# 開発者ガイド

## iPlanet Trustbase Transaction Manager

Release 2.2.1

2001 年 3 月

# 目次

# 図一覧

# 概要

iPlanet Trustbase Transaction Manager フレームワークのドキュメントセットの中の 1 冊であるこのドキュメントは、iPlanet Trustbase Transaction Manager フレームワークを活用してアプリケーションを作成したい設計者および開発者を対象とし、次の 3 つのセクションに大別されています。

- iPlanet Trustbase Transaction Manager が対応する環境の紹介
- フレームワークの概要およびコンポーネント間のインタラクション方法
- iPlanet Trustbase Transaction Manager アプリケーションの開発

# 全体の構成

ドキュメントセットは、次の各ドキュメントから構成されています。

- iTTM2.2-Utility-Guide.pdf。PKI 証明書管理に役立ついくつかのユーティリティについて説明している
- iTTM2.2-Install-Configuration-Guide.pdf。iPlanet Trustbase Transaction Manager のフレームワークを利用するアプリケーションを制作しようとする開発者向け。iPlanet Trustbase Transaction Manager プラットフォームをインストールするために必要な情報を提供するガイド。インストールに先立ち必要なソフトウェアとハードウェア、および CD-ROM から iPlanet Trustbase Transaction Manager をインストールする方法について説明している
- iTTM2.2-Developer-Guide.pdf ( このドキュメント )。独自のサービスを構築および配置する方法について説明している
- API リファレンス (< インストールディレクトリ >/Trustbase/TTM/V2.2/apidocs)。iPlanet Trustbase Transaction Manager インストール時に同時にインストールされる、ソフトコピー形式の Java ドキュメントセット。アプリケーション開発者向けに iPlanet Trustbase Transaction Manager のフレームワークおよびツールを利用するための情報を提供している

このマニュアルは、Java 標準 (`http://www.javasoft.com`) および XML (`http://www.w3.org/TR/REC-xml`) に関する知識を有する読者を対象としています。

また、このマニュアルは、読者が iPlanet Trustbase Transaction Manager 開発者トレーニングコースを受講済みであることを前提としています。

# 関連ドキュメント

- Solaris 8 および Java Development Kit 1.2.1

  英語 :

  http://docs.sun.com

  http://java.sun.com/products/jdk/1.1/docs/index.html

  http://www.sun.com/software/solaris/cover/sol8.html

  日本語 :

  http://www.sun.co.jp/software/solaris/cover/sol8.html

  http://docs.sun.com/ab2/products_ja/INDEX/@ProductViewer/8339/*;td=1?Ab2Lang=ja&Ab2Enc=euc-jp

- Java

  英語 :

  http://www.javasoft.com

  日本語 :

  http://java.sun.com/products/jdk/1.2/download-ja-docs.html

- iPlanet Application Server 6.0

  英語 :

  http://docs.iplanet.com/docs/manuals/ias.html

  http://www.iplanet.com/products/infrastructure/app_servers/index.html

  http://www.iplanet.com/products/iplanet_application/home_2_1_1n.html

  日本語 :

  http://www.iplanet.ne.jp/products/ias6/index.html

  http://docs.iplanet.com/docs/manuals/ias/60/sp2/ja/ReadMe_ja.html

- iPlanet Web Server 6.0[iPlanet Web Server 4.1]

  英語 :

  http://docs.iplanet.com/docs/manuals/enterprise.html

  http://www.iplanet.com/products/infrastructure/web_servers/index.html

  http://www.iplanet.com/products/iplanet_web_enterprise/home_2_1_1m.html

  日本語 :

  http://www.iplanet.ne.jp/products/iws4_1/index.html

http://docs.iplanet.com/docs/manuals/enterprise/41/rn41sp5_JP.htm

http://docs.iplanet.com/docs/manuals/enterprise/41/ja/ag/contents.htm

http://docs.iplanet.com/docs/manuals/enterprise/41/ja/ig/contents.htm

- iPlanet Certificate Management System

  http://docs.iplanet.com/docs/manuals/cms.html

- 『Oracle 8i Installation Guide』および『Oracle 8i Configuration Guide』

  英語 :

  http://www.oracle.com

  日本語 :

  http://www.oracle.co.jp

- Hardware Security nCipher KeySafe 1.0 および CAFast

  英語 :

  http://www.ncipher.com

  日本語 :

  http://www.tel.co.jp

- Identrus メッセージ仕様

  http://www.identrus.com

  Transaction Coordinator requirements (IT-TCFUNC)

  Core messaging specification (IT-TCMPD)

  Certificate Status Check Messaging specification (IT-TCCSC)

# 概要

iPlanet Trustbase Transaction Manager プラットフォームは、銀行業務用および商業用のさまざまなアプリケーションをサポートするメッセージ指向ミドルウェアプラットフォームを提供します。このプラットフォームは、金融機関による Identrus ネットワークの利用を可能にすることを目的として設計されており、Identrus のメッセージング仕様に準拠するメッセージの処理に必要な機能をすべて搭載しています。

## iPlanet Trustbase Transaction Manager プラットフォーム

iPlanet Trustbase Transaction Manager プラットフォームは、次のような高度な性能を備えています。

- SSLv3 上の HTTP および SMTP 上の SMIMEv2 を含む、セキュリティ保護されたチャネル通信をサポートする専用のセキュリティサービスを通じてメッセージを受信
- リクエストはメッセージ解析エンジンと検証エンジンを通過する。プラットフォームにより提供される検証エンジンは入ってきたメッセージをサービスをリクエストすることによって検証し、システムを通じて処理されるサービスリクエストメッセージリクエストを完全に正しくフォーマットおよび構築する。プラットフォームは、XML および HTML を含むさまざまなメッセージコーディングの標準規格をサポートする。ネームサービスは、メッセージオブジェクトの配信先となるアプリケーションロジックを特定する
- アプリケーションロジック。プラットフォームはさまざまなアプリケーションロジックをサポートする。また、サードパーティのものを配置することにより拡張が可能
- サードパーティサービス。iPlanet Trustbase Transaction Manager プラットフォーム上で実行されるアプリケーションは、意思決定プロセス中にサードパーティシステムにリクエストを送ることが可能 ( 組織のオペレーションシステムおよびサードパーティサービスを含む )
- 応答管理。信頼パーティへの応答は、プラットフォームがサポートする各トランザクションに対して定義されているポリシーに基づいて構築され、リクエスト元のパーティに送信される

トランザクション処理プロセスの各段階は、レポジトリの監視機能によって記録可能です。

# Identrus トランザクションコーディネータ

iPlanet Trustbase Transaction Manager は、次の Identrus ドキュメントの指定に従って Identrus トランザクションコーディネータの実装を提供しています。

| 題名 | 説明およびドキュメントリファレンス |
|---|---|
| Transaction Coordinator | Transaction Coordinator requirements (IT-TCFUNC) |
| Transaction Coordinator Messaging Protocol | Core messaging specification (IT-TCMPD) |
| Transaction Coordinator Certificate Status Check Protocol Definition | Certificate Status Check Messaging specification (IT-TCCSC) |

iPlanet Trustbase Transaction Manager は、トランザクションコーディネータのコア CSC 機能を実行する、拡張可能なプラットフォームです。さらに、開発者が Identrus のメッセージング仕様に準拠するアプリケーションを作成するために利用することも可能です。

iPlanet Trustbase Transaction Manager は、開発者による Identrus 準拠アプリケーションの作成を簡易化する次の機能を提供しています。

- トランスポートメカニズムに依存しないメッセージの標準的な提示
- リクエストの標準的な認証および認可
- メッセージ内の Identrus ネットワーク層情報に対する共通妥当性検査およびエラー処理
- Identrus 仕様に準拠する新しいアプリケーションメッセージタイプ生成のためのツール

このガイドの前半では、これらの共通コンポーネントを紹介し、各コンポーネントがサポートする機能について説明します。後半部分では、iPlanet Trustbase Transaction Manager インフラストラクチャ上で Identrus アプリケーションを作成および配置する方法について説明します。

# 製品の特長

iPlanet Trustbase Transaction Manager は、企業間商取引ソリューションのプラットフォームを提供し、次の課題に照準を合わせています。

**製品の特長**

| | |
|---|---|
| 識別 | さまざまなタイプの資格を認識し、リモートまたはサードパーティサービスへのリクエストを含む認可リクエストの処理の基盤としてこれらの資格を使用します。<br>iPlanet Trustbase Transaction Manager サードパーティおよびその他の外部 CA サービスを含む複数パーティ間の信頼関係構築をサポートします。 |
| 資格授与 | 取引相手を識別後、企業が潜在顧客に対してどのサービスを提供すべきかを判別するための手段を提供します。<br>ユーザタイプ、資格タイプ、アクセスチャネル、信用度などに基づいて、サービス供給の条件を指定および執行します。 |
| 紛争解決 | 証明書ベースのテクノロジを使用して、トランザクション署名、監視ログ、および履行拒否に関するサービスを提供します。 |
| 統合 | カスタマーリレーションシップマネジメントおよびキャッシュマネジメントソリューションなど、既存のバックオフィスおよびミドルオフィスソリューションと統合できます。<br>柔軟な電子取引モデルをサポートし、二者またはそれ以上の取引先間のトランザクションを可能にします。 |
| 可用性 | 夜間、週末、祝日を問わないグローバルな企業間取引を可能にします。 |
| スケーラビリティ | 予測外の性能要件にも対応し、顧客層の拡大に伴う需要を満たします。 |
| セキュリティ | すべてのパーティに対して、識別および認証サービス、データの機密保持および統合、トランザクションのセキュリティ、セキュリティ管理サービスなどを含む完全なセキュリティを提供します。 |

第 1 章

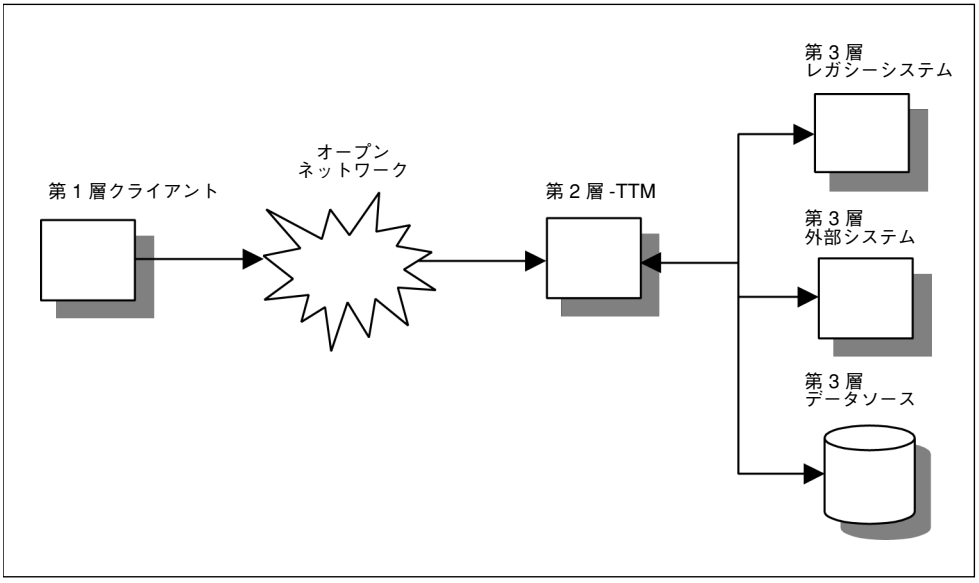
# iPlanet Trustbase Transaction Manager のアーキテクチャ

金融機関の Identrus 対応メッセージ指向ミドルウェアに対するニーズを満たすことを目的として設計された iPlanet Trustbase Transaction Manager は、一貫性のある、再使用可能な金融サービスアプリケーションをインターネット上で提供することを可能にします。

# 概要

多層アーキテクチャの中間層に位置するインフラストラクチャである iPlanet Trustbase Transaction Manager コンポーネントは、新機能を提供しつつ、第 1 層のクライアントを既存の複雑な企業インフラストラクチャから保護しています ( 図　1–1 参照 )。

図　1–1　　3 層アーキテクチャ




このアーキテクチャを実現するため、iPlanet Trustbase Transaction Manager は以下のような特徴を備えています。

- 可用性が高い
- 安全
- 信頼性が高く、かつスケーラブル

これらの特徴は、サーブレットや EJB などのサーバサイド標準規格を使用し、iPlanet Application Server の信頼性およびスケーラビリティ機能を活用することで実現されています。

iPlanet Trustbase Transaction Manager プラットフォームは、特定のタイプや形式のメッセージを処理するために拡張可能なメッセージ処理パイプラインを提供することで、iPlanet Application Server の機能を拡張します。このメッセージ処理パイプラインには、次の 4 つの主要コンポーネントがあります。

- トランスポートリスナー

- プレゼンテーションハンドラおよびフォーマットハンドラ
- ルーティングおよび認可管理
- ビジネスロジックプラグイン

メッセージパイプラインは、Identrus メッセージングプロトコルおよび HTML ベースの通信を使用するアプリケーションのサポートに必要な前処理をすべて行う、一連のコンポーネントから構成されています。

また、iPlanet Trustbase Transaction Manager プラットフォームの操作性は、開発者による Message クラスの生成および完成したアプリケーションの配置を可能にするツールキットによって補強されています。このツールキットは、Identrus 準拠アプリケーションの開発に必要な手間を最低限に抑えることを目的として設計されています。

