

Oracle8i コール・インタフェース

プログラマーズ・ガイド Vol.2

リリース 8.1

ORACLE[®]

Oracle8i コール・インタフェース・プログラマーズ・ガイド Vol.2 リリース 8.1

部品番号 : A62735-1

第 1 版 1999 年 5 月 (第 1 刷)

原本名 : Oracle Call Interface Programmer's Guide, Volume 2, Release 8.1.5

原本部品番号 : A67847-01

原本著者 : Phil Locke

原本協力者 : Eric Belden, Ruth Baylis, Allen Brumm, Sashi Chandrasekaran, Debashish Chatterjee, Ernest Chen, Luxi Chidambaran, Sreenivas Gollapudi, Brajesh Goyal, Radhakrishna Hari, Josef Hasenberger, Don Herkimer, Chin-Heng Hong, Nancy Ikeda, Amit Jasuja, Sanjay Kaluskar, Ravi Kasamsetty, Susan Kotsovolos, Vishu Krishnamurthy, Srinath Krishnaswamy, Ramkumar Krishnan, Sanjeev Kumar, Thomas Kurian, Paul Lane, Shoaib Lari, Chon Lei, Cindy Lim, Nancy Liu, Diana Lorentz, Shailendra Mishra, Valarie Moore, Tin Nguyen, Denise Oertel, Rosanne Park, Jacqui Pons, Den Raphaely, Anindo Roy, Tim Smith, Ekrem Soylemez, Ashwini Surpur, Alan Thiessen, Peter Vasterd, Jyo Wijaya, Sathyam Yanamandram, Allen Zhao

Copyright © 1999, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラムの使用、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。

当ソフトウェア (プログラム) のリバース・エンジニアリング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 Legend が適用されます。

Restricted Rights Legend

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication and disclosure of the Programs 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 52.227-14, Rights in Data -- General, including Alternate III (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	xxvii
------------	-------

第 I 部 OCI の基本概念

1 概要、新機能およびアップグレード

OCI の概要	1-2
OCI の利点	1-3
OCI アプリケーションの構築	1-4
OCI の分類	1-5
プロシージャ型および非プロシージャ型要素	1-5
オブジェクト・サポート	1-6
SQL 文	1-7
新機能	1-12
カプセル化インタフェース	1-13
ユーザー認証およびパスワード管理の簡素化	1-13
アプリケーションのパフォーマンスおよび拡張性を改善する拡張機能	1-14
Oracle OCI オブジェクト・サポート	1-14
クライアント側のオブジェクト・キャッシュ	1-15
アソシエイティブ・インタフェースおよびナビゲーション・インタフェース	1-15
オブジェクト用のランタイム環境	1-16
型管理、マッピングおよび操作関数	1-16
オブジェクト型トランスレータ	1-17
OCI での Oracle アドバンスド・キューイングのサポート	1-17
既存のアプリケーションの移行の簡素化	1-18
互換性、アップグレードおよび移行	1-18

使用されなくなった OCI ルーチン	1-19
サポートされない OCI ルーチン	1-21
互換性.....	1-21
アップグレード.....	1-22
アプリケーションのリンクの問題.....	1-24

2 OCI プログラミングの基本

概要	2-2
OCI プログラム構造	2-3
OCI データ構造	2-5
ハンドル	2-6
ハンドルの割当てと解放.....	2-7
環境ハンドル.....	2-8
エラー・ハンドル.....	2-9
サービス・コンテキスト・ハンドルとそれに関連付けられたハンドル.....	2-9
文ハンドルおよびバインド・ハンドル、定義ハンドル.....	2-10
記述ハンドル.....	2-11
複合オブジェクト検索ハンドル.....	2-11
スレッド・ハンドル.....	2-11
サブスクリプション・ハンドル.....	2-11
ダイレクト・パス・ハンドル.....	2-12
プロセス・ハンドル.....	2-12
ハンドル属性.....	2-12
ユーザー・メモリーの割当て.....	2-13
記述子およびロケータ	2-14
スナップショット記述子.....	2-15
LOB/FILE データ型ロケータ.....	2-15
パラメータ記述子.....	2-16
ROWID 記述子.....	2-16
複合オブジェクト記述子.....	2-16
アドバンスト・キューイング記述子.....	2-16
ユーザー・メモリーの割当て.....	2-17
OCI プログラミング・ステップ	2-17
初期化および接続、セッション作成	2-18
OCI 環境の初期化	2-18

共有データ・モード.....	2-19
ハンドルおよび記述子の割当て.....	2-21
アプリケーションの初期化および接続、セッション作成.....	2-22
SQL 文の処理	2-25
コミットまたはロールバック	2-25
アプリケーションの終了	2-25
エラー処理	2-26
切捨ておよび NULL データのリターン・コードおよびエラー・コード.....	2-28
他の値を戻す関数.....	2-29
その他のコーディング・ガイドライン	2-29
パラメータの型.....	2-29
Null.....	2-30
標識変数.....	2-30
コールの取消し.....	2-32
位置づけ更新および位置づけ削除.....	2-32
予約語.....	2-33
アプリケーションのリンク.....	2-34
非ブロック化モード	2-35
ブロック化モードの設定.....	2-36
非ブロック化コールの取消し.....	2-36
非ブロック化の例.....	2-36
OCI プログラムでの PL/SQL 使用	2-37

3 データ型

Oracle データ型	3-2
内部データ型コード.....	3-3
外部データ型コード.....	3-4
内部データ型	3-4
LONG、RAW、LONG RAW、VARCHAR2.....	3-5
文字列およびバイト配列.....	3-5
ユニバーサル ROWID.....	3-5
外部データ型	3-6
VARCHAR2.....	3-8
NUMBER.....	3-9
INTEGER.....	3-10

FLOAT	3-10
STRING	3-10
VARNUM.....	3-11
LONG.....	3-12
VARCHAR	3-12
ROWID.....	3-12
DATE.....	3-12
RAW	3-13
VARRAW	3-14
LONG RAW	3-14
UNSIGNED.....	3-14
LONG VARCHAR	3-14
LONG VARRAW	3-14
CHAR.....	3-14
CHARZ	3-16
新しい Oracle 外部データ型.....	3-16
名前付きデータ型 (オブジェクト、VARRAY、ネストした表).....	3-16
REF	3-17
LOB.....	3-17
新しい C データ型マップ	3-19
データ変換.....	3-19
型コード.....	3-21
SQLT 値および OCI_TYPECODE 値の関係.....	3-23
oratypes.h の定義.....	3-25

4 SQL 文の処理

概要.....	4-2
SQL 文の処理.....	4-2
文の準備.....	4-4
複数サーバーでの準備済みの文使用方法.....	4-5
バインディング.....	4-5
文の実行.....	4-6
実行スナップショット.....	4-6
実行モード.....	4-7
OCIStmtExecute() 用のバッチ・エラー・モード.....	4-8

選択リスト項目の記述	4-10
暗黙的記述.....	4-11
問合せの明示的記述.....	4-13
定義	4-13
結果のフェッチ	4-14
LOB データのフェッチ.....	4-14
プリフェッチ・カウントの設定.....	4-14

5 バインディングと定義

バインディング	5-2
名前付きバインドおよび定位置バインド.....	5-4
OCI 配列インタフェース.....	5-4
PL/SQL のプレースホルダのバインディング.....	5-5
バインディングで使うステップ.....	5-6
PL/SQL の例.....	5-7
拡張バインド.....	5-9
拡張バインド操作	5-9
静的配列バインド.....	5-9
名前付きデータ型のバインド.....	5-9
REF のバインディング.....	5-10
LOB のバインディング.....	5-10
OCI_DATA_AT_EXEC モードでのバインド.....	5-11
Ref カーソル変数のバインディング.....	5-11
バインド情報のまとめ.....	5-12
定義	5-13
定義に使用するステップ.....	5-14
拡張定義.....	5-15
拡張定義操作	5-15
名前付きデータ型出力変数の定義.....	5-16
REF 出力変数の定義.....	5-16
LOB 出力変数の定義.....	5-16
PL/SQL 出力変数の定義.....	5-16
ピース単位フェッチの定義.....	5-16
構造体配列の定義.....	5-16
構造体の配列	5-17

スキップ・パラメータ	5-18
構造体の配列で使用される OCI コール	5-19
構造体の配列と標識変数	5-19
RETURNING 句を使用した DML	5-20
RETURNING 句を使用した DML の使用方法	5-20
RETURNING...INTO 変数のバインディング	5-21
エラー処理	5-22
RETURNING REF...INTO 句を使用した DML	5-22
コールバックに関するその他の注意	5-24
DML RETURNING 文用の配列インタフェース	5-24
NCHAR および文字変換問題	5-24
NCHAR 問題	5-24
OCI_ATTR_MAXDATA_SIZE 属性	5-25
文字カウント属性	5-26
固定幅 Unicode サポート	5-27
PL/SQL REF CURSOR およびネストしたテーブル	5-28
ランタイム・データ割当てとピース単位操作	5-30
実行時の INSERT または UPDATE データの提供	5-32
PL/SQL でのピース単位操作	5-34
実行時の FETCH 情報の提供	5-34
コールバックなしのピース単位操作に関する追加情報	5-36

6 スキーマ・メタデータの記述

概要	6-2
OCIDescribeAny() の使用	6-2
制限	6-3
型および属性の注意	6-4
パラメータ属性	6-5
表 / ビュー属性	6-7
プロシージャ / ファンクション / サブプログラム属性	6-8
パッケージ属性	6-8
型属性	6-9
型属性の属性	6-10
型メソッド属性	6-11
コレクション属性	6-13

シノニム属性.....	6-14
順序属性.....	6-14
列属性.....	6-15
引数 / 結果属性.....	6-16
リスト属性.....	6-18
スキーマ属性.....	6-19
データベース属性.....	6-19
例	6-20
表用の列データ型の検索.....	6-20
ストアド・プロシージャの記述.....	6-21
オブジェクト型の属性の検索.....	6-23
名前付きコレクション型のコレクション要素のデータ型の検索.....	6-25

7 LOB および FILE 操作

概要	7-2
ロケータ	7-2
LOB ロケータ.....	7-2
FILE ロケータ.....	7-3
内部 LOB の作成と変更	7-4
表内の FILE と OS ファイルの関連付け	7-4
オブジェクトの LOB 属性	7-5
オブジェクトの LOB 属性への書込み.....	7-5
LOB 属性を持つ一時オブジェクト.....	7-5
LOB の配列インタフェース	7-6
LOB および FILE の関数	7-6
LOB の読み込み / 書込みパフォーマンスを向上するための関数.....	7-9
LOB バッファリング関数.....	7-10
LOB のオープンおよびクローズのための関数.....	7-10
LOB 関数のサーバー往復.....	7-12
LOB 読み込みおよび書込みコールバック	7-12
ストリーム転送のコールバック・インタフェース.....	7-12
コールバックを使用して LOB を読み込む.....	7-13
コールバックを使用して LOB を書込む.....	7-14
テンポラリ LOB のサポート	7-16
テンポラリ LOB の作成および解放.....	7-17

テンポラリ LOB の継続時間.....	7-17
テンポラリ LOB の例.....	7-18

8 スケーラブルなプラットフォームの管理

概要	8-2
トランザクション	8-2
トランザクションの複雑性のレベル.....	8-3
トランザクションの例.....	8-8
関連する初期化パラメータ.....	8-10
パスワードおよびセッションの管理	8-10
認証管理.....	8-10
パスワード管理.....	8-12
セッション管理.....	8-12
スレッド・セーフティ	8-13
OCI スレッド・セーフティの利点.....	8-13
スレッド・セーフティと3層アーキテクチャ.....	8-14
マルチスレッド開発の基本概念.....	8-14
スレッド・セーフティのインプリメント.....	8-14

9 OCI プログラミングの応用

概要	9-2
OCIThread パッケージ	9-2
初期化および終了.....	9-3
パッシブ・スレッディング基本型.....	9-4
アクティブ・スレッディング基本型.....	9-7
OCIThread パッケージの使用方法.....	9-8
OCIThread の使用例.....	9-8
ユーザー定義コールバック関数	9-10
ユーザー・コールバックの登録.....	9-10
外部プロシージャからの OCI コールバック.....	9-16
アプリケーション・フェイルオーバー・コールバック	9-17
フェイルオーバー・コールバックの概要.....	9-17
フェイルオーバー・コールバック構造およびパラメータ.....	9-17
フェイルオーバー・コールバックの登録.....	9-19
フェイルオーバー・コールバックの例.....	9-19

OCI_FO_ERROR の処理	9-20
OCI およびアドバンスト・キューイング	9-24
OCI アドバンスト・キューイング関数	9-24
OCI アドバンスト・キューイングの説明	9-24
OCI のアドバンスト・キューイング対 PL/SQL	9-25
パブリッシュ - サブスクライブの通知	9-28
パブリッシュ - サブスクライブの関数	9-29
通知コールバック	9-30
パブリッシュ - サブスクライブの例	9-32
ダイレクト・パス・ロード	9-36
制限と制約	9-37
サポートされるデータ型	9-38
ダイレクト・パスのハンドル	9-38
ダイレクト・パス・インタフェースの関数	9-40
ダイレクト・パス・ロードの例	9-41

第 II 部 OCI オブジェクトの概念

10 OCI オブジェクト・リレーショナル・プログラミング

概要	10-2
OCI オブジェクトの概要	10-3
OCI でのオブジェクトの操作	10-4
基本的なオブジェクト・プログラム構造体	10-4
永続オブジェクトおよび一時オブジェクト、値	10-5
OCI オブジェクト・アプリケーションの開発	10-7
オブジェクトの C アプリケーションでの表現	10-8
環境およびオブジェクト・キャッシュの初期化	10-10
データベース接続の実行	10-10
サーバーからのオブジェクト参照の取得	10-10
オブジェクトの確保	10-11
オブジェクト属性の操作	10-13
オブジェクトのマークおよび変更のフラッシュ	10-14
埋込みオブジェクトのフェッチ	10-15
オブジェクトのメタ属性	10-16
複合オブジェクト検索	10-19

COR プリフェッチ	10-23
OCI と SQL のオブジェクトへのアクセス	10-26
確保カウントおよび確保解除	10-27
NULL	10-27
オブジェクトの作成	10-30
オブジェクトの解放およびコピー	10-31
オブジェクト参照と型参照	10-32
オブジェクト・ビューまたはユーザー定義 OID に基づいたオブジェクトの作成	10-32
オブジェクト・アプリケーションでのエラー処理	10-33

11 オブジェクト・リレーショナル・データ型

概要	11-2
Oracle データ型の C へのマッピング	11-2
OCI 型マッピングの方法論	11-4
OCI での C データ型の操作	11-5
Oracle 数値操作の精度	11-6
日付 (OCIDate)	11-6
日付変換関数	11-6
日付割当ておよび日付取得関数	11-7
日付算術および日付比較関数	11-7
日付情報アクセサ関数	11-7
日付の有効性チェック関数	11-8
日付の例	11-8
数値 (OCINumber)	11-9
数値算出関数	11-10
数値変換関数	11-10
指数関数および対数関数	11-11
三角関数	11-11
数値の代入、比較および評価関数	11-12
数値の例	11-12
固定長または可変長文字列 (OCIStrng)	11-14
文字列関数	11-14
文字列の例	11-15
ロー (OCIRaw)	11-15
ロー・ファンクション	11-16

ローの例.....	11-16
コレクション (OCI Table、OCI Array、OCI Coll、OCI Iter).....	11-17
汎用コレクション関数.....	11-17
コレクション・データ操作関数.....	11-18
コレクション・スキャン・ファンクション.....	11-18
Varray/ コレクション反復子の例.....	11-19
ネストした表の操作関数.....	11-20
ネストした表のロケータ.....	11-21
REF(OCI Ref).....	11-21
REF 操作関数.....	11-22
REF の例.....	11-22
オブジェクト型情報の格納およびアクセス.....	11-23
記述子オブジェクト.....	11-23

12 オブジェクト・アプリケーションでのバインディングおよび定義

バインディング.....	12-2
名前付きデータ型のバインド.....	12-2
REF のバインディング.....	12-3
名前付きデータ型および REF バインドの情報.....	12-3
定義.....	12-4
名前付きデータ型出力変数の定義.....	12-4
REF 出力変数の定義.....	12-4
名前付きデータ型と REF 定義、および PL/SQL OUT バインドの情報.....	12-4
Oracle C データ型のバインドおよび定義.....	12-6
バインドおよび定義の例.....	12-7
給与更新の例.....	12-9
SQLT_NTY バインド / 定義の例.....	12-12
バインドの例.....	12-12
定義の例.....	12-13

13 オブジェクトのキャッシュおよびオブジェクト・ナビゲーション

概要.....	13-2
オブジェクト・キャッシュおよびメモリー管理.....	13-2
キャッシュの一貫性およびコヒーレンス.....	13-5
オブジェクト・キャッシュ・パラメータ.....	13-5

オブジェクト・キャッシュ操作.....	13-6
オブジェクト・コピーをロードおよび削除するための操作.....	13-7
オブジェクト・コピーを変更するための操作.....	13-9
オブジェクト・コピーをサーバーと同期化するための操作.....	13-10
オブジェクトのロック操作.....	13-12
オブジェクト・アプリケーションでのコミットおよびロールバック.....	13-14
オブジェクトの継続時間.....	13-14
インスタンスのメモリー・レイアウト.....	13-17
オブジェクト・ナビゲーション	13-18
単純なオブジェクト・ナビゲーション.....	13-18
OCI ナビゲーション関数	13-20
確保 / 確保解除 / 開放関数.....	13-20
フラッシュおよびリフレッシュ関数.....	13-21
マークおよびマーク解除関数.....	13-21
オブジェクトのメタ属性アクセサ関数.....	13-21
その他の関数.....	13-22

14 オブジェクト型トランスレータの使用法

OTT 概要	14-2
オブジェクト型トランスレータの使用法	14-2
データベースでの型の作成.....	14-4
OTT の起動.....	14-4
OTT のコマンド行	14-5
OTT.....	14-5
userid.....	14-6
intype.....	14-6
outtype.....	14-6
code.....	14-6
hfile.....	14-6
initfile.....	14-7
initfunc.....	14-7
Intype ファイル	14-7
OTT データ型マッピング	14-9
NULL 標識の構造体.....	14-14
Outtype ファイル	14-15

OCI アプリケーションでの OTT の使用方法	14-16
OCI でのオブジェクトへのアクセスおよび操作	14-17
初期化関数のコール	14-18
初期化関数の作業	14-20
OTT の参照	14-20
OTT コマンド行の構文	14-21
OTT パラメータ	14-22
OTT パラメータの指定可能な場所	14-26
Intype ファイルの構造体	14-26
ネストされた #include ファイルの生成	14-28
SCHEMA_NAMES の使用方法	14-30
デフォルトの名前のマッピング	14-32
制限	14-34

第 III 部 OCI 参照

15 OCI リレーショナル関数

概要	15-2
関数の構文	15-2
OCI 関数のコール	15-3
LOB 関数のサーバー・ラウンドトリップ	15-3
アドバンスド・キューイング関数およびパブリッシュ-サブスクライブ関数	15-4
OCIAQDeq()	15-5
OCIAQEnq()	15-7
OCIAQListen()	15-18
OCISubscriptionDisable()	15-20
OCISubscriptionEnable()	15-21
OCISubscriptionPost()	15-22
OCISubscriptionRegister()	15-24
OCISubscriptionUnRegister()	15-26
ハンドル関数および記述子関数	15-27
OCIAttrGet()	15-28
OCIAttrSet()	15-29
OCIDescriptorAlloc()	15-31
OCIDescriptorFree()	15-33
OCIHandleAlloc()	15-34

OCIHandleFree()	15-37
OCIParamGet()	15-39
OCIParamSet()	15-41
バインド関数、定義関数および記述関数	15-42
OCIBindArrayOfStruct()	15-43
OCIBindByName()	15-44
OCIBindByPos()	15-48
OCIBindDynamic()	15-52
OCIBindObject()	15-56
OCIDefineArrayOfStruct()	15-58
OCIDefineByPos()	15-60
OCIDefineDynamic()	15-64
OCIDefineObject()	15-66
OCIDescribeAny()	15-68
OCIStmtGetBindInfo()	15-71
ダイレクト・パス・ロード関数	15-73
OCIDirPathAbort()	15-74
OCIDirPathColArrayEntryGet()	15-75
OCIDirPathColArrayEntrySet()	15-77
OCIDirPathColArrayRowGet()	15-79
OCIDirPathColArrayReset()	15-80
OCIDirPathColArrayToStream()	15-81
OCIDirPathFinish()	15-83
OCIDirPathLoadStream()	15-84
OCIDirPathPrepare()	15-86
OCIDirPathStreamReset()	15-87
接続、認証および初期化関数	15-88
OCIEnvCreate()	15-89
OCIEnvInit()	15-92
OCIInitialize()	15-94
OCILogoff()	15-97
OCILogon()	15-98
OCIServerAttach()	15-100
OCIServerDetach()	15-102
OCISessionBegin()	15-103
OCISessionEnd()	15-106
OCITerminate()	15-107
LOB 関数	15-108

OCIDurationBegin()	15-110
OCIDurationEnd()	15-111
OCILobAppend()	15-112
OCILobAssign()	15-113
OCILobCharSetForm()	15-115
OCILobCharSetId()	15-116
OCILobClose()	15-117
OCILobCopy()	15-118
OCILobCreateTemporary()	15-120
OCILobDisableBuffering()	15-122
OCILobEnableBuffering()	15-123
OCILobErase()	15-124
OCILobFileClose()	15-125
OCILobFileCloseAll()	15-126
OCILobFileExists()	15-127
OCILobFileGetName()	15-128
OCILobFileIsOpen()	15-130
OCILobFileOpen()	15-131
OCILobFileSetName()	15-132
OCILobFlushBuffer()	15-134
OCILobFreeTemporary()	15-136
OCILobGetChunkSize()	15-137
OCILobGetLength()	15-139
OCILobIsEqual()	15-140
OCILobIsOpen()	15-141
OCILobIsTemporary()	15-143
OCILobLoadFromFile()	15-144
OCILobLocatorAssign()	15-146
OCILobLocatorIsInit()	15-148
OCILobOpen()	15-149
OCILobRead()	15-151
OCILobTrim()	15-155
OCILobWrite()	15-156
OCILobWriteAppend()	15-160
文関数	15-163
OCISstmtExecute()	15-164
OCISstmtFetch()	15-167
OCISstmtGetPieceInfo()	15-168

OCISstmtPrepare()	15-170
OCISstmtSetPieceInfo()	15-172
スレッド管理関数	15-174
OCIThreadClose()	15-176
OCIThreadCreate()	15-177
OCIThreadHandleGet()	15-179
OCIThreadHndDestroy()	15-180
OCIThreadHndInit()	15-181
OCIThreadIdDestroy()	15-182
OCIThreadIdGet()	15-183
OCIThreadIdInit()	15-184
OCIThreadIdNull()	15-185
OCIThreadIdSame()	15-186
OCIThreadIdSet()	15-187
OCIThreadIdSetNull()	15-188
OCIThreadInit()	15-189
OCIThreadIsMulti()	15-190
OCIThreadJoin()	15-191
OCIThreadKeyDestroy()	15-192
OCIThreadKeyGet()	15-193
OCIThreadKeyInit()	15-194
OCIThreadKeySet()	15-195
OCIThreadMutexAcquire()	15-196
OCIThreadMutexDestroy()	15-197
OCIThreadMutexInit()	15-198
OCIThreadMutexRelease()	15-199
OCIThreadProcessInit()	15-200
OCIThreadTerm()	15-201
トランザクション関数	15-202
OCITransCommit()	15-203
OCITransDetach()	15-206
OCITransForget()	15-207
OCITransPrepare()	15-208
OCITransRollback()	15-209
OCITransStart()	15-210
その他の関数	15-218
OCIBreak()	15-219
OCIErrorGet()	15-220

OCILdaToSvcCtx()	15-222
OCIPasswordChange().....	15-223
OCIReset()	15-225
OCIServerVersion().....	15-226
OCISvcCtxToLda()	15-227
OCIUserCallbackGet().....	15-228
OCIUserCallbackRegister().....	15-230

16 OCI ナビゲーション関数と型関数

概要	16-2
オブジェクト型と存続期間.....	16-2
用語.....	16-3
関数の構文.....	16-4
ナビゲーション関数の戻り値.....	16-5
キャッシュおよびオブジェクト関数のためのサーバー往復.....	16-5
ナビゲーション関数エラー・コード.....	16-5
OCI フラッシュまたはリフレッシュ関数	16-8
OCICacheFlush()	16-9
OCICacheRefresh()	16-11
OCIObjectFlush().....	16-13
OCIObjectRefresh()	16-14
OCI オブジェクトおよびキャッシュへのマーク設定 / 解除関数	16-16
OCICacheUnmark()	16-17
OCIObjectMarkDelete().....	16-18
OCIObjectMarkDeleteByRef()	16-19
OCIObjectMarkUpdate().....	16-20
OCIObjectUnmark().....	16-22
OCIObjectUnmarkByRef()	16-23
OCI オブジェクト・ステータス取得関数	16-24
OCIObjectExists()	16-25
OCIObjectGetProperty().....	16-26
OCIObjectIsDirty()	16-30
OCIObjectIsLocked()	16-31
その他の OCI オブジェクト関数	16-32
OCIObjectCopy().....	16-33
OCIObjectGetAttr().....	16-35
OCIObjectGetInd()	16-37

OCIObjectGetObjectRef()	16-38
OCIObjectGetTypeRef()	16-39
OCIObjectLock()	16-40
OCIObjectLockNoWait()	16-41
OCIObjectNew()	16-42
OCIObjectSetAttr()	16-45
OCI 確保、確保解除および解放関数	16-47
OCICacheFree()	16-48
OCICacheUnpin()	16-49
OCIObjectArrayPin()	16-50
OCIObjectFree()	16-52
OCIObjectPin()	16-54
OCIObjectPinCountReset()	16-57
OCIObjectPinTable()	16-59
OCIObjectUnpin()	16-61
OCI 型情報アクセサ関数	16-63
OCITypeArrayByName()	16-64
OCITypeArrayByRef()	16-67
OCITypeByName()	16-69
OCITypeByRef()	16-71

17 OCI データ型マッピング関数および操作関数

概要	17-2
関数の構文	17-2
データ型マッピング関数および操作関数の戻り値	17-3
その他の値を戻す関数	17-3
データ型マッピング関数および操作関数のサーバー往復	17-3
例	17-4
OCI コレクションおよび反復関数	17-5
OCICollAppend()	17-6
OCICollAssign()	17-7
OCICollAssignElem()	17-8
OCICollGetElem()	17-10
OCICollIsLocator()	17-13
OCICollMax()	17-14
OCICollSetUpdateStatus()	17-15
OCICollSize()	17-16

OCICollTrim()	17-18
OCIIterCreate()	17-19
OCIIterDelete()	17-20
OCIIterGetCurrent()	17-21
OCIIterInit()	17-22
OCIIterNext()	17-23
OCIIterPrev()	17-25
OCI 日付関数	17-27
OCIDateAddDays()	17-28
OCIDateAddMonths()	17-29
OCIDateAssign()	17-30
OCIDateCheck()	17-31
OCIDateCompare()	17-33
OCIDateDaysBetween()	17-34
OCIDateFromText()	17-35
OCIDateGetDate()	17-37
OCIDateGetTime()	17-38
OCIDateLastDay()	17-39
OCIDateNextDay()	17-40
OCIDateSetDate()	17-41
OCIDateSetTime()	17-42
OCIDateSysDate()	17-43
OCIDateToText()	17-44
OCIDateZoneToZone()	17-46
OCI 数関数	17-48
OCINumberAbs()	17-50
OCINumberAdd()	17-51
OCINumberArcCos()	17-52
OCINumberArcSin()	17-53
OCINumberArcTan()	17-54
OCINumberArcTan2()	17-55
OCINumberAssign()	17-56
OCINumberCeil()	17-57
OCINumberCmp()	17-58
OCINumberCos()	17-59
OCINumberDec()	17-60
OCINumberDiv()	17-61
OCINumberExp()	17-62

OCINumberFloor()	17-63
OCINumberFromInt()	17-64
OCINumberFromReal()	17-65
OCINumberFromText()	17-66
OCINumberHypCos()	17-68
OCINumberHypSin()	17-69
OCINumberHypTan()	17-70
OCINumberInc()	17-71
OCINumberIntPower()	17-72
OCINumberIsInt()	17-73
OCINumberIsZero()	17-74
OCINumberLn()	17-75
OCINumberLog()	17-76
OCINumberMod()	17-77
OCINumberMul()	17-78
OCINumberNeg()	17-79
OCINumberPower()	17-80
OCINumberPrec()	17-81
OCINumberRound()	17-82
OCINumberSetPi()	17-83
OCINumberSetZero()	17-84
OCINumberShift()	17-85
OCINumberSign()	17-86
OCINumberSin()	17-87
OCINumberSqrt()	17-88
OCINumberSub()	17-89
OCINumberTan()	17-90
OCINumberToInt()	17-91
OCINumberToReal()	17-92
OCINumberToText()	17-93
OCINumberTrunc()	17-95
OCI □-関数	17-96
OCIRawAllocSize()	17-97
OCIRawAssignBytes()	17-98
OCIRawAssignRaw()	17-99
OCIRawPtr()	17-100
OCIRawResize()	17-101
OCIRawSize()	17-102

OCI REF 関数	17-103
OCIRefAssign().....	17-104
OCIRefClear()	17-105
OCIRefFromHex().....	17-106
OCIRefHexSize()	17-107
OCIRefsEqual().....	17-108
OCIRefsNull().....	17-109
OCIRefToHex().....	17-110
OCI 文字列関数	17-111
OCIStringAllocSize().....	17-112
OCIStringAssign()	17-113
OCIStringAssignText()	17-114
OCIStringPtr().....	17-115
OCIStringResize().....	17-116
OCIStringSize()	17-117
OCI 表関数	17-118
OCITableDelete().....	17-119
OCITableExists().....	17-120
OCITableFirst()	17-121
OCITableLast().....	17-122
OCITableNext()	17-123
OCITablePrev().....	17-124
OCITableSize().....	17-125

18 OCI 外部プロシージャ関数

概要	18-2
関数の構文.....	18-2
リターン・コード.....	18-3
With_Context 型.....	18-3
OCI 外部プロシージャ関数	18-4
OCIExtProcAllocCallMemory()	18-5
OCIExtProcRaiseExcp().....	18-6
OCIExtProcRaiseExcpWithMsg()	18-7
OCIExtProcGetEnv().....	18-8

第 IV 部 付録

A ハンドルおよび記述子の属性

表記法.....	A-2
環境ハンドルの属性.....	A-3
OCI_ATTR_CACHE_ARRAYFLUSH.....	A-3
エラー・ハンドルの属性.....	A-6
サービス・コンテキスト・ハンドルの属性.....	A-7
サーバー・ハンドルの属性.....	A-9
ユーザー・セッション・ハンドルの属性.....	A-12
トランザクション・ハンドルの属性.....	A-13
文ハンドルの属性.....	A-14
バインド・ハンドルの属性.....	A-20
定義ハンドルの属性.....	A-22
記述ハンドルの属性.....	A-24
パラメータ記述子の属性.....	A-24
LOB ロケータの属性.....	A-25
複合オブジェクトの属性.....	A-25
複合オブジェクト検索ハンドルの属性.....	A-25
複合オブジェクト検索記述子の属性.....	A-26
アドバンスト・キューイング記述子の属性.....	A-27
OCIAQEnqOptions 記述子の属性.....	A-27
OCIAQDeqOptions 記述子の属性.....	A-28
OCIAQMsgProperties 記述子の属性.....	A-31
OCIAQAgent 記述子の属性.....	A-35
サブスクリプション・ハンドルの属性.....	A-37
ダイレクト・パス・ロード・ハンドルの属性.....	A-39
ダイレクト・パス・コンテキスト・ハンドルの属性.....	A-39
ダイレクト・パス列配列ハンドルの属性.....	A-42
ダイレクト・パス・ストリーム・ハンドルの属性.....	A-43
ダイレクト・パス列パラメータの属性.....	A-44
プロセス・ハンドルの属性.....	A-49

B OCI デモ・プログラム

C OCI 関数のサーバー・ラウンドトリップ

概要.....	C-2
リレーショナル関数によるラウンドトリップ.....	C-2
LOB 関数によるラウンドトリップ.....	C-3
オブジェクトおよびキャッシュ関数によるラウンドトリップ.....	C-4
記述操作によるラウンドトリップ.....	C-6
データ型のマップおよび操作関数によるラウンドトリップ.....	C-6
その他のローカル機能.....	C-7

索引

第 III 部

OCI 参照

このマニュアルの第 3 部には、次の OCI 機能参照の章が含まれています。

- 第 15 章の「OCI リレーショナル関数」
- 第 16 章の「OCI ナビゲーション関数と型関数」
- 第 17 章の「OCI データ型マッピング関数および操作関数」
- 第 18 章の「OCI 外部プロシージャ関数」

関連項目 : NLS 環境に適用される OCI 関数の詳細は、『Oracle8i NLS ガイド』を参照してください。カートリッジ・サービスに適用される OCI 関数の詳細は、『Oracle8i データ・カートリッジ開発者ガイド』を参照してください。

15

OCI リレーショナル関数

この章では、C 言語対応の Oracle OCI リレーショナル関数について説明します。各関数コールについて詳細に説明するとともに、ご使用のアプリケーションで OCI 関数をコールする方法についても解説します。この章には次の項があります。

- 概要
- アドバンスト・キューイング関数およびパブリッシュ-サブスクライブ関数
- ハンドル関数および記述子関数
- バインド関数、定義関数および記述関数
- ダイレクト・パス・ロード関数
- 接続、認証および初期化関数
- LOB 関数
- 文関数
- スレッド管理関数
- トランザクション関数
- その他の関数

概要

この章では、OCI リレーショナル関数コールについて説明します。この章で説明する関数コールは、ベーシック OCI におけるコールです。オブジェクトを操作するための関数コールは、この後の 3 つの章で説明します。リターン・コードおよびエラー処理の詳細は、2-26 ページの「[エラー処理](#)」を参照してください。

関数の構文

各関数について、次の情報のリストを示します。

用途

この関数によって実行されるアクションの簡単な説明です。

構文

この関数をコールするための構文を示すコードの断片で、パラメータの順序と種類が含まれています。

パラメータ

各関数のパラメータの説明です。これにはパラメータのモードが含まれます。入力可能なパラメータのモードには、次の 3 つの値があります。

モード	説明
IN	OCI にデータを渡すパラメータ
OUT	このコールで OCI からデータを受信するパラメータ
IN/OUT	このコールでデータを渡し、このコールまたは後続のコールからの戻りでデータを受信するパラメータ

コメント

この関数に関する詳細情報（ある場合）です。関数の使用上の制約やアプリケーション内でこの関数を使用するとき役に立つ情報が記載されています。

例

説明の対象となっている関数コールの使用方法を例示するコード（全体または一部）です。すべての関数に例が記載されているわけではありません。

関連関数

関連する関数コールの一覧です。

OCI 関数のコール

OCIのこれまでのバージョンとは異なり、リリース 8i では、NULL で終わる文字列の文字列長パラメータに -1 を渡すことはできません。

文字列長をパラメータとして渡すときは、NULL ターミネータ・バイト型を組み込まないでください。OCI では、文字列は NULL で終了するとはみなされません。

LOB 関数のサーバー・ラウンドトリップ

個々の OCI LOB 関数に必要なサーバー・ラウンドトリップ回数については、[付録 C の「OCI 関数のサーバー・ラウンドトリップ」](#)の表を参照してください。

アドバンスト・キューイング関数およびパブリッシュ・サブスクライブ関数

この項では、アドバンスト・キューイング関数およびパブリッシュ・サブスクライブ関数について説明します。

表 15-1 OCI クイック・リファレンス

関数	用途	ページ
OCIAQDeq()	アドバンスト・キューイング・デキュー。	15-5 ページ
OCIAQEnq()	アドバンスト・キューイング・エンキュー。	15-7 ページ
OCIAQListen()	リストのエージェントの代理として 1 つ以上のキューをリスニングします。	15-18 ページ
OCISubscriptionEnable()	サブスクリプションについての通知を使用可能にします。	15-21 ページ
OCISubscriptionPost()	通知を受信するサブスクリプションに転記します。	15-22 ページ
OCISubscriptionRegister()	サブスクリプションを登録します。	15-24 ページ
OCISubscriptionUnRegister()	サブスクリプションの登録を解除します。	15-26 ページ

OCIAQDeq()

用途

このコールは、OCI を使ったアドバンスト・キューイング・デキュー操作のために使います。

構文

```
sword OCIAQDeq ( OCISvcCtx          *svch,
                 OCIError           *errh,
                 text                *queue_name,
                 OCIAQDeqOptions     *dequeue_options,
                 OCIAQMsgProperties   *message_properties,
                 OCIType             *payload_tdo,
                 dvoid               **payload,
                 dvoid               **payload_ind,
                 OCIRaw              **msgid,
                 ub4                  flags );
```

パラメータ

svch (IN)

OCI サービス・コンテキストです。

errh (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

queue_name (IN)

デキュー操作のためのターゲット・キューです。

dequeue_options (IN)

デキュー操作のオプションです。OCIAQDeqOptions 記述子に格納されています。

message_properties (OUT)

メッセージ用のメッセージ・プロパティです。OCIAQMsgProperties 記述子に格納されません。

payload_tdo (IN)

オブジェクトタイプの TDO (型記述子オブジェクト) です。ロー・キューでは、このパラメータが SYS.RAW の TDO を指していなければなりません。

payload (IN/OUT)

オブジェクト型のインスタンスであるプログラム変数バッファのポインタへのポインタです。ロー・キューでは、このパラメータが OCIRaw のインスタンスを指していなければなりません。

payload 用のメモリーは、オブジェクト・キャッシュに動的に割り当てられます。payload インスタンスが不要になったときは、アプリケーションから `OCIObjectFree()` をコールし、割り当てを解除することもできます。プログラム変数バッファのポインタ (`*payload`) が NULL として渡された場合、バッファは暗黙のうちにキャッシュに割り当てられます。

アプリケーションによっては、`OCIAQDeq()` が最初にコールされたときに `payload` の値として NULL が渡され、OCI により payload にメモリーが割り当てられます。そしてその後の `OCIAQDeq()` コールでは、前に割り当てられたメモリーのポインタが使われます。

payload のために TDO を取得するには、`OCITypeByName()` または `OCITypeByRef()` を使います。

OCI によって、ユーザーが payload の属性 (テキストなど) を設定できるようにする関数が提供されます。これらの属性の設定については、10-13 ページの「[オブジェクト属性の操作](#)」を参照してください。

payload_ind (IN/OUT)

オブジェクト型のパラレル標識情報構造を含むプログラム変数バッファ・ポインタへのポインタです。

`payload_ind` へのメモリー割当てルールは、上記の `payload` に対するルールと同じです。

msgid (OUT)

メッセージ ID。

flags (IN)

現行では使用されていません。OCI_DEFAULT として渡されます。

コメント

このコールを使うには、ユーザーが `aq_user_role` または `dbms_aq` パッケージを実行する権限を所有していなければなりません。このコールを使うには、OCI 環境をオブジェクト・モードで初期化する必要があります (`OCIInitialize()` を使用)。

OCI およびアドバンスト・キューイングの詳細は、9-24 ページの「[OCI およびアドバンスト・キューイング](#)」を参照してください。

高度なキューについての追加情報は、『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』を参照してください。

例

コード例については、15-7 ページの `OCIAQEnq()` の説明を参照してください。

関連関数

[OCIAQEnq\(\)](#)、[OCIAQListen\(\)](#)、[OCIInitialize\(\)](#)

OCIAQEnq()

用途

このコールは、アドバンスト・キューイング・エンキューに使用します。

構文

```
sword OCIAQEnq ( OCISvcCtx          *svch,
                 OCIError           *errh,
                 text                *queue_name,
                 OCIAQEnqOptions    *enqueue_options,
                 OCIAQMsgProperties *message_properties,
                 OCIType             *payload_tdo,
                 dvoid              **payload,
                 dvoid              **payload_ind,
                 OCIRaw              **msgid,
                 ub4                 flags );
```

パラメータ

svch (IN)

OCI サービス・コンテキストです。

errh (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

queue_name (IN)

エンキュー操作のためのターゲット・キューです。

enqueue_options (IN)

エンキュー操作のオプション。OCIAQEnqOptions 記述子に格納されています。

message_properties (IN)

メッセージ用のメッセージ・プロパティ。OCIAQMsgProperties 記述子に格納されます。

payload_tdo (IN)

オブジェクトタイプの TDO (型記述子オブジェクト) です。ロー・キューでは、このパラメータが SYS.RAW の TDO を指していなければなりません。

payload (IN)

オブジェクト型のインスタンスのポインタへのポインタです。ロー・キューでは、このパラメータが OCIRaw のインスタンスを指していなければなりません。

OCIによって、ユーザーが payload の属性（テキストなど）を設定できるようにする関数が提供されます。これらの属性の設定については、10-13 ページの「[オブジェクト属性の操作](#)」を参照してください。

payload_ind (IN)

オブジェクト型のパラレル標識情報構造を含むプログラム変数バッファ・ポインタへのポインタです。

msgid (OUT)

メッセージ ID です。

flags (IN)

現行では使用されていません。OCI_DEFAULT として渡されます。

コメント

このコールを使用するには、ユーザーが `aq_user_role` または `dbms_aq` パッケージを実行する権限を所有していなければなりません。

このコールを使うには、OCI 環境をオブジェクト・モードで初期化する必要があります (`OCIInitialize()` を使用)。

OCI およびアドバンスト・キューイングの詳細は、9-24 ページの「[OCI およびアドバンスト・キューイング](#)」を参照してください。

高度なキューについての追加情報は、『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』を参照してください。

payload のために TDO を取得するには、`OCITypeByName()` または `OCITypeByRef()` を使います。

例

次に 4 つの異なる状況を想定し、それぞれについて `OCIAQEnq()` と `OCIAQDeq()` を使用する方法を説明します。

これらの例では、データベースが、『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』の「アドバンスト・キューイング」の章の「Oracle アドバンスト・キューイング事例」での説明どおりに設定されているものと想定しています。

例 1

payload オブジェクトのエンキューおよびデキュー

```
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;
    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 **) &svchp, (ub4) OCI_HTYPE_SERVER,
                    52, (dvoid **) &tmp);

    OCIErrorAttach( svchp, 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 *) svchp, (ub4) 0,
                (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

    OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

    /* obtain TDO of message_type */
    OCITypeByName(envhp, errhp, svchp, (CONST text *) "AQ", strlen("AQ"),

```

```

        (CONST text *)"MESSAGE_TYPE", strlen("MESSAGE_TYPE"),
        (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* prepare the message payload */
mesg->subject = (OCIString *)0;
mesg->data = (OCIString *)0;
OCIStringAssignText(envhp, errhp, (CONST text *)"NORMAL MESSAGE",
                    strlen("NORMAL MESSAGE"), &mesg->subject);
OCIStringAssignText(envhp, errhp, (CONST text *)"OCI ENQUEUE",
                    strlen("OCI ENQUEUE"), &mesg->data);
nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* enqueue into the msg_queue */
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* dequeue from the msg_queue */
OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0);
printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
OCITransCommit(svchp, errhp, (ub4) 0);
}

```

例 2**相関 ID を使ったエンキューおよびデキュー**

```

struct message
{
    OCIString    *subject;
    OCIString    *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;

```

```

OCIError *errhp;
OCISvcCtx *svchp;
dvoid *tmp;
OCIType *mesg_tdo = (OCIType *) 0;
message msg;
null_message nmsg;
message *mesg = &nmsg;
null_message *nmesg = &nmsg;
message *deqmesg = (message *) 0;
null_message *ndeqmesg = (null_message *) 0;
OCIRaw*firstmsg = (OCIRaw *) 0;
OCIAQMsgProperties *msgprop = (OCIAQMsgProperties *) 0;
OCIAQDeqOptions *deqopt = (OCIAQDeqOptions *) 0;
text correlation1[30], correlation2[30];

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);

/* allocate message properties descriptor */
OCIDescriptorAlloc(envhp, (dvoid **) &msgprop,
                   OCI_DTYPE_AQMSG_PROPERTIES, 0, (dvoid **) 0);
strcpy(correlation1, "1st message");
OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES, (dvoid *) &correlation1,
           strlen(correlation1), OCI_ATTR_CORRELATION, errhp);

/* obtain TDO of message_type */
OCITypeByName(envhp, errhp, svchp, (CONST text *) "AQ", strlen("AQ"),
              (CONST text *) "MESSAGE_TYPE", strlen("MESSAGE_TYPE"),
              (text *) 0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

```

```
/* prepare the message payload */
msg->subject = (OCIStrng *)0;
msg->data = (OCIStrng *)0;
OCIStrngAssignText(envhp, errhp, (CONST text *)"NORMAL ENQUEUE1",
    strlen("NORMAL ENQUEUE1"), &msg->subject);
OCIStrngAssignText(envhp, errhp, (CONST text *)"OCI ENQUEUE",
    strlen("OCI ENQUEUE"), &msg->data);
nmsg->null_adt = nmsg->null_subject = nmsg->null_data = OCI_IND_NOTNULL;

/* enqueue into the msg_queue, store the message id into firstmsg */
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, msgprop,
    msg_tdo, (dvoid **)&msg, (dvoid **)&nmsg, &firstmsg, 0);

/* enqueue into the msg_queue with a different correlation id */
strcpy(correlation2, "2nd message");
OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES, (dvoid*)&correlation2,
    strlen(correlation2), OCI_ATTR_CORRELATION, errhp);
OCIStrngAssignText(envhp, errhp, (CONST text *)"NORMAL ENQUEUE2",
    strlen("NORMAL ENQUEUE2"), &msg->subject);
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, msgprop,
    msg_tdo, (dvoid **)&msg, (dvoid **)&nmsg, 0, 0);

OCITransCommit(svchp, errhp, (ub4) 0);

/* first dequeue by correlation id "2nd message" */
/* allocate dequeue options descriptor and set the correlation option */
OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
    OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)&0);
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)correlation2,
    strlen(correlation2), OCI_ATTR_CORRELATION, errhp);

/* dequeue from the msg_queue */
OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", deqopt, 0,
    msg_tdo, (dvoid **)&deqmsg, (dvoid **)&ndeqmsg, 0, 0);
printf("Subject: %s\n", OCIStrngPtr(envhp, deqmsg->subject));
printf("Text: %s\n", OCIStrngPtr(envhp, deqmsg->data));
OCITransCommit(svchp, errhp, (ub4) 0);

/* second dequeue by message id */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&firstmsg,
    OCIRawSize(envhp, firstmsg), OCI_ATTR_DEQ_MSGID, errhp);
/* clear correlation id option */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
    (dvoid *)correlation2, 0, OCI_ATTR_CORRELATION, errhp);

/* dequeue from the msg_queue */
```

```

OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", deqopt, 0,
         msg_tdo, (dvoid **)&deqmesg, (dvoid **)&deqmesg, 0, 0);
printf("Subject: %s\n", OCIStrPtr(envhp, deqmesg->subject));
printf("Text: %s\n", OCIStrPtr(envhp, deqmesg->data));
OCITransCommit(svchp, errhp, (ub4) 0);
}

```

例 3

ロー・キューのエンキューおよびデキュー

```

int main()
{
    OCIEnv *envhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    dvoid *tmp;
    OCIType *msg_tdo = (OCIType *) 0;
    char msg_text[100];
    OCIRaw *mesg = (OCIRaw *) 0;
    OCIRaw*deqmesg = (OCIRaw *) 0;
    OCIInd ind = 0;
    dvoid *indptr = (dvoid *)&ind;
    inti;

    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 **) &svchp, (ub4) OCI_HTYPE_SERVER,
                   52, (dvoid **) &tmp);

    OCIErrorAttach( svchp, 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 *) svchp, (ub4) 0,
               (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

    OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

    /* obtain the TDO of the RAW data type */
}

```

```

OCITypeByName(envhp, errhp, svchp, (CONST text *)"SYS", strlen("SYS"),
              (CONST text *)"RAW", strlen("RAW"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* prepare the message payload */
strcpy(msg_text, "Enqueue to a RAW queue");
OCIRawAssignBytes(envhp, errhp, msg_text, strlen(msg_text), &mesg);

/* enqueue the message into raw_msg_queue */
OCIAQEnq(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&mesg, (dvoid **)&indptr, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* dequeue the same message into C variable deqmesg */
OCIAQDeq(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&indptr, 0, 0);
for (i = 0; i < OCIRawSize(envhp, deqmesg); i++)
    printf("%c", *(OCIRawPtr(envhp, deqmesg) + i));
OCITransCommit(svchp, errhp, (ub4) 0);
}

```

例 4**OCIAQAgent を使ったエンキューおよびデキュー**

```

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;
    OCIError *errhp;
    OCISvcCtx *svchp;
    dvoid *tmp;
    OCIType *mesg_tdo = (OCIType *) 0;

```

```

message msg;
null_message nmsg;
message *mesg = &nmsg;
null_message *nmesg = &nmsg;
message *deqmesg = (message *)0;
null_message *ndeqmesg = (null_message *)0;
OCIAQMsgProperties *msgprop = (OCIAQMsgProperties *)0;
OCIAQAgent *agents[2];
OCIAQDeqOptions *deqopt = (OCIAQDeqOptions *)0;
ub4wait = 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);

/* obtain TDO of message_type */
OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
              (CONST text *)"MESSAGE_TYPE", strlen("MESSAGE_TYPE"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* prepare the message payload */
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("mesg for queue subscribers"), &mesg->data);
nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* enqueue MESSAGE 1 for subscribers to the queue i.e. for RED and GREEN */
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, 0,
    mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

/* enqueue MESSAGE 2 for specified recipients i.e. for RED and BLUE */
/* prepare message payload */
OCIStringAssignText(envhp, errhp,
    (CONST text *)"MESSAGE 2", strlen("MESSAGE 2"),
    &mesg->subject);
OCIStringAssignText(envhp, errhp,
    (CONST text *)"mesg for two recipients",
    strlen("mesg for two recipients"), &mesg->data);

/* allocate AQ message properties and agent descriptors */
OCIDescriptorAlloc(envhp, (dvoid **)&msgprop,
    OCI_DTYPE_AQMSG_PROPERTIES, 0, (dvoid **)0);
OCIDescriptorAlloc(envhp, (dvoid **)&agents[0],
    OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
OCIDescriptorAlloc(envhp, (dvoid **)&agents[1],
    OCI_DTYPE_AQAGENT, 0, (dvoid **)0);

/* prepare the recipient list, RED and BLUE */
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);

/* now dequeue the messages using different consumer names */
/* allocate dequeue options descriptor to set the dequeue options */
OCIDescriptorAlloc(envhp, (dvoid **)&deqopt, OCI_DTYPE_AQDEQ_OPTIONS, 0,
    (dvoid **)0);

/* set wait parameter to NO_WAIT so that the dequeue returns immediately */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&wait, 0,
    OCI_ATTR_WAIT, errhp);

/* set navigation to FIRST_MESSAGE so that the dequeue resets the position */
```

```

/* after a new consumer_name is set in the dequeue options */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&navigation, 0,
           OCI_ATTR_NAVIGATION, errhp);

/* dequeue from the msg_queue_multiple as consumer BLUE */
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", OCIStringPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);

/* dequeue from the msg_queue_multiple as consumer RED */
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", OCIStringPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);

/* dequeue from the msg_queue_multiple as consumer GREEN */
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", OCIStringPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);
}

```

関連関数

[OCIAQDeq\(\)](#)、[OCIAQListen\(\)](#)、[OCIInitialize\(\)](#)

OCIAQListen()

用途

リストのエージェントのために 1 つまたは複数のキューをリスニングします。

構文

```
sword OCIAQListen (OCISvcCtx      *svchp,  
                  OCIError       *errhp,  
                  OCIAQAgent     **agent_list,  
                  ub4            num_agents,  
                  sb4            wait,  
                  OCIAQAgent     **agent,  
                  ub4            flags);
```

パラメータ

svchpp (IN/OUT)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

agent_list (IN)

メッセージを監視するエージェントのリストです。

num_agents (IN)

エージェント・リスト内のエージェント数です。

wait (IN)

リスニング・コールのタイムアウトです。

agent (OUT)

メッセージの出力先のエージェントです。OCIAgent は OCI 記述子です。

flags (IN)

現行では使用されていません。OCI_DEFAULT として渡されます。

コメント

メッセージが存在し、リスト内のエージェントに対してそのメッセージを使う準備ができたときに返されるブロッキング・コールです。待機時間が終了してもメッセージが検出されない場合は、エラーが返されます。

関連関数

[OCIAQEnq\(\)](#)、[OCIAQDeq\(\)](#)、[OCISvcCtxToLda\(\)](#)、[OCISubscriptionEnable\(\)](#)、
[OCISubscriptionPost\(\)](#)、[OCISubscriptionRegister\(\)](#)、[OCISubscriptionUnRegister\(\)](#)

OCISubscriptionDisable()

用途

サブスクリプションの登録を無効にして、すべての通知をオフにします。

構文

```
ub4 OCISubscriptionDisable ( OCISubscription  *subscrhp,  
                             OCIError         *errhp  
                             ub4             mode );
```

パラメータ

subscrhp (IN)

OCI_ATTR_SUBSCR_NAME および OCI_ATTR_SUBSCR_NAMESPACE 属性が設定されたサブスクリプション・ハンドルです。詳細は、A-37 ページの「[サブスクリプション・ハンドルの属性](#)」を参照してください。

errhp (OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

mode (IN)

コール固有モードです。有効な値は次のとおりです。

- OCI_DEFAULT - デフォルトのコールを実行します。サブスクリプションが有効になるまで、このサブスクリプションの通知をすべて破棄します。

コメント

このコールは、一時的に通知をオフにするときに使います。コードの重要なセクションを実行しているときに、アプリケーションが中断されないようにするには、この関数を使いません。

この操作を実行するときは、接続または認証は必要ありません。このコールを実行する前に、サブスクリプション・ハンドルによって指定されたサブスクリプションに対して登録が行われていなければなりません。

OCISubscriptionDisable() を実行した後は、*OCISubscriptionEnable()* を実行するまですべての通知が破棄されます。

関連関数

[OCIAQListen\(\)](#)、[OCISubscriptionEnable\(\)](#)、[OCISubscriptionPost\(\)](#)、[OCISubscriptionRegister\(\)](#)、[OCISubscriptionUnRegister\(\)](#)

OCISubscriptionEnable()

用途

無効にしたサブスクリプションの登録を有効にします。すべての通知がオンになります。

構文

```
ub4 OCISubscriptionEnable ( OCISubscription *subscrhp,  
                             OCIError *errhp  
                             ub4 mode );
```

パラメータ

subscrhp (IN)

OCI_ATTR_SUBSCR_NAME および OCI_ATTR_SUBSCR_NAMESPACE 属性が設定されたサブスクリプション・ハンドルです。詳細は、A-37 ページの「[サブスクリプション・ハンドルの属性](#)」を参照してください。

errhp (OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

mode (IN)

コール固有モードです。有効な値は次のとおりです。

- OCI_DEFAULT - デフォルトのコールを実行します。次の有効化操作が実行されるまで、このサブスクリプションの通知はすべてバッファリングされます。

コメント

このコールは、サブスクリプションの登録を無効にした後に、通知をオンにするときに使います。

この操作を実行するときは、接続または認証は必要ありません。このコールを実行する前に、指定されたサブスクリプションに対して登録が行われていなければなりません。

関連関数

OCIAQListen()、*OCISvcCtxToLda()*、*OCISubscriptionPost()*、*OCISubscriptionRegister()*、*OCISubscriptionUnRegister()*

OCISubscriptionPost()

用途

サブスクリプションに転記し、そのサブスクリプションに対して登録されたすべてのクライアントが通知を受信できるようにします。

構文

```
ub4 OCISubscriptionPost ( OCISvcCtx          *svchp,
                          OCISubscription **subscrhpp,
                          ub2              count,
                          OCIError        *errhp,
                          ub4              mode );
```

パラメータ

svchp (IN)

V8 OCI サービス・コンテキストです。このサービス・コンテキストには、認証された有効なユーザー・ハンドルが必要です。

subscrhpp (IN)

サブスクリプション・ハンドルの配列です。この配列の各要素は、OCI_ATTR_SUBSCR_NAME および OCI_ATTR_SUBSCR_NAMESPACE 属性が設定されたサブスクリプション・ハンドルでなければなりません。詳細は、A-37 ページの「[サブスクリプション・ハンドルの属性](#)」を参照してください。

このコールの前に、各サブスクリプション・ハンドルに対して OCI_ATTR_SUBSCR_PAYLOAD 属性を設定する必要があります。設定しない場合は、ペイロードが NULL であると見なされ、関心領域に登録したクライアントが通知を受信するときに、ペイロードが配布されません。OCIAttrSet() コールでは、ペイロードへの参照は記録されますが、内容はコピーされないため、コール側は転記が行われるまでペイロードを保存する必要があります。

count (IN)

サブスクリプション・ハンドル配列の要素数です。

errhp (OUT)

エラー時の診断情報のために OCIErrorGet() に渡せるエラー・ハンドルです。

mode (IN)

コール固有モードです。有効な値は次のとおりです。

- OCI_DEFAULT - デフォルトのコールを実行します。

コメント

サブスクリプションに転記されるときに、必要に応じてサブスクリプション名およびペイロードの識別を行うことができます。ペイロードが関連付けられない場合は、ペイロードの長さは0に設定されます。

このコールでは、ベストエフォート型の保証を提供しています。通知は、登録されたクライアントに対して最大で1回行われます。

このコールは、主に軽量の通知に使い、複数のシステム・イベントが発生する場合に役立ちます。アプリケーションでより確実な保証が必要な場合は、アドバンスト・キューイング機能を使い、キューにエンキューします。

関連関数

[OCIAQListen\(\)](#)、[OCISvcCtxToLda\(\)](#)、[OCISubscriptionEnable\(\)](#)、[OCISubscriptionRegister\(\)](#)、[OCISubscriptionUnRegister\(\)](#)

OCISubscriptionRegister()

用途

メッセージ通知に対するコールバックを登録します。

構文

```
ub4 OCISubscriptionRegister ( OCISvcCtx      *svchp,
                             OCISubscription **subscrhpp,
                             ub2           count,
                             OCIError      *errhp,
                             ub4           mode );
```

パラメータ

svchp (IN)

V8 OCI サービス・コンテキストです。このサービス・コンテキストには、認証された有効なユーザー・ハンドルが必要です。

subscrhpp (IN)

サブスクリプション・ハンドルの配列です。この配列の各要素は、OCI_ATTR_SUBSCR_NAME、OCI_ATTR_SUBSCR_NAMESPACE、OCI_ATTR_SUBSCR_CBACK および OCI_ATTR_SUBSCR_CTX 属性が設定されたサブスクリプション・ハンドルでなければなりません。設定されていない場合は、エラーが戻されます。詳細は、A-37 ページの「[サブスクリプション・ハンドルの属性](#)」を参照してください。

subscrhpp[i] によって示された登録に対して通知を受信した場合は、subscrhpp[i] に対してコンテキスト (OCI_ATTR_SUBSCR_CTX) が設定された状態で、ユーザー定義コールバック関数 (OCI_ATTR_SUBSCR_CBACK) がコールされます。

count (IN)

サブスクリプション・ハンドル配列の要素数です。

errhp (OUT)

エラー時の診断情報のために OCIErrorGet() に渡せるエラー・ハンドルです。

mode (IN)

コール固有モードです。有効な値は次のとおりです。

- OCI_DEFAULT - デフォルトのコールを実行します。登録が切断されていると見なされるように指定します。
- OCI_NOTIFY_CONNECTED - クライアントが接続されている場合にだけ、通知を受信します (このリリースではサポートされていません)。

新しいクライアント・プロセスが発生したとき、または古いクライアント・プロセスのダウンおよび復旧が発生したときは、重要なすべてのサブスクリプションに対する登録が必要で

す。クライアントが接続されたままサーバーがダウンし、その後復旧した場合は、クライアントは、DISCONNECTED した登録の通知を引き続き受信します。ただし、ユーザーは、CONNECTED した登録の通知は受信しません。サーバーのダウンおよび復旧が発生すると、その登録が失われるからです。

コメント

このコールは、サブスクリプションの登録に対して行います。重要なサブスクリプション名およびコールする関連したコールバックを識別します。重要な複数のサブスクリプションを同時に登録できます。

このインタフェースは、非同期モードのメッセージ配信の場合にだけ有効です。非同期モードでは、サブスクライバが登録コールのパブリッシュおよびコールバックの指定を行います。サブスクリプション条件と一致するメッセージを受信すると、コールバックがコールされます。次に、コールバックから明示的な `message_receive` (デキュー) がパブリッシュされ、メッセージが取り出されます。

ユーザーは、登録時に名前領域属性を `OCI_SUBSCR_NAMESPACE_AQ` に設定して、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・キューの登録の場合は文字列 'SCHEMA.QUEUE'、複数コンシューマ・キューの登録の場合は文字列 'SCHEMA.QUEUE:CONSUMER' です。サブスクリプション文字列内の SCHEMA の指定はオプションです。SCHEMA を指定しない場合は、キューはログイン・ユーザーのスキーマであると見なされます。サブスクリプションを登録するには、キューに対する DEQUEUE 特権が必要です。

各名前領域には、独自の特権モデルがあります。登録を行っているユーザーに、指定されたサブスクリプションの名前領域で登録する権限がない場合は、エラーが戻されます。

関連関数

[OCIAQListen\(\)](#)、[OCISvcCtxToLda\(\)](#)、[OCISubscriptionEnable\(\)](#)、[OCISubscriptionPost\(\)](#)、[OCISubscriptionUnRegister\(\)](#)

OCISubscriptionUnRegister()

用途

サブスクリプションの登録を解除して、通知をオフにします。

構文

```
ub4 OCISubscriptionUnRegister ( OCISvcCtx          *svchp,  
                                OCISubscription    *subscrhp,  
                                OCIError           *errhp  
                                ub4                mode );
```

パラメータ

svchp (IN)

V8 OCI サービス・コンテキストです。このサービス・コンテキストには、有効な認証されたユーザー・ハンドルが必要です。

subscrhp (IN)

OCI_ATTR_SUBSCR_NAME および OCI_ATTR_SUBSCR_NAMESPACE 属性が設定されたサブスクリプション・ハンドルです。詳細は、A-37 ページの「[サブスクリプション・ハンドルの属性](#)」を参照してください。

errhp (OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

mode (IN)

コール固有モードです。有効な値は次のとおりです。

- OCI_DEFAULT - デフォルトのコールを実行します。

コメント

サブスクリプションに対する登録を解除すると、その後は特定のサブスクリプションに関する通知を受信できなくなります。通知を再開する場合は、サブスクリプションに対して再登録を行ってください。

関心領域が登録されたクライアントのリストからユーザーが削除されるため、別の登録が行われると、解除前に配信されていた通知はすべて配信されなくなります。

関連関数

[OCIAQListen\(\)](#)、[OCISvcCtxToLda\(\)](#)、[OCISubscriptionEnable\(\)](#)、[OCISubscriptionPost\(\)](#)、[OCISubscriptionRegister\(\)](#)

ハンドル関数および記述子関数

この項では、OCIハンドル関数および記述子関数について説明します。

表 15-2 OCI クイック・リファレンス

関数	用途	ページ
OCIAttrGet()	ハンドルの属性を取得します。	15-28 ページ
OCIAttrSet()	ハンドルまたは記述子の属性を設定します。	15-29 ページ
OCIDescriptorAlloc()	記述子または LOB ロケータの割当てと初期化を行います。	15-31 ページ
OCIDescriptorFree()	割当て済みの記述子を解放します。	15-33 ページ
OCIHandleAlloc()	ハンドルの割当てと初期化を行います。	15-34 ページ
OCIHandleFree()	割当て済みのハンドルを解放します。	15-37 ページ
OCIParamGet()	パラメータ記述子を取得します。	15-39 ページ
OCIParamSet()	COR ハンドルにパラメータ記述子を設定します。	15-41 ページ

OCIAttrGet()

用途

このコールは、ハンドルの特定の属性を取得するときに使用されます。

構文

```
sword OCIAttrGet ( CONST dvoid    *trgthndlp,  
                  ub4            trgthndltyp,  
                  dvoid          *attributep,  
                  ub4            *sizep,  
                  ub4            attrtype,  
                  OCIError       *errhp );
```

パラメータ

trgthndlp (IN)

ハンドル・タイプのポインタです。

trghndltyp (IN)

ハンドル・タイプです。

attributep (OUT)

属性値の格納場所へのポインタです。属性値を指定します。

sizep (OUT)

属性値の記憶領域のサイズです。サイズが明らかなパラメータでは、NULLとして渡すことができます。text* パラメータでは、文字列の長さを取得するために ub4 のポインタを渡す必要があります。

attrtype (IN)

取り出される属性のタイプです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

このコールは、ハンドルの特定の属性を取得するときに使用されます。ハンドル・タイプとその読取り可能な属性については、[付録 A の「ハンドルおよび記述子の属性」](#)のリストを参照してください。

関連関数

[OCIAttrSet\(\)](#)

OCIAttrSet()

用途

このコールは、ハンドルまたは記述子の特定の属性を設定するときに使用されます。

構文

```
sword OCIAttrSet ( dvoid          *trgthndlp,  
                  ub4            trgndltyp,  
                  dvoid          *attributep,  
                  ub4            size,  
                  ub4            attrtype,  
                  OCIError       *errhp );
```

パラメータ

trgthndlp (IN/OUT)

その属性が変更されるハンドル・タイプのポインタです。

trgndltyp (IN/OUT)

ハンドル・タイプです。

attributep (IN)

属性値のポインタです。この属性値が目的のハンドルにコピーされます。属性値がポインタの場合は、そのポインタがコピーされるだけであり、ポインタの内容はコピーされません。

size (IN)

属性値のサイズです。サイズは OCI ライブラリで既にわかっているため、ほとんどの属性に対して 0 (ゼロ) で渡すことができます。text* 属性では、文字列の長さを設定するために ub4 を渡す必要があります。

attrtype (IN)

設定される属性のタイプです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

ハンドル・タイプとその書込み可能な属性については、[付録 A の「ハンドルおよび記述子の属性」](#)のリストを参照してください。

例

次のサンプル・コードは、アプリケーションの開始部分で *OCIAttrSet()* を何回か使用している例です。

```
int main()
{
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
OCISession *usrhp;

OCIInitialize((ub4) OCI_THREADED | OCI_OBJECT, (dvoid *)0,
              (dvoid * (*)()) 0, (dvoid * (*)()) 0, (void (*)()) 0 );
OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                0, (dvoid **) &tmp);
OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 0, (dvoid **) &tmp );
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4)
                OCI_HTYPE_ERROR, 0, (dvoid **) &tmp);
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4)
                OCI_HTYPE_SERVER, 0, (dvoid **) &tmp);
OCIErrorAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp,
                (ub4) OCI_HTYPE_SVCCTX, , (dvoid **) &tmp);
/* set attribute server context in the service context */
OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) srvhp,
            (ub4) 0, (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);
/* allocate a user session handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp,
               (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0);
OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION, (dvoid *)"sherry",
           (ub4)strlen("sherry"), OCI_ATTR_USERNAME, errhp);
OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION, (dvoid *)"penfield",
           (ub4)strlen("penfield"), OCI_ATTR_PASSWORD, errhp);
checkerr(errhp, OCI_SessionBegin( svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                  OCI_DEFAULT));
OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX, (dvoid *)usrhp,
           (ub4)0, OCI_ATTR_SESSION, errhp);
```

関連関数

[OCIAttrGet\(\)](#)

OCIDescriptorAlloc()

用途

記述子または LOB ロケータを格納する記憶領域を割り当てます。

構文

```
sword OCIDescriptorAlloc ( CONST dvoid   *parenth,  
                           dvoid        **descpp,  
                           ub4           type,  
                           size_t       xtrmem_sz,  
                           dvoid        **usrmempp );
```

パラメータ

parenth (IN)

環境ハンドルです。

descpp (OUT)

希望のタイプの記述子または LOB ロケータを戻します。

type (IN)

割り当てられる記述子または LOB ロケータのタイプを指定します。

- OCI_DTYPE_SNAP - C 型のスナップショット記述子 OCISnapshot の生成を指定します。
- OCI_DTYPE_LOB - C 型の LOB 値型ロケータ (BLOB または CLOB の場合) OCILobLocator の生成を指定します。
- OCI_DTYPE_FILE - C 型の FILE 値型ロケータ OCILobLocator の生成を指定します。
- OCI_DTYPE_ROWID - C 型の ROWID 記述子 OCIRowid の生成を指定します。
- OCI_DTYPE_DATETIME - C 型の DATETIME 記述子 OCIDateTime の生成を指定します。
- OCI_DTYPE_INTERVAL - C 型の INTERVAL 記述子 OCIInterval の生成を指定します。
- OCI_DTYPE_COMPLEXOBJECTCOMP - C 型の複合オブジェクト検索記述子 OCIComplexObjectComp の生成を指定します。
- OCI_DTYPE_AQENQ_OPTIONS - C 型のアドバンスト・キューイング・エンキュー・オプション記述子 OCIAQEnqOptions の生成を指定します。
- OCI_DTYPE_AQDEQ_OPTIONS - C 型のアドバンスト・キューイング・デキュー・オプション記述子 OCIAQDeqOptions の生成を指定します。

- OCI_DTYPE_AQMSG_PROPERTIES - C 型のアドバンスド・キューイング・メッセージ・プロパティ記述子 `OCIAQMsgProperties` の生成を指定します。
- OCI_DTYPE_AQAGENT - C 型のアドバンスド・キューイング・エージェント記述子 `OCIAQAgent` の生成を指定します。

xtrmem_sz (IN)

記述子の存続期間中アプリケーションで使用するために割り当てられるユーザー・メモリーの量を指定します。

usrmempp (OUT)

記述子の存続期間中コールによってユーザー用に割り当てられた、サイズ `xtrmem_sz` のユーザー・メモリーのポインタを戻します。

コメント

`type` で指定された型に対応する、割当ておよび初期化済の記述子のポインタを戻します。成功した場合は、非 NULL 記述子または LOB ロケータが戻ります。エラー発生時に診断は利用できません。

このコールは、成功した場合は `OCI_SUCCESS` を、メモリー不足エラーが発生した場合は `OCI_INVALID_HANDLE` を戻します。

`xtrmem_sz` パラメータおよびユーザー・メモリー割当ての詳細は、2-13 ページの「[ユーザー・メモリーの割当て](#)」を参照してください。

関連関数

[OCIDescriptorFree\(\)](#)

OCIDescriptorFree()

用途

割当て済みの記述子の割当てを解除します。

構文

```
sword OCIDescriptorFree ( dvoid    *descp,  
                          ub4      type );
```

パラメータ

descp (IN)

割り当てられた記述子です。

type (IN)

開放する記憶領域のタイプを指定します。指定する型を次に示します。

- OCI_DTYPE_SNAP - スナップショット記述子
- OCI_DTYPE_LOB - LOB 値型記述子
- OCI_DTYPE_FILE - FILE 値型記述子
- OCI_DTYPE_ROWID - ROWID 記述子
- OCI_DTYPE_DATETIME - DATETIME 記述子
- OCI_DTYPE_INTERVAL - INTERVAL 記述子
- OCI_DTYPE_COMPLEXOBJECTCOMP - 複合オブジェクト検索記述子
- OCI_DTYPE_AQENQ_OPTIONS - AQ エンキュー・オプション記述子
- OCI_DTYPE_AQDEQ_OPTIONS - AQ デキュー・オプション記述子
- OCI_DTYPE_AQMSG_PROPERTIES - AQ メッセージ・プロパティ記述子
- OCI_DTYPE_AQAGENT - AQ エージェント記述子

コメント

このコールにより、記述子に対応付けられた記憶領域が解放されます。OCI_SUCCESS または OCI_INVALID_HANDLE を戻します。記述子はすべて明示的に割当てを解除できますが、環境ハンドルの割当てを解除すると、OCI は記述子の割当てを自動的に解除します。

関連関数

[OCIDescriptorAlloc\(\)](#)

OCIHandleAlloc()

用途

このコールは、割当てと初期化がすでに行われたハンドルのポインタを戻します。

構文

```
sword OCIHandleAlloc ( CONST dvoid   *parenth,  
                      dvoid        **hndlpp,  
                      ub4           type,  
                      size_t       xtrmem_sz,  
                      dvoid        **usrmempp );
```

パラメータ

parenth (IN)

環境ハンドルです。

hndlpp (OUT)

ハンドルを戻します。

type (IN)

割り当てられるハンドルのタイプを指定します。使用できる型を次に示します。

- OCI_HTYPE_ERROR - C 型のエラー・レポート・ハンドル OCIError の生成を指定します。
- OCI_HTYPE_SVCCTX - C 型のサービス・コンテキスト・ハンドル OCISvcCtx の生成を指定します。
- OCI_HTYPE_STMT - C 型の文 (アプリケーション要求) ハンドル OCIStmt の生成を指定します。
- OCI_HTYPE_DESCRIBE - C 型の選択リスト説明ハンドル OCIDescribe の生成を指定します。
- OCI_HTYPE_SERVER - C 型のサーバー・コンテキスト・ハンドル OCIServer の生成を指定します。
- OCI_HTYPE_SESSION - C 型のユーザー・セッション・ハンドル OCISession の生成を指定します。
- OCI_HTYPE_TRANS - C 型のトランザクション・コンテキスト・ハンドル OCITrans の生成を指定します。
- OCI_HTYPE_COMPLEXOBJECT - C 型の複合オブジェクト検索ハンドル OCIComplexObject の生成を指定します。

- OCI_HTYPE_SECURITY - C 型のセキュリティ・ハンドル `OCI_Security` の生成を指定します。
- OCI_HTYPE_SUBSCR - C 型のサブスクリプション・ハンドル `OCI_Subscription` の生成を指定します。
- OCI_HTYPE_DIRPATH_CTX - C 型のダイレクト・パス・コンテキスト・ハンドル `OCI_DirPathCtx` の生成を指定します。
- OCI_HTYPE_DIRPATH_COLUMN_ARRAY - C 型のダイレクト・パス列配列ハンドル `OCI_DirPathColArray` の生成を指定します。
- OCI_HTYPE_DIRPATH_STREAM - C 型のダイレクト・パス・ストリーム・ハンドル `OCI_DirPathStream` の生成を指定します。
- OCI_HTYPE_PROCESS - C 型のプロセス・ハンドル `OCI_Process` の生成を指定します。

xtrmem_sz (IN)

割り当てられるべきユーザー・メモリーの量を指定します。

usrmempp (OUT)

コールによりユーザーに割り当てられた、サイズ `xtrmem_sz` のユーザー・メモリーのポインタを戻します。

コメント

type で指定された型に対応する、割当ておよび初期化済みのハンドルのポインタを戻します。成功時には非 NULL ハンドルが戻ります。すべてのハンドルが、親ハンドルとして渡される環境ハンドルと対応付けて割り当てられます。

エラー発生時に診断は利用できません。このコールは、成功した場合は `OCI_SUCCESS` を、エラーが発生した場合は `OCI_INVALID_HANDLE` を戻します。

ハンドルは、OCI コールに渡す前に `OCIHandleAlloc()` を使用して割り当てられていなければなりません。

環境ハンドルの割当ておよび初期化を行うには、`OCIEnvInit()` をコールします。

関連項目 : `xtrmem_sz` パラメータを使ったユーザー・メモリー割当ての詳細は、2-13 ページの「[ユーザー・メモリーの割当て](#)」を参照してください。

例

次のサンプル・コードは、`OCIHandleAlloc()` を使用して、アプリケーションの開始時にさまざまなハンドルを割り当てる例です。

```
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4)
    OCI_HTYPE_ERROR, 0, (dvoid **) &tmp);
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4)
    OCI_HTYPE_SERVER, 0, (dvoid **) &tmp);
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4)
```

OCIHandleAlloc()

```
OCI_HTYPE_SVCCCTX, 0, (dvoid **) &tmp);
```

関連関数

[OCIHandleFree\(\)](#)、[OCIEnvInit\(\)](#)

OCIHandleFree()

用途

このコールは、ハンドルの割当てを明示的に解除します。

構文

```
sword OCIHandleFree ( dvoid      *hndlp,  
                     ub4        type );
```

パラメータ

hndlp (IN)

OCIHandleAlloc() により割り当てられるハンドルです。

type (IN)

開放する記憶領域のタイプを指定します。指定する型を次に示します。

- OCI_HTYPE_ENV - 環境ハンドル
- OCI_HTYPE_ERROR - エラー・レポート・ハンドル
- OCI_HTYPE_SVCCTX - サービス・コンテキスト・ハンドル
- OCI_HTYPE_STMT - 文 (アプリケーション要求) ハンドル
- OCI_HTYPE_DESCRIBE - 選択リスト記述ハンドル
- OCI_HTYPE_SERVER - サーバー・ハンドル
- OCI_HTYPE_SESSION - ユーザー・セッション・ハンドル
- OCI_HTYPE_TRANS - トランザクション・ハンドル
- OCI_HTYPE_COMPLEXOBJECT - 複合オブジェクト検索ハンドル
- OCI_HTYPE_SECURITY - セキュリティ・ハンドル
- OCI_HTYPE_SUBSCR - サブスクリプション・ハンドル
- OCI_HTYPE_DIRPATH_CTX - ダイレクト・パス・コンテキスト・ハンドル
- OCI_HTYPE_DIRPATH_COLUMN_ARRAY - ダイレクト・パス列配列ハンドル
- OCI_HTYPE_DIRPATH_STREAM - ダイレクト・パス・ストリーム・ハンドル
- OCI_HTYPE_PROCESS - プロセス・ハンドル

