS, OCI_DEFAULT);

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
        (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* AQ LISTEN の初期化 - エージェント・ハンドルを割り当てます。 */
for (i = 0; i < 3; i++)
{
    OCIDescriptorAlloc(envhp, (dvoid **)&agent_list[i],
        OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
}

/* MCQ1, MCQ2, MCQ3 は、SCOTT のスキーマの複数コンシューマ・キューです。 */
/* 複数コンシューマ・キューに対して、タイムアウト 0 でリスニングします。 */

SetAgent(agent_list[0], "app1", "MCQ1", errhp);
SetAgent(agent_list[1], "app2", "MCQ2", errhp);
SetAgent(agent_list[2], "app3", "MCQ3", errhp);

checkerr(errhp, OCIAQListen(svchp, errhp, agent_list, 3, 0, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");

/* 複数コンシューマ・キューに対して、タイムアウト 120 秒でリスニングします。 */

SetAgent(agent_list[0], "app1", "SCOTT.MCQ1", errhp);
SetAgent(agent_list[1], "app2", "SCOTT.MCQ2", errhp);
SetAgent(agent_list[2], "app3", "SCOTT.MCQ3", errhp);

checkerr(errhp, OCIAQListen(svchp, errhp, agent_list, 3, 120, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");

/* 単一コンシューマ・キューと複数コンシューマ・キューを組み合わせ、タイムアウト 100 秒でリスニングします。 */

SetAgent(agent_list[0], "app1", "SCOTT.MCQ1", errhp);
SetAgent(agent_list[1], "app2", "SCOTT.MCQ2", errhp);

```

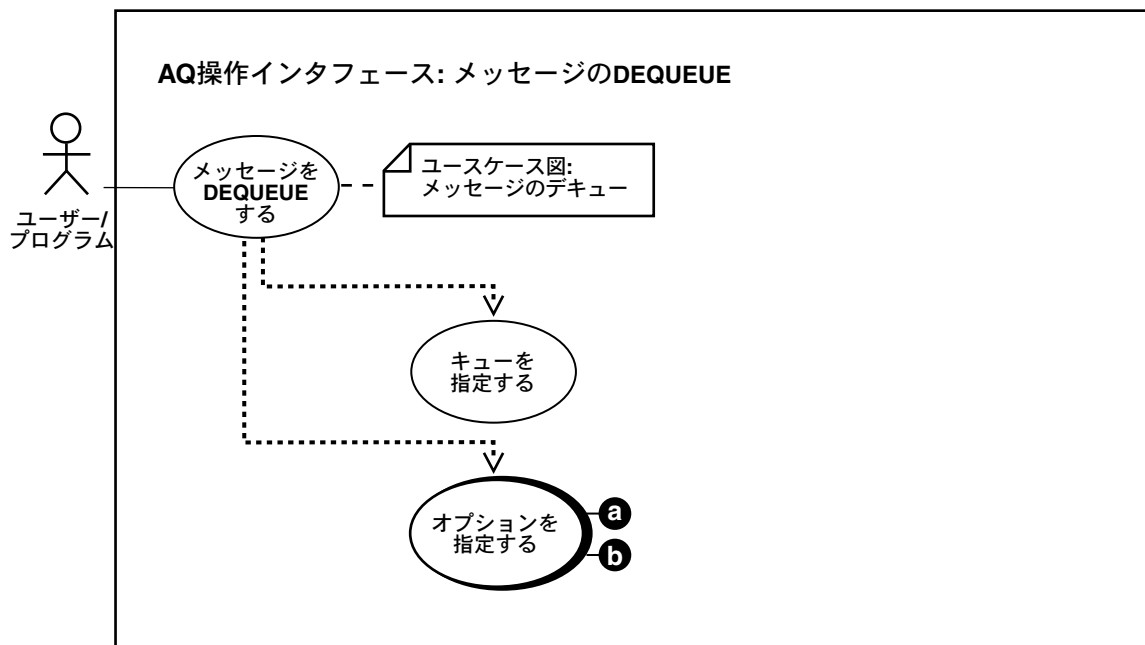
```
SetAgent(agent_list[2], (text *)0, "SCOTT.SQ3", errhp);

checkerr(errhp, OCIAGListen(svchp, errhp, agent_list, 3, 100, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");
}
```

## メッセージのデキュー

図 11-10 ユースケース図：メッセージのデキュー



### 参照：

- 管理インターフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インターフェース – 基本操作」を参照してください。

## 用途

メッセージを、指定したキューからデキューします。

## 使用上の注意

### メッセージの検索基準およびデキュー順序

- デキューするメッセージの検索基準は、デキュー・オプションのコンシューマ名、msgid および correlation パラメータによって決定されます。msgid によって、デキューされるメッセージが一意に識別されます。相関識別子は、AQ では解釈されない、アプリケーション定義の識別子です。
- msgid を指定しない限り、READY 状態のメッセージのみがデキューされます。
- デキューの順序は、デキュー・オプションの msgid および相関識別子でオーバーライドされない限り、キュー・テーブルの作成時に指定された値によって決定されます。
- キュー操作には、データベースの読みみ貫性メカニズムが適用されます。たとえば、トランザクションによる参照が始まってからエンキューされたメッセージを、BROWSE コールでは参照できません。

### キューのナビゲート

デキュー中のデフォルト NAVIGATION パラメータは、NEXT MESSAGE です。つまり、後続のデキューでは、最初のデキューで取得したスナップショットを基礎とするキューからメッセージが取り出されます。特に、最初のデキュー・コマンドの後にエンキューされたメッセージは、キューに残っているメッセージがすべて処理されてから処理されます。すべてのメッセージがすでにキューにエンキューされている場合、またはキューに優先順位がない場合にはこのような処理で十分です。ただし、デキュー・コマンドごとにキューの最初のメッセージを処理する必要がある場合は、アプリケーションで FIRST MESSAGE ナビゲーション・オプションを使用する必要があります。通常は、すでにエンキューされたメッセージの処理中に、優先順位の高いメッセージがキューに到着する場合に、このような処理が必要になります。

---

---

**注意：** 同時にエンキューされたメッセージがある場合には、FIRST\_MESSAGE ナビゲーション・オプションを使用するとより効率的です。FIRST\_MESSAGE オプションが指定されないと、AQ では最初のデキュー・コマンドと同様のスナップショットを継続して生成する必要があり、パフォーマンスの低下につながります。FIRST\_MESSAGE オプションが指定されると、AQ では、すべてのデキュー・コマンドに新しいスナップショットを使用します。

---

---

## メッセージのグループ化によるデキュー

- メッセージをグループ化できるキューに同一トランザクションのメッセージがエンキューされると、グループが形成されます。トランザクションに1つのメッセージしかエンキューされていない場合でも、効率的に1つのグループが形成されます。1つのトランザクションでグループ化できるメッセージの数に、上限はありません。
- メッセージをグループ化できないキューでは、LOCKED または REMOVE モードのキューによって1つのメッセージのみがロックされます。それに対して、グループの一部であるメッセージをデキューする操作では、グループ全体がロックされます。これは、グループのメッセージ全体をアトミック単位として処理する必要がある場合に有効です。
- グループのメッセージ全体をデキューした後でデキュー操作をすると、グループのすべてのメッセージが処理されたことを示すエラーが戻されます。その後、アプリケーションは NEXT TRANSACTION を使用して、次の使用可能なグループからのメッセージのデキューを開始します。使用可能なグループがない場合、指定された WAIT 時間の経過後、デキューはタイムアウトになります。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第4章「DBMS\_AQ」の DEQUEUE プロシージャ
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ」を選択してください。
- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQQueue」の dequeue

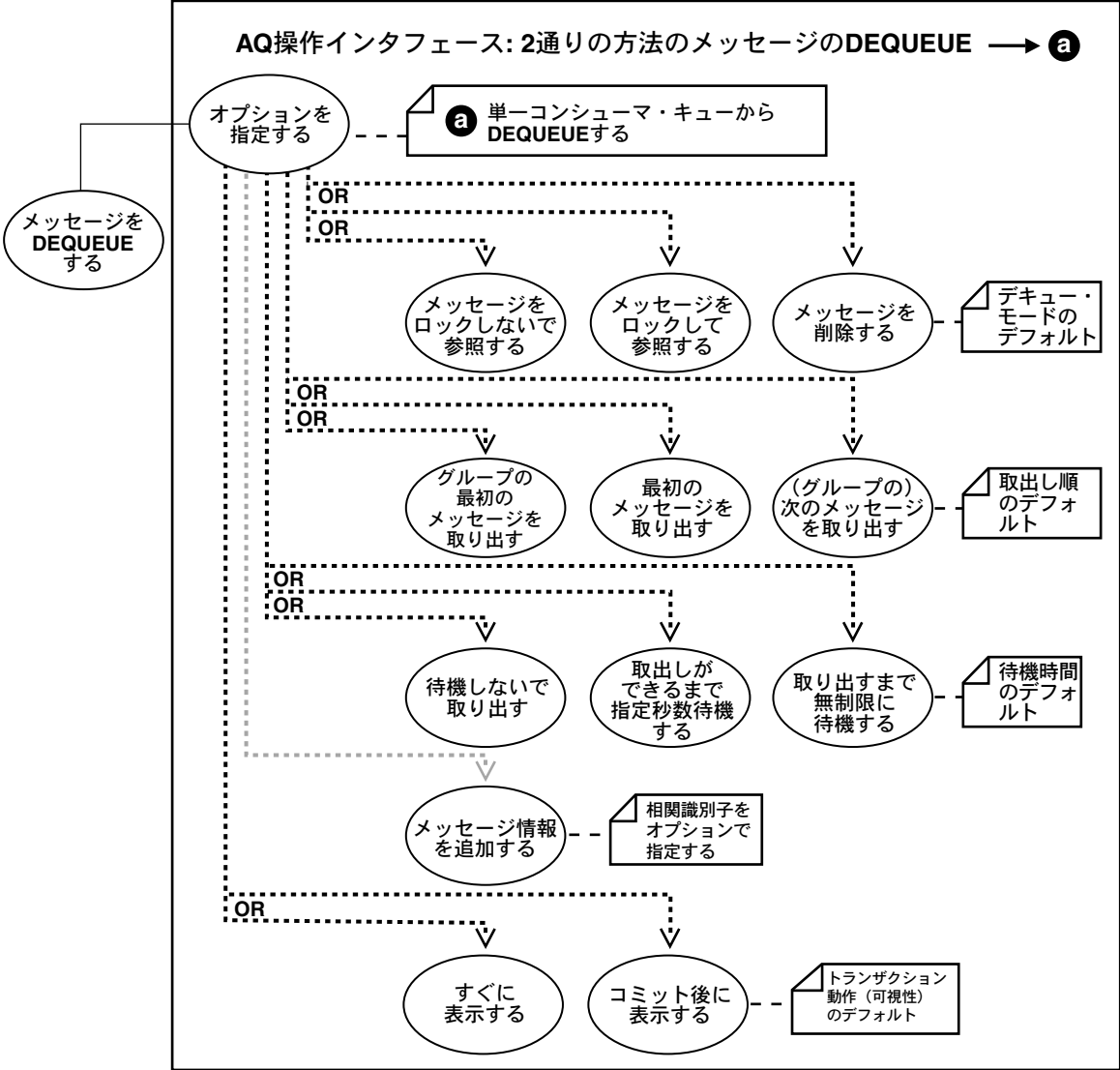
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS\\_AQ パッケージ\) : オブジェクト型メッセージのデキュー \(11-50 ページ\)](#)
- [Java \(JDBC\) : 単一コンシューマ・キューからのメッセージのデキュー \(オプションの指定\) \(11-50 ページ\)](#)
- [Visual Basic \(OO4O\) : メッセージのデキュー \(11-51 ページ\)](#)

# 単一コンシューマ・キューからのメッセージのデキュー（オプションの指定）

図 11-11 ユースケース図：単一コンシューマ・キューからのメッセージのデキュー



## 用途

---

### 参照：

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インタフェース – 基本操作」を参照してください。
- 

デキュー操作で使用可能なオプションを指定します。

## 使用上の注意

一般的には、メッセージのコンシューマはデキュー・インタフェースによってメッセージにアクセスすると考えられます。処理済または処理が終了していないメッセージは、メッセージ ID または SELECT 文を使用して参照できます。

## 構文

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第4章「DBMS\_AQ」の DEQUEUE プロシージャ
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ)：「Help」の「Constents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ」を選択してください。
- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第1章「パッケージ oracle.AQ」の「AQDequeueOption」

## 例

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。次のプログラム環境の例が示されています。

- PL/SQL (DBMS\_AQ パッケージ)：オブジェクト型メッセージのデキュー（11-50 ページ）
- Java (JDBC)：単一コンシューマ・キューからのメッセージのデキュー（オプションの指定）（11-50 ページ）
- Visual Basic (OO4O)：メッセージのデキュー（11-51 ページ）

## PL/SQL（DBMS\_AQ パッケージ）：オブジェクト型メッセージのデキュー

```
/* msg_queue からデキューします。*/  
DECLARE  
  dequeue_options    dbms_aq.dequeue_options_t;  
  message_properties dbms_aq.message_properties_t;  
  message_handle      RAW(16);  
  message             aq.message_typ;  
  
BEGIN  
  DBMS_AQ.DEQUEUE(  
    queue_name      => 'msg_queue',  
    dequeue_options => dequeue_options,  
    message_properties => message_properties,  
    payload          => message,  
    msgid            => message_handle);  
  
  DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||  
                        ' ... ' || message.text );  
  
  COMMIT;  
END;
```

## Java（JDBC）：単一コンシューマ・キューからのメッセージのデキュー（オプションの指定）

```
/* メッセージを相関識別子 'RUSH' でデキューします。*/  
public static void example(AQSession aq_sess) throws AQException, SQLException  
{  
  AQQueue          queue;  
  AQMessage        message;  
  AQRawPayload     raw_payload;  
  AQDequeueOption  deq_option;  
  byte[]           b_array;  
  Connection        db_conn;  
  
  db_conn = ((AQOracleSession) aq_sess).getDBConnection();  
  
  queue = aq_sess.getQueue ("aq", "msg_queue");  
  
  /* AQDequeueOption オブジェクトをデフォルトのオプション付きで作成します。*/  
  deq_option = new AQDequeueOption();  
  
  deq_option.setCorrelation("RUSH");  
  
  /* メッセージをデキューします。*/  
}
```



```
message = queue.dequeue(deq_option);

System.out.println("Successful dequeue");

/* メッセージから RAW データを取り出します。*/
raw_payload = message.getRawPayload();

b_array = raw_payload.getBytes();

db_conn.commit();
}
```

## Visual Basic (0040) : メッセージのデキュー

### RAW 型メッセージのエンキュー

```
'Dequeue the first message available
Q.Dequeue()
Set Msg = Q.QMsg

'Display the message content
MsgBox Msg.Value

'Dequeue the first message available without removing it
' from the queue
Q.DequeueMode = ORAAQ_DEQ_BROWSE

'Dequeue the first message with the correlation identifier
' equal to "RELATIVE_MSG_ID"
Q.Navigation = ORAAQ_DQ_FIRST_MSG
Q.correlate = "RELATIVE_MESSAGE_ID"
Q.Dequeue

'Dequeue the next message with the correlation identifier
' of "RELATIVE_MSG_ID"
Q.Navigation = ORAAQ_DQ_NEXT_MSG
Q.Dequeue()

'Dequeue the first high priority message
Msg.Priority = ORAQMSG_HIGH_PRIORITY
Q.Dequeue()

'Dequeue the message enqueued with message id of Msgid_1
```

```
Q.DequeueMsgid = Msgid_1
Q.Dequeue()

'Dequeue the message meant for "ANDY"
Q.consumer = "ANDY"
Q.Dequeue()

'Return immediately if there is no message on the queue
Q.wait = ORAQ_DQ_NOWAIT
Q.Dequeue()
```

### オブジェクト型メッセージのデキュー

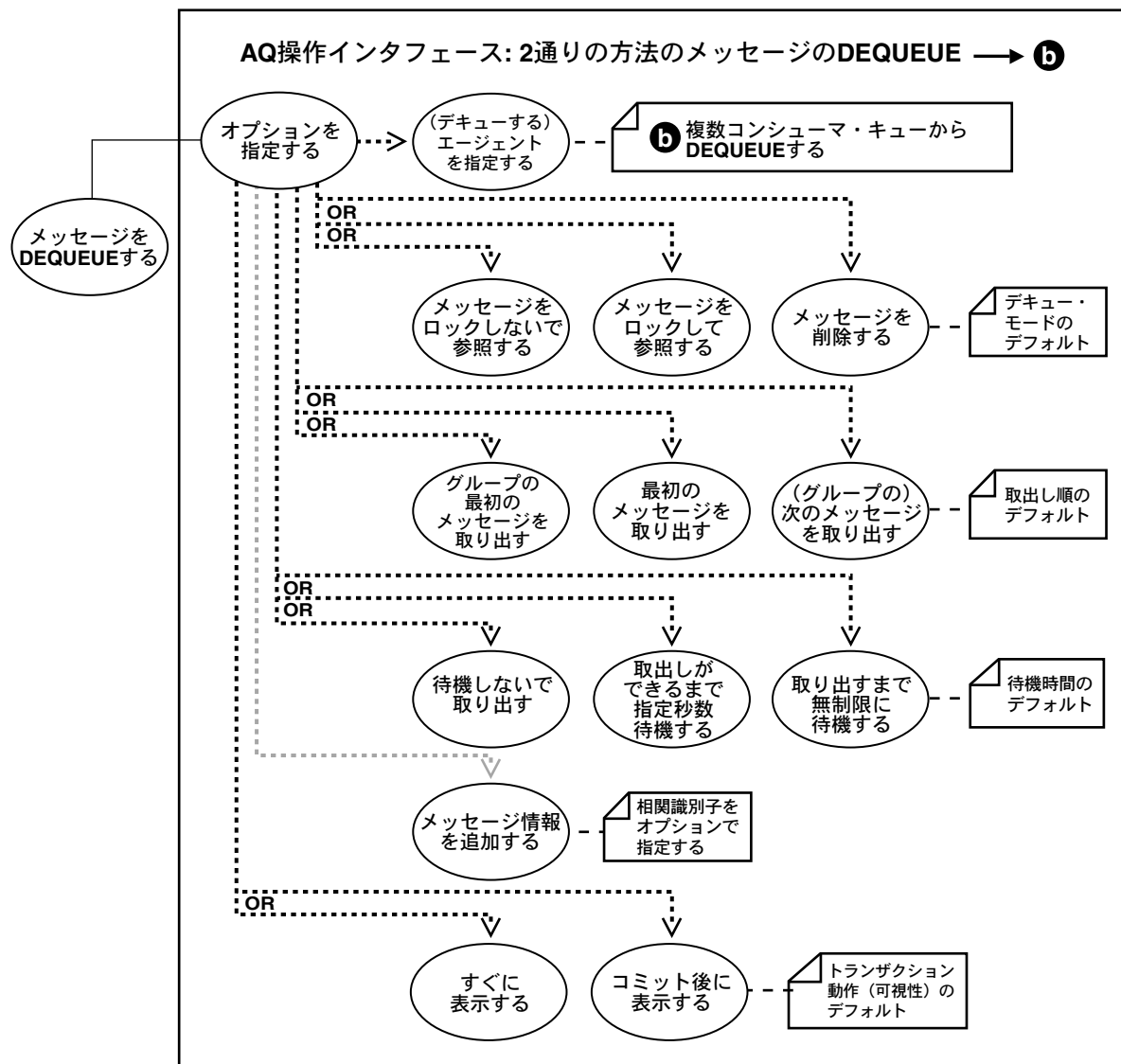
```
Set OraObj = DB.CreateOraObject("MESSAGE_TYPE")
Set QMsg = Q.AQMsg(1, "MESSAGE_TYPE")

'Dequeue the first message available without removing it
Q.Dequeue()
OraObj = QMsg.Value

'Display the subject and data
MsgBox OraObj!subject & OraObj!Data
```

## 複数コンシューマ・キューからのメッセージのデキュー（オプションの指定）

図 11-12 ユースケース図：複数コンシューマ・キューからのメッセージのデキュー



---

---

**参照：**

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「[ユースケース・モデル：操作インタフェース – 基本操作](#)」を参照してください。
- 
- 

## 用途

デキュー操作で使用可能なオプションを指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャ リファレンス』の第 4 章「DBMS\_AQ」の DEQUEUE プロシージャ
- Visual Basic: 参照マニュアルはありません。
- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 1 章「パッケージ oracle.AQ」の「AQDequeueOption」

## 例

次のプログラム環境の例が示されています。

- [Java \(JDBC\)：単一コンシューマ・キューからのメッセージのデキュー（オプションの指定）](#) (11-55 ページ)

## Java（JDBC）：単一コンシューマ・キューからのメッセージのデキュー（オプションの指定）

```
/* BROWSE モードで subscriber1 にメッセージをデキューします。*/
public static void example(AQSession aq_sess) throws AQException, SQLException
{
    AQQueue          queue;
    AQMessage        message;
    AQRawPayload     raw_payload;
    AQDequeueOption  deq_option;
    byte[]           b_array;
    Connection       db_conn;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    queue = aq_sess.getQueue ("aq", "priority_msg_queue");

    /*AQDequeueOption オブジェクトをデフォルトのオプション付きで作成します。*/
    deq_option = new AQDequeueOption();

    /* デキュー・モードを BROWSE に設定します。*/
    deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_BROWSE);

    /* subscriber1 にメッセージをデキューします。*/
    deq_option.setConsumerName("subscriber1");

    /* メッセージをデキューします。*/
    message = queue.dequeue(deq_option);

    System.out.println("Successful dequeue");

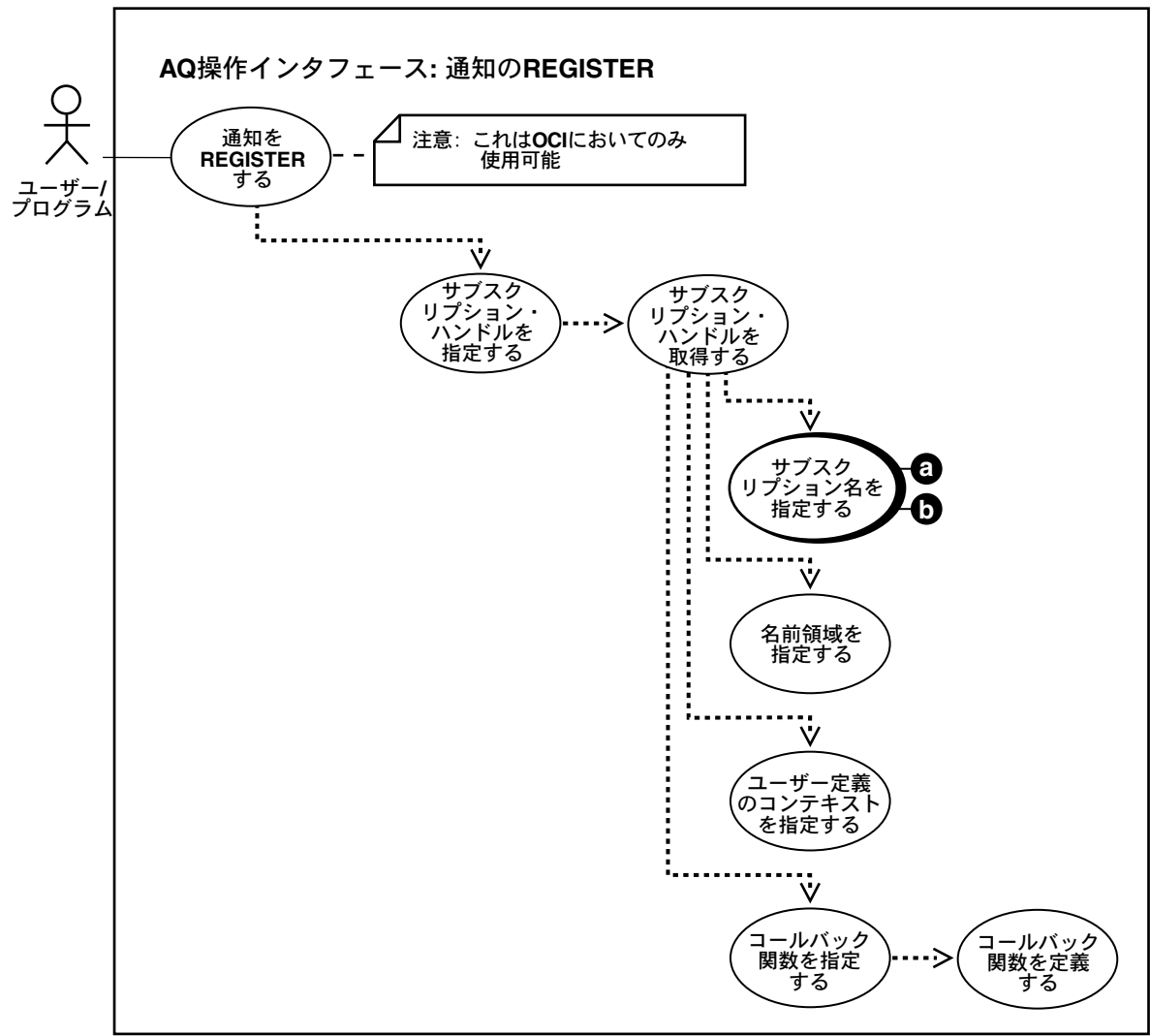
    /* メッセージから RAW データを取り出します。*/
    raw_payload = message.getRawPayload();

    b_array = raw_payload.getBytes();

    db_conn.commit();
}
```

# 通知登録

図 11-13 ユースケース図：通知登録



---

---

**参照：**

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インタフェース – 基本操作」を参照してください。
- 
- 

## 用途

メッセージ通知のコールバックを登録します。

## 使用上の注意

- このコールは、関心があるサブスクリプション名、および起動されるように対応付けられたコールバックを識別するサブスクリプション登録に対して起動されます。一度に、関心があるサブスクリプションを複数個登録できます。
- このインタフェースは、メッセージ配信が非同期モードで行われるときのみ有効です。このモードで、サブスクライバはコールバックを指定する登録呼出しを発行します。サブスクリプションの基準に一致するメッセージが届くと、そのコールバックが起動します。次に、そのコールバックは、メッセージを取り出す明示的な `message_receive` (デキュー) を発行できます。
- ユーザーは登録の際に、`OCI_SUBSCR_NAMESPACE_AQ` に設定された名前領域属性によって、サブスクリプション・ハンドルを指定する必要があります。
- サブスクリプション名は、単一コンシューマ・キューに対する登録の場合は「`schema.queue`」という文字列で、複数コンシューマ・キューに対する登録の場合は「`schema.queue:consumer_name`」になります。
- 関連関数は、`OCIAQListen()`、`OCISubscriptionDisable()`、`OCISubscriptionEnable()` および `OCISubscriptionUnRegister()` です。

---

---

**参照：**

- OCI の通知登録オペレーションの詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。
- 
-

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS\_AQ パッケージ) : 使用できません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第9章「OCI プログラミングの応用」の「パブリッシュ / サブスクライブの関数」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Constents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ Object」>「Monitoring Messages」を選択してください。
- Java (JDBC) : 使用できません。

## 例

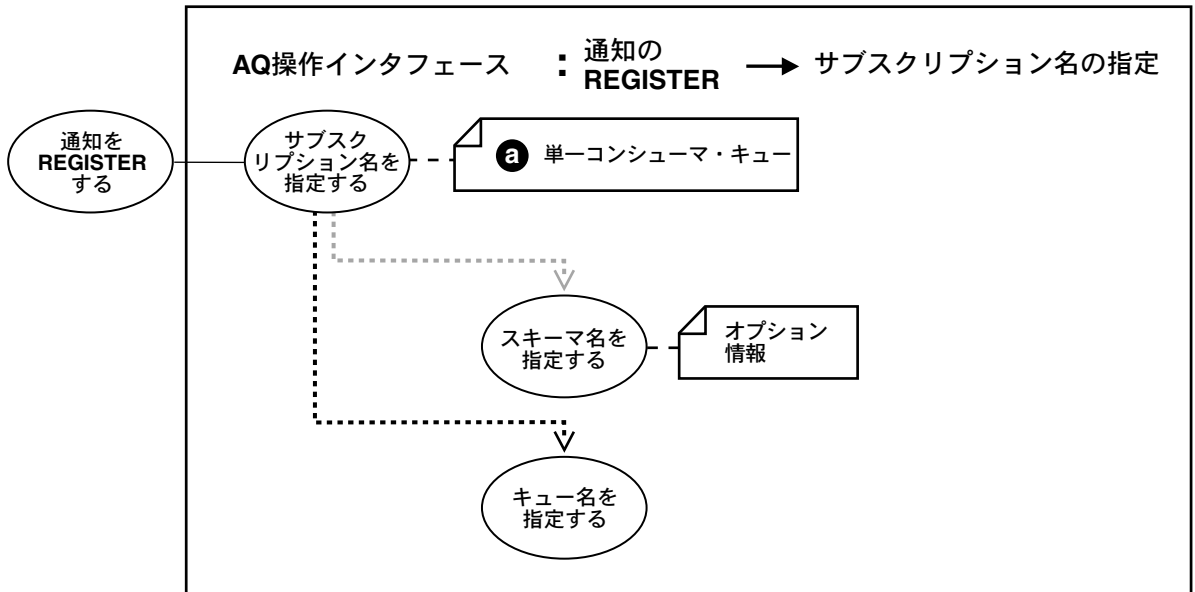
各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。次のプログラム環境の例が示されています。

- [C \(OCI\) : 単一コンシューマおよび複数コンシューマ・キューへの通知登録](#) (11-61 ページ)



## 通知登録（サブスクリプション名の指定 - 単一コンシューマ・キュー）

図 11-14 ユースケース図：サブスクリプション名の指定 - 単一コンシューマ・キュー

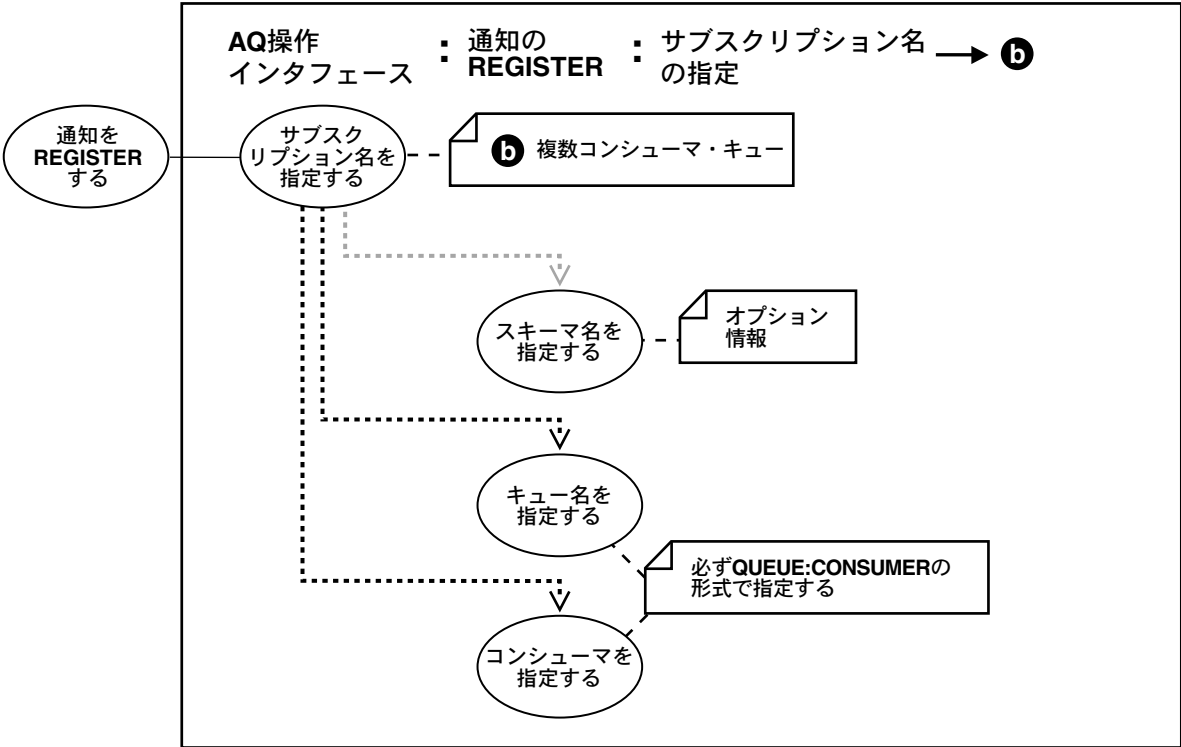


### 参照：

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インタフェース - 基本操作」を参照してください。

# 通知登録（サブスクリプション名の指定 - 複数コンシューマ・キュー）

図 11-15 ユースケース図：サブスクリプション名の指定 - 複数コンシューマ・キュー



**参照：**

- 管理インタフェースに関するすべての基本操作については、11-2 ページの「ユースケース・モデル：操作インタフェース - 基本操作」を参照してください。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL（DBMS\_AQ パッケージ）：使用できません。
- C（OCI）：『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の「OCI プログラミングの高度なトピック」の「パブリッシュ / サブスクライブ通知」
- Visual Basic（Oracle Objects for OLE オンライン・ヘルプ）：「Help」の「Constants」タブから、「OO4O Automation Server」>「OBJECTS」>「OraAQ Object」>「Monitoring Messages」を選択してください。
- Java（JDBC）：使用できません。

## 例

各プログラム環境で使用可能な機能のリストは、第3章「AQ プログラム環境」を参照してください。

## C（OCI）：単一コンシューマおよび複数コンシューマ・キューへの通知登録

/\* OCISubscription を使用すると、クライアントは、メッセージが非永続キューおよび通常のキューにエンキューされたときの通知の受信を登録できます。\*/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>
```

```
static OCISvcCtx *envhp;
static OCIError *errhp;
static OCISvcCtx *svchp;
```

/\* 通知によって起動されるコールバック \*/

```
ub4 notifyCB(ctx, subscrhp, pay, payl, desc, mode)
dvoid *ctx;
OCISubscription *subscrhp; /* サブスクリプション・ハンドル */
dvoid *pay; /* ペイロード */
ub4 payl; /* ペイロードの長さ */
dvoid *desc; /* AQ 通知記述子 */
```

```

ub4          mode;
{
    text          *subname;
    ub4          size;
    ub4          *number = (ub4 *)ctx;
    text          *queue;
    text          *consumer;
    OCIRaw        *msgid;
    OCIAQMsgProperties *msgprop;

    (*number)++;

    /* サブスクリプション名を取得します。*/
    OCIAttrGet((dvoid *)subscrhp, OCI_HTYPE_SUBSCRIPTION,
                (dvoid *)&subname, &size,
                OCI_ATTR_SUBSCR_NAME, errhp);
    printf("got notification number %d for %.*s %d \n",
           *number, size, subname, payl);

    /* AQ 通知記述子からキュー名を取得します。*/
    OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&queue, &size,
                OCI_ATTR_QUEUE_NAME, errhp);

    /* この通知を受信したコンシューマ名を取得します。*/
    OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&consumer, &size,
                OCI_ATTR_CONSUMER_NAME, errhp);

    /* 通知されたメッセージのメッセージ ID を取得します。*/
    OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgid, &size,
                OCI_ATTR_NFY_MSGID, errhp);

    /* 通知されたメッセージのメッセージ・プロパティを取得します。*/
    OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgprop, &size,
                OCI_ATTR_MSG_PROP, errhp);
}

int main(argc, argv)
int argc;
char *argv[];
{
    OCISession *authp = (OCISession *) 0;

    /* サブスクリプション・ハンドル */

```

```

OCISubscription *subscrihp[5];

/* AQ 名前空間用の登録です。*/
ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;

/* コールバック用のコンテキストです。*/
ub4 ctx[5] = {0,0,0,0,0};

printf("Initializing OCI Process\n");

/* OCI プロセス環境は、OCI_EVENTS で初期化する必要があります。*/
/* OCI_OBJECT フラグは、デキュー可能に設定されます。*/
(void) OCIInitialize((ub4) OCI_EVENTS|OCI_OBJECT, (dvoid *)0,
                    (dvoid * (*)(dvoid *, size_t)) 0,
                    (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                    (void (*)(dvoid *, dvoid *)) 0 );

printf("Initialization successful\n");

/* 標準的な OCI の設定 */
printf("Initializing OCI Env\n");
(void) OCIEnvInit((OCIEnv **) &envhp, OCI_DEFAULT, (size_t) 0,
                (dvoid **) 0 );

(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
                    (size_t) 0, (dvoid **) 0);

/* サーバー・コンテキスト */
(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
                    (size_t) 0, (dvoid **) 0);

(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
                    (size_t) 0, (dvoid **) 0);

printf("connecting to server\n");
(void) OCIServerAttach( srvhp, errhp, (text *)"", strlen(""), 0);
printf("connect successful\n");

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
(void) OCIAttrSet( (dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid *)srvhp,
                (ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp);

(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&authp,
                    (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0);

```

```

(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                  (dvoid *) "scott", (ub4) strlen("scott"),
                  (ub4) OCI_ATTR_USERNAME, errhp);

(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                  (dvoid *) "tiger", (ub4) strlen("tiger"),
                  (ub4) OCI_ATTR_PASSWORD, errhp);

checkerr(errhp, OCISessionBegin ( svchp, errhp, authp, OCI_CRED_RDEMS,
                                  (ub4) OCI_DEFAULT));

(void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
                  (dvoid *) authp, (ub4) 0,
                  (ub4) OCI_ATTR_SESSION, errhp);

/* NORMAL 単一コンシューマ・キューへの通知用にサブスクリプション・ハンドルを設定します。*/
printf("allocating subscription handle\n");
subscrhp[0] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[0],
                      (ub4) OCI_HTYPE_SUBSCRIPTION,
                      (size_t) 0, (dvoid **) 0);

printf("setting subscription name\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) "SCOTT.SQ1", (ub4) strlen("SCOTT.SQ1"),
                  (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

printf("setting subscription callback\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) notifyCB, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

printf("setting subscription context \n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) &ctx[0], (ub4) sizeof(ctx[0]),
                  (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

printf("setting subscription namespace\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) &namespace, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

```

```

/* NORMAL 複数コンシューマ・キューへの通知用にサブスクリプション・ハンドルを設定します。*/
subscrhp[1] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[1],
    (ub4) OCI_HTYPE_SUBSCRIPTION,
    (size_t) 0, (dvoid **) 0);

(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
    (dvoid *) "SCOTT.MQ1:APP1",
    (ub4) strlen("SCOTT.MQ1:APP1"),
    (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
    (dvoid *) notifyCB, (ub4) 0,
    (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
    (dvoid *)&ctx[1], (ub4) sizeof(ctx[1]),
    (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
    (dvoid *) &namespace, (ub4) 0,
    (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

/* 非永続単一コンシューマ・キューへの通知用にサブスクリプション・ハンドルを設定します。*/
subscrhp[2] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[2],
    (ub4) OCI_HTYPE_SUBSCRIPTION,
    (size_t) 0, (dvoid **) 0);

(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
    (dvoid *) "SCOTT.NP_SCQ1",
    (ub4) strlen("SCOTT.NP_SCQ1"),
    (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
    (dvoid *) notifyCB, (ub4) 0,
    (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
    (dvoid *)&ctx[2], (ub4) sizeof(ctx[2]),
    (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,

```

```
(dvoid *) &namespace, (ub4) 0,
(ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

/* 非永続複数コンシューマ・キューへの通知用にサブスクリプション・ハンドルを設定します。*/
/* 受信者として指定されたユーザーを待ちます。*/
subscrhp[3] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[3],
(ub4) OCI_HTYPE_SUBSCRIPTION,
(size_t) 0, (dvoid **) 0);

(void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
(dvoid *) "SCOTT.NP_MCQ1",
(ub4) strlen("SCOTT.NP_MCQ1"),
(ub4) OCI_ATTR_SUBSCR_NAME, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
(dvoid *) notifyCB, (ub4) 0,
(ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
(dvoid *)&ctx[3], (ub4) sizeof(ctx[3]),
(ub4) OCI_ATTR_SUBSCR_CTX, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
(dvoid *) &namespace, (ub4) 0,
(ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

printf("Registering for all the subscription\n");
checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 4, errhp,
OCI_DEFAULT));

printf("Waiting for notifications\n");

/* 通知を待ちます。*/
sleep(300);

printf("Exiting\n");
}
```



---

## JMS を使用したアプリケーションの作成

第1章では、架空の会社 BooksOnLine を基にメッセージ・システムを説明しました。この章では、そのシナリオに基づいたサンプル・アプリケーションのコンテキストに沿って、AQ への Oracle JMS インタフェース機能について説明します。

- サンプル・アプリケーション
- 一般的な機能
  - JMS 接続およびセッション
    - \* コネクション・ファクトリ
    - \* 接続
    - \* セッション
  - JMS 宛先 - キューおよびトピック
  - システム・レベルのアクセス制御
  - 宛先レベルのアクセス制御
  - 保存およびメッセージ履歴
  - Oracle Parallel Server のサポート
  - 統計ビューのサポート
  - 構造化ペイロード / メッセージの型
    - \* メッセージ・ヘッダー
    - \* メッセージ・プロパティ
    - \* メッセージ本体
    - \* Stream メッセージ

- 
- \* Byte メッセージ
  - \* Map メッセージ
  - \* Text メッセージ
  - \* Object メッセージ
  - \* ADT メッセージ
  - \* 異なるメッセージ型でのメッセージ・プロパティの使用
  - JMS サンプルで使ったペイロード
  - Point-to-Point モデル機能
    - キュー
    - キュー送信者
    - キュー受信者
    - キュー・ブラウザ
  - パブリッシュ / サブスクライブ・モデル機能
    - トピック
    - 永続サブスクライバ
    - トピック・パブリッシャ
    - 受信者リスト
    - トピック受信者
  - メッセージ・プロデューサ機能
    - メッセージの優先順位および順序付け
    - 時刻指定 : 遅延
    - 時刻指定 : 期限切れ
    - メッセージのグループ化
  - メッセージ・コンシューマ機能
    - メッセージの受信
    - 受信におけるメッセージのナビゲーション
    - メッセージ受信モード
    - 遅延間隔をおいての再試行

- 
- メッセージ・リスナーを使用したメッセージの非同期受信
  - AQ の例外処理
  - 伝播
    - リモート・サブスクライバ
    - 伝播スケジュール
    - 拡張伝播スケジュール機能
    - 伝播中の例外処理

## サンプル・アプリケーション

大型の書籍販売店である BooksOnLine の運営は、販売プロセス全体に関わる様々な部署にまたがるアクティビティを自動化する、オンライン書籍発注システムが基礎となっています。システムのフロント・エンドは、新しい発注を入力するための注文入力アプリケーションです。入力された注文は、注文処理アプリケーションによって妥当性チェックおよび記録されます。地域倉庫に置かれた出荷部門は、発注された書籍をすぐに確実に発送する責任があります。地域倉庫は3箇所、それぞれが東部地域、西部地域、そして海外からの発注に対応しています。発注された書籍の発送完了後、発注情報は支払処理のために中央の請求部門に送られます。各地の顧客サービス部門は、発注情報の状況を管理し、発注に関する問合せを処理する責任があります。

第1章では、架空の会社 BooksOnLine を基にメッセージ・システムの概要を示しました。この章では、そのシナリオに基づいたサンプル・アプリケーションのコンテキストに沿って、AQ への JMS インタフェースの機能について検討します。このサンプル・アプリケーションは、Oracle AQ の機能を説明する目的でのみ作成されたものです。この統合化シナリオを作成するにあたって、単一のコンテキストに絞った説明をすることで、このテクノロジーの可能性を理解しやすくするよう努めました。ただし、考えられる AQ のすべての応用例を、単一の比較的小さいコード・サンプルの範囲内で示すことは不可能なことを理解しておいてください。

## 一般的な機能

- JMS 接続およびセッション
  - コネクション・ファクトリ
  - 接続
  - セッション
- JMS 宛先 - キューおよびトピック
- システム・レベルのアクセス制御
- 宛先レベルのアクセス制御
- 保存およびメッセージ履歴
- Oracle Parallel Server のサポート
- 統計ビューのサポート
- 構造化ペイロード / メッセージの型
  - メッセージ・ヘッダー
  - メッセージ・プロパティ
  - メッセージ本体
  - Stream メッセージ
  - Byte メッセージ
  - Map メッセージ
  - Text メッセージ
  - Object メッセージ
  - ADT メッセージ

## JMS 接続およびセッション

### コネクション・ファクトリ

`ConnectionFactory` は、管理者によって定義された接続構成パラメータの集合をカプセル化します。クライアントは、これを使用して JMS プロバイダとの接続を確立します。Oracle JMS では、Oracle8i が JMS プロバイダです。

`ConnectionFactory` オブジェクトには、次の 2 種類があります。

- `QueueConnectionFactory`
- `TopicConnectionFactory`

`AQjmsFactory` クラスを使用すると、`QueueConnectionFactory` または `TopicConnectionFactory` オブジェクトに対するハンドルを取得できます。

**QueueConnectionFactory** キュー・コネクション・ファクトリを取得するには、次の関数を使用します。

```
AQjmsFactory.getQueueConnectionFactory(...)
```

キュー・コネクション・ファクトリは、ホスト名、ポート番号、SID を使用するか、JDBC URL およびプロパティを使用して作成できます。

**TopicConnectionFactory** トピック・コネクション・ファクトリを取得するには、次の関数を使用します。

```
AQjmsFactory.getTopicConnectionFactory(...)
```

トピック・コネクション・ファクトリは、ホスト名、ポート番号、SID を使用するか、JDBC URL およびプロパティを使用して作成できます。

**JMS 接続** データベースへの接続を確立するには、まず、次の関数を起動します。

```
createQueueConnection(...)  
createTopicConnection(...)
```

このとき、パラメータとしてユーザー名およびパスワードを、それぞれ `QueueConnectionFactory` オブジェクトおよび `TopicConnectionFactory` オブジェクトに明示的に渡します。

### 接続

JMS 接続は、クライアントによる JMS プロバイダへのアクティブな接続を表します。接続によって、次に示すいくつかの重要なサービスが実行されます。

- JMS プロバイダとのオープン接続をカプセル化します。

- クライアントとプロバイダのサービス・デーモン間のオープンな TCP/IP ソケットを表します。
- 接続確立時に、クライアントを認証するための構造を提供します。
- セッションを作成します。
- 接続メタデータを提供します。

### 接続の設定

JMS クライアントは、1つの接続、1つのセッション、および多数のメッセージ・プロデューサとメッセージ・コンシューマを作成します。現在のバージョンでは、1つの接続につき1つのオープン・セッションのみが許可されています。

接続は、確立時は停止モードです。この状態では、メッセージを接続に配信できません。通常、設定が完了するまで、接続は停止モードのままです。設定が完了した時点で、接続の `start()` メソッドがコールされ、メッセージが接続コンシューマに届き始めます。この設定規則によって、設定処理中にもかかわらずメッセージが非同期配信されることによって発生するクライアントの混乱を、最小限に抑えることができます。

接続を開始してから設定を行うことは可能ですが、これを行うクライアントは、設定処理中でもメッセージの非同期配信を処理できるように準備しておく必要があります。メッセージ・プロデューサは、接続が停止中でもメッセージを送信できます。

接続オブジェクトでサポートされているメソッドのいくつかを、次に示します。

- `start()` - 接続による受信メッセージの配信を開始または再開します。
- `stop()` - 接続による受信メッセージの配信を一時的に停止します。配信が停止されると、接続のすべてのメッセージ・コンシューマへの配信が禁止されます。また、同期受信のブロックおよびメッセージは、メッセージ・リスナーに配信されません。
- `close()` - JMS セッションをクローズして、すべての対応付けられたリソースを解放します。
- `createQueueSession(true, 0)` - キュー・セッションを作成します。現在、トランザクション処理済のセッションのみが許可されています。
- `createTopicSession(true, 0)` - トピック・セッションを作成します。現在、トランザクション処理済のセッションのみが許可されています。

### セッション

接続は、JMS プロバイダへの基本接続を使用してメッセージを作成および処理するセッションのファクトリです。JMS セッションは、メッセージの作成および処理用の単一スレッドのコンテキストです。これは、Java 仮想マシンの外でプロバイダ・リソースを割り当てますが、軽量の JMS オブジェクトとみなされます。

セッションには、次の役割があります。

- メッセージ・プロデューサおよびメッセージ・コンシューマのファクトリを構成します。
- 宛先オブジェクト（キュー / トピック）に対するハンドルの取得方法を提供します。
- プロバイダが最適化したメッセージ・ファクトリを提供します。
- このセッションのプロデューサおよびコンシューマにまたがる作業を組み合わせる一連のトランザクションをサポートし、これらをアトミック単位に編成します。
- セッションが処理および作成するメッセージのシリアル順序を定義します。
- セッションに登録されたメッセージ・リスナーの実行をシリアルライズ化します。

1つの接続からは1つのセッションのみを作成できます。プロバイダが、JVM 外のセッションにかわってリソースを割り当てる場合があるため、クライアントは、リソースが必要ないときは、これらをクローズする必要があります。ガーベジ・コレクションによる最終的なリソースの解放を待つ必要はありません。セッションが作成したメッセージ・プロデューサおよびメッセージ・コンシューマについても、同じことが言えます。

セッション・オブジェクトのメソッドには、次のものが含まれます。

- `commit()` - このトランザクションで実行されるすべてのメッセージをコミットして、現在保持されているロックを解放します。
- `rollback()` - トランザクションで実行されたすべてのメッセージをロールバックして、現在保持されているロックを解放します。
- `close()` - セッションをクローズします。
- `getDBConnection(...)` - 基礎となる JDBC 接続に対するハンドルを取得します。このハンドルを使用して、他の SQL DML 操作を同じセッションの一部として実行できます。このメソッドは、Oracle JMS 固有のメソッドです。

次に、JMS に対する Oracle の拡張機能をいくつか示します。どの拡張機能を使用する場合も、セッション・オブジェクトを `AQjmsSession` にキャストする必要があります。

- キュー・テーブル、キューおよびトピックは、セッション・オブジェクトから作成できます。
- `createQueueTable(...)` - キュー・テーブルを作成します。
- `getQueueTable(...)` - 既存のキュー・テーブルに対するハンドルを取得します。
- `createQueue(...)` - キューを作成します。



- `getQueue(...)` - 既存のキューに対するハンドルを取得します。
- `createTopic(...)` - トピックを作成します。
- `getTopic(...)` - 既存のトピックに対するハンドルを取得します。

次に、前述のコールの使用方法を示します。

## サンプル・コード

```
public static void bol_example(String ora_sid, String host, int port,
                               String driver)
{
    QueueConnectionFactory    qc_fact    = null;
    QueueConnection           q_conn     = null;
    QueueSession              q_sess     = null;
    AQQueueTableProperty      qt_prop    = null;
    AQQueueTable              q_table    = null;
    AQjmsDestinationProperty  dest_prop  = null;
    Queue                     queue      = null;
    BytesMessage              bytes_msg  = null;

    try
    {
        /* キュー・コネクション・ファクトリを取得します。*/
        qc_fact = AQjmsFactory.getQueueConnectionFactory(host, ora_sid,
                                                            port, driver);

        /* キュー・コネクションを作成します。*/
        q_conn = qc_fact.createQueueConnection("boluser", "boluser");

        /* キュー・セッションを作成します。*/
        q_sess = q_conn.createQueueSession(true, Session.CLIENT_ACKNOWLEDGE);

        /* キュー・コネクションを開始します。*/
        q_conn.start();

        qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_BYTES_MESSAGE");

        /* キュー・テーブルを作成します。*/
        q_table = ((AQjmsSession)q_sess).createQueueTable("boluser",
                                                            "bol_ship_queue_table",
                                                            qt_prop);

        dest_prop = new AQjmsDestinationProperty();
    }
}
```

```
/* キューを作成します。*/
queue = ((AQjmsSession)q_sess).createQueue(q_table, "bol_ship_queue",
                                             dest_prop);

/* キューを開始します。*/
((AQjmsDestination)queue).start(q_sess, true, true);

/* Byte メッセージを作成します。*/
bytes_msg = q_sess.createBytesMessage();

/* セッションをクローズします。*/
q_sess.close();

/* 接続をクローズします。*/
q_conn.close();
}
catch (Exception ex)
{
    System.out.println("Exception: " + ex);
}
}
```

## JMS 宛先 - キューおよびトピック

宛先は、クライアントがメッセージの送信先および受信元の指定に使用するオブジェクトです。宛先オブジェクトは、セッション・オブジェクトから、ドメイン固有のセッション・メソッドを使用して作成されます。宛先オブジェクトは、同時使用が可能です。

宛先オブジェクトには、キューおよびトピックの2種類があります。AQ では、これらは特定のデータベースで <schema>.<queue> にマップします。キューは AQ の単一コンシューマ・キューに、トピックは AQ の複数コンシューマ・キューにマップします。

宛先オブジェクトのメソッドには、次のものが含まれます。

- `alter(...)` - キューまたはトピックを変更します。
- `schedulePropagation(...)` - ソースから宛先への伝播をスケジュールします。
- `unschedulePropagation(...)` - 設定されていた伝播スケジュールを解除します。
- `enablePropagationSchedule (...)` - 伝播スケジュールを使用可能にします。
- `disablePropagationSchedule (...)` - 伝播スケジュールを使用不可にします。
- `start()` - キューまたはトピックを起動します。キューは、エンキューまたはデキュー（あるいはその両方）の実行のために起動できます。トピックは、パブリッシュまたはサブスクライブ（あるいはその両方）の実行のために起動できます。
- `stop()` - キューまたはトピックを停止します。キューは、エンキューまたはデキュー（あるいはその両方）を実行不可にするために停止できます。トピックは、パブリッシュまたはサブスクライブ（あるいはその両方）を実行不可にするために停止できません。
- `drop()` - キューまたはトピックを削除します。

## サンプル・コード

```
public static void setup_example(TopicSession t_sess)
{
    AQQueueTableProperty qt_prop = null;
    AQQueueTable q_table = null;
    AQjmsDestinationProperty dest_prop = null;
    Topic topic = null;
    TopicConnection t_conn = null;

    try
    {
        qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_BYTES_MESSAGE");
        /* キュー・テーブルを作成します。*/
        q_table = ((AQjmsSession)t_sess).createQueueTable("boluser",
```

```
        "bol_ship_queue_table",
        qt_prop);
dest_prop = new AQjmsDestinationProperty();
/* トピックを作成します。*/
topic = ((AQjmsSession)t_sess).createTopic(q_table, "bol_ship_queue",
        dest_prop);

/* トピックを起動します。*/
((AQjmsDestination)topic).start(t_sess, true, true);

/* トピック "boluser" から宛先 dblink "dba" への伝播をスケジュールします。*/
((AQjmsDestination)topic).schedulePropagation(t_sess, "dba", null,
        null, null, null);

/*
    ここで、いくつかの処理が行われます。
*/
/* 伝播スケジュールを解除します。*/
((AQjmsDestination)topic).unschedulePropagation(t_sess, "dba");
/* トピックを停止します。*/
((AQjmsDestination)topic).stop(t_sess, true, true, true);
/* トピックを削除します。*/
((AQjmsDestination)topic).drop(t_sess);
/* キュー・テーブルを削除します。*/
q_table.drop(true);
/* セッションをクローズします。*/
t_sess.close();
/* 接続をクローズします。*/
t_conn.close();
}
catch(Exception ex)
{
    System.out.println("Exception: " + ex);
}
}
```

## システム・レベルのアクセス制御

Oracle8i は、すべてのキューイング操作に対してシステム・レベルのアクセス制御をサポートします。この機能によって、アプリケーション設計者または DBA は、ユーザーをキュー管理者にできます。キュー/トピック管理者は、データベース上のどのキューに対しても、すべての JMS インタフェース（管理および操作）を起動できます。これによって、データベース上のキュー全体に対するすべての管理スクリプトを 1 つのスキーマで管理できるため、管理作業が単純になります。詳細は、4-4 ページの「[セキュリティ](#)」を参照してください。

### サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、DBA がデータベースのキュー管理者として BOLADM (BooksOnLine の管理者アカウント) を作成します。これによって、BOLADM はデータベース上のすべてのキューを作成、削除、管理および監視できるようになります。どのアプリケーションからでも BOLADM スキーマにエンキューまたはデキューできるように PL/SQL パッケージを作成する場合は、BOLADM にシステム権限 ENQUEUE\_ANY および DEQUEUE\_ANY を付与する必要があります。

```
CREATE USER BOLADM IDENTIFIED BY BOLADM; GRANT CONNECT, RESOURCE, aq_administrator_
role TO BOLADM;
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "BOLADM", false);
((AQjmsSession)t_sess).grantSystemPrivilege("DEQUEUE_ANY", "BOLADM", false)
;where t_sess is the session object.
```

このアプリケーションでは、AQ のプロパゲータは OE (注文入力) スキーマから WS (西部売上)、ES (東部売上) および OS (海外売上) スキーマにメッセージを移入します。次に、WS、ES および OS スキーマから CB (顧客請求) および CS (顧客サービス) スキーマにメッセージを移入します。したがって、OE、WS、ES および OS スキーマのすべてに、プロパゲータのソース・キューとなるキューがあります。

メッセージが宛先キューに届くと、ソース・キューのスキーマ名に基づいたセッションによって、新しく届いたメッセージが宛先キューにエンキューされます。つまり、ソース・キューのスキーマに、宛先キューに対するエンキュー権限を付与する必要があります。

管理を単純化するために、BooksOnLine アプリケーションでソース・キューを持つすべてのスキーマに ENQUEUE\_ANY システム権限を付与します。

```
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "OE", false);
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "WS", false);
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "ES", false);
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "OS", false);
where t_sess is the session object
```

リモートの宛先キューに伝播するために、エージェント構造体のアドレス・フィールドのデータベース・リンクに指定されたログイン・ユーザーには、`ENQUEUE ANY QUEUE` 権限を付与するか、または宛先キューに対するエンキュー権限を付与する必要があります。ただし、データベース・リンクのログイン・ユーザーが宛先のキュー・テーブルを所有している場合は、どのような明示的な権限も付与する必要はありません。

## 宛先レベルのアクセス制御

Oracle8i は、エンキューおよびデキュー操作に対するキュー・レベルまたはトピック・レベルのアクセス制御をサポートします。この機能によって、アプリケーション設計者は、あるスキーマに作成されたキューまたはトピックを、他のスキーマで実行中の他のアプリケーションから保護できます。そのキューまたはトピックが属するスキーマの外で実行しているアプリケーションには、最小限のアクセス権限を付与するのみです。キューまたはトピックに対するアクセス権限でサポートされている ENQUEUE、DEQUEUE および ALL の詳細は、4-4 ページの「[セキュリティ](#)」を参照してください。

### サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、顧客請求情報を CB および CBADM スキーマで処理します。CB（顧客請求情報）スキーマには顧客請求情報アプリケーションがあり、CBADM スキーマには、すべての関連する請求情報データがキュー・テーブルの形で格納されています。請求情報データを保護するために、請求処理アプリケーションと請求情報データは別のスキーマに常駐しています。請求処理アプリケーションは、出荷済の注文情報トピックである CBADM\_shippedorders\_topic からメッセージをデキューする以外の処理はできません。そこでメッセージを処理し、請求済の注文情報トピックである CBADM\_billedorders\_topic に新しいメッセージをエンキューします。

他のアプリケーションの不正な操作からキューを保護するために、次の 2 つの grant コールを行います。

/\* 出荷済注文キューに対するデキュー権限を顧客請求情報アプリケーションに付与します。CB アプリケーションは、請求がまだである出荷済注文を出荷済注文キューから取り出します。\*/

```
((AQjmsDestination)cbadm_shippedorders_topic).grantTopicPrivilege(t_sess, "DEQUEUE",
"CB", false);
where t_sess is the session
```

/\* 請求済キューに対するエンキュー権限を顧客請求情報アプリケーションに付与します。CB アプリケーションは、注文を処理した後、請求済注文をこのキューに置くことができます。\*/

```
((AQjmsDestination)cbadm_billedorders_topic).grantTopicPrivilege(t_sess, "ENQUEUE",
"CB", false);
```

## 保存およびメッセージ履歴

AQ では、ユーザーがメッセージをキュー・テーブルに保存できます。これは、SQL 問合せを使用してこのメッセージを問い合わせて分析できることを意味します。メッセージは、相互に関連していることがよくあります。たとえば、あるメッセージを処理した結果、他のメッセージが生成された場合、両者は関連付けられています。アプリケーション設計者としては、そのような関連を追跡することが必要な場合があります。保存機能およびメッセージ識別子とともに、メッセージ・ジャーナルが AQ によって自動作成され、追跡ジャーナルまたはイベント・ジャーナルとして参照できます。保存、メッセージ識別子および SQL 問合せの協調によって、強力なメッセージ・ウェアハウスが構築できます。

### サンプル・シナリオおよびコード

出荷処理アプリケーションで、注文の平均処理時間を判断する必要があると仮定します。この時間には、backed\_order トピックでの待機時間も含まれます。出荷キューの保存を TRUE に指定し、メッセージの相関フィールドの注文番号を指定した SQL 問合せを作成すると、出荷処理アプリケーション中の注文情報の待機時間を判断できます。

単純化のために、すでに処理された注文情報のみを分析することにします。出荷処理アプリケーション中で注文情報の処理にかかる時間は、WS\_bookedorders\_topic にエンキューされる時刻と WS\_shipped\_orders\_topic にエンキューされる時刻の差です。

```
SELECT SUM(SO.enq_time - BO.enq_time) / count (*) AVG_PRCS_TIME
FROM WS.AQ$WS_orders_pr_mqtab BO , WS.AQ$WS_orders_mqtab SO
WHERE SO.msg_state = 'PROCESSED' and BO.msg_state = 'PROCESSED'
AND SO.corr_id = BO.corr_id and SO.queue = 'WS_shippedorders_topic';
```

/\* 受注残キューの平均待機時間 \*/

```
SELECT SUM(BACK.deq_time - BACK.enq_time)/count (*) AVG_BACK_TIME
FROM WS.AQ$WS_orders_mqtab BACK
WHERE BACK.msg_state = 'PROCESSED' AND BACK.queue = 'WS_backorders_topic';
```



## Oracle Parallel Server のサポート

Oracle Parallel Server を使用すると、異なるキューを別々のインスタンスによって管理できるようにして AQ パフォーマンスを改善できます。このためには、キューを格納するキュー・テーブルに様々なインスタンス親和性（作業環境）を指定します。これによって、様々なキュー / トピックに対するキュー操作（エンキューまたはデキュー）またはトピック操作（パブリッシュ / サブスクライブ）を並行して行うことができますようになります。

AQ のキュー・モニター・プロセスは、キュー・テーブルのインスタンス親和性を継続的に監視します。キュー・モニターは指定されたプライマリ・インスタンスが使用可能な場合はそれにキュー・テーブルの所有権を割り当て、失敗した場合は指定されたセカンダリ・インスタンスに割り当てます。キュー・テーブルを所有しているインスタンスが停止した場合はいつでも、キュー・モニターはそのキュー・テーブルの所有権をセカンダリ・インスタンスに割り当てます。セカンダリ・インスタンスも使用不可な場合は他の使用可能なインスタンスに変更します。

AQ の伝播は OPS でも使用可能になりますが、これはユーザーにとっては完全に透過的です。伝播スケジュールのジョブ親和性は、それぞれのキュー・テーブルの親和性と同じ値に設定されます。このように、キュー・テーブルを所有するインスタンスに対応付けられた `job_queue_process` は、そのキュー・テーブルに格納されているキューからの伝播を処理し、`ping` 操作を最小限に抑えます。詳細は、9-65 ページの「[キューの伝播のスケジュール](#)」を参照してください。

Oracle Parallel Server (OPS) の詳細は、『Oracle8i Parallel Server セットアップおよび構成ガイド』を参照してください。

### サンプル・シナリオおよびコード

BooksOnLine の例では、注文入力 (OE) サイトでの `OE_neworders_que` および `booked_order_topic` の操作は、この 2 つのトピックが異なるインスタンスと対応付けられている場合は高速化できます。そのために、2 つのトピックを別々のキュー・テーブルに作成し、`CreateQueueTable()` コマンドでそれらのキュー・テーブルに別々の親和性を指定します。

この例では、キュー・テーブル `OE_orders_sqtab` にキュー `OE_neworders_que` を格納し、プライマリおよびセカンダリ・インスタンスはそれぞれインスタンス 1 および 2 です。キュー・テーブル `OE_orders_mqtab` にはキュー `booked_order_topic` を格納し、プライマリおよびセカンダリ・インスタンスはそれぞれインスタンス 2 および 1 です。目的は、インスタンス 1 および 2 に 2 つのキューをパラレルで管理させることです。デフォルトでは、1 つのインスタンスのみが使用可能であり、この場合、両方のキュー・テーブルの所有者インスタンスはインスタンス 1 に設定されます。ただし、OPS が正しく設定され、インスタンス 1 および 2 が使用可能な場合は、キュー・テーブル `OE_orders_sqtab` がインスタンス 1 によって所有され、他のキュー・テーブルはインスタンス 2 によって所有されます。キュー・テーブルに対するプライマリおよびセカンダリ・インスタンスの指定は、次の例に示すように `alter_queue_table()` コマンドを使用して動的に変更できます。キュー・テーブルのプライマリ、セカンダリおよび所有者インスタンスに関する情報は、`USER_QUEUE_TABLES` ビューを問い合わせて取得できます（10-25 ページの「[ユーザー・スキーマのキュー・テーブルの選択](#)」を参照）。

```
/* OE でのキュー・テーブル、トピックを作成します。*/

/* キューを保持するキュー・テーブルを作成します。*/
qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_OBJECT_MESSAGE");
qt_prop.setPrimaryInstance(1);
qt_prop.setSecondaryInstance(2);
q_table = createQueueTable("OE", "OE_orders_sqtab", qt_prop);

/* トピックを保持するキュー・テーブルを作成します。*/
qt1_prop = new AQQueueTableProperty("SYS.AQ$_JMS_OBJECT_MESSAGE");
qt1_prop.setMultiConsumer(TRUE);
qt1_prop.setPrimaryInstance(2);
qt1_prop.setSecondaryInstance(1);
q_table1 = createQueueTable("OE", "OE_orders_mqtab", qt1_prop);

dest_prop = new AQjmsDestinationProperty();
queue = ((AQjmsSession)q_sess).createQueue(q_table, "OE_neworders_que",
                                             dest_prop);

dest_prop1 = new AQjmsDestinationProperty();
topic = ((AQjmsSession)q_sess).createTopic(q_table1, "OE_bookedorders_topic",
                                             dest_prop1);

/* AQ 管理ビューから OE キュー・テーブルのインスタンス親和性を確認します。*/
SELECT queue_table, primary_instance, secondary_instance, owner_instance
FROM user_queue_tables;

/* OE キュー・テーブルのインスタンス親和性を変更します。*/
q_table.alter("OE_orders_sqtab", 2, 1);
q_table1.alter("OE_orders_mqtab1", 1, 2);
```

## 統計ビューのサポート

各インスタンスは、それぞれの AQ 統計情報を SGA に保持しますが、他のインスタンスの統計情報については保持しません。そのため、あるインスタンスが GV\$AQ ビューに対して問い合わせると、その他のすべてのインスタンスからそれぞれのその時点の AQ 統計情報が問合せ元のインスタンスに集まります。

### サンプル・シナリオおよびコード

GV\$ ビューは、待機中、準備完了または期限切れの各メッセージ数を必要なときにいつでも問い合わせることができます。また、メッセージが処理されるまでの平均待機秒数も表示します。注文処理アプリケーションは、これを使用して注文処理プロセスの数を動的にチューニングできます（10-39 ページの「[データベース全体における状態ごとのメッセージ数の選択](#)」を参照）。

```
CONNECT oe/oe
```

```
/* メッセージの数およびメッセージが処理されるまでの平均待機時間をカウントします。 */  
SELECT READY, AVERAGE_WAIT  
FROM gv$aq Stats, user_queues Qs  
WHERE Stats.qid = Qs.qid and Qs.Name = 'OE_neworders_que';
```

## 構造化ペイロード / メッセージの型

JMS メッセージは、次の部分から構成されます。

- ヘッダー - すべてのメッセージは、同じヘッダー・フィールドの集合をサポートします。ヘッダー・フィールドには、クライアントおよびプロバイダの両方がメッセージの識別およびルートに使用する値が含まれます。
- プロパティ - 標準ヘッダー・フィールドの他に、オプションのヘッダー・フィールドをメッセージに追加するための機能があります。
  - 標準プロパティ - JMS は、実質的にはオプションのヘッダー・フィールドであるいくつかの標準プロパティを定義します。
  - プロバイダ固有プロパティ - 各 JMS プロバイダが、特定のプロバイダ固有のプロパティをメッセージに追加している場合があります。
  - アプリケーション固有プロパティ - メッセージにアプリケーション固有ヘッダー・フィールドを追加できます。
- 本体 - メッセージ・ペイロードです。JMS は、様々なメッセージ・ペイロードを定義します。

### メッセージ・ヘッダー

メッセージ・ヘッダーには、次のフィールドが含まれます。

- JMSDestination - このフィールドには、メッセージの送信先が含まれます。AQ では、これは宛先キュー / トピックに対応します。
- JMSDeliveryMode - JMS は、PERSISTENT および NON\_PERSISTENT の 2 つのメッセージ配信モードをサポートします。PERSISTENT では、メッセージは決まった記憶域に記録され、NON\_PERSISTENT では、メッセージのログが取られません。Oracle AQ は、PERSISTENT メッセージ配信をサポートします。
- JMSMessageID - この値は、プロバイダのメッセージを一意に識別します。すべてのメッセージ ID は、ID: で始まる必要があります。
- JMSTimeStamp - メッセージが送信先のプロバイダに渡された時刻が含まれます。これは、AQ のメッセージのエンキュー時刻に対応します。
- JMSCorrelationID - クライアントは、このフィールドを使用して、あるメッセージを別のメッセージにリンクできます。
- JMSReplyTo - このフィールドには、メッセージ送信時にクライアントが指定する宛先が含まれます。これは、そのメッセージへの返信先です。クライアントは、ReplyTo 宛先の指定に AQjmsAgent を使用する必要があります。

- **JMSType** - このフィールドには、送信時にクライアントが指定するメッセージ型識別子が含まれます。移植性の問題から、JMSType を記号値にすることをお薦めします。
- **JMSEExpiration** - メッセージが送信されると、その期限切れ時刻が、送信メソッドで指定された **Time-To-Live** 値と現在の GMT の合計値として計算されます。**Time-To-Live** が 0（ゼロ）に指定されると、期限切れは 0（ゼロ）と設定され、メッセージは期限切れになりません。
- **JMSPriority** - このフィールドには、メッセージの優先順位が含まれます。指定できる優先順位の値は 0、1、2、3... で、優先順位が最も高いのは 0（ゼロ）です。JMS では、JMSDeliveryMode、JMSEExpiration および JMSPriority に対してクライアントが指定した値をオーバーライドできるように、管理者による構成が許可されています。

## メッセージ・プロパティ

プロパティは、メッセージにオプションのヘッダー・フィールドを追加する機能です。プロパティによって、クライアントは、メッセージ・セクタを介して、クライアントのかわりにプロバイダにアプリケーション固有基準を使用してメッセージを選択させることができます。プロパティ名は文字列であり、値は Boolean、Byte、Short、Int、Long、Float、Double および String にすることができます。

JMS 定義済プロパティは、JMSX で始まります。

- **JMSXUserID** - メッセージを送信するユーザーの識別情報です。
- **JMSXAppID** - メッセージを送信するアプリケーションの識別情報です。
- **JMSXDeliveryCount** - メッセージ配信の試行回数です。
- **JMSXGroupid** - このフィールドは、このメッセージが一部をなすメッセージ・グループの識別情報に対するクライアント参照によって設定されます。
- **JMSXGroupSeq** - グループ内のメッセージの順序番号です。
- **JMSXRcvTimeStamp** - メッセージがコンシューマに配信された時刻（デキュー時刻）です。
- **JMSXState** - プロバイダによって設定されたメッセージ状態です。メッセージは、WAITING、READY、EXPIRED または RETAINED の状態に設定できます。

Oracle JMS 固有プロパティは、JMS\_Oracle で始まります。次のプロパティは、Oracle 固有です。

- **JMS\_OracleExcpQ** - メッセージを元の宛先に配信できない場合に、そのメッセージの送信先となるキュー名です。EXCEPTION 型の宛先のみを JMS\_OracleExcpQ プロパティに指定できます。

- JMS\_OracleDelay - メッセージ配信の遅延秒数です。これは、メッセージが配信されるときに順序に影響します。
- JMS\_OracleOriginalMessageId - メッセージが1つの宛先から別の宛先に伝播される場合、このプロパティは元のメッセージのメッセージ ID に設定されます。メッセージが伝播されない場合、このプロパティは JMSMessageId と同じ値になります。

クライアントは、プロパティを定義することによって、メッセージにヘッダー・フィールドを追加できます。これらのプロパティをメッセージ・セクタで使用して、特定のメッセージを選択できます。

JMS プロパティまたはヘッダー・フィールドは、クライアントによって明示的に設定されるか、または JMS プロバイダによって自動的に設定されます（通常、これらは読み込み専用です）。JMS プロパティには、送受信操作に指定されたパラメータを使用して設定されるものもあります。

表 12-1 メッセージ・ヘッダー・フィールド

メッセージ・ヘッダー・フィールド	型	設定方法	使用目的
JMSDestination	Destination	Send メソッド完了後に JMS によって設定されます。	メッセージの送信宛先
JMSDeliveryMode	Int	Send メソッド完了後に JMS によって設定されます。	配信モード -PERSISTENT
JMSExpiration	Long	Send メソッド完了後に JMS によって設定されます。	期限切れ時刻  メッセージ・プロデューサに対して指定、あるいは送信または公開中に明示的に指定できます。
JMSPriority	Int	Send メソッド完了後に JMS によって設定されます。	メッセージの優先順位  メッセージ・プロデューサに対して指定、あるいは送信または公開中に明示的に指定できます。
JMSMessageID	String	Send メソッド完了後に JMS によって設定されます。	プロバイダによって送信された各メッセージを一意に識別する値
JMSTimeStamp	Long	Send メソッド完了後に JMS によって設定されます。	送信先のプロバイダにメッセージが渡された時刻
JMSCorrelationID	String	JMS クライアントによって設定されます。	1つのメッセージを別のメッセージにリンクするのに使用できるフィールド

表 12-1 メッセージ・ヘッダー・フィールド (続き)

メッセージ・ヘッダー・フィールド	型	設定方法	使用目的
JMSReplyTo	Destination	JMS クライアントによって設定されます。	メッセージへの応答送信が送信される、クライアントによって設定された宛先  AQjsAgent 型として指定される必要があります。
JMSType	String	JMS クライアントによって設定されます。	メッセージ型識別子
JMSRedelivered	Boolean	JMS プロバイダによって設定されます。	配信済の可能性があるメッセージが、配信時にクライアントに認識されませんでした。

表 12-2 JMS 定義のメッセージ・プロパティ

JMS 定義のメッセージ・プロパティ	型	設定方法	使用目的
JMSXUserID	String	Send メソッド完了後に JMS によって設定されます。	メッセージを送信するユーザーの識別情報
JMSAppID	String	Send メソッド完了後に JMS によって設定されます。	メッセージを送信するアプリケーションの識別情報
JMSDeliveryCount	Int	Receive メソッド完了後に JMS によって設定されます。	メッセージ配信の試行回数 1 回目は 1、2 回目以降は 2、3、4 と続きます。
JMSXGroupID	String	JMS クライアントによって設定されます。	そのメッセージが一部をなすメッセージ・グループの識別情報
JMSXGroupSeq	Int	JMS クライアントによって設定されます。	グループ内のメッセージの順序番号  1 番目のメッセージは 1、2 回目以降は 2、3、4 と続きます。
JMSXRcvTimeStamp	String	Receive メソッド完了後に JMS によって設定されます。	JMS がメッセージをコンシューマに配信した時刻
JMSXState	Int	JMS プロバイダによって設定されます。	プロバイダによって設定されたメッセージ状態

表 12-3 Oracle 定義のメッセージ・プロパティ

ヘッダー・フィールド/ プロパティ	型	設定方法	使用目的
JMS_OracleExcpQ	String	JMS クライアントによって設定されます。	例外キュー名を指定します。
JMS_OracleDelay	Int	JMS クライアントによって設定されます。	コンシューマがメッセージを使用できるようになるまでの時間(秒数)を指定します。
JMS_OracleOriginalMessageID	String	JMS プロバイダによって設定されます。	メッセージが1つの宛先から別の宛先に伝播されるとき、ソースのメッセージのメッセージIDを指定します。

メッセージ本体

JMS では、次の 5 つの形式のメッセージ本体が提供されます。

- **StreamMessage** - 本体に Java 基本データ型の値がストリームとして含まれるメッセージです。このメッセージ本体には順次書込み / 読み込みが可能です。
- **ByteMessage** - 本体にバイト列のストリームが含まれるメッセージです。これは、本体を直接コード化して既存のメッセージ形式に一致させるためのメッセージ型です。
- **MapMessage** - 本体に名前と値の組の集合を含むメッセージです。名前は文字列であり、値は Java 基本データ型です。エントリには、列挙オブジェクトを使用して順次アクセスするか、名前でランダムにアクセスできます。
- **TextMessage** - 本体に java.lang.String が含まれるメッセージです。
- **ObjectMessage** - シリアライズ可能な Java オブジェクトが含まれるメッセージです。
- **AdtMessage** - 本体に Oracle のユーザー定義型オブジェクトが含まれるメッセージです (Oracle JMS では、AdtMessage 型が追加されています)。

Stream メッセージ

StreamMessage は、Java 基本データ型の値をストリームとして送信する場合に使用します。このメッセージ本体には順次書込み / 読み込みが可能です。StreamMessage は、Message から継承され、Stream メッセージ本体を追加します。StreamMessage のメソッドは、主に java.io.DataInputStream および java.io.DataOutputStream のメソッドに基づいています。



基本データ型の値は、それぞれの型ごとのメソッドを使用して、明示的に読み込みまたは書き込みできます。また、抽象的なオブジェクトとして読み込みまたは書き込みされる場合もあります。Stream メッセージを使用するには、ユーザーは、パイロード型 `SYS.AQ$_JMS_STREAM_MESSAGE` を持つキュー・テーブルを作成する必要があります。

Stream メッセージは、次の変換表をサポートします。行の型として書き込まれる値は、列の型として読み込むことができます。

表 12-4 StreamMessage 変換

	Boolean	Byte	Short	Char	Int	Long	Float	Double	String	Byte[]
Boolean	X								X	
Byte		X	X		X	X			X	
Short			X		X	X			X	
Char				X					X	
Int					X	X			X	
Long						X			X	
Float							X	X	X	
Double								X	X	
String	X	X	X	X	X	X	X	X	X	
Byte[]										X

Byte メッセージ

ByteMessage は、未解析バイトの 1 つのストリームを含むメッセージを送信する場合に使用します。ByteMessage は、Message から継承され、Byte メッセージ本体を追加します。メッセージの受信者が、そのバイト列を解析します。このメソッドは、主に `java.io.DataInputStream` および `java.io.DataOutputStream` のメソッドに基づいています。

これは、クライアントが既存のメッセージ形式をコード化するためのメッセージ型です。できるだけ、他の自己定義メッセージ型をかわりに使用する必要があります。

Java の基本データ型の値は、それぞれの型のメソッドを使用して、明示的に書き込みできます。抽象的なオブジェクトとして書き込まれる場合もあります。Byte メッセージを使用するには、ユーザーは、パイロード型 `SYS.AQ$_JMS_BYTES_MESSAGE` を持つキュー・テーブルを作成する必要があります。

Map メッセージ

MapMessage は、名前が文字列で値が Java 基本データ型である名前と値の組の集合を送信する場合に使用します。エントリには、名前で順次またはランダムにアクセスできます。エントリの順序は未定義です。MapMessage は、Message から継承され、Map メッセージ本体を追加します。基本データ型は、それぞれの型のメソッドを使用して、明示的な読み込みまたは書き込みができます。また、抽象的なオブジェクトとして読み込みまたは書き込まれる場合もあります。

Map メッセージを使用するには、ユーザーは、ペイロード型 SYS.AQ\$\_JMS\_MAP\_MESSAGE を持つキュー・テーブルを作成する必要があります。Map メッセージは、次の変換表をサポートします。行の型として書き込まれる値は、列の型として読み込むことができます。

表 12-5 MapMessage 変換

	Boolean	Byte	Short	Char	Int	Long	Float	Double	String	Byte[]
Boolean	X								X	
Byte		X	X		X	X			X	
Short			X		X	X			X	
Char				X					X	
Int					X	X			X	
Long						X			X	
Float							X	X	X	
Double								X	X	
String	X	X	X	X	X	X	X	X	X	
Byte[]										X

## Text メッセージ

TextMessage は、java.lang.StringBuffer を含むメッセージを送信する場合に使用します。TextMessage は、Message から継承され、Text メッセージ本体を追加します。テキスト情報は、getText() および setText(...) メソッドを使用して読み込みまたは書き込みできます。Text メッセージを使用するには、ユーザーは、ペイロード型 SYS.AQ\$\_JMS\_TEXT\_MESSAGE を持つキュー・テーブルを作成する必要があります。

## Object メッセージ

ObjectMessage は、シリアライズ可能な Java オブジェクトを含むメッセージを送信する場合に使用します。ObjectMessage は、Message から継承され、単一の Java オブジェクトへの参照を含む本体を追加します。シリアライズ可能な Java オブジェクトのみを使用できます。Java オブジェクトのコレクションの送信が必要な場合は、JDK 1.2 で提供されたコレクション・クラスの 1 つを使用できます。このオブジェクトは、getObject() および setObject(...) メソッドを使用して読み込みまたは書き込みできます。Object メッセージを使用するには、ユーザーは、ペイロード型 SYS.AQ\$\_JMS\_OBJECT\_MESSAGE を持つキュー・テーブルを作成する必要があります。

## サンプル・コード

```
public void enqueue_new_orders(QueueSession jms_session, BolOrder new_order)
{
    QueueSender    sender;
    Queue           queue;
    ObjectMessage   obj_message;

    try
    {
        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQJmsSession) jms_session).getQueue("OE", "OE_neworders_que");
        sender = jms_session.createSender(queue);
        obj_message = jms_session.createObjectMessage();
        obj_message.setJMSCorrelationID("RUSH");
        obj_message.setObject(new_order);
        jms_session.commit();
    }
    catch (JMSException ex)
    {
        System.out.println("Exception: " + ex);
    }
}
```

## ADT メッセージ

AdtMessage は、Oracle のユーザー定義型に対応する Java オブジェクトを含むメッセージを送信する場合に使用します。これらのオブジェクトは、Message から継承され、CustomDatum インタフェースを実装する Java オブジェクトを含む本体を追加します。CustomDatum インタフェースの詳細は、『Oracle8i JDBC 開発者ガイドおよびリファレンス』を参照してください。

ADT メッセージを使用するには、ユーザーは、Oracle のオブジェクト型としてのペイロード型を持つキュー・テーブルを作成する必要があります。ADT メッセージのペイロードは、getAdtPayload および setAdtPayload メソッドを使用して、読み込みおよび書き込みできます。

## 異なるメッセージ型でのメッセージ・プロパティの使用

- クライアントが setProperty コールを使用して設定できる JMS プロパティは、次のとおりです。
  - StreamMessage、ByteMessage、ObjectMessage、TextMessage、MapMessage の場合  
JMSXAppID  
JMSXGroupID  
JMSXGroupSeq  
JMS\_OracleExcpQ  
JMS\_OracleDelay
  - AdtMessage の場合  
JMS\_OracleExcpQ  
JMS\_OracleDelay
- クライアントが getProperty コールを使用して取得できる JMS プロパティは、次のとおりです。
  - StreamMessage、ByteMessage、ObjectMessage、TextMessage、MapMessage の場合  
JMSXUserID  
JMSXAppID  
JMSXDeliveryCount  
JMSXGroupID  
JMSXGroupSeq

JMSXRecvTimeStamp  
 JMSXState  
 JMS\_OracleExcpQ  
 JMS\_OracleDelay  
 JMS\_OracleOriginalMessageID

– AdtMessage の場合

JMSXDeliveryCount  
 JMSXRecvTimeStamp  
 JMSXState  
 JMS\_OracleExcpQ  
 JMS\_OracleDelay

■ メッセージ・セレクタに含めることができる JMS プロパティ / ヘッダー・フィールドは、次のとおりです。

– (JMS 型または ADT 型ペイロードを含むキューの) キュー受信者およびトピック受信者の場合は、次のとおりです。

JMSMessageID  
 JMSCorrelationID

– (JMS 型ペイロードを含むキューの) トピック・サブスクライバの場合は、次を含む文字列があるすべての SQL92 の WHERE 句。

JMSPriority (Int)  
 JMSCorrelationID (String)  
 JMSTimestamp (Date)  
 JMSType (String)  
 JMSXUserID (String)  
 JMSXAppID (String)  
 JMSXGroupID (String)  
 JMSXGroupSeq (Int)  
 すべてのユーザー定義のプロパティ

- (ADT 型ペイロードを含むキューの) トピック・サブスクライバの場合は、次を含む文字列があるすべての SQL92 の WHERE 句に対して AQ 規則の構文を使用します。
  - \* corrid
  - \* priority
  - \* tab.user\_data.<adt\_field\_name>

## JMS サンプルで使ったペイロード

```
/*
 * BooksOrder - BooksOnline の例でのペイロード
 *
 */

import java.lang.*;
import java.io.*;
import java.util.*;

public class BolOrder implements Serializable
{
    int            orderno;
    String         status;
    String         type;
    String         region;
    BolCustomer    customer;
    String         paymentmethod;
    BolOrderItem[] itemlist;
    String         ccnumber;
    Date           orderdate;

    public BolOrder(int orderno, BolCustomer customer)
    {
        this.customer    = customer;
        this.orderno     = orderno;
    }

    public int getOrderNo()
    {
        return orderno;
    }

    public String getStatus()
    {
        return status;
    }

    public void setStatus(String new_status)
    {
        status = new_status;
    }
}
```

```
public String getRegion()
{
    return region;
}

public void setRegion(String region)
{
    this.region = region;
}

public BolCustomer getCustomer()
{
    return customer;
}

public String getPaymentmethod()
{
    return paymentmethod;
}

public void setPaymentmethod(String paymentmethod)
{
    this.paymentmethod = paymentmethod;
}

public BolOrderItem[] getItemList()
{
    return itemlist;
}

public void setItemList(BolOrderItem[] itemlist)
{
    this.itemlist = itemlist;
}

public String getCCnumber()
{
    return ccnumber;
}
```



```
    }

    public void setCNumber(String ccnumber)
    {
        this.ccnumber = ccnumber;
    }

    public Date getOrderDate()
    {
        return orderdate;
    }

    public void setOrderDate(Date orderdate)
    {
        this.orderdate = orderdate;
    }
}

/*
 * BolOrderItem - BooksOnline の例での注文アイテムの種類
 */

import java.lang.*;
import java.io.*;
import java.util.*;

public class BolOrderItem implements Serializable
{
    BolBook    item;
    int        quantity;

    public BolOrderItem(BolBook book, int quantity)
    {
        item    = book;
        this.quantity = quantity;
    }
}
```

```
        public BolBook getItem()
        {
            return item;
        }

        public int getQuantity()
        {
            return quantity;
        }
    }

    /*
     * BolBook - book type for BooksOnline の例での書籍の種類
     *
     */

    import java.lang.*;
    import java.io.*;
    import java.util.*;

    public class BolBook implements Serializable
    {

        String    title;
        String    authors;
        String    isbn;
        float     price;

        public BolBook(String title)
        {
            this.title    = title;
        }

        public BolBook(String title, String authors, String isbn, float price)
        {
            this.title    = title;
            this.authors  = authors;
            this.isbn     = isbn;
            this.price    = price;
        }
    }
```

```
public String getISBN()
{
    return isbn;
}

public String getTitle()
{
    return title;
}

public String getAuthors()
{
    return authors;
}

public float getPrice()
{
    return price;
}
}

/*
 * BolCustomer - BooksOnline の例での顧客の種類
 *
 */

import java.lang.*;
import java.io.*;
import java.util.*;

public class BolCustomer implements Serializable
{
    int        custno;
    String     custid;
    String     name;
    String     street;
    String     city;
    String     state;
    int        zip;
    String     country;
```

```
public BolCustomer(int custno, String name)
{
    this.custno = custno;
    this.name = name;
}

public BolCustomer(int custno, String custid, String name, String street,
    String city, String state, int zip, String country)
{
    this.custno = custno;
    this.custid = custid;
    this.name = name;
    this.street = street;
    this.city = city;
    this.state = state;
    this.zip = zip;
    this.country = country;
}

public int getCustomerNo()
{
    return custno;
}

public String getCustomerId()
{
    return custid;
}

public String getName()
{
    return name;
}

public String getStreet()
{
    return street;
}

public String getCity()
{

```

```
        return city;
    }

    public String getState()
    {
        return state;
    }

    public int getZipcode()
    {
        return zip;
    }

    public String getCountry()
    {
        return country;
    }
}
```

## Point-to-Point モデル機能

- キュー
- キュー送信者
- キュー受信者
- キュー・ブラウザ

## キュー

Point-to-Point モデルでは、クライアントは、1つのポイントから別のポイントへ、キューを使用してメッセージを交換します。これらのキューは、メッセージ・プロデューサおよびコンシューマによるメッセージの送受信に使用されます。

管理者は、AQjmsSession の createQueue メソッドを使用して、単一コンシューマ・キューを作成します。クライアントは、AQjmsSession の getQueue メソッドを使用して、事前に作成されたキューに対するハンドルを取得する場合があります。

これらのキューは、単一のコンシューマのみがメッセージを処理できるため、**単一コンシューマ・キュー**といわれます。つまり、メッセージは一度に1つのコンシューマでしか処理できません。同じキューから、複数のプロセスまたはオペレーティング・システム・スレッドが同時にデキューしようとする場合を考えてみます。ロックされたメッセージをデキューできるのは、ロックを作成したプロセスのみであるとする、各プロセスは、キューの先頭にあるロックされていない最初のメッセージをデキューします。

キューを使用する前に、AQjmsDestination で start コールを使用して、キューをエンキュー / デキューに対して使用可能にする必要があります。

処理が済むと、キューの保存期間が0（ゼロ）の場合はそのメッセージは削除され、そうでない場合は指定された期間保存されます。メッセージが保存されている間は、次のいずれかが可能です。

- キュー・テーブル・ビューに対して SQL を使用して問合せができます。
- キュー・ブラウザを使用し、処理済のメッセージのメッセージ ID を指定して、デキューできます。

## キュー送信者

クライアントは、キュー送信者を使用して、キューにメッセージを送信します。キュー送信者は、キューをセッションの `createSender` メソッドに渡すことによって作成されます。また、クライアントには、キューを指定せずにキュー送信者を作成するオプションがあります。この場合、キューは、送信操作のたびに指定される必要があります。

クライアントは、キュー送信者によって送信されたすべてのメッセージのデフォルト配信モード、優先順位および `Time-To-Live` を指定できます。また、クライアントは、これらのオプションをメッセージ単位で定義できます。

### サンプル・コード

BooksOnLine アプリケーションでは、新規の注文が `new_orders_queue` に送信されます。JMS 接続およびセッションを作成した後、次のように送信者を作成します。

```
public void enqueue_new_orders(QueueSession jms_session, BolOrder new_order)
{
    QueueSender    sender;
    Queue          queue;
    ObjectMessage  obj_message;

    try
    {
        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQJmsSession) jms_session).getQueue("OE", "OE_neworders_que");
        sender = jms_session.createSender(queue);
        obj_message = jms_session.createObjectMessage();
        obj_message.setJMSCorrelationID("RUSH");
        obj_message.setObject(new_order);
        sender.send(obj_message);
        jms_session.commit();
    }
    catch (JMSException ex)
    {
        System.out.println("Exception: " + ex);
    }
}
```

## キュー受信者

クライアントは、キュー受信者を使用して、キューからメッセージを受信します。キュー受信者は、セッションの `createQueueReceiver` メソッドを使用して作成されます。キュー受信者は、メッセージ・セクタで作成できます。これによって、クライアントは、コン



シューマに配信されるメッセージをセレクトと一致するメッセージに制限できるようになります。

キュー受信者のためのセレクトは、次のいずれかです。

- JMSMessageID = 'ID:23452345'  
これは、指定されたメッセージ ID（すべてのメッセージで接頭語として「ID:」を持ちます）を持つメッセージを取り出します。
- JMSCorrelationID = 'RUSH'
- JMSCorrelationID LIKE 'RE%'  
これは、特定の correlationID を持つメッセージを取り出します。

## サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、新規の注文が `new_orders_queue` から取り出されます。これらの注文は、`OE.OE_bookedorders_topic` に対して公開されます。JMS 接続およびセッションを作成した後、次のようにメッセージの受信者を作成します。

```
public void get_new_orders(QueueSession jms_session)
{
    QueueReceiver    receiver;
    Queue             queue;
    ObjectMessage     obj_message;
    BolOrder          new_order;
    BolCustomer       customer;
    String            state;
    String            cust_name;

    try
    {

        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQJMSession) jms_session).getQueue("OE", "OE_neworders_que");

        receiver = jms_session.createReceiver(queue);

        for(;;)
        {
            /* キューにメッセージが届くのを待ちます。*/
            obj_message = (ObjectMessage)receiver.receive(10);

            new_order = (BolOrder)obj_message.getObject();

            customer = new_order.getCustomer();
            state     = customer.getState();
        }
    }
}
```

```
        obj_message.clearBody();

        /* 顧客領域を判断し、出荷領域を割り当てます。*/
        if((state.equals("CA")) || (state.equals("TX")) ||
            (state.equals("WA")) || (state.equals("NV")))
            obj_message.setStringProperty("Region", "WESTERN");
        else
            obj_message.setStringProperty("Region", "EASTERN");

        cust_name = new_order.getCustomer().getName();

        obj_message.setStringProperty("Customer", cust_name);

        if(obj_message.getJMSCorrelationID().equals("RUSH"))
            book_rush_order(obj_message);
        else
            book_new_order(obj_message);

        jms_session.commit();
    }
}
catch (JMSException ex)
{
    System.out.println("Exception: " + ex);
}
}
```

## キュー・ブラウザ

クライアントが、メッセージを削除せずにキュー上で参照するには、キュー・ブラウザを使用します。このブラウザ・メソッドは、キューのメッセージをスキャンするために使用される `java.util.Enumeration` を戻します。`nextElement` に対する最初のコールが、キューのスナップショットを取得します。キュー・ブラウザも、メッセージをスキャンしているときに、メッセージをオプションでロックする場合があります。これは、メッセージに対する `SELECT ... for UPDATE` コマンドの場合と似ています。これによって、他のコンシューマがスキャン中のメッセージを削除することはなくなります。

キュー・ブラウザも、メッセージ・セクタで作成できます。これによって、クライアントは、コンシューマに配信されるメッセージをセクタと一致するメッセージに制限できるようになります。

キュー・ブラウザのためのセクタは、次のすべての形式で許可されます。

- `JMSMessageID = 'ID:23452345'`  
指定されたメッセージ ID（すべてのメッセージで接頭語として「ID:」を持ちます）を持つメッセージを取り出します。
- `JMSCorrelationID = 'RUSH'`
- `JMSCorrelationID LIKE 'RE%'`  
特定の相関 ID を持つメッセージを取り出します。

## サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、新規の注文が `new_orders_queue` に入れられます。クライアントは、選択されたメッセージをブラウザできます。

```
public void browse_rush_orders(QueueSession jms_session)
{
    QueueBrowser    browser;
    Queue            queue;
    ObjectMessage    obj_message;
    BolOrder         new_order;
    Enumeration      messages;
    String            customer_name;

    try
    {
        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQJMSession) jms_session).getQueue("OE", "OE_neworders_que");

        /* 至急 (RUSH) 注文 を調べるためのブラウザを作成します。*/
        browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'");
```

```
for (messages = browser.getEnumeration() ; messages.hasMoreElements() ;)
{
    obj_message = (ObjectMessage)messages.nextElement();

    new_order = (BolOrder)obj_message.getObject();

    customer_name = new_order.getCustomer().getName();
    System.out.println("Customer " + customer_name +
        " has placed a RUSH order");
}

browser.close();
}
catch (Exception ex)
{
    System.out.println("Exception " + ex);
}
}
```

## パブリッシュ / サブスクライブ・モデル機能

- [トピック](#)
- [永続サブスクライバ](#)
- [トピック・パブリッシャ](#)
- [受信者リスト](#)
- [トピック受信者](#)

## トピック

JMS には様々な機能があり、ユーザーはパブリッシュ / サブスクライブ・モデルに基づいたアプリケーションを開発できます。このアプリケーション・モデルの目的は、パブリッシャ（公開者）の機能を持つアプリケーションとサブスクライバ（予約者）の役割を果たすアプリケーションとの間の柔軟で動的な通信を可能にすることです。具体的な設計のポイント は、そのように異なる役割を果たすアプリケーションを通信上では分離し、メッセージおよびメッセージ内容に基づいた相互作用をさせるということです。

分散メッセージでは、パブリッシャ・アプリケーションが明示的にメッセージ受信者进行处理または管理する必要はありません。このため、パブリッシャ・アプリケーションの論理を変更しなくても、受信メッセージに新しいサブスクライバ・アプリケーションを動的に追加できます。サブスクライバ・アプリケーションは、メッセージを送信しているパブリッシャ・アプリケーションに関係なく、メッセージの内容に基づいてメッセージを受信します。このため、サブスクライバ・アプリケーションの論理を変更しなくても、サブスクライバ・アプリケーションを動的に追加できます。サブスクライバ・アプリケーションは、メッセージ・プロパティまたはトピックのメッセージ内容（あるいはその両方）に対してルールベースのサブスクリプション（予約申込み）を定義することで、関心のあることを指定します。システムは、ルールベースのサブスクリプションを使用して、公開されたメッセージの受信者を計算し、自動的にルーティングします。

パブリッシュ / サブスクライブ・モデルでは、メッセージはトピックに対して公開され、トピックから受信されます。トピックは、AQJMSession の CreateTopic メソッドを使用して作成されます。クライアントは、AQJMSession の getTopic メソッドを使用して、事前に作成されたトピックに対するハンドルを取得する場合があります。

JMS でパブリッシュ / サブスクライブ・モデルの通信を使用するには、次の各ステップを実行します。

- AQJMSDestination で start コールを使用して、トピックに対するエンキュー / デキューを使用可能にします。
- メッセージを保持するために 1 つ以上のトピックを設定します。これらのトピックは、関心がある領域またはサブジェクトを表す必要があります。たとえば、請求済注文情報を表すようにトピックを設定できます。
- **永続サブスクライバ**の集合を作成します。各サブスクライバは、受信を希望するメッセージ仕様（選択）を表すセレクタを指定する場合があります。NULL セレクタは、そのトピックに対して公開されたすべてのメッセージの受信をサブスクライバが希望していることを示します。
- サブスクライバは、ローカルまたはリモートのいずれかです。ローカル・サブスクライバは、メッセージが公開されるトピックと同じトピックに対して定義された永続サブスクライバです。リモート・サブスクライバは、他のトピック、または特定キューのサブスクライバとして定義された他のトピックに対する受信者です。リモート・サブスクライバを使用するには、ローカルおよびリモート・トピックの間に**伝播**を設定する必要があります。伝播の詳細は、[第 9 章「管理インタフェース」](#)を参照してください。

- セッションの `createPublisher` メソッドを使用してトピック・パブリッシャを作成します。メッセージは `publish` コールを使用して公開されます。メッセージは、トピックのすべてのサブスクライバ、またはトピックに対する指定された受信者のサブセットに対して公開されます。
- サブスクライバは、受信メソッドを使用して、トピックに関するメッセージを受信します。
- サブスクライバは、**メッセージ・リスナー**を使用して、非同期にメッセージを受信する場合もあります。**リモート・サブスクライバ**および**伝播**の概念は、JMS に対する Oracle 拡張機能です。

## サンプル・シナリオ

BooksOnLine アプリケーションでは、すべての入力済注文情報が `OE_bookedorders_topic` に対して公開されます。東部地域の顧客の注文情報は `ES.ES_bookedorders_topic` にルーティングされ、西部地域の顧客の注文情報は `WS.WS_bookedorders_topic` にルーティングされます。また、重要な顧客に対するメッセージを追跡するために `OE_bookedorders_topic` にサブスクライブするアプリケーションもあります。以降のページにあるコード例を参照してください。

## 永続サブスクライバ

永続サブスクライバは、次のいずれかの方法で作成されます。

- クライアントが、セッションの `createDurableSubscriber` メソッドを使用して、永続サブスクライバを作成します。
- メッセージ・セレクトアで永続サブスクライバを作成します。これによって、クライアントは、サブスクライバに配信されるメッセージをセレクトアと一致するメッセージに制限できます。

`TextMessage`、`StreamMessage`、`ByteMessage`、`ObjectMessage`、`MapMessage` 型のペイロードを含むトピックのセレクトアには、次の内の 1 つ以上を組み合わせた式を含めることができます。

- JMS メッセージ・ヘッダー・フィールドまたはプロパティ

```
JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
```

- ユーザー定義のメッセージ・プロパティ

```
color IN ('RED', 'BLUE', 'GREEN') AND price < 30000
```

`AdtMessage` を含むトピックの場合、セレクトアはメッセージ・ペイロード内容、優先順位または相関 ID に対する SQL 式である必要があります。

- 優先順位または相関 ID に対するセレクトアは、次のように指定します。

```
priority < 3 AND corr_id = 'Fiction'
```

- メッセージ・ペイロードに対するセレクトアは、次のように指定します。

```
tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
```

セレクトア構文の詳細は、『Oracle8i Java パッケージ・プロシージャ リファレンス』の「`createDurableSubscriber` ユースケース」を参照してください。

リモート・サブスクライバは、`createRemoteSubscriber` コールを使用して定義されます。リモート・サブスクライバは、リモート・トピックの特定のコンシューマまたはすべてのサブスクライバです。

リモート・サブスクライバは、`AQjmsAgent` 構造を使用して定義されます。`AQjmsAgent` は、名前およびアドレスで構成されます。名前は、リモート・トピックで `consumer_name` を参照します。アドレスは、次のようにしてリモート・トピックを参照します。

```
<schema>.<topic_name>[@dblink]
```



- リモート・トピックで特定のコンシューマに対してメッセージを公開するには、リモート・トピックでの受信者のサブスクライバ名が、AQjmsAgent の名前フィールドに指定される必要があります。リモート・トピックは、AQjmsAgent のアドレス・フィールドに指定される必要があります。
- リモート・トピックのすべてのサブスクライバに対してメッセージを公開するには、AQjmsAgent の名前フィールドを NULL に設定する必要があります。リモート・トピックは、AQjmsAgent のアドレス・フィールドに指定される必要があります。

サンプルとして使用している BooksOnLine アプリケーションには、1つのローカル・サブスクライバ SUBS1 および次の2つのリモート・サブスクライバがあります。

- リモート・トピック WS.WS\_bookedorders\_topic での West\_Shipping
- ES.ES\_booked\_orders\_topic での East\_Shipping

## サンプル・コード

```
public void create_booked_orders_subscribers(TopicSession jms_session)
{
    Topic          topic;
    TopicSubscriber tsubs;
    AQjmsAgent      agt_east;
    AQjmsAgent      agt_west;

    try
    {

        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession) jms_session).getTopic("OE",
            "OR_bookedorders_topic");

        /* ローカル・サブスクライバを作成します - 顧客へのメッセージを追跡します。*/
        tsubs = jms_session.createDurableSubscriber(topic, "SUBS1",
            "JMSPriority < 3 AND Customer = 'MARTIN'",
            false);

        /* 西部地区および東部地区のリモート・サブスクライバを作成します。*/
        agt_west = new AQjmsAgent("West_Shipping", "WS.WS_bookedorders_topic");

        ((AQjmsSession) jms_session).createRemoteSubscriber(topic, agt_west,
            "Region = 'WESTERN'");

        agt_east = new AQjmsAgent("East_Shipping", "ES.ES_bookedorders_topic");
    }
}
```

```
((AQjmsSession)jms_session).createRemoteSubscriber(topic, agt_east,
    "Region = 'EASTERN'");

/* bookedorders_topic、WS_bookedorders_topic、ES.ES_bookedorders_topic 間の伝播
をスケジュールします。*/
((AQjmsDestination)topic).schedulePropagation(jms_session,
    "WS.WS_bookedorders_topic",
    null, null, null, null);

((AQjmsDestination)topic).schedulePropagation(jms_session,
    "ES.ES_bookedorders_topic",
    null, null, null, null);
}
catch (Exception ex)
{
    System.out.println("Exception " + ex);
}
}
```

## トピック・パブリッシャ

メッセージは、トピック・パブリッシャを使用して公開されます。

トピック・パブリッシャは、トピックをセッションの `createPublisher` メソッドに渡すことによって作成されます。クライアントには、トピックを指定せずにトピック・パブリッシャを作成するオプションもあります。この場合、トピックは、公開操作のたびに指定される必要があります。クライアントは、トピック・パブリッシャによって送信されたすべてのメッセージのデフォルト配信モード、優先順位および Time-To-Live を指定できます。また、クライアントは、これらのオプションをメッセージ単位で指定できます。

### サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、入力済注文情報が `OE.OE_bookedorders_topic` に対して公開されます。

```
public void book_new_order(TopicSession jms_session, ObjectMessage obj_message)
{
    TopicPublisher publisher;
    Topic topic;

    try
    {
        /* booked_orders トピックへのハンドルを取得します。*/
        topic = ((AQJmsSession) jms_session).getTopic("OE",
            "OE_bookedorders_topic");

        publisher = jms_session.createPublisher(topic);

        publisher.publish(topic, obj_message);

        jms_session.commit();
    }
    catch (JMSException ex)
    {
        System.out.println("Exception: " + ex);
    }
}
```

BooksOnLine アプリケーションでは、各出荷地域は、対応する入力済注文情報トピック (`WS_bookedorder_topic` または `ES_bookedorder_topic`) からメッセージを受信します。ローカル・サブスクライバ `SUBS1` は、`OE_booked_orders_topic` からメッセージを受信します。

```
public void get_martins_orders(TopicSession jms_session)
{
    Topic            topic;
    TopicSubscriber  tsubs;
    ObjectMessage    obj_message;
    BolCustomer      customer;
    BolOrder         new_order;
    String           state;
    int              i = 0;

    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQJmsSession)jms_session).getTopic("OE",
            "OE_bookedorders_topic");

        /* ローカル・サブスクライバを作成します。 - 顧客へのメッセージを追跡します。*/
        tsubs = jms_session.createDurableSubscriber(topic, "SUBS1",
            "JMSPriority < 3 AND Customer = 'MARTIN'",
            false);

        /* 10 個のメッセージを処理します。*/
        for(i=0; i<10; i++)
        {
            /* トピックにメッセージが届くのを待ちます。*/
            obj_message = (ObjectMessage)tsubs.receive(10);

            new_order = (BolOrder)obj_message.getObject();

            customer = new_order.getCustomer();
            state     = customer.getState();

            System.out.println("Order: " + i + " for customer " +
                customer.getName());
            jms_session.commit();
        }
    }
    catch (Exception ex)
    {
        System.out.println("Exception " + ex);
    }
}
```