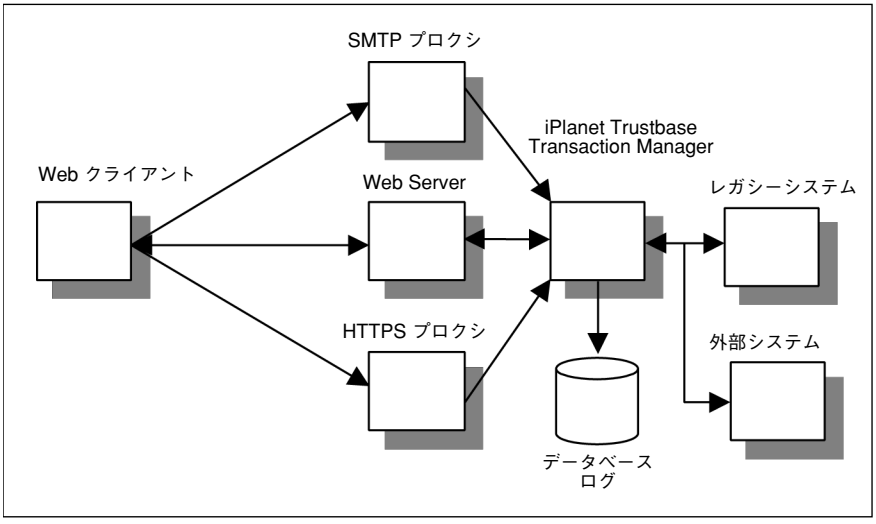
# 外部インターフェイス

iPlanet Trustbase Transaction Manager は、インターネット上の既存のレガシーデータソースへのアクセスおよび使用を可能にするミドルウェアコンポーネントであり、この機能を実現するため、次の能力を備えています。

- HTTP、HTTPS、SMIME などの Web プロトコルを待機および処理する
- JDBC、CORBA、および RMI を介して既存のレガシーシステムにアクセスする
- Web 上の他のサーバが提供するリソースを使用する

図 1-2 に、外部インターフェイスと iPlanet Trustbase Transaction Manager コンポーネントの関係を示します。

図 1–2 iPlanet Trustbase Transaction Manager のインターフェイス




## トランスポートプロトコル

iPlanet Trustbase Transaction Manager は、次の 3 種類の基本的なトランスポートプロトコルをサポートしています。

- SMTP - 電子メールの配信に使用される非同期通信
- HTTP - セキュリティ保護されていない同期通信
- SSL - セキュリティ保護された同期通信

iPlanet Trustbase Transaction Manager のリスナーは、トランスポート固有データの処理およびログを可能にするために、SMTP および SSL トランスポートプロトコルのプロクシとして機能します。HTTP データはすべて Web Server によって直接待機されます。

これら 3 種類のトランスポートプロトコルによって、さまざまなアプリケーションレベルメッセージングが可能となります。iPlanet Trustbase Transaction Manager には、次の種類のデータに直接対応するプレゼンテーションコンポーネントが含まれています。

- SMIME によってラップされた XML または HTML
- HTML
- Identrus 準拠 XML アプリケーションメッセージ

iPlanet Trustbase Transaction Manager プラットフォームは、その他のアプリケーションメッセージングプロトコルをサポートするよう拡張することも可能です。

## 企業接続性

iPlanet Trustbase Transaction Manager のビジネスロジックは、開発者によって実装されたビジネスロジックが、J2EE プラットフォームの標準接続性コンポーネントをすべて使用できるように設計されています。標準 J2EE 接続性コンポーネントには、次のようなものがあります。

- JDBC - リレーショナルデータベースへのアクセス
- RMI および CORBA - リモートオブジェクトおよび EJB へのアクセス
- JNDI - ディレクトリサービスおよびネームサービスへのアクセス
- JMS - MQ Series などのメッセージキューと併用するメッセージ指向インターフェイス

また、iPlanet Trustbase Transaction Manager のビジネスロジックは、iPlanet Application Server が内部システムへのアクセス用に提供している配下の接続性コンポーネントを活用する能力も備えています。これらの企業コネクタには、次のシステムに対応するインターフェイスがあります。

- R/3 - 企業リソース計画
- CICS - IBM メインフレーム統合
- BEA Tuxedo - トランザクションデータシステム
- Peoplesoft - 企業リソース計画

## サーバ間の接続性

iPlanet Trustbase Transaction Manager プラットフォームは、プレゼンテーションコンポーネント内の受信メッセージに使用されているトランスポート形式およびプレゼンテーション形式を抽象化する手段を提供しています。このメカニズムは、異なる API を通して、ビジネスサービスが他のサーバにリクエストを送信できるようにするためにも使用されています。

Identrus 証明書ステータス確認 (CSC) サービスでは、ある特定の時点での証明書の有効性を判別するためにこのメカニズムが使用されています。( 第 2 章、「プレゼンテーションロジック」を参照。)

## ルーティング

ルータは、コードの変更を必要としないセキュリティ保護された方法によって、サービス実行時にそのサービスを構造化および秩序化するためのメカニズムを提供します。また、各サービスが適切なコンテキストで認可された人物によってだけ実行されるように、門番として機能します。システムのユーザは、サーバに接続してメッセージを交換し、場合によっては特定のタスクを実行しようとすることもあります。タスクによっては、認可 ( したがって認証 ) を受けないと実行できないものもあります。また、サービスによっては、リクエストしているユーザの ID に基づいて、リクエストとは多少異なる形でタスクが実行される場合もあります。

ルータは、次の条件を満たすように設計されています。

- 認証および認可をビジネスロジックから切り離す
- ルーティングテーブルの構成および管理を簡易化し、エラーを減少する
- 必要に応じた複雑なソリューションの構築を可能にする
- 単純なソリューションを簡単に実装できるようにする
- サービスが不可分なビジネスレベル機能を実装でき、互いに独立できるようにする

ルータの機能は、iPlanet Trustbase Transaction Manager プラットフォームの中核を成すものです。メッセージはすべてルータを通り、それぞれのコンテキストおよび内容に基づいて受理または拒否されます。

さまざまな複雑な環境下で機能できる柔軟なメカニズムを実現するため、iPlanet Trustbase Transaction Manager は規則ベースのルーティングを提供しています。このため、既存のモジュールやサービスを修正することなく、必要に応じて iPlanet Trustbase Transaction Manager の動作を修正および拡張することが可能です。詳細は、第 3 章、「ルーティング」を参照してください。

## 認可

サービスへのアクセスを制御するという基本条件を満たすには、メッセージのタイプだけでなくその現在の認可レベルに基づいてメッセージを配信する能力が必要となります。iPlanet Trustbase Transaction Manager では、認可は認証の延長として見なされるため、各ユーザの ID に基づいてそのユーザがどのような権限を持っているかを判断できます。

iPlanet Trustbase Transaction Manager の認証メカニズムは独立したコンポーネントではなく、認証データはデフォルトの iPlanet Trustbase Transaction Manager フレームワークによって収集されます。また、ドメイン固有のサービスによってさらにデータが追加されることもあります。iPlanet Trustbase Transaction Manager プラットフォームは、ユーザ名およびパスワード、またはデジタル署名をユーザグループ ( 役割 ) に対応づけるデフォルト認可サービスを提供しています。さらに、ルータによって、サービスにアクセスしている役割が適切なアクセス権を持っているかどうかが確認されます。

デフォルトの認可サービスの代わりに、企業ディレクトリサービスなどの既存リポジトリにユーザ情報を対応づけるメカニズムを使用することも可能です。詳細は、第 3 章、「ルーティング」を参照してください。

## サービス

ビジネスサービスは、電子商取引アプリケーションの中核です。iPlanet Trustbase Transaction Manager では、開発者が記述したサービスを、プラットフォームに登録できます。これらのサービスを記述する際には、トランスポート固有の情報、プレゼンテーション固有の情報、ユーザの認可、ユーザリクエストの認証などの処理を考慮する必要がないため、アプリケーションの機能や、既存システムを Web 対応インフラストラクチャに統合することだけに集中できます。詳細は、第 5 章、「標準的なサービス」を参照してください。

第 2 章

# プレゼンテーションロジック

iPlanet Trustbase Transaction Manager システムは、メッセージを受信すると、そのメッセージを適切なサービスに配信します。配信先のサービスが直接そのメッセージを処理できない場合は、iPlanet Trustbase Transaction Manager が接続マネージャコンポーネントを使用してメッセージを適切に処理します。

# 概要

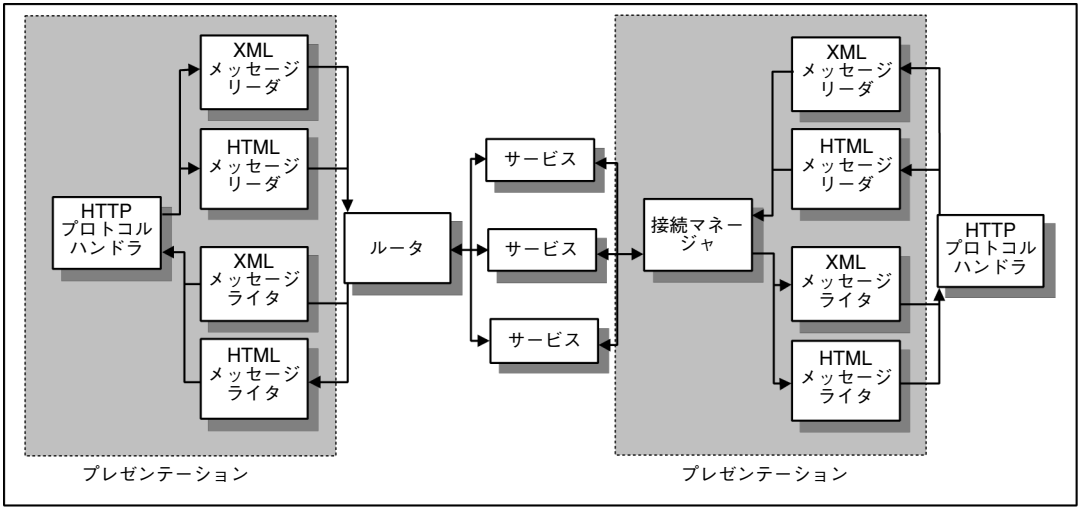
iPlanet Trustbase Transaction Manager のプレゼンテーションコンポーネントは、一連のプロトコルハンドラ、メッセージリーダ、およびメッセージライタから構成されています。プロトコルハンドラは、トランスポート固有のヘッダおよびフッタの抽出、MIME タイプおよびメッセージタイプの判別、およびメッセージ内容の適切なメッセージハンドラへの転送を行います。

メッセージハンドラは、プロトコルハンドラが判別した MIME タイプおよびメッセージタイプに基づいて選択されます。iPlanet Trustbase Transaction Manager には、次の種類のメッセージをそれぞれ処理する 2 つのデフォルトのメッセージハンドラがあります。

- Identrus XML メッセージ
- HTML メッセージ

Identrus XML メッセージの場合は、適切な Identrus ネットワーク処理が実行されます ( 詳細はこの章で後述 )。

図 2–1 プレゼンテーション層コンポーネント




HTML メッセージの場合は、メッセージリーダが HTML フォームのすべてのフィールドを XML 構造に変換します。この XML 構造は、一連の DOM ノードによって表されます。

XML または HTML のどちらの場合でも、ルータへの出力はセッション情報によって iPlanet Trustbase Transaction Manager メッセージにラップされます。その後、iPlanet Trustbase Transaction Manager メッセージは、ルータによって適切なサービスに送信されます。

サービスは、iPlanet Trustbase Transaction Manager メッセージを適切に変換または構築し、ルータに返します。ルータは、その iPlanet Trustbase Transaction Manager メッセージを適切なメッセージライタに送信します。HTML メッセージライタの場合は、このメッセージの DOM 値がメッセージタイプに基づく HTML テンプレートに抽出されます。テンプレートには、フォームデータの条件付き構築または再帰的構築のための一連の指示が含まれていることもあります。

# プロトコルハンドラ

プラットフォームがメッセージを受信すると、受信メッセージの MIME タイプに基づいてプロトコルハンドラが選択されます。このコンポーネントは、次のアプリケーションプロトコル情報を判別します。

- メッセージタイプ
- コンテキスト ID

各 MIME タイプ、したがってプロトコルハンドラは、メッセージのクラス (application/OCSP など ) を表します。クライアントは、適切な MIME タイプを生成する能力を備えている必要があります。クライアントが Web ブラウザの場合は、一般に MIME タイプ「application/x-www-form-url-encoded」が生成されます。iPlanet Trustbase Transaction Manager には、この MIME タイプを処理するデフォルトのプロトコルハンドラが組み込まれています。このプロトコルハンドラのデフォルトアクションは次のとおりです。

- フォームフィールド内の次のデータを検索する
  - メッセージタイプ - メッセージタイプを識別
  - コンテキスト ID - コンテキスト ID を識別
- 応答 MIME タイプを text/html に設定することで、返信が標準的な HTML ページであることをブラウザに伝える

## Identrus プロトコルハンドラ

プロトコルハンドラは、TC に配信されるメッセージのうち、「application/identrus-」文字列で始まる MIME タイプを持つものすべての初期処理を行います。