コメント

このコールは、ハンドルに関連付けられている記憶領域で、*type* パラメータに指定された型に該当するものを解放します。

このコールは、OCI_SUCCESS または OCI_INVALID_HANDLE を戻します。

ハンドルはすべて明示的に割当てを解除できます。OCI では、親ハンドルの割当てを解除すると、子ハンドルの割当てが自動的に解除されます。

関連関数

[OCIHandleAlloc\(\)](#)、[OCIEnvInit\(\)](#)

OCIParamGet()

用途

記述ハンドルまたは文ハンドル内の指定位置にあるパラメータの記述子を戻します。

構文

```
sword OCIParamGet ( CONST dvoid      *hndlp,  
                   ub4              htype,  
                   OCIError        *errhp,  
                   dvoid           **parmdp,  
                   ub4              pos );
```

パラメータ

hndlp (IN)

文ハンドルまたは記述ハンドルです。OCIParamGet() 関数は、このハンドル用のパラメータ記述子を戻します。

htype (IN)

handle パラメータに渡されるハンドルのタイプです。次の型が有効です。

- OCI_DTYPE_PARM、パラメータ記述子用
- OCI_HTYPE_COR、複合オブジェクト検索ハンドル用
- OCI_HTYPE_STMT、文ハンドル用

errhp (IN/OUT)

エラー時の診断情報のために OCIErrorGet() に渡せるエラー・ハンドルです。

parmdp (OUT)

pos パラメータで与えられた位置におけるパラメータの記述子です。

pos (IN)

文ハンドルまたは記述ハンドルにおける位置番号です。この位置にあるパラメータ記述子が戻ります。

注意: 指定位置にパラメータ記述子がない場合は、OCI_NO_DATA が戻ることがあります。

コメント

このコールは、記述ハンドルまたは文ハンドル内の指定位置にあるパラメータの記述子を戻します。パラメータ記述子は、常に OCI ライブラリによって内部的に割り当てられます。パラメータ記述子は読取り専用です。

指定位置にパラメータ記述子がない場合は、OCI_NO_DATA が戻ることがあります。

OCIParamGet()

パラメータ記述子の属性の詳細は、付録 A の「ハンドルおよび記述子の属性」を参照してください。

関連関数

[OCIAttrGet\(\)](#)、[OCIAttrSet\(\)](#)、[OCIParamSet\(\)](#)

OCIParamSet()

用途

COR ハンドルへの複合オブジェクト検索 (COR) 記述子の設定に使用します。

構文

```
sword OCIParamSet ( dvoid          *hndlp,  
                   ub4            htype,  
                   OCIError       *errhp,  
                   CONST dvoid     *dscp,  
                   ub4            dtyp,  
                   ub4            pos );
```

パラメータ

hndlp (IN/OUT)

ハンドル・ポインタです。

htype (IN)

ハンドル・タイプです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

dscp (IN)

複合オブジェクト検索記述子ポインタです。

dtyp (IN)

記述子型です。COR 記述子の記述子の型は、OCI_DTYPE_COMPLEXOBJECTCOMP です。

pos (IN)

位置番号です。

コメント

COR ハンドルは [OCIHandleAlloc\(\)](#) を使って、そして記述子は [OCIDescriptorAlloc\(\)](#) を使って、それぞれ前もって割り当てておく必要があります。記述子の属性は、[OCIAttrSet\(\)](#) を使って設定します。

複合オブジェクト検索の詳細は、10-19 ページの「[複合オブジェクト検索](#)」を参照してください。

関連関数

[OCIParamGet\(\)](#)

バインド関数、定義関数および記述関数

この項では、バインド関数、定義関数および記述関数について説明します。

表 15-3 OCI クイック・リファレンス

関数	用途	ページ
OCIBindArrayOfStruct()	静的配列バインド用のスキップ・パラメータを設定します。	15-43 ページ
OCIBindByName()	名前によりバインドします。	15-44 ページ
OCIBindByPos()	位置によりバインドします。	15-48 ページ
OCIBindDynamic()	OCI_DATA_AT_EXEC モードでバインドした後、追加属性を設定します。	15-52 ページ
OCIBindObject()	名前付きデータ型のバインド用の追加属性を設定します。	15-56 ページ
OCIDefineArrayOfStruct()	静的配列定義用の追加属性を設定します。	15-58 ページ
OCIDefineByPos()	出力変数の関連性を定義します。	15-60 ページ
OCIDefineDynamic()	OCI_DYNAMIC_FETCH モードでの定義用の追加属性を設定します。	15-64 ページ
OCIDefineObject()	名前付きデータ型の定義用の追加属性を設定します。	15-66 ページ
OCIDescribeAny()	既存のスキーマ・オブジェクトを記述します。	15-68 ページ
OCIStmtGetBindInfo()	バインドおよび標識変数名とハンドルを取得します。	15-71 ページ

OCIBindArrayOfStruct()

用途

このコールは、静的配列バインド用のスキップ・パラメータを設定します。

構文

```
sword OCIBindArrayOfStruct ( OCIBind      *bindp,
                             OCIError    *errhp,
                             ub4         pvskip,
                             ub4         indskip,
                             ub4         alskip,
                             ub4         rcskip );
```

パラメータ

bindp (IN/OUT)

バインド構造体へのハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

pvskip (IN)

次のデータ値用のスキップ・パラメータです。

indskip (IN)

次の標識値または構造体のためのスキップ・パラメータです。

alskip (IN)

次の実際の長さ値のためのスキップ・パラメータです。

rcskip (IN)

次の列レベルのリターン・コード値のためのスキップ・パラメータです。

コメント

このコールは、静的配列バインドに必要なスキップ・パラメータを設定します。これは、*OCIBindByName()* または *OCIBindByPos()* の後続コールです。初期バインド・コールによって戻されたバインド・ハンドルが *OCIBindArrayOfStruct()* コールのパラメータとして使用されます。スキップ・パラメータについては、5-17 ページの「[構造体の配列](#)」を参照してください。

関連関数

[OCIBindByName\(\)](#)、[OCIBindByPos\(\)](#)

OCIBindByName()

用途

プログラム変数と SQL 文または PL/SQL ブロック内のプレースホルダを関連付けます。

構文

```
sword OCIBindByName ( OCIStmt      *stmtp,  
                     OCIBind      **bindpp,  
                     OCIError     *errhp,  
                     CONST text   *placeholder,  
                     sb4          placeh_len,  
                     dvoid        *valuep,  
                     sb4          value_sz,  
                     ub2          dty,  
                     dvoid        *indp,  
                     ub2          *alenp,  
                     ub2          *rcodep,  
                     ub4          maxarr_len,  
                     ub4          *curelep,  
                     ub4          mode );
```

パラメータ

stmtp (IN/OUT)

処理中の SQL または PL/SQL 文への文ハンドルです。

bindpp (IN/OUT)

このコールによって暗示的に割り当てられる連結ハンドルです。この特定の入力値に対するバインド情報はすべて、このバインド・ハンドルによってメンテナンスされます。ハンドルは、文ハンドルの割当てが解除されたときに自動的に解放されます。入力時には、このポインタの値は NULL または有効なバインド・ハンドルでなければなりません。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

placeholder (IN)

OCIBindByName() がコールされている場合、プレースホルダの属性が名前指定されます。

placeh_len (IN)

placeholder に指定されたプレースホルダ名の長さです。

valuep (IN/OUT)

データ値のアドレスまたは *dty* パラメータで指定されたタイプのデータ値配列です。データ値配列は、PL/SQL 表にマップする、または SQL 複数行操作のデータを用意するために

指定できます。バインド値配列が用意された場合、これは OCI 用語では配列バインドと呼ばれます。

SQLT_NTY または SQLT_REF バインドの場合、*valuep* パラメータは無視されます。OUT バッファへのポインタは、OCIBindObject() によって初期化される pgvpp パラメータ内に設定されます。

OCI_ATTR_CHARSET_ID 属性が OCI_UCS2ID (Unicode) に設定されている場合は、対応するバインド・コールに対して受渡しされるすべてのデータは、UCS-2 コード化されていると見なされます。詳細は、A-20 ページの [OCI_ATTR_CHARSET_ID](#) を参照してください。

value_sz (IN)

データ値のサイズです。配列バインドの場合、これは、alenp パラメータに指定される実サイズで可能な要素の最大サイズになります。

サイズがクライアント・アプリケーションで不明な記述子、ロケータ、または REF の場合、ユーザーが渡す構造体のサイズが使用されます。例 : sizeof (OCILobLocator *)

dtty (IN)

バインドされる値のデータ型です。名前付きデータ型 (SQLT_NTY) と REF (SQLT_REF) は、アプリケーションがオブジェクト・モードで初期化されている場合だけ有効です。名前付きデータ型または REF の場合、バインド・ハンドルで追加コールが行われ、データ型特定の属性が設定されます。

indp (IN/OUT)

標識変数または配列へのポインタです。SQLT_NTY 以外のデータ型の場合は、sb2 または sb2 の配列へのポインタです。

SQLT_NTY に対してはこのポインタは無視され、標識構造体または標識構造体の配列の実際のポインタが後続のコール OCIBindObject() で初期化されます。このパラメータは、動的バインドでは無視されます。

標識変数の詳細は、2-30 ページの「[標識変数](#)」を参照してください。

alenp (IN/OUT)

配列要素の実際の長さの配列へのポインタです。*alenp* 内の各要素は、実行前後のバインド値配列内の該当要素のデータ長です。このパラメータは、動的バインドでは無視されます。

rcodep (OUT)

列レベル・リターン・コードの配列を指すポインタです。このパラメータは、動的バインドでは無視されます。

maxarr_len (IN)

PL/SQL バインドでタイプ *dtty* の要素の最大可能数です。このパラメータは、非 PL/SQL バインドでは必要ありません。*maxarr_len* がゼロ以外であれば、OCIBindDynamic() または OCIBindArrayOfStruct() を呼び出して、追加バインド属性を設定できます。

curelep (IN/OUT)

実際の要素の数へのポインタです。このパラメータは、PL/SQL バインドでだけです。

mode (IN)

このパラメータに有効なモードは次のとおりです。

OCI_DEFAULT - これはデフォルト・モードです。

OCI_DATA_AT_EXEC - このモードが選択される場合、*value_sz* パラメータは、実行時に提供可能なデータの最大サイズを定義します。アプリケーションは、OCI ライブラリのランタイム IN データ・バッファをいつでも何回でも用意することができるように準備する必要があります。ランタイム・データは、次の 2 つの方法のどちらかを使用して用意されます。

- *OCIBindDynamic()* への後続コールによって登録する必要があるユーザー定義関数を使用するコールバック。
- OCI で用意されているコールを使用するポーリング・メカニズム。このモードは、コールバックが定義されていない場合自動的に選ばれます。

OCI_DATA_AT_EXEC モードの使用の詳細は、5-30 ページの「ランタイム・データ割当てとピース単位操作」を参照してください。

割り当てたバッファは、不要になった際にクライアント側で解放する必要があります。

コメント

このコールは、基本バインド操作を実行するときに使用されます。バインドは、プログラム変数のアドレスと SQL 文または PL/SQL ブロック内のプレースホルダの関連付けを作成します。このバインド・コールは、バインドされるデータの型も指定し、実行時にデータを用意するメソッドも指示できます。

この関数は、*bindpp* パラメータによって指示されたバインド・ハンドルの暗黙の割当ても行います。***bindpp* に非 NULL ポインタが渡されると、OCI では、*OCIHandleAlloc()* または *OCIBindByName()* のコールで以前に割り当てられた有効なハンドルを指します。

OCI アプリケーション内のデータは、プレースホルダに静的または動的にバインドできます。実行の直前にすべての IN バインド・データと OUT バインド・バッファが正しく定義された場合、バインドは静的になります。実行時にクライアント・ライブラリからの要求によりアプリケーションから IN バインド・データと OUT バインド・バッファが供給される場合、バインドは動的になります。動的バインドを指定するには、このコールの *mode* パラメータを *OCI_DATA_AT_EXEC* に設定します。

関連項目: 動的バインディングの詳細は、5-30 ページの「ランタイム・データ割当てとピース単位操作」を参照してください。

OCIBindByName() と *OCIBindByPos()* は、パラメータとしてバインド・ハンドルを取りますが、これは、バインド・コールによって暗黙的に割り当てられます。アプリケーションでバインドしているすべてのプレースホルダに別個のバインド・ハンドルが割り当てられます。

特定のデータ型をバインドする際または入力データを特定の方法で処理する際に必要となる特定の属性を指定する場合は、次の追加バインド・コールが必要です。

- 構造体の配列が使われている場合、*OCIBindArrayOfStruct()* をコールして必要なスキップ・パラメータを設定します。
- 実行時にデータが動的に提供されている場合、アプリケーションでユーザー定義のコールバック関数を使うときには *OCIBindDynamic()* をコールしてコールバックを登録します。
- 名前付きデータ型が結び付けられている場合、*OCIBindObject()* をコールしてその他の必要情報を指定します。
- RETURNING 句を含む文を使用する場合は、このコールの後に *OCIBindDynamic()* をコールする必要があります。

関連関数

OCIBindDynamic()、*OCIBindObject()*、*OCIBindArrayOfStruct()*

OCIBindByPos()

用途

プログラム変数と SQL 文または PL/SQL ブロック内のプレースホルダを関連付けます。

構文

```
sword OCIBindByPos ( OCIStmt      *stmtp,  
                    OCIBind      **bindpp,  
                    OCIError     *errhp,  
                    ub4          position,  
                    dvoid        *valuep,  
                    sb4          value_sz,  
                    ub2          dty,  
                    dvoid        *indp,  
                    ub2          *alenp,  
                    ub2          *rcodep,  
                    ub4          maxarr_len,  
                    ub4          *curelep,  
                    ub4          mode );
```

パラメータ

stmtp (IN/OUT)

処理中の SQL または PL/SQL 文への文ハンドルです。

bindpp (IN/OUT)

このコールによって暗示的に割り当てられる連結ハンドルです。この特定の入力値に対するバインド情報はすべて、このバインド・ハンドルによって メンテナンスされます。ハンドルは、文ハンドルの割当てが解除されたときに暗黙的に解放されます。入力時には、このポインタの値は NULL または有効なバインド・ハンドルでなければなりません。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

position (IN)

OCIBindByPos() がコールされている場合、プレースホルダの属性が位置別に指定されます。

valuep (IN/OUT)

データ値のアドレスまたは *dty* パラメータで指定されたタイプのデータ値配列です。データ値配列は、PL/SQL 表にマップする、または SQL 複数行操作のデータを用意するために指定できます。バインド値配列が用意された場合、これは OCI 用語では配列バインドと呼ばれます。

SQLT_NTY または SQLT_REF バインドの場合、valuep パラメータは無視されます。OUT バッファへのポインタは、OCIBindObject() によって初期化される pgvpp パラメータ内に設定します。

OCI_ATTR_CHARSET_ID 属性が OCI_UCS2ID (Unicode) に設定されている場合は、対応するバインド・コールに対して渡されるか、またはそれらのコールを使って受け取るすべてのデータは、UCS-2 コード化されていると見なされます。詳細は、A-20 ページの [OCI_ATTR_CHARSET_ID](#) を参照してください。

value_sz (IN)

データ値のサイズ。配列バインドの場合、これは、alenp パラメータに指定される実サイズで可能な要素の最大サイズになります。

サイズがクライアント・アプリケーションで不明な記述子、ロケータ、または REF の場合、ユーザーが渡す構造体のサイズが使用されます。例: sizeof (OCILobLocator *)

dtypes (IN)

バインドされる値のデータ型です。名前付きデータ型 (SQLT_NTY) と REF (SQLT_REF) は、アプリケーションがオブジェクト・モードで初期化されている場合だけ有効です。名前付きデータ型または REF の場合、バインド・ハンドルで追加コールが行われ、データ型特定の属性が設定されます。

indp (IN/OUT)

標識変数または配列へのポインタです。すべてのデータ型について、これは sb2 または sb2 の配列へのポインタです。ただ 1 つの例外は SQLT_NTY です。SQLT_NTY に対してはこのポインタは無視され、標識構造体の実際のポインタまたは標識構造体の配列が OCIBindObject() により初期化されます。動的バインドでは無視されます。

標識変数の詳細は、2-30 ページの「[標識変数](#)」を参照してください。

alenp (IN/OUT)

配列要素の実際の長さの配列へのポインタです。alenp 内の各要素は、実行前後のバインド値配列内の該当要素のデータ長です。このパラメータは、動的バインドでは無視されます。

rcodep (OUT)

列レベル・リターン・コードの配列を指すポインタです。このパラメータは、動的バインドでは無視されます。

maxarr_len (IN)

PL/SQL バインドでタイプ dtypes の要素の最大可能数です。このパラメータは、非 PL/SQL バインドでは必要ありません。maxarr_len がゼロ以外であれば、OCIBindDynamic() または OCIBindArrayOfStruct() を呼び出して、追加バインド属性を設定できます。

curelep (IN/OUT)

実際の要素の数へのポインタです。このパラメータは、PL/SQL バインドでだけ必要です。

mode (IN)

このパラメータに有効なモードは次のとおりです。

OCI_DEFAULT - これはデフォルト・モードです。

OCI_DATA_AT_EXEC - このモードが選択される場合、*value_sz* パラメータは、実行時に提供可能なデータの最大サイズを定義します。アプリケーションは、OCI ライブラリのランタイム IN データ・バッファをいつでも何回でも用意することができるように準備する必要があります。ランタイム・データは、次の 2 つの方法のどちらかを使用して用意されます。

- *OCIBindDynamic()* への後続コールによって登録する必要があるユーザー定義関数を使用するコールバック。
- OCI で用意されているコールを使用するポーリング・メカニズム。このモードは、コールバックが定義されていない場合に自動的に選ばれます。

OCI_DATA_AT_EXEC モードの使用方法の詳細は、5-30 ページの「ランタイム・データ割当てとピース単位操作」を参照してください。

割り当てたバッファは、不要になった際にクライアント側で解放する必要があります。

コメント

このコールは、基本バインド操作を実行するときに使用します。バインドは、プログラム変数のアドレスと SQL 文または PL/SQL ブロック内のプレースホルダの関連付けを作成します。このバインド・コールは、バインドされるデータの型も指定し、実行時にデータを用意するメソッドも指示できます。

この関数は、*bindpp* パラメータによって指示されたバインド・ハンドルの暗黙の割当ても行います。***bindpp* に非 NULL ポインタが渡されると、OCI では、*OCIHandleAlloc()* または *OCIBindByPos()* のコールで以前に割り当てられた有効なハンドルを指すとみなします。

OCI アプリケーション内のデータは、プレースホルダに静的または動的にバインドできます。実行の直前にすべての IN バインド・データと OUT バインド・バッファが正しく定義された場合、バインドは静的になります。実行時にクライアント・ライブラリからの要求によりアプリケーションから IN バインド・データと OUT バインド・バッファが供給される場合、バインドは動的になります。動的バインドを指定するには、このコールの *mode* パラメータを *OCI_DATA_AT_EXEC* に設定します。

関連項目：動的バインディングの詳細は、5-30 ページの「ランタイム・データ割当てとピース単位操作」を参照してください。

OCIBindByName() と *OCIBindByPos()* は、パラメータとしてバインド・ハンドルを取りますが、これはバインド・コールによって暗黙的に割り当てられます。アプリケーションでバインドしているすべてのプレースホルダに別個のバインド・ハンドルが割り当てられます。

特定のデータ型をバインドする際または入力データを特定の方法で処理する際に必要となる特定の属性を指定する場合は、次の追加バインド・コールが必要です。

- 構造体の配列が使われている場合、*OCIBindArrayOfStruct()* をコールして必要なスキップ・パラメータを設定します。
- 実行時にデータが動的に提供されている場合、アプリケーションでユーザー定義のコールバック関数を使うときには *OCIBindDynamic()* をコールしてコールバックを登録します。
- 名前付きデータ型が結び付けられている場合、*OCIBindObject()* をコールしてその他の必要情報を指定します。
- RETURNING 句を含む文を使用する場合は、このコールの後に *OCIBindDynamic()* をコールする必要があります。

関連関数

OCIBindDynamic()、*OCIBindObject()*、*OCIBindArrayOfStruct()*

OCIBindDynamic()

用途

このコールは、動的データ割当て用のユーザー・コールバックを登録するときに使用されま
す。

構文

```
sword OCIBindDynamic ( OCIBind      *bindp,  
                      OCIError    *errhp,  
                      dvoid        *ictxp,  
                      OCICallbackInBind (icbfp) /*_  
                        dvoid        *ictxp,  
                        OCIBind      *bindp,  
                        ub4          iter,  
                        ub4          index,  
                        dvoid        **bufpp,  
                        ub4          *alenp,  
                        ub1          *piecep,  
                        dvoid        **indpp */),  
                      dvoid        *octxp,  
                      OCICallbackOutBind (ocbfp) /*_  
                        dvoid        *octxp,  
                        OCIBind      *bindp,  
                        ub4          iter,  
                        ub4          index,  
                        dvoid        **bufpp,  
                        ub4          **alenpp,  
                        ub1          *piecep,  
                        dvoid        **indpp,  
                        ub2          **rcodepp _ */ ) ;
```

パラメータ

bindp (IN/OUT)

OCIBindByName() または *OCIBindByPos()* のコールにより戻されるバインド・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

ictxp (IN)

コールバック関数 *icbfp* に必要なコンテキスト・ポインタです。

icbfp (IN)

実行時に IN バインド値またはピースのポインタを戻すコールバック関数です。コールバックは、次のパラメータを取ります。

ictxp (IN/OUT)

このコールバック関数用のコンテキスト・ポインタ。

bindp (IN)

このバインド変数を一意に指定するために渡されたバインド・ハンドル。

iter (IN)

0 基準の実行繰返値。

index (IN)

現行の配列の索引 (PL/SQL の配列バインド用)。SQL では行の索引です。値は 0 (ゼロ) 以上、バインド・コールの *curletp* パラメータ値以下です。

bufpp (OUT)

バッファまたは記憶領域のポインタ。記述子の場合、*bufpp には記述子を指すポインタが含まれます。たとえば、次のような定義を行うとします。

```
OCILOBLocator *lobp;
```

この場合、*bufpp に *lobp ではなく、lobp を設定します。

REF の場合はそのアドレスを渡します。つまり、*bufpp のかわりに &my_ref を渡します。

OCI_ATTR_CHARSET_ID 属性が OCI_UCS2ID (Unicode) に設定されている場合は、対応するバインド・コールに対して渡されるか、またはそれらのコールを使って受け取るすべてのデータは、UCS-2 コード化されていると見なされます。詳細は、A-20 ページの [OCI_ATTR_CHARSET_ID](#) を参照してください。

alenp (OUT)

バインド値またはバインド・ピースを読み込んだ後それを保存するための OCI 用領域へのポインタ。記述子の場合、記述子にポインタのサイズを渡します。

例 : sizeof(OCILOBLocator *)

piecep (OUT)

バインド値のいずれかのピース。この値は、OCI_ONE_PIECE、OCI_FIRST_PIECE、OCI_NEXT_PIECE、または OCI_LAST_PIECE のいずれかです。ピース単位操作をサポートしないデータ型の場合、OCI_ONE_PIECE を渡す必要があります。OCI_ONE_PIECE を渡さないとエラーになります。

indp (OUT)

標識値を含みます。これは、sb2 値のポインタ、あるいは名前付きデータ型を結び付ける標識構造体のポインタです。

octxp (IN)

コールバック関数 *ocbfp* に必要なコンテキスト・ポインタです。

ocbfp (IN)

実行時に OUT バインド値またはピースのポインタを戻すコールバック関数です。コールバックは、次のパラメータを取ります。

octxp (IN/OUT)

このコールバック関数用のコンテキスト・ポインタです。

bindp (IN)

このバインド変数を一意に指定するために渡されたバインド・ハンドルです。

iter (IN)

0 基準の実行繰返値。

index (IN)

現行の配列の PL/SQL 用索引用 (配列バインド用) です。SQL の場合、現行の繰返しの行番号です。値は 0 (ゼロ) 以上、バインド・コールの *curelep* パラメータ値以下です。

bufpp (OUT)

バインド値またはバインド・ピースを書き込むバッファのポインタ。

OCI_ATTR_CHARSET_ID 属性が OCI_UCS2ID (Unicode) に設定されている場合は、対応するバインド・コールに対して渡されるか、またはそれらのコールを使って受け取るすべてのデータは、UCS-2 コード化されていると見なされます。詳細は、A-20 ページの [OCI_ATTR_CHARSET_ID](#) を参照してください。

alenpp (IN/OUT)

バインド値またはバインド・ピースを読み込んだ後それを保存するための OCI 用領域へのポインタです。

piecep (IN/OUT)

コールバック (アプリケーション) から Oracle へ、次のようにピース値を戻します。

- IN - 値は OCI_ONE_PIECE または OCI_NEXT_PIECE です。
- OUT - IN の値に応じて異なります。

IN 値が OCI_ONE_PIECE の場合、OUT 値は OCI_ONE_PIECE または OCI_FIRST_PIECE です。

IN 値が OCI_NEXT_PIECE の場合、OUT 値は OCI_NEXT_PIECE または OCI_LAST_PIECE です。

indpp (OUT)

標識値 (sb2 値または名前付きデータ型の標識構造体のポインタ) を含むポインタを戻します。

rcodepp (OUT)

リターン・コードを含むポインタを戻します。

コメント

このコールは、*OCIBindByName()* または *OCIBindByPos()* の以前のコールで *OCI_DATA_AT_EXEC* モードが指定された場合に、データの用意または受取り用のユーザー定義コールバック関数を登録するときに使用されます。

コールバック関数ポインタは、コールが成功した場合には *OCI_CONTINUE* を戻します。*OCI_CONTINUE* 以外のリターン・コードは、クライアントが処理の即時停止を求めていることを示します。

OCI_DATA_AT_EXEC モードの詳細は、5-30 ページの「[ランタイム・データ割当てとピース単位操作](#)」を参照してください。

記憶領域のアドレスを渡す際は、コールバックからアプリケーションが復帰した後も必ずその記憶領域が存在するようにしてください。したがって、このような記憶領域はスタック上に割り当てないでください。

関連関数

[OCIBindByName\(\)](#)、[OCIBindByPos\(\)](#)

OCIBindObject()

用途

この関数は、名前付きデータ型（オブジェクト）のバインドに必要な追加属性を設定します。

構文

```
sword OCIBindObject ( OCIBind          *bindp,  
                     OCIError        *errhp,  
                     CONST OCIType   *type,  
                     dvoid           **pgvpp,  
                     ub4             *pvszsp,  
                     dvoid           **indpp,  
                     ub4             *indszp, );
```

パラメータ

bindp (IN/OUT)

OCIBindByName() または *OCIBindByPos()* のコールにより戻されるバインド・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

type (IN)

結び付けられているプログラム変数のタイプを示す TDO を指します。 *OCITypeByName()* のコールにより取り出されます。SQL の REF ではオプションですが、PL/SQL の REF では必須です。

pgvpp (IN/OUT)

プログラム変数バッファのアドレスです。配列では、*pgvpp* はアドレス配列を示します。バインド変数が OUT 変数でもある場合は、オブジェクト・キャッシュ内に OUT 名前付きデータ型値または REF が割り当てられ、REF が戻されます。

pgvpp は、OCI_DATA_AT_EXEC モードが設定されている場合、無視されます。名前付きデータ型バッファは、実行時に要求されます。静的配列バインドでは、*OCIBindArrayOfStruct()* コールを使用してスキップ係数を指定できます。スキップ係数は、値に対する次のポインタのアドレスおよび標識構造体、各自のサイズを計算するために使用されます。

pvszsp (OUT) [optional]

プログラム変数のサイズを指します。名前付きデータ型のサイズは、入力時には必要ありません。配列の場合、*pvszsp* は ub4 の配列です。これは、OUT バインド変数の場合、復帰時に受信した名前付きデータ型および REF のサイズを指します。*pvszsp* は、OCI_DATA_AT_

EXEC モードが設定されている場合は無視されます。この場合、バッファのサイズは実行時に要求されます。

indpp (IN/OUT) [optional]

パラレル標識構造体を含むプログラム変数バッファのアドレスです。配列では、ポインタ配列を示します。バインド変数が OUT バインド変数でもある場合は、オブジェクト・キャッシュにメモリーが割り当てられ、OUT 標識値が格納されます。すべての OUT 値が受け取られた実行終了時には、*indpp* は、新たに割り当てられたこれらの標識構造体のポインタを指しています。SQLT_NTY バインドの場合だけ必要です。*indpp* は、OCI_DATA_AT_EXEC モードが設定されている場合、無視されます。この場合、標識は実行時に要求されます。

indszp (IN/OUT)

IN 標識構造体プログラム変数のサイズを指します。配列では、**ub4** 配列です。OUT バインド変数の場合、戻り時には、受け取った OUT 標識構造体のサイズを指します。*indszp* は、OCI_DATA_AT_EXEC モードが設定されている場合、無視されます。この場合、標識のサイズは実行時に要求されます。

コメント

この関数は、名前付きデータ型（オブジェクト）または REF をバインドする追加属性を設定します。OCI 環境が非オブジェクト・モードで初期化されているときにこの関数をコールした場合は、エラーが戻ります。

この関数は、定義される名前付きデータ型用にデータ型 **OCIType** の型記述子オブジェクト (TDO) をパラメータとして取ります。TDO は、*OCITypeByName()* のコールにより取り出されます。

OCIBindByName() または *OCIBindByPos()* に OCI_DATA_AT_EXEC モードが指定された場合、IN バッファのポインタは、*OCIBindDynamic()* コールに登録されたコールバック *icbfp* を使用するか、*OCIStmtSetPieceInfo()* コールによって取得されます。バッファは、OUT データのために動的に割り当てられ、これらのバッファのポインタは、*OCIBindDynamic()* によって登録された *ocbfp()* をコールするか、*OCIStmtExecute()* が OCI_NEED_DATA を戻したときにコールされる *OCIStmtSetPieceInfo()* によって渡されたバッファにポインタを設定することで戻されます。クライアント・ライブラリが割り当てたバッファのメモリーは、不要になったときに *OCIObjectFree()* コールを使用して解放する必要があります。

関連関数

[OCIBindByName\(\)](#)、[OCIBindByPos\(\)](#)

OCIDefineArrayOfStruct()

用途

このコールは、構造体配列（複数行、複数列）のフェッチで使用される静的な配列定義に必要な追加属性を指定します。

構文

```
sword OCIDefineArrayOfStruct ( OCIDefine   *defnp,  
                               OCIError    *errhp,  
                               ub4         pvskip,  
                               ub4         indskip,  
                               ub4         rlskip,  
                               ub4         rcskip );
```

パラメータ

defnp (IN/OUT)

OCIDefineByPos() のコールにより戻された定義構造体のハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

pvskip (IN)

次のデータ値用のスキップ・パラメータです。

indskip (IN)

次の標識位置のためのスキップ・パラメータです。

rlskip (IN)

次の復帰長さ値のためのスキップ・パラメータです。

rcskip (IN)

次のリターン・コードのためのスキップ・パラメータです。

コメント

このコールは、*OCIDefineByPos()* の後続コールです。

アプリケーションでオブジェクトを含む構造体配列をバインドしている場合、最初に *OCIDefineObject()* をコールして、次に *OCIDefineArrayOfStruct()* をコールする必要があります。

スキップ・パラメータの詳細は、5-18 ページの「[スキップ・パラメータ](#)」を参照してください。

関連関数

OCIDefineByPos()、*OCIDefineObject()*

OCIDefineByPos()

用途

選択リスト内の項目を型と出力データ・バッファに関連付けます。

構文

```
sword OCIDefineByPos ( OCIStmt      *stmtp,
                      OCIDefine    **defnpp,
                      OCIError     *errhp,
                      ub4           position,
                      dvoid         *valuep,
                      sb4           value_sz,
                      ub2           dty,
                      dvoid         *indp,
                      ub2           *rlenp,
                      ub2           *rcodep,
                      ub4           mode );
```

パラメータ

stmtp (IN/OUT)

要求された SQL 問合せ操作へのハンドルです。

defnpp (IN/OUT)

定義ハンドルのポインタへのポインタです。このパラメータを NULL で渡すと、定義ハンドルが暗黙的に割り当てられます。再定義の場合は、非 NULL ハンドルをこのパラメータに渡すことができます。このハンドルは、この列用の定義情報を格納するために使用されます。

注意: ユーザーはこのポインタを記録する必要があります。同じ列位置に対する 2 回目の *OCIDefineByPos()* のコールでは、同じポインタが戻されるとは限りません。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

position (IN)

選択リストでのこの値の位置です。位置は 1 から始まり、左から右へ番号が振られます。たとえば、次の SELECT 文では、

```
SELECT empno, ssn, mgrno FROM employees;
```

empno は位置 1 に、ssn は位置 2 に、mgrno は位置 3 になります。

valuep (IN/OUT)

dtty パラメータに指定されているタイプのバッファまたはバッファの配列のポインタです。単独のフェッチ・コールで複数の行をフェッチしたい場合は、複数のバッファを指定できません。

value_sz (IN)

各 *valuep* バッファのバイト・サイズです。データが VARCHAR2 形式で内部的に格納されている場合、必要な文字数がバイト単位のバッファ・サイズと異なるときは、*OCIAttrSet()* を使用して追加で指定できます。

NLS 変換環境では、指定されたバイト数が小さくて要求された文字数が処理できない場合は切捨てエラーが発生します。

OCI_ATTR_CHARSET_ID 属性が OCI_UCS2ID (Unicode) に設定されている場合は、対応する定義コールに対して渡されるか、またはそれらのコールを使って受け取るすべてのデータは、UCS-2 コード化されていると見なされます。詳細は、A-22 ページの [OCI_ATTR_CHARSET_ID](#) を参照してください。

dtty (IN)

データ型です。名前付きデータ型 (SQLT_NTY) と REF (SQLT_REF) は、環境がオブジェクト・モードで初期化されている場合だけ有効です。データ型コードおよび値については、第 3 章の「データ型」のリストを参照してください。

indp (IN)

標識変数または配列へのポインタです。スカラー・データ型については、sb2 または sb2 の配列へのポインタです。SQLT_NTY 定義は無視されます。SQLT_NTY 定義では、名前付きデータ型の標識構造体または名前付きデータ型の標識構造体配列のポインタは、後続の *OCIDefineObject()* コールによって関連付けられます。

標識変数の詳細は、2-30 ページの「標識変数」を参照してください。

rlenp (IN/OUT)

フェッチされたデータの長さの配列のポインタです。*rlenp* 内の各要素は、フェッチ後の行の中の該当要素内のデータの長さです。

rcodep (OUT)

列レベルのリターン・コードの配列のポインタです。

mode (IN)

次のモードが有効です。

- OCI_DEFAULT - これはデフォルトのモードです。
- OCI_DYNAMIC_FETCH - フェッチするときにデータを動的に割り当てる必要があるアプリケーションでは、このモードを使用する必要があります。ユーザーは、*OCIDefineDynamic()* を追加でコールして、動的に割り当てられたバッファを受け取るた

めに呼び出されるコールバック関数を設定できます。このモードでは、*valuep* および *value_sz* パラメータは無視されます。

コメント

このコールは、Oracle から取り出されたデータを受け取る出力データ・バッファを定義します。この定義は、SELECT 文が OCI アプリケーションにデータを戻すときに必要なローカル・ステップです。

このコールは、選択リスト項目用の定義ハンドルの暗黙の割当ても行います。非 NULL ポインタが **defnpp* に渡されると、OCI では *OCIHandleAlloc()* または *OCIDefineByPos()* のコールで以前に割り当てられた有効なハンドルを指します。別のアドレスに対してハンドルを再定義しているアプリケーションの場合はこれがあてはまるため、複数のフェッチに同じ定義ハンドルを再使用できます。

列をフェッチするための属性の定義は、1 つ以上のコールで実行されます。最初のコールは、フェッチを指定するために必要な最小限の属性を定義する *OCIDefineByPos()* です。

ある種のデータ型またはフェッチ・モードでは、*OCIDefineByPos()* のコールの後に、次の追加定義コールが必要です。

- 複数列を配列フェッチするためのスキップ・パラメータを設定するには、*OCIDefineArrayOfStruct()* のコールが必要です。
- 名前付きデータ型（つまりオブジェクトまたはコレクション）または REF のフェッチに適した属性を設定するには、*OCIDefineObject()* のコールが必要です。この場合、*OCIDefineByPos()* 内のデータ・バッファ・ポインタは無視されます。
- 名前付きデータ型の列を持つ複数の行をフェッチするには、*OCIDefineByPos()* の後に、*OCIDefineArrayOfStruct()* と *OCIDefineObject()* の両方をコールする必要があります。

LOB 定義では、バッファ・ポインタは、*OCIDescriptorAlloc()* コールによって割り当てられる *OCILobLocator* 型の LOB ロケータのポインタでなければなりません。LOB 列には、LOB の値ではなく、常に LOB ロケータが戻ります。LOB 値は、フェッチしたロケータに対して OCI LOB コールを使用するとフェッチできます。これと同じ方式がすべての記述子データ型で使用されます。

NCHAR（固定長および可変長）では、バッファ・ポインタは、必要な NCHAR 文字を保持できるだけのバイト配列を指していなければなりません。

ネストされた表の列は、他のすべての名前付きデータ型と同じように定義およびフェッチされます。

記述子またはロケータの配列を定義するときに、記述子またはロケータのポインタ配列を渡す必要があります。

文字列の配列を定義するときに、文字バッファ配列を渡す必要があります。

このコールの *mode* パラメータに *OCI_DYNAMIC_FETCH* が設定された場合は、クライアント・アプリケーションから実行時にデータを動的にフェッチできます。ランタイム・データは、次の 2 つの方法で用意できます。

- *OCIDefineDynamic()* の後続コールによって登録する必要があるユーザー定義関数を使用するコールバック。クライアント・ライブラリがフェッチしたデータを戻すためにバッファが必要になると、このコールバックが呼び出され、用意されたランタイム・バッファがデータの一部または全部を戻します。
- OCI で用意されているコールを使用するポーリング・メカニズム。このモードは、コールバックが定義されていない場合自動的に選ばれます。この場合、フェッチ・コールにより OCI_NEED_DATA エラー・コードが戻され、データはピース単位のポーリング・メソッドで用意されます。

関連項目 : OCI_DYNAMIC_FETCH モードの使用方法的詳細は、5-30 ページの「[ランタイム・データ割当てとピース単位操作](#)」を参照してください。

定義の詳細は、5-13 ページの「[定義](#)」を参照してください。

関連関数

OCIDefineArrayOfStruct()、*OCIDefineDynamic()*、*OCIDefineObject()*

OCIDefineDynamic()

用途

このコールは、*OCIDefineByPos()* で OCI_DYNAMIC_FETCH モードが選択されたときに必要な追加属性を設定するために使用されます。

構文

```
sword OCIDefineDynamic ( OCIDefine   *defnp,
                        OCIError    *errhp,
                        dvoid       *octxp,
                        OCIcallbackDefine (ocbfp) (/*_
                        dvoid       *octxp,
                        OCIDefine   *defnp,
                        ub4         iter,
                        dvoid       **bufpp,
                        ub4         **alenpp,
                        ub1         *piecep,
                        dvoid       **indpp,
                        ub2         **rcodep _*/) );
```

パラメータ

defnp (IN/OUT)

OCIDefineByPos() のコールにより戻される定義構造体へのハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

octxp (IN)

コールバック関数のコンテキストを指します。

ocbfp (IN)

コールバック関数を指します。この関数は、検索されたフェッチ・データまたはそのピースを格納するバッファを指すポインタを取得するために実行時に起動されます。このコールバックは、標識およびリターン・コード、データ・ピースと標識の長さも指定します。

警告: コールバック・パラメータを使うときは、パラメータ・モードにとって IN と OUT がどのような意味を持つかということを念頭に置いて作業する必要があります。通常、OCI 関数では、IN パラメータは Oracle に渡すデータを表し、OUT パラメータは Oracle から戻されるデータを表します。コールバックの場合、これが逆になります。IN は Oracle からコールバックに渡されるデータ、OUT はコールバックから Oracle に渡されるデータです。

以下にコールバック・パラメータをリストします。

octxp (IN/OUT)

すべてのコールバック関数へ引数として渡されるコンテキスト・ポインタです。

defnp (IN)

定義ハンドルです。

iter (IN)

この現行のフェッチのいずれかの列（基準 0（ゼロ））です。

bufpp (OUT)

列値を格納するバッファのポインタを Oracle に戻します。つまり、*bufpp は列値の適切な記憶領域を示します。

alenpp (IN/OUT)

これは、*bufpp に提供している記憶領域のサイズを設定するために、アプリケーションで使います。データがバッファにフェッチされると、alenpp はデータの実サイズを示します。

piecep (IN/OUT)

コールバック（アプリケーション）から Oracle へ、次のようにピース値を戻します。

- IN - 値は OCI_ONE_PIECE または OCI_NEXT_PIECE です。
- OUT - IN の値に応じて異なります。

IN 値が OCI_ONE_PIECE の場合、OUT 値は OCI_ONE_PIECE または OCI_FIRST_PIECE です。

IN 値が OCI_NEXT_PIECE の場合、OUT 値は OCI_NEXT_PIECE または OCI_LAST_PIECE です。

indpp (IN)

標識変数ポインタ。

rancodep (IN)

リターン・コード変数ポインタ。

コメント

このコールは、*OCIDefineByPos()* のコールで OCI_DYNAMIC_FETCH モードが選択されている場合、必要な追加属性を設定するために使用されます。OCI_DYNAMIC_FETCH モードが選択されているときに *OCIDefineDynamic()* のコールがスキップされた場合、アプリケーションでは、OCI コール（*OCIStmtGetPieceInfo()* および *OCIStmtSetPieceInfo()*）を使ってデータ・ピース単位がフェッチされます。OCI_DYNAMIC_FETCH モードの詳細は、5-30 ページの「ランタイム・データ割当てとピース単位操作」を参照してください。

関連関数

[OCIDefineByPos\(\)](#)

OCIDefineObject()

用途

名前付きデータ型または REF の定義のために必要な追加属性を設定します。

構文

```
sword OCIDefineObject ( OCIDefine      *defnp,  
                        OCIError       *errhp,  
                        CONST OCIType   *type,  
                        dvoid          **pgvpp,  
                        ub4            *pvszsp,  
                        dvoid          **indpp,  
                        ub4            *indszp );
```

パラメータ

defnp (IN/OUT)

以前 *OCIDefineByPos()* のコールで割り当てられた定義ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

type (IN) [optional]

プログラム変数のタイプを示す型記述子オブジェクト (TDO) を指します。型 *SQLT_NTY* のプログラム変数用にだけ使用されます。このパラメータはオプションであり、使用しない場合は *NULL* として渡せます。

pgvpp (IN/OUT)

プログラム変数バッファのポインタを指します。配列では、*pgvpp* はポインタ配列を示します。フェッチした名前付きデータ型インスタンス用のメモリーは、オブジェクト・キャッシュ内に動的に割り当てられます。すべての値を受け取ったフェッチの終了時には、*pgvpp* は、新たに割り当てられた名前付きデータ型インスタンスのポインタを示しています。名前付きデータ型インスタンスが不要になった際には、アプリケーションから *OCIObjectFree()* をコールして、これらの割当てを解除する必要があります。

注意: アプリケーションにおいてバッファが暗黙のうちにキャッシュに割り当てられるようにするには、**pgvpp* を *NULL* として渡す必要があります。

pvszsp (IN/OUT)

プログラム変数のサイズを指します。配列では、*ub4* 配列です。

indpp (IN/OUT)

パラレル標識構造体を含むプログラム変数バッファのポインタを指します。配列では、ポインタ配列を示します。標識構造体を格納するためのメモリーがオブジェクト・キャッシュ内

に割り当てられます。すべての値を受け取ったフェッチの終了時には、*indpp* は、新たに割り当てられた標識構造体のポインタを示しています。

indszp (IN/OUT)

標識構造体プログラム変数のサイズを指します。配列では、**ub4** 配列です。

コメント

この関数は、初期定義情報を設定する *OCIDefineByPos()* の後続コールです。このコールは、名前付きデータ型定義のために必要な追加属性を設定します。OCI 環境が非オブジェクト・モードで初期化されているときにこの関数をコールした場合は、エラーが戻ります。

この関数は、定義される名前付きデータ型用にデータ型 **OCIType** の型記述子オブジェクト (TDO) をパラメータとして取ります。TDO は、*OCIDescribeAny()* をコールして取り出すことができます。

関連項目 : OCI プロセス環境の初期化の詳細は、15-94 ページの *OCIInitialize()* の説明を参照してください。

関連関数

[OCIDefineByPos\(\)](#)

OCIDescribeAny()

用途

既存のスキーマ・オブジェクトおよびサブスキーマ・オブジェクトを記述します。

構文

```
sword OCIDescribeAny ( OCISvcCtx      *svchp,
                      OCIError      *errhp,
                      dvoid          *objptr,
                      ub4            objptr_len,
                      ub1            objptr_typ,
                      ub1            info_level,
                      ub1            objtyp,
                      OCIDescribe   *dschp );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

objptr (IN)

このパラメータは次のいずれかです。

1. 記述するオブジェクトの名前を格納する文字列
2. TDO に対する REF のポインタ (型用)
3. TDO のポインタ (型用)

上記の各値は、*objptr_typ* に適切な値を渡すことで区別されます。このパラメータは非 NULL でなければなりません。

1 の場合は、オブジェクト名を格納する文字列は、*scott.emp.empno@mydb* のように、*<name1>[.<name2>.<name3>...][@<linkname>]* の形式でなければなりません。データベース・リンクは Oracle8i データベースにだけ許可されています。オブジェクト名は、次の SQL ルールによって解釈されます。

- *<name1>* が入力され、*objtyp* が *OCI_PTYPE_SCHEMA* の場合にだけ、名前は、指定されたスキーマを参照します。Oracle データベースは、リリース 8.1 以降でなければなりません。
- *<name1>* が入力され、*objtyp* が *OCI_PTYPE_DATABASE* の場合にだけ、名前は、指定されたデータベースを参照します。*database_name@db_link_name* を使ってリモート・

データベースを記述する場合は、リモート Oracle データベースはリリース 8.1 以降でなければなりません。

- <name1> が入力され、objtyp が OCI_PTYPE_SCHEMA または OCI_PTYPE_DATABASE 以外の場合にだけ、名前は、現行ユーザーの現行スキーマ内の名前付きオブジェクト（表 / ビュー / プロシージャ / ファンクション / パッケージ / 型 / シノニム / シーケンス）を参照します。Oracle7 のサーバーに接続したときは、有効な型はプロシージャおよびファンクションだけです。
- <name1>.<name2>.<name3>... が入力された場合は、オブジェクト名は、<name1> というスキーマ内のスキーマ・オブジェクトまたはサブスキーマ・オブジェクトを参照します。たとえば、文字列 scott.emp.deptno では、scott はスキーマの名前、emp はスキーマ内の表の名前で、deptno は表内の列の名前です。

objnm_len (IN)

objptr が指す名前文字列の長さです。名前が渡される場合、0（ゼロ）以外でなければなりません。objptr が TDO またはその REF のポインタの場合 0（ゼロ）でもかまいません。

objptr_typ (IN)

objptr に渡されるオブジェクトのタイプです。次の値が有効です。

- OCI_OTYPE_NAME、objptr がスキーマ・オブジェクトの名前を示す場合
- OCI_OTYPE_REF、objptr が TDO に対する REF のポインタの場合
- OCI_OTYPE_PTR、objptr が TDO のポインタの場合

info_level (IN)

今後の拡張要素のために確保されています。OCI_DEFAULT を渡します。

objtyp (IN/OUT)

記述されるスキーマ・オブジェクトのタイプです。次の値が有効です。

- OCI_PTYPE_TABLE、表用
- OCI_PTYPE_VIEW、ビュー用
- OCI_PTYPE_PROC、プロシージャ用
- OCI_PTYPE_FUNC、ファンクション用
- OCI_PTYPE_PKG、パッケージ用
- OCI_PTYPE_TYPE、型用
- OCI_PTYPE_SYN、シノニム用
- OCI_PTYPE_SEQ、シーケンス用
- OCI_PTYPE_SCHEMA、スキーマ用
- OCI_PTYPE_DATABASE、データベース用

- OCI_PTYPE_UNK、不明なスキーマ・オブジェクト用

この引数の値は必ず指定しなければなりません。OCI_PTYPE_UNK を指定すると、該当するオブジェクトが存在する場合、現行のスキーマ内の指定した名前のオブジェクトの記述が、実際のオブジェクト型とともに戻されます。

dschp (IN/OUT)

コール後のオブジェクトに関する記述情報とともに移入されている記述ハンドルです。非 NULL でなければなりません。

コメント

このコールは、表、ビュー、シノニム、プロシージャ、ファンクション、パッケージ、シーケンス、および型などの既存のスキーマ・オブジェクトを記述する記述コールの総称です。このコールでは、表内の列などのサブスキーマ・オブジェクトも記述できます。このコールは、*OCIAttrGet()* コールを使用して取得できるオブジェクト固有の属性を記述ハンドルに移入します。

記述ハンドルに対する *OCIParamGet()* は、指定位置のパラメータ記述子を戻します。パラメータ位置は 1 から開始します。パラメータ記述子に対して *OCIAttrGet()* をコールすると、ストアド・プロシージャまたはファンクション・パラメータの特定の属性、あるいは表の列記述子が戻されます。*OCIDescribeAny()* によってスキーマ・オブジェクト記述全体がクライアント側にキャッシュされているため、これらの後続コールは、サーバーへのラウンドトリップを別に行う必要がありません。記述ハンドルに対する *OCIAttrGet()* は、位置の総数も戻します。

記述ハンドルに対して OCI_ATTR_DESC_PUBLIC 属性が設定されている場合、現行のスキーマに名前付きオブジェクトが存在せず <name1> だけが指定されているときは、このオブジェクトはパブリック・シノニムとして参照されます。

記述操作の詳細は、[第 6 章の「スキーマ・メタデータの記述」](#)を参照してください。

関連関数

[OCIAttrGet\(\)](#)、[OCIParamGet\(\)](#)

OCIStmtGetBindInfo()

用途

バインドおよび標識変数名を取得します。

構文

```
sword OCIStmtGetBindInfo ( OCIStmt      *stmtp,
                          OCIError     *errhp,
                          ub4          size,
                          ub4          startloc,
                          sb4          *found,
                          text         *bvnp[] ,
                          ub1          bvn1[] ,
                          text         *invp[] ,
                          ub1          inpl[] ,
                          ub1          dupl[] ,
                          OCIBind      *hndl[] );
```

パラメータ

stmtp (IN)

文ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

size (IN)

各配列の要素数です。

startloc (IN)

バインド情報の取得を開始するバインド変数の位置です。

found (IN)

絶対値関数 (*found*) は、開始位置にかかわらず、文のバインド変数の合計数値を与えます。戻されたバインド変数の数が用意された *size* よりも少ない場合は正数に、そうでない場合は負数になります。

bvnp (OUT)

バインド変数名を保持するポインタの配列です。

bvn1 (OUT)

各 *bvnp* 要素の長さを保持する配列です。

invp (OUT)

標識変数名を保持するポインタの配列です。

inpl (OUT)

各 *invp* 要素の長さを保持するポインタの配列です。

dupl (OUT)

バインド位置が複製であるかどうかにより、その要素値が 0 または 1 になる配列です。

hndl (OUT)

バインド位置のバインドが完了している場合にバインド・ハンドルを戻す配列です。重複では、ハンドルは戻りません。

コメント

このコールは、文が準備された後にバインド変数に関する情報を戻します。バインド名および標識名、重複バインドかどうかなどの情報があります。このコールでは、存在する場合は関連するバインド・ハンドルも戻します。重複しないバインド変数の個数ではなく、バインド変数の総数を *found* パラメータに設定します。

SELECT INTO リスト変数はバインドとはみなされていないため、この関数には含まれません。

このコールの前に、*OCISstmtPrepare()* のコールを使用して文を準備する必要があります。

このコールは、ローカルで処理されます。

関連関数

[OCISstmtPrepare\(\)](#)

ダイレクト・パス・ロード関数

この項では、ダイレクト・パス・ロード関数について説明します。

表 15-4 OCI クイック・リファレンス

関数	用途	ページ
OCIDirPathAbort()	ダイレクト・パス操作を中断します。	15-74 ページ
OCIDirPathColArrayEntryGet()	列配列内の特定のエントリを取得します。	15-75 ページ
OCIDirPathColArrayEntrySet()	列配列内の特定のエントリを特定の値に設定します。	15-77 ページ
OCIDirPathColArrayRowGet()	特定の行番号の基本行ポインタを取得します。	15-79 ページ
OCIDirPathColArrayReset()	行配列の状態をリセットします。	15-80 ページ
OCIDirPathColArrayToStream()	列配列からダイレクト・パス・ストリーム形式に変換します。	15-81 ページ
OCIDirPathFinish()	ロードされたデータを終了およびコミットします。	15-83 ページ
OCIDirPathLoadStream()	ダイレクト・パス・ストリーム形式に変換されたデータをロードします。	15-84 ページ
OCIDirPathPrepare()	行の変換またはロードを行うために、ダイレクト・パス・インタフェースを準備します。	15-86 ページ
OCIDirPathStreamReset()	ダイレクト・ストリームの状態をリセットします。	15-87 ページ

OCIDirPathAbort()

用途

ダイレクト・パス操作を終了します。

構文

```
sword OCIDirPathAbort ( OCIDirPathCtx      *dpctx,  
                        OCIErr          *errhp );
```

パラメータ

dpctx (IN)

ダイレクト・パス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrGet()* に渡せるエラー・ハンドルです。

コメント

ダイレクト・パス操作の代理として、サーバーによってメンテナンスされていたすべての状態は、中断によって破棄されます。ダイレクト・パス・ロードの場合は、中断前にロードされたデータは、問合せでは参照できません。ただし、これらのデータにより、ロードされているセグメント内で空き領域が消費されることがあります。索引メンテナンス操作など、ロード完了操作は行われません。

関連関数

[OCIDirPathFinish\(\)](#)、[OCIDirPathLoadStream\(\)](#)、[OCIDirPathPrepare\(\)](#)、
[OCIDirPathLoadStream\(\)](#)、[OCIDirPathStreamReset\(\)](#)

OCIDirPathColArrayEntryGet()

用途

列配列内の特定のエントリを取得します。

構文

```
sword OCIDirPathColArrayEntryGet ( OCIDirPathColArray *dpca,  
                                   OCIError          *errhp,  
                                   ub4                rownum,  
                                   ub2                colIdx,  
                                   ub1                **cvalpp,  
                                   ub4                *clenp,  
                                   ub1                *cflgp );
```

パラメータ

dpca (IN/OUT)

ダイレクト・パス列配列ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

rownum (IN)

ゼロベースの行オフセットです。

colIdx (IN)

列の識別子 (索引) です。この列 ID は *OCIDirPathColAttrSet()* によって戻されます。

cvalpp (IN/OUT)

列データへのポインタのポインタです。

clenp (IN/OUT)

列データの長さへのポインタです。

cflgp (IN/OUT)

列フラグへのポインタです。

次のいずれかの値が戻されます。

- OCI_DIRPATH_COL_COMPLETE - 列のすべてのデータが存在します。
- OCI_DIRPATH_COL_NULL - 列が NULL です。
- OCI_DIRPATH_COL_PARTIAL - 一部の列データが戻されます。

コメント

cflgp に NULL が設定されている場合は、この操作では *cvalp* および *clenp* パラメータは設定されません。

関連関数

[OCIDirPathColArrayEntrySet\(\)](#)、[OCIDirPathColArrayRowGet\(\)](#)、[OCIDirPathColArrayReset\(\)](#)、[OCIDirPathColArrayToStream\(\)](#)

OCIDirPathColArrayEntrySet()

用途

列配列内の特定のエントリを特定の値に設定します。

構文

```
sword OCIDirPathColArrayEntrySet ( OCIDirPathColArray *dpca,  
                                     OCIError          *errhp,  
                                     ub4                rownum,  
                                     ub2                colIdx,  
                                     ub1                *cvalp,  
                                     ub4                clen,  
                                     ub1                cflg );
```

パラメータ

dpca (IN/OUT)

ダイレクト・パス列配列ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

rownum (IN)

ゼロベースの行オフセットです。

colIdx (IN)

列の識別子 (索引) です。この列 ID は *OCIDirPathColAttrSet()* によって戻されます。

cvalp (IN)

列データへのポインタです。

clen (IN)

列データの長さです。

cflg (IN)

列フラグです。次のいずれかの値が戻されます。

- OCI_DIRPATH_COL_COMPLETE - 列のすべてのデータが存在します。
- OCI_DIRPATH_COL_NULL - 列が NULL です。
- OCI_DIRPATH_COL_PARTIAL - 一部の列データが戻されます。

コメント

cflg に NULL が設定されている場合は、*cval* および *clen* パラメータは使われません。

例

この例では、列配列内の最初の行のデータのソースに *addr*、長さに *len* を設定します。この例では、列は、*colld* によって識別されます。

```
err = OCIDirPathColArrayEntrySet (dpca, errhp, (ub2)0, colld, addr, len,  
                                   OCI_DIRPATH_COL_COMPLETE);
```

関連関数

[OCIDirPathColArrayEntryGet\(\)](#)、[OCIDirPathColArrayRowGet\(\)](#)、[OCIDirPathColArrayReset\(\)](#)、[OCIDirPathColArrayToStream\(\)](#)

OCIDirPathColArrayRowGet()

用途

特定の行番号の列配列行ポインタを取得します。

構文

```
sword OCIDirPathColArrayRowGet ( OCIDirPathColArray *dpca,  
                                OCIError          *errhp,  
                                ub4              rownum,  
                                ub1              ***cvalppp,  
                                ub4              **clenpp,  
                                ub1              **cflgpp );
```

パラメータ

dpca (IN/OUT)

ダイレクト・パス列配列ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

rownum (IN)

ゼロベースの行オフセットです。

cvalppp (IN/OUT)

列データのポインタのベクトルへのポインタです。

clenpp (IN/OUT)

列データの長さのベクトルへのポインタです。

cflgpp (IN/OUT)

列フラグのベクトルへのポインタです。

コメント

アプリケーションでは、簡単なポインタ計算をして特定の行の列内で反復処理を行います。このインタフェースを使うと、各列で *OCIDirPathColArrayEntrySet()* をコールする場合と比べて、行の列配列エントリの取得または設定を効率的に行うことができます。アプリケーションから、列配列の境界を越えてメモリーの参照を解除しないようにする必要があります。列配列の次元は、列配列の属性として取得できます。

関連関数

[OCIDirPathColArrayEntryGet\(\)](#)、[OCIDirPathColArrayEntrySet\(\)](#)、[OCIDirPathColArrayReset\(\)](#)、[OCIDirPathColArrayToStream\(\)](#)

OCIDirPathColArrayReset()

用途

列配列の状態をリセットします。

構文

```
sword OCIDirPathColArrayReset ( OCIDirPathColArray *dpca,  
                                OCIError *errhp );
```

パラメータ

dpca (IN)

ダイレクト・パス列配列ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

列配列の状態は、大きな列内でピース単位操作を行うとき、および列のロード中にエラーが発生したときに、リセットする必要があります。

関連関数

OCIDirPathColArrayEntryGet()、*OCIDirPathColArrayEntrySet()*、
OCIDirPathColArrayRowGet()、*OCIDirPathColArrayToStream()*

OCIDirPathColArrayToStream()

用途

列配列形式からダイレクト・パス・ストリーム形式に変換します。

構文

```
sword OCIDirPathColArrayToStream ( OCIDirPathColArray      *dpca,
                                   OCIDirPathCtx  const  *dpctx,
                                   OCIDirPathStream *dpstr,
                                   OCIError        *errhp,
                                   ub4              rowcnt,
                                   ub4              rowoff );
```

パラメータ

dpca (IN)

ダイレクト・パス列配列ハンドルです。

dpctx (IN)

ロードされているオブジェクトのダイレクト・パス・コンテキスト・ハンドルです。

dpstr (IN/OUT)

ダイレクト・パス・ストリーム・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

rowcnt (IN)

列配列内の行数です。

rowoff (IN)

列配列内の開始索引です。

コメント

このインタフェースは、*OCIDirPathColAttrSet()* によって指定された外部形式の列配列表現データを、ダイレクト・パス・ストリーム形式に変換するときに使います。変換された形式は、*OCIDirPathLoadStream()* を使ったロードに適しています。

ダイレクト・パス・ストリーム形式の列データは、Oracle の内部表現のデータに変換されません。すべての変換は、この 2 タスク・インタフェースのクライアント側で行われ、変換エラーは、このインタフェースへのコールと同時に発生します。エラーが発生した行および列に関する情報は、列配列ハンドルの属性として取得できます。

スレッド環境では、同時に行われる *OCIDirPathColArrayToStream()* 操作によって同じダイレクト・パス・コンテキスト・ハンドルが参照されることがあります。ただし、このインタフェースでは、ダイレクト・パス・コンテキスト・ハンドルは変更されません。

このコールのリターン・コードは次のとおりです。

- OCI_SUCCESS - 列配列内のすべてのデータがストリーム形式に正常に変換されました。列配列属性 OCI_ATTR_ROW_COUNT は、最後に正常に変換された行の列配列への行索引です。
- OCI_ERROR - 変換中にエラーが発生しました。エラー・ハンドルにエラー情報が格納されます。列配列属性 OCI_ATTR_ROW_COUNT または OCI_ATTR_COL_COUNT には、エラーの原因となった行または列、およびその行または列の列配列の索引が格納されます。
- OCI_CONTINUE - 列配列内の一部のデータが、ストリーム形式に変換できませんでした。ストリーム・バッファの領域が不足しているため、すべての列配列データを格納できません。コール側は、データをロードしてファイルに保存するか、または別のストリームを使って *OCIDirPathArrayToStream()* を再度コールし、残りの列配列データを変換する必要があります。列配列には、変換を再開する場所を示す内部状態が格納されています。列配列属性 OCI_ATTR_ROW_COUNT には、完全に変換されなかった行の列配列に対する、行索引が格納されています。
- OCI_NEED_DATA - 列配列内のすべてのデータが正常に変換されましたが、列の一部が見つかりました。コール側では、変換後のストリームをロードし、その行の残りの部分を変換する必要があります。必要に応じて操作を反復します。列配列属性 OCI_ATTR_ROW_COUNT または OCI_ATTR_COL_COUNT には、一部であるとマークされた行または列、およびその行または列の列配列の索引が格納されています。

関連関数

OCIDirPathColArrayEntryGet()、*OCIDirPathColArrayEntrySet()*、
OCIDirPathColArrayRowGet()、*OCIDirPathColArrayReset()*

OCIDirPathFinish()

用途

ダイレクト・パス・ロード操作を終了します。

構文

```
sword OCIDirPathFinish ( OCIDirPathCtx      *dpctx,  
                        OCIError          *errhp );
```

パラメータ

dpctx (IN)

ロードされるオブジェクトのダイレクト・パス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

ロードが完了して、ロードされたデータがコミットされるときに、ダイレクト・パス終了関数がコールされます。

戻り値 OCI_SUCCESS は、バックエンドによるロードが正常に終了したことを示します。

関連関数

[OCIDirPathAbort\(\)](#)、[OCIDirPathLoadStream\(\)](#)、[OCIDirPathPrepare\(\)](#)、
[OCIDirPathLoadStream\(\)](#)、[OCIDirPathStreamReset\(\)](#)

OCIDirPathLoadStream()

用途

ダイレクト・パス・ストリーム形式に変換されたデータをロードします。

構文

```
sword OCIDirPathLoadStream ( OCIDirPathCtx      *dpctx,  
                             OCIDirPathStream   *dpstr,  
                             OCIError           *errhp );
```

パラメータ

dpctx (IN)

ロードされるオブジェクトのダイレクト・パス・コンテキスト・ハンドルです。

dpstr (IN)

ロードするストリームのダイレクト・パス・ストリーム・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

インタフェースからエラーが戻されたときは、ストリームのソースとなった列配列内の列に関する情報を、ダイレクト・パス・ストリームの属性として取得できます。また、エラーが発生したストリーム内のオフセットも、ストリームの属性として取得できます。

この関数のリターン・コードは次のとおりです。

- OCI_SUCCESS - ストリーム内のすべてのデータが正常にロードされました。
- OCI_ERROR - データのロード中にエラーが発生しました。問題は、パーティションのマッピング・エラー、NULL 制約の違反、ファンクション索引評価エラー、またはエクステントを割り当てられないなどの領域条件の不足である可能性があります。ダイレクト・パス・ストリームの属性 OCI_ATTR_STREAM_OFFSET は、違反している行に対応するストリームのオフセットです。ストリームのソースが列配列の場合は、属性 OCI_ATTR_ROW_COUNT には、不正な行に対応する列配列の行索引が格納されます。
- OCI_NEED_DATA - 最後の行が完全な行ではありませんでした。コール側は、行の残りをロードする必要があります。ストリームのソースが列配列の場合は、属性 OCI_ATTR_ROW_COUNT または OCI_ATTR_COL_COUNT には、その一部の行または列に対応する行または列、およびその行または列の列配列の索引が格納されます。
- OCI_NO_DATA - 空のストリームまたは完全に処理されているストリームをロードしようとした。ストリームのソースが列配列の場合は、属性 OCI_ATTR_ROW_COUNT または OCI_ATTR_COL_COUNT には、不正な行または列に対応する行または列、およびその行または列の列配列の索引が格納されます。

関連関数

[OCIDirPathAbort\(\)](#)、[OCIDirPathFinish\(\)](#)、[OCIDirPathLoadStream\(\)](#)、[OCIDirPathPrepare\(\)](#)、[OCIDirPathStreamReset\(\)](#)

OCIDirPathPrepare()

用途

行の変換またはロードを行う前に、ダイレクト・パス・ロード・インタフェースを準備します。

構文

```
sword OCIDirPathPrepare (   OCIDirPathCtx      *dpctx,  
                           OCISvcCtx          *svchp,  
                           OCIError           *errhp );
```

パラメータ

dpctx (IN)

ロードされるオブジェクトのダイレクト・パス・コンテキスト・ハンドルです。

svchp (IN)

サービス・コンテキストです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

操作するオブジェクトの名前、列データの外部属性、およびすべてのロード・オプションの設定が終了したら、行の変換またはロードの前に *OCIDirPathPrepare()* を使ってダイレクト・パス・インタフェースを準備する必要があります。

戻り値 OCI_SUCCESS は、ダイレクト・パス・ロード操作を行うために、バックエンドが適切に初期化されたことを示しています。ゼロ以外の戻り値は、エラーを示します。戻される可能性のあるエラーは次のとおりです。

- コンテキストが無効です。
- サーバーに接続されていません。
- オブジェクト名が設定されていません。
- 既に準備されています。(複数回準備することはできません。)
- オブジェクトがダイレクト・パス操作に適切ではありません。

関連関数

OCIDirPathAbort()、*OCIDirPathFinish()*、*OCIDirPathLoadStream()*、*OCIDirPathLoadStream()*、*OCIDirPathStreamReset()*

OCIDirPathStreamReset()

用途

ダイレクト・パス・ストリームの状態をリセットします。

構文

```
sword OCIDirPathStreamReset ( OCIDirPathStream      *dpstr,  
                             OCIError              *errhp,
```

パラメータ

dpstr (IN)

ダイレクト・パス・ストリーム・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

ダイレクト・パス・ストリームでは、次の *OCIDirPathColArrayToStream()* がストリームに書込みを開始する場所を示す状態をメンテナンスします。通常、データはストリームの末尾に追加されます。ストリームが正常にロードされた後にコール側から新しいストリームを開始する場合、またはストリーム内のデータを破棄する場合は、このコールを使ってストリームをリセットする必要があります。

関連関数

[OCIDirPathAbort\(\)](#)、[OCIDirPathFinish\(\)](#)、[OCIDirPathLoadStream\(\)](#)、[OCIDirPathPrepare\(\)](#)、[OCIDirPathLoadStream\(\)](#)

接続、認証および初期化関数

この項では、OCI 接続関数、認証関数および初期化関数について説明します。

表 15-5 OCI クイック・リファレンス

関数	用途	ページ
OCIEnvCreate()	OCI 環境を作成および初期化します。	15-89 ページ
OCIEnvInit()	環境ハンドルを初期化します。	15-92 ページ
OCIInitialize()	OCI プロセス環境を初期化します。	15-94 ページ
OCILogon()	単純なシングルセッション・ログイン。	15-98 ページ
OCIServerAttach()	サーバーに接続します。サーバー・コンテキスト・ハンドルを初期化します。	15-100 ページ
OCIServerDetach()	サーバーとの接続を切ります。サーバー・コンテキスト・ハンドルの初期化を解除します。	15-102 ページ
OCISessionBegin()	ユーザーを認証します。	15-103 ページ
OCISessionEnd()	ユーザー・セッションを終了します。	15-106 ページ
OCITerminate()	共有メモリー・サブシステムから連結解除します。	15-107 ページ

OCIEnvCreate()

用途

OCI 関数が実行される環境を作成および初期化します。

構文

```

sword OCIEnvCreate ( OCIEnv      **envhpp,
                    ub4         mode,
                    CONST dvoid *ctxp,
                    CONST dvoid *(*malocfp)
                    (dvoid *ctxp,
                     size_t size),
                    CONST dvoid *(*ralocfp)
                    (dvoid *ctxp,
                     dvoid *memptr,
                     size_t newsize),
                    CONST void (*mfreefp)
                    (dvoid *ctxp,
                     dvoid *memptr))
                    size_t      xtramensz,
                    dvoid      **usrmempp );

```

パラメータ

envhpp (OUT)

環境へのハンドルのポインタです。

mode (IN)

モードの初期化を指定します。次のモードが有効です。

- OCI_DEFAULT - デフォルト・モードを使います。
- OCI_THREADED - スレッド環境を使います。ユーザーに隠されている内部データ構造体が複数スレッドで同時にアクセスできないように保護されます。
- OCI_OBJECT - オブジェクト機能を使います。
- OCI_SHARED - 共有データ構造を利用します。
- OCI_EVENTS - パブリッシュ・サブスクライブ通知を利用します。
- OCI_NO_UCB - 動的コールバック・ルーチン *OCIEnvCallback* のコールを抑制します。デフォルトの動作では、環境の作成時に *OCIEnvCallback* のコールが許可されます。詳細は、9-14 ページの「[動的なコールバック登録](#)」を参照してください。

- **OCI_ENV_NO_MUTEX** - このモードでは mutex 化されません。環境ハンドル、または環境ハンドルから導出されたハンドルで行われたすべての OCI コールは、直列化する必要があります。

ctxp (IN)

メモリー・コールバック・ルーチンのユーザー定義コンテキストを指定します。

malocfp (IN)

ユーザー定義のメモリー割当て関数を指定します。モードが OCI_THREADED の場合、このメモリー割当てルーチンは、スレッド・セーフでなければなりません。

ctxp (IN)

ユーザー定義のメモリー割当て関数の、コンテキスト・ポインタを指定します。

size (IN)

ユーザー定義のメモリー割当て関数によって割り当てられるメモリーのサイズを指定します。

ralocfp (IN)

ユーザー定義のメモリー再割当て関数を指定します。モードが OCI_THREADED の場合、このメモリー割当てルーチンは、スレッド・セーフでなければなりません。

ctxp (IN)

ユーザー定義のメモリー再割当て関数の、コンテキスト・ポインタを指定します。

memp (IN)

メモリー・ブロックのポインタです。

newsize (IN)

新しく割り当てられるメモリーのサイズを指定します。

mfreefp (IN)

ユーザー定義のメモリー解放関数を指定します。モードが OCI_THREADED の場合、このメモリー解放ルーチンは、スレッド・セーフでなければなりません。

ctxp (IN)

ユーザー定義のメモリー解放関数の、コンテキスト・ポインタを指定します。

memptr (IN)

解放されるメモリーのポインタです。

xtramemsz (IN)

環境の存続期間中に割り当てられるユーザー・メモリーの量を指定します。

usrmempp (OUT)

コールによりユーザーに割り当てられた、サイズ *xtramemsz* のユーザー・メモリーのポインタを戻します。

コメント

このコールにより、ユーザーによって指定されたモードを使って、すべての OCI コールの環境が作成されます。このコールは、他の OCI コールより先にコールする必要があります。また、*OCIInitialize()* および *OCIEnvInit()* コールのかわりに使ってください。*OCIInitialize()* および *OCIEnvInit()* コールは、下位互換性を保つためにサポートされます。

このコールでは、残りの OCI 関数で使われる環境ハンドルが戻されます。OCI には、それぞれ独自の環境モードを持つ複数の環境が存在する可能性があります。この関数では、どのモードで初期化を要求されても、プロセス・レベルの初期化を実行します。たとえば、環境を *OCI_THREADED* で初期化した場合は、OCI で使われるすべてのライブラリもそのスレッド・モードで初期化されます。

DLL または共有ライブラリを記述している場合は、このコールを使い、*OCIInitialize()* および *OCIEnvInit()* コールは使わないでください。

xtramemsz パラメータおよびユーザー・メモリー割当ての詳細は、2-13 ページの「[ユーザー・メモリーの割当て](#)」を参照してください。

関連関数

[OCIHandleAlloc\(\)](#)、[OCIHandleFree\(\)](#)、[OCIEnvInit\(\)](#)、[OCIInitialize\(\)](#)、[OCITerminate\(\)](#)

OCIEnvInit()

用途

OCI 環境ハンドルを作成および初期化します。

構文

```
sword OCIEnvInit ( OCIEnv      **envhpp,  
                  ub4         mode,  
                  size_t      xtramemsz,  
                  dvoid       **usrmempp );
```

パラメータ

envhpp (OUT)

環境へのハンドルのポインタです。

mode (IN)

環境モードの初期化を指定します。次のモードが有効です。

- OCI_DEFAULT
- OCI_NO_MUTEX
- OCI_ENV_NO_UCB

OCI_DEFAULT モードの場合、OCI ライブラリでハンドルが常に mutex されます。OCI_NO_MUTEX モードの場合、この環境では mutex 化されません。

OCI_NO_MUTEX モードでは、環境ハンドル、または環境ハンドルから導出されたハンドルで行われたすべての OCI コールは、直列化する必要があります。直列化するには、独自の mutex 化を行うか、または環境ハンドル上で操作中のスレッドを 1 つだけにします。

OCI_ENV_NO_UCB モードは、環境の初期化時に動的コールバック・ルーチン *OCIEnvCallback* のコールを抑止するために使います。デフォルトでは、このコールは抑止されません。詳細は、9-14 ページの「[動的なコールバック登録](#)」を参照してください。

xtramemsz (IN)

環境の存続期間中に割り当てられるユーザー・メモリーの量を指定します。

usrmempp (OUT)

環境の存続期間中コールによってユーザー用に割り当てられた、*xtramemsz* サイズのユーザー・メモリーのポインタを戻します。

コメント

注意: `OCIInitialize()` および `OCIEnvInit()` コールは使わずに、[OCIEnvCreate\(\)](#) を使ってください。`OCIInitialize()` および `OCIEnvInit()` コールは、下位互換性を保つためにサポートされます。

このコールは、OCI 環境ハンドルの割当ておよび初期化を行います。既に初期化済みのハンドルには何も行いません。`OCI_ERROR` または `OCI_SUCCESS_WITH_INFO` が戻った場合は、この環境ハンドルを使用し Oracle 固有のエラーおよび診断を取得できます。

これはローカルに処理され、サーバー・ラウンドトリップはありません。

環境ハンドルは、`OCIHandleFree()` を使用して解放できます。

`xtramemsz` パラメータおよびユーザー・メモリー割当ての詳細は、2-13 ページの「[ユーザー・メモリーの割当て](#)」を参照してください。

関連関数

[OCIHandleAlloc\(\)](#)、[OCIHandleFree\(\)](#)、[OCIEnvCreate\(\)](#)、[OCITerminate\(\)](#)

OCIInitialize()

用途

OCI プロセス環境を初期化します。

構文