## 受信者リスト

JMS パブリッシュ / サブスクライブ・モデルでは、クライアントは、トピックのすべてのサブスクライバにメッセージを送信するのではなく、明示的な受信者リストを指定できます。これらの受信者は、トピックの既存のサブスクライバである場合もあれば、そうでない場合もあります。受信者リストは、このメッセージのトピックのサブスクリプション・リストをオーバーライドします。受信者リストの概念は、JMS に対する Oracle 拡張です。

### サンプル・シナリオおよびコード

優先順位が高いメッセージを、OE\_bookedorders\_topic のすべてのサブスクライバに対して公開するのではなく、東部地域の SUBS1 および Fedex\_Shipping のみに送信とします。

```
public void book_rush_order(TopicSession jms_session,
                           ObjectMessage obj_message)
{
    TopicPublisher publisher;
    Topic topic;
    AQjmsAgent[] recp_list = new AQjmsAgent[2];

    try
    {
        /* booked_orders トピックへのハンドルを取得します。*/
        topic = ((AQjmsSession) jms_session).getTopic("OE",
                                                       "OE_bookedorders_topic");

        publisher = jms_session.createPublisher(null);

        recp_list[0] = new AQjmsAgent("SUBS1", null);
        recp_list[1] = new AQjmsAgent("Fedex_Shipping",
                                       "ES.ES_bookedorders_topic");

        publisher.setPriority(1);
        ((AQjmsTopicPublisher)publisher).publish(topic, obj_message, recp_list);

        jms_session.commit();
    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
    }
}
```

## トピック受信者

受信者名が受信者リストに明示的に指定されていても、その受信者がキューのサブスクライバではない場合は、その受信者に送信されるメッセージは、トピック受信者を作成することによって受信できます。トピック受信者はJMSに対するOracle拡張です。

### サンプル・シナリオおよびコード

```
public void ship_rush_orders(TopicSession jms_session)
{
    Topic            topic;
    TopicReceiver    trec;
    ObjectMessage     obj_message;
    BolCustomer      customer;
    BolOrder          new_order;
    String            state;
    int               i = 0;

    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQJmsSession)jms_session).getTopic("ES",
            "ES_bookedorders_topic");

        /* ローカル・サブスクライバを作成します - 顧客へのメッセージを追跡します。*/
        trec = ((AQJmsSession)jms_session).createTopicReceiver(topic,
            "Fedex_Shipping",
            null);

        /* 10 個のメッセージを処理します。*/
        for(i = 0; i < 10; i++)
        {
            /* トピックにメッセージが届くのを待ちます。*/
            obj_message = (ObjectMessage)trec.receive(10);

            new_order = (BolOrder)obj_message.getObject();

            customer = new_order.getCustomer();
            state     = customer.getState();

            System.out.println("Rush Order for customer " +
```

```

        customer.getName();
        jms_session.commit();
    }
}
catch (Exception ex)
{
    System.out.println("Exception ex: " + ex);
}
}

```

リモート・サブスクライバの場合、リモート・トピックでのサブスクライバ名が createRemoteSubscriber コールに明示的に指定されているときは、トピック受信者を使用してメッセージを受信できます。

```

public void get_westernregion_booked_orders(TopicSession jms_session)
{
    Topic            topic;
    TopicReceiver    trec;
    ObjectMessage     obj_message;
    BolCustomer       customer;
    BolOrder          new_order;
    String            state;
    int               i = 0;

    try
    {
        /* WS_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession)jms_session).getTopic("WS",
            "WS_bookedorders_topic");

        /* ローカル・サブスクライバを作成します - 顧客へのメッセージを追跡します。*/
        trec = ((AQjmsSession)jms_session).createTopicReceiver(topic,
            "West_Shipping",
            null);

        /* 10 個のメッセージを処理します。*/
        for(i = 0; i < 10; i++)
        {
            /* トピックにメッセージが届くのを待ちます。*/
            obj_message = (ObjectMessage)trec.receive(10);

            new_order = (BolOrder)obj_message.getObject();

```

```
customer = new_order.getCustomer();
state    = customer.getState();

System.out.println("Received Order for customer " +
    customer.getName());
jms_session.commit();
    }
}
catch (Exception ex)
{
    System.out.println("Exception ex: " + ex);
}
}
```

サブスクライバ名が `createRemoteSubscriber` コールに指定されていない場合、クライアントがメッセージを受信するには、リモート・サイトで永続サブスクライバを使用する必要があります。



## メッセージ・プロデューサ機能

- [メッセージの優先順位および順序付け](#)
- [時刻指定 : 遅延](#)
- [時刻指定 : 期限切れ](#)
- [メッセージのグループ化](#)

## メッセージの優先順位および順序付け

メッセージの順序付けとは、キューまたはトピックからメッセージが受信される順序を示します。優先順位の指定は、キューまたはトピックに対してキュー・テーブルが作成されたときに指定されます (9-5 ページの「[キュー・テーブルの作成](#)」を参照)。現在、AQ は、次の 2 つのメッセージ属性に対する順序付けをサポートしています。

- 優先順位
- エンキュー時刻

これらが組み合されると、次の 4 つの順序付け方法が可能になります。

**メッセージの FIFO による順序付け** エンキュー時刻が順序付け基準として選択されると、メッセージはエンキュー時刻の順序で受信されます。エンキュー時刻は、メッセージの公開 / 送信時に AQ によってメッセージに割り当てられます。これはデフォルトの順序付けです。

**メッセージの優先順位による順序付け** 優先順位による順序付けが選択されると、各メッセージに優先順位が割り当てられます。優先順位は、公開 / 送信時にメッセージ・プロデューサによってメッセージ・プロパティとして指定できます。メッセージは、割り当てられた優先順位の順序で受信されます。

**先入れ先出し (FIFO) 優先順位による順序付け** FIFO 優先順位のトピック / キューも、優先順位およびエンキュー時刻の両方でメッセージのソート順を指定することで作成できます。FIFO 優先順位のキューは、優先順位による順序付けに似ていますが、同じ優先順位のメッセージが 2 つあるときにエンキュー時刻の順序でデキューされるところが異なります。

**エンキュー時刻に続く優先順位による順序付け** 同じエンキュー時刻のメッセージは、優先順位の順序で受信されます。2 つのメッセージの順序付け基準が同じ場合、受信される順序は予想できません。ただし、AQ によって、同じセッション内で同じ順序付け基準で送信 / 公開されたメッセージは、確実に送信された順序で受信されます。

## サンプル・シナリオおよびコード

BooksOnLine アプリケーションを使用すると、顧客は次の方法を要求できます。

- FedEx による出荷 (優先順位 1)
- 優先扱い航空便による出荷 (優先順位 2)
- 通常地上便による出荷 (優先順位 3)

優先順位は、`setPriority` コールを使用してメッセージ・プロデューサ・レベルで指定するか、`send` または `publish` コール中に指定できます。後者は前者をオーバーライドします。

注文入力アプリケーションは、新規の注文情報を FIFO キューに格納します。新規の注文情報は、注文入力アプリケーションによって処理され、入力済注文情報トピックに対して公開されます。注文入力アプリケーションは、エンキュー時刻の順序で新規の注文情報キューからメッセージを取り出します。注文入力アプリケーションは、入力済注文情報を FIFO 優先順位のトピックに格納します。入力済注文情報は、地域の入力済注文情報トピックに伝播されます。各地域では、その地域の入力済注文情報トピックにある注文情報が、出荷の優先順位に従って処理されます。次のコールで注文入力アプリケーションに FIFO 優先順位トピックを作成して、入力済注文情報を格納します。

```
public static void createPriorityTopic(TopicSession jms_session)
{
    AQQueueTableProperty    qt_prop;
    AQQueueTable             pr_qtable;
    AQjmsDestinationProperty dest_prop;
    Topic                    bookedorders_topic;

    try
    {

        /* OE での優先順位キュー・テーブルを作成します。*/
        qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_OBJECT_MESSAGE");
        qt_prop.setComment("Order Entry Priority " +
                           "MultiConsumer Orders queue table");
        qt_prop.setCompatible("8.1");
        qt_prop.setMultiConsumer(true);

        /* FIFO 優先順位による順序付けを設定します。*/
        qt_prop.setSortOrder("priority, enq_time");

        pr_qtable = ((AQjmsSession)jms_session).createQueueTable("OE",
                                                                    "OE_orders_pr_mqtab", qt_prop);

        /* このキュー・テーブルにキューを作成します。*/
        dest_prop = new AQjmsDestinationProperty();

        bookedorders_topic = ((AQjmsSession)jms_session).createTopic(pr_qtable,
                                                                        "OE_bookedorders_topic", dest_prop);

        /* トピックに対するエンキューおよびデキューを使用可能にします。*/
        ((AQjmsDestination)bookedorders_topic).start(jms_session, true, true);

    }
}
```

```

        catch (Exception ex)
        {
            System.out.println("Exception: " + ex);
        }
    }

```

/\* 注文が届くと、注文処理アプリケーションは、次のプロシージャでその注文を入力済注文トピックに公開できます。出荷の優先順位は、注文ごとに指定されています。\*/

```

public static void order_enqueue(TopicSession jms_session, String book_title,
                                int book_qty, int order_num, int cust_no,
                                String cust_name, int ship_priority,
                                String cust_state, String cust_country,
                                String cust_order_type)
{
    BolOrder      order;
    BolCustomer   cust_data;
    BolBook       book_data;
    BolOrderItem[] item_list;
    Topic         topic;
    ObjectMessage  obj_message;
    TopicPublisher tpub;

    try
    {
        book_data = new BolBook(book_title);
        cust_data = new BolCustomer(cust_no, cust_name);

        order = new BolOrder(order_num, cust_data);

        item_list = new BolOrderItem[1];
        item_list[0] = new BolOrderItem(book_data, book_qty);

        order.setItemList(item_list);

        /* OE bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession) jms_session).getTopic("OE",
                                                       "OE_bookedorders_topic");

        /* トピック・パブリッシャを作成します。*/
        tpub = jms_session.createPublisher(topic);

        obj_message = jms_session.createObjectMessage();
        obj_message.setObject(order);
    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
    }
}

```

```
/* メッセージを送信します - 優先順位を指定します。*/
tpub.publish(topic, obj_message, DeliveryMode.PERSISTENT,
             ship_priority,0);

    jms_session.commit();
}
catch (Exception ex)
{
    System.out.println("Exception ex: " + ex);
}
}
```

## 時刻指定：遅延

メッセージをキュー / トピックに対して送信 / 公開するときに、**遅延**を指定できます。遅延は、そのメッセージがメッセージ・コンシューマに対して使用可能になるまでの時間を表します。遅延指定されたメッセージは、遅延が期限切れになり使用可能になるまで待機状態になります。メッセージの遅延は、メッセージ・プロパティ（JMS\_OracleDelay）として指定されます。このプロパティは、JMS 標準では指定されません。これは、JMS メッセージ・プロパティに対する AQ 拡張機能です。

遅延処理には、AQ 用のバックグラウンド・プロセス、キュー・モニターを起動する必要があります。msgid による受信で遅延がオーバーライドされることにも注意してください。

### サンプル・シナリオおよびコード

BooksOnLine アプリケーションでは、遅延は遅延請求処理を実装するために使用できます。請求処理アプリケーションは 1 つのキューを定義して、そのキューの中には、出荷済ですぐには請求されない注文情報が遅延とともに入力されます。たとえば、法人顧客のような顧客アカウントの中には、15 日間経過するまで請求が発行されないクラスがあります。請求アプリケーションは、受け取った出荷済注文情報メッセージを（shippedorders キューから）デキューし、それが法人顧客からの注文である場合は、遅延とともに遅延請求処理キューにエンキューします。サンプル・シナリオは提供されていませんが、遅延は公開についても同様に処理します。

```
public static void defer_billing(QueueSession jms_session,
                                BolOrder deferred_order)
{
    Queue          def_bill_q;
    ObjectMessage   obj_message;
    QueueSender     qsender;

    try
    {
        /* 遅延請求処理キューへのハンドルを取得します。*/
        def_bill_q = ((AQJmsSession)jms_session).getQueue("CBADM",
                                                            "deferbilling_que");

        /* QueueSender を作成します。*/
        qsender = jms_session.createSender(def_bill_q);

        obj_message = jms_session.createObjectMessage();
        obj_message.setObject(deferred_order);

        /* 遅延を 15 日に設定します。
         * 遅延を指定する場合の単位は秒です。*/
    }
}
```

```
obj_message.setIntProperty("JMS_OracleDelay", 15*60*60*24);

qsender.send(obj_message);

jms_session.commit();

    }
    catch (Exception ex)
    {
        System.out.println("Exception " + ex);
    }
}
```

## 時刻指定 : 期限切れ

メッセージ・プロデューサは、期限切れ制限またはメッセージの Time-To-Live (TimeToLive としてコーディングされています) を指定できます。これによって、そのメッセージがメッセージ・コンシューマに対して使用可能な期間が定義されます。

Time-To-Live は、送信 / 公開時に指定するか、メッセージ・プロデューサの Time-To-Live 設定メソッドを使用して指定できます。前者は後者をオーバーライドします。Time-To-Live を実装するには、AQ 用のバックグラウンド・プロセス、キュー・モニターが実行されている必要があります。

### サンプル・シナリオ

BooksOnLine アプリケーションでは、Time-To-Live は受注残処理にあてられる時間を制御するために使用できます。出荷処理アプリケーションは、処理できない書籍注文情報を受注残トピックに送ります。すべての受注残を 1 週間以内に必ず出荷するという方針であれば、1 週間という期限切れとともにメッセージを受注残トピックに公開できます。この場合、1 週間以内に処理されなかった受注残は例外トピックに移されて、EXPIRED というメッセージ状態になります。これは、受注残の出荷方針に従って未出荷になっている注文情報にフラグを付けるために使用できます。

### サンプル・コード

/\* 受注残を back\_order トピックに再度エンキューし、7 日間の Time-To-Live を設定します。すべての受注残は、7 日以内に処理される必要があります。処理されなければ、例外キューに送られます。\*/

```
public static void requeue_back_order(TopicSession jms_session,
                                     String sale_region, BolOrder back_order)
{
    Topic          back_order_topic;
    ObjectMessage   obj_message;
    TopicPublisher  tpub;
    long            timetolive;

    try
    {
        /* 地区に基づく受注残トピックを調べます。*/
        if(sale_region.equals("WEST"))
        {
            back_order_topic = ((AQJmsSession) jms_session).getTopic("WS",
                                                                    "WS_backorders_topic");
        }
        else if(sale_region.equals("EAST"))
```



```
{
    back_order_topic = ((AQJmsSession)jms_session).getTopic("ES",
        "ES_backorders_topic");
}
else
{
    back_order_topic = ((AQJmsSession)jms_session).getTopic("OS",
        "OS_backorders_topic");
}

obj_message = jms_session.createObjectMessage();
obj_message.setObject(back_order);

tpub = jms_session.createPublisher(null);

/* メッセージの期限切れを7日間に設定します。*/
timetolive = 7*60*60*24*1000;           // specified in milliseconds

/* メッセージを公開します。*/
tpub.publish(back_order_topic, obj_message, DeliveryMode.PERSISTENT,
    1, timetolive);

jms_session.commit();
}
catch (Exception ex)
{
    System.out.println("Exception :" + ex);
}
}
```

## メッセージのグループ化

1つのキュー / トピックに属しているメッセージをグループ化して1つのセットにし、一度に1コンシューマしか使用できないようにできます。そのためには、トランザクション処理でのメッセージのグループ化に対応したキュー・テーブルに、そのキュー / トピックを作成する必要があります (9-5 ページの「[キュー・テーブルの作成](#)」を参照)。1つのグループに属するメッセージは、すべて同一のトランザクションで作成される必要があります。また、1つのトランザクションで作成されるメッセージは、すべて同一のグループに属します。この機能によって、ユーザーは複合メッセージを単純メッセージにセグメント化できます。これは AQ 拡張機能であり、JMS 仕様の一部ではありません。

たとえば、あるキュー宛てのメッセージに請求書情報が含まれている場合、そのメッセージは、ヘッダーのメッセージ、詳細情報のメッセージ、フッターのメッセージで構成されるグループとして作成できます。小さいオブジェクトに分割できるイメージやビデオなどの複合ラージ・オブジェクトがメッセージ・ペイロードにある場合は、メッセージのグループ化が非常に有効です。

グループに含まれるメッセージの一般的なメッセージ・プロパティ (優先順位、遅延、期限切れ) は、単にグループの最初のメッセージ (ヘッダー) のプロパティによってのみ判断され、グループの他のメッセージのプロパティは無視されます。

グループ化メッセージのプロパティは、伝播されても保持されます。ただし、メッセージが伝播される宛先トピックも、トランザクション処理でグループ化可能である必要があります。トランザクション処理でグループ化可能なキューからメッセージをデキューするときに、グループ化メッセージのプロパティを保持する場合、他にも認識しておく必要がある制限があります (詳細は、8-55 ページの「[デキューの方法](#)」および 8-68 ページの「[デキューのモード](#)」を参照)。

### サンプル・シナリオ

BooksOnLine アプリケーションでは、メッセージのグループ化は新規の注文を扱うために使用できます。各注文情報には、注文された多数の書籍が1つずつ連続して入っています。Web を経由して注文された項目も同様です。

次の例では、各 send が注文情報の中の個々の書籍に対応し、グループ / トランザクションが1つの完全な注文情報を表します。最初のメッセージにのみ、顧客情報が含まれています。OE\_neworders\_que は、トランザクション処理でグループ化できるキュー・テーブル OE\_orders\_sqtan に定義されることに注意してください。

## サンプル・コード

```

public static void createMsgGroupQueueTable(QueueSession jms_session)
{
    AQQueueTableProperty    sqt_prop;
    AQQueueTable             sq_table;
    AQjmsDestinationProperty dest_prop;
    Queue                    neworders_q;

    try
    {
        /* トランザクション処理でのメッセージのグループ化に対応した単一コンシューマ注文キュー・
        テーブルを作成します。*/
        sqt_prop = new AQQueueTableProperty("BOLADM.order_typ");
        sqt_prop.setComment("Order Entry Single-Consumer Orders queue table");
        sqt_prop.setCompatible("8.1");
        sqt_prop.setMessageGrouping(AQQueueTableProperty.TRANSACTIONAL);

        sq_table = ((AQjmsSession)jms_session).createQueueTable("OE",
            "OE_orders_sqtab", sqt_prop);

        /* OE に対する新しい注文キューを作成します。*/
        dest_prop = new AQjmsDestinationProperty();
        neworders_q = ((AQjmsSession)jms_session).createQueue(sq_table,
            "OE_neworders_que",
            dest_prop);
    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
    }
}

/* このメソッドによって、指定したキューに注文が送られます。*/
public static void enqueue_order(QueueSession jms_session, Queue queue,
    int order_num, String cust_name, int cust_id,
    int book_qty, String book_title)
{
    QueueSender    sender;
    ObjectMessage  obj_message;
    BolOrder       order;
    BolCustomer    cust_data=null;
    BolBook        book_data;
    BolOrderItem[] item_list;

```

```
try
{
    book_data = new BolBook(book_title);

    if(cust_name != null)
    {
        cust_data = new BolCustomer(cust_id, cust_name);
    }

    order = new BolOrder(order_num, cust_data);

    item_list = new BolOrderItem[1];
    item_list[0] = new BolOrderItem(book_data, book_qty);

    order.setItemList(item_list);

    sender = jms_session.createSender(queue);

    obj_message = jms_session.createObjectMessage();

    obj_message.setObject(order);

    sender.send(obj_message);
}
catch (Exception ex)
{
    System.out.println("Exception ex: " + ex);
}
}

/* 注文のグループをエンキューします。*/
public static void enqueue_order_groups(QueueSession jms_session)
{
    Queue neworders_q;

    try
    {
        neworders_q = ((AQJmsSession) jms_session).getQueue("OE",
                                                            "OE_neworders_que");

        /* 最初のグループをエンキューします。*/
        enqueue_order(jms_session, neworders_q, 1, "John", 1000, 2,
```

```
        "John's first book");

enqueue_order(jms_session, neworders_q, 1, null, 0, 1,
    "John's second book");

jms_session.commit();

/* 2 番目のグループをエンキューします。*/
enqueue_order(jms_session, neworders_q, 2, "Mary", 1001, 1,
    "Mary's first book");

enqueue_order(jms_session, neworders_q, 2, null, 0, 1,
    "Mary's second book");

enqueue_order(jms_session, neworders_q, 2, null, 0, 1,
    "Mary's third book");

jms_session.commit();

/* 3 番目のグループをエンキューします。*/
enqueue_order(jms_session, neworders_q, 3, "Scott", 1002, 1,
    "Scott's first book");

enqueue_order(jms_session, neworders_q, 3, null, 0, 2,
    "Scott's second book");

enqueue_order(jms_session, neworders_q, 3, null, 0, 2,
    "Scott's third book");

jms_session.commit();
}
catch (Exception ex)
{
    System.out.println("Exception ex: " + ex);
}
}
```

## メッセージ・コンシューマ機能

- [メッセージの受信](#)
- [受信におけるメッセージのナビゲーション](#)
- [メッセージ受信モード](#)
- [遅延間隔をおいての再試行](#)
- [メッセージ・リスナーを使用したメッセージの非同期受信](#)
- [AQ の例外処理](#)

## メッセージの受信

JMS アプリケーションは、メッセージ・コンシューマを作成することによって、メッセージを受信できます。メッセージは、`receive` コールを使用して同期に受信できるか、メッセージ・リスナーを介して非同期に受信できます。

受信モードには次の 3 つがあります。

- メッセージがコンシューマに届くまでブロック
- 指定時間までブロック
- 非ブロック化

### サンプル・コード：メッセージが届くまでブロック

```
public BolOrder get_new_order1(QueueSession jms_session)
{
    Queue          queue;
    QueueReceiver  qrec;
    ObjectMessage  obj_message;
    BolCustomer    customer;
    BolOrder       new_order = null;
    String         state;

    try
    {
        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQJmsSession) jms_session).getQueue("OE", "OE_neworders_que");

        qrec = jms_session.createReceiver(queue);

        /* キューにメッセージが届くのを待ちます。*/
        obj_message = (ObjectMessage)qrec.receive();

        new_order = (BolOrder)obj_message.getObject();

        customer = new_order.getCustomer();
        state     = customer.getState();

        System.out.println("Order:  for customer " +
                           customer.getName());
    }
    catch (JMSEException ex)
    {
        System.out.println("Exception: " + ex);
    }
}
```

```
    }  
    return new_order;  
}
```

### サンプル・コード : 最長 60 秒間のブロック

```
public BolOrder get_new_order2(QueueSession jms_session)  
{  
    Queue            queue;  
    QueueReceiver    qrec;  
    ObjectMessage     obj_message;  
    BolCustomer       customer;  
    BolOrder          new_order = null;  
    String            state;  
  
    try  
    {  
        /* new_orders キューへのハンドルを取得します。*/  
        queue = ((AQJmsSession) jms_session).getQueue("OE", "OE_neworders_que");  
  
        qrec = jms_session.createReceiver(queue);  
  
        /* キューにメッセージが届くまで 60 秒待ちます。*/  
        obj_message = (ObjectMessage)qrec.receive(60000);  
  
        new_order = (BolOrder)obj_message.getObject();  
  
        customer = new_order.getCustomer();  
        state    = customer.getState();  
  
        System.out.println("Order:  for customer " +  
                           customer.getName());  
    }  
    catch (JMSEException ex)  
    {  
        System.out.println("Exception: " + ex);  
    }  
    return new_order;  
}
```



## サンプル・コード：非ブロック化

```
public BolOrder poll_new_order3(QueueSession jms_session)
{
    Queue          queue;
    QueueReceiver  qrec;
    ObjectMessage  obj_message;
    BolCustomer    customer;
    BolOrder       new_order = null;
    String         state;

    try
    {
        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQJmsSession) jms_session).getQueue("OE", "OE_neworders_que");

        qrec = jms_session.createReceiver(queue);

        /* キューにメッセージが届くのを確認します。*/
        obj_message = (ObjectMessage)qrec.receiveNoWait();

        new_order = (BolOrder)obj_message.getObject();

        customer = new_order.getCustomer();
        state     = customer.getState();

        System.out.println("Order:  for customer " +
                           customer.getName());
    }
    catch (JMSEException ex)
    {
        System.out.println("Exception: " + ex);
    }
    return new_order;
}
```

## 受信におけるメッセージのナビゲーション

コンシューマは、そのセッションで最初に受信するとき、キューまたはトピックで最初のメッセージを取得します。次の受信で次のメッセージを取得し、以後同様に続きます。デフォルトの動作は、FIFO キューおよびトピックでは正常に動作しますが、優先順位によって順序付けられたキューでは正常に動作しません。優先順位が高いメッセージがコンシューマに届く場合、すでに届いているメッセージを削除するまで、このクライアント・プログラムはこのメッセージを受信しません。

コンシューマが、メッセージに対してより効率的にキューのナビゲーションを制御できるように、AQ ナビゲーション・モードが JMS 拡張機能として用意されています。これらのモードは、トピック・サブスクライバ、キュー受信者またはトピック受信者で設定できます。

- `FIRST_MESSAGE` は、コンシューマの位置をキューの先頭にリセットします。このモードでは、コンシューマはキューの一番上にあるメッセージを削除できるため、優先順位によって順序付けられたキューに有効です。
- `NEXT_MESSAGE` は、確立されたコンシューマの位置の後のメッセージを取得します。たとえば、位置が 4 番目のメッセージに確立された後で発行された `NEXT_MESSAGE` は、そのキューの 2 番目のメッセージを取得します。これはデフォルトの動作です。

トランザクションのグループ化については次のとおりです。

- `FIRST_MESSAGE` は、コンシューマの位置をキューの先頭にリセットします。
- `NEXT_MESSAGE` は、位置を同一トランザクションの次のメッセージに設定します。
- `NEXT_TRANSACTION` は、位置を次のトランザクションの最初のメッセージに設定します。

次の方法でメッセージが受信される場合、グループ化トランザクションのプロパティは無効になる場合があります。

- セレクタに相関識別子を指定する受信
- セレクタにメッセージ識別子を指定する受信
- トランザクション・グループのメッセージがすべて受信される前のコミット

キューのナビゲート中に、`NEXT_MESSAGE` または `NEXT_TRANSACTION` オプションを使用したプログラムがキューの最後に到達したとします。ブロッキング受信を指定していた場合は、ナビゲート位置は自動的にそのキューの先頭に変更されます。

デフォルトでは、キュー受信者、トピック受信者またはトピック・サブスクライバは、最初の `receive` コールに `FIRST_MESSAGE` を、次の `receive` コールに `NEXT_MESSAGE` を使用します。

## サンプル・シナリオ

`get_new_orders()` プロシージャは、OE\_neworders\_queue から注文情報を取り出します。各トランザクションはそれぞれの注文情報を参照し、各メッセージはその注文情報に含まれる各書籍に対応しています。`get_orders()` プロシージャは、メッセージをループして書籍注文情報を取り出します。これは、最初の受信の前に、FIRST\_MESSAGE オプションを使用して、位置をキューの先頭にリセットします。それから、「next\_message」ナビゲーション・オプションで、注文情報（トランザクション）から次の書籍（メッセージ）を取り出します。現行のグループまたはトランザクションのすべてのメッセージがフェッチされたという例外が戻されたら、ナビゲーション・オプションを next\_transaction に変更して、次の注文情報の最初の書籍を取り出します。次に、「next\_message」オプションに戻して、同一トランザクションの次のメッセージをフェッチします。同様に、すべての注文情報（トランザクション）がフェッチされるまで、繰り返します。

## サンプル・コード

```
public void get_new_orders(QueueSession jms_session)
{
    Queue          queue;
    QueueReceiver  qrec;
    ObjectMessage  obj_message;
    BolCustomer    customer;
    BolOrder       new_order;
    String          state;
    int             new_orders = 1;

    try
    {

        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQJmsSession) jms_session).getQueue("OE", "OE_neworders_queue");
        qrec = jms_session.createReceiver(queue);

        /* 最初のメッセージにナビゲーションを設定します。*/
        ((AQJmsTopicSubscriber) qrec).setNavigationMode(AQJmsConstants.NAVIGATION_FIRST_
MESSAGE);

        while(new_orders != 0)
        {
            try{

                /* トピックにメッセージが届くのを待ちます。*/
                obj_message = (ObjectMessage) qrec.receiveNoWait();

                if (obj_message != null)    /* キューにはもう注文はありません。*/
```

```
        {
            System.out.println(" No more orders ");
            new_orders = 0;
        }
        new_order = (BolOrder)obj_message.getObject();
        customer = new_order.getCustomer();
        state = customer.getState();

        System.out.println("Order: for customer " +
                           customer.getName());

        /* 次のメッセージを取得します。*/

        ((AQjmsTopicSubscriber)qrec).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_
MESSAGE);

        }catch(AQjmsException ex)
        {   if (ex.getErrorNumber() == 25235)
            {
                System.out.println("End of transaction group");

                ((AQjmsTopicSubscriber)qrec).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_
TRANSACTION);
            }
            else
                throw ex;
        }
    }
    }catch (JMSEException ex)
    {
        System.out.println("Exception: " + ex);
    }
}
```

## メッセージ受信モード

### Point-to-Point モード

デキューするクライアントがキューからメッセージを削除できる通常の受信の他に、JMS では、JMS クライアントがキューで自身に対するメッセージをブラウズできるようにするインタフェースを提供しています。キュー・ブラウザは、キュー・セッションから `createBrowser` メソッドを介して作成できます。

メッセージは、ブラウズ後でも処理可能です。メッセージは、ブラウズされた後は、同時セッションからの `receive` コールがそのメッセージを削除する場合があるため、JMS セッションに再度使用できる保証はないことに注意してください。

一度参照したメッセージが同時 JMS クライアントによって削除されないようにするために、ロック・モードでメッセージを参照できます。このためには、JMS インタフェースに対する AQ 拡張機能を使用して、ロック・モードを持つキュー・ブラウザを作成する必要があります。ロック・モードを持つブラウザによるメッセージのロックは、セッションがコミットまたはロールバックを実行したときに解放されます。

キュー・ブラウザによって参照されたメッセージを削除するには、セッションがキュー受信者を作成し、JMSmessageID をセレクトアとして使用する必要があります。

### サンプル・コード

Point-to-Point 機能のキュー・ブラウザの例を参照してください。

### 取出しを伴わないメッセージの削除

メッセージ・コンシューマは、`receiveNoData` コールを使用して、メッセージを取り出さずにキューまたはトピックから削除できます。これは、アプリケーションがキュー・ブラウザを使用してすでにメッセージを調べている場合に有効です。このモードによって、JMS クライアントは、データベースからペイロードを取り出す場合のオーバーヘッドを回避できます。このオーバーヘッドは、大量のメッセージでは相当量になる可能性があります。

### サンプル・シナリオおよびコード

次の BooksOnLine のシナリオでは、海外からの注文情報（メキシコおよびカナダ向け）が、通商政策および送料割引の理由で別々に処理されます。したがって、（他の同時ユーザが削除できないように）メッセージをキュー・ブラウザを介してロック・モードで参照し、顧客の国（メッセージ・ペイロード）をチェックします。顧客の国がメキシコまたはカナダの場合、（ペイロードはすでにわかっているため）`remove with no data` モードでメッセージをキューから削除します。これを実行しないと、そのメッセージのロックはコミット・コールによって解放されてしまうためです。`receive` コールには、ロック・モードのブラウザから取得したメッセージ識別子を使用することに注意してください。

```

public void process_international_orders(QueueSession jms_session)
{
    QueueBrowser    browser;
    Queue            queue;
    ObjectMessage    obj_message;
    BolOrder         new_order;
    Enumeration      messages;
    String            customer_name;
    String            customer_country;
    QueueReceiver    qrec;
    String            msg_sel;

    try
    {
        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");

        /* 至急 (RUSH) 注文を調べるためのブラウザを作成します。*/
        browser = ((AQjmsSession) jms_session).createBrowser(queue, null, true);

        for (messages = browser.getEnumeration() ; messages.hasMoreElements() ;)
        {
            obj_message = (ObjectMessage) messages.nextElement();

            new_order = (BolOrder) obj_message.getObject();

            customer_name = new_order.getCustomer().getName();

            customer_country = new_order.getCustomer().getCountry();

            if (customer_country equals ("Canada") || customer_country equals (
                "Mexico"))
            {
                System.out.println("Order for Canada or Mexico");
                msg_sel = "JMSMessageID = '" + obj_message.getJMSMessageID() + "'";
                qrec = jms_session.createReceiver(queue, msg_sel);
                ((AQjmsQueueReceiver) qrec).receiveNoData();
            }
        }
    } catch (JMSException ex)
    { System.out.println("Exception " + ex);
    }
}

```

## 遅延間隔をおいての再試行

### 再試行の最大値

キュー / トピックからメッセージを受信するトランザクションが失敗した場合、そのメッセージを削除する試行に失敗したとみなされます。AQ は、メッセージ削除に失敗した試行回数をメッセージ履歴に記録します。

さらに、AQ では、アプリケーションがメッセージに対する再試行の最大回数を、キュー / トピック・レベルで指定できます。メッセージの削除がこの数よりも多く失敗した場合、メッセージは例外キューに移され、アプリケーションで使用できなくなります。

### 再試行の遅延

メッセージを受信するトランザクションが異常終了した場合、たとえば、書籍が在庫不足のため注文を受けることができなかったなどの、不十分な状態が原因である場合があります。在庫更新は 12 時間ごとに行われているため、更新後に再試行することも有効です。4 回試行しても注文を受けられなかった場合は、問題がある可能性があります。

AQ では、ユーザーは `max_retries` とともに `retry_delay` も指定できます。これは、受信の試行に失敗したメッセージを、「`retry_delay`」間隔後に引き続きキューで参照し、デキューできることを意味します。そのときまで、このメッセージは `WAITING` 状態になります。AQ バックグラウンド・プロセス、タイム・マネージャは、再試行のディレイ・プロパティを施行します。

再試行の最大回数および再試行の遅延は、キュー / トピックのプロパティです。このプロパティは、キュー / トピックの作成時、またはキュー / トピックに対する変更メソッドを介して設定できます。`max_retries` のデフォルト値は 5 です。

### サンプル・シナリオおよびコード

在庫不足のため注文を受けることができない場合、その注文を処理するトランザクションは異常終了されます。`booked_orders` トピックは、`max_retries = 4` 時間および `retry_delay = 12` 時間で設定されます。したがって、注文は、2 日間履行されなければ、例外キューに移されます。

```
public BolOrder process_booked_order(TopicSession jms_session)
{
    Topic          topic;
    TopicSubscriber tsubs;
    ObjectMessage   obj_message;
    BolCustomer     customer;
    BolOrder        booked_order = null;
```

```
String          country;
int             i = 0;

try
{
    /* OE_bookedorders_topic へのハンドルを取得します。*/
    topic = ((AQJmsSession)jms_session).getTopic("WS",
                                                "WS_bookedorders_topic");

    /* ローカル・サブスクライバを作成します - 西部地区へのメッセージを追跡します。*/
    tsubs = jms_session.createDurableSubscriber(topic, "SUBS1",
                                                "Region = 'Western' ",
                                                false);

    /* トピックにメッセージが届くのを待ちます。*/
    obj_message = (ObjectMessage)tsubs.receive(10);

    booked_order = (BolOrder)obj_message.getObject();

    customer = booked_order.getCustomer();
    country   = customer.getCountry();

    if (country == "US")
    {
        jms_session.commit();
    }
    else
    {
        jms_session.rollback();
        booked_order = null;
    }
} catch (JMSEException ex)
{ System.out.println("Exception " + ex) ;}

return booked_order;
}
```



## メッセージ・リスナーを使用したメッセージの非同期受信

### メッセージ・コンシューマ用のメッセージ・リスナー

JMS クライアントは、コンシューマで使用可能な `setMessageListener` メソッドを使用してメッセージ・リスナーを設定することによって、メッセージを非同期に受信できます。

メッセージ・コンシューマにメッセージが届いた場合、メッセージ・リスナーの `onMessage` メソッドがそのメッセージで起動されます。メッセージ・リスナーは、メッセージの受信をコミットまたは異常終了できます。メッセージ・リスナーは、JMS 接続が停止されている場合、メッセージを受信しません。一度メッセージ・リスナーがコンシューマに対して設定されると、メッセージの受信に `receive` コールを使用することはできません。

### 例

新規注文のキューを処理するアプリケーションは、そのキューからメッセージを非同期受信するように設定できます。

```
public class OrderListener implements MessageListener
{
    QueueSession    the_sess;

    /* コンストラクタ */
    OrderListener(QueueSession my_sess)
    {
        the_sess = my_sess;
    }

    /* メッセージ・リスナー・インタフェース */
    public void onMessage(Message m)
    {
        ObjectMessage    obj_msg;
        BolCustomer       customer;
        BolOrder          new_order = null;

        try {
            /* JMS オブジェクト・メッセージへキャストします。*/
            obj_msg = (ObjectMessage)m;

            /* 有効な情報を印刷します。*/
            new_order = (BolOrder)obj_msg.getObject();
            customer = new_order.getCustomer();
            System.out.println("Order:  for customer " + customer.getName());
        }
    }
}
```

```

        /* 注文処理メソッドをコールします。
        * 注意：他の場所で定義されていることを前提としています。
        * /
        process_order(new_order);

        /* メッセージの非同期受信をコミットします。*/
        the_sess.commit();
    } catch (JMSEException ex)
    { System.out.println("Exception " + ex) ;}

    }

}

public void setListener1(QueueSession jms_session)
{
    Queue            queue;
    QueueReceiver    qrec;
    MessageListener  ourListener;

    try
    {
        /* new_orders キューへのハンドルを取得します。*/
        queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");

        /* キュー受信者を作成します。*/
        qrec = jms_session.createReceiver(queue);

        /* メッセージ・リスナーを作成します。*/
        ourListener = new OrderListener(jms_session);

        /* 受信者に対してメッセージ・リスナーを設定します。*/
        qrec.setMessageListener(ourListener);
    }
    catch (JMSEException ex)
    {
        System.out.println("Exception: " + ex);
    }
}
}

```

## セッションのすべてのコンシューマ用のメッセージ・リスナー

JMS クライアントは、セッションでメッセージ・リスナーを設定することによって、そのセッションのすべてのコンシューマに対してメッセージを非同期に受信できます。

セッションのどのメッセージ・コンシューマにメッセージが届いても、メッセージ・リスナーの `onMessage` メソッドがそのメッセージで起動されます。メッセージ・リスナーは、メッセージの受信をコミットまたは異常終了できます。メッセージ・リスナーは、JMS 接続が停止されている場合、メッセージを受信しません。一度メッセージ・リスナーが設定されると、そのセッションでは、その他のメッセージ受信モードを使用できません。

## サンプル・シナリオおよびコード

BooksOnLine シナリオの顧客サービス・コンポーネントでは、異なるデータベースから顧客サービス・トピックに届いたメッセージにはその状態が示されます。顧客サービス・アプリケーションはそのトピックを監視し、顧客の注文情報に関するメッセージを検出するたびに、`order_status_table` の注文情報状態を更新します。アプリケーションはセッション・リスナーを使用して様々なトピックを監視します。トピックのいずれかにメッセージがあるときは、常に、セッション・メッセージ・リスナーの `onMessage` メソッドが起動されます。

```
/* メッセージ・リスナーのクラスを定義します。*/
public class CustomerListener implements MessageListener
{
    TopicSession    the_sess;

    /* コンストラクタ */
    CustomerListener(TopicSession my_sess)
    {
        the_sess = my_sess;
    }

    /* メッセージ・リスナー・インタフェース */
    public void onMessage(Message m)
    {
        ObjectMessage    obj_msg;
        BolCustomer       customer;
        BolOrder          new_order = null;

        try
        {
            /* JMS オブジェクト・メッセージへキャストします。*/
            obj_msg = (ObjectMessage)m;

            /* 有効な情報を印刷します。*/
            new_order = (BolOrder)obj_msg.getObject();
            customer = new_order.getCustomer();
            System.out.println("Order:  for customer " + customer.getName());
        }
    }
}
```

```

        /* 更新状態メソッドをコールします。
        * 注意：他の場所で定義されていることを前提としています。
        * /
        update_status(new_order, new_order.getStatus());

        /* メッセージの非同期受信をコミットします。*/
        the_sess.commit();
    }catch (JMSEException ex)
    {
        System.out.println("Exception: " + ex);
    }
}

}

public void monitor_status_topics(TopicSession jms_session)
{
    Topic[]          topic = new Topic[4];
    TopicSubscriber[] tsubs= new TopicSubscriber[4];

    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic[0] = ((AQjmsSession) jms_session).getTopic("CS",
                                                         "CS_bookedorders_topic");
        tsubs[0] = jms_session.createDurableSubscriber(topic[0], "BOOKED_ORDER");

        topic[1] = ((AQjmsSession) jms_session).getTopic("CS",
                                                         "CS_billedorders_topic");
        tsubs[1] = jms_session.createDurableSubscriber(topic[1], "BILLED_ORDER");

        topic[2] = ((AQjmsSession) jms_session).getTopic("CS",
                                                         "CS_backdorders_topic");
        tsubs[2] = jms_session.createDurableSubscriber(topic[2], "BACKED_ORDER");

        topic[3] = ((AQjmsSession) jms_session).getTopic("CS",
                                                         "CS_shippedorders_topic");
        tsubs[3] = jms_session.createDurableSubscriber(topic[3], "SHIPPED_ORDER");

        MessageListener mL = new CustomerListener(jms_session);

        /* セッションのメッセージ・リスナーを設定します。*/
        jms_session.setMessageListener(mL);

    }catch (JMSEException ex)

```

```
{ System.out.println("Exception: " + ex); }  
}
```

## AQ の例外処理

AQ は、例外キュー、期限切れ、再試行の最大回数および再試行の遅延の 4 つの統合メカニズムによって、アプリケーションの例外処理をサポートします。

例外キューは、期限切れまたは処理できないすべてのメッセージのリポジトリになります。アプリケーションから例外キューには直接エンキューできません。ただし、期限切れまたは処理できないメッセージを処理するアプリケーションは、例外キューからこれらのメッセージを受信または削除できます。

例外キューからメッセージを取り出すには、JMS クライアントは Point-to-Point インタフェースを使用する必要があります。トピック用のメッセージの例外キューは、使用可能な複数のコンシューマでキュー・テーブルに作成する必要があります。他のキューと同様に、例外キューも `AQOracleQueue` クラスで `start` メソッドを使用して、メッセージを受信できる必要があります。例外キューをエンキュー可能に設定しようとすると、例外が発生します。

例外キューは、「JMS\_OracleExcpQ」と呼ばれるプロバイダ (Oracle) 固有のメッセージ・プロパティです。これは、メッセージの送信 / 公開前に、そのメッセージで設定できます。例外キューが指定されていないと、デフォルトの例外キューが使用されます。キュー / トピックがキュー・テーブル (たとえば QTAB) の中に作成されている場合、デフォルトの例外キューは `AQ$_QTAB_E` と呼ばれます。デフォルトの例外キューは、キュー・テーブルが作成されるときに自動的に作成されます。

メッセージは、次の条件が成立するときに AQ によって例外キューに移されます。

- そのメッセージが、指定された Time-To-Live 内にデキューされない場合。複数のサブスクリバを指定したメッセージの場合、指定されていないが指定された Time-To-Live 内にそのメッセージをデキューできない受信者が 1 つでもあると、例外キューに移されます。Time-To-Live がメッセージに (publish または send コールでも、またはパブリッシャまたは送信者としても) 指定されていない場合、メッセージは期限切れになりません。
- メッセージが正常に受信された場合。ただし、メッセージ処理中のエラーのため、アプリケーションは受信を実行したトランザクションを異常終了します。そのメッセージはキュー / トピックに戻され、メッセージ受信のために待機中のどのアプリケーションでも使用可能になります。これは失敗したメッセージ受信の試行であるため、その再試行回数は更新されます。

メッセージの再試行回数が、メッセージが常駐するキュー / トピックに指定された最大値を超える場合、そのメッセージは例外キューに移されます。メッセージが複数のサブスクライバを持つ場合、そのメッセージは、すべての受信者が再試行制限を超えたときにのみ、例外キューに移されます。

アプリケーションがトランザクション全体を異常終了するか、受信前のセーブポイントまでロールバックしたときは、受信処理がロールバックまたは UNDO されたとみなされます。

- クライアント・プログラムがメッセージの受信に成功したにもかかわらず、トランザクションをコミットする前に終了した場合。

## サンプル・シナリオ

遅延間隔をおいたセクションの再試行には、MAX\_RETRIES の例があります。

BooksOnLine アプリケーションでは、各出荷地域のビジネス・ルールによって、すぐに応じることができない注文情報は受注残キューに移されます。受注残アプリケーションは、毎日 1 回その注文情報を出荷しようとしています。7 日以内に出荷できない場合、その注文情報は例外キューに移されて特別に処理されます。例外キューにメッセージの Time-To-Live プロパティを使用して、この処理を実装します。

1. 例外キュー WS\_back\_order\_exp\_que を作成します。

```
public void create_excp_que(TopicSession jms_session)
{
    AQQueueTable    q_table;
    Queue           excpq;

    try {
        /* 複数コンシューマ・フラグが TRUE であるキュー・テーブルに例外キューを作成します。*/
        q_table = ((AQJmsSession)jms_session).getQueueTable("WS", "WS_orders_
mqtab");

        AQJmsDestinationProperty dest_prop = new AQJmsDestinationProperty();

        dest_prop.setQueueType(AQJmsDestinationProperty.EXCEPTION_QUEUE);
        excpq = ((AQJmsSession)jms_session).createQueue(q_table,
            "WS_back_orders_excp_que",
            dest_prop);
        /* メッセージを受信（デキュー）するための例外キューを起動します。*/
        ((AQJmsDestination)excpq).start(jms_session, false, true);
    }
}
```

```

        catch (JMSEException ex)
        { System.out.println("Exception " + ex); }
    }

```

2. 受注残キューに関するメッセージを、WS\_back\_orders\_excp\_que に設定した例外キューとともに公開します。

```

public static void requeue_back_order(TopicSession jms_session,
                                     String sale_region, BolOrder back_order)
{
    Topic            back_order_topic;
    ObjectMessage    obj_message;
    TopicPublisher   tpub;
    long             timetolive;

    try
    {
        back_order_topic = ((AQJMSSession) jms_session).getTopic("WS",
                                                                "WS_backorders_topic");
        obj_message = jms_session.createObjectMessage();
        obj_message.setObject(back_order);

        /* 例外キューを設定します。*/
        obj_message.setStringProperty("JMS_OracleExcpQ", "WS.WS_back_orders_excp_
que");

        tpub = jms_session.createPublisher(null);

        /* メッセージの期限切れを7日間に設定します。*/
        timetolive = 7*60*60*24*1000;           // specified in milliseconds

        /* メッセージを公開します。*/
        tpub.publish(back_order_topic, obj_message, DeliveryMode.PERSISTENT,
                    1, timetolive);
        jms_session.commit();
    }
    catch (Exception ex)
    {
        System.out.println("Exception :" + ex);
    }
}

```

3. Point-to-Point インタフェースを使用して、例外キューから期限切れメッセージを受信します。

```

public BolOrder get_expired_order(QueueSession jms_session)
{
    Queue          queue;
    QueueReceiver  qrec;
    ObjectMessage  obj_message;
    BolCustomer    customer;
    BolOrder       exp_order = null;

    try
    {
        /* 例外キューへのハンドルを取得します。*/
        queue = ((AQjmsSession) jms_session).getQueue("WS", "WS_back_orders_excp_
que");

        qrec = jms_session.createReceiver(queue);

        /* キューにメッセージが届くのを待ちます。*/
        obj_message = (ObjectMessage)qrec.receive();

        exp_order = (BolOrder)obj_message.getObject();

        customer = exp_order.getCustomer();

        System.out.println("Expired Order:  for customer " +
                           customer.getName());

    }
    catch (JMSEException ex)
    {
        System.out.println("Exception: " + ex);
    }
    return exp_order;
}

```



## 伝播

- リモート・サブスクライバ
- 伝播スケジュール
- 拡張伝播スケジュール機能
- 伝播中の例外処理

## リモート・サブスクライバ

この機能によって、同じデータベースに接続しなくても、アプリケーション同士が互いに通信できます。

AQ によって、リモート・サブスクライバ（他のデータベースにあるサブスクライバ）がトピックにサブスクライブできます。トピックに対して公開されたメッセージがリモート・サブスクライバの基準を満たしている場合、AQ はリモート・サブスクライバに指定されているリモート・データベースにあるキュー / トピックにメッセージを自動的に伝播します。

スナップショット（ジョブ・キュー）のバックグラウンド・プロセスが伝播を実行します。伝播は、データベース・リンクおよび Net8 を使用して行われます。

リモート・サブスクライバを実装するには、次の 2 つの方法があります。

- `createRemoteSubscriber` メソッドは、トピック上またはトピックに対してリモート・サブスクライバを作成するために使用します。このリモート・サブスクライバは、クラス `AQjmsAgent` のインスタンスとして指定されます。
- `AQjmsAgent` には、名前およびアドレスがあります。アドレスは、キュー / トピックおよびサブスクライバのデータベースへのデータベース・リンク（`dblink`）で構成されます。

リモート・サブスクライバには、次の 2 種類があります。

**ケース 1** リモート・サブスクライバがトピックである場合。これは、`AQjmsAgent` オブジェクトのリモート・サブスクライバに名前が指定されず、アドレスがトピックである場合に発生します。サブスクライバのサブスクリプションを満たすメッセージが、リモート・トピックに伝播されます。伝播されたメッセージは、それが満たすリモート・トピックのすべてのサブスクリプションに対して使用可能になります。

**ケース 2** メッセージに対して特定のリモート受信者を指定する場合。リモート・サブスクリプションは、リモート・データベースにある特定のコンシューマに対して指定できます。リモート受信者の名前が（`AQjmsAgent` オブジェクトに）指定される場合、サブスクリプションを満たすメッセージが、その受信者専用のリモート・データベースに伝播されます。リモート・データベースにある受信者は、`TopicReceiver` インタフェースを使用してメッセージを取り出します。リモート・サブスクリプションは、Point-to-Point キューに対して指定することもできます。

### ケース 1 のサンプル・シナリオ

注文入力アプリケーションおよび西部向け出荷処理アプリケーションが、`db1` および `db2` という異なるデータベース上にあるとします。また、データベース `db1`（注文入力データベース）からデータベース `db2`（西部向け出荷処理データベース）へのデータベース・リンク `dblink_oe_ws` があるとします。`db2` にある `WS_bookedorders_topic` は、`db1` にある `OE_bookedorders_topic` のリモート・サブスクライバです。

## ケース 2 のサンプル・シナリオ

注文入力アプリケーションおよび西部向け出荷処理アプリケーションが、db1 および db2 という異なるデータベース上にあるとします。さらに、db1（ローカル注文入力データベース）から db2（西部向け出荷処理データベース）へのデータベース・リンク dblink\_oe\_ws があるとします。db2 の WS\_bookedorders\_topic にあるエージェント「優先順位」は、db1 にある OE\_bookedorders\_topic のリモート・サブスクライバです。WS\_bookedorders\_topic に伝播されたメッセージは、「優先順位」専用です。

```
public void remote_subscriber(TopicSession jms_session)
{
    Topic          topic;
    ObjectMessage   obj_message;
    AQjmsAgent      remote_sub;

    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession)jms_session).getTopic("OE",
                                                    "OE_bookedorders_topic");
        /* リモート・サブスクライバ（未定義で db2 のトピック WS_booked_orders_topic を指す名前）を作成します。*/
        remote_sub = new AQjmsAgent(null, "WS.WS_bookedorders_topic@dblink_oe_ws");

        /* 西部地区の注文をサブスクライブします。*/
        ((AQjmsSession)jms_session).createRemoteSubscriber(topic, remote_sub, "Region
= 'Western' ");
    }
    catch (JMSEException ex)
    { System.out.println("Exception :" + ex); }
    catch (java.sql.SQLException ex1)
    {System.out.println("SQL Exception :" + ex1); }
}
```

データベース db2 - 出荷処理データベース:WS\_booked\_orders\_topic は 2 つのサブスクライバを持ち、1 つは優先順位による出荷用で、もう 1 つは通常の出荷用です。注文入力データベースからのメッセージは、出荷処理データベースに伝播され、正しいサブスクライバに配信されます。優先順位による注文には、優先順位が 1 のメッセージがあります。

```
public void get_priority_messages(TopicSession jms_session)
{
    Topic          topic;
    TopicSubscriber tsubs;
    ObjectMessage   obj_message;
```

```

        BolCustomer      customer;
        BolOrder         booked_order;

    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession)jms_session).getTopic("WS",
                                                    "WS_bookedorders_topic");

        /* ローカル・サブスクライバを作成します - 優先するメッセージ用です。*/
        tsubs = jms_session.createDurableSubscriber(topic, "PRIORITY",
                                                    " JMSPriority = 1 ", false);

        obj_message = (ObjectMessage) tsubs.receive();

        booked_order = (BolOrder)obj_message.getObject();
        customer = booked_order.getCustomer();
        System.out.println("Priority Order:  for customer " + customer.getName());

        jms_session.commit();
    }
    catch (JMSEException ex)
    { System.out.println("Exception :" + ex); }
}

public void  get_normal_messages(TopicSession jms_session)
{
    Topic          topic;
    TopicSubscriber tsubs;
    ObjectMessage  obj_message;
    BolCustomer    customer;
    BolOrder       booked_order;

    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession)jms_session).getTopic("WS",
                                                    "WS_bookedorders_topic");

        /* ローカル・サブスクライバを作成します - 優先するメッセージ用です。*/
        tsubs = jms_session.createDurableSubscriber(topic, "PRIORITY",
                                                    " JMSPriority > 1 ", false);

        obj_message = (ObjectMessage) tsubs.receive();
    }
}

```

```

        booked_order = (BolOrder)obj_message.getObject();
        customer = booked_order.getCustomer();
        System.out.println("Normal Order: for customer " + customer.getName());

        jms_session.commit();
    }
    catch (JMSEException ex)
    { System.out.println("Exception :" + ex); }
}

public void remote_subscriber1(TopicSession jms_session)
{
    Topic            topic;
    ObjectMessage    obj_message;
    AQjmsAgent       remote_sub;

    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession)jms_session).getTopic("OE",
                                                    "OE_bookedorders_topic");
        /* リモート・サブスクライバ（優先順位が高く、db2 のトピック WS_booked_orders_topic
        を指す名前）を作成します。*/
        remote_sub = new AQjmsAgent("Priority", "WS.WS_bookedorders_topic@dblink_oe_
ws");

        /* 西部地区の注文をサブスクライブします。*/
        ((AQjmsSession)jms_session).createRemoteSubscriber(topic, remote_sub, "Region
= 'Western' ");
    }
    catch (JMSEException ex)
    { System.out.println("Exception :" + ex); }
    catch (java.sql.SQLException ex1)
    {System.out.println("SQL Exception :" + ex1); }
}

Remote database:
database db2 - Western Shipping database.
/* サブスクライバの優先順位が高いメッセージを取得します。*/
public void get_priority_messages1(TopicSession jms_session)
{

```

```

Topic          topic;
TopicReceiver  treds;
ObjectMessage  obj_message;
BolCustomer    customer;
BolOrder       booked_order;

try
{
    /* OE_bookedorders_topic へのハンドルを取得します。*/
    topic = ((AQjmsSession)jms_session).getTopic("WS",
                                                  "WS_bookedorders_topic");

    /* WS_bookedorders_topic へのリモート・サブスクリプション用のローカル受信者
    「Priority」を作成します。*/
    treds = ((AQjmsSession)jms_session).createTopicReceiver(topic, "Priority",
null);

    obj_message = (ObjectMessage) treds.receive();

    booked_order = (BolOrder)obj_message.getObject();
    customer = booked_order.getCustomer();
    System.out.println("Priority Order:  for customer " +  customer.getName());

    jms_session.commit();
}
catch (JMSEException ex)
{ System.out.println("Exception :" + ex); }
}

```

## 伝播スケジュール

伝播は、メッセージがターゲットの接続先データベースに伝播されるすべてのトピックについて、`schedule_propagation` メソッドを介してスケジュールされる必要があります。

スケジュールは時間の枠を示し、メッセージはその枠内でソース・トピックから伝播されます。この時間枠は、ネットワーク通信量、ソース・データベースの負荷、接続先データベースの負荷などの多くの要因に左右されます。したがって、スケジュールは特定のソースおよび宛先に合せて作成する必要があります。スケジュールが作成されると、ジョブは自動的にジョブ・キューに発行され、伝播が処理されます。

伝播スケジュールのための運用管理コールによって、スケジュール管理の柔軟性が向上します (9-65 ページの「[キューの伝播のスケジュール](#)」を参照)。あるスケジュールの永続時間または伝播枠パラメータによって、伝播が行われる時間枠が指定されます。永続時間が指定されない場合、時間枠は無制限の単一枠になります。枠を定期的に繰り返す必要がある場合、連続する枠の間の周期的間隔を定義する `next_time` 関数を使用して有限の永続時間を指定します。

スケジュールの待ち時間パラメータが関係するのは、特定のキューに伝播する必要があるメッセージが1つもないときのみです。このパラメータは、キューを再チェックする時間間隔を指定します。待ち時間パラメータが適用された場合、`job_queue_processes` 用の `job_queue_interval` パラメータは、待ち時間パラメータより小さい必要がある点に注意してください。特定のキューに定義された伝播スケジュールは、そのキューの有効期間中いつでも変更または削除できます。さらに、(スケジュールを削除するかわりに) 一時的に使用禁止にするコール、および使用禁止のスケジュールを使用可能にするコールがあります。メッセージがスケジュール内で伝播されているとき、そのスケジュールはアクティブです。すべての運用管理コールは、スケジュールがアクティブかどうかに関係なく実行されます。アクティブなスケジュールでは、コールが実行されるまでに数秒かかります。

伝播が開始されるには、ジョブ・キュー・プロセスを起動する必要があります。少なくとも2つのジョブ・キュー・プロセスを起動する必要があります。接続先データベースへのデータベース・リンクも、有効である必要があります。伝播のソースおよび宛先トピックは、同じメッセージ型である必要があります。リモート・トピックは、エンキューできる必要があります。データベース・リンクのユーザーも、リモート・トピックに対するエンキュー権限を持っている必要があります。

### サンプル・コード

```
public void schedule_propagation(TopicSession jms_session)
{
    Topic          topic;
```

```

try
{
    /* OE_bookedorders_topic へのハンドルを取得します。*/
    topic = ((AQjmsSession)jms_session).getTopic("WS",
                                                "WS_bookedorders_topic");

    /* 存続期間が 5 分、待ち時間が 20 秒で伝播を直接スケジュールします。*/
    ((AQjmsDestination)topic).schedulePropagation(jms_session, "dba", null,
                                                new Double(5*60), null, new Double(20));
}catch (JMSEException ex)
{System.out.println("Exception: " + ex);}
}

```

Propagation schedule parameters can also be altered.

```

/* 存続時間を 10 分、待ち時間を 0 に変更します。*/
public void alter_propagation(TopicSession jms_session)
{
    Topic          topic;
    try
    {
        /* OE_bookedorders_topic へのハンドルを取得します。*/
        topic = ((AQjmsSession)jms_session).getTopic("WS",
                                                        "WS_bookedorders_topic");

        /* 存続期間が 5 分、待ち時間が 20 秒で伝播を直接スケジュールします。*/
        ((AQjmsDestination)topic).alterPropagationSchedule(jms_session, "dba",
                                                            new Double(10*60), null, new Double(0));
    }catch (JMSEException ex)
    {System.out.println("Exception: " + ex);}
}

```



## 拡張伝播スケジュール機能

スケジュールに関する詳細情報は、伝播のために定義されたカタログ・ビューから取得できます。アクティブ・スケジュールに関する情報、たとえば、そのスケジュールを処理しているバックグラウンド・プロセス名、伝播を処理しているセッションのSID（セッションのシリアル番号）、スケジュールを処理する Oracle インスタンス（OPS が使用されているときは関連があります）などの情報は、そのカタログ・ビューから取得できます。同じカタログ・ビューによって、先行して正常に実行されたスケジュール（最後に正常に伝播されたメッセージ）および次に実行されるスケジュールに関する情報も得られます。

各スケジュールには伝播の詳細情報が保持されます。これには、スケジュールの中で伝播されたメッセージの合計数およびバイトの合計数、伝播枠の中で伝播されたメッセージの最大数および平均数、伝播されたメッセージの平均サイズおよび平均時間が含まれます。このような統計は、キュー管理者が最も効果的なスケジュール調整のために利用できるように設計されています。

伝播機能には、障害対処およびエラー・レポートが組み込まれています。たとえば、指定されたデータベース・リンクが無効な場合、リモート・データベースが使用できない場合、またはリモート・トピック / キューにエンキューできない場合、適切なエラー・メッセージがレポートされます。伝播は指数バックオフ・スキームを使用して、障害が発生したスケジュールからの伝播を再試行します。あるスケジュールが続けて障害が発生したときは、最初の再試行は 30 秒後、次の再試行は 60 秒後、3 回目の再試行は 120 秒後、というように続きます。再試行時間が現行の伝播枠の期限切れ時刻を超える場合は、次の再試行は、次の伝播枠の開始時刻に行われます。最大 16 回の再試行が行われた後、そのスケジュールは自動的に使用禁止になります。障害のためにスケジュールが自動的に使用禁止になると、関連情報がアラート・ログに書き込まれます。どの場合でも、スケジュールに障害が発生しているか、発生している場合は継続的な障害がいくつあるかということが、障害の原因および直前の障害の発生時刻を示すエラー・メッセージによってチェックできます。この情報を調べることで、管理者は障害を回復し、スケジュールを使用可能にできます。再試行の間に伝播が成功したときは、障害の数は 0（ゼロ）にリセットされます。伝播機能には OPS サポートが組み込まれていますが、ユーザーおよび管理者には完全に透過的です。伝播を処理するジョブは、ソース・トピックが常駐しているキュー・テーブルの所有者と同じインスタンスに送られます。あるインスタンスに障害があってトピックを保存しているキュー・テーブルが他のインスタンスに移される場合は、伝播ジョブも必ず自動的に新しいインスタンスに移行されます。これによって、インスタンス間の ping 操作は最小限に抑えられ、パフォーマンスが向上します。伝播は、同時スケジュールをいくつでも処理できるように設計されています。

`job_queue_processes` の数は、最大 36 に制限され、その中のいくつかは伝播以外の関連ジョブを処理するために使用される場合があることに注意してください。このように、伝播にはマルチタスキングおよびロード・バランスのサポートが組み込まれています。伝播アルゴリズムは、複数スケジュールが単一スナップショット（ジョブ・キュー）のプロセスによって処理できるように設計されています。ジョブ・キュー・プロセスに対する伝播の負荷は、異なるソース・トピックからのメッセージ到着割合に基づいて偏りが発生する場合があります。あるプロセスが数個のアクティブ・スケジュールによって過負荷になっている一方で、別のプロセスは受動的なスケジュールが多いために余力があるというとき、伝播はプロセス間で負荷が均等になるようにスケジュールを自動的に再分配します。

## サンプル・シナリオ

BooksOnLine の例では、`OE_bookedorders_topic` は、そこに含まれるメッセージが別の出荷サイトに伝播されるためビジーになります。エラー・チェックおよびスケジュール監視のための拡張伝播スケジューリングをサポートしているコールを、次のサンプル・コードで示します。

## サンプル・コード

```
CONNECT OE/OE;
/* 平均を求めます。*/
select avg_time, avg_number, avg_size from user_queue_schedules;

/* 合計を求めます。*/
select total_time, total_number, total_bytes from user_queue_schedules;

/* 枠の最大数を求めます。*/
select max_number, max_bytes from user_queue_schedules;

/* 現在のスケジュールに関する詳細情報を取得します。*/
select process_name, session_id, instance, schedule_disabled
       from user_queue_schedules;

/* 最後に実行した日時、次に実行する日時を求めます。*/
select last_run_date, last_run_time, next_run_date, next_run_time
       from user_queue_schedules;

/* 最新のエラー情報を取得します（ある場合）。*/
select failures, last_error_msg, last_error_date, last_error_time
       from user_queue_schedules;
```

## 伝播中の例外処理

ネットワーク障害のようなシステム・エラーが発生した場合、AQ は指数関数的に実行間隔をあけてメッセージを伝播する試みを継続します。状況がアプリケーション・エラーを示しているとき、メッセージの伝播でエラーが発生した場合は、AQ はそのメッセージに UNDELIVERABLE というマークを付けます。

このようなエラーは、リモート・キュー / トピックが存在しないときやソース・キュー / トピックとリモート・キュー / トピックの型が一致しない場合に発生します。このような状況では、ユーザーは DBA\_SCHEDULER ビューを問い合わせ、特定の宛先への伝播中に発生した最新のエラーを調べます。\$ORACLE\_HOME/rdbms/log ディレクトリのトレース・ファイルには、そのエラーに関する追加情報があります。



---

## JMS 管理インタフェース：基本操作

### ユースケース・モデル

この章では、Oracle アドバンスド・キューイングの管理インタフェースをユースケースに沿って説明します。それぞれの操作（[キュー・テーブルの作成](#)など）を、その操作名ごとにユースケースの点から説明します。この章の先頭に、すべてのユースケースを示します（13-2 ページの「[ユースケース・モデル：JMS 管理インタフェース – 基本操作](#)」を参照）。

### ユースケース・モデルの図形概要

[図 13-1 「ユースケース図：JMS 管理者インタフェース – 基本操作」](#) に、すべてのユースケースを 1 つの図にまとめています。

### 個々のユースケース

個々のユースケースは、次のように配置されています。

- **ユースケース図：**ユースケースを表す図
- **用途：**このユースケースの用途
- **使用上の注意：**実装に有効なガイドライン
- **構文：**このアクティビティの実行に使用する主な構文
- **例：**各プログラム環境でのユースケースの例

# ユースケース・モデル: JMS 管理インタフェース – 基本操作

表 13-1 ユースケース・モデル: JMS 管理インタフェース – 基本操作

ユースケース
13-5 ページの「 <a href="#">Point-to-Point: キュー・コネクション・ファクトリを作成する 2 つの方法</a> 」
13-6 ページの「 <a href="#">JDBC URL でのキュー・コネクション・ファクトリの取得</a> 」
13-8 ページの「 <a href="#">JDBC 接続パラメータでのキュー・コネクション・ファクトリの取得</a> 」
13-10 ページの「 <a href="#">パブリッシュ / サブスクライブ: トピック・コネクション・ファクトリを作成する 2 つの方法</a> 」
13-11 ページの「 <a href="#">JDBC URL でのトピック・コネクション・ファクトリの取得</a> 」
13-13 ページの「 <a href="#">JDBC 接続パラメータでのトピック・コネクション・ファクトリの取得</a> 」
13-15 ページの「 <a href="#">キュー・テーブルの作成</a> 」
13-17 ページの「 <a href="#">キュー・テーブルの作成 (キュー・テーブルのプロパティの指定)</a> 」
13-19 ページの「 <a href="#">キュー・テーブルの取得</a> 」
13-21 ページの「 <a href="#">宛先プロパティの指定</a> 」
13-23 ページの「 <a href="#">Point-to-Point: キューの作成</a> 」
13-25 ページの「 <a href="#">パブリッシュ / サブスクライブ: トピックの作成</a> 」
13-27 ページの「 <a href="#">システム権限の付与</a> 」
13-29 ページの「 <a href="#">システム権限の取消し</a> 」
13-31 ページの「 <a href="#">パブリッシュ / サブスクライブ: トピック権限の付与</a> 」
13-33 ページの「 <a href="#">パブリッシュ / サブスクライブ: トピック権限の取消し</a> 」
13-35 ページの「 <a href="#">Point-to-Point: キュー権限の付与</a> 」
13-37 ページの「 <a href="#">Point-to-Point: キュー権限の取消し</a> 」
13-39 ページの「 <a href="#">宛先の開始</a> 」
13-41 ページの「 <a href="#">宛先の停止</a> 」
13-43 ページの「 <a href="#">宛先の変更</a> 」
13-45 ページの「 <a href="#">宛先の削除</a> 」
13-47 ページの「 <a href="#">伝播のスケジュール</a> 」
13-49 ページの「 <a href="#">伝播スケジュールの使用可能化</a> 」

表 13-1 (続き) ユースケース・モデル: JMS 管理インタフェース – 基本操作

---

**ユースケース**

---

13-51 ページの「[伝播スケジュールの変更](#)」

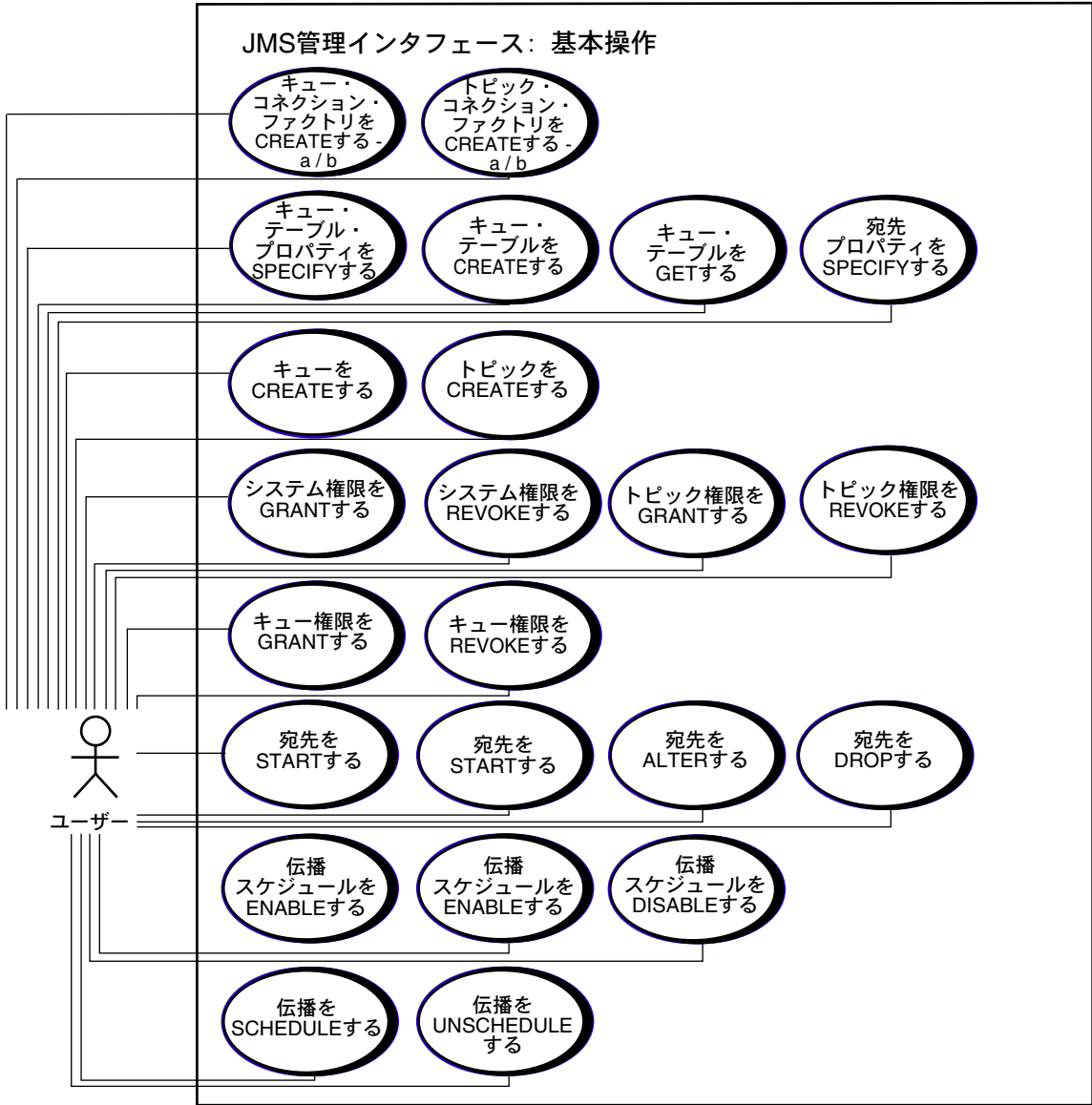
13-53 ページの「[伝播スケジュールの使用不可](#)」

13-55 ページの「[伝播スケジュールの解除](#)」

---

# ユースケース図 : JMS 管理インタフェース - 基本操作

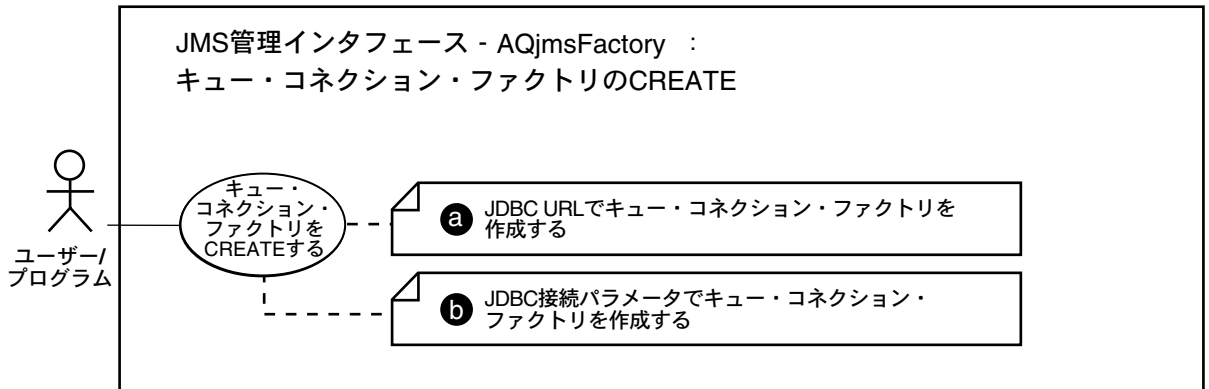
図 13-1 ユースケース図 : JMS 管理者インタフェース - 基本操作





## Point-to-Point: キュー・コネクション・ファクトリを作成する 2 つの方法

図 13-2 ユースケース図 : Point-to-Point: キュー・コネクション・ファクトリを作成する 2 つの方法

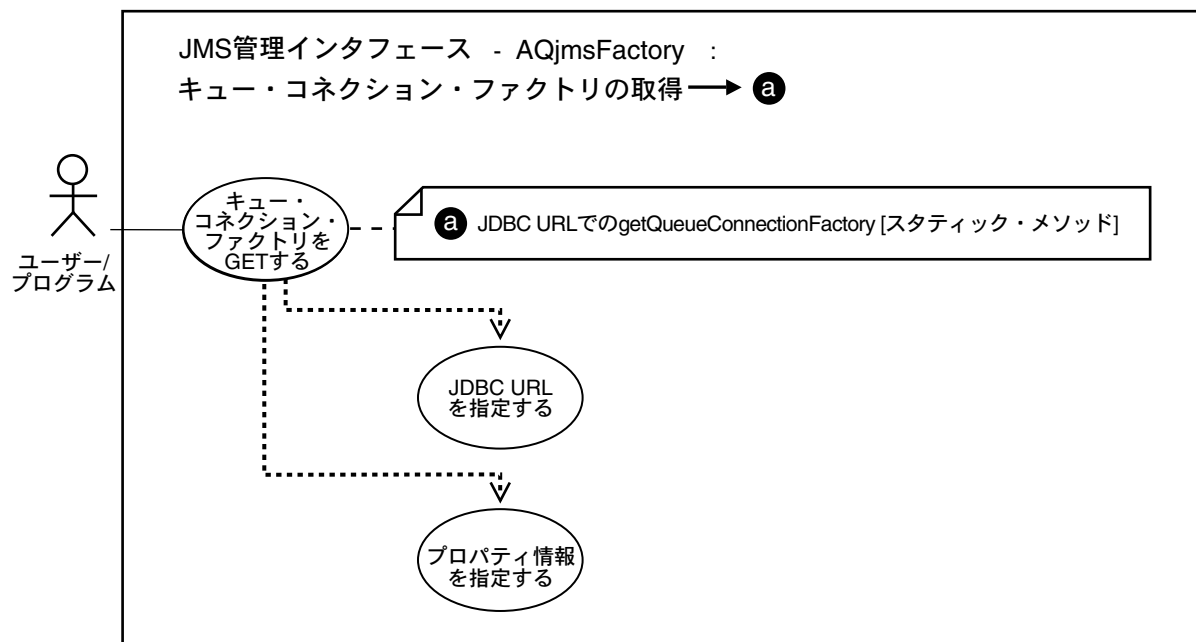


次の 2 つの方法でキュー・コネクション・ファクトリを作成できます。

- a. JDBC URL: 13-6 ページの「[JDBC URL でのキュー・コネクション・ファクトリの取得](#)」
- b. JDBC 接続パラメータ: 13-8 ページの「[JDBC 接続パラメータでのキュー・コネクション・ファクトリの取得](#)」

## JDBC URL でのキュー・コネクション・ファクトリの取得

図 13-3 ユースケース図：JDBC URL でのキュー・コネクション・ファクトリの取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。

### 用途

JDBC URL で、キュー・コネクション・ファクトリを取得します。

### 使用上の注意

`getQueueConnectionFactory` はスタティック・メソッドです。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsFactory」の `getQueueConnectionFactory` を参照してください。

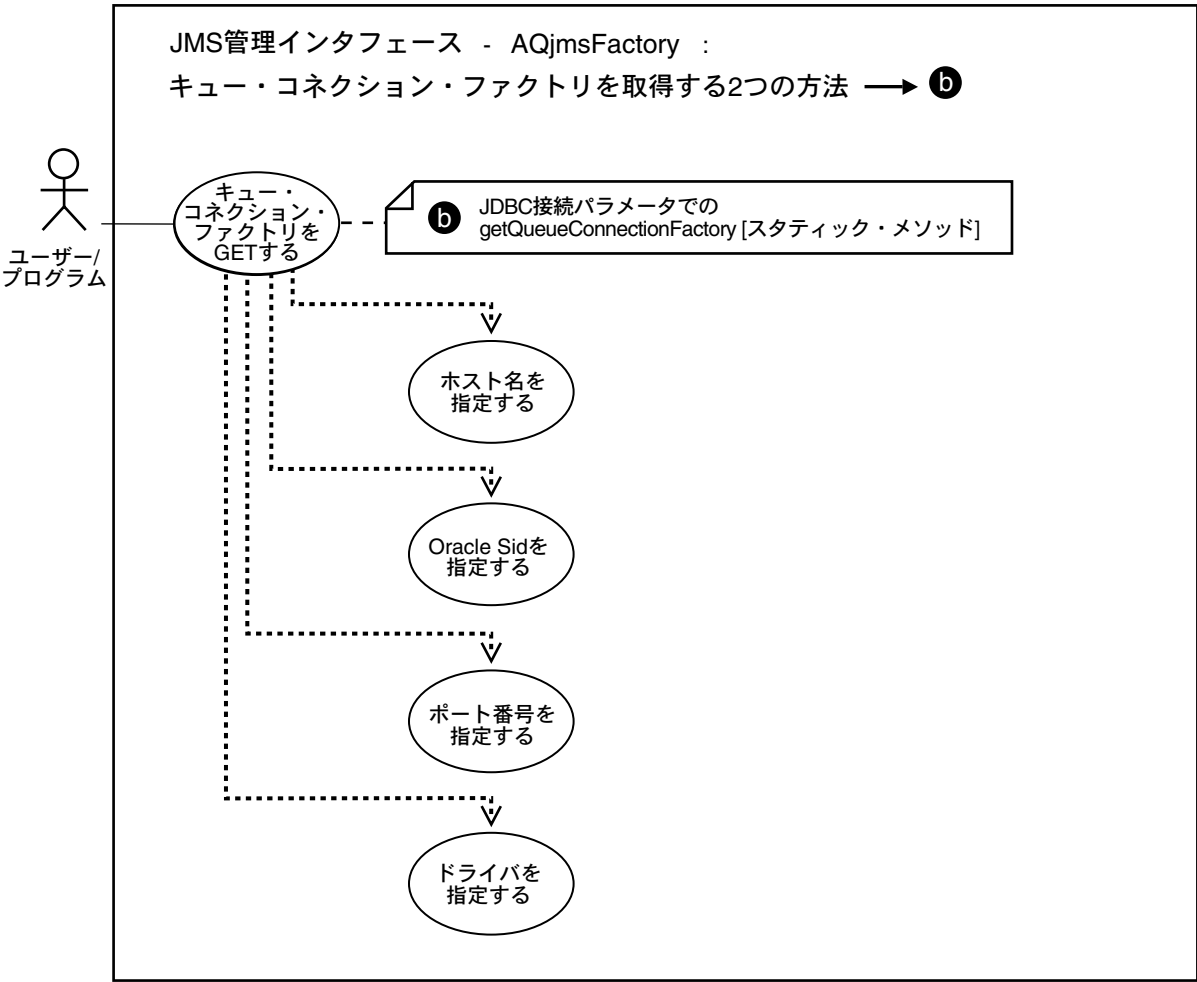
## 例

```
String      url          = "jdbc:oracle:oci8:internal/oracle"
Properties  info         = new Properties();
QueueConnectionFactory qc_fact;

info.put("internal_logon", "sysdba");
qc_fact = AQjmsFactory.getQueueConnectionFactory(url, info);
```

# JDBC 接続パラメータでのキュー・コネクション・ファクトリの取得

図 13-4 ユースケース図 : Point-to-Point: キュー・コネクション・ファクトリを作成する 2 つの方法



---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

JDBC 接続パラメータで、キュー・コネクション・ファクトリを取得します。

## 使用上の注意

`getQueueConnectionFactory` はスタティック・メソッドです。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ `oracle.jms`」の「`AQjmsFactory`」の `getQueueConnectionFactory` を参照してください。

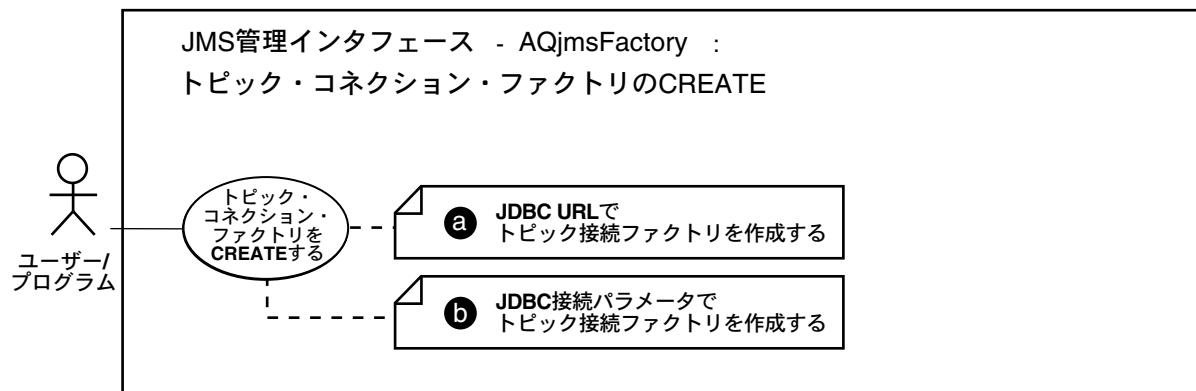
## 例

```
String      host      = "dlsun";
String      ora_sid   = "rdbms8i"
String      driver    = "thin";
int         port      = 5521;
QueueConnectionFactory qc_fact;

qc_fact = AQjmsFactory.getQueueConnectionFactory(host, ora_sid, port, driver);
```

## パブリッシュ / サブスクライブ : トピック・コネクション・ファクトリを作成する 2 つの方法

図 13-5 ユースケース図 : パブリッシュ / サブスクライブ : トピック・コネクション・ファクトリを作成する 2 つの方法



---

### 参照 :

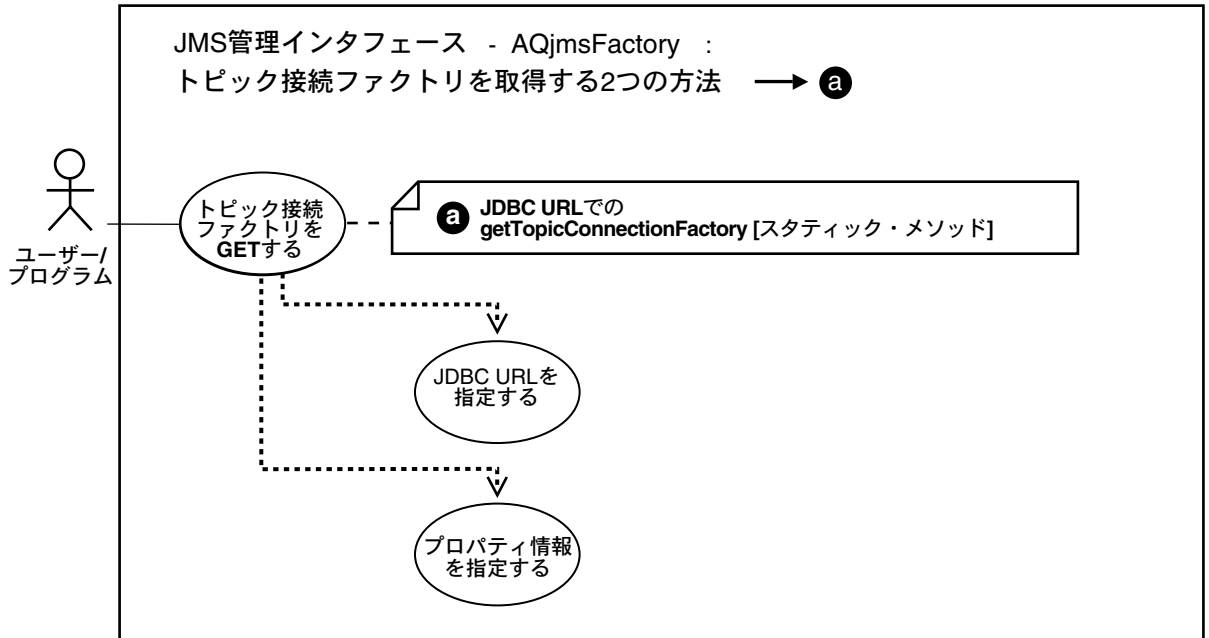
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 

次の 2 つの方法でトピック・コネクション・ファクトリを作成できます。

- JDBC URL: 13-11 ページの「[JDBC URL でのトピック・コネクション・ファクトリの取得](#)」
- JDBC 接続パラメータ : 13-13 ページの「[JDBC URL でのトピック・コネクション・ファクトリの取得](#)」

## JDBC URL でのトピック・コネクション・ファクトリの取得

図 13-6 ユースケース図：JDBC URL でのトピック・コネクション・ファクトリの取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

### 用途

JDBC URL で、トピック・コネクション・ファクトリを取得します。

### 使用上の注意

`getTopicConnectionFactory` はスタティック・メソッドです。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsFactory」の `getTopicConnectionFactory` を参照してください。

## 例

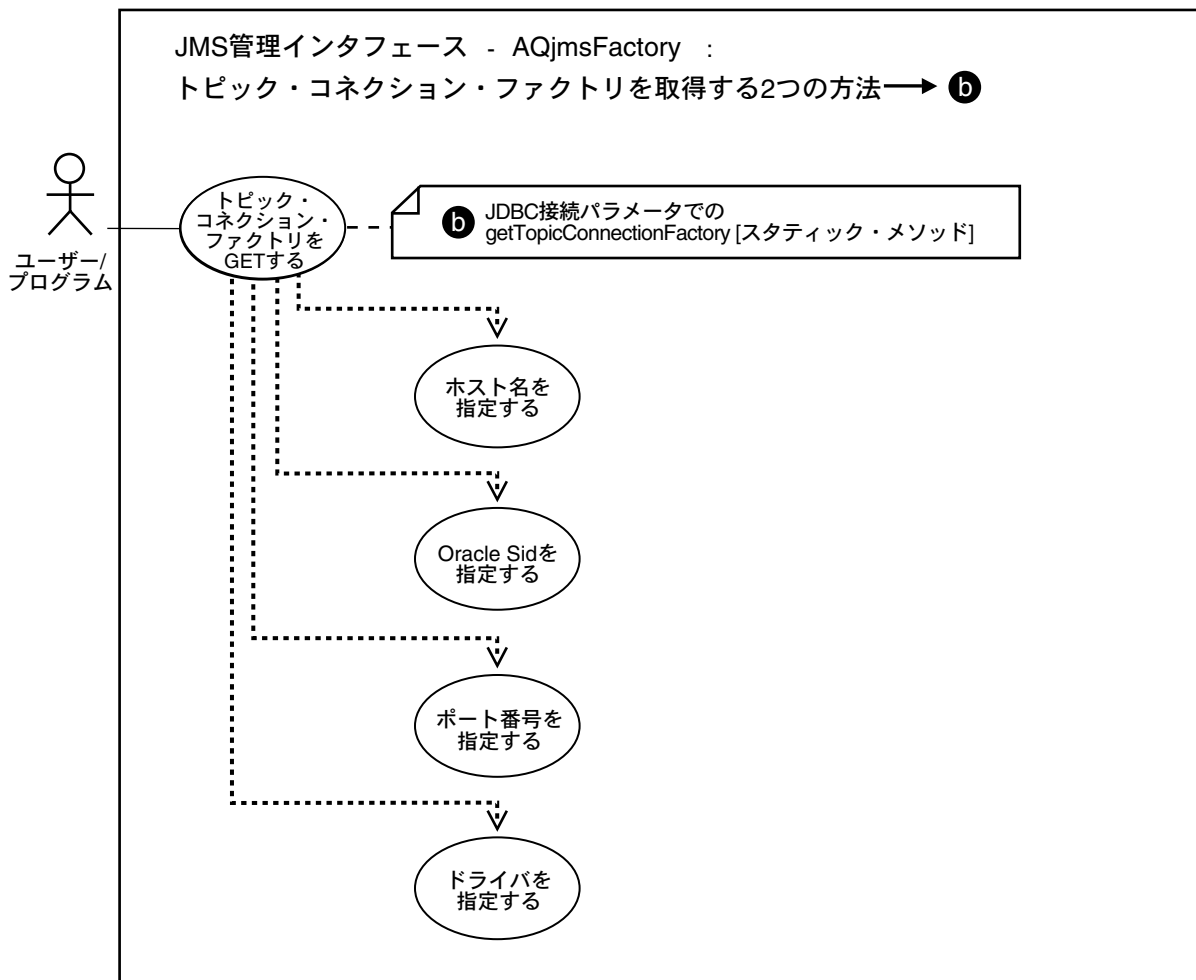
```
String      url          = "jdbc:oracle:oci8:internal/oracle"
Properties  info          = new Properties();
TopicConnectionFactory tc_fact;

info.put("internal_logon", "sysdba");
tc_fact = AQjmsFactory.getTopicConnectionFactory(url, info);
```



## JDBC 接続パラメータでのトピック・コネクション・ファクトリの取得

図 13-7 ユースケース図：JDBC 接続パラメータでのトピック・コネクション・ファクトリの取得



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 使用上の注意

GetTopicConnectionFactory はスタティック・メソッドです。

## 用途

JDBC 接続パラメータで、トピック・コネクション・ファクトリを取得します。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsFactory」の getTopicConnectionFactory を参照してください。

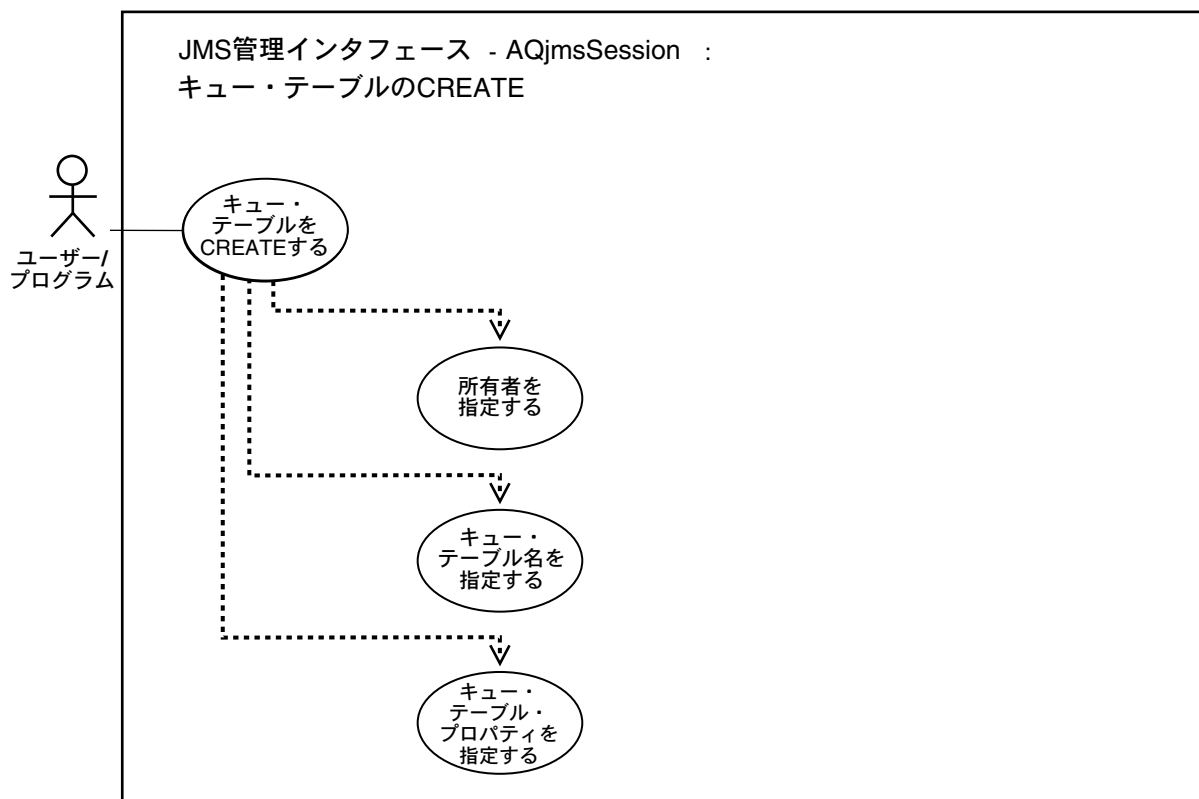
## 例

```
String      host      = "dlsun";
String      ora_sid   = "rdbms8i"
String      driver    = "thin";
int         port      = 5521;
TopicConnectionFactory tc_fact;

tc_fact = AQjmsFactory.getTopicConnectionFactory(host, ora_sid, port, driver);
```

## キュー・テーブルの作成

図 13-8 ユースケース図：キュー・テーブルの作成



---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

キュー・テーブルを作成します。

## 使用上の注意

AQ オブジェクト型では、CLOB、BLOB、BFILE オブジェクトが有効な属性です。ただし、Oracle8i では、CLOB および BLOB のみ AQ 伝播を使用して伝播できます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createQueueTable を参照してください。

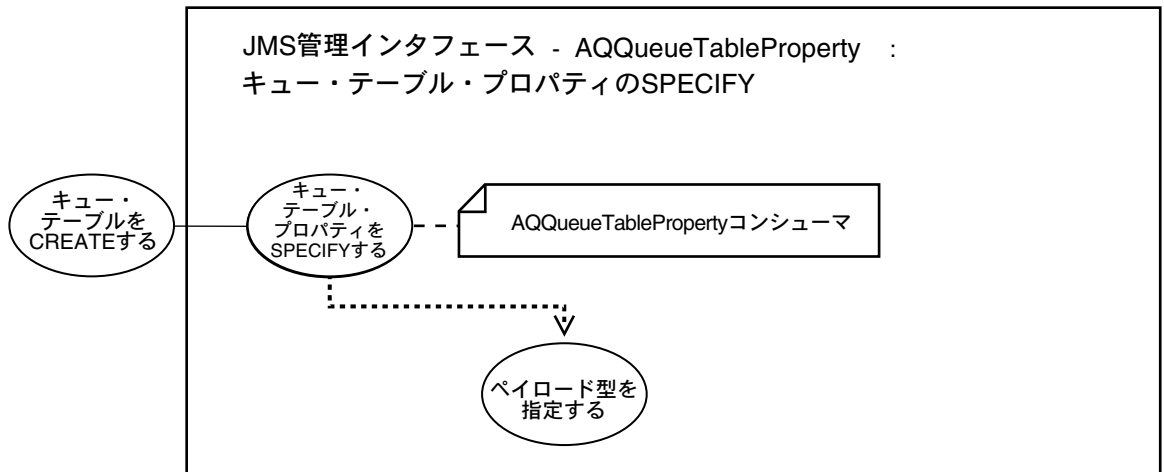
## 例

```
QueueSession          q_sess    = null;
AQQueueTable           q_table   = null;
AQQueueTableProperty   qt_prop   = null;

qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_BYTES_MESSAGE");
q_table = ((AQjmsSession)q_sess).createQueueTable("boluser",
                                                    "bol_ship_queue_table",
                                                    qt_prop);
```

## キュー・テーブルの作成（キュー・テーブルのプロパティの指定）

図 13-9 ユースケース図：キュー・テーブルのプロパティの指定



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

キュー・テーブルのプロパティを指定します。

## 使用上の注意

ありません。

## 構文

Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 1 章「パッケージ oracle.AQ」の「AQQueueTableProperty」を参照してください。

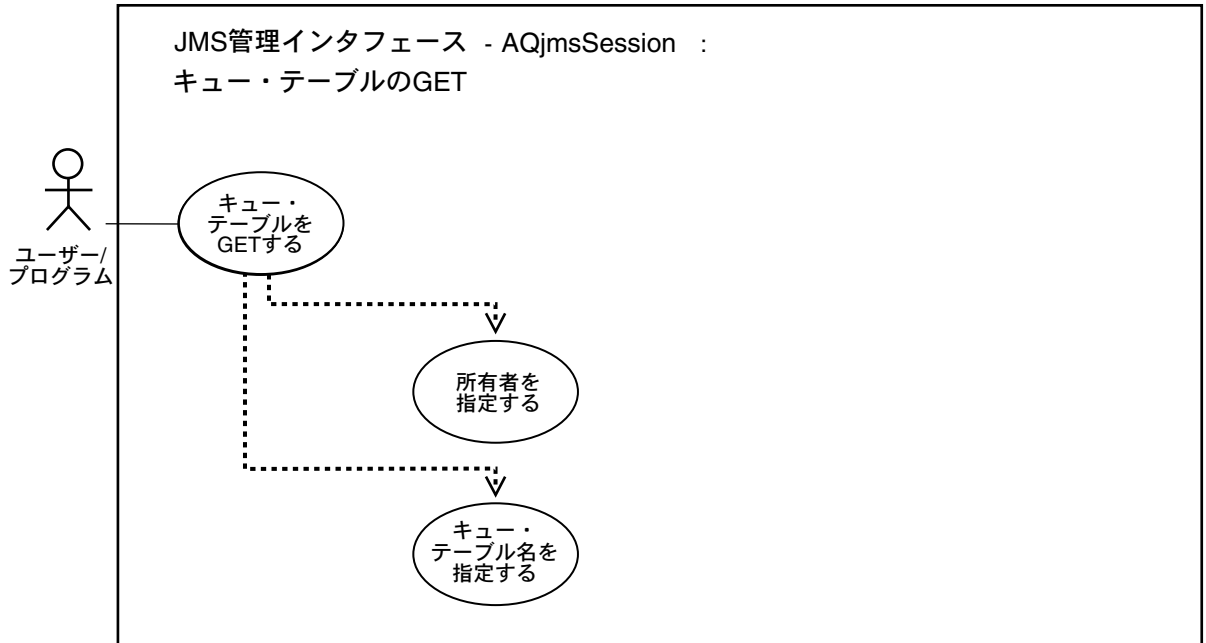
## 例

```
QueueSession          q_sess    = null;
AQQueueTable           q_table   = null;
AQQueueTableProperty   qt_prop   = null;

qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_BYTES_MESSAGE");
q_table = ((AQJmsSession)q_sess).createQueueTable("boluser",
                                                    "bol_ship_queue_table",
                                                    qt_prop);
```

## キュー・テーブルの取得

図 13-10 ユースケース図：キュー・テーブルの取得



---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

キュー・テーブルを取得します。

## 使用上の注意

接続をオープンしたコール側がキュー・テーブルの所有者でない場合、コール側には、キュー・テーブル内のキュー / トピックに対する AQ のエンキュー / デキュー権限が必要です。この権限がない場合、キュー・テーブルは取得できません。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQJmsSession」の `getQueueTable` を参照してください。

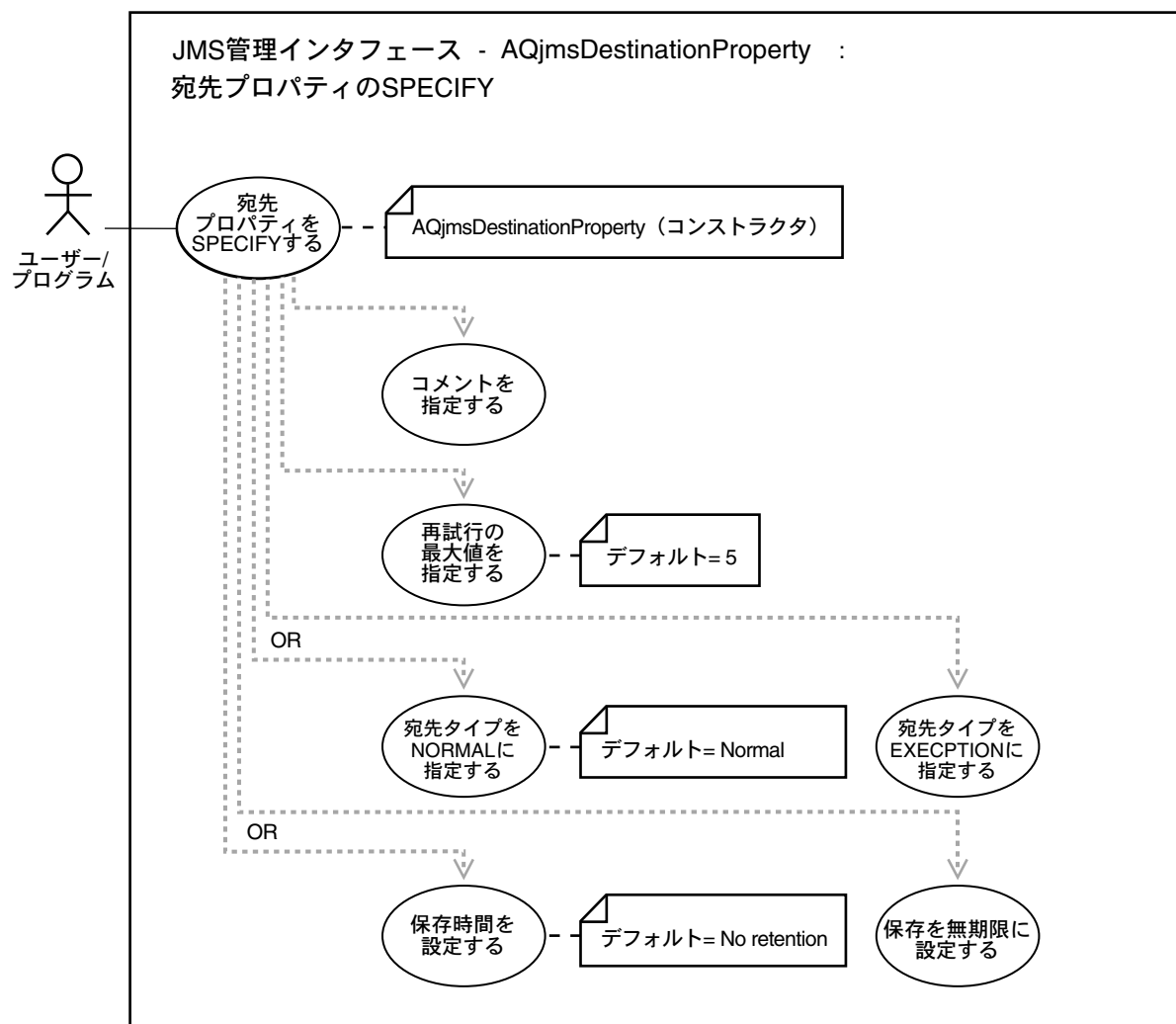
## 例

```
QueueSession          q_sess;  
AQQueueTable          q_table;  
  
q_table = ((AQJmsSession)q_sess).getQueueTable("boluser",  
"bol_ship_queue_table");
```



## 宛先プロパティの指定

図 13-11 ユースケース図：宛先プロパティの指定



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

宛先プロパティを指定します。

## 使用上の注意

ありません。

## 構文

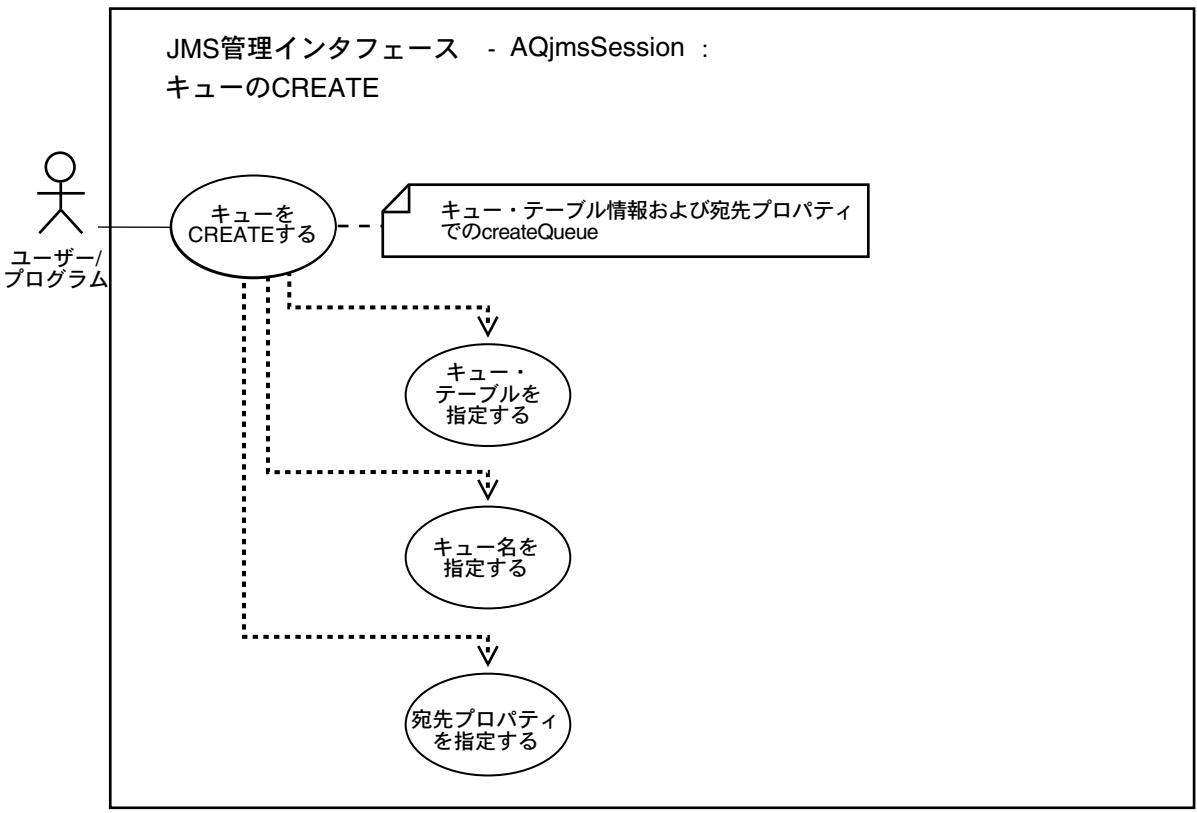
Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsDestinationProperty」を参照してください。