プロトコルハンドラが順次実行するアクションは次のとおりです。

- クライアントから原初 XML ストリームを読み取る
- 原初 XML を内部 iPlanet オブジェクト構造 ( メッセージツリー ) に解釈する
- XML メッセージ内の DOCTYPE によって指定されている DTD に照らし合わせて、メッセージツリーを確認する
- メッセージツリーをトラバースし、ID 属性がある場合は重複するものがないかどうかを確認する (XML 処理に必須のステップ。XML 構造がデジタル署名される場合は特に重要 )
- IdentrusConstants.SYSTEM_ID_ATTR が特定する iPlanet Trustbase Transaction Manager メッセージ属性に DOCTYPE タグのシステム識別子を入れる
- NIB から原初メッセージログ処理可能な要素を抽出する
- メッセージから証明書バンドルを抽出し、証明書ログにすべての証明書を記録する
- メッセージの証明書バンドル以外の部分を原初メッセージログに送信する

- ログから返された原初ログ ID を、IdentrusConstants.RAW_RECORD_MARKER が特定する iPlanet Trustbase Transaction Manager メッセージ属性に入れる
- iPlanet Trustbase Transaction Manager メッセージのタイプを IdentrusConstants.IDENTRUS_MESSAGE に設定する
- メッセージツリーをシリアル化された内容として iPlanet Trustbase Transaction Manager メッセージに付加することで、メッセージを適切なメッセージリーダに送信可能な状態にする

---

**注**　変数 DOCTYPE は、XML の概念です。`http://www.w3.org/TR/REC-xml` などを参照してください。
定数 SYSTEM_ID_ATTR、RAW_RECORD_MARKER および IDENTRUS_MESSAGE は、Identrus 定数です。Identrus 定数については、com.iplanet.trustbase.identrus.IdentrusConstants を参照してください。

---

# メッセージリーダ

適切なメッセージリーダの選択は、プロトコルハンドラが収集した情報に基づいて行われます。各メッセージリーダは、それぞれどのメッセージのクラスを内部 iPlanet Trustbase Transaction Manager メッセージに変換できるかを暗黙のうちに理解しています。

## デフォルトメッセージリーダ、HTTP リーダ

iPlanet Trustbase Transaction Manager には、MIME タイプ `application/x-www-form-url-encoded` を処理する HTTP/POST メッセージリーダが組み込まれています。HTTP/POST メッセージリーダはすでにデフォルトのメッセージリーダとして登録されているため、ホスト環境アダプタによって検知されたメッセージタイプに一致するものがメッセージリーダレジストリ内にない場合は、このリーダが返されます。MIME タイプが `application/x-www-form-url-encoded` でない場合は、エラーが返されます。

適切なメッセージリーダが見つかると、iPlanet Trustbase Transaction Manager メッセージオブジェクトおよび未処理のメッセージ内容を含む入力ストリームがそのメッセージリーダに渡されます。メッセージリーダは入力ストリームからメッセージ内容を読み取って解釈し、メッセージ内容の内部表現を含むデータブロックを作成します。このデータブロックは、適切なサービスが使用できるように、iPlanet Trustbase Transaction Manager メッセージオブジェクト内に格納されます。

メッセージリーダが入力ストリームの解釈および iPlanet Trustbase Transaction Manager メッセージの構築を完了すると、メッセージアナライザがその iPlanet Trustbase Transaction Manager メッセージをルータに送り、その後の iPlanet Trustbase Transaction Manager メッセージ処理で発生したエラーや例外に対処します。

## デフォルト HTTP リーダの使用

HTTP リーダは、HTTP Post または Get 文字列から各入力フィールドを抽出して DOM 構造に入れるという、比較的単純な方法で入力ストリームから情報を抽出します。すべてのフィールドが抽出されると、この DOM オブジェクトは iPlanet Trustbase Transaction Manager メッセージ内に格納されます。DOM オブジェクトの構造がユーザによって指定されていない場合、各フィールドはノード HTTP_VALUES の属性として格納されます。

```
HTTP
HTTP_VALUES Attributes [< フィールド名 > = < 値 >]*
```

ただし、特定のエスケープシーケンスを使用することで、フィールドをより複雑な構造に格納するよう強制できます。

HTTPReader クラスは入力フィールド名 <input name="$|xml:<ノード1>...<ノードn>|$"> に含まれるエスケープシーケンスを認識するため、開発者は独自の DOM 構造を作成できます。次に例を示します。

```
<input name="$|xml:message.user.name|$">
<input name="$|xml:message.user.address|$">
<input name="$|xml:message.user?jobtitle|$">
```

これによって、ルート HTTP ノードの下に、次のデータを含む一連のノードが作成されます。

```
HTTP
HTTP_VALUES Attributes [< フィールド名 > = < 値 >]*
message
user Attribute jobtitle=< 値 >
name = < 値 >
address = < 値 >
```

# デフォルトメッセージリーダ、Identrus リーダ

このメッセージリーダは、「application/identrus-」で始まる MIME タイプおよびメッセージタイプ IdentrusConstants.IDENTRUS_MESSAGE を持つメッセージを処理します。

メッセージリーダのアクションは次のとおりです。

- 必須の DSIG 署名をチェックする
- メッセージ内の証明書チェーンをチェックし、チェーンのルート証明書がデータベース内に存在することを確認する。チェーンのルートが証明書ストア属性 IdentrusConstants.IR_CA を持っていることを確認する
- ルータでの認可段階に対する準備が整っている iPlanet Trustbase Transaction Manager メッセージのセキュリティコンテキストを設定する
- メッセージの請求ログエントリを記録する

その後、メッセージはルータに送信されます。

# デフォルトメッセージリーダ、Identrus エラーリーダ

このメッセージリーダは、MIME タイプ application/identrus-identrustransporterror およびメッセージタイプ IdentrusConstants.IDENTRUS_TRANSPORT_ERROR を持つメッセージを処理します。

メッセージリーダのアクションは次のとおりです。

- 受信する「Identrus トランスポートエラー」にはチェックすべき署名がないため、iPlanet Trustbase Transaction Manager メッセージのセキュリティコンテキストを設定しない。このリーダに送られるメッセージはコネクタだけを経由しているため、ルータにこれらのメッセージを送らない

# メッセージライタ

メッセージが 1 つ以上のサービスに配信され、その後ルータに返されると、そのメッセージはメッセージアナライザに返されます ( 例外またはエラーが発生しなかった場合 )。この時点で、メッセージアナライザは、返された iPlanet Trustbase Transaction Manager メッセージをユーザへの送信に適切な形にするにはどのメッセージライタを使用すべきかを判別しなくてはなりません。メッセージアナライザは、メッセージリーダを選択する場合と同様、メッセージの返されたタイプおよび応答 MIME タイプの値に基づいて、適切なメッセージライタを選択します。

メッセージライタは iPlanet Trustbase Transaction Manager メッセージに含まれる情報を抽出し、メッセージをクライアントに送信可能な形式に処理します。

iPlanet Trustbase Transaction Manager には、メッセージライタをレジストリに登録するための HTML 管理インターフェイスがあります。したがって、iPlanet Trustbase Transaction Manager に関連付けられている内部構成ストアに新しいメッセージライタのサブクラスを登録できます。または、tbase.properties ファイルに適切なエントリを追加することでも登録できます。その場合は、たとえば次のように新しいメッセージライタを [MessageWriter] セクションに追加します。

```
message.writer=< 名前 >:< クラスパス >
message.writer=
Script:uk.co.jcp.tbaseimpl.parse.message.http.ScriptWriter
```

< 名前 > は新しい MessageWriter インスタンスのユーザ定義名を示し、< クラスパス > は完全指定のクラスパスを示します。管理インターフェイスを通じて新しいメッセージライタを追加し、その後 properties ファイルを使用して iPlanet Trustbase Transaction Manager を再起動すると、変更がすべて失われるので注意してください。このような事態を避けるため、管理インターフェイスを使用する場合でも、properties ファイルにエントリを追加することをお勧めします。

## デフォルト HTML メッセージライタ

iPlanet Trustbase Transaction Manager には、MIME タイプ text/html を処理する HTML メッセージライタが組み込まれています。この HTML メッセージライタはすでにデフォルトのメッセージライタとして登録されているため、ホスト環境アダプタによって検知されたメッセージタイプおよび MIME タイプに一致するものがメッセージライタレジストリ内にない場合は、このライタが返されます。MIME タイプが text/HTML でない場合は、エラーが返されます。

このメッセージライタは、他の Dynamic HTML ライタに類似のパターンマッチングおよびテンプレートを使用して、メッセージに含まれる XML フィールドを HTML 形式に変換する能力を備えています。

デフォルトの HTML メッセージライタである ScriptWriter は、完全な XSL に頼らず単純な情報を迅速および能率的に表示するためのスクリプト言語を使用します。

ScriptWriter は、応答にテキスト ( 通常は HTML) が含まれると想定され、かつルータから返された情報が XML からテンプレートへそのまま値を代入できるような単純なものである場合に使用されるべきライタです。

ScriptWriter クラスは、返されたメッセージに含まれる DOM オブジェクトを取り、その中のフィールドをすべて抽出することで機能します。これらのフィールドは、返されたメッセージのタイプおよび形式に基づいて適切なテンプレートに代入されます。この代入は、一連のスクリプトタグによって行われます。開発者は、これらのスクリプトタグを使用して、メッセージの各部分をそれぞれテンプレートのどの場所に入れるかを指定できます。ScriptWriter が使用するテンプレートを構成するには、iPlanet Trustbase Transaction Manager HTML 管理インターフェイスを使用するか、または次のように tbase.properties ファイルの [ScriptWriter] セクションに適切なエントリを追加します。

```
writer.typeandformat=< 形式 >:< タイプ >:< テンプレート >
writer.typeandformat=text/html:TimeServiceTimeResponse:Config\Templates\Time
Service\TimeServiceTimeResponse.html
```

< 形式 > は text/html です。< タイプ > は、ライタに返される前に最後にそのメッセージを処理したサービスが設定するメッセージタイプを示します。< テンプレート > は、テンプレートファイル名を示します。このファイル名には絶対パスまたは相対パスを使用できますが、相対パスの場合は 'root':template.directory=.\ ( 最後の「\」は必須 ) の場所を指定するためのエントリが必要です。

# スクリプトタグ

スクリプトテンプレートでは、主に次の 4 種類のタグが使用されます。

- $$xml:<XML 変数名 >$$

これは、1 つの変数が XML 文書からテンプレートに 1 度代入されることを表すタグです。<XML 変数名 > は、ルータから返された XML 文書に含まれるノードの完全指定の「場所」であり、テンプレートに代入可能な文字列値が含まれていることが想定されています。たとえば、XML 文書から 名前 (name) を取得するタグは、次のようになります。

```
$$xml:message.content.user.name$$
```

また、このタグでは、エスケープシーケンス「?」を使用して XML ノードの属性を指定することも可能です。たとえば、名前 (name) の属性として格納されているミドルイニシャル (middle_initials) を取得するタグは、次のようになります。

```
$$xml:message.content.user.name?middle_initials$$
```

- $$repeat:<XML 配列名 >[< ユーザ定義のタグ名 >:< 反復子 >]$$...$$/repeat$$

これは、XML 文書に含まれる、値の繰り返し配列の始点を表すタグです。

<XML 配列名 > は、XML 文書内の配列の完全指定の「場所」です。

< ユーザ定義のタグ名 > によって、この配列の場所に省略名を付けられます。たとえば、配列がユーザ (user) の住所 Response.Content.User.Address を表す場合、繰り返しのブロック内で簡単に使用できるように、これを住所 (address) としてタグできます。

< 反復子 > によって、配列のどの「部分」を表示するかを指定できます。

**表 2–1 XML Repeat 反復子**

| 反復子 | 説明 | 例 |
|---|---|---|
| * | 指定の配列内に含まれるすべてのアイテムを反復 | * |
| <n> | 配列内の n 番目の要素を選択。配列内に含まれる要素が n 個未満の場合は、何も表示しない | 2 |
| <m>-<n> | m 番目から n 番目までの要素 (m および n を含む) を選択。配列に含まれる要素が n 個未満の場合は、m から最後までを選択。要素が m 個未満の場合は何も表示しない | 3-5 |
| <m>-* | 配列の m 番目から最後の要素までをすべて選択 | 2-* |
| < 属性 >=< 値 > | 指定の属性が指定の値に等しい配列内の要素を「すべて」選択 | location=London |
| , | 複数の反復子を同時に指定することを許可 | 0,3-5,9,postcode=EC1 |

繰り返しの各ブロックは $$/repeat$$ 文によって中断されます。

- $$xif:<XML 変数名 >[="< 値 >"]$$ ... $$/xif$$

これは、条件を表すタグです。最も単純な形で記述された場合、このタグは、XML 内にノード ( または同じ「?」エスケープシーケンス使用の属性 ) が存在するときに条件文に含まれるすべてを反復するように指示します。

ただし、「=」が使用されている場合は、ノードまたは属性が存在するだけではなく、その値が指定されている値と正確に一致している必要があります (大文字と小文字の区別を含む)。たとえば、職名 (jobtitle) が「SysAdmin」に一致する場合だけユーザ詳細情報を表示するよう指示する条件ブロックは、次のようになります。

```
$$xif:message.content.user?jobtitle="SysAdmin"$$
```

$$xif:_$$ タグに関連するタグとして、任意の条件分岐を行うための $$xelse:_$$ があります。このタグは、2 とおりの方法で使用できます。

```
$$xelse$$
```

最も単純な使用法です。これは、$$xif:_$$ 条件が満たされていない場合に、代わりに適用する条件を表すタグです。XML ノードの内容によって、2 種類の表示形式を使用する必要がある場合などに使用します。

```
$$xelse:<XML 変数名 >[="< 値 >"]$$
```

このタグの記述の仕方は xif 文と同じです。これは、このブランチを実行する条件を表すタグです。テンプレートのどのセクションを使用するかをより詳細に指定するために、追加の条件分岐を行う場合に使用します。