```
sword OCIInitialize ( ub4          mode,
                    CONST dvoid *ctxp,
                    CONST dvoid *(*malocfp)
                    /* dvoid *ctxp,
                     size_t size _*/),
                    CONST dvoid *(*ralocfp)
                    /* _ dvoid *ctxp,
                     dvoid *memptr,
                     size_t newsize _*/),
                    CONST void (*mfreefp)
                    /* _ dvoid *ctxp,
                     dvoid *memptr _/));
```

パラメータ

mode (IN)

モードの初期化を指定します。次のモードが有効です。

- OCI_DEFAULT - デフォルト・モード。
- OCI_THREADED - スレッド環境。このモードでは、ユーザーに隠されている内部データ構造体が複数スレッドで同時にアクセスできないように保護されます。
- OCI_OBJECT - オブジェクト機能を使用します。
- OCI_SHARED - 共有データ構造を利用します。
- OCI_EVENTS - パブリッシュ - サブスクライブ通知を利用します。

ctxp (IN)

メモリー・コールバック・ルーチン用のユーザー定義コンテキストです。

malocfp (IN)

ユーザー定義のメモリー割当て関数です。*mode* が OCI_THREADED の場合、このメモリー割当てルーチンは、スレッド・セーフでなければなりません。

ctxp (IN/OUT)

ユーザー定義のメモリー割当て関数のためのコンテキスト・ポインタです。

size (IN)

ユーザー定義のメモリー割当て関数によって割り当てられるメモリーのサイズです。

ralocfp (IN)

ユーザー定義のメモリー再割当て関数です。 *mode* が OCI_THREADED の場合、このメモリー割当てルーチンは、スレッド・セーフでなければなりません。

ctxp (IN/OUT)

ユーザー定義のメモリー再割当て関数のためのコンテキスト・ポインタです。

memptr (IN/OUT)

メモリー・ブロックのポインタです。

newsize (IN)

新しく割り当てられるメモリーのサイズです。

mfrefp (IN)

ユーザー定義のメモリー解放関数です。 *mode* が OCI_THREADED の場合、このメモリー解放ルーチンは、スレッド・セーフでなければなりません。

ctxp (IN/OUT)

ユーザー定義のメモリー解放関数のためのコンテキスト・ポインタです。

memptr (IN/OUT)

解放されるメモリーのポインタです。

コメント

注意: *OCIInitialize()* および *OCIEnvInit()* コールは使わずに、*OCIEnvCreate()* を使ってください。 *OCIInitialize()* および *OCIEnvInit()* コールは、下位互換性を保つためにサポートされます。

このコールは、OCI プロセス環境を初期化します。 *OCIInitialize()* は、他の OCI をコールする前にコールする必要があります。

この関数を使用すると、アプリケーションでコールバックを使用して固有のメモリー管理関数を定義できます。アプリケーションでこのようなメモリー管理関数（つまりメモリー割当て、メモリー再割当て、メモリー解放）が既に定義されている場合は、この関数のコールバック・パラメータを使用して登録する必要があります。

これらのメモリー・コールバックはオプションです。アプリケーションで、この関数のメモリー・コールバックに NULL 値を渡すと、デフォルトのプロセス・メモリー割当て方法が使用されます。

共有データ構造モード

SQL 文が処理されると、特定の基礎となるデータが文に関連付けられます。このデータには、問合せの情報の定義データおよび記述データとともに、文のテキストおよびバインド・

データの情報が格納されています。文を何度実行しても、別のユーザーが実行しても、このデータは変わりません。

OCIアプリケーションがOCI_SHAREDモードで初期化された場合は、複数の文ハンドル間で共通の文データが共有されるため、アプリケーションのメモリーを節約できます。このメモリーの節約は、複数の文ハンドルを作成するアプリケーションに特に有効です。このようなアプリケーションでは、同じSQL文が、同一または複数の接続で、複数のユーザーのセッション上で実行されるためです。詳細は、2-19ページの「[共有データ・モード](#)」を参照してください。

関連項目 : マルチスレッド・アプリケーションの記述にOCIを使う方法の詳細は、8-13ページの「[スレッド・セーフティ](#)」を参照してください。

オブジェクトを使ったOCIプログラミングの詳細は、[第10章の「OCIオブジェクト・リレーショナル・プログラミング」](#)を参照してください。

例

次の文は、ユーザー定義メモリー関数がない場合に、スレッド・モードおよびオブジェクト・モードでOCIInitialize()をコールする方法の例です。

```
OCIInitialize((ub4) OCI_THREADED | OCI_OBJECT, (dvoid *)0,  
             (dvoid * (*)()) 0, (dvoid * (*)()) 0, (void (*)()) 0 );
```

関連関数

[OCIHandleAlloc\(\)](#)、[OCIHandleFree\(\)](#)、[OCIEnvCreate\(\)](#)、[OCIEnvInit\(\)](#)、[OCITerminate\(\)](#)

OCILogoff()

用途

この関数は、[OCILogon\(\)](#) で作成された接続およびセッションを終了するために使います。

構文

```
sword OCILogoff ( OCISvcCtx      *svchp  
                  OCIError      *errhp );
```

パラメータ

svchp (IN)

[OCILogon\(\)](#) のコールに使われたサービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために [OCIErrorGet\(\)](#) に渡せるエラー・ハンドルです。

コメント

このコールは、[OCILogon\(\)](#) で作成されたセッションおよび接続を終了するために使います。サーバーおよびユーザー・セッション、サービス・コンテキスト・ハンドルを暗黙的に解放します。

注意: アプリケーションでのログインおよびログアウトの詳細は、2-22 ページの「[アプリケーションの初期化および接続、セッション作成](#)」を参照してください。

関連関数

[OCILogon\(\)](#)

OCILogon()

用途

この関数は、ログイン・セッションを作成するときに使用します。

構文

```
sword OCILogon ( OCIEnv          *envhp,  
                 OCIError       *errhp,  
                 OCISvcCtx       **svchp,  
                 CONST text      *username,  
                 ub4             uname_len,  
                 CONST text      *password,  
                 ub4             passwd_len,  
                 CONST text      *dbname,  
                 ub4             dbname_len );
```

パラメータ

envhp (IN)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

svchp (IN/OUT)

サービス・コンテキスト・ポインタです。

username (IN)

ユーザー名です。

uname_len (IN)

username の長さです。

password (IN)

ユーザーのパスワードです。

passwd_len (IN)

password の長さです。

dbname (IN)

接続先のデータベース名です。

dbname_len (IN)

dbname の長さです。

コメント

この関数は、アプリケーションのためのログイン・セッションを作成するときに使用しません。

注意: TP モニタ・アプリケーションなどのより複雑なセッションを必要とするユーザーは、2-22 ページの「[アプリケーションの初期化および接続、セッション作成](#)」を参照してください。

このコールは、渡されたサービス・コンテキスト・ハンドルの割当てを行います。このコールは、セッションに関連付けられるサーバーとユーザー・セッション・ハンドルの暗黙の割当ても実行します。これらのハンドルは、サービス・コンテキスト・ハンドルに対して [OCIAttrGet\(\)](#) をコールすることにより取り出すことができます。

関連関数

[文関数](#)

OCI_{ServerAttach}()

用途

OCI 操作の対象となるデータ・ソースへのアクセス・パスを作成します。

構文

```
sword OCIServerAttach ( OCIServer      *srvhp,  
                      OCIError       *errhp,  
                      CONST text     *dblink,  
                      sb4            dblink_len,  
                      ub4            mode );
```

パラメータ

srvhp (IN/OUT)

このコールにより初期化される未初期化サーバー・ハンドルです。初期化済みのサーバー・ハンドルを渡すとエラーが発生します。

errhp (IN/OUT)

エラー時の診断情報のために *OCI_{ErrorGet}()* に渡せるエラー・ハンドルです。

dblink (IN)

使用するデータベース（サーバー）を指定します。このパラメータは、接続文字列またはサービス・ポイントを指定する文字列を指示します。接続文字列が NULL の場合、このコールは、デフォルト・ホストに接続します。dblink の長さは、dblink_len で指定します。dblink ポインタは、戻り時にコール側によって解放されます。

dblink_len (IN)

dblink により指される文字列の長さです。接続文字列または別名を有効にするには、dblink_len は 0（ゼロ）以外でなければなりません。

mode (IN)

異なるモードの操作を指定します。リリース 8.0 では、OCI_DEFAULT が渡されます。このモードでは、このサーバー・コンテキストでサーバーに対して行われたコールは、ブロック・モードで実行されます。

コメント

このコールは、OCI アプリケーションと特定のサーバー間に関連性を作成するときに使用されます。

このコールは、サーバー・コンテキスト・ハンドルを初期化しますが、ハンドルは *OCI_{HandleAlloc}()* のコールを使用して事前に割り当てられていなければなりません。このコールによって初期化されたサーバー・コンテキスト・ハンドルは、*OCI_{AttrSet}()* のコール

を介してサービス・コンテキストに関連付けることができます。関連性が作成された後、そのサーバーに対して OCI 操作を実行できます。

複数のサーバーに対してアプリケーションを実行している場合、複数のサーバー・コンテキスト・ハンドルを維持できます。OCI 操作は、サービス・コンテキストに現在関連付けられているサーバー・コンテキストに対して実行されます。

OCI`ServerAttach()` が正常に完了すると、Oracle シャドウ・プロセスが開始します。Oracle シャドウ・プロセスをクリーン・アップするには、OCI`SessionEnd()` および OCI`ServerDetach()` をコールする必要があります。コールしないと、シャドウ・プロセスが蓄積され、Unix システムのプロセス数が足りなくなります。データベースが再起動したときにプロセス数が不足している場合は、データベースが起動しないことがあります。

例

次のコードは `OCIServerAttach()` の使用方法の例です。この例では、サーバー・ハンドルを割り当て、接続コールを行い、サービス・コンテキスト・ハンドルを割り当て、最後にそれにサーバー・コンテキストを設定します。

```
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4)
    OCI_HTYPE_SERVER, 0, (dvoid **) &tmp);
OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4)
    OCI_HTYPE_SVCCIX, 0, (dvoid **) &tmp);
/* set attribute server context in the service context */
OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCIX, (dvoid *) srvhp,
    (ub4) 0, (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);
```

関連関数

[OCI`ServerDetach\(\)`](#)

OCI`ServerDetach()`

用途

OCI 操作の対象となるデータ・ソースへのアクセスを削除します。

構文

```
sword OCIServerDetach ( OCIServer   *srvhp,  
                        OCIError    *errhp,  
                        ub4          mode );
```

パラメータ

srvhp (IN)

未初期化状態にリセットされる初期化済みサーバー・コンテキストへのハンドルです。ハンドルの割当ては解除されません。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

mode (IN)

異なるモードの操作を指定します。有効なモードはデフォルト・モード用の OCI_DEFAULT だけです。

コメント

このコールは、*OCI`ServerAttach()`* のコールによって確立された OCI 操作の対象となるデータ・ソースへのアクセスを削除します。

関連関数

[OCI`ServerAttach\(\)`](#)

OCISessionBegin()

用途

ユーザー・セッションを作成し、指定のサーバーに対してユーザー・セッションを開始します。

構文

```
sword OCISessionBegin ( OCISvcCtx      *svchp,  
                        OCIError       *errhp,  
                        OCISession     *usrhp,  
                        ub4             credt,  
                        ub4             mode );
```

パラメータ

svchp (IN)

サービス・コンテキストへのハンドルです。svchp には、有効なサーバー・ハンドルを設定する必要があります。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

usrhp (IN/OUT)

このコールによって初期化されるユーザー・セッション・コンテキストへのハンドルです。

credt (IN)

ユーザー・セッションの作成に使用する資格証明のタイプを指定します。credt の有効な値は次のとおりです。

- **OCI_CRED_RDBMS** - 資格証明としてデータベースのユーザー名とパスワードを使用して認証します。このコールの前に、ユーザー・セッション・コンテキストに **OCI_ATTR_USERNAME** と **OCI_ATTR_PASSWORD** 属性が設定されていなければなりません。
- **OCI_CRED_EXT** - 外部資格証明を使用して認証します。ユーザー名とパスワードは用意されません。

mode (IN)

異なるモードの操作を指定します。次のモードが有効です。

- **OCI_DEFAULT** - このモードでは、戻されるユーザー・セッション・コンテキストは、*svchp* に指定された同じサーバー・コンテキストとともにだけ設定されます。
- **OCI_MIGRATE** - このモードでは、新しいユーザー・セッション・コンテキストは、別のサーバー・ハンドルとともにサービス・ハンドルに設定されます。このモードは、

ユーザー・セッション・コンテキストを確立します。移行可能なセッションを作成するには、サービス・ハンドルが移行不可能なセッションとともに既に設定されていなければなりません。移行可能なセッションの親は、移行不可能なセッションでなければなりません。

- OCI`_SYSDBA` - このモードでは、ユーザーは、SYSDBA アクセス用に認証されます。
- OCI`_SYSOPER` - このモードでは、ユーザーは、SYSOPER アクセス用に認証されます。
- OCI`_PRELIM_AUTH` - このモードは、ある種の管理タスクを認証するために OCI`_SYSDBA` または OCI`_SYSOPER` とともにだけ使用できます。

コメント

`OCISessionBegin()` コールは、サービス・コンテキスト・ハンドルに設定されたサーバーに対して、ユーザーを認証するときに使用します。

注意: セッションを開始するときは、戻されたエラーをすべてチェックしてください。たとえば、アカウントのパスワードの期限が切れた場合は、ORA-28001 エラーが戻されます。

Oracle8i では、サーバー・ハンドルに要求を行う前にそれに対して `OCISessionBegin()` をコールする必要があります。`OCISessionBegin()` は、サーバー・ハンドルによってサービス・コンテキスト内に指定されたユーザーによる Oracle サーバーへのアクセスだけを認証します。つまり、`OCIServerAttach()` をコールしてサーバー・ハンドルを初期化した後、指定のサーバーに対してユーザーを認証するために `OCISessionBegin()` をコールする必要があります。

指定のサーバー・ハンドルに対して、最初に `OCISessionBegin()` をコールしたとき、ユーザー・セッションは、移動可能モード (OCI`_MIGRATE`) では作成されません。

サーバー・ハンドルに対して `OCISessionBegin()` をコールした後、アプリケーションでは `OCISessionBegin()` を再度コールし、別のユーザー・セッション・ハンドルを、別の (または同じ) 資格証明と別の (または同じ) 操作モードを使用して初期化できます。アプリケーションでユーザーを OCI`_MIGRATE` モードで認証する場合、サービス・ハンドルは、移動可能ではないユーザー・ハンドルに既に関連付けられている必要があります。このユーザー・ハンドルのユーザー ID は、移動可能なユーザー・セッションの所有者 ID になります。移動可能なすべてのセッションは、移動可能ではない親セッションを持つ必要があります。

OCI`_MIGRATE` を指定しない場合、ユーザー・セッション・コンテキストは、`svchp` に設定されたサーバー・ハンドルと同じサーバー・ハンドルでしか使用できません。OCI`_MIGRATE` モードを指定した場合、ユーザー認証は、異なるサーバー・ハンドルで設定されます。しかし、ユーザー・セッション・コンテキストは、同じデータベース・インスタンスを要求するサーバー・ハンドルでだけ使用できます。セキュリティ・チェックは、セッションの切替え中に行われます。プロセスまたは回路は、セッションの所有者 ID が、現在同じプロセスまたは回路に接続された移動可能ではないセッションのユーザー ID と一致する場合だけ (セッションの作成者でない限り) 移動可能なセッションに切り替えることができます。

OCI`_SYSDBA` および OCI`_SYSOPER`、OCI`_PRELIM_AUTH` は、1 次ユーザー・セッション・コンテキストでだけ使用できます。

`OCISessionBegin()` のコール用の資格証明を用意するには、2通りの方法のうち1つがサポートされています。最初の方法は、`OCISessionBegin()` に渡されるユーザー・セッション・ハンドル内にデータベース認証用の有効なユーザー名とパスワードのペアを用意することです。この方法では、`OCIAttrSet()` を使用して、ユーザー・セッション・ハンドルに対して `OCI_ATTR_USERNAME` および `OCI_ATTR_PASSWORD` 属性を設定します。すると、`OCI_CRED_RDBMS` を指定して `OCISessionBegin()` がコールされます。

注意: `OCISessionEnd()` を使ってユーザー・セッション・ハンドルを終了しても、ユーザー名とパスワード属性は残留します。その後の `OCISessionBegin()` のコールにはこれらをまた使うことができます。再使用しない場合は、次の `OCISessionBegin()` コールの前に新しい値を設定し直す必要があります。

もう1つの資格証明の方法は、外部資格証明です。この方法では、`OCISessionBegin()` をコールする前に、ユーザー・セッション・ハンドルについて属性を設定する必要はありません。資格証明型は `OCI_CRED_EXT` です。これは、Oracle7 の 'connect /' 構文と等価です。既に `OCI_ATTR_USERNAME` および `OCI_ATTR_PASSWORD` に値が設定してある場合、`OCI_CRED_EXT` を使用すると、これらの値は無視されます。

資格証明を設定するもう1つの方法では、`OCI_MIGSESSION` 属性とともに、既に認証されているユーザーのセッション ID を使います。この ID は、`OCIAttrGet()` コールを使って認証されたユーザーのセッション・ハンドルから抽出できます。

例

次のコードは `OCISessionBegin()` の使用方法の例です。この例では、ユーザー・セッション・ハンドルを割り当て、ユーザー名とパスワードを設定し、`OCISessionBegin()` をコールして、最後にサービス・コンテキスト内にユーザー・セッションを設定します。

```
/* allocate a user session handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4)
    OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0);
OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION, (dvoid *)"jessica",
    (ub4)strlen("jessica"), OCI_ATTR_USERNAME, errhp);
OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION, (dvoid *)"doogie",
    (ub4)strlen("doogie"), 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);
```

関連関数

[OCISessionEnd\(\)](#)

OCISessionEnd()

用途

OCISessionBegin() によって作成されたユーザー・セッション・コンテキストを終了します。

構文

```
sword OCISessionEnd ( OCISvcCtx      *svchp,  
                     OCIError      *errhp,  
                     OCISession    *usrhp,  
                     ub4           mode );
```

パラメータ

svchp (IN/OUT)

サービス・コンテキスト・ハンドルです。*svchp* には、有効なサーバー・ハンドルおよびユーザー・セッション・ハンドルを関連付ける必要があります。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

usrhp (IN)

このユーザーの認証を解除します。このパラメータが NULL として渡された場合は、サービス・コンテキスト・ハンドル内のユーザーの認証が解除されます。

mode (IN)

有効なモードは OCI_DEFAULT だけです。

コメント

サービス・コンテキストに関連付けられているユーザー・セキュリティ・コンテキストがこのコールによって無効化されます。ユーザー・セッション・コンテキスト用の記憶領域は解放されません。サービス・コンテキストによって指定されたトランザクションが暗黙のうちにコミットされます。トランザクション・ハンドルが明示的に割り当てられていて、使用されていない場合は、解放されます。このユーザーのためにサーバー上に割り当てられていたリソースが解放されます。ユーザー・セッション・ハンドルは、*OCISessionBegin()* の次のコールで再使用できます。

関連関数

[OCISessionBegin\(\)](#)

OCITerminate()

用途

共有メモリー・サブシステムからプロセスを連結解除します。

構文

```
sword OCITerminate ( ub4 mode );
```

パラメータ

mode (IN)

コール固有モードです。有効な値は次のとおりです。

- OCI_DEFAULT - デフォルトのコールを実行します。

コメント

OCITerminate() は、各プロセス内で一度だけコールしてください。*OCIInitialize()* コールと対でコールします。このコールは、共有メモリー・サブシステムからプロセスを連結解除します。他のプロセスが連結していない場合は、そのプロセスを終了します。また、プロセス・クリーンアップ操作も行います。

関連関数

[OCIInitialize\(\)](#)

LOB 関数

この項では、LOB 関数について説明します。

表 15-6 OCI クイック・リファレンス

関数	用途	ページ
OCIDurationBegin()	テンポラリ LOB のためのユーザー期間を開始します。	15-110 ページ
OCIDurationEnd()	テンポラリ LOB のためのユーザー期間を終了します。	15-111 ページ
OCILobAppend()	1 つの REF を別の REF の後に追加します。	15-112 ページ
OCILobAssign()	1 つの LOB ロケータを別の LOB ロケータに割り当てます。	15-113 ページ
OCILobCharSetForm()	LOB ロケータからキャラクタ・セットを取得します。	15-115 ページ
OCILobCharSetId()	LOB ロケータからキャラクタ・セット ID を取得します。	15-116 ページ
OCILobClose()	オープン済みの LOB をクローズします。	15-117 ページ
OCILobCopy()	LOB の一部または全部を別の LOB にコピーします。	15-118 ページ
OCILobCreateTemporary()	テンポラリ LOB を作成します。	15-120 ページ
OCILobDisableBuffering()	LOB バッファリングをオフにします。	15-122 ページ
OCILobEnableBuffering()	LOB バッファリングをオンにします。	15-123 ページ
OCILobErase()	LOB の一部を消去します。	15-124 ページ
OCILobFileClose()	オープン済みの FILE をクローズします。	15-125 ページ
OCILobFileCloseAll()	オープン済みのすべての FILE をクローズします。	15-126 ページ
OCILobFileExists()	サーバー上のファイルの存在をチェックします。	15-127 ページ
OCILobFileGetName()	LOB ロケータからディレクトリ別名とファイル名を取得します。	15-128 ページ
OCILobFileIsOpen()	サーバー上のファイルがこのロケータを介してオープンされたかどうかをチェックします。	15-130 ページ
OCILobFileOpen()	FILE をオープンします。	15-131 ページ
OCILobFileSetName()	LOB ロケータにディレクトリ別名とファイル名を設定します。	15-132 ページ
OCILobFlushBuffer()	LOB バッファをフラッシュします。	15-134 ページ
OCILobFreeTemporary()	テンポラリ LOB を解放します。	15-136 ページ
OCILobGetChunkSize()	LOB のチャンク・サイズを取得します。	15-137 ページ
OCILobGetLength()	LOB の長さを取得します。	15-139 ページ
OCILobIsEqual()	2 つの LOB ロケータが等しいか比較します。	15-140 ページ

表 15-6 OCI クイック・リファレンス (続き)

関数	用途	ページ
OCILobIsOpen()	LOB がオープンされているかどうかをチェックします。	15-141 ページ
OCILobIsTemporary()	特定の LOB がテンポラリ LOB かどうかを特定します。	15-143 ページ
OCILobLoadFromFile()	FILE から LOB をロードします。	15-144 ページ
OCILobLocatorAssign()	1 つの LOB ロケータを別の LOB ロケータに割り当てます。	15-146 ページ
OCILobLocatorIsInit()	LOB ロケータが初期化されているかどうかをチェックします。	15-148 ページ
OCILobOpen()	LOB をオープンします。	15-149 ページ
OCILobRead()	LOB の一部を読み込みます。	15-151 ページ
OCILobTrim()	LOB を切り捨てます。	15-155 ページ
OCILobWrite()	LOB に書き込みます。	15-156 ページ
OCILobWriteAppend()	LOB の末尾からデータを書き込みます。	15-160 ページ

OCI LOB コールのパラメータについて、次の点に注意してください。

- 固定幅のクライアント側キャラクタ・セットの場合、オフセットおよび量パラメータは、CLOB と NCLOB では常に文字数、BLOB と BFILE では常にバイト数です。
- 可変幅のクライアント側キャラクタ・セットの場合、一般に次のルールが適用されません。
 - 量 (*amtp*) パラメータ - 量パラメータがサーバー側 LOB を参照する場合は、量は文字数です。量パラメータがクライアント側バッファを参照する場合は、量はバイト数です。詳細は、[OCILobGetLength\(\)](#)、[OCILobRead\(\)](#)、[OCILobWrite\(\)](#) などの各 LOB コールを参照してください。
 - オフセット (*offset*) パラメータ - クライアント側キャラクタ・セットが可変幅かどうかに関わらず、オフセット・パラメータは、CLOB と NCLOB では常に文字数、BLOB と BFILE では常にバイト数です。
- LOB 操作では、多くの場合、クライアント側キャラクタ・セットかどうかに関わらず、量パラメータは CLOB と NCLOB では文字数です。これらの LOB 操作には、[OCILobCopy\(\)](#)、[OCILobErase\(\)](#)、[OCILobGetLength\(\)](#)、[OCILobLoadFromFile\(\)](#)、[OCILobTrim\(\)](#) などがあります。これらのすべての操作では、サーバー上の LOB データの量を参照します。

OCIDurationBegin()

用途

ユーザー期間を開始します。

構文

```
sword OCIDurationBegin ( OCIEnv          *env,  
                        OCIError       *err,  
                        CONST OCISvcCtx *svc,  
                        OCIDuration    parent,  
                        OCIDuration    *duration );
```

パラメータ

env (IN/OUT)

NULL ポインタとして渡します。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。診断情報は、*OCIErrorGet()* をコールして取得できます。

svc (IN)

OCI サービス・コンテキスト・ハンドルです。非 NULL でなければなりません。

parent (IN)

親期間の期間番号です。

duration (OUT)

新しく作成されたユーザー期間固有の識別子です。

コメント

この関数によってユーザー期間が開始されます。リリース 8.1 では、テンポラリ LOB の作成時にユーザー期間を使うことができます。ユーザーは複数のアクティブなユーザー期間を同時に利用することができます。ユーザー期間をネストする必要はありません。*dur* パラメータは、このコールによって作成された継続時間を識別するための一意の番号を戻すのに使われます。ユーザー期間の詳細は、7-17 ページの「[テンポラリ LOB の継続時間](#)」を参照してください。

関連関数

[OCIDurationEnd\(\)](#)

OCIDurationEnd()

用途

ユーザー期間を終了します。

構文

```
sword OCIDurationEnd ( OCIEnv          *env,  
                      OCIError       *err,  
                      CONST OCISvcCtx *svc,  
                      OCIDuration    duration );
```

パラメータ

env (IN/OUT)

NULL ポインタとして渡します。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCL_ERROR が戻されます。診断情報は、*OCIErrorGet()* をコールして取得できます。

svc (IN)

OCI サービス・コンテキスト・ハンドルです。非 NULL でなければなりません。

duration (IN)

ユーザー期間を識別するための番号です。

コメント

この関数によってユーザー期間が終了します。ユーザー期間に割り当てられていたテンポラリー LOB が解放されます。

ユーザー期間の詳細は、7-17 ページの「[テンポラリー LOB の継続時間](#)」を参照してください。

関連関数

[OCIDurationBegin\(\)](#)

OCILobAppend()

用途

別の LOB の最後に LOB 値を指定どおりに追加します。

構文

```
sword OCILobAppend ( OCISvcCtx      *svchp,  
                    OCIError      *errhp,  
                    OCILobLocator *dst_locp,  
                    OCILobLocator *src_locp );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

dst_locp (IN/OUT)

一意に宛先 LOB を参照する内部 LOB ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

src_locp (IN)

一意にソース LOB を参照する内部 LOB ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

コメント

別の LOB の最後に LOB 値を指定どおりに追加します。データは、ソースから宛先の末尾にコピーされます。コピー元 LOB およびコピー先 LOB は、既に存在している必要があります。宛先 LOB は、新たに書き込まれるデータに合わせて拡張されます。許される最大長 (4 ギガバイト) を超える宛先 LOB の拡張と NULL LOB のコピーはエラーになります。

ソースおよび宛先の LOB ロケータの型は同じでなければなりません (たとえば、両方が BLOB または両方が CLOB)。LOB パッファリングは、ロケータの両方の型に対して使用不可でなければなりません。この関数では、ソースまたは宛先として FILE ロケータを受け入れません。

関連関数

[OCILobTrim\(\)](#)、[OCILobWrite\(\)](#)、[OCILobCopy\(\)](#)、[OCIErrorGet\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobAssign()

用途

LOB/FILE ロケータを別のロケータに割り当てます。

構文

```
sword OCILobAssign ( OCIEnv          *envhp,  
                    OCIError       *errhp,  
                    CONST OCILobLocator *src_locp,  
                    OCILobLocator   **dst_locpp );
```

パラメータ

envhp (IN/OUT)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

src_locp (IN)

コピー元となる LOB/FILE ロケータです。

dst_locpp (IN/OUT)

コピー先となる LOB/FILE ロケータです。コール側で、*OCIDescriptorAlloc()* をコールしてコピー先のロケータに領域を割り当てする必要があります。

コメント

ソース・ロケータを宛先ロケータに割り当てます。割り当て後は、両方のロケータは同じ LOB 値を参照します。内部 LOB の場合、宛先ロケータが表に格納されているときだけ、ソース・ロケータの LOB 値が宛先ロケータの LOB 値にコピーされます。したがって、宛先ロケータが格納されているオブジェクトのフラッシュをパブリッシュすると、LOB 値がコピーされます。

OCILobAssign() はテンポラリ LOB には使用できません。OCI_INVALID_HANDLE エラーが生成されます。テンポラリ LOB には、*OCILobLocatorAssign()* を使います。

FILE では、ファイルを参照するロケータだけが表にコピーされます。OS ファイルそのものはコピーされません。

FILE ロケータを LOB ロケータに割り当てたり、LOB ロケータを FILE ロケータに割り当てたりするとエラーになります。

ソース・ロケータがバッファリング可能な内部 LOB 用で LOB バッファリング・サブシステムを介した LOB データの変更に使用されており、そのバッファが書き込み後フラッシュされていない場合には、ソース・ロケータが宛先ロケータに割り当てられない場合があります。

これは、LOB ごとに1つのロケータしかLOBバッファリング・サブシステムを介してLOBデータを変更できないためです。

入力宛先ロケータの値は、*OCIDescriptorAlloc()* コールによってあらかじめ割り当てられている必要があります。たとえば、次のように宣言します。

```
OCILobLocator    *source_loc = (OCILobLocator *) 0;
OCILobLocator    *dest_loc = (OCILobLocator *) 0;
```

アプリケーションでは、*source_loc* ロケータを次のように割り当てます。

```
if (OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &source_loc,
    (ub4) OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0))
    handle_error;
```

次に初期化するために、表のLOBを*source_loc*に選択します。アプリケーションでは、*OCILobAssign()* コールをバブリッシュして*source_loc*の値を*dest_loc*に割り当てる前に、宛先ロケータ*dest_loc*を割り当てる必要があります。たとえば、次のとおりです。

```
if (OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
    (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0))
    handle_error;
if (OCILobAssign(envhp, errhp, source_loc, &dest_loc))
    handle_error;
```

関連関数

[OCIErrorGet\(\)](#)、[OCILobIsEqual\(\)](#)、[OCILobLocatorAssign\(\)](#)、[OCILobLocatorIsInit\(\)](#)、[OCILobEnableBuffering\(\)](#)

OCILobCharSetForm()

用途

LOB ロケータのキャラクタ・セット・フォームを取得します。

構文

```
sword OCILobCharSetForm ( OCIEnv           *envhp,  
                          OCIError        *errhp,  
                          CONST OCILobLocator *locp,  
                          ub1              *csfrm );
```

パラメータ

envhp (IN/OUT)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN)

キャラクタ・セット・フォームを取得するための LOB ロケータです。

csfrm (OUT)

入力 LOB ロケータのキャラクタ・セット・フォームです。入力ロケータが BLOB または BFILE 用の場合、バイナリ LOB/FILE にはキャラクタ・セットという概念がないため *csfrm* には 0 (ゼロ) が設定されます。コール側は、*csfrm ub1* 用の領域を割り当てる必要があります。

コメント

入力 LOB ロケータのキャラクタ・セット・フォームを *csfrm* 出力パラメータに戻します。この関数は、文字 LOB 型 (つまり CLOB および NCLOB) にだけ有効です。

関連関数

[OCIErrorGet\(\)](#)、[OCILobCharSetId\(\)](#)、[OCILobLocatorIsInit\(\)](#)

OCILobCharSetId()

用途

LOB ロケータのキャラクタ・セット ID を取得します。

構文

```
sword OCILobCharSetId ( OCIEnv           *envhp,  
                        OCIError        *errhp,  
                        CONST OCILobLocator *locp,  
                        ub2              *csid );
```

パラメータ

envhp (IN/OUT)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN)

キャラクタ・セット ID を取得するための LOB ロケータです。

csid (OUT)

入力 LOB ロケータのキャラクタ・セット ID です。入力ロケータが BLOB または BFILE 用の場合、バイナリ LOB/FILE にはキャラクタ・セットという概念がないため *csid* には 0 (ゼロ) が設定されます。コール側は、*csidub2* 用の領域を割り当てる必要があります。

コメント

入力 LOB ロケータのキャラクタ・セット ID を *csid* 出力パラメータに戻します。

この関数は、文字 LOB 型 (つまり CLOB および NCLOB) にだけ有効です。

関連関数

[OCIErrorGet\(\)](#)、[OCILobCharSetForm\(\)](#)、[OCILobLocatorIsInit\(\)](#)

OCILobClose()

用途

オープンしている LOB または FILE をクローズします。

構文

```
sword OCILobClose ( OCISvcCtx      *svchp,  
                   OCIError       *errhp,  
                   OCILobLocator  *locp );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

クローズする LOB です。ロケータは、内部 LOB または外部 LOB を参照できます。

コメント

オープンしている内部または外部 LOB をクローズします。BFILE が存在していてオープンしていないときは、エラーは戻されません。内部 LOB がオープンしていない場合は、エラーが戻されます。

LOB をクローズするときは、内部 LOB および外部 LOB からサーバーへのラウンドトリップを行なう必要があります。内部 LOB の場合は、LOB をクローズすると、クローズ・コールに依存する他のコードが実行されます。外部 LOB (BFILE) の場合は、LOB をクローズすると、サーバー側のオペレーティング・システム・ファイルが実際にクローズされます。

関連項目 : 詳細は 7-10 ページの「[LOB のオープンおよびクローズのための関数](#)」を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobOpen\(\)](#)、[OCILobIsOpen\(\)](#)

OCILobCopy()

用途

LOB 値の全体または一部を別の LOB 値にコピーします。

構文

```
sword OCILobCopy ( OCISvcCtx      *svchp,  
                  OCIError       *errhp,  
                  OCILobLocator  *dst_locp,  
                  OCILobLocator  *src_locp,  
                  ub4            amount,  
                  ub4            dst_offset,  
                  ub4            src_offset );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

dst_locp (IN/OUT)

一意に宛先 LOB を参照する内部 LOB ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

src_locp (IN)

一意にソース LOB を参照する内部 LOB ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

amount (IN)

コピー元の LOB からコピー先の LOB へコピーされる文字数 (CLOB/NCLOB) またはバイト数 (BLOB) です。

dst_offset (IN)

これは、宛先 LOB 用の絶対オフセットです。キャラクタ LOB では、LOB の先頭から書き込みを開始する位置までの文字数です。バイナリ LOB では、LOB の先頭から書き込みを開始する位置までのバイト数です。オフセットは 1 から始まります。

src_offset (IN)

これは、ソース LOB の絶対表示オフセットです。キャラクタ LOB では LOB の先頭からの文字数であり、バイナリ LOB ではバイト数です。オフセットは 1 から始まります。

コメント

内部 LOB 値全体または一部を別の内部 LOB 値に指定されたとおりにコピーします。データは、ソースから宛先にコピーされます。ソース (*src_locp*) と宛先 (*dst_locp*) LOB は既に存在していなければなりません。

宛先のコピー開始位置に既にデータが存在する場合は、ソース・データによって上書きされます。宛先のコピー開始位置が現データの末尾の位置を超えている場合は、宛先 LOB のデータの末尾とソースから新たに書き込まれるデータの先頭との間に 0 バイト充てん文字 (BLOB 用) または空白 (CLOB 用) が書き込まれます。新規に書き込むデータが追加先 LOB の現行の長さよりも大きい場合は、追加先 LOB は、そのデータに合わせて拡張されず。許される最大長 (4 ギガバイト) を超える宛先 LOB の拡張と NULL LOB のコピーはエラーになります。

ソースおよび宛先の LOB ロケータの型は同じでなければなりません (たとえば、両方が BLOB または両方が CLOB)。LOB バッファリングは、ロケータの両方に対して使用不可でなければなりません。

この関数では、ソースまたは宛先として FILE ロケータを受け入れません。

注意: ソース LOB の長さを判断するには、*OCILobGetLength()* をコールします。

関連関数

[OCIErrorGet\(\)](#)、[OCILobRead\(\)](#)、[OCILobAppend\(\)](#)、[OCILobCopy\(\)](#)、[OCILobWrite\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobCreateTemporary()

用途

テンポラリ LOB を作成します。

構文

```
sword OCILobCreateTemporary(OCISvcCtx          *svchp,  
                             OCIError          *errhp,  
                             OCILobLocator     *locp,  
                             ub2               csid,  
                             ub1               csfrm,  
                             ub1               lobtype,  
                             boolean           cache,  
                             OCIDuration      duration);
```

パラメータ

svchp (IN)

OCI サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

テンポラリ LOB を指すロケータです。ロケータをこの関数に渡す前に、*OCIDescriptorAlloc()* を使ってロケータを割り当てる必要があります。このロケータは、LOB を指しているかどうかに関わらず上書きされます。

csid (IN)

LOB キャラクタ・セット ID です。Oracle8i の場合は、OCI_DEFAULT を渡します。

csfrm (IN)

バッファ・データの LOB キャラクタ・セット・フォームです。Oracle8i の場合は、OCI_DEFAULT を渡します。

lobtype (IN)

作成する LOB の型です。次の値が有効です。

- OCI_TEMP_BLOB - テンポラリ BLOB の場合
- OCI_TEMP_CLOB - テンポラリ CLOB の場合
- OCI_TEMP_NCLOB - テンポラリ NCLOB の場合

cache (IN)

テンポラリ LOB をキャッシュに読み込む必要がある場合は TRUE、その必要がない場合は FALSE を渡します。NOCACHE 機能の場合は、デフォルトは FALSE です。

duration (IN)

テンポラリ LOB の期間です。次に示す値が有効です。

- OCI_DURATION_SESSION
- OCI_DURATION_CALL

コメント

この関数は、ユーザーのテンポラリ表領域にテンポラリ LOB および対応する索引を作成します。

この関数が完了すると、*locp* パラメータは、長さが 0 (ゼロ) の空のテンポラリ LOB を指します。

テンポラリ LOB の存続期間は、*duration* パラメータによって決まります。テンポラリ LOB は、この期間の最後に解放されます。アプリケーションから *OCILobFreeTemporary()* コールを使うと、テンポラリ LOB をより早く解放できます。

LOB が BLOB の場合は、*csid* および *csfrm* パラメータは無視されます。

テンポラリ LOB とその存続期間の詳細は、7-16 ページの「[テンポラリ LOB のサポート](#)」を参照してください。

関連関数

[OCILobFreeTemporary\(\)](#)、[OCILobIsTemporary\(\)](#)、[OCIDescriptorAlloc\(\)](#)、[OCIErrorGet\(\)](#)

OCILobDisableBuffering()

用途

入力ロケータの LOB バッファリングを使用禁止にします。

構文

```
sword OCILobDisableBuffering ( OCISvcCtx      *svchp,  
                               OCIError      *errhp,  
                               OCILobLocator *locp );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

一意に LOB を参照する内部 LOB ロケータです。

コメント

入力内部 LOB ロケータの LOB バッファリングを使用禁止にします。次回入力ロケータを通して LOB に対してデータの読書きを行うときは、LOB バッファリング・サブシステムは使用されません。このコールでは、バッファリング・サブシステムで行われた変更は暗黙的にフラッシュされません。ユーザーは、*OCILobFlushBuffer()* をコールして変更を明示的にフラッシュする必要があります。

この関数は、FILE ロケータを受け入れません。

関連関数

[OCILobEnableBuffering\(\)](#)、[OCIErrorGet\(\)](#)、[OCILobFlushBuffer\(\)](#)

OCILobEnableBuffering()

用途

入力ロケータの LOB バッファリングを使用可能にします。

構文

```
sword OCILobEnableBuffering ( OCISvcCtx      *svchp,  
                              OCIError      *errhp,  
                              OCILobLocator *locp );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

一意に LOB を参照する内部 LOB ロケータです。

コメント

入力内部 LOB ロケータの LOB バッファリングを使用可能にします。次回入力ロケータを通して LOB に対してデータの読書きを行うときは、LOB バッファリング・サブシステムが使用されます。

ロケータに対して LOB バッファリングを使用可能にして、そのロケータを *OCILobAppend()*、*OCILobCopy()*、*OCILobErase()*、*OCILobGetLength()*、*OCILobLoadFromFile()*、*OCILobTrim()* または *OCILobWriteAppend()* のいずれかのルーチンに渡した場合は、エラーが戻されます。

この関数は、FILE ロケータを受け入れません。

関連関数

OCILobDisableBuffering()、*OCIErrorGet()*、*OCILobWrite()*、*OCILobRead()*、*OCILobFlushBuffer()*、*OCILobWriteAppend()*

OCILobErase()

用途

内部 LOB データの一部を指定のオフセットの位置から消去します。

構文

```
sword OCILobErase ( OCISvcCtx      *svchp,  
                   OCIError      *errhp,  
                   OCILobLocator *locp,  
                   ub4            *amount,  
                   ub4            offset );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

一意に LOB を参照する内部 LOB ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

amount (IN/OUT)

消去する文字数 (CLOB/NCLOB の場合) またはバイト数 (BLOB の場合) です。IN では、値は消去する文字数またはバイト数を表します。OUT では、値は消去された実際の文字数またはバイト数です。

offset (IN)

データの消去を開始する LOB 値の、先頭からの絶対オフセット (CLOB/NCLOB の場合は文字数、BLOB の場合はバイト数) です。オフセットは 1 から始まります。

コメント

実際に消去した文字数またはバイト数が戻ります。BLOB の場合、消去するということは、0 (ゼロ) バイトの充てん文字で既存の LOB 値を上書きするということです。CLOB の場合、空白で上書きされます。

この関数は内部 LOB にだけ有効で、FILE には使用できません。

関連関数

[OCIErrorGet\(\)](#)、[OCILobRead\(\)](#)、[OCILobAppend\(\)](#)、[OCILobCopy\(\)](#)、[OCILobWrite\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobFileClose()

用途

オープンされている FILE をクローズします。

構文

```
sword OCILobFileClose ( OCISvcCtx          *svchp,  
                        OCIError          *errhp,  
                        OCILobLocator     *filep );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

filep (IN/OUT)

閉じる FILE を参照する FILE ロケータのポインタです。

コメント

オープンされている FILE をクローズします。内部 LOB に対してこの関数をコールするとエラーになります。FILE が存在していてオープンしていないときは、エラーは戻りません。

この関数は、特定の FILE ロケータに対して初めてコールしたときだけ有効です。同じ FILE ロケータを使用してこの関数を続けてコールしても何も行われません。

関連項目 : FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobClose\(\)](#)、[OCILobFileCloseAll\(\)](#)、[OCILobFileExists\(\)](#)、[OCILobFileIsOpen\(\)](#)、[OCILobFileOpen\(\)](#)、[OCILobOpen\(\)](#)、[OCILobIsOpen\(\)](#)

OCILobFileCloseAll()

用途

指定のサービス・コンテキストでオープンしているすべての FILE をクローズします。

構文

```
sword OCILobFileCloseAll ( OCISvcCtx   *svchp,  
                           OCIError    *errhp );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

指定のサービス・コンテキストでオープンしているすべての FILE をクローズします。内部 LOB に対してこの関数をコールするとエラーになります。

関連項目 : FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

関連関数

[OCILobFileClose\(\)](#)、[OCIErrorGet\(\)](#)、[OCILobFileExists\(\)](#)、[OCILobFileIsOpen\(\)](#)

OCILobFileExists()

用途

サーバーのオペレーティング・システムに FILE が存在するかどうかをテストします。

構文

```
sword OCILobFileExists ( OCISvcCtx      *svchp,  
                        OCIError       *errhp,  
                        OCILobLocator  *filep,  
                        boolean        *flag );
```

パラメータ

svchp (IN)

OCI サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

filep (IN)

ファイルを参照する FILE ロケータのポインタです。

flag (OUT)

サーバーに FILE が存在する場合は TRUE、存在しない場合は FALSE を戻します。

コメント

サーバーのファイル・システムに FILE が存在するかどうかをチェックします。内部 LOB に対してこの関数をコールするとエラーになります。

関連項目 : FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobFileClose\(\)](#)、[OCILobFileCloseAll\(\)](#)、[OCILobFileIsOpen\(\)](#)、[OCILobOpen\(\)](#)、[OCILobIsOpen\(\)](#)

OCILobFileName()

用途

FILE ロケータのディレクトリ別名およびファイル名を取得します。

構文

```
sword OCILobFileName ( OCIEnv           *envhp,  
                      OCIErrror        *errhp,  
                      CONST OCILobLocator *filep,  
                      text              *dir_alias,  
                      ub2               *d_length,  
                      text              *filename,  
                      ub2               *f_length );
```

パラメータ

envhp (IN/OUT)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrrorGet()* に渡せるエラー・ハンドルです。

filep (IN)

ディレクトリ別名およびファイル名を取得するための FILE ロケータです。

dir_alias (OUT)

ディレクトリ別名が配置されているバッファです。コール側は、ディレクトリ別名用の十分な領域を割り当てる必要があります。ディレクトリ別名の最大長は 30 バイトです。

d_length (IN/OUT)

次の用途に使用できます。

- IN: 入力 *dir_alias* 文字列の長さ
- OUT: 戻される *dir_alias* 文字列の長さ

filename (OUT)

ファイル名が配置されているバッファです。コール側は、ファイル名用の十分な領域を割り当てる必要があります。ファイル名の最大長は 255 バイトです。

f_length (IN/OUT)

次の用途に使用できます。

- IN: 入力 *filename* バッファの長さ
- OUT: 戻される *filename* 文字列の長さ

コメント

この FILE ロケータに対応付けられたディレクトリ別名とファイル名を戻します。内部 LOB に対してこの関数をコールするとエラーになります。

関連項目 : FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

関連関数

[OCILobFileSetName\(\)](#)、[OCIErrorGet\(\)](#)

OCILobFileIsOpen()

用途

FILE がオープンしているかどうかをテストします。

構文

```
sword OCILobFileIsOpen ( OCISvcCtx      *svchp,  
                        OCIError       *errhp,  
                        OCILobLocator   *filep,  
                        boolean        *flag );
```

パラメータ

svchp (IN)

OCI サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

filep (IN)

対象となる FILE ロケータのポインタです。

flag (OUT)

この特定のロケータを使って FILE が開かれた場合は TRUE、開かれなかった場合は FALSE が戻されます。

コメント

サーバー上のファイルが *filep* FILE ロケータを使用してオープンされたかどうかをチェックします。内部 LOB に対してこの関数をコールするとエラーになります。

OCILobFileOpen() または *OCILobOpen()* コマンドに入力 FILE ロケータが渡されなかった場合、ファイルはそのロケータによってオープンされていないとみなされます。ただし、別のロケータがそのファイルをオープンしていることがあります。オープンは、特定のロケータに関連付けられます。

関連項目 : FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobClose\(\)](#)、[OCILobFileCloseAll\(\)](#)、[OCILobFileExists\(\)](#)、[OCILobFileClose\(\)](#)、[OCILobFileOpen\(\)](#)、[OCILobOpen\(\)](#)、[OCILobIsOpen\(\)](#)

OCILobFileOpen()

用途

読み専用アクセス用にサーバーのファイル・システム上の FILE をオープンします。

構文

```
sword OCILobFileOpen ( OCISvcCtx          *svchp,  
                      OCIError          *errhp,  
                      OCILobLocator     *filep,  
                      ub1                mode );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

filep (IN/OUT)

オープンする FILE です。ロケータが FILE を参照していない場合はエラーが発生します。

mode (IN)

ファイルをオープンするモードです。有効なモードは OCI_FILE_READONLY だけです。

コメント

サーバーのファイル・システムの FILE をオープンします。FILE は、読み専用アクセス用にオープンできません。Oracle を介して FILE に書き込むことはできません。内部 LOB に対してこの関数をコールするとエラーになります。

この関数は、特定の FILE ロケータに対して初めてコールしたときだけ有効です。同じ FILE ロケータを使用してこの関数を続けてコールしても何も行われません。

関連項目 : FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobClose\(\)](#)、[OCILobFileCloseAll\(\)](#)、[OCILobFileExists\(\)](#)、[OCILobFileClose\(\)](#)、[OCILobFileIsOpen\(\)](#)、[OCILobOpen\(\)](#)、[OCILobIsOpen\(\)](#)

OCILobFileSetName()

用途

FILE ロケータ内にディレクトリ別名とファイル名を設定します。

構文

```
sword OCILobFileSetName ( OCIEnv          *envhp,  
                          OCIError        *errhp,  
                          OCILobLocator   **filepp,  
                          CONST text      *dir_alias,  
                          ub2             d_length,  
                          CONST text      *filename,  
                          ub2             f_length );
```

パラメータ

envhp (IN/OUT)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

filepp (IN/OUT)

ディレクトリ別名およびファイル名を設定する FILE ロケータです。

dir_alias (IN)

FILE ロケータに設定するディレクトリ別名を含むバッファです。

d_length (IN)

入力 *dir_alias* パラメータの長さです。

filename (IN)

FILE ロケータに設定するファイル名を含むバッファです。

f_length (IN)

入力 *filename* パラメータの長さです。

コメント

内部 LOB に対してこの関数をコールするとエラーになります。

関連項目 : FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

関連関数

[OCILobFileName\(\)](#)、[OCIErrorGet\(\)](#)

OCILobFlushBuffer()

用途

この LOB のすべてのバッファをサーバーにフラッシュするか、もしくは書き込みます。

構文

```
sword OCILobFlushBuffer ( OCISvcCtx      *svchp,  
                          OCIError      *errhp,  
                          OCILobLocator *locp,  
                          ub4           flag );
```

パラメータ

svchp (IN/OUT)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

LOB を一意に参照する内部ロケータです。

flag (IN)

OCI_LOB_BUFFER_FREE を設定すると、LOB のバッファ・リソースは、フラッシュの後に解放されます。次のコメントを参照してください。

コメント

入力ロケータによって参照される LOB に関連するバッファリング・サブシステムに加えられた変更を、サーバーにフラッシュします。このルーチンでは、バッファのデータをデータベースの LOB に実際に書き込みます。LOB バッファリングが入力 LOB ロケータに対してあらかじめ使用可能でなければなりません。

このフラッシュ操作のデフォルトでは、別のバッファ LOB 操作への再割当て用にバッファ・リソースは解放されません。ただし、バッファを明示的に解放する場合は、flag パラメータを OCI_LOB_BUFFER_FREE に設定できます。

クライアント・アプリケーションでフラッシュ後のバッファ値を読み込む予定で、あらかじめバッファの現行値が必要な値であることがわかっている場合、サーバーからデータを再度読み込む必要はありません。

バッファの解放による影響はユーザーにはほとんど意識されません。ただし、LOB 内の同じ範囲に次にアクセスするときに、サーバーをラウンドトリップすることになり、バッファ・リソースの取得および LOB から読み込まれるデータによる初期化のコストもかかります。このオプションは、オンボード・メモリーが少ないクライアント環境用です。

関連関数

OCILobEnableBuffering()、*OCIErrorGet()*、*OCILobWrite()*、*OCILobRead()*、*OCILobDisableBuffering()*、*OCILobWriteAppend()*

OCILobFreeTemporary()

用途

テンポラリ LOB を解放します。

構文

```
sword OCILobFreeTemporary( OCISvcCtx          *svchp,  
                           OCIError          *errhp,  
                           OCILobLocator     *locp);
```

パラメータ

svchp (IN/OUT)

OCI サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

解放する LOB を一意に参照するロケータです。

コメント

このロケータが指すテンポラリ LOB の内容を解放します。ロケータ自体は、*OCIDescriptorFree()* がコールされるまで解放されません。

locp パラメータで渡される LOB ロケータがテンポラリ LOB を指していない場合は、エラーを戻します。次のいずれかが原因である可能性があります。

- ロケータが、永続 LOB を指しています。
- ロケータが、既に解放されたテンポラリ LOB を指しています。
- ロケータが、何も指していません。

関連関数

OCILobCreateTemporary()、*OCILobIsTemporary()*、*OCIErrorGet()*

OCILobGetChunkSize()

用途

LOB のチャンク・サイズを取得します。

構文

```
sword OCILobGetChunkSize ( OCISvcCtx      *svchp,  
                           OCIError      *errhp,  
                           OCILobLocator *locp,  
                           ub4           *chunk_size );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

利用可能なチャンク・サイズを取得する内部 LOB です。

chunk_size (OUT)

内部 LOB 値を格納するために使うチャンク領域の量です。これは、ユーザーが LOB 値の読み書き時に使う量です。可能な場合は、チャンクの先頭などのチャンクの境界で書き込みを開始し、1 つのチャンクを 1 度で書き込みます。

chunk_size は、BLOB ではバイト数、CLOB と NCLOB では文字数で戻されます。可変幅のキャラクタ・セットの場合は、チャンクと同じ大きさの Unicode キャラクタ数です。

コメント

内部 LOB が格納される表を作成するときに、チャンク係数を指定できます。チャンク係数は、Oracle ブロックの倍数で、LOB 値に対するアクセスまたは修正時に、LOB データ層によって使われるチャンク・サイズに対応しています。チャンクの一部はシステム関連情報に使われ、残りのチャンクには LOB 値が格納されます。この関数では、LOB 値が格納される LOB チャンクで使われる領域の量を戻します。アプリケーションからこのチャンク・サイズの倍数を使って読み込みまたは書き込み要求をパブリッシュすると、パフォーマンスが向上します。書き込みの場合は、他にも利点があります。LOB チャンクは世代管理されているので、すべての書き込みをチャンク単位で行うと、不要または重複した世代管理が行われることはありません。同じチャンクに対して複数の書き込みコールをパブリッシュするかわりに、チャンクがいっぱいになるまで書き込みを蓄積することもできます。

関連項目 : 詳細は 7-9 ページの「[LOB の読み込み / 書き込みパフォーマンスを向上するための関数](#)」を参照してください。

OCILobGetChunkSize()

関連関数

[OCIErrorGet\(\)](#)、[OCILobRead\(\)](#)、[OCILobAppend\(\)](#)、[OCILobCopy\(\)](#)、[OCILobWrite\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobGetLength()

用途

LOB の長さを取得します。

構文

```
sword OCILobGetLength ( OCISvcCtx      *svchp,  
                        OCIError      *errhp,  
                        OCILobLocator *locp,  
                        ub4            *lenp );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN)

一意に LOB を参照する LOB ロケータです。内部 LOB では、このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。FILE の場合は、*OCILobFileSetName()*、または SELECT 文、*OCIObjectPin* を使用して設定できます。

lenp (OUT)

出力時に LOB が NULL でない場合は、LOB の長さです。文字の LOB の場合は、LOB 内の文字数です。バイナリの LOB および BFILE の場合は、LOB 内のバイト数です。

コメント

LOB の長さを取得します。LOB が NULL の場合、長さは未定義です。FILE の長さが存在する場合、その長さには EOF が含まれます。空の内部 LOB の長さは 0 (ゼロ) です。

クライアント側キャラクタ・セットが可変幅かどうかに関わりなく、出力の長さは、CLOB と NCLOB では文字数、BLOB と BFILE ではバイト数です。

注意: 以前の *OCILobErase()* または *OCILobWrite()* のコールによって LOB に書き込まれた、ゼロ・バイトまたは空白の充填文字も、長さの一部になります。

関連関数

[OCIErrorGet\(\)](#)、[OCILobFileSetName\(\)](#)、[OCILobRead\(\)](#)、[OCILobWrite\(\)](#)、[OCILobCopy\(\)](#)、[OCILobAppend\(\)](#)、[OCILobLoadFromFile\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobIsEqual()

用途

2 つの LOB/FILE ロケータの同等性を比較します。

構文

```
sword OCILobIsEqual ( OCIEnv          *envhp,  
                      CONST OCILobLocator *x,  
                      CONST OCILobLocator *y,  
                      boolean          *is_equal );
```

パラメータ

envhp (IN)

OCI 環境ハンドルです。

x (IN)

比較する LOB ロケータです。

y (IN)

比較する LOB ロケータです。

is_equal (OUT)

LOB ロケータが等価の場合は TRUE、等価でない場合は FALSE です。

コメント

指定された LOB/FILE ロケータの同等性を比較します。2 つの LOB/FILE ロケータは、どちらも同じ LOB/FILE 値を参照している場合だけ同等になります。

この関数では、2 つの NULL ロケータは、同等とはみなされません。

関連関数

[OCILobAssign\(\)](#)、[OCILobLocatorIsInit\(\)](#)

OCILobIsOpen()

用途

LOB または FILE がオープンされているかどうかをテストします。

構文

```
sword OCILobIsOpen ( OCISvcCtx      *svchp,  
                    OCIError      *errhp,  
                    OCILobLocator *locp,  
                    boolean        *flag );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー発生時に診断情報として *OCIErrorGet()* に渡されるエラー・ハンドルです。

locp (IN)

対象となる LOB ロケータのポインタです。ロケータは、内部 LOB または外部 LOB を参照できます。

flag (OUT)

LOB がオープンしている場合、または入力ロケータ内部を使って BFILE がオープンされた場合は、TRUE を戻します。それ以外は FALSE を戻します。

コメント

LOB がオープンされている場合、または入力ロケータ内部を使って BFILE がオープンされた場合は TRUE を戻します。

BFILES の場合

OCILobOpen() または *OCILobFileOpen()* に入力 BFILE ロケータが渡されなかった場合、BFILE はそのロケータによってオープンされていないと見なされます。ただし、別の BFILE からその BFILE がオープンされていることがあります。別のロケータを使って、その BFILE に対して複数のオープンが実行されることがあります。つまり、オープンは BFILE の特定のロケータに関連付けられます。

内部 LOB の場合

オープンは、ロケータではなく LOB に関連付けられます。*locator1* が LOB をオープンした場合は、*locator2* もオープン時にその LOB を参照します。

内部 LOB の場合、このコールでは、サーバー上の状態によって、LOB がオープンしているかどうかを確認するため、サーバー・ラウンドトリップが必要です。外部 LOB (BFILE) の

場合も、サーバー側のオペレーティング・システム・ファイルによって、その LOB がオープンされているかどうかを確認するため、ラウンドトリップが必要です。

関連項目：詳細は 7-10 ページの「LOB のオープンおよびクローズのための関数」を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobClose\(\)](#)、[OCILobFileCloseAll\(\)](#)、[OCILobFileExists\(\)](#)、[OCILobFileClose\(\)](#)、[OCILobFileIsOpen\(\)](#)、[OCILobFileOpen\(\)](#)、[OCILobOpen\(\)](#)

OCILobIsTemporary()

用途

ロケータがテンポラリ LOB を指すかどうかをテストします。

構文

```
sword OCILobIsTemporary(OCIEnv          *envhp,  
                        OCIError        *errhp,  
                        OCILobLocator   *locp,  
                        boolean         *is_temporary);
```

パラメータ

envhp (IN)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN)

テストするロケータです。

is_temporary (OUT)

ロケータがテンポラリ LOB を指す場合は TRUE、そうでない場合は FALSE を戻します。

コメント

この関数では、ロケータがテンポラリ LOB を指すかどうかを調べるためにロケータをテストします。テンポラリ LOB を指す場合は、*is_temporary* が TRUE に設定されます。そうでない場合は、*is_temporary* が FALSE に設定されます。

関連関数

[OCILobCreateTemporary\(\)](#)、[OCILobFreeTemporary\(\)](#)

OCILobLoadFromFile()

用途

ファイルの全体または一部を内部 LOB にロード / コピーします。

構文

```
sword OCILobLoadFromFile ( OCISvcCtx      *svchp,  
                           OCIError      *errhp,  
                           OCILobLocator *dst_locp,  
                           OCILobLocator *src_locp,  
                           ub4           amount,  
                           ub4           dst_offset,  
                           ub4           src_offset );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

dst_locp (IN/OUT)

タイプ BLOB または CLOB、NCLOB であり得る宛先内部 LOB を一意に参照するロケータです。

src_locp (IN/OUT)

ソース FILE を一意に参照するロケータです。

amount (IN)

ロードされるバイト数です。

dst_offset (IN)

これは、宛先 LOB 用の絶対オフセットです。キャラクタ LOB では、LOB の先頭から書込みを開始する位置までの文字数です。バイナリ LOB では、LOB の先頭から読取りを開始する位置までのバイト数です。オフセットは 1 から始まります。

src_offset (IN)

これは、ソース FILE にとっての絶対表示オフセットです。FILE の先頭からのバイト数です。オフセットは 1 から始まります。

コメント

FILE 値の一部またはすべてを指定どおりに内部 LOB にロード / コピーします。データは、ソース FILE から宛先の内部 LOB(BLOB/CLOB) にコピーされます。FILE データを

CLOB/NCLOB にコピーするときに、キャラクタ・セットの変換は実行されません。また、バイナリ・データが BLOB にロードされた場合は、キャラクタ・セット変換が実行されません。このため、FILE データは、あらかじめデータベースの LOB と同じキャラクタ・セットでなければなりません。これを検証するエラー・チェックは実行されません。

ソース (*src_locp*) と宛先 (*dst_locp*) LOB は既に存在していなければなりません。宛先のコピー開始位置に既にデータが存在する場合は、ソース・データによって上書きされます。宛先のコピー開始位置が現データの末尾の位置を超えている場合は、宛先 LOB の現データの末尾とソースから新たに書き込まれるデータの先頭との間に 0 バイト充填文字 (BLOB 用) または空白 (CLOB 用) が書き込まれます。新規に書き込むデータが追加先 LOB の現行の長さよりも大きい場合は、追加先 LOB は、そのデータに合わせて拡張されます。

許される最大長 (4 ギガバイト) を超える宛先 LOB の拡張と NULL FILE のコピーはエラーになります。

関連関数

[OCIErrorGet\(\)](#)、[OCILobAppend\(\)](#)、[OCILobWrite\(\)](#)、[OCILobTrim\(\)](#)、[OCILobCopy\(\)](#)、[OCILobGetLength\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobLocatorAssign()

用途

LOB/FILE ロケータを別のロケータに割り当てます。

構文

```
sword OCILobLocatorAssign ( OCISvcCtx          *svchp,  
                             OCIError          *errhp,  
                             CONST OCILobLocator *src_locp,  
                             OCILobLocator     **dst_locpp );
```

パラメータ

svchp (IN/OUT)

OCI サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

src_locp (IN)

コピー元となる LOB/FILE ロケータです。

dst_locpp (IN/OUT)

コピー先となる LOB/FILE ロケータです。コール側で、*OCIDescriptorAlloc()* をコールして *OCILobLocator* に領域を割り当ててする必要があります。

コメント

このコールは、ソース・ロケータを宛先ロケータに割り当てます。割当て後は、両方のロケータは同じ LOB データを参照します。内部 LOB の場合、宛先ロケータが表に格納されているときだけ、ソース・ロケータの LOB データが宛先ロケータの LOB データにコピーされます。このため、宛先ロケータが含まれているオブジェクトにフラッシュをパブリッシュすると、LOB データがコピーされます。FILE の場合、OS ファイルを参照するロケータだけが表にコピーされ、OS ファイルはコピーされません。

このコールは *OCILobAssign()* と似ていますが、*OCILobLocatorAssign()* は OCI 環境ハンドル・ポインタではなく、OCI サービス・ハンドル・ポインタを使います。また、*OCILobLocatorAssign()* はテンポラリー LOB に対して使うことができますが、*OCILobAssign()* は使うことができません。

注意 : *OCILobLocatorAssign()* 関数が正常に終了しなかった場合は、ターゲット・ロケータは以前の状態には復元されません。ターゲット・ロケータは、再度初期化するまで、後続の操作で使わないでください。

宛先ロケータがテンポラリー LOB に対するロケータの場合は、宛先テンポラリー LOB は、ソース LOB を割り当てる前に解放されます。ソース LOB ロケータからテンポラリー LOB を参照

すると、ソース・テンポラリ LOB がディープ・コピーされ、そのテンポラリ LOB の新しいディープ・コピーを参照するために宛先ロケータが作成されます。このディープ・コピーされないようにするには、等号を使って、2つの LOB ロケータ・ポインタが同じ LOB ロケータを参照するようにします。

関連関数

[OCIErrorGet\(\)](#)、[OCILobAssign\(\)](#)、[OCILobIsEqual\(\)](#)、[OCILobLocatorIsInit\(\)](#)

OCILobLocatorIsInit()

用途

指定された LOB/FILE ロケータが初期化されているかどうかをテストします。

構文

```
sword OCILobLocatorIsInit ( OCIEnv           *envhp,  
                           OCIError        *errhp,  
                           CONST OCILobLocator *locp,  
                           boolean         *is_initialized );
```

パラメータ

envhp (IN/OUT)

OCI 環境ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN)

テスト中の LOB/FILE ロケータです。

is_initialized (OUT)

所定の LOB/FILE ロケータが初期化されると TRUE、初期化されないと FALSE を戻します。

コメント

指定された LOB/FILE ロケータが初期化されているかどうかをテストします。

内部 LOB ロケータは、次のいずれかの方法で初期化できます。

- 非 NULL の LOB をロケータに SELECT します。
- *OCIObjectPin()* を使用して非 NULL の LOB 属性を含むオブジェクトを確保します。
- *OCIAttrSet()* (詳細は、A-25 ページの「LOB ロケータの属性」を参照) 経由でロケータを Empty 値に設定します。

FILE ロケータは、次のいずれかの方法で初期化できます。

- 非 NULL の FILE をロケータに SELECT します。
- *OCIObjectPin()* を使用して非 NULL の FILE 属性を含むオブジェクトを確保します。
- *OCILobFileSetName()* をコールします。

関連関数

[OCIErrorGet\(\)](#)、[OCILobIsEqual\(\)](#)

OCILobOpen()

用途

指定されたモードで LOB（内部または外部）をオープンします。

構文

```
sword OCILobOpen ( OCISvcCtx      *svchp,  
                  OCIError      *errhp,  
                  OCILobLocator *locp,  
                  ub1           mode );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

オープンする LOB です。ロケータは、内部 LOB または外部 LOB を参照できます。

mode (IN)

LOB/BFILE をオープンするモードです。Oracle8i では、LOB に対して有効なモードは OCI_LOB_READONLY および OCI_LOB_READWRITE です。OCI_FILE_READONLY は、Oracle8 では、*OCILobFileOpen()* の入力モードとして使います。入力ロケータが BFILE に対するロケータの場合は、OCI_FILE_READONLY は *OCILobOpen()* とともに使うことができます。

コメント

同じ LOB を 2 度オープンするとエラーが発生します。BFILE は、読み書きモードでオープンできません。LOB/BFILE が読み専用モードでオープンされている場合、LOB/BFILE に書き込みを行うと、エラーが戻されます。

LOB のオープンでは、内部 LOB と外部 LOB のどちらに対してもサーバーへのラウンドトリップが必要です。内部 LOB の場合は、オープンにより、オープン・コールに依存する他のコードが実行されます。外部 LOB (BFILE) の場合は、サーバー側のオペレーティング・システム・ファイルがオープンしているので、オープンにはラウンドトリップが必要です。

関連項目 : 詳細は 7-10 ページの「[LOB のオープンおよびクローズのための関数](#)」を参照してください。

OCILobOpen()

関連関数

OCIErrorGet()、*OCILobClose()*、*OCILobFileCloseAll()*、*OCILobFileExists()*、*OCILobFileClose()*、*OCILobFileIsOpen()*、*OCILobFileOpen()*、*OCILobIsOpen()*

OCILobRead()

用途

LOB/FILE の一部をコールの指定どおりにバッファに読み込みます。

構文

```

sword OCILobRead ( OCISvcCtx          *svchp,
                  OCIError           *errhp,
                  OCILobLocator       *locp,
                  ub4                 *amtp,
                  ub4                 offset,
                  dvoid               *bufp,
                  ub4                 buf1,
                  dvoid               *ctxp,
                  OCICallbackLobRead (cbfp)
                  ( dvoid             *ctxp,
                    CONST dvoid      *bufp,
                    ub4              len,
                    ub1              piece )
                  ub2                 csid,
                  ub1                 csfrm );

```

パラメータ

svchp (IN/OUT)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN)

一意に LOB/FILE を参照する LOB/FILE ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

amtp (IN/OUT)

入力時は、読み込まれる文字数 (CLOB または NCLOB の場合) またはバイト数 (BLOB および BFILE の場合) です。出力時は、実際に読み込まれたバイト数または文字数です。

次の場合、**amtp* は読み込まれたデータの総量です。

- データがストリーム・モードで読み込まれない場合 (1 ピースだけ読み込まれ、ポーリングまたはコールバックは行われません。)
- データがコールバックによってストリーム・モードで読み込まれる場合

データがポーリングを使ってストリーム・モードで読み込まれる場合は、**amtp* は読み込まれた最後のピースの長さです。

読み込まれるバイト数がバッファの長さより大きい場合、LOB は、入力オフセットからLOBの終わりまで、あるいは指定されたバイト数までの範囲で、いずれかが先にくるところまでストリーム・モードで読み込まれると仮定されています。入力時にこの値が0(ゼロ)の場合、入力オフセットからLOBの末尾までストリーム・モードでデータが読み込まれます。

ストリーム・モード(ポーリングまたはコールバックのいずれかで実行)の場合、LOB 値は入力オフセットから連続して読み込まれます。

データが細分されて読み込まれる場合、**amtp* には常に読み込まれた各ピースの長さが含まれます。

コールバック関数が定義された場合は、パイプから *bufi* バイトが読み込まれるたびにそのコールバック関数が呼び出されます。各ピースが *bufp* に書き込まれます。

コールバック関数が定義されていない場合は、OCI_NEED_DATA エラー・コードが戻ります。アプリケーションでは、OCI_NEED_DATA エラー・コードが戻らなくなるまで何度も OCILobRead() をコールし、LOB のピースを読み込み続けなければなりません。ピースのサイズを変えながら複数の場所を読み込む場合は、コールごとにバッファ・ポインタと長さを変えることができます。

クライアント側のキャラクタ・セットが可変幅の場合は、CLOB および NCLOB の入力は文字数、出力はバイト数で表されます。入力量は、サーバー側 CLOB/NCLOB から読み込む文字数を示します。出力量は、バッファ *bufp* に読み込まれたバイト数を示します。

offset (IN)

入力時、これはLOB 値の先頭からの絶対表示オフセットです。キャラクタLOB (CLOB、NCLOB) では、LOB の先頭からの文字数、バイナリLOB/FILE ではバイト数です。先頭位置は、1 です。

bufp (IN/OUT)

ピースの読み込み先バッファのポインタです。*bufi* のメモリー長が割り当てられるとみなされます。

bufi (IN)

オクテットで示したバッファの長さです。*bufi* パラメータがバイト数で指定され、*amtp* パラメータが文字列で指定されている場合は、この値は、CLOB および NCLOB の *amtp* 値 (*csfrm*=SQLCS_NCHAR) とは異なります。

ctxp (IN)

コールバック関数用のコンテキスト・ポインタです。NULL でもかまいません。

cbfp (IN)

各ピースに対してコールされるように登録できるコールバックです。これが NULL の場合は、ピースごとに OCI_NEED_DATA が戻ります。

コールバック関数は、OCI_CONTINUE を戻して読みを続行する必要があります。これ以外のエラー・コードが戻った場合、LOB の読みは強制終了します。

ctxp (IN)

コールバック関数用のコンテキストです。NULL でもかまいません。

bufp (IN/OUT)

ピース用のバッファ・ポインタです。

len (IN)

bufp における現行のピースのバイト長です。

piece (IN)

ピース : OCI_FIRST_PIECE または OCI_NEXT_PIECE、OCI_LAST_PIECE。

csid (IN)

バッファ・データのキャラクタ・セット ID です。

csfrm (IN)

バッファ・データのキャラクタ・セット・フォームです。*csfrm* パラメータは、LOB の種類と一致していなければなりません。つまり、LOB が CLOB の場合は、*csfrm* には NCHAR を指定できません。LOB が NCLOB の場合は、*csfrm* は NCHAR でなければなりません。

コメント

LOB/FILE の一部をコールの指定どおりにバッファに読み込みます。NULL の LOB または FILE から読み取るとエラーになります。

注意 : LOB を読み込むか、または書き込む場合、指定されたキャラクタ・セット・フォーム (*csfrm*) がロケータのフォームと合致する必要があります。

FILE の場合、サーバー上に既にオペレーティング・システム・ファイルが存在していて、入力ロケータを使用して *OCILobFileOpen()* または *OCILobOpen()* でオープンされていなければなりません。OS ファイルの読み込み許可が Oracle にあること、およびディレクトリ・オブジェクトの読み込み許可がユーザーにあることが必要です。

OCILobRead() に対してポーリング・モードを使用するとき、ユーザーは最初のコールでは *offset* および *amtp* の値を指定する必要がありますが、その後の *OCILobRead()* のポーリング・コールでは指定する必要はありません。

LOB が BLOB の場合は、*csid* および *csfrm* パラメータは無視されます。

注意 : *OCILobRead()* 操作を中止して文ハンドルを解放するには、*OCIBreak()* コールを使います。

以下のルールは、CLOB および NCLOB のクライアント側可変幅キャラクタ・セットに適用されます。

- ポーリング・モードを使う場合は、*OCILobRead()* コール後に *amtp* パラメータの値を参照し、バッファに読み込まれたバイト数を確認してください。バッファの空き領域を確認するためです。
- コールバックを使う場合は、*len* パラメータはコールバックへの入力で、バッファ内で格納されたバイト数を示します。バッファの空き領域を確認するため、コールバック処理中に *len* パラメータをチェックします。

以下のルールは、CLOB および NCLOB のクライアント側固定幅キャラクタ・セットおよびクライアント側可変幅キャラクタ・セットに適用されます。

- CLOB または NCLOB 値を読み込むときに、データベースの CHAR または NCHAR キャラクタ・セットが可変幅の場合は、ユーザー・バッファのすべての領域をデータの格納に使うことはできません。*amtp* パラメータは、実際にユーザー・バッファに読み込まれたバイト数を示します。

UCS-2 形式でデータを読み込むには、*csid* パラメータを OCL_UCS2ID に設定します。*csid* パラメータが設定された場合は、このパラメータによって NLS_LANG 環境変数が上書きされます。UCS-2 (unicode) 形式の追加情報については、5-27 ページの「[固定幅 Unicode サポート](#)」を参照してください。

関連項目： FILE についての追加情報は、『Oracle8i アプリケーション開発者ガイド - ラージ・オブジェクト』の BFILE の説明を参照してください。

LOB の読み込みおよび書き込み方法を示したコード・サンプルについては、Oracle インストールに含まれているデモンストレーション・プログラムを参照してください。追加情報については、[付録 B の「OCI デモ・プログラム」](#)を参照してください。

一般的なピース単位 OCI 操作については、5-30 ページの「[ランタイム・データ割当てとピース単位操作](#)」を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobWrite\(\)](#)、[OCILobFileSetName\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobTrim()

用途

LOB 値を短い長さにトリムします。(切り捨てます。)

構文

```
sword OCILobTrim ( OCISvcCtx      *svchp,  
                  OCIError       *errhp,  
                  OCILobLocator  *locp,  
                  ub4            newlen );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

一意に LOB を参照する内部 LOB ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

newlen (IN)

LOB 値の新しい長さ (現行の長さ以下) です。

コメント

この関数では、LOB データを指定の長さにトリムします。*newlen* が現行の LOB の長さより長い場合は、エラーが戻されます。この関数は内部 LOB にだけ有効です。FILE には使用できません。

関連関数

[OCIErrorGet\(\)](#)、[OCILobRead\(\)](#)、[OCILobAppend\(\)](#)、[OCILobCopy\(\)](#)、[OCILobErase\(\)](#)、[OCILobWrite\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobWrite()

用途

バッファを LOB に書き込みます。

構文

```

sword OCILobWrite ( OCISvcCtx      *svchp,
                   OCIError      *errhp,
                   OCILobLocator *locp,
                   ub4            *amtp,
                   ub4            offset,
                   dvoid          *bufp,
                   ub4            buflen,
                   ub1            piece,
                   dvoid          *ctxp,
                   OCICallbackLobWrite (cbfp)
                   (/*_
                   dvoid *ctxp,
                   dvoid *bufp,
                   ub4 *lenp,
                   ub1 *piecep */)
                   ub2            csid,
                   ub1            csfrm );

```

パラメータ

svchp (IN/OUT)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

一意に LOB を参照する内部 LOB ロケータです。このロケータは、*svchp* に指定したサーバーから取得されたロケータでなければなりません。

amtp (IN/OUT)

入力時は、書き込まれる文字数 (CLOB および NCLOB の場合) またはバイト数 (BLOB の場合) です。出力時は、実際に書き込まれたバイト数または文字数を戻します。このパラメータは常に非 NULL ポインタです。ファイル終りまでの書込みを指定する場合は、変数を宣言してゼロを設定し、そのアドレスをこのパラメータに渡します。

入力時に量が指定されていて、データがピース単位で書き込まれる場合、**amtp* にはコール終了時 (最後のピースが書き込まれた時点) に各ピースの長さの合計が格納され、書込み途

中は未定義になります。ピース単位の読み込みとは異なることに注意してください。指定された量がサーバーに送信されない場合、エラーが戻されます。

amtp が 0 (ゼロ) の場合、ストリーム転送モードが仮定され、ユーザーが `OCI_LAST_PIECE` を指定するまでデータが書き込まれます。

クライアント側のキャラクタ・セットが可変幅の場合は、CLOB および NCLOB に対して入力バイト数、出力は文字数で表されます。入力量は、LOB に書き込むデータのバイト数で、*bufp* 内のバイト数ではありません。*bufp* 内のバイト数は、*buflen* によって指定します。データがピース単位に読み込まれる場合は、書き込むバイト数は、*buflen* よりも大きくなります。出力量は、サーバー側の CLOB/NCLOB に書き込む文字数です。

offset (IN)

入力時、これは LOB 値の先頭からの絶対表示オフセットです。キャラクタ LOB では LOB の先頭からの文字数であり、バイナリ LOB ではバイト数です。先頭位置は 1 です。

bufp (IN)

ピースの書き込み元バッファのポインタです。バッファ内のデータの長さは、*buflen* に渡された値であるとみなされます。データがポーリング・メソッドを使用してピース単位で書き込まれる場合でも、*bufp* はこのコールが呼び出された時の LOB の最初のピースを格納する必要があります。コールバックが使用される場合、データの提供に *bufp* は使用できません。使用するとエラーになります。

buflen (IN)