## 例

今回のリリースでは、例は記載していません。

# Point-to-Point: キューの作成

図 13-12 ユースケース図 : Point-to-Point: キューの作成



- 参照：**
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

指定されたキュー・テーブル内にキューを作成します。

## 使用上の注意

キューが作成されるキュー・テーブルは、単一コンシューマ・キュー・テーブルである必要があります。

## 構文

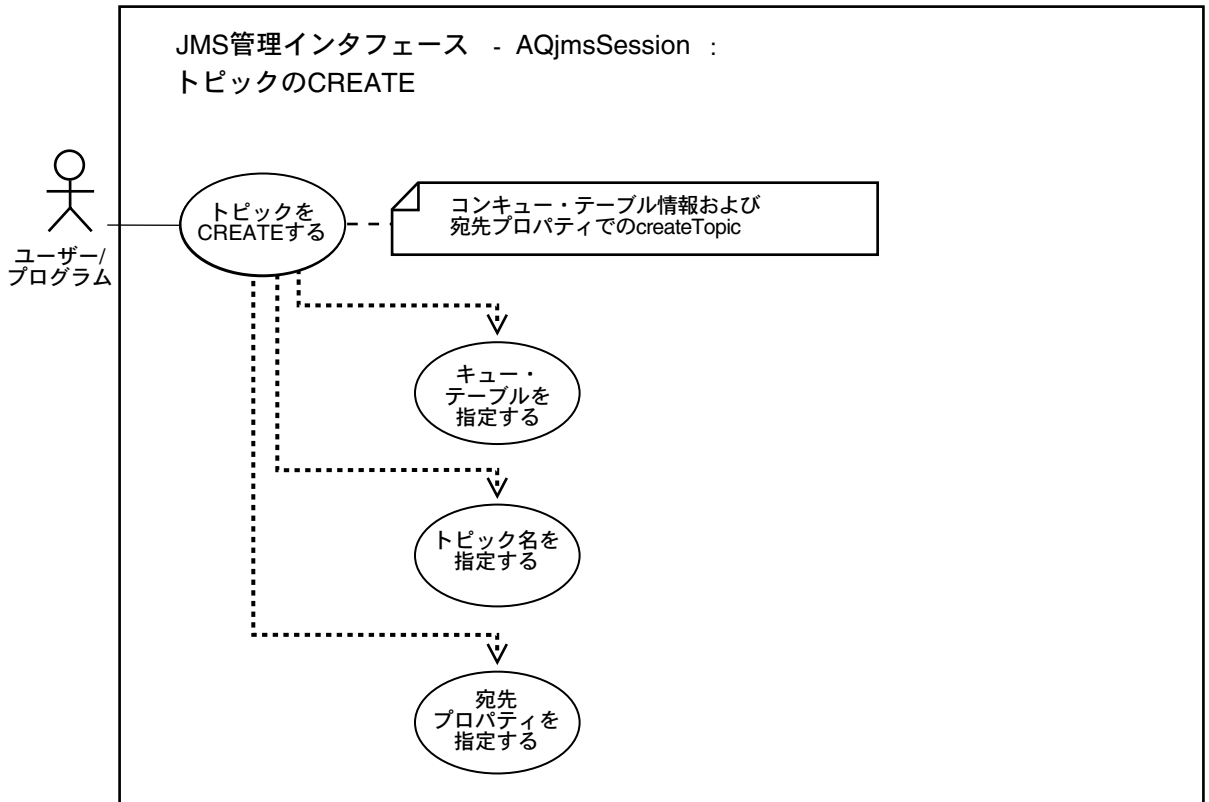
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の `createQueue` を参照してください。

## 例

```
QueueSession          q_sess;  
AQQueueTable          q_table;  
AQjmsDestinationProperty dest_prop;  
Queue                 queue;  
  
queue = ((AQjmsSession)q_sess).createQueue(q_table, "jms_q1", dest_prop);
```

# パブリッシュ / サブスクライブ : トピックの作成

図 13-13 ユースケース図 : パブリッシュ / サブスクライブ : トピックの作成



## 参照:

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

パブリッシュ / サブスクライブ・モデルにおいてトピックを作成します。

## 使用上の注意

ありません。

## 構文

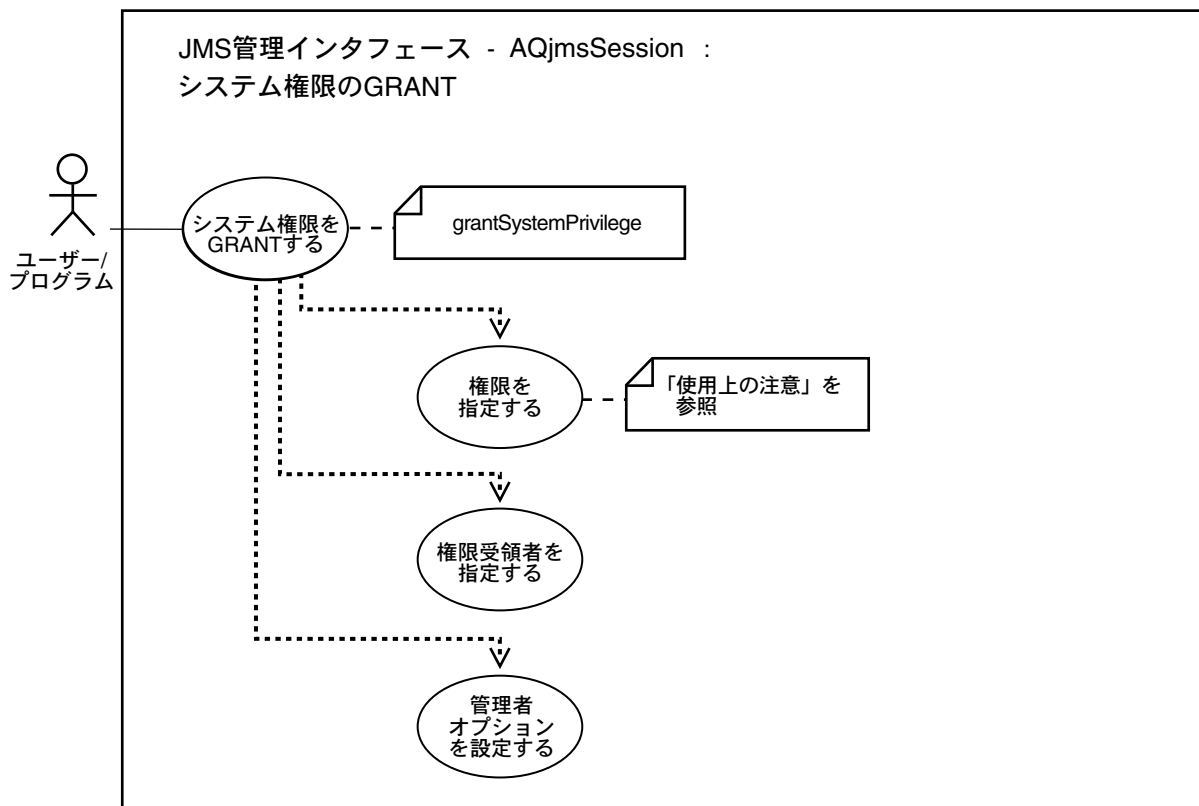
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createTopic を参照してください。

## 例

```
TopicSession          t_sess;  
AQQueueTable          q_table;  
AQjmsDestinationProperty dest_prop;  
Topic                 topic;  
  
topic = ((AQjmsSessa)t_sess).createTopic(q_table, "jms_t1", dest_prop);
```

## システム権限の付与

図 13-14 ユースケース図：システム権限の付与

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

ユーザー / ロールに AQ システム権限を付与します。

## 使用上の注意

最初は、SYS および SYSTEM のみがこのプロシージャを正常に使用できます。

この権限とは、ENQUEUE\_ANY、DEQUEUE\_ANY および MANAGE\_ANY です。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の grantSystemPrivilege を参照してください。

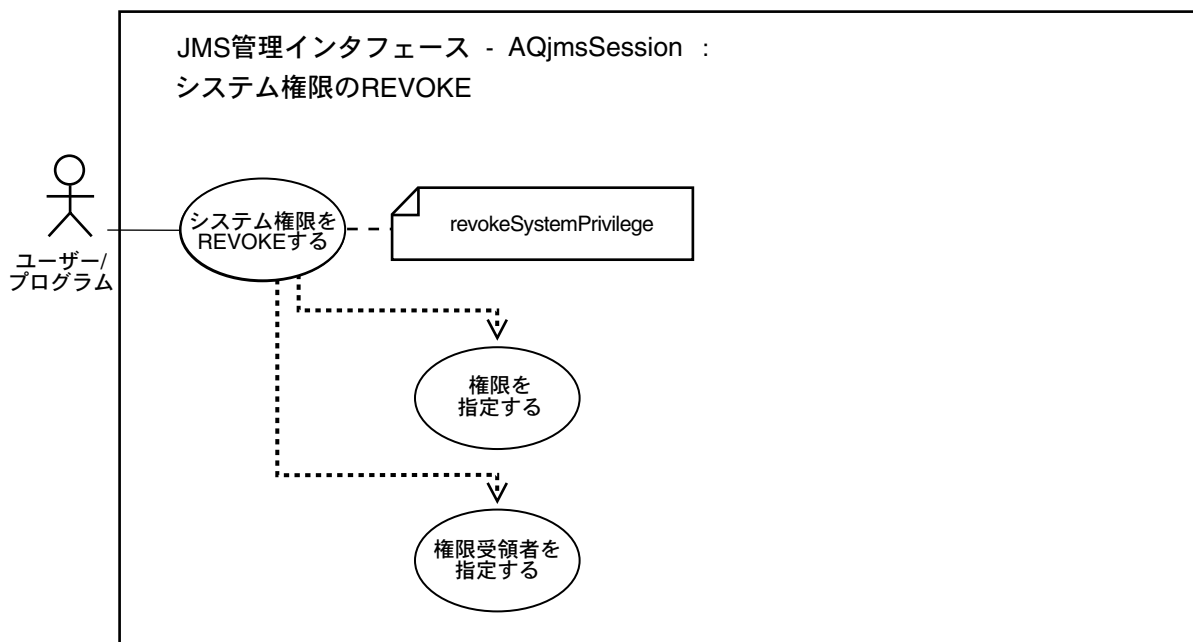
## 例

```
TopicSession          t_sess;  
  
( (AQjmsSession)t_sess ).grantSystemPrivilege("ENQUEUE_ANY", "scott", false);
```



## システム権限の取消し

図 13-15 ユースケース図：システム権限の取消し



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

ユーザー / ロールの AQ システム権限を取り消します。

## 使用上の注意

ここでの権限とは、ENQUEUE\_ANY、DEQUEUE\_ANY および MANAGE\_ANY です。

### 構文

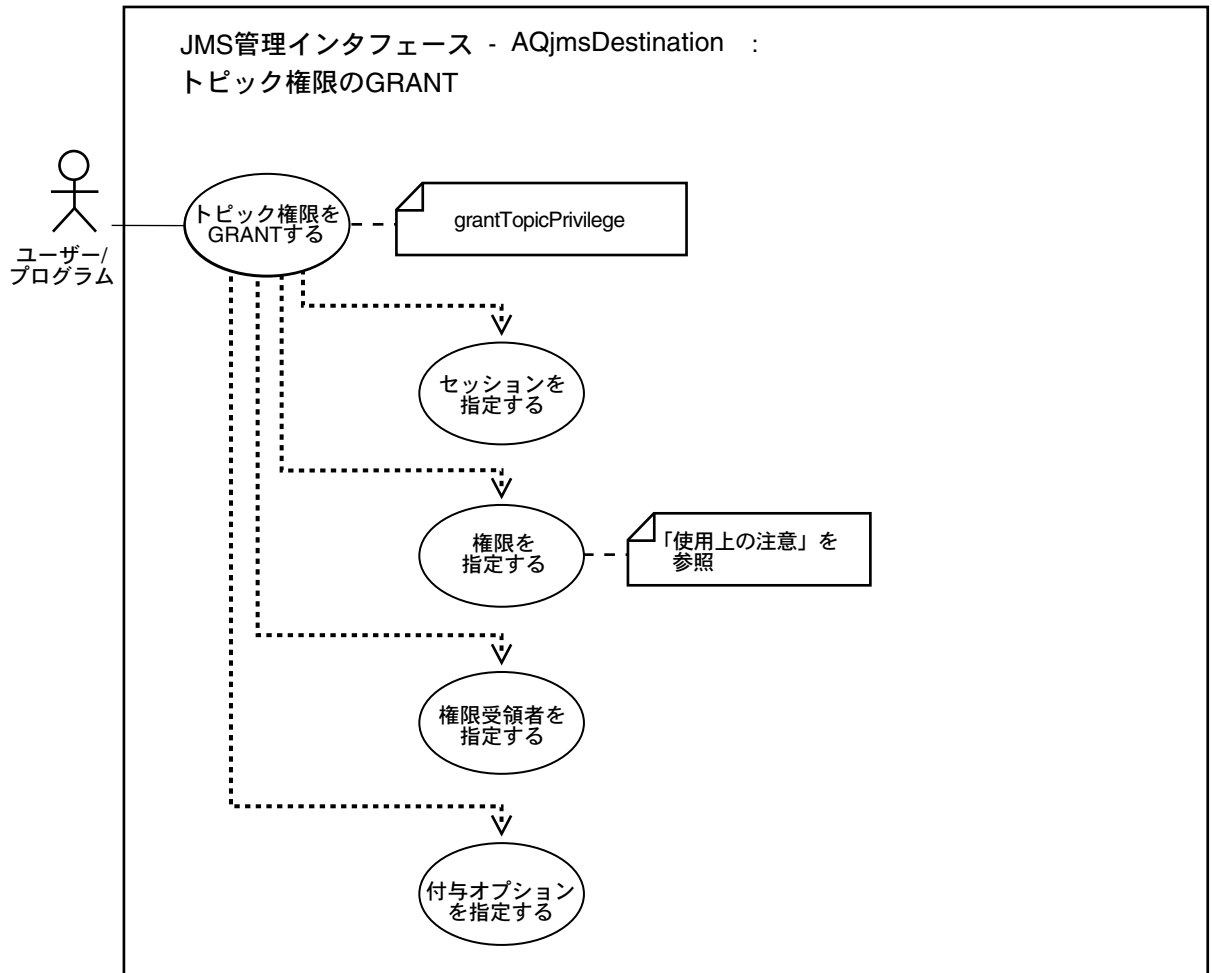
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の `revokeSystemPrivilege` を参照してください。

### 例

```
TopicSession          t_sess;  
  
((AQjmsSession)t_sess).revokeSystemPrivilege("ENQUEUE_ANY", "scott");
```

## パブリッシュ / サブスクライブ : トピック権限の付与

図 13-16 ユースケース図 : パブリッシュ / サブスクライブ : トピック権限の付与



---

---

**参照 :**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

パブリッシュ / サブスクライブ・モデルにトピック権限を付与します。

## 使用上の注意

この権限とは、ENQUEUE、DEQUEUE および ALL です。ALL は両方を意味します。最初は、キュー・テーブル所有者のみが、このプロシージャを使用してトピックに対する権限を付与できます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsDestination」の grantTopicPrivilege を参照してください。

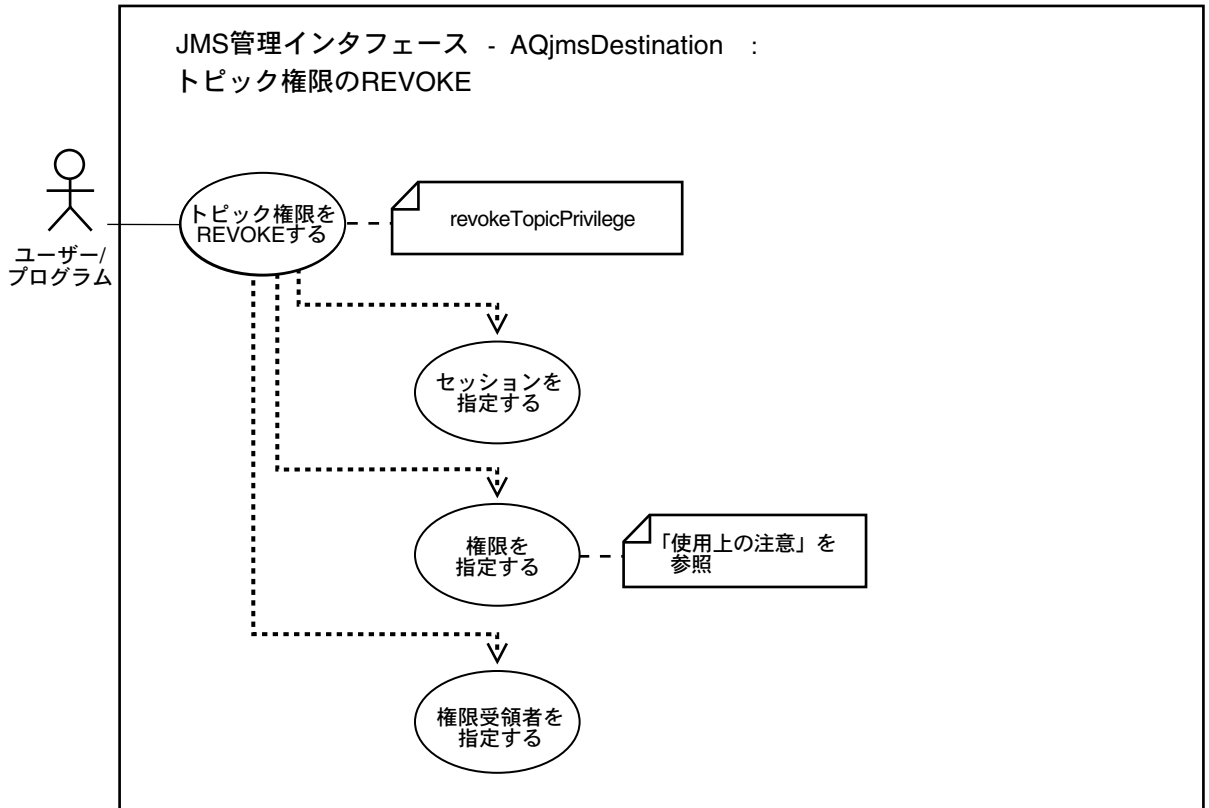
## 例

```
TopicSession      t_sess;
Topic              topic;

((AQjmsDestination)topic).grantTopicPrivilege(t_sess, "ENQUEUE", "scott", false);
```

# パブリッシュ / サブスクライブ : トピック権限の取消し

図 13-17 ユースケース図 : パブリッシュ / サブスクライブ : トピック権限の取消し

**参照 :**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

パブリッシュ / サブスクライブ・モデルにおいてトピック権限を取り消します。

## 使用上の注意

この権限とは、ENQUEUE、DEQUEUE および ALL です。ALL は両方を意味します。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の revokeTopicPrivilege を参照してください。

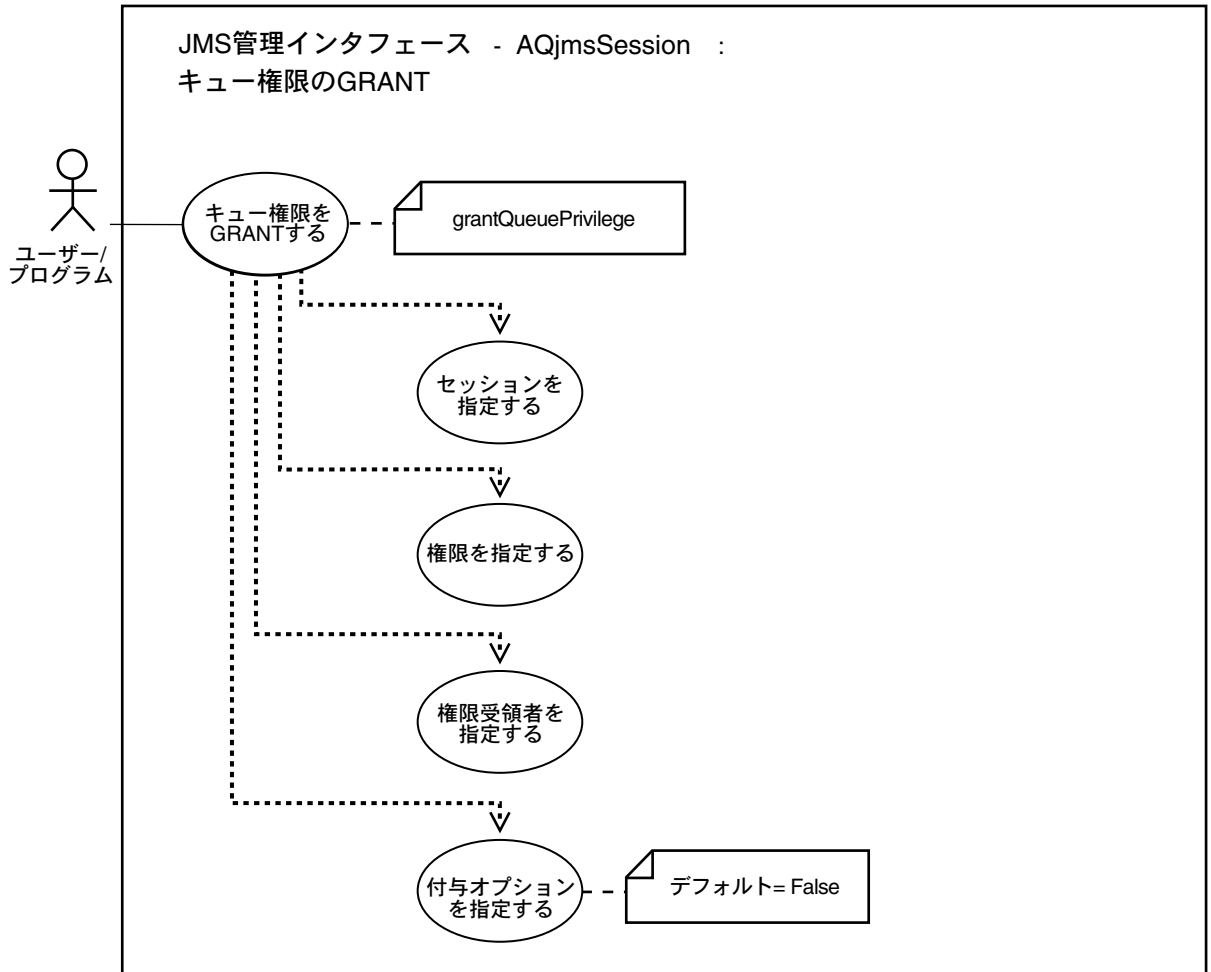
## 例

```
TopicSession      t_sess;
Topic              topic;

((AQjmsDestination)topic).revokeTopicPrivilege(t_sess, "ENQUEUE", "scott");
```

## Point-to-Point: キュー権限の付与

図 13-18 ユースケース図 : Point-to-Point: キュー権限の付与



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

Point-to-Point モデルでキュー権限を付与します。

## 使用上の注意

この権限とは、ENQUEUE、DEQUEUE および ALL です。ALL は両方を意味します。最初は、キュー・テーブル所有者のみが、このプロシージャを使用してキューに対する権限を付与できます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsDestination」の `grantQueuePrivilege` を参照してください。

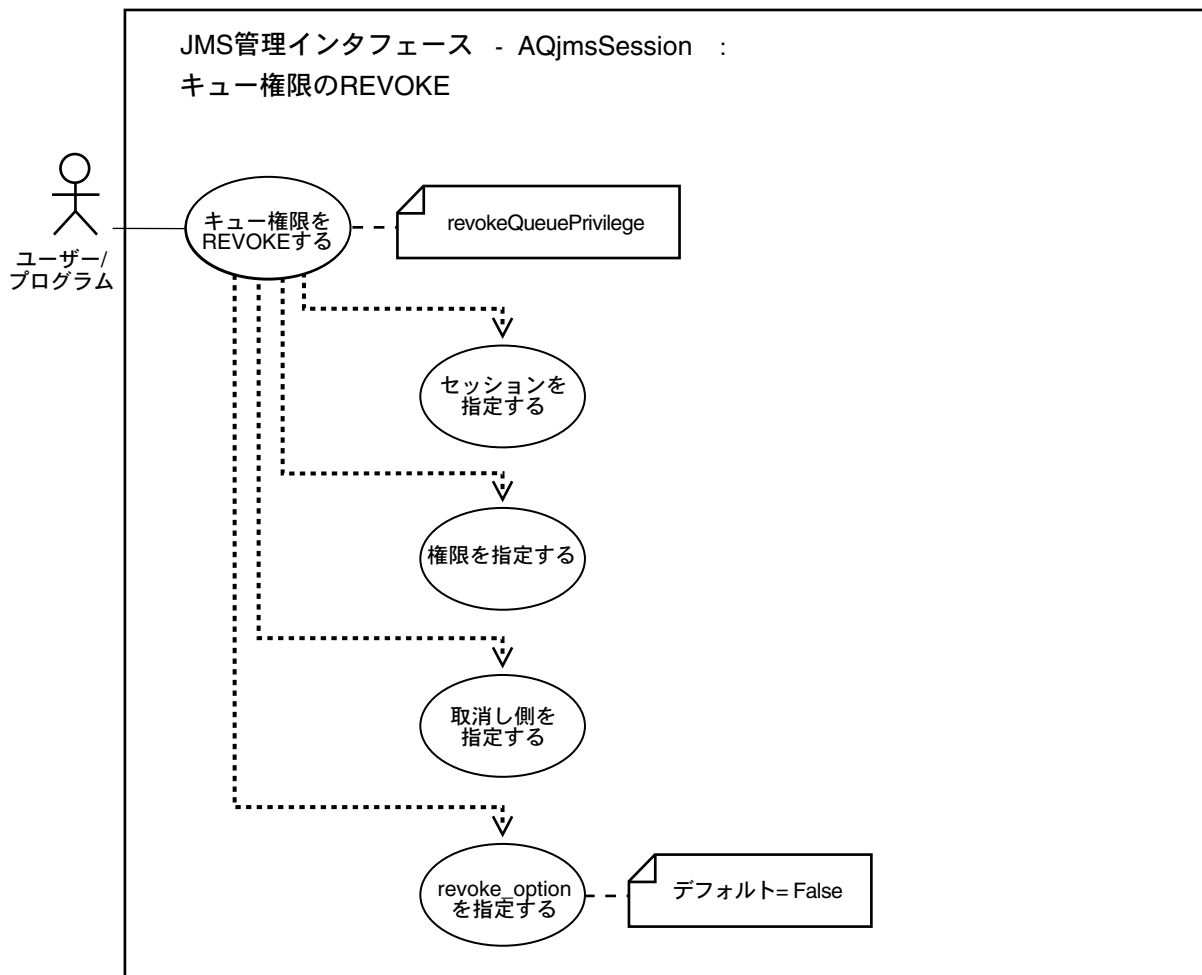
## 例

```
QueueSession      q_sess;  
Queue              queue;  
  
(AQjmsDestination)queue).grantQueuePrivilege(q_sess, "ENQUEUE", "scott", false);
```



## Point-to-Point: キュー権限の取消し

図 13-19 ユースケース図 : Point-to-Point: キュー権限の取消し



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

Point-to-Point モデルでキュー権限を取り消します。

## 使用上の注意

この権限とは、ENQUEUE、DEQUEUE および ALL です。ALL は両方を意味します。権限を取り消すユーザーは、取消しの対象となる権限の付与者である必要があります。GRANT オプションによって伝播された権限は、伝播させた付与者の権限が取り消されたときに取り消されます。

## 構文

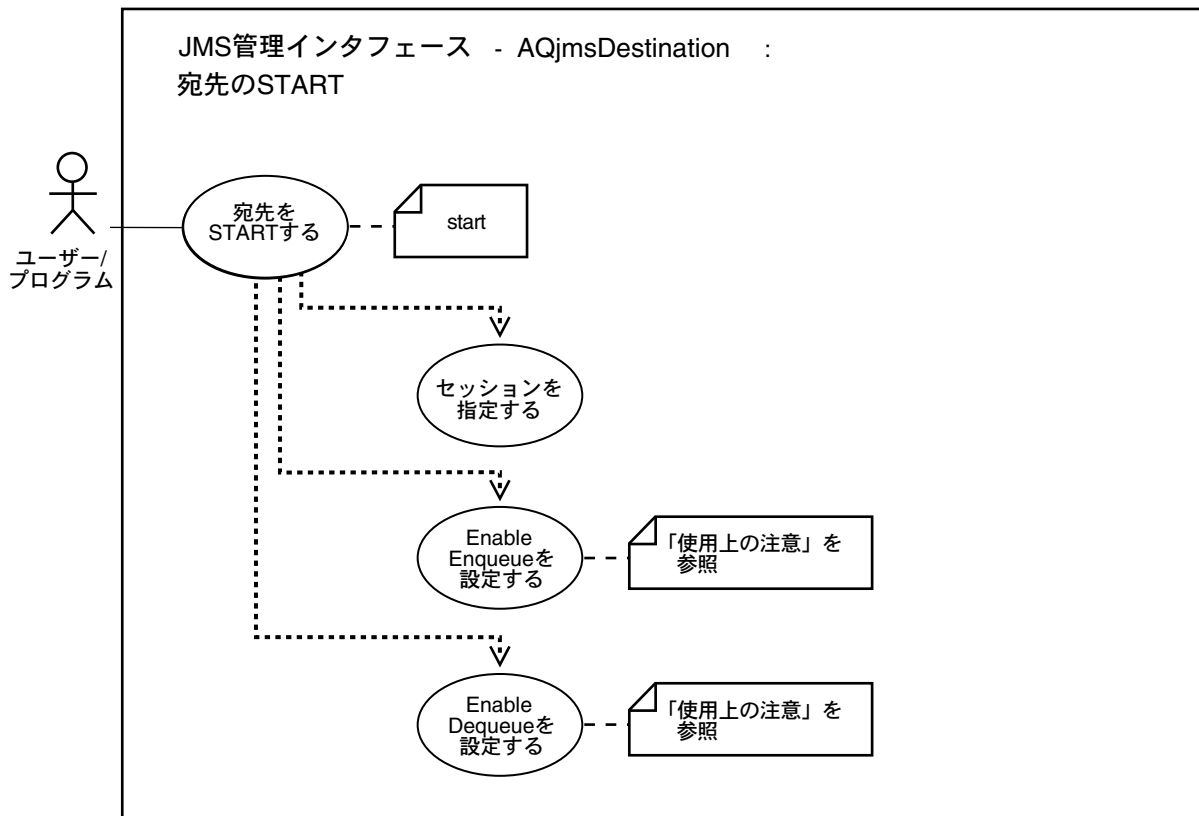
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsDestination」の `revokeQueuePrivilege` を参照してください。

## 例

```
QueueSession      q_sess;  
Queue              queue;  
  
( (AQjmsDestination)queue ).revokeQueuePrivilege (q_sess, "ENQUEUE", "scott");
```

## 宛先の開始

図 13-20 ユースケース図：宛先の開始



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

宛先を開始します。

## 使用上の注意

宛先の作成後、管理者は、start メソッドを使用して宛先を使用可能にする必要があります。Enable Enqueue が TRUE に設定されている場合、宛先はエンキュー可能になります。Enable Enqueue が FALSE に設定されている場合、宛先はエンキュー禁止になります。同様に、Enable Dequeue が TRUE に設定されている場合、宛先はデキュー可能になります。Enable Dequeue が FALSE に設定されている場合、宛先はデキュー禁止になります。

## 構文

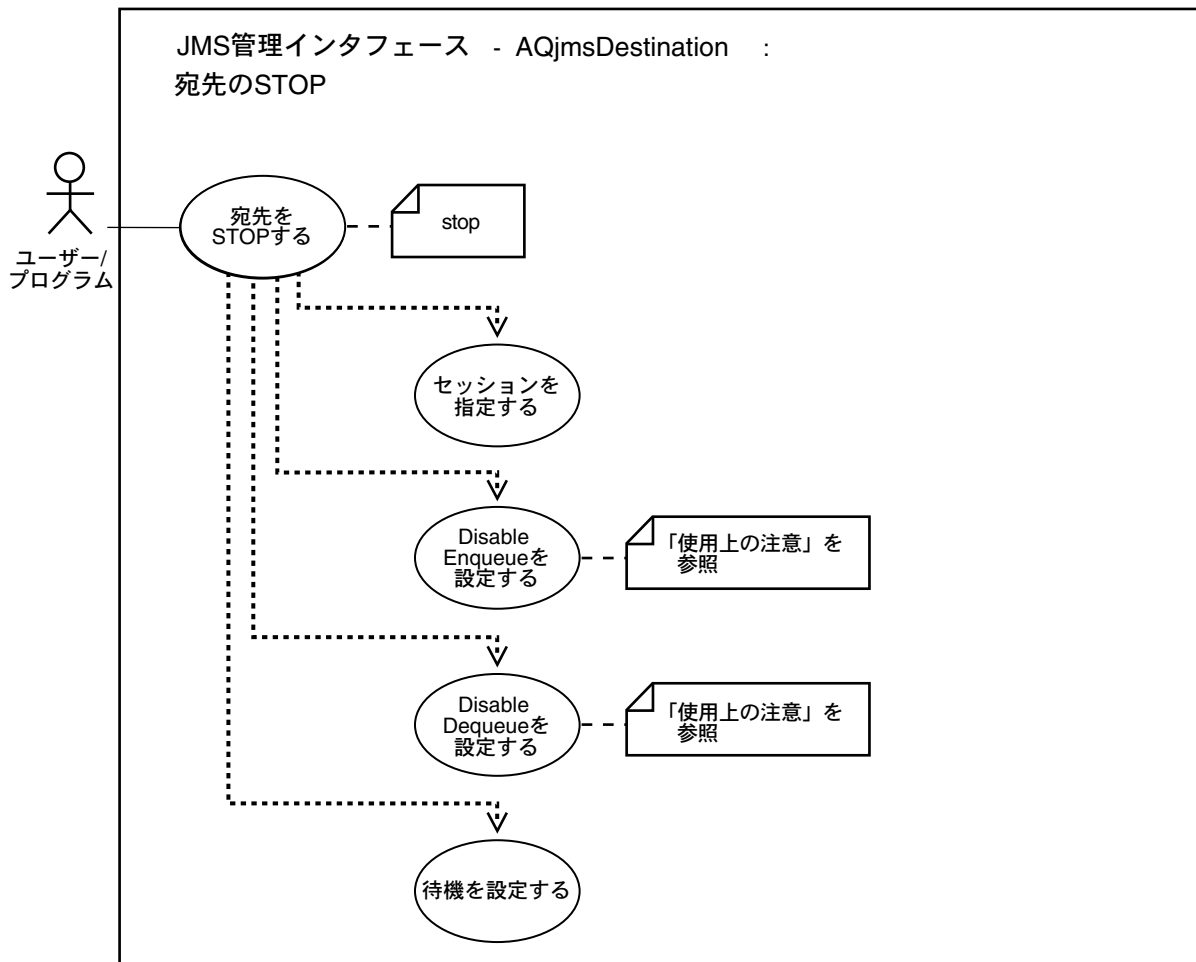
Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の start を参照してください。

## 例

```
TopicSession t_sess;  
QueueSession q_sess;  
Topic        topic;  
Queue        queue;  
  
(AQjmsDestination)topic.start(t_sess, true, true);  
(AQjmsDestination)queue.start(q_sess, true, true);
```

## 宛先の停止

図 13-21 ユースケース図：宛先の停止



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

宛先を停止します。

## 使用上の注意

Disable Dequeue が TRUE に設定されている場合、宛先はデキュー禁止になります。Disable Dequeue が FALSE に設定されている場合、現在の設定は変更されません。同様に、Disable Enqueue が TRUE に設定されている場合、宛先はエンキューに使用禁止になります。Disable Enqueue が FALSE に設定されている場合、現在の設定は変更されません。

## 構文

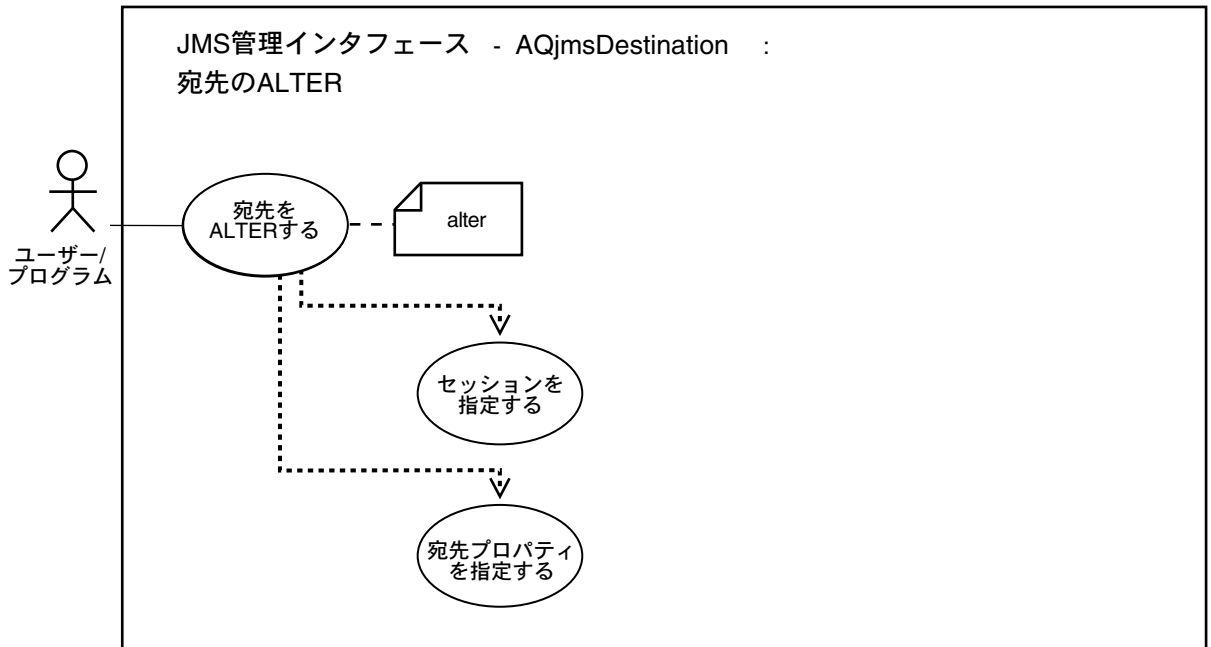
Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsDestination」の stop を参照してください。

## 例

```
TopicSession t_sess;  
Topic        topic;  
  
(AQjmsDestination)topic).stop(t_sess, true, false);
```

## 宛先の変更

図 13-22 ユースケース図：宛先の変更



---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

宛先を変更します。

## 使用上の注意

ありません。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の alter を参照してください。

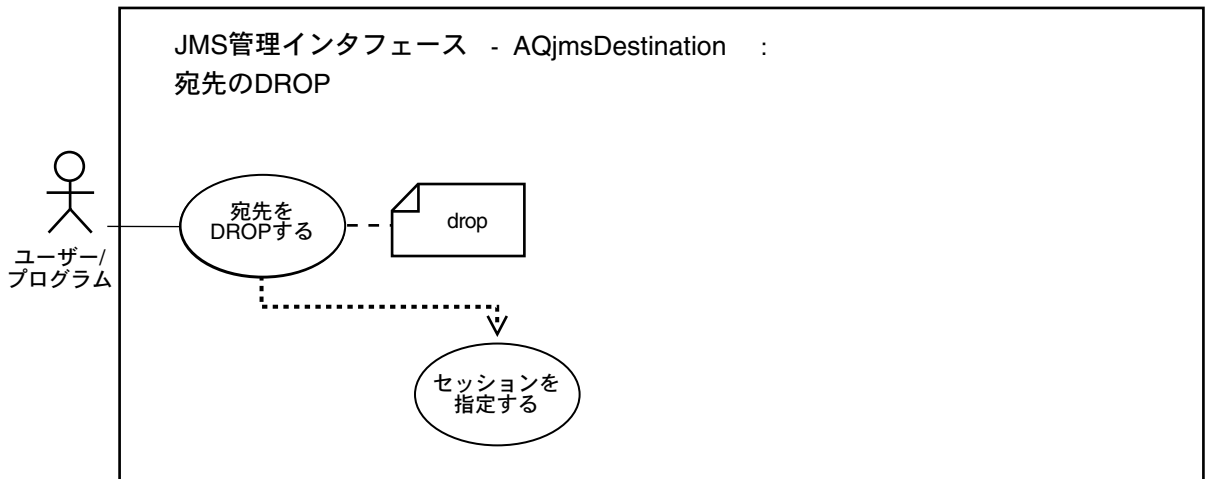
## 例

```
QueueSession q_sess;  
Queue         queue;  
TopicSession t_sess;  
Topic         topic;  
  
AQjmsDestinationProperty dest_prop1, dest_prop2;  
  
( (AQjmsDestination)queue ).alter (dest_prop1);  
( (AQjmsDestination)topic ).alter (dest_prop2);
```



## 宛先の削除

図 13-23 ユースケース図：宛先の削除



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

宛先を削除します。

## 使用上の注意

ありません。

## 構文

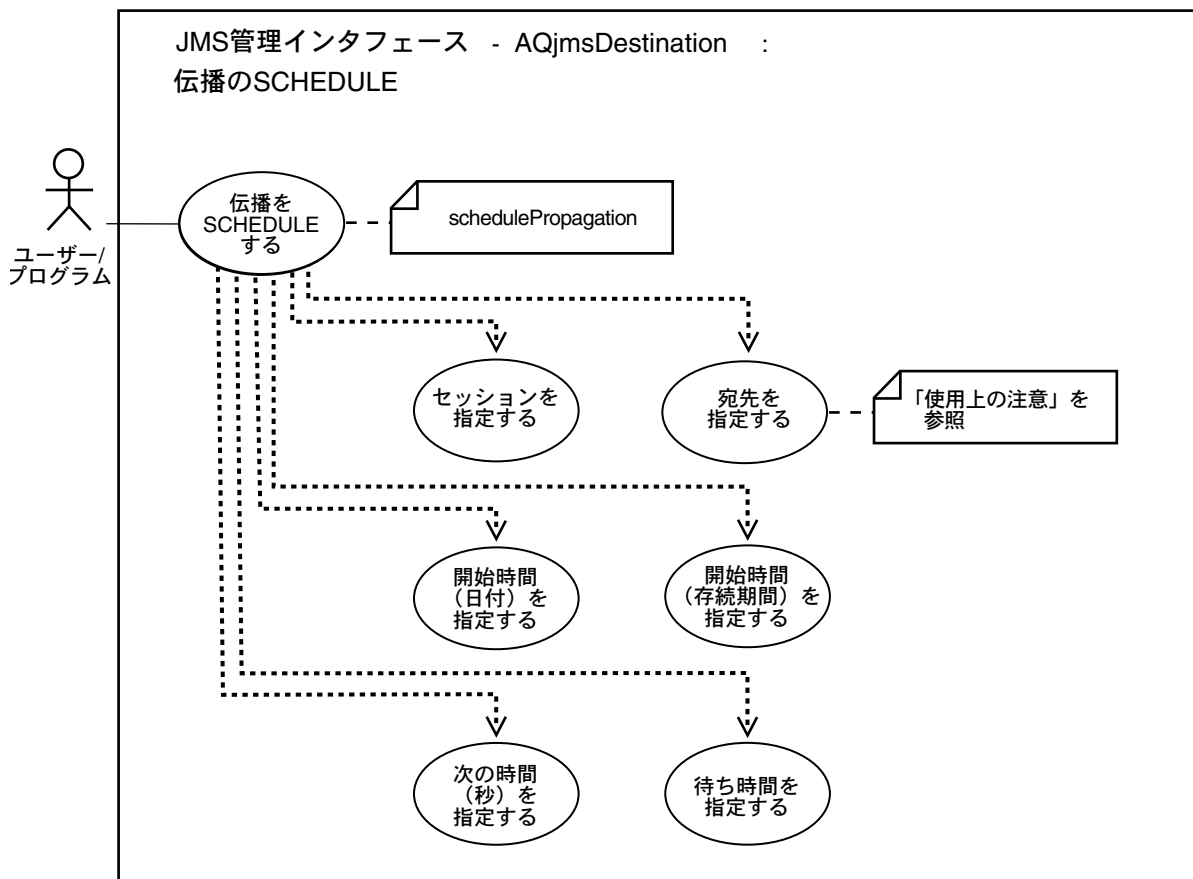
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の drop を参照してください。

## 例

```
QueueSession q_sess;  
Queue         queue;  
TopicSession t_sess;  
Topic         topic;  
  
( (AQjmsDestination) queue ).drop (q_sess);  
( (AQjmsDestination) topic ).drop (t_sess);
```

## 伝播のスケジュール

図 13-24 ユースケース図：伝播のスケジュール

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

### 用途

伝播をスケジュールします。

### 使用上の注意

NULL の宛先を指定すると、同じデータベース内の他のトピックにメッセージを伝播できます。メッセージ受信者が、同一または異なるキュー内の同じ宛先に複数ある場合、メッセージはすべて受信者に同時に伝播されます。

### 構文

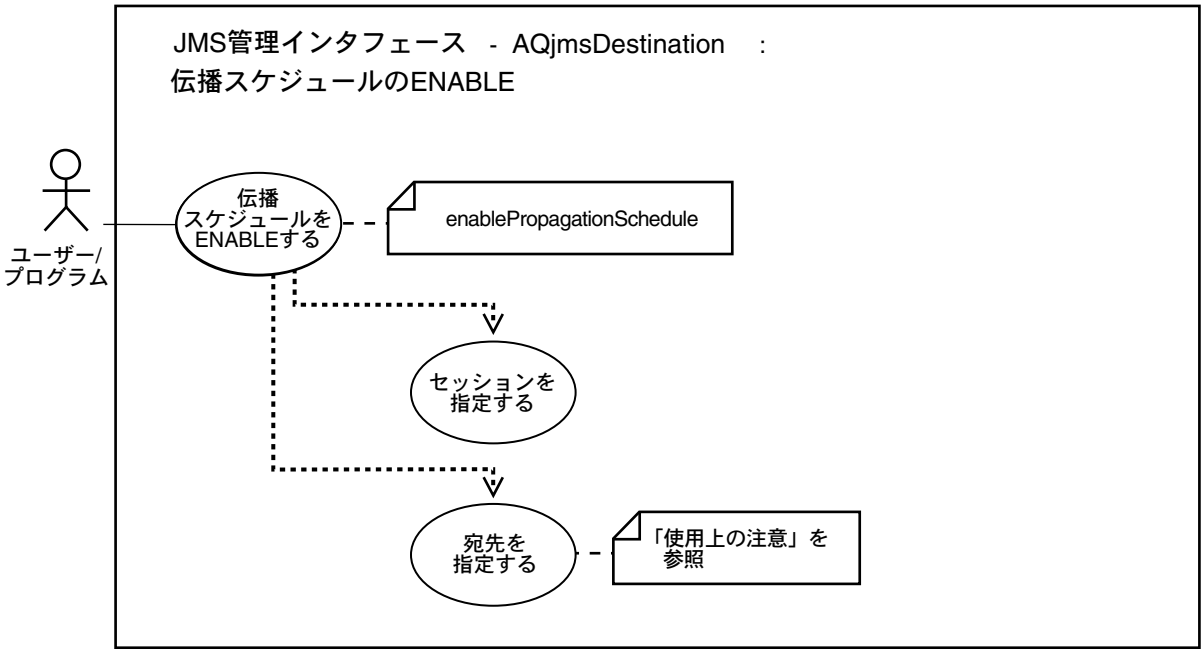
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の `schedulePropagation` を参照してください。

### 例

```
TopicSession t_sess;  
Topic        topic;  
  
((AQjmsDestination)topic).schedulePropagation(t_sess, null, null, null, null, new  
Double(0));
```

# 伝播スケジュールの使用可能化

図 13-25 ユースケース図：伝播スケジュールの使用可能化



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

伝播スケジュールを使用可能にします。

## 使用上の注意

NULL の宛先は、ローカル・データベースに伝播されることを示します。

## 構文

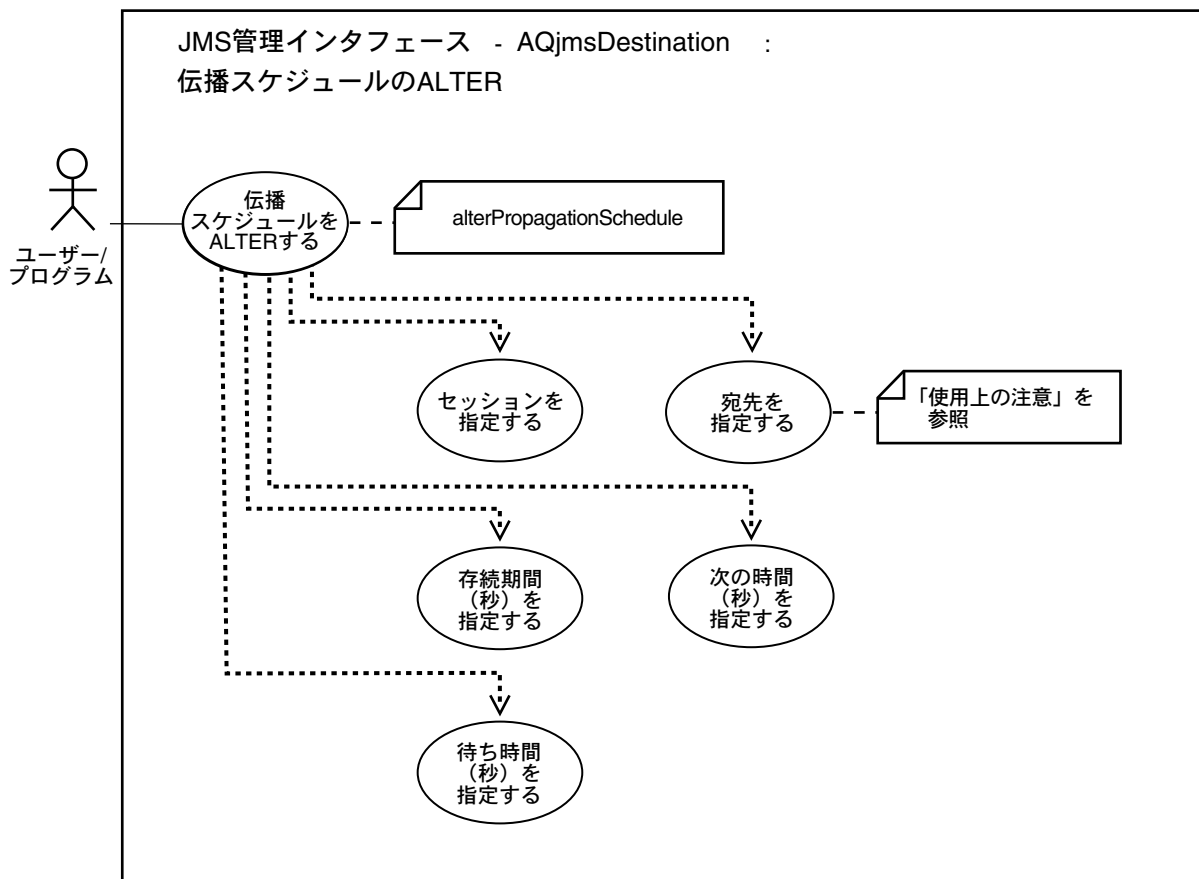
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の enablePropagationSchedule を参照してください。

## 例

```
TopicSession      t_sess;  
Topic              topic;  
  
( (AQjmsDestination) topic ).enablePropagationSchedule(t_sess, "dbs1");
```

## 伝播スケジュールの変更

図 13-26 ユースケース図：伝播スケジュールの変更

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

伝播スケジュールを変更します。

## 使用上の注意

NULL の宛先は、ローカル・データベースに伝播されることを示します。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の alterPropagationSchedule を参照してください。

## 例

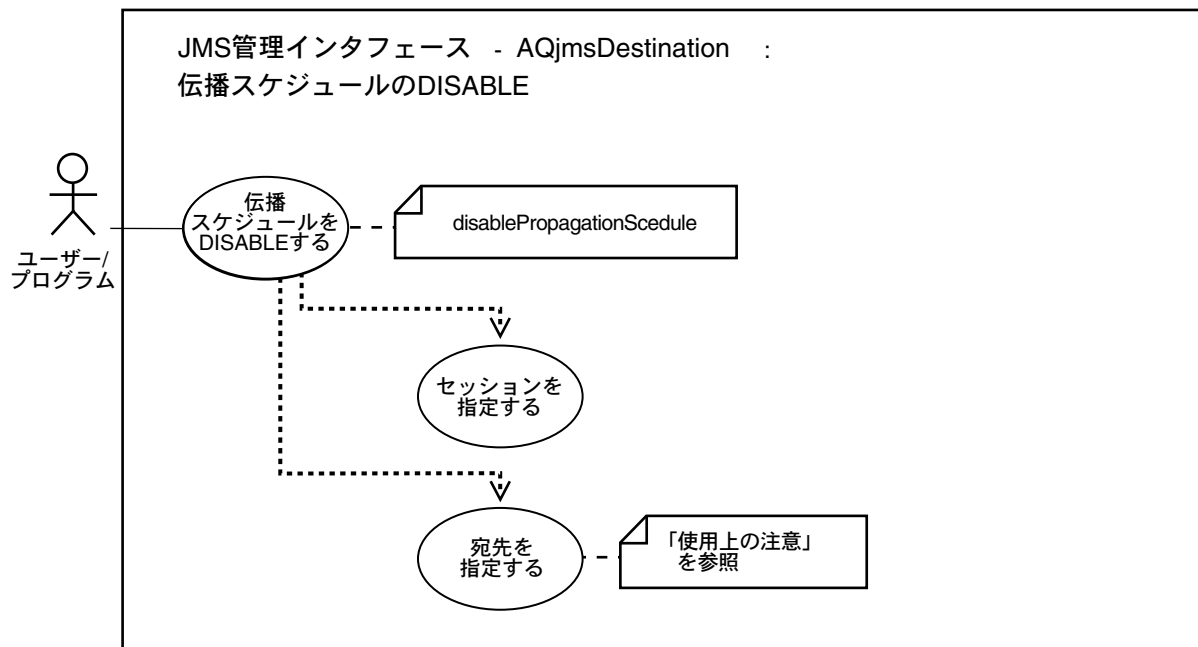
```
TopicSession          t_sess;
Topic                  topic;

((AQjmsDestination)topic).alterPropagationSchedule(t_sess, null, 30, null, new
Double(30));
```



## 伝播スケジュールの使用不可

図 13-27 ユースケース図：伝播スケジュールの使用不可



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

伝播スケジュールを使用不可にします。

## 使用上の注意

NULL の宛先は、ローカル・データベースに伝播されることを示します。

## 構文

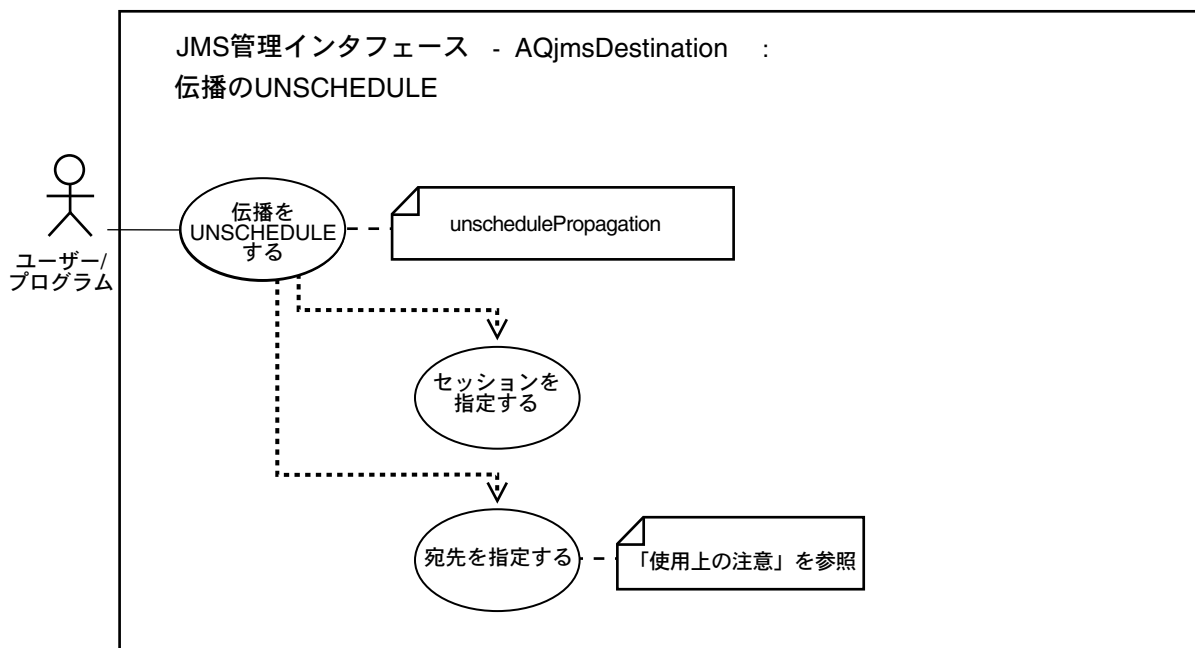
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の `disablePropagationSchedule` を参照してください。

## 例

```
TopicSession      t_sess;  
Topic              topic;  
  
( (AQjmsDestination) topic ).disablePropagationSchedule(t_sess, "dbs1");
```

## 伝播スケジュールの解除

図 13-28 ユースケース図：伝播スケジュールの解除



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

伝播スケジュールを解除します。

## 使用上の注意

以前にスケジュールされた伝播スケジュールを解除してください。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsDestination」の `unschedulePropagation` を参照してください。

## 例

```
TopicSession  t_sess;  
Topic          topic;  
  
((AQjmsDestination)topic).unschedulePropagation(t_sess, "dbs1");
```

---

## JMS 操作インタフェース：基本操作 (Point-to-Point)

### ユースケース・モデル

この章では、Oracle アドバンスド・キューイングの操作インタフェースをユースケースに沿って説明します。それぞれの操作（[キュー送信者の作成](#)など）を、その操作名ごとにユースケースの点から説明します。この章の先頭に、すべてのユースケースを示します（14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作（Point-to-Point）](#)」を参照）。

### ユースケース・モデルの図形概要

[図 14-1「ユースケース図：JMS 操作インタフェース – 基本操作（Point-to-Point）」](#)に、すべてのユースケースを 1 つの図にまとめています。

### 個々のユースケース

各ユースケースは、次のように配置されています。

- **ユースケース図**：ユースケースを表す図
- **用途**：このユースケースの用途
- **使用上の注意**：実装に有効なガイドライン
- **構文**：このアクティビティの実行に使用する主な構文
- **例**：各プログラム環境でのユースケースの例

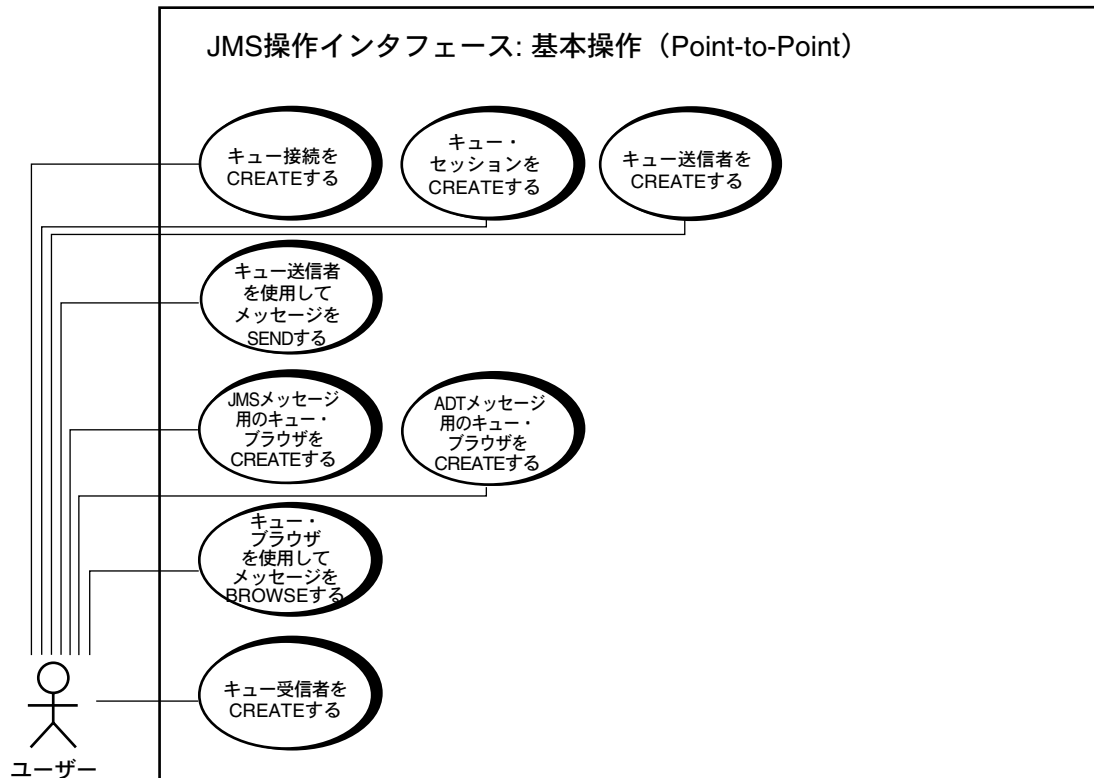
# ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)

表 14-1 ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)

ユースケース
14-4 ページの「キュー接続を確立する 3 つの方法」
14-5 ページの「ユーザー名 / パスワードでのキュー接続の確立」
14-7 ページの「オープン JDBC 接続でのキュー接続の確立」
14-9 ページの「デフォルトのコネクション・ファクトリ・パラメータでのキュー接続の確立」
14-11 ページの「キュー・セッションの作成」
14-13 ページの「キュー送信者の作成」
14-15 ページの「キュー送信者を使用してメッセージを送信する 2 つの方法」
14-16 ページの「デフォルトの送信オプションを持つキュー送信者を使用したメッセージの送信」
14-18 ページの「送信オプションの指定によるキュー送信者を使用したメッセージの送信」
14-21 ページの「JMS メッセージ・キュー用のキュー・ブラウザを作成する 2 つの方法」
14-22 ページの「Text、Stream、Object、Byte または Map メッセージを含むキュー用のキュー・ブラウザの作成」
14-24 ページの「Text、Stream、Object、Byte または Map メッセージを含むキュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザの作成」
14-26 ページの「Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザを作成する 2 つの方法」
14-27 ページの「Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザの作成」
14-29 ページの「Oracle オブジェクト型 (ADT) メッセージ・キュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザの作成」
14-31 ページの「キュー・ブラウザを使用したメッセージのブラウズ」
14-33 ページの「キュー受信者を作成する 2 つの方法」
14-34 ページの「標準 JMS 型メッセージ・キューに対するキュー受信者の作成」
14-36 ページの「Oracle オブジェクト型 (ADT) メッセージ・キューに対するキュー受信者の作成」

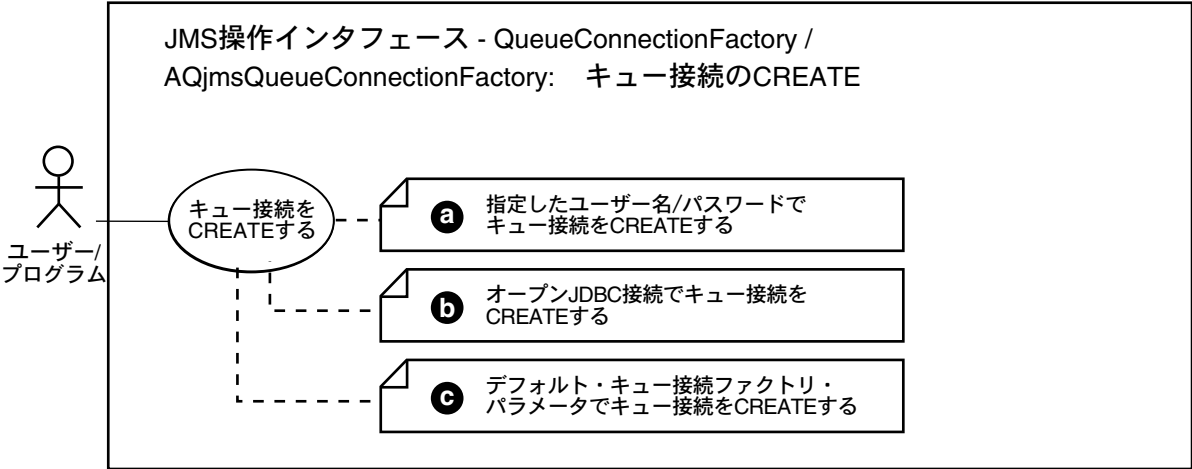
## ユースケース図 : JMS 操作インタフェース - 基本操作 (Point-to-Point)

図 14-1 ユースケース図 : JMS 操作インタフェース - 基本操作 (Point-to-Point)



# キュー接続を確立する 3 つの方法

図 14-2 ユースケース図：キュー接続を確立する 3 つの方法



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

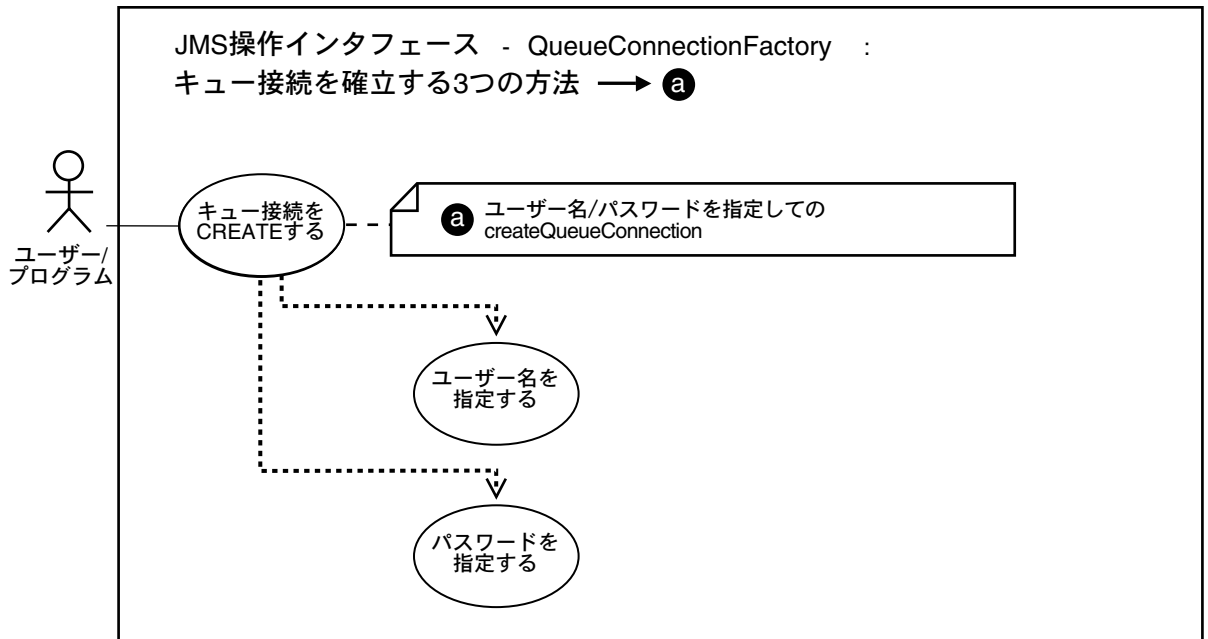
ここでは、キュー接続を確立する 3 つの方法について説明します。

- a. 14-5 ページの「ユーザー名 / パスワードでのキュー接続の確立」
- b. 14-7 ページの「オープン JDBC 接続でのキュー接続の確立」
- c. 14-9 ページの「デフォルトのコネクション・ファクトリ・パラメータでのキュー接続の確立」



## ユーザー名 / パスワードでのキュー接続の確立

図 14-3 ユースケース図：ユーザー名 / パスワードでのキュー接続の確立



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

ユーザー名 / パスワードで、キュー接続を確立します。

## 使用上の注意

ありません。

### 構文

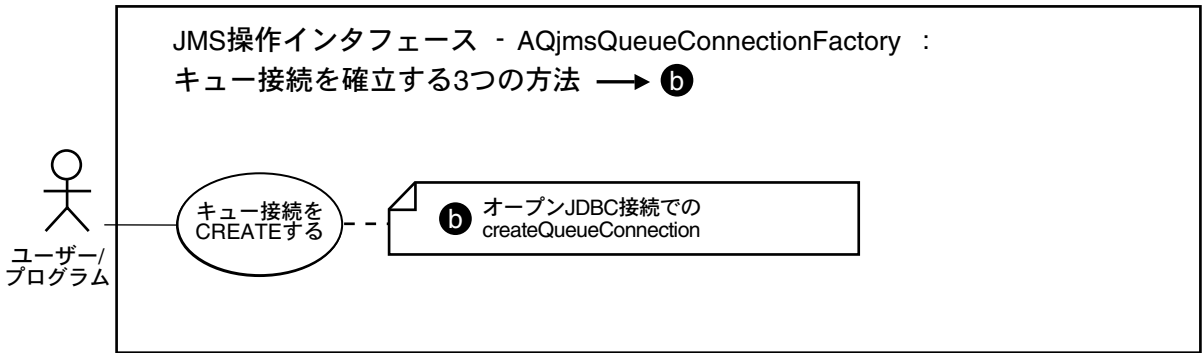
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsQueueConnectionFactory」の createQueueConnection を参照してください。

### 例

```
QueueConnectionFactory qc_fact = AQjmsFactory.getQueueConnectionFactory("sun123",  
"oratest", 5521, "thin");  
/* ユーザー名 / パスワードを使用して、キュー・コネクションを作成します。*/  
QueueConnection qc_conn = qc_fact.createQueueConnection("jmsuser", "jmsuser");
```

# オープン JDBC 接続でのキュー接続の確立

図 14-4 ユースケース図：オープン JDBC 接続でのキュー接続の確立



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

オープン JDBC 接続で、キュー接続を確立します。

## 使用上の注意

これはスタティック・メソッドです。

## 構文

Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsQueueConnectionFactory」の createQueueConnection を参照してください。

## 例

### 例 1

ユーザーが JMS 操作に対して既存の（たとえば接続プールの）JDBC 接続を使用するとき、このメソッドが使用されます。この場合、JMS は新しい接続をオープンするのではなく、提供された JDBC 接続を使用して、JMS キュー接続オブジェクトを作成します。

```
Connection db_conn;      /* 事前にオープンされた JDBC 接続 */
QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(db_
conn);
```

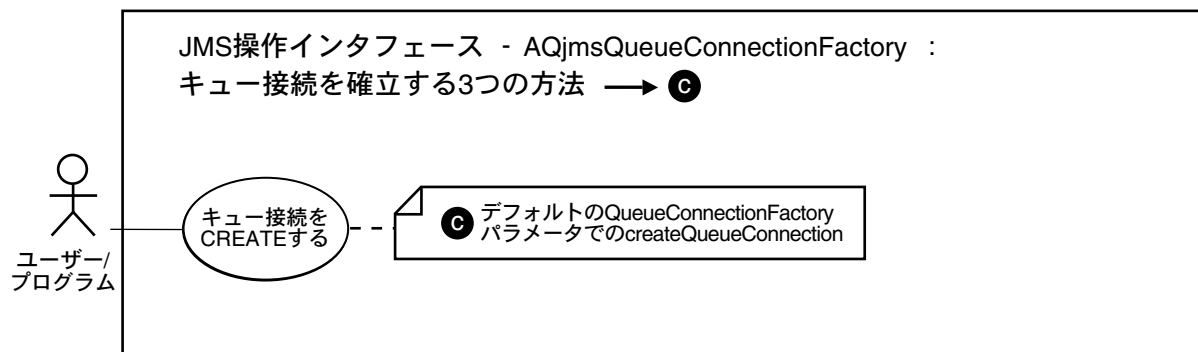
### 例 2

このメソッドは、データベース（JDBC サーバー・ドライバ）内の Java ストアド・プロシージャから JMS を使用する場合に、JMS キュー接続を確立する唯一の方法です。

```
OracleDriver ora = new OracleDriver();
QueueConnection qc_conn =
AQjmsQueueConnectionFactory.createQueueConnection(ora.defaultConnection());
```

## デフォルトのコネクション・ファクトリ・パラメータでのキュー接続の確立

図 14-5 ユースケース図：デフォルトのコネクション・ファクトリ・パラメータでのキュー接続の確立



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

デフォルトのコネクション・ファクトリ・パラメータで、キュー接続を確立します。

## 使用上の注意

QueueConnectionFactory のプロパティには、デフォルトのユーザー名 / パスワードを含める必要があります。そうでない場合、このメソッドでは JMS 例外が発生します。

## 構文

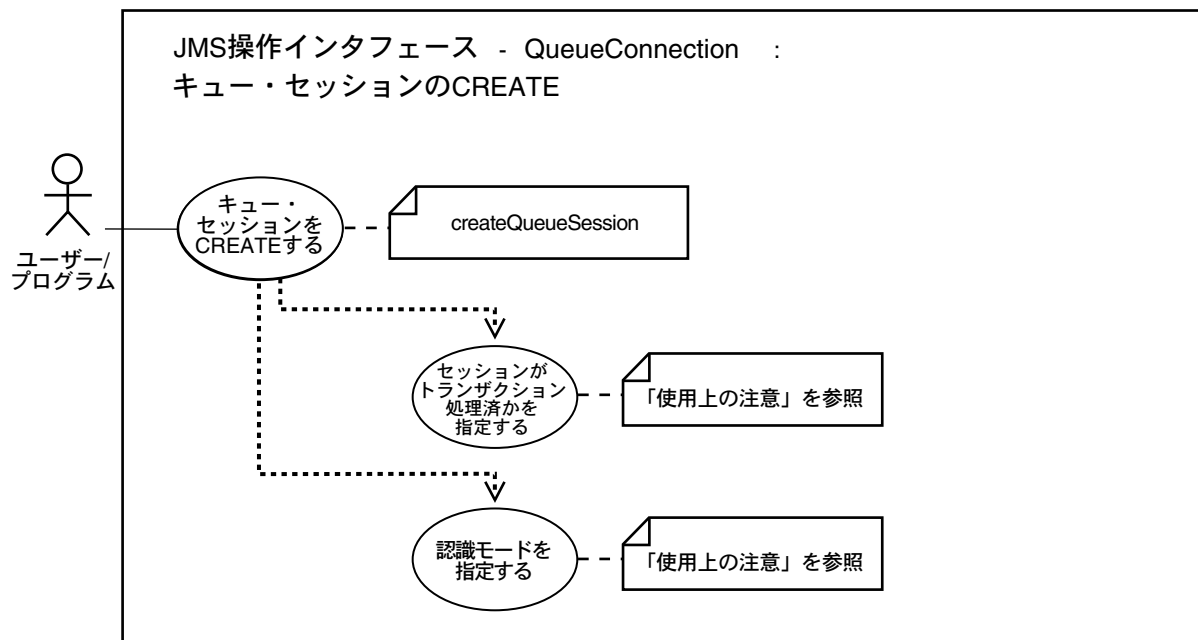
Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsQueueConnectionFactory」の createQueueConnection を参照してください。

## 例

今回のリリースでは、例は記載していません。

## キュー・セッションの作成

図 14-6 ユースケース図：キュー・セッションの作成



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

キュー・セッションを作成します。

## 使用上の注意

現行のバージョンでは、トランザクション処理済のセッションのみがサポートされます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsConnection」の `createQueueSession` を参照してください。

## 例

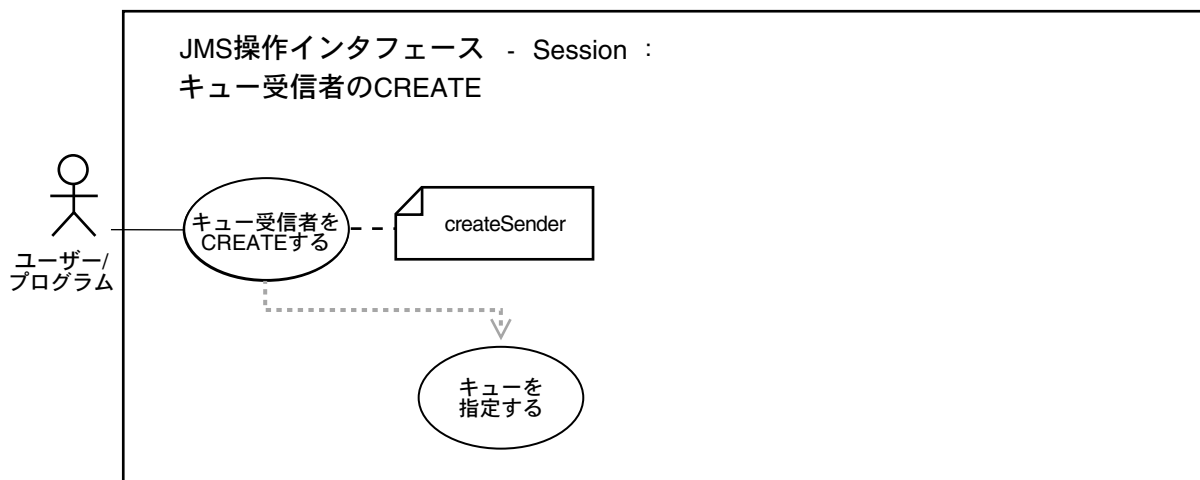
現行のバージョンでは、トランザクション処理済のセッションのみがサポートされます。

```
QueueConnection qc_conn;  
QueueSession q_sess = qc_conn.createQueueSession(true, 0);
```



## キュー送信者の作成

図 14-7 ユースケース図：キュー送信者の作成



---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

キュー送信者を作成します。

## 使用上の注意

デフォルトのキューを使用しないで送信者が作成された場合、送信操作を行うたびに宛先キューを指定する必要があります。

## 構文

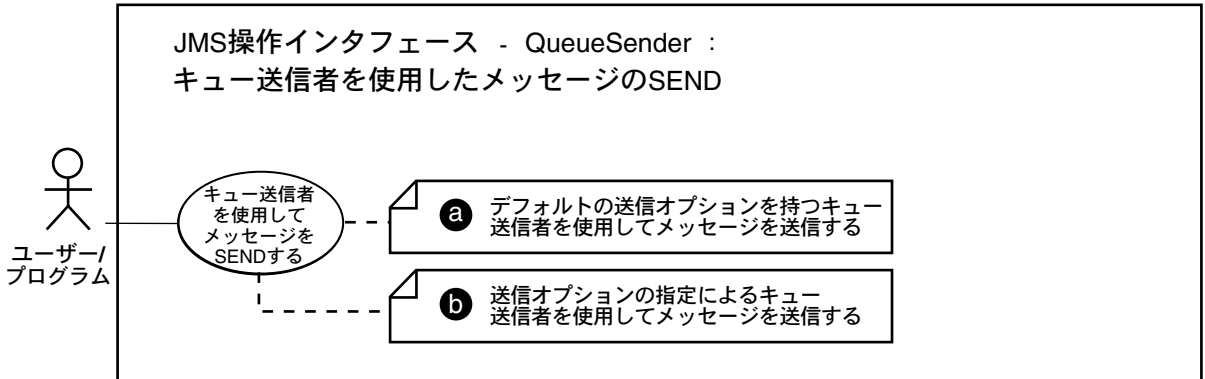
Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の createSender を参照してください。

## 例

今回のリリースでは、例は記載していません。

## キュー送信者を使用してメッセージを送信する2つの方法

図 14-8 ユースケース図：キュー送信者を使用してメッセージを送信する2つの方法



### 参照：

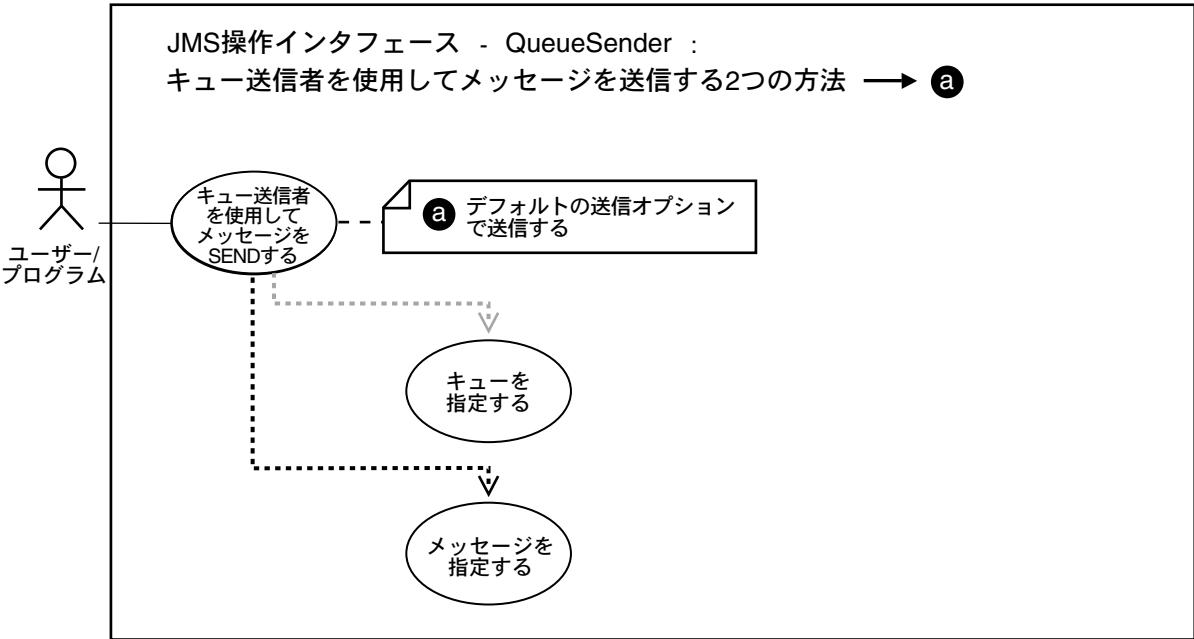
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、キュー送信者を使用してメッセージを送信する2つの方法について説明します。

- 14-16 ページの「デフォルトの送信オプションを持つキュー送信者を使用したメッセージの送信」
- 14-18 ページの「送信オプションの指定によるキュー送信者を使用したメッセージの送信」

# デフォルトの送信オプションを持つキュー送信者を使用したメッセージの送信

図 14-9 ユースケース図：デフォルトの送信オプションを持つキュー送信者を使用したメッセージの送信



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

デフォルトの送信オプションを持つキュー送信者を使用して、メッセージを送信します。

## 使用上の注意

キュー送信者がデフォルト・キューで作成されている場合、必ずしも `send` コールにキュー・パラメータを指定する必要はありません。キューが送信操作で指定されている場合、この値はキュー送信者のデフォルト・キューをオーバーライドします。

キュー送信者がデフォルト・キューを使用しないで作成されている場合、`send` コールごとにキュー・パラメータを指定する必要があります。

この送信操作では、メッセージ・プロパティ (1) および Time-To-Live (無制限) のデフォルト値が使用されます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャリファレンス』の第2章「パッケージ `oracle.jms`」の「`AQjmsQueueSender`」の `send` を参照してください。

## 例

### 例 1

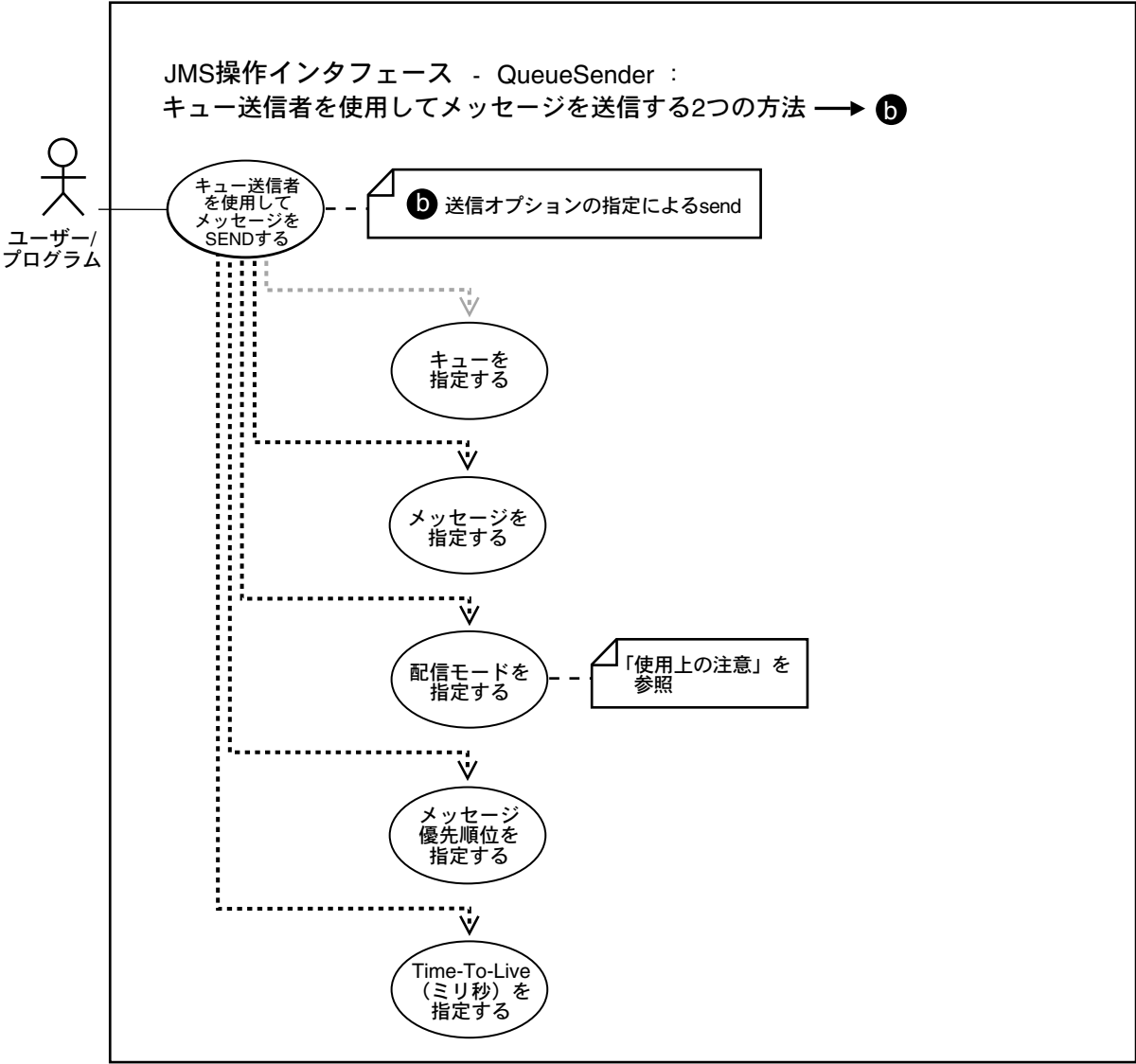
```
/* すべてのキューにメッセージを送る送信者を作成します。*/  
QueueSession jms_sess;  
QueueSender sender1;  
TextMessage message;  
sender1 = jms_sess.createSender(null);  
sender1.send(queue, message);
```

### 例 2

```
/* 特定のキューにメッセージを送る送信者を作成します。*/  
QueueSession jms_sess;  
QueueSender sender2;  
Queue billed_orders_que;  
TextMessage message;  
sender2 = jms_sess.createSender(billed_orders_que);  
sender2.send(queue, message);
```

# 送信オプションの指定によるキュー送信者を使用したメッセージの送信

図 14-10 ユースケース図：送信オプションの指定によるキュー送信者を使用したメッセージの送信



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

送信オプションを指定することによって、キュー送信者を使用してメッセージを送信します。

## 使用上の注意

キュー送信者がデフォルト・キューで作成されている場合、必ずしも send コールにキュー・パラメータを指定する必要はありません。キューが送信操作で指定されている場合、この値はキュー送信者のデフォルト・キューをオーバーライドします。

キュー送信者がデフォルト・キューを使用しないで作成されている場合、send コールごとにキュー・パラメータを指定する必要があります。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsQueueSender」の send を参照してください。

## 例

### 例 1

```
/* すべてのキューにメッセージを送る送信者を作成します。*/  
/* 優先順位が 2 で、Time-To-Live が 100000 ミリ秒の new_orders_que にメッセージを送信しま  
す。*/  
QueueSession jms_sess;  
QueueSender sender1;  
TextMessage mesg;  
Queue new_orders_que  
sender1 = jms_sess.createSender(null);  
sender1.send(new_orders_que, mesg, DeliveryMode.PERSISTENT, 2, 100000);
```

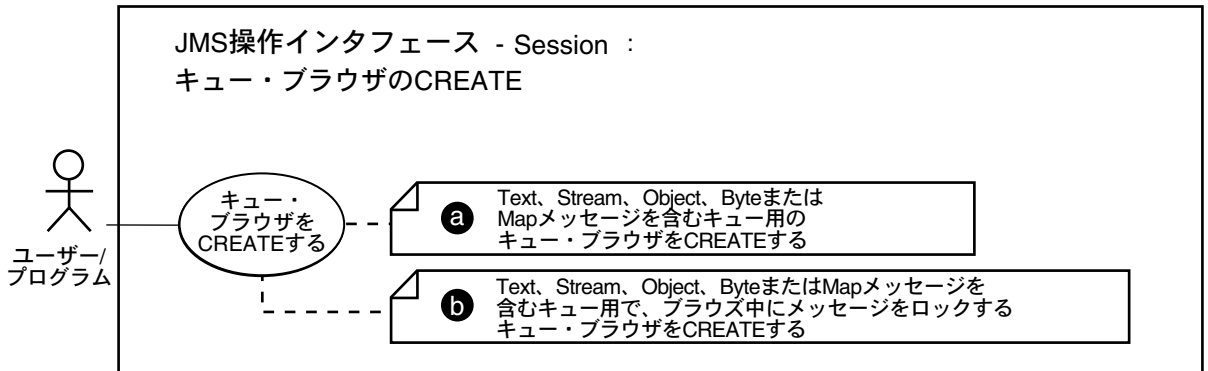
### 例 2

```
/* 特定のキューにメッセージを送る送信者を作成します。*/  
/* 優先順位が 1 で、Time-To-Live が 400000 ミリ秒の billed_orders_que にメッセージを送信しま  
す。*/  
QueueSession jms_sess;  
QueueSender sender2;  
Queue billed_orders_que;  
TextMessage mesg;  
sender2 = jms_sess.createSender(billed_orders_que);  
sender2.send(mesg, DeliveryMode.PERSISTENT, 1, 400000);
```



## JMS メッセージ・キュー用のキュー・ブラウザを作成する 2 つの方法

図 14-11 ユースケース図：標準 JMS メッセージ・キュー用のキュー・ブラウザを作成する 2 つの方法



### 参照：

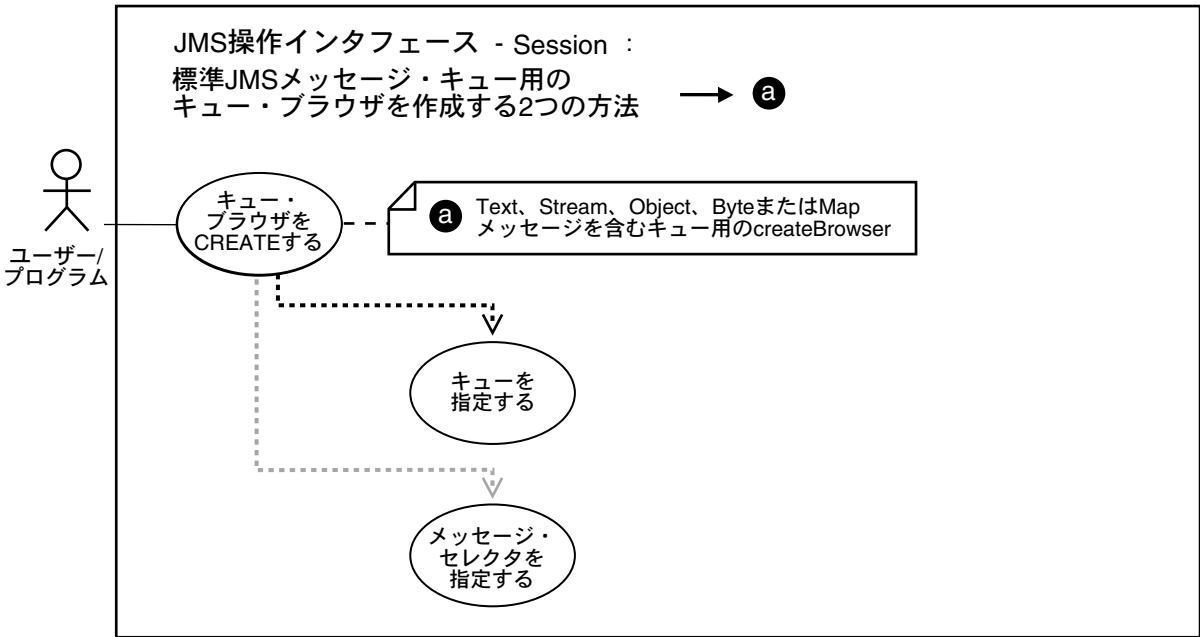
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、JMS メッセージ・キュー用のキュー・ブラウザを作成する 2 つの方法について説明します。

- 14-22 ページの「Text、Stream、Object、Byte または Map メッセージを含むキュー用のキュー・ブラウザの作成」
- 14-24 ページの「Text、Stream、Object、Byte または Map メッセージを含むキュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザの作成」

# Text、Stream、Object、Byte または Map メッセージを含む キュー用のキュー・ブラウザの作成

図 14-12 ユースケース図：Text、Stream、Object、Byte または Map メッセージを含むキュー用の  
キュー・ブラウザの作成



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

Text、Stream、Object、Byte または Map メッセージを含むキュー用の、キュー・ブラウザを作成します。

## 使用上の注意

ある関連 ID を持つメッセージを取り出すには、キュー・ブラウザに対するセクタを次のいずれかに指定します。

- JMSMessageID = 'ID:2345234523452345'  
指定されたメッセージ ID を持つメッセージを取り出します。
- JMSCorrelationID = 'RUSH'
- JMSCorrelationID LIKE 'RE%'

すべてのメッセージ ID には、接頭辞「ID:」を付ける必要があります。

java.util Enumeration 内のメソッドを使用して、メッセージのリストを参照してください。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQJmsSession」の createBrowser を参照してください。

## 例

### 例 1

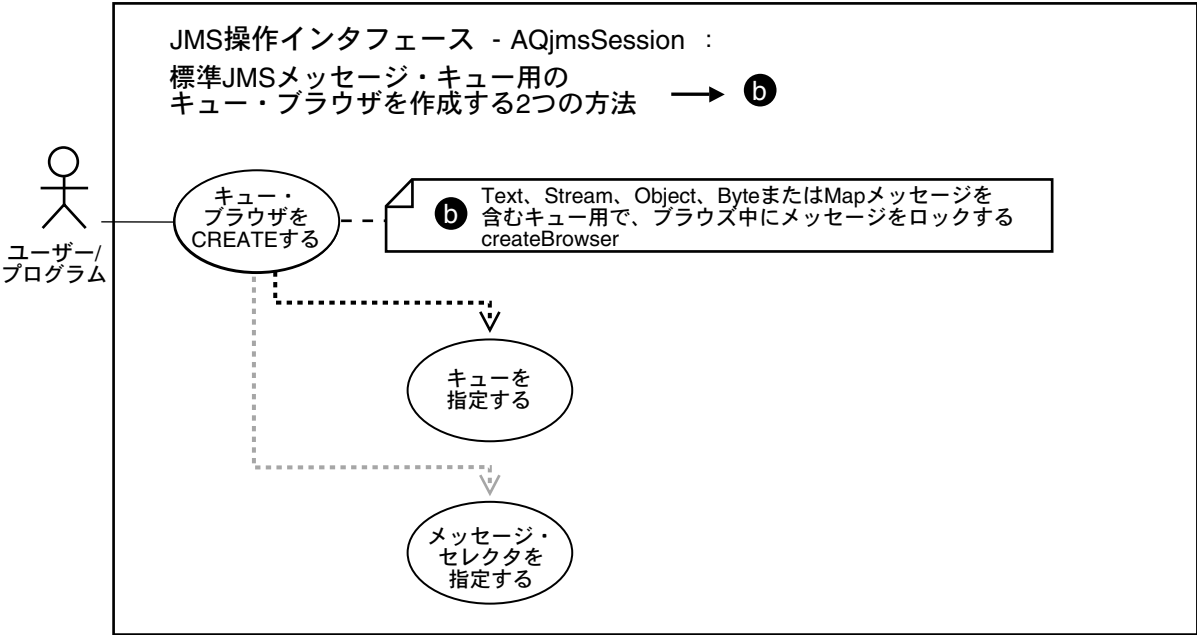
```
/* セクタを指定せずにブラウザを作成します。*/  
QueueSession jms_session;  
QueueBrowser browser;  
Queue queue;  
  
browser = jms_session.createBrowser(queue);
```

### 例 2

```
/* セクタを指定して、キューのブラウザを作成します。*/  
QueueSession jms_session;  
QueueBrowser browser;  
Queue queue;  
  
/* correlationID = RUSH のメッセージを調べるためのブラウザを作成します。*/  
browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'");
```

# Text、Stream、Object、Byte または Map メッセージを含む キュー用で、ブラウズ中にメッセージをロックするキュー・ブ 라우저の作成

図 14-13 ユースケース図：Text、Stream、Object、Byte または Map メッセージを含むキュー用  
で、ブラウズ中にメッセージをロックするキュー・ブラウザの作成



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

Text、Stream、Object、Byte または Map メッセージを含むキュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザを作成します。

## 使用上の注意

locked パラメータが TRUE に指定されている場合、ブラウズ中のメッセージはロックされます。したがって、ブラウズ中のセッションがトランザクションを終了するまで、他のコンシューマがこれらのメッセージを削除することはできません。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQJmsSession」の createBrowser を参照してください。

## 例

### 例 1

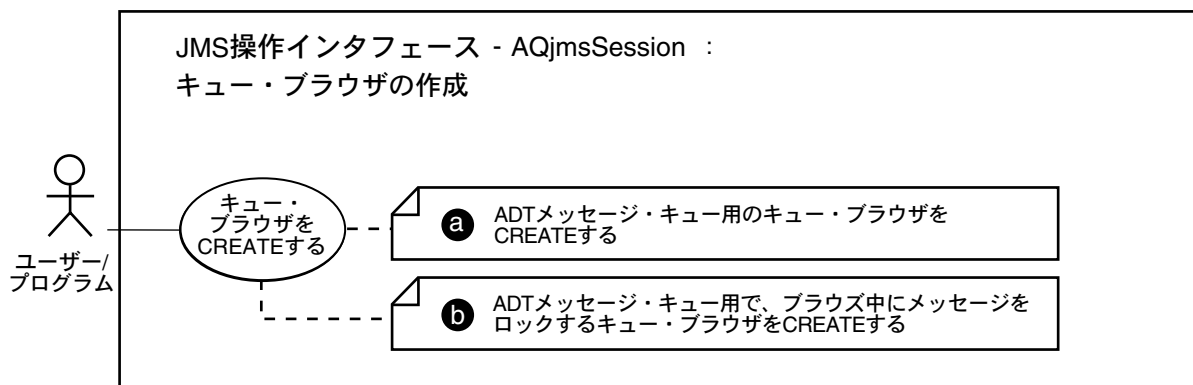
```
/* セレクタを指定せずにブラウザを作成します。*/  
QueueSession    jms_session;  
QueueBrowser     browser;  
Queue            queue;  
  
browser = jms_session.createBrowser(queue, null, true);
```

### 例 2

```
/* セレクタを指定して、キューのブラウザを作成します。*/  
QueueSession    jms_session;  
QueueBrowser     browser;  
Queue            queue;  
  
/* correlationID = RUSH でロック・モードのメッセージを調べるために、ブラウザを作成します。*/  
  
browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'", true);
```

## Oracle オブジェクト型 (ADT) メッセージ・キュー用の キュー・ブラウザを作成する 2 つの方法

図 14-14 ユースケース図 : Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザを作成する 2 つの方法



### 参照：

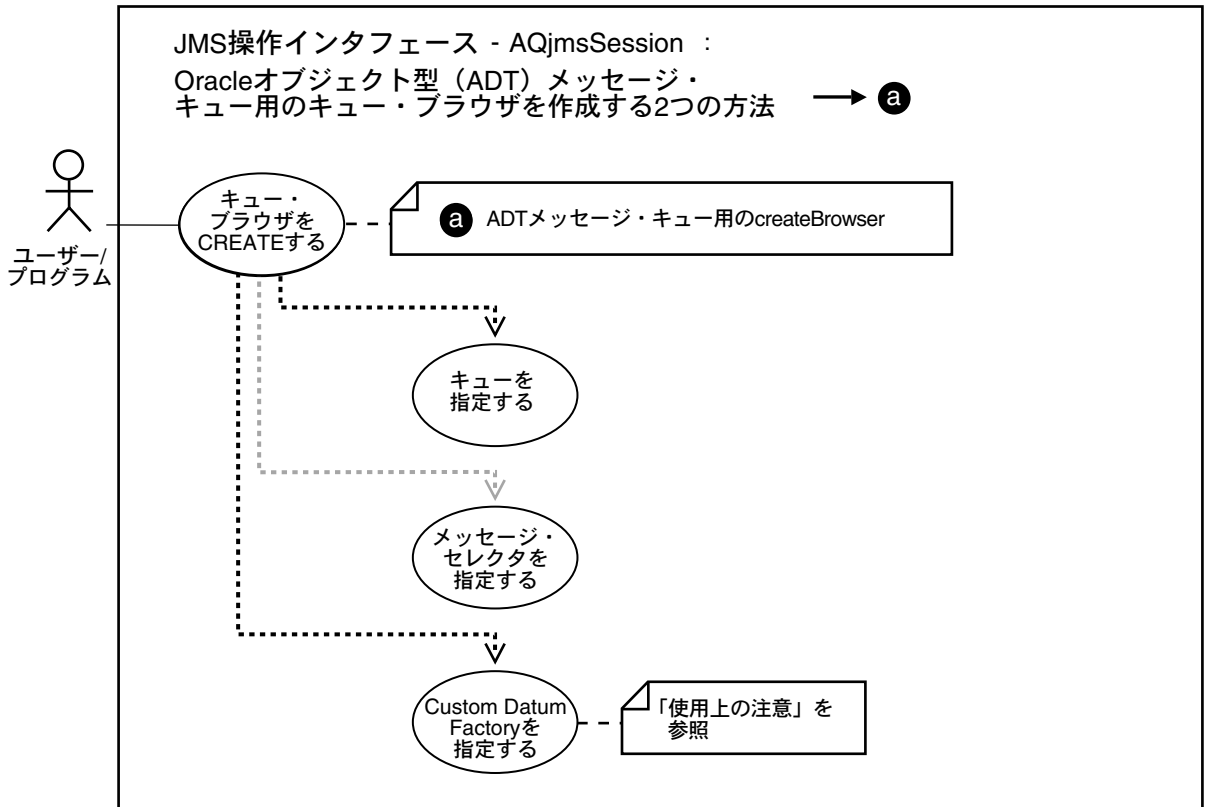
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

ここでは、Oracle オブジェクト型 (ADT) メッセージ・キュー用のブラウザを作成する 2 つの方法について説明します。

- 14-27 ページの「Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザの作成」
- 14-29 ページの「Oracle オブジェクト型 (ADT) メッセージ・キュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザの作成」

## Oracle オブジェクト型 (ADT) メッセージ・キュー用の キュー・ブラウザの作成

図 14-15 ユースケース図：Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザの作成



## 用途

---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

Oracle オブジェクト型 (ADT) メッセージ・キュー用のキュー・ブラウザを作成します。

## 使用上の注意

例を参照してください。

## 構文

Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の createBrowser を参照してください。

## 例

SQL ADT ペイロードにマップする特定の Java クラスに対する CustomDatum ファクトリは、getFactory スタティック・メソッドを介して取得できます。

キュー test\_queue は SCOTT.EMPLOYEE 型のペイロードを持ち、この ADT に対して、Employee という Java クラスが JPublisher によって生成されたと想定します。Employee クラスは、CustomDatum インタフェースを実装します。このクラスに対する CustomDatumFactory は、Employee.getFactory() メソッドを使用して取得できます。

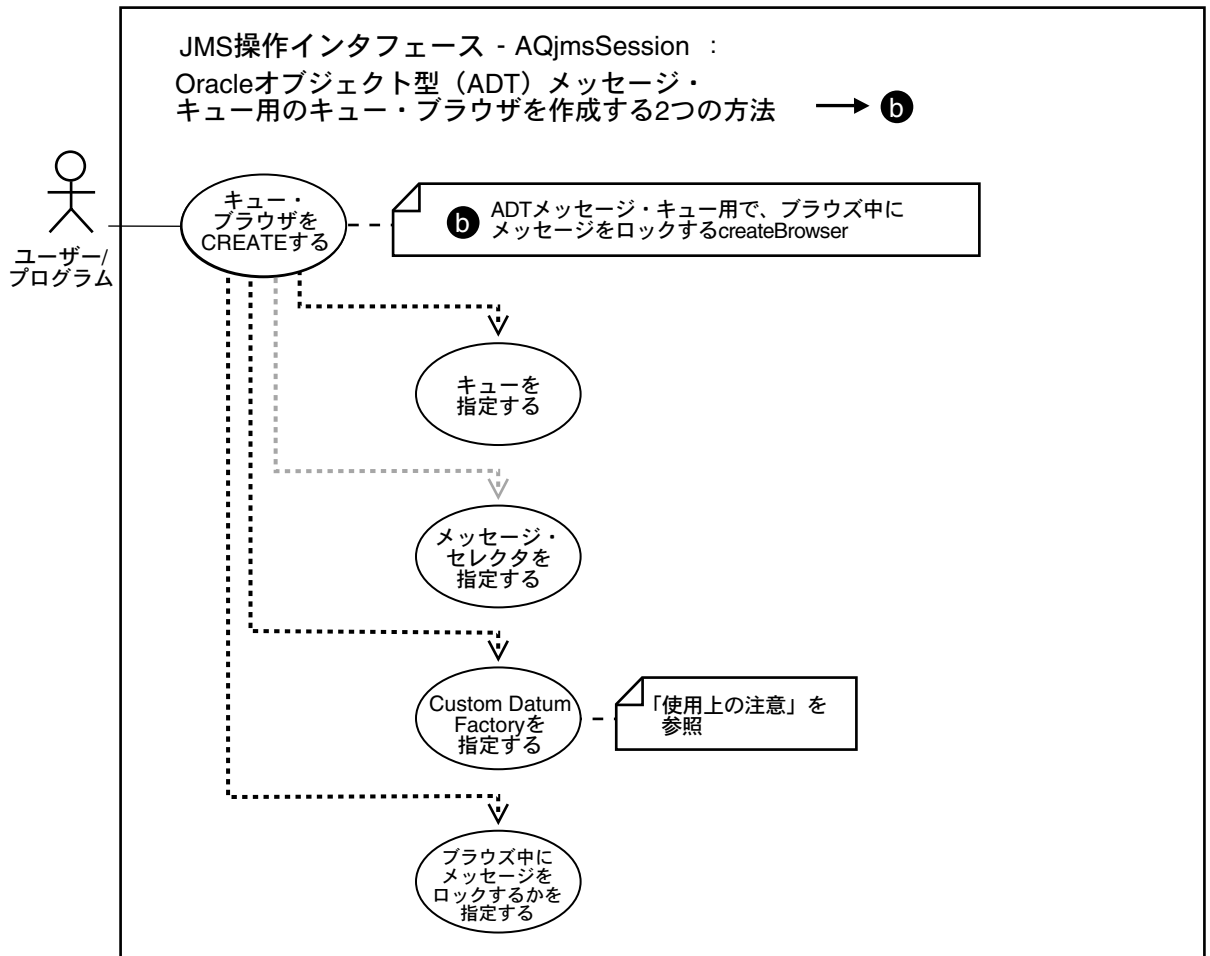
```
/* EMPLOYEE 型の ADT メッセージを持つキューのブラウザを作成します。*/
QueueSession jms_session
QueueBrowser browser;
Queue        test_queue;

browser = ((AQjmsSession)jms_session).createBrowser(test_queue, null,
Employee.getFactory());
```