- $$xprop:[< セクション >]< プロパティ >[=< デフォルト値 >]$$

これは、プロパティの値を tbase.properties ファイルからテンプレートに取り入れるための特殊なタグです。このタグによって、同じテンプレートが異なるアプリケーションサーバ環境をサポートできるようになります。プロパティ値は各サーバのタイプによって変化します。また、$$xprop:...$$ タグは、他のタグに先立って前処理されるため、プロパティに他のタグの情報 ($$xprop:...$$ そのものを除く ) を含むことができるという特徴を持っています。

- < セクション > は、プロパティを含むセクションの名前です。指定されていない場合は、ScriptWriter セクションが使用されます。
- < プロパティ > は、使用されるプロパティの名前です。
- < デフォルト値 > は、プロパティが見つからない場合に使用されるデフォルト値です。プロパティが見つからず、かつデフォルト値が指定されていない場合は、空白文字列が返されます。

# ScriptWriter タグの使用

- `$$xml:...$$` substitution

前述のとおり、このタグは返された XML 文書に含まれる 1 つの値がそのまま HTML テンプレートに代入されることを表します。ルータから返される XML 文書にサポートデスクのメールアドレス (supportaddress) を示すデータが含まれるという前提のもとに記述されたテンプレートの一部を次に示します。

```
<P>
<FONT SIZE=-2>
お問い合わせは、
<A HREF="mailto:$$xml:response.content.supportaddress$$">サポートデスク
</A>
までお送りください。
</FONT>
</P>
```

- `$$xml:...$$` filtering

次のように、絶対パスに属性フィルタを挿入することも可能です。

```
<P>
ログインの詳細については、
<A HREF="mailto:$$xml:response.content.user?title="SysAdmin".email$$">
システム管理者 </A>
まで電子メールでお問い合わせください。
</P>
```

この例では、XML 文書に 1 人以上のユーザ (user) の詳細情報の配列が含まれ、各ユーザにそれぞれの電子メールアドレス (email) を含むノードが含まれています。ユーザの title 属性を使用して職名 (title) が「SysAdmin」であるユーザを判別し、その結果から電子メールアドレス (email) を抽出します。

- `$$repeat:...$$` loop

`$$/repeat$$` を含む繰り返しのブロックを実際にどのように使用できるかを、次に示します。この例では、XML 文書がノード http_response.content.transaction の下にトランザクション (transaction) の配列を含んでいると想定されています。さらに、XML 文書ツリーのこのセクションは、txn としてタグ付けされています。

繰り返しのブロック内では、通常の `$$xml:...$$` 表記によって、返された配列内の各要素に含まれるデータが選択されています。

各トランザクションは、日付 (date)、説明 (description)、および金額 (amount) の 3 つのノードを含みます。これらのノードは 1 つのテーブル行に挿入され、テーブルには返されたトランザクションと同じ数の行が含まれます。

```
<table BORDER="1" BGCOLOR="#fff0a0">
     <th COLSPAN="3" BGCOLOR="ffd080"><b><i><font SIZE="+1"
COLOR="#00a000">明細</font></i></b></th>
     <tr>
          <td WIDTH="50" ALIGN="center"><B>日付</B></td>
          <td WIDTH="400" ><B>説明</B></td>
          <td WIDTH="200" ALIGN="center"><B>金額</B></td>
     </tr>
     $$repeat:message.content.transaction[txn:*]$$
          <tr>
          <td>$$xml:[txn].date$$</td>
          <td>$$xml:[txn].description$$</td>
          <td ALIGN="right">$$xml:[txn].amount$$</td>
          </tr>
     $$/repeat$$
</table>
```

- `$$repeat: ...$$` ブロックの入れ子

繰り返しのブロックは、入れ子にすることも可能です。前述のユーザの例に当てはめてみましょう。各ユーザ (user) のノードが住所 (address) のノード ( 住所 1 行 あたり 1 つずつ ) の配列を含んでいるため、各ユーザに対してすべての住所行を反復したいと仮定します。この場合、テンプレートの記述はたとえば次のようになります。

```
<table BORDER="1" BGCOLOR="#fff0a0">
     <th COLSPAN="3" BGCOLOR="ffd080"><b><i><font SIZE="+1"
COLOR="#00a000"> システムユーザ </font></i></b></th>
     <tr>
          <td WIDTH="100" ALIGN="center"><B> 名前 </B></td>
          <td WIDTH="400" ALIGN="center"><B> 住所 </B></td>
          <td WIDTH="100" ALIGN="center"><B> 電話番号 </B></td>
     </tr>
     $$repeat:message.content.user[user:*]$$
          <tr>
          <td>$$xml:[user].name$$</td>
          <td>
          $$repeat:[user].address[addr:*]$$
               $$xml:[addr]$$<br>
          $$/repeat$$
          </td>
          <td>$$xml:[user].telno$$</td>
          </tr>
     $$/repeat$$
</table>
```

- $$xif:…$$

ここでは、繰り返しのブロックの例をさらに発展させて、条件付き表示を実行する方法を示します。この場合、電話番号の列には、在宅勤務の従業員の自宅番号 (homePhone) またはオフィス勤務の従業員の内線番号 (workExtn) が表示されます。

```
<table BORDER="1" BGCOLOR="#fff0a0">
     <th COLSPAN="3" BGCOLOR="ffd080"><b><i><font SIZE="+1"
COLOR="#00a000"> システムユーザ </font></i></b></th>
     <tr>
          <td WIDTH="100" ALIGN="center"><B> 名前 </B></td>
          <td WIDTH="400" ALIGN="center"><B> 住所 </B></td>
          <td WIDTH="100" ALIGN="center"><B> 電話番号 </B></td>
     </tr>
     $$repeat:http_response.content.user[user:*]$$
          <tr>
          <td>$$xml:[user].name$$</td>
          <td>
          $$repeat:[user].address[addr:*]$$
               $$xml:[addr]$$<br>
          $$/repeat$$
          </td>
          <td>
               $$xif:[user].workType=" 在宅勤務 "$$
                    $$xml:[user].homePhone$$
               $$xelse$$
                    $$xml:[user].workExtn$$
               $$/xif$$
          </td>
          </tr>
     $$/repeat$$
</table>
```

- `$$else:…$$`

さらに、xelse 構造を使用して複数の条件文を実行することも可能です。前述の例に当てはめてみると、条件分岐を拡張して、営業担当の従業員を含むことができます。

```
<td>
     $$xif:[user].workType=" 在宅勤務 "$$
                    $$xml:[user].homePhone$$
     $$xelse:[user].workType=" 営業 "$$
                    $$xml:[user].mobile$$
     $$xelse$$
                    $$xml:[user].workExtn$$
     $$/xif$$
</td>
```

- `$$xprop:...$$` プロパティ代入

tbase.properties ファイルからテンプレートにプロパティを取り込むためのコードの例を次に示します。

```
<hr>
<form METHOD=post
ACTION="$$xprop:[ApplicationServcer]form.string$$/appservlet">
     <h3>
ID 確認のため、ご使用のカードに関する情報を入力してください。
</h3>
```

# デフォルト Identrus メッセージライタ

このメッセージライタは、「application/identrus-」で始まる MIME タイプおよびメッセージタイプ IdentrusConstants.IDENTRUS_MESSAGE を持つメッセージを処理します。

メッセージライタのアクションは次のとおりです。

- IdentrusConstants.SIGNING_CERT_ATTR が特定する iPlanet Trustbase Transaction Manager メッセージ属性が存在しない場合は、IdentrusConstants.L1_IP_SC 証明書を使用してメッセージに署名する。属性が存在する場合は、その値を送信メッセージの署名に使用される証明書または鍵の目的 ID として取る
- DOCTYPE に基づいて送信メッセージの DTD を確認し、不正なメッセージが TC の外部に出されないようにする
- DOCTYPE タグを設定し、DOCTYPE からメッセージの MIME タイプを判別する
- メッセージから証明書バンドルを抽出し、証明書ログに記録する
- 証明書バンドルを除く送信メッセージの XML をメッセージログに記録する
- メッセージをクライアントに送信可能な形式にする

# デフォルト Identrus エラーライタ

このメッセージライタは、MIME タイプ application/identrus-identrustransporterror およびメッセージタイプ IdentrusConstants.IDENTRUS_TRANSPORT_ERROR を持つメッセージを処理します。

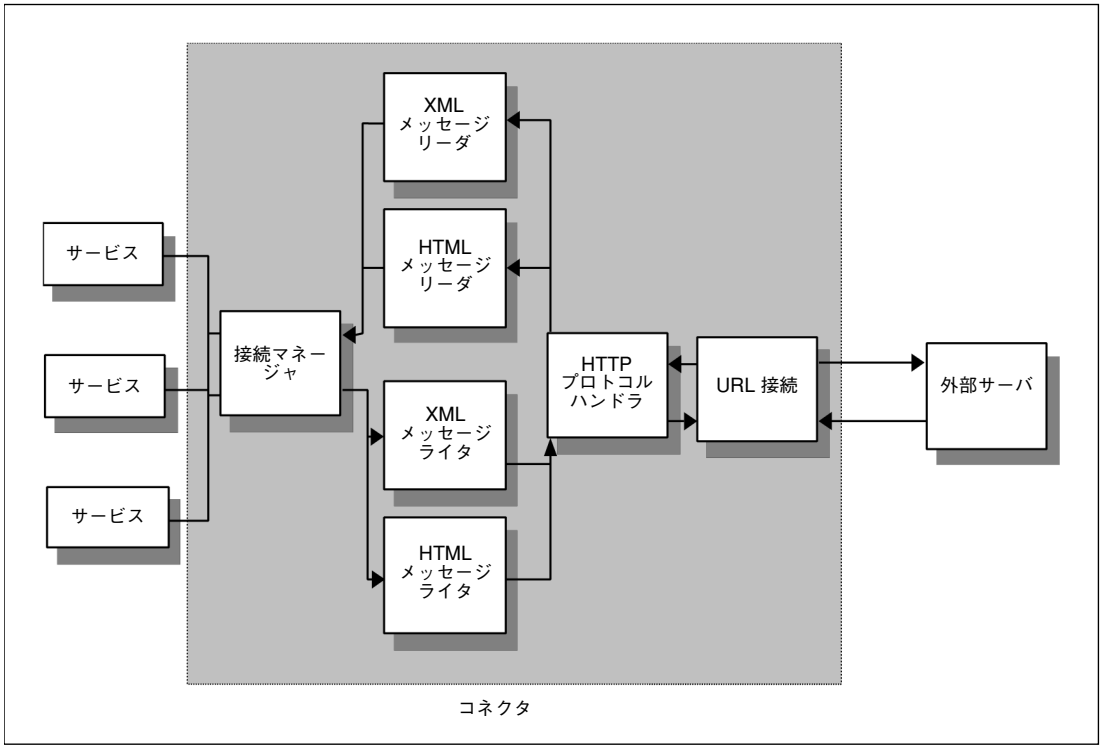
メッセージライタのアクションは次のとおりです。

- DOCTYPE に基づいて送信メッセージの DTD を確認し、不正なメッセージが TC の外部に出されないようにする
- DOCTYPE タグを設定し、DOCTYPE からメッセージの MIME タイプを判別する
- 送信メッセージの XML を原初ログに記録する
- メッセージをクライアントに送信可能な形式にする

# 接続マネージャ

iPlanet Trustbase Transaction Manager は、メッセージを受信すると、それを適切に処理できるサービスに配信し、必要に応じて応答メッセージを送ります。しかし、iPlanet Trustbase Transaction Manager システムが受信したメッセージを適切なサービスに配信しても、そのサービスが 他のソースから追加情報を取得しないとメッセージの処理を完了できない場合もあります。したがって、iPlanet Trustbase Transaction Manager は、特定のメッセージトランスポートプロトコルを使用して新しいメッセージを生成し、そのメッセージを特定の外部目的地に送るための機能を備えている必要があります。この機能を提供するのが「接続マネージャ」であり、サービスは接続マネージャを呼び出すことでメッセージを処理できるようになります。

図 2–2 接続マネージャのアーキテクチャ



注　詳細は、次の各 API を参照してください。

- uk.co.jcp.tbase.connector
- uk.co.jcp.tbase.xurl
- uk.co.jcp.tbaseimpl.connector

接続マネージャは、基本的に、外部エンティティとのメッセージの送受信に使用可能なサービスです。接続マネージャは、iPlanet Trustbase Transaction Manager メッセージおよび Destination オブジェクトを受け取り、これらの情報に基づいて、関連するメッセージの送受信を行います。接続マネージャのメッセージ処理の流れは次のとおりです。