バッファ内のデータの長さ (バイト数) です。*buflen* パラメータがバイト数で指定され、*amtp* パラメータが文字列で指定されている場合は、この値は、CLOB および NCLOB の *amtp* 値とは異なります。

注意: このパラメータは 8 ビット (1 バイト) を仮定します。現行のプラットフォームでこれより長いバイトを使用している場合は、*buflen* の値をそれに合わせて調整する必要があります。

piece (IN)

書き込み中のバッファのピースです。このパラメータのデフォルト値は、バッファが単独のピースとして書き込まれることを示す `OCI_ONE_PIECE` です。

ピース単位またはコールバック・モードの値には、`OCI_FIRST_PIECE`、`OCI_NEXT_PIECE`、`OCI_LAST_PIECE` も使うことができます。

ctxp (IN)

コールバック関数用のコンテキストです。NULL でもかまいません。

cbfp (IN)

ピース単位書き込みで各ピースに対してコールされるように登録できるコールバックです。これが NULL の場合は、標準ポーリング・メソッドが使用されます。

コールバック関数は、OCI_CONTINUE を戻して書込みを続行する必要があります。これ以外のエラー・コードが戻った場合、LOB の書込みは強制終了します。コールバックは、次のパラメータを取ります。

ctxp (IN)

コールバック関数用のコンテキストです。NULL でもかまいません。

bufp (IN/OUT)

ピース用のバッファ・ポインタです。このパラメータは、OCILobWrite() ルーチンに入力として渡される *bufp* と同じです。

lenp (IN/OUT)

buffer (IN) 内のデータのバイト数、および *bufp* (OUT) 内の現行ピースのバイト数です。

piecep (OUT)

ピース: OCI_NEXT_PIECE または OCI_LAST_PIECE です。

csid (IN)

バッファ・データのキャラクタ・セット ID です。

csfrm (IN)

バッファ・データのキャラクタ・セット・フォームです。 *csfrm* パラメータは、LOB の型と一貫していなければなりません。つまり、LOB が CLOB の場合は、*csfrm* は NCHAR を示してはならず、LOB が NCLOB の場合は、*csfrm* は NCHAR を示す必要があります。

コメント

バッファを指定どおりに内部 LOB に書き込みます。LOB に既にデータが存在する場合は、バッファに格納されたデータによって上書きされます。バッファは、このコールによって単独のピースとして LOB に書き込むことも、コールバックまたは標準ポーリング・メソッドを使用してピース単位で用意することもできます。

注意: LOB を読み込むか、または書き込む場合、指定されたキャラクタ・セット・フォーム (*csfrm*) がロケータのフォームと合致する必要があります。

OCILobWrite() に対してポーリング・モードを使っているとき、ユーザーは最初のコールでは *offset* および *amtp* の値を指定する必要がありますが、その後の OCILobWrite() のコールでは指定する必要はありません。

piece パラメータの値が OCI_FIRST_PIECE である場合、データによっては、コールバックまたはポーリングを介して提供される必要があります。

cbfp パラメータにコールバック関数が定義された場合は、パイプに 1 ピースが書き込まれると、このコールバック関数が呼び出され、次のピースが取得されます。各ピースが *bufp* から書き込まれます。コールバック関数が定義されていない場合、OCILobWrite() は、OCI_NEED_DATA エラー・コードを戻します。LOB のピースの書き込みを続けるには、アプリケーションで OCILobWrite() を再度コールする必要があります。このモードでは、ピースが

別々のサイズで複数の場所に読み込まれる場合は、コールごとにバッファ・ポインタと長さを変えることができます。

piece パラメータの値が `OCI_LAST_PIECE` のときは、ポーリングまたはコールバック・メソッドが使用されているかどうかにかかわらず、ピース単位の書込み操作は終了します。

(どの入力方法を使用した場合も) Oracle に渡されるデータの量が、*amt* パラメータに指定された量より少ない場合は、ORA-22993 エラーが戻されます。

この関数は内部 LOB にだけ有効です。FILE は読み専用なので、取り扱えません。LOB が BLOB の場合は、*csid* および *csfrm* パラメータは無視されます。

クライアント側のキャラクタ・セットが可変幅の場合は、CLOB および NCLOB に対して入力はバイト数、出力は文字数で表されます。入力量は、ユーザーが LOB に書き込みたいデータのバイト数で、*bufp* 内のバイト数ではありません。*bufp* 内のバイト数は、*buflen* によって指定されます。データがピースごとに読み込まれる場合は、書き込むバイトの量は、*buflen* よりも大きくできます。出力量は、サーバー側 CLOB/NCLOB に書き込む文字数を指します。

UCS-2 形式でデータを書き込むには、*csid* パラメータを `OCI_UCS2ID` に設定します。*csid* パラメータが設定された場合は、このパラメータによって `NLS_LANG` 環境変数が上書きされます。UCS-2 (unicode) 形式の詳細は、5-27 ページの「[固定幅 Unicode サポート](#)」を参照してください。

関連項目 : LOB の読みおよび書込み方法を示したコード・サンプルについては、Oracle インストラクションに含まれているデモンストレーション・プログラムを参照してください。追加情報については、[付録 B の「OCI デモ・プログラム」](#)を参照してください。

一般的なピース単位 OCI 操作については、5-30 ページの「[ランタイム・データ割当てとピース単位操作](#)」を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobRead\(\)](#)、[OCILobAppend\(\)](#)、[OCILobCopy\(\)](#)、[OCILobWriteAppend\(\)](#)

OCILobWriteAppend()

用途

LOB の末尾からデータを書き込みます。

構文

```
sword OCILobWriteAppend ( OCISvcCtx *svchp,
                          OCIError *errhp,
                          OCILobLocator *locp,
                          ub4 *amtp,
                          dvoid *bufp,
                          ub4 buflen,
                          ub1 piece,
                          dvoid *ctxp,
                          OCICallbackLobWrite (cbfp)
                          /*_
                          dvoid *ctxp,
                          dvoid *bufp,
                          ub4 *lenp,
                          ub1 *piecep */)
                          ub2 csid,
                          ub1 csfrm);
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

locp (IN/OUT)

一意に LOB を参照する内部 LOB ロケータです。

amtp (IN/OUT)

入力時は、書き込まれる文字数 (CLOB/NCLOB の場合) またはバイト数 (BLOB の場合) です。出力時は、実際に書き込まれたバイト数または文字数を戻します。入力時に量が指定されていて、データがピース単位で書き込まれる場合、*amtp にはコール終了時 (最後のピースが書き込まれた時点) に各ピースの長さの合計が格納され、書込み途中は未定義になります。(ピース単位の読み込みとは異なることに注意してください。) 指定された量がサーバーに送信されない場合、エラーが戻されます。amtp が 0 (ゼロ) の場合、ストリーム転送モードが仮定され、ユーザーが OCI_LAST_PIECE を指定するまでデータが書き込まれます。

クライアント側のキャラクタ・セットが可変幅の場合は、CLOB/NCLOB の入力量は文字数ではなくバイト数で表されます。

bufp (IN)

ピースの書き込み元バッファのポインタです。バッファ内のデータの長さが *buflen* に渡された値であると見なされます。データがピース単位で書き込まれる場合でも、*bufp* には、この関数がコールされたときに、LOB の最初のピースを格納する必要があります。コールバックが使用される場合、データの提供に *bufp* は使用できません。使用するとエラーになります。

buflen (IN)

バッファ内のデータの長さ (バイト数) です。このパラメータは 8 ビット (1 バイト) を仮定します。現行のプラットフォームでこれより長いバイトを使用している場合は、*buflen* の値をそれに合わせて調整する必要があります。

piece (IN)

書き込み中のバッファのピースです。このパラメータのデフォルト値は、バッファが単独のピースとして書き込まれることを示す `OCI_ONE_PIECE` です。ピース単位またはコールバック・モードで可能なその他の値としては、`OCI_FIRST_PIECE` および `OCI_NEXT_PIECE`、`OCI_LAST_PIECE` などがあります。

ctxp (IN)

コールバック関数用のコンテキストです。NULL でもかまいません。

cbfp (IN)

ピース単位書き込みで各ピースに対してコールされるように登録できるコールバック。これが NULL の場合は、標準ポーリング・メソッドが使用されます。コールバック関数は、`OCI_CONTINUE` を戻して書き込みを続行する必要があります。これ以外のエラー・コードが戻った場合、LOB の書き込みは強制終了します。コールバックは、次のパラメータを取ります。

ctxp (IN)

コールバック関数用のコンテキストです。NULL でもかまいません。

bufp (IN/OUT)

ピース用のバッファ・ポインタです。

lenp (IN/OUT)

buffer (IN) 内のデータのバイト数、および *bufp* (OUT) 内の現行ピースのバイト数です。

piecep (OUT)

ピース : `OCI_NEXT_PIECE` または `OCI_LAST_PIECE`。

csid (IN)

バッファ・データのキャラクタ・セット ID です。

csfrm (IN)

バッファ・データのキャラクタ・セット・フォームです。

コメント

バッファは、このコールによって単独のピースとしてLOBに書き込むことも、コールバックまたは標準ポーリング・メソッドを使用してピース単位で用意することもできます。pieceパラメータの値がOCI_FIRST_PIECEである場合、データによっては、コールバックまたはポーリングを介して提供される必要があります。cbfpパラメータにコールバック関数が定義された場合は、パイプに1ピースが書き込まれるとこのコールバック関数が呼び出され、次のピースが取得されます。各ピースがbufpから書き込まれます。コールバック関数が定義されていない場合、OCILobWriteAppend()は、OCI_NEED_DATAエラー・コードを戻します。

LOBのピースの書込みを続けるには、アプリケーションでOCILobWriteAppend()を再度コールする必要があります。このモードでは、ピースが別々のサイズで複数の場所に読み込まれる場合は、コールごとにバッファ・ポインタと長さを変えることができます。pieceパラメータの値がOCI_LAST_PIECEのときは、ピース単位の書込み操作は終了します。

LOBバッファリングが有効な場合は、OCILobWriteAppend()はサポートされません。

LOBがBLOBの場合は、csidおよびcsfrmパラメータは無視されます。

クライアント側キャラクタ・セットが可変幅の場合は、CLOB/NCLOBの入力量は文字数ではなくバイト数で表されます。

関連項目 : 詳細は7-9ページの「LOBの読み / 書込みパフォーマンスを向上するための関数」を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCILobRead\(\)](#)、[OCILobAppend\(\)](#)、[OCILobCopy\(\)](#)、[OCILobWrite\(\)](#)

文関数

この項では、文関数について説明します。

表 15-7 OCI クイック・リファレンス

関数	用途	ページ
OCIStmtExecute()	実行する文をサーバーに送信します。	15-164 ページ
OCIStmtFetch()	問合せから行をフェッチします。	15-167 ページ
OCIStmtGetPieceInfo()	ピース単位で操作するためのピース情報を取得します。	15-168 ページ
OCIStmtPrepare()	アプリケーション要求を確立します。	15-170 ページ
OCIStmtSetPieceInfo()	ピース単位で操作するためのピース情報を設定します。	15-172 ページ

OCIStmtExecute()

用途

このコールは、アプリケーション要求をサーバーに関連付けます。

構文

```
sword OCIStmtExecute ( OCISvcCtx          *svchp,
                      OCIStmt          *stmtp,
                      OCIError         *errhp,
                      ub4               iters,
                      ub4               rowoff,
                      CONST OCISnapshot *snap_in,
                      OCISnapshot      *snap_out,
                      ub4               mode );
```

パラメータ

svchp (IN/OUT)

サービス・コンテキスト・ハンドルです。

stmtp (IN/OUT)

文ハンドルです。サーバーで実行される文および対応付けられたデータを定義します。svchpが Oracle7 のサーバーを示しているときに、リリース 8 でだけサポートされるデータ型のバインドを持つ文ハンドルを渡しても無効になります。

errhp (IN/OUT)

エラー時の診断情報のために OCIErrorGet() に渡せるエラー・ハンドルです。

iters (IN)

SELECT 文以外の文の場合、この文が実行される回数です。

SELECT 文では、iters がゼロ以外の場合は、文ハンドルに対する定義を行う必要があります。実行すると、iters が事前定義バッファにフェッチされ、プリフェッチ行カウントに従ってさらに行がプリフェッチされます。SELECT 文に取り出される行数がわからない場合は、iters をゼロに設定します。

この関数は、SELECT 文以外に対して iters=0 の場合はエラーを戻します。

rowoff (IN)

この複数行実行に関連する配列バインドのデータが始まる開始索引です。

snap_in (IN)

このパラメータはオプションです。指定する場合は、OCI_DTYPE_SNAP 型のスナップショット記述子を指示する必要があります。この記述子の内容は、直前のコールの snap_out パラメータから取得する必要があります。この記述子は、SQL が SELECT でない場合は無

視されます。この機能を使うと、Oracle への複数のサービスキュエリ・コンテキストにより、データベースのコミット済みデータに対し一貫性のあるスナップショットを見ることができます。ただし、あるコンテキストでコミットされていないデータは、同じスナップショットを使用しても別のコンテキストで参照することはできません。

snap_out (OUT)

このパラメータはオプションです。指定する場合は、OCI_DTYPE_SNAP 型の記述子を指示する必要があります。この記述子には、現行の Oracle 「システム変更番号」が暗号化されて格納されており、後続の `OCIStmtExecute()` コールの `snap_in` への入力値として使うことができます。この記述子は必要以上に長く使わないでください。「スナップショットが古すぎます」というエラーが発生します。

mode (IN)

次のモードが有効です。

- OCI_DEFAULT - このモードで `OCIStmtExecute()` をコールすると文が実行されます。また、選択リストに関する記述情報が暗黙的に戻されます。
- OCI_DESCRIBE_ONLY - このモードは、実行前に問合せを記述するユーザー用です。`OCIStmtExecute()` をこのモードでコールすると、文は実行されませんが、選択リスト記述は戻されます。パフォーマンスを最大にするために、アプリケーションではデフォルト・モードで文を実行し、実行に伴う暗黙的な記述を使用することをお勧めします。
- OCI_COMMIT_ON_SUCCESS - このモードで文を実行した場合、実行が正常に終了すると、実行後に現行のトランザクションがコミットされます。
- OCI_EXACT_FETCH - アプリケーションでフェッチされる行数があらかじめ正確にわかっているときに使用します。このモードは Oracle リリース 8 モード用のプリフェッチをオフにします。また、実行がコールされる前に定義されていなければなりません。このモードを使用すると要求した行がフェッチされた後カーソルが取り消されるため、サーバー側のリソース使用量を削減できます。
- OCI_BATCH_ERRORS - このモードの詳細は、4-8 ページの「[OCIStmtExecute\(\) 用のバッチ・エラー・モード](#)」を参照してください。

これらのモードは相互排他的ではなく、組み合わせて使うことができます。

コメント

この関数は、準備された SQL 文を実行するために使用します。アプリケーションは、実行コールを使用して要求をサーバーに関連付けます。

SELECT 文が実行されると、選択リストの記述がレスポンスとして暗黙的に使用可能になります。この記述は、記述およびフェッチ、型変換定義用にクライアント側にバッファ処理されます。したがって、選択リストの記述は、実行後だけに行うのが最善の方法です。詳細は、4-10 ページの「[選択リスト項目の記述](#)」を参照してください。

SELECT 文では、一部の結果も暗黙のうちに取得されます。実行終了の時点で行が受け取られ、バッファ処理されます。行数が少ない問合せでは、プリフェッチすることでフェッチの

最後に達したときにサーバー内のメモリーが解放され、これによってメモリーの使用量が削減されるように最適化できます。プリフェッチする行数を結果セットごとに設定する属性設定コールが定義されています。

SELECT 文では、文ハンドルは、実行終了の時点では、それが実行されたサービス・コンテキストに対する参照を暗黙のうちに保持しています。サービス・コンテキストの完全性は、ユーザーに保守の義務があります。暗黙参照は、文ハンドルが解放されるまたはフェッチが取り消される、あるいはフェッチ条件の最後に達するまで保持されます。

注意: *OCIStmtExecute()* のコール前に SELECT 文に対して出力変数が定義されると、*iters* によって指定された行数が、定義済み出力バッファに直接フェッチされ、プリフェッチ件数と等価であるその他の行がプリフェッチされます。追加行がない場合、フェッチは *OCIStmtFetch()* をコールしないで完了します。

関連関数

[*OCIStmtPrepare\(\)*](#)

OCIStmtFetch()

用途

問合せから行をフェッチします。

構文

```
sword OCIStmtFetch ( OCIStmt      *stmtp,  
                    OCIError    *errhp,  
                    ub4          nrows,  
                    ub2          orientation,  
                    ub4          mode );
```

パラメータ

stmtp (IN)

文 (アプリケーション要求) ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

nrows (IN)

現行の位置からフェッチされる行数です。

orientation (IN)

リリース 8.0 での許容値は、デフォルト値でもある *OCI_FETCH_NEXT* だけです。

mode (IN)

OCI_DEFAULT として渡します。

コメント

プリフェッチされた行で間に合う場合、フェッチ・コールはローカル・コールになります。ただしこれは、アプリケーションに対して透過的に行われます。

LOB 列が読み込まれる場合、LOB ロケータに対して実行される後続の LOB 操作のために、それらのロケータがフェッチされます。プリフェッチは、Long 列が関係する場合はオフに切り替わります。

この関数は、データが切り捨てられる場合またはファイルが終了した場合に *OCI_SUCCESS_WITH_INFO* を戻します。*nrows* パラメータにゼロを設定して *OCIStmtFetch()* をコールした場合は、カーソルが取り消されます。

関連関数

[OCIStmtExecute\(\)](#)

OCIStmtGetPieceInfo()

用途

ピース単位操作のピース情報を戻します。

構文

```
sword OCIStmtGetPieceInfo( CONST OCIStmt  *stmtp,
                           OCIError      *errhp,
                           dvoid          **hndlpp,
                           ub4           *typep,
                           ub1           *in_outp,
                           ub4           *iterp,
                           ub4           *idxp,
                           ub1           *piecep );
```

パラメータ

stmtp (IN)

戻された OCI_NEED_DATA が実行される際の文です。

errhp (OUT)

エラー時の診断情報のために OCIErrorGet() に渡せるエラー・ハンドルです。

hndlpp (OUT)

バインド、バインドの定義ハンドル、またはそのランタイム・データが要求または提供されている定義のポインタを戻します。

typep (OUT)

hndlpp が指すハンドルのタイプです。タイプには、OCI_HTYPE_BIND (バインド・ハンドル用) または OCI_HTYPE_DEFINE (定義ハンドル用) があります。

in_outp (OUT)

IN バインド値に対してデータが必要な場合、OCI_PARAM_IN を戻します。データが OUT バインド変数または定義位置値として取得できる場合は OCI_PARAM_OUT が戻ります。

iterp (OUT)

複数行操作の行数を戻します。

idxp (OUT)

PL/SQL 配列バインド操作の配列要素の索引です。

piecep (OUT)

OCI_ONE_PIECE、OCI_FIRST_PIECE、OCI_NEXT_PIECE、または OCI_LAST_PIECE のいずれかの事前定義値を戻します。

コメント

実行 / フェッチ・コールから OCI_NEED_DATA が戻され、動的なバインド / 定義の値またはピースが取得される / 戻されると、*OCIStmtGetPieceInfo()* から、バインド / 定義ハンドル、反復、索引番号、ピース情報などの関連情報が戻されます。

OCIStmtGetPieceInfo() の使用方法の詳細は、5-30 ページの「ランタイム・データ割当てとピース単位操作」を参照してください。

関連関数

OCIAttrGet()、*OCIAttrSet()*、*OCIStmtExecute()*、*OCIStmtFetch()*、*OCIStmtSetPieceInfo()*

OCIStmtPrepare()

用途

このコールは、実行する SQL 文または PL/SQL 文を準備します。

構文

```
sword OCIStmtPrepare ( OCIStmt      *stmtp,  
                      OCIError     *errhp,  
                      CONST text    *stmt,  
                      ub4           stmt_len,  
                      ub4           language,  
                      ub4           mode );
```

パラメータ

stmtp (IN)

文ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

stmt (IN)

実行される SQL 文または PL/SQL 文です。NULL で終わる文字列でなければなりません。文が実行される、またはそこからデータがフェッチされる限り、文のテキストに対するポインタが利用できなければなりません。

stmt_len (IN)

文の長さです。0 (ゼロ) 以外でなければなりません。

language (IN)

V7 構文またはネイティブ構文を指定します。可能な値は次のとおりです。

- OCI_V7_SYNTAX - V7 Oracle 解析構文。
- OCI_NTV_SYNTAX - サーバーのバージョンに依存する構文。

mode (IN)

次の値があります。

- OCI_DEFAULT - デフォルト・モード。
- OCI_NO_SHARING - SQL 文の共有モードを無効にします。2-19 ページの「[共有データ・モード](#)」を参照してください。

コメント

このコールは、OCI アプリケーションで実行する SQL 文または PL/SQL 文を準備するために使用します。 *OCIStmtPrepare()* コールは、アプリケーション要求を定義します。

これは、純粹にローカル・コールです。後続のバインド・コールで初期化されるこの文のためのデータ値は、この文ハンドルをハング・オフするバインド・ハンドル内に格納されません。

このコールは、この文ハンドルと特定のサーバー間の関連性は作成しません。

このコールの使用方法の詳細は、4-4 ページの「[文の準備](#)」を参照してください。

関連関数

[OCIAttrGet\(\)](#)、[OCIStmtExecute\(\)](#)

OCIStmtSetPieceInfo()

用途

ピース単位操作のピース情報を設定します。

構文

```
sword OCIStmtSetPieceInfo ( dvoid          *hdlp,  
                           ub4            type,  
                           OCIError      *errhp,  
                           CONST dvoid    *bufp,  
                           ub4            *alenp,  
                           ub1            piece,  
                           CONST dvoid    *indp,  
                           ub2           *rcodep );
```

パラメータ

hdlp (IN/OUT)

バインド・ハンドルまたは定義ハンドルです。

type (IN)

ハンドルのタイプです。

errhp (OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

bufp (IN/OUT)

IN バインド変数の場合はデータ値またはピースを含む記憶領域のポインタです。それ以外の場合、*bufp* は OUT バインドおよび定義変数に対してピースまたはデータ値を取得するための記憶領域のポインタです。名前付きデータ型または REF では、オブジェクトまたは REF のポインタが戻ります。

alenp (IN/OUT)

ピースまたは値の長さです。

piece (IN)

ピース・パラメータです。有効な値は次のとおりです。

- OCI_ONE_PIECE
- OCI_FIRST_PIECE
- OCI_NEXT_PIECE
- OCI_LAST_PIECE

このパラメータは、IN バインド変数でだけ使用されます。

indp (IN/OUT)

標識です。名前付きデータ型 (SQLT_NTY) と REF(SQLT_REF) 用の sb2 値のポインタまたは標識構造体のポインタです。つまり *indp は、データ型に応じて sb2 または dvoid* のどちらかになります。

rancodep (IN/OUT)

リターン・コードです。

コメント

実行コールから OCI_NEED_DATA が戻され、動的 IN/OUT バインドの値またはピースが取得されると、OCIStmtSetPieceInfo() により、バッファ、長さ、現在処理されているピース、標識、およびこの列のリターン・コードなどのピース情報が設定されます。

OCIStmtSetPieceInfo() の使用方法の詳細は、5-30 ページの「ランタイム・データ割当てとピース単位操作」を参照してください。

関連関数

[OCIAttrGet\(\)](#)、[OCIAttrSet\(\)](#)、[OCIStmtExecute\(\)](#)、[OCIStmtFetch\(\)](#)、[OCIStmtGetPieceInfo\(\)](#)

スレッド管理関数

この項では、スレッド管理関数について説明します。

表 15-8 OCI クイック・リファレンス

関数	用途	ページ
OCIThreadClose()	スレッド・ハンドルをクローズします。	15-176 ページ
OCIThreadCreate()	新しいスレッドを作成します。	15-177 ページ
OCIThreadHandleGet()	コールが行われたスレッドの <code>OCIThreadHandle</code> を検索します。	15-179 ページ
OCIThreadHndDestroy()	スレッド・ハンドルの破棄および割当て解除を行います。	15-180 ページ
OCIThreadHndInit()	スレッド・ハンドルの割当ておよび初期化を行います。	15-181 ページ
OCIThreadIdDestroy()	スレッド ID の破棄および割当て解除を行います。	15-182 ページ
OCIThreadIdGet()	コールが行われたスレッドの <code>OCIThreadId</code> を検索します。	15-183 ページ
OCIThreadIdInit()	スレッド ID の割当ておよび初期化を行います。	15-184 ページ
OCIThreadIdNull()	特定の <code>OCIThreadId</code> が NULL スレッド ID かどうかを決定します。	15-185 ページ
OCIThreadIdSame()	2 つの <code>OCIThreadId</code> が同じスレッドを表しているかどうかを決定します。	15-186 ページ
OCIThreadIdSet()	<code>OCIThreadId</code> を別の <code>OCIThreadId</code> に設定します。	15-187 ページ
OCIThreadIdSetNull()	特定の <code>OCIThreadId</code> に NULL スレッド ID を設定します。	15-188 ページ
OCIThreadInit()	<code>OCIThread</code> コンテキストを初期化します。	15-189 ページ
OCIThreadIsMulti()	アプリケーションがマルチスレッド環境とシングルスレッド環境のどちらで動作しているかを、コール側に伝えます。	15-190 ページ
OCIThreadJoin()	コール側スレッドから別のスレッドに結合できるようにします。	15-191 ページ
OCIThreadKeyDestroy()	<code>key</code> によってポイントされているキーの破棄および割当て解除を行います。	15-192 ページ
OCIThreadKeyGet()	キーに対してコール側スレッドの現行の値を取得します。	15-193 ページ
OCIThreadKeyInit()	キーを作成します。	15-194 ページ
OCIThreadKeySet()	キーに対してコール側スレッドの値を設定します。	15-195 ページ
OCIThreadMutexAcquire()	コールが行われたスレッドに対して <code>mutex</code> を取得します。	15-196 ページ
OCIThreadMutexDestroy()	<code>mutex</code> の破棄および割当て解除を行います。	15-197 ページ
OCIThreadMutexInit()	<code>mutex</code> の割り当ておよび初期化を行います。	15-198 ページ

表 15-8 OCI クイック・リファレンス (続き)

関数	用途	ページ
OCIThreadMutexRelease()	mutex を解放します。	15-199 ページ
OCIThreadProcessInit()	OCIThread プロセスの初期化を行います。	15-200 ページ
OCIThreadTerm()	OCIThread コンテキストを解放します。	15-201 ページ

OCIThreadClose()

用途

スレッド・ハンドルをクローズします。

構文

```
sword OCIThreadClose ( dvoid          *hndl,  
                      OCIError       *err,  
                      OCIThreadHandle *tHnd );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。診断情報は、*OCIErrorGet()* をコールして取得できます。

tHnd (IN/OUT)

クローズする OCIThread スレッド・ハンドルです。

コメント

tHnd は、*OCIThreadHndInit()* を使って初期化する必要があります。同一 *OCIThreadCreate()* コールから戻されたスレッド・ハンドルおよびスレッド ID は、*OCIThreadClose()* コール後はどちらも無効になります。

関連関数

[OCIThreadCreate\(\)](#)

OCIThreadCreate()

用途

新しいスレッドを作成します。

構文

```
sword OCIThreadCreate ( dvoid          *hndl,  
                       OCIError      *err,  
                       void (*start)  (dvoid  
                       dvoid          *arg,  
                       OCIThreadId   *tid,  
                       OCIThreadHandle *tHnd );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。診断情報は、OCIErrorGet() をコールして取得できます。

start (IN)

新しいスレッドが実行を開始する関数です。

arg (IN)

start によってポイントされている関数に渡す引数です。

tid (IN/OUT)

NULL でない場合は、新しいスレッドに ID を取得します。

tHnd (IN/OUT)

NULL でない場合は、新しいスレッドにハンドルを取得します。

コメント

arg から渡された引数を使って、*start* によってポイントされている関数コールを実行すると、新しいスレッドが開始します。その関数が復帰すると、新しいスレッドは終了します。関数は値を戻さず、パラメータ *dvoid* を受け入れます。*tHnd* が非 NULL の場合に限り、OCIThreadCreate() コールおよび OCIThreadClose() コールは対でなければなりません。

tHnd が NULL の場合は、生成されるスレッドの終了のタイミングがわからないので、**tid* 内に配置されたスレッド ID はコール側スレッドでは無効になります。

OCIThreadCreate()

tid は *OCIThreadIdInit()* によって初期化します。また、*tHnd* は *OCIThreadHndInit()* によって初期化します。

関連関数

[OCIThreadClose\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadHndInit\(\)](#)

OCIThreadHandleGet()

用途

コールが行われたスレッドの `OCIThreadHandle` を検索します。

構文

```
sword OCIThreadHandleGet ( dvoid          *hdl,  
                           OCIError      *err,  
                           OCIThreadHandle *tHnd );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、`OCI_ERROR` が戻されます。診断情報は、`OCIErrorGet()` をコールして取得できます。

tHnd (IN/OUT)

NULL でない場合は、スレッドのスレッド・ハンドルを配置する場所です。

コメント

tHnd は、`OCIThreadHndInit()` を使って初期化する必要があります。

この関数によって検索されるスレッド・ハンドル *tHnd* は、使用後に `OCIThreadClose()` を使ってクローズし、`OCIThreadHndDestroy()` を使って破棄する必要があります。

関連関数

[OCIThreadHndDestroy\(\)](#)、[OCIThreadHndInit\(\)](#)

OCIThreadHndDestroy()

用途

スレッド・ハンドルの破棄および割当て解除を行います。

構文

```
sword OCIThreadHndDestroy ( dvoid          *hndl,  
                             OCIError      *err,  
                             OCIThreadHandle **thnd );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。診断情報は、*OCIErrorGet()* をコールして取得できます。

thnd (IN/OUT)

破棄するスレッド・ハンドルへのポインタのアドレスです。

コメント

thnd は、*OCIThreadHndInit()* を使って初期化する必要があります。

関連関数

[OCIThreadHandleGet\(\)](#)、[OCIThreadHndInit\(\)](#)

OCIThreadHndInit()

用途

スレッド・ハンドルの割当ておよび初期化を行います。

構文

```
sword OCIThreadHndInit ( dvoid          *hndl,  
                        OCIError       *err,  
                        OCIThreadHandle **thnd );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。診断情報は、*OCIErrorGet()* をコールして取得できます。

thnd (OUT)

初期化するスレッド・ハンドルへのポインタのアドレスです。

関連関数

[OCIThreadHandleGet\(\)](#)、[OCIThreadHndDestroy\(\)](#)

OCIThreadIdDestroy()

用途

スレッド ID の破棄および割当て解除を行います。

構文

```
sword OCIThreadIdDestroy (dvoid          *hndl,  
                          OCIError      *err,  
                          OCIThreadId  **tid );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

tid (IN/OUT)

破棄するスレッド ID へのポインタです。

コメント

tid は、OCIThreadIdInit() を使って初期化する必要があります。

関連関数

[OCIThreadIdGet\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)、[OCIThreadIdSetNull\(\)](#)

OCIThreadIdGet()

用途

コールされるスレッドの `OCIThreadId` を検索します。

構文

```
sword OCIThreadIdGet ( dvoid          *hndl,  
                      OCIError      *err,  
                      OCIThreadId   *tid );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、`OCL_ERROR` が戻されます。診断情報は、`OCIErrorGet()` をコールして取得できます。

tid (OUT)

コール側スレッドの ID を配置する場所をポイントする必要があります。

コメント

tid は、`OCIThreadIdInit()` を使って初期化する必要があります。OCIThread がシングルスレッド環境で使われた場合は、*tid* によってポイントされている場所に、`OCIThreadIdGet()` によって常に同じ値が配置されます。正確な値であることも重要ですが、この値が NULL スレッド ID とは異なり、常に同じ値であるということが重要です。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)、[OCIThreadIdSetNull\(\)](#)

OCIThreadIdInit()

用途

スレッド ID tid の割当ておよび初期化を行ないます。

構文

```
sword OCIThreadIdInit ( dvoid          *hdl,  
                       OCIError      *err,  
                       OCIThreadId   **tid );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

tid (OUT)

初期化するスレッド ID へのポインタです。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdGet\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)、[OCIThreadIdSetNull\(\)](#)

OCIThreadIdNull()

用途

特定の OCIThreadId が NULL スレッド ID かどうかを決定します。

構文

```
sword OCIThreadIdNull ( dvoid          *hdl,  
                       OCIError      *err,  
                       OCIThreadId   *tid,  
                       boolean        *result );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCL_ERROR が戻されます。診断情報は、*OCIErrorGet()* をコールして取得できます。

tid (IN)

チェックする OCIThreadId へのポインタです。

result (IN/OUT)

結果へのポインタです。

コメント

tid が NULL スレッド ID の場合は、*result* は TRUE に設定されます。そうでない場合は、*result* は FALSE に設定されます。*tid* は、*OCIThreadIdInit()* を使って初期化する必要があります。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdGet\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)、[OCIThreadIdSetNull\(\)](#)

OCIThreadIdSame()

用途

2 つの OCIThreadId が同じスレッドを表しているかどうかを決定します。

構文

```
sword OCIThreadIdSame ( dvoid          *hndl,  
                        OCIError       *err,  
                        OCIThreadId    *tid1,  
                        OCIThreadId    *tid2,  
                        boolean         *result );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。診断情報は、*OCIErrorGet()* をコールして取得できます。

tid1 (IN)

最初の OCIThreadId へのポインタです。

tid2 (IN)

2 番目の OCIThreadId へのポインタです。

result (IN/OUT)

結果へのポインタです。

コメント

tid1 および *tid2* が同じスレッドを表す場合は、*result* は TRUE に設定されます。そうでない場合は、*result* は FALSE に設定されます。*tid1* および *tid2* がどちらも NULL スレッド ID の場合は、*result* は TRUE に設定されます。*tid1* および *tid2* は、*OCIThreadIdInit()* を使って初期化する必要があります。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdGet\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSet\(\)](#)、[OCIThreadIdSetNull\(\)](#)

OCIThreadIdSet()

用途

OCIThreadId を別の OCIThreadId に設定します。

構文

```
sword OCIThreadIdSet ( dvoid          *hndl,  
                      OCIError       *err,  
                      OCIThreadId    *tidDest,  
                      OCIThreadId    *tidSrc );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

tidDest (OUT)

設定後の OCIThreadId の場所をポイントする必要があります。

tidSrc (IN)

設定前の OCIThreadId をポイントする必要があります。

コメント

tid は、OCIThreadIdInit() を使って初期化する必要があります。

関連関数

OCIThreadIdDestroy()、OCIThreadIdGet()、OCIThreadIdInit()、OCIThreadIdNull()、OCIThreadIdSame()、OCIThreadIdSetNull()

OCIThreadIdSetNull()

用途

特定の `OCIThreadId` に NULL スレッド ID を設定します。

構文

```
sword OCIThreadIdSetNull ( dvoid          *hndl,  
                           OCIError       *err,  
                           OCIThreadId    *tid );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、`OCI_ERROR` が戻されます。診断情報は、`OCIErrorGet()` をコールして取得できます。

tid (OUT)

NULL スレッド ID を設定する `OCIThreadId` をポイントする必要があります。

コメント

tid は、`OCIThreadIdInit()` を使って初期化する必要があります。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdGet\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)

OCIThreadInit()

用途

OCIThread コンテキストを初期化します。

構文

```
sword OCIThreadInit ( dvoid      *hndl,  
                     OCIError  *err );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

コメント

戻されたポインタによってポイントされているメモリーを、OCIThread クライアントから検査しないでください。同時に OCIThreadInit() コールを実行してください。OCIThreadProcessInit() と異なり、他のコールより先に初期コールを行う必要はありません。

OCIThreadInit() の 1 回目のコールでは、OCI スレッド・コンテキストが初期化されます。また、そのコンテキストへのポインタを、システム情報も含めて保存します。2 回目以降の OCIThreadInit() コールでは、同じコンテキストが戻されます。

各 OCIThreadInit() コールは、OCIThreadTerm() コールと対になっている必要があります。

関連関数

[OCIThreadTerm\(\)](#)

OCIThreadIsMulti()

用途

アプリケーションがマルチスレッド環境とシングル・スレッド環境のどちらで動作しているかを、コール側に伝えます。

構文

```
boolean OCIThreadIsMulti ( );
```

戻り値

TRUE - 環境がマルチスレッドの場合。

FALSE - 環境がシングルスレッドの場合。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdGet\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)

OCIThreadJoin()

用途

コール側スレッドが別のスレッドと結合できるようにします。

構文

```
sword OCIThreadJoin ( dvoid          *hdl,  
                     OCIError       *err,  
                     OCIThreadHandle *tHnd );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。診断情報は、OCIErrorGet() をコールして取得できます。

tHnd (IN)

結合するスレッドの OCIThreadHandle です。

コメント

この関数は、指定されたスレッドが終了するまでコール側をブロックします。

tHnd は、OCIThreadHndInit() を使って初期化する必要があります。複数のスレッドをすべて同一のスレッドに結合した場合、結果は保証されていません。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdGet\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)

OCIThreadKeyDestroy()

用途

key によってポイントされているキーの破棄および割当て解除を行います。

構文

```
sword OCIThreadKeyDestroy ( dvoid          *hdl,  
                           OCIError      *err,  
                           OCIThreadKey **key );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは *err* に記録され、*OCIErrorGet()* のコールによって診断情報を取得できます。

key (IN/OUT)

キーを破棄する *OCIThreadKey* です。

コメント

キー作成ルーチンに渡されるデストラクタ関数コールバックとは異なります。この新しい破棄関数 *OCIThreadKeyDestroy()* は、*key* を作成したときに、OCI THREAD によって取得されたリソースの終了に使用します。*OCIThreadKeyInit()* の *OCIThreadKeyDestFunc* コールバックは、キー VALUE のデストラクタです。キー自体の操作は行いません。

この関数は、キーの使用が終了したときにコールする必要があります。キー破棄関数をコールしないと、メモリー・リークが発生する可能性があります。

関連関数

[OCIThreadKeyGet\(\)](#)、[OCIThreadKeyInit\(\)](#)、[OCIThreadKeySet\(\)](#)

OCIThreadKeyGet()

用途

キーに対してコール側スレッドの現行の値を取得します。

構文

```
sword OCIThreadKeyGet ( dvoid          *hndl,
                       OCIError       *err,
                       OCIThreadKey   *key,
                       dvoid          **pValue );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

key (IN)

キーです。

pValue (IN/OUT)

スレッド固有のキー値を配置する場所です。

コメント

OCIThreadKeyInit() を使って作成されなかったキーに対して、この関数は使わないでください。

コール側スレッドからキーに値を割り当てられていない場合は、pValue によってポイントされている場所に NULL が配置されます。

関連関数

OCIThreadKeyDestroy()、*OCIThreadKeyInit()*、*OCIThreadKeySet()*

OCIThreadKeyInit()

用途

キーを作成します。

構文

```
sword OCIThreadKeyInit (dvoid          *hdl,
                       OCIError       *err,
                       OCIThreadKey   **key,
                       OCIThreadKeyDestFunc destFn );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

key (OUT)

新しいキーの作成を行う OCIThreadKey です。

destFn (IN)

キーのデストラクタです。NULL が許可されています。

コメント

このルーチンをコールするたびに、他のキーと異なる新しいキーの割当ておよび生成が行われます。この関数が正常に実行されると、割当ておよび初期化が行われたキーへのポインタが戻されます。このキーは、OCIThreadKeyGet() および OCIThreadKeySet() とともに使います。すべてのスレッドで、このキーの初期値は NULL です。

この関数を、key パラメータに同じ値を指定して、複数回コールしないでください。

destFn パラメータが NULL 以外の場合は、キーの値が NULL 以外のスレッドが終了するたびに、destFn によってポイントされているルーチンがコールされます。そのルーチンがコールされるときのパラメータは、1 つです。そのパラメータは、スレッド終了時のそのスレッドに対するキーの値です。キーにデストラクタ関数が必要ない場合は、destFn に NULL を渡します。

関連関数

[OCIThreadKeyDestroy\(\)](#)、[OCIThreadKeyGet\(\)](#)、[OCIThreadKeySet\(\)](#)

OCIThreadKeySet()

用途

キーに対してコール側スレッドの値を設定します。

構文

```
sword OCIThreadKeySet ( dvoid          *hdl,  
                       OCIError       *err,  
                       OCIThreadKey   *key,  
                       dvoid          *value );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、*OCIErrorGet()* のコールによって診断情報を取得できます。

key (IN/OUT)

キーです。

value (IN)

キーに設定するスレッド固有の値です。

コメント

OCIThreadKeyInit() を使って作成されなかったキーに対して、この関数は使わないでください。

関連関数

[OCIThreadKeyDestroy\(\)](#)、[OCIThreadKeyGet\(\)](#)、[OCIThreadKeyInit\(\)](#)

OCIThreadMutexAcquire()

用途

コールが行われたスレッドに対して mutex を取得します。

構文

```
sword OCIThreadMutexAcquire ( dvoid          *hndl,  
                              OCIError      *err,  
                              OCIThreadMutex *mutex );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、err に記録され、OCI_ERROR が戻されます。OCIErrorGet() のコールによって診断情報を取得できます。

mutex (IN/OUT)

取得する mutex です。

コメント

mutex が別のスレッドによって保持されている場合は、mutex が取得できるまでコール側スレッドはブロックされます。

初期化されていない mutex は取得しないでください。

スレッドによって既に保持されている mutex を取得するときに、この関数が使われるかどうかは定義されていません。

関連関数

[OCIThreadMutexDestroy\(\)](#)、[OCIThreadMutexInit\(\)](#)、[OCIThreadMutexRelease\(\)](#)

OCIThreadMutexDestroy()

用途

mutex の破棄および割当て解除を行います。

構文

```
sword OCIThreadMutexDestroy ( dvoid          *hdl,  
                              OCIError      *err,  
                              OCIThreadMutex **mutex );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、*OCIErrorGet()* のコールによって診断情報を取得できます。

mutex (IN/OUT)

破棄する mutex です。

コメント

各 mutex は、必要がなくなったら破棄する必要があります。

初期化されていない mutex、またはスレッドに現在保持されている mutex を破棄しないでください。mutex の破棄は、その mutex に対する他の操作と同時に行わないでください。また、mutex の破棄後は、その mutex を使わないでください。

関連関数

[OCIThreadMutexAcquire\(\)](#)、[OCIThreadMutexInit\(\)](#)、[OCIThreadMutexRelease\(\)](#)

OCIThreadMutexInit()

用途

mutex の割当ておよび初期化を行います。

構文

```
sword OCIThreadMutexInit ( dvoid          *hdl,  
                           OCIError      *err,  
                           OCIThreadMutex **mutex );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

mutex (OUT)

初期化する mutex です。

コメント

すべての mutex は、使う前に初期化する必要があります。

複数のスレッドから同時に 1 つの mutex を初期化しないでください。また、mutex は、破壊されるまで再度初期化しないでください (OCIThreadMutexDestroy() を参照)。

関連関数

[OCIThreadMutexDestroy\(\)](#)、[OCIThreadMutexAcquire\(\)](#)、[OCIThreadMutexRelease\(\)](#)

OCIThreadMutexRelease()

用途

mutex を解放します。

構文

```
sword OCIThreadMutexRelease ( dvoid          *hdl,  
                              OCIError      *err,  
                              OCIThreadMutex *mutex );
```

パラメータ

hdl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、*OCIErrorGet()* のコールによって診断情報を取得できます。

mutex (IN/OUT)

解放する mutex です。

コメント

mutex にブロックされているスレッドが存在する場合は、スレッドの1つが mutex を取得し、スレッドのブロックが解除されます。

初期化されていない mutex を解放しないでください。また、スレッドから、そのスレッドが保持していない mutex を解放しないでください。

関連関数

[OCIThreadMutexDestroy\(\)](#)、[OCIThreadMutexInit\(\)](#)、[OCIThreadMutexAcquire\(\)](#)

OCIThreadProcessInit()

用途

OCIThread プロセスの初期化を実行します。

構文

```
void OCIThreadProcessInit ( );
```

コメント

この関数をコールする必要があるかどうかは、OCI Thread の使用方法によって決まります。

シングルスレッド・アプリケーションでは、この関数はコールする必要はありません。コールする場合は、1 回目のコールは、その他の OCIThread 関数よりも先に行う必要があります。2 回目以降のコールには制限はありません。コールしても何も処理されません。

マルチスレッド・アプリケーションでは、この関数は必ずコールする必要があります。1 回目のコールは、その他の OCIThread コールより先に行う必要があります。1 回目のコールと同時に、他の OCIThread 関数コール（この関数コールも含まれます）を行うことはできません。

2 回目以降のコールには制限はありません。コールしても何も処理されません。

関連関数

[OCIThreadIdDestroy\(\)](#)、[OCIThreadIdGet\(\)](#)、[OCIThreadIdInit\(\)](#)、[OCIThreadIdNull\(\)](#)、[OCIThreadIdSame\(\)](#)、[OCIThreadIdSet\(\)](#)

OCIThreadTerm()

用途

OCIThread コンテキストを解放します。

構文

```
sword OCIThreadTerm ( dvoid      *hndl,  
                     OCIError   *err );
```

パラメータ

hndl (IN/OUT)

OCI 環境ハンドルまたはセッション・ハンドルです。

err (IN/OUT)

OCI エラー・ハンドルです。エラーが発生して OCI_ERROR が戻された場合は、エラーは err に記録され、OCIErrorGet() のコールによって診断情報を取得できます。

コメント

この関数は、OCIThreadInit() への各コールに対して 1 回だけコールする必要があります。

同時に OCIThreadTerm() コールを行ってください。OCIThreadTerm() のコール回数が OCIThreadInit() のコール回数と異なる場合は、処理は行われません。コール回数と同じときは、OCIThread 層を終了し、コンテキストに割り当てられていたメモリーを解放します。この操作が行われたら、そのコンテキストは再利用できません。OCIThreadInit() をコールして新しいコンテキストを取得する必要があります。

関連関数

[OCIThreadInit\(\)](#)

トランザクション関数

この項では、トランザクション関数について説明します。

表 15-9 OCI クイック・リファレンス

関数	用途	ページ
OCITransCommit()	サービス・コンテキスト上のトランザクションをコミットします。	15-203 ページ
OCITransDetach()	サービス・コンテキストからトランザクションを連結解除します。	15-206 ページ
OCITransForget()	準備したグローバル・トランザクションを放棄します。	15-207 ページ
OCITransPrepare()	グローバル・トランザクションをコミットのために準備します。	15-208 ページ
OCITransRollback()	トランザクションをロールバックします。	15-209 ページ
OCITransStart()	サービス・コンテキスト上のトランザクションを開始します。	15-210 ページ

OCITransCommit()

用途

指定のサービス・コンテキストに関連付けられているトランザクションをコミットします。

構文

```
sword OCITransCommit ( OCISvcCtx   *svchp,  
                       OCIError    *errhp,  
                       ub4          flags );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

flags (IN)

グローバル・トランザクションの1フェーズ・コミットを最適化するためのフラグです。

トランザクションが分散型ではない場合、flags パラメータは無視され、その値として OCI_DEFAULT を渡すことができます。2フェーズ・コミットでは、グローバル・トランザクションを管理している OCI アプリケーションは、flags に対して値 OCI_TRANS_TWOPHASE を渡す必要があります。デフォルトは1フェーズ・コミットです。

コメント

サービス・コンテキストに現在関連付けられているトランザクションをコミットします。トランザクションがサーバーによるコミットが不可能なグローバル・トランザクションである場合、このコールは、トランザクションの状態をデータベースから検索し、エラー・ハンドルを使用してユーザーに戻します。

アプリケーションが複数のトランザクションを定義している場合、このコールは、サービス・コンテキストに現在関連付けられているトランザクションを操作します。アプリケーションがデータベースの変更時に作成される暗黙ローカル・トランザクションだけを操作している場合は、その暗黙トランザクションがコミットされます。

アプリケーションをオブジェクト・モードで実行している場合、このトランザクションに対してオブジェクト・キャッシュで変更または更新されたオブジェクトもフラッシュされ、コミットされます。

正常な状況では、*OCITransCommit()* は、トランザクションがコミットされたかロールバックされたかを示すステータスを戻します。グローバル・トランザクションでは、トランザクションがインダウトの状態、つまり、コミットも異常終了もされていない状態の場合もあり


```
OCITransCommit(svchp, errhp, (ub4) 0);

/* update scott.emp empno=7902, increment salary again, but rollback */
OCIStmtExecute(svchp, stmthp, errhp, 1, 0, 0, 0, 0);
OCITransRollback(svchp, errhp, (ub4) 0);
}
```

関連関数

[OCITransRollback\(\)](#)

OCITransDetach()

用途

トランザクションの連結を解除します。

構文

```
sword OCITransDetach ( OCISvcCtx   *svchp,  
                       OCIError    *errhp,  
                       ub4          flags );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

flags (IN)

このパラメータ用に OCI_DEFAULT の値を渡す必要があります。

コメント

サービス・コンテキスト・ハンドルからグローバル・トランザクションを連結解除します。このコールが終了した時点で、サービス・コンテキスト・ハンドルに現在アタッチされているトランザクションが非活動状態になります。このトランザクションは、後で *OCITransStart()* をコールしながら OCI_TRANS_RESUME のフラグ値を指定するときに再開することができます。

トランザクションの連結が解除されると、トランザクションの開始時に *OCITransStart()* のタイムアウト・パラメータに指定された値を使って、サーバーの PMON プロセスによって削除されるまでブランチを非活動状態にしておける時間が判断されます。

注意: トランザクションが同一の権限を持っている場合、トランザクションはその連結を解除したプロセス以外のプロセスによっても再開することができます。トランザクションが実際に開始される前にこの関数がコールされると、この関数は空命令になります。

OCITransDetach() の使用方法を説明するコード例については、[OCITransStart\(\)](#) の説明を参照してください。

関連関数

[OCITransStart\(\)](#)

OCITransForget()

用途

完了したグローバル・トランザクションをサーバーに放棄させます。

構文

```
sword OCITransForget ( OCISvcCtx      *svchp,  
                      OCIError      *errhp,  
                      ub4            flags );
```

パラメータ

svchp (IN)

トランザクションが常駐するサービス・コンテキスト・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

flags (IN)

このパラメータ用に OCI_DEFAULT を渡す必要があります。

コメント

完了したグローバル・トランザクションを放棄します。サーバーにより、システムのペンディング状態のトランザクション・テーブルからトランザクションの状態が削除されます。

放棄されるトランザクションの XID をトランザクション・ハンドルの属性として設定します (OCI_ATTR_XID)。

関連関数

[OCITransCommit\(\)](#)、[OCITransRollback\(\)](#)

OCITransPrepare()

用途

指定のトランザクションをコミットできるように準備します。

構文

```
sword OCITransPrepare ( OCISvcCtx   *svchp,  
                        OCIError    *errhp,  
                        ub4          flags );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

flags (IN)

このパラメータ用に OCI_DEFAULT を渡す必要があります。

コメント

指定のグローバル・トランザクションをコミットできるように準備します。

このコールは、グローバル・トランザクションに対してだけ有効です。

このコールは、トランザクションが変更されていない場合は OCI_SUCCESS_WITH_INFO を戻します。エラー・ハンドルは、トランザクションが読取り専用であることを示します。flags パラメータは、現在使用されていません。

関連関数

[OCITransCommit\(\)](#)、[OCITransForget\(\)](#)

OCITransRollback()

用途

現トランザクションをロールバックします。

構文

```
sword OCITransRollback ( dvoid          *svchp,  
                        OCIError      *errhp,  
                        ub4            flags );
```

パラメータ

svchp (IN)

サービス・コンテキスト・ハンドルです。サービス・コンテキスト・ハンドル内に現在設定されているトランザクションがロールバックされます。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

flags (IN)

このパラメータ用に OCI_DEFAULT の値を渡す必要があります。

コメント

現行トランザクション (最後の *OCITransCommit()* 以降、または *OCISessionBegin()* 以降に実行された一連の文として定義される) がロールバックされます。

アプリケーションがオブジェクト・モードで実行されている場合は、このトランザクションで修正または更新されたオブジェクト・キャッシュ内のオブジェクトも同様にロールバックされます。

現在アクティブでないグローバル・トランザクションをロールバックしようとすると、エラーになります。

例

OCITransRollback() の使用方法を説明するコード例については、[OCITransCommit\(\)](#) の説明を参照してください。

関連関数

[OCITransCommit\(\)](#)

OCITransStart()

用途

トランザクションの開始を設定します。

構文

```
sword OCITransStart ( OCISvcCtx      *svchp,  
                     OCIError      *errhp,  
                     uword         timeout,  
                     ub4           flags );
```

パラメータ

svchp (IN/OUT)

サービス・コンテキスト・ハンドルです。flags パラメータに新規トランザクションの開始が指定されている場合は、このコールの終了時点で、サービス・コンテキスト・ハンドル内のトランザクション・コンテキストが初期化されます。

errhp (IN/OUT)

OCI エラー・ハンドルです。エラーがある場合は、err に記録され、OCI_ERROR が戻されます。診断情報は、OCIErrorGet() をコールして取得できます。

timeout (IN)

OCI_TRANS_RESUME が指定されたときに、トランザクションが再開できるようになるまで待機する時間 (秒) です。OCI_TRANS_NEW が指定された時は、非活動状態が timeout パラメータの指示する秒数続いたトランザクションは、システムによって自動的に中止されます。トランザクションは、(OCITransDetach() により) 連結解除されたときから OCITransStart() によって再開されるまでの間は非活動状態にあります。

flags (IN)

新規トランザクションが開始中か、または既存のトランザクションが再開中であるかを指定します。直列可能または読取り専用ステータスも指定します。複数の値を指定できます。デフォルトでは、読取りトランザクションまたは書込みトランザクションが開始されます。フラグ値には次のものがあります。

- OCI_TRANS_NEW - 新規トランザクション・ブランチを開始します。デフォルトでは、緊密に結合された移行可能なブランチが開始されます。
- OCI_TRANS_TIGHT - 緊密に結合されたブランチを明示的に指定します。
- OCI_TRANS_LOOSE - 疎結合ブランチを指定します。
- OCI_TRANS_RESUME - 既存のトランザクション・ブランチを再開します。
- OCI_TRANS_READONLY - 読取り専用トランザクションを開始します。


```
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                52, (dvoid **) &tmp);

OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp1, OCI_HTYPE_STMT, 0, 0);
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp2, OCI_HTYPE_STMT, 0, 0);

OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)srvhp, 0,
           OCI_ATTR_SERVER, errhp);

/* set the external name and internal name in server handle */
OCIAttrSet((dvoid *)srvhp, OCI_HTYPE_SERVER, (dvoid *) "demo", 0,
           OCI_ATTR_EXTERNAL_NAME, errhp);
OCIAttrSet((dvoid *)srvhp, OCI_HTYPE_SERVER, (dvoid *) "txn demo", 0,
           OCI_ATTR_INTERNAL_NAME, errhp);

/* allocate a user context handle */
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);

OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS, 0);

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
           (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* allocate transaction handle 1 and set it in the service handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&txnhp1, OCI_HTYPE_TRANS, 0, 0);
OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp1, 0,
           OCI_ATTR_TRANS, errhp);

/* start a transaction with global transaction id = [1000, 123, 1] */
gxid.formatID = 1000; /* format id = 1000 */
gxid.gtrid_length = 3; /* gtrid = 123 */
gxid.data[0] = 1; gxid.data[1] = 2; gxid.data[2] = 3;
gxid.bqual_length = 1; /* bqual = 1 */
gxid.data[3] = 1;

OCIAttrSet((dvoid *)txnhp1, OCI_HTYPE_TRANS, (dvoid *)&gxid, sizeof(XID),
           OCI_ATTR_XID, errhp);

/* start global transaction 1 with 60 second time to live when detached */
OCITransStart (svchp, errhp, 60, OCI_TRANS_NEW);
```

```
/* update scott.emp empno=7902, increment salary */
sprintf((char *)sqlstmt, "UPDATE EMP SET SAL = SAL + 1 WHERE EMPNO = 7902");
OCIStmtPrepare(stmthp1, errhp, sqlstmt, strlen(sqlstmt), OCI_NTV_SYNTAX, 0);
OCIStmtExecute(svchp, stmthp1, errhp, 1, 0, 0, 0, 0);

/* detach the transaction */
OCITransDetach(svchp, errhp, 0);

/* allocate transaction handle 2 and set it in the service handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&txnhp2, OCI_HTYPE_TRANS, 0, 0);
OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp2, 0,
           OCI_ATTR_TRANS, errhp);

/* start a transaction with global transaction id = [1000, 124, 1] */
gxid.formatID = 1000; /* format id = 1000 */
gxid.gtrid_length = 3; /* gtrid = 124 */
gxid.data[0] = 1; gxid.data[1] = 2; gxid.data[2] = 4;
gxid.bqual_length = 1; /* bqual = 1 */
gxid.data[3] = 1;

OCIAttrSet((dvoid *)txnhp2, OCI_HTYPE_TRANS, (dvoid *)&gxid, sizeof(XID),
           OCI_ATTR_XID, errhp);

/* start global transaction 2 with 90 second time to live when detached */
OCITransStart(svchp, errhp, 90, OCI_TRANS_NEW);

/* update scott.emp empno=7934, increment salary */
sprintf((char *)sqlstmt, "UPDATE EMP SET SAL = SAL + 1 WHERE EMPNO = 7934");
OCIStmtPrepare(stmthp2, errhp, sqlstmt, strlen(sqlstmt), OCI_NTV_SYNTAX, 0);
OCIStmtExecute(svchp, stmthp2, errhp, 1, 0, 0, 0, 0);

/* detach the transaction */
OCITransDetach(svchp, errhp, 0);

/* Resume transaction 1, increment salary and commit it */
/* Set transaction handle 1 into the service handle */
OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp1, 0,
           OCI_ATTR_TRANS, errhp);

/* attach to transaction 1, wait for 10 seconds if the transaction is busy */
/* The wait is clearly not required in this example because no other */
/* process/thread is using the transaction. It is only for illustration */
OCITransStart(svchp, errhp, 10, OCI_TRANS_RESUME);
OCIStmtExecute(svchp, stmthp1, errhp, 1, 0, 0, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* attach to transaction 2 and commit it */
```

```
/* set transaction handle2 into the service handle */
OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp2, 0,
           OCI_ATTR_TRANS, errhp);
OCITransCommit(svchp, errhp, (ub4) 0);
}
```

例 2

この例では、同一のトランザクションを共有する複数のブランチ上で動作するシングル・セッションについて説明します。

```
int main()
{
    OCIEnv *envhp;
    OCIServer *srvhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCISession *usrhp;
    OCIStmt *stmthp;
    OCITrans *txnhp1, *txnhp2;
    dvoid *tmp;
    XID gxid;
    text sqlstmt[128];

    OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0, (dvoid * (*)()) 0,
                 (dvoid * (*)()) 0, (void (*)()) 0 );

    OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                   0, (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);

    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT, 0, 0);

    OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)srvhp, 0,
              OCI_ATTR_SERVER, errhp);

    /* set the external name and internal name in server handle */
```

```
OCIAttrSet((dvoid *)srvhp, OCI_HTYPE_SERVER, (dvoid *) "demo", 0,
           OCI_ATTR_EXTERNAL_NAME, errhp);
OCIAttrSet((dvoid *)srvhp, OCI_HTYPE_SERVER, (dvoid *) "txn demo2", 0,
           OCI_ATTR_INTERNAL_NAME, errhp);

/* allocate a user context handle */
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);

OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS, 0);

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
           (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* allocate transaction handle 1 and set it in the service handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&txnhp1, OCI_HTYPE_TRANS, 0, 0);
OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp1, 0,
           OCI_ATTR_TRANS, errhp);

/* start a transaction with global transaction id = [1000, 123, 1] */
gxid.formatID = 1000; /* format id = 1000 */
gxid.gtrid_length = 3; /* gtrid = 123 */
gxid.data[0] = 1; gxid.data[1] = 2; gxid.data[2] = 3;
gxid.bqual_length = 1; /* bqual = 1 */
gxid.data[3] = 1;

OCIAttrSet((dvoid *)txnhp1, OCI_HTYPE_TRANS, (dvoid *)&gxid, sizeof(XID),
           OCI_ATTR_XID, errhp);

/* start global transaction 1 with 60 second time to live when detached */
OCITransStart (svchp, errhp, 60, OCI_TRANS_NEW);

/* update scott.emp empno=7902, increment salary */
sprintf((char *)sqlstmt, "UPDATE EMP SET SAL = SAL + 1 WHERE EMPNO = 7902");
OCIStmtPrepare(stmthp, errhp, sqlstmt, strlen(sqlstmt), OCI_NIV_SYNTAX, 0);
OCIStmtExecute(svchp, stmthp, errhp, 1, 0, 0, 0, 0);

/* detach the transaction */
OCITransDetach(svchp, errhp, 0);

/* allocate transaction handle 2 and set it in the service handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&txnhp2, OCI_HTYPE_TRANS, 0, 0);
```

```
OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp2, 0,
           OCI_ATTR_TRANS, errhp);

/* start a transaction with global transaction id = [1000, 123, 2] */
/* The global transaction will be tightly coupled with earlier transaction */
/* There is not much practical value in doing this but the example */
/* illustrates the use of tightly-coupled transaction branches */
/* In a practical case the second transaction that tightly couples with */
/* the first can be executed from a different process/thread */

gxid.formatID = 1000; /* format id = 1000 */
gxid.gtrid_length = 3; /* gtrid = 123 */
gxid.data[0] = 1; gxid.data[1] = 2; gxid.data[2] = 3;
gxid.bqual_length = 1; /* bqual = 2 */
gxid.data[3] = 2;

OCIAttrSet((dvoid *)txnhp2, OCI_HTYPE_TRANS, (dvoid *)&gxid, sizeof(XID),
           OCI_ATTR_XID, errhp);

/* start global transaction 2 with 90 second time to live when detached */
OCITransStart(svchp, errhp, 90, OCI_TRANS_NEW);

/* update scott.emp empno=7902, increment salary */
/* This is possible even if the earlier transaction has locked this row */
/* because the two global transactions are tightly coupled */
OCIStmtExecute(svchp, stmthp, errhp, 1, 0, 0, 0, 0);

/* detach the transaction */
OCITransDetach(svchp, errhp, 0);

/* Resume transaction 1 and prepare it. This will return */
/* OCI_SUCCESS_WITH_INFO because all branches except the last branch */
/* are treated as read-only transactions for tightly-coupled transactions */

OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp1, 0,
           OCI_ATTR_TRANS, errhp);
if (OCITransPrepare(svchp, errhp, (ub4) 0) == OCI_SUCCESS_WITH_INFO)
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("OCITransPrepare - %s\n", errbuf);
}
```

```
/* attach to transaction 2 and commit it */
/* set transaction handle2 into the service handle */
OCIAttrSet((dvoid *)svchp, OCI_HTYPE_SVCCTX, (dvoid *)txnhp2, 0,
           OCI_ATTR_TRANS, errhp);
OCITransCommit(svchp, errhp, (ub4) 0);
}
```

関連関数

[*OCITransDetach\(\)*](#)

その他の関数

この項では、その他の OCI 関数について説明します。

表 15-10 OCI クイック・リファレンス

関数	用途	ページ
OCIBreak()	即時非同期ブレークを実行します。	15-219 ページ
OCIErrorGet()	エラーを戻します。	15-220 ページ
OCILdaToSvcCtx()	Lda_Def をサービス・コンテキスト・ハンドルに切り替えます。	15-222 ページ
OCIPasswordChange()	パスワードを変更します。	15-223 ページ
OCIReset()	OCIBreak() の後にコールされて非同期操作およびプロトコルをリセットします。	15-225 ページ
OCIServerVersion()	Oracle バージョン文字列を取得します。	15-226 ページ
OCISvcCtxToLda()	サービス・コンテキスト・ハンドルを Lda_Def に切り替えます。	15-227 ページ
OCIUserCallbackGet()	ハンドルに対して登録されたコールバックを識別します。	15-228 ページ
OCIUserCallbackRegister()	ユーザー作成コールバック関数を登録します。	15-230 ページ

OCIBreak()

用途

このコールは、サーバーに関連付けられて現在実行中である OCI 関数の即時（非同期）終了を実行します。

構文

```
sword OCIBreak ( dvoid      *hdlp,  
                 OCIError  *errhp);
```

パラメータ

hdlp (IN/OUT)

サービス・コンテキスト・ハンドルまたはサーバー・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

コメント

このコールは、サーバーに関連付けられて現在実行中である OCI 関数の即時（非同期）終了を実行します。これは通常、サーバーで長時間実行されている処理中の OCI コールを停止するために使用します。

このコールは、終了させる関数を識別するためのパラメータとして、サービス・コンテキスト・ハンドルまたはサーバー・コンテキスト・ハンドルを取ることができます。

関連関数

[OCIReset\(\)](#)

OCIErrorGet()

用途

用意されたバッファ内のエラー・メッセージと Oracle エラーを戻します。

構文

```
sword OCIErrorGet ( dvoid      *hndlp,  
                   ub4        recordno,  
                   text       *sqlstate,  
                   sb4        *errcodep,  
                   text       *bufp,  
                   ub4        bufsiz,  
                   ub4        type );
```

パラメータ

hndlp (IN)

エラー・ハンドルまたは環境ハンドル (*OCIEnvInit()* および *OCIHandleAlloc()*) に対するエラー用) です。エラー・ハンドルの場合がほとんどです。

recordno (IN)

アプリケーションが情報を検索する検索先の状態レコードを示します。1 から始まります。

sqlstate (OUT)

リリース 8.0 ではサポートされていません。

errcodep (OUT)

Oracle エラーが戻されます。

bufp (OUT)

エラー・メッセージ・テキストが戻されます。

bufsiz (IN)

エラー・メッセージの取得に使用するバッファのサイズです。

type (IN)

ハンドルのタイプ (*OCI_HTYPE_ERR* または *OCI_HTYPE_ENV*) です。

コメント

用意されたバッファ内のエラー・メッセージと Oracle エラー・コードを戻します。SQL ステータスはサポートしていません。この関数は、1 つのエラーに対して複数の診断レコードがある場合は、複数回コールできます。

エラー・ハンドルは、最初は *OCIHandleAlloc()* のコールで割り当てられます。

例

次のサンプル・コードは、エラー処理ルーチンで *OCIErrorGet()* を使用方法の例です。このルーチンは、OCI 関数から戻されるステータス・コードの型を印刷し、エラーが発生した場合は、*OCIErrorGet()* を使用して、印刷するメッセージのテキストを取り出します。

```
static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{ text errbuf[512];
  ub4 buflen;
  ub4 errcode;
  switch (status)
  { case OCI_SUCCESS:
    break;
    case OCI_SUCCESS_WITH_INFO:
    printf("ErrorOCI_SUCCESS_WITH_INFO\n");
    break;
    case OCI_NEED_DATA:
    printf("ErrorOCI_NEED_DATA\n");
    break;
    case OCI_NO_DATA:
    printf("ErrorOCI_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("ErrorOCI_INVALID_HANDLE\n");
    break;
    case OCI_STILL_EXECUTING:
    printf("ErrorOCI_STILL_EXECUTE\n");
    break;
    case OCI_CONTINUE:
    printf("ErrorOCI_CONTINUE\n");
    break;
    default:
    break;
  }
}
```

関連関数

[*OCIHandleAlloc\(\)*](#)

OCILdaToSvcCtx()

用途

V7 の Lda_Def を V8 のサービス・コンテキスト・ハンドルに変換します。

構文

```
sword OCILdaToSvcCtx ( OCISvcCtx  **svchpp,  
                      OCIError   *errhp,  
                      Lda_Def    *ldap );
```

パラメータ

svchpp (IN/OUT)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

ldap (IN/OUT)

OCISvcCtxToLda() によりこのサービス・コンテキストから戻された Oracle7 ログイン・データ領域 (LDA) です。

コメント

Oracle7 Lda_Def を Oracle リリース 8 サービス・コンテキスト・ハンドルに変換します。変換されたサービス・コンテキスト・ハンドルを *OCISvcCtxToLda()* 関数に渡すと、このコールの逆の変換ができます。

サービス・コンテキストが Lda_Def に変換されている場合は、Oracle7 のコールしか使用できません。Lda_Def をサービス・コンテキストにリセットせずに OracleOCI リリース 8 コールを行っても実行されません。

サーバー・ハンドルまたはサービス・コンテキスト・ハンドルの OCI_ATTR_IN_V8_MODE 属性により、現行のモードが Oracle リリース 7 モードか Oracle リリース 8 モードかをアプリケーションで判断できます。詳細は、[付録 A の「ハンドルおよび記述子の属性」](#)を参照してください。

関連関数

[OCISvcCtxToLda\(\)](#)

OCIPasswordChange()

用途

このコールは、アカウントのパスワードの変更を許可します。

構文

```
sword OCIPasswordChange ( OCISvcCtx      *svchp,
                          OCIError      *errhp,
                          CONST text    *user_name,
                          ub4           usernm_len,
                          CONST text    *opasswd,
                          ub4           opasswd_len,
                          CONST text    *npasswd,
                          sb4           npasswd_len,
                          ub4           mode );
```

パラメータ

svchp (IN/OUT)

サービス・コンテキストへのハンドルです。サービス・コンテキスト・ハンドルは初期化済みであり、サーバー・コンテキスト・ハンドルがそれに関連付けられていなければなりません。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

user_name (IN)

ユーザー名を指定します。これは文字列を指し、その長さは *usernmm_len* に指定します。このパラメータは、サービス・コンテキストがユーザー・セッション・ハンドルを使用して初期化されている場合は、NULL でなければなりません。

usernmm_len (IN)

user_name で指定されたユーザー名文字列の長さです。ユーザー名文字列を有効にするには、*usernmm_len* は 0 (ゼロ) 以外でなければなりません。

opasswd (IN)

ユーザーの旧パスワードを指定します。これは文字列を示し、その長さは *opasswd_len* に指定されます。

opasswd_len (IN)

opasswd で指定された旧パスワード文字列の長さです。パスワード文字列を有効にするには、*opasswd_len* は 0 (ゼロ) 以外でなければなりません。

npasswd (IN)

ユーザーの新しいパスワードを指定します。これは文字列を示し、その長さは *npasswd_len* に指定され、パスワード文字列を有効にするには 0 (ゼロ) 以外でなければなりません。パスワードの複雑さ検証ルーチンがユーザーのプロファイルに指定されていて、それを使って新しいパスワードの複雑さが検証されるようになっている場合には、新しいパスワードは検証関数の複雑さ要件に合致しなければなりません。

npasswd_len (IN)

npasswd で指定された新しいパスワード文字列の長さです。パスワード文字列を有効にするには、*npasswd_len* は 0 (ゼロ) 以外でなければなりません。

mode (IN)

OCI_DEFAULT または OCI_AUTH、あるいはその両方です。OCI_AUTH を設定すると、次のことが発生します。

- ユーザー・セッション・コンテキストが作成されていない場合は、このコールで作成され、パスワードが変更されます。コールが終了しても、ユーザー・セッション・コンテキストは消去されません。ユーザーはログインされたままです。

ユーザー・セッション・コンテキストが既に作成されている場合は、パスワードの変更だけが実行され、このフラグによるセッションへの影響はありません。ユーザーはログインされたままです。

コメント

このコールは、アカウントのパスワードの変更を許可します。このコールは *OCISessionBegin()* に類似していますが、次の点が異なります。

- ユーザー・セッションが既に確立されている場合は、旧パスワードを使用しているアカウントを認証した後、パスワードを新規パスワードに変更します。
- ユーザー・セッションが確立されていない場合は、ユーザー・セッションを確立して旧パスワードを使用しているアカウントを認証した後、パスワードを新規パスワードに変更します。

このコールは、アカウントのパスワードが期限切れになり、*OCISessionBegin()* がエラー (ORA-28001) またはパスワードが期限切れであることを示す警告を戻したときに役に立ちます。

関連関数

[OCISessionBegin\(\)](#)

OCIReset()

用途

中断された非同期操作およびプロトコルをリセットします。非ブロッキング操作の処理中に OCIBreak コールがパブリッシュされた場合に、コールする必要があります。

構文

```
sword OCIReset ( dvoid      *hdlp,  
                 OCIError  *errhp);
```

コメント

このコールは、非ブロッキング・モードでだけコールされます。中断された非同期操作およびプロトコルをリセットします。非ブロッキング操作の処理中に OCIBreak コールがパブリッシュされた場合に、コールする必要があります。

パラメータ

hdlp (IN)

サービス・コンテキスト・ハンドルまたはサーバー・コンテキスト・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

関連関数

[OCIBreak\(\)](#)

OCIServerVersion()

用途

Oracle サーバーのバージョン文字列を戻します。

構文

```
sword OCIServerVersion ( dvoid          *hndlp,  
                        OCIError      *errhp,  
                        text           *bufp,  
                        ub4            bufksz,  
                        ub1            hndltype );
```

パラメータ

hndlp (IN)

サービス・コンテキスト・ハンドルまたはサーバー・コンテキスト・ハンドルです。

errhp (IN)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

bufp (IN)

バージョン情報の復帰先バッファです。

bufksz (IN)

バッファの長さです。

hndltype (IN)

関数に渡されたハンドル型です。

コメント

このコールは、Oracle サーバーのバージョン文字列を戻します。たとえば、アプリケーションが 7.3.2 サーバー上で実行されている場合は、バージョン文字列として次のものが戻ります。

```
Oracle7 Server Release 7.3.2.0.0 Production Release  
PL/SQL Release 2.3.2.0.0 Production  
CORE Version 3.5.2.0.0 Production  
TNS for SEQUENT DYNLX/ptx: Version 2.3.2.0.0 Production  
NLSRTL Version 3.2.2.0.0 Production
```

関連関数

[OCIErrorGet\(\)](#)

OCISvcCtxToLda()

用途

V8 サービス・コンテキスト・ハンドルを V7 Lda_Def に切り替えます。

構文

```
sword OCISvcCtxToLda ( OCISvcCtx   *srvhp,  
                      OCIError    *errhp,  
                      Lda_Def     *ldap );
```

パラメータ

svchp (IN/OUT)

サービス・コンテキスト・ハンドルです。

errhp (IN/OUT)

エラー時の診断情報のために *OCIErrorGet()* に渡せるエラー・ハンドルです。

ldap (IN/OUT)

このコールによって初期化される Oracle7 型 OCI コールのためのログイン・データ領域 (LDA)。

コメント

Oracle OCI リリース 8 サービス・コンテキスト・ハンドルと Oracle7Lda_Def とを切り替えます。

この関数は、サービス・コンテキストが正常に初期化された後でだけコールできます。

サービス・コンテキストは、Lda_Def に変換した後、リリース 7.x の OCI コール (例: *obindps()*、*ofen()*) で使用できます。

注意: 同じサーバー・ハンドルを共有する複数のサービス・コンテキストがある場合、Oracle7 モードは常に 1 つのみに可能です。

変換された Lda_Def を *OCILdaToSvcCtx()* 関数に渡すと、このコールの逆の変換ができます。

サーバー・ハンドルまたはサービス・コンテキスト・ハンドルの OCI_ATTR_IN_V8_MODE 属性により、現行のモードが Oracle リリース 7 モードか Oracle リリース 8 モードかをアプリケーションで判断できます。詳細は、[付録 A の「ハンドルおよび記述子の属性」](#)を参照してください。

関連関数

[OCILdaToSvcCtx\(\)](#)

OCIUserCallbackGet()

用途

ハンドルに対して登録されたコールバックを決定します。

構文

```
sword OCIUserCallbackGet ( dvoid   *hndlp,
                          ub4     type,
                          dvoid   *ehndlp,
                          ub4     fcode,
                          ub1     when
                          OCIUserCallback (*callbackp)
                          (/*_
                            dvoid *ctxp,
                            dvoid *hndlp,
                            ub4 type,
                            ub4 fcode,
                            ub1 when,
                            sword returnCode,
                            ub4 *errnop,
                            va_list arglist
                            _*/),
                          dvoid   **ctxpp );
```

パラメータ

hndlp (IN)

type パラメータによって指定された型のハンドルです。

type (IN)

ハンドル・タイプです。次のハンドル・タイプが有効です。

- OCI_HTYPE_ENV - コールバックは、環境ハンドル上で生成された *fcode* によって指定される関数の、すべてのコールに対して登録されます。

ehndlp (IN)

OCI エラー・ハンドルまたは環境ハンドルです。エラーがある場合は、*ehndlp* に記録され、OCI_ERROR が戻されます。OCIErrorGet() のコールによって診断情報を取得できます。

fcode (IN)

OCI 関数の一意の関数コードです。これらは、15-234 ページの表 15-11 の「OCI 関数コード」に示されています。

when (IN)

コールバックがいつ起動されるかを定義します。次のモードが有効です。

- OCI_CBTYPE_ENTRY - コールバックは、OCI 関数の開始時に起動します。
- OCI_CBTYPE_EXIT - コールバックは、OCI 関数の終了時前に起動します。

callbackp (OUT)

コールバック関数ポインタへのポインタです。 *fcode*、*when* および *hndlp* の値に対して現在登録されている関数を戻します。コールバックが登録されていない場合は、戻り値は NULL です。 *callbackp* のパラメータについては、15-230 ページの [OCIUserCallbackRegister\(\)](#) の説明を参照してください。

ctxpp (OUT)

現在登録されているコールバックに対して戻されるコンテキストへのポインタです。

コメント

この関数は、特定のハンドルに対して登録されているコールバックを検出します。

コールバック関数の使用制限については、9-15 ページの「[コールバック関数の制限](#)」を参照してください。

関連関数

[OCIUserCallbackRegister\(\)](#)

OCIUserCallbackRegister()

用途

ユーザー作成コールバック関数を登録します。

構文