## Oracle オブジェクト型 (ADT) メッセージ・キュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザの作成

図 14-16 ユースケース図：Oracle オブジェクト型 (ADT) メッセージ・キュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザの作成



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: [JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

Oracle オブジェクト型 (ADT) メッセージ・キュー用で、ブラウズ中にメッセージをロックするキュー・ブラウザを作成します。

## 使用上の注意

ありません。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の createBrowser を参照してください。

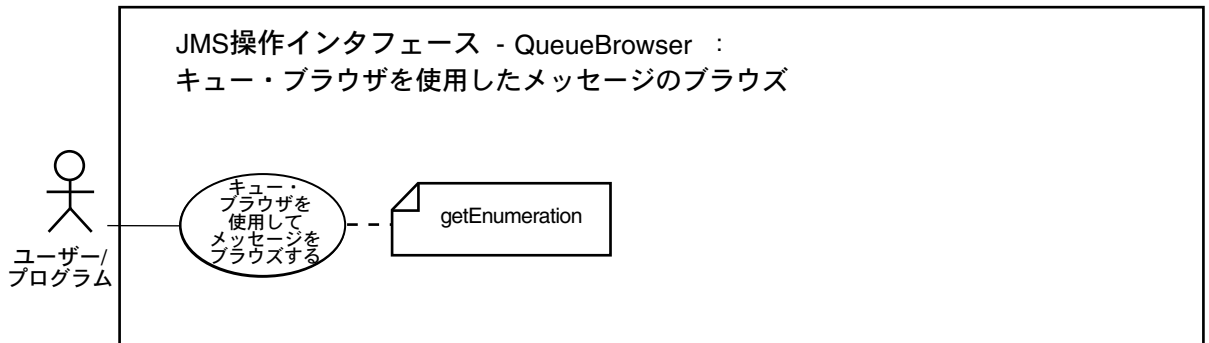
## 例

```
/* ロック・モードで EMPLOYEE 型の ADT メッセージを持つキューのブラウザを作成します。*/
QueueSession jms_session
QueueBrowser browser;
Queue         test_queue;

browser = ((AQjmsSession)jms_session).createBrowser(test_queue, null,
Employee.getFactory(), true);
```

## キュー・ブラウザを使用したメッセージのブラウズ

図 14-17 ユースケース図：キュー・ブラウザを使用したメッセージのブラウズ



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

### 用途：

キュー・ブラウザを使用して、メッセージをブラウズします。

### 使用上の注意

java.util.Enumeration 内のメソッドを使用して、メッセージのリストを参照してください。

### 構文

Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsQueueBrowser」を参照してください。

### 例

```
/* セレクタを指定して、キューのブラウザを作成します。*/
public void browse_rush_orders(QueueSession jms_session)
{
    QueueBrowser    browser;
    Queue            queue;
    ObjectMessage    obj_message;
    BolOrder         new_order;
    Enumeration      messages;

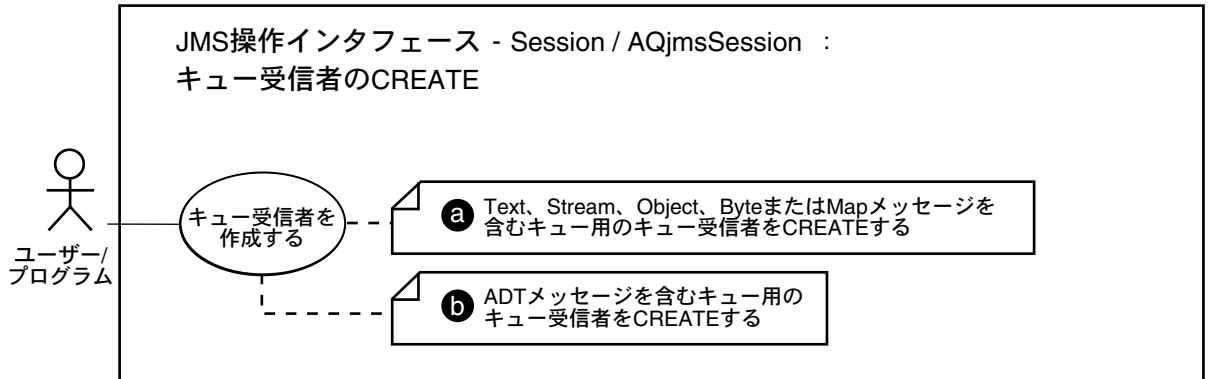
    /* new_orders キューへのハンドルを取得します。*/
    queue = ((AQJMSSession) jms_session).getQueue("OE", "OE_neworders_que");

    /* 至急 (RUSH) 注文を調べるためのブラウザを作成します。*/
    browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'");

    /* メッセージをブラウズします。*/
    for (messages = browser.elements() ; message.hasMoreElements() ;)
    {
        obj_message = (ObjectMessage)message.nextElement();
    }
}
```

## キュー受信者を作成する 2 つの方法

図 14-18 ユースケース図：キュー受信者を作成する 2 つの方法



### 参照：

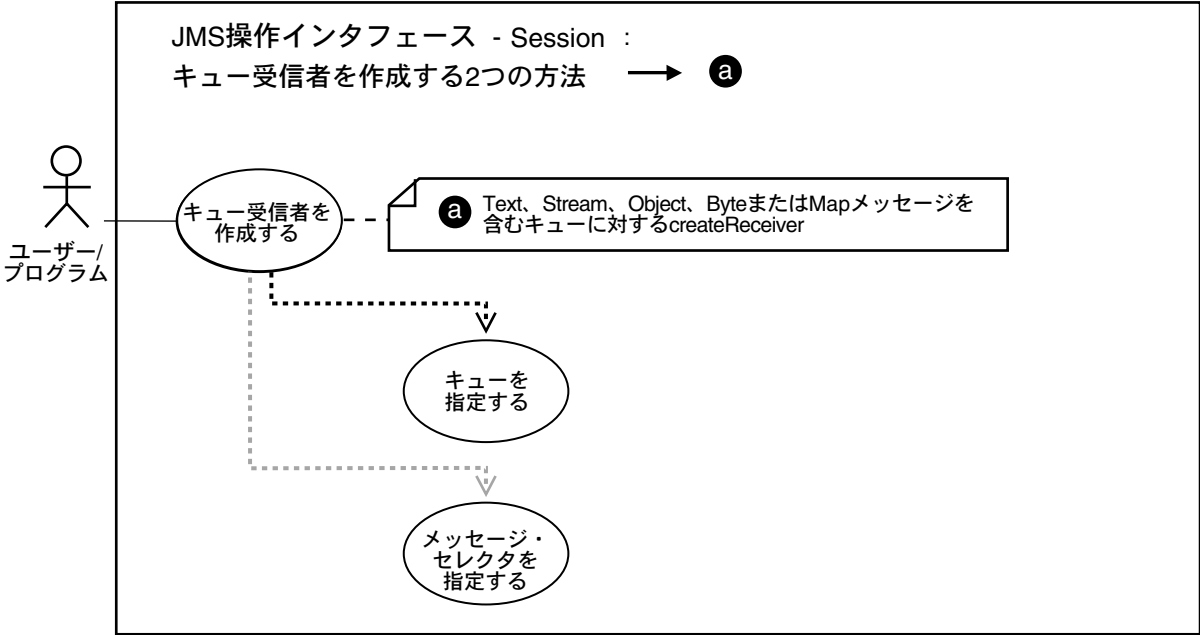
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、キュー受信者を作成する 2 つの方法について説明します。

- 14-34 ページの「標準 JMS 型メッセージ・キューに対するキュー受信者の作成」
- 14-36 ページの「Oracle オブジェクト型 (ADT) メッセージ・キューに対するキュー受信者の作成」

# 標準 JMS 型メッセージ・キューに対するキュー受信者の作成

図 14-19 ユースケース図：標準 JMS 型メッセージ・キューに対するキュー受信者の作成



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

標準 JMS 型メッセージ・キューに対するキュー受信者を作成します。

## 使用上の注意

キュー受信者に対するセレクトは、次のいずれかに指定します。

- JMSMessageID = 'ID:2345234523452345'  
指定されたメッセージ ID を持つメッセージを取り出します。すべてのメッセージ ID には、接頭辞「ID:」を付ける必要があります。
- JMSCorrelationID = 'EXPRESS'
- JMSCorrelationID LIKE 'RE%'  
ある相関 ID を持つメッセージを取り出します。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQJmsSession」の createReceiver を参照してください。

## 例

### 例 1

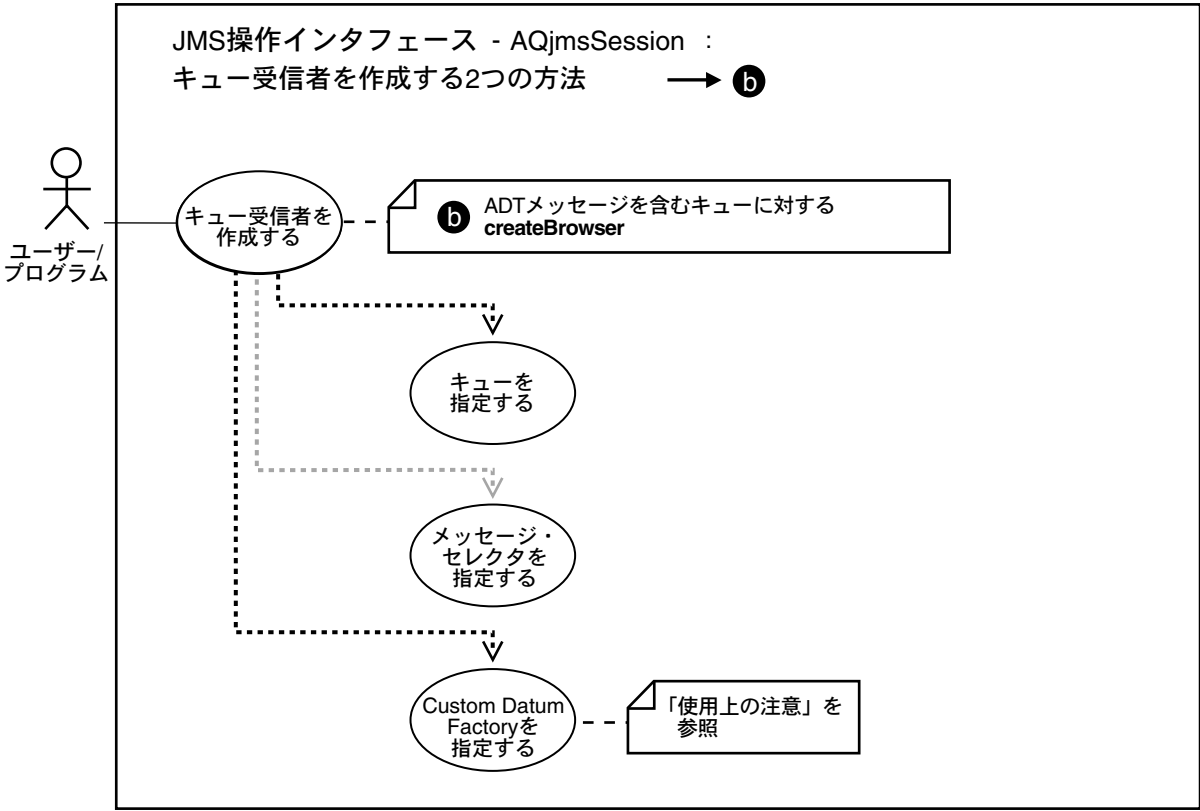
```
/* セレクトを指定せずに受信者を作成します。*/  
QueueSession    jms_session  
QueueReceiver    receiver;  
Queue            queue;  
  
receiver = jms_session.createReceiver(queue);
```

### 例 2

```
/* セレクトを指定して、キューの受信者を作成します。*/  
QueueSession    jms_session;  
QueueReceiver    receiver;  
Queue            queue;  
  
/* correlationID の先頭が EXP のメッセージを受信する受信者を作成します。*/  
browser = jms_session.createReceiver(queue, "JMSCorrelationID LIKE 'EXP%'");
```

# Oracle オブジェクト型 (ADT) メッセージ・キューに対する キュー受信者の作成

図 14-20 ユースケース図：Oracle オブジェクト型 (ADT) メッセージ・キューに対するキュー  
受信者の作成





---

**参照:**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

Oracle オブジェクト型 (ADT) メッセージ・キューに対するキュー受信者を作成します。

## 使用上の注意

SQL ADT ペイロードにマップする特定の Java クラスに対する CustomDatum ファクトリは、getFactory スタティック・メソッドを介して取得できます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の createReceiver を参照してください。

## 例

キュー test\_queue は SCOTT.EMPLOYEE 型のペイロードを持ち、このユーザー定義型に対して、Employee という Java クラスが JPublisher によって生成されたと想定します。Employee クラスは、CustomDatum インタフェースを実装します。このクラスに対する CustomDatumFactory は、Employee.getFactory() メソッドを使用して取得できます。

```
/* EMPLOYEE 型の ADT メッセージを持つキューの受信者を作成します。*/
QueueSession jms_session
QueueReceiver receiver;
Queue        test_queue;

browser = ((AQjmsSession) jms_session).createReceiver(test_queue, "JMSCorrelationID =
'MANAGER', Employee.getFactory());
```



---

## JMS 操作インタフェース：基本操作 (パブリッシュ / サブスクライブ)

### ユースケース・モデル

この章では、Oracle アドバンスト・キューイングの操作インタフェースをユースケースに沿って説明します。それぞれの操作（メッセージの公開など）を、その操作名ごとにユースケースの点から説明します。この章の先頭に、すべてのユースケースを示します（14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作（Point-to-Point）」を参照）。

### ユースケース・モデルの図形概要

図 15-1 「ユースケース図：JMS 操作インタフェース – 基本操作（パブリッシュ / サブスクライブ）」に、すべてのユースケースを 1 つの図にまとめています。

### 個々のユースケース

個々のユースケースは、次のとおり配置されています。

- **ユースケース図**：ユースケースを表す図
- **用途**：このユースケースの用途
- **使用上の注意**：実装に有効なガイドライン
- **構文**：このアクティビティの実行に使用する主な構文
- **例**：各プログラム環境でのユースケースの例

# ユースケース・モデル: JMS 操作インタフェース – 基本操作 (パブリッシュ / サブスクライブ)

表 15-1 ユースケース・モデル: JMS 操作インタフェース – 基本操作 (パブリッシュ / サブスクライブ)

ユースケース
15-5 ページの「トピック接続を確立する 3 つの方法」
15-6 ページの「ユーザー名 / パスワードでのトピック接続の確立」
15-8 ページの「既存の JDBC 接続でのトピック接続の確立」
15-10 ページの「デフォルトのコネクション・ファクトリ・パラメータでのトピック接続の確立」
15-11 ページの「トピック・セッションの作成」
15-13 ページの「トピック・パブリッシャの作成」
15-15 ページの「トピック・パブリッシャを使用してメッセージを公開する 4 つの方法」
15-16 ページの「最小限の指定でのメッセージの公開」
15-19 ページの「相関および遅延を指定したメッセージの公開」
15-16 ページの「最小限の指定でのメッセージの公開」
15-19 ページの「相関および遅延を指定したメッセージの公開」
15-22 ページの「優先順位および Time-To-Live を指定したメッセージの公開」
15-25 ページの「トピック・サブスクライバをオーバーライドする受信者リストを指定したメッセージの公開」
15-28 ページの「標準 JMS 型メッセージのトピックに対して永続サブスクライバを作成する 2 つの方法」
15-29 ページの「セクタを指定しない JMS トピックに対する永続サブスクライバの作成」
15-31 ページの「セクタを指定しての JMS トピックに対する永続サブスクライバの作成」
15-34 ページの「Oracle オブジェクト型 (ADT) メッセージのトピックに対する永続サブスクライバを作成する 2 つの方法」
15-35 ページの「セクタを指定しない JMS トピックに対する永続サブスクライバの作成」
15-37 ページのセクタを指定しての JMS トピックに対する永続サブスクライバの作成
15-40 ページの「リモート・サブスクライバを作成する 2 つの方法」
15-41 ページの「JMS メッセージのトピックに対するリモート・サブスクライバの作成」
15-44 ページの「Oracle オブジェクト型 (ADT) メッセージのトピックに対するリモート・サブスクライバの作成」

**表 15-1 (続き) ユースケース・モデル: JMS 操作インタフェース – 基本操作 (パブリッシュ / サブスクライブ)**

---

**ユースケース**

---

15-47 ページの「[永続サブスクリプションのサブスクライブを解除する 2 つの方法](#)」

15-48 ページの「[ローカル・サブスクライバに対する永続サブスクリプションのサブスクライブの解除](#)」

15-50 ページの「[リモート・サブスクライバに対する永続サブスクリプションのサブスクライブの解除](#)」

15-52 ページの「[トピック受信者を作成する 2 つの方法](#)」

15-53 ページの「[標準 JMS 型メッセージのトピックに対するトピック受信者の作成](#)」

15-55 ページの「[Oracle オブジェクト型 \(ADT\) メッセージのトピックに対するトピック受信者の作成](#)」

---

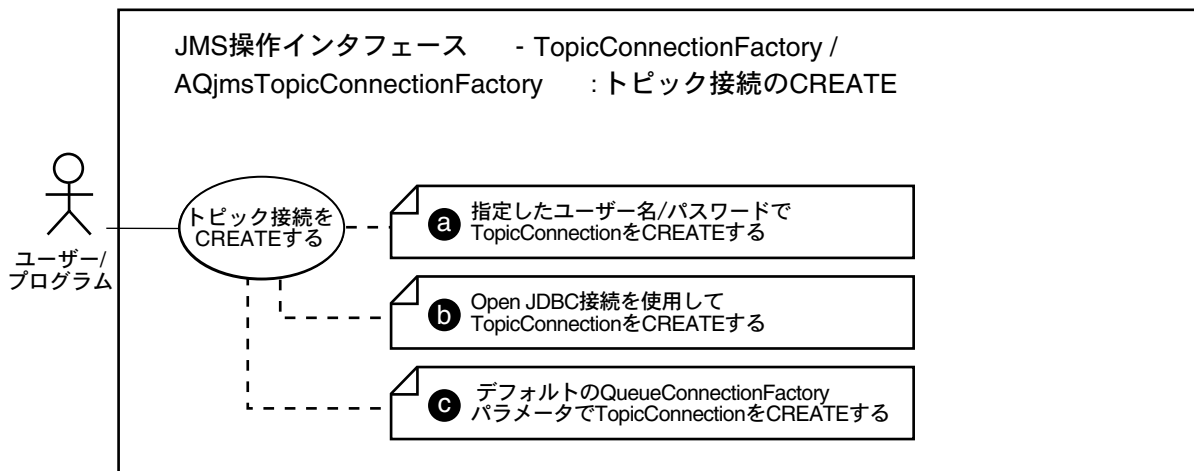
# ユースケース図 : JMS 操作インタフェース – 基本操作 (パブリッシュ / サブスクライブ)

図 15-1 ユースケース図 : JMS 操作インタフェース – 基本操作 (パブリッシュ / サブスクライブ)



## トピック接続を確立する 3 つの方法

図 15-2 ユースケース図：トピック接続を確立する 3 つの方法（パブリッシュ / サブスクライブ）



### 参照：

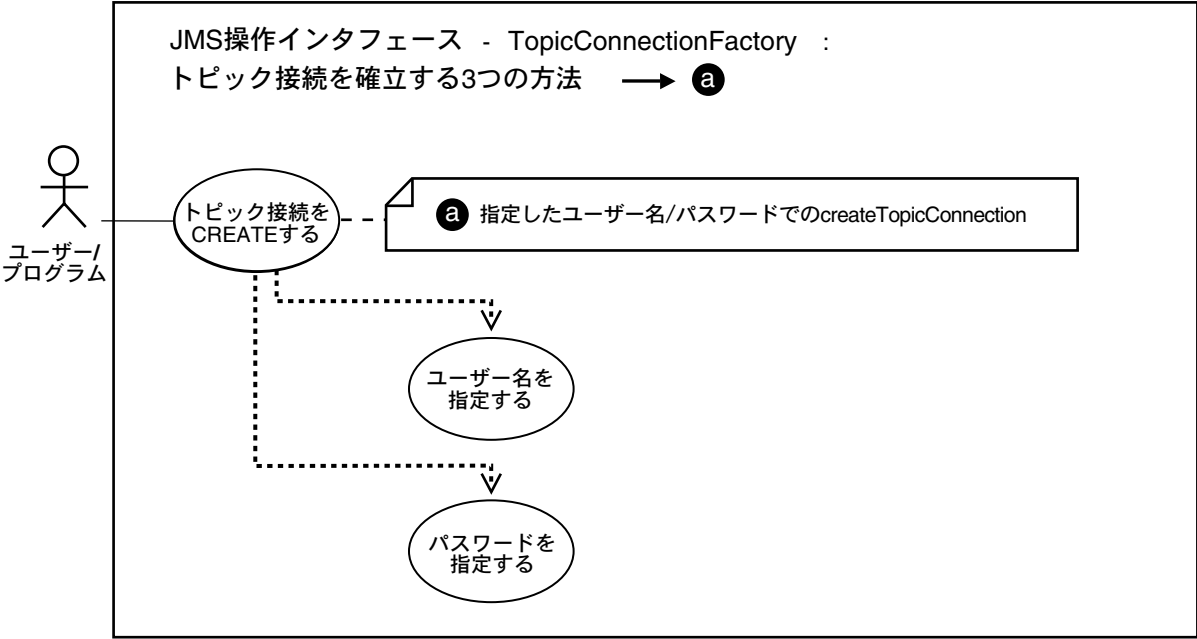
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、トピック接続を確立する 3 つの方法について説明します。

- 15-6 ページの「ユーザー名 / パスワードでのトピック接続の確立」
- 15-8 ページの「既存の JDBC 接続でのトピック接続の確立」
- 15-11 ページの「デフォルトのコネクション・ファクトリ・パラメータでのトピック接続の確立」

# ユーザー名 / パスワードでのトピック接続の確立

図 15-3 ユースケース図：ユーザー名 / パスワードでのトピック接続の確立（パブリッシュ / サブスクライブ）



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

ユーザー名 / パスワードで、トピック接続を確立します。

## 使用上の注意

ありません。



## 構文

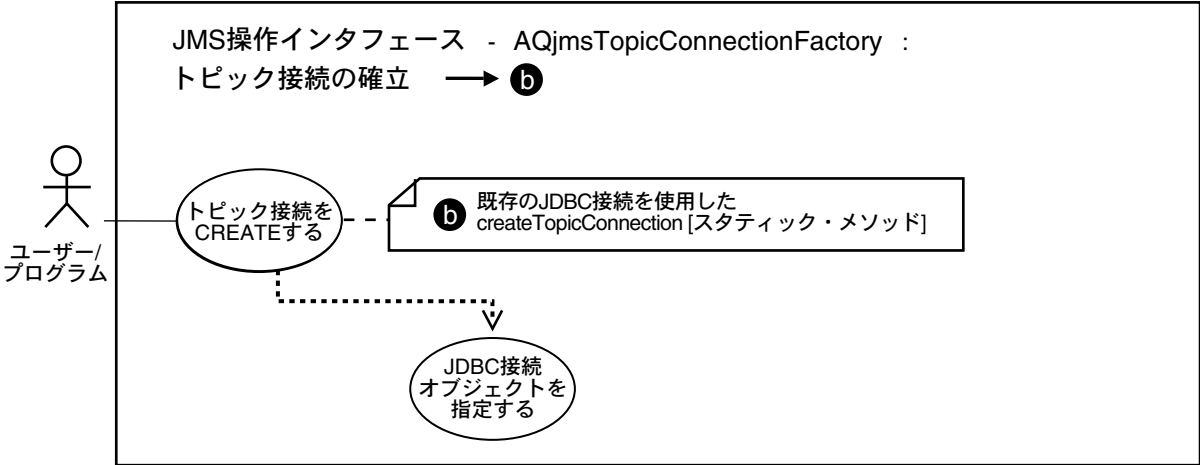
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsTopicConnectionFactory」の createTopicConnection を参照してください。

## 例

```
TopicConnectionFactory tc_fact = AQjmsFactory.getTopicConnectionFactory("sun123",
"oratest", 5521, "thin");
/* ユーザー名 / パスワードを使用して、トピック接続を作成します。*/
TopicConnection tc_conn = tc_fact.createTopicConnection("jmsuser", "jmsuser");
```

# 既存の JDBC 接続でのトピック接続の確立

図 15-4 ユースケース図：既存の JDBC 接続でのトピック接続の確立（パブリッシュ / サブスクライプ）



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

既存の JDBC 接続で、トピック接続を作成します。

## 使用上の注意

ありません。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsTopicConnectionFactory」の createTopicConnection を参照してください。

## 例

### 例 1

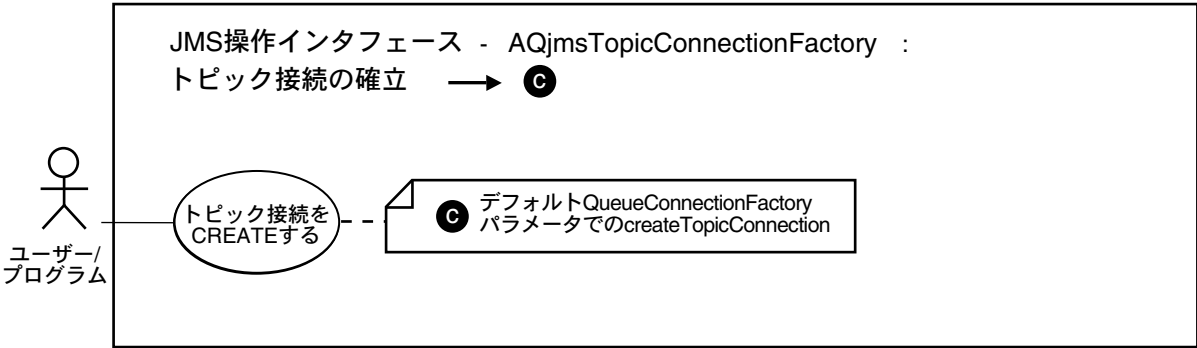
```
Connection db_conn;    /* 事前にオープンされた JDBC 接続 */
TopicConnection tc_conn = AQjmsTopicConnectionFactory.createTopicConnection(db_
conn);
```

### 例 2

```
OracleDriver ora = new OracleDriver();
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(ora.defaultConnection());
```

# デフォルトのコネクション・ファクトリ・パラメータでのトピック接続の確立

図 15-5 ユースケース図：デフォルトのコネクション・ファクトリ・パラメータでのトピック接続の確立（パブリッシュ/サブスクライブ）



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

デフォルトのコネクション・ファクトリ・パラメータで、トピック接続を確立します。

## 使用上の注意

ありません。

## 構文

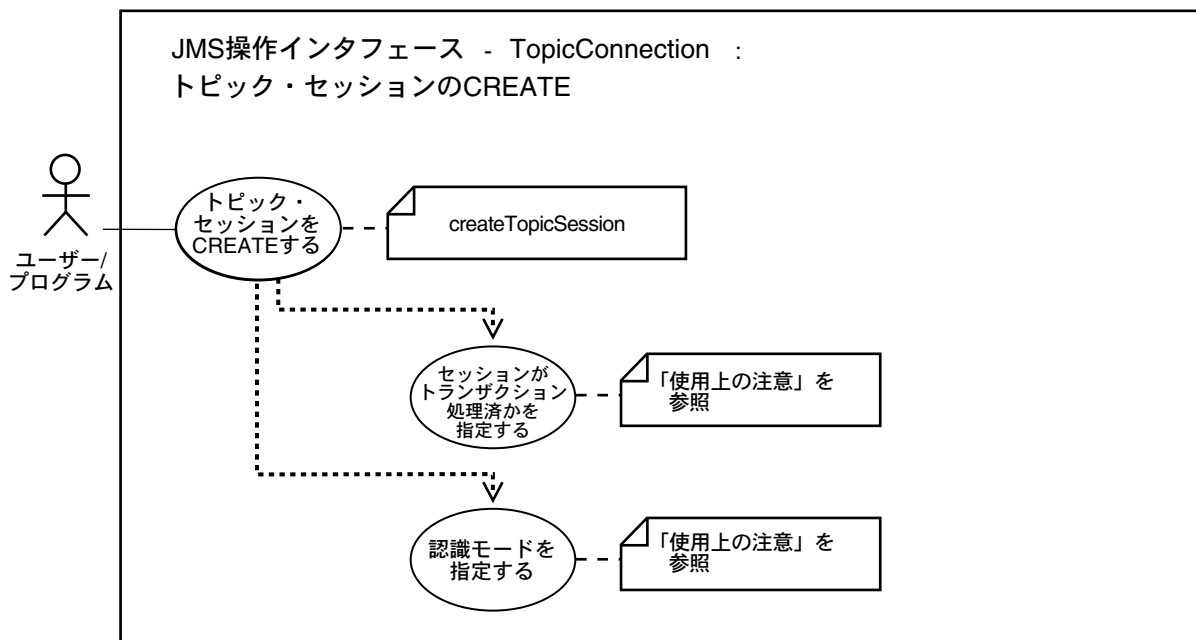
Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsTopicConnectionFactory」の createTopicConnection を参照してください。

## 例

例はありません。

## トピック・セッションの作成

図 15-6 ユースケース図：トピック・セッションの作成（パブリッシュ/サブスクライブ）



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

トピック・セッションを作成します。

## 使用上の注意

ありません。

## 構文

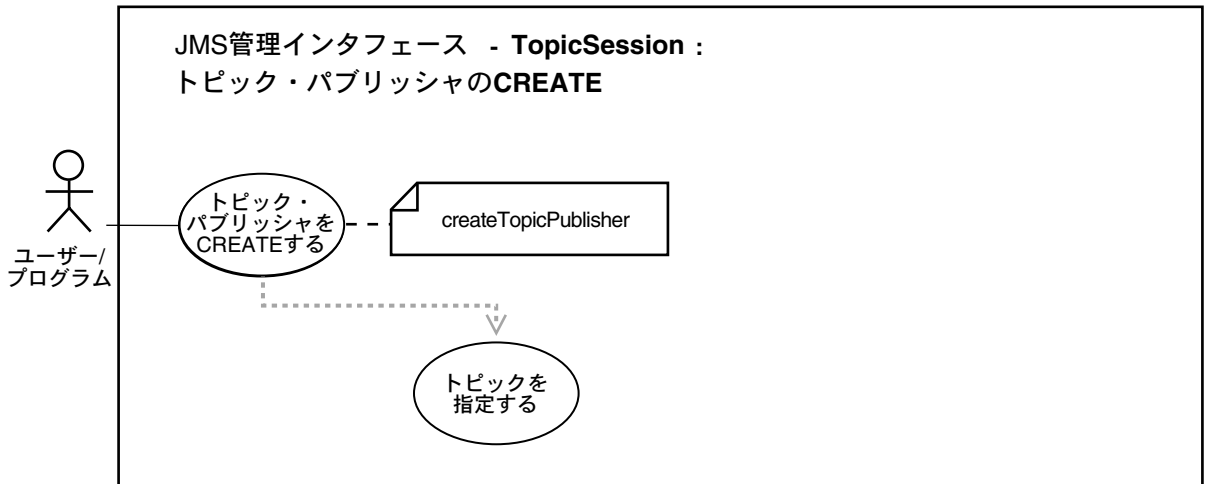
Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsConnection」の createTopicSession を参照してください。

## 例

```
TopicConnection tc_conn;  
TopicSession t_sess = tc_conn.createTopicSession(true,0);
```

## トピック・パブリッシャの作成

図 15-7 ユースケース図：トピック・パブリッシャの作成（パブリッシュ / サブスクライブ）



---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース - 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 

## 用途

トピック・パブリッシャを作成します。

## 使用上の注意

ありません。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の createPublisher を参照してください。

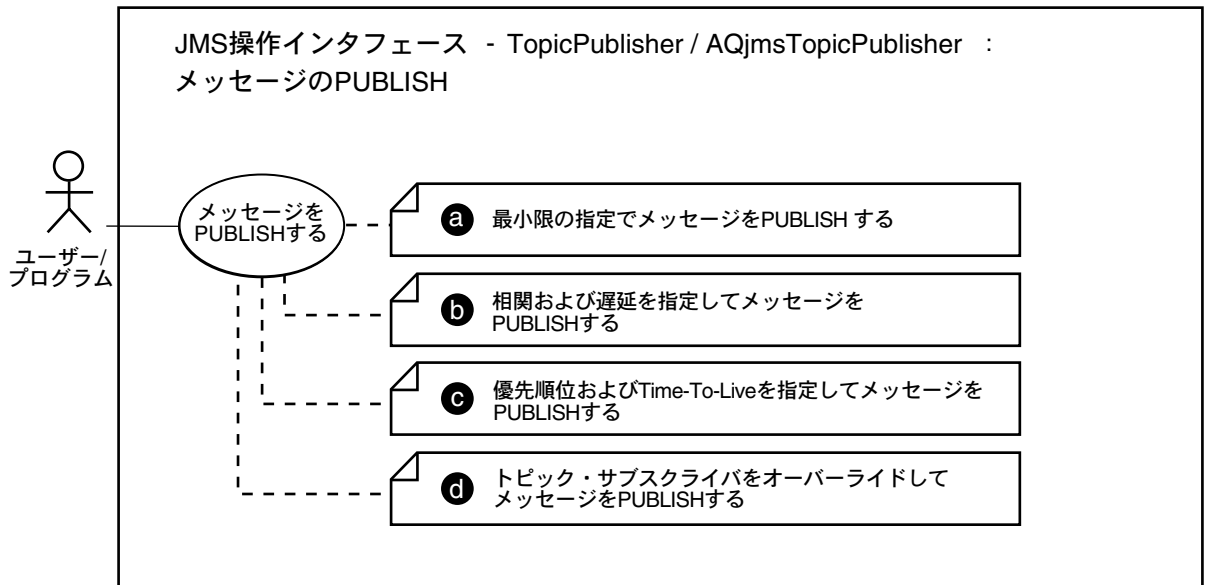
## 例

今回のリリースでは、例は記載していません。



## トピック・パブリッシャを使用してメッセージを公開する 4 つの方法

図 15-8 ユースケース図：トピック・パブリッシャを使用してメッセージを公開する 4 つの方法（パブリッシュ / サブスクライブ）



### 参照：

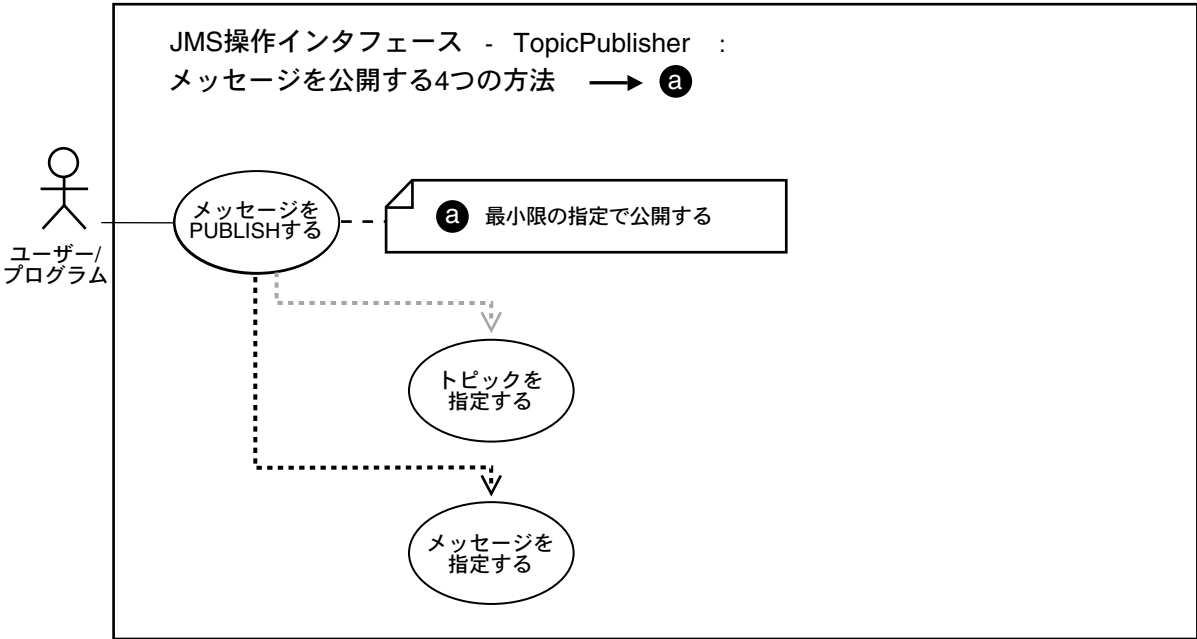
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、トピック・パブリッシャを使用してメッセージを公開する 4 つの方法について説明します。

- a. 15-16 ページの「[最小限の指定でのメッセージの公開](#)」
- b. 15-19 ページの「[相関および遅延を指定したメッセージの公開](#)」
- c. 15-22 ページの「[優先順位および Time-To-Live を指定したメッセージの公開](#)」
- d. 15-25 ページの「[トピック・サブスクライバをオーバーライドする受信者リストを指定したメッセージの公開](#)」

# 最小限の指定でのメッセージの公開

図 15-9 ユースケース図：最小限の指定でのメッセージの公開（パブリッシュ / サブスクライブ）



- 参照：
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

最小限の指定でメッセージを公開します。

## 使用上の注意

トピック・パブリッシャがデフォルト・トピックで作成されている場合、必ずしも publish コールにトピック・パラメータを指定する必要はありません。トピックが送信操作で指定されている場合、その値はトピック・パブリッシャのデフォルト・トピックをオーバーライドします。トピック・パブリッシャがデフォルト・トピックを使用しないで作成されている場合、publish コールごとにトピックを指定する必要があります。トピック・パブリッシャでは、メッセージの優先順位 (1) および Time-To-Live (無制限) のデフォルト値が使用されます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsTopicPublisher」の publish を参照してください。

## 例

Example 1 - publish specifying topic

```

TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              jms_sess;
TopicPublisher             publisher1;
Topic                     shipped_orders;
int                        myport = 5521;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory('MYHOSTNAME',
                                                    'MYSID', myport, 'oci8');
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

/* トピック・パブリッシャを作成します。*/
publisher1 = jms_sess.createPublisher(null);

/* トピック・オブジェクトを取得します。*/
shipped_orders = ((AQjmsSession) jms_sess).getTopic('WS', 'Shipped_Orders_Topic');

/* テキスト・メッセージを作成します。*/
TextMessage text_message = jms_sess.createTextMessage();

/* トピックを指定して公開します。*/
publisher1.publish(shipped_orders, text_message);

```

Example 2 - publish without specifying topic

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              jms_sess;
TopicPublisher            publisher1;
Topic                     shipped_orders;
int                        myport = 5521;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jms_topic", "jms_topic");

/* トピック・セッションを作成します。*/
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

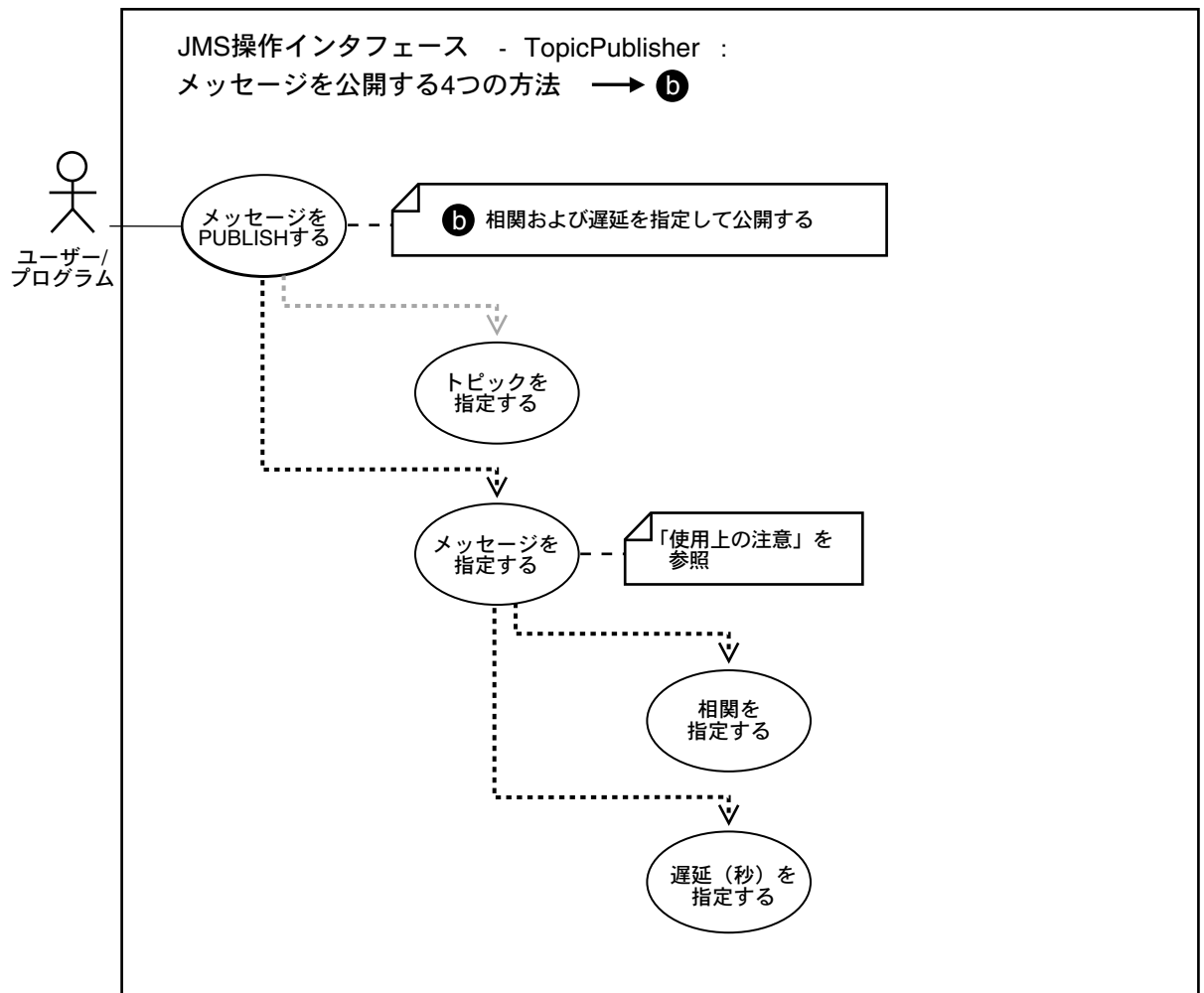
/* 出荷済注文トピックを取得します。*/
shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);

/* テキスト・メッセージを作成します。*/
TextMessage    jms_sess.createTextMessage();

/* トピックを指定せずに公開します。*/
publisher1.publish(text_message);
```

## 相関および遅延を指定したメッセージの公開

図 15-10 ユースケース図：相関および遅延を指定したメッセージの公開（パブリッシュ / サブスクライブ）



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

相関および遅延を指定して、メッセージを公開します。

## 使用上の注意

パブリッシャは、公開前に遅延や相関などのメッセージ・プロパティを設定できます。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsTopicPublisher」の publish を参照してください。

## 例

Example 1 - publish specifying delay, correlation

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection          t_conn     = null;
TopicSession              jms_sess;
TopicPublisher            publisher1;
Topic                     shipped_orders;
int                        myport    = 5521;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jms_topic", "jms_topic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);

/* テキスト・メッセージを作成します。*/
TextMessage    jms_sess.createTextMessage();
```

```

/* 相関および遅延を指定します。*/

/* 相関を指定します。*/
jms_sess.setJMSCorrelationID("FOO");

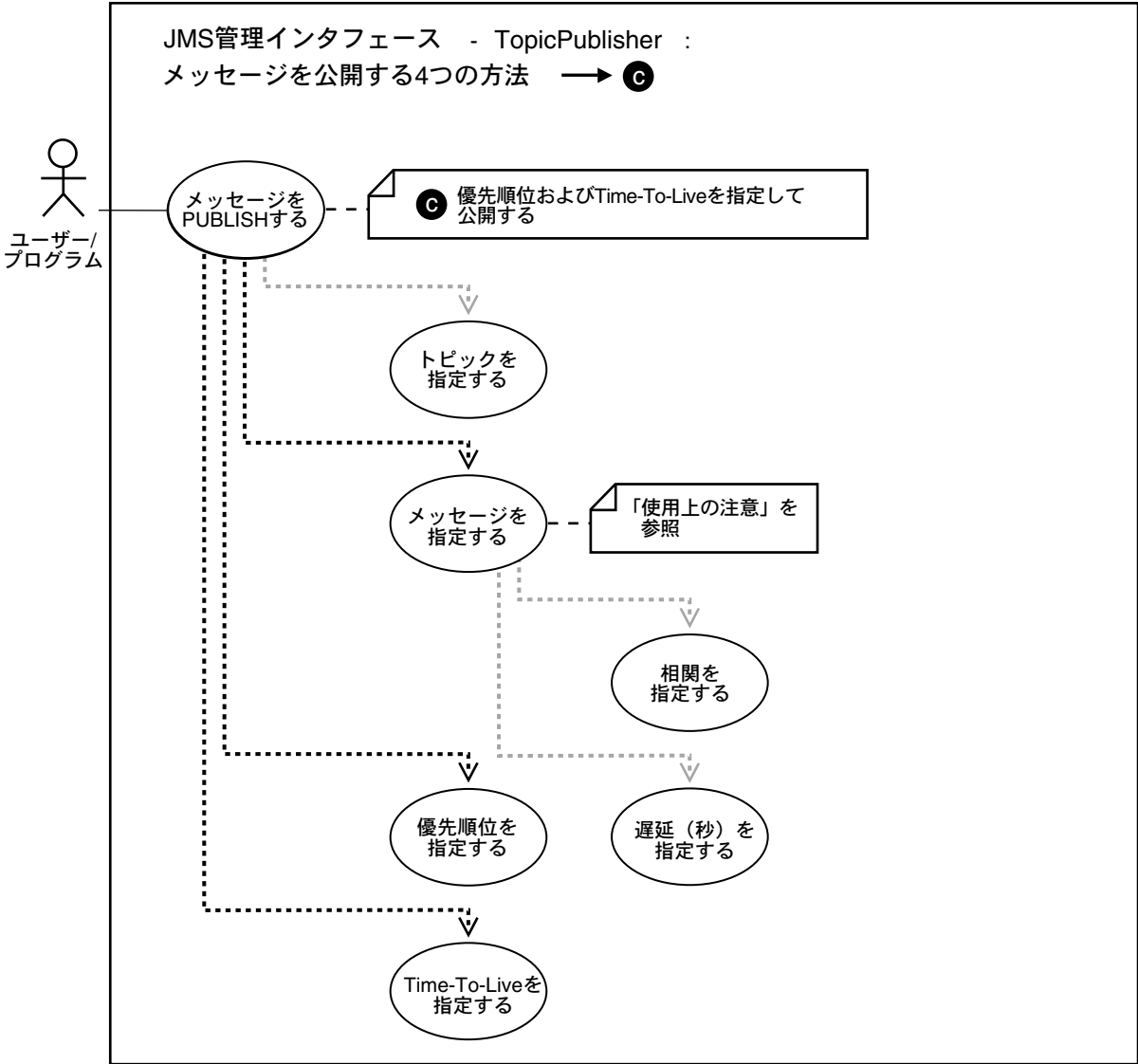
/* 遅延を 30 秒に指定します。*/
jms_sess.setLongProperty("JMS_OracleDelay", 30);

/* 公開します。*/
publisher1.publish(text_message);

```

# 優先順位および Time-To-Live を指定したメッセージの公開

図 15-11 ユースケース図：優先順位および Time-To-Live を指定したメッセージの公開  
(パブリッシュ / サブスクライブ)





---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

優先順位および Time-To-Live を指定して、メッセージを公開します。

## 使用上の注意

メッセージの優先順位および Time-To-Live は、publish コールで指定できます。このリリースでは、配信モード PERSISTENT のみがサポートされています。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsTopicPublisher」の publish を参照してください。

## 例

Example 1 - publish specifying priority, timeToLive

```
TopicConnectionFactory  tc_fact  = null;
TopicConnection        t_conn   = null;
TopicSession           jms_sess;
TopicPublisher         publisher1;
Topic                  shipped_orders;
int                     myport = 5521;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);

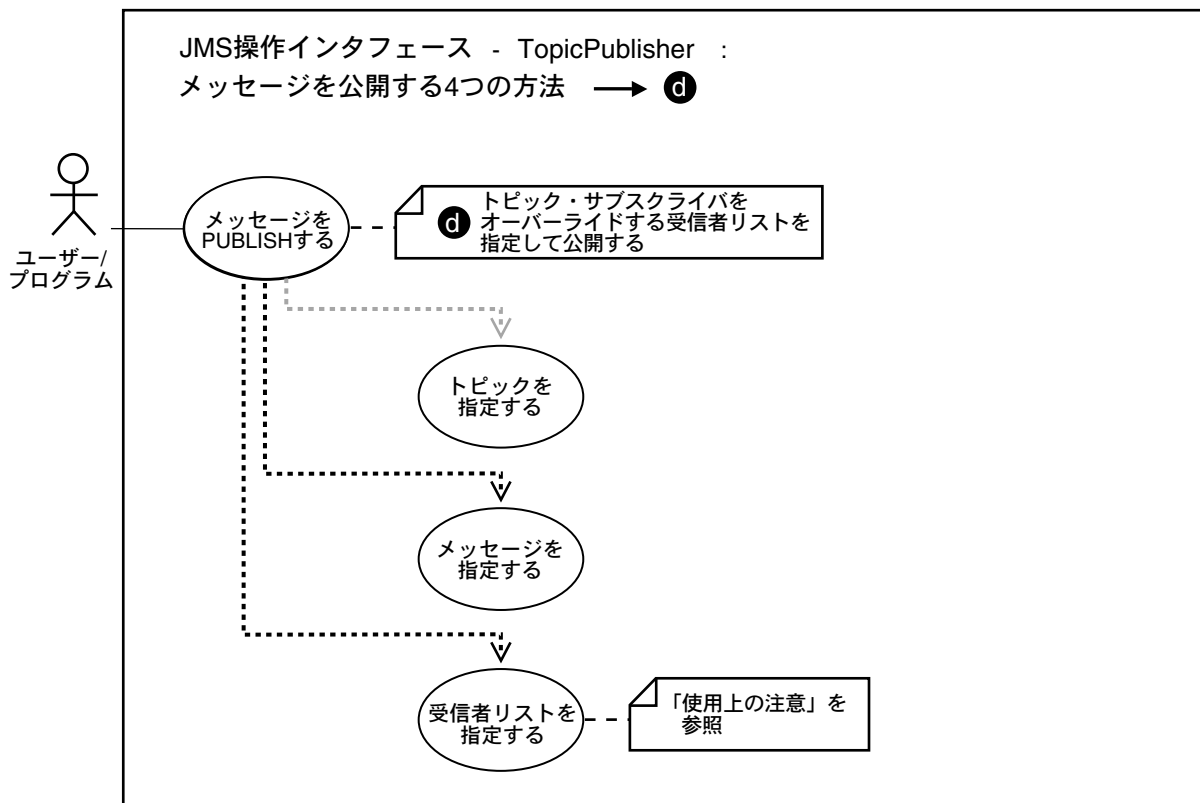
/* テキスト・メッセージを作成します。*/
```

```
TextMessage    jms_sess.createTextMessage();

/* 優先順位が1、Time-To-Live が200 秒でメッセージを公開します。*/
publisher1.publish(text_message, DeliveryMode.PERSISTENT, 1, 200000);
```

## トピック・サブスクライバをオーバーライドする受信者リストを指定したメッセージの公開

図 15-12 ユースケース図：トピック・サブスクライバをオーバーライドする受信者リストを指定したメッセージの公開（パブリッシュ/サブスクライブ）



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

トピック・サブスクライバをオーバーライドする受信者リストを指定して、メッセージを公開します。

## 使用上の注意

トピックのサブスクリプション・リストは、publish コールで受信者リストを指定することによって、オーバーライドできます。

## 構文

Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsTopicPublisher」の publish を参照してください。

## 例

Example 1 - publish specifying priority, timeToLive

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection          t_conn     = null;
TopicSession             jms_sess;
TopicPublisher            publisher1;
Topic                    shipped_orders;
int                       myport = 5521;
AQjmsAgent []            recipList;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);
```

```
/* テキスト・メッセージを作成します。*/
TextMessage      jms_sess.createTextMessage();

/* 2 人の受信者を作成します。*/
recipList = new AQjmsAgent[2];

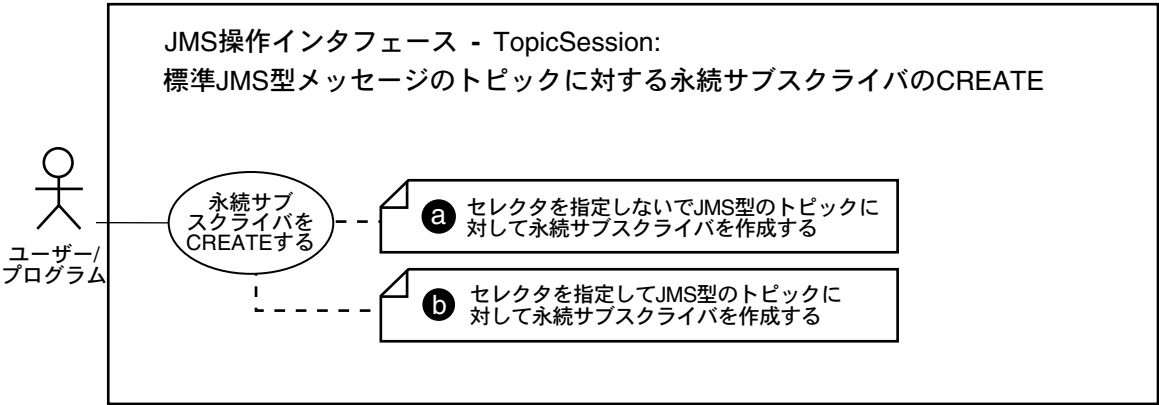
recipList[0] = new AQjmsAgent("ES", "ES.shipped_orders_topic",
                              AQAgent.DEFAULT_AGENT_PROTOCOL);

recipList[1] = new AQjmsAgent("WS", "WS.shipped_orders_topic",
                              AQAgent.DEFAULT_AGENT_PROTOCOL);

/* 受信者リストを指定してメッセージを公開します。*/
publisher1.publish(text_message, recipList);
```

# 標準 JMS 型メッセージのトピックに対して永続サブスクライバを作成する 2 つの方法

図 15-13 ユースケース図：標準 JMS 型メッセージのトピックに対して永続サブスクライバを作成する 2 つの方法（パブリッシュ / サブスクライブ）



**参照：**

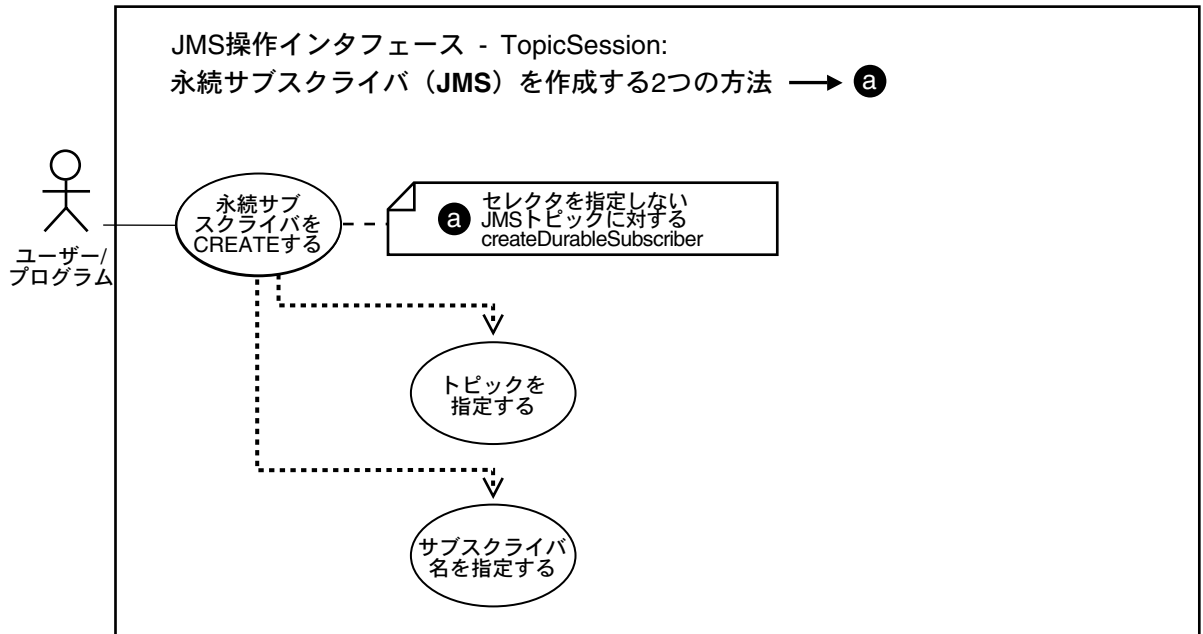
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、標準 JMS 型メッセージのトピックに対して永続サブスクライバを作成する 2 つの方法について説明します。

- a. 15-29 ページの「セレクトタを指定しない JMS トピックに対する永続サブスクライバの作成」
- b. 15-31 ページの「セレクトタを指定しての JMS トピックに対する永続サブスクライバの作成」

## セレクタを指定しない JMS トピックに対する永続サブスクライバの作成

図 15-14 ユースケース図：セレクタを指定しない JMS トピックに対する永続サブスクライバの作成（パブリッシュ / サブスクライブ）



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

セレクタを指定しないで、JMS トピックに対する永続サブスクライバを作成します。

## 使用上の注意

永続サブスクライバを作成するには、サブスクライバ名および JMS トピックを指定する必要があります。トピックのサブスクリプションを終了するには、unsubscribe コールが必要です。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の CreateDurableSubscriber を参照してください。

## 例

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection          t_conn     = null;
TopicSession             jms_sess;
TopicSubscriber          subscriber1;
Topic                    shipped_orders;
int                       myport = 5521;
AQjmsAgent []            recipList;

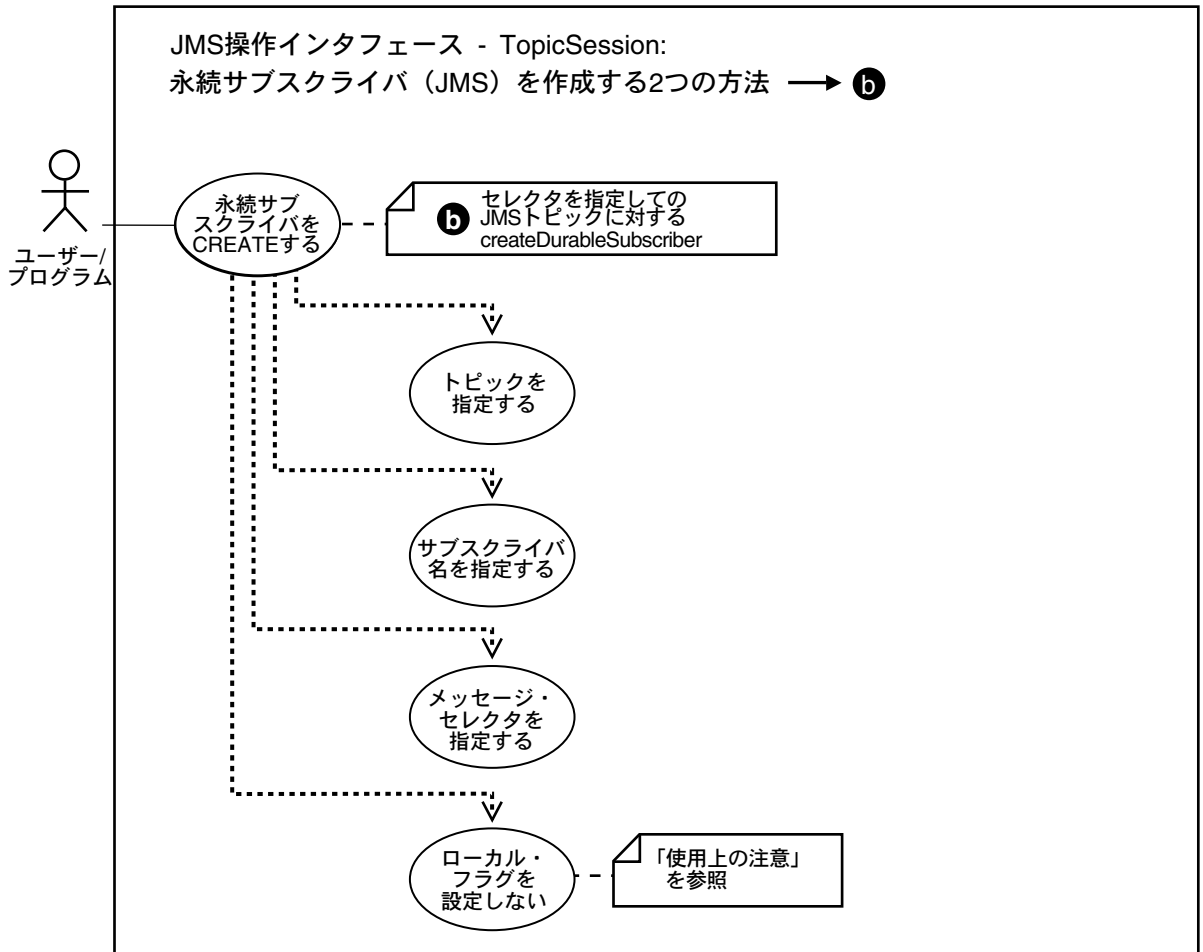
/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");
/* shipped_orders トピックに永続サブスクライバを作成します。*/
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders, 'WesternShipping');
```



# セクタを指定しての JMS トピックに対する永続サブスクライバの作成

図 15-15 ユースケース図：セクタを指定しての JMS トピックに対する永続サブスクライバの作成（パブリッシュ / サブスクライブ）



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

セレクトを指定して、JMS トピックに対する永続サブスクライバを作成します。

## 使用上の注意

クライアントは、サブスクライバ名および JMS トピックを指定することによって、永続サブスクライバを作成します。オプションで、メッセージ・セレクトを指定できます。メッセージ・セレクト式と一致したプロパティを持つメッセージのみ、サブスクライバに配信されます。セレクト値は NULL である場合があります。セレクトには、次のうち 1 つ以上が組み合わされた SQL92 式を含めることができます。

- JMS メッセージ・ヘッダー・フィールドまたはプロパティ: JMSPriority (Int)、JMSCorrelationID (String)、JMSType (String)、JMSXUserID (String)、JMSXAppID (String)、JMSXGroupID (String)、JMSXGroupSeq (Int)

使用例を次に示します。

```
JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
```

- ユーザー定義のメッセージ・プロパティ

使用例を次に示します。

```
color IN ('RED', 'BLUE', 'GREEN') AND price < 30000
```

使用可能な演算子は次のとおりです。

- 優先順位 NOT、AND、OR の論理演算子
- 比較演算子 =、>、>=、<、<=、<>、! (<> および ! は、不等号として使用できます)
- 優先順位 +、- unary、\*、/、+、- の算術演算子
- 識別子 [NOT] IN (文字リテラル 1, 文字リテラル 2, ..)
- 算術式 1 [NOT] BETWEEN 算術式 2 and 算術式 3
- 識別子 [NOT] LIKE パターン値 [ESCAPE エスケープ文字]
- パターン値は文字列リテラルで、% は連続するすべての文字を表します。

- \_ はすべての単一文字を表します。
- オプションのエスケープ文字は、パターン値内の「\_」および「%」を文字列として処理する場合に使用します。
- 識別子 IS [NOT] NULL

クライアントは、同一の名前および異なるメッセージ・セレクトタで永続トピック・サブスクライバを作成することによって、既存の永続サブスクリプションを変更できます。トピックのサブスクリプションを終了するには、unsubscribe コールが必要です。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsTopicPublisher」の publish を参照してください。

## 例

Example 1 - subscribe specifying selector

```

TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              jms_sess;
TopicSubscriber           subscriber1;
Topic                     shipped_orders;
int                        myport = 5521;
AQjmsAgent []             recipList;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

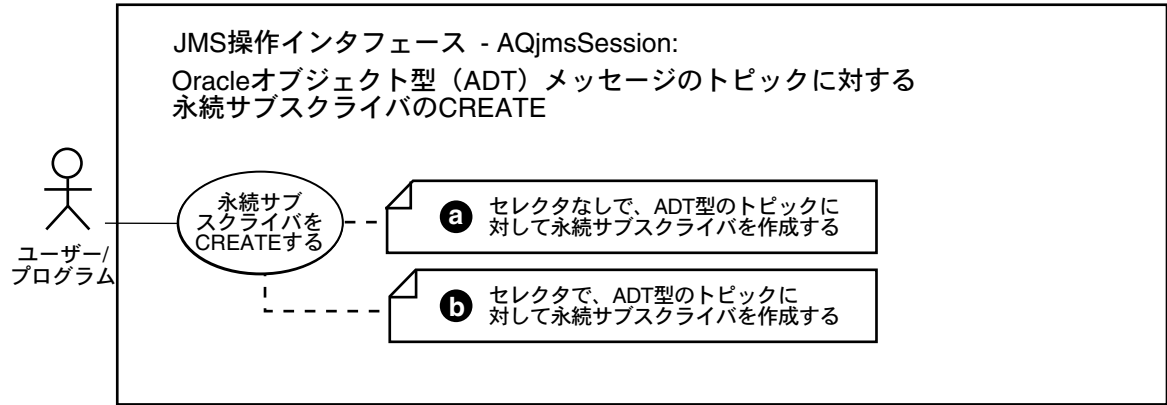
shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");

/* サブスクライバを作成します。*/
/* JMSPriority およびユーザー・プロパティの 'Region' に基づく条件付きです。*/
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders, 'WesternShipping',
                                                "JMSPriority > 2 and Region like 'Western%'",
                                                false);

```

# Oracle オブジェクト型 (ADT) メッセージのトピックに対する永続サブスクライバを作成する 2 つの方法

図 15-16 ユースケース図 : Oracle オブジェクト型 (ADT) メッセージのトピックに対する永続サブスクライバを作成する 2 つの方法 (パブリッシュ / サブスクライブ)



**参照 :**

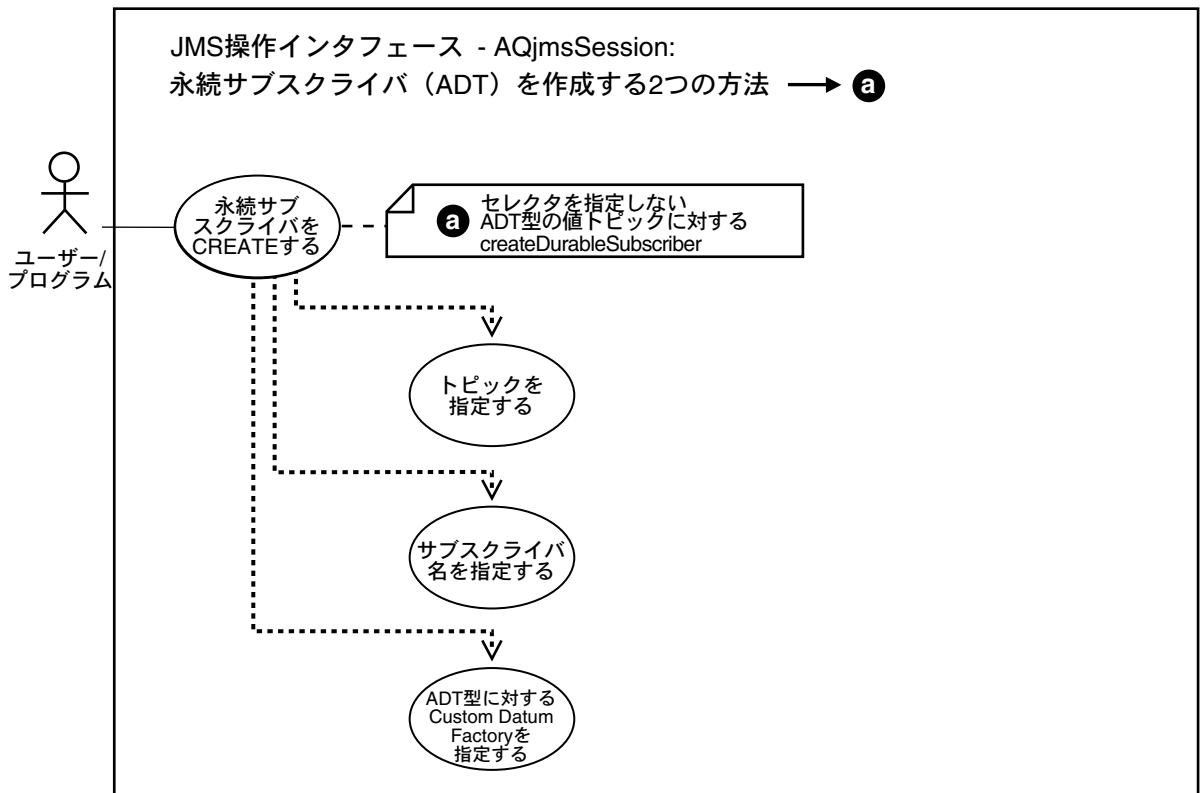
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、Oracle オブジェクト型 (ADT) メッセージに対する永続サブスクライバを作成する 2 つの方法について説明します。

- a. 15-35 ページの「セレクトタを指定しない JMS トピックに対する永続サブスクライバの作成」
- b. 15-37 ページの「セレクトタを指定しての JMS トピックに対する永続サブスクライバの作成」

# セクタを指定しない JMS トピックに対する永続サブスクライバの作成

図 15-17 ユースケース図：セクタを指定しない JMS トピックに対する永続サブスクライバの作成（パブリッシュ / サブスクライブ）



## 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

セクタを指定しないで、トピックに対する永続サブスクライバを作成します。

## 使用上の注意

Oracle オブジェクト型のトピックに対して永続サブスクライバを作成するために、クライアントは、トピックおよびサブスクライバ名の他に、Oracle オブジェクト型に対する CustomDatumFactory を指定する必要があります。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createDurableSubscriber を参照してください。

## 例

```
subscribe to an ADT queue

TopicConnectionFactory    tc_fact    = null;
TopicConnection          t_conn    = null;
TopicSession             t_sess    = null;
TopicSession             jms_sess;
TopicSubscriber          subscriber1;
Topic                   shipped_orders;
int                      myport = 5521;
AQjmsAgent[]             recipList;
/* JPublisher で作成された oracle オブジェクト型の javam アップリング */
ADTMessage               message;

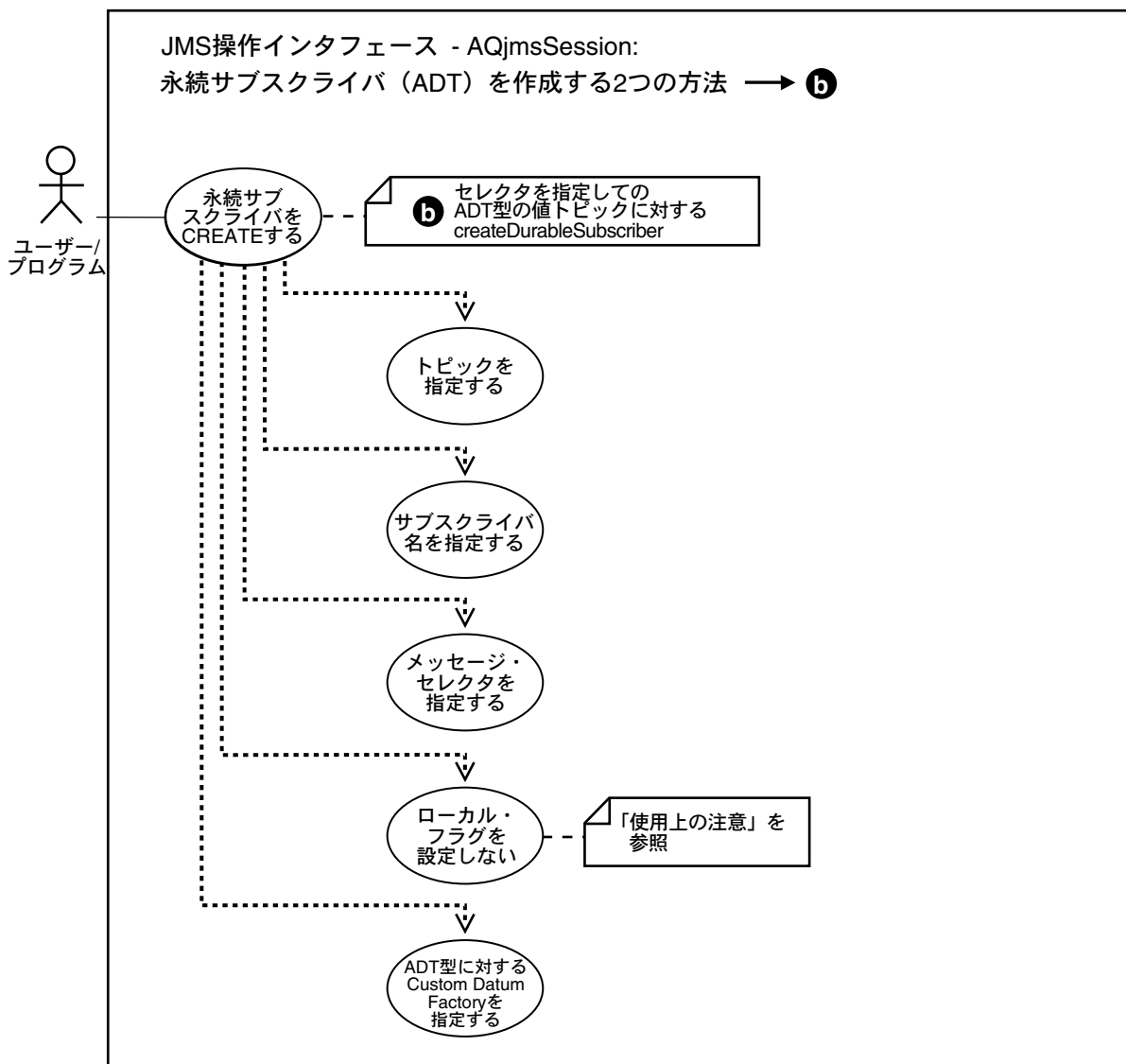
/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");

/* 正確な CustomDatumFactory を指定して、サブスクライバを作成します。*/
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders, 'WesternShipping',
                                                AQjmsAgent.getFactory());
```

# セクタを指定しての JMS トピックに対する永続サブスクライバの作成

図 15-18 ユースケース図：セクタを指定しての JMS トピックに対する永続サブスクライバの作成（パブリッシュ/サブスクライブ）



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

セレクトタを指定して、トピックに対する永続サブスクライバを作成します。

## 使用上の注意

Oracle オブジェクト型のトピックに対して永続サブスクライバを作成するために、クライアントは、トピックおよびサブスクライバ名の他に、Oracle オブジェクト型に対する CustomDatumFactory を指定する必要があります。

オプションで、メッセージ・セレクトタを指定できます。セレクトタと一致したメッセージのみ、サブスクライバに配信されます。

ADT メッセージには、ユーザー定義のプロパティが含まれません。ただし、セレクトタを使用して、優先順位、相関識別子またはメッセージ・ペイロードの属性値に基づいて、メッセージを選択できます。

ADT メッセージを含むキューに対するセレクトタの構文は、標準 JMS ペイロード（テキスト、ストリーム、バイト、マップ）を含むキューに対するセレクトタの構文とは異なります。

セレクトタは、AQ ルールの構文に似ています。

- a. 優先順位または相関識別子のセレクトタは、次のように指定されます。

```
priority > 3 AND corr_id = 'Fiction'
```

- b. メッセージ・ペイロードのセレクトタは、次のように指定されます。属性名には、接頭辞 tab.user\_data を付ける必要があります。

```
tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
```



## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createDurableSubscriber を参照してください。

## 例

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection          t_conn     = null;
TopicSession             jms_sess;
TopicSubscriber          subscriber1;
Topic                    shipped_orders;
int                       myport = 5521;
AQjmsAgent []            recipList;
/* JPublisher で作成された oracle オブジェクト型の java マッピング */
ADTMessage                message;

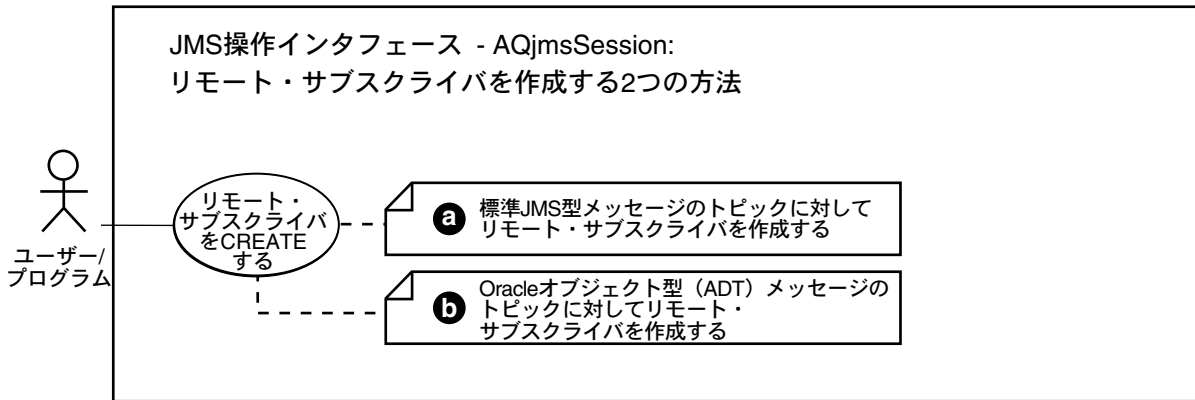
/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");

/* 正確な CustomDatumFactory およびセレクタを指定して、サブスクライバを作成します。*/
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders, "WesternShipping",
                                                " priority > 1 and tab.user_data.region like 'WESTERN %'",
                                                false, AQjmsAgent.getFactory());
```

## リモート・サブスクライバを作成する2つの方法

図 15-19 ユースケース図：リモート・サブスクライバを作成する2つの方法（パブリッシュ / サブスクライブ）



---

### 参照：

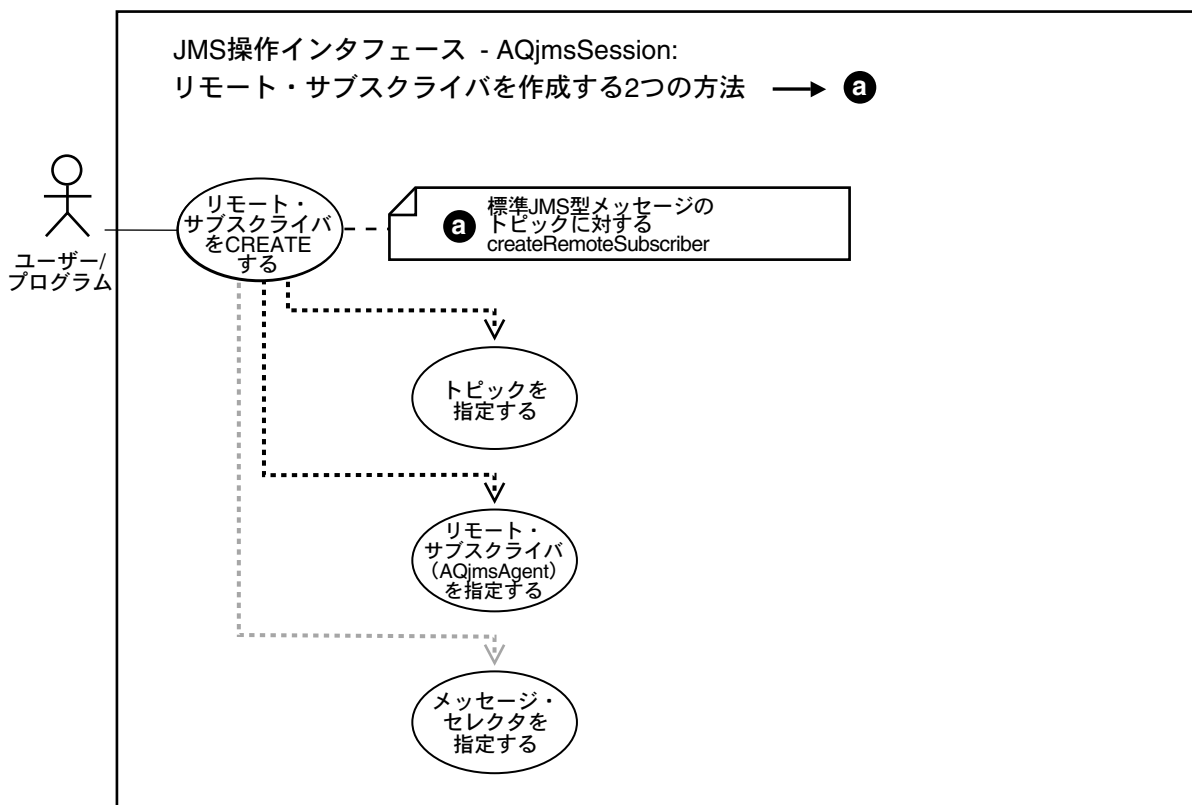
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

ここでは、標準 JMS 型メッセージのトピックに対するサブスクライバを作成する 2 つの方法について説明します。

- 15-41 ページの「JMS メッセージのトピックに対するリモート・サブスクライバの作成」
- 15-44 ページの「Oracle オブジェクト型（ADT）メッセージのトピックに対するリモート・サブスクライバの作成」

## JMS メッセージのトピックに対するリモート・サブスクライバの作成

図 15-20 ユースケース図：JMS メッセージのトピックに対するリモート・サブスクライバの作成  
(パブリッシュ/サブスクライブ)



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

セレクトなしで、JMS メッセージのトピックに対するリモート・サブスクライバを作成します。

## 使用上の注意

AQ では、トピックがリモート・サブスクライバ（同一または異なるデータベース内の他のトピックのサブスクライバ）を持つことができます。リモート・サブスクライバを使用するには、ローカル・トピックとリモート・トピック間の伝播を設定する必要があります。

リモート・サブスクライバは、リモート・トピックの特定のコンシューマまたはリモート・トピックのすべてのサブスクライバである場合があります。リモート・サブスクライバは、AQjmsAgent 構造を使用して定義されます。AQjmsAgent は、名前およびアドレスで構成されます。名前は、リモート・トピックのコンシューマ名を参照します。アドレスは、(schema).(topic\_name) [@dblink] 構文のリモート・トピックを参照します。

- a. メッセージをリモート・トピックの特定のコンシューマに公開するには、リモート・トピックの受信者のサブスクリプション名を AQjmsAgent の name フィールドに指定する必要があります。リモート・トピックは、AQjmsAgent の address フィールドに指定する必要があります。
- b. メッセージをリモート・トピックのすべてのサブスクライバに公開するには、AQjmsAgent の name フィールドを NULL に設定する必要があります。リモート・トピックは、AQjmsAgent の address フィールドに指定する必要があります。

また、メッセージ・セレクトも指定できます。セレクトを満たすメッセージのみ、リモート・サブスクライバに配信されます。メッセージ・セレクトは NULL である可能性があります。セレクトに対する構文は、createDurableSubscriber の構文と同じです。メッセージ・セレクトには NULL が許可されています。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createRemoteSubscriber を参照してください。

## 例

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection          t_conn     = null;
TopicSession             t_sess     = null;
TopicSession             jms_sess;
TopicSubscriber          subscriber1;
Topic                   shipped_orders;
int                      my[port = 5521;
AQjmsAgent               remoteAgent;
```

```
/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                  "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

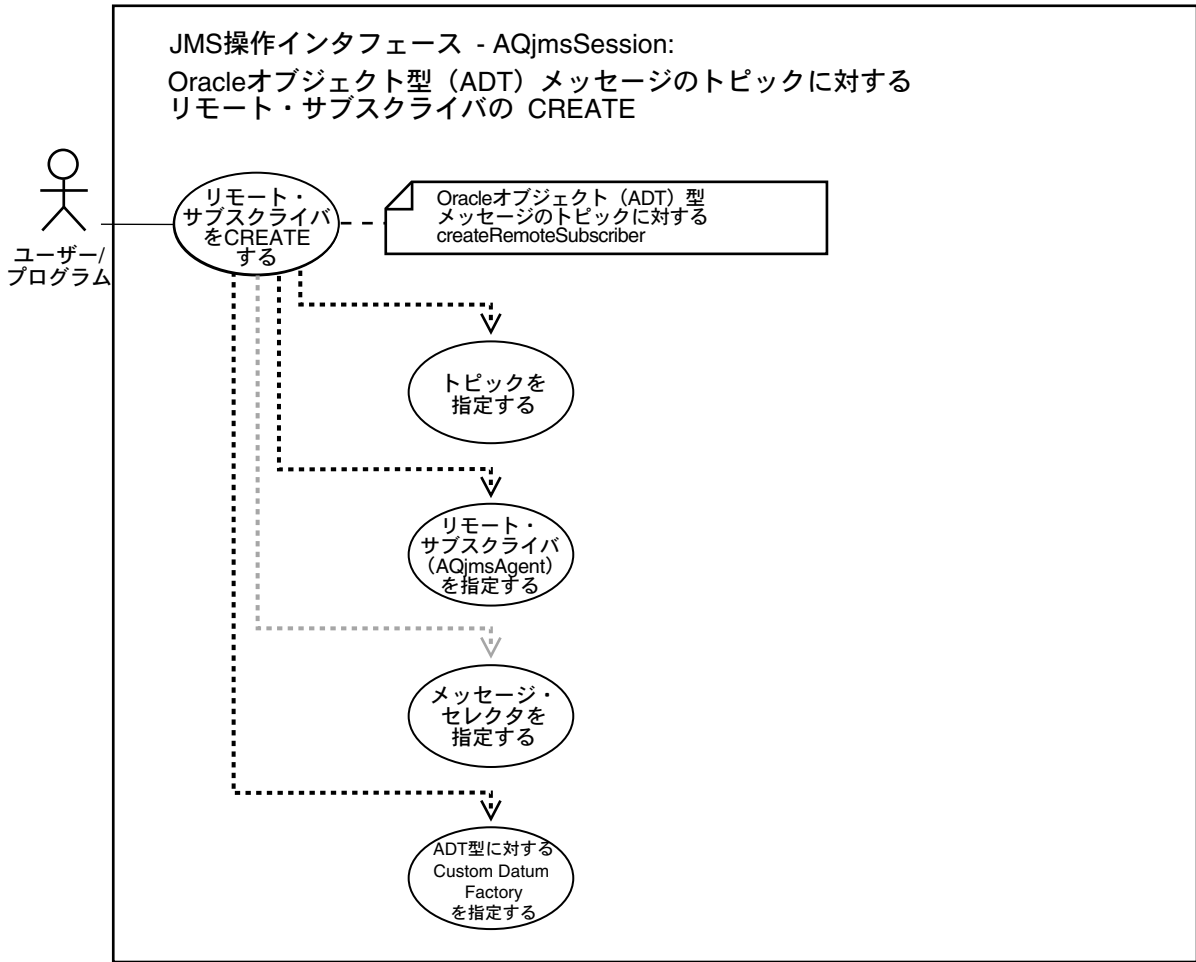
shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");

remoteAgent = new AQjmsAgent("WesternRegion", "WS.shipped_orders_topic", null);

/* リモート・サブスクライバを作成します（セレクトはNULLです）。*/
subscriber1 = ((AQjmsSession) jms_sess).createRemoteSubscriber(shipped_orders,
                                                                remoteAgent, null);
```

# Oracle オブジェクト型 (ADT) メッセージのトピックに対する リモート・サブスクライバの作成

図 15-21 ユースケース図：Oracle オブジェクト型 (ADT) メッセージのトピックに対する  
リモート・サブスクライバの作成 (パブリッシュ / サブスクライブ)



---

**参照:**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

Oracle オブジェクト型 (ADT) メッセージのトピックに対するリモート・サブスクライバを作成します。

## 使用上の注意

AQ では、トピックがリモート・サブスクライバ（同一または異なるデータベース内の他のトピックのサブスクライバ）を持つことができます。リモート・サブスクライバを使用するには、ローカル・トピックとリモート・トピック間の伝播を設定する必要があります。

リモート・サブスクライバは、リモート・トピックの特定のコンシューマまたはリモート・トピックのすべてのサブスクライバである場合があります。リモート・サブスクライバは、AQjmsAgent 構造を使用して定義されます。

AQjmsAgent は、名前およびアドレスで構成されます。名前は、リモート・トピックのコンシューマ名を参照します。アドレスは、(schema).(topic\_name)[@dblink] 構文のリモート・トピックを参照します。

- メッセージをリモート・トピックの特定のコンシューマに公開するには、リモート・トピックの受信者のサブスクリプション名を AQjmsAgent の name フィールドに指定する必要があります。リモート・トピックは、AQjmsAgent の address フィールドに指定する必要があります。
- メッセージをリモート・トピックのすべてのサブスクライバに公開するには、AQjmsAgent の name フィールドを NULL に設定する必要があります。リモート・トピックは、AQjmsAgent の address フィールドに指定する必要があります。

トピックの Oracle オブジェクト型の CustomDatumFactory を指定する必要があります。また、メッセージ・セレクトタも指定できます。セレクトタを満たすメッセージのみ、リモート・サブスクライバに配信されます。メッセージ・セレクトタは NULL である可能性があります。セレクトタに対する構文は、createDurableSubscriber の構文と同じです。メッセージ・セレクトタには NULL が許可されています。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createRemoteSubscriber を参照してください。

## 例

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection          t_conn     = null;
TopicSession             t_sess     = null;
TopicSession             jms_sess;
TopicSubscriber          subscriber1;
Topic                    shipped_orders;
int                       myport = 5521;
AQjmsAgent               remoteAgent;
ADTMessage               message;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                  "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");

/* トピック・セッションを作成します。*/
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

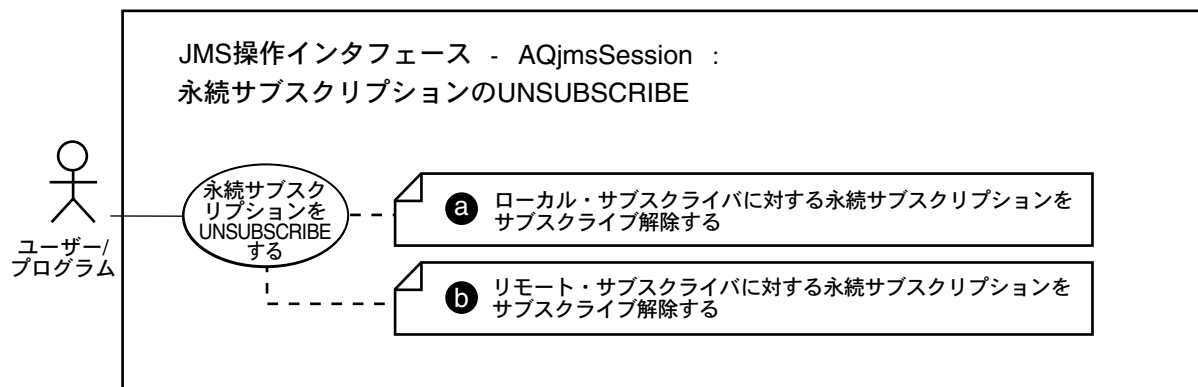
/* 出荷済注文トピックを取得します。*/
shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");
/* リモート・エージェントを作成します。*/
remoteAgent = new AQjmsAgent("WesternRegion", "WS.shipped_orders_topic", null);

/* セレクタを NULL でリモート・サブスクライバを作成します。*/
subscriber1 = ((AQjmsSession) jms_sess).createRemoteSubscriber(shipped_orders,
                                                                remoteAgent, null, message.getFactory());
```



## 永続サブスクリプションのサブスクライブを解除する2つの方法

図 15-22 ユースケース図：永続サブスクリプションのサブスクライブを解除する2つの方法  
(パブリッシュ/サブスクライブ)



### 参照：

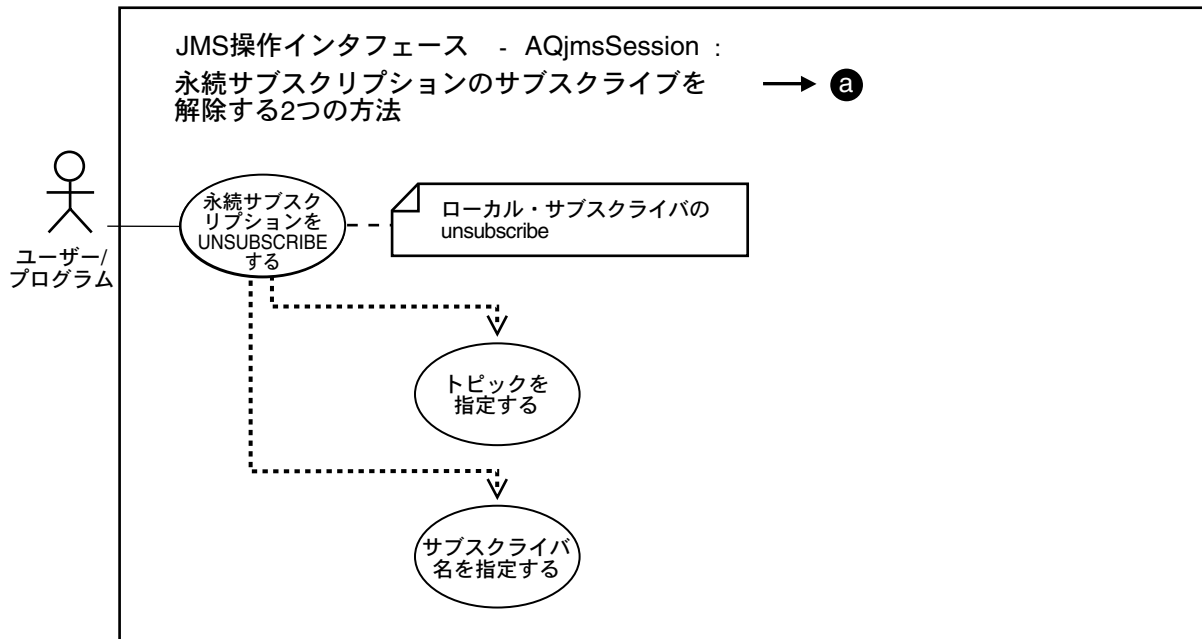
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

ここでは、永続サブスクリプションのサブスクライブを解除する2つの方法について説明します。

- 15-48 ページの「ローカル・サブスクライバに対する永続サブスクリプションのサブスクライブの解除」
- 15-50 ページの「リモート・サブスクライバに対する永続サブスクリプションのサブスクライブの解除」

## ローカル・サブスクライバに対する永続サブスクリプションのサブスクライブの解除

図 15-23 ユースケース図：ローカル・サブスクライバに対する永続サブスクリプションのサブスクライブの解除（パブリッシュ / サブスクライブ）



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

ローカル・サブスクライバに対する永続サブスクリプションのサブスクライブを解除します。

## 使用上の注意

特定のトピックのクライアントによって作成された永続サブスクリプションのサブスクライブを解除します。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の unsubscribe を参照してください。

## 例

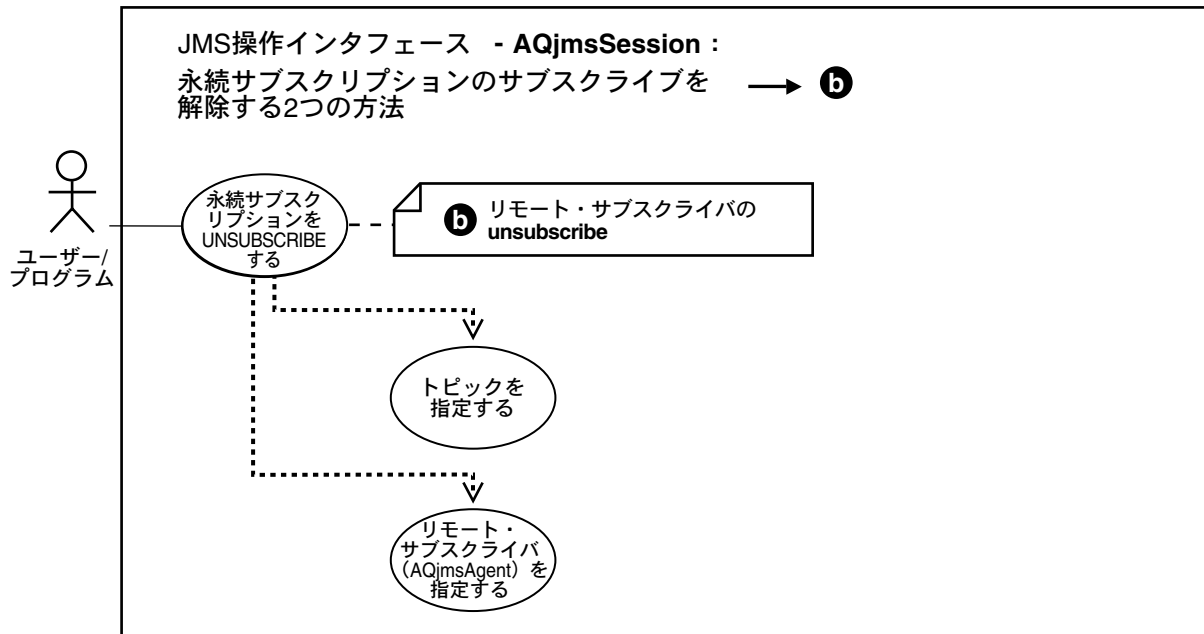
```
TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              jms_sess;
TopicSubscriber           subscriber1;
Topic                     shipped_orders;
int                        myport = 5521;
AQjmsAgent[]              recipList;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");
/* shipped_orders から "WesternShipping" のサブスクライブを解除します。*/
jms_sess.unsubscribe(shipped_orders, "WesternShipping");
```

## リモート・サブスクライバに対する永続サブスクリプションのサブスクライブの解除

図 15-24 ユースケース図：リモート・サブスクライバに対する永続サブスクリプションのサブスクライブの解除（パブリッシュ / サブスクライブ）



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

リモート・サブスクライバに対する永続サブスクリプションのサブスクライブを解除します。

## 使用上の注意

ありません。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の unsubscribe を参照してください。

## 例

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              t_sess     = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                        myport = 5521;
AQjmsAgent                 remoteAgent;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

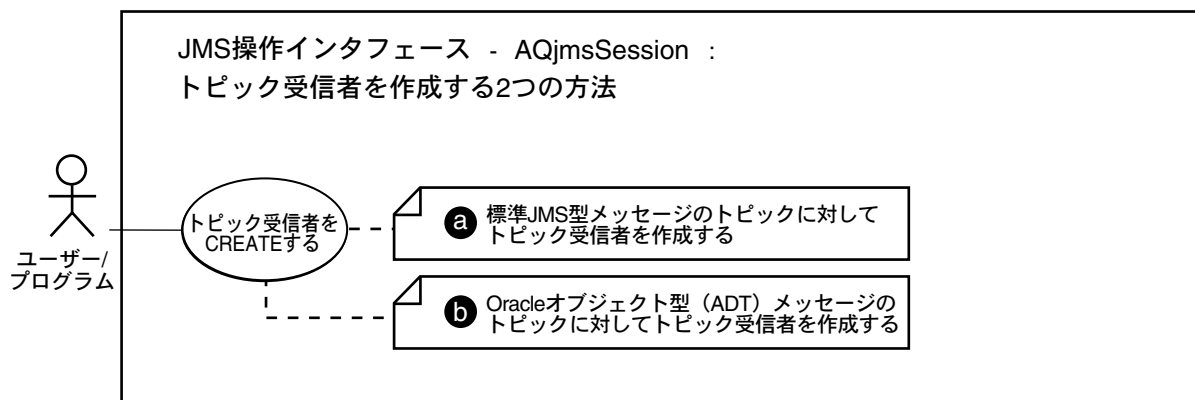
shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE", "Shipped_Orders_Topic");

remoteAgent = new AQjmsAgent("WS", "WS.Shipped_Orders_Topic", null);

/* shipped_orders からリモート・エージェントのサブスクライブを解除します。*/
((AQjmsSession) jms_sess).unsubscribe(shipped_orders, remoteAgent);
```

## トピック受信者を作成する 2 つの方法

図 15-25 ユースケース図：トピック受信者を作成する 2 つの方法（パブリッシュ / サブスクライブ）



---

### 参照：

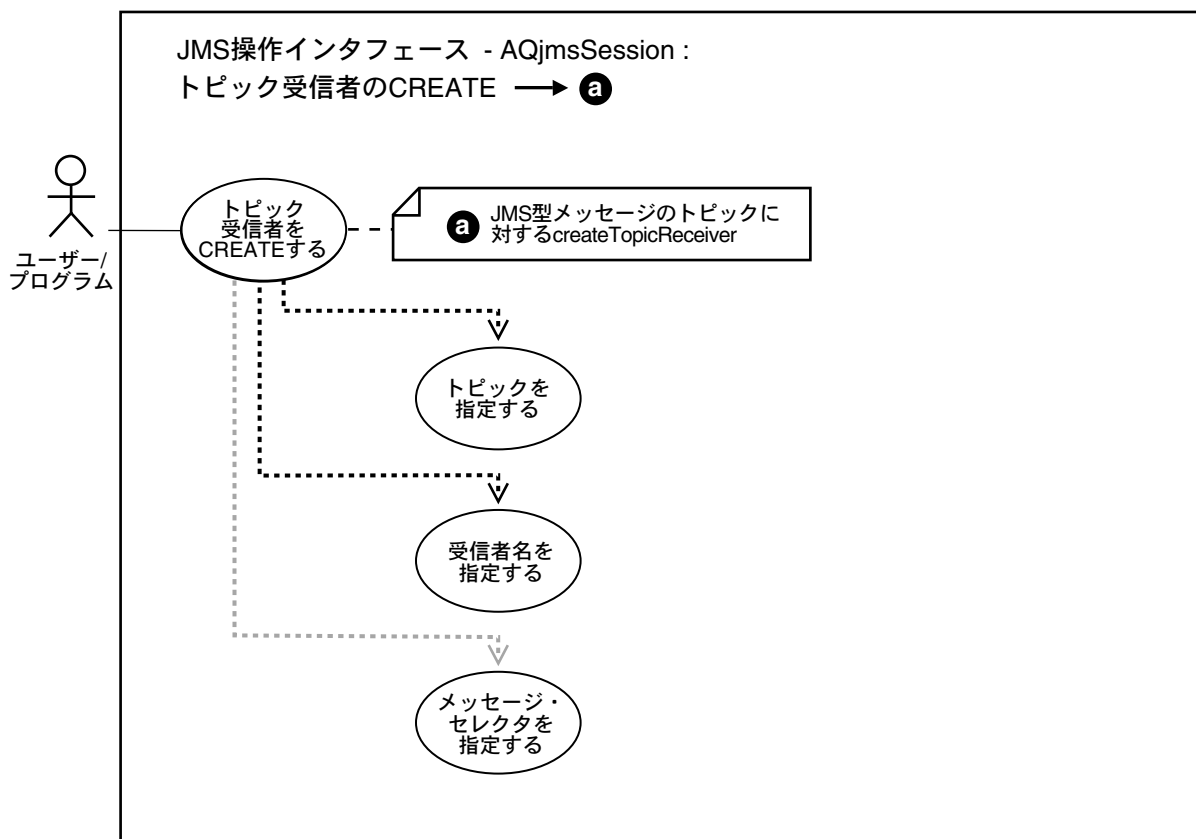
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 

ここでは、トピック受信者を作成する 2 つの方法について説明します。

- 15-53 ページの「[標準 JMS 型メッセージのトピックに対するトピック受信者の作成](#)」
- 15-55 ページの「[Oracle オブジェクト型 \(ADT\) メッセージのトピックに対するトピック受信者の作成](#)」

## 標準 JMS 型メッセージのトピックに対するトピック受信者の作成

図 15-26 ユースケース図：標準 JMS 型メッセージのトピックに対するトピック受信者の作成  
(パブリッシュ / サブスクライブ)



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

標準 JMS 型メッセージのトピックに対するトピック受信者を作成します。

## 使用上の注意

AQ では、特定の受信者にメッセージを送信できます。この受信者は、トピックのサブスクライバである場合とそうでない場合があります。受信者がトピックに対するサブスクライバでない場合、明示的にアドレス指定されているメッセージのみを受信します。

この方法は、永続サブスクライバ以外の顧客に対するトピック受信者オブジェクトの作成に使用する必要があります。メッセージ・セレクトアが指定できます。メッセージ・セレクトアの構文は、キュー受信者の構文と同じです。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の createTopicReceiver を参照してください。

## 例

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              t_sess     = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                        myport = 5521;
TopicReceiver              receiver;
```