- サービスが Connector を呼び出し、外部リソースへのリクエストを含む iPlanet Trustbase Transaction Manager メッセージおよび外部リソースの説明である Destination オブジェクトを渡します。Destination オブジェクトはアプリケーション定義の実装であり、Connector のアプリケーション定義のプラグインである ProtocolMap オブジェクトによって認識されます。無効なメッセージパラメータが渡された場合は、接続が確立されても URLConnection オブジェクトへのデータの書き込みは行われません。
- Connector は、受け取った Destination オブジェクトを ProtocolDescriptor オブジェクトに変換するため、ProtocolMapManager を呼び出します。(ProtocolDescriptor オブジェクトは、接続のための URL およびリクエスト内容のための MIME タイプを指定します。) ProtocolMapManager は、( 管理者によって ) 登録されている ProtocolMap の中から、受け取った Destination 実装の処理を担当しているものを選び出し、その ProtocolMap に Destination オブジェクトを渡します。
- ProtocolMap が Destination オブジェクトを ProtocolDescriptor に変換します。このアクションは、単純でローカル処理が可能な場合もあり、複雑でさらに外部へのリクエストが必要な場合もあります。
- Connector が ProtocolDescriptor 内の URL を使用して外部リソースに接続し、URLConnection を形成します。
- Connector にメッセージが送られた場合、Connector は URLConnection の doOutput 変数を true に設定することで、接続にデータを書き込むことを明示します。Connector は次に、記述されるべきメッセージのタイプ ( メッセージ の Type 属性が指定 ) および形式 (ProtocolDescriptor が指定 ) に従って、MessageWriter のレジストリから MessageWriter を選択します。その MessageWriter が呼び出され、メッセージを適切な形式に変換して URLConnection の OutputStream に書き込みます。Connector にメッセージが送られていない場合は、URLConnection の doOutputfield が false に設定されるので、URLConnection には何も書き込まれません。
- Connector が URLConnection の InputStream および応答の MIME タイプ (URLConnection に基づいて判別 ) を ConnectionProtocolAnalyser に渡します。ConnectionProtocolAnalyser は、応答の MIME タイプに基づいて、どの ProtocolHandler を呼び出すかを判別します。

- ConnectionProtocolAnalyser が、選択した ProtocolHandler に InputStream および新しい iPlanet Trustbase Transaction Manager メッセージを与えます。ProtocolHandler は、InputStream から応答のメッセージタイプを判別し、新しいメッセージのタイプをそれと同じものに設定します ( 同時にコンテキスト識別子も設定することがあるが、ここでは無視される )。
- Connector が 応答メッセージタイプおよび応答 InputStream MIME タイプに基づいて MessageReader を選択します。MessageReader が呼び出され、応答の解釈を完了します。MessageReader の作業が完了すると、Connector が外部との接続確立に使用されたリソースをすべて開放し、解釈済みのメッセージをリクエスト元であるサービスに返します。

---

**注** iPlanet Trustbase Transaction Manager は、uk.co.jcp.tbaseimpl.connector.DefaultConnector で定義されているコネクタインターフェイスを実装するデフォルトコネクタを提供しています。これらのクラスの大部分は、iPlanet Trustbase Transaction Manager の一部として提供されていますが、そうでないものもあります。たとえば、java.net.URLConnection はコア Java API の一部です。

---

# プロトコルマップマネージャ

接続マネージャは、iPlanet Trustbase Transaction Manager フレームワーク内のサービスが他の外部エンティティと通信することを可能にします。通信の際、Service は Connector ( 接続マネージャ ) に メッセージおよび Destination を渡します。Connector は指定の外部エントリにメッセージを配信し、通信先のアドレスを見つけるために ProtocolMapManager に Destination を渡します。ProtocolMapManager は登録されている ProtocolMap の中からその Destination に関連付けられているものを選択し、その ProtocolMap に Destination を渡します。ProtocolMap は Destination に関連付けられている ProtocolDescriptor オブジェクトを返します。受け取った Destination に対してどの ProtocolDescriptor を返すかは、各 ProtocolMap が判別します。

新しいアプリケーション固有プロトコルマッピングシステムを作成するには、次のステップを実行する必要があります。

- 特定の ProtocolMap の作成
- 特定の Destination の作成
- 新しい ProtocolMap の ProtocolMapManager への登録
- ProtocolMap へのデータ提供

---

**注** フレームワーク uk.co.jcp.tbase.connector を参照してください。また、デフォルトの接続マネージャクラス実装については、uk.co.jcp.tbaseimpl.connector を参照してください。

---

各ステップの詳細は次のとおりです。

- 特定の ProtocolMap の作成

ProtocolMap オブジェクトはすべて ProtocolMap インターフェイスを実装する必要があります。また、空白またはデフォルトのコンストラクタを提供する必要もあります。これは、ProtocolMapManager がクラス登録のために ProtocolMap オブジェクトを動的にインスタンス化するからです。

ProtocolMap は次のメソッドを呼び出します。

```
public Enumeration getDestinationTypes()
```

これは、ProtocolMap が認識できる Destination 実装のパッケージ指定クラス名のリストを返すメソッドです。

```
public ProtocolDescriptor getProtocolDescriptor(Destination
destination)
     throws InvalidDestinationException
```

これは、アプリケーションの指定された Destination オブジェクトを ProtocolDescriptor に変換し、接続に必要な URL および MIME タイプを指定するメソッドです。

---

**注** アプリケーション固有の ProtocolMap は、シリアル化できないオブジェクトを含むことはできません。これは、ProtocolMapManager に永続性があるためです。詳細は、API uk.co.jcp.tbase.connector.ProtocolMap を参照してください。

---

- 特定の Destination の作成

Destination はすべて Destination インターフェイスを実装する必要があります。このインターフェイスはメソッドを呼び出しません。すべてのメソッドおよびデータはアプリケーション固有であり、関連付けられている ProtocolMap が前もってインターフェイスの形式を認識している必要があります。

---

**注** アプリケーション固有の Destination がシリアル化できないオブジェクトを含むことはできません。これは、ProtocolMapManager に永続性があるためです。詳細は、API uk.co.jcp.tbase.connector.Destination を参照してください。

---

- 新しい ProtocolMap の ProtocolMapManager への登録

アプリケーション固有の ProtocolMap および Destination を構築したら、iPlanet Trustbase Transaction Manager 内のサービスが ProtocolMap の機能を活用できるように、ProtocolMap を ProtocolMapManager に登録する必要があります。

---

**注** ProtocolMapManager に登録されている既存 ProtocolMap が使用する特定の Destination への参照がアプリケーション固有の新しい ProtocolMap に含まれている場合、新しい ProtocolMap は ProtocolMapManager に登録されません。

---

プロトコルマップは、「ProtocolMapManager」というプロパティセクションを含む tbase.properties ファイル内で登録できます。このセクションにアプリケーション固有の ProtocolMap オブジェクトのエントリを記述します。次に例を示します。

```
[ProtocolMapManager]
protocol.map=uk.co.jcp.tbase.connector.SimpleProtocolMap
```

---

**注** iPlanet Trustbase Transaction Manager のインスタンスに関連付けられている構成オブジェクトを削除しない限り、この `tbase.properties` ファイルが読み込まれるのは 1 度だけです。また、完全なパッケージ名を指定する必要があります。

---

# URL 接続の実装

接続マネージャ内部では、ProtocolMapManager が受け取った Destination に基づいて ProtocolDescriptor を返すと、ProtocolDescriptor から URL の文字列表現が取得されます。この URL の文字列表現からは URLConnection オブジェクトが取得されます。接続マネージャ (Connector) は、URLConnection からの出力ストリームおよび入力ストリームを通じてメッセージを送受信します。URL の文字列表現から URLConnection オブジェクトへの変換は、「XURLxxxxx」の形式を持つクラスによって行われます。使用されるクラスは次のとおりです。

- XURL
- XURLStreamHandler
- XURLStreamHandlerFactory

これらのクラスが必要なのは、javasoft によって提供されている既存の URL クラス階層は EJB 形式では拡張できないからです。既存のフレームワークでは、URL クラスは、URL 形式を認識できないと XURLStreamHandlerFactory が読み込まれているかどうかを確認しますが、この際に XURLStreamHandlerFactory が不明な URL 形式を提供することがあります。このファクトリを設定できるのは 1 度だけなので、EJB 環境では URLStreamHandlerFactory がインスタンス化されているかどうかを確認できず、その結果フレームワークを使用できないということになります。この問題を解決するため、どの環境でも動作可能な拡張フレームワークが作られました。これらのオブジェクトは、XURL、XURLStreamHandler、および XURLStreamHandlerFactory と呼ばれます。

プロセスの流れは次のとおりです。

- ProtocolDescriptor が提供する URL の文字列表現に基づいて、XURL オブジェクトが構築されます。
- URLConnection オブジェクト取得のため、XURL のメソッド openConnection() が呼び出されます。
- このメソッド内で XURLStreamHandlerFactory が呼び出されます。
- 初めて呼び出された場合、XURLStreamHandlerFactory は tbase.properties ファイルの「XURLStreamHandlerFactory」セクションに各 XURLStreamHandler クラスを登録し、それぞれを値「url.stream.protocol」に対応づけて、自己初期化を行います。
- 指定された URL への接続に使用されるプロトコルをキーとして、XURLStreamHandlerFactory 内の検索が行われます。
- このプロトコルが XURLStreamHandlerFactory に登録されている XURLStreamHandler に対応づけられている場合は、XURLStreamHandler オブジェクトが呼び出され、必要な URLConnection オブジェクトを提供します。
- プロトコルが XURLStreamHandlerFactory 内のどの XURLStreamHandlers とも一致しない場合は、XURL の文字列表現から URL オブジェクトが作成され、そこから直接 URLConnection オブジェクトが取得されます。

この拡張フレームワークによって、JCP は既存のプロトコル形式を無効にし、特定の拡張機能 ( 例 : HTTPS。SSL が証明書ストアインターフェイスに直接結び付けられている ) を組み込むことを可能にしています。

---

**注**　　詳細は、API uk.co.jcp.tbase.xurl を参照してください。

---

第 3 章

# ルーティング

ルータコンポーネントには、2 つの特定の機能があります。

- 特定のサービスへのリクエストを認可する手段を提供する
- トランザクションの流れを制御する手段を提供する

ルータのアーキテクチャは規則ベースになっています。通常の場合、トランザクションの動きを変更するために開発者が Java コードを記述する必要はありません。これは、システム内でのメッセージ処理の流れを判別する XML 規則を作成し、これらの規則と iPlanet Trustbase Transaction Manager メッセージ内の値を比較するメカニズムが存在するためです。

この章では、ルータのアーキテクチャの基盤となっている概念、ルータの機能、および Identrus メッセージ処理のデフォルト実装について説明します。

# メッセージ

iPlanet Trustbase Transaction Manager メッセージは、iPlanet Trustbase Transaction Manager のある部分から別の部分へと移動するエンティティであり、iPlanet Trustbase Transaction Manager が受信したメッセージおよびそのメッセージに関する内部データをカプセル化したものです。iPlanet Trustbase Transaction Manager メッセージを構成する基本要素は次のとおりです。

- メッセージタイプ
- 属性
- データの内容

| | |
|---|---|
| 注 | iPlanet Trustbase Transaction Manager メッセージの要素に関する詳細は、uk.co.jcp.tbase.environment.Message の API ドキュメントを参照してください。 |

フレームワークそのものが iPlanet Trustbase Transaction Manager メッセージをどう処理するかは、メッセージのタイプおよび属性によって決定されます。iPlanet Trustbase Transaction Manager フレームワークでは、データの内容は不透明なものとして扱われます。つまり、データの内容が変更されても、iPlanet Trustbase Transaction Manager フレームワークはその変更を認識しません。

メッセージの状態を判別するのはルータの役目です。ルータは、属性値に基づく一連の前提条件を使用してメッセージの状態を判別し、その状態に従ってどのアクションを適用すべきかを判別します。これらの前提条件およびアクションは、ルータ規則として格納されています ( 詳細は後述 )。

## メッセージの属性

メッセージの属性は、Java プロパティと同様、名前と値が対になったものです。属性の名前と値は、必ず文字列によって表されます。

メッセージ属性の読み取りおよび書き込みは、プロトコルハンドラ、メッセージリーダ、メッセージライタ、ルータ、またはサービスによって行われます。したがって、メッセージ処理パイプラインの各段階で、その時点でのメッセージの状態に関する情報が追加されます。

属性は任意のユーザアプリケーションコードによって定義できますが、iPlanet Trustbase Transaction Manager フレームワークでは、次のような 標準的な属性が使用されています。

- messageType
  Message API クラスが処理する属性。メッセージの MIME タイプとともに、メッセージ処理に使用するメッセージリーダ、メッセージライタ、およびプロトコルハンドラの判別に使用される。MIME タイプがこの属性を適切に設定するように管理するのは、プロトコルハンドラの役目。また、messageType 属性は、メッセージ処理中に変化することがある
- security.role
  メッセージが帰するセキュリティ役割を含む属性。認証サービス (Authenticator) によって、認証に成功した場合は役割名に、役割を割り当てられない場合はデフォルト値に設定される
- security.cert
  認証の証拠として使用される証明書のエンコード。通常は SSL クライアント証明書だが、Identrus メッセージの場合は、必須であるレベル 1 署名の確認に使用される証明書
- security.cert_encoding
  必ず BASE64 で表される
- security.cert_present
  認証に使用できる有効な証明書が存在しているかどうか
- security.username
  認証に使用されるユーザ名
- security.password
  認証に使用されるパスワード
- security.user_pass_present
  有効なユーザ名 / パスワードの証拠がメッセージに存在しているかどうか
- security.auth_failed
  executeServiceDirective が呼び出された際に、認可に失敗したことを示す。iPlanet Trustbase Transaction Manager 内に役割 / サービス名の対応づけ ( マッピング ) が存在しない、つまりメッセージをリクエスト先のサービスに送信する認可が与えられていないことを意味する

---

**注**　セキュリティに固有の属性名は、uk.co.jcp.tbaseimpl.authenticator.SecurityContext 内で定数として宣言されます。また、uk.co.jcp.tbase.environment.attribute も参照してください。