```

sword OCIUserCallbackRegister ( dvoid   *hndlp,
                               ub4     type,
                               dvoid   *ehndlp,
                               OCIUserCallback (callback)
                                   (/*_
                                   dvoid *ctxp,
                                   dvoid *hndlp,
                                   ub4  type,
                                   ub4  fcode,
                                   ub1  when,
                                   sword returnCode,
                                   ub4  *ermop,
                                   va_list arglist
                                   _*/),
                               dvoid   *ctxp,
                               ub4     fcode,
                               ub1     when );

```

パラメータ

hndlp (IN)

type パラメータによって指定された型のハンドルです。

type (IN)

ハンドル・タイプです。次のハンドル・タイプが有効です。

- OCI_HTYPE_ENV - コールバックは、環境ハンドル上で生成された *fcode* によって指定される関数の、すべてのコールに対して登録されます。

ehndlp (IN)

OCI エラー・ハンドルまたは環境ハンドルです。エラーがある場合は、*ehndlp* に記録され、OCI_ERROR が戻されます。OCIErrorGet() のコールによって診断情報を取得できます。OCIEnvCallback では、エラー・ハンドルを使うことができないので、環境ハンドルは *ehndlp* として渡されます。

callback (IN)

コールバック関数ポインタです。OCIUserCallback 関数プロトタイプの変数引数のリストは、OCI 関数に渡されるパラメータです。OCIUserCallback の typedef については、後で説明します。

最初のコールバックで OCI_CONTINUE 以外が戻された場合は、OCI コードはスキップされます。最後のコールバックが存在する場合は、制御が最後のコールバックに移ります。

最後のコールバックから OCI_CONTINUE 以外が戻された場合は、OCI 関数から戻り値が戻されます。OCI_CONTINUE が戻された場合は、OCI コードまたは最初のコールバック（最初のコールバックが OCI_CONTINUE を戻さずに OCI コードを迂回した場合）の戻り値がコールから戻されます。

コールバックに対して NULL 値が渡された場合は、when 値および指定されたハンドルのコールバックが削除されます。

ctxp (IN)

コールバックのコンテキスト・ポインタです。

fcode (IN)

OCI 関数の一意の関数コードです。これらは、15-234 ページの表 15-11 の「OCI 関数コード」に示されています。

when (IN)

コールバックがいつ起動されるかを定義します。次のモードが有効です。

- OCI_CBTYPPE_ENTRY - コールバックは、OCI 関数の開始時に起動します。
- OCI_CBTYPPE_EXIT - コールバックは、OCI 関数の終了時前に起動します。

コメント

この関数は、ユーザー作成のコールバック関数の登録に使用します。OCI では、OCI 環境にユーザー作成のコールバック関数を登録できます。このコールバックにより、アプリケーションで次の処理を行うことができます。

1. デバッグおよびパフォーマンス測定のために OCI コールをトレースします。
2. 選択した OCI コールの後に、前処理または後処理を追加実行します。
3. 指定された関数のコードを、外部キー・データ・ソースで実行するコードに置き換えます。

OCI では、最初のコールバックおよび最後のコールバックの 2 種類のコールバックがサポートされています。

最初のコールバックは、プログラムが OCI 関数に入るときに実行されます。最初のコールバックから値 OCI_CONTINUE が戻された場合は、通常の OCI 固有のコードが実行されません。このコールバックから OCI_CONTINUE 以外が戻された場合は、OCI コードは実行されません。

OCI 関数が正常に実行された後またはコールバックから OCI_CONTINUE 以外が戻された後は、プログラムの制御は最後のコールバック（登録されている場合）に移ります。

注意: 最後のコールバックが登録されていないときに、最初のコールバックから OCI_CONTINUE 以外が戻された場合は、最初のコールバックからのリターン・コードは、

関連付けられた OCI コールから戻されます。同様に、最後のコールバックから OCI_CONTINUE 以外が戻された場合は、リターン・コードは OCI コールから戻されます。

ハンドルに対して登録されているコールバックを検出するには、*OCIUserCallbackGet()* を使います。このコールのプロトタイプを次に示します。

OCIUserCallback typedef のプロトタイプは次のとおりです。

```
typedef sword (*OCIUserCallback)
    (dvoid *ctxp,
     dvoid *hdlp,
     ub4 type,
     ub4 fcode,
     ub1 when,
     sword returnCode,
     ub4 *errnop,
     va_list arglist
    );
```

OCIUserCallback 関数プロトタイプへのパラメータは次のとおりです。

ctxp (IN)

登録コールバック関数内で *ctxp* として渡されるコンテキストです。

hdlp(IN)

type パラメータで指定された型のハンドルです。コールバックを起動するためのハンドルです。OCI_HTYPE_ENV 型以外は使わないので、環境ハンドル *env* はここに渡されます。

type (IN)

hdlp に対して登録された型です。次のハンドル・タイプが有効です。

- OCI_HTYPE_ENV - コールバックは、環境ハンドル上で生成された *fcode* によって指定される関数の、すべてのコールに対して登録されます。

fcode (IN)

OCI コールの関数コードです。関数コードは、15-234 ページの表 15-11 の「OCI 関数コード」のリストを参照してください。コールバックは、表 15-11 の「OCI 関数コード」に示されている OCI コールに対してだけ登録できます。

when (IN)

コールバックの *when* 値です。

returnCode (IN)

直前のコールバックまたは OCI コードからのリターン・コードです。最初のコールバックの場合は、常に OCI_SUCCESS が渡されます。最後のコールバックの場合は、OCI コードまたは最初のコールバック（最初のコールバックが OCI_CONTINUE を戻さなかった場合）からのリターン・コードが渡されます。

errnop (IN/OUT)

最初のコールバックがコールされた場合は、**errnop* の入力値は 0 です。最初のコールバックから OCI_CONTINUE 以外が戻された場合は、このコールバックで **errnop* にエラー番号を設定する必要があります。この値は、OCI コール内に渡されるエラー・ハンドルで設定されます。

最後のコールバックの場合は、**errnop* の入力値はエラー・ハンドル内のエラー番号の値です。このため、最後のコールバックから OCI_CONTINUE が戻されなかった場合は、最初のコールバックからの **errnop* の出力値はエラー・ハンドル内の値で、この値はここで最後のコールバックに渡されます。一方、最初のコールバックから OCI_CONTINUE が戻され、通常の OCI コードが実行された場合は、その OCI コールの結果発生したエラー・ハンドル内の値はここで渡されます。

最初のコールバックと同様に、最後のコールバックから OCI_CONTINUE 以外が戻された場合は、**errnop* の値を設定する必要があります。最後のコールバックから OCI_CONTINUE が戻されない場合は、エラー・ハンドル内に **errnop* の値が設定されます。

**errnop* に Oracle 以外のエラー番号が戻された場合は、そのエラー番号に対して *OCIErrorGet()* 関数から適切なテキストが戻されるように、コールバックを登録する必要があります。

arglist (IN)

ここで引数として渡される、OCI コールへのパラメータです。これらのパラメータは、*va_arg* を使って参照を解除する必要があります。ユーザー・コールバックのデモンストレーション・プログラムを参照してください。デモンストレーション・プログラムのリストについては、[付録 B の「OCI デモ・プログラム」](#)を参照してください。

表 15-11 OCI 関数コード

#	OCI ルーチン	#	OCI ルーチン	#	OCI ルーチン
1	OCIInitialize	33	OCITransStart	65	OCIDefineByPos
2	OCIHandleAlloc	34	OCITransDetach	66	OCIBindByPos
3	OCIHandleFree	35	OCITransCommit	67	OCIBindByName
4	OCIDescriptorAlloc	36	(使用されていません。)	68	OCILobAssign
5	OCIDescriptorFree	37	OCIErrorGet	69	OCILobIsEqual
6	OCIEnvInit	38	OCILobFileOpen	70	OCILobLocatorIsInit
7	OCIserverAttach	39	OCILobFileClose	71	OCILobEnableBuffering
8	OCIserverDetach	40	(使用されていません。)	72	OCILobCharSetID
9	(使用されていません。)	41	(使用されていません。)	73	OCILobCharSetForm
10	OCISessionBegin	42	OCILobCopy	74	OCILobFileSetName
11	OCISessionEnd	43	OCILobAppend	75	OCILobFileGetName
12	OCIPasswordChange	44	OCILobErase	76	OCILogon
13	OCIStmtPrepare	45	OCILobGetLength	77	OCILogoff
14	(使用されていません。)	46	OCILobTrim	78	OCILobDisableBuffering
15	(使用されていません。)	47	OCILobRead	79	OCILobFlushBuffer
16	(使用されていません。)	48	OCILobWrite	80	OCILobLoadFromFile
17	OCIBindDynamic	49	(使用されていません。)	81	OCILobOpen
18	OCIBindObject	50	OCIBreak	82	OCILobClose
19	(使用されていません。)	51	OCIserverVersion	83	OCILobIsOpen
20	OCIBindArrayOfStruct	52	(使用されていません。)	84	OCILobFileIsOpen
21	OCIStmtExecute	53	(使用されていません。)	85	OCILobFileExists
22	(使用されていません。)	54	OCIAttrGet	86	OCILobFileCloseAll
23	(使用されていません。)	55	OCIAttrSet	87	OCILobCreateTemporary
24	(使用されていません。)	56	OCIParamSet	88	OCILobFreeTemporary
25	OCIDefineObject	57	OCIParamGet	89	OCILobIsTemporary
26	OCIDefineDynamic	58	OCIStmtGetPieceInfo	90	OCIAQEnq
27	OCIDefineArrayOfStruct	59	OCILdaToSvcCtx	91	OCIAQDeq
28	OCIStmtFetch	60	(使用されていません。)	92	OCIReset
29	OCIStmtGetBindInfo	61	OCIStmtSetPieceInfo	93	OCISvcCtxToLda
30	(使用されていません。)	62	OCITransForget	94	OCILobLocatorAssign

#	OCI ルーチン	#	OCI ルーチン	#	OCI ルーチン
31	(使用されていません。)	63	OCITransPrepare	95	(使用されていません。)
32	OCIDescribeAny	64	OCITransRollback	96	OCIAQListen

関連関数

[OCIUserCallbackGet\(\)](#)

OCIUserCallbackRegister()

OCI ナビゲーション関数と型関数

この章では、Oracle データベース・サーバーから取り出されたオブジェクトをナビゲートするために使う OCI ナビゲーション関数について説明します。さらに、ここには型記述子オブジェクト (TDO) を取得するために使う関数の説明も含まれています。この章には、次の項があります。

- [概要](#)
- [OCI フラッシュまたはリフレッシュ関数](#)
- [OCI オブジェクトおよびキャッシュへのマーク設定 / 解除関数](#)
- [OCI オブジェクト・ステータス取得関数](#)
- [その他の OCI オブジェクト関数](#)
- [OCI 確保、確保解除および解放関数](#)
- [OCI 型情報アクセサ関数](#)

注意 : この章で説明する関数は、Oracle8i Enterprise Edition をインストールしている場合にだけ使用可能です。

概要

オブジェクト・ナビゲーション・パラダイムの中では、データは参照によって結合されたオブジェクトのグラフとして表現されます。このグラフの中のオブジェクトに到達するには、参照をたどります。OCIは、オブジェクトに対するナビゲーション・インタフェースをOracle Server内に用意しています。この章では、これらのコールについて説明します。

OCIオブジェクト環境は、OCI_OBJECTモードでアプリケーションがOCIInitialize()をコールすると初期化されます。

関連項目: この章で説明しているコールの使用の詳細は、第10章の「OCIオブジェクト・リレーショナル・プログラミング」および第13章の「オブジェクトのキャッシュおよびオブジェクト・ナビゲーション」を参照してください。

オブジェクト型と存続期間

オブジェクト・インスタンスとは、Oracleデータベースに定義されている型のオカレンスです。この項では、OCIでのオブジェクト・インスタンスの記述方法について説明します。16-3ページの図16-1を参照してください。OCIでは、オブジェクト・インスタンスは型、存続期間および参照可能性によって次のように分類されます。

- 永続オブジェクトはオブジェクト型のインスタンスの1つです。永続オブジェクトは、サーバー内の表の行中に常駐し、セッション（接続）の継続時間より長く存在できます。永続オブジェクトは、オブジェクト識別子を格納するオブジェクト参照によって識別できます。永続オブジェクトは、そのオブジェクト参照を確保することで取得できます。
- 一時オブジェクトはオブジェクト型のインスタンスです。一時オブジェクトは、セッション（接続）の継続時間より長く存在することはできず、一時的な計算結果を格納するために使用されます。一時オブジェクトは、一時オブジェクト識別子を格納するオブジェクト参照によって識別できます。
- 値はユーザー定義型（オブジェクト型またはコレクション型）のインスタンスの1つであるか、ビルトインOracle型のいずれかです。オブジェクトとは異なり、オブジェクト型の値は、参照ではなくメモリー・ポインタによって識別されます。

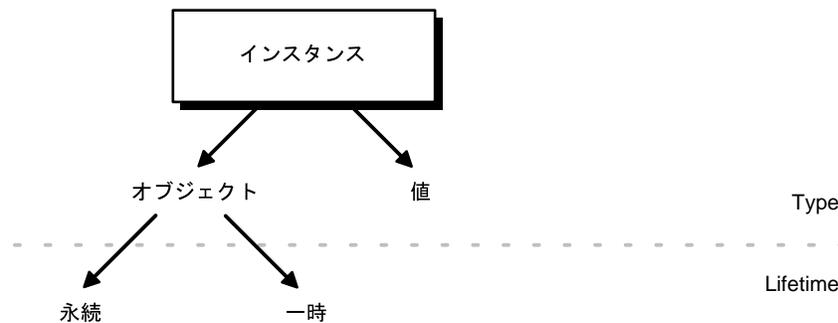
値はスタンドアロン値または埋込み値になります。スタンドアロン値は、通常選択文を発行することで取得されます。OCIでは、クライアント・プログラムがSQL文を発行することで値の中にオブジェクト表の行を選択することも許されています。データベース内の参照可能オブジェクトは、参照によって識別できない値として表現できます。また、スタンドアロン値を、VARCHARやrawなど、オブジェクト内のアウト・ラインの属性にしたり、VARCHAR、raw、オブジェクトなど、コレクション内のアウト・ラインの要素にすることができます。

埋込み値は、格納インスタンス内に物理的に格納されます。埋込み値は、数値やネストしたオブジェクトなどのオブジェクト内のイン・ラインの属性、またはコレクション内のイン・ラインの要素にすることができます。

すべての値は、OCI では一時的なものであると見なされます。つまり、OCI は値のデータベースへの自動フラッシュをサポートしないため、クライアントは、値をデータベースに格納するための SQL 文を明示的に実行する必要があります。埋込み値は、格納インスタンスがフラッシュされるときにフラッシュされます。

図 16-1 に、インスタンスの型と存続期間による分類方法を示します。

図 16-1 インスタンスの型と存続期間による分類



各インスタンスの違いを次の表に示します。

	永続オブジェクト	一時オブジェクト	値
型	オブジェクト型	オブジェクト型	オブジェクト型、ビルトイン、コレクション
最長存続期間	オブジェクトが削除されるまで	セッション	セッション
参照可能性	あり	あり	なし
埋込み可能性	なし	なし	あり

用語

この章では、次の用語が使用されます。

- オブジェクトは、一般に、永続オブジェクト、一時オブジェクト、オブジェクト型のスタンドアロン値またはオブジェクト型の埋込み値を指すために使用します。
- 参照可能オブジェクトは、永続オブジェクトまたは一時オブジェクトを指します。
- スタンドアロン・オブジェクトは、永続オブジェクト、一時オブジェクトまたはオブジェクト型のスタンドアロン値を指します。
- 埋込みオブジェクトは、オブジェクト型の埋込み値を指します。

- オブジェクトは、作成（新規作成）された場合、または更新または削除のマークが設定された場合に、使用済みと表現されます。

異なるオブジェクト型を指すために使用する用語の詳細は、10-5 ページの「[永続オブジェクトおよび一時オブジェクト、値](#)」を参照してください。

関数の構文

関数ごとに次の情報が記載されています。

用途

この関数によって実行されるアクションの簡単な説明。

構文

この関数をコールするための構文を示すコードの断片で、パラメータの順序と種類が含まれています。

コメント

この関数に関する詳細情報（ある場合）、関数の使用上の制約やアプリケーション内でこの関数を使用するときに役に立つ情報が記載されています。

パラメータ

この関数の各パラメータの説明。これにはパラメータのモードが含まれます。引数のモードには、以下のように3つの可能な値があります。

モード	説明
IN	Oracle にデータを渡すパラメータ
OUT	このコールまたは後続のコールで Oracle からデータを受け取るパラメータ
IN/OUT	このコールでデータを渡し、このコールまたは後続のコールからの戻りでデータを受け取るパラメータ

戻り値

上記にリストされた標準リターン・コード以外が関数によって戻される場合の、関数によって戻される値についての説明です。

関連関数

役に立つ可能性がある追加情報を提供する関連コールの一覧です。

ナビゲーション関数の戻り値

OCI ナビゲーション関数は、通常次の値を戻します。

戻り値	意味
OCI_SUCCESS	操作は成功しました。
OCI_ERROR	操作は失敗しました。関数に渡されたエラー・ハンドルに対して <i>OCIErrorGet()</i> をコールすることで特定のエラーを検索できます。
OCI_INVALID_HANDLE	関数に渡された環境ハンドルまたはエラー・ハンドルが NULL です。

この章では、各関数に関する記述の後で関数固有の戻り値についての情報を説明します。各関数から戻される特定のエラー・コードについての情報は、次の項で記述します。

関連項目 : リターン・コードおよびエラー処理の詳細は、2-26 ページの「[エラー処理](#)」を参照してください。

キャッシュおよびオブジェクト関数のためのサーバー往復

個々の OCI キャッシュ関数およびオブジェクト関数に必要なサーバー・ラウンドトリップ回数を示す表は、[付録 C の「OCI 関数のサーバー・ラウンドトリップ」](#)を参照してください。

ナビゲーション関数エラー・コード

[表 16-1](#) に、各 OCI ナビゲーション関数によって戻される可能性がある外部 Oracle エラー・コードのリストを示します。表の後に各エラーの内容を説明します。

表 16-1 OCI ナビゲーション関数エラー・コード

関数	発生する可能性のある ORA エラー
OCICacheFlush()	24350、21560、21705
OCICacheFree()	24350、21560、21705
OCICacheRefresh()	24350、21560、21705
OCICacheUnmark()	24350、21560、21705
OCICacheUnpin()	24350、21560、21705
OCIObjectArrayPin()	24350、21560
OCIObjectCopy()	24350、21560、21705、21710

表 16-1 OCI ナビゲーション関数エラー・コード (続き)

関数	発生する可能性のある ORA エラー
OCIObjectExists()	24350、21560、21710
OCIObjectFlush()	24350、21560、21701、21703、21708、21710
OCIObjectFree()	24350、21560、21603、21710
OCIObjectGetAttr()	21560、21600、22305
OCIObjectGetInd()	24350、21560、21710
OCIObjectGetTypeRef()	24350、21560、21710
OCIObjectIsDirty()	24350、21560、21710
OCIObjectIsLocked()	24350、21560、21710
OCIObjectLock()	24350、21560、21701、21708、21710
OCIObjectLockNoWait()	24350、21560、21701、21708、21710
OCIObjectMarkDelete()	24350、21560、21700、21701、21702、21710
OCIObjectMarkDeleteByRef()	24350、21560
OCIObjectMarkUpdate()	24350、21560、21700、21701、21710
OCIObjectNew()	24350、21560、21705、21710
OCIObjectPin()	24350、21560、21700、21702
OCIObjectPinCountReset()	24350、21560、21710
OCIObjectPinTable()	24350、21560、21705
OCIObjectRefresh()	24350、21560、21709、21710
OCIObjectSetAttr()	21560、21600、22305、22279、21601
OCIObjectUnmark()	24350、21560、21710
OCIObjectUnmarkByRef()	24350、21560
OCIObjectUnpin()	24350、21560、21710
OCIObjectGetObjectRef()	24350、21560、21710

表 16-1 の ORA エラーには、次のような意味があります。

- ORA-21560 - 引数 : %s が NULL または無効、範囲外です。
- ORA-21600 - パス式が長すぎます。

- ORA-21601 - 属性はオブジェクトではありません。
- ORA-21603 - プロパティ ID[%s] が無効です。
- ORA-21700 - オブジェクトが存在しないか、削除マークが設定されています。
- ORA-21701 - オブジェクトを別のサーバーへフラッシュしようとした。
- ORA-21702 - オブジェクトが未インスタンス化またはキャッシュ内で逆インスタンス化されています。
- ORA-21703 - 変更されていないオブジェクトはフラッシュできません。
- ORA-21704 - フラッシュを最初に行わずに、キャッシュまたは接続を終了できません。
- ORA-21705 - サービス・コンテキストが無効です。
- ORA-21708 - 一時オブジェクトで不適切な操作がありました。
- ORA-21709 - 変更されたオブジェクトをリフレッシュできません。
- ORA-21710 - 引数にはオブジェクトの有効なメモリー・アドレスが必要です。
- ORA-22279 - LOB バッファが使用可能な状態では、操作を実行できません。
- ORA-22305 - 属性 / メソッド / パラメータ ¥"%s¥" が見つかりません。
- ORA-24350 - OCI コールは使用できません。

OCI フラッシュまたはリフレッシュ関数

この項では、OCI フラッシュまたはリフレッシュ関数について説明します。

表 16-2 OCI フラッシュまたはリフレッシュ関数クイック・リファレンス

関数	用途	ページ
OCICacheFlush()	キャッシュ内の変更済み永続オブジェクトをサーバーにフラッシュします。	16-9 ページ
OCICacheRefresh()	確保されている永続オブジェクトをリフレッシュします。	16-11 ページ
OCIObjectFlush()	単一の変更済み永続オブジェクトをサーバーにフラッシュします。	16-13 ページ
OCIObjectRefresh()	永続オブジェクトをリフレッシュします。	16-14 ページ

OCICacheFlush()

用途

この関数は、修正済みの永続オブジェクトをサーバーにフラッシュします。

構文

```
sword OCICacheFlush ( OCIEnv          *env,
                     OCIError       *err,
                     CONST OCISvcCtx *svc,
                     dvoid          *context,
                     OCIRef         *(*get)
                               ( dvoid *context,
                               ub1   *last ),
                     OCIRef         **ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキストです。

context (IN) [optional]

クライアント・コールバック関数 *get* への引数であるユーザー・コンテキストを指定します。このパラメータは、ユーザー・コンテキストがない場合、NULL が設定されます。

get (IN) [optional]

フラッシュを必要とする使用済みオブジェクトのバッチを取り出す反復子として機能するクライアント定義関数です。関数が NULL でない場合はこの関数がコールされ、使用済みオブジェクトの参照が取得されます。これは、クライアント関数によって NULL 参照が戻されるか、パラメータ *last* が TRUE に設定されるまで繰り返されます。クライアント関数が呼び出されるたびに *get()* に *context* パラメータが渡されます。このパラメータは、ユーザー・コールバックを指定しない場合は NULL にする必要があります。クライアント関数によって戻されたオブジェクトが使用済み永続オブジェクトでない場合、そのオブジェクトは無視されません。

クライアント関数から戻されるすべてのオブジェクトは、同一のサービス・コンテキストを使用する新しいまたは確保済みのものでなければなりません。そうでない場合はエラーが通

知されます。戻されたオブジェクトは、それらが使用済みとマークされた順序でフラッシュされることに注意してください。

このパラメータに NULL を渡すと（クライアント定義関数が指定されない場合など）指定されたサービス・コンテキストのすべての使用済み永続オブジェクトが、使用済みとマークされた順序にフラッシュされます。

ref (OUT) [optional]

オブジェクトのフラッシュでエラーが起きると、(**ref*) によってエラーの原因となったオブジェクトが指し示されます。*ref* が NULL の場合、オブジェクトは戻りません。**ref* が NULL の場合は、オブジェクトの参照が割り当てられ、そのオブジェクトを指すように設定されます。**ref* が NULL でない場合は、オブジェクトの参照が指定の領域にコピーされます。エラーの原因が使用済みオブジェクトではない場合は、指定の REF が NULL 参照になるように初期化されます（*OCIRefsNull (*ref)* が TRUE になります）。

REF はセッション継続期間（*OCI_DURATION_SESSION*）に対して割り当てられます。*OCIObjectFree()* 関数を使用すると、アプリケーションは割り当てられた REF を解放できません。

コメント

この関数は、オブジェクト・キャッシュの修正済みの永続オブジェクトをサーバーにフラッシュします。オブジェクトは、新たに作成された順序、あるいは更新か削除のマークが設定された順序でフラッシュされます。フラッシュの詳細は、「[OCIObjectFlush\(\)](#)」を参照してください。

この関数はネットワークを 1 回だけ往復します。

関連関数

[OCIObjectFlush\(\)](#)

OCICacheRefresh()

用途

キャッシュ内のすべての確保済み永続オブジェクトをリフレッシュします。

構文

```
sword OCICacheRefresh ( OCIEnv          *env,
                       OCIError       *err,
                       CONST OCISvcCtx *svc,
                       OCIRefreshOpt  option,
                       dvoid          *context,
                       OCIRef         *(*get) (dvoid *context),
                       OCIRef         **ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキストです。

option (IN) [optional]

OCI_REFRESH_LOADED が指定されると、そのトランザクションの間にロードされたすべてのオブジェクトをリフレッシュします。オプションが *OCI_REFRESH_LOADED* で、パラメータ *get* が NULL ではない場合、パラメータは無視されます。

context (IN) [optional]

クライアント・コールバック関数 *get* への引数であるユーザー・コンテキストを指定します。このパラメータは、ユーザー・コンテキストがない場合、NULL が設定されます。

get (IN) [optional]

リフレッシュされる必要があるオブジェクトのバッチを取り出す反復子として機能するクライアント定義関数。関数が NULL でない場合はこの関数がコールされ、オブジェクトの参照が取得されます。参照が NULL でない場合、オブジェクトはリフレッシュされます。このステップは、この関数から NULL 参照が戻されるまで繰り返されます。クライアント関数が呼び出されるたびに *get()* に *context* パラメータが渡されます。このパラメータは、ユーザー・コールバックを指定しない場合は NULL にする必要があります。

ref (OUT) [optional]

オブジェクトのリフレッシュ中にエラーが起きると、(*ref)によってエラーの原因となったオブジェクトが指し示されます。refがNULLの場合、オブジェクトは戻りません。*refがNULLの場合は、オブジェクトの参照が割り当てられ、そのオブジェクトを指すように設定されます。*refがNULLでない場合は、オブジェクトの参照が指定の領域にコピーされます。エラーの原因がオブジェクトでない場合は、指定されたREFがNULL参照になるように初期化されます (OCISqlIsNull (*ref) が TRUE になります)。

コメント

この関数は、すべての確保済み永続オブジェクトをリフレッシュします。確保が解除されたすべての永続オブジェクトは、オブジェクト・キャッシュから解放されます。

リフレッシュの詳細は、「[OCIObjectRefresh\(\)](#)」の説明、および13-11ページの「[オブジェクト・コピーのリフレッシュ](#)」を参照してください。

警告: オブジェクトがリフレッシュされると、それらのオブジェクトの2次メモリーがメモリー内の別の場所に移動することがあります。このため、このコール前に保存された属性へのポインタが無効になる可能性があります。2次メモリーを使用する属性には、OCIStr *、OCIColl *、OCIRaw * があります。

関連関数

[OCIObjectRefresh\(\)](#)

OCIObjectFlush()

用途

修正済みの永続オブジェクトをサーバーにフラッシュします。

構文

```
sword OCIObjectFlush ( OCIEnv      *env,  
                      OCIError    *err,  
                      dvoid        *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

object (IN)

永続オブジェクトのポインタです。このコールの前に、オブジェクトが確保されている必要があります。

コメント

この関数は、修正済みの永続オブジェクトをサーバーにフラッシュします。フラッシュ時には、オブジェクトの排他ロックが暗黙のうちに取得されます。オブジェクトがサーバーに書き込まれるときにトリガーを起動できます。この関数は、一時オブジェクトと値、および修正されていない永続オブジェクトではエラーを戻します。

サーバーでのトリガーによってオブジェクトを修正できます。キャッシュにあるオブジェクトをデータベースと整合性を保つようするため、アプリケーションによってキャッシュにあるオブジェクトを解放したり、リフレッシュしたりできます。

フラッシュするオブジェクトに内部 LOB 属性があり、その LOB 属性が [OCIObjectCopy\(\)](#)、[OCILobAssign\(\)](#) または [OCILobLocatorAssign\(\)](#) によって、あるいは別の LOB ロケータの割当てによって修正された場合は、フラッシュによって、割当て時にソース LOB 内に存在した LOB 値のコピーか、内部 LOB のロケータまたはオブジェクトのコピーが生成されます。LOB 関数の詳細は、15-108 ページの「[LOB 関数](#)」を参照してください。

関連関数

[OCIObjectPin\(\)](#)、[OCICacheFlush\(\)](#)

OCIObjectRefresh()

用途

現行データベースの最新のスナップショットから永続オブジェクトをリフレッシュします。

構文

```
sword OCIObjectRefresh ( OCIEnv      *env,  
                        OCIError    *err,  
                        dvoid        *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

object (IN)

永続オブジェクトへのポインタで、それがすでに確保されている必要があります。

コメント

この関数は、サーバー内の最新のスナップショットから検索されたデータを使用して、オブジェクトをリフレッシュします。オブジェクトは、オブジェクト・キャッシュ内のオブジェクトとサーバー側のオブジェクトとの間に整合性がなくなったときに、リフレッシュされなければなりません。

注意: オブジェクトをサーバーにフラッシュするときに、トリガーを起動してさらに多くのサーバー内のオブジェクトを変更できます。その場合、オブジェクト・キャッシュ内の（トリガーによって変更したオブジェクトと）同一のオブジェクトは最新の状態ではなく、リフレッシュしないとロックまたはフラッシュできません。

これは、ユーザーが SQL 文または PL/SQL プロシージャを発行してサーバー内のオブジェクトを修正したときに発生します。

警告: 最後のフラッシュ以降にオブジェクト（使用済みオブジェクト）に対して行われた修正は、そのオブジェクトがこの関数によってリフレッシュされると失われます。

リフレッシュの後、オブジェクトのさまざまなメタ属性フラグと継続時間が、次のように修正されます。

オブジェクト属性	リフレッシュ後の状態
既存	適切な値にセット
確保済み	変更なし
割当て継続時間	変更なし
確保継続時間	変更なし

リフレッシュされたオブジェクトはその場で置き換えられます。オブジェクトがその場で置き換えられると、オブジェクトのトップ・レベル・メモリーが再利用されるので、新しいデータが同じメモリー・アドレスにロードされるようになります。NULL 構造体のトップ・レベル・メモリーも再利用されます。トップ・レベル・メモリーとは異なり、2次メモリーは解放され、再割当てが行われます。

2次メモリーのアドレスをローカル変数に割り当てるなど、2次メモリー・チャンクを指すポインタを保持する機能を作成しているときは、このポインタがオブジェクトのリフレッシュ後は無効になるので、注意してください。

この関数は、一時オブジェクトまたは値に対して何も影響を与えません。

関連関数

[OCICacheRefresh\(\)](#)

OCI オブジェクトおよびキャッシュへのマーク設定 / 解除関数

この項では、OCI オブジェクトおよびキャッシュへのマーク設定 / 解除関数について説明します。

表 16-3 OCI ナビゲーション関数クイック・リファレンス

関数	用途	ページ
OCICacheUnmark()	キャッシュ内のオブジェクトのマークを解除します。	16-17 ページ
OCIObjectMarkDelete()	オブジェクトに削除済みのマークを付けます。または値インスタンスを削除します。	16-18 ページ
OCIObjectMarkDeleteByRef()	参照によって指定されたオブジェクトに削除マークを設定します。	16-19 ページ
OCIObjectMarkUpdate()	オブジェクトに更新済み / 使用済みのマークを付けます。	16-20 ページ
OCIObjectUnmark()	オブジェクトのマークを解除します。	16-22 ページ
OCIObjectUnmarkByRef()	参照によって指定されたオブジェクトのマークを解除します。	16-23 ページ

OCICacheUnmark()

用途

オブジェクト・キャッシュ内のすべての使用済みオブジェクトのマークを解除します。

構文

```
sword OCICacheUnmark ( OCIEnv          *env,  
                      OCIError       *err,  
                      CONST OCISvcCtx *svc );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキストです。

コメント

接続が指定された場合、この関数は、その接続内のすべての使用済みオブジェクトのマークを解除します。それ以外の場合は、キャッシュ内のすべての使用済みオブジェクトのマークを解除します。オブジェクトのマーク解除の詳細は、16-22 ページの [OCIObjectUnmark\(\)](#) を参照してください。

関連関数

[OCIObjectUnmark\(\)](#)

OCIObjectMarkDelete()

用途

スタンドアロン・インスタンスへのポインタを指定して、そのインスタンスに削除マークを設定します。

構文

```
sword OCIObjectMarkDelete ( OCIEnv          *env,  
                             OCIError       *err,  
                             dvoid          *instance );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

instance (IN)

インスタンスへのポインタです。これはスタンドアロン型でなければならず、オブジェクトの場合は確保されていなければなりません。

コメント

この関数は、スタンドアロン・インスタンスのポインタを受け取り、そのオブジェクトに削除マークを設定します。オブジェクトは、次の規則に従って解放されます。

永続オブジェクト

オブジェクトに削除マークが設定されます。オブジェクトのメモリーは解放されません。オブジェクトは、そのオブジェクトがフラッシュされたときにサーバー内で削除されます。

一時オブジェクト

オブジェクトに削除マークが設定されます。オブジェクトのメモリーは解放されません。

値

この関数は、ただちに値を解放します。

関連関数

[OCIObjectMarkDeleteByRef\(\)](#)、[OCIObjectGetProperty\(\)](#)

OCIObjectMarkDeleteByRef()

用途

オブジェクトへの参照を指定して、そのオブジェクトに削除マークを設定します。

構文

```
sword OCIObjectMarkDeleteByRef ( OCIEnv          *env,  
                                OCIError       *err,  
                                OCIRef         *object_ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

object_ref (IN)

削除されるオブジェクトへの参照です。

コメント

この関数は、オブジェクトへの参照を受け取り、*object_ref* によって指定されたオブジェクトに削除マークを設定します。オブジェクトはマーク付けされて、次のように解放されます。

永続オブジェクト

オブジェクトがロードされていない場合は、一時オブジェクトが作成され、それに対して削除マークが設定されます。それ以外の場合は、オブジェクトに削除マークが設定されます。

オブジェクトは、そのオブジェクトがフラッシュされたときにサーバー内で削除されます。

一時オブジェクト

オブジェクトに削除マークが設定されます。オブジェクトは、確保が解除されるまで解放されません。

関連関数

[OCIObjectMarkDelete\(\)](#)、[OCIObjectGetProperty\(\)](#)

OCIObjectMarkUpdate()

用途

永続オブジェクトに更新または使用済みマークを設定します。

構文

```
sword OCIObjectMarkUpdate ( OCIEnv      *env,  
                           OCIError    *err,  
                           dvoid       *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

object (IN)

永続オブジェクトへのポインタで、それがすでに確保されている必要があります。

コメント

この関数は、永続オブジェクトに更新または使用済みマークを設定します。オブジェクトの型によって次の規則が適用されます。オブジェクトの使用済み状態は、[OCIObjectIsLocked\(\)](#) をコールすることによりチェックできます。

永続オブジェクト

この関数は、指定された永続オブジェクトに更新マークを設定します。

永続オブジェクトは、オブジェクト・キャッシュがフラッシュされたときにサーバーに書き込まれます。この関数は、オブジェクトのロックもフラッシュも行いません。削除済みオブジェクトを更新しようとするエラーが発生します。

オブジェクトの更新マークの設定とフラッシュが実行された後、オブジェクトがフラッシュ後も使用済みになっている場合は、この関数を再度コールしてオブジェクトに更新マークを設定する必要があります。

一時オブジェクト

この関数は、指定された一時オブジェクトに更新マークを設定します。一時オブジェクトは、サーバーには書き込まれません。削除済みオブジェクトを更新しようとするエラーが発生します。

値

値に対しては何も行われません。

この関数の使用方法の詳細は、10-14 ページの「[オブジェクトのマークおよび変更のフラッシュ](#)」を参照してください。

関連関数

[OCIObjectPin\(\)](#)、[OCIObjectGetProperty\(\)](#)

OCIObjectUnmark()

用途

オブジェクトの使用済みマークを解除します。

構文

```
sword OCIObjectUnmark ( OCIEnv      *env,  
                        OCIError    *err,  
                        dvoid        *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

object (IN)

永続オブジェクトへのポインタです。オブジェクトは確保されていなければなりません。

コメント

永続オブジェクトと一時オブジェクト

この関数は、指定された永続オブジェクトの使用済みマークを解除します。オブジェクトに対して行われた変更は、サーバーに書き込まれません。オブジェクトにロック・マークが設定されている場合は、ロックされたままになります。オブジェクトに対してこれまで実行された変更は、暗黙のうちに元に戻されます。

値

値に対しては何も行われません。これは、値に対して関数をコールしても何の効果もないことを意味しています。

関連関数

[OCIObjectUnmarkByRef\(\)](#)

OCIObjectUnmarkByRef()

用途

オブジェクトへの REF が与えられたら、オブジェクトの使用済みマークを解除します。

構文

```
sword OCIObjectUnmarkByRef ( OCIEnv      *env,  
                             OCIError    *err,  
                             OCIRef      *ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

ref (IN)

オブジェクトの参照です。オブジェクトは確保されていない限りなりません。

コメント

この関数は、オブジェクトの使用済みマークを解除します。オブジェクトへの REF を引数として取ることを除けば、この関数は [OCIObjectUnmark\(\)](#) とまったく同じです。

永続オブジェクトと一時オブジェクト

この関数は、指定された永続オブジェクトの使用済みマークを解除します。オブジェクトに対して行われた変更は、サーバーに書き込まれません。オブジェクトにロック・マークが設定されている場合は、ロックされたままになります。オブジェクトに対してこれまで実行された変更は、暗黙のうちに元に戻されます。

値

値に対しては何も行われません。

関連関数

[OCIObjectUnmark\(\)](#)

OCI オブジェクト・ステータス取得関数

この項では、OCI オブジェクト・ステータス取得関数について説明します。

表 16-4 OCI ナビゲーション関数クイック・リファレンス

関数	用途	ページ
OCIObjectExists()	インスタンスの存在ステータスを取得します。	16-25 ページ
OCIObjectGetProperty()	特定のオブジェクト・プロパティのステータスを取得します。	16-26 ページ
OCIObjectIsLocked()	インスタンスの使用済みステータスを取得します。	16-31 ページ
OCIObjectIsLocked()	インスタンスのロック・ステータスを取得します。	16-31 ページ

OCIObjectExists()

用途

スタンドアロン・インスタンスの存在メタ属性を戻します。

構文

```
sword OCIObjectExists ( OCIEnv      *env,  
                        OCIError    *err,  
                        dvoid        *ins,  
                        boolean      *exist );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

ins (IN)

インスタンスへのポインタです。オブジェクトの場合は、確保されていなければなりません。

exist (OUT)

存在ステータスの戻り値です。

コメント

この関数は、インスタンスの存在を戻します。インスタンスが値の場合、この関数は常に TRUE を戻します。インスタンスは、スタンドアロン型の永続または一時オブジェクトでなければなりません。

オブジェクトのメタ属性の詳細は、10-16 ページの「[オブジェクトのメタ属性](#)」を参照してください。

関連関数

[OCIObjectPin\(\)](#)

OCIObjectGetProperty()

用途

オブジェクトの指定されたプロパティを取り出します。

構文

```
sword OCIObjectGetProperty ( OCIEnv          *envh,  
                             OCIError       *errh,  
                             CONST dvoid    *obj,  
                             OCIObjectPropId propertyId,  
                             dvoid         *property,  
                             ub4           *size );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

obj (IN)

そのプロパティが戻されるオブジェクトです。

propertyId (IN)

目的のプロパティを識別する識別子です。

property (OUT)

目的のプロパティがコピーされるバッファです。

size (IN/OUT)

入力の際は、コール側から渡されるプロパティ・バッファのサイズがこのパラメータによって指定されます。

出力の際は、戻されるプロパティのサイズがバイト単位でこれに含まれます。このパラメータは、OCI_OBJECTPROP_SCHEMA、OCI_OBJECTPROP_TABLE などの文字列型のプロパティだけに必要です。非文字列型のプロパティの場合は、サイズが固定なのでこのパラメータは無視されます。

コメント

この関数によりオブジェクトの指定されたプロパティが戻されます。目的のプロパティは *propertyId* で識別されます。プロパティの値は *property* にコピーされ、文字列型プロパティの場合は、文字列のサイズが *size* によって戻されます。

オブジェクトは、その存続期間と参照可能性に基づいて永続、一時、値に分類されます。プロパティのいくつかは永続オブジェクトのみに適用され、また他のプロパティは永続オブジェクトと一時オブジェクト（の両方）に適用されます。指定したオブジェクトに適用されないプロパティを取得しようとすると、エラーが戻されます。このようなエラーを回避するには、まずそのオブジェクト（の OCI_OBJECTPROP_LIFETIME プロパティ）が永続、一時、値のどれであるかをチェックし、それに基づいてその他のプロパティを適切に問い合わせるようにします。

異なるプロパティ Id とそれに対応する *property* 引数の型を次に示します。

OCI_OBJECTPROP_LIFETIME

このプロパティによって指定されたオブジェクトが永続オブジェクト、一時オブジェクトまたは、値のインスタンスかが識別されます。 *property* 引数は OCIObjectLifetime 型の変数へのポインタでなければなりません。可能な値は、次のとおりです。

- OCI_OBJECT_PERSISTENT
- OCI_OBJECT_TRANSIENT
- OCI_OBJECT_VALUE

OCI_OBJECTPROP_SCHEMA

このプロパティにより、その中にオブジェクトが存在する表のスキーマ名が戻されます。指定されたオブジェクトが一時的インスタンスまたは値をポイントする場合、エラーが戻されます。スキーマ名の保持にバッファのサイズが足りない場合は、エラーが戻され、エラー・メッセージで必要なサイズが表示されます。成功した場合は、戻り値のスキーマ名のサイズがバイト単位で *size* によって戻されます。 *property* 引数は、text 型の配列でなければなりません。コール側は、 *size* をバイト単位で配列のサイズに設定する必要があります。

OCI_OBJECTPROP_TABLE

このプロパティによってそのオブジェクトが存在する表名が戻されます。指定されたオブジェクトが一時的インスタンスまたは値をポイントする場合、エラーが戻されます。表名の保持にバッファが足りない場合は、エラーが戻され、エラー・メッセージで必要なサイズが表示されます。成功した場合は、戻り値のスキーマ名のサイズがバイト単位で *size* によって戻されます。 *property* 引数は、text 型の配列で、コール側は、その *size* をバイト単位で配列のサイズに設定しなければなりません。

OCI_OBJECTPROP_PIN_DURATION

このプロパティによってオブジェクトの確保継続時間が戻されます。指定されたオブジェクトが値のインスタンスを指し示している場合は、エラーが戻されます。 *property* 引数は OCIDuration 型の変数へのポインタでなければなりません。有効な値は、次のとおりです。

- OCI_DURATION_SESSION
- OCI_DURATION_TRANS

継続時間の詳細は、13-14 ページの「[オブジェクトの継続時間](#)」を参照してください。

OCI_OBJECTPROP_ALLOC_DURATION

このプロパティによりオブジェクトの割当て継続時間が戻されます。*property* 引数は OCIDuration 型の変数へのポインタでなければなりません。次の値が有効です。

- OCI_DURATION_SESSION
- OCI_DURATION_TRANS

継続時間の詳細は、13-14 ページの「[オブジェクトの継続時間](#)」を参照してください。

OCI_OBJECTPROP_LOCK

このプロパティによってオブジェクトのロック・ステータスが戻されます。可能なロック・ステータスは OCILockOpt によって列挙されます。指定されたオブジェクトが一時的インスタンスまたは値インスタンスをポイントする場合、エラーが戻されます。*property* 引数は OCILockOpt 型の変数へのポインタでなければなりません。オブジェクトのロック・ステータスは、OCIObjectIsLocked() 関数をコールすることでも取り出せます。次の値が有効です。

- OCI_LOCK_NONE - ロックなし
- OCI_LOCK_X - 排他的ロック
- OCI_LOCK_X_NOWAIT - NOWAIT オプションを使用した排他ロック

関連項目 : NOWAIT オプションの詳細は、13-13 ページの「[NOWAIT オプションを使ったロック](#)」を参照してください。

OCI_OBJECTPROP_MARKSTATUS

このプロパティによって使用済みステータスが戻され、オブジェクトが新規のオブジェクトであるか、更新されたオブジェクトであるか、削除されたオブジェクトであるかが示されます。指定されたオブジェクトが一時的インスタンスまたは値インスタンスをポイントする場合、エラーが戻されます。*property* 引数は、OCIObjectMarkStatus 型でなければなりません。次の値が有効です。

- OCI_OBJECT_NEW
- OCI_OBJECT_DELETED
- OCI_OBJECT_UPDATED

次のマクロを使ってマーク・ステータスをテストすることが可能です。

- OCI_OBJECT_IS_UPDATED (flag)
- OCI_OBJECT_IS_DELETED (flag)
- OCI_OBJECT_IS_NEW (flag)

- OCI_OBJECT_IS_DIRTY (flag)

OCI_OBJECTPROP_VIEW

このプロパティによって指定されたオブジェクトがビュー・オブジェクトであるかどうか
識別されます。戻されたプロパティ値が TRUE であれば、オブジェクトはビューであり、
TRUE でない場合はオブジェクトはビューではありません。指定されたオブジェクトが一時的
インスタンスまたは値インスタンスをポイントする場合、エラーが戻されます。この
property 引数の型はブール式でなければなりません。

関連関数

[OCIObjectLock\(\)](#)、[OCIObjectMarkDelete\(\)](#)、[OCIObjectMarkUpdate\(\)](#)、[OCIObjectPin\(\)](#)、
[OCIObjectPin\(\)](#)

OCIObjectIsDirty()

用途

オブジェクトに使用済みマークが設定されているかどうかをチェックします。

構文

```
sword OCIObjectIsDirty ( OCIEnv      *env,  
                        OCIError    *err,  
                        dvoid       *ins,  
                        boolean     *dirty );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

ins (IN)

インスタンスへのポインタです。

dirty (OUT)

使用済みステータスに対する値を戻します。

コメント

この関数に渡すインスタンスはスタンドアロンでなければなりません。インスタンスがオブジェクトの場合、そのインスタンスは確保されていなければなりません。

この関数は、インスタンスの使用済みステータスを戻します。インスタンスが値の場合、この関数は常に FALSE を戻します。

関連関数

[OCIObjectMarkUpdate\(\)](#)、[OCIObjectGetProperty\(\)](#)

OCIObjectIsLocked()

用途

オブジェクトのロック・ステータスを取得します。

構文

```
sword OCIObjectIsLocked ( OCIEnv      *env,  
                          OCIError   *err,  
                          dvoid      *ins,  
                          boolean    *lock );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

ins (IN)

インスタンスへのポインタです。インスタンスはスタンドアロン型でなければならず、オブジェクトの場合は確保されていなければなりません。

lock (OUT)

ロック・ステータスに対する値を戻します。

コメント

この関数は、インスタンスのロック・ステータスを戻します。インスタンスが値の場合、この関数は常に FALSE を戻します。

関連関数

[OCIObjectLock\(\)](#)、[OCIObjectGetProperty\(\)](#)

その他の OCI オブジェクト関数

この項では、その他ののオブジェクト関数について説明します。

表 16-5 OCI ナビゲーション関数クイック・リファレンス

関数	用途	ページ
OCIObjectCopy()	1 つのインスタンスを別のインスタンスにコピーします。	16-33 ページ
OCIObjectGetAttr()	オブジェクト属性を取得します。	16-35 ページ
OCIObjectGetInd()	インスタンスの NULL 構造体を取得します。	16-37 ページ
OCIObjectGetObjectRef()	指定のオブジェクトへの参照を戻します。	16-38 ページ
OCIObjectGetTypeRef()	インスタンスの TDO への参照を取得します。	16-39 ページ
OCIObjectLock()	永続オブジェクトをロックします。	16-40 ページ
OCIObjectLockNoWait()	永続オブジェクトをロックしますが、ロック待機は行いません。	16-41 ページ
OCIObjectPin()	新規インスタンスを作成します。	16-54 ページ
OCIObjectSetAttr()	オブジェクト属性を設定します。	16-45 ページ

OCIObjectCopy()

用途

ソース・インスタンスを宛先インスタンスにコピーします。

構文

```
sword OCIObjectCopy ( OCIEnv          *env,
                     OCIError       *err,
                     CONST OCISvcCtx *svc,
                     dvoid          *source,
                     dvoid          *null_source,
                     dvoid          *target,
                     dvoid          *null_target,
                     OCIType        *tdo,
                     OCIDuration    duration,
                     ub1             option );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキスト・ハンドルで、その場合にコピー処理が起きるサービス・コンテキストを指定します。

source (IN)

ソース・インスタンスに対するポインタです。それがオブジェクトであれば、確保される必要があります。16-54 ページの「[OCIObjectPin\(\)](#)」を参照してください。

null_source (IN)

ソース・オブジェクトの NULL 構造体に対するポインタです。

target (IN)

ターゲット・インスタンスに対するポインタです。それがオブジェクトであれば、確保される必要があります。

null_target (IN)

ターゲット・オブジェクトの NULL 構造体に対するポインタです。

tdo (IN)

ソースとターゲットの両方に対する TDO です。 *OCIDescribeAny()* で検索できます。

duration (IN)

ターゲット・メモリーの割当て継続時間です。

option (IN)

このパラメータは使用されていません。0 (ゼロ) または OCI_DEFAULT として渡します。

コメント

この関数は、*source* インスタンスの内容を *target* インスタンスにコピーします。この関数は、次のすべてがコピーされるディープ・コピーを実行します。

- すべての最上位属性 (後述の例外参照)
- 最上位属性から到達可能な (ソースの) すべての 2 次メモリー
- インスタンスの NULL 構造体

メモリーは、*duration* パラメータに指定された継続時間の間、割り当てられます。

ある種のデータ項目は、コピーされません。

- *option* パラメータにオプション OCI_OBJECTCOPY_NOREF が指定されている場合は、ソース内のすべての参照はコピーされません。ターゲット内の参照には NULL が設定されます。
- 属性が内部 LOB の場合、ソース・オブジェクトの LOB ロケータだけがコピーされます。LOB データのコピーは *OCIObjectFlush()* がコールされるまで作成されません。ターゲット・オブジェクトがフラッシュされるまでは、ソースとターゲットのロケータはともに同じ LOB 値を参照します。

ターゲットまたはターゲットの格納インスタンスは、事前に作成されていなければなりません。これは *OCIObjectPin()* によって行うことができます。

source インスタンスと *target* インスタンスは、同じ型でなければなりません。ソースとターゲットが別々のデータベースに置かれている場合は、両方のデータベースに同じ型が存在していなければなりません。

関連関数

[OCIObjectPin\(\)](#)

OCIObjectGetAttr()

用途

オブジェクト属性を検索します。

構文

```
sword OCIObjectGetAttr ( OCIEnv          *env,
                        OCIError        *err,
                        dvoid           *instance,
                        dvoid           *null_struct,
                        struct OCIType  *tdo,
                        CONST text      **names,
                        CONST ub4       *lengths,
                        CONST ub4       name_count,
                        CONST ub4       *indexes,
                        CONST ub4       index_count,
                        OCIInd          *attr_null_status,
                        dvoid           **attr_null_struct,
                        dvoid           **attr_value,
                        struct OCIType  **attr_tdo );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

instance (IN)

オブジェクトへのポインタです。

null_struct (IN)

オブジェクトまたは配列の NULL 構造体です。

tdo (IN)

TDO へのポインタです。

names (IN)

属性名の配列です。パス式に属性名を指定するために使用されます。

lengths (IN)

属性名長の配列です。

name_count (IN)

配列名の要素の数です。

indexes (IN) [optional]

現行ではサポートされていません。(ub4 *)0 を渡します。

index_count (IN) [optional]

現行ではサポートされていません。(ub4)0 を渡します。

attr_null_status (OUT)

属性タイプがプリミティブの場合、属性は NULL ステータスです。

attr_null_struct (OUT)

オブジェクトまたは収集属性の NULL 構造です。

attr_value (OUT)

属性値へのポインタです。

attr_tdo (OUT)

属性の TDO に対するポインタです。

コメント

この関数は、オブジェクトまたは配列から値を取得します。パラメータ *instance* がオブジェクトを示すと、パス式はそのオブジェクトの属性の位置を指定します。オブジェクトが確保され、戻された値はオブジェクトの確保が解除されるまで有効であるとみなされます。

関連関数

[OCIObjectSetAttr\(\)](#)

OCIObjectGetInd()

用途

スタンドアロン・インスタンスの NULL 構造体を取得します。

構文

```
sword OCIObjectGetInd ( OCIEnv      *env,  
                        OCIError    *err,  
                        dvoid        *instance,  
                        dvoid        **null_struct );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

instance (IN)

NULL 構造体を取り込み中のインスタンスへのポインタです。インスタンスは、スタンドアロン型でなければなりません。*instance* がオブジェクトの場合は、事前に確保されていなければなりません。

null_struct (OUT)

インスタンスの NULL 構造体です。

コメント

なし

関連関数

[OCIObjectPin\(\)](#)

OCIObjectGetObjectRef()

用途

指定された永続オブジェクトへの参照を戻します。

構文

```
sword OCIObjectGetObjectRef ( OCIEnv      *env,  
                             OCIError    *err,  
                             dvoid       *object,  
                             OCIRef      *object_ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

object (IN)

永続オブジェクトへのポインタです。事前に確保されていなければなりません。

object_ref (OUT)

object に指定されているオブジェクトへの参照です。参照は、事前に割り当てられていなければなりません。これは、[OCIObjectNew\(\)](#) によって実行できます。

コメント

この関数は、オブジェクトのポインタによって指定された特定の永続オブジェクトへの参照を戻します。(オブジェクトではなく) 値をこの関数に引き渡すと、エラーになります。

関連項目 : オブジェクトのメタ属性の詳細は、10-16 ページの「[オブジェクトのメタ属性](#)」を参照してください。

関連関数

[OCIObjectPin\(\)](#)、[OCIObjectPin\(\)](#)

OCIObjectGetTypeRef()

用途

スタンドアロン・インスタンスの型記述子オブジェクト (TDO) への参照を戻します。

構文

```
sword OCIObjectGetTypeRef ( OCIEnv      *env,  
                           OCIError    *err,  
                           dvoid       *instance,  
                           OCIRef      *type_ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

instance (IN)

スタンドアロン・インスタンスに対するポインタです。これはスタンドアロン型でなければならず、オブジェクトの場合は事前に確保されていなければなりません。

type_ref (OUT)

オブジェクトの型に対する参照です。参照は事前に割り当てられていなければなりません。これは、*OCIObjectNew()* によって実行できます。

コメント

なし

関連関数

[OCIObjectPin\(\)](#)、[OCIObjectPin\(\)](#)

OCIObjectLock()

用途

永続オブジェクトをサーバー側でロックします。

構文

```
sword OCIObjectLock ( OCIEnv      *env,  
                     OCIError   *err,  
                     dvoid      *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

object (IN)

ロックされている永続オブジェクトへのポインタです。事前に確保されていなければなりません。

コメント

この関数は、一時オブジェクトと値ではエラーを戻します。また、オブジェクトが存在しない場合にもエラーを戻します。

オブジェクトのロックの詳細は、13-12 ページの「[更新するためのオブジェクトのロック](#)」を参照してください。

関連関数

[OCIObjectPin\(\)](#)、[OCIObjectIsLocked\(\)](#)、[OCIObjectGetProperty\(\)](#)、[OCIObjectLockNoWait\(\)](#)

OCIObjectLockNoWait()

用途

永続オブジェクトをサーバー側でロックしますが、ロック待機は行いません。ロックが使用不能である場合は、エラーを戻します。

構文

```
sword OCIObjectLockNoWait ( OCIEnv          *env,  
                             OCIError       *err,  
                             dvoid          *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

object (IN)

ロックされている永続オブジェクトへのポインタです。事前に確保されていなければなりません。

コメント

この関数は、永続オブジェクトをサーバー側でロックします。ただし、[OCIObjectLock\(\)](#) とは異なり、この関数では、目的のオブジェクトに対するロックを別のユーザーが保持している場合に待機が行われません。オブジェクトが現在別のユーザーによってロックされている場合は、エラーが戻されます。また、この関数は、一時オブジェクトおよび値、または存在していないオブジェクトに対してエラーを戻します。

オブジェクトのロックは、トランザクションの終了時に解除されます。オブジェクトのロックの詳細は、13-12 ページの「[更新するためのオブジェクトのロック](#)」を参照してください。

[OCIObjectLockNoWait\(\)](#) は、次の値を戻します。

- OCI_INVALID_HANDLE (環境ハンドルまたはエラー・ハンドルが NULL である場合)
- OCI_SUCCESS (操作が成功した場合)
- OCI_ERROR (操作が失敗した場合)

関連関数

[OCIObjectPin\(\)](#)、[OCIObjectIsLocked\(\)](#)、[OCIObjectGetProperty\(\)](#)、[OCIObjectLock\(\)](#)

OCIObjectNew()

用途

スタンドアロン・インスタンスを作成します。

構文

```
sword OCIObjectNew ( OCIEnv          *env,
                    OCIError       *err,
                    CONST OCISvcCtx *svc,
                    OCITypeCode    typecode,
                    OCIType        *tdo,
                    dvoid           *table,
                    OCIDuration     duration,
                    boolean         value,
                    dvoid           **instance );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCL_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

svc (IN) [optional]

OCI サービス・ハンドルです。プログラムでインスタンスの継続時間と OCI サービスを関連付ける（例：トランザクションのコミット時に文字列を解放する）場合は、これを必ず指定します。このパラメータは、TDO が指定された場合は無視されます。

typecode (IN)

インスタンスの型の型コードです。詳細は、3-21 ページの「[型コード](#)」を参照してください。

tdo (IN) [optional]

記述子オブジェクト型に対するポインタです。TDO は、作成されるインスタンスの型を記述します。TDO の取得の詳細は、OCITypeByName() を参照してください。TDO は、オブジェクトやコレクションなどの名前付き型の作成に必要です。

table (IN) [optional]

サーバーにある表を指定しているオブジェクト表に対するポインタです。表を指定しない場合は、このパラメータに NULL を設定できます。作成するインスタンスの種類（永続、一

時、値)を判断するためのオブジェクト表と TDO の使用方法は、次の説明を参照してください。オブジェクト表の検索については、`OCIObjectPinTable()` を参照してください。

duration (IN)

これはオーバーロードされたパラメータです。このパラメータの使用方法は、作成されるインスタンスの種類によって決まります。

- 永続オブジェクト。このパラメータは、確保継続時間を指定します。
- 一時オブジェクト。このパラメータは、割当て継続時間および確保継続時間を指定します。
- 値。このパラメータは、割当て継続時間を指定します。

value (IN)

作成されたオブジェクトが値かどうかを指定します。TRUE の場合は値が作成されます。それ以外の場合は、参照可能オブジェクトが作成されます。インスタンスがオブジェクトではない場合、このパラメータは無視されます。

instance (OUT)

新規に作成されたインスタンスのアドレスです。

コメント

この関数は、型コードまたは TDO によって指定された型のインスタンスを新たに作成します。型コードの詳細は、3-21 ページの「[型コード](#)」を参照してください。パラメータ `typecode` (または `tdo`)、`value` および `table` に基づいて、次のような異なる種類のインスタンスが作成されます。

	<i>table</i> パラメータの値	
TYPE	NULL 以外	NULL
オブジェクト型 (<code>value=TRUE</code>)	値	値
オブジェクト型 (<code>value=FALSE</code>)	永続オブジェクト	一時オブジェクト
ビルトイン型	値	値
コレクション型	値	値

この関数は、インスタンスのトップ・レベル・メモリーのチャンクを割り当てます。トップレベル・メモリー内の属性が初期化されます。つまり、`varchar2` の属性が長さ 0 (ゼロ) の `OCIString` に初期化されます。インスタンスがオブジェクトの場合、そのオブジェクトに存在マークが設定されますがオブジェクトはアトミック NULL になります。

関連項目 : オブジェクト・ビューまたはユーザー作成 OID に基づいた新規オブジェクト作成の詳細は、10-32 ページの「[オブジェクト・ビューまたはユーザー定義 OID に基づいたオブジェクトの作成](#)」を参照してください。

永続オブジェクト

オブジェクトに使用済みおよび存在マークが設定されます。オブジェクトの割当て継続時間はセッションです。オブジェクトは、パラメータ *duration* に指定された確保継続時間の間、確保されます。永続オブジェクトが作成されても、そのオブジェクトがサーバーにフラッシュされるまでは、データベース表へのエントリは行われません。

一時オブジェクト

オブジェクトが確保されます。割当て継続時間および確保継続時間は、パラメータ *duration* によって指定されます。

値

割当て継続時間は、パラメータ *duration* によって指定されます。

新規オブジェクトの属性値

デフォルトでは、新規作成されたオブジェクトのすべての属性に NULL 値があります。属性データを初期化した後で、各属性の対応する NULL ステータスを非 NULL に変更する必要があります。

オブジェクトの作成時に、属性を非 NULL 値に設定できます。これは、*OCIAttrSet()* を使って環境ハンドルの *OCI_OBJECT_NEWNOTNULL* 属性を TRUE に設定することにより実行できます。その後、この属性を FALSE に設定することにより、このモードを無効にすることができます。*OCI_OBJECT_NEWNOTNULL* が TRUE に設定されている場合は、*OCIObjectNew()* によって非 NULL オブジェクトが作成されます。詳細は、10-30 ページの「[新しいオブジェクトの属性値](#)」を参照してください。

LOB 属性を持つオブジェクト

オブジェクトに内部 LOB 属性がある場合、LOB は Empty 値に設定されます。LOB へのデータ書き込みを開始する前に、そのオブジェクトに使用済みマークを設定して（オブジェクトを表に挿入するために）フラッシュし、再確保しなければなりません。作成後にオブジェクトを確保するときは、新しく更新された LOB ロケータをサーバーから検索するために、*OCI_PIN_LATEST* 確保オプションを使用する必要があります。

オブジェクトに外部 LOB 属性（FILE）がある場合は、FILE ロケータが割り当てられますが、初期化はされません。オブジェクトをデータベースにフラッシュする前に、*OCILobFileSetName()* をコールして FILE 属性を初期化する必要があります。最初にディレクトリ別名およびファイル名を指定しないで FILE の INSERT または UPDATE を実行すると、エラーが発生します。ファイル名を設定したら、FILE の読み込みを開始できます。

注意 : Oracle8i では、バイナリ FILE (BFILE) だけをサポートします。

関連関数

[OCIObjectPinTable\(\)](#)、[OCIObjectFree\(\)](#)

OCIObjectSetAttr()

用途

オブジェクト属性を設定します。

構文

```

sword OCIObjectSetAttr ( OCIEnv          *env,
                        OCIError       *err,
                        dvoid          *instance,
                        dvoid          *null_struct,
                        struct OCIType *tdo,
                        CONST text     **names,
                        CONST ub4      *lengths,
                        CONST ub4      name_count,
                        CONST ub4      *indexes,
                        CONST ub4      index_count,
                        CONST OCIInd   null_status,
                        CONST dvoid    *attr_null_struct,
                        CONST dvoid    *attr_value );

```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

instance (IN)

オブジェクトのインスタンスへのポインタです。

null_struct (IN)

オブジェクトのインスタンスまたは配列の NULL 構造体です。

tdo (IN)

TDO へのポインタです。

names (IN)

属性名の配列です。パス式に属性名を指定するために使用されます。

lengths (IN)

属性名長の配列です。

name_count (IN)

配列名の要素の数です。

indexes (IN) [optional]

現行ではサポートされていません。(ub4 *)0 を渡します。

index_count (IN) [optional]

現行ではサポートされていません。(ub4)0 を渡します。

attr_null_status (IN)

属性タイプがプリミティブの場合、属性は NULL ステータスです。

attr_null_struct (IN)

オブジェクトまたは収集属性の NULL 構造です。

attr_value (IN)

属性値へのポインタです。

コメント

この関数は、所定のオブジェクトの属性に所定の値を設定します。属性の位置は名前配列および索引配列であるパス式で指定されます。

例

パス式 stanford.cs.stu[5].addr の場合、配列は次のようになります。

```
names = {"stanford", "cs", "stu", "addr"}
```

```
lengths = {8, 2, 3, 4}
```

```
indexes = {5}
```

関連関数

[*OCIObjectMarkDelete\(\)*](#)

OCI 確保、確保解除および解放関数

この項では、OCI 確保、確保解除および解放関数について説明します。

表 16-6 OCI 確保、確保解除および解放関数クイック・リファレンス

関数	用途	ページ
OCICacheFree()	キャッシュ内のオブジェクトを解放します。	16-48 ページ
OCICacheUnpin()	キャッシュまたは接続で永続オブジェクトを確保解除します。	16-49 ページ
OCIObjectArrayPin()	参照の配列を確保します。	16-50 ページ
OCIObjectFree()	割当て済みのオブジェクトを解放します。	16-52 ページ
OCIObjectPin()	オブジェクトを確保します。	16-54 ページ
OCIObjectPinCountReset()	オブジェクトを確保解除して確保カウントをゼロにします。	16-57 ページ
OCIObjectPinTable()	継続時間を指定して表オブジェクトを確保します。	16-59 ページ
OCIObjectUnpin()	オブジェクトを確保解除します。	16-61 ページ

OCI`CacheFree()`

用途

指定された接続に対してキャッシュ内のオブジェクトと値をすべて解放します。

構文

```
sword OCICacheFree ( OCIEnv          *env,  
                    OCIError       *err,  
                    CONST OCISvcCtx *svc );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「`OCIEnvCreate()`」および「`OCIInitialize()`」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、`err` に記録され、`OCI_ERROR` が戻されます。`OCIErrorGet()` をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキストです。

コメント

接続が指定された場合、この関数は、その接続に対して割り当てられている永続オブジェクトおよび一時オブジェクト値を解放します。それ以外の場合は、オブジェクト・キャッシュ内の永続オブジェクトおよび一時オブジェクト値をすべて解放します。オブジェクトは、確保カウントに関係なく解放されます。

インスタンスの解放の詳細は、`OCIObjectFree()` を参照してください。

関連関数

[OCIObjectFree\(\)](#)

OCICacheUnpin()

用途

永続オブジェクトの確保を解除します。

構文

```
sword OCICacheUnpin ( OCIEnv          *env,  
                     OCIError       *err,  
                     CONST OCISvcCtx *svc );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキスト・ハンドルです。指定された接続のオブジェクトの確保が解除されます。

コメント

この関数は、所定の接続に対するすべての永続オブジェクトの確保を完全に解除します。オブジェクトの確保カウントは 0 (ゼロ) にリセットされます。

確保および確保解除の詳細は、10-11 ページの「[オブジェクトの確保](#)」および 10-27 ページの「[確保カウントおよび確保解除](#)」を参照してください。

関連関数

[OCIObjectUnpin\(\)](#)

OCIObjectArrayPin()

用途

参照配列を確保します。

構文

```
sword OCIObjectArrayPin ( OCIEnv          *env,
                          OCIError       *err,
                          OCIRef         **ref_array,
                          ub4            array_size,
                          OCIComplexObject **cor_array,
                          ub4            cor_array_size,
                          OCIPinOpt      pin_option,
                          OCIDuration    pin_duration,
                          OCILockOpt     lock,
                          dvoid          **obj_array,
                          ub4            *pos );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

ref_array (IN)

確保される参照の配列です。

array_size (IN)

参照の配列にある要素の数です。

cor_array

確保されるオブジェクトに対応する COR ハンドルの配列です。

cor_array_size

cor_array にある要素の数です。

pin_option (IN)

確保オプションです。16-54 ページの「*OCIObjectPin()*」を参照してください。

pin_duration (IN)

確保継続時間です。「*OCIObjectPin()*」を参照してください。

lock (IN)

ロック・オプションです。「*OCIObjectPin()*」を参照してください。

obj_array (OUT)

この引数が NULL でなければ、確保されたオブジェクトをこの配列に戻します。dvoid * 型の要素をこの配列に割り当てる必要があります。この配列のサイズは *array_size* と同じです。

pos (OUT)

エラーが起きると、この引数によってエラーの原因となった要素が示されます。この引数には、*ref_array* の第 1 要素に対して、1 が設定されます。

コメント

確保されたすべてのオブジェクトが 1 回のネットワーク往復でデータベースから検索されます。ユーザーが出力配列 (*obj_array*) を指定した場合は、確保されたオブジェクトのアドレスが配列内の各要素に割り当てられます。

関連関数

[OCIObjectPin\(\)](#)

OCIObjectFree()

用途

オブジェクト・インスタンスの解放と確保解除を行います。

構文

```
sword OCIObjectFree ( OCIEnv          *env,  
                     OCIError       *err,  
                     dvoid          *instance,  
                     ub2            flags );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

instance (IN)

スタンドアロン・インスタンスに対するポインタです。オブジェクトの場合は、確保されていなければなりません。

flags (IN)

OCI_OBJECTFREE_FORCE が引き渡された場合は、オブジェクトが確保されているか使用済みであっても解放します。OCI_OBJECTFREE_NONULL が引き渡されると、NULL 構造体は解放されません。

コメント

この関数は、オブジェクト・インスタンスのために割り当てられたすべてのメモリの割当てを解除します (NULL 構造体も含まれます)。インスタンスの型によって、次の規則が適用されます。

永続オブジェクト

この関数は、フラッシュされていない使用済み永続オブジェクトをクライアントが解放しようとした場合はエラーを戻します。クライアントは、永続オブジェクトをフラッシュするか、マークを解除するか、*flags* パラメータに OCI_OBJECTFREE_FORCE を設定する必要があります。

この関数は、オブジェクトの確保を完全に解除できるかどうかを調べるために OCIObjectUnpin() を 1 回コールします。成功した場合は、関数の残りを続行しオブジェクト

を解放します。失敗した場合は、*flag* パラメータに `OCI_OBJECTFREE_FORCE` が設定されていない限りエラーが戻ります。

メモリー内の永続オブジェクトを解放しても、そのオブジェクトのサーバー側での永続状態が変化することはありません。たとえば、オブジェクトが解放された後でも、そのオブジェクトはロックされたままになります。

一時オブジェクト

この関数は、オブジェクトの確保を完全に解除できるかどうかを調べるために `OCIObjectUnpin()` を 1 回コールします。成功した場合は、関数の残りを続行しオブジェクトを解放します。失敗した場合は、*flag* パラメータに `OCI_OBJECTFREE_FORCE` が設定されていない限りエラーが戻ります。

値

オブジェクトのメモリーがただちに解放されます。

関連関数

[OCICacheFree\(\)](#)

OCIObjectPin()

用途

参照可能オブジェクトを確保します。

構文

```
sword OCIObjectPin ( OCIEnv          *env,  
                    OCIError       *err,  
                    OCIRef         *object_ref,  
                    OCIComplexObject *corhdl,  
                    OCIPinOpt      pin_option,  
                    OCIDuration     pin_duration,  
                    OCILockOpt     lock_option,  
                    dvoid           **object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

object_ref (IN)

オブジェクトに対する参照です。

corhdl (IN)

複合オブジェクト検索に対するハンドルです。

pin_option (IN)

検索されるオブジェクトのコピーの指定に使用します。

pin_duration (IN)

オブジェクトがクライアントによってアクセスされる継続時間です。オブジェクトの確保は、確保継続時間の終了時点で暗黙のうちに解除されます。OCI_DURATION_NULL が渡された場合は、オブジェクトがすでにキャッシュ内にロードされているときは、確保の促進はありません。オブジェクトが未ロードで OCI_DURATION_NULL の場合、確保継続時間には OCI_DURATION_DEFAULT が設定されます。

lock_option (IN)

ロック・オプション（たとえば、排他的ロック）です。ロック・オプションが指定された場合、オブジェクトはサーバー内でロックされます。オブジェクトのロック・ステータスは、`OCIObjectIsLocked()` 関数をコールすることでも取り出せます。次の値が有効です。

- OCI_LOCK_NONE - ロックなし
- OCI_LOCK_X - 排他的ロック
- OCI_LOCK_X_NOWAIT - NOWAIT オプションを使用した排他ロック

関連項目 : NOWAIT オプションの詳細は、13-13 ページの「[NOWAIT オプションを使ったロック](#)」を参照してください。

object (OUT)

確保されたオブジェクトに対するポインタです。

コメント

この関数は、オブジェクト参照によって指示された参照可能オブジェクト・インスタンスを確保します。この確保のプロセスで、次の2つの目的を達成します。

- 参照によって指示されたオブジェクトの場所を特定します。これは、オブジェクト・キャッシュ内のオブジェクトを追跡し記録するオブジェクト・キャッシュによって実行されます。
- 永続オブジェクトは使用中のため、エイジングが不可であることをオブジェクト・キャッシュに通知します。永続オブジェクトは必要に応じていつでもサーバーからロードできるため、確保が完全に解除された永続オブジェクトを割当て継続時間の経過前に解放する（エイジングさせる）ことができる場合は、利用できるメモリーを増やすことができます。オブジェクトは複数回確保できます。確保されたオブジェクトは、完全に確保解除されるまでメモリー内に残ります。16-54 ページの「[OCIObjectPin\(\)](#)」を参照してください。

確保解除の詳細は、「[OCIObjectUnpin\(\)](#)」を参照してください。

永続オブジェクト

永続オブジェクトを確保したときにそのオブジェクトがキャッシュ内に存在しない場合は、永続ストアからフェッチされます。オブジェクトの割当て継続時間はセッションです。オブジェクトがキャッシュ内に存在する場合は、そのオブジェクトがクライアントに戻されず。ロック・オプションが指定されている場合、オブジェクトは、サーバー内でロックされます。

この関数は、存在しないオブジェクトに対してはエラーを戻します。

確保オプションを使用して、検索されるオブジェクトのコピーを指定します。

- `pin_option` が `OCI_PIN_ANY` (pin any) であり、オブジェクトがすでにオブジェクト・キャッシュ内にある場合は、そのオブジェクトが戻されます。それ以外の場合、オブジェクトはデータベースから検索されます。これは、`pin_option` が `OCI_PIN_LATEST`

である場合と同じです。このオプションは、セッションのデータに排他的アクセス権を持っているのを、クライアントが認識しているときに有用です。

- *pin_option* が OCI_PIN_LATEST (pin latest) であると、オブジェクトがロックされていなければデータベースから取り出されます。オブジェクトがキャッシュ処理されている場合は、最新バージョンによってリフレッシュされます。リフレッシュの詳細は、「OCIObjectRefresh()」を参照してください。
- *pin_option* が OCI_PIN_RECENT (pin recent) であり、オプションが現行のトランザクションでキャッシュにロードされた場合は、そのオブジェクトが戻されます。オブジェクトが現行のトランザクション中にロードされていない場合は、オブジェクトはサーバーからリフレッシュされます。

一時オブジェクト

この関数は、一時オブジェクトがすでに解放されている場合はエラーを戻します。ロック・オプションに排他ロックが指定されている場合は、エラーを戻しません。

関連関数

[OCIObjectUnpin\(\)](#)、[OCIObjectPinCountReset\(\)](#)

OCIObjectPinCountReset()

用途

オブジェクトの確保を完全に解除し、その確保カウントを 0 (ゼロ) に設定します。

構文

```
sword OCIObjectPinCountReset ( OCIEnv      *env,  
                               OCIError    *err,  
                               dvoid       *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

object (IN)

オブジェクトへのポインタで、それがすでに確保されている必要があります。

コメント

この関数は、オブジェクトの確保を完全に解除し、その確保カウントに 0 (ゼロ) を設定します。オブジェクトの確保が完全に解除されると、OCI は、そのオブジェクトをエラーなしにいつでも暗黙的に解放します。オブジェクトの型によって次の規則が適用されます。

永続オブジェクト

永続オブジェクトの確保が完全に解除された場合、そのオブジェクトはエイジングの対象になります。オブジェクトのメモリーは、オブジェクトのエイジングが終了したときに解放されます。エイジングは、メモリーを最大限に利用するために使用されます。使用済みオブジェクトは、それがフラッシュされるまでエイジングを終了できません。

一時オブジェクト

オブジェクトの確保カウントが減分されます。一時オブジェクトは、その割当て継続時間が経過したとき、または OCIObjectFree() をコールすることで明示的に解放されたときだけ解放できます。

値

この関数は、値に対してはエラーを戻します。

この関数の使用方法の詳細は、10-27 ページの「[確保カウントおよび確保解除](#)」を参照してください。

OCIObjectPinCountReset()

関連関数

OCIObjectPin()、*OCIObjectUnpin()*

OCIObjectPinTable()

用途

表オブジェクトを指定された継続時間の間、確保します。

構文