```
/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

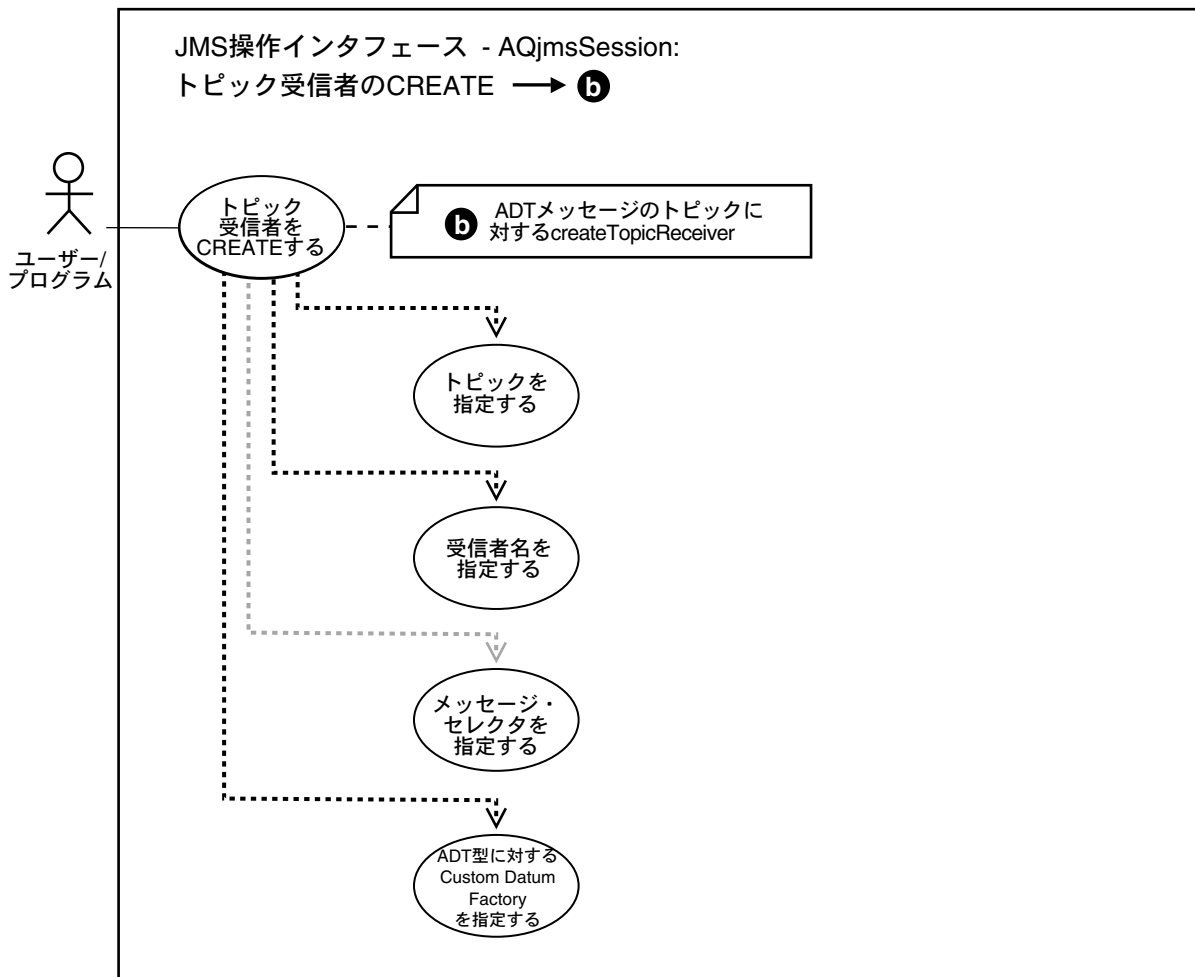
shipped_orders = ((AQjmsSession) jms_sess).getTopic("WS", "Shipped_Orders_Topic");

receiver = ((AQjmsSession) jms_sess).createTopicReceiver(shipped_orders,
"WesternRegion", null);
```



## Oracle オブジェクト型 (ADT) メッセージのトピックに対するトピック受信者の作成

図 15-27 ユースケース図：Oracle オブジェクト型 (ADT) メッセージのトピックに対するトピック受信者の作成 (パブリッシュ / サブスクライブ)



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

セレクタを指定して、ADT メッセージのトピックに対するトピック受信者を作成します。

## 使用上の注意

AQ では、トピックのすべてのサブスクライバ、または特定の受信者にメッセージを送信できます。これらの受信者は、トピックのサブスクライバである場合とそうでない場合があります。受信者がトピックに対するサブスクライバでない場合、明示的にアドレス指定されているメッセージのみを受信します。

この方法は、永続サブスクライバ以外の顧客に対するトピック受信者オブジェクトの作成に使用する必要があります。また、メッセージ・セレクタも指定できます。メッセージ・セレクタの構文は、キュー受信者の構文と同じです。

## 構文

Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の createTopicReceiver を参照してください。

## 例

```
TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              t_sess     = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                        myport = 5521;
TopicReceiver             receiver;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");

t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
```

```
shipped_orders = ((AQJmsSession)jms_sess).getTopic("WS", "Shipped_Orders_Topic");  
  
receiver = ((AQJmsSession)jms_sess).createTopicReceiver(shipped_orders,  
"WesternRegion", null);
```



---

## JMS 操作インタフェース：基本操作 (共有インタフェース)

### ユースケース・モデル

この章では、Oracle アドバンスド・キューイングの操作インタフェースをユースケースに沿って説明します。それぞれの操作（メッセージのエンキューなど）を、その操作名ごとにユースケースの点から説明します。この章の先頭に、すべてのユースケースを示します（16-2 ページの「[ユースケース・モデル：操作インタフェース – 基本操作（共有インタフェース）](#)」を参照）。

### ユースケース・モデルの図式概要

図 16-1「[ユースケース・モデル図：操作インタフェース](#)」に、すべてのユースケースを1つの図にまとめています。

### 個々のユースケース

個々のユースケースは、次のとおり配置されています。

- **ユースケース図**：ユースケースを表す図
- **用途**：このユースケースの用途
- **使用上の注意**：実装に有効なガイドライン
- **構文**：このアクティビティの実行に使用する主な構文
- **例**：各プログラム環境でのユースケースの例

# ユースケース・モデル: JMS 操作インタフェース – 基本操作 (共有インタフェース)

表 16-1 ユースケース・モデル: 操作インタフェース – 基本操作 (共有インタフェース)

---

ユースケース

---

- 16-6 ページの「[JMS 接続の開始](#)」
- 16-8 ページの「[セッションからの JMS 接続の取得](#)」
- 16-10 ページの「[トランザクションを実行したセッションにおけるすべての操作のコミット](#)」
- 16-12 ページの「[トランザクションを実行したセッションにおけるすべての操作のロールバック](#)」
- 16-14 ページの「[JMS セッションからの JDBC 接続の取得](#)」

異なる型のメッセージの作成

- 16-16 ページの「[Byte メッセージの作成](#)」
- 16-18 ページの「[Map メッセージの作成](#)」
- 16-20 ページの「[Stream メッセージの作成](#)」
- 16-22 ページの「[Object メッセージの作成](#)」
- 16-24 ページの「[Text メッセージの作成](#)」
- 16-26 ページの「[ADT メッセージの作成](#)」
- 16-28 ページの「[メッセージの関連識別子の指定](#)」

- 16-30 ページの「[JMS メッセージ・プロパティの指定](#)」

JMS プロパティを異なる型として指定

- 16-32 ページの「[メッセージ・プロパティを Boolean として指定](#)」
- 16-34 ページの「[メッセージ・プロパティを String として指定](#)」
- 16-36 ページの「[メッセージ・プロパティを Int として指定](#)」
- 16-38 ページの「[メッセージ・プロパティを Double として指定](#)」
- 16-40 ページの「[メッセージ・プロパティを Float として指定](#)」
- 16-42 ページの「[メッセージ・プロパティを Byte として指定](#)」
- 16-44 ページの「[メッセージ・プロパティを Long として指定](#)」
- 16-48 ページの「[メッセージ・プロパティを Object として指定](#)」

表 16-1 ユースケース・モデル : 操作インタフェース – 基本操作（共有インタフェース）

## ユースケース

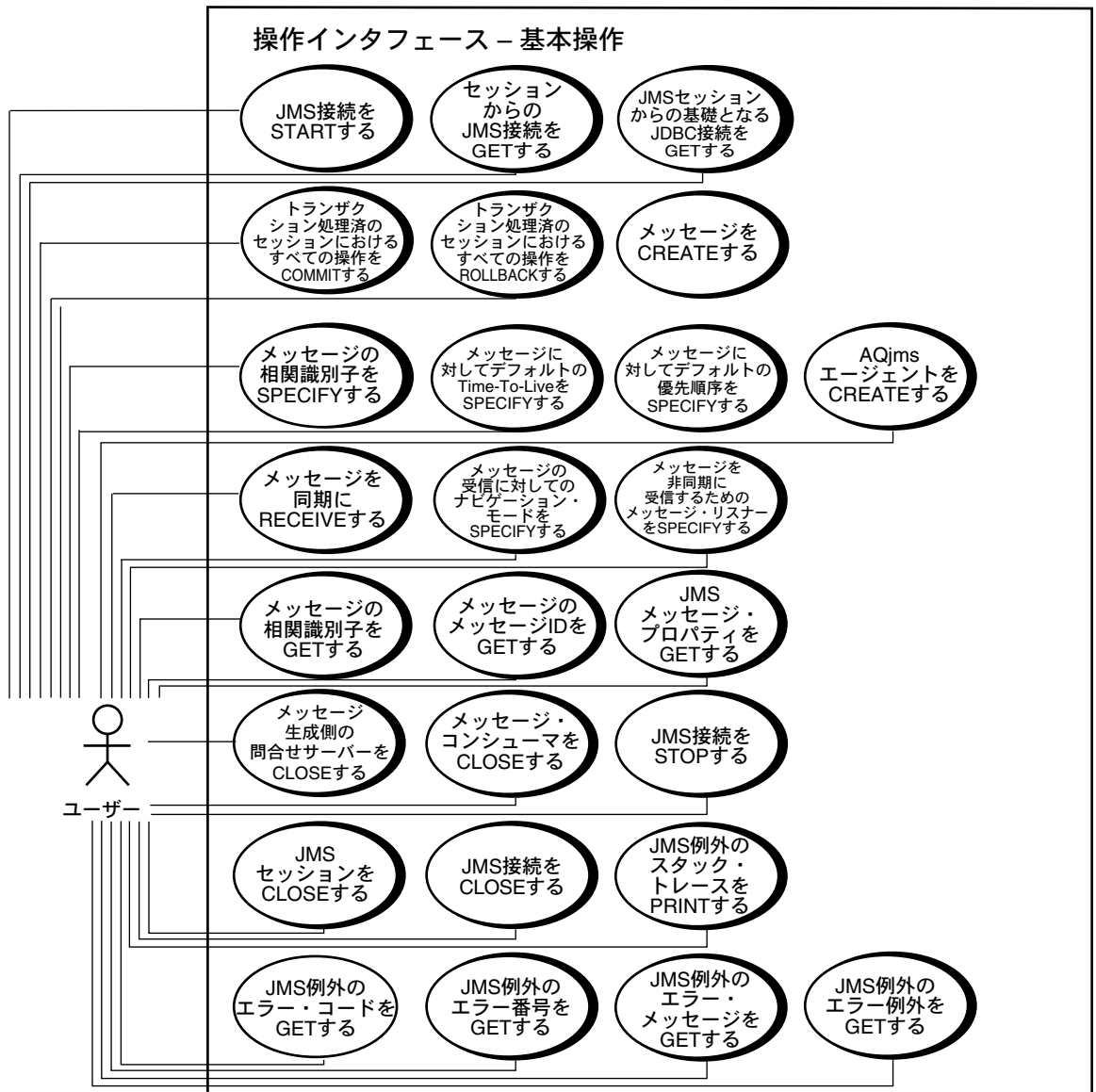
- 16-50 ページの「メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの Time-To-Live の設定」
- 16-52 ページの「メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの優先順位の設定」
- 16-54 ページの「AQjms エージェントの作成」
- 16-56 ページの「メッセージ・コンシューマを使用してメッセージを同期受信する 2 つの方法」
  - 16-58 ページの「タイムアウトを指定してのメッセージ・コンシューマを使用したメッセージの受信」
  - 16-60 ページの「待機なしでのメッセージ・コンシューマを使用したメッセージの受信」
- 16-62 ページの「メッセージの受信に対するナビゲーション・モードの指定」
- 16-65 ページの「メッセージを非同期受信するためにメッセージ・リスナーを指定する 2 つの方法」
  - 16-66 ページの「メッセージ・コンシューマでのメッセージ・リスナーの指定」
  - 16-69 ページの「セッションでのメッセージ・リスナーの指定」
- 16-71 ページの「メッセージの相関識別子の取得」
- 16-73 ページの「メッセージのメッセージ ID を取得する 2 つの方法」
  - 16-74 ページの「メッセージのメッセージ ID を Byte として取得」
  - 16-76 ページの「メッセージのメッセージ ID を String として取得」
- 16-78 ページの「JMS メッセージ・プロパティの取得」
- 異なる型の JMS メッセージ・プロパティの取得
  - 16-80 ページの「メッセージ・プロパティを Boolean として取得」
  - 16-82 ページの「メッセージ・プロパティを String として取得」
  - 16-84 ページの「メッセージ・プロパティを Int として取得」
  - 16-86 ページの「メッセージ・プロパティを Double として取得」
  - 16-88 ページの「メッセージ・プロパティを Float として取得」
  - 16-90 ページの「メッセージ・プロパティを Byte として取得」
  - 16-92 ページの「メッセージ・プロパティを Long として取得」
  - 16-94 ページの「メッセージ・プロパティを Short として取得」
  - 16-94 ページの「メッセージ・プロパティを Object として取得」
- 16-98 ページの「メッセージ・プロデューサのクローズ」

表 16-1 ユースケース・モデル: 操作インタフェース – 基本操作（共有インタフェース）

ユースケース	
16-100	ページの「メッセージ・コンシューマのクローズ」
16-102	ページの「JMS 接続の停止」
16-104	ページの「JMS セッションのクローズ」
16-106	ページの「JMS 接続のクローズ」
16-108	ページの「JMS 例外のエラー・コードの取得」
16-110	ページの「JMS 例外のエラー番号の取得」
16-112	ページの「JMS 例外のエラー・メッセージの取得」
16-114	ページの「JMS 例外にリンクされた例外の取得」
16-116	ページの「JMS 例外のスタック・トレースの出力」

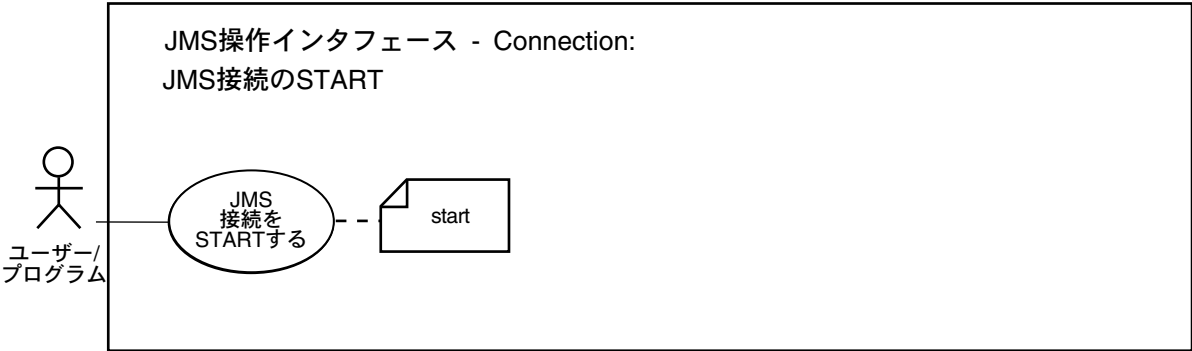


図 16-1 ユースケース・モデル図: 操作インタフェース



# JMS 接続の開始

図 16-2 ユースケース図 : JMS 接続の開始



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージを受信するために、JMS 接続を開始します。

## 使用上の注意

start メソッドを使用して、受信メッセージの接続配信を開始（または再開）します。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsConnection」の start

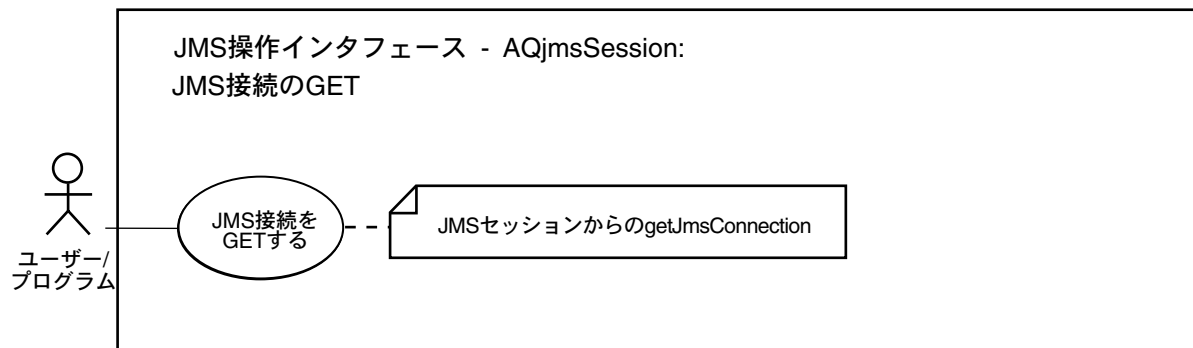
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## セッションからの JMS 接続の取得

図 16-3 ユースケース図：セッションからの JMS 接続の取得



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

セッションから JMS 接続を取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の getJmsConnection

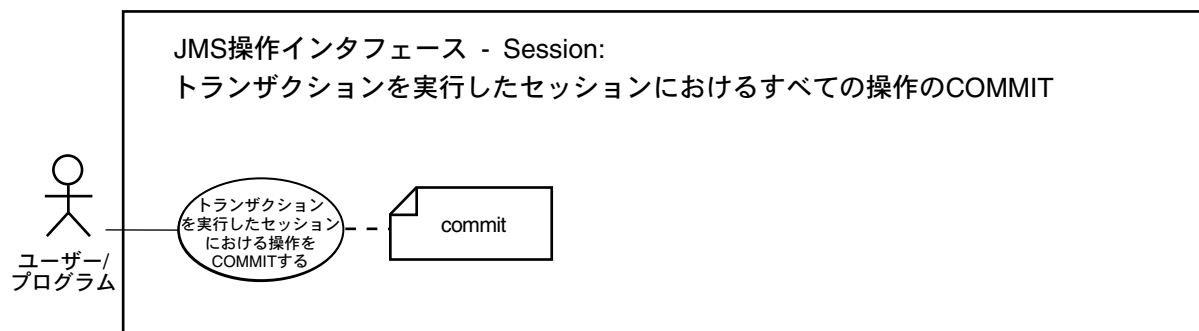
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## トランザクションを実行したセッションにおけるすべての操作のコミット

図 16-4 ユースケース図：トランザクションを実行したセッションにおけるすべての操作のコミット



---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

トランザクションを実行したセッションにおけるすべての操作をコミットします。

## 使用上の注意

このメソッドは、このセッションで実行されているすべての JMS および SQL 操作をコミットします。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQJmsSession」の commit

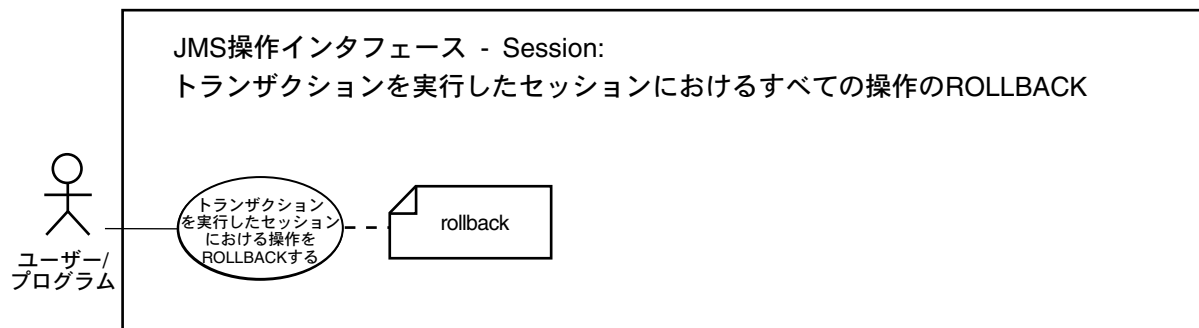
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## トランザクションを実行したセッションにおけるすべての操作のロールバック

図 16-5 ユースケース図：トランザクションを実行したセッションにおけるすべての操作のロールバック



---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

トランザクションを実行したセッションにおけるすべての操作をロールバックします。

## 使用上の注意

このメソッドは、このセッションで実行されているすべての JMS および SQL 操作を異常終了させます。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の rollback



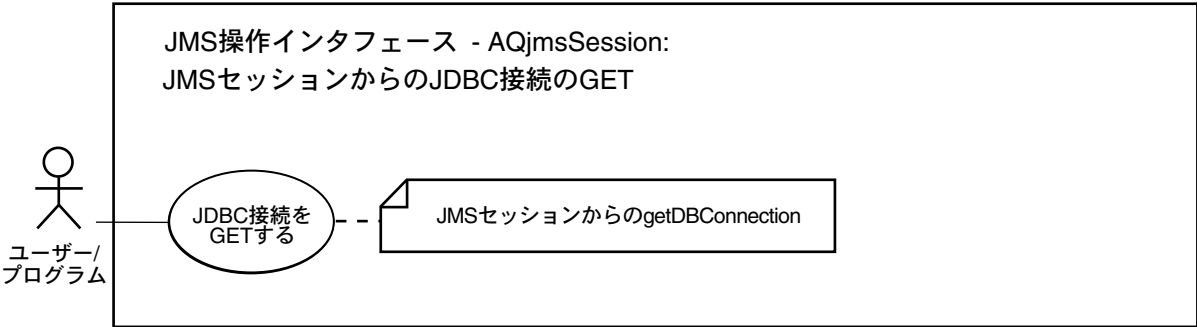
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS セッションからの JDBC 接続の取得

図 16-6 ユースケース図：JMS セッションからの JDBC 接続の取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS セッションから、JDBC 接続を取得します。

## 使用上の注意

このメソッドは、JMS セッションから、JDBC 接続を取得するために使用されます。JDBC 接続は、JMS 操作が実行される同一のトランザクションの一部として、SQL 操作を実行するために使用される場合があります。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の getConnection

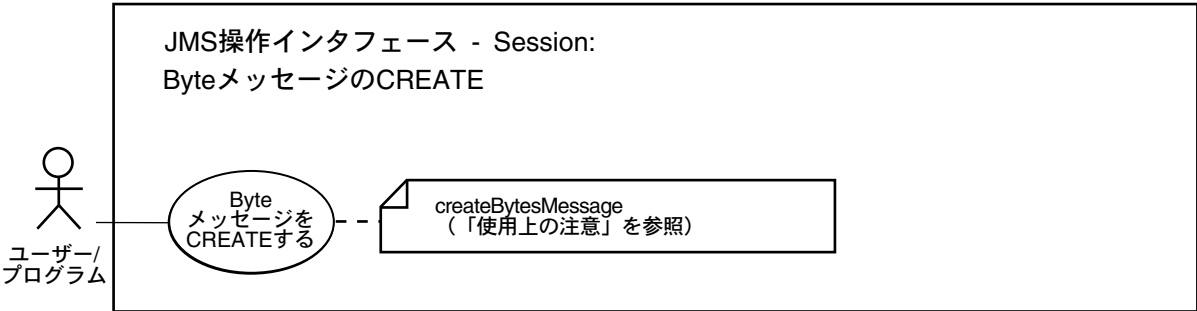
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

```
java.sql.Connection db_conn;  
QueueSession      jms_sess;  
db_conn = ((AQjmsSession) jms_sess).getDBConnection();
```

# Byte メッセージの作成

図 16-7 ユースケース図 : Byte メッセージの作成



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

Byte メッセージを作成します。

## 使用上の注意

このメソッドは、宛先のキュー / トピックを含むキュー・テーブルが、ペイロード型 `SYS.AQ$_JMS_BYTES_MESSAGE` で作成された場合にのみ、使用できます。

`ByteMessage` の移入に使用されるメソッドについては、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ `oracle.jms`」の「`AQjmsSession`」の `createBytesMessage`

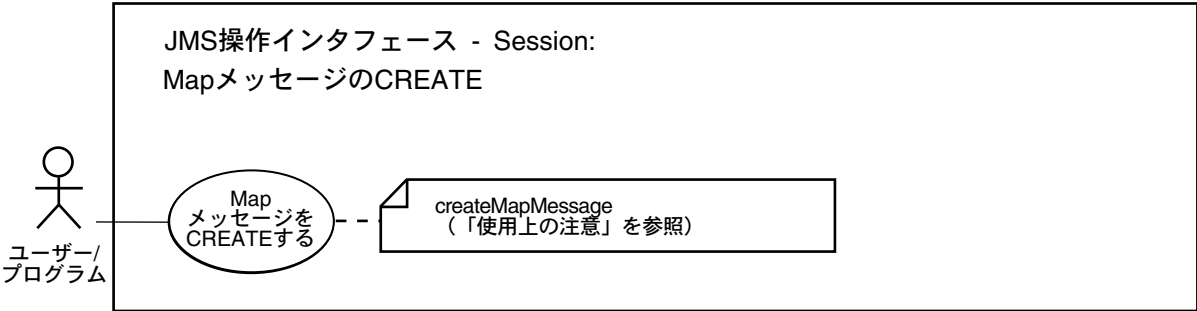
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# Map メッセージの作成

図 16-8 ユースケース図 : Map メッセージの作成



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

Map メッセージを作成します。

## 使用上の注意

このメソッドは、宛先のキュー / トピックを含むキュー・テーブルが、ペイロード型 `SYS.AQ$_JMS_MAP_MESSAGE` で作成された場合のみ、使用できます。

`MapMessage` の移入に使用されるメソッドについては、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ `oracle.jms`」の「`AQjmsSession`」の `createMapMessage`

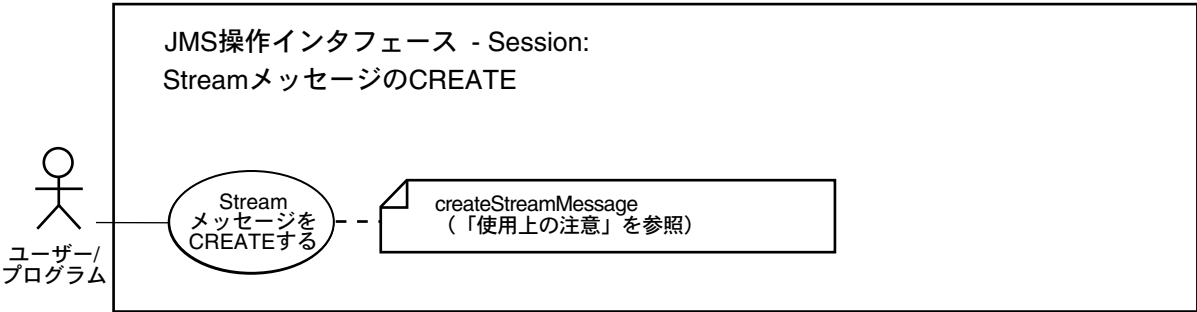
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# Stream メッセージの作成

図 16-9 ユースケース図 : Stream メッセージの作成



**参照 :**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

Stream メッセージを作成します。

## 使用上の注意

このメソッドは、宛先のキュー / トピックを含むキュー・テーブルが、ペイロード型 `SYS.AQ$_JMS_STREAM_MESSAGE` で作成された場合にのみ、使用できます。

`StreamMessage` の移入に使用されるメソッドについては、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ `oracle.jms`」の「`AQJmsSession`」の `createStreamMessage`



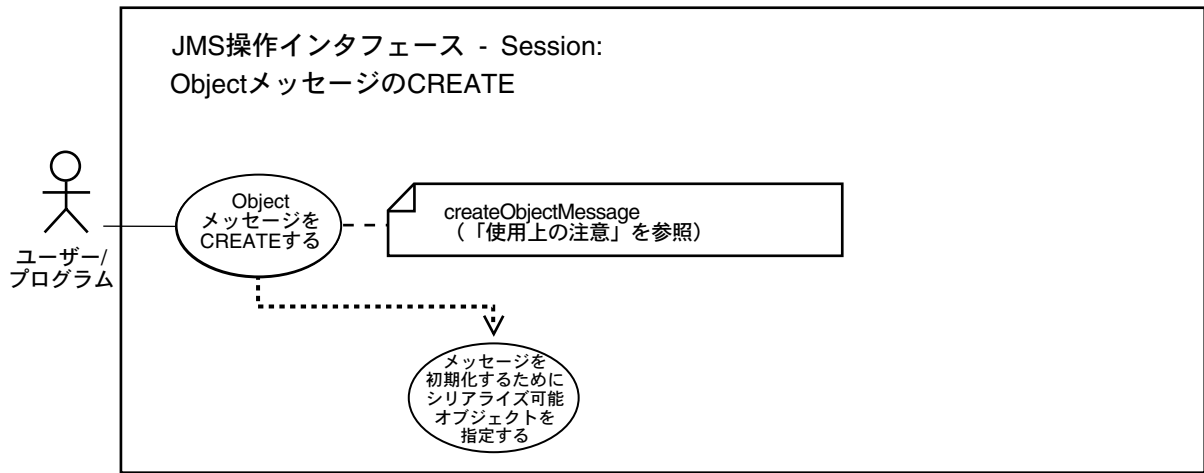
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# Object メッセージの作成

図 16-10 ユースケース図：Object メッセージの作成



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

Object メッセージを作成します。

## 使用上の注意

このメソッドは、宛先のキュー / トピックを含むキュー・テーブルが、ペイロード型 `SYS.AQ$_JMS_OBJECT_MESSAGE` で作成された場合にのみ、使用できます。

ObjectMessage の移入に使用されるメソッドについては、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createObjectMessage

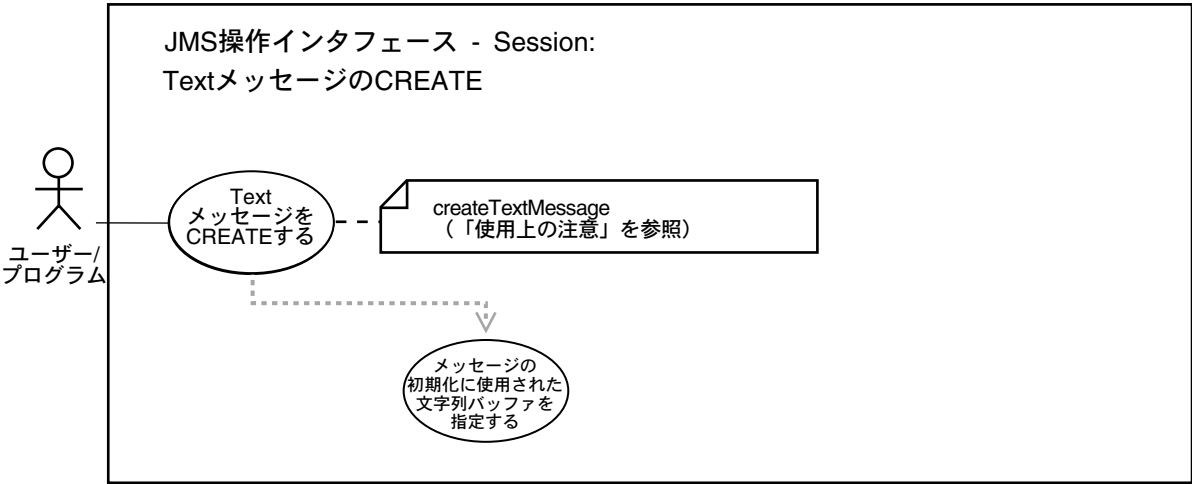
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# Text メッセージの作成

図 16-11 ユースケース図 : Text メッセージの作成



- 参照:**
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

Text メッセージを作成します。

## 使用上の注意

このメソッドは、宛先のキュー / トピックを含むキュー・テーブルが、ペイロード型 `SYS.AQ$_JMS_TEXT_MESSAGE` で作成された場合にのみ、使用できます。

`TextMessage` の移入に使用されるメソッドは、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createTextMessage

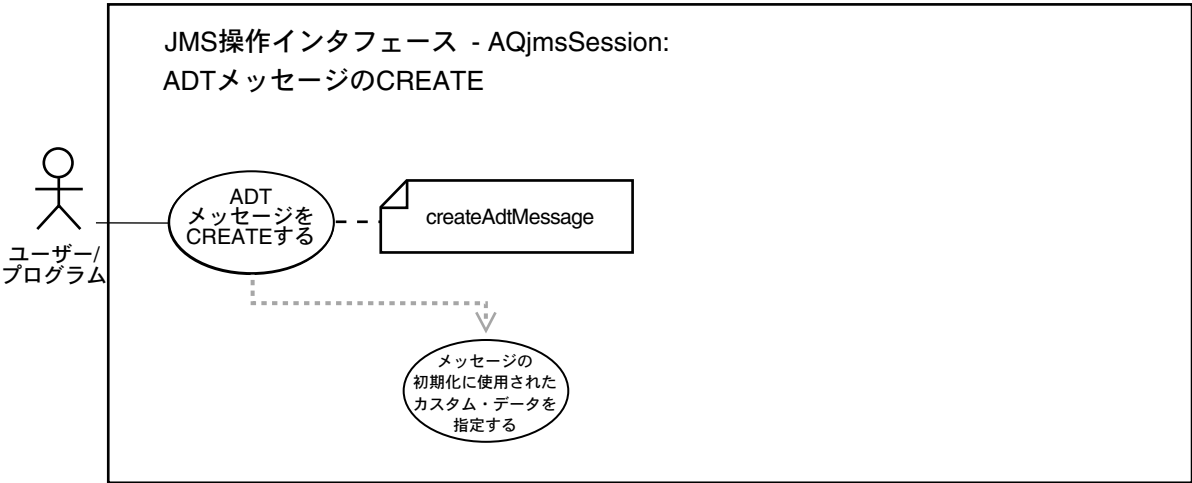
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# ADT メッセージの作成

図 16-12 ユースケース図：ADT メッセージの作成



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

ADT メッセージを作成します。

## 使用上の注意

このメソッドは、キュー / トピックを含むキュー・テーブルが、Oracle のユーザー定義型 (SYS.AQ\$\_JMS\* 型ではない) で作成された場合にのみ、使用できます。

ADT メッセージは、CustomDatum インタフェースを実装するオブジェクトで移入される必要があります。このオブジェクトは、キュー / トピックのペイロードとして定義されている、SQL ADT の Java マッピングである必要があります。SQL のユーザー定義型に対応する Java クラスは、JPublisher ツールを使用して生成される場合があります。CustomDatum インタフェースおよび Jpublisher の詳細は、JDBC ドキュメントを参照してください。

AdtMessage の移入に使用されるメソッドについては、『Oracle8i Java パッケージ・プロシージャ リファレンス』を参照してください。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の createAdtMessage

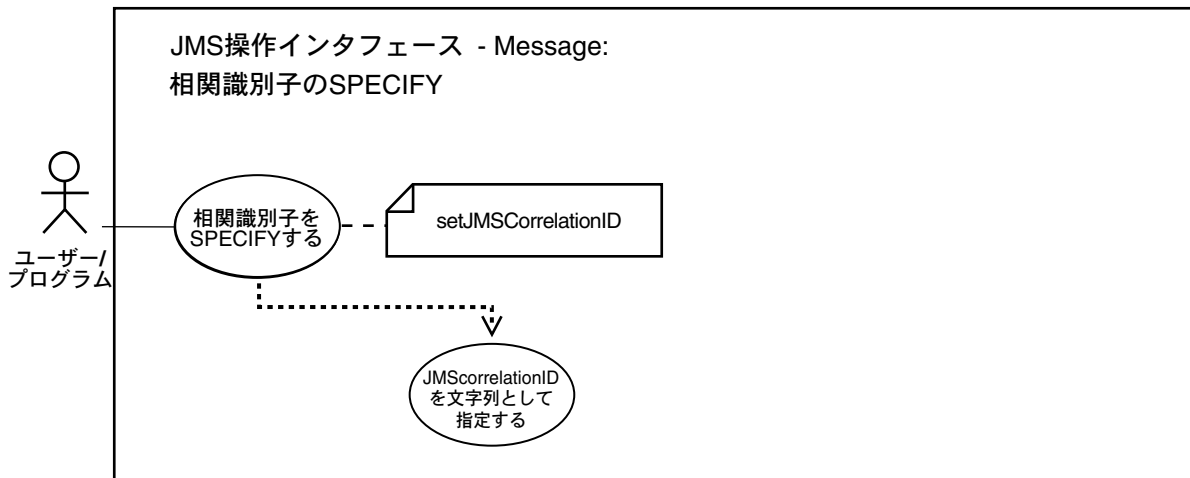
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージの関連識別子の指定

図 16-13 ユースケース図：メッセージの関連識別子の指定



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージの関連識別子を指定します。

## 使用上の注意

ありません。



## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setJMSCorrelationID

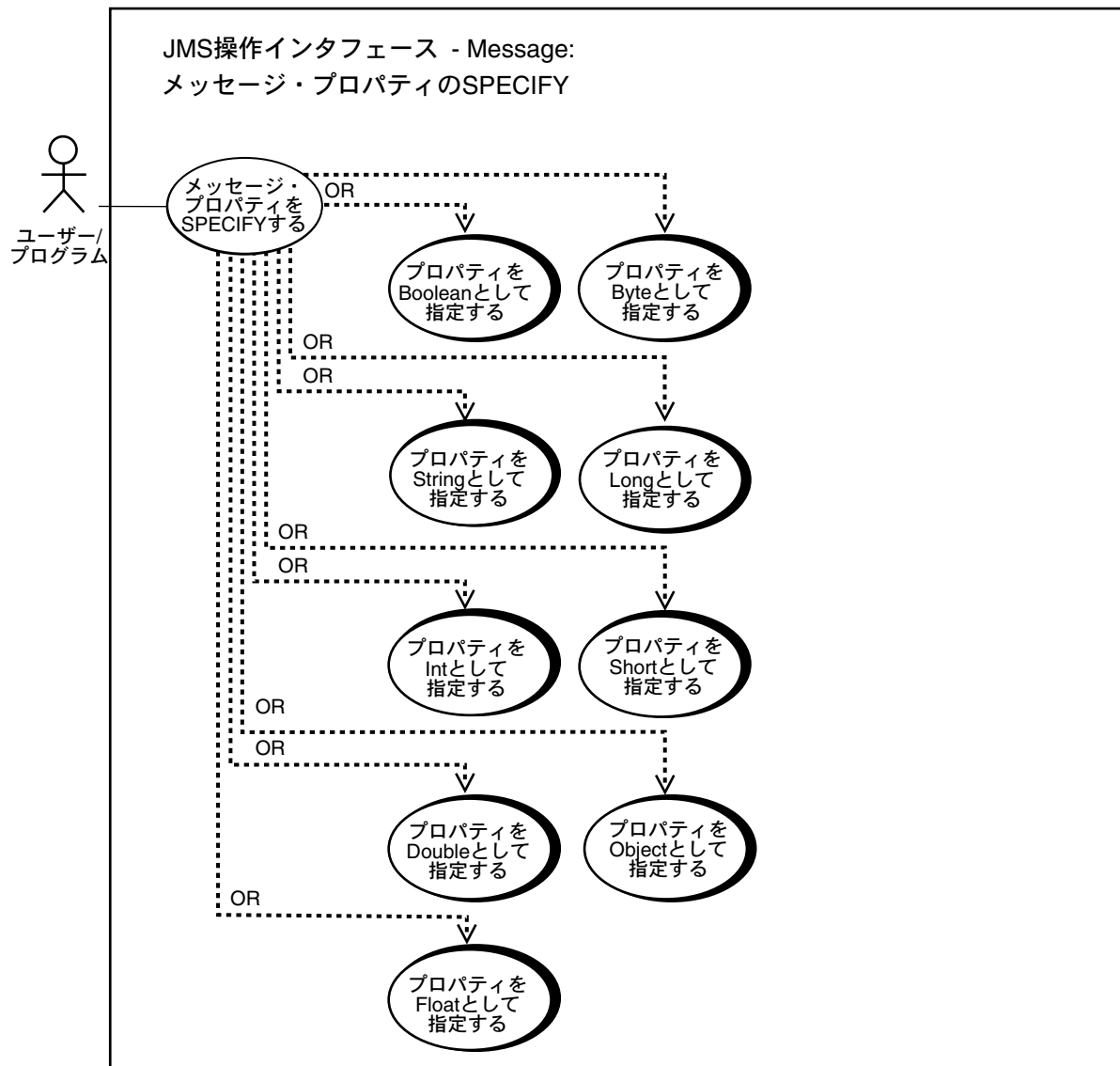
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## JMS メッセージ・プロパティの指定

図 16-14 ユースケース図：JMS メッセージ・プロパティの指定



## 使用上の注意

JMS で始まるプロパティ名は、プロバイダ固有です。ユーザー定義のプロパティは、JMS で始めることができません。

次のプロバイダのプロパティは、クライアントが Text メッセージ、Stream メッセージ、Object メッセージ、Byte メッセージまたは Map メッセージを使用して、設定する場合があります。

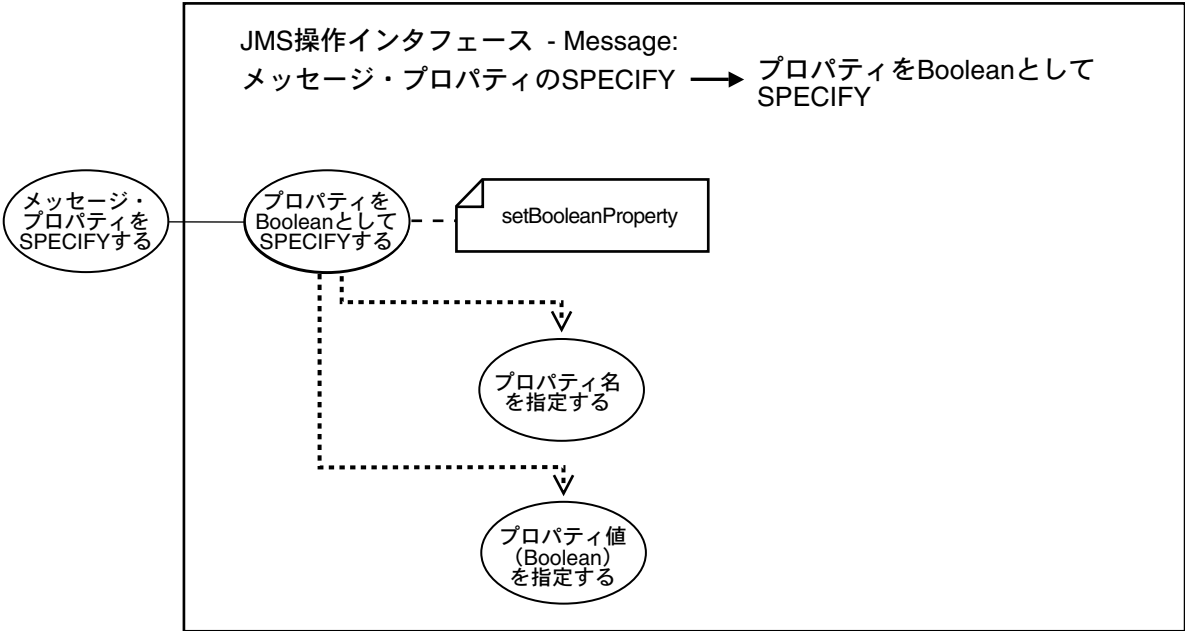
- JMSXAppID (String)
- JMSXGroupID (String)
- JMSXGroupSeq (Int)
- JMS\_OracleExcpQ (String) - 例外キュー
- JMS\_OracleDelay (Int) - メッセージ遅延 (秒)

次のプロパティは、ADT メッセージで設定されます。

- JMS\_OracleExcpQ (String) - 例外キュー - <schema>.queue\_name として指定
- JMS\_OracleDelay (Int) - メッセージ遅延 (秒)

# メッセージ・プロパティを Boolean として指定

図 16-15 ユースケース図：メッセージ・プロパティを Boolean として指定



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Boolean として指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setBooleanProperty

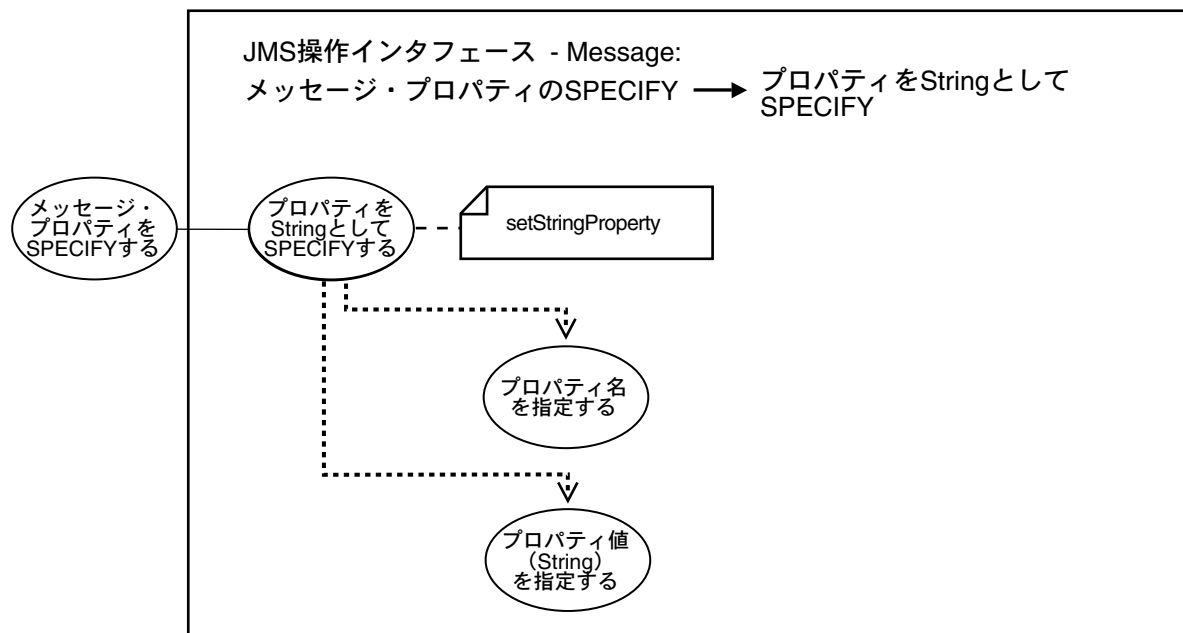
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを String として指定

図 16-16 ユースケース図：メッセージ・プロパティを String として指定



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージ・プロパティを String として指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の `setStringProperty`

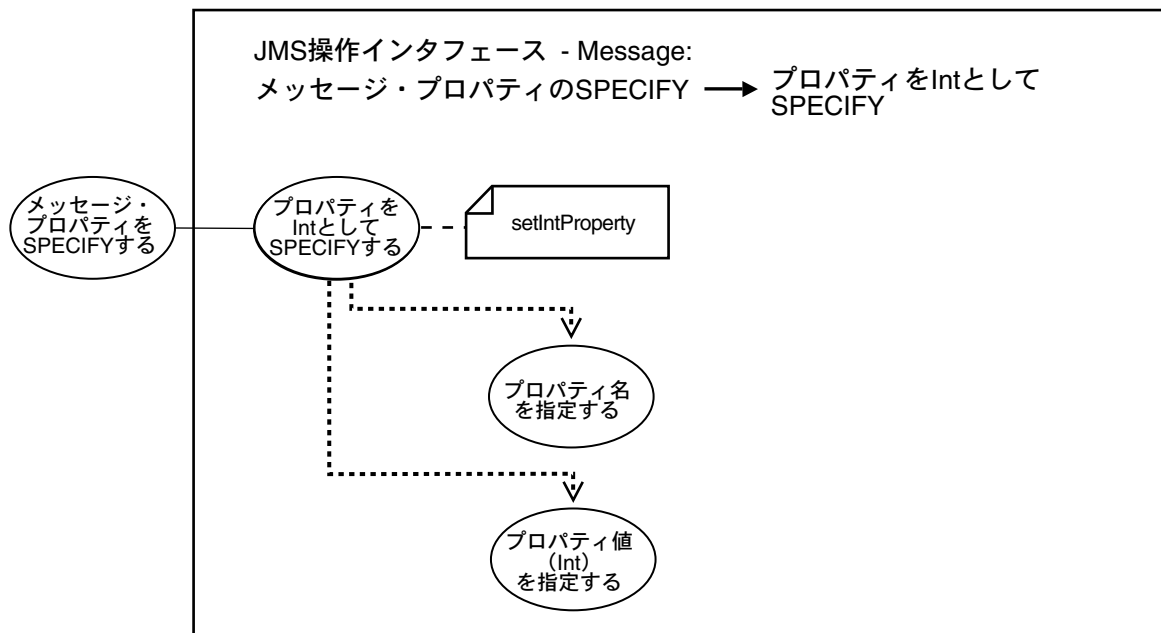
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを Int として指定

図 16-17 ユースケース図：メッセージ・プロパティを Int として指定



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース - 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージ・プロパティを Int として指定します。

## 使用上の注意

ありません。



## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setIntProperty

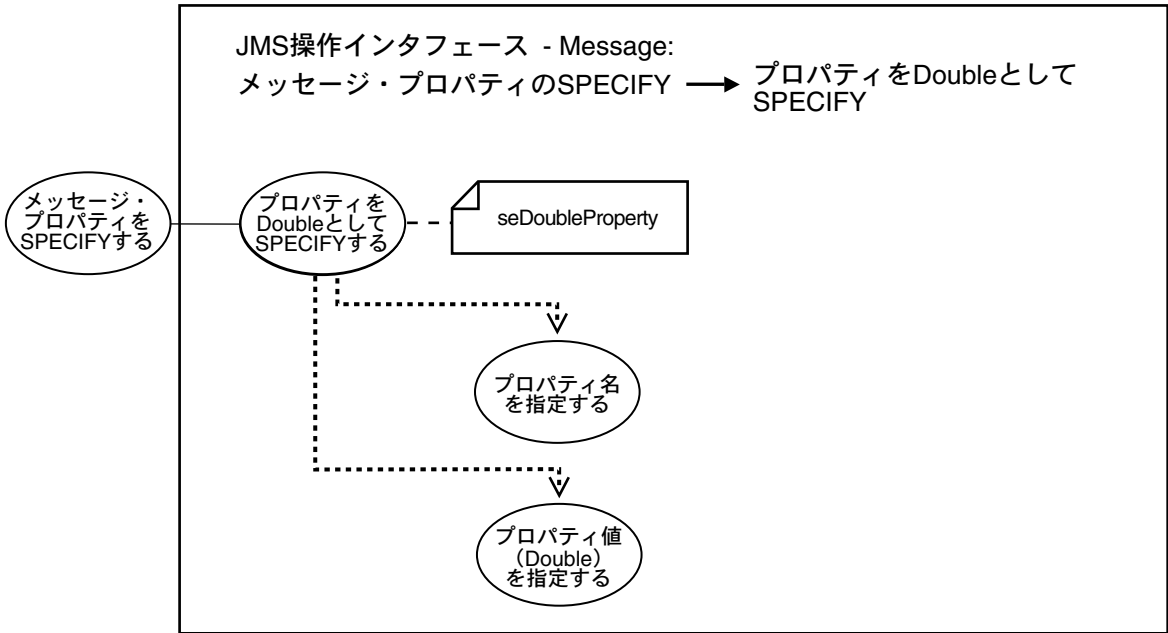
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# メッセージ・プロパティを Double として指定

図 16-18 ユースケース図：メッセージ・プロパティを Double として指定



- 参照：
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Double として指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setDoubleProperty

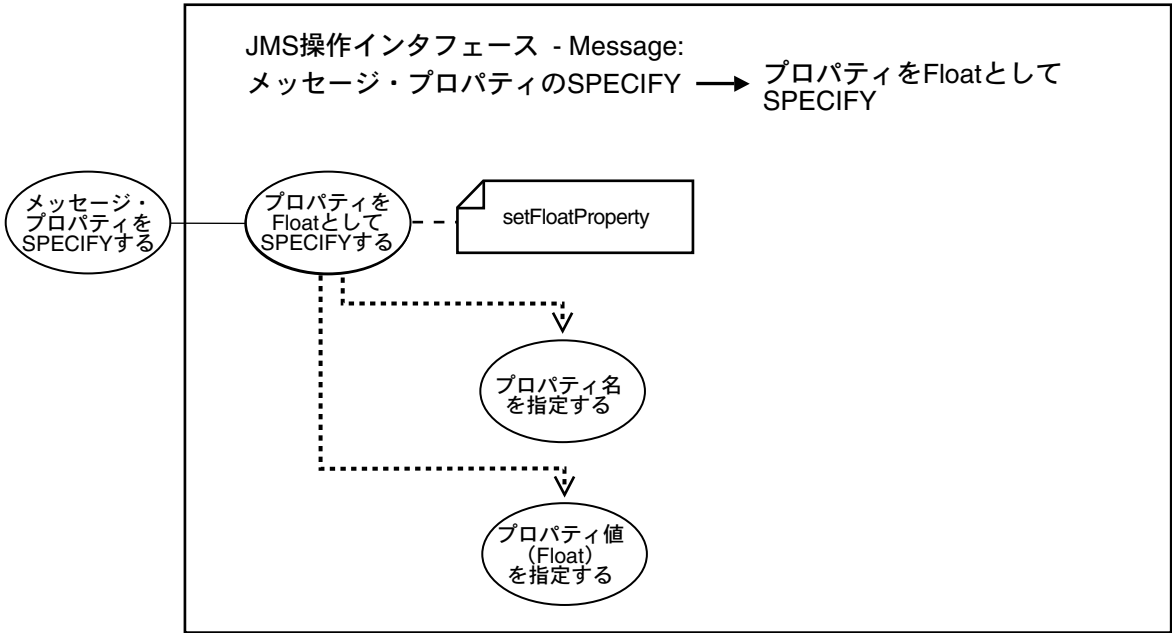
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# メッセージ・プロパティを Float として指定

図 16-19 ユースケース図：メッセージ・プロパティを Float として指定



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Float として指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setFloatProperty

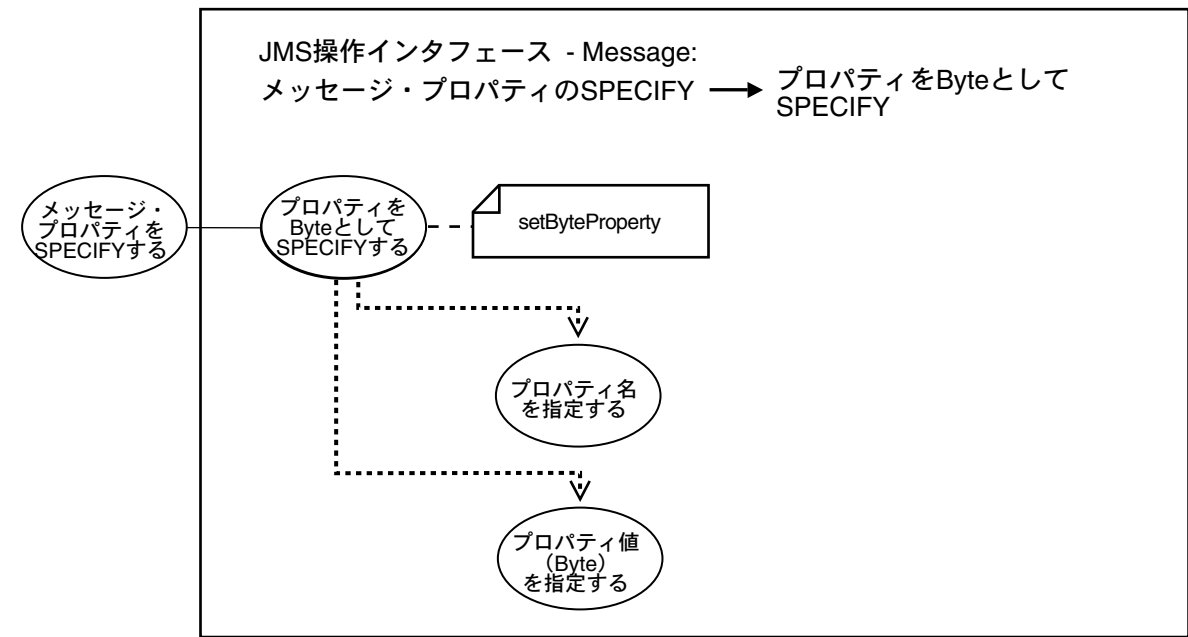
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# メッセージ・プロパティを Byte として指定

図 16-20 ユースケース図：メッセージ・プロパティを Byte として指定



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Byte として指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setByteProperty

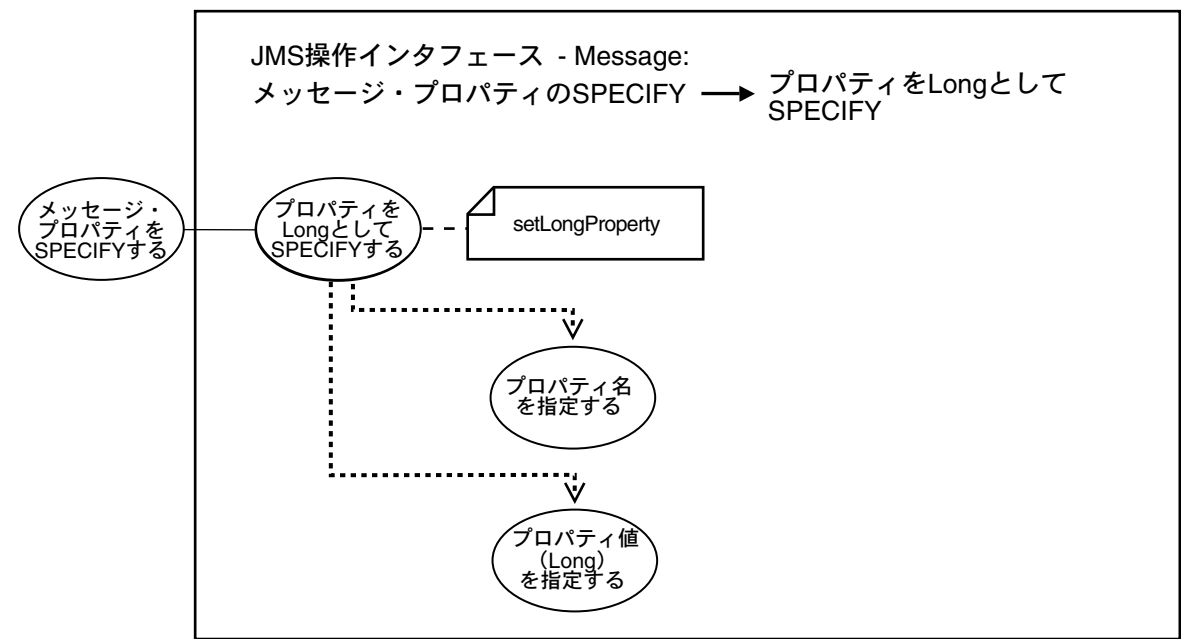
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# メッセージ・プロパティを Long として指定

図 16-21 ユースケース図：メッセージ・プロパティを Long として指定



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Long として指定します。

## 使用上の注意

ありません。



## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setLongProperty

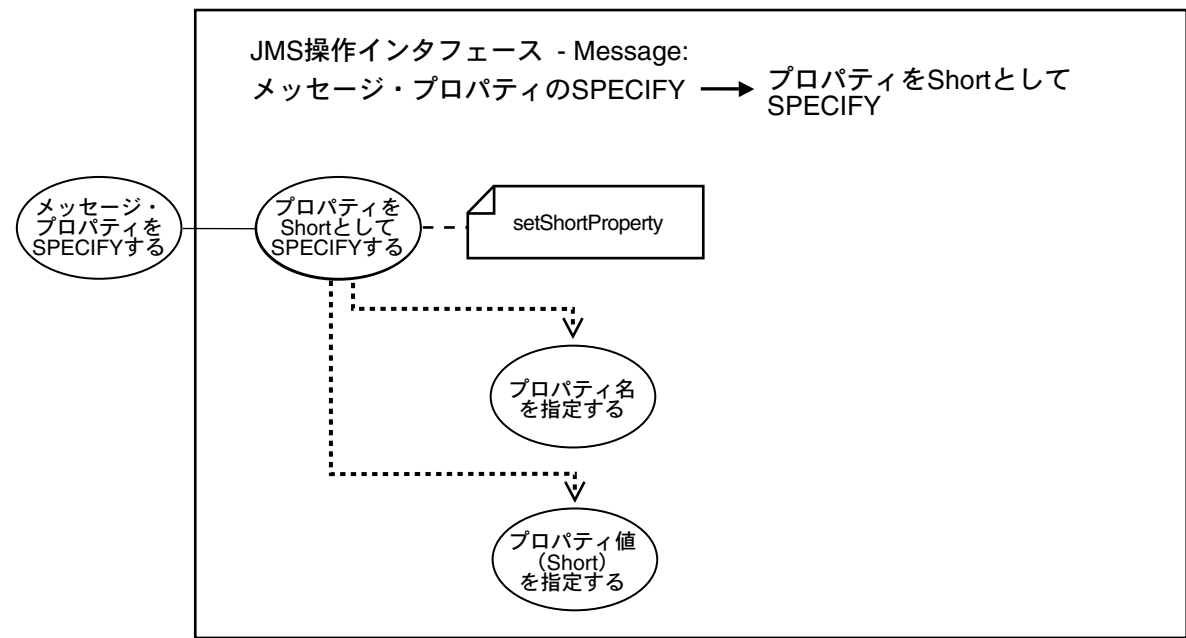
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# メッセージ・プロパティを Short として指定

図 16-22 ユースケース図：メッセージ・プロパティを Short として指定



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Short として指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setShortProperty

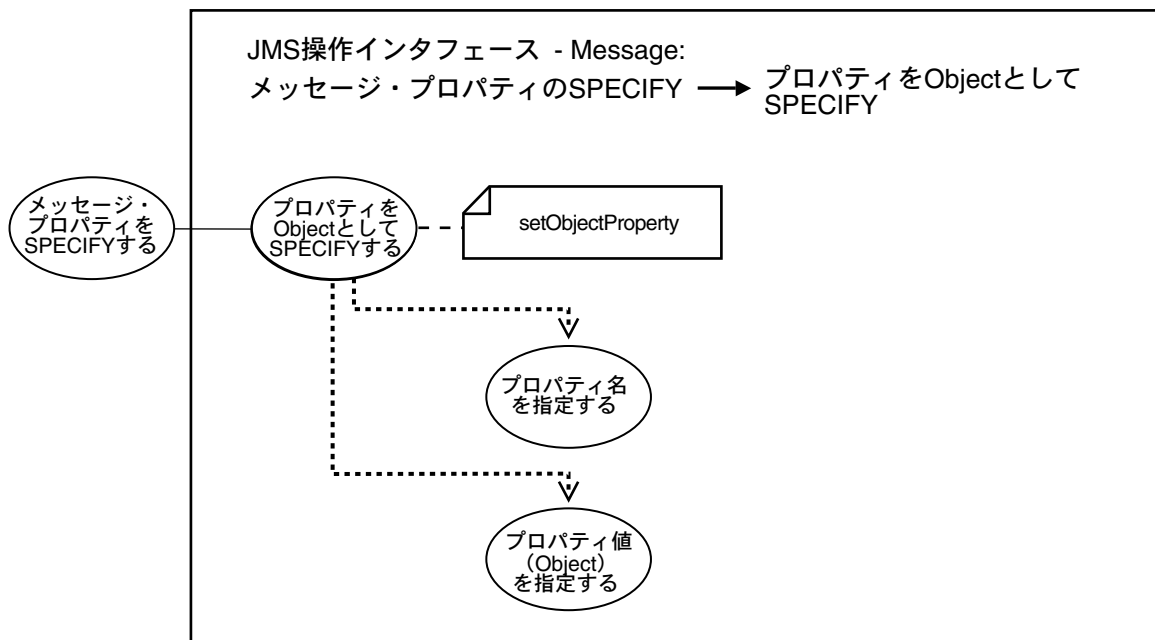
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを Object として指定

図 16-23 ユースケース図：メッセージ・プロパティを Object として指定



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージ・プロパティを Object として指定します。

## 使用上の注意

Java 基本型の値 (Boolean、Byte、Short、Int、Long、Float、Double および String) のみを含むオブジェクトがサポートされています。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の setObjectProperty

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの Time-To-Live の設定

図 16-24 ユースケース図：メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの Time-To-Live の設定



---

---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル : JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージ・プロデューサが送信するすべてのメッセージに対して、デフォルトの Time-To-Live を設定します。

## 使用上の注意

Time-To-Live は、1000 分の 1 秒単位で指定されます。これは、メッセージが ready 状態になった後（メッセージ遅延が有効になった後）に計算されます。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsProducer」の setTimeToLive

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

/\* QueueSender から送信されるすべてのメッセージのデフォルトの Time-To-Live 値を 100000 ミリ秒に設定します。\*/

```
QueueSender sender;  
sender.setTimeToLive(100000);
```

## メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの優先順位の設定

図 16-25 ユースケース図：メッセージ・プロデューサが送信するすべてのメッセージに対するデフォルトの優先順位の設定



---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

メッセージ・プロデューサが送信するすべてのメッセージに対して、デフォルトの優先順位を設定します。

## 使用上の注意

優先順位の値には、どの整数値でも指定できます。数値が小さいほど、優先順位が高いことを表します。

優先順位の値が、送信操作中に明示的に指定されている場合は、このメソッドで設定されたプロデューサのデフォルト値をオーバーライドします。



## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsProducer」の setPriority

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

### 例 1

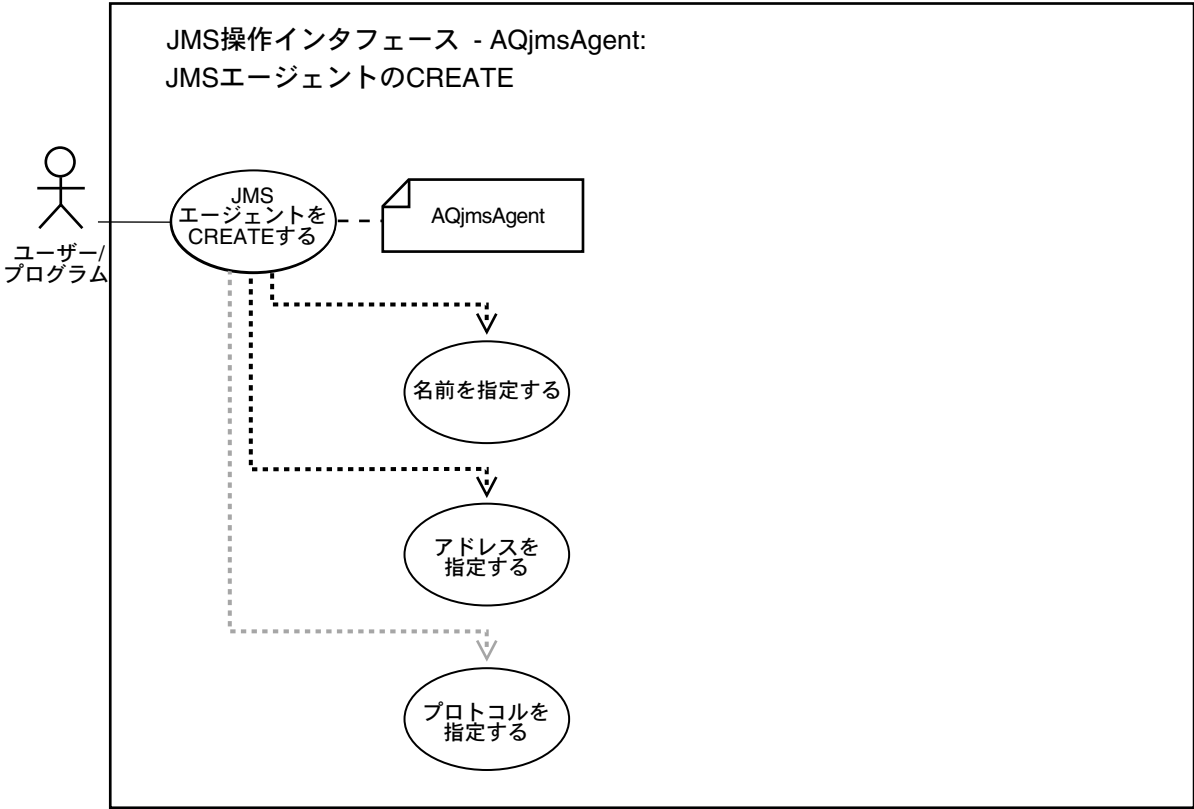
```
/* QueueSender から送信されるすべてのメッセージのデフォルトの優先順位を 2 に設定します。*/  
QueueSender sender;  
sender.setPriority(2);
```

### 例 2

```
/* TopicPublisher から送信されるすべてのメッセージのデフォルトの優先順位を 2 に設定します。*/  
TopicPublisher publisher;  
publisher.setPriority(1);
```

# AQjms エージェントの作成

図 16-26 ユースケース図 : AQjms エージェントの作成



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

AQjms エージェントを作成します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsAgent」

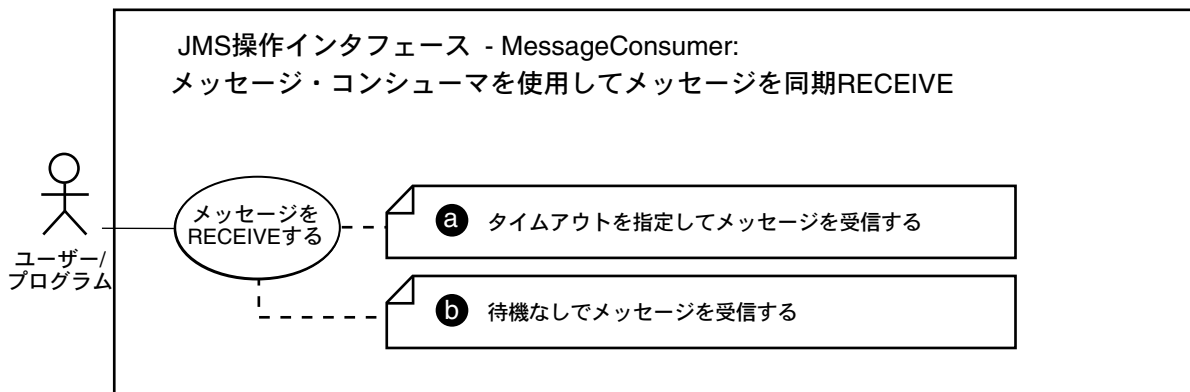
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・コンシューマを使用してメッセージを同期受信する2つの方法

図 16-27 ユースケース図：メッセージ・コンシューマを使用してメッセージを同期受信する2つの方法



---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

## 用途

2つの方法で、メッセージ・コンシューマを使用してメッセージを同期受信します。

- 16-58 ページの「タイムアウトを指定してのメッセージ・コンシューマを使用したメッセージの受信」
- 16-60 ページの「待機なしでのメッセージ・コンシューマを使用したメッセージの受信」

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsConsumer」の receive

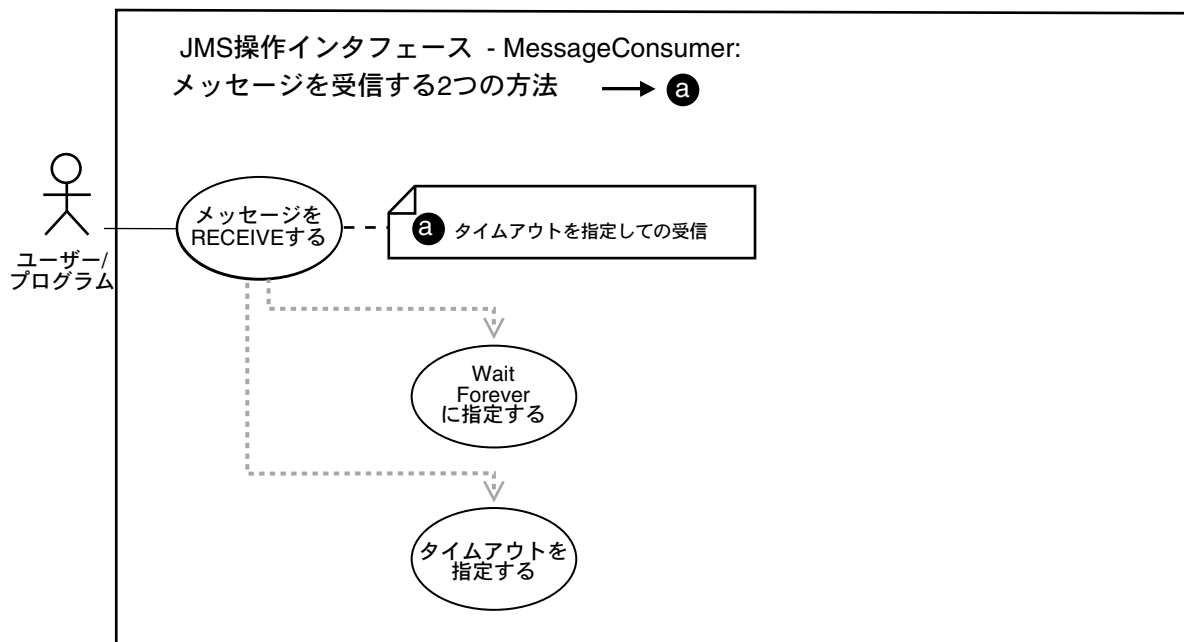
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## タイムアウトを指定してのメッセージ・コンシューマを使用したメッセージの受信

図 16-28 ユースケース図：タイムアウトを指定してのメッセージ・コンシューマを使用したメッセージの受信



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

タイムアウトを指定し、メッセージ・コンシューマを使用してメッセージを受信します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsConsumer」の receive

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

```

TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              t_sess     = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                        myport = 5521;

/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");

t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession ) jms_sess).getTopic("WS",
"Shipped_Orders_Topic");

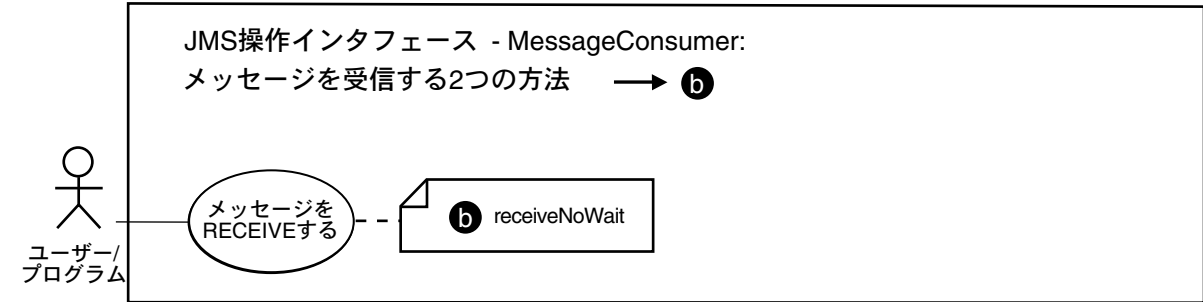
/* 正確な CustomDatumFactory およびセクタを指定して、サブスクライバを作成します。*/
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders,
'WesternShipping',
        " priority > 1 and tab.user_data.region like 'WESTERN %'",
        false,AQjmsAgent.getFactory());

/* メッセージがない場合は、30 秒間ブロックして受信します。*/
Message = subscriber.receive(30000);

```

# 待機なしでのメッセージ・コンシューマを使用したメッセージの受信

図 16-29 ユースケース図：待機なしでのメッセージ・コンシューマを使用したメッセージの受信



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

待機なしで、メッセージ・コンシューマを使用してメッセージを受信します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsConsumer」の receiveNoWait



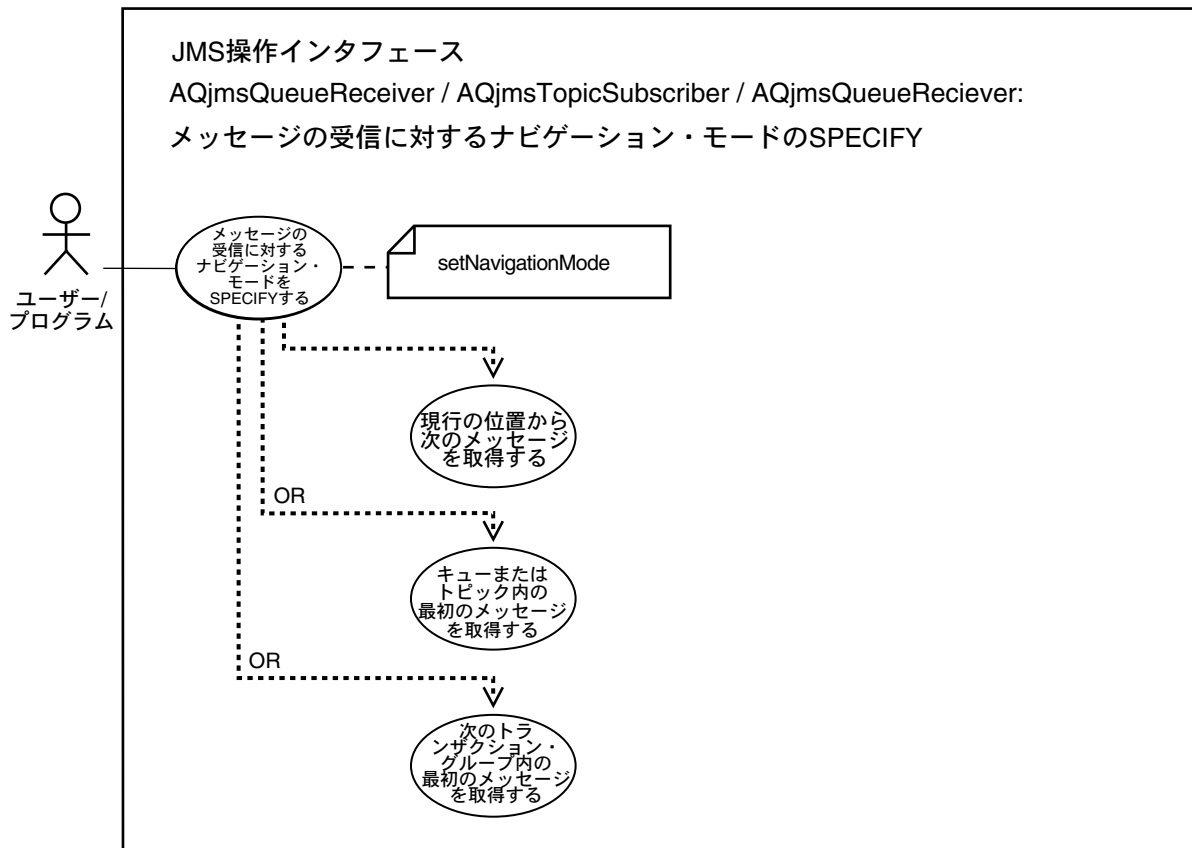
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージの受信に対するナビゲーション・モードの指定

図 16-30 ユースケース図：メッセージの受信に対するナビゲーション・モードの指定



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージの受信に対して、ナビゲーション・モードを指定します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsQueueReceiver」の setNavigationMode、 「AQjmsTopicReceiver」の setNavigationMode、 「AQjmsTopicSubscriber」の setNavigationMode

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

```

TopicConnectionFactory  tc_fact  = null;
TopicConnection         t_conn   = null;
TopicSession            t_sess   = null;
TopicSession            jms_sess;
Topic                   shipped_orders;
int                      myport = 5521;

/* 接続およびセッションを作成します。*/

tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                  "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("WS",
"Shipped_Orders_Topic");
/* 正確な CustomDatumFactory およびセレクトを指定して、サブスクライバを作成します。*/

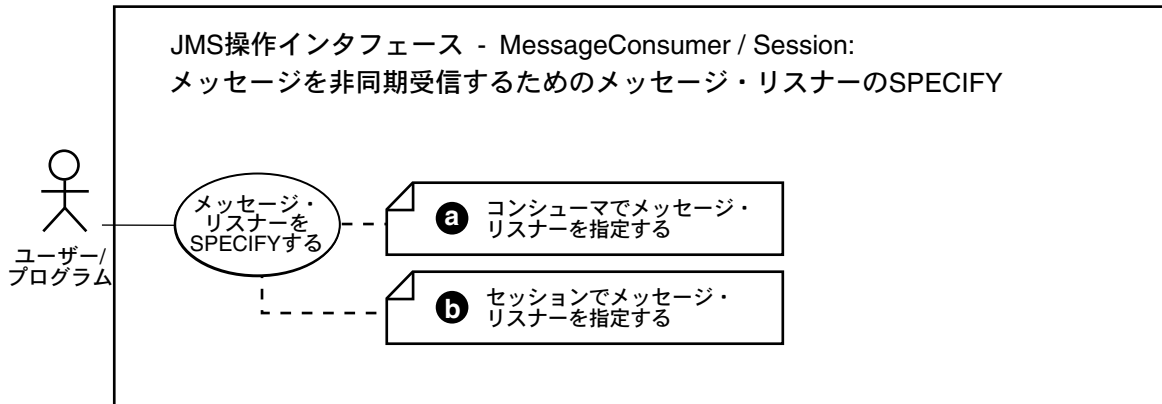
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders,
'WesternShipping',
        " priority > 1 and tab.user_data.region like 'WESTERN %'",

```

```
        false, AQjmsAgent.getFactory());  
  
subscriber1.setNavigationMode(AQjmsConstants.NAVIGATION_FIRST_MESSAGE);  
  
/* メッセージがない場合は、すぐに戻ってサブスクライバのメッセージを取得します。*/  
Message = subscriber.receive();
```

## メッセージを非同期受信するためにメッセージ・リスナーを指定する2つの方法

図 16-31 ユースケース図：メッセージを非同期受信するためにメッセージ・リスナーを指定する2つの方法



### 参照：

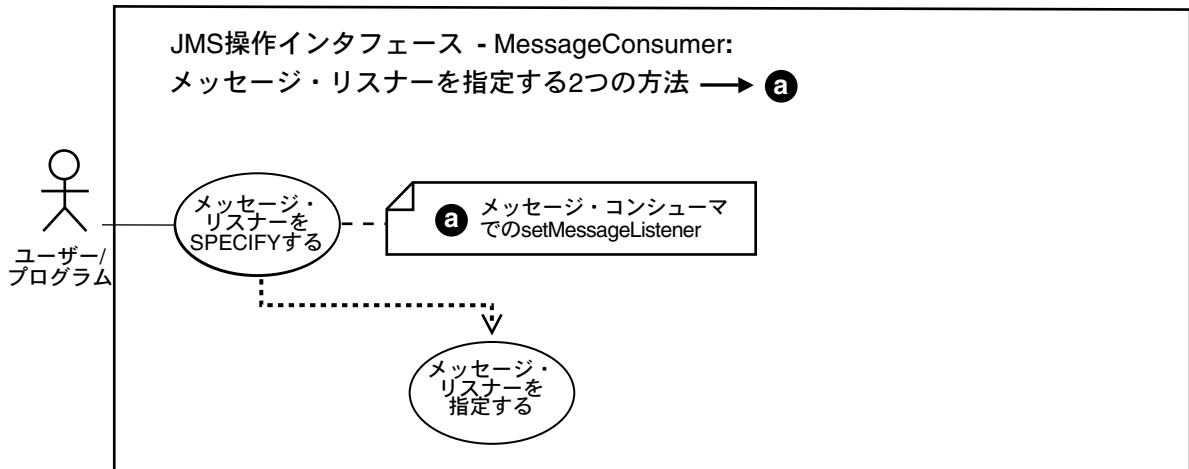
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

2つの方法で、メッセージ・リスナーを指定して、メッセージを非同期受信します。

- 16-66 ページの「メッセージ・コンシューマでのメッセージ・リスナーの指定」
- 16-69 ページの「セッションでのメッセージ・リスナーの指定」

## メッセージ・コンシューマでのメッセージ・リスナーの指定

図 16-32 ユースケース図：メッセージ・コンシューマでのメッセージ・リスナーの指定



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 
- 

### 用途

メッセージ・コンシューマでメッセージ・リスナーを指定します。

### 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsConsumer」の setMessageListener

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

```

TopicConnectionFactory    tc_fact    = null;
TopicConnection           t_conn     = null;
TopicSession              t_sess     = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                        myport = 5521;
MessageListener            mLis = null;
/* 接続およびセッションを作成します。*/
tc_fact = AQjmsFactory.getTopicConnectionFactory("MYHOSTNAME",
                                                    "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

shipped_orders = ((AQjmsSession) jms_sess).getTopic("WS",
"Shipped_Orders_Topic");
/* 正確な CustomDatumFactory およびセクタを指定して、サブスクライバを作成します。*/
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders,
'WesternShipping',
        " priority > 1 and tab.user_data.region like 'WESTERN %'",
        false, AQjmsAgent.getFactory());

mLis = new myListener(jms_sess, "foo");
/* メッセージがない場合は、すぐに戻ってサブスクライバのメッセージを取得します。*/
subscriber.setMessageListener(mLis);

The definition of the myListener class
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import java.util.*;
public class myListener implements MessageListener
{

```

```
TopicSession  mySess;
String         myName;

/* コンストラクタ */
myListener(TopicSession t_sess, String t_name)
{
    mySess = t_sess;
    myName = t_name;
}

public onMessage(Message m)
{
    System.out.println("Retrieved message with correlation: " ||
m.getJMSCorrelationID());

    try{
        /* デキューをコミットします。*/
        mySession.commit();
    } catch (java.sql.SQLException e)
    {System.out.println("SQL Exception on commit"); }

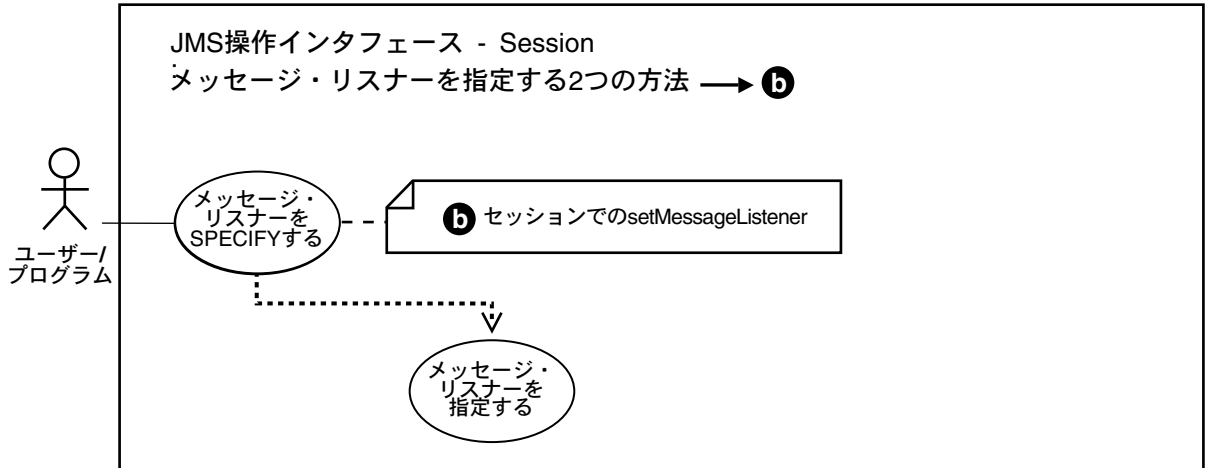
}

}
```



## セッションでのメッセージ・リスナーの指定

図 16-33 ユースケース図：セッションでのメッセージ・リスナーの指定



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 
- 

### 用途

セッションでメッセージ・リスナーを指定します。

### 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsSession」の setMessageListener

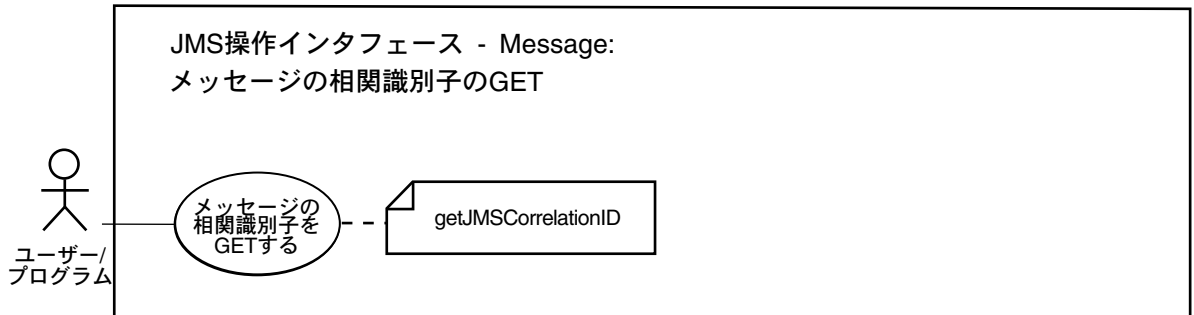
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージの相関識別子の取得

図 16-34 ユースケース図：メッセージの相関識別子の取得



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージの相関識別子を取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsMessage」の getJMSCorrelationID

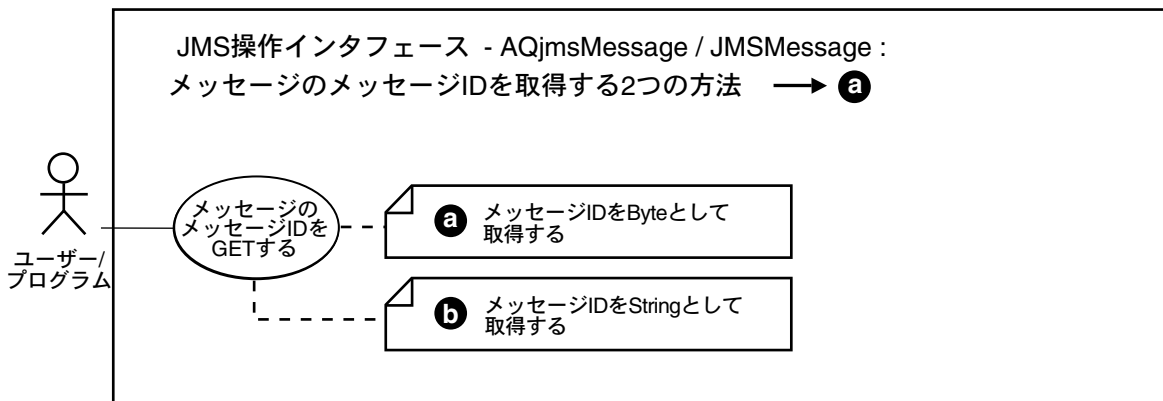
### 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージのメッセージ ID を取得する 2 つの方法

図 16-35 ユースケース図：メッセージのメッセージ ID を取得する 2 つの方法



### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

2 つの方法で、メッセージのメッセージ ID を取得します。

- Byte を使用する場合：16-74 ページの「メッセージのメッセージ ID を Byte として取得」
- String を使用する場合：16-76 ページの「メッセージのメッセージ ID を String として取得」

# メッセージのメッセージ ID を Byte として取得

図 16-36 ユースケース図：メッセージのメッセージ ID を Byte として取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージのメッセージ ID を Byte として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsMessage」の getJMSMessageID

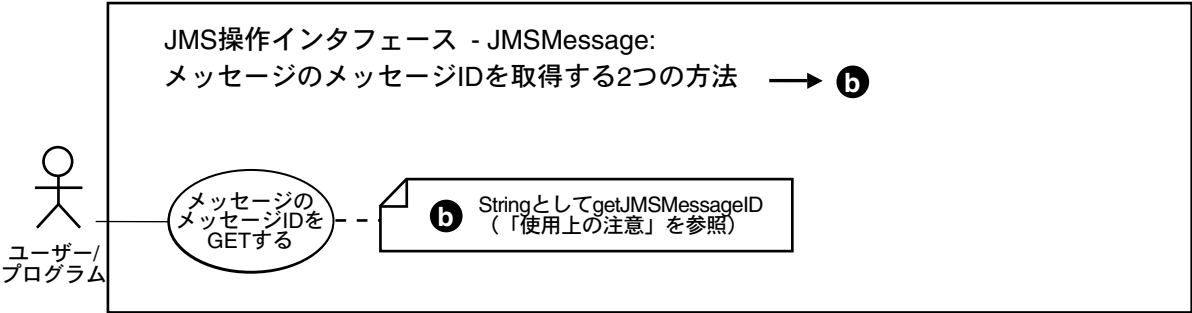
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# メッセージのメッセージ ID を String として取得

図 16-37 ユースケース図：メッセージのメッセージ ID を String として取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージのメッセージ ID を String として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsMessage」の getJMSMessageID



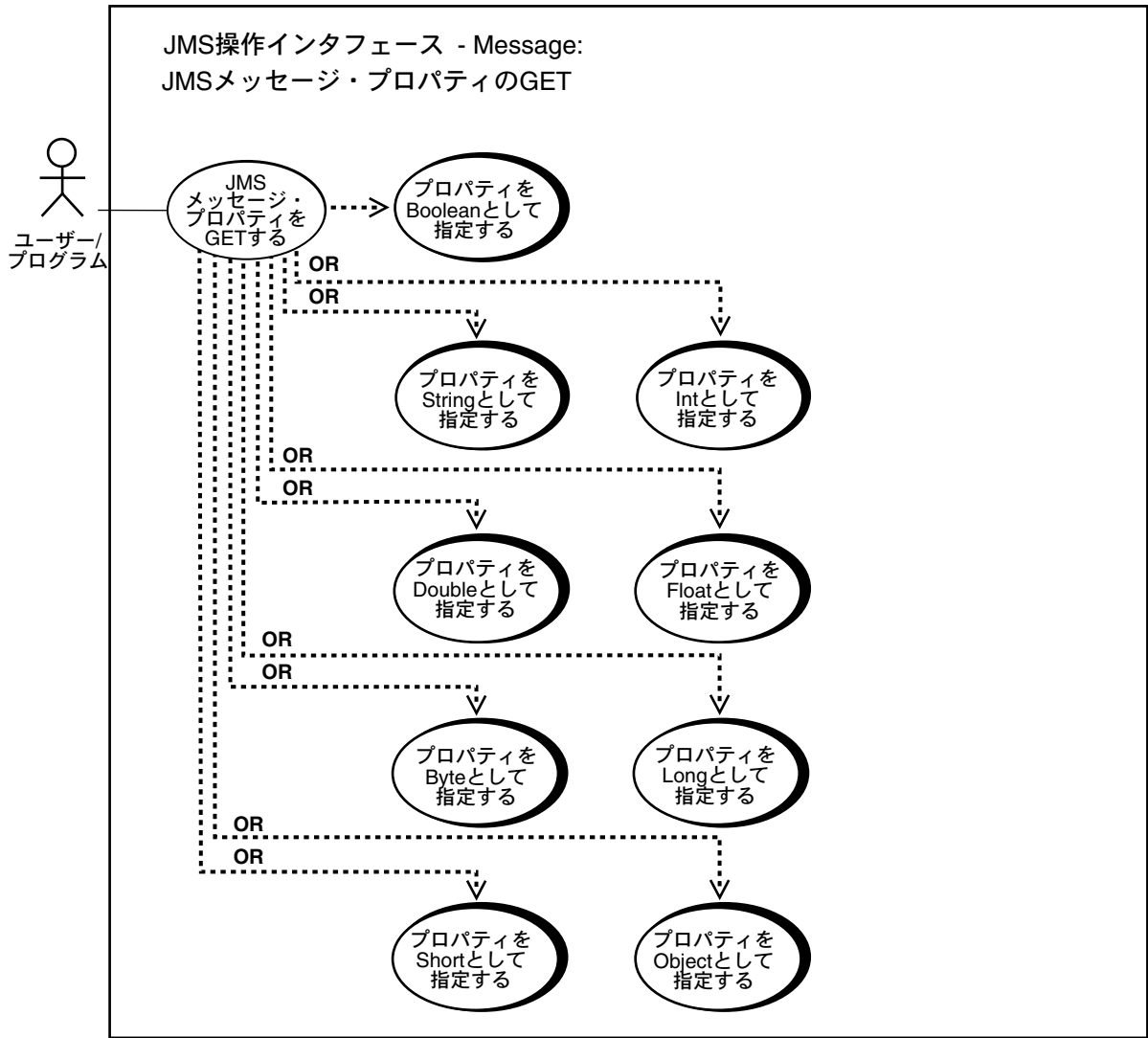
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS メッセージ・プロパティの取得

図 16-38 ユースケース図：JMS メッセージ・プロパティの取得



---

**参照：**

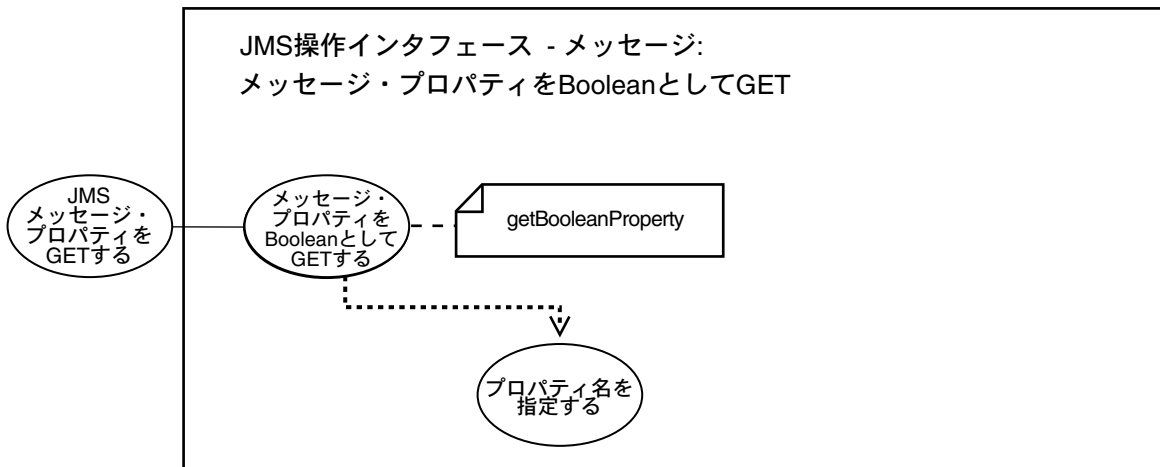
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 

次の方法で、JMS メッセージ・プロパティを取得します。

- 16-80 ページの「メッセージ・プロパティを Boolean として取得」
- 16-82 ページの「メッセージ・プロパティを String として取得」
- 16-84 ページの「メッセージ・プロパティを Int として取得」
- 16-86 ページの「メッセージ・プロパティを Double として取得」
- 16-88 ページの「メッセージ・プロパティを Float として取得」
- 16-90 ページの「メッセージ・プロパティを Byte として取得」
- 16-92 ページの「メッセージ・プロパティを Long として取得」
- 16-94 ページの「メッセージ・プロパティを Short として取得」
- 16-96 ページの「メッセージ・プロパティを Object として取得」

## メッセージ・プロパティを Boolean として取得

図 16-39 ユースケース図：メッセージ・プロパティを Boolean として取得



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: [JMS 操作インタフェース - 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージ・プロパティを Boolean として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の `getBooleanProperty`

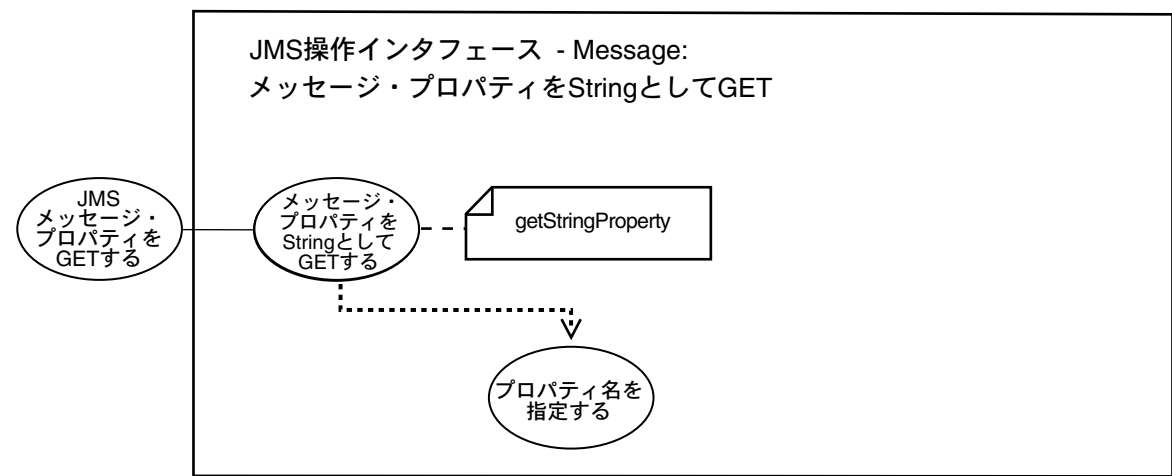
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# メッセージ・プロパティを String として取得

図 16-40 ユースケース図：メッセージ・プロパティを String として取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを String として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の getStringProperty

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

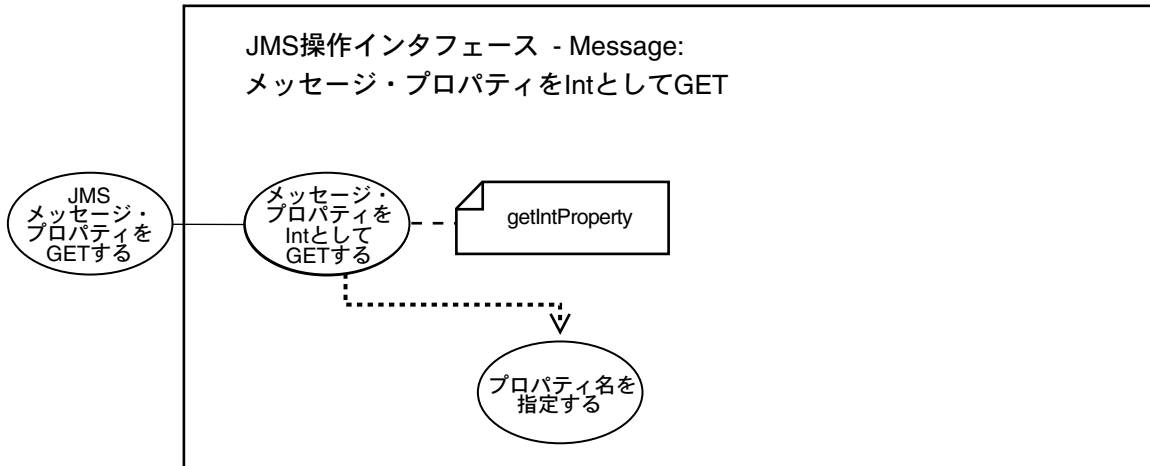
```
TextMessage message;
```

```
message.setStringProperty("JMS_OracleExcpQ", "scott.text_ecxcp_queue"); /* メッセージ  
に例外キューを設定します。*/
```

```
message.setStringProperty("color", "red"); /* ユーザー定義プロパティの color を設定します。  
*/
```

## メッセージ・プロパティを Int として取得

図 16-41 ユースケース図：メッセージ・プロパティを Int として取得



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。
- 
- 

## 用途

メッセージ・プロパティを Int として取得します。

## 使用上の注意

ありません。



## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の getIntProperty

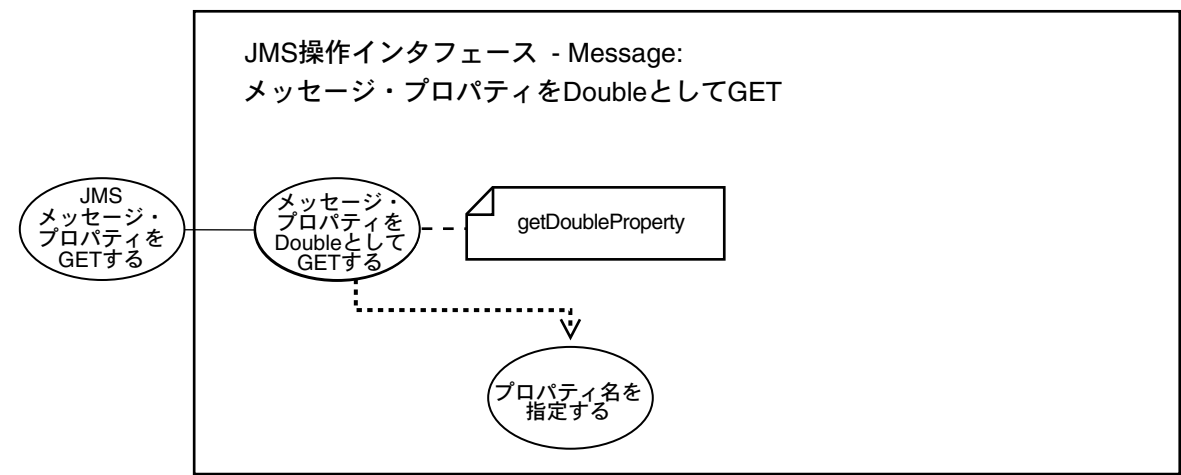
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