---

# Identrus メッセージの属性

Identrus メッセージ処理の場合は、さらに次のメッセージ属性が使用されます。

- DocType
  メッセージのタイプを特定する XML メッセージの DOCTYPE (CSCRequest や PingRequest など )
- SigningCertPurposeId
  デフォルト Identrus メッセージライタだけが使用する省略可能な属性。指定されている場合、この属性は署名証明書および秘密鍵の抽出に使用する証明書ストア属性を指定する。この属性が指定されていない場合は、メッセージライタは Level 1 Inter-participant Signing Certificate を使用する
- XMLSystemId
  受信 XML メッセージの「システム識別子」属性を含む属性。対応する応答メッセージに XML システム識別子を埋め込むために Identrus メッセージライタが使用する

---

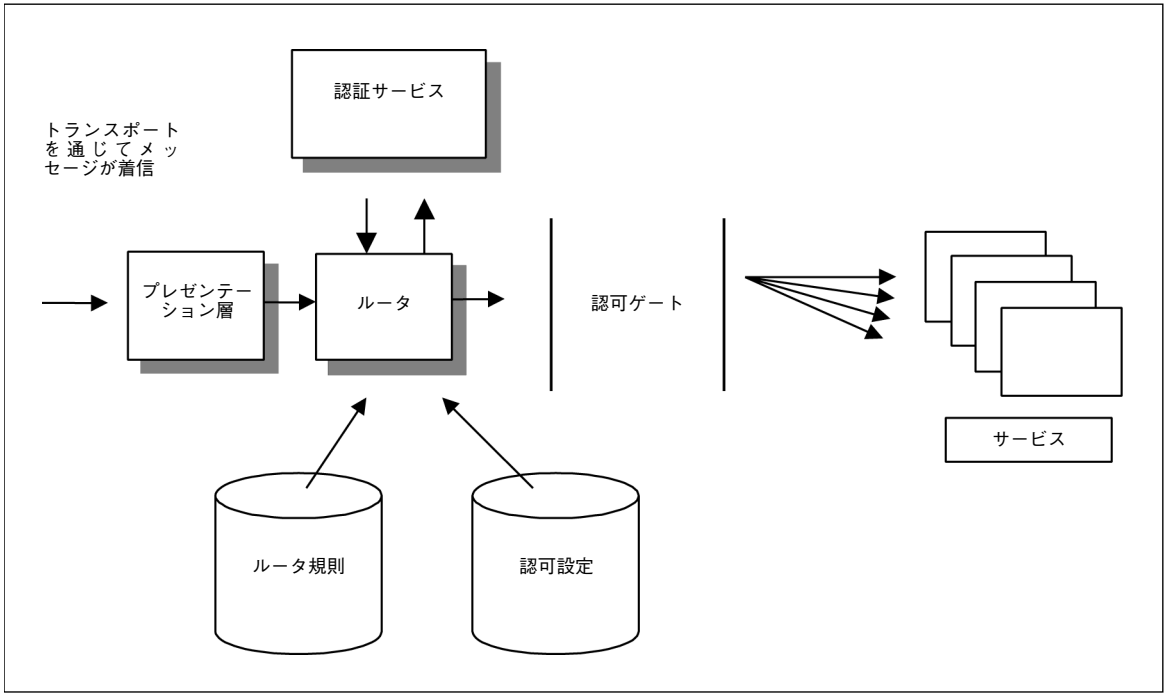
**注** Identrus に固有の属性名は、com.iplanet.trustbase.identrus.IdentrusConstants 内で定数として宣言されます。

---

# ルータのアーキテクチャ

ルータは、iPlanet Trustbase Transaction Manager の柔軟性の中核であるコア規則ベースのメッセージルーティングを行いますが、認証情報および認可設定に基づいてサービスへのアクセスを制御するゲートとしての役割も担っています。図 3-1 に、iPlanet Trustbase Transaction Manager 内でメッセージを移動し、適切なサービスに転送するための基幹要素を示します。

図 3–1 ルータのアーキテクチャ




- iPlanet Trustbase Transaction Manager に着信したメッセージは、プレゼンテーション層 ( プロトコルハンドラおよびメッセージリーダ ) を通過します。この間に、messageType の識別と設定、およびその他の標準的な属性の設定が行われます。これらの属性は、メッセージの認証証拠の基盤となります。
- その後、メッセージはルータに送られます。ルータは受け取ったすべてのメッセージをまず認証サービス (Authenticator) に送ります。iPlanet Trustbase Transaction Manager にはデフォルトの認証サービスが実装されていますが、必要に応じて変更することも可能です。

- デフォルト認証サービスは、メッセージの属性を確認することで、そのメッセージの認証レベルを判別します。認証サービスの機能に関する詳細は、下記の説明を参照してください。
- セキュリティコンテキスト属性 (security.role) を設定されたメッセージが、ルータに送り返されます。このコンテキスト属性によって、メッセージがどの役割に対して認証されたかが判別されます。
- ルータは、ルーティング規則に従ってメッセージを処理することでターゲットサービスを検出し、そのサービスにメッセージを送ります。ルーティング規則の詳細は、この章後述の説明を参照してください。
- メッセージは、サービスに到達する前に認可ゲートを通過する必要があります。この認可ゲートでは、メッセージに割り当てられた役割属性とターゲットサービスが対応づけられているか ( マッピングが存在するかどうか ) が確認されます。マッピングが存在する場合、メッセージはサービスに送られ、存在しない場合は認可エラーサービスに送られます。

iPlanet Trustbase Transaction Manager メッセージはすべて上記の各段階を通過します。また、Identrus XML メッセージ処理のためのパイプラインは、開発者の観点から合理化されています。詳細は、この章の「デフォルトルーティング」を参照してください。

# 認証と認可

iPlanet Trustbase Transaction Manager の認証メカニズムは独立したコンポーネントではありません。認証データはデフォルト iPlanet Trustbase Transaction Manager フレームワークによって収集されます。また、ドメイン固有のサービスによってさらにデータが追加されることもあります。これらのサービスが識別情報を特定の属性セットに対応づけ、iPlanet Trustbase Transaction Manager はこれらの属性を使用して他のサービスへのアクセスを制御します。

## 認証

可能な限りさまざまな組織に対応できる認証および認可メカニズムを維持するため、iPlanet Trustbase Transaction Manager フレームワークそのものは認証作業を行いません。識別情報の確認および特定の役割への対応付けは、サービスによって行われます。システムにはデジタル署名データアイテムおよびそれに付随する X509 デジタル証明書があり、サービスはこのデータを役割に対応づける機能を備えています。基本的な認証情報は、次の 2 種類の方法で提示されます。

- 暗黙的 - トランスポートプロトコル内 (SSL など ) で提示
- 明示的 - メッセージ内容の中 (Identrus XML メッセージの署名済みメッセージブロックなど ) で提示

ルータコンポーネントのアーキテクチャは、SSL ハンドシェイク内で使用されている証明書を削除し、認証サービスに送られるメッセージにこの証明書を付加することで、認証情報を明示化するように設計されています。このため、トランスポートレベルの認証に対して暗黙の「ブラックボックス」アプローチが取られる場合とは異なり、ターゲットシステムがさまざまなレベルの認証を実行できます。

また、デフォルトサービスは、ルータがリクエスト認可時に使用できるように、ユーザ名およびパスワードを役割に対応づける手段も提供しています。このメカニズムは、HTML テンプレートメカニズムを使用して構成情報および管理情報を提示する場合に一般的に使用されるものです。大規模な HTML ベースアプリケーションに対してプラットフォームを使用する場合は、排他的コミュニティメカニズムの代わりに、企業レベルのユーザ名およびパスワードメカニズムを使用することをお勧めします。

# 認可

デジタル証明書またはその他の方法でメッセージの送信元を判別したら、次の各操作を実行するためのメカニズムが必要となります。

- ユーザに対して適切な認可レベルを特定する
- 認可に基づいて各サービスにアクセスする

適切な認可レベル ( または役割 ) を特定するのは、iPlanet Trustbase Transaction Manager の認証サービス (Authenticator) の役目です。デフォルトとして実装されているこのサービスは、認可データベース内のユーザ名 / パスワードテーブルおよび証明書テーブルを使用して、メッセージとセキュリティ属性の対応づけ ( マッピング ) を判別します。マッピングの詳細は、『iPlanet Trustbase Transaction Manager 構成およびインストールガイド』を参照してください。各サービスへのアクセスは、ルータによって自動的に行われます。一致するルーティング規則が存在し、かつ本文に「executeServiceDirective」が含まれている場合、ルータはこのディレクティブを実行する前に、メッセージに割り当てられた役割とターゲットサービスの間にマッピングが存在するかどうかを確認します。これらのマッピングは認可テーブル内に維持されます。テーブルがどのように使用されるかについては、『iPlanet Trustbase Transaction Manager 構成およびインストールガイド』を参照してください。マッピングが存在する場合、メッセージはサービスに送られます。存在しない場合は、メッセージに新しい属性「security.auth_failed」が追加され、ルータに送り返されます。ルーティング規則は、認可失敗を必ず検知し、クライアントに「認可失敗」の応答を送るためのサービスを実行します。「unauthorisedExecuteServiceDirective」が含まれる場合は、認可チェックは行われず、無条件にサービスが実行されます。

# デフォルトルーティング

Identrus メッセージサービスの開発を簡易化するため、iPlanet Trustbase Transaction Manager は Identrus メッセージ用のデフォルトルーティングメカニズムを提供しています。このデフォルトルーティングは、開発および配置されるすべての Identrus サービスに適用されます。詳細は、第 8 章、「Identrus ソリューションの構築」を参照してください。

デフォルトルーティングは大部分の Identrus サービスに対応できるように設計されていますが、このルーティングでは不十分な場合は、開発者が新しいルーティング規則を記述する必要があります。

## ルータ規則

iPlanet Trustbase Transaction Manager バージョン 2.2 のルータはより高度な機能を搭載しているため、基本的な規則処理が非常に簡易化されています。以前のバージョンでは、各サービスがサービス間でのメッセージ移送を明示的に対応づける規則セットを定義している必要がありました。しかし、新バージョンではシステムが動的に規則を定義できるため、この基本的な動作は自動的に定義されます。下記の 2 つの項で、このメカニズムについて説明します。

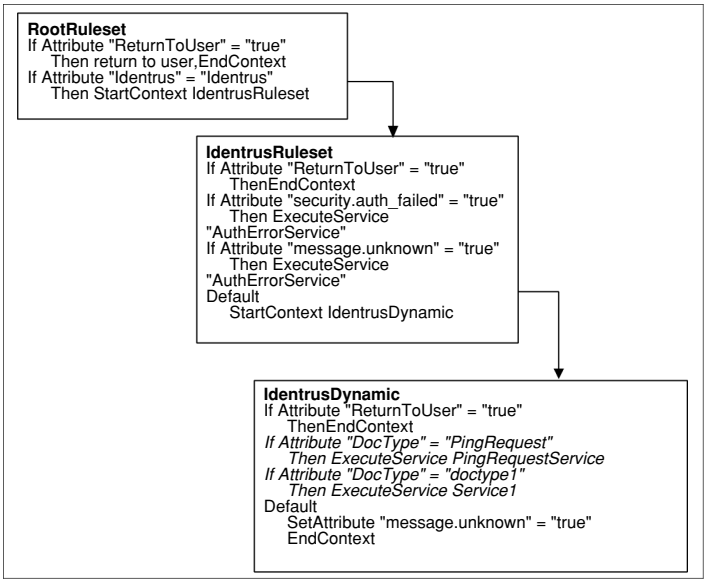
## サービスへのルーティング

サービスの名前は、サービス JAR ファイルに含まれる配置記述子によって特定されます。システム起動時に、規則ハンドラがどのサービスを読み込むかを判別し、該当する各サービスに対して、指定の doctype とともにメッセージを転送するためのルーティング規則を動的に設定します。サービスがユーザ定義の規則セットを持つ場合は、動的に作成された規則によって、サービスではなくこの規則セットにメッセージが送られます。ユーザ定義の規則セットはサービスの JAR ファイル内に含まれます。単純なメッセージ処理の場合はこの規則セットは不要です。

# 返送パス

ルーティングをできるだけ単純にするため、「ReturnToUser」属性が設定されているメッセージはユーザに送り返されます。この属性は、処理が完了した時点でサービスによって設定されます。サービスレベルで作成されるエラーメッセージの場合も同様です。

図　3–2　　デフォルトルーティング規則



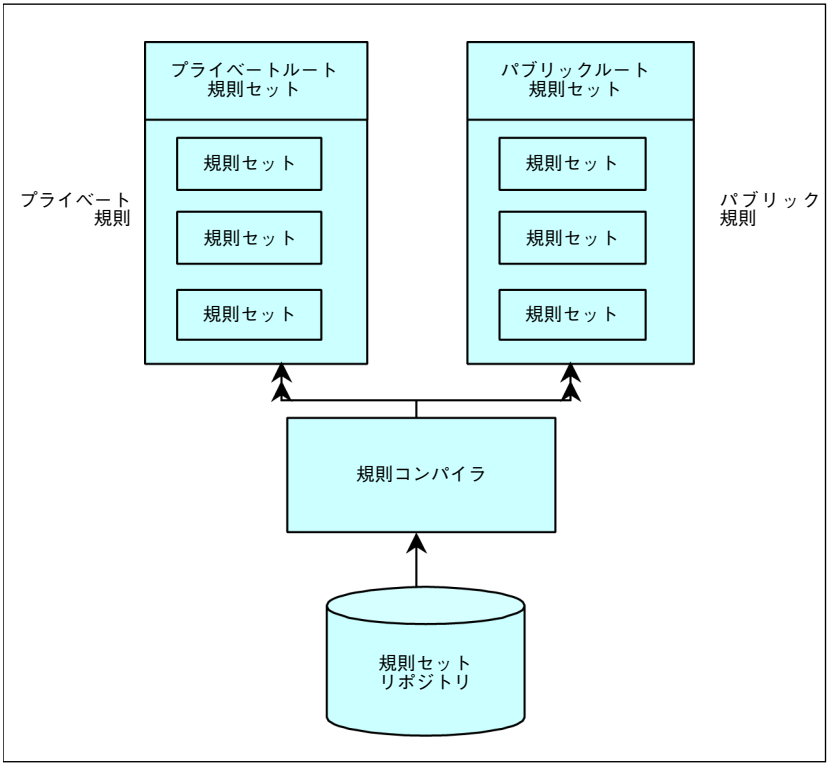
# 高度なルーティング

この節では、ルーティング規則について詳しく説明します。単純な Identrus サービスの場合、メッセージをサービスに送るために開発者がルーティング規則を記述する必要はありませんが、デフォルトルーティングが不適切な場合、または Identrus 以外のメッセージを処理する場合は、新しく規則を記述する必要があります。