```
sword OCIObjectPinTable ( OCIEnv          *env,  
                          OCIError       *err,  
                          CONST OCISvcCtx *svc,  
                          CONST text     *schema_name,  
                          ub4           s_n_length,  
                          CONST text     *object_name,  
                          ub4           o_n_length,  
                          dvoid         *not_used,  
                          OCIDuration   pin_duration,  
                          dvoid         **object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキスト・ハンドルです。

schema_name (IN) [optional]

表のスキーマ名です。

s_n_length (IN) [optional]

schema_name に指定されたスキーマ名の長さです。

object_name (IN)

表の名前です。

o_n_length (IN)

object_name に指定された表名の長さです。

not_used (IN/OUT)

このパラメータは使われていません。NULL を渡します。

pin_duration (IN)

確保継続時間です。16-54 ページの「[OCIObjectPin\(\)](#)」の説明を参照してください。

object (OUT)

確保されたオブジェクト表です。

コメント

この関数は、表オブジェクトを指定の継続時間の間、確保します。クライアントは、[OCIObjectUnpin\(\)](#) をコールすることでこのオブジェクトの確保を解除できます。

このコールによって確保されるオブジェクト表は、スタンドアロン型の永続オブジェクトを作成するために、パラメータとして [OCIObjectNew\(\)](#) に渡すことができます。

関連関数

[OCIObjectPin\(\)](#)、[OCIObjectUnpin\(\)](#)

OCIObjectUnpin()

用途

オブジェクトの確保を解除します。

構文

```
sword OCIObjectUnpin ( OCIEnv      *env,  
                      OCIError    *err,  
                      dvoid        *object );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

object (IN)

オブジェクトへのポインタで、それがすでに確保されている必要があります。

コメント

各オブジェクトには、そのオブジェクトが確保されるたびに増分される確保カウントが関連付けられています。オブジェクトの確保カウントが 0 (ゼロ) になったとき、そのオブジェクトの確保は完全に解除されたと表現されます。確保が解除されたオブジェクトは、OCI によって、エラーなしにいつでも暗黙的に解放されます。

この関数は、オブジェクトの確保を解除します。次のいずれかが true であるとき、オブジェクトの確保は完全に解除されています。

1. オブジェクトの確保カウントが 0 (ゼロ) に達しています (つまり、全部で N 回確保された後に全部で N 回確保解除されています)。
2. オブジェクトの確保継続時間が終了しています。
3. オブジェクトに対して OCIObjectPinCountReset() 関数がコールされました。

オブジェクトの確保が完全に解除されると、OCI は、そのオブジェクトをエラーなしにいつでも暗黙的に解放します。

オブジェクトの型によって次の規則が適用されます。

永続オブジェクト

永続オブジェクトの確保が完全に解除された場合、そのオブジェクトはエイジングの対象になります。オブジェクトのメモリーは、オブジェクトのエイジングが終了したときに解放されます。エイジングは、メモリーを最大限に利用するために使用されます。使用済みオブジェクトは、それがフラッシュされるまでエイジングを終了できません。

一時オブジェクト

オブジェクトの確保カウントが減分されます。一時オブジェクトは、その割当て継続時間が終わったとき、または *OCIObjectFree()* をコールすることで明示的に削除したときだけ解放できます。

値

この関数は値に対してはエラーを戻します。

関連関数

OCIObjectPin()、*OCIObjectPinCountReset()*

OCI 型情報アクセサ関数

この項では、OCI 型情報アクセサ関数について説明します。

表 16-7 OCI 型情報アクセサ関数クイック・リファレンス

関数	用途	ページ
OCITypeArrayByName()	オブジェクト名配列を指定して TDO 配列を取得します。	16-64 ページ
OCITypeArrayByRef()	オブジェクト参照配列を指定して TDO 配列を取得します。	16-67 ページ
OCITypeByName()	オブジェクト名を指定して TDO を取得します。	16-69 ページ
OCITypeByRef()	オブジェクト参照を指定して TDO を取得します。	16-71 ページ

OCITypeArrayByName()

用途

名前配列を指定して型配列を取得します。

構文

```
sword OCITypeArrayByName ( OCIEnv          *envhp,  
                           OCIError       *errhp,  
                           CONST OCISvcCtx *svc,  
                           ub4            array_len,  
                           CONST text     *schema_name[],  
                           ub4            s_length[],  
                           CONST text     *type_name[],  
                           ub4            t_length[],  
                           CONST text     *version_name[],  
                           ub4            v_length[],  
                           OCIDuration    pin_duration,  
                           OCITypeGetOpt  get_option,  
                           OCIType       *tdo[] );
```

パラメータ

envhp (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

errhp (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

svc (IN)

OCI サービス・ハンドルです。

array_len (IN)

取り出される *schema_name/type_name/version_name* エントリの数です。

schema_name (IN, optional)

取り出される型に関連付けられているスキーマ名の配列です。この配列を指定する場合は、*array_len* 個の要素が必要です。0 (ゼロ) を指定した場合はデフォルト・スキーマであると見なされます。それ以外の場合は、*array_len* 個の要素が必要です。デフォルト・スキーマが必要であることを指示するために 1 つ以上のエントリに 0 (ゼロ) を指定できます。

s_length (IN)

schema_name 長さの配列で、それぞれのエントリが、*schema_name* 配列の対応する *schema_name* エントリのバイト単位の長さに対応しています。この配列には、*array_len* 個の要素があるか、*schema_name* が指定されない場合は 0 (ゼロ) でなければなりません。

type_name (IN)

取り出される型の名前の配列です。*array_len* 個の要素が必要です。

t_length (IN)

type_name 配列にある型名のバイト単位の長さの配列です。

version_name (IN)

取り出される型に対応するバージョン名の配列です。これは、最新の現バージョンの検索を示す 0 (ゼロ) にするか、*array_len* 個の要素が必要です。

0 (ゼロ) が指定された場合は最新の現バージョンと見なされます。それ以外の場合は、*array_len* 個の要素が必要です。現バージョンが必要であることを指示するために複数のエントリに 0 (ゼロ) を指定できます。

注意: リリース 8.0 では、バージョン・パラメータは無視されます。

v_length (IN)

version_name 配列にあるバージョン名のバイト単位の長さの配列です。

注意: リリース 8.0 では、バージョン・パラメータは無視されます。

pin_duration (IN)

取り出された型用の確保継続時間 (たとえば、現行のトランザクションの終わりまで) です。各オプションの説明は、*oro.h* を参照してください。

get_option (IN)

型をロードするためのオプションは、次のどちらかの値です。

- OCI_TYPEGET_HEADER - ロードされるヘッダー用のみ
- OCI_TYPEGET_ALL - TDO、およびロードされるすべての ADO と MDO 用

tdo (OUT)

オブジェクト・キャッシュに確保された各型へのポインタ用出力配列です。これには、*array_len* ポインタ分の領域が必要です。OCIObjectGetObjectRef() を使用して確保された各型記述子に対する CREF を取得します。

コメント

スキーマ / 型名配列に関連付けられている既存の型のポインタを取得します。

get_option パラメータを使用すると、往復ごとにロードされる TDO の部分を制御できます。

この関数は、必須パラメータに NULL がある場合、あるいはスキーマ名または型名に関連付けられたオブジェクト型が存在しない場合はエラーを戻します。

配列ではなく単一の型を取り出すには、*OCITypeByName()* を使用します。

関連関数

[OCITypeArrayByRef\(\)](#)、[OCITypeByName\(\)](#)、[OCITypeByRef\(\)](#)

OCITypeArrayByRef()

用途

参照配列を指定して型配列を取得します。

構文

```
sword OCITypeArrayByRef ( OCIEnv          *envhp,
                          OCIError       *errhp,
                          ub4            array_len,
                          CONST OCIRef   *type_ref [],
                          OCIDuration    pin_duration,
                          OCITypeGetOpt  get_option,
                          OCIType        *tdo [] );
```

パラメータ

envhp (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

errhp (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

array_len (IN)

取り出される schema_name/type_name/version_name エントリの数です。

type_ref (IN)

取り出される型記述子オブジェクトの特定のバージョンを指し示す OCIRef* の配列です。この配列を指定する場合は、*array_len* 個の要素が必要です。

pin_duration (IN)

取り出された型用の確保継続時間（たとえば、現行のトランザクションの終わりまで）です。各オプションの説明は、'oro.h' を参照してください。

get_option (IN)

型をロードするためのオプションは、次のどちらかの値です。

- OCI_TYPEGET_HEADER - ロードされるヘッダーのみ
- OCI_TYPEGET_ALL - TDO、およびロードされるすべての ADO と MDO 用

tdo (OUT)

オブジェクト・キャッシュに確保された各型へのポインタ用出力配列です。これには、*array_len* ポインタ分の領域が必要です。OCIObjectGetObjectRef() を使用して確保された各型記述子に対する CREF を取得します。

コメント

スキーマ / 型名配列に関連付けられている既存の型のポインタを取得します。

この関数は、次の場合にエラーを戻します。

- 必須パラメータに NULL がある場合
- スキーマ / 型名エントリに関連付けられている 1 つ以上のオブジェクト型が存在しない場合

型の配列ではなく、単一の型を取り出すには、OCITypeByName() を使用します。

関連関数

[OCITypeArrayByName\(\)](#)、[OCITypeByRef\(\)](#)、[OCITypeByName\(\)](#)

OCITypeByName()

用途

既存の型の最新バージョンを名前を指定して取得します。

構文

```
sword OCITypeByName ( OCIEnv          *env,
                     OCIError       *err,
                     CONST OCISvcCtx *svc,
                     CONST text     *schema_name,
                     ub4            s_length,
                     CONST text     *type_name,
                     ub4            t_length,
                     CONST text     *version_name,
                     ub4            v_length,
                     OCIDuration    pin_duration,
                     OCITypeGetOpt  get_option,
                     OCIType        **tdo );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

svc (IN)

OCI サービス・ハンドルです。

schema_name (IN, optional)

型に関連付けられているスキーマ名です。デフォルトでは、ユーザーのスキーマ名が使用されます。

s_length (IN)

schema_name パラメータの長さです。

type_name (IN)

取得する型の名前です。

t_length (IN)

type_name パラメータの長さです。

version_name (IN, optional)

ユーザーが読み出し可能な、型のバージョンです。最新のバージョンを取り出すには (text *) 0 を渡します。リリース 8.0 では、シングル・バージョンだけがサポートされています。

v_length (IN)

version_name のバイト単位の長さです。最新のバージョンを取り出す場合は 0 (ゼロ) を渡します。

pin_duration (IN)

確保継続時間です。詳細は、13-14 ページの「[オブジェクトの継続時間](#)」を参照してください。

get_option (IN)

型をロードするためのオプションは、次のどちらかの値です。

- OCI_TYPEGET_HEADER - ヘッダーだけロードします。
- OCI_TYPEGET_ALL - TDO、およびすべての ADO と MDO をロードします。

tdo (OUT)

オブジェクト・キャッシュに確保された型へのポインタです。

コメント

この関数は、スキーマ / 型名に関連付けられている既存の型へのポインタを取得します。必須パラメータに NULL がある場合、またはスキーマ / 型名に関連付けられたオブジェクト型が存在しない場合はエラーを戻します。

注意: スキーマ名と型名は大文字小文字を区別します。SQL を使って作成された場合は、大文字の名前を使用する必要があります。

[OCTypeArrayByName\(\)](#) または [OCTypeArrayByRef\(\)](#) をコールすることにより、アプリケーションで TDO の配列を取り出せます。

関連関数

[OCTypeByRef\(\)](#)、[OCTypeArrayByName\(\)](#)、[OCTypeArrayByRef\(\)](#)

OCITypeByRef()

用途

参照を指定して型を取得します。

構文

```
sword OCITypeByRef ( OCIEnv          *env,  
                    OCIError       *err,  
                    CONST OCIRef    *type_ref,  
                    OCIDuration     pin_duration,  
                    OCITypeGetOpt   get_option,  
                    OCIType         *tdo );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

type_ref (IN)

取得する型記述子オブジェクトのバージョンを指す *OCIRef** です。

pin_duration (IN)

取り出す型に対する、現行のトランザクションの終了までの確保継続時間です。各オプションの説明は、'oro.h' を参照してください。

get_option (IN)

型をロードするためのオプションは、次のどちらかの値です。

- *OCI_TYPEGET_HEADER* - ロードされるヘッダー用のみ
- *OCI_TYPEGET_ALL* - TDO、およびロードされるすべての ADO と MDO 用

tdo (OUT)

オブジェクト・キャッシュに確保された型へのポインタです。

コメント

OCITypeByRef() は、必須パラメータに NULL がある場合、またはスキーマ / 型名に関連付けられたオブジェクト型が存在しない場合はエラーを戻します。

OCITypeByRef()

関連関数

[OCITypeByName\(\)](#)、[OCITypeArrayByName\(\)](#)、[OCITypeArrayByRef\(\)](#)

OCI データ型マッピング関数および操作関数

この章では、OCI データ型マッピング関数および操作関数について説明します。これらは Oracle 事前定義型に対する、Oracle の外部 C 言語インタフェースです。この章には、次の項目が含まれています。

- [概要](#)
- [OCI コレクションおよび反復関数](#)
- [OCI 日付関数](#)
- [OCI 数関数](#)
- [OCI ロー関数](#)
- [OCI REF 関数](#)
- [OCI 文字列関数](#)
- [OCI 表関数](#)

注意：この章で説明する関数は、Oracle8i Enterprise Edition をオブジェクト・オプションとともにインストールした場合にだけ使用可能です。

概要

この章では、OCI データ型マッピングおよび操作関数の詳細を説明します。

関連項目 : この章にリストしている関数の詳細は、[第 11 章の「オブジェクト・リレーショナル・データ型」](#)を参照してください。

関数の構文

関数ごとに次の情報が記載されています。

用途

関数の用途に関する簡単な説明。

構文

この関数をコールするための構文を示すコードの断片で、パラメータの順序と種類が含まれています。

コメント

この関数に関する詳細情報（ある場合）、関数の使用上の制約やアプリケーション内でこの関数を使用するときに役に立つ情報が記載されています。

パラメータ

この関数の各パラメータの説明。これにはパラメータのモードが含まれます。引数のモードには、以下のように 3 つの可能な値があります。

モード	説明
IN	Oracle にデータを渡すパラメータ
OUT	このコールまたは後続のコールで Oracle からデータを受け取るパラメータ
IN/OUT	このコールでデータを渡し、このコールまたは後続のコールからの戻りでデータを受け取るパラメータ

戻り値

17-3 ページの表 17-1 の「[関数の戻り値](#)」にリストされた標準リターン・コード以外の値が関数によって戻される場合の、戻り値についての説明。

関連関数

関連する関数の一覧。

データ型マッピング関数および操作関数の戻り値

OCI データ型マッピング関数および操作関数は、通常、次の値の 1 つを戻します。

表 17-1 関数の戻り値

戻り値	意味
OCI_SUCCESS	操作は成功しました。
OCI_ERROR	操作は失敗しました。関数に渡されたエラー・ハンドルに対して <i>OCIErrorGet()</i> をコールすることで特定のエラーを検索できます。
OCI_INVALID_HANDLE	関数に渡された環境ハンドルまたはエラー・ハンドルが NULL です。

この章では、各関数に関する記述の後で関数固有の戻り値についての情報を説明します。リターン・コードおよびエラー処理の詳細は、2-26 ページの「[エラー処理](#)」を参照してください。

その他の値を戻す関数

一部の関数は、表 17-1 にリストされていない値を戻します。これらの関数を使用する場合は、OUT パラメータからではなく、ファンクション・コールから直接値が戻されることを必ず考慮してください。

- `OCICollMax()`
- `OCIRawPtr()`
- `OCIRawSize()`
- `OCIRefHexSize()`
- `OCIRefsEqual()`
- `OCIRefsNull()`
- `OCIStringPtr()`
- `OCIStringSize()`

データ型マッピング関数および操作関数のサーバー往復

個々の OCI データ型マッピング関数および操作関数に必要なサーバー往復回数を示す表は、付録 C の「[OCI 関数のサーバー・ラウンドトリップ](#)」を参照してください。

例

コード例など、これらの関数の詳細は、[第 11 章の「オブジェクト・リレーショナル・データ型」](#)を参照してください。

OCI コレクションおよび反復関数

この項では、コレクションおよび反復関数について説明します。

表 17-2 OCI コレクションおよび反復関数クイック・リファレンス

関数	用途	ページ
OCICollAppend()	コレクションに要素を追加します。	17-6 ページ
OCICollAssign()	コレクションを割り当てます。	17-7 ページ
OCICollAssignElem()	コレクションに要素を割り当てます。	17-8 ページ
OCICollGetElem()	要素を指すポインタを取得します。	17-10 ページ
OCICollIsLocator()	コレクションがロケータベースであるかどうかを示します。	17-13 ページ
OCICollMax()	コレクションの最大要素数を戻します。	17-14 ページ
OCICollSetUpdateStatus()	コレクションの更新状況を設定します。	17-15 ページ
OCICollSize()	コレクションの現在のサイズ（要素数単位）を取得します。	17-16 ページ
OCICollTrim()	コレクションから要素を切り捨てます。	17-18 ページ
OCIIterCreate()	varray 要素をスキャンするためのイテレータを作成します。	17-19 ページ
OCIIterDelete()	イテレータを削除します。	17-20 ページ
OCIIterGetCurrent()	現在のコレクション要素を取得します。	17-21 ページ
OCIIterInit()	イテレータを初期化して指定のコレクションをスキャンします。	17-22 ページ
OCIIterNext()	次のコレクション要素を取得します。	17-23 ページ
OCIIterPrev()	前のコレクション要素を取得します。	17-25 ページ

OCICollAppend()

用途

コレクションの最後に要素を追加します。

構文

```
sword OCICollAppend ( OCIEnv          *env,  
                     OCIError       *err,  
                     CONST dvoid     *elem,  
                     CONST dvoid     *elemind,  
                     OCIColl         *coll );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

elem (IN)

与えられたコレクションの最後に追加される要素へのポインタ。

elemind (IN) [optional]

要素の NULL 標識情報へのポインタ。(*elemind* == NULL) の場合、追加された要素の NULL 標識情報は、非 NULL に設定されます。

coll (IN/OUT)

更新されたコレクション。

コメント

要素の追加は、コレクションのサイズを 1 要素分増やし、最終要素のデータを指定の要素のデータで更新（ディープコピー）することと等価です。指定の要素 *elem* へのポインタはこの関数によって保存されないことに注意してください。つまり、*elem* は厳密に入力パラメータです。

この関数は、コレクションの現在のサイズが、要素を追加する前のコレクションの最大サイズ（上限）と等しい場合、エラーを戻します。また、入力パラメータに NULL がある場合にもエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)

OCICollAssign()

用途

あるコレクションを別のコレクションに割り当てます。(ディープコピーします。)

構文

```
sword OCICollAssign ( OCIEnv          *env,  
                     OCIError       *err,  
                     CONST OCIColl   *rhs,  
                     OCIColl        *lhs );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

rhs (IN)

割当て元となる右側 (ソース) コレクション。

lhs (OUT)

割当て先となる左側 (ターゲット) コレクション。

コメント

rhs (ソース) を *lhs* (ターゲット) に割り当てます。*lhs* コレクションは、*rhs* のサイズに応じて変更されます。*lhs* に要素が格納されている場合は、割当てに先立ってその要素が削除されます。この関数はディープコピーを実行します。要素用のメモリーはオブジェクト・キャッシュから取られます。

lhs コレクションと *rhs* コレクションの要素の型が一致しない場合はエラーが戻ります。*lhs* の上限が *rhs* コレクション内の現在の要素数より小さい場合もエラーが戻ります。次の場合にもエラーが戻されます。

- 入力パラメータに NULL がある
- *lhs* コレクションと *rhs* コレクションの型が一致しない
- *lhs* コレクションの上限が *rhs* コレクション内の現在の要素数より小さい

関連関数

[OCIErrorGet\(\)](#)、[OCICollAssignElem\(\)](#)

OCICollAssignElem()

用途

指定の要素値 *elem* を *coll[Index]* の要素に割り当てます。

構文

```
sword OCICollAssignElem ( OCIEnv          *env,  
                          OCIError       *err,  
                          sb4             index,  
                          CONST dvoid     *elem,  
                          CONST dvoid     *elemind,  
                          OCIColl        *coll );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

index (IN)

割当て先となる要素の索引。

elem (IN)

割当て元となる要素 (ソース要素)。

elemind (IN) [optional]

要素の NULL 標識情報へのポインタ。(*elemind* == NULL) の場合、割り当てられた要素の NULL 標識情報は、非 NULL に設定されます。

coll (IN/OUT)

更新されるコレクション。

コメント

コレクションの型がネストした表である場合、要素が削除されているときと同様に、指定の索引位置に要素が存在しない場合があります。このような場合は、*index* の位置に指定の要素が挿入されます。それ以外の場合は、*index* の位置にある要素が *elem* の値で更新されます。

指定の要素はディープコピーされること、*elem* は厳密に入力パラメータであることに注意してください。

この関数は、入力パラメータに NULL があるか、指定の索引が指定のコレクションの上限を超えている場合にエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCICollAssign\(\)](#)

OCICollGetElem()

用途

指定の索引位置にある要素のポインタを取得します。

構文

```
sword OCICollGetElem ( OCIEnv          *env,  
                      OCIError        *err,  
                      CONST OCIColl    *coll,  
                      sb4              index,  
                      boolean          *exists,  
                      dvoid            **elem,  
                      dvoid            **elemind );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

coll (IN)

このコレクション内の要素のポインタが戻されます。

index (IN)

ポインタが戻される要素の索引。

exists (OUT)

指定した索引の要素が存在しない場合は FALSE に設定されます。そうでない場合は TRUE に設定されます。

elem (OUT)

要求する要素のアドレスが戻されます。

elemind (OUT) [optional]

NULL 標識情報のアドレスが戻されます。(*elemind* == NULL) の場合、NULL 標識情報は戻されません。

コメント

指定の位置にある要素のアドレスを取得します。また、この関数は、オプションで要素の NULL 標識情報のアドレスを戻します。

コレクションの各要素の型と対応する要素ポインタの型を次の表に示します。要素ポインタは `OCICollGetElem()` の `elem` パラメータによって戻されます。

要素型	*elem の設定
Oracle 数値 (OCINumber)	OCINumber*
日付 (OCIDate)	OCIDate*
可変長文字列 (OCIStrng*)	OCIStrng**
可変長 raw (CIRaw*)	OCIRaw**
オブジェクト参照 (OCIStrng*)	OCIStrng**
LOB ロケータ (OCILobLocator*)	OCILobLocator**
オブジェクト型 (person など)	person*

`OCICollGetElem()` で戻される要素ポインタは、要素データにアクセスするために使用できる書式だけでなく、割当て文のターゲットとして使用できる書式の場合もあります。

たとえば、要素型がオブジェクト参照 (OCIStrng*) であるコレクションの要素を反復するとします。`OCICollGetElem()` のコールにより、参照ハンドル (OCIStrng**) のポインタが戻ります。コレクション要素のポインタを取得した後、新しい参照を割り当てることでそれを修正できます。

これは、次のような REF 割当て関数によって実行できます。

```
sword OCIStrngAssign( OCIStrng      *env,
                     OCIStrng      *err,
                     CONST OCIStrng *source,
                     OCIStrng      **target );
```

`OCIStrngAssign()` の `target` パラメータの型は `OCIStrng**` であることに注意してください。したがって `OCICollGetElem()` は、`OCIStrng**` を戻します。`*target` が NULL の場合は、`OCIStrngAssign()` によって新しい REF が割り当てられ、`target` パラメータを介して戻されます。

同様に、コレクション要素がタイプ文字列 (OCIStrng*) のものである場合、`OCICollGetElem()` はポインタを文字列ハンドル (つまり `OCIStrng**`) に戻します。`OCIStrngAssign()` または `OCIStrngAssignText()` を介して新しい文字列を割り当てる場合は、ターゲットの型は `OCIStrng**` でなければなりません。

コレクションの要素型が Oracle 数値の場合、`OCICollGetElem()` は `OCINumber*` を戻します。`OCINumberAssign()` のプロトタイプを次に示します。

```
sword OCINumberAssign(OCIStrng      *err,
                     CONST OCINumber *from,
                     OCINumber      *to);
```

この関数は、入力パラメータに NULL がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCICollAssignElem\(\)](#)

OCICollIsLocator()

用途

コレクションがロケータベースであるかどうかを示します。

構文

```
sword OCICollIsLocator ( OCIEnv *env,  
                        OCIError *err,  
                        CONST OCIColl *coll,  
                        boolean *result );
```

パラメータ

env (IN)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

coll (IN)

コレクション項目。

result (OUT)

コレクション項目がロケータベースである場合は TRUE を戻します。それ以外の場合は FALSE を戻します。

コメント

この関数は、コレクションがロケータベースであるかどうかを確認するためのテストを行います。コレクション項目がロケータベースである場合は *result* パラメータに TRUE を戻します。それ以外の場合は FALSE を戻します。

関連関数

[OCIErrorGet\(\)](#)

OCICollMax()

用途

指定のコレクションの最大サイズを要素数単位で取得します。

構文

```
sb4 OCICollMax ( OCIEnv          *env,  
                 CONST OCIColl   *coll );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

coll (IN)

要素数が戻されるコレクション。coll は有効なコレクション記述子を指していなければなりません。

コメント

指定されたコレクションが保持できる要素の最大数を戻します。値 0 (ゼロ) は、そのコレクションには上限がないことを示します。

戻り値

指定のコレクションの上限。

関連関数

[OCIErrorGet\(\)](#)、[OCICollSize\(\)](#)

OCICollSetUpdateStatus()

用途

コレクションの更新状況を設定します。

構文

```
sword OCICollSetUpdateStatus ( OCIEnv *env,  
                               OCIError *err,  
                               OCIColl *coll,  
                               ub1 status );
```

パラメータ

env (IN)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

coll (IN)

更新状況が設定されるコレクション。

status (IN)

コレクションの状況。可能な値は次のとおりです。

- OCICOLL_DIRTY - 更新マークの設定
- OCICOLL_NOT_DIRTY - 非使用済みマークの設定

コメント

この関数は、コレクションの状況を設定して、コレクションに更新（使用済み）マークを設定する必要があるかどうか、または非使用済みマークを設定する必要があるかどうかを示します。

関連関数

[OCIErrorGet\(\)](#)

OCICollSize()

用途

指定のコレクションの現在のサイズを要素数単位で取得します。

構文

```
sword OCICollSize ( OCIEnv          *env,
                   OCIError        *err,
                   CONST OCIColl    *coll
                   sb4              *size );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

coll (IN)

要素の数が戻されるコレクション。有効なコレクション識別子を指していなければなりません。

size (OUT)

コレクション内の現在の要素数。

コメント

指定されたコレクションの現在の要素数を戻します。ネストした表の場合、要素を削除してもこのカウントは減分されません。したがって、このカウントには要素の削除によって発生した空きが含まれています。切捨て操作 (*OCICollTrim()*) は、切り捨てられた要素の数だけこのカウントを減分します。削除済みの要素を差し引いたカウントを取得するには、*OCITableSize()* を使用します。

次の擬似コードで例をいくつか示します。

```
OCICollSize(...);
// assume 'size' returned is equal to 5
OCITableDelete(...); // delete one element
OCICollSize(...);
// 'size' returned is still 5
```

削除済みの要素を差し引いたカウントを取得するには、*OCITableSize()* を使用します。上記の例を続けます。

```
OCITableSize(...)  
// 'size' returned is equal to 4
```

切捨て操作 (*OCICollTrim()*) は、切り捨てられた要素の数だけこのカウントを減分します。
上記の例を続けます。

```
OCICollTrim(...,1..); // trim one element  
OCICollSize(...);  
// 'size' returned is equal to 4
```

この関数は、コレクションのオブジェクト・キャッシュへのロード中にエラーが発生した場合または入力パラメータに NULL がある場合にエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCICollMax\(\)](#)

OCICollTrim()

用途

指定数の要素をコレクションの最後から切り捨てます。

構文

```
sword OCICollTrim ( OCIEnv          *env,  
                   OCIError        *err,  
                   sb4              trim_num,  
                   OCIColl         *coll );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

trim_num (IN)

切り捨てる要素数。

coll (IN/OUT)

この関数は、*trim_num* 要素を *coll* の最後から削除（解放）します。

コメント

要素は、コレクションの末尾から削除されます。*trim_num* がコレクションの現在のサイズよりも大きい場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCICollSize\(\)](#)

OCIIterCreate()

用途

要素またはコレクションをスキャンするためのイテレータを作成します。

構文

```
sword OCIIterCreate ( OCIEnv          *env,  
                    OCIError       *err,  
                    CONST OCIColl   *coll,  
                    OCIIter        **itr );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

coll (IN)

スキャンするコレクション。リリース *si* では、有効なコレクション型として *VARRAY* とネストした表が含まれています。

itr (OUT)

割り当てられたコレクション・イテレータのアドレスは、この関数で戻されます。

コメント

イテレータは、オブジェクト・キャッシュの中に作成されます。イテレータは、コレクションの先頭を指すように初期化されます。

イテレータの作成直後に *OCIIterNext()* をコールした場合は、コレクションの最初の要素が戻ります。イテレータの作成直後に *OCIIterPrev()* をコールした場合は、「コレクションの先頭にいる」ことを示すエラーが戻ります。

この関数は、入力パラメータに *NULL* がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIIterDelete\(\)](#)

OCIIterDelete()

用途

コレクション・イテレータを削除します。

構文

```
sword OCIIterDelete ( OCIEnv          *env,  
                     OCIError       *err,  
                     OCIIter        **itr );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

itr (IN/OUT)

戻す前に破棄され NULL に設定されたコレクション・イテレータ。

コメント

[OCIIterCreate\(\)](#) のコールで以前に作成されたイテレータを削除します。

この関数は、入力パラメータに NULL がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIIterCreate\(\)](#)

OCIIterGetCurrent()

用途

現在のイテレータ・コレクション要素のポインタを取得します。

構文

```
sword OCIIterGetCurrent ( OCIEnv          *env,  
                          OCIError       *err,  
                          CONST OCIIter   *itr,  
                          dvoid          **elem,  
                          dvoid          **elemind );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

itr (IN)

現行の要素を指すイテレータ。

elem (OUT)

イテレータが参照した要素のアドレスが戻されます。

elemind (OUT) [optional]

要素の NULL 標識情報のアドレスが戻されます。(*elem_ind* == NULL) の場合、NULL 標識情報は戻されません。

コメント

現在のイテレータ・コレクション要素のポインタとそれに対応する NULL 情報を戻します。この関数は、入力パラメータに NULL がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIIterNext\(\)](#)、[OCIIterPrev\(\)](#)

OCIIterInit()

用途

コレクションをスキャンするためのイテレータを初期化します。

構文

```
sword OCIIterInit ( OCIEnv          *env,  
                  OCIError        *err,  
                  CONST OCIColl    *coll,  
                  OCIIter         *itr );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

coll (IN)

スキャンするコレクション。リリース *si* では、有効なコレクション型として VARRAY とネストした表が含まれています。

itr (IN/OUT)

割り当てられたコレクション・イテレータへのポインタ。

コメント

イテレータが、指定したコレクションの先頭を指すように初期化します。入力パラメータに NULL がある場合はエラーを戻します。この関数を使うと次のことができます。

- コレクションの先頭を指すようにイテレータをリセットします。
- 割当て済みのイテレータを再利用して別のコレクションをスキャンします。

関連関数

[OCIErrorGet\(\)](#)

OCIIterNext()

用途

次のイテレータ・コレクション要素のポインタを取得します。

構文

```
sword OCIIterNext ( OCIEnv          *env,
                   OCIError       *err,
                   OCIIter        *itr,
                   dvoid          **elem,
                   dvoid          **elemind,
                   boolean        *eoc );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCIEnvCreate()」および「OCIInitialize()」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCIErrorGet() をコールして診断情報を取得します。

itr (IN/OUT)

次の要素を指すようにイテレータを更新します。

elem (OUT)

イテレータが次要素を指すように更新された後、要素のアドレスを戻します。

elemind (OUT) [optional]

要素の NULL 標識情報のアドレスが戻されます。(*elem_ind* == NULL) の場合、NULL 標識情報は戻されません。

eoc (OUT)

イテレータがコレクションの末尾にある（つまり次の要素が存在しない）場合は TRUE、それ以外の場合は FALSE となります。

コメント

この関数は、次のイテレータ・コレクション要素のポインタおよびそれに対応する NULL 情報を戻します。また、次の要素を指すようにイテレータを更新します。

この関数の実行前にイテレータがコレクションの最後の要素を指している場合は、この関数をコールすると *eoc* フラグに TRUE が設定されます。この場合、イテレータは更新されません。

この関数は、入力パラメータに NULL がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIIterGetCurrent\(\)](#)、[OCIIterPrev\(\)](#)

OCIIterPrev()

用途

直前のイテレータ・コレクション要素のポインタを取得します。

構文

```
sword OCIIterPrev ( OCIEnv          *env,  
                   OCIError       *err,  
                   OCIIter        *itr,  
                   dvoid          **elem,  
                   dvoid          **elemind,  
                   boolean        *boc );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

itr (IN/OUT)

前の要素を指すように更新されるイテレータ。

elem (OUT)

イテレータが前の要素を指すように更新された後に戻される、前の要素のアドレス。

elemind (OUT) [optional]

要素の NULL 標識のアドレス。(*elem_ind* == NULL) の場合、NULL 標識は戻されません。

boc (OUT)

イテレータがコレクションの最初にある（つまり前の要素が存在しない）場合は TRUE、それ以外の場合は FALSE となります。

コメント

この関数は、直前のイテレータ・コレクション要素のポインタおよびそれに対応する NULL 情報を戻します。イテレータは、前の要素を指すように更新されます。

この関数の実行前にイテレータがコレクションの最初の要素を指している場合は、この関数をコールすると *boc* フラグに TRUE が設定されます。この場合、イテレータは更新されません。

OCIIterPrev()

この関数は、入力パラメータに NULL がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIIterGetCurrent\(\)](#)、[OCIIterNext\(\)](#)

OCI 日付関数

この項では、OCI 日付関数について説明します。

表 17-3 OCI 日付関数クイック・リファレンス

関数	用途	ページ
OCIDateAddDays()	日数の加算または減算を行います。	17-28 ページ
OCIDateAddMonths()	月の加算または減算を行います。	17-29 ページ
OCIDateAssign()	日付を割り当てます。	17-30 ページ
OCIDateCheck()	指定の日付が有効かどうかをチェックします。	17-31 ページ
OCIDateCompare()	日付を比較します。	17-33 ページ
OCIDateDaysBetween()	2 つの日付の間にある日数を取得します。	17-34 ページ
OCIDateFromText()	文字列を日付に変換します。	17-35 ページ
OCIDateGetDate()	日付の日付部分を取得します。	17-37 ページ
OCIDateGetTime()	日付の時刻部分を取得します。	17-38 ページ
OCIDateLastDay()	月末の日付を取得します。	17-39 ページ
OCIDateNextDay()	翌日の日付を取得します。	17-40 ページ
OCIDateSetDate()	日付の日付部分を設定します。	17-41 ページ
OCIDateSetTime()	日付の時刻部分を設定します。	17-42 ページ
OCIDateSysDate()	現在のシステム日付および時刻を取得します。	17-43 ページ
OCIDateToText()	日付を文字列に変換します。	17-44 ページ
OCIDateZoneToZone()	あるタイム・ゾーンの日付を別のゾーンの日付に変換します。	17-46 ページ

OCIDateAddDays()

用途

指定の日付に日数を加算または減算します。

構文

```
sword OCIDateAddDays ( OCIError          *err,  
                      CONST OCIDate      *date,  
                      sb4                num_days,  
                      OCIDate            *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date (IN)

加算または減算の対象となる指定の日付。

num_days (IN)

加算または減算される日数。マイナス値の場合に減算となります。

result (IN/OUT)

date に対して日数を加算または減算した結果。

コメント

この関数は、無効な日付が渡された場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateAddMonths\(\)](#)

OCIDateAddMonths()

用途

指定の日付に月数を加算または減算します。

構文

```
sword OCIDateAddMonths ( OCIError          *err,  
                        CONST OCIDate      *date,  
                        sb4                 num_months,  
                        OCIDate            *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date (IN)

加算または減算の対象となる指定の日付。

num_months (IN)

加算または減算される月数。マイナス値の場合減算となります。

result (IN/OUT)

date に対して日数を加算または減算した結果。

コメント

入力された *date* が月末の場合は、出力される日付も月末になるように適切な調整が行われま
す。たとえば、2月28日 + 1か月 = 3月31日、11月30日 - 3か月 = 8月31日となりま
す。それ以外の場合、*result* 日付の日は *date* の日と同じになります。

この関数は、無効な日付が渡された場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateAddDays\(\)](#)

OCIDateAssign()

用途

日付の割当てを実行します。

構文

```
sword OCIDateAssign ( OCLError          *err,  
                     CONST OCIDate     *from,  
                     OCIDate           *to );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

from (IN)

割り当てる日付。

to (OUT)

割当てのターゲット。

コメント

この関数は、ある *OCIDate* 変数の値を別の変数に割り当てます。

関連関数

[OCLErrorGet\(\)](#)、[OCIDateCheck\(\)](#)

OCIDateCheck()

用途

指定の日付が有効かどうかをチェックします。

構文

```
sword OCIDateCheck ( OCIError      *err,
                    CONST OCIDate  *date,
                    uword          *valid );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date (IN)

チェックする日付。

valid (OUT)

有効な日付に対して 0 (ゼロ) を戻します。それ以外の場合は、次のように指定されたすべてのエラー・ビットの OR による組合せを戻します。

マクロ名	ビット数	エラー
OCI_DATE_INVALID_DAY	0x1	無効な日
OCI_DATE_DAY_BELOW_VALID	0x2	無効な日の、有効な日に対する大 / 小を示すビット (1= 小)
OCI_DATE_INVALID_MONTH	0x4	無効な月
OCI_DATE_MONTH_BELOW_VALID	0x8	無効な月の、有効な月に対する大 / 小を示すビット (1= 小)
OCI_DATE_INVALID_YEAR	0x10	無効な年
OCI_DATE_YEAR_BELOW_VALID	0x20	無効な年の、有効な年に対する大 / 小を示すビット (1= 小)
OCI_DATE_INVALID_HOUR	0x40	無効な時刻 (時)
OCI_DATE_HOUR_BELOW_VALID	0x80	無効な時刻 (時) の、有効な時刻 (時) に対する大 / 小を示すビット (1= 小)
OCI_DATE_INVALID_MINUTE	0x100	無効な時刻 (分)

マクロ名	ビット数	エラー
OCI_DATE_MINUTE_BELOW_VALID	0x200	無効な時刻（分）の、有効な時刻（分）に対する大 / 小を示すビット（1=小）
OCI_DATE_INVALID_SECOND	0x400	無効な時刻（秒）
OCI_DATE_SECOND_BELOW_VALID	0x800	無効な時刻（秒）の、有効な時刻（秒）に対する大 / 小を示すビット（1=小）
OCI_DATE_DAY_MISSING_FROM_1582	0x1000	日が 1582 から欠落しています。
OCI_DATE_YEAR_ZERO	0x2000	年が 0（ゼロ）に等しくありません。
OCI_DATE_INVALID_FORMAT	0x8000	無効な日付書式入力

たとえば、渡された日付が 2/0/1990 25:61:10（月 / 日 / 年 時 : 分 : 秒書式）の場合は、次のエラーが戻されます。

```
OCI_DATE_INVALID_DAY | OCI_DATE_DAY_BELOW_VALID | OCI_DATE_INVALID_HOUR |
OCI_DATE_INVALID_MINUTE.
```

コメント

この関数は、*date* または *valid* ポインタが NULL の場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateCompare\(\)](#)

OCIDateCompare()

用途

2 つの日付を比較します。

構文

```
sword OCIDateCompare ( OCIError      *err,  
                      CONST OCIDate  *date1,  
                      CONST OCIDate  *date2,  
                      sword          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date1、date2 (IN)

比較する日付。

result (OUT)

以下の比較結果。

比較結果	<i>result</i> パラメータの出力
date1 < date2	-1
date1 = date2	0
date1 > date2	1

コメント

この関数は、無効な日付が渡された場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateCheck\(\)](#)

OCIDateDaysBetween()

用途

2つの日付間の日数を取得します。

構文

```
sword OCIDateDaysBetween ( OCIError          *err,  
                           CONST OCIDate      *date1,  
                           CONST OCIDate      *date2,  
                           sb4                *num_days );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date1 (IN)

入力日付。

date2 (IN)

入力日付。

num_days (OUT)

date1 と *date2* の間の日数。

コメント

date1 と *date2* の間の日数の計算時には、時刻は無視されます。

この関数は、無効な日付が渡された場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateCheck\(\)](#)

OCIDateFromText()

用途

指定された書式に従って、文字列を日付型に変換します。

構文

```
sword OCIDateFromText ( OCIError          *err,  
                        CONST text        *date_str,  
                        ub4               d_str_length,  
                        CONST text        *fmt,  
                        ub1               fmt_length,  
                        CONST text        *lang_name,  
                        ub4               lang_length,  
                        OCIDate           *date );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date_str (IN)

Oracle 日付に変換される入力文字列。

d_str_length (IN)

入力文字列のサイズ。長さが -1 の場合は、*date_str* は NULL 終了の文字列として処理されません。

fmt (IN)

変換書式。*fmt* が NULL ポインタの場合、文字列は 'DD-MON-YY' 書式になります。

fmt_length (IN)

fmt パラメータの長さ。

lang_name (IN)

日および月の名前と省略が指定される言語。*lang_name* が NULL 文字列 (text *) 0 の場合は、そのセッションのデフォルト言語が使われます。

lang_length (IN)

lang_name パラメータの長さ。

date (OUT)

日付に変換された指定文字列。

コメント

書式および NLS 引数についての説明は、『Oracle8i SQL リファレンス』の第 3 章にある TO_DATE 変換関数を参照してください。

この関数は、無効な書式または入力文字列を受け取った場合にエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateToText\(\)](#)

OCIDateGetDate()

用途

Oracle 日付に記憶されている年月日を取得します。

構文

```
void OCIDateGetDate ( CONST OCIDate   *date,  
                      sb2             *year,  
                      ub1             *month,  
                      ub1             *day );
```

パラメータ

date (IN)

年、月、日データの取出し元となる Oracle 日付。

year (OUT)

戻される年値。

month (OUT)

戻される月値。

day (OUT)

戻される日付値。

コメント

なし

関連関数

[OCIDateSetDate\(\)](#)、[OCIDateGetTime\(\)](#)

OCIDateGetTime()

用途

Oracle 日付に記憶されている時刻を取得します。

構文

```
void OCIDateGetTime ( CONST OCIDate   *date,  
                      ub1             *hour,  
                      ub1             *min,  
                      ub1             *sec );
```

パラメータ

date (IN)

時間データの取出し元となる Oracle 日付。

hour (OUT)

戻される時間値。

min (OUT)

戻される分値。

sec (OUT)

戻される秒値。

コメント

時、分、秒の時刻情報を戻します。

関連関数

OCIDateSetTime()、*OCIDateGetDate()*

OCIDateLastDay()

用途

指定された日付の月の最後の日の日付を取得します。

構文

```
sword OCIDateLastDay ( OCIError          *err,  
                      CONST OCIDate      *date,  
                      OCIDate            *last_day );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date (IN)

入力日付。

last_day (OUT)

date の月の最後の日付。

コメント

この関数は、無効な日付が渡された場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateGetDate\(\)](#)

OCIDateNextDay()

用途

指定した日付以降で最初の曜日の日付を取得します。

構文

```
sword OCIDateNextDay ( OCIError          *err,  
                      CONST OCIDate      *date,  
                      CONST text         *day,  
                      ub4                day_length,  
                      OCIDate            *next_day );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、`OCL_ERROR` が戻されます。`OCIErrorGet()` をコールして診断情報を取得します。

date (IN)

戻される日付は、この日付よりも後になります。

day (IN)

これによって指定された週の最初の日が返されます。

day_length (IN)

文字列 *day* のバイト長。

next_day (OUT)

date 以降最初の、*day* によって指定された曜日の日付。

コメント

date 以降最初の、*day* によって指定された曜日の日付を戻します。

例

1996 年 4 月 18 日 (木曜日) の次の月曜日の日付を取得します。

```
OCIDateNextDay(&err, '18-APR-96', 'MONDAY', strlen('MONDAY'), &next_day)
```

`OCIDateNextDay()` は '22-APR-96' を戻します。

この関数は、無効な日付または曜日が渡された場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateGetDate\(\)](#)

OCIDateSetDate()

用途

Oracle 日付に値を設定します。

構文

```
void OCIDateSetDate ( OCIDate      *date,  
                      sb2          year,  
                      ub1          month,  
                      ub1          day );
```

パラメータ

date (OUT)

時間データ設定の対象となる Oracle 日付。

year (IN)

設定する年値。

month (IN)

設定する月値。

day (IN)

設定する日値。

コメント

なし

関連関数

[OCIDateGetDate\(\)](#)

OCIDateSetTime()

用途

Oracle 日付に時刻情報を設定します。

構文

```
void OCIDateSetTime ( OCIDate   *date,  
                      ub1       hour,  
                      ub1       min,  
                      ub1       sec );
```

パラメータ

date (OUT)

時間データ設定の対象となる Oracle 日付。

hour (IN)

設定する時間値。

min (IN)

設定する分値。

sec (IN)

設定する秒値。

コメント

なし

関連関数

[OCIDateGetTime\(\)](#)

OCIDateSysDate()

用途

現在のシステム日付およびシステム時刻を取得します。

構文

```
sword OCIDateSysDate ( OCIError      *err,  
                      OCIDate       *sys_date );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

sys_date (OUT)

現在のシステム日付およびシステム時刻。

コメント

なし

関連関数

[OCIErrorGet\(\)](#)

OCIDateToText()

用途

日付型を文字列に変換します。

構文

```
sword OCIDateToText ( OCIError          *err,  
                     CONST OCIDate     *date,  
                     CONST text        *fmt,  
                     ub1               fmt_length,  
                     CONST text        *lang_name,  
                     ub4               lang_length,  
                     ub4               *buf_size,  
                     text              *buf );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date (IN)

変換する Oracle 日付。

fmt (IN)

変換書式。NULL 文字列ポインタ、つまり (*text **) 0 の場合、日付はデフォルト日付書式 DD-MON-YY の文字列に変換されます。

fmt_length (IN)

fmt パラメータの長さ。

lang_name (IN)

月および日の名前と省略形が戻される言語を指定します。*lang_name* が NULL ((*text **) 0) の場合、セッションのデフォルト言語が使用されます。

lang_length (IN)

lang_name パラメータの長さ。

buf_size (IN/OUT)

- バッファ (IN) のサイズ。
- このパラメータ (OUT) に、結果の文字列のサイズが戻されます。

buf (OUT)

変換された文字列が配置されるバッファ。

コメント

指定された日付を指定の書式の文字列に変換します。変換された NULL で終わる日付文字列は、*buf* に記憶されます。

書式および NLS 引数の説明は、『Oracle8i SQL リファレンス』の第 3 章にある TO_DATE 変換関数を参照してください。

この関数は、バッファが小さすぎる場合または無効な書式や認識できない言語が渡された場合はエラーを戻します。オーバーフローもエラーの原因となります。たとえば、値 10 を書式 '9' に変換すると、エラーが発生します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateFromText\(\)](#)

OCIDateZoneToZone()

用途

あるタイム・ゾーンの日付を別のタイム・ゾーンの日付に変換します。

構文

```
sword OCIDateZoneToZone ( OCLError          *err,  
                          CONST OCIDate      *date1,  
                          CONST text         *zon1,  
                          ub4                zon1_length,  
                          CONST text         *zon2,  
                          ub4                zon2_length,  
                          OCIDate           *date2 );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

date1 (IN)

変換する日付。

zon1 (IN)

入力日付のゾーン。

zon1_length (IN)

zon1 のバイト長。

zon2 (IN)

変換先のゾーン。

zon2_length (IN)

zon2 のバイト長。

date2 (OUT)

変換日付 (*zon2* での)。

コメント

タイム・ゾーン *zon1* の指定の日付 *date1* を、タイム・ゾーン *zon2* の日付 *date2* に変換します。

有効なゾーン文字列のリストは、『Oracle8i SQL リファレンス』の第3章の NEW_TIME 関数の説明を参照してください。有効なゾーン文字列には次のようなものがあります。

- AST (大西洋標準時)
- ADT (大西洋夏時間)
- BST (ベーリング標準時)
- BDT (ベーリング夏時間)

この関数は、無効な日付またはタイム・ゾーンが渡された場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIDateCheck\(\)](#)

OCI 数関数

この項では、OCI 数関数について説明します。

表 17-4 OCI 数関数クイック・リファレンス

関数	用途	ページ
OCINumberAbs()	絶対値を計算します。	17-50 ページ
OCINumberAdd()	数値を加算します。	17-51 ページ
OCINumberArcCos()	アーク・コサインを計算します。	17-52 ページ
OCINumberArcSin()	アーク・サインを計算します。	17-53 ページ
OCINumberArcTan()	アーク・タンジェントを計算します。	17-54 ページ
OCINumberArcTan2()	2 つの数値のアーク・タンジェントを計算します。	17-55 ページ
OCINumberAssign()	ある数値を別の数値に代入します。	17-56 ページ
OCINumberCeil()	数値の上限を計算します。	17-57 ページ
OCINumberCmp()	数値を比較します。	17-58 ページ
OCINumberCos()	コサインを計算します。	17-59 ページ
OCINumberDec()	OCI 数値を減分します。	17-60 ページ
OCINumberDiv()	2 つの数を除算します。	17-61 ページ
OCINumberExp()	e を指定の Oracle 数値で累乗します。	17-62 ページ
OCINumberFloor()	数値の下限を計算します。	17-63 ページ
OCINumberFromInt()	整数を Oracle 数値に変換します。	17-64 ページ
OCINumberFromReal()	実数を Oracle 数値に変換します。	17-65 ページ
OCINumberFromText()	文字列を Oracle 数値に変換します。	17-66 ページ
OCINumberHypCos()	双曲線コサインを計算します。	17-68 ページ
OCINumberHypSin()	双曲線サインを計算します。	17-69 ページ
OCINumberHypTan()	双曲線タンジェントを計算します。	17-70 ページ
OCINumberInc()	Oracle 数値を増分します。	17-71 ページ
OCINumberIntPower()	指定の底を整数で累乗します。	17-72 ページ
OCINumberIsInt()	数値が整数かどうかをテストします。	17-73 ページ
OCINumberIsZero()	数値が 0 (ゼロ) かどうかをテストします。	17-74 ページ

表 17-4 OCI 数関数クイック・リファレンス(続き)

関数	用途	ページ
OCINumberLn()	自然対数を計算します。	17-75 ページ
OCINumberLog()	任意の底に対する対数を計算します。	17-76 ページ
OCINumberMod()	除算の余り	17-77 ページ
OCINumberMul()	数値を乗算します。	17-78 ページ
OCINumberNeg()	数値を負の数にします。	17-79 ページ
OCINumberPower()	底 e を累乗します。	17-80 ページ
OCINumberPrec()	数値を指定の小数点以下の桁数に丸めます。	17-81 ページ
OCINumberRound()	Oracle 数値を指定の小数点以下の桁数に丸めます。	17-82 ページ
OCINumberSetPi()	数値を Pi に初期化します。	17-83 ページ
OCINumberSetZero()	数値を 0 (ゼロ) に初期化します。	17-84 ページ
OCINumberShift()	10 の累乗を乗算し、小数点以下を指定の桁数にします。	17-85 ページ
OCINumberSign()	Oracle 数値の符号を取得します。	17-86 ページ
OCINumberSin()	サインを計算します。	17-87 ページ
OCINumberSqrt()	数値の平方根を計算します。	17-88 ページ
OCINumberSub()	数値を減算します。	17-89 ページ
OCINumberTan()	タンジェントを計算します。	17-90 ページ
OCINumberToInt()	Oracle 数値を整数に変換します。	17-91 ページ
OCINumberToReal()	Oracle 数値を実数に変換します。	17-92 ページ
OCINumberToText()	Oracle 数値を文字列に変換します。	17-93 ページ
OCINumberTrunc()	Oracle 数値を指定の小数点以下の桁数で切り捨てます。	17-95 ページ

OCINumberAbs()

用途

Oracle 数値の絶対値を計算します。

構文

```
sword OCINumberAbs ( OCIError          *err,  
                    CONST OCINumber    *number,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

入力数値。

result (OUT)

入力数値の絶対値。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberSign\(\)](#)

OCINumberAdd()

用途

2 つの Oracle 数値を加算します。

構文

```
sword OCINumberAdd ( OCIError          *err,  
                    CONST OCINumber    *number1,  
                    CONST OCINumber    *number2,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number1、number2 (IN)

足し合わされる数値。

result (OUT)

number1 を *number2* に加算した結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberSub\(\)](#)

OCINumberArcCos()

用途

Oracle 数値のアーク・コサインをラジアン単位で求めます。

構文

```
sword OCINumberArcCos ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

アーク・コサインの引数。

result (OUT)

ラジアンのアーク・コサインの結果。

コメント

この関数は、NULL の数値引数がある場合、または *number* が -1 より小さいか *number* が 1 より大きい場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberCos\(\)](#)

OCINumberArcSin()

用途

Oracle 数値のアーク・サインをラジアン単位で求めます。

構文

```
sword OCINumberArcSin ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

アーク・サインの引数。

result (OUT)

ラジアンのアーク・サインの結果。

コメント

この関数は、NULL の数値引数がある場合、または *number* が -1 より小さいか *number* が 1 より大きい場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberSin\(\)](#)

OCINumberArcTan()

用途

Oracle 数値のアーク・タンジェントをラジアン単位で求めます。

構文

```
sword OCINumberArcTan ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

アーク・タンジェントの引数。

result (OUT)

ラジアンのアーク・タンジェントの結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberTan\(\)](#)

OCINumberArcTan2()

用途

2 つの Oracle 数値のアーク・タンジェントを求めます。

構文

```
sword OCINumberArcTan2 ( OCLError          *err,  
                        CONST OCINumber    *number1,  
                        CONST OCINumber    *number2,  
                        OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

number1 (IN)

アーク・タンジェントの引数 1。

number2 (IN)

アーク・タンジェントの引数 2。

result (OUT)

ラジアンのアーク・タンジェントの結果。

コメント

この関数は、NULL の数値引数がある場合、または *number2* が 0 (ゼロ) の場合はエラーを戻します。

関連関数

[OCLErrorGet\(\)](#)、[OCINumberTan\(\)](#)

OCINumberAssign()

用途

ある Oracle 数値を別の Oracle 数値に割り当てます。

構文

```
sword OCINumberAssign ( OCIError          *err,  
                        CONST OCINumber    *from,  
                        OCINumber          *to );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

from (IN)

割り当てる数値。

to (OUT)

コピー先の数値。

コメント

from によって識別される数値を、*to* によって識別される数値に代入します。

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberCmp\(\)](#)

OCINumberCeil()

用途

Oracle 数値の上限値を計算します。

構文

```
sword OCINumberCeil ( OCIError          *err,  
                     CONST OCINumber    *number,  
                     OCINumber         *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

入力数値。

result (OUT)

入力数値の上限値を含む出力。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberFloor\(\)](#)

OCINumberCmp()

用途

2 つの Oracle 数値を比較します。

構文

```
sword OCINumberCmp ( OCIError          *err,  
                    CONST OCINumber    *number1,  
                    CONST OCINumber    *number2,  
                    sword                *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number1、number2 (IN)

比較する数値。

result (OUT)

以下の比較結果。

比較結果	<i>result</i> パラメータの出力
number1 < number2	負数
number1 = number2	0
number1 > number2	正数

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberAssign\(\)](#)

OCINumberCos()

用途

Oracle 数値のコサインをラジアン単位で計算します。

構文

```
sword OCINumberCos ( OCLError          *err,  
                    CONST OCINumber    *number,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

number (IN)

ラジアンのコサインの引数。

result (OUT)

コサインの結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCLErrorGet\(\)](#)、[OCINumberArcCos\(\)](#)

OCINumberDec()

用途

OCINumber を減分します。

構文

```
OCINumberDec ( OCLError *err,  
               OCINumber *number );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

number (IN/OUT)

減分される正数の Oracle 数値。

コメント

Oracle 数値を適切に減分します。入力値は 0 ~ 100²¹-2 の整数であると見なされます。入力値が大きすぎる場合は、0 (ゼロ) として扱われ、その結果、Oracle 数値は 1 になります。入力値が正の整数でない場合は、予想できない結果になります。

この関数は、入力数値が NULL の場合はエラーを戻します。

関連関数

[OCINumberDec\(\)](#)、[OCINumberInc\(\)](#)

OCINumberDiv()

用途

2 つの Oracle 数値を除算します。

構文

```
sword OCINumberDiv ( OCIError          *err,  
                    CONST OCINumber    *number1,  
                    CONST OCINumber    *number2,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number1 (IN)

分子へのポインタ。

number2 (IN)

分母へのポインタ。

result (OUT)

除算結果。

コメント

number1 を *number2* で除算し、その結果を *result* に戻します。

この関数は、次の場合にエラーを戻します。

- NULL の数値引数がある
- アンダーフロー・エラーがある
- ゼロによる除算エラーがある

関連関数

[OCIErrorGet\(\)](#)、[OCINumberMul\(\)](#)

OCI_{NumberExp}()

用途

e を指定の Oracle 数値で累乗します。

構文

```
sword OCINumberExp ( OCIError          *err,  
                    CONST OCINumber   *number,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

この関数は、*e* をこの Oracle 数値で累乗します。

result (OUT)

指数の出力。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCI_{NumberLn}\(\)](#)

OCINumberFloor()

用途

Oracle 数値の下限値を計算します。

構文

```
sword OCINumberFloor ( OCLError          *err,  
                      CONST OCINumber    *number,  
                      OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

number (IN)

入力数値。

result (OUT)

入力数値の下限値。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCLErrorGet\(\)](#)、[OCINumberCeil\(\)](#)

OCINumberFromInt()

用途

整数を Oracle 数値に変換します。

構文

```
sword OCINumberFromInt ( OCIError          *err,  
                        CONST dvoid       *inum,  
                        uword             inum_length,  
                        uword             inum_s_flag,  
                        OCINumber        *number );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、`OCL_ERROR` が戻されます。`OCIErrorGet()` をコールして診断情報を取得します。

inum (IN)

変換する整数へのポインタ。

inum_length (IN)

整数のサイズ。

inum_s_flag (IN)

整数の符号を指定するフラグで、次のものがあります。

- `OCL_NUMBER_UNSIGNED` - 符号なしの値
- `OCL_NUMBER_SIGNED` - 符号付きの値

number (OUT)

Oracle 数値に変換される指定の整数。

コメント

これは、ネイティブな型変換関数です。`ub4` や `sb2` などの Oracle 標準マシンネイティブ整数型を Oracle 数値に変換します。

この関数は、数値が大きすぎて Oracle 数値に適合しない場合、*number* または *inum* が NULL の場合、*inum_s_flag* に無効な符号フラグの値が渡された場合は、エラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberToInt\(\)](#)

OCINumberFromReal()

用途

実数（浮動小数点数）型を Oracle 数値に変換します。

構文

```
sword OCINumberFromReal ( OCLError          *err,  
                          CONST dvoid       *rnum,  
                          uword             rnum_length,  
                          OCINumber         *number );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

rnum (IN)

変換する浮動小数点数へのポインタ。

rnum_length (IN)

希望の結果のサイズ。*sizeof({float | double | long double})* に等しくなります。

number (OUT)

Oracle 数値に変換される指定の float。

コメント

これは、ネイティブな型変換関数です。マシンネイティブ浮動小数点型を Oracle 数値に変換します。

この関数は、*number* または *rnum* が NULL であるか、*rnum_length* が 0（ゼロ）の場合はエラーを戻します。

関連関数

[OCLErrorGet\(\)](#)、[OCINumberToReal\(\)](#)

OCINumberFromText()

用途

文字列を Oracle 数値に変換します。

構文

```

sword OCINumberFromText ( OCIError          *err,
                          CONST text        *str,
                          ub4               str_length,
                          CONST text        *fmt,
                          ub4               fmt_length,
                          CONST text        *nls_params,
                          ub4               nls_p_length,
                          OCINumber        *number );
    
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

str (IN)

Oracle 数値に変換する入力文字列。

str_length (IN)

入力文字列のサイズ。

fmt (IN)

変換書式。

fmt_length (IN)

fmt パラメータの長さ。

nls_params (IN)

NLS 書式指定。NULL 文字列 ("") の場合は、セッションのデフォルト・パラメータが使用されます。

nls_p_length (IN)

nls_params パラメータの長さ。

number (OUT)

数値に変換された指定文字列。

コメント

指定された文字列を指定の書式の数値に変換します。書式および NLS パラメータの詳細は、『Oracle8i SQL リファレンス』の TO_NUMBER 変換関数を参照してください。

この関数は、無効な書式または NLS 書式、入力文字列がある場合、*number* または *str* が NULL である場合、*str_length* が 0 (ゼロ) である場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberToText\(\)](#)

OCINumberHypCos()

用途

Oracle 数値の双曲線コサインを計算します。

構文

```
sword OCINumberHypCos ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

コサイン双曲線の引数。

result (OUT)

コサイン双曲線の結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

警告 : Oracle 数値のオーバーフローは、予想できない結果値が出力される原因となります。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberHypSin\(\)](#)、[OCINumberHypTan\(\)](#)

OCINumberHypSin()

用途

Oracle 数値の双曲線サインを計算します。

構文

```
sword OCINumberHypSin ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

サイン双曲線の引数。

result (OUT)

サイン双曲線の結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

警告: Oracle 数値のオーバーフローは、予想できない結果値が出力される原因となります。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberHypCos\(\)](#)、[OCINumberHypTan\(\)](#)

OCINumberHypTan()

用途

Oracle 数値の双曲線タンジェントを計算します。

構文

```
sword OCINumberHypTan ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        OCINumber         *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

タンジェント双曲線の引数。

result (OUT)

タンジェント双曲線の結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

警告 : Oracle 数値のオーバーフローは、予想できない結果値が出力される原因となります。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberHypCos\(\)](#)、[OCINumberHypSin\(\)](#)

OCINumberInc()

用途

OCINumber を増分します。

構文

```
OCINumberInc ( OCLError   *err,  
               OCINumber  *number );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN/OUT)

増分される正数の Oracle 数値。

コメント

Oracle 数値を適切に増分します。入力値は $0 \sim 100^{21-2}$ の整数であると見なされます。入力値が大きすぎる場合は、0 (ゼロ) として扱われ、その結果、Oracle 数値は 1 になります。入力値が正の整数でない場合は、予想できない結果になります。

この関数は、入力数値が NULL の場合はエラーを戻します。

関連関数

[OCINumberDec\(\)](#)

OCINumberIntPower()

用途

指定の底を指定の整数で累乗します。

構文

```
sword OCINumberIntPower ( OCLError          *err,  
                          CONST OCINumber    *base,  
                          CONST sword        exp,  
                          OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

base (IN)

指数の基数。

exp (IN)

基数が累乗される指数。

result (OUT)

指数の出力。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCLErrorGet\(\)](#)、[OCINumberPower\(\)](#)

OCINumberIsInt()

用途

OCINumber が整数かどうかをテストします。

構文

```
sword OCINumberIsInt ( OCIError          *err,  
                      CONST OCINumber   *number,  
                      boolean           *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

テストされる数値。

result (OUT)

整数値の場合は TRUE、それ以外の場合は FALSE に設定されます。

コメント

この関数は、*number* または *result* が NULL の場合にエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberRound\(\)](#)、[OCINumberTrunc\(\)](#)

OCINumberIsZero()

用途

指定の数値がゼロに等しいかどうかをテストします。

構文

```
sword OCINumberIsZero ( OCIError          *err,  
                        CONST OCINumber   *number,  
                        boolean           *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

比較する数値。

result (OUT)

ゼロの場合は TRUE、それ以外の場合は FALSE が設定されます。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberSetZero\(\)](#)

OCINumberLn()

用途

Oracle 数値の自然対数 (底 e) を求めます。

構文

```
sword OCINumberLn ( OCIError          *err,  
                   CONST OCINumber    *number,  
                   OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

この数値の対数を計算します。

result (OUT)

対数結果。

コメント

この関数は、NULL の数値引数があるか、*number1* が 0 (ゼロ) 以下の場合はエラーを戻しません。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberExp\(\)](#)、[OCINumberLog\(\)](#)

OCI_{NumberLog}()

用途

Oracle 数値の任意の底に対する対数を求めます。

構文

```
sword OCINumberLog ( OCIError          *err,  
                    CONST OCINumber   *base,  
                    CONST OCINumber   *number,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCI_{ErrorGet}() をコールして診断情報を取得します。

base (IN)

対数の基数。

number (IN)

演算子。

result (OUT)

対数結果。

コメント

この関数は、次の場合にエラーを戻します。

- NULL の数値引数がある
- *number* <= 0
- *base* <= 0

関連関数

[OCI_{ErrorGet}\(\)](#)、[OCI_{NumberLn}\(\)](#)

OCINumberMod()

用途

2 つの Oracle 数値の除算の余りを取得します。

構文

```
sword OCINumberMod ( OCLError          *err,  
                    CONST OCINumber    *number1,  
                    CONST OCINumber    *number2,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

number1 (IN)

分子へのポインタ。

number2 (IN)

分母へのポインタ。

result (OUT)

結果の剰余。

コメント

この関数は、*number1* または *number2* が NULL であるか、ゼロ割りエラーがある場合にエラーを戻します。

関連関数

[OCLErrorGet\(\)](#)、[OCINumberDiv\(\)](#)

OCINumberMul()

用途

2 つの Oracle 数値を乗算します。

構文

```
sword OCINumberMul ( OCIError          *err,  
                    CONST OCINumber   *number1,  
                    CONST OCINumber   *number2,  
                    OCINumber         *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number1 (IN)

乗算する数値。

number2 (IN)

乗算する数値。

result (OUT)

乗算結果。

コメント

number1 に *number2* を乗算し、結果を *result* に戻します。

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberDiv\(\)](#)

OCINumberNeg()

用途

Oracle 数値を負の数値にします。

構文

```
sword OCINumberNeg ( OCIError          *err,  
                    CONST OCINumber    *number,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

負にする数値。

result (OUT)

number の負の値が入ります。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberAbs\(\)](#)、[OCINumberSign\(\)](#)

OCI_{NumberPower}()

用途

指定の底を指定の指数で累乗します。

構文

```
sword OCINumberPower ( OCIError          *err,  
                      CONST OCINumber   *base,  
                      CONST OCINumber   *number,  
                      OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

base (IN)

指数の基数。

number (IN)

基数が累乗される指数。

result (OUT)

指数の出力。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCI_{NumberExp}\(\)](#)

OCINumberPrec()

用途

OCINumber を指定の小数点以下の桁数に丸めます。

構文

```
sword OCINumberPrec ( OCIError *err,  
                     CONST OCINumber *number,  
                     sword nDigs,  
                     OCINumber *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

精度を設定する数値。

nDigs (IN)

希望の結果の小数点以下の桁数。

result (OUT)

結果。

コメント

桁数に基づいて、浮動小数点を丸めます。

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberRound\(\)](#)

OCI_{NumberRound}()

用途

Oracle 数値を指定の桁数で丸めます。

構文

```
sword OCINumberRound ( OCIError          *err,  
                      CONST OCINumber   *number,  
                      sword              decplace,  
                      OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。OCI_{ErrorGet}() をコールして診断情報を取得します。

number (IN)

丸める数値。

decplace (IN)

丸める小数点以下の桁数。負数も可能です。

result (OUT)

丸めの出力。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCI_{ErrorGet}\(\)](#)、[OCI_{NumberTrunc}\(\)](#)

OCINumberSetPi()

用途

OCINumber を Pi に設定します。

構文

```
void OCINumberSetPi ( OCIError *err,  
                     OCINumber *num );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

num (OUT)

Pi の値に設定される数値。

コメント

指定の数値を Pi の値に初期化します。

関連関数

[OCIErrorGet\(\)](#)

OCINumberSetZero()

用途

Oracle 数値を 0 (ゼロ) に初期化します。

構文

```
void OCINumberSetZero ( OCIError      *err  
                        OCINumber     *num );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

num (IN/OUT)

ゼロ値に初期化する数値。

コメント

なし

関連関数

[OCIErrorGet\(\)](#)

OCINumberShift()

用途

数値を 10 の累乗で乗算し、小数点を指定の桁に移動します。

構文

```
sword OCINumberShift ( OCLError      *err,  
                       CONST OCINumber *number,  
                       CONST sword    nDig,  
                       OCINumber      *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

桁を移動する Oracle 数値。

nDig (IN)

移動する小数点以下の桁数。

result (OUT)

桁を移動した結果。

コメント

数値を 10^{nDig} で乗算し、*product* に結果を設定します。

この関数は、入力数値が NULL の場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)

OCINumberSign()

用途

Oracle 数値の符号を取得します。

構文

```
sword OCINumberSign ( OCLError          *err,  
                     CONST OCINumber    *number,  
                     sword              *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCLErrorGet()* をコールして診断情報を取得します。

number (IN)

符号が戻される数値。

result (OUT)

以下の可能値。

<i>number</i> の値	<i>result</i> パラメータの出力
<i>number</i> < 0	-1
<i>number</i> == 0	0
<i>number</i> > 0	1

コメント

この関数は、*number* または *result* が NULL の場合にエラーを戻します。

関連関数

[OCLErrorGet\(\)](#)、[OCINumberAbs\(\)](#)

OCINumberSin()

用途

Oracle 数値のサインをラジアン単位で計算します。

構文

```
sword OCINumberSin ( OCIError          *err,  
                    CONST OCINumber    *number,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

ラジアン単位でのサインの引数。

result (OUT)

サインの結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberArcSin\(\)](#)

OCINumberSqrt()

用途

Oracle 数値の平方根を計算します。

構文

```
sword OCINumberSqrt ( OCIError          *err,  
                     CONST OCINumber   *number,  
                     OCINumber        *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

入力数値。

result (OUT)

入力数値の平方根を含む出力。

コメント

この関数は、*number* が NULL または負数の場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberPower\(\)](#)

OCINumberSub()

用途

2 つの Oracle 数値を減算します。

構文

```
sword OCINumberSub ( OCIError          *err,  
                    CONST OCINumber    *number1,  
                    CONST OCINumber    *number2,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number1、number2 (IN)

この関数は、*number1* から *number2* を減算します。

result (OUT)

減算結果。

コメント

number1 から *number2* を減算し、その結果を *result* に戻します。

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberAdd\(\)](#)

OCINumberTan()

用途

Oracle 数値のタンジェントをラジアン単位で計算します。

構文

```
sword OCINumberTan ( OCIError          *err,  
                    CONST OCINumber    *number,  
                    OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

ラジアン単位でのタンジェントの引数。

result (OUT)

タンジェントの結果。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberArcTan\(\)](#)、[OCINumberArcTan2\(\)](#)

OCINumberToInt()

用途

Oracle 数値型を整数に変換します。

構文

```
sword OCINumberToInt ( OCIError          *err,  
                      CONST OCINumber    *number,  
                      uword              rsl_length,  
                      uword              rsl_flag,  
                      dvoid              *rsl );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

変換する数値。

rsl_length (IN)

適切な結果のサイズ。

rsl_flag (IN)

出力値の符号を指定するフラグで、可能な値は次のとおりです。

- OCI_NUMBER_UNSIGNED - 符号なしの値
- OCI_NUMBER_SIGNED - 符号付きの値

rsl (OUT)

結果を保存するスペースへのポインタ。

コメント

これは、ネイティブな型変換関数です。指定の Oracle 数値を *ub2*、*ub4*、*sb2* など、*xbn* という書式の整数に変換します。

この関数は、*number* または *rsl* が NULL の場合、または *number* が大きすぎる（オーバーフロー）か小さすぎる（アンダーフロー）場合、無効な符号フラグの値が *rsl_flag* に渡されるとエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberFromInt\(\)](#)

OCINumberToReal()

用途

Oracle 数値型を実数に変換します。

構文

```
sword OCINumberToReal ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        uword              rsl_length,  
                        dvoid               *rsl );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

変換する数値。

rsl_length (IN)

希望の結果のサイズ。sizeof ({ float | double | long double }) に等しくなります。

rsl (OUT)

結果を保存するスペースへのポインタ。

コメント

これは、ネイティブな型変換関数です。Oracle 数値をマシンネイティブ実数型に変換します。この関数は、単に最大 LDBL_DIG または DBL_DIG、FLT_DIG 桁の精度の数値を変換し、後続するゼロを削除します。これらの定数は、*float.h* に定義されます。

この関数は、*number* または *rsl* が NULL であるか、*rsl_length* = 0 の場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberFromReal\(\)](#)

OCINumberToText()

用途

指定された書式に従って、Oracle 数値を文字列に変換します。

構文

```
sword OCINumberToText ( OCIError          *err,  
                        CONST OCINumber    *number,  
                        CONST text         *fmt,  
                        ub4                 fmt_length,  
                        CONST text         *nls_params,  
                        ub4                 nls_p_length,  
                        ub4                 *buf_size,  
                        text                *buf );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

変換する Oracle 数値。

fmt (IN)

変換書式。

fmt_length (IN)

fmt パラメータの長さ。

nls_params (IN)

NLS 書式指定。NULL 文字列 ((text *)0) の場合は、そのセッションのデフォルト・パラメータが使われます。

nls_p_length (IN)

nls_params パラメータの長さ。

buf_size (IN)

バッファのサイズ。

buf (OUT)

変換された文字列が配置されるバッファ。

コメント

書式および NLS パラメータの詳細は、『Oracle8i SQL リファレンス』の TO_NUMBER 変換関数を参照してください。

変換された数字文字列は、最大 *buf_size* バイトまで *buf* に保存されます。この関数は、次の場合にエラーを戻します。

- *number* または *buf* が NULL である。
- バッファが小さすぎる。
- 書式または NLS 書式が無効である。
- 数値を指定の書式のテキストに翻訳するときにオーバーフローが発生した。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberFromText\(\)](#)

OCINumberTrunc()

用途

Oracle 数値を指定の桁数で切り捨てます。

構文

```
sword OCINumberTrunc ( OCIError          *err,  
                      CONST OCINumber    *number,  
                      sword              decplace,  
                      OCINumber          *result );
```

パラメータ

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

number (IN)

入力数値。

decplace (IN)

切り捨てる小数点以下の桁数。負数も可能です。

result (OUT)

切捨ての出力。

コメント

この関数は、NULL の数値引数がある場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCINumberRound\(\)](#)

OCI ロー関数

この項では、OCI ロー関数について説明します。

表 17-5 OCI ロー関数クイック・リファレンス

関数	用途	ページ
OCIRawAllocSize()	ロー・メモリーに割り当てられたサイズをバイト単位で取得します。	17-97 ページ
OCIRawAssignBytes()	ローにロー・バイトを割り当てます。	17-98 ページ
OCIRawAssignRaw()	ローにローを割り当てます。	17-99 ページ
OCIRawPtr()	ロー・データのポインタを取得します。	17-100 ページ
OCIRawResize()	可変長ローのメモリー・サイズを変更します。	17-101 ページ
OCIRawSize()	ローのサイズを取得します。	17-102 ページ

OCIRawAllocSize()

用途

ロー・メモリーの割当てサイズをバイト単位で取得します。

構文

```
sword OCIRawAllocSize ( OCIEnv          *env,  
                        OCIError       *err,  
                        CONST OCIRaw    *raw,  
                        ub4             *allocsize );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

raw (IN)

割り当てられたサイズがバイト単位で返されるロー・データ。これは、非 NULL ポインタでなければなりません。

allocsize (OUT)

ロー・メモリーの割当てサイズをバイト単位で戻します。

コメント

割り当てられるサイズは、実際のロー・サイズより大きいかそれに等しくなります。

関連関数

[OCIErrorGet\(\)](#)、[OCIRawResize\(\)](#)、[OCIRawSize\(\)](#)

OCIRawAssignBytes()

用途

型 `ub1*` の raw バイトを Oracle の `OCIRaw*` データ型に割り当てます。

構文

```
sword OCIRawAssignBytes ( OCIEEnv          *env,  
                          OCIError        *err,  
                          CONST ub1       *rhs,  
                          ub4              rhs_len,  
                          OCIRaw          **lhs );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「`OCIEnvCreate()`」および「`OCIInitialize()`」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、`err` に記録され、`OCI_ERROR` が戻されます。`OCIErrorGet()` をコールして診断情報を取得します。

rhs (IN)

割当ての右側（ソース）となるデータ型 `ub1`。

rhs_len (IN)

`rhs` ロー・バイトの長さ。

lhs (IN/OUT)

割当て `OCIRaw` データの左側（ターゲット）。

コメント

`rhs` ロー・バイトを `lhs` ロー・データ型に割り当てます。`lhs` ローのサイズは、`rhs` のサイズに応じて変更されます。割り当てられたロー・バイトの型は `ub1` になります。

関連関数

[OCIErrorGet\(\)](#)、[OCIRawAssignRaw\(\)](#)

OCIRawAssignRaw()

用途

ある Oracle RAW データ型を別の Oracle RAW データ型に割り当てます。

構文