```
StreamMessage message;  
message.setIntProperty("MMS_OracleDelay", 10); /* メッセージの遅延に 10 秒を設定します。*/  
  
message.setIntProperty("empid", 1000); /* ユーザー定義プロパティ empId を設定します。*/
```

# メッセージ・プロパティを Double として取得

図 16-42 ユースケース図：メッセージ・プロパティを Double として取得



- 参照：**
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Double として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の `getDoubleProperty`

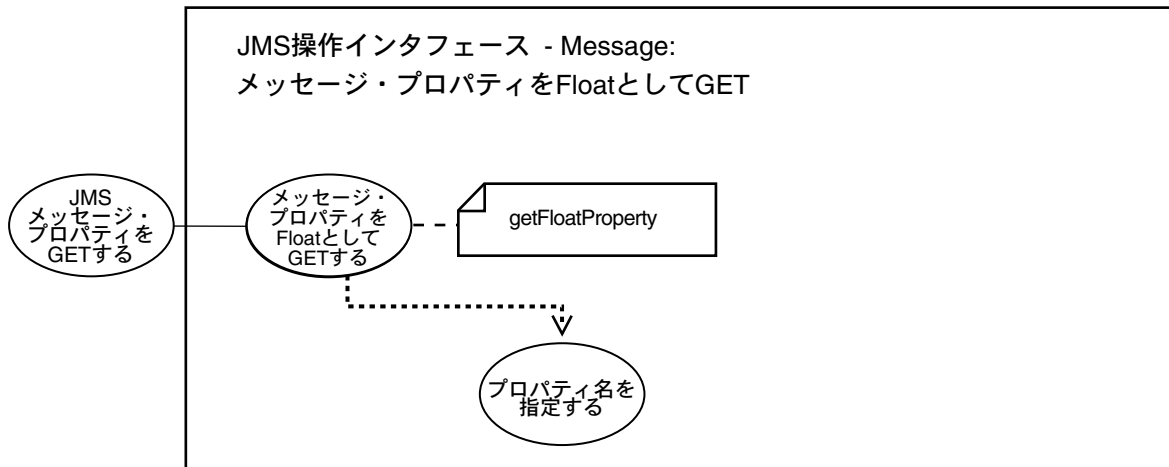
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを Float として取得

図 16-43 ユースケース図：メッセージ・プロパティを Float として取得



---

### 参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 

## 用途

メッセージ・プロパティを Float として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の `getFloatProperty`

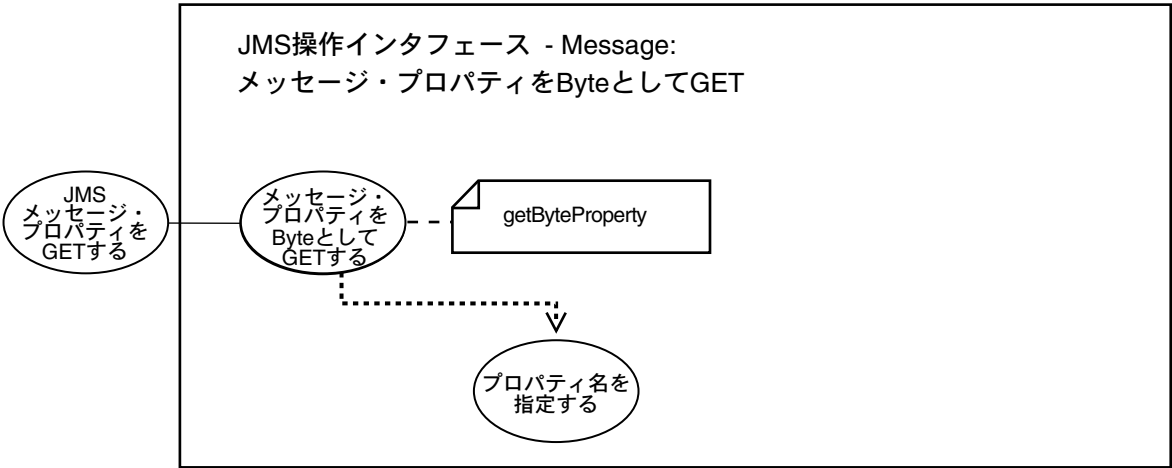
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを Byte として取得

図 16-44 ユースケース図：メッセージ・プロパティを Byte として取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース - 基本操作 (Point-to-Point)」を参照してください。

## 用途

メッセージ・プロパティを Byte として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の `getBytesProperty`

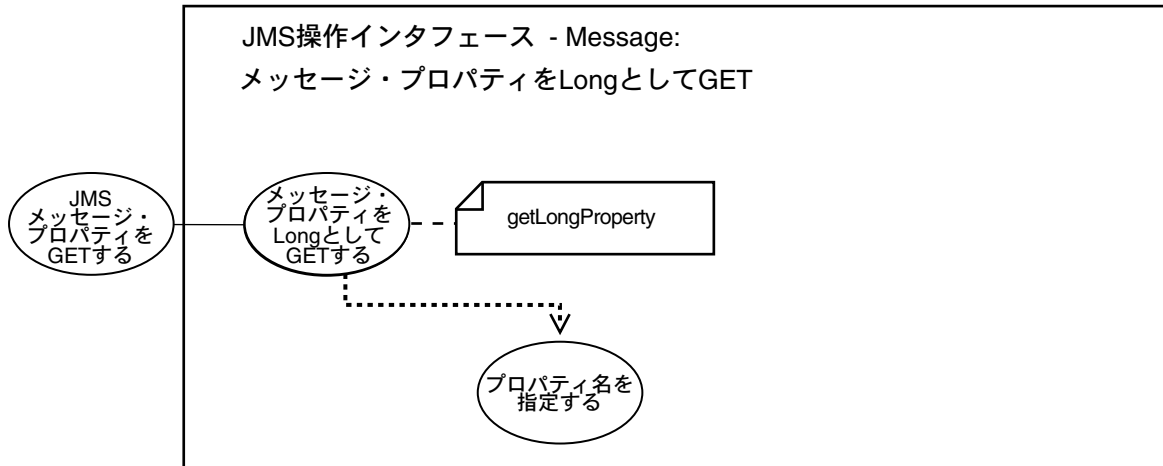
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを Long として取得

図 16-45 ユースケース図：メッセージ・プロパティを Long として取得



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル : JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージ・プロパティを Long として取得します。

## 使用上の注意

ありません。



## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の getLongProperty

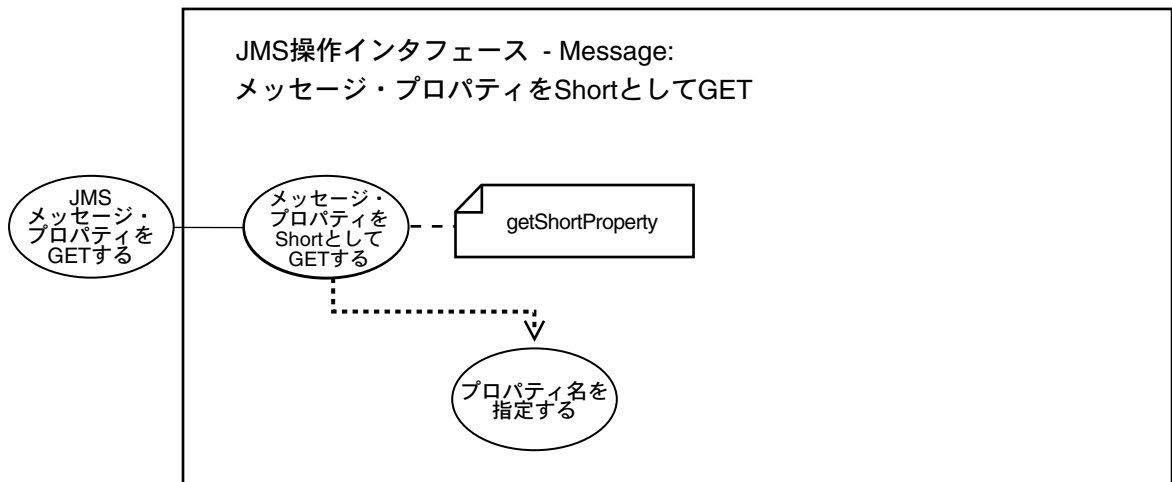
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを Short として取得

図 16-46 ユースケース図：メッセージ・プロパティを Short として取得



---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル: JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 

## 用途

メッセージ・プロパティを Short として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の getShortProperty

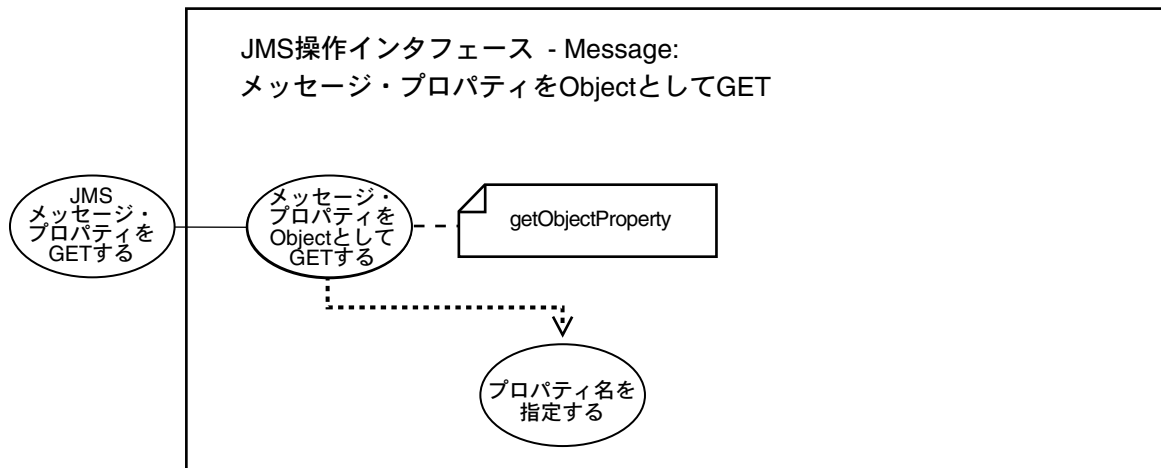
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・プロパティを Object として取得

図 16-47 ユースケース図：メッセージ・プロパティを Object として取得



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル：[JMS 操作インタフェース - 基本操作 \(Point-to-Point\)](#)」を参照してください。
- 
- 

## 用途

メッセージ・プロパティを Object として取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsMessage」の getObjectProperty

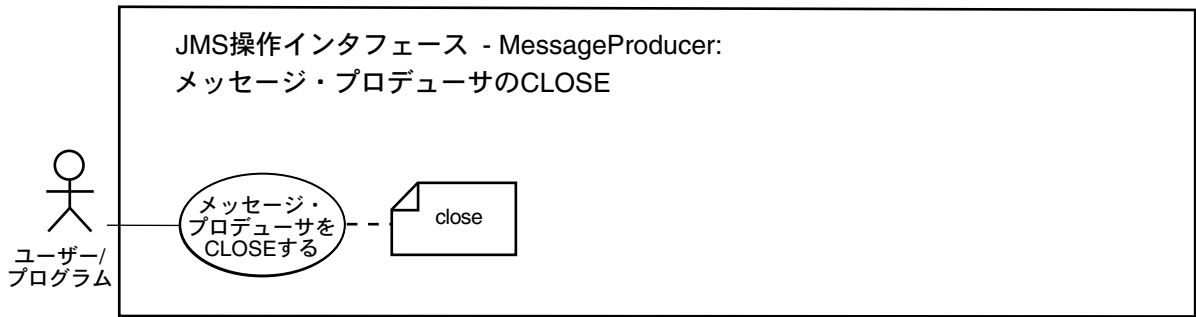
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

```
TextMessage message;  
message.setObjectProperty("empid", new Integer(1000));
```

# メッセージ・プロデューサのクローズ

図 16-48 ユースケース図：メッセージ・プロデューサのクローズ



- 参照：**
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル：JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。

## 用途

メッセージ・プロデューサをクローズします。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsProducer」の close

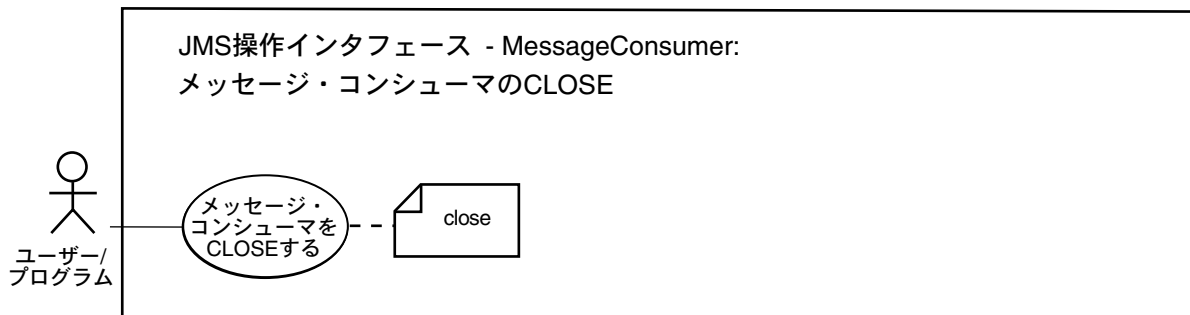
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

## メッセージ・コンシューマのクローズ

図 16-49 ユースケース図：メッセージ・コンシューマのクローズ



---

---

**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。
- 
- 

### 用途

メッセージ・コンシューマをクローズします。

### 使用上の注意

ありません。

### 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsConsumer」の close



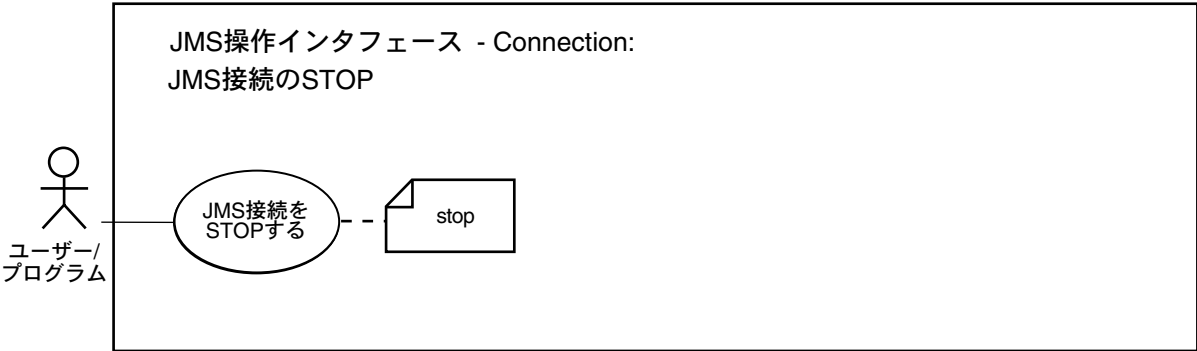
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS 接続の停止

図 16-50 ユースケース図 : JMS 接続の停止



- 参照：**
- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS 接続を停止します。

## 使用上の注意

このメソッドは、受信メッセージの接続配信を一時的に停止するために使用します。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsConnection」の stop

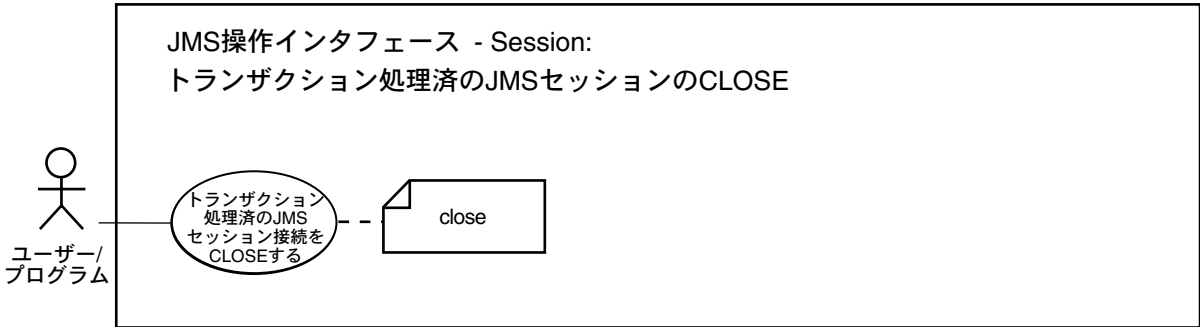
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS セッションのクローズ

図 16-51 ユースケース図：トランザクション処理済の JMS セッションのクローズ



参照：

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS セッションをクローズします。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsSession」の close

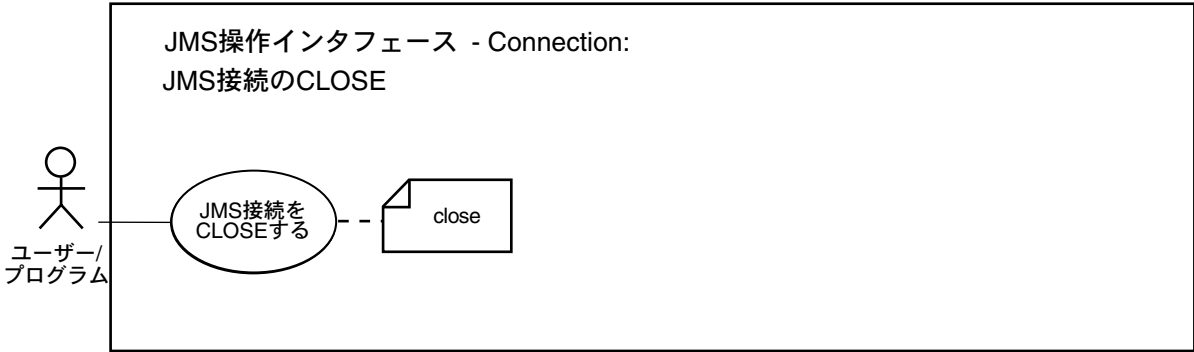
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS 接続のクローズ

図 16-52 ユースケース図 : JMS 接続のクローズ



**参照 :**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「[ユースケース・モデル : JMS 操作インタフェース – 基本操作 \(Point-to-Point\)](#)」を参照してください。

## 用途

JMS 接続をクローズします。

## 使用上の注意

このメソッドは接続をクローズし、接続のために割り当てられているすべてのリソースを解放します。通常、JMS のプロバイダは重要なリソースを接続のために JVM の外部に割り当てるため、クライアントは、これらのリソースが必要ない場合はクローズする必要があります。ガーベジ・コレクションに依存して、最終的にこれらのリソースを再生するのは、時間が不足する場合があります。

## 構文

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第2章「パッケージ oracle.jms」の「AQjmsConnection」の close

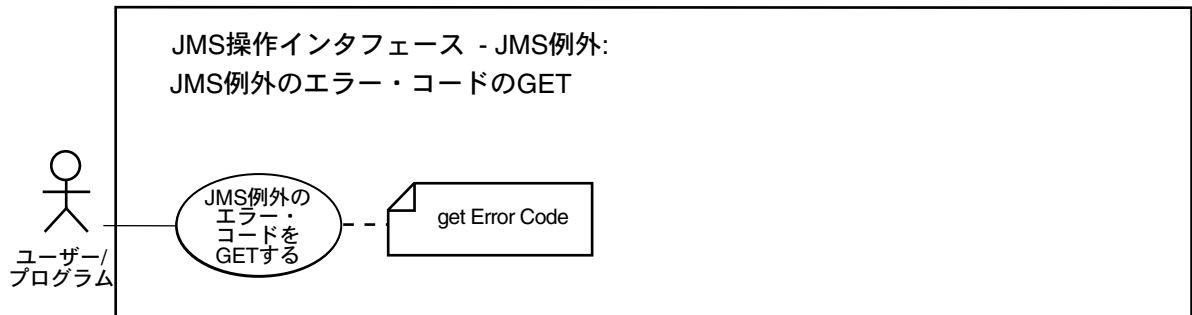
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS 例外のエラー・コードの取得

図 16-53 ユースケース図：JMS 例外のエラー・コードの取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル:JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS 例外のエラー・コードを取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsException」の getErrorCode



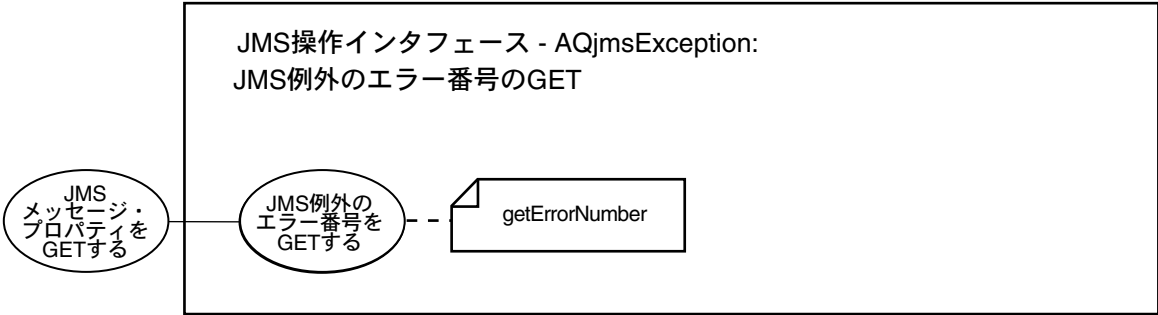
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS 例外のエラー番号の取得

図 16-54 ユースケース図 : JMS 例外のエラー番号の取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル : JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS 例外のエラー番号を取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC) : 『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsException」の getErrorNumber

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS 例外のエラー・メッセージの取得

図 16-55 ユースケース図：JMS 例外のエラー・メッセージの取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS 例外のエラー・メッセージを取得します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQJmsException」の getMessage

## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS 例外にリンクされた例外の取得

図 16-56 ユースケース図：JMS 例外にリンクされた例外の取得



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS 例外にリンクされた例外を取得します。

## 使用上の注意

このメソッドは、この JMS 例外にリンクされた例外を取得するために使用します。一般に、これにはデータベースが呼び出す SQL 例外が含まれます。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsException」の getLinkedException

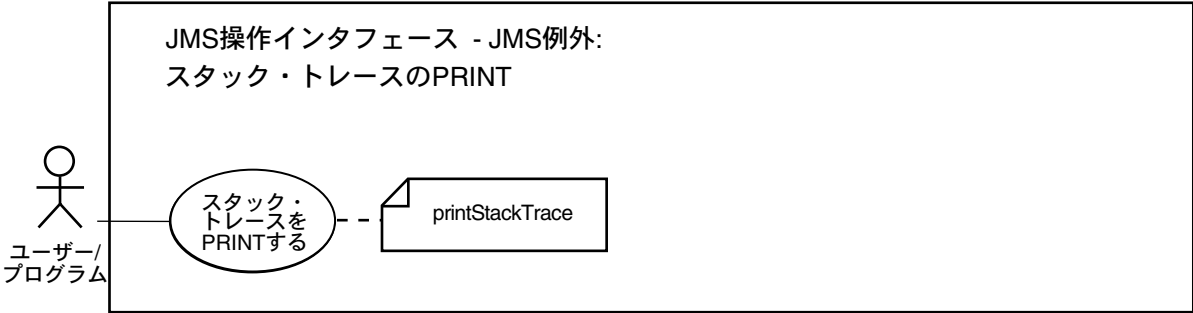
## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。

# JMS 例外のスタック・トレースの出力

図 16-57 ユースケース図：JMS 例外のスタック・トレースの出力



**参照：**

- 操作インタフェースに関するすべての基本操作については、14-2 ページの「ユースケース・モデル: JMS 操作インタフェース – 基本操作 (Point-to-Point)」を参照してください。

## 用途

JMS 例外のスタック・トレースを出力します。

## 使用上の注意

ありません。

## 構文

各プログラム環境で使用可能な機能のリストは、第 3 章「AQ プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- Java (JDBC)：『Oracle8i Java パッケージ・プロシージャ リファレンス』の第 2 章「パッケージ oracle.jms」の「AQjmsException」の printStackTrace



## 例

各プログラム環境で使用可能な機能のリストは、[第3章「AQ プログラム環境」](#)を参照してください。

今回のリリースでは、例は記載していません。



---

# Oracle アドバンスト・キューイングの使用例

この付録では、様々なプログラミング環境での使用例を掲載します。

- キュー・テーブルおよびキューの作成
  - オブジェクト型のキュー・テーブルおよびキューの作成
  - RAW 型のキュー・テーブルおよびキューの作成
  - 優先順位指定メッセージのキュー・テーブルおよびキューの作成
  - 複数コンシューマのキュー・テーブルおよびキューの作成
  - 伝播を実証するためのキューの作成
  - Java AQ の例の設定
  - Java AQ セッションの作成
  - Java を使用したキュー・テーブルおよびキューの作成
  - Java を使用したキューの作成およびエンキュー / デキューの開始
  - Java を使用した複数コンシューマ・キューの作成およびサブスクライバの追加
- メッセージのエンキューおよびデキュー
  - PL/SQL を使用したオブジェクト型メッセージのエンキューおよびデキュー
  - Pro\*C/C++ を使用したオブジェクト型メッセージのエンキューおよびデキュー
  - OCI を使用したオブジェクト型メッセージのエンキューおよびデキュー
  - Java を使用したオブジェクト型メッセージ (CustomDatum インタフェース) のエンキューおよびデキュー

- 
- Java を使用したオブジェクト型メッセージ（SQLData インタフェース使用）のエンキューおよびデキュー
  - PL/SQL を使用した RAW 型メッセージのエンキューおよびデキュー
  - Pro\*C/C++ を使用した RAW 型メッセージのエンキューおよびデキュー
  - OCI を使用した RAW 型メッセージのエンキューおよびデキュー
  - Java を使用した RAW 型メッセージのエンキュー
  - Java を使用したメッセージのデキュー
  - Java を使用したブラウズ・モードでのメッセージのデキュー
  - PL/SQL を使用した優先順位によるメッセージのエンキューおよびデキュー
  - Java を使用した優先順位によるメッセージのエンキュー
  - PL/SQL を使用したプレビュー後のメッセージのデキュー
  - PL/SQL を使用した遅延および期限切れによるメッセージのエンキューおよびデキュー
  - Pro\*C/C++ を使用した相関識別子およびメッセージ ID によるメッセージのエンキューおよびデキュー
  - OCI を使用した相関識別子およびメッセージ ID によるメッセージのエンキューおよびデキュー
  - PL/SQL を使用した複数コンシューマ・キューでのメッセージのエンキューおよびデキュー
  - OCI を使用した複数コンシューマ・キューでのメッセージのエンキューおよびデキュー
  - PL/SQL を使用したメッセージのグループ化によるメッセージのエンキューおよびデキュー
  - PL/SQL を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー
  - Java を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー
- 伝播
- PL/SQL を使用したリモートのサブスクライバ / 受信者用のメッセージの複数コンシューマ・キューへのエンキューおよび伝播のスケジュール
  - PL/SQL を使用した同一データベース内の 1 つのキューから他のキューへの伝播管理

- 
- PL/SQL を使用した 1 つのキューから他のデータベース内の他のキューへの伝播管理
  - PL/SQL を使用した伝播スケジュールの解除
  - AQ オブジェクトの削除
  - ロールおよび権限の取消し
  - AQ による XA の使用
  - AQ およびメモリーの使用
    - OCI を使用したメッセージのエンキュー（各コール後のメモリー解放）
    - OCI を使用したメッセージのエンキュー（メモリーの再使用）
    - OCI を使用したメッセージのデキュー（各コール後のメモリー解放）
    - OCI を使用したメッセージのデキュー（メモリーの再使用）

## キュー・テーブルおよびキューの作成

---

**注意：** 次のような SQL 文を実行しないと機能しない場合があります。

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
GRANT CONNECT, RESOURCE TO aqadm;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
```

---

## オブジェクト型のキュー・テーブルおよびキューの作成

```
/* メッセージ型を作成します。*/
CREATE type aq.Message_typ as object (
  subject    VARCHAR2(30),
  text       VARCHAR2(80));

/* オブジェクト型のキュー・テーブルおよびキューを作成します。*/
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
  queue_table      => 'aq.objmsgs80_qtab',
  queue_payload_type => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
  queue_name       => 'msg_queue',
  queue_table      => 'aq.objmsgs80_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
  queue_name       => 'msg_queue');
```

## RAW 型のキュー・テーブルおよびキューの作成

```
/* RAW 型のキュー・テーブルおよびキューを作成します。*/
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table          => 'aq.RawMsgs_qtab',
queue_payload_type   => 'RAW');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name           => 'raw_msg_queue',
queue_table          => 'aq.RawMsgs_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name           => 'raw_msg_queue');
```

## 優先順位指定メッセージのキュー・テーブルおよびキューの作成

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table          => 'aq.priority_msg',
sort_list            => 'PRIORITY,ENQ_TIME',
queue_payload_type   => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name           => 'priority_msg_queue',
queue_table          => 'aq.priority_msg');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name           => 'priority_msg_queue');
```

## 複数コンシューマのキュー・テーブルおよびキューの作成

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table          => 'aq.MultiConsumerMsgs_qtab',
multiple_consumers   => TRUE,
queue_payload_type   => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name           => 'msg_queue_multiple',
queue_table          => 'aq.MultiConsumerMsgs_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name           => 'msg_queue_multiple');
```

## 伝播を実証するためのキューの作成

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (  
queue_name      => 'another_msg_queue',  
queue_table     => 'aq.MultiConsumerMsgs_qtab');  
  
EXECUTE DBMS_AQADM.START_QUEUE (  
queue_name      => 'another_msg_queue');
```

## Java AQ の例の設定

```
CONNECT system/manager  
  
DROP USER aqjava CASCADE;  
GRANT CONNECT, RESOURCE, AQ_ADMINISTRATOR_ROLE TO aqjava IDENTIFIED BY aqjava;  
GRANT EXECUTE ON DBMS_AQADM TO aqjava;  
GRANT EXECUTE ON DBMS_AQ TO aqjava;  
CONNECT aqjava/aqjava  
  
/* 後続の例およびハンドル例外のコール元となるメイン・クラスを設定します。*/  
import java.sql.*;  
import oracle.AQ.*;  
  
public class test_aqjava  
{  
    public static void main(String args[])  
    {  
        AQSession aq_sess = null;  
        try  
        {  
            aq_sess = createSession(args);  
  
            /* テストを実行します。*/  
            runTest(aq_sess);  
        }  
        catch (Exception ex)  
        {  
            System.out.println("Exception-1: " + ex);  
            ex.printStackTrace();  
        }  
    }  
}
```



## Java AQ セッションの作成

```
/* 前述の AQDriverManager の項で示した 'aqjava' ユーザーに Java AQ セッションを作成します。*/
public static AQSession createSession(String args[])
{
    Connection db_conn;
    AQSession aq_sess = null;

    try
    {

        Class.forName("oracle.jdbc.driver.OracleDriver");
        /* 実際のホスト名、ポート番号およびSIDは、次のものとは異なります。ここでは、明示
           的に 'dlsun736'、'5521' および 'test' を指定します。*/

        db_conn =
            DriverManager.getConnection(
                "jdbc:oracle:thin:@dlsun736:5521:test",
                "aqjava", "aqjava");

        System.out.println("JDBC Connection opened ");
        db_conn.setAutoCommit(false);

        /* Oracle8i AQ ドライバをロードします。*/
        Class.forName("oracle.AQ.AQOracleDriver");

        /* AQ セッションを作成します。*/
        aq_sess = AQDriverManager.createAQSession(db_conn);
        System.out.println("Successfully created AQSession ");
    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
        ex.printStackTrace();
    }
    return aq_sess;
}
```

## Java を使用したキュー・テーブルおよびキューの作成

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;

    /* AQQueueTableProperty オブジェクト (ペイロード型 RAW) を作成します。*/
    qtable_prop = new AQQueueTableProperty("RAW");

    /* aqjava スキーマにキュー・テーブル aq_table1 を作成します。*/
    q_table = aq_sess.createQueueTable ("aqjava", "aq_table1", qtable_prop);
    System.out.println("Successfully created aq_table1 in aqjava schema");

    /* 新しいAQQueueProperty オブジェクトを作成します。*/
    queue_prop = new AQQueueProperty();

    /* aq_table1 にキュー aq_queue1 を作成します。*/
    queue = aq_sess.createQueue (q_table, "aq_queue1", queue_prop);
    System.out.println("Successfully created aq_queue1 in aq_table1");
}

/* 既存のキュー・テーブルおよびキューへのハンドルを取得します。*/
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable            q_table;
    AQQueue                 queue;

    /* aqjava スキーマのキュー・テーブル aq_table1 へのハンドルを取得します。*/
    q_table = aq_sess.getQueueTable ("aqjava", "aq_table1");
    System.out.println("Successful getQueueTable");

    /* aqjava スキーマのキュー aq_queue1 へのハンドルを取得します。*/
    queue = aq_sess.getQueue ("aqjava", "aq_queue1");
    System.out.println("Successful getQueue");
}
```

## Java を使用したキューの作成およびエンキュー / デキューの開始

```
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;

    /* AQQueueTable プロパティ・オブジェクト（バイロード型 RAW）を作成します。*/
    qtable_prop = new AQQueueTableProperty("RAW");
    qtable_prop.setCompatible("8.1");

    /* aqjava スキーマにキュー・テーブル aq_table3 を作成します。*/
    q_table = aq_sess.createQueueTable ("aqjava", "aq_table3", qtable_prop);
    System.out.println("Successful createQueueTable");

    /* 新しいAQQueueProperty オブジェクトを作成します。*/
    queue_prop = new AQQueueProperty();

    /* aq_table3 にキュー aq_queue3 を作成します。*/
    queue = aq_sess.createQueue (q_table, "aq_queue3", queue_prop);
    System.out.println("Successful createQueue");

    /* このキューに対するエンキュー / デキューを使用可能にします。*/
    queue.start();
    System.out.println("Successful start queue");

    /* ユーザー scott に、このキューに対する enqueue_any 権限を付与します。*/
    queue.grantQueuePrivilege("ENQUEUE", "scott");
    System.out.println("Successful grantQueuePrivilege");
}
```

## Java を使用した複数コンシューマ・キューの作成およびサブスクライバの追加

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;
    AQAgent                 subs1, subs2;

    /* AQQueueTable プロパティ・オブジェクト（バイロード型 RAW）を作成します。*/
    qtable_prop = new AQQueueTableProperty("RAW");
    System.out.println("Successful setCompatible");
```

```
/* 複数コンシューマのフラグを TRUE に設定します。*/
qtable_prop.setMultiConsumer(true);

/* aqjava スキーマにキュー・テーブル aq_table4 を作成します。*/
q_table = aq_sess.createQueueTable ("aqjava", "aq_table4", qtable_prop);
System.out.println("Successful createQueueTable");

/* 新しいAQQueueProperty オブジェクトを作成します。*/
queue_prop = new AQQueueProperty();
/* aq_table4 にキュー aq_queue4 を作成します。*/
queue = aq_sess.createQueue (q_table, "aq_queue4", queue_prop);
System.out.println("Successful createQueue");

/* このキューに対するエンキュー / デキューを使用可能にします。*/
queue.start();
System.out.println("Successful start queue");

/* このキューにサブスクライバを追加します。*/
subs1 = new AQAgent("GREEN", null, 0);
subs2 = new AQAgent("BLUE", null, 0);

queue.addSubscriber(subs1, null); /* ルールはありません。*/
System.out.println("Successful addSubscriber 1");

queue.addSubscriber(subs2, "priority < 2"); /* ルールがあります。*/
System.out.println("Successful addSubscriber 2");
}
```

## メッセージのエンキューおよびデキュー

### PL/SQL を使用したオブジェクト型メッセージのエンキューおよびデキュー

他のパラメータを使用せずに 1 つのメッセージをエンキューするには、キュー名およびペイロードを指定します。

```
/* msg_queue にエンキューします。*/
DECLARE
    enqueue_options      dbms_aq.enqueue_options_t;
    message_properties    dbms_aq.message_properties_t;
    message_handle        RAW(16);
    message               aq.message_typ;

BEGIN
    message := message_typ('NORMAL MESSAGE',
        'enqueued to msg_queue first. ');

    dbms_aq.enqueue(queue_name => 'msg_queue',
        enqueue_options      => enqueue_options,
        message_properties    => message_properties,
        payload               => message,
        msgid                 => message_handle);

    COMMIT;

/* msg_queue からデキューします。*/
DECLARE
    dequeue_options      dbms_aq.dequeue_options_t;
    message_properties    dbms_aq.message_properties_t;
    message_handle        RAW(16);
    message               aq.message_typ;

BEGIN
    DBMS_AQ.DEQUEUE(queue_name => 'msg_queue',
        dequeue_options      => dequeue_options,
        message_properties    => message_properties,
        payload               => message,
        msgid                 => message_handle);

    DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
        ' ... ' || message.text );

    COMMIT;
END;
```

## Pro\*C/C++ を使用したオブジェクト型メッセージのエンキューおよびデキュー

---

**注意：** 次のような設定を実行しないと機能しない例もあります。

```
$ cat >> message.typ
case=lower
type aq.message_typ
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \ code=c
hfile=demo.h
$
$ proc intyp=message_o.typ iname=<program name> \
config=<config file> SQLCHECK=SEMANTICS userid=aq/aq
```

---

```
#include <stdio.h>
#include <string.h>
#include <sqlca.h>
#include <sql2oci.h>
/* object type オブジェクト型 'aq.Message_typ' を処理することによって生成されるヘッダー・
ファイル */
#include "pceg.h"

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
Message_typ      *message = (Message_typ*)0; /* ペイロード */
char              user[60]="aq/AQ"; /* ユーザー・ログイン・パスワード */
char              subject[30]; /* コンポーネント */
char              txt[80]; /* ペイロード */

/* OBJECT QUEUE への ENQUEUE および DEQUEUE */

/* データベースに接続します。*/
EXEC SQL CONNECT :user;
```

```
/* Oracle エラーが発生すると、エラー番号を印刷します。*/
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* オブジェクト・キャッシュからホスト変数にメモリーを割り当てます。*/
EXEC SQL ALLOCATE :message;

/* エンキューします。*/

strcpy(subject, "NORMAL ENQUEUE");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* メッセージのコンポーネントを初期化します。*/
EXEC SQL OBJECT SET subject, text OF :message TO :subject, :txt;

/* 埋込み PL/SQL が AQ エンキュー・プロシージャをコールします。*/
EXEC SQL EXECUTE
DECLARE
message_properties    dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
msgid                RAW(16);
BEGIN
/* ホスト変数 'message' をペイロードにバインドします。*/
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;

printf("Enqueued Message \n");
printf("Subject   :%s\n",subject);
printf("Text      :%s\n",txt);

/* デキューします。*/

/* 埋込み PL/SQL が AQ デキュー・プロシージャをコールします。*/
EXEC SQL EXECUTE
DECLARE
message_properties    dbms_aq.message_properties_t;
dequeue_options       dbms_aq.dequeue_options_t;
msgid                RAW(16);
```

```
BEGIN
/* ホスト変数 'message' にペイロードを戻します。*/
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;

/* メッセージのコンポーネントを抽出します。*/
EXEC SQL OBJECT GET SUBJECT,TEXT FROM :message INTO :subject,:txt;

printf("Dequeued Message \n");
printf("Subject   :%s\n",subject);
printf("Text      :%s\n",txt);
}
```

## OCI を使用したオブジェクト型メッセージのエンキューおよびデキュー

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
    OCISString    *subject;
    OCISString    *data;
};
typedef struct message message;

struct null_message
{
    OCIIInd      null_adt;
    OCIIInd      null_subject;
    OCIIInd      null_data;
};
typedef struct null_message null_message;

int main()
{
```



```

OCIEnv      *envhp;
OCIServer   *srvhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
dvoid       *tmp;
OCIType     *mesg_tdo = (OCIType *) 0;
message     msg;
null_message nmsg;
message     *mesg      = &msg;
null_message *nmesg    = &nmsg;
message     *deqmesg   = (message *) 0;
null_message *ndeqmesg = (null_message *) 0;

OCIInitialize((ub4) OCI_OBJECT, (dvoid *) 0, (dvoid * (*)()) 0,
              (dvoid * (*)()) 0, (void (*)()) 0 );

OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit(&envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* message_typ の TDO を取得します。*/
OCITypeByName(envhp, errhp, svchp, (CONST text *) "AQ", strlen("AQ"),
              (CONST text *) "MESSAGE_TYP", strlen("MESSAGE_TYP"),
              (text *) 0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* メッセージ・ペイロードを準備します。*/
mesg->subject = (OCIStrng *) 0;
mesg->data = (OCIStrng *) 0;
OCIStrngAssignText(envhp, errhp,

```

```
(CONST text *)"NORMAL MESSAGE", strlen("NORMAL MESSAGE"),
&mesg->subject);

OCIStrAssignText(envhp, errhp,
    (CONST text *)"OCI ENQUEUE", strlen("OCI ENQUEUE"),
    &mesg->data);
nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* msg_queue にエンキューします。*/
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
    mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* msg_queue からデキューします。*/
OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
    mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0);
printf("Subject: %s\n", OCIStrPtr(envhp, deqmesg->subject));
printf("Text: %s\n", OCIStrPtr(envhp, deqmesg->data));
OCITransCommit(svchp, errhp, (ub4) 0);
}
```

## Java を使用したオブジェクト型メッセージ (CustomDatum インタフェース) のエンキューおよびデキュー

オブジェクト型のメッセージをエンキューおよびデキューするには、次のステップに従います。

- a. キュー・ペイロードに対する SQL 型を作成します。

```
connect aquser/aquser
create type ADDRESS as object (street VARCHAR (30), city VARCHAR(30));
create type PERSON as object (name VARCHAR (30), home ADDRESS);
```

- b. PERSON オブジェクト型に対応した、CustomDatum インタフェースを実装する Java クラスを (JPublisher ツールを使用して) 生成します。

```
jpub -user=aquser/aquser -sql=ADDRESS,PERSON -case=mixed -usertypes=oracle
-methods=false
```

これにより、PERSON および ADDRESS オブジェクト型に対応する 2 つのクラス、PERSON.java および ADDRESS.java が作成されます。

- c. オブジェクト型ペイロードで、キュー・テーブルおよびキューを作成します。
- d. オブジェクト型ペイロードを含むメッセージをエンキューおよびデキューします。

```
public static void AQObjectPayloadTest(AQSession aq_sess)
    throws AQException, SQLException, ClassNotFoundException
{
    Connection          db_conn   = null;
    AQQueue             queue     = null;
    AQMessage           message   = null;
    AQObjectPayload      payload   = null;
    AQEnqueueOption     eq_option = null;
    AQDequeueOption     dq_option = null;
    PERSON              pers      = null;
    PERSON              pers2     = null;
    ADDRESS              addr      = null;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    queue = aq_sess.getQueue("aquser", "test_queue2");

    /* このキューへのエンキュー / デキューを使用可能にします。*/
    queue.start();

    /* test_queue2 のメッセージをエンキューします。*/
    message = queue.createMessage();

    pers = new PERSON();
    pers.setName("John");
    addr = new ADDRESS();
    addr.setStreet("500 Easy Street");
    addr.setCity("San Francisco");
    pers.setHome(addr);

    payload = message.getObjectPayload();
    payload.setPayloadData(pers);
    eq_option = new AQEnqueueOption();

    /* test_queue2 にメッセージをエンキューします。*/
    queue.enqueue(eq_option, message);

    db_conn.commit();

    /* test_queue2 からメッセージをデキューします。*/
    dq_option = new AQDequeueOption();
    message = ((AQOracleQueue)queue).dequeue(dq_option, PERSON.getFactory());
}
```

```
payload = message.getObjectPayload();
pers2 = (PERSON) payload.getPayloadData();

System.out.println("Object data retrieved: [PERSON]");
System.out.println("Name:   " + pers2.getName());
System.out.println("Address ");
System.out.println("Street: " + pers2.getHome().getStreet());
System.out.println("City:   " + pers2.getHome().getCity());

db_conn.commit();
}
```

## Java を使用したオブジェクト型メッセージ（SQLData インタフェース使用）のエンキューおよびデキュー

オブジェクト型のメッセージをエンキューおよびデキューするには、次のステップに従います。

- a. キュー・ペイロードに対する SQL 型を作成します。

```
connect aquser/aquser
create type EMPLOYEE as object (empname VARCHAR (50), empno INTEGER);
```

- b. EMPLOYEE オブジェクト型に対応した、SQLData インタフェースを実装する Java クラスを生成します。このクラスは、次の構文を使用して、JPublisher で生成することもできます。

```
jpub -user=aquser/aquser -sql=EMPLOYEE -case=mixed -usertypes=jdbc
-methods=false
```

```
import java.sql.*;
import oracle.jdbc2.*;

public class Employee implements SQLData
{
    private String sql_type;
    public String empName;
    public int empNo;
    public Employee()
    {}
    public Employee (String sql_type, String empName, int empNo)
    {
        this.sql_type = sql_type;
        this.empName = empName;
    }
}
```

```

        this.empNo = empNo;
    }

    ///// implements SQLData /////
    public String getSQLTypeName() throws SQLException
    { return sql_type;
    }

    public void readSQL(SQLInput stream, String typeName)
        throws SQLException
    {
        sql_type = typeName;
        empName = stream.readString();
        empNo = stream.readInt();
    }

    public void writeSQL(SQLOutput stream)
        throws SQLException
    {
        stream.writeString(empName);
        stream.writeInt(empNo);
    }

    public String toString()
    {
        String ret_str = "";
        ret_str += "[Employee]\n";
        ret_str += "Name: " + empName + "\n";
        ret_str += "Number: " + empNo + "\n";

        return ret_str;
    }
}

```

- c. オブジェクト型ペイロードで、キュー・テーブルおよびキューを作成します。

```

public static void createEmployeeObjQueue(AQSession aq_sess)
    throws AQException
{
    AQQueueTableProperty qt_prop = null;
    AQQueueProperty      q_prop  = null;
    AQQueueTable          q_table = null;
    AQQueue               queue   = null;

    /* メッセージ・ペイロード型は aquser.EMPLOYEE です。 */
    qt_prop = new AQQueueTableProperty("AQUSER.EMPLOYEE");
    qt_prop.setComment("queue-table1");
}

```

```

/* aqTable1 を作成します。*/
System.out.println("\nCreate QueueTable: [aqtable1]");
q_table = aq_sess.createQueueTable("aquser", "aqtable1", qt_prop);

/* test_queue1 を作成します。*/
q_prop = new AQQueueProperty();
queue = q_table.createQueue("test_queue1", q_prop);

/* このキューへのエンキュー / デキューを使用可能にします。*/
queue.start();
}

```

- d. オブジェクト型ペイロードを含むメッセージをエンキューおよびデキューします。

```

public static void AQObjectPayloadTest2(AQSession aq_sess)
    throws AQException, SQLException, ClassNotFoundException
{
    Connection          db_conn    = null;
    AQQueue             queue      = null;
    AQMessage           message    = null;
    AQObjectPayload     payload    = null;
    AQEnqueueOption     eq_option  = null;
    AQDequeueOption     dq_option  = null;
    Employee            emp        = null;
    Employee            emp2       = null;
    Hashtable            map;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    /* キュー・オブジェクトを取得します。*/
    queue = aq_sess.getQueue("aquser", "test_queue1");

    /* (EMPLOYEE オブジェクト型に対応する) Employee クラスを接続タイプ・マップに登録
    します。*/
    try
    {
        map = (java.util.Hashtable)((OracleConnection)db_conn).getTypeMap();
        map.put("AQUSER.EMPLOYEE", Class.forName("Employee"));
    }
    catch(Exception ex)
    {
        System.out.println("Error registering type: " + ex);
    }
}

```

```

/* test_queue1 のメッセージをエンキューします。*/
message = queue.createMessage();
emp = new Employee("AQUSER.EMPLOYEE", "Mark", 1007);

/* オブジェクト型ペイロードを設定します。*/
payload = message.getObjectPayload();
payload.setPayloadData(emp);

/* test_queue1 にメッセージをエンキューします。*/
eq_option = new AQEnqueueOption();
queue.enqueue(eq_option, message);
db_conn.commit();

/* test_queue1 からメッセージをデキューします。*/
dq_option = new AQDequeueOption();

message = queue.dequeue(dq_option, Class.forName("Employee"));
payload = message.getObjectPayload();
emp2 = (Employee) payload.getPayloadData();
System.out.println("\nObject data retrieved:  [EMPLOYEE] ");
System.out.println("Name   : " + emp2.empName);
System.out.println("EmpId  : " + emp2.empNo);

db_conn.commit();
}

```

## PL/SQL を使用した RAW 型メッセージのエンキューおよびデキュー

```

DECLARE
    enqueue_options      dbms_aq.enqueue_options_t;
    message_properties    dbms_aq.message_properties_t;
    message_handle        RAW(16);
    message               RAW(4096);

BEGIN
    message := HEXTORAW(RPAD('FF',4095,'FF'));
    DBMS_AQ.ENQUEUE(queue_name => 'raw_msg_queue',
        enqueue_options      => enqueue_options,
        message_properties    => message_properties,
        payload               => message,
        msgid                 => message_handle);

```

```
        COMMIT;
    END;

    /* raw_msg_queue からデキューします。*/
    /* raw_msg_queue からデキューします。*/
    DECLARE
        dequeue_options    DBMS_AQ.dequeue_options_t;
        message_properties  DBMS_AQ.message_properties_t;
        message_handle      RAW(16);
        message             RAW(4096);

    BEGIN
        DBMS_AQ.DEQUEUE(queue_name => 'raw_msg_queue',
            dequeue_options => dequeue_options,
            message_properties => message_properties,
            payload          => message,
            msgid            => message_handle);

        COMMIT;
    END;
```

## Pro\*C/C++ を使用した RAW 型メッセージのエンキューおよびデキュー

---

---

**注意：** 次のような設定を実行しないと機能しない例もあります。

```
$ cat >> message.typ
case=lower
type aq.message_type
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \ code=c
hfile=demo.h
$
$ proc intyp=message_o.typ iname=<program name> \
config=<config file> SQLCHECK=SEMANTICS userid=aq/aq
```

---

---

```
#include <stdio.h>
#include <string.h>
#include <sqlca.h>
#include <sql2oci.h>

void sql_error(msg)
char *msg;
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
```



```

printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
OCIEnv          *oeh;   /* OCI 環境ハンドル */
OCIError        *err;   /* OCI エラー・ハンドル */
OCIRaw          *message= (OCIRaw*)0; /* ペイロード */
ub1             message_txt[100]; /* ペイロードのデータ */
char            user[60]="aq/AQ"; /* ユーザー・ログイン・パスワード */
int             status; /* OCI コールの状態を返します。*/

/* RAW キューに対してエンキューおよびデキューを行います。*/

/* データベースに接続します。*/
EXEC SQL CONNECT :user;

/* Oracle エラーが発生すると、エラー番号を印刷します。*/
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* OCI 環境ハンドルを取得します。*/
if (SQLEnvGet(SQL_SINGLE_RCTX, &oeh) != OCI_SUCCESS)
{
printf(" error in SQLEnvGet \n");
exit(1);
}
/* OCI エラー・ハンドルを取得します。*/
if (status = OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
(ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)&0))
{
printf(" error in OCIHandleAlloc %d \n", status);
exit(1);
}

/* エンキューします。*/
/* The bytes to be put into the raw payload:*/
strcpy(message_txt, "Enqueue to a Raw payload queue ");

/* OCIRaw ポインタにバイトを割り当てます。
メモリーは、明示的に OCIRaw* に割り当てる必要があります。*/
if (status=OCIRawAssignBytes(oeh, err, message_txt, 100,
&message))
{
printf(" error in OCIRawAssignBytes %d \n", status);

```

```
exit(1);
}

/* 埋込み PL/SQL が AQ エンキュー・プロシージャをコールします。*/
EXEC SQL EXECUTE
DECLARE
message_properties    dbms_aq.message_properties_t;
enqueue_options       dbms_aq.enqueue_options_t;
msgid                 RAW(16);
BEGIN
/* ホスト変数メッセージを RAW ペイロードにバインドします。*/
dbms_aq.enqueue(queue_name => 'raw_msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;

/* デキューします。*/
/* 埋込み PL/SQL が AQ デキュー・プロシージャをコールします。*/
EXEC SQL EXECUTE
DECLARE
message_properties    dbms_aq.message_properties_t;
dequeue_options       dbms_aq.dequeue_options_t;
msgid                 RAW(16);
BEGIN
/* ホスト変数 'message' に RAW ペイロードを戻します。*/
dbms_aq.dequeue(queue_name => 'raw_msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;
}
```

## OCI を使用した RAW 型メッセージのエンキューおよびデキュー

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

int main()
{
    OCIEnv      *envhp;
    OCIServer    *srvhp;
    OCIError     *errhp;
    OCISvcCtx    *svchp;
    dvoid        *tmp;
    OCIType      *mesg_tdo = (OCIType *) 0;
    char         msg_text[100];
    OCIRaw       *mesg = (OCIRaw *) 0;
    OCIRaw       *deqmesg = (OCIRaw *) 0;
    OCIInd       ind = 0;
    dvoid        *indptry = (dvoid *)&ind;
    int          i;

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *) 0, (dvoid * (*)()) 0,
                  (dvoid * (*)()) 0, (void (*)()) 0 );

    OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                   52, (dvoid **) &tmp);

    OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp );

    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                   52, (dvoid **) &tmp);
    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                   52, (dvoid **) &tmp);

    OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                   52, (dvoid **) &tmp);

    OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) srvhp, (ub4) 0,
               (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

```

```
OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* RAW データ型の TDO を取得します。*/
OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQADM", strlen("AQADM"),
              (CONST text *)"RAW", strlen("RAW"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* メッセージ・ペイロードを準備します。*/
strcpy(msg_text, "Enqueue to a RAW queue");
OCIRawAssignBytes(envhp, errhp, msg_text, strlen(msg_text), &mesg);

/* raw_msg_queue にメッセージをエンキューします。*/
OCIAQEng(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&mesg, (dvoid **)&indptr, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* C 変数 degmesg に同じメッセージをデキューします。*/
OCIAQDeq(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&degmesg, (dvoid **)&indptr, 0, 0);
for (i = 0; i < OCIRawSize(envhp, degmesg); i++)
    printf("%c", *(OCIRawPtr(envhp, degmesg) + i));
OCITransCommit(svchp, errhp, (ub4) 0);
}
```

## Java を使用した RAW 型メッセージのエンキュー

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable      q_table;
    AQQueue           queue;
    AQMessage          message;
    AQRawPayload       raw_payload;
    AQEnqueueOption    enq_option;
    String             test_data = "new message";
    byte[]             b_array;
    Connection         db_conn;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    /* aqjava スキーマのキュー・テーブル aq_table4 へのハンドルを取得します。*/
    q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
    System.out.println("Successful getQueueTable");
}
```

```

/* aquser スキーマのキュー aq_queue4 へのハンドルを取得します。*/
queue = aq_sess.getQueue ("aqjava", "aq_queue4");
System.out.println("Successful getQueue");

/*RAW ペイロードを含むメッセージを作成します。*/
message = queue.createMessage();

/* AQRawPayload オブジェクトへのハンドルを取得し、RAW データとともに移入します。*/
b_array = test_data.getBytes();

raw_payload = message.getRawPayload();

raw_payload.setStream(b_array, b_array.length);

/* AQEnqueueOption オブジェクトをデフォルトのオプション付きで作成します。*/
enq_option = new AQEnqueueOption();
/* メッセージをエンキューします。*/
queue.enqueue(enq_option, message);

db_conn.commit();
}

```

## Java を使用したメッセージのデキュー

```

public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable      q_table;
    AQQueue            queue;
    AQMessage          message;
    AQRawPayload       raw_payload;
    AQEnqueueOption    enq_option;
    String             test_data = "new message";
    AQDequeueOption    deq_option;
    byte[]             b_array;
    Connection         db_conn;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    /* aqjava スキーマのキュー・テーブル aq_table4 へのハンドルを取得します。*/
    q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
    System.out.println("Successful getQueueTable");

    /* aquser スキーマのキュー aq_queue4 へのハンドルを取得します。*/
    queue = aq_sess.getQueue ("aqjava", "aq_queue4");

```

```
System.out.println("Successful getQueue");

/* RAW ペイロードを含むメッセージを作成します。*/
message = queue.createMessage();

/* AQRawPayload オブジェクトへのハンドルを取得し、RAW データとともに移入します。*/
b_array = test_data.getBytes();

raw_payload = message.getRawPayload();

raw_payload.setStream(b_array, b_array.length);

/* AQEnqueueOption オブジェクトをデフォルトのオプション付きで作成します。*/
enq_option = new AQEnqueueOption();

/* メッセージをエンキューします。*/
queue.enqueue(enq_option, message);
System.out.println("Successful enqueue");

db_conn.commit();

/* AQDequeueOption オブジェクトをデフォルトのオプション付きで作成します。*/
deq_option = new AQDequeueOption();

/* メッセージをデキューします。*/
message = queue.dequeue(deq_option);
System.out.println("Successful dequeue");

/* メッセージから RAW データを取り出します。*/
raw_payload = message.getRawPayload();

b_array = raw_payload.getBytes();

db_conn.commit();
}
```

## Java を使用したブラウズ・モードでのメッセージのデキュー

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable      q_table;
    AQQueueTable      q_table;
    AQQueue            queue;
    AQMessage          message;
```

```
AQRawPayload          raw_payload;
AQEnqueueOption        enq_option;
String                test_data = "new message";
AQDequeueOption        deq_option;
byte[]                b_array;
Connection            db_conn;

db_conn = ((AQOracleSession)aq_sess).getDBConnection();

/* aqjava スキーマのキュー・テーブル aq_table4 へのハンドルを取得します。*/
q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
System.out.println("Successful getQueueTable");

/* aquser スキーマのキュー aq_queue4 へのハンドルを取得します。*/
queue = aq_sess.getQueue ("aqjava", "aq_queue4");
System.out.println("Successful getQueue");

/* RAW ペイロードを含むメッセージを作成します。*/
message = queue.createMessage();

/* AQRawPayload オブジェクトへのハンドルを取得し、RAW データとともに移入します。*/
b_array = test_data.getBytes();

raw_payload = message.getRawPayload();

raw_payload.setStream(b_array, b_array.length);

/* AQEnqueueOption オブジェクトをデフォルトのオプション付きで作成します。*/
enq_option = new AQEnqueueOption();

/* メッセージをエンキューします。*/
queue.enqueue(enq_option, message);
System.out.println("Successful enqueue");

db_conn.commit();

/* AQDequeueOption オブジェクトをデフォルトのオプション付きで作成します。*/
deq_option = new AQDequeueOption();

/* デキュー・モードを BROWSE に設定します。*/
deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_BROWSE);

/* 待機時間を 10 秒に設定します。*/
deq_option.setWaitTime(10);
```

```
/* メッセージをデキューします。*/
message = queue.dequeue(deq_option);

/* メッセージから RAW データを取り出します。*/
raw_payload = message.getRawPayload();
b_array = raw_payload.getBytes();

String ret_value = new String(b_array);
System.out.println("Dequeued message: " + ret_value);

db_conn.commit();
}
```

## PL/SQL を使用した優先順位によるメッセージのエンキューおよびデキュー

2つのメッセージが同じ優先順位でエンキューされると、先にエンキューされた方が先にデキューされます。ただし、2つのメッセージの優先順位が異なる場合は、値の小さい（優先順位の低い）メッセージが先にデキューされます。

```
/* 2つのメッセージを優先順位 30 と 5 でエンキューします。*/
DECLARE
    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties  dbms_aq.message_properties_t;
    message_handle      RAW(16);
    message             aq.message_typ;

BEGIN
    message := message_typ('PRIORITY MESSAGE',
        'enqueued at priority 30.');
```

message\_properties.priority := 30;

```
    DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',
        enqueue_options    => enqueue_options,
        message_properties => message_properties,
        payload            => message,
        msgid              => message_handle);

    message := message_typ('PRIORITY MESSAGE',
        'Enqueued at priority 5.');
```

message\_properties.priority := 5;



```

DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',
                 enqueue_options => enqueue_options,
                 message_properties => message_properties,
                 payload => message,
                 msgid => message_handle);

END;

/* 優先順位キューからデキューします。*/
DECLARE
    dequeue_options    DBMS_AQ.dequeue_options_t;
    message_properties  DBMS_AQ.message_properties_t;
    message_handle      RAW(16);
    message             aq.message_typ;

BEGIN
    DBMS_AQ.DEQUEUE(queue_name => 'priority_msg_queue',
                    dequeue_options => dequeue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => message_handle);

    DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                          ' ... ' || message.text );

    COMMIT;

    DBMS_AQ.DEQUEUE(queue_name => 'priority_msg_queue',
                    dequeue_options => dequeue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => message_handle);

    DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                          ' ... ' || message.text );
    COMMIT;
END;

/* 戻るとすぐ、優先順位が5の2番目のメッセージが、優先順位30のメッセージの前に取り出されま
す。これは、優先順位がエンキュー時刻より優先されるためです。*/

```

## Java を使用した優先順位によるメッセージのエンキュー

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable      q_table;
    AQQueue           queue;
    AQMessage         message;
    AQMessageProperty m_property;
    AQRawPayload      raw_payload;
    AQEnqueueOption   enq_option;
    String            test_data;
    byte[]            b_array;
    Connection        db_conn;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    /* aqjava スキーマのキュー・テーブル aq_table4 へのハンドルを取得します。*/
    qtable = aq_sess.getQueueTable ("aqjava", "aq_table4");
    System.out.println("Successful getQueueTable");

    /* aqjava スキーマのキュー aq_queue4 へのハンドルを取得します。*/
    queue = aq_sess.getQueue ("aqjava", "aq_queue4");
    System.out.println("Successful getQueue");

    /* 優先順位が異なる 5 つのメッセージをエンキューします。*/
    for (int i = 0; i < 5; i++ )
    {
        /* RAW ペイロードを含むメッセージを作成します。*/
        message = queue.createMessage();

        test_data = "Small_message_" + (i+1);          /* テスト・データ */

        /* AQRawPayload オブジェクトへのハンドルを取得し、RAW データとともに移入します。*/
        b_array = test_data.getBytes();

        raw_payload = message.getRawPayload();

        raw_payload.setStream(b_array, b_array.length);

        /* メッセージの優先順位を設定します。*/
        m_property = message.getMessageProperty();

        if ( i < 2)
            m_property.setPriority(2);
    }
}
```

```

else
    m_property.setPriority(3);

    /* AQEnqueueOption オブジェクトをデフォルトのオプション付きで作成します。*/
    enq_option = new AQEnqueueOption();

    /* メッセージをエンキューします。*/
    queue.enqueue(enq_option, message);
    System.out.println("Successful enqueue");
}

db_conn.commit();
}

```

## PL/SQL を使用したプレビュー後のメッセージのデキュー

アプリケーションでは、ブラウズ・モードまたはロック・モードで、メッセージを削除せずにプレビューできます。その後で、対象メッセージをキューから削除できます。

/\* 6つのメッセージGREEN、GREEN、YELLOW、VIOLET、BLUE、RED をmsg\_queueにエンキューします。\*/

```

DECLARE
    enqueue_options    DBMS_AQ.enqueue_options_t;
    message_properties DBMS_AQ.message_properties_t;
    message_handle      RAW(16);
    message             aq.message_typ;

BEGIN
    message := message_typ('GREEN',
        'GREEN enqueued to msg_queue first.');
```

```

    DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
        enqueue_options    => enqueue_options,
        message_properties  => message_properties,
        payload             => message,
        msgid               => message_handle);

    message := message_typ('GREEN',
        'GREEN also enqueued to msg_queue second.');
```

```

    DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
        enqueue_options    => enqueue_options,
        message_properties  => message_properties,

```

```
        payload          => message,
        msgid            => message_handle);

message := message_typ('YELLOW',
'YELLOW enqueued to msg_queue third.');
```

```
DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload          => message,
    msgid            => message_handle);

DBMS_OUTPUT.PUT_LINE ('Message handle: ' || message_handle);

message := message_typ('VIOLET',
'VIOLET enqueued to msg_queue fourth.');
```

```
DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload          => message,
    msgid            => message_handle);

message := message_typ('BLUE',
'BLUE enqueued to msg_queue fifth.');
```

```
DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload          => message,
    msgid            => message_handle);

message := message_typ('RED',
'RED enqueued to msg_queue sixth.');
```

```
DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload          => message,
    msgid            => message_handle);

COMMIT;
END;
```

```
/* RED が見つかるまで BROWSE モードでデキューし、キューから RED を削除します。*/
DECLARE
    dequeue_options    DBMS_AQ.dequeue_options_t;
```

```

message_properties DBMS_AQ.message_properties_t;
message_handle     RAW(16);
message            aq.message_typ;

BEGIN
    dequeue_options.dequeue_mode := DBMS_AQ.BROWSE;

    LOOP
        DBMS_AQ.DEQUEUE(queue_name      => 'msg_queue',
                        dequeue_options  => dequeue_options,
                        message_properties => message_properties,
                        payload           => message,
                        msgid             => message_handle);

        DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                               ' ... ' || message.text );

        EXIT WHEN message.subject = 'RED';

    END LOOP;

    dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
    dequeue_options.msgid        := message_handle;

    DBMS_AQ.DEQUEUE(queue_name => 'msg_queue',
                    dequeue_options => dequeue_options,
                    message_properties => message_properties,
                    payload           => message,
                    msgid             => message_handle);

    DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                           ' ... ' || message.text );

    COMMIT;
END;

/* BLUEが見つかるまで LOCKED モードでデキューし、キューから BLUE を削除します。*/
DECLARE
    dequeue_options    dbms_aq.dequeue_options_t;
    message_properties dbms_aq.message_properties_t;
    message_handle     RAW(16);

```

```
message          aq.message_typ;

BEGIN
dequeue_options.dequeue_mode := dbms_aq.LOCKED;

    LOOP

dbms_aq.dequeue(queue_name => 'msg_queue',
                dequeue_options => dequeue_options,
                message_properties => message_properties,
                payload          => message,
                msgid            => message_handle);

dbms_output.put_line ('Message: ' || message.subject ||
                      ' ... ' || message.text );

EXIT WHEN message.subject = 'BLUE';
    END LOOP;

dequeue_options.dequeue_mode := dbms_aq.REMOVE;
dequeue_options.msgid        := message_handle;

dbms_aq.dequeue(queue_name => 'msg_queue',
                dequeue_options => dequeue_options,
                message_properties => message_properties,
                payload          => message,
                msgid => message_handle);

DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                      ' ... ' || message.text );

    COMMIT;

END;
```

## PL/SQL を使用した遅延および期限切れによるメッセージのエンキューおよびデキュー

---

---

**注意：** 期限切れは最も早いデキュー時刻から計算されています。たとえば、アプリケーション側でメッセージを今から 1 週間後以降 3 週間以内にデキューする場合には、期限切れを 2 週間に設定する必要があります。このシナリオを、次のサンプル・コードで説明します。

---

---

```

/* 遅延のために、メッセージをエンキューします。*/
DECLARE
enqueue_options      dbms_aq.enqueue_options_t;
message_properties   dbms_aq.message_properties_t;
message_handle       RAW(16);
message              aq.Message_typ;

BEGIN
message := Message_typ('DELAYED',
'This message is delayed one week. ');
message_properties.delay := 7*24*60*60;
message_properties.expiration := 2*7*24*60*60;

dbms_aq.enqueue(queue_name => 'msg_queue',
enqueue_options      => enqueue_options,
message_properties   => message_properties,
payload             => message,
msgid               => message_handle);

COMMIT;

END;

```

## Pro\*C/C++ を使用した相関識別子およびメッセージ ID によるメッセージのエンキューおよびデキュー

---

**注意：** 次のような設定を実行しないと機能しない例もあります。

```

$ cat >> message.typ
case=lower
type aq.message_typ
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \ code=c
hfile=demo.h
$
$ proc intyp=message_o.typ iname=<program name> \
config=<config file> SQLCHECK=SEMANTICS userid=aq/aq

```

---

```

#include <stdio.h>
#include <string.h>
#include <sqlca.h>
#include <sql2oci.h>
/* オブジェクト型 'aq.Message_typ' を処理することによって生成されるヘッダー・ファイル */
#include "pceg.h"

```

```

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
OCIEnv          *oeh; /* OCI 環境ハンドル */
OCIError        *err; /* OCI エラー・ハンドル */
Message_typ     *message = (Message_typ*)0; /* キュー・ペイロード */
OCIRaw         *msgid = (OCIRaw*)0; /* メッセージ ID */
ub1             msgmem[16]=""; /* msgid 用のメモリー */
char            user[60]="aq/AQ"; /* ユーザー・ログイン・パスワード */
char            subject[30]; /* コンポーネント */
char            txt[80]; /* Message_typ */
char            correlation1[30]; /* メッセージの相関 */
char            correlation2[30];
int             status; /* OCI コールによって戻されるコード */

/* 相関およびmsgidによるデキュー */

/* データベースに接続します。*/
EXEC SQL CONNECT :user;
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* ホスト変数用にオブジェクト・キャッシュに領域を割り当てます。*/
EXEC SQL ALLOCATE :message;

/* OCI 環境ハンドルを取得します。*/
if (SQLEnvGet(SQL_SINGLE_RCTX, &oeh) != OCI_SUCCESS)
{
printf(" error in SQLEnvGet \n");
exit(1);
}
/* OCI エラー・ハンドルを取得します。*/
if (status = OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
(ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0))

```



```

{
printf(" error in OCIHandleAlloc %d \n", status);
exit(1);
}

/* msgid にメモリーを割り当てます。
メモリーは、明示的に OCIRaw* に割り当てる必要があります。*/
if (status=OCIRawAssignBytes(oeh, err, msgmem, 16, &msgid))
{
printf(" error in  OCIRawAssignBytes  %d \n", status);
exit(1);
}

/* 1 回目のエンキュー */

strcpy(correlation1, "1st message");
strcpy(subject, "NORMAL ENQUEUE1");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* メッセージのコンポーネントを初期化します。*/
EXEC SQL OBJECT SET subject, text OF :message TO :subject, :txt;

/* 埋込み PL/SQL が AQ キュー・プロシージャをコールします。*/
EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
enqueue_options     dbms_aq.enqueue_options_t;
BEGIN
/* ホスト変数 'correlation1' をメッセージ相関にバインドします。*/
message_properties.correlation := :correlation1;

/* ホスト変数 'message' をペイロードにバインドし、ホスト変数 'msgid' にメッセージ ID を戻しま
す。*/
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message,
msgid => :msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;

printf("Enqueued Message \n");

```

```
printf("Subject  :%s\n",subject);
printf("Text      :%s\n",txt);

/* 2 回目のエンキュー */

strcpy(correlation2, "2nd message");
strcpy(subject, "NORMAL ENQUEUE2");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* メッセージのコンポーネントを初期化します。*/
EXEC SQL OBJECT SET subject, text OF :message TO :subject,:txt;

/* 埋込み PL/SQL が AQ エンキュー・プロシージャをコールします。*/
EXEC SQL EXECUTE
DECLARE
message_properties    dbms_aq.message_properties_t;
enqueue_options       dbms_aq.enqueue_options_t;
msgid                 RAW(16);
BEGIN
/* ホスト変数 'correlation2' をメッセージ相関にバインドします。*/
message_properties.correlation := :correlation2;

/* ホスト変数 'message' をペイロードにバインドします。*/
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;
printf("Enqueued Message \n");
printf("Subject  :%s\n",subject);
printf("Text      :%s\n",txt);

/* 相関による 1 回目のデキュー */

EXEC SQL EXECUTE
DECLARE
message_properties    dbms_aq.message_properties_t;
dequeue_options       dbms_aq.dequeue_options_t;
msgid                 RAW(16);
BEGIN
/* ホスト変数 'correlation2' で相関によるデキューを行います。*/
```

```
dequeue_options.correlation := :correlation2;

/* ホスト変数 'message' にペイロードを戻します。*/
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;

/* メッセージのコンポーネントの値を抽出します。*/
EXEC SQL OBJECT GET subject, text FROM :message INTO :subject,:txt;

printf("Dequeued Message \n");
printf("Subject   :%s\n",subject);
printf("Text      :%s\n",txt);

/* msgid による 2 回目のデキュー */

EXEC SQL EXECUTE
DECLARE
message_properties dbms_aq.message_properties_t;
dequeue_options   dbms_aq.dequeue_options_t;
msgid             RAW(16);
BEGIN
/* ホスト変数 'msgid' で msgid によるデキューを行います。*/
dequeue_options.msgid := :msgid;

/* ホスト変数 'message' にペイロードを戻します。*/
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* 処理をコミットします。*/
EXEC SQL COMMIT;
}
```

## OCI を使用した相関識別子およびメッセージ ID によるメッセージのエンキューおよびデキュー

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
    OCIStrng    *subject;
    OCIStrng    *data;
};
typedef struct message message;

struct null_message
{
    OCIInd      null_adt;
    OCIInd      null_subject;
    OCIInd      null_data;
};
typedef struct null_message null_message;

int main()
{
    OCIEnv      *envhp;
    OCIServer    *srvhp;
    OCIErrr      *errhp;
    OCISvcCtx    *svchp;
    dvoid        *tmp;
    OCIType      *mesg_tdo = (OCIType *) 0;
    message      msg;
    null_message nmsg;
    message      *mesg      = &msg;
    null_message *nmesg     = &nmsg;
    message      *deqmesg   = (message *)0;
    null_message *ndeqmesg  = (null_message *)0;

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
                  (dvoid * (*)()) 0, (void (*)()) 0 );

    OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                   52, (dvoid **) &tmp);

    OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp );
```

```

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* message_typ の TDO を取得します。*/
OCITypeByName(envhp, errhp, svchp, (CONST text *) "AQ", strlen("AQ"),
              (CONST text *) "MESSAGE_TYP", strlen("MESSAGE_TYP"),
              (text *) 0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* メッセージ・ペイロードを準備します。*/
mesg->subject = (OCIStrng *) 0;
mesg->data = (OCIStrng *) 0;
OCIStrngAssignText(envhp, errhp,
                  (CONST text *) "NORMAL MESSAGE", strlen("NORMAL MESSAGE"),
                  &mesg->subject);
OCIStrngAssignText(envhp, errhp,
                  (CONST text *) "OCI ENQUEUE", strlen("OCI ENQUEUE"),
                  &mesg->data);
nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* msg_queue にエンキューします。*/
OCIAQEnq(svchp, errhp, (CONST text *) "msg_queue", 0, 0,
         mesg_tdo, (dvoid **) &mesg, (dvoid **) &nmesg, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* msg_queue からデキューします。*/
OCIAQDeq(svchp, errhp, (CONST text *) "msg_queue", 0, 0,
         mesg_tdo, (dvoid **) &deqmesg, (dvoid **) &ndeqmesg, 0, 0);
printf("Subject: %s\n", OCIStrngPtr(envhp, deqmesg->subject));
printf("Text: %s\n", OCIStrngPtr(envhp, deqmesg->data));
OCITransCommit(svchp, errhp, (ub4) 0);
}

```

## PL/SQL を使用した複数コンシューマ・キューでのメッセージのエンキューおよびデキュー

```

/* サブスクライバ・リストを作成します。*/
DECLARE
    subscriber aq$_agent;

    /* サブスクライバ RED および GREEN をサブスクライバ・リストに追加します。*/
BEGIN
    subscriber := aq$_agent('RED', NULL, NULL);
    DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
        subscriber => subscriber);

    subscriber := aq$_agent('GREEN', NULL, NULL);
    DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
        subscriber => subscriber);
END;

DECLARE
    enqueue_options    DBMS_AQ.enqueue_options_t;
    message_properties DBMS_AQ.message_properties_t;
    recipients         DBMS_AQ.aq$_recipient_list_t;
    message_handle     RAW(16);
    message            aq.message_typ;

    /* サブスクライバ (RED および GREEN) に対する MESSAGE 1 をキューにエンキューします。*/
BEGIN
    message := message_typ('MESSAGE 1',
        'This message is queued for queue subscribers.');


DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload          => message,
    msgid            => message_handle);



/* 指定した受信者 (RED および BLUE) に対する MESSAGE 2 をエンキューします。*/
    message := message_typ('MESSAGE 2',
        'This message is queued for two recipients.');



recipients(1) := aq$_agent('RED', NULL, NULL);
    recipients(2) := aq$_agent('BLUE', NULL, NULL);
    message_properties.recipient_list := recipients;



DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
    enqueue_options => enqueue_options,


```

```

        message_properties => message_properties,
        payload            => message,
        msgid              => message_handle);

    COMMIT;
END;
```

RED はキューのサブスクライバであると同時に、MESSAGE 2 の指定された受信者であることに注意してください。それとは対照的に GREEN は、受信者が指定されていないキュー内のメッセージ（この場合 MESSAGE）に対するサブスクライバでしかありません。BLUE は、キューのサブスクライバではありませんが、MESSAGE 2 の宛先に指定されています。

```

/* msg_queue_multiple からメッセージをデキューします。*/
DECLARE
    dequeue_options    DBMS_AQ.dequeue_options_t;
    message_properties  DBMS_AQ.message_properties_t;
    message_handle      RAW(16);
    message             aq.message_typ;
    no_messages         exception;
    pragma exception_init (no_messages, -25228);

BEGIN

    dequeue_options.wait := DBMS_AQ.NO_WAIT;
    BEGIN
    /* コンシューマ BLUE は MESSAGE 2 を受け取ります。*/
    dequeue_options.consumer_name := 'BLUE';
    dequeue_options.navigation := FIRST_MESSAGE;

    LOOP

        DBMS_AQ.DEQUEUE(queue_name => 'msg_queue_multiple',
            dequeue_options => dequeue_options,
            message_properties => message_properties,
            payload => message,
            msgid => message_handle);

        DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
            ' ... ' || message.text );
        dequeue_options.navigation := NEXT_MESSAGE;

    END LOOP;
    EXCEPTION
    WHEN no_messages THEN
```

```
DBMS_OUTPUT.PUT_LINE ('No more messages for BLUE');
COMMIT;
END;

BEGIN
/* コンシューマ RED はMESSAGE 1 およびMESSAGE 2 を取得します。*/
  dequeue_options.consumer_name := 'RED';
  dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE
  LOOP
    DBMS_AQ.DEQUEUE(queue_name => 'msg_queue_multiple',
                     dequeue_options => dequeue_options,
                     message_properties => message_properties,
                     payload => message,
                     msgid => message_handle);

    DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                          ' ... ' || message.text );
    dequeue_options.navigation := NEXT_MESSAGE;
  END LOOP;
EXCEPTION
  WHEN no_messages THEN
    DBMS_OUTPUT.PUT_LINE ('No more messages for RED');
  COMMIT;
END;

BEGIN
/* コンシューマ GREEN は、MESSAGE 1 を受け取ります。*/
  dequeue_options.consumer_name := 'GREEN';
  dequeue_options.navigation := FIRST_MESSAGE;
  LOOP
    DBMS_AQ.DEQUEUE(queue_name => 'msg_queue_multiple',
                     dequeue_options => dequeue_options,
                     message_properties => message_properties,
                     payload => message,
                     msgid => message_handle);

    DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                          ' ... ' || message.text );
    dequeue_options.navigation := NEXT_MESSAGE;
  END LOOP;
EXCEPTION
  WHEN no_messages THEN
    DBMS_OUTPUT.PUT_LINE ('No more messages for GREEN');
  COMMIT;
END;
```



## OCI を使用した複数コンシューマ・キューでのメッセージのエンキュー およびデキュー

**注意：** 次のような SQL 文を実行しないと機能しない例もあります。

```
CONNECT aqadm/aqadm
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'aq.qtable_multi',
    multiple_consumers => true,
    queue_payload_type => 'aq.message_typ');
EXECUTE DBMS_AQADM.START_QUEUE('aq.msg_queue_multiple');
CONNECT aq/aq
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
    OCIStrng    *subject;
    OCIStrng    *data;
};
typedef struct message message;

struct null_message
{
    OCIInd      null_adt;
    OCIInd      null_subject;
    OCIInd      null_data;
};
typedef struct null_message null_message;

int main()
{
    OCIEnv          *envhp;
    OCIServer        *srvhp;
    OCIErr          *errhp;
    OCISvcCtx        *svchp;
    dvoid            *tmp;
    OCIType          *mesg_tdo = (OCIType *) 0;
    message          msg;
    null_message      rmsg;
    message          *mesg = &msg;
```

```

null_message      *nmesg = &nmesg;
message           *deqmesg = (message *)0;
null_message      *ndeqmesg = (null_message *)0;
OCIAQMsgProperties *msgprop = (OCIAQMsgProperties *)0;
OCIAQAgent        *agents[2];
OCIAQDeqOptions   *deqopt = (OCIAQDeqOptions *)0;
ub4               wait = OCI_DEQ_NO_WAIT;
ub4               navigation = OCI_DEQ_FIRST_MSG;

OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
              (dvoid * (*)()) 0, (void (*)()) 0 );

OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* message_typ の TDO を取得します。*/
OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
              (CONST text *)"MESSAGE_TYP", strlen("MESSAGE_TYP"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* メッセージ・ペイロードを準備します。*/
mesg->subject = (OCIStrng *)0;
mesg->data = (OCIStrng *)0;
OCIStrngAssignText(envhp, errhp,
                  (CONST text *)"MESSAGE 1", strlen("MESSAGE 1"),
                  &mesg->subject);
OCIStrngAssignText(envhp, errhp,
                  (CONST text *)"mesg for queue subscribers",

```

```

        strlen("msg for queue subscribers"), &mesg->data);
nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* サブスクライバ (RED および GREEN) に対する MESSAGE 1 をキューにエンキューします。*/
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, 0,
        mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

/* 指定した受信者 (RED および BLUE) に対する MESSAGE 2 をエンキューします。*/
/* メッセージ・ペイロードを準備します。*/
OCIStrAssignText(envhp, errhp,
        (CONST text *)"MESSAGE 2", strlen("MESSAGE 2"),
        &mesg->subject);
OCIStrAssignText(envhp, errhp,
        (CONST text *)"msg for two recipients",
        strlen("msg for two recipients"), &mesg->data);

/* AQ メッセージ・プロパティおよびエージェント記述子を割り当てます。*/
OCIDDescriptorAlloc(envhp, (dvoid **)&msgprop,
        OCI_DTYPE_AQMSG_PROPERTIES, 0, (dvoid **)&0);
OCIDDescriptorAlloc(envhp, (dvoid **)&agents[0],
        OCI_DTYPE_AQAGENT, 0, (dvoid **)&0);
OCIDDescriptorAlloc(envhp, (dvoid **)&agents[1],
        OCI_DTYPE_AQAGENT, 0, (dvoid **)&0);

/* 受信者リスト RED および BLUE を準備します。*/
OCIAttrSet(agents[0], OCI_DTYPE_AQAGENT, "RED", strlen("RED"),
        OCI_ATTR_AGENT_NAME, errhp);
OCIAttrSet(agents[1], OCI_DTYPE_AQAGENT, "BLUE", strlen("BLUE"),
        OCI_ATTR_AGENT_NAME, errhp);
OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES, (dvoid *)agents, 2,
        OCI_ATTR_RECIPIENT_LIST, errhp);

OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, msgprop,
        mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

OCITransCommit(svchp, errhp, (ub4) 0);

/* 異なるコンシューマ名でメッセージをデキューします。*/
/* デキュー・オプション記述子を割り当てて、デキュー・オプションを設定します。*/
OCIDDescriptorAlloc(envhp, (dvoid **)&deqopt, OCI_DTYPE_AQDEQ_OPTIONS, 0,
        (dvoid **)&0);

/* デキューがすぐに戻るように、wait パラメータを NO_WAIT に設定します。*/
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&wait, 0,
        OCI_ATTR_WAIT, errhp);

/* デキューの位置がリセットされるように、ナビゲーションを FIRST_MESSAGE に設定します。*/

```

```

/* 新しい consumer_name がデキュー・オプションに設定された後 */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&navigation, 0,
            OCI_ATTR_NAVIGATION, errhp);

/* コンシューマ BLUE として msg_queue_multiple からデキューします。*/
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)"BLUE", strlen("BLUE"),
            OCI_ATTR_CONSUMER_NAME, errhp);

while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
                msg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
        == OCI_SUCCESS)
{
    printf("Subject: %s\n", OCIStrPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStrPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);

/* コンシューマ RED として msg_queue_multiple からデキューします。*/
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)"RED", strlen("RED"),
            OCI_ATTR_CONSUMER_NAME, errhp);
while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
                msg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
        == OCI_SUCCESS)
{
    printf("Subject: %s\n", OCIStrPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStrPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);

/* コンシューマ GREEN として msg_queue_multiple からデキューします。*/
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)"GREEN", strlen("GREEN"),
            OCI_ATTR_CONSUMER_NAME, errhp);
while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
                msg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
        == OCI_SUCCESS)
{
    printf("Subject: %s\n", OCIStrPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStrPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);
}

```

## PL/SQL を使用したメッセージのグループ化によるメッセージのエンキューおよびデキュー

```

CONNECT aq/aq

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
    queue_table      => 'aq.msggroup',
    queue_payload_type => 'aq.message_typ',
    message_grouping => DBMS_AQADM.TRANSACTIONAL);

EXECUTE DBMS_AQADM.CREATE_QUEUE (
    queue_name      => 'msggroup_queue',
    queue_table     => 'aq.msggroup');

EXECUTE DBMS_AQADM.START_QUEUE (
    queue_name => 'msggroup_queue');

/* 各トランザクションで3つのメッセージをエンキューします。*/
DECLARE
    enqueue_options    DBMS_AQ.enqueue_options_t;
    message_properties DBMS_AQ.message_properties_t;
    message_handle     RAW(16);
    message            aq.message_typ;

BEGIN

    /* 反復ごとにコミットし、3回ループします。*/
    FOR txnno in 1..3 LOOP

        /* 反復ごとにエンキューし、3回ループします。*/
        FOR mesgno in 1..3 LOOP
            message := message_typ('GROUP#' || txnno,
                                   'Message#' || mesgno || ' in group' || txnno);

            DBMS_AQ.ENQUEUE(queue_name      => 'msggroup_queue',
                           enqueue_options => enqueue_options,
                           message_properties => message_properties,
                           payload          => message,
                           msgid           => message_handle);
        END LOOP;
        /* トランザクションをコミットします。*/
        COMMIT;
    END LOOP;
END;

/* グループとしてメッセージをデキューします。*/
DECLARE

```

```

    dequeue_options      DBMS_AQ.dequeue_options_t;
    message_properties    DBMS_AQ.message_properties_t;
    message_handle        RAW(16);
    message               aq.message_typ;

    no_messages    exception;
    end_of_group   exception;

    PRAGMA EXCEPTION_INIT (no_messages, -25228);
    PRAGMA EXCEPTION_INIT (end_of_group, -25235);

BEGIN
    dequeue_options.wait      := DBMS_AQ.NO_WAIT;
    dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;

    LOOP
        BEGIN
            DBMS_AQ.DEQUEUE(queue_name => 'msggroup_queue',
                           dequeue_options => dequeue_options,
                           message_properties => message_properties,
                           payload => message,
                           msgid => message_handle);

            DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                  ' ... ' || message.text );

            dequeue_options.navigation := DBMS_AQ.NEXT_MESSAGE;

        EXCEPTION
            WHEN end_of_group THEN
                DBMS_OUTPUT.PUT_LINE ('Finished processing a group of messages');
                COMMIT;
                dequeue_options.navigation := DBMS_AQ.NEXT_TRANSACTION;
            END;
        END LOOP;
    EXCEPTION
        WHEN no_messages THEN
            DBMS_OUTPUT.PUT_LINE ('No more messages');
    END;
```

## PL/SQL を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー

/\* 1 つ以上の LOB 属性を含むメッセージ・ペイロード・オブジェクト型を作成します。エンキュー時に、LOB 属性を EMPTY\_BLOB に設定します。エンキューが完了した後で、トランザクションをコミットする前に、キュー・テーブルまたはキュー・テーブル・ビューの user\_data 列から LOB 属性を選択します。この時点で、(OCI および PL/SQL の両方で使用可能な) LOB インタフェースを使用して、キューに LOB データを書き込むことができます。デキュー時には、メッセージ・ペイロードに LOB ロケータが含まれます。デキュー後 (ただし、トランザクションのコミット前)、この LOB ロケータを使用して、LOB データを読み込むことができます。\*/

/\* アカウントを設定します。\*/

```
connect system/manager
```

```
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT aq_administrator_role TO aqadm;
```

```
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON DBMS_AQ TO aq;
CREATE TYPE aq.message AS OBJECT(id          NUMBER,
                                   subject VARCHAR2(100),
                                   data       BLOB,
                                   trailer  NUMBER);
CREATE TABLESPACE aq_tbs DATAFILE 'aq.dbs' SIZE 2M REUSE;
```

/\* キュー・テーブル、キューを作成し、キューを開始します。\*/

```
CONNECT aqadm/aqadm
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table      => 'aq.qt1',
    queue_payload_type => 'aq.message');
EXECUTE DBMS_AQADM.CREATE_QUEUE(
    queue_name      => 'aq.queue1',
    queue_table     => 'aq.qt1');
EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'aq.queue1');
```

/\* 設定を終了します。\*/

/\* ラージ・データ型のエンキュー \*/

```
CONNECT aq/aq
CREATE OR REPLACE PROCEDURE blobenqueue(msgno IN NUMBER) AS
enq_userdata aq.message;
enq_msgid    RAW(16);
```

```
enqopt      DBMS_AQ.enqueue_options_t;
msgprop     DBMS_AQ.message_properties_t;
lob_loc     BLOB;
buffer      RAW(4096);

BEGIN

    buffer := HEXTORAW(RPAD('FF', 4096, 'FF'));
    enq_userdata := aq.message(msgno, 'Large Lob data', EMPTY_BLOB(), msgno);
    DBMS_AQ.ENQUEUE('aq.queue1', enqopt, msgprop, enq_userdata, enq_msgid);

    -- キュー・テーブルの LOB ロケータを選択します。
    SELECT t.user_data.data INTO lob_loc
        FROM qt1 t
        WHERE t.msgid = enq_msgid;

    DBMS_LOB.WRITE(lob_loc, 2000, 1, buffer );
    COMMIT;
END;

/* LOB データをデキューします。*/

CREATE OR REPLACE PROCEDURE blobdequeue AS
    dequeue_options  DBMS_AQ.dequeue_options_t;
    message_properties DBMS_AQ.message_properties_t;
    mid              RAW(16);
    pload            aq.message;
    lob_loc          BLOB;
    amount           BINARY_INTEGER;
    buffer           RAW(4096);

BEGIN
    DBMS_AQ.DEQUEUE('aq.queue1', dequeue_options, message_properties,
                    pload, mid);
    lob_loc := pload.data;

    -- LOB データ info パックファを読み込みます。
    amount := 2000;
    DBMS_LOB.READ(lob_loc, amount, 1, buffer);
    DBMS_OUTPUT.PUT_LINE('Amount of data read: '||amount);
    COMMIT;
END;

/* エンキューおよびデキューを行います。*/

SET SERVEROUTPUT ON
```



```
BEGIN
  FOR i IN 1..5 LOOP
    blobenqueue(i);
  END LOOP;
END;
```

```
BEGIN
  FOR i IN 1..5 LOOP
    blobdequeue();
  END LOOP;
END;
```

## Java を使用した LOB 属性を含むオブジェクト型メッセージのエンキューおよびデキュー

1. メッセージ型（CLOB および BLOB を含むオブジェクト型）を作成します。

```
connect aquser/aquser

create type LobMessage as object(id          NUMBER,
                                subject      varchar2(100),
                                data         blob,
                                cdata       clob,
                                trailer     number);
```

2. キュー・テーブルおよびキューを作成します。

```
connect aquser/aquser
execute dbms_aqadm.create_queue_table(
    queue_table => 'qt_adt',
    queue_payload_type => 'LOBMESSAGE',
    comment => 'single-consumer, default sort ordering, ADT Message',
    compatible => '8.1.0'
);

execute dbms_aqadm.create_queue(
    queue_name => 'q1_adt',
    queue_table => 'qt_adt'
);

execute dbms_aqadm.start_queue(queue_name => 'q1_adt');
```

3. JPublisher を実行し、LobMessage に対応する Java クラスを生成します。

```
Oracle object type

jpub -user=aquser/aquser -sql=LobMessage -case=mixed -methods=false
```

4. メッセージをエンキューおよびデキューします。

```
public static void runTest(AQSession aq_sess)
{
    Connection          db_conn    = null;
    AQEnqueueOption     eq_option  = null;
    AQDequeueOption     dq_option  = null;
    AQQueue             queue1     = null;
    AQMessage           adt_msg    = null;
    AQMessage           adt_msg2   = null;
    AQObjectPayload     sPayload   = null;
```

```

AQObjectPayload      sPayload2 = null;
LobMessage           sPayl      = null;
LobMessage           sPayl2     = null;
AQObjectPayload      rPayload   = null;
LobMessage           rPayl      = null;
byte[]               msggid;
AQMessage            rMessage   = null;
int                  i          = 0;
int                  j          = 0;
int                  id         = 0;
boolean              more       = false;
byte[]               b_array;
char[]               c_array;
String               mStr       = null;
BLOB                 b1         = null;
CLOB                 c1         = null;
BLOB                 b2         = null;
CLOB                 c2         = null;
BLOB                 b3         = null;
CLOB                 c3         = null;
int                  b_len      = 0;
int                  c_len      = 0;
OracleCallableStatement blob_stmt0= null;
OracleCallableStatement clob_stmt0= null;
OracleResultSet      rset0      = null;
OracleResultSet      rset1      = null;
OracleCallableStatement blob_stmt = null;
OracleResultSet      rset2      = null;
OracleCallableStatement clob_stmt = null;
OracleResultSet      rset3      = null;

try
{

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    queue1 = aq_sess.getQueue("aquser", "q1_adt");

    b_array = new byte[5000];
    c_array = new char[5000];
    for (i = 0; i < 5000; i++)
    {
        b_array[i] = 67;
        c_array[i] = 'c';
    }
    sPayl = new LobMessage();

```

```
System.out.println("Enqueue Long messages");

eq_option = new AQEnqueueOption();

/* LOB 属性を持つメッセージをエンキューします。*/
for ( i = 0; i < 10; i++)
{
    adt_msg = queue1.createMessage();

    sPayload = adt_msg.getObjectPayload();

    /* Empty BLOB ハンドルを取得します。*/
    blob_stmt0 = (OracleCallableStatement)db_conn.prepareCall(
        "select empty_blob() from dual");
    rset0 = (OracleResultSet) blob_stmt0.executeQuery ();
    try
    {
        if (rset0.next())
        {
            b1 = (oracle.sql.BLOB)rset0.getBlob(1);
        }
        if (b1 == null)
        {
            System.out.println("select empty_blob() from dual failed");
        }
    }
    catch (Exception ex)
    {
        System.out.println("Exception during select from dual " + ex);
        ex.printStackTrace();
    }

    /* Empty CLOB ハンドルを取得します。*/
    clob_stmt0 = (OracleCallableStatement)db_conn.prepareCall(
        "select empty_clob() from dual");
    rset1 = (OracleResultSet) clob_stmt0.executeQuery ();
    try
    {
        if (rset1.next())
        {
            c1 = (oracle.sql.CLOB)rset1.getClob(1);
        }
        if (c1 == null)
        {
            System.out.println("select empty_clob() from dual failed");
        }
    }
}
```

```
    }
}
catch (Exception ex)
{
    System.out.println("Exception2 during select from dual " + ex);
    ex.printStackTrace();
}
id = i+1;
mStr = "Message #" + id;
sPayl.setId(new BigDecimal(id));
sPayl.setTrailer(new BigDecimal(id));
sPayl.setSubject(mStr);
sPayl.setData(b1);
sPayl.setCdata(c1);

/* オブジェクト・ペイロード・データを設定します。*/
sPayload.setPayloadData(sPayl);

/* メッセージをエンキューします。*/
queue1.enqueue(eq_option, adt_msg);
System.out.println("Enqueued Message: " + id);
msgid = adt_msg.getMessageId();

/*
 * 注意：メッセージは、EMPTY BLOB および CLOB とともにはじめにエンキュー
 * されます。メッセージをエンキューした後、LOB ロケータを取得し、LOB を移入
 * します。
 */
blob_stmt = (OracleCallableStatement)db_conn.prepareCall(
    "SELECT user_data FROM qt_adt where msgid = ?");
blob_stmt.setBytes(1,msgid);
rset2 = (OracleResultSet) blob_stmt.executeQuery ();
try
{
    if (rset2.next())
    {
        /* メッセージの内容を取得します。*/
        sPayl2 = (LobMessage)rset2.getCustomDatum(1,
            ((CustomDatumFactory)LobMessage.getFactory()));

        /* BLOB ロケータを取得します。*/
        b2 = sPayl2.getData();

        /* BLOB を移入します。*/
        if (b2 == null)
        {
            System.out.println("Blob select null");
        }
    }
}
```

```

    }
    if ((i % 3) == 0)
    {
        b_len = b2.putBytes(1000,b_array);
    }
    else
    {
        b_len = b2.putBytes(1,b_array);
    }

    /* CLOB ロケータを取得します。*/
    c2 = sPay12.getCdata();

    /* CLOB を移入します。*/
    if (c2 == null)
    {
        System.out.println("Clob select null");
    }
    if ((i % 4) == 0)
    {
        c_len = c2.putChars(2500,c_array);
    }
    else
    {
        c_len = c2.putChars(1,c_array);
    }
}
}
catch (Exception ex)
{
    System.out.println("Blob or Clob exception: " + ex);
}

}

Thread.sleep(30000);

// dequeue messages
dq_option = new AQDequeueOption();
dq_option.setWaitTime(AQDequeueOption.WAIT_NONE);

for (i = 0 ; i < 10 ; i++)
{
    /* メッセージをデキューします。*/
    adt_msg2 = ((AQOracleQueue)queue1).dequeue(dq_option,

```

```
LobMessage.getFactory());

/* LOB データを含むペイロードを取得します。*/
rPayload = adt_msg2.getObjectPayload();
rPayl = (LobMessage) rPayload.getPayloadData();

System.out.println("\n Message: #" + (i+1));
System.out.println("    Id: " + rPayl.getId());
System.out.println("    Subject: " + rPayl.getSubject());

/* BLOB データを取得します。*/
b3 = rPayl.getData();
System.out.println("    " + b3.length() + " bytes of data");

/* CLOB データを取得します。*/
c3 = rPayl.getCdata();
System.out.println("    " + c3.length() + " chars of data");
System.out.println("    Trailer: " + rPayl.getTrailer());
db_conn.commit();
}

}

catch (java.sql.SQLException sql_ex)
{
    System.out.println("SQL Exception: " + sql_ex);
    sql_ex.printStackTrace();
}

catch (Exception ex)
{
    System.out.println("Exception-2: " + ex);
    ex.printStackTrace();
}

}
```

## 伝播

---

**注意：** キューやキュー・テーブルを作成したり、キューを開始または使用可能にしないと、機能しない例もあります。

---

### PL/SQL を使用したリモートのサブスクライバ/受信者用のメッセージの複数コンシューマ・キューへのエンキューおよび伝播のスケジュール

```
/* サブスクライバ・リストを作成します。*/
DECLARE
    subscriber aq$_agent;

    /* アドレスが異なるサブスクライバ RED および GREEN をサブスクライバ・リストに追加します。*/
BEGIN
    BEGIN
        /* 同じデータベースの another_msg_queue キューからメッセージをデキューするサブスクライバ RED を追加します。*/
        subscriber := aq$_agent('RED', 'another_msg_queue', NULL);
        DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
            subscriber => subscriber);

        /* 同じデータベースの msg_queue_multiple から他のキューへの伝播をスケジュールします。*/
        DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'msg_queue_multiple');

        /* データベース・リンク another_db.world で接続されている他のデータベースの msg_queue キューからメッセージをデキューするサブスクライバ GREEN を追加します。*/
        subscriber := aq$_agent('GREEN', 'msg_queue@another_db.world', NULL);
        DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
            subscriber => subscriber);

        /* msg_queue_multiple からデータベース "another_database" の他のキューへの伝播をスケジュールします。*/
    END;
    BEGIN
        DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'msg_queue_multiple',
            destination => 'another_db.world');
    END;
END;
```



```

DECLARE
    enqueue_options    DBMS_AQ.enqueue_options_t;
    message_properties  DBMS_AQ.message_properties_t;
    recipients          DBMS_AQ.aq$_recipient_list_t;
    message_handle      RAW(16);
    message             aq.message_typ;

/* サブスライバ (アドレスが another_msg_queue の RED および msg_queue@another_db.world の
GREEN) に対する MESSAGE 1 をエンキューします。*/
BEGIN
    message := message_typ('MESSAGE 1',
        'This message is queued for queue subscribers.');
```

DBMS\_AQ.ENQUEUE(queue\_name => 'msg\_queue\_multiple',  
 enqueue\_options => enqueue\_options,  
 message\_properties => message\_properties,  
 payload => message,  
 msgid => message\_handle);

/\* 指定した受信者 (BLUE およびアドレスが another\_msg\_queue の RED) に対する MESSAGE 2 を  
エンキューします。\*/

```

    message := message_typ('MESSAGE 2',
        'This message is queued for two recipients.');
```

recipients(1) := aq\$\_agent('RED', 'another\_msg\_queue', NULL);  
recipients(2) := aq\$\_agent('BLUE', NULL, NULL);  
message\_properties.recipient\_list := recipients;

DBMS\_AQ.ENQUEUE(queue\_name => 'msg\_queue\_multiple',  
 enqueue\_options => enqueue\_options,  
 message\_properties => message\_properties,  
 payload => message,  
 msgid => message\_handle);

```

    COMMIT;
END;
```

---

---

**注意：** アドレス `another_msg_queue` の RED は、キューのサブスクライバであると同時に、MESSAGE 2 が指定された受信者です。それとは対照的に、アドレス `msg_queue@another_db.world` の GREEN は、受信者が指定されていないキュー内のメッセージ（この場合 MESSAGE 1）に対するサブスクライバでしかありません。BLUE はキューのサブスクライバではありませんが、MESSAGE 2 の宛先に指定されています。

---

---

## PL/SQL を使用した同一データベース内の 1 つのキューから他のキューへの伝播管理

```
/* 同じキュー内の q1def から他のキューへの伝播をスケジュールします。*/
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'q1def');

/* 同じデータベース内のキュー q1def から他のキューへの伝播を使用不可にします。*/
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
    queue_name => 'q1def');

/* 同じデータベース内のキュー q1def から他のキューへのスケジュールを変更します。*/
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
    queue_name => 'q1def',
    duration    => '2000',
    next_time   => 'SYSDATE + 3600/86400',
    latency     => '32');

/* 同じデータベース内のキュー q1def から他のキューへの伝播を使用可能にします。*/
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
    queue_name => 'q1def');

/* 同じデータベース内のキュー q1def から他のキューへの伝播のスケジュールを解除します。*/
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
    queue_name => 'q1def');
```

## PL/SQL を使用した 1 つのキューから他のデータベース内の他のキューへの伝播管理

```
/* キュー q1def から、データベース・リンク another_db.world で接続されている他のデータベース内の他のキューへの伝播をスケジュールします。*/
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(
```

```
queue_name => 'q1def',
destination => 'another_db.world');

/* キュー q1def から、データベース・リンク another_db.world で接続されている他のデータベース
の他のキューへの伝播を使用不可にします。*/
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
    queue_name => 'q1def',
    destination => 'another_db.world');

/* キュー q1def から、データベース・リンク another_db.world で接続されている他のデータベース
の他のキューへのスケジュールを変更します。*/
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
    queue_name => 'q1def',
    destination => 'another_db.world',
    duration => '2000',
    next_time => 'SYSDATE + 3600/86400',
    latency => '32');

/* キュー q1def から、データベース・リンク another_db.world で接続されている他のデータベース
の他のキューへの伝播を使用可能にします。*/
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
    queue_name => 'q1def',
    destination => 'another_db.world');

/* キュー q1def から、データベース・リンク another_db.world で接続されている他のデータベース
の他のキューへの伝播のスケジュールを解除します。*/
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
    queue_name => 'q1def',
    destination => 'another_db.world');
```

## PL/SQL を使用した伝播スケジュールの解除

```
/* msg_queue_multiple から宛先 another_db.world への伝播のスケジュールを解除します。*/
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
    queue_name => 'msg_queue_multiple',
    destination => 'another_db.world');
```

---

---

### 参照：

- 9-77 ページの「[PL/SQL \(DBMS\\_AQADM\) : 伝播スケジュールの変更](#)」を参照してください。
  - 9-80 ページの「[PL/SQL \(DBMS\\_AQADM\) : 伝播の使用可能化](#)」を参照してください。
  - 9-83 ページの「[PL/SQL \(DBMS\\_AQADM\) : 伝播の使用不可](#)」を参照してください。
- 
- 

## AQ オブジェクトの削除

---

**注意：** キューやキュー・テーブルを作成したり、キューを開始、停止または使用可能にしないと、機能しない例もあります。

---

---

```
/* オブジェクト型に関連するすべてのオブジェクトをクリーン・アップします。*/
CONNECT aq/aq

EXECUTE DBMS_AQADM.STOP_QUEUE (
    queue_name => 'msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE (
    queue_name => 'msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
    queue_table => 'aq.objmsgs80_qtab');

/* RAW 型に関連するすべてのオブジェクトをクリーン・アップします。*/
EXECUTE DBMS_AQADM.STOP_QUEUE (
    queue_name      => 'raw_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE (
    queue_name      => 'raw_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
    queue_table => 'aq.RawMsgs_qtab');

/* 優先順位キューに関連するすべてのオブジェクトをクリーン・アップします。*/
EXECUTE DBMS_AQADM.STOP_QUEUE (
    queue_name      => 'priority_msg_queue');
```

```
EXECUTE DBMS_AQADM.DROP_QUEUE (
    queue_name      => 'priority_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
    queue_table     => 'aq.priority_msg');

/* 複数コンシューマ・キューに関連するすべてのオブジェクトをクリーン・アップします。*/
EXECUTE DBMS_AQADM.STOP_QUEUE (
    queue_name      => 'msg_queue_multiple');

EXECUTE DBMS_AQADM.DROP_QUEUE (
    queue_name      => 'msg_queue_multiple');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
    queue_table     => 'aq.MultiConsumerMsgs_qtab');

DROP TYPE aq.message_typ;
```

## ロールおよび権限の取消し

```
CONNECT sys/change_on_install
DROP USER aq;
```

## AQ による XA の使用

---

**注意：** 次のような SQL 文を実行しないと機能しない例もあります。

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
GRANT CONNECT, RESOURCE TO aqadm;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'aq.qtable',
    queue_payload_type => 'RAW');

EXECUTE DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'aq.aqqueue',
    queue_table => 'aq.qtable');

EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'aq.aqqueue');
```

---

```
/*
 * このプログラムは、XA インタフェースを使用して 100 のメッセージをエンキューした後、デキュー
 * します。
 * ログイン： aq/aq
 * 必要条件： aq に AQ_USER_ROLE が付与される必要があります。
 *             RAW 型のキュー "aqqueue" が aqs スキーマに作成される必要があります。
 *             (これらの処理は、 aqaq.sql を実行することによって行います。)
 * メッセージ・フォーマット： Msgno: [0-1000] HELLO, WORLD!
 * 作成者： schandra@us.oracle.com
 */

#ifdef OCI_ORACLE
#include <oci.h>
#endif

#include <xa.h>
```

```

/* XA オープン文字列 */
char xaoinfo[] = "oracle_xa+ACC=P/AQ/AQ+SESTM=30+Objects=T";

/* XA XIDs を生成するためのテンプレート */
XID xidtempl = { 0x1e0a0a1e, 12, 8, "GTRID001BQual001" };

/* Oracle XA ファンクション・テーブルへのポインタ */
extern struct xa_switch_t xaosw; /* Oracle XA スイッチ */
static struct xa_switch_t *xafunc = &xaosw;

/* ax_reg および ax_unreg のダミー・スタブ */
int ax_reg(rmid, xid, flags)
int rmid;
XID *xid;
long flags;
{
    xid->formatID = -1;
    return 0;
}

int ax_unreg(rmid, flags)
int rmid;
long flags;
{
    return 0;
}

/* XID を生成します。 */
void xidgen(xid, serialno)
XID *xid;
int serialno;
{
    char seq [11];

    sprintf(seq, "%d", serialno);
    memcpy((void *)xid, (void *)&xidtempl, sizeof(XID));
    strncpy((&xid->data[5]), seq, 3);
}

/* XA の操作が正常に行われたかどうかを確認します。 */
#define checkXAerr(action, funcname) \
    if ((action) != XA_OK) \
    { \
        printf("%s failed!\n", funcname); \
    }

```

```
        exit(-1);          \
    } else

/* OCI の操作が正常に行われたかどうかを確認します。*/
static void checkOCIerr(errhp, status)
OCIError *errhp;
sword      status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    if (status == OCI_SUCCESS) return;

    if (status == OCI_ERROR)
    {
        OCIErrorGet((dvoid *) errhp, 1, (text *)0, &errcode, errbuf,
                    (ub4)sizeof(errbuf), OCI_HTYPE_ERROR);
        printf("Error - %s\n", errbuf);
    }
    else
        printf("Error - %d\n", status);
    exit (-1);
}

void main(argc, argv)
int      argc;
char **argv;
{
    int      msgno = 0;          /* エンキューされるメッセージ */
    OCIEnv   *envhp;            /* OCI 環境ハンドル */
    OCIError  *errhp;            /* OCI エラー・ハンドル */
    OCISvcCtx *svchp;            /* OCI サービス・ハンドル */
    char      message[128];      /* メッセージ・バッファ */
    ub4      msglen;             /* メッセージの長さ */
    OCIRaw    *rawmesg = (OCIRaw *)0; /* OCI RAW フォーマットのメッセージ */
    OCIInd     ind = 0;          /* OCI NULL 標識 */
    dvoid      *indpctr = (dvoid *)&ind; /* NULL 標識ポインタ */
    OCIType     *mesg_tdo = (OCIType *) 0; /* RAW データ型の TDO */
    XID         xid;             /* XA のグローバル・トランザクション ID */
    ub4         i;               /* 配列索引 */

    checkXAerr(xafunc->xa_open_entry(xaoinfo, 1, TMNOFLAGS), "xaopen");
```



```

svchp = xaoSvcCtx((text *)0);          /* XA からサービス・ハンドルを取得します。*/
envhp = xaoEnv((text *)0);             /* XA から環境ハンドルを取得します。*/

if (!svchp || !envhp)
{
    printf("Unable to obtain OCI Handles from XA!\n");
    exit (-1);
}

OCIHandleAlloc((dvoid *)envhp, (dvoid **)&errhp,
               OCI_HTYPE_ERROR, 0, (dvoid **)0); /* エラー・ハンドルを割り当てます。*/

/* XA トランザクションごとに1つのメッセージで、1000 メッセージをエンキューします。*/
for (msgno = 0; msgno < 1000; msgno++)
{
    sprintf((const char *)message, "Msgno: %d, Hello, World!", msgno);
    msglen = (ub4)strlen((const char *)message);
    xidgen(&xid, msgno);                /* XA xid を生成します。*/

    checkXAerr(xafunc->xa_start_entry(&xid, 1, TMNOFLAGS), "xaostart");

    checkOCIerr(errhp, OCIRawAssignBytes(envhp, errhp, (ub1 *)message, msglen,
                                         &rawmesg));

    if (!mesg_tdo)                      /* RAW 型の型記述子 (TDO) を取得します。*/
        checkOCIerr(errhp, OCITypeByName(envhp, errhp, svchp,
                                         (CONST text *)"AQADM", strlen("AQADM"),
                                         (CONST text *)"RAW", strlen("RAW"),
                                         (text *)0, 0, OCI_DURATION_SESSION,
                                         OCI_TYPEGET_ALL, &mesg_tdo));

    checkOCIerr(errhp, OCIAQEnq(svchp, errhp, (CONST text *)"aqqueue",
                                0, 0, mesg_tdo, (dvoid **)&rawmesg, &indptr,
                                0, 0));

    checkXAerr(xafunc->xa_end_entry(&xid, 1, TMSUCCESS), "xaoend");
    checkXAerr(xafunc->xa_commit_entry(&xid, 1, TMONEPHASE), "xaocommit");
    printf("%s Enqueued\n", message);
}

/* 1回の XA トランザクション内で1000 のメッセージをデキューします。*/
xidgen(&xid, msgno);                  /* XA xid を生成します。*/
checkXAerr(xafunc->xa_start_entry(&xid, 1, TMNOFLAGS), "xaostart");
for (msgno = 0; msgno < 1000; msgno++)
{