## ルーティング規則セット

ルーティングテーブルには、プライベートおよびパブリックの 2 つの論理領域があります。パブリック規則セットは管理インターフェイスを通じて追加または構成できます。プライベート規則セットは基本的なサービスを提供するために iPlanet Trustbase Transaction Manager が使用するものであり、開発者またはシステム管理者が構成することはできません。この構造の最上部には、プライベート規則がどのように実行されるかを管理するルート規則セットがあります。パラメータに一致するプライベート規則が存在しない場合は、パブリックルート規則セットが実行されます。システム開発者は、このパブリックルート規則セットから独自のタスク規則セットをリンクできます ( 規則セットの定義については、次項を参照 )。

図 3–3 規則セット




各領域には複数の規則セットが含まれています。各パブリック規則セットには名前があり、通常の場合はその名前が規則セットが行うビジネスタスクを示しています。パブリック規則セットはプラットフォームが起動するたびにディスクから読み込まれます。ルート規則セット内の規則が実行された結果として、名前そのものが明示的に読み込まれることもあります。

メッセージは、ルータに送られた時点ですでにコンテキストに関連付けられています。各コンテキストは、それぞれ規則セットに関連付けられています。これは、次のことを意味しています。

- メッセージはルートコンテキストで動作し、ルート規則セットが関連付けられている
- メッセージはサブコンテキストで動作し、パブリック規則領域の規則セットが関連付けられている
- ただし、iPlanet Trustbase Transaction Manager メッセージが処理されている場合は例外であり、その場合はプライベート規則セットが関連付けられる

規則セットは順次処理され、現在の条件に当てはまる最初の規則が使用されます。ルータは規則セット内の各規則を最初から順に取り、メッセージタイプが規則前提条件に一致するかどうかを確認します。一致するものが見つかると、アクションが実行されます。一致するものがない場合は、エラー条件が返されます。規則は順次処理されるため、より明示的な規則を規則セットの始めの部分に、一般的な規則を後の部分に設定することが重要です。そうしないと、明示的な規則が一般的な規則によって実行対象外として排除されてしまう可能性があります。

ルート規則セットはできるだけ小さくしておくのが理想的です。そのためには、ルート規則セット内の規則がビジネスタスク内の最初のメッセージ ( 非請求メッセージ ) だけに一致するようにしておく必要があります。アクションは、必ず新しいコンテキストを開始するためのものである必要があります。また、メッセージに適用する規則セットの名前 ( および場合によっては開始位置となる規則の名前 ) を示していなくてはなりません。その後、メッセージが同様の方法で処理されます。

規則セットは、次の項で説明する DTD に準拠する、1 つの完全な XML 文書です。ルータは、起動時にまず iPlanet Trustbase Transaction Manager 初期化ファイル内の Router セクションを参照し、開発者によって記述された規則を格納しているディレクトリを探します。次に .xml ファイルを探し、これらのファイルをパブリック規則セット領域にコンパイルします。次の DTD の例に示されるように、各規則セットは、規則を含まない場合や、1 つあるいは複数の規則を含む場合があります。

```
<!ELEMENT ruleSet (rule*)>
<!ATTLIST ruleSet name CDATA #REQUIRED>
```

規則セット内の規則の構造については、次の項で説明します。

# ルータ規則の構文

この項では、各要素の目的を定義しながら規則セットの DTD について説明します。各規則には、次の 3 つの要素があります。

- 規則名を示す属性。省略可能。特定の規則を使用するかどうかを判別する必要がある場合は、新しいコンテキスト開始時にその名前を指定できる
- 一連の前提条件。本文のディレクティブが実行されるためには、これらの条件がすべて満たされている必要がある
- 規則の本文。前提条件がすべて満たされている場合に実行されるアクションのリスト

典型的な規則名は次のようになります。

```
<!ELEMENT rule ( preconditions, body )>
<!ATTLIST rule name CDATA #IMPLIED>
<!ELEMENT preconditions (attributeCondition*)>
```

前提条件は、属性条件 (attributeCondition) を含まない場合や、1 つあるいは複数の attributeCondition を含む場合があります。すべての attributeCondition が満たされていない場合、本文は実行されません。

```
<!ELEMENT attributeCondition EMPTY>
<!ATTLIST attributeCondition name CDATA #REQUIRED>
<!ATTLIST attributeCondition operator ( greaterThan |
greaterThanEqualTo | lessThan | lessThanEqualTo | notEqual |
startsWith | contains | endsWith | equals ) "equals">
<!ATTLIST attributeCondition valueType ( any | null | string ) "string">
<!ATTLIST attributeCondition value CDATA #IMPLIED>
```

iPlanet Trustbase Transaction Manager は、各 attributeCondtion を次のように解釈して処理します。

```
IF ( name operator value ) then pre-condition met
```

通常の場合、valueType は「string」に設定されます。value フィールドを持つ規則の場合は、valueType は不要です。属性の存在を確認するよう指定されている場合、または null に対してテストするよう指定されている場合は、valueType フィールドが使用されます。このような場合、value フィールドは不要であり、指定されていても無視されます。

```
<!ELEMENT body ((setAttribute | executeServiceDirective |
unauthorisedExecuteServiceDirective )*, (startContextDirective |
endContextDirective | returnToUserDirective )?)>
```

前提条件が満たされると、規則の本文が実行されます。規則の本文には、複数のアクションが含まれていることがあり、これらのアクションは記述順に実行されます。アクションの中には「中断」アクションとして認識されるものが 3 種類あり、これらのアクションは最終アクションとして規則の末尾部分に 1 度だけ記述されます。表　3-1 に、使用可能な本文ディレクティブを示します。

表 3–1 **DTD** の規則本文

| ディレクティブ | 説明 |
|---|---|
| SetAttribute | メッセージに iPlanet Trustbase Transaction Manager 属性を設定する |
| executeServiceDirective | 必須の認可チェックを実行してルータから指定のサービスに制御を移す |
| unauthorisedExecuteServiceDirective | 必須の認可チェックを実行せずにルータから指定のサービスに制御を移す |
| StartContextDirective | ある規則セットから別の規則セットにルータ制御を移す |
| EndContextDirective | 現在のコンテキストを終了し、規則の検索を再開する |
| ReturnToUserDirective | ルータからシステムユーザに制御を返す |

```
<!ELEMENT setAttribute EMPTY>
<!ATTLIST setAttribute name  CDATA #REQUIRED>
<!ATTLIST setAttribute value  CDATA #REQUIRED>
<!ATTLIST setAttribute location (MESSAGE | CONTEXT) "CONTEXT">
<!ATTLIST setAttribute type  (ASCEND_TRANSITIVE | NONE |
INHERIT_TRANSITIVE | INHERIT) "ASCEND_TRANSITIVE">
```

setAttribute アクションによって、ルータがメッセージまたはコンテキストに属性を追加することが可能になります。name、value、location、および type フィールドは、API を通じて属性を作成する場合の設定と同じです。

**注** 詳細は、uk.co.jcp.tbase.environment.attribute.Attribute の API ドキュメントを参照してください。

```
<!ELEMENT executeServiceDirective EMPTY>
<!ATTLIST executeServiceDirective name CDATA #REQUIRED>
```

executeServiceDirective には、起動するサービス名を指定する必要があります。この方法でサービスを実行すると、メッセージがリクエスト先のサービスに送られる前に、まず認可チェックが実行されます。

このサービス名は、tbase.properties ファイルの ServiceRegistry セクションでサービス名として指定されているものに関連しています。tbase.properties ファイル内の新しい service.description エントリは、そのサービスの名前およびクラスパスによって構成されます。

```
<!ELEMENT unauthorisedExecuteServiceDirective EMPTY>
<!ATTLIST unauthorisedExecuteServiceDirective name CDATA #REQUIRED>
```

unauthorisedExecuteServiceDirective には、起動するサービス名を指定する必要があります。この方法でサービスを実行すると、認可チェックは実行されずにメッセージがリクエスト先のサービスに送られます。

このサービス名は、tbase.properties ファイルの ServiceRegistry セクションでサービス名として指定されているものと関連しています。tbase.properties ファイル内の新しい service.description エントリは、そのサービスの名前およびクラスパスによって構成されます。

```
<!ELEMENT startContextDirective EMPTY>
<!ATTLIST startContextDirective ruleset CDATA #REQUIRED>
<!ATTLIST startContextDirective rule CDATA #IMPLIED>
```

ルータは、startContextDirective を実行する際に、検索していた現在の規則セットを破棄し、ディレクティブ内で指定されている新しい規則セットを読み込みます。ディレクティブ内で個々の規則が指定されている場合は、新しい規則セット内にその規則が存在するかどうかを確認し、それ以外の場合は、規則セットの最初からチェックを開始して前提条件が満たされている最初の規則を見つけようとします。

startContextDirective には、使用する規則セットの名前が含まれている必要があります。また、指定の規則セット内で実行される規則名が含まれることもあります。

```
<!ELEMENT endContextDirective EMPTY>
```

endContextDirective には、属性または子はありません。

```
<!ELEMENT returnToUserDirective EMPTY>
<!ATTLIST returnToUserDirective endContext ( true | false ) #REQUIRED>
```

ルータは、returnToUserDirective によって、現在のコンテキストを終了するかどうかを判断します。

ユーザのためにいったんサービスが起動された後は、コンテキストは存在意義を失うため、破棄できることもあります。ただし、ユーザがサービスを実行するために認証を受ける必要があり、インタラクション間で「認可済み」コンテキストを維持したい場合など、ユーザが毎回認可プロセスを通さずにサービスを繰り返し使用できるような設定が必要な場合もあります。

---

**注** これらのデータ構造に準拠するコア Trustbase サービス用の規則および規則セットの例は、< インストールディレクトリ >/Trustbase/TTM/current/Config/Rules/Public にあります。

---

# ルータ規則の完全な DTD

iPlanet Trustbase Transaction Manager 内で使用可能なすべてのルーティング規則を指定する完全な DTD を次に示します。

```
<!-- A rule set is zero or more rules -->
<!ELEMENT ruleSet (rule*)>
<!ATTLIST ruleSet name CDATA #REQUIRED>
<!-- A rule optionally has a name attribute, preconditions & body elements -->
<!ELEMENT rule ( preconditions, body )>
<!ATTLIST rule name CDATA #IMPLIED>
<!-- preconditions element is a set of zero or more attribute condition elements -->
<!ELEMENT preconditions (attributeCondition*)>
<!-- An attribute condition is an empty tag with name, valueType & value attributes -->
<!-- the name is mandatory. valueType may be any, null or string. if string is -->
<!-- selected then value must be present. If any is selected then the value -->
<!-- matches any value and if null is selected then the value is disregarded -->
<!-- and assumed to be null -->
<!ELEMENT attributeCondition EMPTY>
<!ATTLIST attributeCondition name CDATA #REQUIRED>
<!ATTLIST attributeCondition operator ( greaterThan | greaterThanEqualTo | lessThan |
lessThanEqualTo | notEqual | startsWith | contains | endsWith | equals ) "equals">
<!ATTLIST attributeCondition valueType ( any | null | string ) "string">
<!ATTLIST attributeCondition value CDATA #IMPLIED>
<!-- The body contains one of four directives -->
<!ELEMENT body ((setAttribute | executeServiceDirective |
unauthorisedExecuteServiceDirective )*, (startContextDirective | endContextDirective
| returnToUserDirective )?)>
<!-- A setAttribute directive -->
<!ELEMENT setAttribute EMPTY>
<!ATTLIST setAttribute name  CDATA #REQUIRED>
<!ATTLIST setAttribute value  CDATA #REQUIRED>
<!ATTLIST setAttribute location (MESSAGE | CONTEXT) "CONTEXT">
<!ATTLIST setAttribute type  (ASCEND_TRANSITIVE | NONE | INHERIT_TRANSITIVE |
INHERIT) "ASCEND_TRANSITIVE">
<!-- A start context directive must contain a name for the rule set to be used. -->
<!-- optionally contains a rule name to be executed in the specified rule set -->
<!ELEMENT startContextDirective EMPTY>
<!ATTLIST startContextDirective ruleset CDATA #REQUIRED>
<!ATTLIST startContextDirective rule CDATA #IMPLIED>
<!-- An end context directive has no attributes or children -->
<!ELEMENT endContextDirective EMPTY>
<!-- execute directive must specify a name for the service to be executed -->
<!ELEMENT executeServiceDirective EMPTY>
<!ATTLIST executeServiceDirective name CDATA #REQUIRED>
<!-- unauthorisedExecute directive must specify service name to be executed -->
<!ELEMENT unauthorisedExecuteServiceDirective EMPTY>
<!ATTLIST unauthorisedExecuteServiceDirective name CDATA #REQUIRED>
<!-- A return to user directive is empty and has no attributes -->
<!ELEMENT returnToUserDirective EMPTY>
<!ATTLIST returnToUserDirective endContext ( true | false ) #REQUIRED>
```