```
sword OCIRawAssignRaw ( OCIEnv          *env,  
                        OCIError       *err,  
                        CONST OCIRaw    *rhs,  
                        OCIRaw         **lhs );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

rhs (IN)

割当ての右辺 (ソース)、OCIRaw データ。

lhs (IN/OUT)

割当ての左辺 (ターゲット)、OCIRaw データ。

コメント

rhs ローを *lhs* ローに割り当てます。*lhs* ローのサイズは、*rhs* のサイズに応じて変更されません。

関連関数

[OCIErrorGet\(\)](#)、[OCIRawAssignBytes\(\)](#)

OCIRawPtr()

用途

ロー・データへのポインタを取得します。

構文

```
ub1 *OCIRawPtr ( OCIEnv          *env,  
                 CONST OCIRaw    *raw );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

raw (IN)

指定された raw データのポインタを戻します。

コメント

なし

関連関数

[OCIErrorGet\(\)](#)、[OCIRawAssignRaw\(\)](#)

OCIRawResize()

用途

指定の可変長ローのメモリー・サイズを変更します。

構文

```
sword OCIRawResize ( OCIEnv          *env,  
                    OCIError       *err,  
                    ub2            new_size,  
                    OCIRaw        **raw );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

new_size (IN)

バイト単位での新規ロー・データのサイズ。

raw (IN)

可変長ロー・ポインタ。ローのサイズは *new_size* に変更されます。

コメント

この関数は、オブジェクト・キャッシュ内の指定の可変長ローのメモリー・サイズを変更します。そのローの以前の内容は保持されません。この関数は、新しいメモリー領域にローを割り当てることがありますが、この場合は指定のローが占有していたメモリーは解放されません。入力ローが NULL (*raw* == NULL) の場合、この関数は、ロー・データ用のメモリーを割り当てます。

new_size が 0 (ゼロ) の場合、この関数は *raw* が占有していたメモリーを解放し、NULL ポインタ値を戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIRawAllocSize\(\)](#)、[OCIRawSize\(\)](#)

OCIRawSize()

用途

指定のローのサイズをバイト単位で戻します。

構文

```
ub4 OCIRawSize ( OCIEnv          *env,  
                 CONST OCIRaw    *raw );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

raw (IN/OUT)

サイズが戻される raw。

コメント

なし

関連関数

[OCIErrorGet\(\)](#)、[OCIRawAllocSize\(\)](#)、[OCIRawSize\(\)](#)

OCI REF 関数

この項では、OCI REF 関数について説明します。

表 17-6 OCI データ型マッピング関数および操作関数クイック・リファレンス

関数	用途	ページ
OCIRefAssign()	1 つの REF を別の REF に代入します。	17-104 ページ
OCIRefClear()	REF を消去または NULL にします。	17-105 ページ
OCIRefFromHex()	16 進文字列を REF に変換します。	17-106 ページ
OCIRefHexSize()	REF の 16 進表現のサイズを戻します。	17-107 ページ
OCIRefsEqual()	2 つの REF が等しいか比較します。	17-108 ページ
OCIRefsNull()	REF が NULL かどうかをテストします。	17-109 ページ
OCIRefToHex()	REF を 16 進文字列に変換します。	17-110 ページ

OCIRefAssign()

用途

ある REF を別の REF に割り当て、両方が同じオブジェクトを参照するようにします。

構文

```
sword OCIRefAssign ( OCIEnv          *env,  
                    OCIError       *err,  
                    CONST OCIRef    *source,  
                    OCIRef          **target );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

source (IN)

コピー元の REF。

target (IN/OUT)

コピー先の REF。

コメント

source REF を *target* REF にコピーし、両方が同じオブジェクトを参照するようにします。*target* REF ポインタが NULL (つまり **target* == NULL) の場合、*OCIRefAssign()* は、コピーを実行する前に OCI オブジェクト・キャッシュ内に *target* REF 用のメモリーを割り当てません。

関連関数

[OCIErrorGet\(\)](#)、[OCIRefsEqual\(\)](#)

OCIRefClear()

用途

指定の REF を消去または無効化します。

構文

```
void OCIRefClear ( OCIEnv      *env,  
                  OCIRef      *ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

ref (IN/OUT)

消去する REF。

コメント

オブジェクトを指さない REF は、NULL REF とみなされます。論理的には、NULL REF はぶら下がり REF になります。

NULL REF は依然として有効な SQL 値であり、SQL の NULL ではないことに注意してください。これは、表内の非 NULL 列または行属性用の有効な非 NULL 定数 REF として使うことができます。

REF として NULL ポインタ値が渡された場合、この関数による操作は実行されません。

関連関数

[OCIErrorGet\(\)](#)、[OCIRefIsNull\(\)](#)

OCIRefFromHex()

用途

指定の 16 進文字列を REF に変換します。

構文

```
sword OCIRefFromHex ( OCIEnv          *env,  
                     OCIError       *err,  
                     CONST OCISvcCtx *svc,  
                     CONST text     *hex,  
                     ub4             length,  
                     OCIRef         **ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

svc (IN)

OCI サービス・コンテキスト・ハンドル。結果の REF がこのサービス・コンテキストで初期化される場合。

hex (IN)

REF に変換する 16 進文字列で、以前に *OCIRefToHex()* によって出力されたもの。

length (IN)

16 進文字列の長さ。

ref (IN/OUT)

16 進文字列の変換先となる REF。入力時に **ref* が NULL の場合は、その REF 用の領域がオブジェクト・キャッシュ内に割り当てられます。それ以外の場合は、指定の REF によって占有されているメモリーが再利用されます。

コメント

この関数は、変換後の REF の書式が正しいことを保証します。ただし、変換後の REF によって指示されるオブジェクトが存在するかどうかについては保証しません。

関連関数

[OCIErrorGet\(\)](#)、[OCIRefToHex\(\)](#)

OCIRefHexSize()

用途

REF の 16 進表現のサイズを戻します。

構文

```
ub4 OCIRefHexSize ( OCIEnv          *env,  
                   CONST OCIRef     *ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

ref (IN)

16 進表現のサイズをバイト単位に戻される REF。

戻り値

REF の 16 進表現のサイズ。

コメント

REF の 16 進表現のために必要なバッファ・サイズをバイト単位で戻します。最低このサイズのバッファを、REF から 16 進に変換する関数 (*OCIRefToHex()*) に渡す必要があります。

関連関数

[OCIErrorGet\(\)](#)、[OCIRefFromHex\(\)](#)

OCIRefsEqual()

用途

2 つの REF を比較してそれらが同等であるかどうかを判別します。

構文

```
boolean OCIRefsEqual ( OCIEnv          *env,  
                      CONST OCIRef     *x,  
                      CONST OCIRef     *y );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

x (IN)

比較する REF。

y (IN)

比較する REF。

戻り値

2 つの REF が同等な場合は TRUE。

2 つの REF が同等でない場合、または *x* が *y* が NULL の場合は FALSE。

コメント

2 つの REF は、オブジェクトの種類（永続または一時）に関係なく同じオブジェクトを参照している場合だけ同等になります。

注意: 2 つの NULL REF は、この関数では等しくないと見なされます。

関連関数

[OCIErrorGet\(\)](#)、[OCIRefAssign\(\)](#)

OCIRefIsNull()

用途

REF が NULL であるかどうかをテストします。

構文

```
boolean OCIRefIsNull ( OCIEnv          *env,  
                      CONST OCIRef    *ref );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

ref (IN)

NULL であるかテストする REF。

戻り値

指定された REF が NULL の場合は TRUE を戻します。それ以外の場合は FALSE を戻します。

コメント

REF は、次に該当する場合だけ NULL になります。

- 永続オブジェクトを参照するが、オブジェクトの識別子が NULL である場合。
- 一時オブジェクトを参照するが、現時点ではオブジェクトを指していない場合。

注意: 指すオブジェクトが存在しない場合、REF はぶら下がり REF です。

関連関数

[OCIErrorGet\(\)](#)、[OCIRefClear\(\)](#)

OCIRefToHex()

用途

REF を 16 進文字列に変換します。

構文

```
sword OCIRefToHex ( OCIEnv          *env,  
                  OCIError        *err,  
                  CONST OCIRef     *ref,  
                  text             *hex,  
                  ub4              *hex_length );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

ref (IN)

16 進文字列に変換する REF。ref が NULL REF (つまり、*OCIRefsIsNull(ref) == TRUE*) の場合、0 (ゼロ) *hex_length* 値が戻されます。

hex (OUT)

結果の 16 進文字列を含むだけのサイズを持つバッファ。文字列の内容は、コール側にとっては不透明になります。

hex_length (IN/OUT)

入力では *hex* バッファのサイズを指定し、出力では *hex* で戻される 16 進文字列実際のサイズを指定。

コメント

指定された REF を 16 進文字列に変換し、その文字列の長さを戻します。変換された文字列は、コール側にとっては不透明になります。

この関数は、指定されたバッファが変換後の文字列を保持できるだけの大きさが無い場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIRefFromHex\(\)](#)、[OCIRefHexSize\(\)](#)、[OCIRefsIsNull\(\)](#)

OCI 文字列関数

この項では、OCI 文字列関数について説明します。

表 17-7 OCI 文字列関数クイック・リファレンス

関数	用途	ページ
OCIStringAllocSize()	文字列メモリの割り当てられたサイズ (バイト単位) を取得します。	17-112 ページ
OCIStringAssign()	文字列に文字列を割り当てます。	17-113 ページ
OCIStringAssignText()	テキスト文字列を文字列に代入します。	17-114 ページ
OCIStringPtr()	文字列ポインタを取得します。	17-115 ページ
OCIStringResize()	文字列メモリーをサイズ変更します。	17-116 ページ
OCIStringSize()	文字列サイズを取得します。	17-117 ページ

OCIStringAllocSize()

用途

文字列の割当てメモリー・サイズをバイト単位で取得します。

構文

```
sword OCIStringAllocSize ( OCIEnv          *env,  
                           CONST OCIString *vs,  
                           ub4            *allocsize );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

vs (IN)

割り当てられたサイズがバイト単位で戻される文字列。*vs* は非 NULL ポインタでなければなりません。

allocsize (OUT)

文字列メモリーの割当てサイズをバイト単位で戻します。

コメント

割り当てられるサイズは、実際の文字列サイズより大きいかそれに等しくなります。

関連関数

[OCIErrorGet\(\)](#)、[OCIStringResize\(\)](#)、[OCIStringSize\(\)](#)

OCIStringAssign()

用途

文字列を別の文字列に割り当てます。

構文

```
sword OCIStringAssign ( OCIEnv          *env,
                        OCIError       *err,
                        CONST OCIString *rhs,
                        OCIString      **lhs );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

rhs (IN)

割当ての右辺 (ソース)。

lhs (IN/OUT)

割当ての左辺 (ターゲット)。

コメント

rhs 文字列を *lhs* 文字列に割り当てます。*lhs* 文字列のサイズは、*rhs* のサイズに応じて変更されます。割り当てられる文字列は、NULL で終わります。長さのフィールドには、NULL 終了に必要な余分なバイトは含まれません。

この関数は、割当て操作で領域が足りなくなった場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIStringAssignText\(\)](#)

OCIStringAssignText()

用途

ソース・テキスト文字列をターゲット文字列に割り当てます。

構文

```
sword OCIStringAssignText ( OCIEnv          *env,  
                           OCIError       *err,  
                           CONST text     *rhs,  
                           ub2            rhs_len,  
                           OCIString      **lhs );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

rhs (IN)

割当ての右辺（ソース）となる文字列。

rhs_len (IN)

rhs 文字列の長さ。

lhs (IN/OUT)

割当ての左辺（ターゲット）。

コメント

rhs 文字列を *lhs* 文字列に割り当てます。*lhs* 文字列のサイズは、*rhs* のサイズに応じて変更されます。割り当てられる文字列は、NULL で終わります。長さのフィールドには、NULL 終了に必要な余分なバイトは含まれません。

関連関数

[OCIErrorGet\(\)](#)、[OCIStringAssign\(\)](#)

OCIStringPtr()

用途

指定の文字列のテキストへのポインタを取得します。

構文

```
text *OCIStringPtr ( OCIEnv          *env,  
                    CONST OCIString *vs );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

vs (IN)

この文字列へのポインタを戻します。

コメント

なし

関連関数

[OCIErrorGet\(\)](#)、[OCIStringAssign\(\)](#)

OCIStringResize()

用途

指定の文字列のメモリー・サイズを変更します。

構文

```
sword OCIStringResize ( OCIEnv          *env,  
                       OCIError       *err,  
                       ub4             new_size,  
                       OCIString      **str );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

new_size (IN)

バイト単位での文字列の新規メモリー・サイズ。*new_size* には、文字列終了記号としての NULL 文字 ('\0 ') 用の領域を含める必要があります。

str (IN/OUT)

OCI オブジェクト・キャッシュから解放された文字列の割当てメモリー。

コメント

この関数は、オブジェクト・キャッシュ内の指定された可変長文字列のメモリー・サイズを変更します。文字列の内容は保持されません。この関数は、新しいメモリー領域に文字列を割り当てることがありますが、この場合指定の文字列が占有していたメモリーは解放されません。*str* が NULL の場合、この関数は文字列のメモリーを割り当てます。*new_size* が 0 の場合、この関数は、*str* が占有していたメモリーを解放し、NULL ポインタ値を戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCIStringAllocSize\(\)](#)、[OCIStringSize\(\)](#)

OCIStringSize()

用途

指定の文字列のサイズをバイト単位で取得します。

構文

```
ub4 OCIStringSize ( OCIEnv          *env,  
                   CONST OCIString *vs );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

vs (IN)

サイズを調べる文字列。

コメント

戻されたサイズには、NULL 終了用の余分なバイトは含まれていません。

関連関数

[OCIErrorGet\(\)](#)、[OCIStringResize\(\)](#)

OCI 表関数

この項では、OCI 表関数について説明します。

表 17-8 OCI 表関数クイック・リファレンス

関数	用途	ページ
OCI Table Delete()	要素を削除します。	17-119 ページ
OCI Table Exists()	要素が存在するかどうかをテストします。	17-120 ページ
OCI Table First()	表の先頭の索引を戻します。	17-121 ページ
OCI Table Last()	表の最後の索引を戻します。	17-122 ページ
OCI Table Next()	利用可能な次の表索引を戻します。	17-123 ページ
OCI Table Prev()	利用可能なひとつ前の表索引を戻します。	17-124 ページ
OCI Table Size()	表の現在のサイズを戻します。	17-125 ページ

OCITableDelete()

用途

指定の索引位置にある要素を削除します。

構文

```
sword OCITableDelete ( OCIEnv          *env,  
                      OCIError       *err,  
                      sb4             index,  
                      OCITable       *tbl );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

index (IN)

削除する必要がある要素の索引。

tbl (IN)

要素を削除する表。

コメント

この関数は、指定した索引の要素がすでに削除されている場合、または指定した索引が表に対して無効の場合には、エラーを戻します。入力パラメータに NULL がある場合もエラーとなります。

注意: 表の残りの要素の位置序数は、[OCITableDelete\(\)](#) で変更されません。この削除操作により、表の中に空きが生成されます。

関連関数

[OCIErrorGet\(\)](#)、[OCITableExists\(\)](#)

OCI`TableExists()`

用途

指定の索引位置に要素が存在するかどうかをテストします。

構文

```
sword OCITableExists ( OCIEnv          *env,  
                       OCIError       *err,  
                       CONST OCITable *tbl,  
                       sb4             index,  
                       boolean        *exists );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCI`EnvCreate()`」および「OCI`Initialize()`」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI`_ERROR` が戻されます。OCI`ErrorGet()` をコールして診断情報を取得します。

tbl (IN)

指定の索引をチェックする表。

index (IN)

要素の存在をチェックする索引。

exists (OUT)

指定する *index* に要素が存在する場合は TRUE。それ以外の場合は、FALSE。

コメント

この関数は、入力パラメータに NULL がある場合はエラーを戻します。

関連関数

[OCI`ErrorGet\(\)`](#)、[OCI`TableDelete\(\)`](#)

OCITableFirst()

用途

指定の表の先頭の既存要素の索引を戻します。

構文

```
sword OCITableFirst ( OCIEnv          *env,  
                     OCIError       *err,  
                     CONST OCITable *tbl,  
                     sb4            *index );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

tbl (IN)

スキャンする表。

index (OUT)

指定の表に存在する最初の要素の索引が戻されます。

コメント

たとえば、[OCITableDelete\(\)](#) によって表の先頭の 5 つの要素が削除されている場合は、[OCITableFirst\(\)](#) は 6 を戻します。表内のデータのない空きの詳細は、[OCITableDelete\(\)](#) を参照してください。

この関数は、表が空の場合はエラーを戻します。

関連関数

[OCIErrorGet\(\)](#)、[OCITableDelete\(\)](#)、[OCITableLast\(\)](#)

OCITableLast()

用途

表の最後の既存要素の索引を戻します。

構文

```
sword OCITableLast ( OCIEnv          *env,  
                    OCIError        *err,  
                    CONST OCITable  *tbl,  
                    sb4              *index );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

tbl (IN)

スキャンする表。

index (OUT)

表の最後の既存要素の索引。

コメント

この関数は、表が空の場合はエラーを戻します。

関連関数

OCIErrorGet()、*OCITableFirst()*、*OCITableNext()*、*OCITablePrev()*

OCITableNext()

用途

表の次の既存要素の索引を戻します。

構文

```
sword OCITableNext ( OCIEnv          *env,
                    OCIError        *err,
                    sb4              index,
                    CONST OCITable  *tbl,
                    sb4              *next_index
                    boolean          *exists );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「[OCIEnvCreate\(\)](#)」および「[OCIInitialize\(\)](#)」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、OCI_ERROR が戻されます。[OCIErrorGet\(\)](#) をコールして診断情報を取得します。

index (IN)

スキヤンの開始位置の索引。

tbl (IN)

スキヤンする表。

next_index (OUT)

tbl(index) の次の既存要素の索引。

exists (OUT)

次の索引が存在しない場合は FALSE、存在する場合は TRUE。

コメント

index よりも大きく *exists(j)* が TRUE になるような最小位置 *j* を戻します。

関連項目 : 表内のデータの不在の存在の詳細は、「[OCIStringAllocSize\(\)](#)」の説明を参照してください。

関連関数

[OCIErrorGet\(\)](#)、[OCITablePrev\(\)](#)

OCI`TablePrev()`

用途

表の前の既存要素の索引を戻します。

構文

```
sword OCITablePrev ( OCIEnv           *env,  
                    OCIError        *err,  
                    sb4               index,  
                    CONST OCITable  *tbl,  
                    sb4               *prev_index  
                    boolean           *exists );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「OCI`EnvCreate()`」および「OCI`Initialize()`」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、`err` に記録され、OCI`_ERROR` が戻されます。OCI`ErrorGet()` をコールして診断情報を取得します。

index (IN)

スキャンの開始位置の索引。

tbl (IN)

スキャンする表。

prev_index (OUT)

`tbl(index)` の直前の既存要素の索引。

exists (OUT)

前の索引が存在しない場合は FALSE、存在する場合は TRUE に設定されます。

コメント

`index` よりも小さく `exists(j)` が TRUE になるような最大位置 `j` を戻します。

関連項目 : 表内のデータの無い空きの存在の詳細は、「OCI`StringAllocSize()`」の説明を参照してください。

関連関数

[OCI`TableNext\(\)`](#)

OCITableSize()

用途

指定の表のサイズを戻します。削除済みの要素は含まれません。

構文

```
sword OCITableSize ( OCIEnv          *env,
                    OCIError        *err,
                    CONST OCITable   *tbl,
                    sb4              *size );
```

パラメータ

env (IN/OUT)

オブジェクト・モードで初期化された OCI 環境ハンドル。詳細は、第 15 章の「*OCIEnvCreate()*」および「*OCIInitialize()*」の説明を参照してください。

err (IN/OUT)

OCI エラー・ハンドル。エラーがある場合は、*err* に記録され、*OCI_ERROR* が戻されます。*OCIErrorGet()* をコールして診断情報を取得します。

tbl (IN)

要素の数が戻されるネストした表。

size (OUT)

ネストした表内の現在の要素数。削除された要素はカウントに含まれません。

コメント

カウントは、ネストした表から要素を削除すると減分されます。したがって、このカウントには要素の削除によって生成された空きは含まれていません。削除した要素を含まないカウントを得るには、*OCICollSize()* を使います。

たとえば、次のとおりです。

```
OCITableSize(...);
// assume 'size' returned is equal to 5
OCITableDelete(...); // delete one element
OCITableSize(...);
// 'size' returned is equal to 4
```

削除された要素を含めたカウントを取得するには、*OCICollSize()* を使用します。上記の例を続けます。

```
OCICollSize(...)
// 'size' returned is still equal to 5
```

この関数は、ネストした表のオブジェクト・キャッシュへのロード中にエラーが発生した場合または入力パラメータに NULL がある場合はエラーを戻します。

関連関数

[OCICollSize\(\)](#)

18

OCI 外部プロシージャ関数

この章には、次の項目があります。

- [概要](#)
- [OCI 外部プロシージャ関数](#)

概要

この章では、OCI 外部プロシージャ関数について説明します。これらの関数により、外部プロシージャのユーザーは、エラーの通知、メモリーの割当ておよび OCI コンテキスト情報の取得が行えます。これらの関数の使用方法の詳細は、『Oracle8i アプリケーション開発者ガイド - 基礎編』を参照してください。

関数の構文

各関数について、次の情報をリストします。

用途

この関数によって実行されるアクションの簡単な説明。

構文

この関数をコールするための構文を示すコードの断片で、パラメータの順序と種類が含まれています。

パラメータ

関数の各パラメータの説明。これにはパラメータのモードが含まれます。パラメータのモードには、以下のように 3 つの可能な値があります。

モード	説明
IN	Oracle にデータを渡すパラメータ
OUT	このコールまたは後続のコールで Oracle からデータを受け取るパラメータ
IN/OUT	このコールでデータを渡し、このコールまたは後続のコールからの戻りでデータを受け取るパラメータ

コメント

この関数に関する詳細情報（ある場合）、関数の使用上の制約やアプリケーション内でこの関数を使用するときに役に立つ情報が記載されています。

戻り値

この関数に対する可能戻り値の一覧です。

例

説明の対象となっている関数コールの使用方法を例示するコード（全体または一部）、すべての関数に例が記載されているわけではありません。

関連関数

関連する関数コールの一覧です。

リターン・コード

成功とエラーのリターン・コードは、特定の外部プロシージャ・インタフェース関数に対して定義されます。特定のインタフェース関数が `OCIEXTPROC_SUCCESS` または `OCIEXTPROC_ERROR` を戻した場合、アプリケーションはこれらのマクロを使用して戻り値をチェックする必要があります。

- `OCIEXTPROC_SUCCESS` - 外部プロシージャの成功した場合のリターン・コード
- `OCIEXTPROC_ERROR` - 外部プロシージャの失敗した場合のリターン・コード

With_Context 型

PL/SQL 外部プロシージャへの C コール可能インタフェースには、`with_context` パラメータを渡す必要があります。この構造体の型は、`OCIExtProcContext` で、ユーザーには不透明です。

ユーザーは、`with_context` パラメータをアプリケーションで次のように宣言できます。

```
OCIExtProcContext *with_context;
```

OCI 外部プロシージャ関数

この章の残りの部分では、C に対応する OCI 外部プロシージャ関数について説明します。

表 18-1 OCI 外部プロシージャ関数クイック・リファレンス

関数	用途	ページ
OCIExtProcAllocCallMemory()	外部プロシージャの継続中にメモリーを割り当てます。	18-5 ページ
OCIExtProcRaiseExcp()	PL/SQL に例外を通知します。	18-6 ページ
OCIExtProcRaiseExcpWithMsg()	例外をメッセージ付きで通知します。	18-7 ページ
OCIExtProcGetEnv()	OCI 環境ハンドル、サービス・コンテキスト・ハンドルおよびエラー・ハンドルを取得します。	18-8 ページ

OCIExtProcAllocCallMemory()

用途

外部プロシージャの継続中は N バイトのメモリーを割り当てます。

構文

```
dvoid * OCIExtProcAllocCallMemory ( OCIExtProcContext    *with_context,  
                                     size_t              amount )
```

パラメータ

with_context (IN)

C 外部プロシージャに渡される `with_context` ポインタ。18-3 ページの「[With_Context 型](#)」を参照してください。

amount (IN)

割り当てるバイト数。

コメント

このコールは、外部プロシージャのコール継続中に `amount` バイトのメモリーを割り当てません。

このコールで割り当てられたメモリーは、外部プロシージャから戻るとすぐに PL/SQL によって解放されます。アプリケーションでは、`OCIExtProcAllocCallMemory()` で割り当てたメモリーに解放関数を使用しないでください。この関数を使用して、関数の戻り値用のメモリーを割り当ててください。

0 (ゼロ) の戻り値はエラーとして扱われます。

戻り値

割り当てたメモリーを示すタイプ化されていない (不透明な) ポインタ。

例

```
text *ptr = (text *)OCIExtProcAllocCallMemory(wctx, 1024)
```

関連関数

[OCIErrorGet\(\)](#)

OCIExtProcRaiseExcp()

用途

PL/SQL に例外を通知します。

構文

```
size_t OCIExtProcRaiseExcp ( OCIExtProcContext  *with_context,  
                             int                errnum )
```

パラメータ

with_context (IN)

C 外部プロシージャに渡される **with_context** ポインタ。18-3 ページの「[With_Context 型](#)」を参照してください。

errnum (IN)

PL/SQL に通知する Oracle エラー番号。*errnum* は、1 ~ 32767 の正数でなければなりません。

コメント

この関数をコールすると、例外が PL/SQL に通知されます。この関数が問題なく戻されると、外部プロシージャは終了処理を開始し、PL/SQL に戻ります。いったん例外が PL/SQL に通知されると、IN/OUT 引数と OUT 引数（ある場合）は処理されません。

戻り値

この関数は、コールが成功した場合は、OCIEXTPROC_SUCCESS を戻します。コールが失敗した場合は、OCIEXTPROC_ERROR を戻します。

関連関数

[OCIExtProcRaiseExcpWithMsg\(\)](#)

OCIExtProcRaiseExcpWithMsg()

用途

例外をメッセージ付きで通知します。

構文

```
size_t OCIExtProcRaiseExcpWithMsg ( OCIExtProcContext  *with_context,  
                                     int                errnum,  
                                     char                *errmsg,  
                                     size_t              msglen )
```

パラメータ

with_context (IN)

C 外部プロシージャに渡される `with_context` ポインタ。18-3 ページの「[With_Context 型](#)」を参照してください。

errnum (IN)

PL/SQL へ信号を送る Oracle エラー番号。 `errnum` は、1 ~ 32767 の正数でなければなりません。

errmsg (IN)

`errnum` に関連付けられたエラー・メッセージ。

len (IN)

エラー・メッセージの長さ。 `errmsg` が NULL で終わる文字列の場合は、0 (ゼロ) が渡されます。

コメント

PL/SQL に例外が呼び出されます。さらに、後続のエラー・メッセージ文字列を標準の Oracle エラー・メッセージ内で代入します。詳細は、`OCIExtProcRaiseExcp()` の説明を参照してください。

戻り値

この関数は、コールが成功した場合は、`OCIEXTPROC_SUCCESS` を戻します。コールが失敗した場合は、`OCIEXTPROC_ERROR` を戻します。

関連関数

[OCIExtProcRaiseExcp\(\)](#)

OCIExtProcGetEnv()

用途

OCI 環境、サービス・コンテキストおよびエラー・ハンドルを取得します。

構文

```
sword OCIExtProcGetEnv ( OCIExtProcContext  *with_context,  
                        OCIEnv              envh,  
                        OCISvcCtx          svch,  
                        OCIError           errh )
```

パラメータ

with_context (IN)

C 外部プロシージャに渡される with_context ポインタ。18-3 ページの「[With_Context 型](#)」を参照してください。

envh (OUT)

OCI 環境ハンドル。

svch (OUT)

OCI サービス・ハンドル。

errh (OUT)

OCI エラー・ハンドル。

コメント

この関数の主な目的は、OCI コールバックで同じトランザクションのデータベースを使用できるようにすることです。この関数で取得する OCI ハンドルは、データベースに対する OCI コールバックで使用しなければなりません。これらのハンドルを標準の OCI コールを介して取得した場合、これらのハンドルはデータベースへの新規接続を使用するので、同一トランザクション内のコールバックには使用できません。1 つの外部プロシージャで使用できるのは、コールバックと新規接続のどちらか一方で、両方は使用できません。

戻り値

この関数は、コールが成功した場合は、OCI_SUCCESS を戻します。コールが失敗した場合は、OCI_ERROR を戻します。

関連関数

[OCIEnvCreate\(\)](#)、[OCIAttrGet\(\)](#)、[OCIHandleAlloc\(\)](#)

第 IV 部

付録

このマニュアルの第 4 部には、次の付録が含まれています。

- [付録 A の「ハンドルおよび記述子の属性」](#)には、各種の OCI ハンドルの属性のリストを示します。
- [付録 B の「OCI デモ・プログラム」](#)には、OCI 機能のコード例の重要なデモ・プログラムを示します。
- [付録 C の「OCI 関数のサーバー・ラウンドトリップ」](#)では、ほとんどの OCI 関数に必要なサーバー・ラウンドトリップの詳細を説明します。

ハンドルおよび記述子の属性

この付録では、OCI のハンドルおよび記述子の属性を説明します。この属性は `OCIAttrGet()` で読み込み、`OCIAttrSet()` 関数で変更することができます。

- 表記法
- 環境ハンドルの属性
- エラー・ハンドルの属性
- サービス・コンテキスト・ハンドルの属性
- サーバー・ハンドルの属性
- ユーザー・セッション・ハンドルの属性
- トランザクション・ハンドルの属性
- 文ハンドルの属性
- バインド・ハンドルの属性
- 定義ハンドルの属性
- 記述ハンドルの属性
- パラメータ記述子の属性
- LOB ロケータの属性
- 複合オブジェクトの属性
- アドバンスド・キューイング記述子の属性
- サブスクリプション・ハンドルの属性
- ダイレクト・パス・ロード・ハンドルの属性
- プロセス・ハンドルの属性

表記法

それぞれのハンドル型で、読みまたは変更が可能な属性をリストします。各属性リストには、次の情報が含まれています。

モード

使われるモードは次の3つです。

読み - *OCIAttrGet()* を使用して属性を読み込むことができます。

書込み - *OCIAttrSet()* を使用して属性を変更することができます。

読み / 書込み - *OCIAttrGet()* を使用して属性を読み込み、*OCIAttrSet()* を使用して属性を変更することができます。

説明

これは、属性の用途の説明です。

属性データ型

これは、属性のデータ型です。必要に応じ、読みモードと書込みモードのデータ型が区別されます。

可能な値

一定の値だけが実現される場合には、その値をここに表示します。

例

一部の属性については例が記載されています。

環境ハンドルの属性

OCI_ATTR_CACHE_ARRAYFLUSH

モード

読み込み / 書き込み

説明

この属性が TRUE に設定されている場合は、*OCICacheFlush()* のコールによって、共通の表に属するオブジェクトと一緒にフラッシュされるため、パフォーマンスを大幅に向上することができます。このモードは、オブジェクトのフラッシュの順序が重要でない場合にだけ使用します。このモードの間は、オブジェクトに使用済みマークが設定される順序が保持される保証はありません。詳細は、13-5 ページの「[オブジェクト・キャッシュ・パラメータ](#)」および 13-10 ページの「[変更をサーバーにフラッシュ](#)」を参照してください。

属性データ型

boolean

OCI_ATTR_CACHE_MAX_SIZE

モード

読み込み / 書き込み

説明

最適サイズのパーセンテージとして、クライアント側オブジェクト・キャッシュの最大サイズ（高水位標）を設定します。デフォルト値は 10% です。詳細は、13-5 ページの「[オブジェクト・キャッシュ・パラメータ](#)」を参照してください。

属性データ型

ub4 *

OCI_ATTR_CACHE_OPT_SIZE

モード

読み込み / 書き込み

説明

クライアント側オブジェクトのキャッシュの最適サイズをバイト数で設定します。デフォルト値は 200KB です。詳細は、13-5 ページの「[オブジェクト・キャッシュ・パラメータ](#)」を参照してください。

属性データ型

ub4 *

OCI_ATTR_OBJECT

モード
読み込み

説明
環境がオブジェクト・モードで初期化されていた場合は、TRUE を戻します。

属性データ型
boolean *

OCI_ATTR_PINOPTION

モード
読み込み / 書き込み

説明
この属性は、環境ハンドルに関連付けられたアプリケーションに OCI_PIN_DEFAULT の値を設定します。

たとえば、OCI_ATTR_PINOPTION が OCI_PIN_RECENT に設定されている場合に、*pin_option* パラメータが OCI_PIN_DEFAULT に設定されている *OCIObjectPin()* をコールすると、オブジェクトは OCI_PIN_RECENT モードで確保されます。

属性データ型
OCIPinOpt *

OCI_ATTR_ALLOC_DURATION

モード
読み込み / 書き込み

説明
この属性は、環境ハンドルに関連付けられたアプリケーションの割当て継続時間に OCI_DURATION_DEFAULT の値を設定します。

属性データ型
OCIDuration *

OCI_ATTR_PIN_DURATION

モード
読み込み / 書き込み

説明

この属性は、環境ハンドルに関連付けられたアプリケーションの確保継続時間に OCI_DURATION_DEFAULT の値を設定します。

属性データ型

OCIDuration *

OCI_ATTR_HEAPALLOC**モード**

読みみ

説明

環境ハンドルから割り当てられたメモリの現行サイズです。これは、アプリケーション内でメモリーが最も使用されている部分を追跡するのに役立ちます。

属性データ型

ub4 *

OCI_ATTR_OBJECT_NEWNOTNULL**モード**

読みみ / 書込み

説明

この属性が TRUE に設定されている場合は、新規作成されるオブジェクトに非 NULL 属性が設定されます。詳細は、10-30 ページの「[オブジェクトの作成](#)」を参照してください。

属性データ型

boolean *

OCI_ATTR_OBJECT_DETECTCHANGE**モード**

読みみ / 書込み

説明

この属性が TRUE に設定されている場合は、コミットされた別のトランザクションによりサーバー内で修正されたオブジェクトのフラッシュを試行すると、アプリケーションが ORA-08179 エラーを受け取ります。

詳細は、13-13 ページの「[最適ロックの実現](#)」を参照してください。

属性データ型

boolean *

OCI_ATTR_SHARED_HEAP_ALLOC

モード

読み込み

説明

共有プールから現在割り当てられているメモリのサイズを戻します。この属性はどの環境ハンドルでも使用できますが、有効な値を戻すには、プロセスを共有モードで初期化する必要があります。この属性は次のように読み込まれます。

```
ub4 heapsz = 0;
OCIAttrGet((dvoid *)envhp, (ub4)OCI_HTYPE_ENV,
           (dvoid *) &heapsz, (ub4 *) 0,
           (ub4)OCI_ATTR_SHARED_HEAPALLOC, errhp);
```

属性データ型

ub4 *

エラー・ハンドルの属性

OCI_ATTR_DML_ROW_OFFSET

モード

読み込み

説明

エラーが発生した（DML 配列の）オフセットを戻します。

属性データ型

ub4 *

サービス・コンテキスト・ハンドルの属性

OCI_ATTR_ENV

モード
読み込み

説明
サービス・コンテキストに関連付けられた環境コンテキストを戻します。

属性データ型
OCIEnv **

OCI_ATTR_SERVER

モード
読み込み / 書き込み

説明
読み込みモードでは、サービス・コンテキストのサーバー・コンテキスト属性を指すポインタを戻します。

書き込みモードでは、サービス・コンテキストのサーバー・コンテキスト属性を設定します。

属性データ型
OCIServer ** (読み込み) / OCIServer * (書き込み)

OCI_ATTR_SESSION

モード
読み込み / 書き込み

説明
読み込みモードでは、サービス・コンテキストの認証コンテキスト属性を指すポインタを戻します。

書き込みモードでは、サービス・コンテキストの認証コンテキスト属性を設定します。

属性データ型
OCISession ** (読み込み) / OCISession * (書き込み)

OCI_ATTR_TRANS

モード

読み込み / 書き込み

説明

読み込みモードでは、サービス・コンテキストのトランザクション・コンテキスト属性を指すポインタを戻します。

書き込みモードでは、サービス・コンテキストのトランザクション・コンテキスト属性を設定します。

属性データ型

OCITrans ** (読み込み) / OCITrans * (書き込み)

OCI_ATTR_IN_V8_MODE

モード

読み込み

説明

アプリケーションが (OCI`SvcCtxToLda()` コールによって) Oracle リリース 7 モードに切り替えられているかどうかを判別できます。戻り値がゼロ以外 (TRUE) であれば、アプリケーションは現在 Oracle リリース 8 モードで実行されており、戻り値がゼロ (FALSE) であればアプリケーションは現在 Oracle リリース 7 モードで実行されています。

属性データ型

ub1 *

例

次のコード・サンプルで、このパラメータの使用法を示します。

```
in_v8_mode = 0;
OCIAttrGet ((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX, (ub1 *)&in_v8_mode,
            (ub4) 0, OCI_ATTR_IN_V8_MODE, errhp);

if (in_v8_mode)
    fprintf (stdout, "In V8 mode\n");
else
    fprintf (stdout, "In V7 mode\n");
```

サーバー・ハンドルの属性

OCI_ATTR_NONBLOCKING_MODE

モード

読み込み / 書き込み

説明

この属性は、ブロック化モードを判別します。

サーバー・コンテキストが非ブロック化モードであれば、読み込み時にこの属性値は TRUE を戻します。設定すると、非ブロック化モード属性が切り替えられます。詳細は、2-35 ページの「[非ブロック化モード](#)」を参照してください。

属性データ型

属性値はタイプ `ub1` です。

OCI_ATTR_ENV

モード

読み込み

説明

サーバー・コンテキストに関連付けられた環境コンテキストを戻します。

属性データ型

OCIEnv **

OCI_ATTR_EXTERNAL_NAME

モード

読み込み / 書き込み

説明

外部名は使いやすいグローバルな名前です。これは `sys.props$.value$` に格納されています (name = 'GLOBAL_DB_NAME')。外部名は、すべてのデータベースでネットワーク・ディレクトリ・サービスを使って登録しない限り、一意である保証はありません。

分散トランザクションを調整する場合に、データベース名をサーバーと交換できます。サーバー・データベース名へは、`OCISessionBegin()` コールが発行されたときにデータベースが開いていた場合に限り、アクセスすることができます。

属性データ型

text ** (読み込み) / text * (書き込み)

OCI_ATTR_INTERNAL_NAME

モード

読み込み / 書き込み

説明

グローバル・トランザクションの実行時に記録されるクライアント・データベース名を設定します。この名前は、故障のために準備状態で保留になっている可能性があるトランザクションを追跡するために、DBA で使用できます。

属性データ型

text ** (読み込み) / text * (書き込み)

OCI_ATTR_IN_V8_MODE

モード

読み込み

説明

アプリケーションが (*OCISvcCtxToLda()* コールによって) Oracle リリース 7 モードに切り替えられているかどうかを判別できます。戻り値がゼロ以外 (TRUE) であれば、アプリケーションは現在 Oracle リリース 8 モードで実行されており、戻り値がゼロ (FALSE) であればアプリケーションは現在 Oracle リリース 7 モードで実行されています。

属性データ型

ub1 *

OCI_ATTR_FOCBK

モード

読み込み / 書き込み

説明

詳細は、9-17 ページの「[アプリケーション・フェイルオーバー・コールバック](#)」を参照してください。

属性データ型

OCIFocbkStruct *

OCI_ATTR_SERVER_GROUP

モード

読み込み / 書き込み

説明

サーバー・グループを指定する、30文字以内の英数字文字列。詳細は、8-10ページの「[パスワードおよびセッションの管理](#)」を参照してください。

属性データ型

ub4

ユーザー・セッション・ハンドルの属性

OCI_ATTR_USERNAME

モード
書込み

説明
認証に使用するユーザー名を指定します。

属性データ型
text *

OCI_ATTR_MIGSESSION

モード
読込み / 書込み

説明
セッション・ハンドル用に識別されたセッションを指定します。同じプロセス内または複数プロセス間で、セッションをある環境から別の環境へ複製することができます。これは、同じマシン上のプロセスでも異なるマシン上のプロセスでも可能です。複製するセッションは、移行可能と認証されている必要があります。詳細は、8-10 ページの「[パスワードおよびセッションの管理](#)」を参照してください。

属性データ型
ub1 *

例
次のコード・サンプルで、この属性の使用方法を示します。

```
OCIAttrSet ((dvoid *) authp, (ub4) OCI_HTYPE_SESSION, (dvoid *) mig_session,  
(ub4) sz, (ub4) OCI_ATTR_MIGSESSION, errhp);
```

OCI_ATTR_PASSWORD

モード
書込み

説明
認証に使用するパスワードを指定します。

属性データ型
text *

トランザクション・ハンドルの属性

OCI_ATTR_TRANS_NAME

モード

読み込み / 書き込み

説明

この属性を使用して、トランザクションを識別するテキスト文字列の構築または読み込みができません。これは、XID を使用してトランザクションを識別するかわりに使用される方法です。テキスト文字列は、64 バイトまで指定できます。

属性データ型

text ** (読み込み) / text * (書き込み)

OCI_ATTR_XID

モード

読み込み / 書き込み

説明

トランザクションを識別する XID の設定または読み込みができません。

属性データ型

XID ** (読み込み) / XID * (書き込み)

文ハンドルの属性

OCI_ATTR_NUM_DML_ERRORS

モード
読み込み

説明
DML 操作でのエラーの数を返します。

属性データ型
ub4 *

OCI_ATTR_ROW_COUNT

モード
読み込み

説明
これまでに処理した行の数を返します。デフォルト値は 1 です。

属性データ型
ub4 *

OCI_ATTR_SQLFNCODE

モード
読み込み

説明
文に関連付けられた SQL コマンドの関数コードを返します。

属性データ型
ub2 *

注意
SQL コマンド・コードのリストを A-15 ページの表 A-1 の「SQL コマンド・コード」に示します。

表 A-1 SQL コマンド・コード

コード	SQL 関数	コード	SQL 関数	コード	SQL 関数
01	CREATE TABLE	43	DROP EXTERNAL DATABASE	85	TRUNCATE TABLE
02	SET ROLE	44	CREATE DATABASE	86	TRUNCATE CLUSTER
03	INSERT	45	ALTER DATABASE	87	CREATE BITMAPFILE
04	SELECT	46	CREATE ROLLBACK SEGMENT	88	ALTER VIEW
05	UPDATE	47	ALTER ROLLBACK SEGMENT	89	DROP BITMAPFILE
06	DROP ROLE	48	DROP ROLLBACK SEGMENT	90	SET CONSTRAINTS
07	DROP VIEW	49	CREATE TABLESPACE	91	CREATE FUNCTION
08	DROP TABLE	50	ALTER TABLESPACE	92	ALTER FUNCTION
09	DELETE	51	DROP TABLESPACE	93	DROP FUNCTION
10	CREATE VIEW	52	ALTER SESSION	94	CREATE PACKAGE
11	DROP USER	53	ALTER USER	95	ALTER PACKAGE
12	CREATE ROLE	54	COMMIT (WORK)	96	DROP PACKAGE
13	CREATE SEQUENCE	55	ROLLBACK	97	CREATE PACKAGE BODY
14	ALTER SEQUENCE	56	SAVEPOINT	98	ALTER PACKAGE BODY
15	(使用されていません)	57	CREATE CONTROL FILE	99	DROP PACKAGE BODY
16	DROP SEQUENCE	58	ALTER TRACING	157	CREATE DIRECTORY
17	CREATE SCHEMA	59	CREATE TRIGGER	158	DROP DIRECTORY
18	CREATE CLUSTER	60	ALTER TRIGGER	159	CREATE LIBRARY
19	CREATE USER	61	DROP TRIGGER	160	CREATE JAVA
20	CREATE INDEX	62	ANALYZE TABLE	161	ALTER JAVA
21	DROP INDEX	63	ANALYZE INDEX	162	DROP JAVA
22	DROP CLUSTER	64	ANALYZE CLUSTER	163	CREATE OPERATOR
23	VALIDATE INDEX	65	CREATE PROFILE	164	CREATE INDEXTYPE
24	CREATE PROCEDURE	66	DROP PROFILE	165	DROP INDEXTYPE
25	ALTER PROCEDURE	67	ALTER PROFILE	166	ALTER INDEXTYPE

表 A-1 SQL コマンド・コード (続き)

コード	SQL 関数	コード	SQL 関数	コード	SQL 関数
26	ALTER TABLE	68	DROP PROCEDURE	167	DROP OPERATOR
27	EXPLAIN	69	(使用されていません)	168	ASSOCIATE STATISTICS
28	GRANT	70	ALTER RESOURCE COST	169	DISASSOCIATE STATISTICS
29	REVOKE	71	CREATE SNAPSHOT LOG	170	CALL METHOD
30	CREATE SYNONYM	72	ALTER SNAPSHOT LOG	171	CREATE SUMMARY
31	DROP SYNONYM	73	DROP SNAPSHOT LOG	172	ALTER SUMMARY
32	ALTER SYSTEM SWITCH LOG	74	CREATE SNAPSHOT	173	DROP SUMMARY
33	SET TRANSACTION	75	ALTER SNAPSHOT	174	CREATE DIMENSION
34	PL/SQL EXECUTE	76	DROP SNAPSHOT	175	ALTER DIMENSION
35	LOCK	77	CREATE TYPE	176	DROP DIMENSION
36	NOOP	78	DROP TYPE	177	CREATE CONTEXT
37	RENAME	79	ALTER ROLE	178	DROP CONTEXT
38	COMMENT	80	ALTER TYPE	179	ALTER OUTLINE
39	AUDIT	81	CREATE TYPE BODY	180	CREATE OUTLINE
40	NOAUDIT	82	ALTER TYPE BODY	181	DROP OUTLINE
41	ALTER INDEX	83	DROP TYPE BODY	182	UPDATE INDEXES
42	CREATE EXTERNAL DATABASE	84	DROP LIBRARY	183	ALTER OPERATOR

OCI_ATTR_ENV**モード**

読み込み

説明

文に関連付けられた環境コンテキストを戻します。

属性データ型

OCIEnv **

OCI_ATTR_STMT_TYPE

モード
読み込み

説明

ハンドルに関連付けられた文のタイプ。可能な値は次のとおりです。

- OCI_STMT_SELECT
- OCI_STMT_UPDATE
- OCI_STMT_DELETE
- OCI_STMT_INSERT
- OCI_STMT_CREATE
- OCI_STMT_DROP
- OCI_STMT_ALTER
- OCI_STMT_BEGIN (PL/SQL 文)
- OCI_STMT_DECLARE (PL/SQL 文)

属性データ型
ub2 *

OCI_ATTR_ROWID

モード
読み込み

説明

OCIDescriptorAlloc() によって割り当てられる ROWID 記述子を戻します。詳細は、2-32 ページの「[位置づけ更新および位置づけ削除](#)」および 3-12 ページの「[ROWID](#)」を参照してください。

属性データ型
OCIRowid *

OCI_ATTR_PARAM_COUNT

モード
読み込み

説明

この属性を使用して、文ハンドルに関連付けられた文の選択リストにある列数を取得できません。

属性データ型

ub4 *

例

次のコード・サンプルで、この属性の使用方法を示します。

```
/* Describe of a select-list */
text *selstmt = "SELECT * FROM EMP";
ub4 paramcnt;
OCIParam *parmdp;

err = OCIStmtPrepare (stmhp, errhp, selstmt,
                     (ub4)strlen((char *)selstmt),
                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
err = OCIStmtExecute (svchp, stmhp, errhp, (ub4)1, (ub4)0,
                     (const OCISnapshot*) 0, (OCISnapshot*)0, OCI_DESCRIBE_ONLY);

/* get the number of columns in the select list */
err = OCIAttrGet ((dvoid *)stmhp, (ub4)OCI_HTYPE_STMT, (dvoid *)
                 &paramcnt, (ub4 *) 0, (ub4)OCI_ATTR_PARAM_COUNT, errhp);

/* get describe information for each column */
for (i = 0; i < paramcnt; i++) {
    OCIParamGet (dvoid *)stmhp, OCI_HTYPE_STMT, errhp, &parmdp, i);
/* get the attributes for each column */
}
```

OCI_ATTR_PREFETCH_ROWS

モード

書込み

説明

プリフェッチされるトップ・レベル行の数を設定します。デフォルト値は1行です。

属性データ型

ub4 *

OCI_ATTR_PREFETCH_MEMORY

モード
書込み

説明
プリフェッチされるトップ・レベル行のメモリー・レベルを設定します。このメモリー・レベルが、指定したメモリー使用量の制限以下である場合は、指定したトップ・レベル行数までの行がフェッチされます。デフォルト値は0です。これは、プリフェッチされた行の数の計算にメモリー・サイズが含まれていないことを意味します。

属性データ型
ub4 *

OCI_ATTR_PARSE_ERROR_OFFSET

モード
読み込み

説明
文の解析エラー・オフセットを戻します。

属性データ型

バインド・ハンドルの属性

OCI_ATTR_CHAR_COUNT

モード
書込み

説明
5-26 ページの「[文字カウント属性](#)」を参照してください。

属性データ型
ub4 *

OCI_ATTR_CHARSET_ID

モード
読み込み / 書込み

説明
バインド・ハンドルのキャラクタ・セット ID。入力データのキャラクタ・セットが UCS-2 (Unicode) の場合、ユーザーはキャラクタ・セット ID を OCI_UCS2ID に設定する必要があります。バインド値バッファは utext バッファであると見なされ、入力長ポインタおよび戻り値の長さのセマンティクスは、文字セマンティクス (utext の数) に変更されます。ただし、先行する *OCIBind* コールでは、バインド値バッファのサイズをバイト単位で指定する必要があります。詳細は、5-27 ページの「[固定幅 Unicode サポート](#)」を参照してください。

バインド・ハンドルのキャラクタ・セットを変更する場合は、データベース (スキーマ定義) 内の対応する列の最大長を明示的に指定する必要があります。これは、*OCIAttrSet()* を使用して定義ハンドルの OCI_ATTR_MAXDATA_SIZE 属性の値を指定することにより実行できます。

属性データ型
ub2 *

OCI_ATTR_CHARSET_FORM

モード
読み込み / 書込み

説明
バインド・ハンドルのキャラクタ・セット形式

属性データ型
ub1 *

OCI_ATTR_MAXDATA_SIZE

モード
読み込み / 書き込み

説明
5-25 ページの「[OCI_ATTR_MAXDATA_SIZE 属性](#)」を参照してください。

属性データ型
sb4 *

OCI_ATTR_PDPRC

モード
書き込み

説明
パック 10 進数の精度を指定します。SQLT_PDN 値の場合、精度は $2^{(\text{value_sz}-1)}$ に等しくなければなりません。SQLT_SLS 値の場合、精度は $(\text{value_sz}-1)$ に等しくなければなりません。

バインドまたは定義後に、この値は 0 (ゼロ) に初期化されます。最初に OCI_ATTR_PDPRC 属性を設定して、次に OCI_ATTR_PDSCL を設定します。これらの値のどちらかを変更する必要がある場合は、最初に再バインドまたは再定義を行って、次にこれらの属性を適切に再設定する必要があります。

属性データ型
ub2 *

OCI_ATTR_PDSCL

モード
書き込み

説明
パック 10 進値のスケールを指定します。

バインドまたは定義後に、この値は 0 (ゼロ) に初期化されます。最初に OCI_ATTR_PDPRC 属性を設定して、次に OCI_ATTR_PDSCL を設定します。これらの値のどちらかを変更する必要がある場合は、最初に再バインドまたは再定義を行って、次にこれらの属性を適切に再設定する必要があります。

属性データ型
sb2 *

OCI_ATTR_ROWS_RETURNED

モード
読み込み

説明

この属性は、RETURNING 句で DML 文をバインディングする OUT のコールバック関数を実行している時、現在の繰返して戻される行数を戻します。

属性データ型
ub4 *

定義ハンドルの属性

OCI_ATTR_CHAR_COUNT

モード
書き込み

説明

文字型データの文字数を設定します。この属性は、定義バッファに格納したい文字の数を指定します。define コールで指定した定義バッファの長さは、文字数より大きくなければなりません。

属性データ型
ub4 *

OCI_ATTR_CHARSET_ID

モード
読み込み / 書き込み

説明

定義ハンドルのキャラクタ・セット ID。出力データのキャラクタ・セットが UCS-2 (Unicode) の場合、ユーザーはキャラクタ・セット ID を OCI_UCS2ID に設定する必要があります。定義値バッファは utext バッファであると見なされ、標識および戻り値の長さのセマンティクスは、文字セマンティクス (utext の数) に変更されます。ただし、先行する OCIDefine コールでは、定義値バッファのサイズをバイト単位で指定する必要があります。詳細は、5-27 ページの「[固定幅 Unicode サポート](#)」を参照してください。

属性データ型
ub2 *

OCI_ATTR_CHARSET_FORM

モード
読み込み / 書き込み

説明
定義ハンドルのキャラクタ・セット形式

属性データ型
ub1 *

OCI_ATTR_PDPRC

モード
書き込み

説明
パック 10 進数の精度を指定します。SQLT_PDN 値の場合、精度は $2^{(\text{value_sz}-1)}$ に等しくなければなりません。SQLT_SLS 値の場合、精度は $(\text{value_sz}-1)$ に等しくなければなりません。

バインドまたは定義後に、この値は 0 (ゼロ) に初期化されます。最初に OCI_ATTR_PDPRC 属性を設定して、次に OCI_ATTR_PDSCL を設定します。これらの値のどちらかを変更する必要がある場合は、最初に再バインドまたは再定義を行って、次にこれらの属性を適切に再設定する必要があります。

属性データ型
ub2 *

OCI_ATTR_PDSCL

モード
書き込み

説明
パック 10 進値のスケールを指定します。

バインドまたは定義後に、この値は 0 (ゼロ) に初期化されます。最初に OCI_ATTR_PDPRC 属性を設定して、次に OCI_ATTR_PDSCL を設定します。これらの値のどちらかを変更する必要がある場合は、最初に再バインドまたは再定義を行って、次にこれらの属性を適切に再設定する必要があります。

属性データ型
sb2 *

記述ハンドルの属性

OCI_ATTR_PARAM

モード
読み込み

説明
記述のルートを指します。後続の `OCIAttrGet()` および `OCIParamGet()` コールのために使用します。

属性データ型
`ub4 *`

OCI_ATTR_PARAM_COUNT

モード
読み込み

説明
記述ハンドル内のパラメータ数を戻します。記述ハンドルが選択リストの記述である場合は、選択リスト内の列数を示します。

属性データ型
`ub4 *`

パラメータ記述子の属性

パラメータ記述子の属性の詳細なリストは、[第 6 章の「スキーマ・メタデータの記述」](#)を参照してください。

LOB ロケータの属性

OCI_ATTR_LOBEMPTY

モード
書込み

説明
内部 LOB ロケータを Empty 値に設定します。すると、そのロケータは INSERT または UPDATE 文のバインド変数として使用でき、LOB を Empty 値に初期化します。LOB が Empty 値になると、*OCIlobWrite()* をコールしてデータを LOB に移入できます。この属性は内部 LOB (BLOB、CLOB、NCLOB) に対してだけ有効です。

アプリケーションから、0 (ゼロ) の値を持つ `ub4` のアドレスを渡します。たとえば、

```
ub4 lobEmpty = 0
```

と宣言した後にアドレス `&lobEmpty` を渡します。

属性データ型
`ub4 *`

複合オブジェクトの属性

複合オブジェクト検索の詳細は、10-19 ページの「[複合オブジェクト検索](#)」を参照してください。

複合オブジェクト検索ハンドルの属性

OCI_ATTR_COMPLEXOBJECT_LEVEL

モード
書込み

説明
複合オブジェクト検索のレベルの理解。

属性データ型
`ub4 *`

OCI_ATTR_COMPLEXOBJECT_COLL_OUTOFFLINE

モード
書込み

説明
オブジェクト型ライン外にあるコレクション属性をフェッチするかどうか。

属性データ型
ub1 *

複合オブジェクト検索記述子の属性

OCI_ATTR_COMPLEXOBJECTCOMP_TYPE

モード
書込み

説明
複合オブジェクト検索に続く REF のタイプ。

属性データ型
dvoid *

OCI_ATTR_COMPLEXOBJECTCOMP_TYPE_LEVEL

モード
書込み

説明
後続の OCI_ATTR_COMPLEXOBJECTCOMP_TYPE 型の REF のレベルの理解。

属性データ型
ub4 *

アドバンスト・キューイング記述子の属性

アドバンスト・キューイング、プロパティおよびオプションの詳細は、『Oracle8i アプリケーション開発者ガイド - 基礎編』のアドバンスト・キューイングの章を参照してください。

OCIAQEnqOptions 記述子の属性

次に示す属性は、OCIAQEnqOptions 記述子のプロパティです。

OCI_ATTR_RELATIVE_MSGID

モード

読み込み / 書き込み

説明

順序逸脱操作で参照されるメッセージのメッセージ識別子を指定します。この値は、OCI_ATTR_SEQUENCE_DIVISION で OCI_ENQ_BEFORE が指定されている場合にだけ有効です。順序逸脱が未指定の場合には、この値は無視されます。

属性データ型

OCIRaw *

OCI_ATTR_SEQUENCE_DEVIATION

モード

読み込み / 書き込み

説明

現在エンキューされているメッセージを、すでにキューにあるその他のメッセージより先にデキューするかどうかを指定します。

属性データ型

ub4

可能な値

次に示す値だけが有効です。

- OCI_ENQ_BEFORE: メッセージは、OCI_ATTR_RELATIVE_MSGID で指定したメッセージより先にエンキューされます。
- OCI_ENQ_TOP: メッセージは、その他のあらゆるメッセージより先にエンキューされます。

OCI_ATTR_VISIBILITY

モード

読み込み / 書き込み

説明

エンキュー要求のトランザクション動作を指定します。

属性データ型

ub4

可能な値

次に示す値だけが有効です。

- OCI_ENQ_ON_COMMIT: エンキューは現行のトランザクションの一部です。操作は、トランザクションがコミットするときに完了します。これは、デフォルトです。
- OCI_ENQ_IMMEDIATE: エンキューは現行のトランザクションの一部ではありません。固有のトランザクションで操作が行われます。

OCIAQDeqOptions 記述子の属性

次に示す属性は、OCIAQDeqOptions 記述子のプロパティです。

OCI_ATTR_CONSUMER_NAME

モード

読み込み / 書き込み

説明

利用者名。利用者に合致するメッセージだけがアクセスされます。キューが複数の利用者に対して設定されていない場合、このフィールドは NULL に設定します。

属性データ型

text *

OCI_ATTR_CORRELATION

モード

読み込み / 書き込み

説明

デキューするメッセージの相関識別子を指定します。パーセント符号 (%) やアンダースコア (_) などの特殊なパターン・マッチング文字を使用することができます。2 つ以上のメッセージがパターンに合致する場合、デキューの順序は確定されません。

属性データ型
text *

OCI_ATTR_DEQ_MODE

モード
読み込み / 書き込み

説明
デキューに関連付けられたロック動作を指定します。

属性データ型
ub4

可能な値
次に示す値だけが有効です。

- OCI_DEQ_BROWSE: ロックを取得することなくメッセージを読み込みます。これは、SELECT 文と同じです。
- OCI_DEQ_LOCKED: メッセージを読み込み、書き込みロックを取得します。ロックは、トランザクションの間有効です。これは、SELECT FOR UPDATE 文と同じです。
- OCI_DEQ_REMOVE: メッセージを読み込み、それを更新または削除します。これは、デフォルトです。メッセージは、保存プロパティに基づいてキュー・テーブルに保存されます。
- OCI_DEQ_NO_DATA: メッセージの受取りを確認しますが、実際のメッセージ内容は送付しません。

OCI_ATTR_DEQ_MSGID

モード
読み込み / 書き込み

説明
デキューするメッセージのメッセージ識別子を指定します。

属性データ型
OCIRaw *

OCI_ATTR_NAVIGATION

モード
読み込み / 書き込み

説明

取り出すメッセージの位置を指定します。最初に位置が決定されます。次に検索基準が適用されます。最後にメッセージが取り出されます。

属性データ型

ub4

可能な値

次に示す値だけが有効です。

- OCI_DEQ_FIRST_MSG: 使用可能なメッセージの中から、検索基準に合致する最初のメッセージを取り出します。これにより、位置はキューの先頭にリセットされます。
- OCI_DEQ_NEXT_MSG: 使用可能なメッセージの中から、検索基準に合致する次のメッセージを取り出します。前のメッセージがメッセージ・グループに属していた場合は、検索基準に合致し、なおかつメッセージ・グループに属するメッセージが AQ により取り出されます。これは、デフォルトです。
- OCI_DEQ_NEXT_TRANSACTION: 残りの現行のトランザクション・グループをスキップし、次のトランザクション・グループから最初のメッセージを取り出します。このオプションは、現行のキューに対してメッセージをグループ化できる場合にだけ使用できます。

OCI_ATTR_VISIBILITY

モード

読み込み / 書き込み

説明

新しいメッセージを現行のトランザクションの一部としてデキューするかどうかを指定します。BROWSE モードを使用しているときは、この可視性 (visibility) パラメータは無視されます。

属性データ型

ub4

可能な値

次に示す値のみ有効です。

- OCI_DEQ_ON_COMMIT: デキューは現行のトランザクションの一部になります。これは、デフォルトです。
- OCI_DEQ_IMMEDIATE: デキューされたメッセージは現行のトランザクションの一部ではありません。固有のトランザクションが構成されます。

OCI_ATTR_WAIT

モード

読み込み / 書き込み

説明

検索基準に合致するメッセージがないときの待機時間を指定します。同じグループのメッセージがデキューされている場合、このパラメータは無視されます。

属性データ型

ub4

可能な値

ub4 値はいずれも有効ですが、さらに次のような定数が事前定義されています。

- OCI_DEQ_WAIT_FOREVER: いつまでも待ち続けます。これは、デフォルトです。
- OCI_DEQ_NO_WAIT: 待ちません。

OCIAQMsgProperties 記述子の属性

次に示す属性は、OCIAQMsgProperties 記述子のプロパティです。

OCI_ATTR_ATTEMPTS

モード

読み込み

説明

あるメッセージに対して今までに何回デキューが試みられたかを指定します。このパラメータを、エンキュー時刻に設定することはできません。

属性データ型

sb4

可能な値

sb4 値はいずれも有効です。

OCI_ATTR_CORRELATION

モード

読み込み / 書き込み

説明

エンキュー時に製作者がメッセージに付けた ID を指定します。

属性データ型

text *

可能な値

文字列は 128 バイトまで有効です。

OCI_ATTR_DELAY

モード

読み込み / 書き込み

説明

エンキューしたメッセージを遅延させる秒数を指定します。遅延はメッセージがデキュー可能になった後の秒数を示します。msgid によりデキューすると、遅延指定は上書きされません。遅延を設定すると、エンキューされたメッセージは WAITING 状態となり、遅延時間が終了するとともに READY 状態となります。DELAY 処理にはキュー・モニターを起動する必要があります。遅延は、あくまでもメッセージをエンキューする製作者が設定します。

属性データ型

sb4

可能な値

すべての sb4 値が有効ですが、次の事前定義済みの定数が使用可能です。

- OCI_MSG_NO_DELAY: メッセージが即時にデキューできることを示します。

OCI_ATTR_ENQ_TIME

モード

読み込み

説明

メッセージがエンキューされた時間を指定します。この値はシステムにより判断されるものであり、ユーザーは設定できません。

属性データ型

OCIDate

OCI_ATTR_EXCEPTION_QUEUE

モード

読み込み / 書き込み

説明

無事に処理されなかったメッセージを移動する移動先のキューの名前を指定します。メッセージが移動されるのは次の2つの場合です。デキューが *max_retries* 以内の回数で成功しなかった場合とメッセージの有効期限が切れた場合です。例外キューのメッセージはすべて EXPIRED 状態です。

デフォルトは、キュー・テーブルと関連付けられた例外キューです。移動時に指定の例外キューが存在しない場合、メッセージはキュー・テーブルと関連付けられたデフォルトの例外キューに移動し、警告が警告ファイルにログされます。デフォルトの例外キューが使用されている場合は、デキュー時に NULL 値が戻されます。

この属性は有効なキュー名を参照しなければなりません。

属性データ型

text *

OCI_ATTR_EXPIRATION**モード**

読み込み / 書き込み

説明

メッセージの時間切れを指定します。この時間（秒）を経過したメッセージはデキューされません。このパラメータは、遅延からのオフセットです。時間切れ処理にはキュー・モニターを起動させる必要があります。

時間切れ前のメッセージは READY 状態です。時間切れ前にデキューされないメッセージは、EXPIRED 状態となって例外デキューに移動します。

属性データ型

sb4

可能な値

すべての sb4 値が有効ですが、次の事前定義済みの定数が使用可能です。

- OCI_MSG_NO_EXPIRATION: メッセージは時間切れになりません。

OCI_ATTR_MSG_STATE**モード**

読み込み

説明

デキュー時のメッセージの状態を指定します。このパラメータを、エンキュー時刻に設定することはできません。

属性データ型

ub4

可能な値

戻される値は次の 4 つだけです。

- OCI_MSG_WAITING: メッセージの遅延時間がまだ経過していません。
- OCI_MSG_READY: メッセージは処理できる状態にあります。
- OCI_MSG_PROCESSED: メッセージは処理され、保存されています。
- OCI_MSG_EXPIRED: メッセージは例外キューに移動しました。

OCI_ATTR_PRIORITY

モード

読み込み / 書き込み

説明

メッセージの優先順位を指定します。数値が小さいほど高い優先順位を示します。優先順位は、負数も含めたあらゆる数値で指定できます。

デフォルト値は 0 (ゼロ) です。

属性データ型

sb4

OCI_ATTR_RECIPIENT_LIST

モード

書き込み

説明

このパラメータは、複数の利用者を許すキューに対してだけ有効です。デフォルトの受信者はキューの予約者です。このパラメータはデキュー時に利用者に戻されません。

属性データ型

OCIAQAgent **

OCI_ATTR_SENDER_ID

モード

読み込み / 書き込み

説明

メッセージの最初の送信者を識別します。

属性データ型

OCIAgent *

OCI_ATTR_ORIGINAL_MSGID**モード**

読み込み / 書き込み

説明

そのメッセージを生成した最後のキューの ID。メッセージがあるキューから別のキューに伝播される際に、この属性はメッセージの最後の伝播元であるキューの ID を識別します。メッセージが複数のキューを経由して伝播されている場合は、この属性は、最初のキューではなく最後のキューを識別します。

属性データ型

OCIRaw *

OCIAQAgent 記述子の属性

次に示す属性は、OCIAQAgent 記述子のプロパティです。

OCI_ATTR_AGENT_ADDRESS**モード**

読み込み / 書き込み

説明

プロトコル特定の受信者アドレス。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] となります。

属性データ型

text *

可能な値

128 バイト以下のあらゆる文字列。

OCI_ATTR_AGENT_NAME**モード**

読み込み / 書き込み

説明

メッセージ製作者名または利用者名。

属性データ型

text *

可能な値

30 バイト以下のあらゆる Oracle 識別子。

OCI_ATTR_AGENT_PROTOCOL

モード

読み込み / 書き込み

説明

アドレスを解釈しメッセージを伝播させるプロトコル。デフォルト値（現在はこれだけがサポートされています）は0（ゼロ）です。

属性データ型

ub1

可能な値

有効な値は0（ゼロ）だけです。これはデフォルトでもあります。

サブスクリプション・ハンドルの属性

関連項目 : [ダイレクト・パス・ロードおよびダイレクト・パス・ハンドルの割当ての詳細は、9-28 ページの「パブリッシュ - サブスクライブの通知」を参照してください。](#)

OCI_ATTR_SUBSCR_CALLBACK

モード

読み込み / 書き込み

説明

サブスクリプション・コールバック。登録コール `OCISubscriptionRegister()` にサブスクリプション・ハンドルを渡すには、この属性を設定する必要があります。

属性データ型

OCISubscriptionNotify *

OCI_ATTR_SUBSCR_CTX

モード

読み込み / 書き込み

説明

システムにより起動された際に、クライアントから `OCI_ATTR_SUBSCR_CALLBACK` が示すユーザー・コールバックに渡されるコンテキスト。登録コール `OCISubscriptionRegister()` にサブスクリプション・ハンドルを渡すには、この属性を設定する必要があります。

属性データ型

dvoid *

OCI_ATTR_SUBSCR_NAMESPACE

モード

読み込み / 書き込み

説明

サブスクリプション・ハンドルが使用される名前領域。リリース 8i では、`OCI_SUBSCR_NAMESPACE_AQ` だけがサポートされ、この値は明示的に設定する必要があります。サブスクリプション・ハンドルに設定されるサブスクリプション名は、その名前領域と一貫性を持たせる必要があります。

属性データ型

ub4 *

OCI_ATTR_SUBSCR_NAME

モード

読み込み / 書き込み

説明

サブスクリプション名。すべてのサブスクリプションは、サブスクリプション名によって識別されます。サブスクリプション名は、指定された長さの一連のバイト数から構成されます。サブスクリプション名のバイト長は、この名前が NULL で終了しないことを前提として指定する必要があります。名前に NLS 文字が含まれる可能性があるため、この前提は重要です。

クライアントでは、OCIAttrSet() コールを使用して、OCI_HTYPE_SUBSCR のハンドル型と OCI_ATTR_SUBSCR_NAME の属性タイプを指定することにより、サブスクリプション・ハンドルのサブスクリプション名属性を設定できます。

すべてのサブスクリプション・コールバックには、OCI_ATTR_SUBSCR_NAME 属性および OCI_ATTR_SUBSCR_NAMESPACE 属性が設定されたサブスクリプション・ハンドルが必要です。これらの属性が設定されていない場合は、エラーが戻されます。サブスクリプション・ハンドルに設定されるサブスクリプション名は、その名前領域と一貫性を持たせる必要があります。

属性データ型

text *

OCI_ATTR_SUBSCR_PAYLOAD

モード

読み込み / 書き込み

説明

通知とともに送信する必要があるペイロードに対応するバッファ。バッファ長も同じ属性設定コールに指定できます。サブスクリプションで通知を実行するには、この属性を設定する必要があります。リリース 8i では、タイプ化されていない (ub1 *) ペイロードだけをサポートします。

属性データ型

ub1 *

ダイレクト・パス・ロード・ハンドルの属性

関連項目: [ダイレクト・パス・ロード](#)および[ダイレクト・パス・ハンドルの割当ての詳細](#)は、9-36 ページの「[ダイレクト・パス・ロード](#)」を参照してください。

ダイレクト・パス・コンテキスト・ハンドルの属性

OCI_ATTR_BUF_SIZE

モード

読み込み / 書き込み

説明

ストリーム転送バッファのサイズを設定します。デフォルト値は 64KB です。

属性データ型

ub4 */ub4 *

OCI_ATTR_CHARSET_ID

モード

読み込み / 書き込み

説明

文字データのデフォルトのキャラクタ・セット ID。このキャラクタ・セット ID は、列レベルで上書きできることに注意してください。キャラクタ・セット ID が列レベルまたは表レベルで指定されていない場合は、NLS 環境設定が使用されます。

属性データ型

ub2 */ub2 *

OCI_ATTR_DATEFORMAT

モード

読み込み / 書き込み

説明

SQLT_CHAR から DTYDAT に変換する際の、デフォルトの日付書式文字列。この日付書式文字列は、列レベルで上書きできることに注意してください。日付書式文字列が列レベルまたは表レベルで指定されていない場合は、NLS 環境設定が使用されます。

属性データ型

text **/text *

OCI_ATTR_DIRPATH_MODE

モード

読み込み / 書き込み

説明

ダイレクト・パス・コンテキストのモード。

- OCI_DIRPATH_LOAD: ロード操作 (デフォルト)
- OCI_DIRPATH_CONVERT: 変換専用操作

属性データ型

ub1 */ub1 *

OCI_ATTR_DIRPATH_NOLOG

モード

読み込み / 書き込み

説明

各セグメントの NOLOG 属性は、イメージ REDO と無効化 REDO のどちらが生成されるかを次のように判別します。

- 0: ロードされるセグメントの属性を使用します。
- 1: ログ記録を行いません。必要があれば DDL 文を上書きします。

属性データ型

ub1 */ub1 *

OCI_ATTR_DIRPATH_PARALLEL

モード

読み込み / 書き込み

説明

この値を 1 に設定すると、複数ロード・セッションで、共通のセグメントを同時にロードできます。デフォルトは 0 (ゼロ) です (非パラレル)。

属性データ型

ub1 */ub1 *

OCI_ATTR_LIST_COLUMNS

モード
読み込み

説明
ダイレクト・パス・コンテキストに関連付けられた列リストのパラメータ記述子に、ハンドルを戻します。OCI_ATTR_NUM_COLS 属性で列数を設定した後で、列リスト・パラメータ記述子を取り出すことができます。A-44 ページの「[列パラメータ属性へのアクセス](#)」を参照してください。

属性データ型
OCIParam**

OCI_ATTR_NAME

モード
読み込み / 書き込み

説明
ロードされる表の名前。

属性データ型
text**/text *

OCI_ATTR_NUM_COLS

モード
読み込み / 書き込み

説明
表にロードされる列の数。

属性データ型
ub2 */ub2 *

OCI_ATTR_SCHEMA_NAME

モード
読み込み / 書き込み

説明

ロードされる表が存在するスキーマの名前。指定しない場合は、デフォルトで、接続ユーザーのスキーマが使用されます。

属性データ型

text **/text *

OCI_ATTR_SUB_NAME

モード

読み込み / 書き込み

説明

ロードされるパーティションまたはサブパーティションの名前。指定しない場合は、表全体がロードされます。名前は、その表に属する有効なパーティションまたはサブパーティションの名前にする必要があります。

属性データ型

text **/text *

ダイレクト・パス列配列ハンドルの属性

OCI_ATTR_COL_COUNT

モード

読み込み

説明

最後に処理された行の最終列。

属性データ型

ub2 *

OCI_ATTR_NUM_COLS

モード

読み込み

説明

列配列の列ディメンション。

属性データ型

ub2 *

OCI_ATTR_NUM_ROWS

モード
読み込み

説明
列配列の行ディメンション。

属性データ型
ub4 *

OCI_ATTR_ROW_COUNT

モード
読み込み

説明
最後に処理された行。

属性データ型
ub4 *

ダイレクト・パス・ストリーム・ハンドルの属性

OCI_ATTR_BUF_ADDR

モード
読み込み

説明
ストリーム・データの先頭のバッファ・アドレス。

属性データ型
ub1 **

OCI_ATTR_BUF_SIZE

モード
読み込み

説明
ストリーム・データのバイト単位のサイズ。

属性データ型

ub4 *

OCI_ATTR_ROW_COUNT

モード

読み込み

説明

最後に処理された行の列配列索引。この属性は、データ・ソースが列配列である場合にだけ有効です。

属性データ型

ub4 *

OCI_ATTR_STREAM_OFFSET

モード

読み込み

説明

最後に処理された行の、ストリーム・バッファへのオフセット。

属性データ型

ub4 *

ダイレクト・パス列パラメータの属性

アプリケーションでは、各列パラメータ記述子に対して属性を設定することにより、ロードする列、およびデータの外部書式を指定します。列パラメータ記述子は、*OCIParamGet()* によって、列パラメータ・リストのパラメータとして取得されます。列パラメータ・リストは、ダイレクト・パス・コンテキストの *OCI_ATTR_LIST_COLUMNS* 属性から取得されます。すべてのパラメータは 1 を基準としていることに注意してください。

列パラメータ属性へのアクセス

次のコード・サンプルは、ダイレクト・パス列パラメータの属性の使用方法を示しています。この属性にアクセスするには、最初に、ロードする列の数を設定して、*OCI_ATTR_LIST_COLUMNS* 属性から列パラメータ・リストを取得する必要があります。

```
/* set number of columns to be loaded */
OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
          OCIAttrSet((dvoid *)dpctx, (ub4)OCI_HTYPE_DIRPATH_CTX,
                    (dvoid *)&tblp->ncol_tbl,
                    (ub4)0, (ub4)OCI_ATTR_NUM_COLS, ctlp->errhp_ctl));
```

```

/* get the column parameter list */
OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
          OCIAttrGet((dvoid *)dpctx,
                    OCI_HTYPE_DIRPATH_CTX,
                    (dvoid *)&ctlp->colLstDesc_ctl, (ub4 *)0,
                    OCI_ATTR_LIST_COLUMNS, ctlp->errhp_ctl));