```

```
        checkOCIerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"aqsgueue",
        0, 0, msg_tdo, (dvoid **)&rawmsg, &indptr,
        0, 0));
    if (ind)
        printf("Null Raw Message");
    else
        for (i = 0; i < OCIRawSize(envhp, rawmsg); i++)
            printf("%c", *(OCIRawPtr(envhp, rawmsg) + i));
        printf("\n");

    }
    checkXAerr(xafunc->xa_end_entry(&xid, 1, TMSUCCESS), "xaend");
    checkXAerr(xafunc->xa_commit_entry(&xid, 1, TMONEPHASE), "xaocommit");
}
```

## AQ およびメモリーの使用

### Creat\_types.sql: Scott のスキーマへのペイロード型およびキューの作成

**注意：** 次のような SQL 文を実行しないと機能しない例もあります。

```
/* Create_types.sql */
CONNECT system/manager
GRANT AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE TO scott;
CONNECT scott/tiger
CREATE TYPE MESSAGE AS OBJECT (id NUMBER, data VARCHAR2(80));
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table      => 'qt',
    queue_payload_type => 'message');
EXECUTE DBMS_AQADM.CREATE_QUEUE('msgqueue', 'qt');
EXECUTE DBMS_AQADM.START_QUEUE('msgqueue');
```

### OCI を使用したメッセージのエンキュー（各コール後のメモリー解放）

このプログラム enqgnoreuse.c は、前述の create\_types.sql を介して scott のスキーマに作成されたキュー msgqueue から、テキストの各行をデキューします。メッセージは、enqgnoreuse.c または enqgreuse.c を使用してエンキューされます（次の例を参照）。メッセージがない場合は、60 秒待機してからタイムアウトします。このプログラムでは、デキュー・サブルーチンはクライアント側のオブジェクトのメモリーを再使用しません。必要なメモリーをデキューの前に割り当て、デキューの完了後にそのメモリーを解放します。

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;

struct message
{
```

```

    OCINumber    id;
    OCISString    *data;
};
typedef struct message message;

struct null_message
{
    OCIIInd    null_adt;
    OCIIInd    null_id;
    OCIIInd    null_data;
};
typedef struct null_message null_message;

static void deqmesg(buf, buflen)
text    *buf;
ub4     *buflen;
{
    OCIType        *mesgtdo = (OCIType *)0;    /* SCOTT.MESSAGE の型記述子 */
    message        *mesg    = (dvoid *)0;      /* SCOTT.MESSAGE のインスタンス */
    null_message    *mesgind = (dvoid *)0;      /* NULL 標識 */
    OCIAQDeqOptions *deqopt  = (OCIAQDeqOptions *)0;
    ub4            wait      = 60;              /* 60 秒後にタイムアウト */
    ub4            navigation = OCI_DEQ_FIRST_MSG; /* 常に q の先頭を取得 */

    /* SCOTT.MESSAGE 型の型記述子オブジェクトを取得します。*/
    checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
        (CONST text *)"SCOTT", strlen("SCOTT"),
        (CONST text *)"MESSAGE", strlen("MESSAGE"),
        (text *)0, 0, OCI_DURATION_SESSION,
        OCI_TYPEGET_ALL, &mesgtdo));

    /* SCOTT.MESSAGE のインスタンスを割り当て、その NULL 標識を取得します。*/
    checkerr(errhp, OCIObjectNew(envhp, errhp, svchp, OCI_TYPECODE_OBJECT,
        mesgtdo, (dvoid *)0, OCI_DURATION_SESSION,
        TRUE, (dvoid **)&mesg));
    checkerr(errhp, OCIObjectGetInd(envhp, errhp, (dvoid *)mesg,
        (dvoid **)&mesgind));

    /* デキュー・オプションに記述子を割り当て、待機時間、ナビゲーションを設定します。*/
    checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
        OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));
    checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
        (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
    checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,

```

```

        (dvoid *)&navigation, 0,
        OCI_ATTR_NAVIGATION, errhp));

/* メッセージをデキューしてコミットします。*/
checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
        deqopt, 0, mesgtdo, (dvoid **)&mesg,
        (dvoid **)&mesgind, 0, 0));

checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

/* メッセージ・ペイロード・テキストをユーザー・バッファにコピーします。*/
if (mesgind->null_data)
    *buflen = 0;
else
    memcpy((dvoid *)buf, (dvoid *)OCIStrPtr(envhp, mesg->data),
        (size_t) (*buflen = OCIStrSize(envhp, mesg->data)));

/* デキュー・オプション記述子を解放します。*/
checkerr(errhp, OCIDescriptorFree((dvoid *)deqopt, OCI_DTYPE_AQDEQ_OPTIONS));

/* オブジェクトのメモリーを解放します。*/
checkerr(errhp, OCIObjectFree(envhp, errhp, (dvoid *)mesg,
        OCI_OBJECTFREE_FORCE));
}
/* deqmesg を終了します。*/

void main()
{
    OCIServer      *srvhp;
    OCISession     *usrhp;
    dvoid          *tmp;
    text           buf[80];
    ub4            buflen;

    /* ペイロード・テキスト*/

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
        (dvoid * (*)()) 0, (void (*)()) 0);

    OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
        52, (dvoid **) &tmp);

    OCIEnvInit(&envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp);

    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
        52, (dvoid **) &tmp);
    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
        52, (dvoid **) &tmp);

```

```
OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* ユーザー・コンテキスト・ハンドルを割り当てます。*/
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &usrhp, (ub4) OCI_HTYPE_SESSION,
               (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION,
           (dvoid *) "scott", (ub4) strlen("scott"), OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION,
           (dvoid *) "tiger", (ub4) strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                OCI_DEFAULT));

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
           (dvoid *) usrhp, (ub4) 0, OCI_ATTR_SESSION, errhp);

do {
    deqmesg(buf, &buflen);
    printf("%.5s\n", buflen, buf);
} while(1);
/* main を終了します。*/

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    if (status == OCI_SUCCESS) return;

    switch (status)
    {
        case OCI_ERROR:
```

```

        OCLErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                     errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
        printf("Error - %s\n", errbuf);
        break;
case OCI_INVALID_HANDLE:
        printf("Error - OCI_INVALID_HANDLE\n");
        break;
default:
        printf("Error - %d\n", status);
        break;
    }
    exit(-1);
}
/* checkerr を終了します。*/

```

## OCI を使用したメッセージのエンキュー（メモリーの再使用）

このプログラム enqreus.c は、create\_types.sql を実行して scott のスキーマに作成されたキュー msgqueue に、テキストの各行をエンキューします。ユーザーが入力するテキストの各行は、ユーザーが EOF を入力するまでキューの中に格納されます。このプログラムでは、エンキュー・サブルーチンは AQ メッセージ・プロパティ記述子のみでなくメッセージ・ペイロード用のメモリーも再使用します。

```

#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCLError *errhp, sword status);
static void enqmesg(ub4 msgno, text *buf);

struct message
{
    OCINumber    id;
    OCISString   *data;
};
typedef struct message message;

struct null_message
{
    OCInd        null_adt;
    OCInd        null_id;
    OCInd        null_data;
};
typedef struct null_message null_message;

```

```

/* エンキューのためのコールで再利用するグローバル・データ */
OCIEnv          *envhp;
OCIError         *errhp;
OCISvcCtx        *svchp;
message          msg;
null_message     rmsg;
OCIAQMsgProperties *msgprop;

static void enqmesg(msgno, buf)
ub4      msgno;
text     *buf;
{
    OCIType      *mesgtdo = (OCIType *)0; /* SCOTT.MESSAGE の型記述子 */
    message      *mesg = &msg;           /* SCOTT.MESSAGE のインスタンス */
    null_message *mesgind = &rmsg;        /* NULL 標識 */
    text         corrid[128];             /* 相関識別子 */

    /* SCOTT.MESSAGE 型の型記述子オブジェクトを取得します。*/
    checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
        (CONST text *)"SCOTT", strlen("SCOTT"),
        (CONST text *)"MESSAGE", strlen("MESSAGE"),
        (text *)0, 0, OCI_DURATION_SESSION,
        OCI_TYPEGET_ALL, &mesgtdo));

    /* SCOTT.MESSAGE の属性を設定します。*/
    checkerr(errhp, OCINumberFromInt(errhp, &msgno, sizeof(ub4), 0, &mesg->id));
    checkerr(errhp, OCIStrAssignText(envhp, errhp, buf, strlen(buf),
        &mesg->data));
    mesgind->null_adt = mesgind->null_id = mesgind->null_data = 0;

    /* メッセージ・プロパティ記述子に相関識別子を設定します。*/
    sprintf((char *)corrid, "Msg#: %d", msgno);
    checkerr(errhp, OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES,
        (dvoid *)&corrid, strlen(corrid),
        OCI_ATTR_CORRELATION, errhp));

    /* メッセージをエンキューしてコミットします。*/
    checkerr(errhp, OCIAQEnq(svchp, errhp, (CONST text *)"msgqueue",
        0, msgprop, mesgtdo, (dvoid **)&mesg,
        (dvoid **)&mesgind, 0, 0));

    checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));
}
/* enqmesg を終了します。*/

```



```

void main()
{
    OCIServer      *srvhp;
    OCISession     *usrhp;
    dvoid          *tmp;
    text           buf[80];          /* ユーザー定義テキスト */
    int            msgno = 0;

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
                  (dvoid * (*)()) 0, (void (*)()) 0 );

    OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                   52, (dvoid **) &tmp);

    OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp );

    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                   52, (dvoid **) &tmp);
    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                   52, (dvoid **) &tmp);

    OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                   52, (dvoid **) &tmp);

    /* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
    OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
               (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

    /* ユーザー・コンテキスト・ハンドルを割り当てます。*/
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                   (size_t) 0, (dvoid **) 0);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
               (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
               (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

    checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                     OCI_DEFAULT));

    OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
               (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

```

```
/* 関連識別子を充填するメッセージ・プロパティ記述子を割り当てます。*/
checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&msgprop,
    OCI_DTYPE_AQMSG_PROPERTIES,
    0, (dvoid **)0));

do {
    printf("Enter a line of text (max 80 chars):");
    if (!gets((char *)buf))
        break;
    enqmesg((ub4)msgno++, buf);
} while(1);

/* メッセージ・プロパティ記述子を解放します。*/
checkerr(errhp, OCIDescriptorFree((dvoid *)msgprop,
    OCI_DTYPE_AQMSG_PROPERTIES));

} /* main を終了します。*/

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    if (status == OCI_SUCCESS) return;

    switch (status)
    {
    case OCI_ERROR:
        OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
            errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
        printf("Error - %s\n", errbuf);
        break;
    case OCI_INVALID_HANDLE:
        printf("Error - OCI_INVALID_HANDLE\n");
        break;
    default:
        printf("Error - %d\n", status);
        break;
    }
    exit(-1);
} /* checkerr を終了します。*/
```

## OCI を使用したメッセージのデキュー（各コール後のメモリー解放）

このプログラム deqgnoreuse.c は、create\_types.sql を実行して scott のスキーマに作成されたキュー msgqueue から、テキストの各行をデキューします。メッセージは、enqgnoreuse または enqgreuse を使用してエンキューされます。メッセージがない場合は、60 秒待機してからタイムアウトします。このプログラムでは、デキュー・サブルーチンはクライアント側のオブジェクトのメモリーを再使用しません。必要なメモリーをデキューの前に割り当て、デキューの完了後にそのメモリーを解放します。

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

OCIEnv      *envhp;
OCIError     *errhp;
OCISvcCtx    *svchp;

struct message
{
    OCINumber    id;
    OCIString     *data;
};
typedef struct message message;

struct null_message
{
    OCIInd      null_adt;
    OCIInd      null_id;
    OCIInd      null_data;
};
typedef struct null_message null_message;

static void deqmesg(buf, buflen)
text      *buf;
ub4       *buflen;
{
    OCIType      *mesgtdo = (OCIType *)0; /* SCOTT.MESSAGE の型記述子 */
    message      *mesg = (dvoid *)0;      /* SCOTT.MESSAGE のインスタンス */
    null_message *mesgind = (dvoid *)0;    /* NULL 標識 */
    OCIAQDeqOptions *deqopt = (OCIAQDeqOptions *)0;
```

```
ub4          wait          = 60;                /* 60 秒後にタイムアウト */
ub4          navigation = OCI_DEQ_FIRST_MSG; /* 常に q の先頭を取得 */

/* SCOTT.MESSAGE 型の型記述子オブジェクトを取得します。 */
checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
    (CONST text *) "SCOTT", strlen("SCOTT"),
    (CONST text *) "MESSAGE", strlen("MESSAGE"),
    (text *) 0, 0, OCI_DURATION_SESSION,
    OCI_TYPEGET_ALL, &mesgtdo));

/* SCOTT.MESSAGE のインスタンスを割り当て、その NULL 標識を取得します。 */
checkerr(errhp, OCIObjectNew(envhp, errhp, svchp, OCI_TYPECODE_OBJECT,
    mesgtdo, (dvoid *) 0, OCI_DURATION_SESSION,
    TRUE, (dvoid **) &mesg));
checkerr(errhp, OCIObjectGetInd(envhp, errhp, (dvoid *) mesg,
    (dvoid **) &mesgind));

/* デキュー・オプションに記述子を割り当て、待機時間、ナビゲーションを設定します。 */
checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **) &deqopt,
    OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **) 0));
checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
    (dvoid *) &wait, 0, OCI_ATTR_WAIT, errhp));
checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
    (dvoid *) &navigation, 0,
    OCI_ATTR_NAVIGATION, errhp));

/* メッセージをデキューしてコミットします。 */
checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *) "msgqueue",
    deqopt, 0, mesgtdo, (dvoid **) &mesg,
    (dvoid **) &mesgind, 0, 0));

checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

/* メッセージ・ペイロード・テキストをユーザー・バッファにコピーします。 */
if (mesgind->null_data)
    *buflen = 0;
else
    memcpy((dvoid *) buf, (dvoid *) OCIStrPtr(envhp, mesg->data),
        (size_t) (*buflen = OCIStrSize(envhp, mesg->data)));

/* デキュー・オプション記述子を解放します。 */
checkerr(errhp, OCIDescriptorFree((dvoid *) deqopt, OCI_DTYPE_AQDEQ_OPTIONS));

/* オブジェクトのメモリーを解放します。 */
checkerr(errhp, OCIObjectFree(envhp, errhp, (dvoid *) mesg,
```

```

        OCI_OBJECTFREE_FORCE));
    }
    /* deqmesg を終了します。*/

void main()
{
    OCIServer      *srvhp;
    OCISession     *usrhp;
    dvoid          *tmp;
    text           buf[80];
    ub4            buflen;

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
                  (dvoid * (*)()) 0, (void (*)()) 0 );

    OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                   52, (dvoid **) &tmp);

    OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp );

    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                   52, (dvoid **) &tmp);
    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                   52, (dvoid **) &tmp);

    OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                   52, (dvoid **) &tmp);

    /* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
    OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
               (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

    /* ユーザー・コンテキスト・ハンドルを割り当てます。*/
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                   (size_t) 0, (dvoid **) 0);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
               (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
               (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

    checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                     OCI_DEFAULT));

```

```

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
           (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

do {
    deqmesg(buf, &buflen);
    printf("%.s\n", buflen, buf);
} while(1);
} /* main を終了します。*/

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    if (status == OCI_SUCCESS) return;

    switch (status)
    {
    case OCI_ERROR:
        OCIErrorGet((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                   errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
        printf("Error - %s\n", errbuf);
        break;
    case OCI_INVALID_HANDLE:
        printf("Error - OCI_INVALID_HANDLE\n");
        break;
    default:
        printf("Error - %d\n", status);
        break;
    }
    exit(-1);
} /* checkerr を終了します。*/

```

## OCI を使用したメッセージのデキュー（メモリーの再使用）

このプログラム `degreuse.c` は、`create_types.sql` を実行して `scott` のスキーマに作成されたキュー `msgqueue` から、テキストの各行をデキューします。メッセージは `engnignoreuse.c` または `engreuse.c` を使用してエンキューされます。メッセージがない場合は、60 秒待機してからタイムアウトします。このプログラムでは、デキュー・サブルーチンは、`OCIAQDeq` の起動間でクライアント側のオブジェクトのメモリーを再使用します。`OCIAQDeq` に対する最初のコール中に、OCI はメッセージ・ペイロード用のメモリーを自動的に割り当てます。`OCIAQDeq` に対する後続のコール中に、同一のペイロード・ポインタが渡され、OCI は必要に応じてペイロード・メモリーのサイズを自動的に変更します。

```

#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void degmesg(text *buf, ub4 *buflen);

struct message
{
    OCINumber    id;
    OCIStrng     *data;
};
typedef struct message message;

struct null_message
{
    OCIInd       null_adt;
    OCIInd       null_id;
    OCIInd       null_data;
};
typedef struct null_message null_message;

/* エンキューのためのコールで再利用するグローバル・データ */
OCIEnv          *envhp;
OCIError         *errhp;
OCISvcCtx        *svchp;
OCIAQDeqOptions  *deqopt;
message          *mesg = (message *)0;
null_message     *mesgind = (null_message *)0;

static void degmesg(buf, buflen)
text             *buf;
ub4              *buflen;
{

```

```

OCIType      *mesgtdo    = (OCIType *)0; /* SCOTT.MESSAGE の型記述子 */
ub4          wait       = 60;           /* 60 秒後にタイムアウト */
ub4          navigation = OCI_DEQ_FIRST_MSG; /* 常に q の先頭を取得 */

/* SCOTT.MESSAGE 型の型記述子オブジェクトを取得します。 */
checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
    (CONST text *)"SCOTT", strlen("SCOTT"),
    (CONST text *)"MESSAGE", strlen("MESSAGE"),
    (text *)0, 0, OCI_DURATION_SESSION,
    OCI_TYPEGET_ALL, &mesgtdo));

/* デキュー・オプションに待機時間、ナビゲーションを設定します。 */
checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
    (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
    (dvoid *)&navigation, 0,
    OCI_ATTR_NAVIGATION, errhp));

/*
 * メッセージをデキューしてコミットします。ペイロードのメモリーは、OCI によって自動的に
 * 割当て / サイズ変更されます。
 */
checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
    deqopt, 0, mesgtdo, (dvoid **)&mesg,
    (dvoid **)&mesgind, 0, 0));

checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

/* メッセージ・ペイロード・テキストをユーザー・バッファにコピーします。 */
if (mesgind->null_data)
    *buflen = 0;
else
    memcpy((dvoid *)buf, (dvoid *)OCIStrPtr(envhp, mesg->data),
        (size_t) (*buflen = OCIStrSize(envhp, mesg->data)));
} /* deqmesg を終了します。 */

void main()
{
    OCIServer      *srvhp;
    OCISession     *usrhp;
    dvoid          *tmp;
    text           buf[80]; /* ペイロード・テキスト */
    ub4            buflen;

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
        (dvoid * (*)()) 0, (void (*)()) 0);

```



```
OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

/* サービス・コンテキストに属性サーバー・コンテキストを設定します。*/
OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* ユーザー・コンテキスト・ハンドルを割り当てます。*/
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
               (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
           (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
           (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                OCI_DEFAULT));

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
           (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* デキュー・オプション記述子を割り当てます。*/
checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
                                   OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));

do {
    deqmesg(buf, &buflen);
    printf("%s.\n", buflen, buf);
} while(1);
```

```
/*
 * このプログラムが、デキュー・タイムアウトおよび終了としてこの地点に到達することはあり
 * ません。到達した場合は、OCIDescriptorFree を使用してデキュー・オプション記述子を解放
 * し、OCI によって割り当てられたメモリーを、OCIObjectFree を使用してペイロードに解放して
 * ください。
 */
}                               /* main を終了します。*/

static void checkerr(errhp, status)
OCIError *errhp;
sword      status;
{
    text errbuf[512];
    ub4   buflen;
    sb4   errcode;

    if (status == OCI_SUCCESS) return;

    switch (status)
    {
    case OCI_ERROR:
        OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                     errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
        printf("Error - %s\n", errbuf);
        break;
    case OCI_INVALID_HANDLE:
        printf("Error - OCI_INVALID_HANDLE\n");
        break;
    default:
        printf("Error - %d\n", status);
        break;
    }
    exit(-1);
}                               /* checkerr を終了します。*/
```

---

## Oracle JMS インタフェース、クラス および例外

---

**表 B-1 インタフェース、クラスおよび例外**

---

**インタフェース / クラス / 例外**

---

B-5 ページの「 <a href="#">Oracle JMS クラス (パート 1)</a> 」
B-6 ページの「 <a href="#">Oracle JMS クラス (パート 2)</a> 」
B-7 ページの「 <a href="#">Oracle JMS クラス (パート 3)</a> 」
B-8 ページの「 <a href="#">Oracle JMS クラス (パート 4)</a> 」
B-9 ページの「 <a href="#">Oracle JMS クラス (パート 5)</a> 」
B-10 ページの「 <a href="#">Oracle JMS クラス (パート 6)</a> 」
B-12 ページの「 <a href="#">Oracle JMS クラス (パート 7)</a> 」
B-13 ページの「 <a href="#">Oracle JMS クラス (パート 8)</a> 」
B-14 ページの「 <a href="#">Oracle JMS クラス (パート 9)</a> 」
B-15 ページの「 <a href="#">Oracle JMS クラス (パート 10)</a> 」
B-17 ページの「 <a href="#">インタフェース - javax.jms.BytesMessage</a> 」
B-18 ページの「 <a href="#">インタフェース - javax.jms.Connection</a> 」
B-19 ページの「 <a href="#">インタフェース - javax.jms.ConnectionFactory</a> 」
B-20 ページの「 <a href="#">インタフェース - javax.jms.ConnectionMetaData</a> 」
B-21 ページの「 <a href="#">インタフェース - javax.jms.DeliveryMode</a> 」
B-22 ページの「 <a href="#">インタフェース - javax.jms.Destination</a> 」
B-23 ページの「 <a href="#">インタフェース - javax.jms.MapMessage</a> 」
B-24 ページの「 <a href="#">インタフェース - javax.jms.Message</a> 」
B-25 ページの「 <a href="#">インタフェース - javax.jms.MessageConsumer</a> 」
B-26 ページの「 <a href="#">インタフェース - javax.jms.MessageListener</a> 」
B-27 ページの「 <a href="#">インタフェース - javax.jms.MessageProducer</a> 」
B-28 ページの「 <a href="#">インタフェース - javax.jms.ObjectMessage</a> 」
B-29 ページの「 <a href="#">インタフェース - javax.jms.Queue</a> 」
B-30 ページの「 <a href="#">インタフェース - javax.jms.QueueBrowser</a> 」
B-31 ページの「 <a href="#">インタフェース - javax.jms.QueueConnection</a> 」
B-32 ページの「 <a href="#">インタフェース - javax.jms.QueueConnectionFactory</a> 」

---

**表 B-1 インタフェース、クラスおよび例外（続き）**

---

**インタフェース / クラス / 例外**

---

B-33	ページの「 <a href="#">インタフェース - javax.jms.QueueReceiver</a> 」
B-34	ページの「 <a href="#">インタフェース - javax.jms.QueueSender</a> 」
B-35	ページの「 <a href="#">インタフェース - javax.jms.QueueSession</a> 」
B-36	ページの「 <a href="#">インタフェース - javax.jms.Session</a> 」
B-37	ページの「 <a href="#">インタフェース - javax.jms.StreamMessage</a> 」
B-38	ページの「 <a href="#">インタフェース - javax.jms.TextMessage</a> 」
B-39	ページの「 <a href="#">インタフェース - javax.jms.Topic</a> 」
B-40	ページの「 <a href="#">インタフェース - javax.jms.TopicConnection</a> 」
B-41	ページの「 <a href="#">インタフェース - javax.jms.TopicConnectionFactory</a> 」
B-42	ページの「 <a href="#">インタフェース - javax.jms.TopicPublisher</a> 」
B-43	ページの「 <a href="#">インタフェース - javax.jms.TopicSession</a> 」
B-44	ページの「 <a href="#">インタフェース - javax.jms.TopicSubscriber</a> 」
B-45	ページの「 <a href="#">例外 - javax.jms.InvalidDestinationException</a> 」
B-46	ページの「 <a href="#">例外 - javax.jms.InvalidSelectorException</a> 」
B-47	ページの「 <a href="#">例外 - javax.jms.JMSEException</a> 」
B-48	ページの「 <a href="#">例外 - javax.jms.MessageEOFException</a> 」
B-49	ページの「 <a href="#">例外 - javax.jms.MessageFormatException</a> 」
B-50	ページの「 <a href="#">例外 - javax.jms.MessageNotReadableException</a> 」
B-51	ページの「 <a href="#">例外 - javax.jms.MessageNotWriteableException</a> 」
B-52	ページの「 <a href="#">インタフェース - oracle.jms.AdtMessage</a> 」
B-53	ページの「 <a href="#">インタフェース - oracle.jms.AQjmsQueueReceiver</a> 」
B-54	ページの「 <a href="#">インタフェース - oracle.jms.AQjmsQueueSender</a> 」
B-55	ページの「 <a href="#">インタフェース - oracle.jms.AQjmsTopicPublisher</a> 」
B-56	ページの「 <a href="#">インタフェース - oracle.jms.TopicReceiver</a> 」
B-57	ページの「 <a href="#">インタフェース - oracle.jms.AQjmsTopicSubscriber</a> 」
B-58	ページの「 <a href="#">インタフェース - oracle.jms.AQjmsTopicReceiver</a> 」

---

**表 B-1 インタフェース、クラスおよび例外（続き）**

---

**インタフェース / クラス / 例外**

---

B-59	ページの「クラス - <a href="#">oracle.jms.AQjmsAdtMessage</a> 」
B-60	ページの「クラス - <a href="#">oracle.jms.AQjmsAgent</a> 」
B-61	ページの「クラス - <a href="#">oracle.jms.AQjmsBytesMessage</a> 」
B-62	ページの「クラス - <a href="#">oracle.jms.AQjmsConnection</a> 」
B-63	ページの「インタフェース - <a href="#">oracle.jms.AQjmsConnectionMetadata</a> 」
B-64	ページの「クラス - <a href="#">oracle.jms.AQjmsConstants</a> 」
B-65	ページの「インタフェース - <a href="#">oracle.jms.AQjmsConsumer</a> 」
B-66	ページの「クラス - <a href="#">oracle.jms.AQjmsDestination</a> 」
B-67	ページの「クラス - <a href="#">oracle.jms.AQjmsDestinationProperty</a> 」
B-68	ページの「クラス - <a href="#">oracle.jms.AQjmsFactory</a> 」
B-69	ページの「クラス - <a href="#">oracle.jms.AQjmsMapMessage</a> 」
B-70	ページの「クラス - <a href="#">oracle.jms.AQjmsMessage</a> 」
B-71	ページの「クラス - <a href="#">oracle.jms.AQjmsObjectMessage</a> 」
B-72	ページの「クラス - <a href="#">oracle.jms.AQjmsOracleDebug</a> 」
B-73	ページの「クラス - <a href="#">oracle.jms.AQjmsProducer</a> 」
B-74	ページの「クラス - <a href="#">oracle.jms.AQjmsQueueBrowser</a> 」
B-75	ページの「クラス - <a href="#">AQjmsQueueConnectionFactory</a> 」
B-76	ページの「クラス - <a href="#">oracle.jms.AQjmsSession</a> 」
B-77	ページの「クラス - <a href="#">oracle.jms.AQjmsStreamMessage</a> 」
B-78	ページの「クラス - <a href="#">oracle.jms.AQjmsTextMessage</a> 」
B-79	ページの「クラス - <a href="#">oracle.jms.AQjmsTopicConnectionFactory</a> 」
B-81	ページの「例外 - <a href="#">oracle.jms.AQjmsInvalidDestinationException</a> 」
B-82	ページの「例外 - <a href="#">oracle.jms.AQjmsInvalidSelectorException</a> 」
B-83	ページの「例外 - <a href="#">oracle.jms.AQjmsMessageEOFException</a> 」
B-84	ページの「例外 - <a href="#">oracle.jms.AQjmsMessageFormatException</a> 」
B-85	ページの「例外 - <a href="#">oracle.jms.AQjmsMessageNotReadableException</a> 」

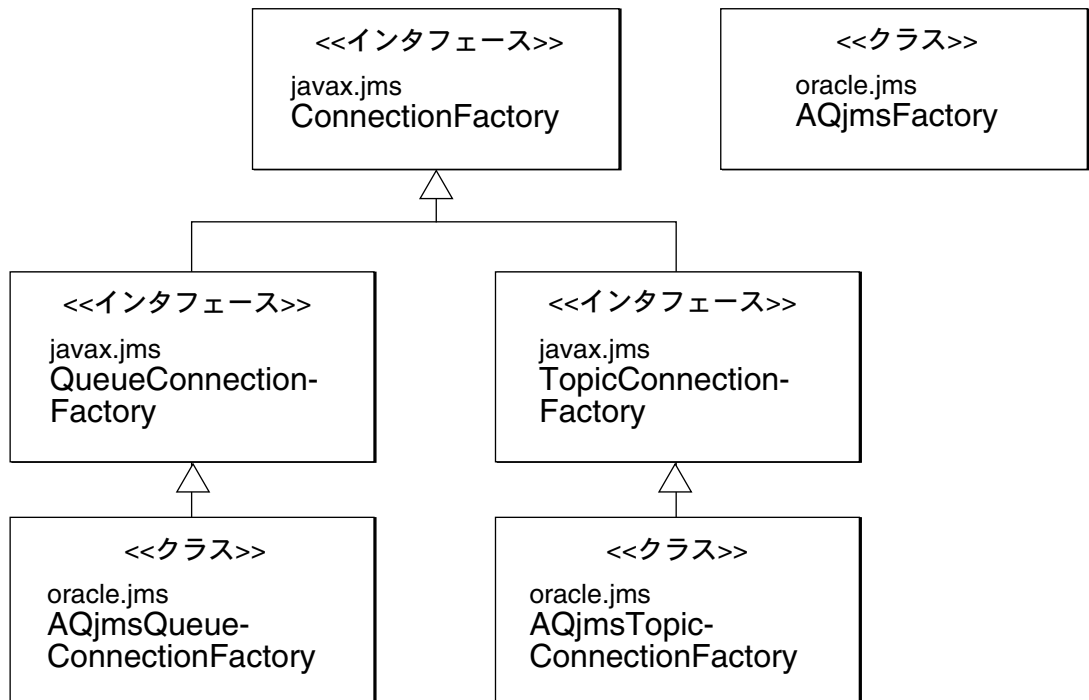
表 B-1 インタフェース、クラスおよび例外 (続き)

## インタフェース / クラス / 例外

B-86 ページの「例外 - [oracle.jms.AQjmsMessageNotWriteableException](#)」B-87 ページの「インタフェース - [oracle.AQ.AQQueueTable](#)」B-88 ページの「クラス - [oracle.AQ.AQQueueTableProperty](#)」

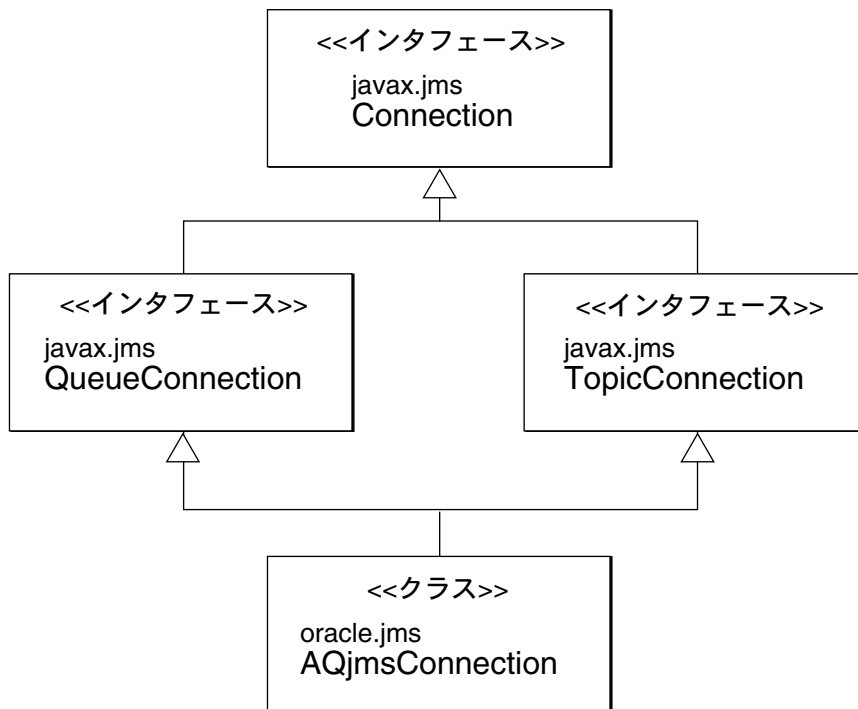
## Oracle JMS クラス (パート 1)

図 B-1 クラス図 : Oracle JMS クラス (パート 1)



## Oracle JMS クラス (パート 2)

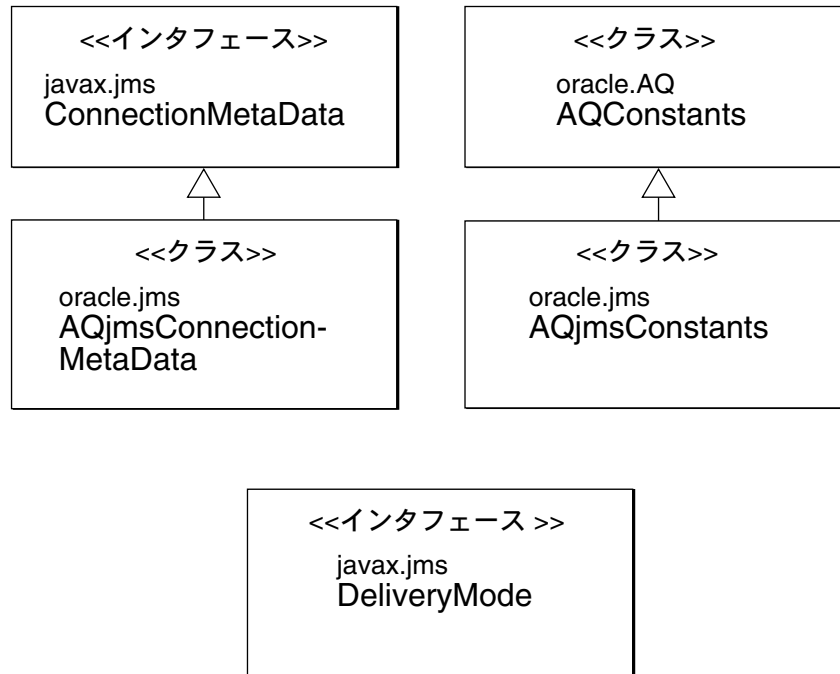
図 B-2 クラス図 : Oracle クラスのクラス (パート 2)





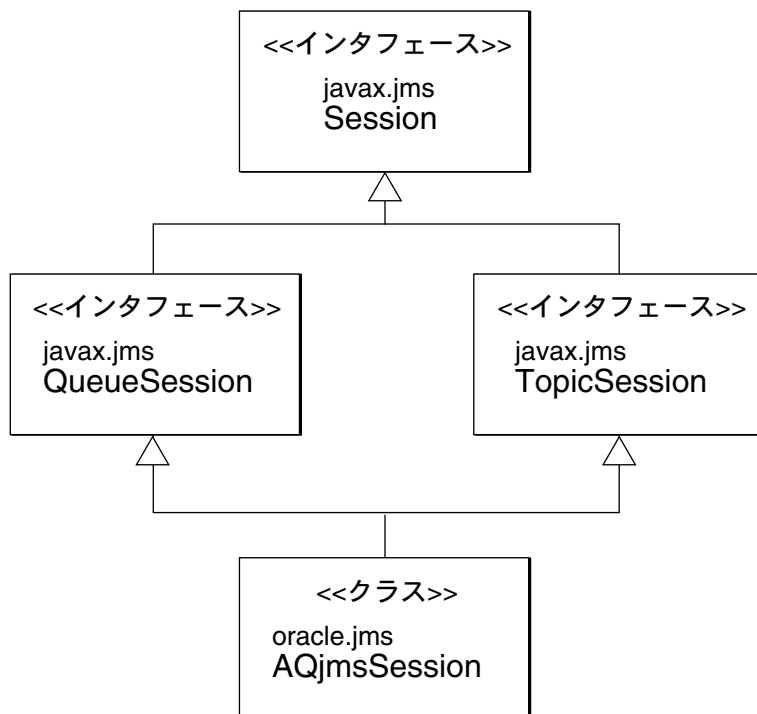
## Oracle JMS クラス (パート 3)

図 B-3 クラス図 : Oracle クラスのクラス (パート 3)



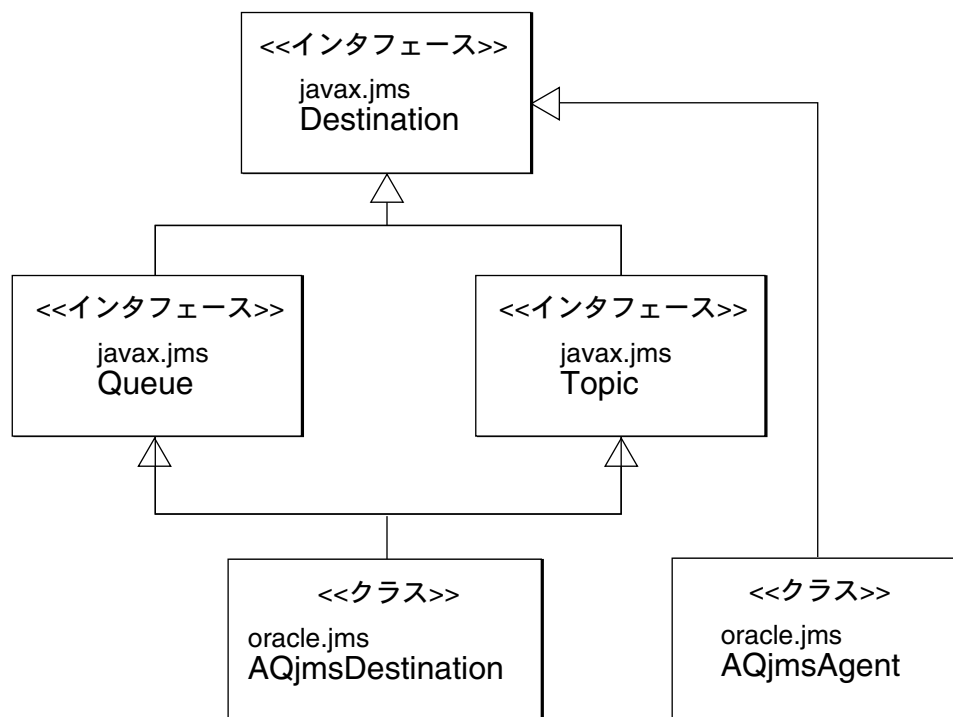
## Oracle JMS クラス (パート 4)

図 B-4 クラス図 : Oracle クラスのクラス (パート 4)



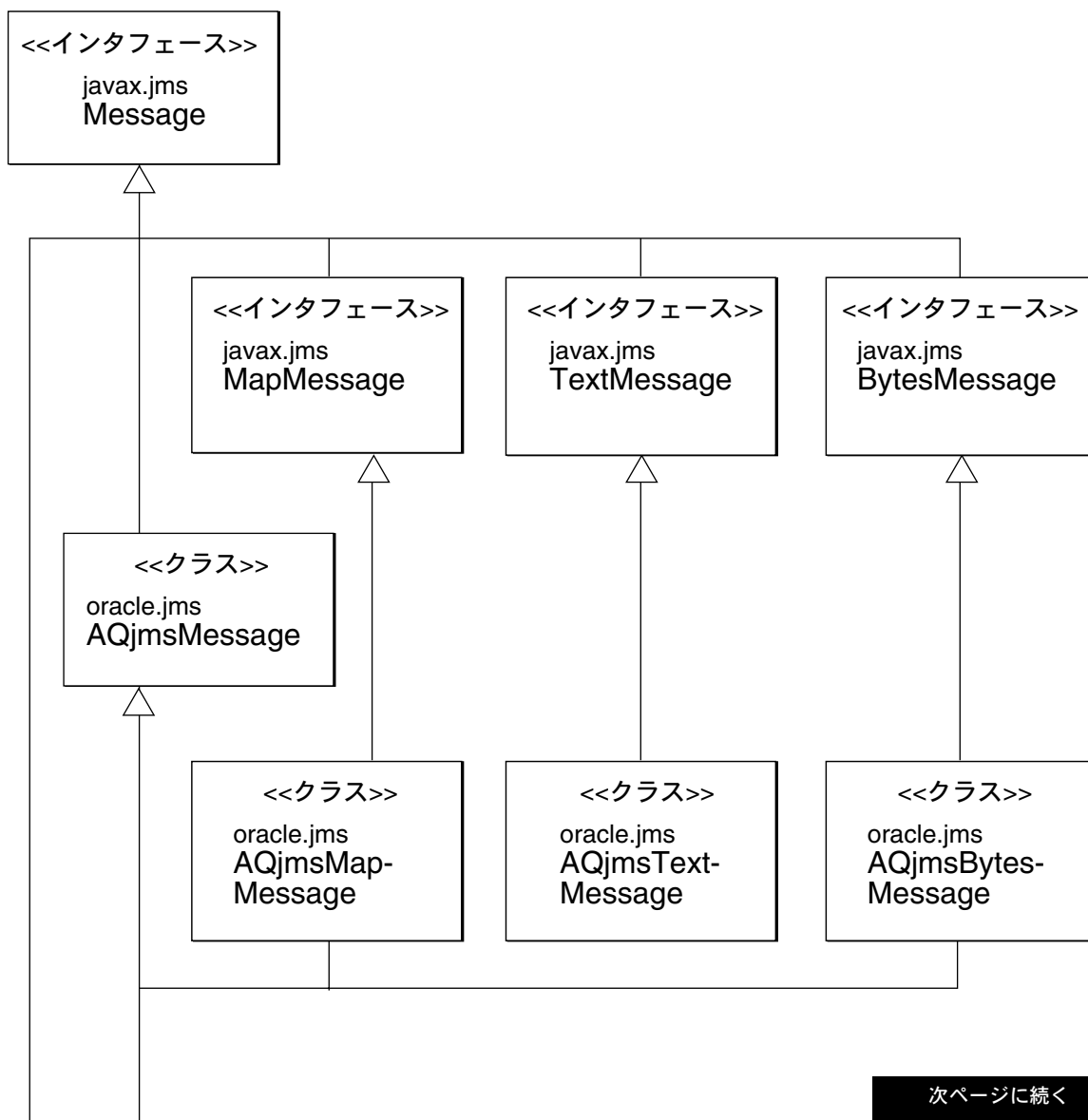
## Oracle JMS クラス (パート 5)

図 B-5 クラス図 : Oracle クラスのクラス (パート 5)



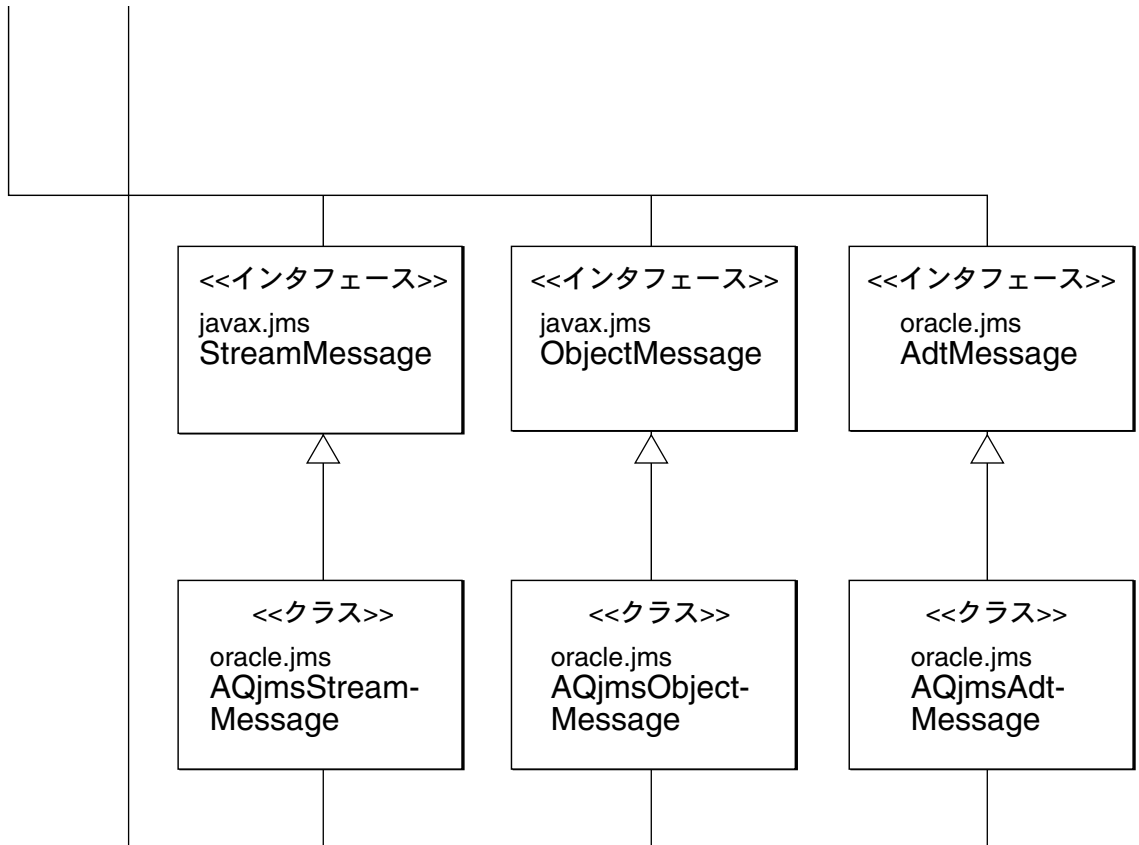
## Oracle JMS クラス (パート 6)

図 B-6 クラス図 : Oracle クラスのクラス (パート 6)



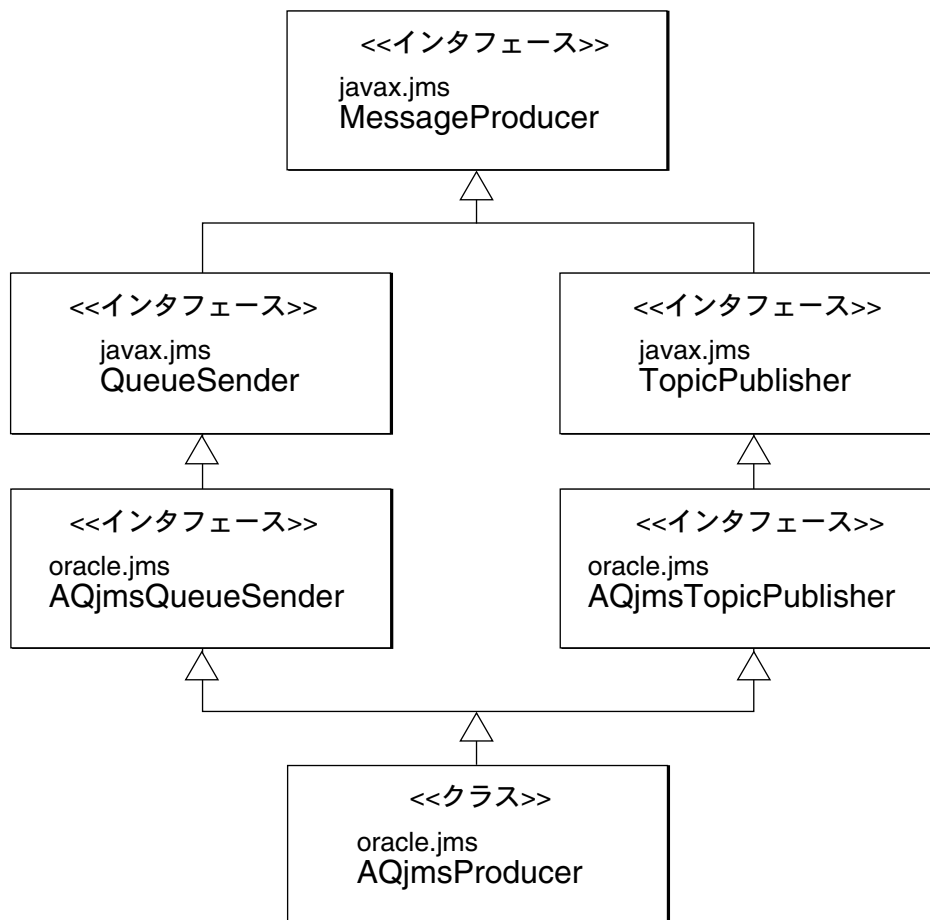
## Oracle JMS クラス（パート 6 の続き）

図 B-7 クラス図：Oracle クラスのクラス（パート 6）



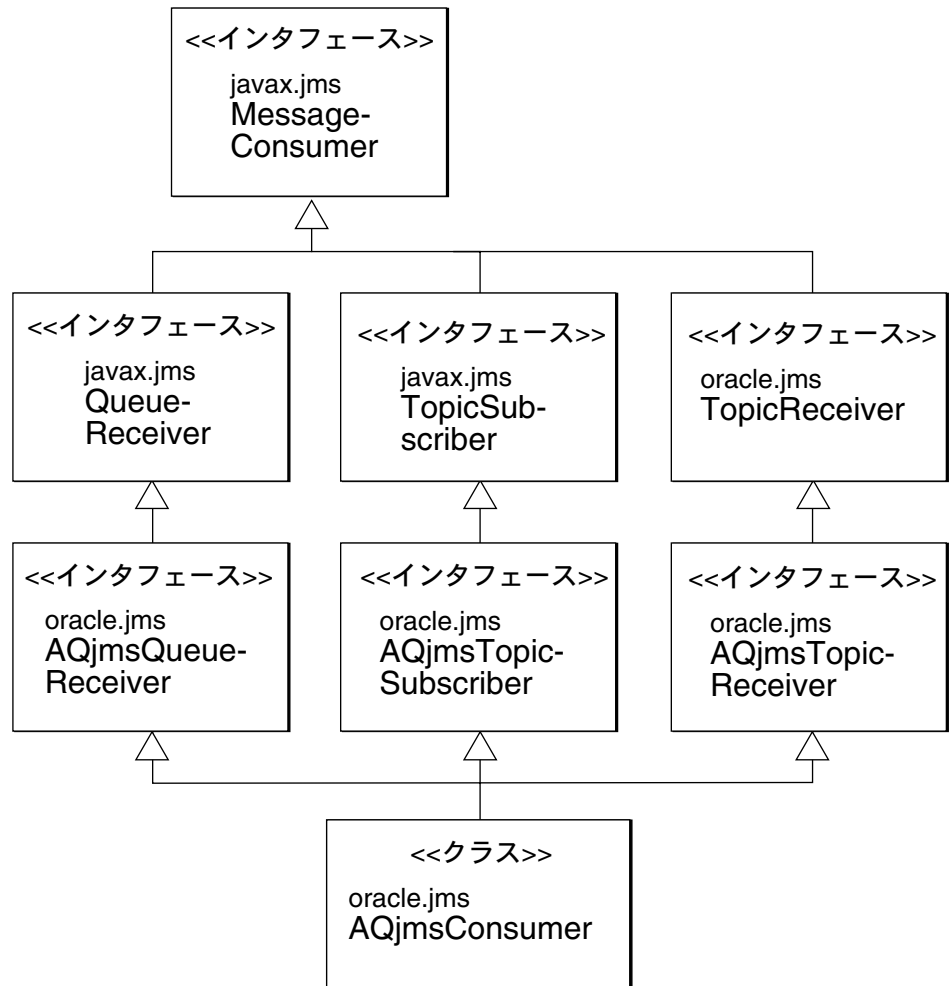
## Oracle JMS クラス (パート 7)

図 B-8 クラス図 : Oracle クラスのクラス (パート 7)



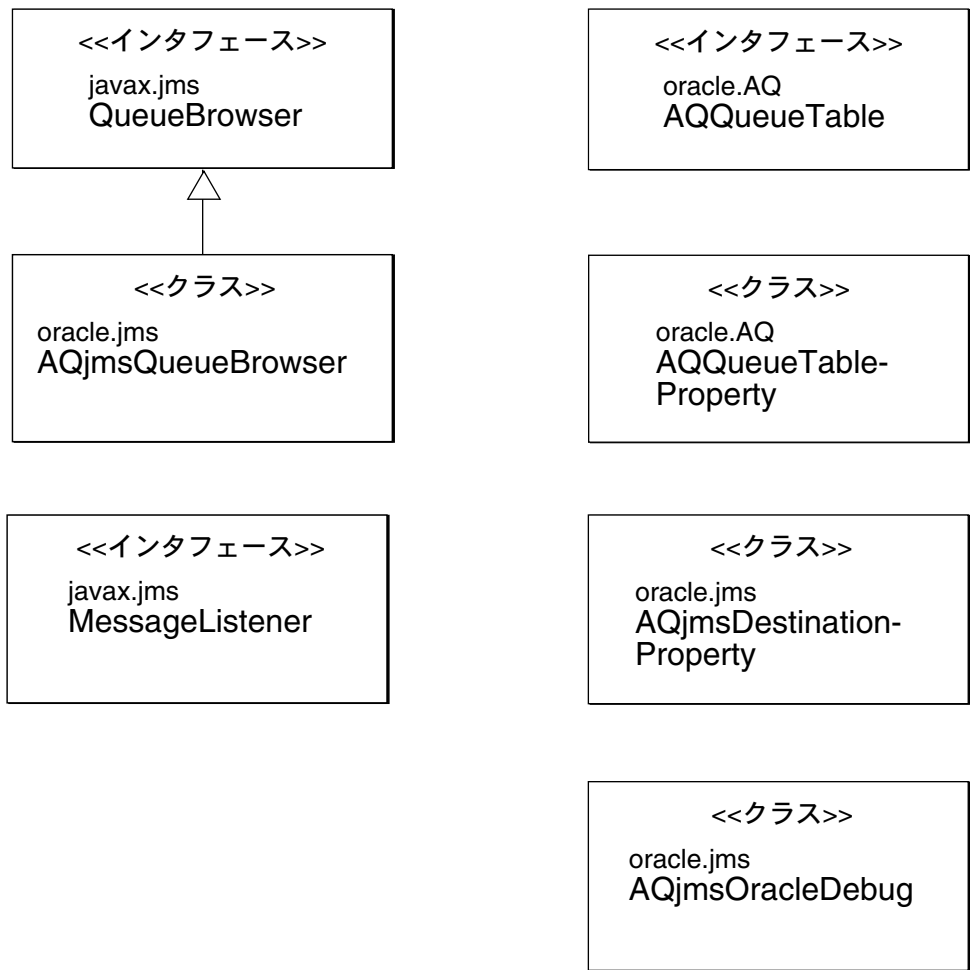
## Oracle JMS クラス (パート 8)

図 B-9 クラス図 : Oracle クラスのクラス (パート 8)



# Oracle JMS クラス (パート 9)

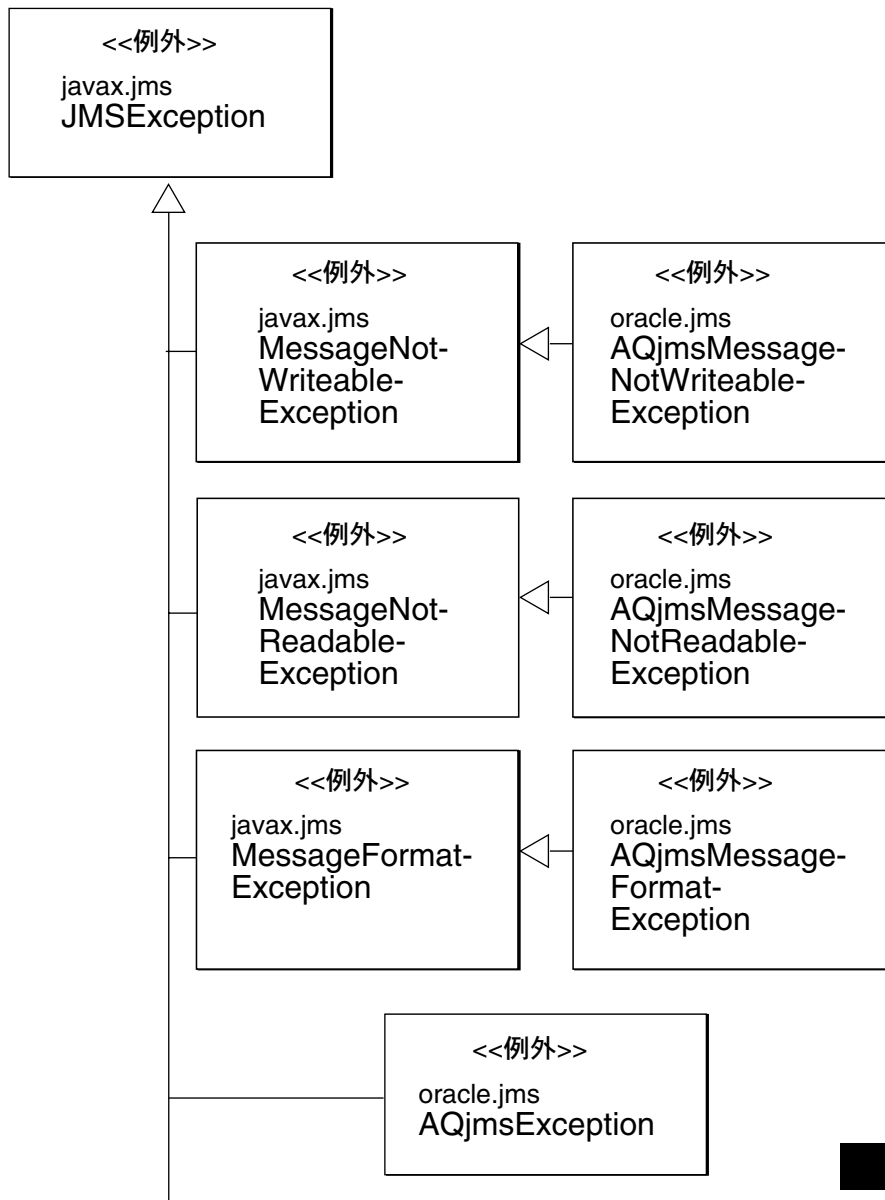
図 B-10 クラス図 : Oracle クラスのクラス (パート 9)





## Oracle JMS クラス (パート 10)

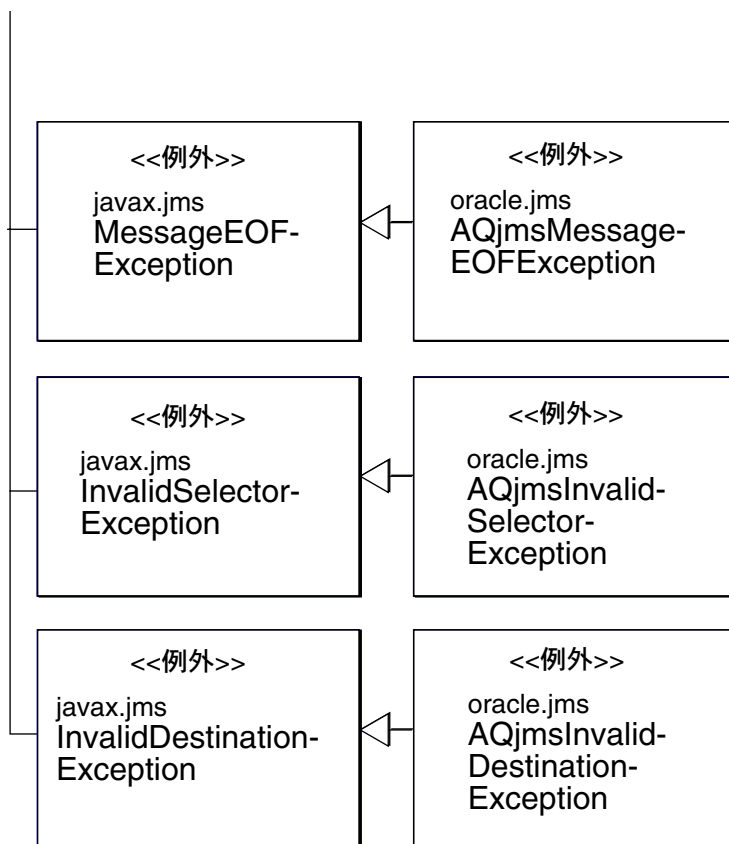
図 B-11 クラス図: Oracle クラスのクラス (パート 10)



次ページに続く

## Oracle JMS クラス（パート 10 の続き）

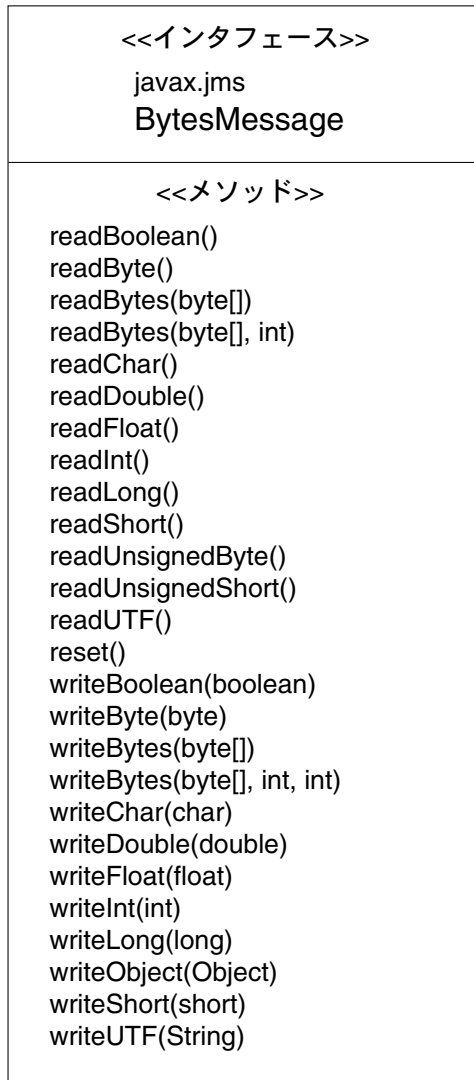
図 B-12 クラス図：Oracle クラスのクラス（パート 10）



## インタフェース - javax.jms.BytesMessage

図 B-13 クラス図: インタフェース - javax.jms.BytesMessage

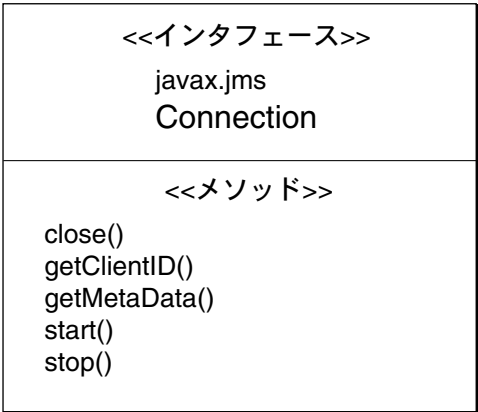
Oracle JMSクラス: javax.jms.BytesMessage



# インタフェース - javax.jms.Connection

図 B-14 クラス図：インタフェース - javax.jms.Connection

Oracle JMSクラス: javax.jms.Connection



# インタフェース - javax.jms.ConnectionFactory

図 B-15 クラス図：インタフェース - javax.jms.ConnectionFactory

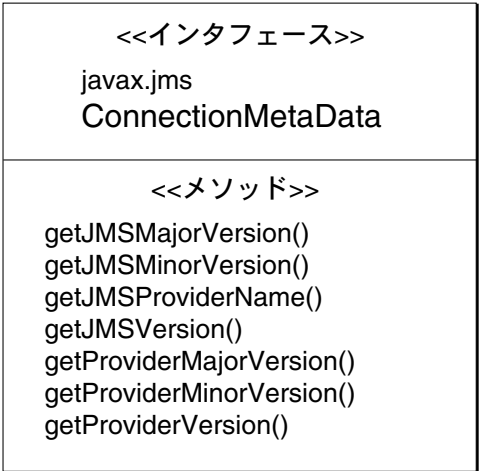
Oracle JMSクラス: javax.jms.ConnectionFactory



# インタフェース - javax.jms.ConnectionMetaData

図 B-16 クラス図: インタフェース - javax.jms.ConnectionMetaData

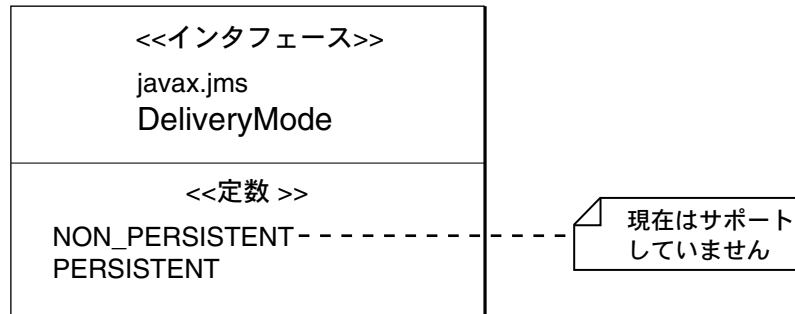
Oracle JMSクラス: javax.jms.ConnectionMetaData



# インタフェース - javax.jms.DeliveryMode

図 B-17 クラス図：インタフェース - javax.jms.DeliveryMode

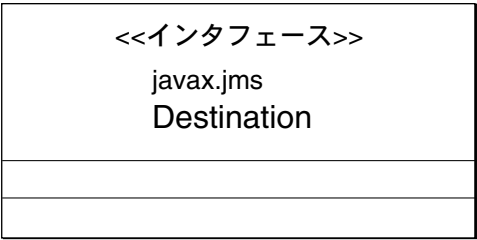
Oracle JMSクラス: javax.jms.DeliveryMode



# インタフェース - javax.jms.Destination

図 B-18 クラス図：インタフェース - javax.jms.Destination

Oracle JMSクラス: javax.jms.Destination

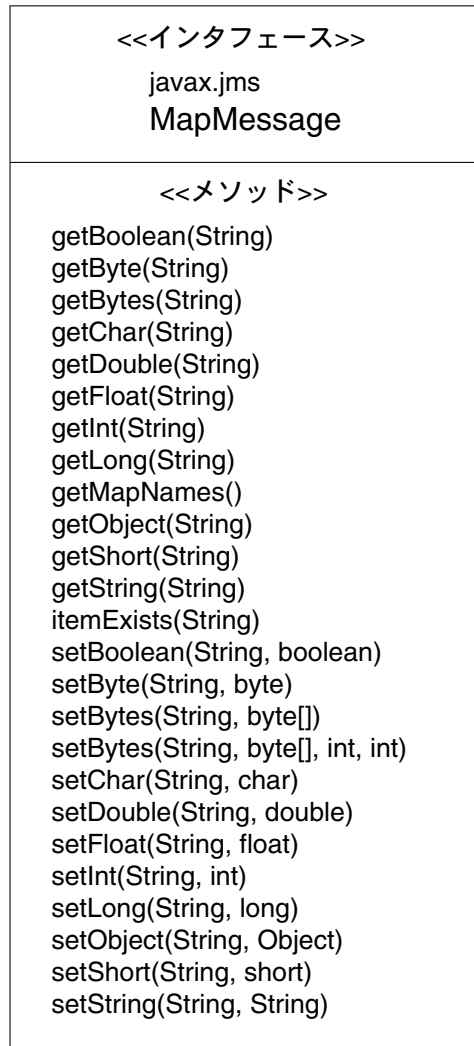




## インタフェース - javax.jms.MapMessage

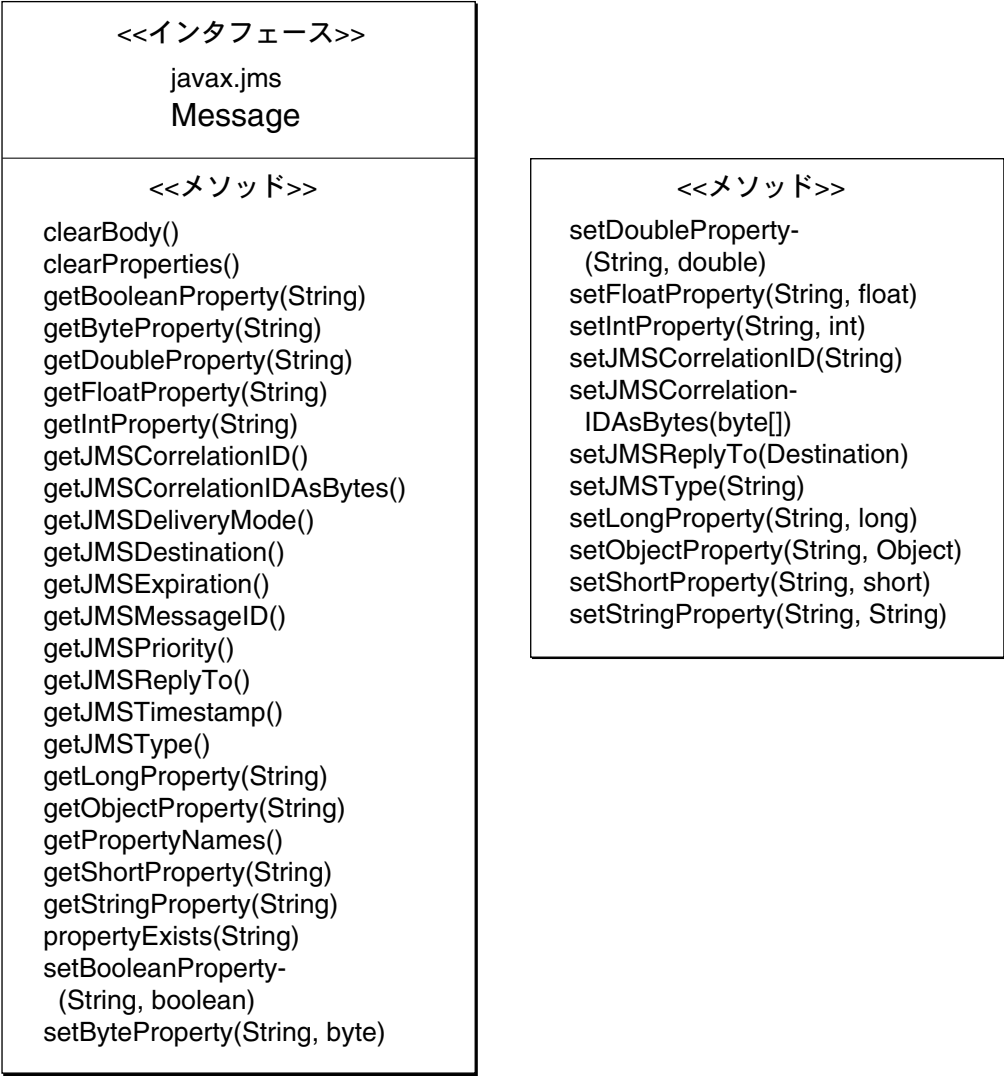
図 B-19 クラス図：インタフェース - javax.jms.MapMessage

Oracle JMSクラス: javax.jms.MapMessage



# インタフェース - javax.jms.Message

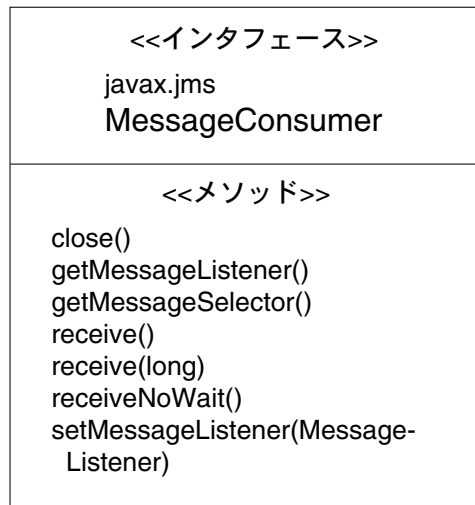
図 B-20 クラス図: インタフェース - javax.jms.Message



## インタフェース - javax.jms.MessageConsumer

図 B-21 クラス図：インタフェース - javax.jms.MessageConsumer

Oracle JMSクラス: javax.jms.MessageConsumer



## インタフェース - javax.jms.MessageListener

図 B-22 クラス図: インタフェース - javax.jms.MessageListener

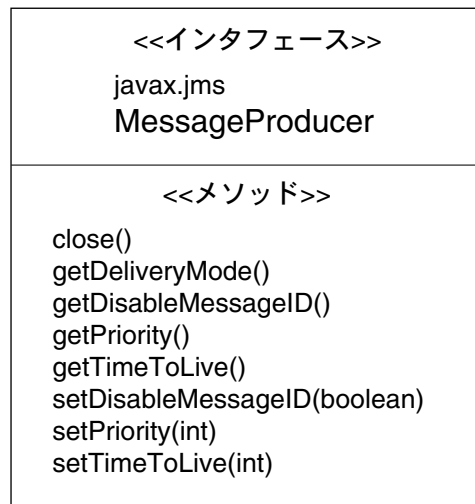
Oracle JMSクラス: javax.jms.MessageListener



## インタフェース - javax.jms.MessageProducer

図 B-23 クラス図: インタフェース - javax.jms.MessageProducer

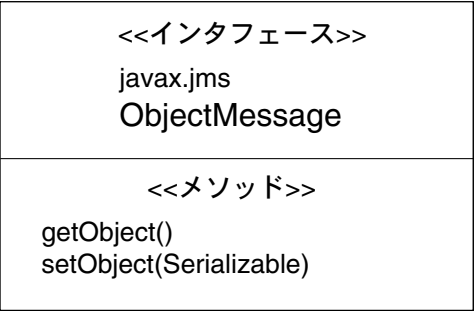
Oracle JMSクラス: javax.jms.MessageProducer



# インタフェース - javax.jms.ObjectMessage

図 B-24 クラス図：インタフェース - javax.jms.ObjectMessage

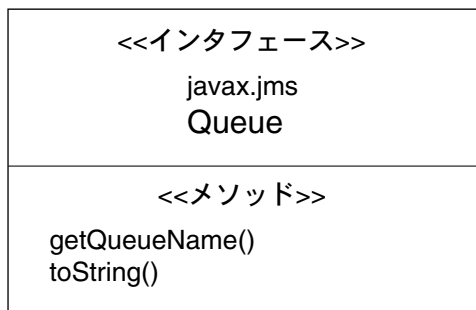
Oracle JMSクラス: javax.jms.ObjectMessage



## インタフェース - javax.jms.Queue

図 B-25 クラス図：インタフェース - javax.jms.Queue

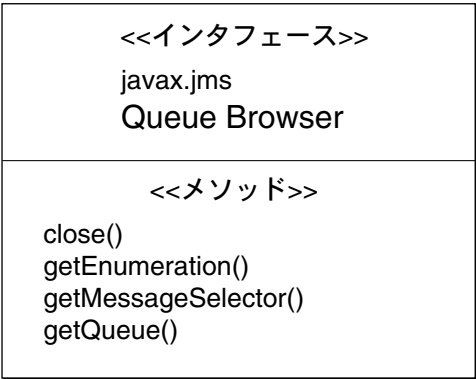
Oracle JMSクラス: javax.jms.Queue



# インタフェース - javax.jms.QueueBrowser

図 B-26 クラス図: インタフェース - javax.jms.QueueBrowser

Oracle JMSクラス: javax.jms.QueueBrowser

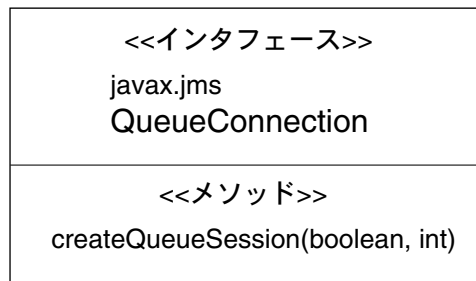




## インタフェース - javax.jms.QueueConnection

図 B-27 クラス図：インタフェース - javax.jms.QueueConnection

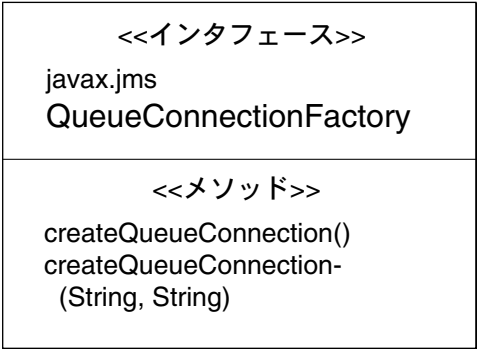
Oracle JMSクラス: javax.jms.QueueConnection



# インタフェース - javax.jms.QueueConnectionFactory

図 B-28 クラス図：インタフェース - javax.jms.QueueConnectionFactory

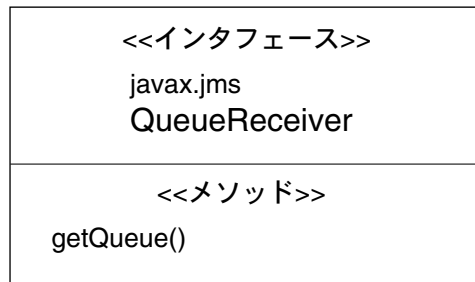
Oracle JMSクラス: javax.jms.QueueConnectionFactory



## インタフェース - javax.jms.QueueReceiver

図 B-29 クラス図：インタフェース - QueueReceiver

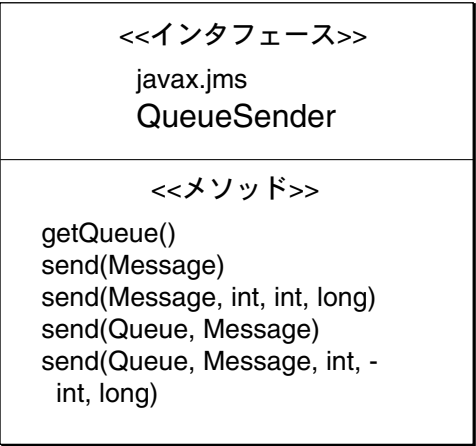
Oracle JMSクラス: javax.jms.QueueReceiver



# インタフェース - javax.jms.QueueSender

図 B-30 クラス図：インタフェース - javax.jms.QueueSender

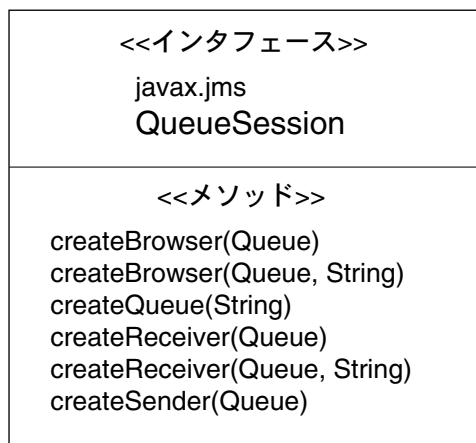
Oracle JMSクラス: javax.jms.QueueSender



## インタフェース - javax.jms.QueueSession

図 B-31 クラス図：インタフェース - javax.jms.QueueSession

Oracle JMSクラス: javax.jms.QueueSession



# インタフェース - javax.jms.Session

図 B-32 クラス図: インタフェース - javax.jms.Session

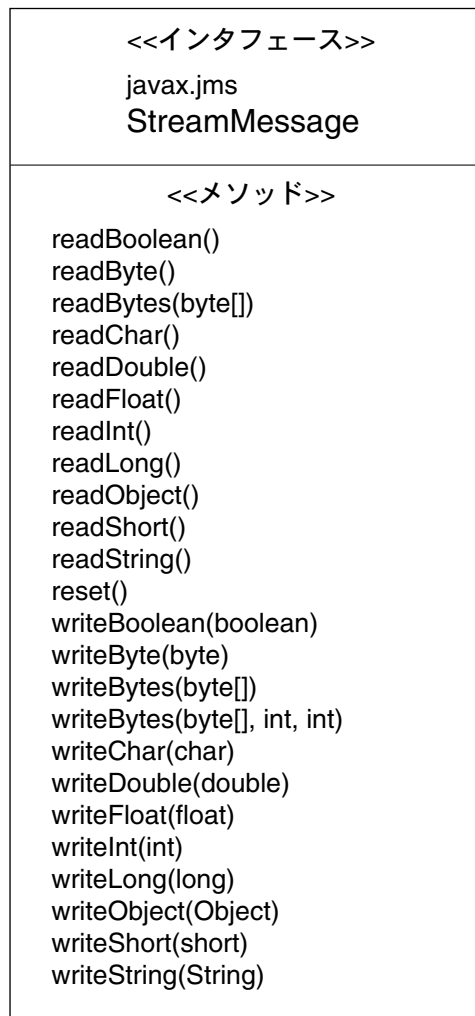
Oracle JMSクラス: javax.jms.Session



## インタフェース - javax.jms.StreamMessage

図 B-33 クラス図：インタフェース - javax.jms.StreamMessage

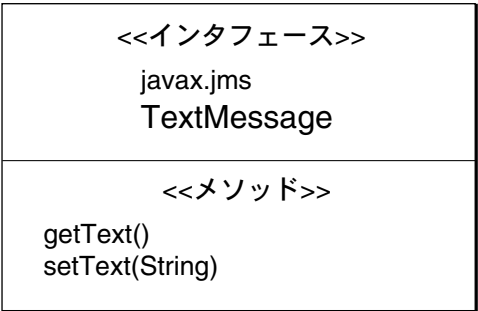
Oracle JMSクラス: javax.jms.StreamMessage



# インタフェース - javax.jms.TextMessage

図 B-34 クラス図: インタフェース - javax.jms.TextMessage

Oracle JMSクラス: javax.jms.TextMessage

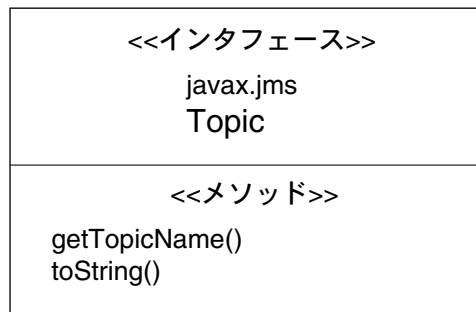




## インタフェース - javax.jms.Topic

図 B-35 クラス図：インタフェース - javax.jms.Topic

Oracle JMSクラス: javax.jms.Topic



## インタフェース - javax.jms.TopicConnection

図 B-36 クラス図: インタフェース - javax.jms.TopicConnection

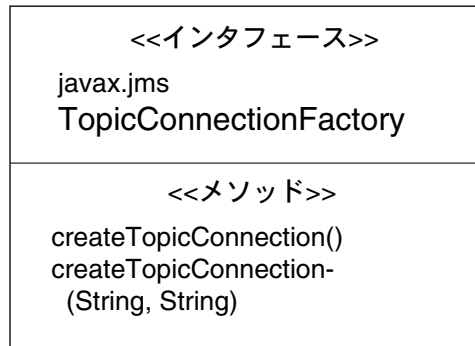
Oracle JMSクラス: javax.jms.TopicConnection



# インタフェース - javax.jms.TopicConnectionFactory

図 B-37 クラス図: インタフェース - javax.jms.TopicConnectionFactory

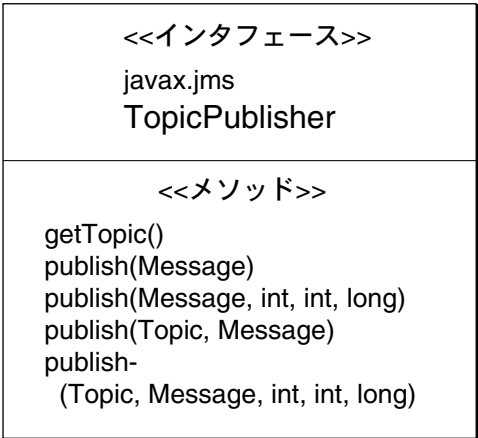
Oracle JMSクラス: javax.jms.TopicConnectionFactory



# インタフェース - javax.jms.TopicPublisher

図 B-38 クラス図：インタフェース - javax.jms.TopicPublisher

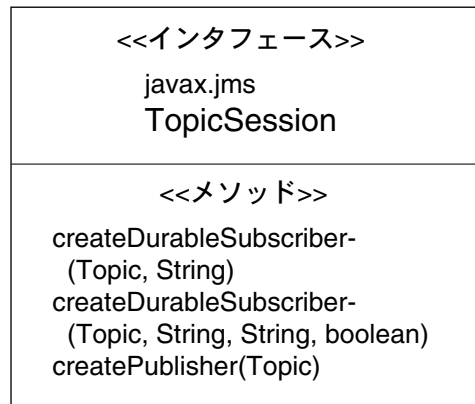
Oracle JMSクラス: javax.jms.TopicPublisher



## インタフェース - javax.jms.TopicSession

図 B-39 クラス図：インタフェース - javax.jms.TopicSession

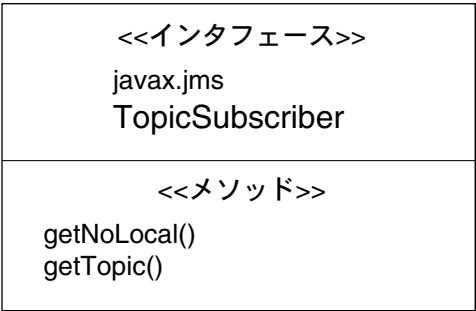
Oracle JMSクラス: javax.jms.TopicSession



# インタフェース - javax.jms.TopicSubscriber

図 B-40 クラス図：インタフェース - javax.jms.TopicSubscriber

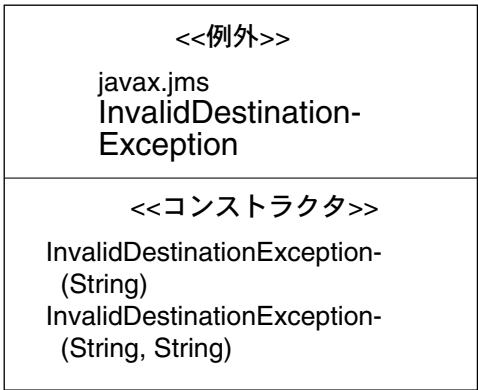
Oracle JMSクラス: javax.jms.TopicSubscriber



# 例外 - javax.jms.InvalidDestinationException

図 B-41 クラス図：例外 - javax.jms.InvalidDestinationException

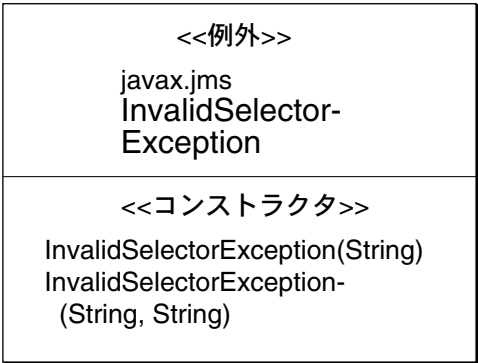
Oracle JMSクラス: javax.jms.InvalidDestinationException



## 例外 - javax.jms.InvalidSelectorException

図 B-42 クラス図：例外 - javax.jms.InvalidSelectorException

Oracle JMSクラス: javax.jms.InvalidSelectorException





## 例外 - javax.jms.JMSEException

図 B-43 クラス図：例外 - javax.jms.JMSEException

Oracle JMSクラス: javax.jms.JMSEException

<p>&lt;&lt;例外&gt;&gt;</p> <p>javax.jms JMSEException</p>
<p>&lt;&lt;コンストラクタ&gt;&gt;</p> <p>JMSEException(String) JMSEException(String, String)</p>
<p>&lt;&lt;メソッド&gt;&gt;</p> <p>getErrorCode() getLinkedException() setLinkedException(Exception)</p>

## 例外 - javax.jms.MessageEOFException

図 B-44 例外 - javax.jms.MessageEOFException

Oracle JMSクラス: javax.jms.MessageEOFException

<div>&lt;&lt;例外&gt;&gt;</div> <div>javax.jms MessageEOFException</div>
<div>&lt;&lt;コンストラクタ&gt;&gt;</div> <div>MessageEOFException(String) MessageEOFException- (String, String)</div>

## 例外 - javax.jms.MessageFormatException

図 B-45 例外 - javax.jms.MessageFormatException

Oracle JMSクラス: javax.jms.MessageFormatException

<p>&lt;&lt;例外&gt;&gt;</p> <p>javax.jms MessageFormatException</p>
<p>&lt;&lt;コンストラクタ&gt;&gt;</p> <p>MessageFormatException(String) MessageFormatException- (String, String)</p>

## 例外 - javax.jms.MessageNotReadableException

図 B-46 例外 - javax.jms.MessageNotReadableException

Oracle JMSクラス: javax.jms.MessageNotReadableException

<div>&lt;&lt;例外&gt;&gt;</div> <div>javax.jms MessageNotReadable- Exception</div>
<div>&lt;&lt;コンストラクタ&gt;&gt;</div> <div>MessageNotReadableException- (String) MessageNotReadableException- (String, String)</div>

# 例外 - javax.jms.MessageNotWriteableException

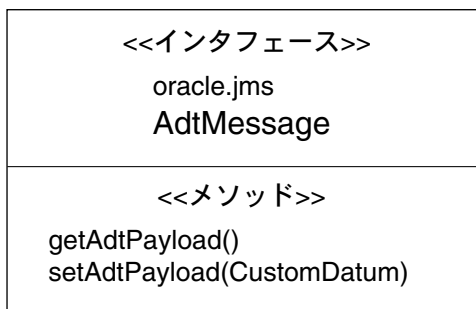
図 B-47 例外 - javax.jms.MessageNotWriteableException

<p>&lt;&lt;例外&gt;&gt;</p> <p>javax.jms MessageNotWriteable- Exception</p>
<p>&lt;&lt;コンストラクタ&gt;&gt;</p> <p>MessageNotWriteableException- (String) MessageNotWriteableException- (String, String)</p>

## インタフェース - oracle.jms.AdtMessage

図 B-48 インタフェース - oracle.jms.AdtMessage

Oracle JMSクラス: oracle.jms.AdtMessage



## インタフェース - oracle.jms.AQjmsQueueReceiver

図 B-49 インタフェース - oracle.jms.AQjmsQueueReceiver

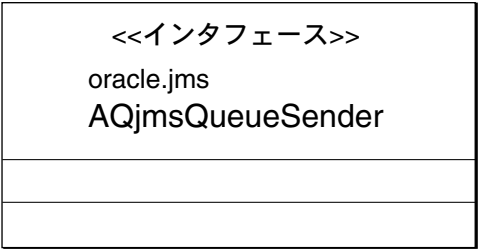
Oracle JMSクラス: oracle.jms.AQjmsQueueReceiver

<p>&lt;&lt;インタフェース&gt;&gt;</p> <p>oracle.jms AQjmsQueueReceiver</p>
<p>&lt;&lt;メソッド&gt;&gt;</p> <p>getNavigationMode() receiveNoData() receiveNoData(long) setNavigationMode(int)</p>

# インタフェース - oracle.jms.AQjmsQueueSender

図 B-50 インタフェース - oracle.jms.AQjmsQueueSender

Oracle JMSクラス: oracle.jms.AQjmsQueueSender





## インタフェース - oracle.jms.AQjmsTopicPublisher

図 B-51 インタフェース - oracle.jms.AQjmsTopicPublisher

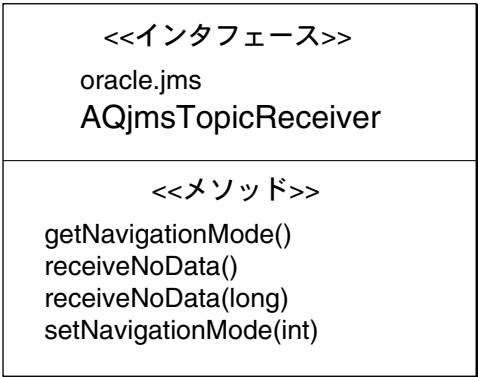
Oracle JMSクラス: oracle.jms.AQjmsTopicPublisher

<p>&lt;&lt;インタフェース&gt;&gt;</p> <p>oracle.jms AQjmsTopicPublisher</p>
<p>&lt;&lt;メソッド&gt;&gt;</p> <p>publish(Message, AQjmsAgent[]) publish(Message, - AQjmsAgent[], int, int, long) publish(Topic, Message, - AQjmsAgent[]) publish(Topic, Message, - AQjmsAgent[], int, int, long)</p>

# インタフェース - oracle.jms.TopicReceiver

図 B-52 インタフェース - oracle.jms.TopicReceiver

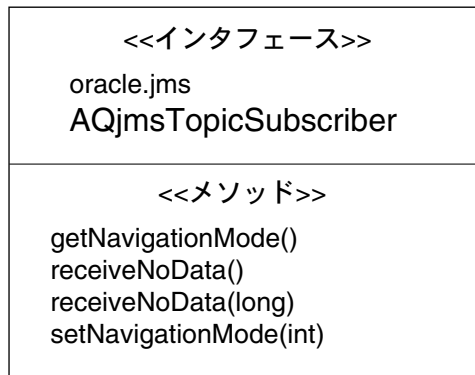
Oracle JMSクラス: oracle.jms.AQjmsTopicReceiver



## インタフェース - oracle.jms.AQjmsTopicSubscriber

図 B-53 クラス図：インタフェース - oracle.jms.AQjmsTopicSubscriber

Oracle JMSクラス: oracle.jms.AQjmsTopicSubscriber



## インタフェース - oracle.jms.AQjmsTopicReceiver

図 B-54 インタフェース - oracle.jms.

Oracle JMSクラス: oracle.jms.TopicReceiver



## クラス - oracle.jms.AQjmsAdtMessage

図 B-55 クラス - oracle.jms.AQjmsAdtMessage

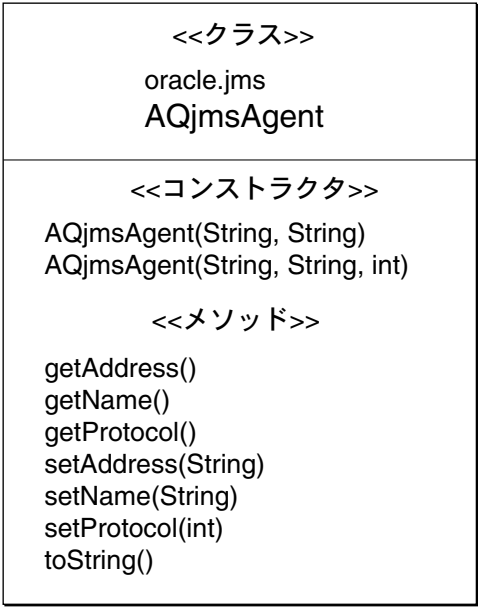
Oracle JMSクラス: oracle.jms.AQjmsAdtMessage

<p>&lt;&lt;クラス&gt;&gt;</p> <p>oracle.jms AQjmsAdtMessage</p>
<p>&lt;&lt;メソッド&gt;&gt;</p> <p>getAdtPayload() setAdtPayload(CustomDatum)</p>

## クラス - oracle.jms.AQjmsAgent

図 B-56 クラス図 : クラス - oracle.jms.AQjmsAgent

Oracle JMSクラス: oracle.jms.AQjmsAgent



# クラス - oracle.jms.AQjmsBytesMessage

図 B-57 クラス図：クラス - oracle.jms.AQjmsBytesMessage

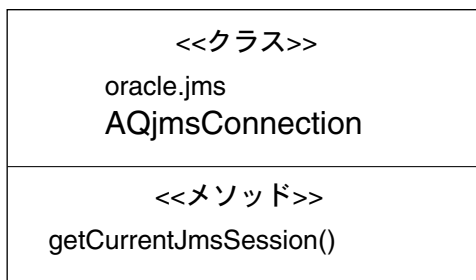
Oracle JMSクラス: oracle.jms.AQjmsBytesMessage



## クラス - oracle.jms.AQjmsConnection

図 B-58 クラス図 : oracle.jms.AQjmsConnection

Oracle JMSクラス: oracle.jms.AQjmsConnection

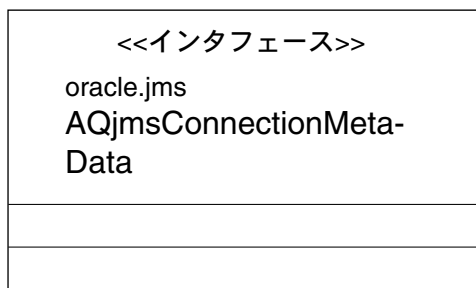




## インタフェース - oracle.jms.AQjmsConnectionMetadata

図 B-59 インタフェース - oracle.jms.AQjmsConnectionMetadata

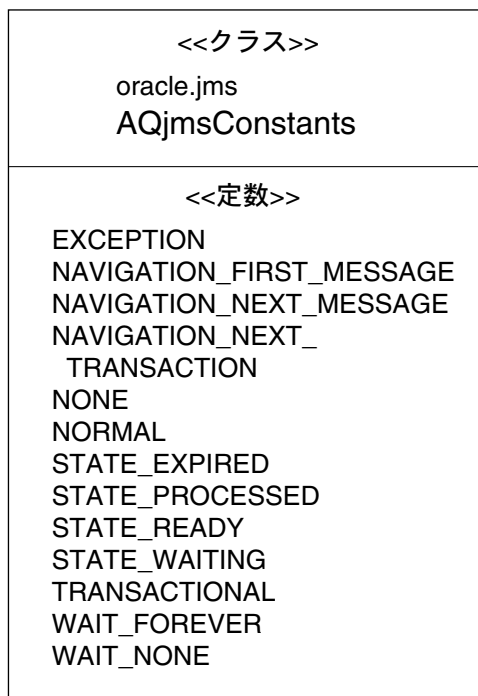
Oracle JMSクラス: oracle.jms.AQjmsConnectionMetadata



## クラス - oracle.jms.AQjmsConstants

図 B-60 クラス図: クラス - oracle.jms.AQjmsConstants

Oracle JMSクラス: oracle.jms.AQjmsConstants



# インタフェース - oracle.jms.AQjmsConsumer

図 B-61 インタフェース - oracle.jms.AQjmsConsumer

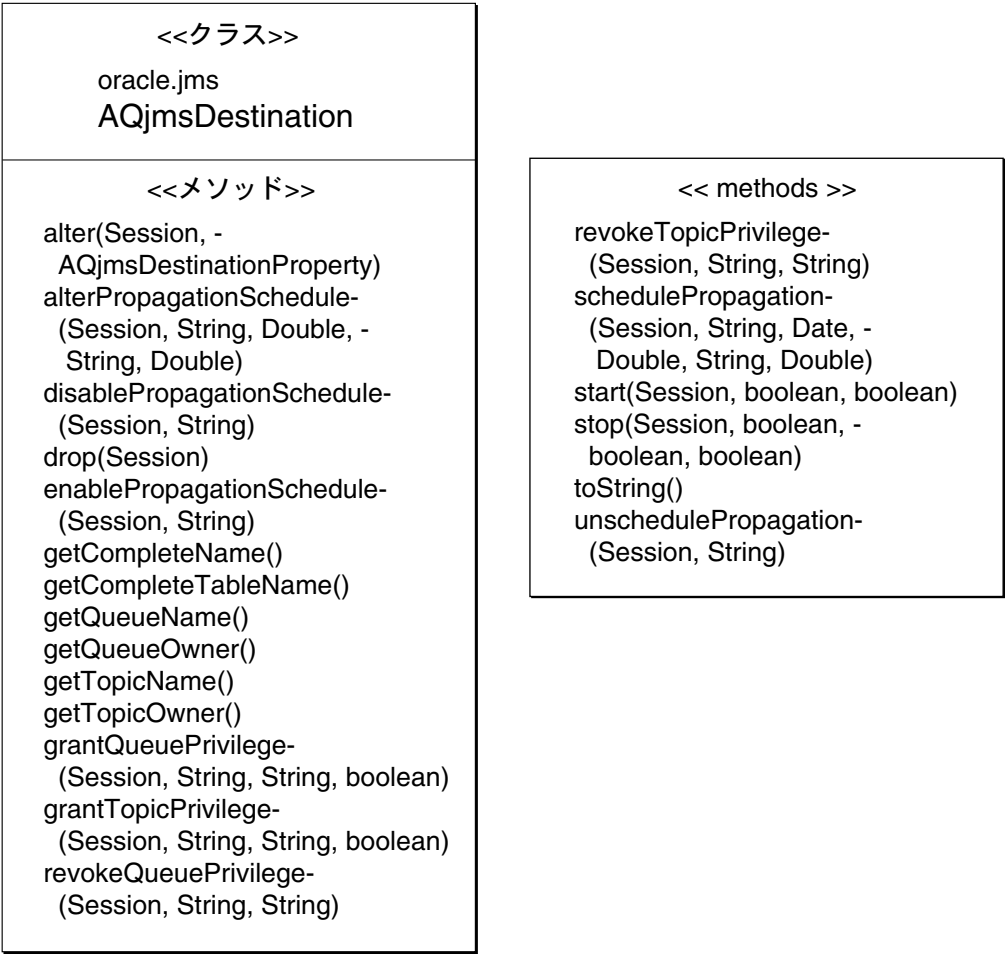
Oracle JMSクラス: oracle.jms.AQjmsConsumer

<div>&lt;&lt;インタフェース&gt;&gt;</div> <div>oracle.jms</div> <div>AQjmsConsumer</div>

# クラス - oracle.jms.AQjmsDestination

図 B-62 クラス図 : クラス - oracle.jms.AQjmsDestination

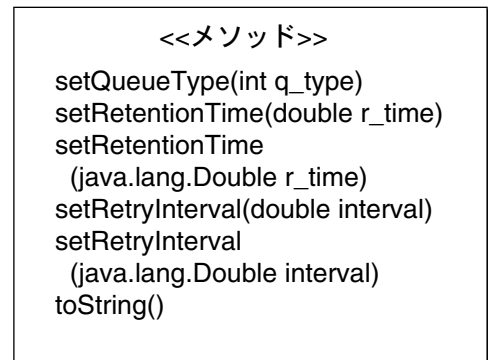
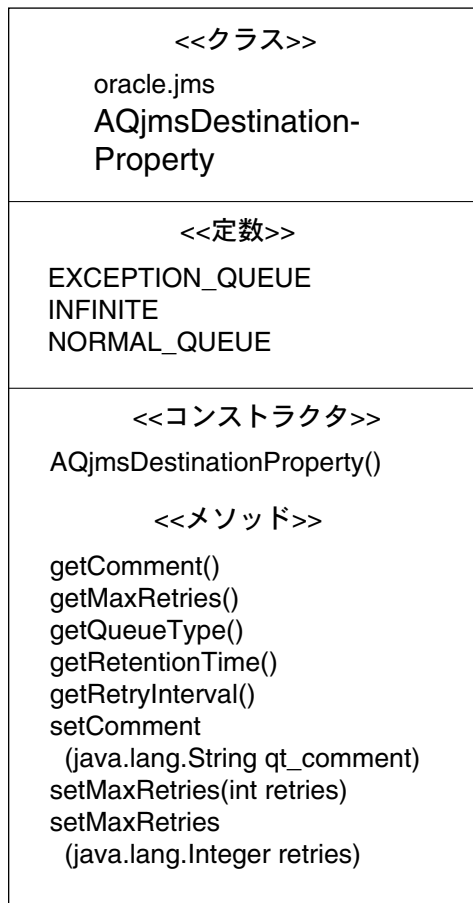
Oracle JMSクラス: oracle.jms.AQjmsDestination



## クラス - oracle.jms.AQjmsDestinationProperty

図 B-63 インタフェース - oracle.jms.AQjmsDestinationProperty

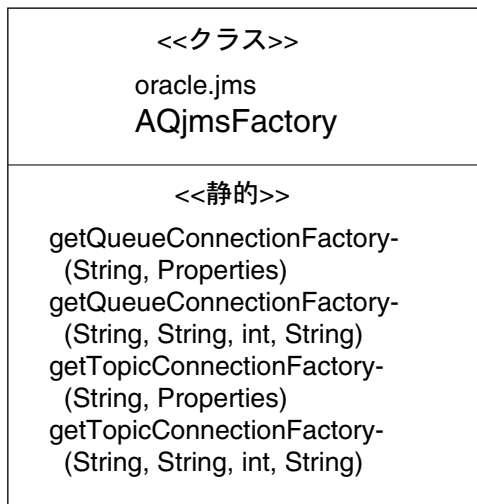
Oracle JMSクラス: oracle.jms.AQjmsDestinationProperty



## クラス - oracle.jms.AQjmsFactory

図 B-64 クラス図 : クラス - oracle.jms. AQjmsFactory

Oracle JMSクラス: oracle.jms.AQjmsFactory



# クラス - oracle.jms.AQjmsMapMessage

図 B-65 クラス図 : クラス - oracle.jms.AQjmsMapMessage

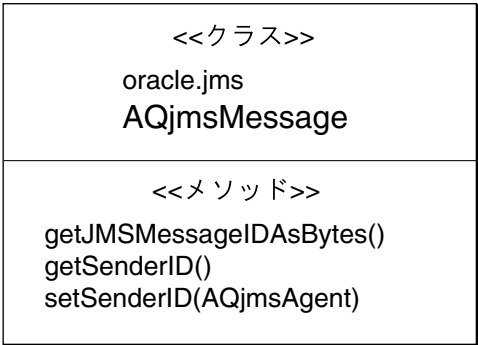
Oracle JMSクラス: oracle.jms.AQjmsMapMessage



# クラス - oracle.jms.AQjmsMessage

図 B-66 クラス図 : クラス - oracle.jms.AQjmsMessage

Oracle JMSクラス: oracle.jms.AQjmsMessage





## クラス - oracle.jms.AQjmsObjectMessage

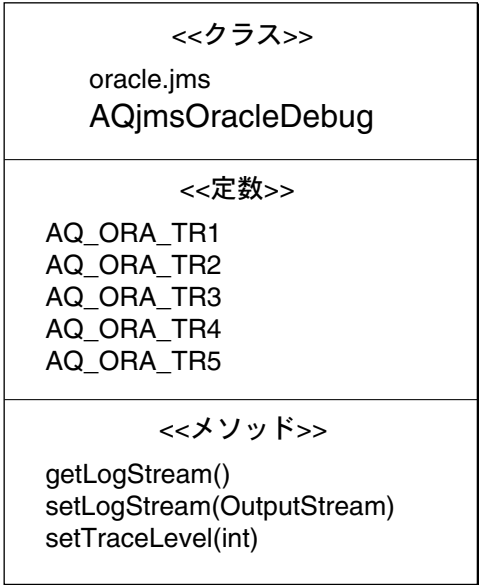
図 B-67 クラス図：クラス - oracle.jms.AQjmsObjectMessage

Oracle JMSクラス: oracle.jms.AQjmsObjectMessage



## クラス - oracle.jms.AQjmsOracleDebug

図 B-68 クラス図：クラス - oracle.jms.AQjmsOracleDebug



## クラス - oracle.jms.AQjmsProducer

図 B-69 クラス図：クラス - oracle.jms.AQjmsProducer

Oracle JMSクラス: oracle.jms.AQjmsProducer



## クラス - oracle.jms.AQjmsQueueBrowser

図 B-70 クラス図：クラス - oracle.jms.AQjmsQueueBrowser

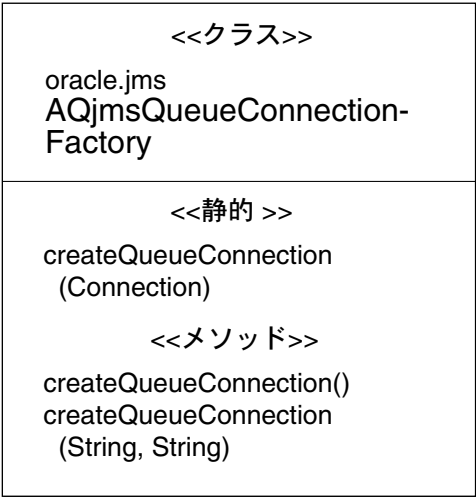
Oracle JMSクラス: oracle.jms.AQjmsQueueBrowser



# クラス - AQjmsQueueConnectionFactory

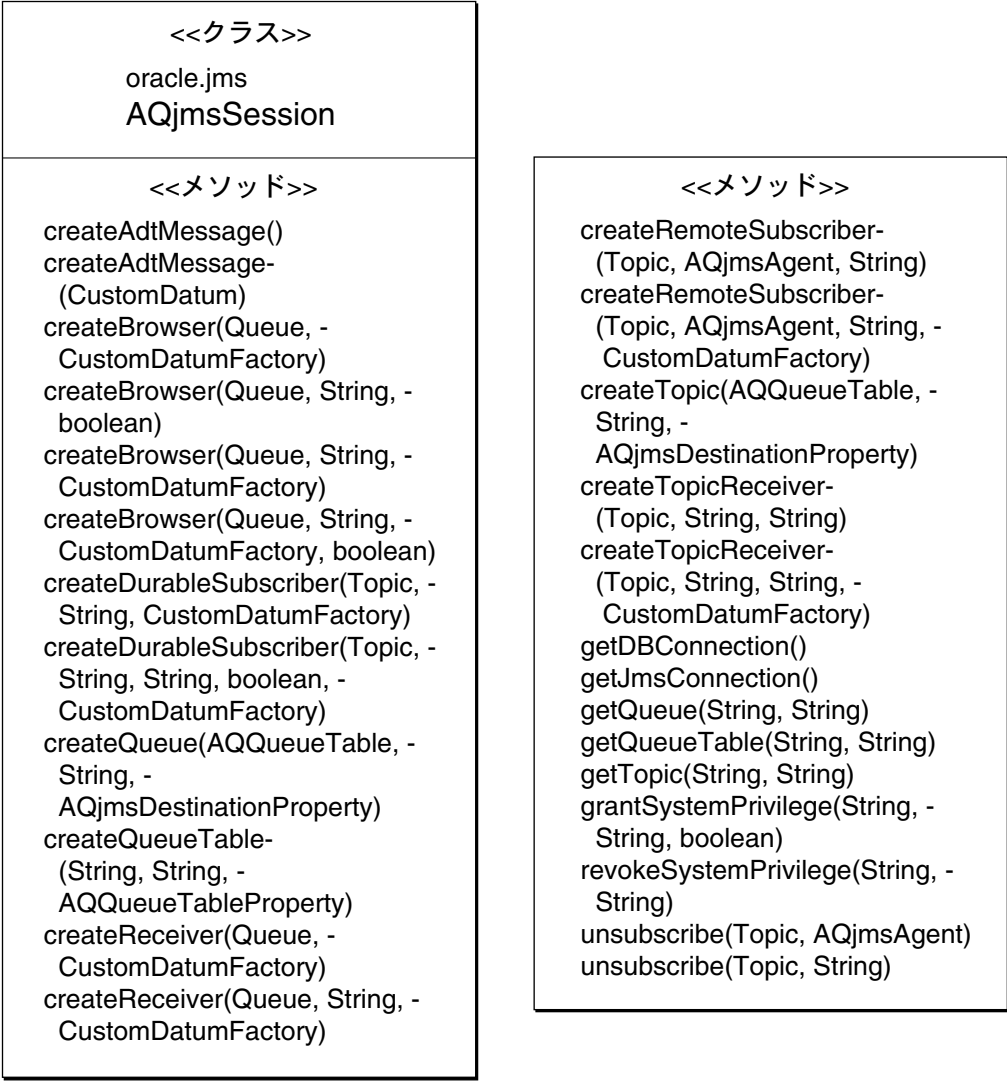
図 B-71 クラス図 : クラス - oracle.jms.AQjmsQueueConnectionFactory

Oracle JMSクラス: oracle.jms.AQjmsQueueConnectionFactory



# クラス - oracle.jms.AQjmsSession

図 B-72 クラス図 : クラス - oracle.jms.AQjmsSession



# クラス - oracle.jms.AQjmsStreamMessage

図 B-73 クラス図 : クラス - oracle.jms.AQjmsStreamMessage

Oracle JMSクラス: oracle.jms.AQjmsStreamMessage



## クラス - oracle.jms.AQjmsTextMessage

図 B-74 クラス図：クラス - oracle.jms.AQjmsTextMessage

Oracle JMSクラス: oracle.jms.AQjmsTextMessage

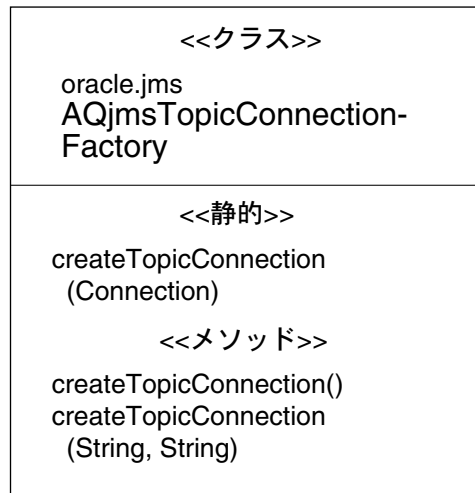




## クラス - oracle.jms.AQjmsTopicConnectionFactory

図 B-75 クラス - oracle.jms.AQjmsTopicConnectionFactory

Oracle JMSクラス: oracle.jms.AQjmsTopicConnectionFactory



## 例外 - oracle.jms.AQjmsException

図 B-76 クラス - oracle.jms.AQjmsException

Oracle JMSクラス: oracle.jms.AQjmsException

<p>&lt;&lt;例外&gt;&gt;</p> <p>oracle.jms AQjmsException</p>
<p>&lt;&lt;メソッド&gt;&gt;</p> <p>getErrorNumber()</p>

## 例外 - oracle.jms.AQjmsInvalidDestinationException

図 B-77 例外 - oracle.jms.AQjmsInvalidDestinationException

Oracle JMSクラス: oracle.jms.AQjmsInvalidDestinationException

<<例外>>
oracle.jms AQjmsInvalidDestination- Exception

## 例外 - oracle.jms.AQjmsInvalidSelectorException

図 B-78 例外 - oracle.jms.AQjmsInvalidSelectorException

Oracle JMSクラス: oracle.jms.AQjmsInvalidSelectorException

<<例外>> oracle.jms AQjmsInvalidSelector- Exception

# 例外 - oracle.jms.AQjmsMessageEOFException

図 B-79 例外 - oracle.jms.AQjmsMessageEOFException

Oracle JMSクラス: oracle.jms.AQjmsMessageEOFException

<div>&lt;&lt;例外&gt;&gt;</div> <div>oracle.jms AQjmsMessageEOF- Exception</div>

## 例外 - oracle.jms.AQjmsMessageFormatException

図 B-80 例外 - oracle.jms.AQjmsMessageFormatException

Oracle JMSクラス: oracle.jms.AQjmsMessageFormatException

<div>&lt;&lt;例外&gt;&gt;</div> <div>oracle.jms AQjmsMessageFormat- Exception</div>

## 例外 - oracle.jms.AQjmsMessageNotReadableException

図 B-81 例外 - oracle.jms.AQjmsMessageNotReadableException

Oracle JMSクラス: oracle.jms.AQjmsMessageNotReadableException

<<例外>> oracle.jms AQjmsMessageNot- ReadableException

## 例外 - oracle.jms.AQjmsMesssageNotWriteableException

図 B-82 例外 - oracle.jms.AQjmsMessageNotWriteableException

Oracle JMSクラス: oracle.jms.AQjmsMessageNotWriteableException

<<例外>> oracle.jms AQjmsMessageNot- WriteableException



# インタフェース - oracle.AQ.AQQueueTable

図 B-83 インタフェース - oracle.AQ.AQQueueTable

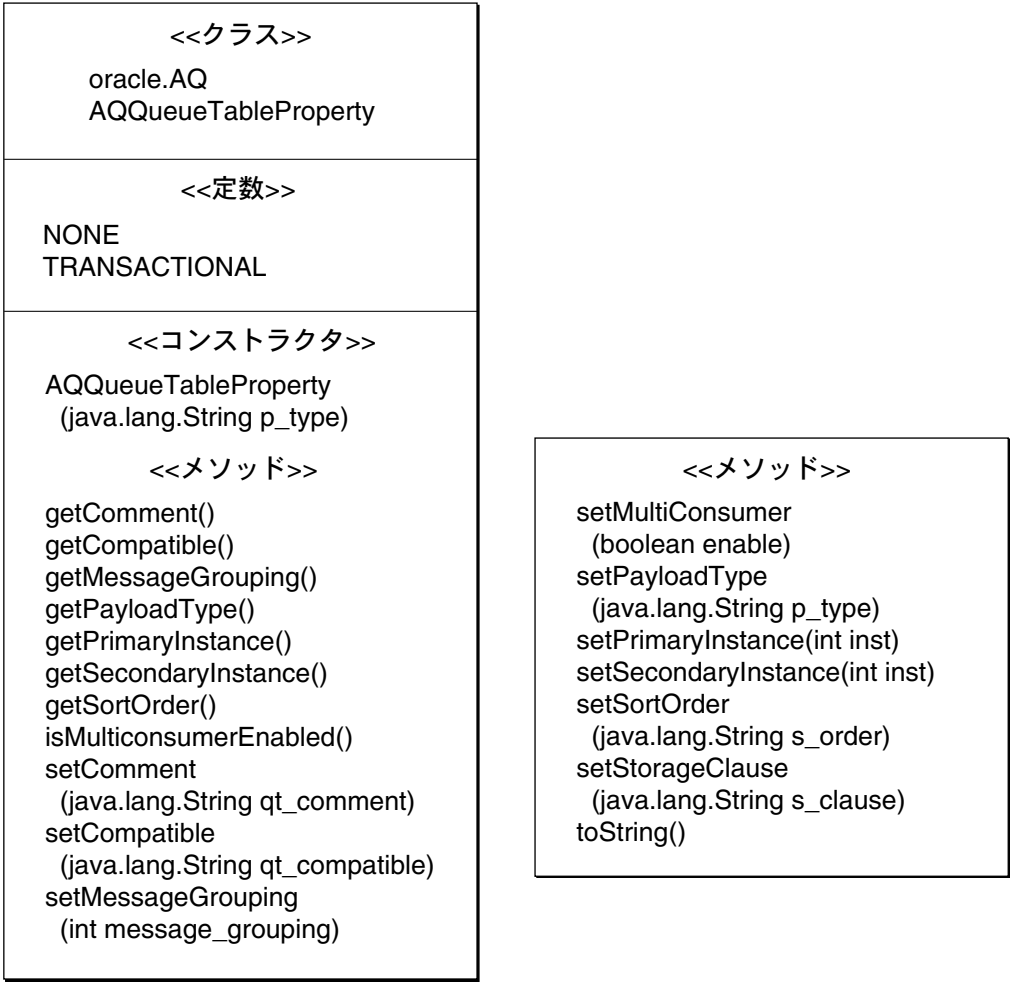
Oracle JMSクラス: oracle.AQ.AQQueueTable

<div>&lt;&lt;インタフェース&gt;&gt;</div> <div>oracle.AQ AQQueueTable</div>
<div>&lt;&lt;メソッド&gt;&gt;</div> <div>alter(java.lang.String comment) alter(java.lang.String com- ment, int primary_in- stance, int secondary_instance) drop(boolean force) getName() getOwner() getProperty()</div>

# クラス - oracle.AQ.AQQueueTableProperty

図 B-84 クラス図：クラス - oracle.AQ.AQQueueTableProperty

Oracle JMSクラス: oracle.AQ.AQQueueTableProperty



---

## BooksOnLine 用スクリプト

この付録には、次のスクリプトが掲載されています。

- `tkaqdoca.sql`: ユーザー、オブジェクト、キュー・テーブル、キューおよびサブスクリイ  
パ作成用のスクリプト
- `tkaqdocd.sql`: 管理インタフェースおよび操作インタフェースの例
- `tkaqdoce.sql`: 操作例
- `tkaqdocp.sql`: 操作インタフェースの例
- `tkaqdocc.sql`: クリーンアップ・スクリプト

## tkaqdoca.sql: ユーザー、オブジェクト、キュー・テーブル、キューおよびサブスクリバ作成用のスクリプト

```
Rem $Header: tkaqdoca.sql 26-jan-99.17:50:37 aquser1 Exp $
Rem
Rem tkaqdoca.sql
Rem
Rem Copyright (c) Oracle Corporation 1998, 1999. All Rights Reserved.
Rem
Rem      NAME
Rem      tkaqdoca.sql - TKAQ DOCumentation Admin examples file

Rem Set up a queue admin account and individual accounts for each application
Rem
connect system/manager
set serveroutput on;
set echo on;

Rem Create a common admin account for all BooksOnLine applications
Rem
create user BOLADM identified by BOLADM;
grant connect, resource, aq_administrator_role to BOLADM;
grant execute on dbms_aq to BOLADM;
grant execute on dbms_aqadm to BOLADM;
execute dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','BOLADM',FALSE);
execute dbms_aqadm.grant_system_privilege('DEQUEUE_ANY','BOLADM',FALSE);

Rem Create the application schemas and grant appropriate permission
Rem to all schemas

Rem Create an account for Order Entry
create user OE identified by OE;
grant connect, resource to OE;
grant execute on dbms_aq to OE;
grant execute on dbms_aqadm to OE;

Rem Create an account for WR Shipping
create user WS identified by WS;
grant connect, resource to WS;
grant execute on dbms_aq to WS;
grant execute on dbms_aqadm to WS;

Rem Create an account for ER Shipping
create user ES identified by ES;
grant connect, resource to ES;
```

```
grant execute on dbms_aq to ES;
grant execute on dbms_aqadm to ES;

Rem Create an account for Overseas Shipping
create user OS identified by OS;
grant connect, resource to OS;
grant execute on dbms_aq to OS;
grant execute on dbms_aqadm to OS;

Rem Create an account for Customer Billing
Rem Customer Billing, for security reason, has an admin schema that
Rem hosts all the queue tables and an application schema from where
Rem the application runs.
create user CBADM identified by CBADM;
grant connect, resource to CBADM;
grant execute on dbms_aq to CBADM;
grant execute on dbms_aqadm to CBADM;

create user CB identified by CB;
grant connect, resource to CB;
grant execute on dbms_aq to CB;
grant execute on dbms_aqadm to CB;

Rem Create an account for Customer Service
create user CS identified by CS;
grant connect, resource to CS;
grant execute on dbms_aq to CS;
grant execute on dbms_aqadm to CS;

Rem All object types are created in the administrator schema.
Rem All application schemas that host any propagation source
Rem queues are given the ENQUEUE_ANY system level privilege
Rem allowing the application schemas to enqueue to the destination
Rem queue.
Rem
connect BOLADM/BOLADM;

Rem Create objects

create or replace type customer_typ as object (
    custno          number,
    name            varchar2(100),
    street          varchar2(100),
    city            varchar2(30),
```

```

        state          varchar2(2),
        zip            number,
        country        varchar2(100));
/

create or replace type book_type as object (
    title             varchar2(100),
    authors           varchar2(100),
    ISBN              number,
    price             number);
/

create or replace type orderitem_type as object (
    quantity          number,
    item              book_type,
    subtotal           number);
/

create or replace type orderitemlist_vartyp as varray (20) of orderitem_type;
/

create or replace type order_type as object (
    orderno            number,
    status             varchar2(30),
    ordertype          varchar2(30),
    orderregion        varchar2(30),
    customer           customer_type,
    paymentmethod       varchar2(30),
    items              orderitemlist_vartyp,
    total              number);
/

grant execute on order_type to OE;
grant execute on orderitemlist_vartyp to OE;
grant execute on orderitem_type to OE;
grant execute on book_type to OE;
grant execute on customer_type to OE;
execute dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','OE',FALSE);

grant execute on order_type to WS;
grant execute on orderitemlist_vartyp to WS;
grant execute on orderitem_type to WS;
grant execute on book_type to WS;
grant execute on customer_type to WS;
execute dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','WS',FALSE);

```

```
grant execute on order_typ to ES;
grant execute on orderitemlist_vartyp to ES;
grant execute on orderitem_typ to ES;
grant execute on book_typ to ES;
grant execute on customer_typ to ES;
execute dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','ES',FALSE);

grant execute on order_typ to OS;
grant execute on orderitemlist_vartyp to OS;
grant execute on orderitem_typ to OS;
grant execute on book_typ to OS;
grant execute on customer_typ to OS;
execute dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','OS',FALSE);

grant execute on order_typ to CBADM;
grant execute on orderitemlist_vartyp to CBADM;
grant execute on orderitem_typ to CBADM;
grant execute on book_typ to CBADM;
grant execute on customer_typ to CBADM;

grant execute on order_typ to CB;
grant execute on orderitemlist_vartyp to CB;
grant execute on orderitem_typ to CB;
grant execute on book_typ to CB;
grant execute on customer_typ to CB;

grant execute on order_typ to CS;
grant execute on orderitemlist_vartyp to CS;
grant execute on orderitem_typ to CS;
grant execute on book_typ to CS;
grant execute on customer_typ to CS;

Rem Create queue tables, queues for OE
Rem
connect OE/OE;
begin
dbms_aqadm.create_queue_table(
    queue_table => 'OE_orders_sqtab',
    comment => 'Order Entry Single Consumer Orders queue table',
    queue_payload_type => 'BOLADM.order_typ',
    message_grouping => DBMS_AQADM.TRANSACTIONAL,
    compatible => '8.1',
    primary_instance => 1,
    secondary_instance => 2);
```

```
end;
/

Rem Create a priority queue table for OE
begin
dbms_aqadm.create_queue_table(
    queue_table => 'OE_orders_pr_mqtab',
    sort_list => 'priority,enq_time',
    comment => 'Order Entry Priority MultiConsumer Orders queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1',
    primary_instance => 2,
    secondary_instance => 1);
end;
/

begin
dbms_aqadm.create_queue (
    queue_name          => 'OE_neworders_que',
    queue_table         => 'OE_orders_sqtab');
end;
/

begin
dbms_aqadm.create_queue (
    queue_name          => 'OE_bookedorders_que',
    queue_table         => 'OE_orders_pr_mqtab');
end;
/

Rem Orders in OE_bookedorders_que are being propagated to WS_bookedorders_que,
Rem ES_bookedorders_que and OS_bookedorders_que according to the region
Rem the books are shipped to. At the time an order is placed, the customer
Rem can request Fed-ex shipping (priority 1), priority air shipping (priority
Rem 2) and ground shipping (priority 3). An priority queue is created in
Rem each region, the shipping applications will dequeue from these priority
Rem queues according to the orders' shipping priorities, processes the orders
Rem and enqueue the processed orders into
Rem the shipped_orders queues or the back_orders queues. Both the shipped_
Rem orders queues and the back_orders queues are FIFO queues. However,
Rem orders put into the back_orders_queues are enqueued with delay time
Rem set to 1 day, so that each order in the back_order_queues is processed
Rem only once a day until the shipment is filled.
```



```
Rem Create queue tables, queues for WS Shipping
connect WS/WS;

Rem Create a priority queue table for WS shipping
begin
dbms_aqadm.create_queue_table(
    queue_table => 'WS_orders_pr_mqtab',
    sort_list => 'priority,enq_time',
    comment => 'West Shipping Priority MultiConsumer Orders queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');
end;
/

Rem Create a FIFO queue tables for WS shipping
begin
dbms_aqadm.create_queue_table(
    queue_table => 'WS_orders_mqtab',
    comment => 'West Shipping Multi Consumer Orders queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');
end;
/

Rem Booked orders are stored in the priority queue table
begin
dbms_aqadm.create_queue (
    queue_name          => 'WS_bookedorders_que',
    queue_table         => 'WS_orders_pr_mqtab');
end;
/

Rem Shipped orders and back orders are stored in the FIFO queue table
begin
dbms_aqadm.create_queue (
    queue_name          => 'WS_shippedorders_que',
    queue_table         => 'WS_orders_mqtab');
end;
/

begin
dbms_aqadm.create_queue (
```

```

        queue_name          => 'WS_backorders_que',
        queue_table         => 'WS_orders_mqtab');
end;
/

Rem
Rem In order to test history, set retention to 1 DAY for the queues
Rem in WS

begin
dbms_aqadm.alter_queue(
    queue_name => 'WS_bookedorders_que',
    retention_time => 86400);
end;
/

begin
dbms_aqadm.alter_queue(
    queue_name => 'WS_shippedorders_que',
    retention_time => 86400);
end;
/

begin
dbms_aqadm.alter_queue(
    queue_name => 'WS_backorders_que',
    retention_time => 86400);
end;
/

Rem Create queue tables, queues for ES Shipping
connect ES/ES;

Rem Create a priority queue table for ES shipping
begin
dbms_aqadm.create_queue_table(
    queue_table => 'ES_orders_mqtab',
    comment => 'East Shipping Multi Consumer Orders queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');
end;
/

```

```
Rem Create a FIFO queue tables for ES shipping
begin
dbms_aqadm.create_queue_table(
    queue_table => 'ES_orders_pr_mqtab',
    sort_list => 'priority,enq_time',
    comment => 'East Shipping Priority Multi Consumer Orders queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');
end;
/

Rem Booked orders are stored in the priority queue table
begin
dbms_aqadm.create_queue (
    queue_name          => 'ES_bookedorders_que',
    queue_table         => 'ES_orders_pr_mqtab');
end;
/

Rem Shipped orders and back orders are stored in the FIFO queue table
begin
dbms_aqadm.create_queue (
    queue_name          => 'ES_shippedorders_que',
    queue_table         => 'ES_orders_mqtab');
end;
/

begin
dbms_aqadm.create_queue (
    queue_name          => 'ES_backorders_que',
    queue_table         => 'ES_orders_mqtab');
end;
/

Rem Create queue tables, queues for Overseas Shipping
connect OS/OS;

Rem Create a priority queue table for OS shipping
begin
dbms_aqadm.create_queue_table(
    queue_table => 'OS_orders_pr_mqtab',
    sort_list => 'priority,enq_time',
    comment => 'Overseas Shipping Priority MultiConsumer Orders queue table',
```

```
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');
end;
/

Rem Create a FIFO queue tables for OS shipping
begin
dbms_aqadm.create_queue_table(
    queue_table => 'OS_orders_mqtab',
    comment => 'Overseas Shipping Multi Consumer Orders queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');
end;
/

Rem Booked orders are stored in the priority queue table
begin
dbms_aqadm.create_queue (
    queue_name          => 'OS_bookedorders_que',
    queue_table         => 'OS_orders_pr_mqtab');
end;
/

Rem Shipped orders and back orders are stored in the FIFO queue table
begin
dbms_aqadm.create_queue (
    queue_name          => 'OS_shippedorders_que',
    queue_table         => 'OS_orders_mqtab');
end;
/

begin
dbms_aqadm.create_queue (
    queue_name          => 'OS_backorders_que',
    queue_table         => 'OS_orders_mqtab');
end;
/

Rem Create queue tables, queues for Customer Billing
connect CBADM/CBADM;
begin
```

```
dbms_aqadm.create_queue_table(
    queue_table => 'CBADM_orders_sqtan',
    comment => 'Customer Billing Single Consumer Orders queue table',
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');

dbms_aqadm.create_queue_table(
    queue_table => 'CBADM_orders_mqtan',
    comment => 'Customer Billing Multi Consumer Service queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');

dbms_aqadm.create_queue (
    queue_name          => 'CBADM_shippedorders_que',
    queue_table         => 'CBADM_orders_sqtan');

end;
/

Rem Grant dequeue privilege on the shipped orders queue to the Customer Billing
Rem application. The CB application retrieves shipped orders (not billed yet)
Rem from the shipped orders queue.
execute dbms_aqadm.grant_queue_privilege('DEQUEUE', 'CBADM_shippedorders_que', 'CB',
FALSE);

begin
dbms_aqadm.create_queue (
    queue_name          => 'CBADM_billedorders_que',
    queue_table         => 'CBADM_orders_mqtan');

end;
/

Rem Grant enqueue privilege on the billed orders queue to Customer Billing
Rem application. The CB application is allowed to put billed orders into
Rem this queue.
execute dbms_aqadm.grant_queue_privilege('ENQUEUE', 'CBADM_billedorders_que', 'CB',
FALSE);

Rem Customer support tracks the state of the customer request in the system
Rem
Rem At any point, customer request can be in one of the following states
Rem A. BOOKED B. SHIPPED C. BACKED D. BILLED
Rem Given the order number the customer support will return the state
```

```

Rem the order is in. This state is maintained in the order_status_table

connect CS/CS;

CREATE TABLE Order_Status_Table(customer_order      boladm.order_typ,
                                status                varchar2(30));

Rem Create queue tables, queues for Customer Service

begin
dbms_aqadm.create_queue_table(
    queue_table => 'CS_order_status_qt',
    comment => 'Customer Status multi consumer queue table',
    multiple_consumers => TRUE,
    queue_payload_type => 'BOLADM.order_typ',
    compatible => '8.1');

dbms_aqadm.create_queue (
    queue_name          => 'CS_bookedorders_que',
    queue_table         => 'CS_order_status_qt');

dbms_aqadm.create_queue (
    queue_name          => 'CS_backorders_que',
    queue_table         => 'CS_order_status_qt');

dbms_aqadm.create_queue (
    queue_name          => 'CS_shippedorders_que',
    queue_table         => 'CS_order_status_qt');

dbms_aqadm.create_queue (
    queue_name          => 'CS_billedorders_que',
    queue_table         => 'CS_order_status_qt');

end;
/

Rem Create the Subscribers for OE queues
Rem Add the Subscribers for the OE booked_orders queue

connect OE/OE;

Rem Add a rule-based subscriber for West Shipping
Rem West Shipping handles Western region US orders
Rem Rush Western region orders are handled by East Shipping
declare

```

```
subscriber      aq$_agent;
begin
  subscriber := aq$_agent('West_Shipping', 'WS.WS_bookedorders_que', null);
  dbms_aqadm.add_subscriber(queue_name => 'OE.OE_bookedorders_que',
                             subscriber => subscriber,
                             rule       => 'tab.user_data.orderregion = ''WESTERN''
AND tab.user_data.ordertype != ''RUSH''');
end;
/

Rem Add a rule-based subscriber for East Shipping
Rem East shipping handles all Eastern region orders
Rem East shipping also handles all US rush orders
declare
  subscriber      aq$_agent;
begin
  subscriber := aq$_agent('East_Shipping', 'ES.ES_bookedorders_que', null);
  dbms_aqadm.add_subscriber(queue_name => 'OE.OE_bookedorders_que',
                             subscriber => subscriber,
                             rule       => 'tab.user_data.orderregion = ''EASTERN''
OR (tab.user_data.ordertype = ''RUSH'' AND tab.user_data.customer.country = ''USA''
)');
end;
/

Rem Add a rule-based subscriber for Overseas Shipping
Rem Intl Shipping handles all non-US orders
declare
  subscriber      aq$_agent;
begin
  subscriber := aq$_agent('Overseas_Shipping', 'OS.OS_bookedorders_que', null);
  dbms_aqadm.add_subscriber(queue_name => 'OE.OE_bookedorders_que',
                             subscriber => subscriber,
                             rule       => 'tab.user_data.orderregion =
''INTERNATIONAL''');
end;
/

Rem Add the Customer Service order queues as a subscribers to the
Rem corresponding queues in OrderEntry, Shipping and Billing

declare
  subscriber      aq$_agent;
begin
  /* Subscribe to the booked orders queue */
```

```
subscriber := aq$_agent('BOOKED_ORDER', 'CS.CS_bookedorders_que', null);
dbms_aqadm.add_subscriber(queue_name => 'OE.OE_bookedorders_que',
                           subscriber => subscriber);

end;
/

connect WS/WS;

declare
  subscriber      aq$_agent;
begin
  /* Subscribe to the WS back orders queue */
  subscriber := aq$_agent('BACK_ORDER', 'CS.CS_backorders_que', null);
  dbms_aqadm.add_subscriber(queue_name => 'WS.WS_backorders_que',
                           subscriber => subscriber);

end;
/

declare
  subscriber      aq$_agent;
begin
  /* Subscribe to the WS shipped orders queue */
  subscriber := aq$_agent('SHIPPED_ORDER', 'CS.CS_shippedorders_que', null);
  dbms_aqadm.add_subscriber(queue_name => 'WS.WS_shippedorders_que',
                           subscriber => subscriber);

end;
/

connect CBADM/CBADM;
declare
  subscriber      aq$_agent;
begin
  /* Subscribe to the BILLING billed orders queue */
  subscriber := aq$_agent('BILLED_ORDER', 'CS.CS_billedorders_que', null);
  dbms_aqadm.add_subscriber(queue_name => 'CBADM.CBADM_billedorders_que',
                           subscriber => subscriber);

end;
/

Rem
Rem BOLADM will Start all the queues
Rem
```



```
connect BOLADM/BOLADM
execute dbms_aqadm.start_queue(queue_name => 'OE.OE_neworders_que');
execute dbms_aqadm.start_queue(queue_name => 'OE.OE_bookedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'WS.WS_bookedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'WS.WS_shippedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'WS.WS_backorders_que');
execute dbms_aqadm.start_queue(queue_name => 'ES.ES_bookedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'ES.ES_shippedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'ES.ES_backorders_que');
execute dbms_aqadm.start_queue(queue_name => 'OS.OS_bookedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'OS.OS_shippedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'OS.OS_backorders_que');
execute dbms_aqadm.start_queue(queue_name => 'CBADM.CBADM_shippedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'CBADM.CBADM_billedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'CS.CS_bookedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'CS.CS_backorders_que');
execute dbms_aqadm.start_queue(queue_name => 'CS.CS_shippedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'CS.CS_billedorders_que');

connect system/manager

Rem
Rem Start job_queue_processes to handle AQ propagation
Rem

alter system set job_queue_processes=4;
```

## tkagdocd.sql: 管理インタフェースおよび操作インタフェースの例

```
Rem
Rem $Header: tkagdocd.sql 26-jan-99.17:51:23 aquser1 Exp $
Rem
Rem tkagdocd.sql
Rem
Rem Copyright (c) Oracle Corporation 1998, 1999. All Rights Reserved.
Rem
Rem NAME
Rem     tkagdocd.sql - <one-line expansion of the name>
Rem
Rem DESCRIPTION
Rem     <short description of component this file declares/defines>
Rem
Rem NOTES
Rem     <other useful comments, qualifications, etc.>
Rem
Rem
Rem
Rem
Rem Schedule propagation for the shipping, billing, order entry queues
Rem
Rem
Rem connect OE/OE;
Rem
Rem execute dbms_aqadm.schedule_propagation(queue_name => 'OE.OE_bookedorders_que');
Rem
Rem connect WS/WS;
Rem execute dbms_aqadm.schedule_propagation(queue_name => 'WS.WS_backorders_que');
Rem execute dbms_aqadm.schedule_propagation(queue_name => 'WS.WS_shippedorders_que');
Rem
Rem connect CBADM/CBADM;
Rem execute dbms_aqadm.schedule_propagation(queue_name => 'CBADM.CBADM_billedorders_
Rem que');
Rem
Rem Customer service application
Rem
Rem This application monitors the status queue for messages and updates
Rem the Order Status table.
```

```
connect CS/CS

Rem
Rem Dequeue messages from the 'queue' for 'consumer'

CREATE OR REPLACE PROCEDURE DEQUEUE_MESSAGE(
                                queue      IN   VARCHAR2,
                                consumer   IN   VARCHAR2,
                                message    OUT  BOLADM.order_typ)
IS
    dopt          dbms_aq.dequeue_options_t;
    mprop         dbms_aq.message_properties_t;
    deq_msgid     raw(16);
BEGIN
    dopt.dequeue_mode := dbms_aq.REMOVE;
    dopt.navigation := dbms_aq.FIRST_MESSAGE;
    dopt.consumer_name := consumer;

    dbms_aq.dequeue(
        queue_name => queue,
        dequeue_options => dopt,
        message_properties => mprop,
        payload => message,
        msgid => deq_msgid);

    commit;
END;
/

Rem
Rem Updates the status of the order in the status table
Rem

CREATE OR REPLACE PROCEDURE update_status(
                                new_status  IN VARCHAR2,
                                order_msg   IN BOLADM.ORDER_TYP)
IS
    old_status    VARCHAR2(30);
    dummy         NUMBER;
BEGIN
    BEGIN
```

```
/* query old status from the table */
SELECT st.status INTO old_status from order_status_table st
      where st.customer_order.orderno = order_msg.orderno;

/* Status can be 'BOOKED_ORDER', 'SHIPPED_ORDER', 'BACK_ORDER'
 *           and 'BILLED_ORDER'
 */

IF new_status = 'SHIPPED_ORDER' THEN
  IF old_status = 'BILLED_ORDER' THEN
    return;          /* message about a previous state */
  END IF;
ELSIF new_status = 'BACK_ORDER' THEN
  IF old_status = 'SHIPPED_ORDER' OR old_status = 'BILLED_ORDER' THEN
    return;          /* message about a previous state */
  END IF;
END IF;

/* update the order status */
UPDATE order_status_table st
      SET st.customer_order = order_msg, st.status = new_status
      where st.customer_order.orderno = order_msg.orderno;

COMMIT;

EXCEPTION
WHEN OTHERS THEN      /* change to no data found */
  /* first update for the order */
  INSERT INTO order_status_table(customer_order, status)
  VALUES (order_msg, new_status);
  COMMIT;

END;
END;
/

Rem
Rem  Monitors the customer service queues for 'time' seconds
Rem

CREATE OR REPLACE PROCEDURE MONITOR_STATUS_QUEUE(time IN NUMBER)
IS
  agent_w_message  aq$_agent;
  agent_list       dbms_aq.agent_list_t;
```

```

wait_time          INTEGER := 120;
no_message          EXCEPTION;
pragma EXCEPTION_INIT(no_message, -25254);
order_msg           boladm.order_typ;
new_status          VARCHAR2(30);
monitor             BOOLEAN := TRUE;
begin_time          number;
end_time            number;
BEGIN

begin_time := dbms_utility.get_time;
WHILE (monitor)
LOOP
BEGIN
agent_list(1) := aq$_agent('BILLED_ORDER', 'CS_billedorders_que', NULL);
agent_list(2) := aq$_agent('SHIPPED_ORDER', 'CS_shippedorders_que', NULL);
agent_list(3) := aq$_agent('BACK_ORDER', 'CS_backorders_que', NULL);
agent_list(4) := aq$_agent('Booked_ORDER', 'CS_bookedorders_que', NULL);

/* wait for order status messages */
dbms_aq.listen(agent_list, wait_time, agent_w_message);

dbms_output.put_line('Agent' || agent_w_message.name || ' Address ' || agent_w_
message.address);
/* dequeue the message from the queue */
dequeue_message(agent_w_message.address, agent_w_message.name, order_msg);

/* update the status of the order depending on the type of the message
 * the name of the agent contains the new state
 */
update_status(agent_w_message.name, order_msg);

/* exit if we have been working long enough */
end_time := dbms_utility.get_time;
IF (end_time - begin_time > time) THEN
EXIT;
END IF;

EXCEPTION
WHEN no_message THEN
dbms_output.put_line('No messages in the past 2 minutes');
end_time := dbms_utility.get_time;
/* exit if we have done enough work */
IF (end_time - begin_time > time) THEN
EXIT;

```

```
        END IF;
    END;

    END LOOP;
END;
/

Rem
Rem  History queries
Rem

Rem
Rem  Average processing time for messages in western shipping:
Rem  Difference between the ship- time and book-time for the order
Rem
Rem  NOTE: we assume that order id is the correlation identifier
Rem          Only processed messages are considered.

Connect WS/WS

SELECT SUM(SO.eng_time - BO.eng_time) / count (*) AVG_PRCS_TIME
FROM WS.AQ$WS_orders_pr_mqtab BO , WS.AQ$WS_orders_mqtab SO
WHERE SO.msg_state = 'PROCESSED' and BO.msg_state = 'PROCESSED'
AND SO.corr_id = BO.corr_id and SO.queue = 'WS_shippedorders_que';

Rem
Rem  Average backed up time (again only processed messages are considered
Rem

SELECT SUM(BACK.deq_time - BACK.eng_time)/count (*) AVG_BACK_TIME
FROM WS.AQ$WS_orders_mqtab BACK
WHERE BACK.msg_state = 'PROCESSED' and BACK.queue = 'WS_backorders_que';
```

## tkaqdoce.sql: 操作例

```

Rem
Rem $Header: tkaqdoce.sql 26-jan-99.17:51:28 aquser1 Exp $
Rem
Rem tkaqdocl.sql
Rem
Rem Copyright (c) Oracle Corporation 1998, 1999. All Rights Reserved.
Rem

set echo on

Rem =====
Rem      Demonstrate enqueueing a backorder with delay time set
Rem      to 1 day. This will guarantee that each backorder will
Rem      be processed only once a day until the order is filled.
Rem =====

Rem Create a package that enqueue with delay set to one day
connect BOLADM/BOLADM
create or replace procedure requeue_unfilled_order(sale_region varchar2,
                                                    backorder order_typ)
as
    back_order_queue_name    varchar2(62);
    enqopt                   dbms_aq.enqueue_options_t;
    msgprop                   dbms_aq.message_properties_t;
    enq_msgid                 raw(16);
begin
    -- Choose a back order queue based the the region
    IF sale_region = 'WEST' THEN
        back_order_queue_name := 'WS.WS_backorders_que';
    ELSIF sale_region = 'EAST' THEN
        back_order_queue_name := 'ES.ES_backorders_que';
    ELSE
        back_order_queue_name := 'OS.OS_backorders_que';
    END IF;

    -- Enqueue the order with delay time set to 1 day
    msgprop.delay := 60*60*24;
    dbms_aq.enqueue(back_order_queue_name, enqopt, msgprop,
                    backorder, enq_msgid);
end;
```

## tkaqdocp.sql: 操作インタフェースの例

```

Rem
Rem $Header: tkaqdocp.sql 26-jan-99.17:50:54 aquser1 Exp $
Rem
Rem tkaqdocp.sql
Rem
Rem Copyright (c) Oracle Corporation 1998, 1999. All Rights Reserved.
Rem
Rem NAME
Rem tkaqdocp.sql - <one-line expansion of the name>
Rem

set echo on;

Rem =====
Rem                      Illustrating Support for OPS
Rem =====

Rem Login into OE account
connect OE/OE;
set serveroutput on;

Rem check instance affinity of OE queue tables from AQ administrative view

select queue_table, primary_instance, secondary_instance, owner_instance
from user_queue_tables;

Rem alter instance affinity of OE queue tables

begin
dms_aqadm.alter_queue_table(
    queue_table => 'OE.OE_orders_sqt'ab',
    primary_instance => 2,
    secondary_instance => 1);
end;
/

begin
dms_aqadm.alter_queue_table(
    queue_table => 'OE.OE_orders_pr_mqt'ab',
    primary_instance => 1,
    secondary_instance => 2);
end;
/

```



Rem check instance affinity of OE queue tables from AQ administrative view

```
select queue_table, primary_instance, secondary_instance, owner_instance
from user_queue_tables;
```

```
Rem =====
Rem                               Illustrating Propagation Scheduling
Rem =====
```

Rem Login into OE account

```
set echo on;
connect OE/OE;
set serveroutput on;
```

Rem

Rem Schedule Propagation from bookedorders\_que to shipping

Rem

```
execute dbms_aqadm.schedule_propagation(queue_name => 'OE.OE_bookedorders_que');
```

Rem Login into boladm account

```
set echo on;
connect boladm/boladm;
set serveroutput on;
```

Rem create a procedure to enqueue an order

```
create or replace procedure order_enq(book_title  in varchar2,
                                     book_qty    in number,
                                     order_num    in number,
                                     shipping_priority in number,
                                     cust_state   in varchar2,
                                     cust_country in varchar2,
                                     cust_region  in varchar2,
                                     cust_ord_typ in varchar2) as
```

OE_enq_order_data	BOLADM.order_typ;
OE_enq_cust_data	BOLADM.customer_typ;
OE_enq_book_data	BOLADM.book_typ;
OE_enq_item_data	BOLADM.orderitem_typ;
OE_enq_item_list	BOLADM.orderitemlist_vartyp;
enqopt	dbms_aq.enqueue_options_t;
msgprop	dbms_aq.message_properties_t;
enq_msgid	raw(16);

```

begin

    msgprop.correlation := cust_ord_typ;
    OE_enq_cust_data := BOLADM.customer_typ(NULL, NULL, NULL, NULL,
        cust_state, NULL, cust_country);
    OE_enq_book_data := BOLADM.book_typ(book_title, NULL, NULL, NULL);
    OE_enq_item_data := BOLADM.orderitem_typ(book_qty,
        OE_enq_book_data, NULL);
    OE_enq_item_list := BOLADM.orderitemlist_vartyp(
        BOLADM.orderitem_typ(book_qty,
            OE_enq_book_data, NULL));
    OE_enq_order_data := BOLADM.order_typ(order_num, NULL,
        cust_ord_typ, cust_region,
        OE_enq_cust_data, NULL,
        OE_enq_item_list, NULL);

    -- Put the shipping priority into message property before
    -- enqueueing the message
    msgprop.priority := shipping_priority;
    dbms_aq.enqueue('OE.OE_bookedorders_que', enqopt, msgprop,
        OE_enq_order_data, enq_msgid);

end;
/

show errors;

grant execute on order_enq to OE;

Rem now create a procedure to dequeue booked orders for shipment processing
create or replace procedure shipping_bookedorder_deq(
    consumer in varchar2,
    deqmode in binary_integer) as

    deq_cust_data      BOLADM.customer_typ;
    deq_book_data      BOLADM.book_typ;
    deq_item_data      BOLADM.orderitem_typ;
    deq_msgid          RAW(16);
    dopt               dbms_aq.dequeue_options_t;
    mprop              dbms_aq.message_properties_t;
    deq_order_data     BOLADM.order_typ;
    qname              varchar2(30);
    no_messages        exception;
    pragma exception_init (no_messages, -25228);
    new_orders         BOOLEAN := TRUE;

```

```
begin

    dopt.consumer_name := consumer;
    dopt.wait := DBMS_AQ.NO_WAIT;
    dopt.dequeue_mode := deqmode;
    dopt.navigation := dbms_aq.FIRST_MESSAGE;

    IF (consumer = 'West_Shipping') THEN
        qname := 'WS.WS_bookedorders_que';
    ELSIF (consumer = 'East_Shipping') THEN
        qname := 'ES.ES_bookedorders_que';
    ELSE
        qname := 'OS.OS_bookedorders_que';
    END IF;

    WHILE (new_orders) LOOP
        BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_data,
                msgid => deq_msgid);

            deq_item_data := deq_order_data.items(1);
            deq_book_data := deq_item_data.item;
            deq_cust_data := deq_order_data.customer;

            dbms_output.put_line(' **** next booked order **** ');
            dbms_output.put_line('order_num: ' || deq_order_data.orderno ||
                ' book_title: ' || deq_book_data.title ||
                ' quantity: ' || deq_item_data.quantity);
            dbms_output.put_line('ship_state: ' || deq_cust_data.state ||
                ' ship_country: ' || deq_cust_data.country ||
                ' ship_order_type: ' || deq_order_data.ordertype);
            dopt.navigation := dbms_aq.NEXT_MESSAGE;
        EXCEPTION
            WHEN no_messages THEN
                dbms_output.put_line(' ---- NO MORE BOOKED ORDERS ---- ');
                new_orders := FALSE;
        END;
    END LOOP;

end;
```

```
/
show errors;

Rem now create a procedure to dequeue rush orders for shipment
create or replace procedure get_rushtitles(consumer in varchar2) as

deq_cust_data          BOLADM.customer_typ;
deq_book_data          BOLADM.book_typ;
deq_item_data          BOLADM.orderitem_typ;
deq_msgid              RAW(16);
dopt                   dbms_aq.dequeue_options_t;
mprop                  dbms_aq.message_properties_t;
deq_order_data         BOLADM.order_typ;
qname                  varchar2(30);
no_messages            exception;
pragma exception_init  (no_messages, -25228);
new_orders             BOOLEAN := TRUE;

begin

    dopt.consumer_name := consumer;
    dopt.wait := 1;
    dopt.correlation := 'RUSH';

    IF (consumer = 'West_Shipping') THEN
        qname := 'WS.WS_bookedorders_que';
    ELSIF (consumer = 'East_Shipping') THEN
        qname := 'ES.ES_bookedorders_que';
    ELSE
        qname := 'OS.OS_bookedorders_que';
    END IF;

    WHILE (new_orders) LOOP
        BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_data,
                msgid => deq_msgid);

            deq_item_data := deq_order_data.items(1);
            deq_book_data := deq_item_data.item;

            dbms_output.put_line(' rushorder book_title: ' ||
```

```

                                deq_book_data.title ||
                                ' quantity: ' || deq_item_data.quantity);
EXCEPTION
    WHEN no_messages THEN
        dbms_output.put_line (' ---- NO MORE RUSH TITLES ---- ');
        new_orders := FALSE;
    END;
END LOOP;

end;
/
show errors;

Rem now create a procedure to dequeue orders for handling North American
Rem orders
create or replace procedure get_northamerican_orders as

    deq_cust_data          BOLADM.customer_typ;
    deq_book_data          BOLADM.book_typ;
    deq_item_data          BOLADM.orderitem_typ;
    deq_msgid              RAW(16);
    dopt                   dbms_aq.dequeue_options_t;
    mprop                  dbms_aq.message_properties_t;
    deq_order_data         BOLADM.order_typ;
    deq_order_nodata       BOLADM.order_typ;
    qname                   varchar2(30);
    no_messages            exception;
    pragma exception_init   (no_messages, -25228);
    new_orders              BOOLEAN := TRUE;

begin

    dopt.consumer_name := 'Overseas_Shipping';
    dopt.wait := DBMS_AQ.NO_WAIT;
    dopt.navigation := dbms_aq.FIRST_MESSAGE;
    dopt.dequeue_mode := DBMS_AQ.LOCKED;

    qname := 'OS.OS_bookedorders_que';

    WHILE (new_orders) LOOP
        BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,

```

```
        payload => deq_order_data,
        msgid => deq_msgid);

deq_item_data := deq_order_data.items(1);
deq_book_data := deq_item_data.item;
deq_cust_data := deq_order_data.customer;

IF (deq_cust_data.country = 'Canada' OR
    deq_cust_data.country = 'Mexico' ) THEN

    dopt.dequeue_mode := dbms_aq.REMOVE_NODATA;
    dopt.msgid := deq_msgid;
    dbms_aq.dequeue(
        queue_name => qname,
        dequeue_options => dopt,
        message_properties => mprop,
        payload => deq_order_nodata,
        msgid => deq_msgid);

    dbms_output.put_line(' **** next booked order **** ');
    dbms_output.put_line('order_no: ' || deq_order_data.orderno ||
        ' book_title: ' || deq_book_data.title ||
        ' quantity: ' || deq_item_data.quantity);
    dbms_output.put_line('ship_state: ' || deq_cust_data.state ||
        ' ship_country: ' || deq_cust_data.country ||
        ' ship_order_type: ' || deq_order_data.ordertype);

END IF;

commit;
dopt.dequeue_mode := DBMS_AQ.LOCKED;
dopt.msgid := NULL;
dopt.navigation := dbms_aq.NEXT_MESSAGE;
EXCEPTION
    WHEN no_messages THEN
        dbms_output.put_line (' ---- NO MORE BOOKED ORDERS ---- ');
        new_orders := FALSE;
END;
END LOOP;

end;
/
show errors;

grant execute on shipping_bookedorder_deq to WS;
```

```
grant execute on shipping_bookedorder_deq to ES;
grant execute on shipping_bookedorder_deq to OS;
grant execute on shipping_bookedorder_deq to CS;

grant execute on get_rushtitles to ES;

grant execute on get_northamerican_orders to OS;

Rem Login into OE account
connect OE/OE;
set serveroutput on;

Rem
Rem Enqueue some orders into OE_bookedorders_que
Rem

execute BOLADM.order_enq('My First   Book', 1, 1001, 3, 'CA', 'USA', 'WESTERN',
'NORMAL');
execute BOLADM.order_enq('My Second Book', 2, 1002, 3, 'NY', 'USA', 'EASTERN',
'NORMAL');
execute BOLADM.order_enq('My Third   Book', 3, 1003, 3, '',   'Canada',
'INTERNATIONAL', 'NORMAL');
execute BOLADM.order_enq('My Fourth  Book', 4, 1004, 2, 'NV', 'USA', 'WESTERN',
'RUSH');
execute BOLADM.order_enq('My Fifth   Book', 5, 1005, 2, 'MA', 'USA', 'EASTERN',
'RUSH');
execute BOLADM.order_enq('My Sixth   Book', 6, 1006, 3, '',   'UK',   'INTERNATIONAL',
'NORMAL');
execute BOLADM.order_enq('My Seventh Book', 7, 1007, 1, '',   'Canada',
'INTERNATIONAL', 'RUSH');
execute BOLADM.order_enq('My Eighth  Book', 8, 1008, 3, '',   'Mexico',
'INTERNATIONAL', 'NORMAL');
execute BOLADM.order_enq('My Ninth   Book', 9, 1009, 1, 'CA', 'USA', 'WESTERN',
'RUSH');
execute BOLADM.order_enq('My Tenth   Book', 8, 1010, 3, '',   'UK',   'INTERNATIONAL',
'NORMAL');
execute BOLADM.order_enq('My Last    Book', 7, 1011, 3, '',   'Mexico',
'INTERNATIONAL', 'NORMAL');
commit;
/

Rem
Rem Wait for Propagation to Complete
Rem
```

```
execute dbms_lock.sleep(100);

Rem =====
Rem           Illustrating Dequeue Modes/Methods
Rem =====

connect WS/WS;
set serveroutput on;

Rem Dequeue all booked orders for West_Shipping
execute BOLADM.shipping_bookedorder_deq('West_Shipping', DBMS_AQ.REMOVE);
commit;
/

connect ES/ES;
set serveroutput on;

Rem Browse all booked orders for East_Shipping
execute BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.BROWSE);

Rem Dequeue all rush order titles for East_Shipping
execute BOLADM.get_rushtitles('East_Shipping');
commit;
/

Rem Dequeue all remaining booked orders (normal order) for East_Shipping
execute BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.REMOVE);
commit;
/

connect OS/OS;
set serveroutput on;

Rem Dequeue all international North American orders for Overseas_Shipping
execute BOLADM.get_northamerican_orders;
commit;
/

Rem Dequeue rest of the booked orders for Overseas_Shipping
execute BOLADM.shipping_bookedorder_deq('Overseas_Shipping', DBMS_AQ.REMOVE);
commit;
/
```



```
Rem =====
Rem          Illustrating Enhanced Propagation Capabilities
Rem =====

connect OE/OE;
set serveroutput on;

Rem
Rem Get propagation schedule information & statistics
Rem

Rem get averages
select avg_time, avg_number, avg_size from user_queue_schedules;

Rem get totals
select total_time, total_number, total_bytes from user_queue_schedules;

Rem get status information of schedule (present only when active)
select process_name, session_id, instance, schedule_disabled
       from user_queue_schedules;

Rem get information about last and next execution
select last_run_date, last_run_time, next_run_date, next_run_time
       from user_queue_schedules;

Rem get last error information if any
select failures, last_error_msg, last_error_date, last_error_time
       from user_queue_schedules;

Rem disable propagation schedule for booked orders

execute dbms_aqadm.disable_propagation_schedule(queue_name => 'OE_bookedorders_
que');
execute dbms_lock.sleep(30);
select schedule_disabled from user_queue_schedules;

Rem alter propagation schedule for booked orders to execute every
Rem 15 mins (900 seconds) for a window duration of 300 seconds

begin
dbms_aqadm.alter_propagation_schedule(
        queue_name => 'OE_bookedorders_que',
        duration => 300,
        next_time => 'SYSDATE + 900/86400',
        latency => 25);
```

```
end;
/

execute dbms_lock.sleep(30);
select next_time, latency, propagation_window from user_queue_schedules;

Rem enable propagation schedule for booked orders

execute dbms_aqadm.enable_propagation_schedule(queue_name => 'OE_bookedorders_que');
execute dbms_lock.sleep(30);
select schedule_disabled from user_queue_schedules;

Rem unschedule propagation for booked orders

execute dbms_aqadm.unschedule_propagation(queue_name => 'OE.OE_bookedorders_que');

set echo on;

Rem =====
Rem                               Illustrating Message Grouping
Rem =====

Rem Login into boladm account
set echo on;
connect boladm/boladm;
set serveroutput on;

Rem now create a procedure to handle order entry
create or replace procedure new_order_enq(book_title   in varchar2,
                                           book_qty    in number,
                                           order_num    in number,
                                           cust_state   in varchar2) as

    OE_enq_order_data      BOLADM.order_typ;
    OE_enq_cust_data       BOLADM.customer_typ;
    OE_enq_book_data       BOLADM.book_typ;
    OE_enq_item_data       BOLADM.orderitem_typ;
    OE_enq_item_list       BOLADM.orderitemlist_vartyp;
    enqopt                 dbms_aq.enqueue_options_t;
    msgprop                 dbms_aq.message_properties_t;
    enq_msgid               raw(16);

begin
```

```

OE_enq_cust_data := BOLADM.customer_typ(NULL, NULL, NULL, NULL,
                                         cust_state, NULL, NULL);
OE_enq_book_data := BOLADM.book_typ(book_title, NULL, NULL, NULL);
OE_enq_item_data := BOLADM.orderitem_typ(book_qty,
                                         OE_enq_book_data, NULL);
OE_enq_item_list := BOLADM.orderitemlist_vartyp(
    BOLADM.orderitem_typ(book_qty,
    OE_enq_book_data, NULL));
OE_enq_order_data := BOLADM.order_typ(order_num, NULL,
    NULL, NULL,
    OE_enq_cust_data, NULL,
    OE_enq_item_list, NULL);
dbms_aq.enqueue('OE.OE_neworders_que', enqopt, msgprop,
    OE_enq_order_data, enqmsgid);

end;
/
show errors;

Rem now create a procedure to handle order enqueue
create or replace procedure same_order_enq(book_title  in varchar2,
                                         book_qty    in number) as

OE_enq_order_data    BOLADM.order_typ;
OE_enq_book_data     BOLADM.book_typ;
OE_enq_item_data     BOLADM.orderitem_typ;
OE_enq_item_list     BOLADM.orderitemlist_vartyp;
enqopt               dbms_aq.enqueue_options_t;
msgprop             dbms_aq.message_properties_t;
enqmsgid             raw(16);

begin

    OE_enq_book_data := BOLADM.book_typ(book_title, NULL, NULL, NULL);
    OE_enq_item_data := BOLADM.orderitem_typ(book_qty,
        OE_enq_book_data, NULL);
    OE_enq_item_list := BOLADM.orderitemlist_vartyp(
        BOLADM.orderitem_typ(book_qty,
        OE_enq_book_data, NULL));
    OE_enq_order_data := BOLADM.order_typ(NULL, NULL,
        NULL, NULL,
        NULL, NULL,
        OE_enq_item_list, NULL);
    dbms_aq.enqueue('OE.OE_neworders_que', enqopt, msgprop,
        OE_enq_order_data, enqmsgid);

```

```
end;
/
show errors;

grant execute on new_order_enq to OE;
grant execute on same_order_enq to OE;

Rem now create a procedure to get new orders by dequeuing
create or replace procedure get_new_orders as

    deq_cust_data          BOLADM.customer_typ;
    deq_book_data          BOLADM.book_typ;
    deq_item_data          BOLADM.orderitem_typ;
    deq_msgid              RAW(16);
    dopt                   dbms_aq.dequeue_options_t;
    mprop                  dbms_aq.message_properties_t;
    deq_order_data         BOLADM.order_typ;
    qname                  varchar2(30);
    no_messages            exception;
    end_of_group           exception;
    pragma exception_init  (no_messages, -25228);
    pragma exception_init  (end_of_group, -25235);
    new_orders             BOOLEAN := TRUE;

begin

    dopt.wait := 1;
    dopt.navigation := DBMS_AQ.FIRST_MESSAGE;
    qname := 'OE.OE_neworders_queue';
    WHILE (new_orders) LOOP
        BEGIN
            LOOP
                BEGIN
                    dbms_aq.dequeue(
                        queue_name => qname,
                        dequeue_options => dopt,
                        message_properties => mprop,
                        payload => deq_order_data,
                        msgid => deq_msgid);

                    deq_item_data := deq_order_data.items(1);
                    deq_book_data := deq_item_data.item;
                    deq_cust_data := deq_order_data.customer;

                    IF (deq_cust_data IS NOT NULL) THEN
```

```

        dbms_output.put_line(' **** NEXT ORDER **** ');
        dbms_output.put_line('order_num: ' ||
                               deq_order_data.orderno);
        dbms_output.put_line('ship_state: ' ||
                               deq_cust_data.state);
    END IF;
    dbms_output.put_line(' ---- next book ---- ');
    dbms_output.put_line(' book_title: ' ||
                           deq_book_data.title ||
                           ' quantity: ' || deq_item_data.quantity);
EXCEPTION
    WHEN end_of_group THEN
        dbms_output.put_line ('*** END OF ORDER ***');
        commit;
        dopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
    END;
END LOOP;
EXCEPTION
    WHEN no_messages THEN
        dbms_output.put_line (' ---- NO MORE NEW ORDERS ---- ');
        new_orders := FALSE;
    END;
END LOOP;

end;
/

show errors;

grant execute on get_new_orders to OE;

Rem Login into OE account
connect OE/OE;
set serveroutput on;

Rem
Rem Enqueue some orders using message grouping into OE_neworders_que
Rem

Rem First Order
execute BOLADM.new_order_enq('My First  Book', 1, 1001, 'CA');
execute BOLADM.same_order_enq('My Second  Book', 2);
commit;
/

```

```
Rem Second Order
execute BOLADM.new_order_enq('My Third   Book', 1, 1002, 'WA');
commit;
/

Rem Third Order
execute BOLADM.new_order_enq('My Fourth  Book', 1, 1003, 'NV');
execute BOLADM.same_order_enq('My Fifth   Book', 3);
execute BOLADM.same_order_enq('My Sixth   Book', 2);
commit;
/

Rem Fourth Order
execute BOLADM.new_order_enq('My Seventh Book', 1, 1004, 'MA');
execute BOLADM.same_order_enq('My Eighth  Book', 3);
execute BOLADM.same_order_enq('My Ninth   Book', 2);
commit;
/

Rem
Rem Dequeue the neworders
Rem

execute BOLADM.get_new_orders;
```

## tkaqdocc.sql: クリーンアップ・スクリプト

```
Rem
Rem $Header: tkaqdocc.sql 26-jan-99.17:51:05 aquer1 Exp $
Rem
Rem tkaqdocc.sql
Rem
Rem Copyright (c) Oracle Corporation 1998, 1999. All Rights Reserved.
Rem
Rem NAME
Rem      tkaqdocc.sql - <one-line expansion of the name>
Rem

set echo on;
connect system/manager
set serveroutput on;

drop user WS cascade;
drop user ES cascade;
drop user OS cascade;
drop user CB cascade;
drop user CBADM cascade;
drop user CS cascade;
drop user OE cascade;
drop user boladm cascade;
```





---

## JMS エラー・メッセージ

この付録では、トラブルシューティングに有効なエラー・メッセージを掲載します。

---

### **JMS-101 Invalid delivery mode (string)**

**原因:** 配信モードがサポートされていません。

**処置:** 有効な配信モードは、AQjmsConstants.PERSISTENT です。

### **JMS-102 Feature not supported (string)**

**原因:** この機能は、現行のリリースではサポートされていません。

**処置:** 処置はありません。

### **JMS-104 Message Payload must be specified**

**原因:** メッセージ・ペイロードが NULL です。

**処置:** メッセージに、NULL 以外のペイロードを指定してください。

### **JMS-105 Agent must be specified**

**原因:** AqjmsAgent オブジェクトが NULL です。

**処置:** リモート・サブスクライバを表す有効な AqjmsAgent を指定してください。

### **JMS-106 Cannot have more than one open Session on a JMSConnection**

**原因:** 接続上に、オープンされている JMS セッションがあります。接続上には、セッションを1つしかオープンできません。

**処置:** オープン・セッションをクローズしてから、新しいセッションをオープンしてください。

### **JMS-107 Operation not allowed on (string)**

**原因:** このオブジェクトでは使用できない操作が指定されています。

**処置:** 処置はありません。

### **JMS-108 Messages of type (string) not allowed with Destinations containing payload of type (string)**

**原因:** 使用されているメッセージ型と、宛先に指定されているペイロード型が一致していません。

**処置:** この宛先を含むキュー・テーブルに指定されているペイロードに割り当てられているメッセージ型を使用してください。

---

**JMS-109 Class not found: (string)**

**原因:** 指定されたクラスが見つかりません。

**処置:** CLASSPATH に、指定されたクラスが含まれているかどうかを確認してください。

**JMS-110 Property (string) not writeable**

**原因:** 読み込み専用メッセージのヘッダー・フィールドまたはプロパティを更新しようとした。

**処置:** 処置はありません。

**JMS-111 Connection must be specified**

**原因:** 接続オブジェクトが NULL です。

**処置:** NULL 以外の JDBC 接続を指定してください。

**JMS-112 Connection is invalid**

**原因:** JDBC 接続が無効です。

**処置:** NULL 以外の Oracle JDBC 接続を指定してください。

**JMS-113 Connection is in stopped state**

**原因:** 停止状態の接続上でメッセージを受信しようとした。

**処置:** 接続を開始してください。

**JMS-114 Connection is closed**

**原因:** クローズされた接続を使用しようとした。

**処置:** 新しい接続を確立してください。

**JMS-115 Consumer is closed**

**原因:** クローズされたコンシューマを使用しようとした。

**処置:** 新しいメッセージ・コンシューマを作成してください。

**JMS-116 Subscriber name must be specified**

**原因:** サブスクライバ名が NULL です。

**処置:** NULL 以外のサブスクリプション名を指定してください。

---

### **JMS-117   Conversion failed - invalid property type**

**原因:** 要求された型にプロパティを変換中に、エラーが発生しました。

**処置:** プロパティのデータ型に対応したメソッドを使用して、プロパティを取り出してください。

### **JMS-119   Invalid Property value**

**原因:** 無効なプロパティの値が指定されています。

**処置:** 設定中のプロパティに、適切な型の値を使用してください。

### **JMS-120   Dequeue failed**

**原因:** メッセージの受信中にエラーが発生しました。

**処置:** 詳細は、JMSException およびリンクされた SQLException 内のメッセージを参照してください。

### **JMS-121   DestinationProperty must be specified**

**原因:** キュー / トピックの作成中に、NULL の AqjmsDestinationProperty が指定されました。

**処置:** 宛先に、NULL 以外の AQjmsDestinationProperty を指定してください。

### **JMS-122   Internal error (string)**

**原因:** 内部エラーが発生しました。

**処置:** オラクル社カスタマ・サポート・センターに連絡してください。

### **JMS-123   Interval must be at least (integer) seconds**

**原因:** 無効な間隔が指定されました。

**処置:** 間隔は、30 秒より大きい値に指定してください。

### **JMS-124   Invalid Dequeue mode**

**原因:** 無効なデキュー・モードが指定されました。

**処置:** 有効なデキュー・モードは、AQConstants.DEQUEUE\_BROWSE、AQConstants.DEQUEUE\_REMOVE、AQConstants.DEQUEUE\_LOCKED および AQConstants.DEQUEUE\_REMOVE\_NODATA です。

---

### **JMS-125 Invalid Queue specified**

**原因:** 無効なキュー・オブジェクトが指定されました。

**処置:** 有効なキュー・ハンドルを指定してください。

### **JMS-126 Invalid Topic specified**

**原因:** 無効なトピック・オブジェクトが指定されました。

**処置:** 有効なトピック・ハンドルを指定してください。

### **JMS-127 Invalid Destination**

**原因:** 無効な宛先オブジェクトが指定されました。

**処置:** 有効な宛先（キュー / トピック）オブジェクトを指定してください。

### **JMS-128 Invalid Navigation mode**

**原因:** 無効なナビゲーション・モードが指定されました。

**処置:** 有効なナビゲーション・モードは、  
AQjmsConstants.NAVIGATION\_FIRST\_MESSAGE、  
AQjmsConstants.NAVIGATION\_NEXT\_MESSAGE および  
AQjmsConstants.NAVIGATION\_NEXT\_TRANSACTION です。

### **JMS-129 Invalid Payload type**

**原因:** 使用されているメッセージ型と、宛先に指定されているペイロードの型が一致していません。

**処置:** この宛先を含むキュー・テーブルに指定されたペイロードに割り当てられているメッセージ型を使用してください。オブジェクト型メッセージでは、適切な CustomDatum ファクトリを使用して、メッセージ・コンシューマを作成してください。

### **JMS-130 JMS queue cannot be multi-consumer enabled**

**原因:** AQ 複数コンシューマ・キューを、JMS キューとして取得しようとしてしました。

**処置:** JMS キューでは、複数コンシューマを使用できません。

### **JMS-131 Session is closed**

**原因:** クローズされたセッションを使用しようとしてしました。

**処置:** 新しいセッションをオープンしてください。

---

### **JMS-132 Maximum number of properties (integer) exceeded**

**原因:** メッセージに対するユーザー定義のプロパティの数が、最大値を超えました。

**処置:** 処置はありません。

### **JMS-133 Message must be specified**

**原因:** NULL のメッセージが指定されました。

**処置:** NULL 以外のメッセージを指定してください。

### **JMS-134 Name must be specified**

**原因:** NULL のキューまたはキュー・テーブル名が指定されました。

**処置:** NULL 以外の名前を指定してください。

### **JMS-135 Driver (string) not supported**

**原因:** サポートされていないドライバが指定されています。

**処置:** 有効な JDBC ドライバは、OCI8 および Thin です。サーバー・ドライバを使用するには、getDefaultConnection() を使用して接続を取得し、静的な createTopicConnection および createQueueConnection メソッドを使用してください。

### **JMS-136 Payload factory can only be specified for destinations with ADT payloads**

**原因:** オブジェクト型ペイロードを含まない宛先のコンシューマに、CustomDatumFactory が指定されました。

**処置:** ペイロード型が SYS.AQ\$\_JMS\_TEXT\_MESSAGE、SYS.AQ\$\_JMS\_BYTES\_MESSAGE、SYS.AQ\$\_JMS\_MAP\_MESSAGE、SYS.AQ\$\_JMS\_OBJECT\_MESSAGE または SYS.AQ\$\_JMS\_STREAM\_MESSAGE の宛先では、このフィールドを NULL に設定してください。

### **JMS-137 Payload factory must be specified for destinations with ADT payloads**

**原因:** オブジェクト型ペイロードを含む宛先に、CustomDatumFactory が指定されませんでした。

**処置:** オブジェクト型メッセージを含む宛先では、オブジェクト型の宛先に割り当てる Java クラスに対して CustomDatumFactory を指定してください。

### **JMS-138 Producer is closed**

**原因:** クローズされたプロデューサを使用しようとしてしました。

**処置:** 新しいメッセージ・プロデューサを作成してください。

---

**JMS-139 Property name must be specified**

**原因:** プロパティ名が NULL です。

**処置:** NULL 以外のプロパティ名を指定してください。

**JMS-140 Invalid System property**

**原因:** 無効なシステム・プロパティ名が指定されました。

**処置:** 有効な JMS システム・プロパティを指定してください。

**JMS-142 JMS topic must be created in multi-consumer enabled queue tables**

**原因:** 単一コンシューマ・キュー・テーブル内に、JMS トピックを作成しようとした。

**処置:** JMS トピックは、複数コンシューマが使用可能なキュー・テーブルにのみ作成してください。

**JMS-143 Queue must be specified**

**原因:** NULL のキューが指定されました。

**処置:** NULL でないキューを指定してください。

**JMS-144 JMS queue cannot be created in multi-consumer enabled queue tables**

**原因:** 複数コンシューマ・キュー・テーブル内に、JMS キューを作成しようとした。

**処置:** JMS キューは、複数コンシューマが使用可能でないキュー・テーブルにのみ作成してください。

**JMS-145 Invalid recipient list**

**原因:** 空の受信者リストが指定されました。

**処置:** 1 つ以上の受信者を含む受信者リストを指定してください。

**JMS-146 Registration failed**

**原因:** 型をタイプ・マップに登録中に、エラーが発生しました。

**処置:** 処置はありません。

**JMS-147 Invalid ReplyTo destination type**

**原因:** ReplyTo 宛先のオブジェクト型が無効です。

**処置:** ReplyTo 宛先の型は、AqjmsAgent にしてください。

---

### **JMS-148 Property name size exceeded**

**原因:** プロパティ名が最大サイズを超えています。

**処置:** プロパティ名は、100 文字未満で指定してください。

### **JMS-149 Subscriber must be specified**

**原因:** NULL のサブスクライバが指定されました。

**処置:** NULL 以外のサブスクライバを指定してください。

### **JMS-150 Property not supported**

**原因:** サポートされていないプロパティを使用しようとしてしました。

**処置:** 処置はありません。

### **JMS-151 Topics cannot be of type EXCEPTION**

**原因:** トピックは、AQjmsConstants.EXCEPTION タイプにできません。

**処置:** トピックが、AQjmsConstants.NORMAL タイプになるように指定してください。

### **JMS-153 Invalid System property type**

**原因:** 指定された値の型が、設定中のシステム・プロパティに定義された型と一致していません。

**処置:** システム・プロパティの設定と一致した型を使用してください。

### **JMS-154 Invalid value for sequence deviation**

**原因:** 順序逸脱が無効です。

**処置:** 有効な値は、AQEnqueueOption.DEVIATION\_BEFORE および AQEnqueueOption.DEVIATION\_TOP です。

### **JMS-155 AQ Exception (string)**

**原因:** AQ Java レイヤーでエラーが発生しました。

**処置:** 詳細は、JMSException およびリンクされた Exception 内のメッセージを参照してください。

### **JMS-156 Invalid Class (string)**

**原因:** 無効なクラスが指定されています。

**処置:** CLASSPATH に、指定されたクラスが含まれているかどうかを確認してください。



---

### **JMS-157 IO Exception (string)**

**原因:** I/O の例外です。

**処置:** 詳細は、JMSEException 内のメッセージを参照してください。

### **JMS-158 SQL Exception (string)**

**原因:** SQL の例外です。

**処置:** 詳細は、リンクされた SQLException 内のメッセージを参照してください。

### **JMS-159 Invalid selector (string)**

**原因:** 無効なセクタまたは長すぎるセクタが指定されています。

**処置:** セクタの構文を確認してください。

### **JMS-160 EOF Exception (string)**

**原因:** バイト・ストリームの読み込み中に EOF 例外が発生しました。

**処置:** 処置はありません。

### **JMS-161 MessageFormat Exception: (string)**

**原因:** ストリーム・データを指定された型に変換中に、エラーが発生しました。

**処置:** ストリーム上で予測されるデータの型を確認して、適切な読み込みメソッドを使用してください。

### **JMS-162 Message not Readable**

**原因:** メッセージが書き込み専用モードです。

**処置:** リセット・メソッドをコールして、メッセージを読み込み可能にしてください。

### **JMS-163 Message not Writeable**

**原因:** メッセージが読み込み専用モードです。

**処置:** clearBody メソッドを使用して、メッセージを書込み可能にしてください。

### **JMS-164 No such element**

**原因:** 指定された名前の要素が、Map メッセージ内に見つかりません。

**処置:** 処置はありません。

---

### **JMS-165 Maximum size of property value exceeded**

**原因:** プロパティの値が、最大長を超えています。

**処置:** JMS 定義のプロパティの値の最大長は 100 です。ユーザー定義のプロパティの値の最大長は 2000 です。

### **JMS-166 Topic must be specified**

**原因:** NULL のトピックが指定されました。

**処置:** NULL 以外のトピックを指定してください。

### **JMS-167 Payload factory or Sql\_data\_class must be specified**

**原因:** オブジェクト・ペイロードを含むキューに、ペイロード・ファクトリまたは Sql\_data\_class が指定されていません。

**処置:** CustomDatumFactory、またはキューに定義された ADT 型にマップする Java オブジェクトの SQLData クラスを指定してください。

### **JMS-168 Cannot specify both payload factory and sql\_data\_class**

**原因:** デキュー中に、CustomDatumFactory および SQLData クラスが指定されました。

**処置:** CustomDatumFactory、またはキューに定義された ADT 型にマップする Java オブジェクトの SQLData クラスのいずれかを指定してください。

### **JMS-169 Sql\_data\_class cannot be null**

**原因:** NULL の SQLData クラスが指定されています。

**処置:** キューに定義された ADT 型にマップする SQLData クラスを指定してください。

### **JMS-171 Message is not defined to contain (string)**

**原因:** メッセージ内のペイロード型が無効です。

**処置:** キューが RAW または OBJECT ペイロードを含むように定義されているかどうかを確認し、メッセージ内に適切なペイロード型を使用してください。

### **JMS-172 More than one queue table matches query (string)**

**原因:** 問合せに 2 つ以上のキュー・テーブルが一致します。

**処置:** 所有者およびキュー・テーブル名を指定してください。

### **JMS-173 Queue Table (string) not found**

**原因:** 指定されたキュー・テーブルが見つかりません。

**処置:** 有効なキュー・テーブルを指定してください。

---

**JMS-174 Class must be specified for queues with object payloads\n Use dequeue(deq\_option,payload\_fact) or dequeue(deq\_option, sql\_data\_cl)**

**原因:** このデキュー・メソッドは、OBJECT ペイロードを含むキューからのデキューには使用できません。

**処置:** dequeue(deq\_option, payload\_fact) または dequeue(deq\_option, sql\_data\_cl) のいずれかを使用してください。

**JMS-175 DequeueOption must be specified**

**原因:** NULL のデキュー・オプションが指定されています。

**処置:** NULL 以外のデキュー・オプションを指定してください。

**JMS-176 EnqueueOption must be specified**

**原因:** NULL のエンキュー・オプションが指定されています。

**処置:** NULL 以外のエンキュー・オプションを指定してください。

**JMS-177 Invalid payload type: Use dequeue(deq\_option) for raw payload queues**

**原因:** この方法は、RAW ペイロードを含むキューからのデキューには使用できません。

**処置:** dequeue(deq\_option) メソッドを使用してください。

**JMS-178 Invalid Queue name - (string)**

**原因:** NULL または無効なキュー名が指定されています。

**処置:** NULL 以外のキュー名を使用してください。キュー名はスキーマ名で修飾しないでください。スキーマ名は、owner パラメータの値として指定してください。

**JMS-179 Invalid Queue Table name - (string)**

**原因:** NULL または無効なキュー・テーブル名が指定されています。

**処置:** NULL 以外のキュー・テーブル名を使用してください。キュー・テーブル名はスキーマ名で修飾しないでください。スキーマ名は、owner パラメータの値として指定してください。

**JMS-180 Invalid Queue Type**

**原因:** キュー・タイプが無効です。

**処置:** 有効なタイプは、AQConstants.NORMAL または AQConstants.EXCEPTION です。

---

### **JMS-181 Invalid value for wait\_time**

**原因:** 待機タイプの値が無効です。

**処置:** 待機時間は、AQDequeueOption.WAIT\_FOREVER、AQDequeueOption.WAIT\_NONE または 0（ゼロ）より大きいすべての値に指定できます。

### **JMS-182 More than one queue matches query**

**原因:** 問合せに 2 つ以上のキューが一致します。

**処置:** キューの所有者および名前を指定してください。

### **JMS-183 No AQ driver registered**

**原因:** AQDriver が登録されていません。

**処置:** AQ Java ドライバが登録されているかどうかを確認してください。  
Class.forName("oracle.AQ.AQOracleDriver") を使用してください。

### **JMS-184 Queue object is invalid**

**原因:** キュー・オブジェクトが無効です。

**処置:** 基礎となる JDBC 接続がクローズされている可能性があります。キュー・ハンドルを再度取得してください。

### **JMS-185 QueueProperty must be specified**

**原因:** NULL の AQQueueProperty が指定されています。

**処置:** NULL 以外の AQQueueProperty を指定してください。

### **JMS-186 QueueTableProperty must be specified**

**原因:** NULL の QueueTableProperty が指定されています。

**処置:** NULL 以外の AQQueueTableProperty を指定してください。

### **JMS-187 Queue Table must be specified**

**原因:** NULL のキュー・テーブルが指定されています。

**処置:** NULL 以外のキュー・テーブルを指定してください。

### **JMS-188 QueueTable object is invalid**

**原因:** キュー・テーブル・オブジェクトが無効です。

**処置:** 基礎となる JDBC 接続がクローズされている可能性があります。キュー・テーブル・ハンドルを再度取得してください。

---

### **JMS-189    Byte array too small**

**原因:** 指定されたバイト配列が小さすぎるため、要求されたデータを保持できません。

**処置:** 要求されたデータを保持できる大きいバイト配列を指定するか、または要求を短くしてください。

### **JMS-190    Queue (string) not found**

**原因:** 指定されたキューが見つかりません。

**処置:** 有効なキューを指定してください。

### **JMS-191    sql\_data\_cl must be a class that implements SQLData interface**

**原因:** java.sql.SQLData インタフェースをサポートしないクラスが指定されています。

**処置:** 処置はありません。

### **JMS-192    Invalid Visibility value**

**原因:** 無効な値の可視性が指定されています。

**処置:** 有効な値は、AQConstants.VISIBILITY\_ONCOMMIT および AQConstants.VISIBILITY\_IMMEDIATE です。

### **JMS-193    JMS queues cannot contain payload of type RAW**

**原因:** RAW ペイロードを含む JMS キューを作成しようとしてしました。

**処置:** JMS キュー / トピックには、RAW ペイロードを含めないでください。

### **JMS-194    Session object is invalid**

**原因:** セッション・オブジェクトが無効です。

**処置:** 基礎となる JDBC 接続がクローズされている可能性があります。新しいセッションを確立してください。

### **JMS-195    Invalid object type: object must implement CustomDatum or SQLData interface**

**原因:** 無効なオブジェクト型が指定されています。

**処置:** オブジェクトには、CustomDatum または SQLData インタフェースを実装してください。

---

### **JMS-196 Cannot have more than one open QueueBrowser for the same destination on a JMS Session**

**原因:** このセッションのこのキューでは、キュー・ブラウザがすでにオープンされています。

**処置:** 特定のセッションの同じキューでは、2つ以上のキュー・ブラウザをオープンできません。既存のキュー・ブラウザをクローズしてから、新しくオープンしてください。

### **JMS-197 Agent address must be specified for remote subscriber**

**原因:** リモート・サブスクライバのアドレス・フィールドが NULL です。

**処置:** アドレス・フィールドには、完全に修飾されたリモート・トピック名を含めてください。

### **JMS-198 Invalid operation: Privileged message listener set for the Session**

**原因:** セッション・メッセージ・リスナーが設定されたときに、クライアントがメッセージ・コンシューマを使用して、メッセージを受信しようとした。

**処置:** セッションのメッセージ・リスナーを使用して、メッセージを使用してください。コンシューマのメッセージ受信メソッドは、使用しないでください。

### **JMS-199 Registration for notification failed**

**原因:** リスナー登録に失敗しました。

**処置:** 詳細は、リンクされた Exception 内のエラー・メッセージを参照してください。

### **JMS-200 Destination must be specified**

**原因:** 宛先が NULL です。

**処置:** NULL 以外の宛先を指定してください。

### **JMS-201 All Recipients in recipient\_list must be specified**

**原因:** 受信者リスト内の1つ以上の要素が NULL です。

**処置:** 受信者リスト内のすべての AqjmsAgents を指定してください。

---

# 索引

## A

---

ADT メッセージ, 12-28  
Agent address must be specified for remote subscriber -  
エラー, D-14  
Agent must be specified - エラー, D-2  
All Recipients in recipient\_list must be specified -  
エラー, D-14  
AQ Excetpion (string) - エラー, D-8  
AQ\_TM\_PROCESSES, 2-7  
AQjmsQueueConnectionFactory, B-75  
AQ エージェント・リスト型, 2-4  
AQ オブジェクト型、アクセス, 4-7  
AQ オブジェクトの削除, A-66  
AQ サブスクライバ・リスト型, 2-4  
AQ 受信者リスト型, 2-4  
AQ の例, A-1  
AQ を使用したアプリケーション例, 8-1  
AQ を使用したサンプル・アプリケーション, 8-1  
AQ を使用したパブリッシュ / サブスクライブ, 7-13

## B

---

BooksOnLine のサンプル・アプリケーション, 8-1  
Byte array too small - エラー, D-13  
Byte メッセージ, 12-25

## C

---

C, 11-27, 11-39, 11-61  
Cannot have more than one open QueueBrowser for  
the same destination - エラー, D-14  
Cannot have more than one open Session on a  
JMSConnection - エラー, D-2

Cannot specify both payload factory and sql\_data\_class  
- エラー, D-10  
Class must be specified for queues with object  
payloads\n Use dequeue - エラー, D-11  
Class not found  
(string) - エラー, D-3  
Connection is closed - エラー, D-3  
Connection is in stopped state - エラー, D-3  
Connection is invalid - エラー, D-3  
Connection must be specified - エラー, D-3  
Consumer is closed - エラー, D-3  
Conversion failed - invalid property type - エラー, D-4

## D

---

DBA\_QUEUE\_TABLES, 10-5, 10-8, 10-26  
DBA\_QUEUEUES, 10-11  
DBMS\_AQADM.DROP\_QUEUE, 9-18  
Dequeue failed - エラー, D-4  
dequeue mode, 2-6  
DequeueOption must be specified - エラー, D-11  
DEQUEUE 機能, 8-54  
Destination must be specified - エラー, D-14  
DestinationProperty must be specified - エラー, D-4  
Driver (string) not supported - エラー, D-6

## E

---

EnqueueOption must be specified - エラー, D-11  
ENQUEUE 機能, 8-34  
EOF Exception (string) - エラー, D-9  
expiration, 2-6

## F

---

FAQ, 6-1

Feature not supported (string) - エラー, D-2

## I

---

INIT.ORA パラメータ, 2-7

Internal error (string) - エラー, D-4

Interval must be atleast (integer) seconds - エラー, D-4

Invalid Class (string) - エラー, D-8

Invalid delivery mode (string) - エラー, D-2

Invalid Dequeue mode - エラー, D-4

Invalid Destination - エラー, D-5

Invalid Navigation mode - エラー, D-5

Invalid object type

object must implement CustomDatum or SQLData  
interface - エラー, D-13

Invalid operation

Privileged message listener - エラー, D-14

Invalid payload type

Use dequeue - エラー, D-11

Invalid Payload type - エラー, D-5

Invalid Property value - エラー, D-4

Invalid Queue name - (string) - エラー, D-11

Invalid Queue specified - エラー, D-5

Invalid Queue Table name - (string) - エラー, D-11

Invalid Queue Type - エラー, D-11

Invalid recipient list - エラー, D-7

Invalid ReplyTo destination type - エラー, D-7

Invalid selector (string) - エラー, D-9

Invalid System property type - エラー, D-8

Invalid System property - エラー, D-7

Invalid Topic specified - エラー, D-5

Invalid value for sequence deviation - エラー, D-8

Invalid value for wait\_time - エラー, D-12

Invalid Visibility value - エラー, D-13

IO Exception (string) - エラー, D-9

## J

---

Java

「JDBC」を参照

javax.jms.BytesMessage, B-17

javax.jms.Connection, B-18

javax.jms.ConnectionFactory, B-19

javax.jms.ConnectionMetaData, B-20

javax.jms.DeliveryMode, B-21

javax.jms.Destination, B-22

javax.jms.InvalidDestinationException, B-45

javax.jms.InvalidSelectorException, B-46

javax.jms.JMSEException, B-47

javax.jms.MapMessage, B-23

javax.jms.MessageNotWriteableException, B-51

javax.jms.Message, B-24

javax.jms.MessageConsumer, B-25

javax.jms.MessageEOFException, B-48

javax.jms.MessageFormatException, B-49

javax.jms.MessageListener, B-26

javax.jms.MessageNotReadableException, B-50

javax.jms.MessageProducer, B-27

javax.jms.ObjectMessage, B-28

javax.jms.Queue, B-29

javax.jms.QueueBrowser, B-30

javax.jms.QueueConnection, B-31

javax.jms.QueueConnectionFactory, B-32

javax.jms.QueueReceiver, B-33

javax.jms.QueueSender, B-34

javax.jms.QueueSession, B-35

javax.jms.Session, B-36

javax.jms.StreamMessage, B-37

javax.jms.TextMessage, B-38

javax.jms.Topic, B-39

javax.jms.TopicConnection, B-40

javax.jms.TopicSession, B-43

javax.jms.TopicSubscriber, B-44

JMS queue cannot be created in multi-consumer  
enabled queue tables - エラー, D-7

JMS queue cannot be multi-consumer enabled - エラー,  
D-5

JMS queues cannot contain payload of type RAW -  
エラー, D-13

JMS topic must be created in multi-consumer enabled  
queue tables - エラー, D-7

JMS インタフェース, B-2

JMS エラーのトラブルシューティング, D-1

JMS エラー・メッセージ, D-1

JMS クラス, B-2, B-5

JMS サンプル・アプリケーション, 12-1

JMS サンプルのペイロード, 12-31

JMS でのアプリケーションの作成, 12-1

JMS の拡張機能, 3-10

JMS 例外, B-2

JOB\_QUEUE\_PROCESSES, 2-7



## L

---

Listen 機能, 8-97  
LOB 属性を伴うメッセージの伝播, 8-107

## M

---

Map メッセージ, 12-26  
Maximum number of properties (integer) exceeded - エラー, D-6  
Maximum size of property value exceeded - エラー, D-10  
Message is not defined to contain (string) - エラー, D-10  
Message must be specified - エラー, D-6  
Message not Readable - エラー, D-9  
Message not Writeable - エラー, D-9  
Message Payload must be specified - エラー, D-2  
message\_grouping, 2-5  
MessageFormat Exception (string) - エラー, D-9  
Messages of type (string) not allowed with Destinations containing payload of type (string) - エラー, D-2  
More than one queue matches query - エラー, D-12  
More than one queue table matches query (string) - エラー, D-10

## N

---

Name must be specified - エラー, D-6  
No AQ driver registered - エラー, D-12  
No such element - エラー, D-9

## O

---

Object メッセージ, 12-27  
OO4O, 3-6, 9-9  
Operation not allowed on (string) - エラー, D-2  
Oracle JMS クラス, B-5  
Oracle Parallel Server のサポート, 12-17  
Oracle Parallel Server (OPS) のサポート, 8-28  
oracle.AQ.AQQueueTable, B-87  
oracle.AQ.AQQueueTableProperty, B-88  
oracle.jms.AdtMessage, B-52  
oracle.jms.AQjmsAdtMessage, B-59  
oracle.jms.AQjmsAgent, B-60  
oracle.jms.AQjmsBytesMessage, B-61

oracle.jms.AQjmsConnection, B-62  
oracle.jms.AQjmsConstants, B-64  
oracle.jms.AQjmsConsumer, B-65  
oracle.jms.AQjmsDestination, B-66  
oracle.jms.AQjmsDestinationProperty, B-67  
oracle.jms.AQjmsException, B-80  
oracle.jms.AQjmsFactory, B-68  
oracle.jms.AQjmsInvalidDestinationException, B-81  
oracle.jms.AQjmsInvalidSelectorException, B-82  
oracle.jms.AQjmsMapMessage, B-69  
oracle.jms.AQjmsMessageEOFException, B-83  
oracle.jms.AQjmsMessageFormatException, B-84  
oracle.jms.AQjmsMessageNotReadableException, B-85  
oracle.jms.AQjmsMessageNotWriteableException, B-86  
oracle.jms.AQjmsObjectMessage, B-71  
oracle.jms.AQjmsOracleDebug, B-72  
oracle.jms.AQjmsProducer, B-73  
oracle.jms.AQjmsQueueBrowser, B-74  
oracle.jms.AQjmsQueueReceiver, B-53  
oracle.jms.AQjmsQueueSender, B-54  
oracle.jms.AQjmsSession, B-76  
oracle.jms.AQjmsStreamMessage, B-77  
oracle.jms.AQjmsTextMessage, B-78  
oracle.jms.AQjmsTopicConnectionFactory, B-79  
oracle.jms.AQjmsTopicPublisher, B-55  
oracle.jms.AQjmsTopicReceiver, B-58  
oracle.jms.AQjmsTopicSubscriber, B-57  
oracle.jms.TopicReceiver, B-56  
Oracle の拡張機能, 3-10

## P

---

Payload factory can only be specified for destinations with ADT payloads - エラー, D-6  
Payload factory must be specified for destinations with ADT payloads - エラー, D-6  
Payload factory or Sql\_data\_class must be specified - エラー, D-10  
PL/SQL, 3-2  
Producer is closed - エラー, D-6  
Property (string) not writeable - エラー, D-3  
Property name must be specified - エラー, D-7  
Property name size exceeded - エラー, D-8  
Property not supported - エラー, D-8

## Q

---

Queue object is invalid - エラー, D-12  
Queue (string) not found - エラー, D-13  
Queue Table must be specified - エラー, D-12  
Queue Table (string) not found - エラー, D-10  
queue\_type, 2-5  
QueueProperty must be specified - エラー, D-12  
QueueTable object is invalid - エラー, D-12  
QueueTableProperty must be specified - エラー, D-12

## R

---

RAW 型のキュー・テーブルおよびキューの作成,  
9-23, A-5, 9-9  
Registration failed - エラー, D-7  
Registration for notification failed - エラー, D-14

## S

---

sequence\_deviation, 2-6  
Session is closed - エラー, D-5  
Session object is invalid - エラー, D-13  
SQL Exception (string) - エラー, D-9  
sql\_data\_cl must be a class that implements SQLData  
interface - エラー, D-13  
Sql\_data\_class cannot be null - エラー, D-10  
Stream メッセージ, 12-24  
Subscriber must be specified - エラー, D-8  
Subscriber name must be specified - エラー, D-3

## T

---

Text メッセージ, 12-27  
Topic must be specified - エラー, D-10  
Topics cannot be of type EXCEPTION - エラー, D-8

## V

---

Visual Basic  
「OO4O」を参照

## あ

---

アクセス制御, 4-5  
アクセス制御、システム・レベル, 8-5  
宛先レベルのアクセス制御, 12-15

## アドバンスト・キューイング

DBMS\_AQADM パッケージ, 4-4

管理インタフェース

権限およびアクセス制御, 4-5

機能, xxxviii

エンキューにおけるメッセージの優先順位および順序付け, 1-11

構造化ペイロード, 1-7

サブスクリプションおよび受信者リスト, 1-10

時刻指定, 1-12

相関識別子, 1-10

送信者識別, 1-12

遅延を伴う再試行, 1-14

追跡およびイベント・ジャーナル, 1-8

デキューにおけるメッセージのナビゲーション,  
1-13

デキューのモード, 1-13

伝播, 1-12

統合されたトランザクション, 1-8

統合データベース・レベルのサポート, 1-7

トランザクション保護のオプション, 1-14

複数の受信者, 1-13

保存およびメッセージ履歴, 1-8

メッセージ待機の最適化, 1-14

メッセージのグループ化, 1-11

例外処理, 1-14

ローカルおよびリモートの受信者, 1-13

キュー・テーブルおよびキューの作成, A-4

メッセージ・プロパティ, 2-3

アドバンスト・キューイングの複数コンシューマによる  
同一メッセージのデキュー, 7-6

アドバンスト・キューイング、基本, 7-3

アドバンスト・キューイング

ロールおよび権限の取消し, A-67

アプリケーションのサンプル, 8-3

## い

---

インタフェース - javax.jms.BytesMessage, B-17

インタフェース - javax.jms.Connection, B-18

インタフェース - javax.jms.ConnectionFactory, B-19

インタフェース - javax.jms.ConnectionMetaData, B-20

インタフェース - javax.jms.DeliveryMode, B-21

インタフェース - javax.jms.Destination, B-22

インタフェース - javax.jms.MapMessage, B-23

インタフェース - javax.jms.Message, B-24

インタフェース - javax.jms.MessageConsumer, B-25

インタフェース - javax.jms.MessageListener, B-26  
インタフェース - javax.jms.MessageProducer, B-27  
インタフェース - javax.jms.ObjectMessage, B-28  
インタフェース - javax.jms.Queue, B-29  
インタフェース - javax.jms.QueueBrowser, B-30  
インタフェース - javax.jms.QueueConnection, B-31  
インタフェース - javax.jms.QueueConnectionFactory, B-32  
インタフェース - javax.jms.QueueReceiver, B-33  
インタフェース - javax.jms.QueueSender, B-34  
インタフェース - javax.jms.QueueSession, B-35  
インタフェース - javax.jms.Session, B-36  
インタフェース - javax.jms.StreamMessage, B-37  
インタフェース - javax.jms.TextMessage, B-38  
インタフェース - javax.jms.Topic, B-39  
インタフェース - javax.jms.TopicSession, B-43  
インタフェース - javax.jms.TopicSubscriber, B-44  
インタフェース - oracle.AQ.AQQueueTable, B-87  
インタフェース - oracle.jms.AdtMessage, B-52  
インタフェース - oracle.jms.AQjmsConnectionMetadata, B-63  
インタフェース - oracle.jms.AQjmsConsumer, B-65  
インタフェース - oracle.jms.AQjmsQueueReceiver, B-53  
インタフェース - oracle.jms.AQjmsQueueSender, B-54  
インタフェース - oracle.jms.AQjmsTopicPublisher, B-55  
インタフェース - oracle.jms.AQjmsTopicReceiver, B-58  
インタフェース - oracle.jms.AQjmsTopicSubscriber, B-57  
インタフェース - oracle.jms.TopicReceiver, B-56  
インタフェース、クラスおよび例外, B-2

## え

永続サブスクリバ, 12-48  
エージェント, 2-3  
エラー・メッセージ, xlii, D-1

## お

オブジェクト型のキュー・テーブルおよびキューの作成, 9-23, A-4, 9-8  
オブジェクトのネーミング, 2-2  
オブジェクト名, 2-2

## か

開始、キュー, 9-36  
可視性, 2-6  
可用性, 5-2  
管理インタフェース, 9-1  
ビュー, 10-1  
管理インタフェースの列挙定数, 2-5  
管理インタフェース、ビュー, 10-2  
管理者ロール, 4-6

## き

期限切れ、時刻指定, 8-48  
機能  
    アドバンスト・キューイング, xxxviii  
    キュー・レベルのアクセス制御, 1-8  
    パブリッシュ / サブスクライブ・サポート, 1-9  
    非永続キュー, 1-9  
基本操作、操作インタフェース, 11-1  
キュー、Point-to-Point, 12-39  
キューイング  
    DBMS\_AQADM パッケージ, 4-4  
キュー・エンティティのモデリング, 7-2  
キュー権限の取消し, 9-50  
キュー権限の付与, 9-47, 9-50  
キュー・タイプの検証, 9-72  
キュー・テーブルおよびキューの作成, A-4  
キュー・テーブルおよびキューの作成例, A-4  
キュー・テーブル・データのエクスポート, 4-2  
キュー・テーブルの削除, 9-17  
キュー・テーブルの作成, 9-13  
キュー・テーブルの定義, 1-17  
キュー・テーブルの変更, 9-14  
キュー・テーブルのメッセージの選択, 10-21  
キューの開始, 9-36  
キューの削除, 9-33  
キューの作成, 9-20  
キューのサブスクリバおよびそのルールを選択, 10-37  
キューのサブスクリバの選択, 10-35  
キューのサブスクリバの定義, 7-6  
キューのセキュリティ, 4-4  
キューの定義, 1-17  
キューの停止, 9-39  
キューの伝播スケジュールの解除, 9-69  
キューの伝播のスケジュール, 9-65

キューの変更, 9-30  
キュー・レベルのアクセス制御, 8-10

## く

---

クラス, B-2  
クラス - AQjmsQueueConnectionFactory, B-75  
クラス, JMS, B-5  
クラス - oracle.AQ.AQQueueTableProperty, B-88  
クラス - oracle.jms.AQjmsAdtMessage, B-59  
クラス - oracle.jms.AQjmsAgent, B-60  
クラス - oracle.jms.AQjmsBytesMessage, B-61  
クラス - oracle.jms.AQjmsConnection, B-62  
クラス - oracle.jms.AQjmsConstants, B-64  
クラス - oracle.jms.AQjmsStreamMessage, B-77  
クラス - oracle.jms.AQjmsDestination, B-66  
クラス - oracle.jms.AQjmsDestinationProperty, B-67  
クラス - oracle.jms.AQjmsException, B-80  
クラス - oracle.jms.AQjmsFactory, B-68  
クラス - oracle.jms.AQjmsMapMessage, B-69  
クラス - oracle.jms.AQjmsObjectMessage, B-71  
クラス - oracle.jms.AQjmsOracleDebug, B-72  
クラス - oracle.jms.AQjmsProducer, B-73  
クラス - oracle.jms.AQjmsQueueBrowser, B-74  
クラス - oracle.jms.AQjmsSession, B-76  
クラス - oracle.jms.AQjmsTextMessage, B-78  
クラス - oracle.jms.AQjmsTopicConnectionFactory,  
B-79  
グループ化、メッセージ, 12-66

## け

---

検証、キュー・タイプ, 9-72

## こ

---

構造化ペイロード, 1-7, 8-7  
構造化ペイロードまたはメッセージ型, 12-20  
コンシューマ, 7-4

## さ

---

索引および表構造, 5-2  
削除、キュー・テーブル, 9-17  
削除、サブスクライバ, 9-62  
作成、キュー, 9-20  
作成、キュー・テーブル, 9-5

サブスクライバの削除, 9-62  
サブスクライバの追加, 9-53  
サブスクライバの変更, 9-58  
サブスクライブ・サポート, 8-25  
サブスクリプションおよび受信者リスト, 1-10, 8-35  
サブスクリプション、ルールベース, 8-93  
サンプル JMS アプリケーション, 12-1  
サンプル・アプリケーション, 8-3

## し

---

時刻指定  
    期限切れ, 8-48, 12-64  
    遅延, 8-45, 12-62  
システム権限の取消し, 9-45  
システム権限の付与, 9-42  
システム・レベルのアクセス制御, 8-5, 12-13  
指定、メッセージ・プロパティ, 2-3  
受信者, 1-18  
受信者のリストの長さの制限, 6-3  
受信者リスト, 8-35, 12-53  
受信者、複数, 8-60  
受信者、ローカルおよびリモート, 8-62  
受信におけるメッセージのナビゲーション, 12-74  
障害、伝播, 5-4  
使用可能化、伝播スケジュール, 9-79  
使用禁止、伝播スケジュール, 9-82  
状態, 2-6

## す

---

スケーラビリティ、キュー操作, 5-3  
スケジューリング伝播, 8-103  
スケジューリング伝播における wait パラメータ, 6-2  
スケジュールの解除、キューの伝播, 9-69  
スケジュール、キューの伝播, 9-65  
すべての伝播スケジュールの選択, 10-12  
スループット, 5-2

## せ

---

セキュリティ, 4-4  
設計およびモデリング, xl, 7-1  
選択、キュー・テーブルのメッセージ, 10-21  
選択、キューのサブスクライバ, 10-35  
選択、キューのサブスクライバおよびそのルール,  
10-37

選択、すべての伝播スケジュール, 10-12  
選択、データベース内のすべてのキュー・テーブル, 10-4  
選択、データベースのすべてのキュー, 10-10  
選択、ユーザーがキュー権限を持っているキュー, 10-19  
選択、ユーザーが何らかの権限を持っているキュー, 10-17  
選択、ユーザー・スキーマのキュー, 10-28  
選択、ユーザー・スキーマのキュー・テーブル, 10-25  
選択、ユーザー・スキーマの伝播スケジュール, 10-30  
選択、ユーザーのキュー・テーブル, 10-7

## そ

---

関連識別子, 1-10  
操作インタフェース - 基本操作, 11-1  
操作インタフェースの列挙定数, 2-6

## ち

---

遅延, 2-6  
遅延間隔をおいての再試行, 8-83  
遅延間隔、再試行, 8-83  
遅延、時刻指定, 12-62, 8-45

## つ

---

追加、サブスクライバ, 9-53  
通知登録とリスナー, 6-3  
通知、非同期, 8-76

## て

---

定義、エージェント, 1-17  
定義、メッセージ, 1-17  
停止、キュー, 9-39  
データベース全体、状態ごとのメッセージ数の選択, 10-39  
データベース内のすべてのキュー・テーブルの選択, 10-4  
データベースのすべてのキューの選択, 10-10  
デキューにおける ORA-1555 エラー, 6-4  
デキューにおけるメッセージのナビゲーション, 8-64  
デキューの方法, 8-55  
デキューのモード, 8-68  
デキュー、メッセージのナビゲーション, 8-64

伝播, 1-12, 8-102, 12-89  
伝播機能, 8-101  
伝播障害, 5-4  
伝播スケジュールリング, 1-15, 8-103  
伝播スケジュールリング機能の拡張, 8-109  
伝播スケジュール, 12-95  
伝播スケジュール機能の拡張, 12-97  
伝播スケジュールの使用可能化, 9-79  
伝播スケジュールの使用禁止, 9-82  
伝播スケジュールの変更, 9-75  
伝播中の例外処理, 12-99  
伝播のスケジュール, 12-95  
伝播の問題点, 5-3  
伝播、スケジュール, 1-15  
伝播、メッセージ, 4-7  
伝播、例外処理, 12-99

## と

---

同一メッセージのデキュー、複数コンシューマ, 7-6  
統計ビューのサポート, 8-33, 12-19  
特定のインスタンスにおける状態ごとのメッセージ数の選択, 10-41  
トピック・パブリッシャ, 12-51  
トピック、パブリッシュ / サブスクライブ, 12-46  
トラブルシューティング JMS エラー, xlii  
取消し、システム権限, 9-45

## な

---

ナビゲーション, 2-6

## は

---

パフォーマンス, 5-2  
パブリッシュ / サブスクライブ・サポート, 8-25

## ひ

---

非永続キュー, 9-27  
非永続キューの作成, 9-27  
非永続キューの使用方法, 6-3  
非同期通知, 8-76  
ビュー, 10-1  
ビューの属性, 10-1  
表および索引構造, 5-2

## ふ

---

複合, 7-15  
複数コンシューマによる同一メッセージのデキュー,  
7-6  
複数の受信者, 8-60  
複数のプロデューサ、1つのコンシューマ, 7-3  
複数のプロデューサ、複数のコンシューマ - 不連続  
メッセージ, 7-3  
付与、システム権限, 9-42  
プレビュー後のメッセージのデキュー, A-33  
プログラム環境, 3-2  
プロデューサ, 7-3  
プロデューサ、コンシューマ, 7-3

## へ

---

ペイロード、構造化, 8-7  
変更、キュー, 9-30  
変更、伝播スケジュール, 9-75

## ほ

---

保存, 2-5  
保存およびメッセージ履歴, 1-8, 8-23, 12-16

## ま

---

待ち状態, 2-6

## め

---

メッセージ  
プロデューサおよびコンシューマ, 1-17  
メッセージ、エラー, D-1  
メッセージ・エンキュー, 11-5  
メッセージ・システム, 1-2  
メッセージ到着待機, 8-74  
メッセージ到着待機の最適化, 8-74  
メッセージのエンキュー, 11-5, 11-7, 11-10, 11-13  
メッセージのエンキューおよびデキュー  
Pro\*C/C++を使用したRAW型, A-25, A-22  
Pro\*C/C++を使用した相関識別子およびメッセー  
ジIDによる, A-37  
RAW型, A-14, A-16, A-18  
オブジェクト型, A-11  
遅延および期限切れによる, A-36

複数コンシューマ・キューでの, A-44, A-47  
優先順位による, A-14, A-16, A-18  
メッセージのエンキューおよびデキュー例, A-11  
メッセージのグループ化, 1-11, 8-51, 12-66  
メッセージの受信, 12-71  
メッセージの受信者、定義, 7-6  
メッセージの順序付け, 8-37, 12-58  
メッセージのデキュー, 11-45  
メッセージの伝播, 4-7, 7-15  
メッセージの伝播、LOB属性, 8-107  
メッセージのファネルイン, 7-15  
メッセージのファンアウト, 7-15  
メッセージの優先順位および順序付け, 8-37, 12-58  
メッセージ・プロデューサ機能, 12-57  
メッセージ・リスナーを使用したメッセージの非同期  
受信, 12-81  
メッセージ履歴, 8-23  
メッセージ履歴および保存, 12-16  
メッセージ、ファネルイン, 7-15  
メッセージ、ファンアウト, 7-15

## も

---

モード、デキュー, 8-68  
モデリングおよび設計, xl, 7-1

## ゆ

---

ユーザーがキュー権限を持っているキューの選択,  
10-19  
ユーザーが何らかの権限を持っているキューの選択,  
10-17  
ユーザー・スキーマのキュー・テーブルの選択, 10-25  
ユーザー・スキーマのキューの選択, 10-28  
ユーザー・スキーマの伝播スケジュールの選択, 10-30  
ユーザーのキュー・テーブルの選択, 10-7  
ユーザー・ロール, 4-7  
ユースケース  
内部LOBのすべてのリスト, 11-2  
モデルの図形概要, 9-1, 11-1, 16-1  
優先順位指定メッセージのキュー・テーブルおよび  
キューの作成, A-5  
優先メッセージのキュー・テーブルおよびキューの作  
成, 9-23, 9-9  
キュー・テーブルのメッセージ、選択, 10-21

## り

---

リスナーと通知登録, 6-3  
リモートの受信者, 8-62

## る

---

ルール, 1-19  
ルールのエクスポート、キュー・テーブル, 4-2  
ルールベースのサブスクリプション, 8-93  
ルール、サブスクライバの選択, 10-37

## れ

---

例外, B-2  
例外 - javax.jms.InvalidDestinationException, B-45  
例外 - javax.jms.InvalidSelectorException, B-46  
例外 - javax.jms.JMSEException, B-47  
例外 - javax.jms.MessageNotWriteableException, B-51  
例外 - javax.jms.MessageEOFException, B-48  
例外 - javax.jms.MessageFormatException, B-49  
例外 - javax.jms.MessageNotReadableException, B-50  
例外 - oracle.jms.AQjmsInvalidDestinationException,  
B-81  
例外 - oracle.jms.AQjmsInvalidSelectorException, B-82  
例外 - oracle.jms.AQjmsMessageEOFException, B-83  
例外 - oracle.jms.AQjmsMessageFormatException,  
B-84  
例外 - oracle.jms.AQjmsMessageNotReadableException,  
B-85  
例外 - oracle.jms.AQjmsMessageNotWriteableException,  
B-86  
例外処理, 8-87  
例外処理、AQ, 12-85

## ろ

---

ローカルおよびリモートの受信者, 8-62  
ロールおよび権限の取消し (AQ), A-67  
ロール、管理者, 4-6  
ロール、ユーザー, 4-7