```

これで、パラメータ属性を設定できます。

```

/* set the attributes of each column by getting a parameter handle on each
 * column, then setting attributes on the parameter handle for the column.
 * Note that positions within a column list descriptor are 1-based. */

for (i = 0, pos = 1, colp = tblp->col_tbl, fldp = tblp->fld_tbl;
     i < tblp->ncol_tbl;
     i++, pos++, colp++, fldp++)
{
    /* get parameter handle on the column */
    OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
              OCIParamGet((CONST dvoid *)ctlp->colLstDesc_ctl,
                          (ub4)OCI_DTYPE_PARAM, ctlp->errhp_ctl,
                          (dvoid **)&colDesc, pos));

    colp->id_col = i;                /* position in column array */

    /* set external attributes on the column */
    /* column name */
    OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
              OCIAttrSet((dvoid *)colDesc, (ub4)OCI_DTYPE_PARAM,
                        (dvoid *)colp->name_col,
                        (ub4)strlen((const char *)colp->name_col),
                        (ub4)OCI_ATTR_NAME, ctlp->errhp_ctl));

    /* column type */
    OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
              OCIAttrSet((dvoid *)colDesc, (ub4)OCI_DTYPE_PARAM,
                        (dvoid *)&colp->extttyp_col, (ub4)0,
                        (ub4)OCI_ATTR_DATA_TYPE, ctlp->errhp_ctl));

    /* max data size */
    OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
              OCIAttrSet((dvoid *)colDesc, (ub4)OCI_DTYPE_PARAM,
                        (dvoid *)&fldp->maxlen_fld, (ub4)0,
                        (ub4)OCI_ATTR_DATA_SIZE, ctlp->errhp_ctl));

    if (colp->datemask_col) /* set column (input field) date mask */

```

```

    {
        OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
            OCIAttrSet((dvoid *)colDesc, (ub4)OCI_DTYPE_PARAM,
                (dvoid *)colp->datemask_col,
                (ub4)strlen((const char *)colp->datemask_col),
                (ub4)OCI_ATTR_DATEFORMAT, ctlp->errhp_ctl));
    }
    if (colp->prec_col)
    {
        OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
            OCIAttrSet((dvoid *)colDesc, (ub4)OCI_DTYPE_PARAM,
                (dvoid *)&colp->prec_col, (ub4)0,
                (ub4)OCI_ATTR_PRECISION, ctlp->errhp_ctl));
    }
    if (colp->scale_col)
    {
        OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
            OCIAttrSet((dvoid *)colDesc, (ub4)OCI_DTYPE_PARAM,
                (dvoid *)&colp->scale_col, (ub4)0,
                (ub4)OCI_ATTR_SCALE, ctlp->errhp_ctl));
    }
    if (colp->csid_col)
    {
        OCI_CHECK(ctlp->errhp_ctl, OCI_HTYPE_ERROR, ociret, ctlp,
            OCIAttrSet((dvoid *)colDesc, (ub4)OCI_DTYPE_PARAM,
                (dvoid *)&colp->csid_col, (ub4)0,
                (ub4)OCI_ATTR_CHARSET_ID, ctlp->errhp_ctl));
    }
    /* free the parameter handle to the column descriptor */
    OCI_CHECK((dvoid *)0, 0, ociret, ctlp,
        OCIDescriptorFree((dvoid *)colDesc, OCI_DTYPE_PARAM));
}

```

OCI_ATTR_CHARSET_ID

モード

読み込み / 書き込み

説明

キャラクタ列のキャラクタ・セット ID。設定しない場合は、デフォルトで、ダイレクト・パス・コンテキスト内に設定されたキャラクタ・セット ID が使用されます。

属性データ型

ub2 */ub2 *

OCI_ATTR_DATA_SIZE

モード
読み込み / 書き込み

説明
列の外部データのバイト単位の最大サイズ。これは、変換バッファ・サイズに影響する可能性があります。

属性データ型
ub4 */ub4 *

OCI_ATTR_DATA_TYPE

モード
読み込み / 書き込み

説明
列の外部データ型を戻すか、または設定します。有効なデータ型は、SQLT_CHR、SQLT_DAT、SQLT_INT、SQLT_UIN、SQLT_FLT、SQLT_PDN、SQLT_BIN、SQLT_NUM です。

属性データ型
ub2 */ub2 *

OCI_ATTR_DATEFORMAT

モード
読み込み / 書き込み

説明
列の日付変換マスク。設定しない場合、日付書式は、デフォルトでダイレクト・パス・コンテキスト内に設定された日付変換マスクになります。

属性データ型
text **/text *

OCI_ATTR_NAME

モード
読み込み / 書き込み

説明
ロードされる列の名前を戻すか、または設定します。

属性データ型

text **/text *

OCI_ATTR_PRECISION

モード

読み込み / 書き込み

説明

精度を戻すか、または設定します。

属性データ型

ub1 */ub1 *

OCI_ATTR_SCALE

モード

読み込み / 書き込み

説明

バック 10 進数およびゾーン 10 進数の入力データ型からの変換のスケール (小数点以下の桁数) を戻すか、または設定します。

属性データ型

sb1 */sb1 *

プロセス・ハンドルの属性

共有システム用のパラメータは、OCIAttrSet() コールおよびOCIAttrGet() コールを使用して設定および読み込みを行うことができます。使用するハンドル型は、プロセス・ハンドル OCI_HTYPE_PROC です。A-6 ページの「OCI_ATTR_SHARED_HEAP_ALLOC」も参照してください。

OCI_ATTR_MEMPOOL_APPNAME 属性、OCI_ATTR_MEMPOOL_HOMENAME 属性および OCI_ATTR_MEMPOOL_INSTNAME 属性は、アプリケーション名、ホーム名およびインスタンス名を指定します。これらの名前を一緒に使用して、プロセスを適切な共有プール領域にマップすることができます。これらの属性を設定しない場合は、内部デフォルト値が使用されます。特定の動作のための、可能な属性設定を次に示します。

- インスタンス名、アプリケーション名（非修飾）：この設定では、特定の名前の実行可能ファイルだけが、共通の共有サブシステムに接続できます。たとえば、Office という名前の OCI アプリケーションが、Office が存在するディレクトリに関係なく、共通の共有サブシステムに接続できます。
- インスタンス名、ホーム名：この設定では、特定のホーム・ディレクトリ内の一連の実行可能ファイルが、共有サブシステムの同じインスタンスに接続できます。たとえば、ORACLE_HOME ディレクトリに存在するすべての OCI アプリケーションが、共通の共有サブシステムを使用できます。
- インスタンス名、ホーム名、アプリケーション名（非修飾）：この設定では、特定の実行可能ファイルだけが、共有サブシステムに接続できます。たとえば、ORACLE_HOME ディレクトリ内の Office という名前の 1 つのアプリケーションが、指定の共有サブシステムに接続できます。

OCI_ATTR_MEMPOOL_APPNAME

モード

読み込み / 書き込み

説明

実行可能ファイルのファイル名または完全修飾パス名。

属性データ型

text *

OCI_ATTR_MEMPOOL_HOMENAME

モード

読み込み / 書き込み

説明

共通の共有サブシステム・インスタンスを使用する実行可能ファイルが存在するディレクトリ名。

属性データ型

text *

OCI_ATTR_MEMPOOL_INSTNAME

モード

読み込み / 書き込み

説明

共有サブシステムのインスタンスを識別する、任意のユーザー定義名。

属性データ型

text *

OCI_ATTR_MEMPOOL_SIZE

モード

読み込み / 書き込み

説明

共有プールのバイト単位のサイズ。この属性は次のように設定されます。

```
ub4 plsz = 1000000;  
OCIAttrSet((dvoid *)0, (ub4) OCI_HTYPE_PROC,  
           (dvoid *)&plsz, (ub4) 0, (ub4) OCI_ATTR_POOL_SIZE, 0)
```

属性データ型

ub4 *

OCI_ATTR_PROC_MODE

モード

読み込み

説明

現在設定されているすべてのプロセス・モードを戻します。読み込まれる値には、現在設定されているすべての OCI プロセス・モードの論理和をとった値が含まれます。特定のモードが設定されているかどうかを判別するには、そのモードで値の論理和をとります。たとえば、次のとおりです。

```
ub4 mode;
```

```
boolean is_shared;  
  
OCIAttrGet((dvoid *)0, (ub4)OCI_HTYPE_PROC,  
           (dvoid *) &mode, (ub4 *) 0,  
           (ub4)OCI_ATTR_PROC_MODE, 0);  
  
is_shared = (mode & OCI_SHARED);
```

属性データ型

ub4 *

B

OCI デモ・プログラム

Oracle は、OCI コールの使用方法を示すコード例を提供しています。これらのプログラムはあくまでも例にすぎないので、すべてのプラットフォームで実行できるとは限りません。

デモ・プログラムは、Oracle インストールで使用できます。デモ・プログラムの位置、名前および可用性は、プラットフォームによって異なります。Unix ワークステーションでは、デモ・プログラムは `ORACLE_HOME/demo` ディレクトリにインストールされています。Windows NT マシンでは、デモ・プログラムは `ORACLE_HOME%\oci\Samples` ディレクトリにあります。

特定のヘッダー・ファイルまたは SQL ファイルがアプリケーションで必要になった場合には、これらのファイルも用意されています。設定やプログラム実行のヒントは、デモ・プログラムの最初に記載された情報を参照してください。

表 B-1 の「OCI デモ・プログラム」に、重要なデモ・プログラムおよびそのプログラムで示される OCI 機能のリストを示します。

表 B-1 OCI デモ・プログラム

プログラム名	示される機能
<code>cdemo1.c</code>	基本 SQL 処理の使用
<code>cdemo81.c</code>	リリース 8 の機能性を利用した基本 SQL 処理の使用
<code>cdemo82.c</code>	ユーザー定義オブジェクトの基本処理の実行
<code>cdmocer.c</code>	複合オブジェクト検索 (COR) を使用したパフォーマンス向上
<code>cdemodr1.c</code> 、 <code>cdemodr2.c</code> 、 <code>cdemodr3.c</code>	LOB および REF で使用される RETURNING 句を伴う INSERT/UPDATE/DELETE 文の使用
<code>cdemodsa.c</code>	表の情報の記述
<code>cdemodsc.c</code>	ユーザー定義型の情報の記述

プログラム名	示される機能
cdemofo.c	アプリケーション・フェイルオーバー・コールバックの登録および操作
cdemolb.c	LOB オブジェクトの作成、アクセスおよび操作
cdemolb2.c	ストリーム・モードおよびコールバック関数を使用した CLOB/BLOB 列の書き込みおよび読み込み
cdemolbs.c	LOB バッファリング・システムを使用した LOB の書き込みおよび読み込み
cdemobj.c	REF オブジェクトの確保およびナビゲーション
cdemorid.c	INSERT/UPDATE/DELETE 文およびフェッチを使用した 1 ラウンドトリップでの複数 ROWID 取得
cdemoses.c	セッション切替えおよび移行の使用
cdemothr.c	OCIThread パッケージの使用
cdemosyev.c	事前定義済みのサブスクリプションの登録およびクライアント通知用に起動するコールバック関数の指定
cdemodp_lip.c	ダイレクト・パス・ロード関数を使用したデータのロード
cdemoucbl.c、cdemoucbl.c	動的および静的ユーザー・コールバックの使用

OCI 関数のサーバー・ラウンドトリップ

この付録では、さまざまな OCI コール中に起こるサーバー・ラウンドトリップの情報を提供します。この情報は、プログラマがアプリケーションで特定の作業を実行するために最も効果的な方法を判断するのに役立ちます。

この付録には、次の項があります。

- [概要](#)
- [リレーショナル関数によるラウンドトリップ](#)
- [LOB 関数によるラウンドトリップ](#)
- [オブジェクトおよびキャッシュ関数によるラウンドトリップ](#)
- [記述操作によるラウンドトリップ](#)
- [データ型のマップおよび操作関数によるラウンドトリップ](#)
- [その他のローカル機能](#)

概要

この付録では、さまざまな OCI コール中に起こるサーバー・ラウンドトリップの情報を提供します。この情報は、アプリケーションで特定の作業を実行するための最も効率的な方法を判断するのに役立ちます。

リレーショナル関数によるラウンドトリップ

OCI リレーショナル関数に必要なサーバー・ラウンドトリップ回数のリストを[表 C-1](#) に示します。

表 C-1 リレーショナル操作のサーバー・ラウンドトリップ

関数	サーバー・ラウンドトリップの回数
OCISstmtGetPieceInfo()	1
OCISstmtSetPieceInfo()	1

LOB 関数によるラウンドトリップ

表 C-2 に、*OCILob*()* コールで発生するサーバー・ラウンドトリップのリストを示します。表の後に、読みおよび書き込みコールに関する情報を記載します。

表 C-2 *OCILob*()* コールのサーバー・ラウンドトリップ

関数	サーバー・ラウンドトリップの回数
<i>OCILobAppend()</i>	1
<i>OCILobAssign()</i>	0
<i>OCILobCharSetForm()</i>	0
<i>OCILobCharSetId()</i>	0
<i>OCILobCopy()</i>	1
<i>OCILobCreateTemporary()</i>	
<i>OCILobDisableBuffering()</i>	0
<i>OCILobEnableBuffering()</i>	0
<i>OCILobErase()</i>	1
<i>OCILobFileClose()</i>	1
<i>OCILobFileCloseAll()</i>	1
<i>OCILobFileExists()</i>	1
<i>OCILobFileGetName()</i>	0
<i>OCILobFileIsOpen()</i>	1
<i>OCILobFileOpen()</i>	1
<i>OCILobFileSetName()</i>	0
<i>OCILobFlushBuffer()</i>	この LOB のバッファにある変更ページごとに 1。
<i>OCILobFreeTemporary()</i>	
<i>OCILobGetLength()</i>	1
<i>OCILobIsEqual()</i>	0
<i>OCILobIsTemporary()</i>	
<i>OCILobLoadFromFile()</i>	1
<i>OCILobLocatorAssign()</i>	ソース・ロケータまたは宛先ロケータ、あるいはその両方がテンポラリ LOB を参照している場合は 1。
<i>OCILobLocatorIsInit()</i>	0

表 C-2 OCILob*() コールのサーバー・ラウンドトリップ (続き)

関数	サーバー・ラウンドトリップの回数
OCILobTrim()	1
OCILobOpen()	1
OCILobClose()	1
OCILobIsOpen()	1
OCILobGetChunkSize()	1

OCILobRead()

必要なラウンドトリップの回数は、コールの使用方法によって異なります。

- コールバックを行わないポーリング・モードでは、パイプをオープンしてデータをストリームするために、各 *OCILobRead()* コールにつき 1 ラウンドトリップが必要です。
- コールバックを行うポーリング・モードでは、1 ラウンドトリップが必要となり、次にすべてのデータが読み込まれるまでコールバック関数がコールされます。
- 入力バッファを使用してデータを 1 まとめにして読み込む場合、1 ラウンドトリップが必要です。

OCILobWrite()、OCILobWriteAppend()

必要なラウンドトリップの回数は、コールの使用方法によって異なります。

- コールバックを行わないポーリング・モードでは、パイプをオープンしてデータをストリームするために、各 *OCILobWrite()* コールにつき 1 ラウンドトリップが必要です。
- コールバックを行うポーリング・モードでは、1 ラウンドトリップが必要となり、次にすべてのデータが書き込まれるまでコールバック関数がコールされます。
- 入力バッファを使用してデータを 1 まとめにして書き込む場合、1 ラウンドトリップが必要です。

オブジェクトおよびキャッシュ関数によるラウンドトリップ

表 C-3 に、オブジェクト関数およびキャッシュ関数に必要なサーバー・ラウンドトリップ回数のリストを示します。これらは、キャッシュがウォーム状態にあるときの値です。つま

り、アプリケーションに必要な型記述子オブジェクトがロードされていることを想定しています。

表 C-3 オブジェクト関数およびキャッシュ関数のサーバー・ラウンドトリップ

関数	サーバー・ラウンドトリップの回数
OCIObjectNew()	0
OCIObjectPin()	1 必要なオブジェクトがすでにキャッシュにある場合は 0。
OCIObjectUnpin()	0
OCIObjectPinCountReset()	0
OCIObjectLock()	1
OCIObjectMarkUpdate()	0
OCIObjectUnmark()	0
OCIObjectUnmarkByRef()	0
OCIObjectFree()	0
OCIObjectMarkDelete()	0
OCIObjectMarkDeleteByRef()	0
OCIObjectFlush()	1
OCIObjectRefresh()	1
OCIObjectCopy()	0
OCIObjectGetTypeRef()	0
OCIObjectGetObjectRef()	0
OCIObjectGetInd()	0
OCIObjectExists()	0
OCIObjectIsLocked()	0
OCIObjectIsDirty()	0
OCIObjectPinTable()	1
OCIObjectArrayPin()	1
OCICacheFlush()	1
OCICacheRefresh()	1
OCICacheUnpin()	0
OCICacheFree()	0

表 C-3 オブジェクト関数およびキャッシュ関数のサーバー・ラウンドトリップ (続き)

関数	サーバー・ラウンドトリップの回数
OCI <code>CacheUnmark()</code>	0

記述操作によるラウンドトリップ

`OCIDescribeAny()`、`OCIAttrGet()` および `OCIParamGet()` に必要なサーバー・ラウンドトリップの回数のリストを表 C-4 に示します。

表 C-4 記述操作のサーバー・ラウンドトリップ

関数	サーバー・ラウンドトリップの回数
OCI <code>DescribeAny()</code>	型記述子オブジェクトの REF 取得のために 1 ラウンドトリップ。
OCI <code>AttrGet()</code>	<p>型オブジェクトがオブジェクト・キャッシュにない場合、型を記述するために 2 ラウンドトリップ。</p> <p>各コレクション要素、型属性、メソッドまたはメソッド引数記述子につき 1 ラウンドトリップ。コレクション要素、型属性またはメソッド引数に対して <code>OCI_ATTR_TYPE_NAME</code> または <code>OCI_ATTR_SCHEMA_NAME</code> を使用している場合はさらに 1 ラウンドトリップ。</p> <p>最初の <code>OCIAttrGet()</code> コールの後で、オブジェクト・キャッシュに記述されるすべての型オブジェクトがすでにある場合は 0。</p>
OCI <code>ParamGet()</code>	0

データ型のマップおよび操作関数によるラウンドトリップ

データ型マッピングおよび操作関数によるラウンドトリップ回数のリストを表 C-5 に示します。表内のアスタリスクは、特定の接頭辞を持つすべての関数のサーバー・ラウンドトリップ回数が同じになることを示します。たとえば、`OCINumberAdd()`、`OCINumberPower()` および `OCINumberFromText()` では、サーバー・ラウンドトリップはすべて 0 となります。

表 C-5 データ型操作関数のサーバー・ラウンドトリップ

関数	サーバー・ラウンドトリップの回数
OCI <code>Number*</code> ()	0
OCI <code>Date*</code> ()	0
OCI <code>String*</code> ()	0
OCI <code>Raw*</code> ()	0

表 C-5 データ型操作関数のサーバー・ラウンドトリップ (続き)

関数	サーバー・ラウンドトリップの回数
OCIRef*()	0
OCIColl*()	0 コレクションがキャッシュにロードされていない場合は1。
OCITable*()	0 ネストした表がキャッシュにロードされていない場合は1。
OCIIter*()	0 コレクションがキャッシュにロードされていない場合は1。

その他のローカル機能

表 C-6 に示す関数はローカルであるため、サーバー・ラウンドトリップは不要です。

表 C-6 ローカルに処理される関数

ローカル関数名	注意
OCIAttrGet()	
OCIAttrSet()	
OCIBindByName()	
OCIBindByPos()	
OCIBindDynamic()	
OCIBindObject()	
OCIBindArrayOfStruct()	
OCIDefineByPos()	
OCIDefineDynamic()	
OCIDefineArrayOfStruct()	
OCIDefineObject()	
OCIDescriptorAlloc()	
OCIDescriptorFree()	
OCIEnvInit()	
OCIErrorGet()	
OCIHandleAlloc()	
OCIHandleFree()	
OCILdaToSvcCtx()	
OCISvcCtxToLda()	

表 C-6 ローカルに処理される関数 (続き)

ローカル関数名	注意
OCIStmtGetBindInfo()	
OCIStmtPrepare()	
OCIStmtGetBindInfo()	
OCIStmtPrepare()	
OCIStmtFetch()	プリフェッチされた行を取り出す場合はローカルです。

索引

数字

2 次メモリー

 オブジェクト, 13-17

3 層アーキテクチャ

 スレッド・セーフティ, 8-14

A

ADO, 「属性記述子オブジェクト」を参照。

ADT, 「抽象データ型」を参照。

AQ, 「アドバンスド・キューイング」を参照。

B

BFILE

 データ型, 3-18

BLOB

 データ型, 3-18

C

CASE OTT パラメータ, 14-25

CHAR

 外部データ型, 3-14

CHARZ

 外部データ型, 3-16

checkerr() 関数

 コード・リスト, 2-27

CLOB

 データ型, 3-19

CODE OTT パラメータ, 14-23

CONFIG OTT パラメータ, 14-24

COR, 「複合オブジェクト検索」を参照。

C データ型

OCI での操作, 11-5

D

DATE

 外部データ型, 3-12

DDL, 「データ定義言語」を参照。

DML, 「データ操作言語」を参照。

E

ERRTYPE OTT パラメータ, 14-24

F

FILE

 データ型, 3-18

 OS ファイルとの関連付け, 7-4

 ロケータ, 7-3

FLOAT

 外部データ型, 3-10

G

GTRID, 「トランザクション識別子」を参照。

H

HFILE OTT パラメータ, 14-24

I

INITFILE OTT パラメータ, 14-23

INITFUNC OTT パラメータ, 14-24

INTEGER

- 外部データ型, 3-10
- INTYPE OTT パラメータ, 14-22
- intype ファイル
 - OTT の実行時に提供, 14-7
 - 構造, 14-26

L

- LOB, 7-2
 - OCI 関数, 7-6
 - 一時オブジェクトの属性, 7-5
 - 外部データ型, 3-17
 - 可変幅キャラクタ・セット, 15-109
 - キャラクタ・セット, 15-109
 - コールバック, 7-12
 - 固定幅キャラクタ・セット, 15-109
 - 作成, 7-4
 - 定義, 5-16
 - データのフェッチ, 4-14
 - テンポラリ, 7-16
 - テンポラリ LOB の解放, 7-17
 - テンポラリ LOB の継続時間, 7-17
 - テンポラリ LOB の作成, 7-17
 - テンポラリ LOB の例, 7-18
 - バインディング, 5-10
 - バッファリング, 7-10
 - 変更, 7-4
 - 量およびオフセット・パラメータ, 15-109
 - ロケータ, 2-15
- LOB 関数, 15-108
 - サーバー・ラウンドトリップ, C-4
- LOB 操作のバッファリング, 7-10
- LOB ロケータ, 2-15, 7-2
 - 属性, A-25
- LONG
 - 外部データ型, 3-12
- LONG RAW
 - 外部データ型, 3-14
- LONG VARCHAR
 - 外部データ型, 3-14
- LONG VARRAW
 - 外部データ型, 3-14

M

MDO, 「方法記述子オブジェクト」を参照。

N

- NCHAR
 - 問題, 5-24
- NCLOB
 - データ型, 3-19
- NLS
 - OCI 関数, xxxii, 2-3
- no-op
 - 定義, 16-22
- NULL
 - アトミック, 10-27
 - オブジェクトの, 10-27
 - 検出, 2-31
 - 挿入, 2-31
 - データベースに挿入, 2-30
 - 標識変数を使用して挿入, 2-30
- NULL 標識構造体, 10-27
 - OTT で生成された, 10-9
- NUMBER
 - 外部データ型, 3-9

O

- OCI
 - オブジェクト・サポート, 1-6
 - オブジェクトのアクセスと操作, 14-17
 - 概要, 1-2
 - 新機能, 1-12
 - 部分, 1-5
 - 利点, 1-3
- OCIAQAgent
 - 記述子の属性, A-35
- OCIAQDeq(), 15-5
- OCIAQDeqOptions
 - 記述子の属性, A-28
- OCIAQEnq(), 15-7
- OCIAQEnqOptions
 - 記述子の属性, A-27
- OCIAQListen(), 15-18
- OCIAQMsgProperties
 - 記述子の属性, A-31
- OCIArray, 11-17
 - バインディングと定義, 11-17, 12-6
- OCIArray の操作
 - コード例, 11-19
- OCI_ATTR_ALLOC_DURATION

環境ハンドルの属性, A-4
 OCI_ATTR_AUTOCOMMIT_DDL
 属性, 6-20
 OCI_ATTR_BUF_ADDR, A-43
 OCI_ATTR_BUF_SIZE, A-39, A-43
 OCI_ATTR_CACHE
 属性, 6-15
 OCI_ATTR_CACHE_ARRAYFLUSH
 環境ハンドルの属性, A-3
 OCI_ATTR_CACHE_MAX_SIZE
 環境ハンドルの属性, A-3
 OCI_ATTR_CACHE_OPT_SIZE
 環境ハンドルの属性, A-3
 OCI_ATTR_CATALOG_LOCATION
 属性, 6-19
 OCI_ATTR_CHAR_COUNT
 使用, 5-26
 定義ハンドルの属性, A-22
 バインド・ハンドルの属性, A-20
 OCI_ATTR_CHARSET_FORM
 属性, 6-11, 6-14, 6-16
 定義ハンドルの属性, A-23
 バインド・ハンドルの属性, A-20
 OCI_ATTR_CHARSET_ID, A-39, A-46
 属性, 6-11, 6-14, 6-16, 6-18, 6-19
 定義ハンドルの属性, A-22
 バインド・ハンドルの属性, A-20
 OCI_ATTR_CLUSTERED
 属性, 6-7
 OCI_ATTR_COL_COUNT, A-42
 OCI_ATTR_COLLECTION_ELEMENT
 属性, 6-9
 OCI_ATTR_COLLECTION_TYPECODE
 属性, 6-9
 OCI_ATTR_COMPLEXOBJECT_COLL_OUTOFLINE
 COR ハンドルの属性, A-26
 OCI_ATTR_COMPLEXOBJECTCOMP_TYPE
 COR 記述子の属性, A-26
 OCI_ATTR_COMPLEXOBJECTCOMP_TYPE_LEVEL
 COR 記述子の属性, A-26
 OCI_ATTR_COMPLEXOBJECT_LEVEL
 COR ハンドルの属性, A-25
 OCI_ATTR_CURSOR_COMMIT_BEHAVIOR
 属性, 6-19
 OCI_ATTR_DATA_SIZE, A-47
 属性, 6-10, 6-13, 6-15, 6-16
 OCI_ATTR_DATA_TYPE, A-47
 属性, 6-10, 6-13, 6-15, 6-16
 OCI_ATTR_DATE_FORMAT, A-39
 OCI_ATTR_DATEFORMAT, A-47
 OCI_ATTR_DBA
 属性, 6-7
 OCI_ATTR_DIRPATH_MODE, A-40
 OCI_ATTR_DIRPATH_NOLOG, A-40
 OCI_ATTR_DIRPATH_PARALLEL, A-40
 OCI_ATTR_DML_ROW_OFFSET, A-6
 OCI_ATTR_DURATION
 属性, 6-7
 OCI_ATTR_ENCAPSULATION
 属性, 6-11
 OCI_ATTR_ENV
 サーバー・ハンドルの属性, A-9
 サービス・コンテキスト・ハンドルの属性, A-7
 OCI_ATTR_EXTERNAL_NAME
 サーバー・ハンドルの属性, A-9
 OCI_ATTR_FOCBK
 サーバー・ハンドルの属性, A-10
 OCIAttrGet(), 15-28
 記述用に使用, 4-11
 OCI_ATTR_HAS_DEFAULT
 属性, 6-17
 OCI_ATTR_HAS_FILE
 属性, 6-9
 OCI_ATTR_HAS_LOB
 属性, 6-9
 OCI_ATTR_HAS_NESTED_TABLE
 属性, 6-9
 OCI_ATTR_HW_MARK
 属性, 6-15
 OCI_ATTR_INCR
 属性, 6-14
 OCI_ATTR_INDEX_ONLY
 属性, 6-7
 OCI_ATTR_INTERNAL_NAME
 サーバー・ハンドルの属性, A-10
 OCI_ATTR_IN_V8_MODE
 サーバー・ハンドルの属性, A-10
 サービス・コンテキスト・ハンドルの属性, A-8
 OCI_ATTR_IOMODE
 属性, 6-17
 OCI_ATTR_IS_CONSTRUCTOR
 属性, 6-12
 OCI_ATTR_IS_DESTRUCTOR
 属性, 6-12

OCI_ATTR_IS_INCOMPLETE_TYPE
属性, 6-9
OCI_ATTR_IS_INVOKER_RIGHTS
属性, 6-8, 6-10
OCI_ATTR_IS_MAP
属性, 6-12
OCI_ATTR_IS_NULL
属性, 6-15, 6-17
OCI_ATTR_IS_OPERATOR
属性, 6-12
OCI_ATTR_IS_ORDER
属性, 6-12
OCI_ATTR_IS_PREDEFINED_TYPE
属性, 6-9
OCI_ATTR_IS_RNDS
属性, 6-12
OCI_ATTR_IS_RNPS
属性, 6-12
OCI_ATTR_IS_SELFSH
属性, 6-12
OCI_ATTR_IS_SYSTEM_GENERATED_TYPE
属性, 6-9
OCI_ATTR_IS_SYSTEM_TYPE
属性, 6-9
OCI_ATTR_IS_TEMPORARY
属性, 6-7
OCI_ATTR_IS_TRANSIENT_TYPE
属性, 6-9
OCI_ATTR_IS_WNDS
属性, 6-12
OCI_ATTR_IS_WNPS
属性, 6-12
OCI_ATTR_LEVEL
属性, 6-17
OCI_ATTR_LINK
属性, 6-14, 6-18
OCI_ATTR_LIST_ARGUMENTS
属性, 6-8, 6-11
OCI_ATTR_LIST_COLUMNS, A-41
属性, 6-7
OCI_ATTR_LIST_OBJECTS
属性, 6-19
OCI_ATTR_LIST_SCHEMAS
属性, 6-19
OCI_ATTR_LIST_SUBPROGRAMS
属性, 6-8
OCI_ATTR_LIST_TYPE
属性, 6-18
OCI_ATTR_LIST_TYPE_ATTRS
属性, 6-10
OCI_ATTR_LIST_TYPE_METHODS
属性, 6-10
OCI_ATTR_LOBEMPTY
LOB ロケータの属性, A-25
OCI_ATTR_LOCKING_MODE
属性, 6-20
OCI_ATTR_MAP_METHOD
属性, 6-10
OCI_ATTR_MAX
属性, 6-14
OCI_ATTR_MAX_CATALOG_NAMELEN
属性, 6-19
OCI_ATTR_MAX_COLUMN_NAMELEN
属性, 6-19
OCI_ATTR_MAXDATA_SIZE
バインドで使用, 5-25
バインド・ハンドルの属性, A-21
OCI_ATTR_MAX_PROC_NAMELEN
属性, 6-19
OCI_ATTR_MEMPOOL_APPNAME, A-49
OCI_ATTR_MEMPOOL_HOMENAME, A-49
OCI_ATTR_MEMPOOL_INSTNAME, A-50
OCI_ATTR_MEMPOOL_SIZE, A-50
OCI_ATTR_MIGSESSION
ユーザー・セッション・ハンドルの属性, A-12
OCI_ATTR_MIN
属性, 6-14
OCI_ATTR_NAME, A-41, A-47
属性, 6-8, 6-10, 6-11, 6-13, 6-14, 6-15, 6-16
OCI_ATTR_NCHARSET_ID
属性, 6-19
OCI_ATTR_NONBLOCKING_MODE
サーバー・ハンドルの属性, A-9
OCI_ATTR_NOWAIT_SUPPORT
属性, 6-20
OCI_ATTR_NUM_ATTRS
属性, 6-5
OCI_ATTR_NUM_COLS, A-41, A-42
属性, 6-7
OCI_ATTR_NUM_DML_ERRORS, A-14
OCI_ATTR_NUM_ELEMENTS
属性, 6-13
OCI_ATTR_NUM_HANDLES
属性, 6-18

OCI_ATTR_NUM_PARAMS
属性, 6-5
OCI_ATTR_NUM_ROWS, A-43
OCI_ATTR_NUM_TYPE_ATTRS
属性, 6-10
OCI_ATTR_NUM_TYPE_METHODS
属性, 6-10
OCI_ATTR_OBJECT
環境ハンドルの属性, A-4
OCI_ATTR_OBJECT_DETECTCHANGE
環境ハンドル属性, 13-13
OCI_ATTR_OBJID
属性, 6-7, 6-14
OCI_ATTR_ORDER
属性, 6-15
OCI_ATTR_ORDER_METHOD
属性, 6-10
OCI_ATTR_OVERLOAD
属性, 6-8
OCI_ATTR_PARAM
記述ハンドルの属性, A-24
OCI_ATTR_PARAM_COUNT
記述ハンドルの属性, A-24
文ハンドルの属性, A-17
OCI_ATTR_PARTITIONED
属性, 6-7
OCI_ATTR_PASSWORD
ユーザー・セッション・ハンドルの属性, A-12
OCI_ATTR_PDSCL
バインド・ハンドルの属性, A-21, A-23
OCI_ATTR_PIN_DURATION
環境ハンドルの属性, A-4
OCI_ATTR_PINOPTION
環境ハンドルの属性, A-4
OCI_ATTR_POSITION
属性, 6-16
OCI_ATTR_PRECISION, A-48
属性, 6-4, 6-10, 6-13, 6-15, 6-16
OCI_ATTR_PREFETCH_MEMORY
文ハンドルの属性, A-19
OCI_ATTR_PREFETCH_ROWS
文ハンドルの属性, A-18
OCI_ATTR_PROC_MODE, A-50
OCI_ATTR_PTYPE
属性, 6-6
OCI_ATTR_RADIX
属性, 6-17
OCI_ATTR_REF_TDO
属性, 6-7, 6-9, 6-11, 6-14, 6-16, 6-18
OCI_ATTR_ROW_COUNT, A-43, A-44
OCI_ATTR_ROWID
文ハンドルの属性, A-17
OCI_ATTR_ROWS_RETURNED
コールバックに使用, 5-24
バインド・ハンドルの属性, A-22
OCI_ATTR_SAVEPOINT_SUPPORT
属性, 6-20
OCI_ATTR_SCALE, A-48
属性, 6-11, 6-13, 6-15, 6-17
OCI_ATTR_SCHEMA_NAME, A-41
属性, 6-11, 6-13, 6-14, 6-16, 6-17
OCI_ATTR_SEQ
属性, 6-14
OCI_ATTR_SERVER
サービス・コンテキスト・ハンドルの属性, A-7
OCI_ATTR_SERVER_GROUP
サーバー・ハンドルの属性, A-11
OCI_ATTR_SESSION
サービス・コンテキスト・ハンドルの属性, A-7
OCIAttrSet(), 15-29
OCI_ATTR_SHARED_HEAP_ALLOC, A-6
OCI_ATTR_STMT_TYPE
文ハンドルの属性, A-17
OCI_ATTR_STREAM_OFFSET, A-44
OCI_ATTR_SUB_NAME, A-42
属性, 6-17
OCI_ATTR_SUBSCR_CALLBACK, A-37
OCI_ATTR_SUBSCR_CTX, A-37
OCI_ATTR_SUBSCR_NAME, A-38
OCI_ATTR_SUBSCR_NAMESPACE, A-37
OCI_ATTR_SUBSCR_PAYLOAD, A-38
OCI_ATTR_TABLESPACE
属性, 6-7
OCI_ATTR_TIMESTAMP
属性, 6-6
OCI_ATTR_TRANS
サービス・コンテキスト・ハンドルの属性, A-8
OCI_ATTR_TRANS_NAME
トランザクション・ハンドルの属性, A-13
OCI_ATTR_TYPECODE
属性, 6-9, 6-10, 6-13, 6-16
OCI_ATTR_TYPE_NAME
属性, 6-11, 6-13, 6-15, 6-17
OCI_ATTR_USRNAME

- ユーザー・セッション・ハンドルの属性, A-12
- OCI_ATTR_VERSION
 - 属性, 6-9, 6-19
- OCI_ATTR_XID
 - トランザクション・ハンドルの属性, A-13
- OCIBindArrayOfStruct(), 15-43
- OCIBindByName(), 15-44
- OCIBindByPos(), 15-48
- OCIBindDynamic(), 15-52
- OCIBindObject(), 15-56
- OCIBreak(), 15-219
 - 使用, 2-32, 2-36
- OCICacheFlush(), 16-9
- OCICacheFree(), 16-48
- OCICacheRefresh(), 16-11
- OCICacheUnmark(), 16-17
- OCICacheUnpin(), 16-49
- OCIColl, 11-17
 - バインディングと定義, 11-17
- OCICollAppend(), 17-6
- OCICollAssign(), 17-7
- OCICollAssignElem(), 17-8
- OCICollGetElem(), 17-10
- OCICollIsLocator(), 17-13
- OCICollMax(), 17-14
- OCICollSetUpdateStatus(), 17-15
- OCICollSize(), 17-16
- OCICollTrim(), 17-18
- OCIComplexObject
 - 使用, 10-22
- OCIComplexObjectComp
 - 使用, 10-22
- OCIDate, 11-6
 - バインディングと定義, 11-6, 12-6
- OCIDateAddDays(), 17-28
- OCIDateAddMonths(), 17-29
- OCIDateAssign(), 17-30
- OCIDateCheck(), 17-31
- OCIDateCompare(), 17-33
- OCIDateDaysBetween(), 17-34
- OCIDateFromText(), 17-35
- OCIDateGetDat(), 17-37
- OCIDateGetTime(), 17-38
- OCIDateLastDay(), 17-39
- OCIDateNextDay(), 17-40
- OCIDateSetDate(), 17-41
- OCIDateSetTime(), 17-42
- OCIDateSysDate(), 17-43
- OCIDateToText(), 17-44
- OCIDateZoneToZone(), 17-46
- OCIDate の操作
 - コード例, 11-8
- OCIDefineArrayOfStruct(), 15-58
- OCIDefineByPos(), 15-60
- OCIDefineDynamic(), 15-64
- OCIDefineObject(), 15-66
- OCIDescAlloc(), 15-31
- OCIDescFree(), 15-33
- OCIDescribeAny(), 15-68
 - 使用方法, 6-2
 - 使用例, 6-20
- OCIDirPathAbort(), 15-74
- OCIDirPathColArrayEntryGet(), 15-75
- OCIDirPathColArrayEntrySet(), 15-77
- OCIDirPathColArrayReset(), 15-80
- OCIDirPathColArrayRowGet(), 15-79
- OCIDirPathColArrayToStream(), 15-81
- OCIDirPathFinish(), 15-83
- OCIDirPathFlushRow(), 15-84
- OCIDirPathPrepare(), 15-86
- OCIDirPathStreamLoad(), 15-84
- OCIDirPathStreamReset(), 15-87
- OCIDuration
 - 使用, 13-7, 13-14
- OCIDurationBegin(), 15-110
- OCIDurationEnd(), 15-111
- OCIEnvCallback, 9-14
- OCIEnvCreate(), 15-89
- OCIEnvInit(), 15-92
- OCIErrorGet(), 15-220
- OCI_EVENTS
 - 通知受信のモード, 9-29
- OCIExtProcAllocCallmemory(), 18-5
- OCIExtProcGetEnv(), 18-8
- OCIExtProcRaiseExcp(), 18-6
- OCIExtProcRaiseExcpWithMsg(), 18-7
- OCIHandleAlloc(), 15-34
- OCIHandleFree(), 15-37
- OCIInd
 - 使用, 10-28
- OCIInitialize(), 15-94
 - 共有モード, 2-20
- OCIIter, 11-17
 - 使用例, 11-19

- バイディングと定義, 11-17
- OCIIterCreate(), 17-19
- OCIIterDelete(), 17-20
- OCIIterGetCurrent(), 17-21
- OCIIterInit(), 17-22
- OCIIterNext(), 17-23
- OCIIterPrev(), 17-25
- OCILdaToSvcCtx(), 15-222
- OCILobAppend(), 15-112
- OCILobAssign(), 15-113
- OCILobCharSet(), 15-115, 15-116
- OCILobClose(), 15-117
- OCILobCopy(), 15-118
- OCILobCreateTemporary(), 15-120
- OCILobDisableBuffering(), 15-122
- OCILobEnableBuffering(), 15-123
- OCILobErase(), 15-124
- OCILobFileClose(), 15-125
- OCILobFileCloseAll(), 15-126
- OCILobFileExists(), 15-127
- OCILobFileIsOpen(), 15-130
- OCILobFileOpen(), 15-131
- OCILobFlushBuffer(), 15-134
- OCILobFreeTemporary(), 15-136
- OCILobGetChunkSize(), 15-137
- OCILobGetFileName(), 15-128
- OCILobGetLength(), 15-139
- OCILobIsEqual(), 15-140
- OCILobIsOpen(), 15-141
- OCILobIsTemporary(), 15-143
- OCILobLoadFromFile(), 15-144
- OCILobLocatorAssign(), 15-146
- OCILobLocatorIsInit(), 15-148
- OCILobOpen(), 15-149
- OCILobRead(), 15-151
- OCILobSetFileName(), 15-132
- OCILobTrim(), 15-155
- OCILobWrite(), 15-156
- OCILobWriteAppend(), 15-160
- OCILockOpt
 - 可能な値, 16-28, 16-55
- OCI_LOCK_X_NOWAIT
 - パラメータの使用方法, 13-13
- OCILogoff(), 15-97
- OCILogon(), 15-98
 - 使用方法, 2-22
- OCINumber, 11-9

- 定義の例, 12-7
- バイディングと定義, 11-9, 12-6
- バインドの例, 12-7
- OCINumberAbs(), 17-50
- OCINumberAdd(), 17-51
- OCINumberArcCos(), 17-52
- OCINumberArcSin(), 17-53
- OCINumberArcTan(), 17-54
- OCINumberArcTan2(), 17-55
- OCINumberAssign(), 17-56
- OCINumberCeil(), 17-57
- OCINumberCompare(), 17-58
- OCINumberCos(), 17-59
- OCINumberDec(), 17-60
- OCINumberDiv(), 17-61
- OCINumberExp(), 17-62
- OCINumberFloor(), 17-63
- OCINumberFromInt(), 17-64
- OCINumberFromReal(), 17-65
- OCINumberFromText(), 17-66
- OCINumberHypCos(), 17-68
- OCINumberHypSin(), 17-69
- OCINumberHypTan(), 17-70
- OCINumberInc(), 17-71
- OCINumberIntPower(), 17-72
- OCINumberIsInt(), 17-73
- OCINumberIsZero(), 17-74
- OCINumberLn(), 17-75
- OCINumberLog(), 17-76
- OCINumberMod(), 17-77
- OCINumberMul(), 17-78
- OCINumberNeg(), 17-79
- OCINumberPower(), 17-80
- OCINumberPrec(), 17-81
- OCINumberRound(), 17-82
- OCINumberSetPi(), 17-83
- OCINumberSetZero(), 17-84
- OCINumberShift(), 17-85
- OCINumberSign(), 17-86
- OCINumberSin(), 17-87
- OCINumberSqrt(), 17-88
- OCINumberSub(), 17-89
- OCINumberTan(), 17-90
- OCINumberToInt(), 17-91
- OCINumberToReal(), 17-92
- OCINumberToText(), 17-93
- OCINumberTrunc(), 17-95

OCINumber の操作
 コード例, 10-13, 11-12
 OCI_NUM_SHARED_PROCS, 2-21
 OCIObjectArrayPin(), 16-50
 OCIObjectCopy(), 16-33
 OCIObjectExists(), 16-25
 OCIObjectFlush(), 16-13
 OCIObjectFree(), 16-52
 OCIObjectGetAttr(), 16-35
 OCIObjectGetInd(), 16-37
 OCIObjectGetObjectRef(), 16-38
 OCIObjectGetTypeRef(), 16-39
 OCIObjectIsDirty(), 16-30
 OCIObjectIsLocked(), 16-31
 OCIObjectLifetime
 可能な値, 16-27
 OCIObjectLock(), 16-40
 OCIObjectLockNoWait(), 16-41
 OCIObjectMarkDelete(), 16-18
 OCIObjectMarkDeleteByRef(), 16-19
 OCIObjectMarkStatus
 可能な値, 16-28
 OCIObjectMarkUpdate(), 16-20
 OCIObjectNew(), 16-42
 OCIObjectPin(), 16-54
 OCIObjectPinCountReset(), 16-57
 OCIObjectPinTable(), 16-59
 OCIObjectRefresh(), 16-14
 OCIObjectSetAttr(), 16-45
 OCIObjectUnmark(), 16-22
 OCIObjectUnmarkByRef(), 16-23
 OCIObjectUnpin(), 16-61
 OCIParamGet(), 15-39
 記述用に使用, 4-11
 OCIParamSet(), 15-41
 OCIPasswordChange(), 15-223
 OCIPinOpt
 使用, 13-7
 OCI_PTYPE_ARG
 属性, 6-16
 OCI_PTYPE_COL
 属性, 6-15
 OCI_PTYPE_COLL
 属性, 6-13
 OCI_PTYPE_DATABASE
 属性, 6-19
 OCI_PTYPE_FUNC
 属性, 6-8
 OCI_PTYPE_LIST
 属性, 6-18
 OCI_PTYPE_PKG
 属性, 6-8
 OCI_PTYPE_PROC
 属性, 6-8
 OCI_PTYPE_SCHEMA
 属性, 6-19
 OCI_PTYPE_SYN
 属性, 6-14
 OCI_PTYPE_TABLE
 属性, 6-7
 OCI_PTYPE_TYPE
 属性, 6-9
 OCI_PTYPE_TYPE_ATTR
 属性, 6-10
 OCI_PTYPE_TYPE_FUNC
 属性, 6-11
 OCI_PTYPE_TYPE_PROC
 属性, 6-11
 OCI_PTYPE_VIEW
 属性, 6-7
 OCIRaw, 11-15
 バインディングと定義, 11-15, 12-6
 OCIRawAllocSize(), 17-97
 OCIRawAssignBytes(), 17-98
 OCIRawAssignRaw(), 17-99
 OCIRawPtr(), 17-100
 OCIRawResize(), 17-101
 OCIRawSize(), 17-102
 OCIRaw の操作
 コード例, 11-16
 OCIRef, 11-21
 使用例, 11-22
 バインディングと定義, 11-21
 OCIRefAssign(), 17-104
 OCIRefClear(), 17-105
 OCIRefFromHex(), 17-106
 OCIRefHexSize(), 17-107
 OCIRefIsEqual(), 17-108
 OCIRefIsNull(), 17-109
 OCIRefToHex(), 17-110
 OCIReset(), 15-225
 使用, 2-36
 OCIRowid, 2-16
 OCIServerAttach(), 15-100

- シャドウ・プロセス, 15-101
- OCIServerDetach(), 15-102
- OCIServerVersion(), 15-226
- OCISessionBegin(), 15-103
- OCISessionEnd(), 15-106
- OCI_SHARED_MODE, 2-21
- OCIStmtExecute(), 15-164
 - iters パラメータの使用, 4-6
 - プリフェッチ, 4-6
- OCIStmtFetch(), 15-167
- OCIStmtGetBind(), 15-71
- OCIStmtGetPieceInfo(), 15-168
- OCIStmtPrepare(), 15-170
 - SQL 文の準備, 4-4
 - 共有モード, 2-20
- OCIStmtSetPieceInfo(), 15-172
- OCIString, 11-14
 - バインディングと定義, 11-14, 12-6
- OCIStringAllocSize(), 17-112
- OCIStringAssign(), 17-113
- OCIStringAssignText(), 17-114
- OCIStringPtr(), 17-115
- OCIStringResize(), 17-116
- OCIStringSize(), 17-117
- OCIString の操作
 - コード例, 11-15
- OCISubscriptionDisable(), 15-20
- OCISubscriptionEnable(), 15-21
- OCISubscriptionPost(), 15-22
- OCISubscriptionRegister(), 15-24
- OCISubscriptionUnRegister(), 15-26
- OCISvcCtxToLda(), 15-227
- OCITable, 11-17
 - バインディングと定義, 11-17, 12-6
- OCITableDelete(), 17-119
- OCITableExists(), 17-120
- OCITableFirst(), 17-121
- OCITableLast(), 17-122
- OCITableNext(), 17-123
- OCITablePrev(), 17-124
- OCITableSize(), 17-125
- OCITerminate(), 15-107
- OCIThreadClose(), 15-176
- OCIThreadCreate(), 15-177
- OCIThreadHandleGet(), 15-179
- OCIThreadHndDestroy(), 15-180
- OCIThreadHndInit(), 15-181
- OCIThreadIdDestroy(), 15-182
- OCIThreadIdGet(), 15-183
- OCIThreadIdInit(), 15-184
- OCIThreadIdNull(), 15-185
- OCIThreadIdSame(), 15-186
- OCIThreadIdSet(), 15-187
- OCIThreadIdSetNull(), 15-188
- OCIThreadInit(), 15-189
- OCIThreadIsMulti(), 15-190
- OCIThreadJoin(), 15-191
- OCIThreadKeyDestroy(), 15-192
- OCIThreadKeyGet(), 15-193
- OCIThreadKeyInit(), 15-194
- OCIThreadKeySet(), 15-195
- OCIThreadMutexAcquire(), 15-196
- OCIThreadMutexDestroy(), 15-197
- OCIThreadMutexInit(), 15-198
- OCIThreadMutexRelease(), 15-199
- OCIThreadProcessInit(), 15-200
- OCIThreadTerm(), 15-201
- OCITransCommit(), 15-203
- OCITransDetach(), 15-206
- OCITransForget(), 15-207
- OCITransPrepare(), 15-208
- OCITransRollback(), 15-209
- OCITransStart(), 15-210
- OCIType
 - 説明, 11-23
- OCITypeArrayByName(), 16-64
- OCITypeArrayByRef(), 16-67
- OCITypeByName(), 16-69
- OCITypeByRef(), 16-71
- OCL_TYPECODE
 - 値, 3-21, 3-23
- OCITypeElem
 - 説明, 11-23
- OCITypeMethod
 - 説明, 11-23
- OCIUserCallbackGet(), 15-228
- OCIUserCallbackRegister(), 15-230
- OCI アプリケーション
 - オブジェクト付き
 - 初期化, 10-10
 - オブジェクトを使用した構造体, 10-4
 - での OTT の使用, 14-16
- OCI 環境
 - オブジェクトの初期化, 10-10

OCI 関数
NLS, xxxii, 2-3
新しいコールおよび更新されたコール, 1-12
コード, 15-234
コールの取消し, 2-32
サポートされない, 1-21
使用されなくなった, 1-19
データ・カートリッジ, xxxii, 2-3
リターン・コード, 2-26, 2-29
OCI コールの中止, 2-32
OCI コールの取消し, 2-32
OCI ナビゲーション・ファンクション, 13-20
確保 / 確保解除 / 解放関数, 13-20
その他の関数, 13-22
名前付けスキーム, 13-20
フラッシュ関数, 13-21
マーク関数, 13-21
メタ属性アクセス関数, 13-21
OCI プログラム, 「OCI アプリケーション」を参照。
OCI プロセス
オブジェクトの初期化, 10-10
OCI リレーショナル関数
アドバンスド・キューイングおよびパブリッシュ・サブスクライブ, 15-4
参照項目の概要, 18-2
接続、認証および初期化, 15-88
OID, 「オブジェクト識別子」を参照。
Oracle8 データ型
バインディングと定義, 12-6
Oracle コール・インタフェース, 「OCI」を参照。
Oracle データ型, 3-2
C へのマッピング, 11-2
oratypes.h
内容, 3-25
ORE, 「オブジェクト・ランタイム環境」を参照。
OTT
intype ファイル, 14-26
intype ファイルの提供, 14-7
outtype ファイル, 14-15
概要, 14-2
コマンド行, 14-5
コマンド行構文, 14-21
参照, 14-20
使用方法, 14-1, 14-2
制限, 14-34
データ型マッピング, 14-9
データベースでの型の作成, 14-4

パラメータ, 14-22
OTT, 「オブジェクト型トランスレータ」を参照。
OTT 初期化関数
コール, 14-18
作業, 14-20
OTT パラメータ
CASE, 14-25
CODE, 14-23
CONFIG, 14-24
ERRTYPE, 14-24
HFILE, 14-24
INITFILE, 14-23
INITFUNC, 14-24
INTYPE, 14-22
OUTTYPE, 14-23
SCHEMA_NAMES, 14-26
USERID, 14-22
表示される場所, 14-26
OUTTYPE OTT パラメータ, 14-23
outtype ファイル, 14-26
OTT の実行時, 14-15

P

PDO, 「パラメータ記述子オブジェクト」を参照。
PL/SQL, 1-9
OCI アプリケーションでの使用, 2-37
OCI プログラムでの使用, 5-7
REF カーソルのバインディングと定義, 5-28
出力変数の定義, 5-16
ネストした表のバインディングと定義, 5-28
ピース単位操作, 5-34
プレースホルダのバインド, 2-37

R

RAW
外部データ型, 3-13
REF
外部データ型, 3-17
サーバーからの取得, 10-10
定義, 5-16, 12-4
バインディング, 5-10, 12-3
標識変数, 2-30, 2-31
REF カーソル
バインディングと定義, 5-28
REF カーソル変数

バインディング, 5-11
RETURNING 句
OCI の使用, 5-20
REF で, 5-22
エラー処理, 5-22
バインディング, 5-21
変数の初期化, 5-21
RETURNING 句を使用する DML
「RETURNING 句」を参照。
ROWID
位置づけ更新および位置づけ削除に使用, 2-32
外部データ型, 3-12
ユニバーサル ROWID, 3-5
論理, 3-5
ROWID 記述子, 2-16

S

sb1
定義, 3-25
sb2
定義, 3-25
sb4
定義, 3-25
SCHEMA_NAMES OTT パラメータ, 14-26
使用方法, 14-30
SQLT_NTY
オブジェクト・メモリの事前割当て, 12-5
説明, 3-16
定義の例, 12-13
バインドの例, 12-12
SQLT_REF
説明, 3-17
定義, 3-17
SQLT 型コード, 3-23
SQL 問合せ
結果のフェッチ, 4-14
出力変数の定義, 4-13, 5-13, 12-4
出力変数の定義, 「操作の定義」を参照。
プレースホルダのバインド, 「バインド操作」を参照。
文の種類, 1-8
SQL 文, 1-7
型
PL/SQL, 1-9
埋込み SQL, 1-10
制御文, 1-8

データ操作言語, 1-8
データ定義言語, 1-7
問合せ, 1-8
実行, 4-6
実行の準備, 4-4
準備するタイプの決定, 4-4
処理, 4-2
バインディング・プレースホルダ, 4-5, 5-2, 12-2
SQL 文の実行, 4-6
STRING
外部データ型, 3-10
sword
定義, 3-25

T

TDO
型記述子オブジェクト, 「TDO」を参照。
取得, 11-23
説明, 11-23
定義, 12-2
TDO, 「型記述子オブジェクト」を参照。

U

ub1
定義, 3-25
ub2
定義, 3-25
ub4
定義, 3-25
Unicode
OCILobRead(), 15-154
OCILobWrite(), 15-159
および UTF-8, 5-27
キャラクタ・セット ID, A-20, A-22
固定幅サポート, 5-27
対応, 5-27
UNSIGNED
外部データ型, 3-14
UROWID
ユニバーサル ROWID, 3-5
USERID OTT パラメータ, 14-22
utext
Unicode データ型, 5-27

V

- VARCHAR
 - 外部データ型, 3-12
- VARCHAR2
 - 外部データ型, 3-8
- VARNUM
 - 外部データ型, 3-11
- VARRAW
 - 外部データ型, 3-14

W

- with_context
 - 外部プロシージャ関数の引数, 18-3

X

- XID, 「トランザクション識別子」を参照。
- xtrammem_sz パラメータ
 - 使用方法, 2-17

あ

- 値, 10-5
 - オブジェクト・アプリケーション, 10-7
- アップグレード
 - 7.x から 8.0, 1-21
 - 7.x から 8.0 OCI, 1-22
- アドバンスト・キューイング
 - OCI および, 9-24
 - OCI 関数, 9-24
 - OCI 対 PL/SQL, 9-25
 - OCI の説明, 9-24
 - エンキュー関数, 15-7
 - 説明, 9-24
 - デキュー関数, 15-5
 - 例, 15-8
- アドバンスト・キューイング関数, 15-4
- アトミック NULL, 10-27
- アプリケーション
 - リンク, 2-34
- アプリケーション・フェイルオーバー
 - OCI コールバック, 9-17
 - コールバックの登録, 9-19
 - コールバックの例, 9-19

い

- 移行
 - 7.x から 8.0, 1-21
 - セッション, 8-10
- 移植
 - セッション, 15-104
- 一時オブジェクト, 10-6
 - LOB
 - 属性, 7-5
 - メタ属性, 10-19
- 位置づけ, 2-32
 - 削除, 2-32
- 一貫性
 - オブジェクト・キャッシュ, 13-5

う

- 埋込み SQL, 1-10
 - OCI コールとの混在, 1-10
- 埋込みオブジェクト
 - フェッチ, 10-15

え

- 永続オブジェクト, 10-5
 - メタ属性, 10-16
- エラー
 - オブジェクト・アプリケーションでの処理, 10-33
 - 処理, 2-26
 - 処理例, 2-27
- エラー・コード
 - 定義コール, 2-28
 - ナビゲーション関数, 16-5
- エラー・ハンドル
 - 説明, 2-9
 - 属性, A-6

お

- オブジェクト
 - 2次メモリー, 13-17
 - LOB 属性, 7-5
 - NULL, 10-27
 - OCI オブジェクト・アプリケーション構造体, 10-4
 - OCI でのアクセス, 14-17
 - OCI での使用方法, 10-3

- OCI での操作, 14-17
 - 一時, 10-5, 10-6
 - 一時オブジェクトの LOB 属性, 7-5
 - インスタンスのメモリー・レイアウト, 13-17
 - 永続, 10-5
 - 解放, 10-30, 13-9
 - 確保, 10-11, 13-7
 - 確保解除, 10-27, 13-8
 - 確保カウント, 10-27
 - 確保継続時間, 13-14
 - 型, 10-5, 16-2
 - クライアント側キャッシュ, 13-2
 - 継続時間, 13-14
 - コピー, 10-30
 - 作成, 10-30
 - 属性, 10-16
 - 操作, 10-13
 - 存続期間, 16-2
 - トップレベル・メモリー, 13-17
 - ナビゲーション, 13-18
 - 単純, 13-18
 - 配列確保, 10-12
 - フラッシュ, 13-10
 - 変更のフラッシュ, 10-14
 - マーク, 10-14, 13-9
 - マーク解除, 13-10
 - メタ属性, 10-16
 - メモリー管理, 13-2
 - 用語, 10-5, 16-2
 - リフレッシュ, 13-11
 - ロック, 13-12
 - 割当て継続時間, 13-14
- オブジェクト・アプリケーション
 - コミット, 13-14
 - データベース接続, 10-10
 - ロールバック, 13-14
- オブジェクト型トランスレータ
 - OCI での使用方法, 10-8
 - 「OTT」を参照。
 - サンプル出力, 10-9
- オブジェクト関数
 - サーバー・ラウンドトリップ, C-4
 - 「ナビゲーション関数」を参照。
- オブジェクト・キャッシュ, 13-2
 - 一貫性, 13-5
 - オブジェクトの削除, 13-7
 - オブジェクトのロード, 13-7

- コヒーレンス, 13-5
- サイズの設定, 13-5
- 初期化, 10-10
- 操作, 13-6
 - メモリー・パラメータ, 13-5
- オブジェクト参照, 10-32
- オブジェクト参照, 「REF」を参照。
- オブジェクト識別子
 - 永続オブジェクト用, 10-5
- オブジェクト付き OCI アプリケーション
 - 一般的構造, 2-3
 - 構造, 2-3
 - コンパイル, 1-4
 - 終了, 2-25
 - 初期化例, 2-23
 - ステップ, 2-17
 - リンク, 1-4
- オブジェクトの確保, 13-7
- オブジェクト・ランタイム環境
 - 初期化, 10-10

か

- 外部データ型, 3-4, 3-6
 - CHAR, 3-14
 - CHARZ, 3-16
 - DATE, 3-12
 - FLOAT, 3-10
 - INTEGER, 3-10
 - LOB, 3-17
 - LONG, 3-12
 - LONG RAW, 3-14
 - LONG VARCHAR, 3-14
 - LONG VARRAW, 3-14
 - NUMBER, 3-9
 - RAW, 3-13
 - REF, 3-17
 - ROWID, 3-12
 - SQLT_BLOB, 3-17
 - SQLT_CLOB, 3-17
 - SQLT_NCLOB, 3-17
 - SQLT_NTY, 3-16
 - SQLT_REF, 3-17
 - STRING, 3-10
 - UNSIGNED, 3-14
 - VARCHAR, 3-12
 - VARCHAR2, 3-8

- VARNUM, 3-11
- VARRAW, 3-14
 - 名前付きデータ型, 3-16
 - 変換, 3-19
- 外部プロシージャ
 - OCI コールバック, 9-16
- 外部プロシージャ関数
 - with_context 型, 18-3
 - リターン・コード, 18-3
- 解放
 - オブジェクト, 10-30, 13-9
- 拡張子
 - OTT のデフォルトのファイル名, 14-32
- 確保, 13-7
- 確保解除, 10-27, 13-8
 - オブジェクト, 13-8
- 確保カウント, 10-27
- 確保継続時間
 - オブジェクトの, 13-14
 - 例, 13-15
- 型
 - 記述, 6-2
 - 属性, 6-9
- 型関数
 - 属性, 6-11
- 型記述子オブジェクト, 11-23
- 型コード, 3-21
- 型参照, 10-32
- 型属性
 - 属性, 6-10
- 型プロシージャ
 - 属性, 6-11
- 環境ハンドル
 - 説明, 2-8
 - 属性, A-3
- 関数
 - コード, 15-234
- 関数名
 - コーディング・ガイドライン, 2-34

き

- キーワード, xxxiii, 2-33
- 記述
 - 暗黙的, 4-11
 - 型の, 6-2
 - コレクションの, 6-2

- シノニムの, 6-2
- 順序の, 6-2
- スキーマの, 6-2
- ストアド・ファンクション, 6-2
- ストアド・プロシージャの, 6-2
- 選択リスト, 4-10
- データベースの, 6-2
- パッケージの, 6-2
- ビュー, 6-2
- 表, 6-2
- 明示的, 4-13
- 明示的および暗黙的, 6-4
- 記述関数, 15-42
- 記述子, 2-14
 - ROWID, 2-16
 - オブジェクト, 11-23
 - スナップショット, 2-15
 - パラメータ, 2-16
 - 複合オブジェクト検索, 2-16
 - 割当て, 2-21
- 記述子オブジェクト, 11-23
- 記述子関数, 15-27
- 記述操作
 - サーバー・ラウンドトリップ, C-6
- 記述ハンドル
 - 説明, 2-11
 - 属性, A-24
- キャッシュ関数
 - サーバー・ラウンドトリップ, C-4
- キャラクタ・セット ID, 5-24
 - Unicode, A-20, A-22
- キャラクタ・セット・フォーム, 5-24
- 共有データ構造モード, 2-19
- 共有モード, 2-19
 - OCIInitialize(), 2-20
 - OCIStmtPrepare(), 2-20
 - 環境変数の使用方法, 2-20

<

- グローバル・トランザクション, 8-4

け

- 継続時間
 - オブジェクトの, 13-14
 - 例, 13-15

こ

- 更新, 2-32
 - 位置づけ, 2-32
 - ピース単位, 5-30, 5-32
- 構造体
 - 配列, 5-17
- 構造体の配列, 5-17
 - 使用される OCI コール, 5-19
 - スキップ・パラメータ, 5-18
 - 標識変数, 5-19
- コーディング・ガイドライン
 - 関数名, 2-34
 - 予約語, 2-33
- コード
 - 関数, 15-234
 - デモ・プログラムのリスト, B-1
 - 例題プログラム, B-1
- コールバック
 - LOB の書込み用, 7-14
 - LOB ストリーム・インタフェース, 7-12
 - LOB 操作作用, 7-12
 - LOB の読み込み用, 7-13
 - アプリケーション・フェイルオーバー, 9-17
 - アプリケーション・フェイルオーバーのための登録, 9-19
 - 外部プロシージャから, 9-16
 - 制限, 9-15
 - 動的な登録, 9-14
 - パラメータ・モード, 15-64
 - ユーザー定義関数, 9-10
- コピー
 - オブジェクト, 10-30
- コヒーレンス
 - オブジェクト・キャッシュ, 13-5
- コミット, 2-25
 - オブジェクト・アプリケーション, 13-14
 - グローバル・トランザクションの 1 フェーズ, 8-7
 - グローバル・トランザクションの 2 フェーズ, 8-7
- コレクション
 - 記述, 6-2
 - スキャン・ファンクション, 11-18
 - 説明, 11-17
 - 操作作用関数, 11-17
 - 属性, 6-13
 - データ操作関数, 11-18

さ

- サーバー・ハンドル
 - サービス・コンテキストでの設定, 2-10
 - 説明, 2-9
 - 属性, A-9
- サーバー・ラウンドトリップ
 - LOB 関数, C-4
 - オブジェクト関数, C-4
 - 記述操作, C-6
 - キャッシュ関数, C-4
 - データ型のマッピングおよび操作関数, C-6
 - リレーショナル関数, C-2, C-7
- サービス・コンテキスト・ハンドル
 - 説明, 2-9
 - 属性, A-7
 - 要素, 2-9
- 最適ロック
 - 実現, 13-13
- 削除
 - 位置づけ, 2-32
- 作成
 - オブジェクト, 10-30
- サブスクリプション・ハンドル, 2-11
 - 属性, A-37
- 参照, 「REF」を参照。
- サンプル・プログラム, B-1

し

- 実行
 - 複数サーバーに対する, 4-5
 - モード, 4-7
- シノニム
 - 記述, 6-2
 - 属性, 6-14
- 順序
 - 記述, 6-2
 - 属性, 6-14
- 初期化関数, 15-88
- 新機能, 1-12
 - 紹介, 1-12
 - 利点, 1-12
- シングルタスク・リンク
 - サポート, 1-26

す

スキーマ

- 記述, 6-2
- 属性, 6-19

スキップ・パラメータ

- 構造体の配列, 5-18
- 標準配列, 5-19

ストアド・ファンクション

- 記述, 6-2

ストアド・プロシージャ

- 記述, 6-2

スナップショット

- 実行, 4-6

スナップショット記述子, 2-15

スナップショットの実行, 4-6

スレッド管理関数, 15-174

スレッド・セーフティ, 8-13

- 3層アーキテクチャ, 8-14
- 7.x と 8.0 コールの混在, 8-16
- OCI でのインプリメント, 8-14
- 基本概念, 8-14
- 必要な OCI コール, 8-14
- 利点, 8-13

せ

静的配列

- 定義, 5-16
- バインディング, 5-9

セッション

- 移行, 8-10
- 移植, 15-104

セッション管理, 8-10, 8-12

接続関数, 15-88

接続モード

- 非ブロック化, 2-35

選択リスト

- 記述, 4-10

そ

操作の定義, 4-13, 5-13, 12-4

- LOB, 5-16
- PL/SQL 出力変数, 5-16
- REF, 5-16, 12-4
- 使用するステップ, 5-14

静的配列, 5-16

名前付きデータ型, 5-16, 12-4

ピース単位フェッチ, 5-16

例, 5-14

挿入

- ピース単位, 5-30, 5-32

属性

- オブジェクトの, 10-16
- パラメータ記述子の, 6-5
- パラメータの, 6-5
- ハンドル, 2-12

属性記述子オブジェクト, 11-23

その他の関数, 15-218

た

ダイレクト・パス・ハンドル, 2-12

ダイレクト・パス・ロード, 9-36

- コンテキスト・ハンドルの属性, A-39

- ストリーム・ハンドルの属性, A-43

制限, 9-37

ダイレクト・パス・コンテキスト・ハンドル, 9-38

ダイレクト・パス・ストリーム・ハンドル, 9-39

ダイレクト・パス列配列ハンドル, 9-39

ハンドル, 9-38

ハンドルの属性, A-39

ファンクション, 9-40

例, 9-41

列のデータ型, 9-38, A-47

列配列ハンドルの属性, A-42

列パラメータの属性, A-44

ダイレクト・パス・ロード関数, 15-73

ち

抽象データ型

- C アプリケーションでの表現, 10-8

て

定義

- OCINumber, 12-7

配列, 12-6

- リターン・コードとエラー・コード, 2-28

定義関数, 15-42

定義ハンドル

- 説明, 2-10

- 属性, A-22
- データ・カートリッジ
 - OCI 関数, xxxii, 2-3
- データ型
 - BFILE, 3-18
 - BLOB, 3-18
 - CLOB, 3-19
 - FILE, 3-18
 - NCLOB, 3-19
 - OCI での操作, 11-5
 - Oracle, 3-2
 - Oracle から C へのマッピング, 11-2
 - 外部, 3-4, 3-6
 - ダイレクト・パス・ロード, 9-38, A-47
 - 内部, 3-3, 3-4
 - ピース単位操作作用, 5-31
 - 変換, 3-19
- データ型コード
 - 内部, 3-4
- データ型のマッピングおよび操作関数
 - サーバー・ラウンドトリップ, C-6
- データ型マッピング
 - Oracle 方法論, 11-4
 - OTT, 14-9
- データ構造
 - 8.0 用の新規, 2-5
- データ操作言語
 - SQL 文, 1-8
- データ定義言語
 - SQL 文, 1-7
- データベース
 - 記述, 6-2
 - 属性, 6-19
- データベース接続
 - オブジェクト・アプリケーション用, 10-10
- デフォルトのファイル名拡張子
 - OTT, 14-32
- デフォルト名のマッピング
 - OTT, 14-32
- デモ・プログラム, B-1
 - リスト, B-1

と

- 問合せ
 - 明示的記述, 4-13
- 問合せ, 「SQL 問合せ」を参照。

- 登録
 - ユーザー・コールバック, 9-10
- トップレベル・メモリー
 - オブジェクト, 13-17
- トランザクション
 - OCI 関数
 - トランザクション用, 8-2
 - グローバル, 8-4
 - 1 フェーズ・コミット, 8-7
 - 2 フェーズ・コミット, 8-7
 - トランザクション識別子, 8-4
 - ブランチ, 8-4
 - ブランチの状態, 8-6
 - グローバルな例, 8-8
 - コミット, 2-25
 - 初期化パラメータ, 8-10
 - 直列可能, 8-3
 - 読み込み専用, 8-3
 - ローカル, 8-3
 - ローリング・バック, 2-25
- トランザクション関数, 15-202
- トランザクション識別子, 8-4
- トランザクションの複雑性
 - OCI でのレベル, 8-3
- トランザクション・ハンドル
 - 説明, 2-9
 - 属性, A-13

な

- 内部データ型, 3-3, 3-4
 - データ型コード, 3-4
 - 変換, 3-19
- ナビゲーション関数
 - エラー・コード, 16-5
 - 戻り値, 16-5
 - 用語, 16-3
- ナビゲーション, 13-18
- 名前付きデータ型
 - 外部データ型, 3-16
 - 定義, 3-16, 5-16, 12-4
 - バインディング, 5-9, 12-2
 - バインディングと定義, 12-6
 - 標識変数, 2-30, 2-31

に

認証関数, 15-88
認証管理, 8-10

ね

ネストした表
操作関数, 11-20
要素順序, 11-20

は

配列

スキップ・パラメータ, 5-19
定義, 12-6
バインド, 12-3

バインディング

OCINumber, 12-7
PL/SQL プレースホルダ, 2-37
配列, 12-3
まとめ, 5-12

バインド関数, 15-42

バインド操作, 4-5, 5-2, 12-2

LOB, 5-10
OCI_DATA_AT_EXEC モード, 5-11
OCI 配列インタフェース, 5-4
PL/SQL, 5-5
REF, 5-10, 12-3
REF カーソル変数, 5-11
行われた関係付け, 5-3
使用するステップ, 5-6
静的配列, 5-9
定位置対名前付き, 5-4
名前付き対定位置, 5-4
名前付きデータ型, 5-9, 12-2
変数の初期化, 5-3
例, 5-6

バインド・ハンドル

説明, 2-10
属性, A-20

パスワード管理, 8-10, 8-12

パッケージ

記述, 6-2
属性, 6-8

パブリッシュ - サブスクライブ

COMPATIBLE パラメータ, 9-29

_SYSTEM_TRIG_ENABLED パラメータ, 9-32
関数, 9-29

サブスクリプション・ハンドル, 9-29

通知機能, 9-28

通知コールバック, 9-30

ハンドル属性, 9-30

ハンドルの属性, A-37

例, 9-32

パブリッシュ - サブスクライブ関数, 15-4

パラメータ

属性, 6-5
モード, 15-2, 18-2
文字列長, 15-3
文字列を渡す, 2-30

パラメータ記述子, 2-16

属性, 6-5, A-24

パラメータ記述子オブジェクト, 11-23

ハンドル

C データ型, 2-7
エラー・ハンドル, 2-9
親が解放される際に解放される子, 2-7
階層, 2-8
解放, 2-7
型, 2-7
環境ハンドル, 2-8
記述ハンドル, 2-11
サーバー・ハンドル, 2-9
サービス・コンテキスト・ハンドル, 2-9
サブスクリプション, 2-11, 9-29
ダイレクト・パス, 2-12
定義ハンドル, 2-10
トランザクション・ハンドル, 2-9
バインド・ハンドル, 2-10
プロセス, 2-12
プロセスの属性, A-49
文ハンドル, 2-10
ユーザー・セッション・ハンドル, 2-9
利点, 2-8
割当て, 2-7, 2-21

ハンドル関数, 15-27

ハンドル属性, 2-12

設定, 2-12

読み込み, 2-12

ひ

ピース単位操作, 5-32

- PL/SQL, 5-34
- 更新, 5-30
- 挿入, 5-30
 - フェッチ, 5-30, 5-35
 - 有効なデータ型, 5-31
- ピース単位フェッチ, 5-34
- 引数
 - 属性, 6-16
- 非ブロック化モード, 2-35
 - 例, 2-36
- ビュー
 - 記述, 6-2
 - 属性, 6-7
- 表
 - 記述, 6-2
 - 属性, 6-7
- 標識変数, 2-30
 - PL/SQL OUT バインド, 12-4
 - REF 定義, 12-4
 - REF バインド, 12-3
 - REF 用, 2-30, 2-31
 - 構造体の配列, 5-19
 - 名前付きデータ型の定義, 12-4
 - 名前付きデータ型のバインド, 12-3
 - 名前付きデータ型用, 2-30, 2-31

ふ

- ファンクション
 - 属性, 6-8
- 関数
 - 新しいコールおよび更新されたコール, 1-12
- フェッチ
 - ピース単位, 5-30, 5-34
- フェッチ操作, 4-14
 - LOB データ, 4-14
 - プリフェッチ・カウントの設定, 4-14
- 複合オブジェクト検索, 10-19
 - 実現, 10-22
 - ナビゲーション・プリフェッチ, 10-23
- 複合オブジェクト検索 (COR) 記述子, 2-16
 - 属性, A-26
- 複合オブジェクト検索 (COR) ハンドル, 2-11
 - 属性, A-25
- 複数サーバー
 - 文の実行, 4-5
- フラッシュ, 13-10

- オブジェクト, 13-10
- オブジェクトの変更, 10-14
- ブランチ
 - 再開, 8-7
 - 連結解除, 8-7
- ブランチの再開, 8-7
- ブランチの連結解除, 8-7
- プリフェッチ
 - OCIStmtExecute() 中に, 4-6
 - 行カウントの設定, 4-14
 - プリフェッチ・メモリー・サイズの設定, 4-14
- プロシージャ
 - 属性, 6-8
- プロセス
 - ハンドルの属性, A-49
- プロセス・ハンドル, 2-12
- ブロック化モード, 2-35
- 文ハンドル
 - 説明, 2-10
 - 属性, A-14

ほ

- 方法記述子オブジェクト, 11-23

ま

- マーク
 - オブジェクト, 13-9
- マーク解除, 13-10
 - オブジェクト, 13-10
- マルチスレッド開発
 - 基本概念, 8-14

め

- メタ属性
 - 一時オブジェクト, 10-19
 - 永続オブジェクト, 10-16
 - オブジェクトの, 10-16

も

- 文字列
 - パラメータとして渡す, 2-30
- 戻り値
 - ナビゲーション関数, 16-5

ゆ

- ユーザー・セッション・ハンドル
 - サービス・コンテキストでの設定, 2-10
 - 説明, 2-9
 - 属性, A-12
- ユーザー定義コールバック関数, 9-10
 - 登録, 9-10
- ユーザー・メモリー
 - 割当て, 2-17
- ユニバーサル ROWID, 3-5

よ

- 用語
 - このマニュアルで使用される, 1-10
 - ナビゲーション関数, 16-3
- 予約語, xxxiii, 2-33
- 予約済み名前領域, 2-33

ら

- ラウンドトリップ
 - 「サーバー・ラウンドトリップ」を参照。

り

- リスト
 - 属性, 6-18
- 利点
 - OCI, 1-3
- リフレッシュ, 13-11
 - オブジェクト, 13-11
- リリースの機能拡張, 1-12
- リレーショナル関数
 - サーバー・ラウンドトリップ, C-2, C-7
- リンク, 2-34
 - シングルタスクのサポート, 1-26
 - モード, 1-24
 - 問題, 1-24

れ

- 例
 - OCIThread の使用方法, 9-8
 - デモ・プログラム, B-1
 - 非ブロック化モード, 2-36

列

- 属性, 6-4, 6-15

ろ

- ロールバック, 2-25
 - オブジェクト・アプリケーション, 13-14
- ロケータ, 2-14
 - FILE 用, 7-3
 - LOB データ型用, 2-15, 7-2
- ロック, 13-12
 - オブジェクト, 13-12
 - 最適モデル, 13-13

わ

- 割当て継続時間
 - オブジェクトの, 13-14
 - 例, 13-15