第 4 章

# 構成管理

構成マネージャのサブシステムには、構成マネージャそのものおよび構成ストアの 2 つの要素があります。構成マネージャは、すべての構成リクエストの中央アクセスポイントです。また、ストア内の構成オブジェクトの維持も担当し、読み取りおよび更新のためにこれらのオブジェクトへのアクセスを同期します。

# 構成オブジェクト

構成オブジェクトは、サービスおよびその構成サービス用に永続データを維持します。1 つのサービスが複数の構成オブジェクトを参照することもあれば、複数のサービスまたは構成サービスが 1 つの構成オブジェクトを参照することもあります。また、各 iPlanet Trustbase Transaction Manager コンポーネントも構成オブジェクトを使用します。構成オブジェクトおよび構成マネージャを使用するサービスまたはコンポーネントは、構成可能エンティティと呼ばれます。構成オブジェクトに読み取り / 書き込みインターフェイスを実装するサービスは、構成サービスと呼ばれます。通常の場合は、ユーザが構成オブジェクト内に格納された値を変更できるようにするための、グラフィックまたは HTML ベースのインターフェイスが実装されます。

構成マネージャが適切に操作できるように、構成オブジェクトはすべて ConfigurationObject インターフェイスを実装している必要があります。構成オブジェクトには、ストア内で ConfigUID オブジェクトがインデックスとして付けられます。

---

**注**　ConfigurationObject が提示しなくてはならないインターフェイスに関する詳細は、uk.co.jcp.tbase.config.ConfigurationObject の API ドキュメントを参照してください。

---

# 構成マネージャ

サービスは、SingletonConfigManager クラスを使用して構成マネージャへのリファレンスを取得します。

構成マネージャは、ConfigUID オブジェクトが特定する読み取りまたは書き込み可能なバージョンの構成オブジェクトを、呼び出し元が取得することを許可します。

構成オブジェクトの取得には、2 つのメソッドが使用されます。

- 1 つ目のメソッドは、サービスの設定または構成サービス表示へのデータ挿入に使用できる、構成オブジェクトの読み取り可能ディープコピーを返します。このメソッドによって取得された構成オブジェクトに変更が加えられても、その変更は構成ストアには反映されません。これは、返された構成オブジェクトには有効な ConfigurationLock オブジェクトが含まれていないためです。また、リクエストされたオブジェクトに未処理のロックがあるかどうかに関わらず、この形の構成オブジェクトは必ず提供されます。つまり、構成マネージャが、ConfigurationObject の読み取り専用コピーに対するリクエストを拒否することはあり得ないということです。
- 2 つ目のメソッドは、一定期間有効な構成ロックを含む構成オブジェクトのディープコピーを返します。このロック期間そのものも構成可能です。指定の期間が過ぎるとロックは失効し、構成ストアに変更を適用しようとする試みは拒否されます。ロックの期限切れは、開発者が割り当てられたロックを開放しなかったことに起因するデッドロック問題に対する、単純な解決策を提供します。

構成オブジェクトを更新または削除できるのは、そのオブジェクトが有効なロックオブジェクトを含む場合だけです。

---

**注** サービスが ConfigUID を持つ構成オブジェクトを登録できるのは、同じ ConfigUID を持つオブジェクトが他に存在していない場合だけです。

---

構成可能エンティティは、構成オブジェクトに変更があった場合に通知を受け取るよう登録できます。構成オブジェクトに変更があった場合、登録済みエンティティには変更後の構成オブジェクトの読み取り専用コピーを含むイベントオブジェクトが送られます。

---

**注** 構成マネージャが ConfigurationObject 操作用に提供しているメソッドの詳細は、uk.co.jcp.tbase.config.SingletonConfigManager および uk.co.jcp.tbase.config.ConfigManager の API ドキュメントを参照してください。

---

# 構成ストア

構成ストアは、構成オブジェクトを永続的に格納します。iPlanet Trustbase Transaction Manager に付属の構成ストアは、配下に実装されている JDBC を使用してデータを格納しますが、その他の構成ストアを実装して、構成オブジェクトを任意の形式で格納することも可能です。

構成ストアが実装するのは、ConfigUID キーを通じて構成オブジェクトにアクセスするための、読み取り、書き込み、および削除のための基本的なメソッドだけです。ロッキングなどのより高度な意味論は、構成マネージャによって処理されます。

開発者がデフォルト実装の JDBC ストアを他のものに変えることはできません。また、JDBC ストアのインターフェイスはパブリックではありません。

# 構成サービス

構成サービスは ConfigurationObject の構成可能オブジェクトについての知識を持つ要素であり、これらの属性をユーザインターフェイスを通じて管理者に提示する機能を提供します。iPlanet Trustbase Transaction Manager は、HTML フォームを使用して構成データを提示します。

iPlanet Trustbase Transaction Manager に付属の構成サービスは、HTTPReader および ScriptWriter クラスを使用して、管理者による情報変更が可能な HTML ページまたはフォームを生成します。構成サービスは、プロトコルアナライザ、メッセージアナライザ、メッセージリーダおよびライタ、ルータおよびサービスを含む、個々の iPlanet Trustbase Transaction Manager コンポーネントを構成できます。更新された属性を受け取ると、構成サービスは構成マネージャにこれらの変更を登録し、その後、構成マネージャが他の関連エンティティに通知を送ります。

プラットフォームに登録されている各構成可能エンティティに対して、最低 1 つの構成サービスが存在します。ただし、構成サービスが構成するエンティティのインスタンス数に関わらず、各構成サービスの実行時のインスタンスは 1 つだけです。

---

**注** 構成サービスに関連付けられている API クラスはありません。構成サービスは通常の iPlanet Trustbase Transaction Manager サービスであり、uk.co.jcp.tbase.service.Service インターフェイスの実装が必要です。

---

第 5 章

# 標準的なサービス

サービスは、意思決定プロセスにおいてアクションを提供します。各サービスは iPlanet Trustbase Transaction Manager フレームワークのプラグインコンポーネントであり、それぞれ iPlanet Trustbase Transaction Manager のサービスインターフェイスの実装を提供します ( 詳細は『構成およびインストールガイド』を参照 )。

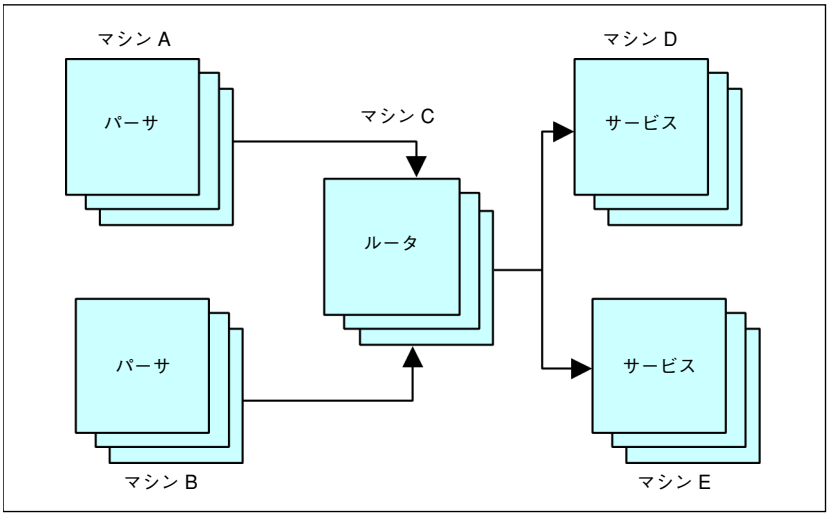
iPlanet Trustbase Transaction Manager は、さまざまなプラットフォーム、特にアプリケーションサーバ上で動作できるように設計されており、同時に多数のユーザに対応できるソリューションを提供しています。これは、コンポーネントの反復、すなわち複数の Java 仮想マシンまたは物理マシン上でクラスを実行することによって実現されています。iPlanet Trustbase Transaction Manager フレームワークは、このために必要なすべての条件を満たすように設計されています。各主要サブシステムは複数の物理マシン上で何度も反復できますが、大量処理を可能にするため、iPlanet Trustbase Transaction Manager フレームワークはサービスに対して多数の制限を設けています。

# 概要

サービスは次の条件を満たしている必要があります。

- 状態を持たない ( ステートレスである ) - 各コンポーネントはコンテキストに関わらずメッセージを処理できなくてはならない
- 細分化されている - シングルポイント障害のリスクを避けるため、各コンポーネントが実行するタスクの数は限られていなくてはならない

図 5–1 コンポーネントの反復




サービスはメッセージ処理に直接関わります。また、状態を持つことはできません。これは、反復を可能にするためです。

通常の場合、サービスは次の 3 つのカテゴリに分けられます。

- 認証および認可
- 既存のシステムへの統合を含むビジネス処理
- ユーザとの対話

複数のサービスにそれぞれ異なる分野の機能を割り当てることで、共通のインフラストラクチャ上に多数のアプリケーションを構築できるようになります。

特に、認証および認可サービスは企業全体で共有されるため、この 2 つのタスクの両方またはどちらかだけを専門に実行するサービスを考案することで、各アプリケーションを開発する際の作業量を軽減できます。

第 6 章

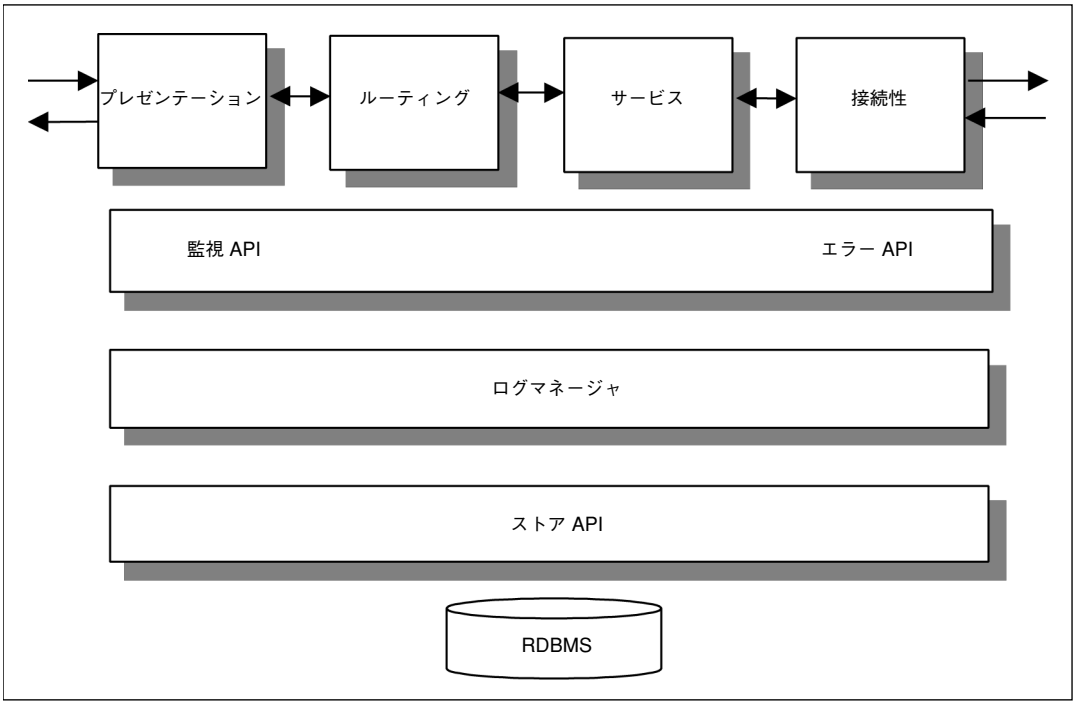
# エラーと監視のログ

iPlanet Trustbase Transaction Manager には、エラーや監視のログに使用できる一連のライブラリがあります。これらのライブラリのデフォルト実装により、フレームワークやビジネスコンポーネントで情報をリレーショナルデータベース内に格納して、外部のブラウザやエラー管理コンソールから使用することが可能です。

エラーログと監視ログは、どちらもログマネージャによって制御されます。ログマネージャは、現在の構成に従って、データを適切な場所に格納します。この設計によって、エラーや監視の情報を記録するための、場所に依存しないメカニズムが提供され、iPlanet Trustbase Transaction Manager の複数の実装間でコンポーネントを再使用できます。このログマネージャは、iPlanet Trustbase Transaction Manager を起動するたびにアクティブになります。ログマネージャの構成は、最初は初期化ファイルから読み込まれますが、次の起動からはログマネージャによって生成された永続構成オブジェクトから読み込まれます。

# 概要

図 6–1 iPlanet Trustbase Transaction Manager ログマネージャ




ログマネージャによって記録されるデータは、リクエスト元のコンポーネントにより、エラーログオブジェクトまたは監視ログオブジェクトとして提供されます。ログマネージャでは記録されるオブジェクトのタイプはチェックされますが、内容はチェックされません。タイプのチェックでは、受け渡すオブジェクトが iPlanet Trustbase Transaction Manager で定義されたエラーログクラスまたは監視ログクラスのサブクラスであることが確認されます。したがって、開発者はソリューションで特別に使用する新規のエラーログタイプまたは監視ログタイプ ( 請求ログなど ) を定義できます。

ログマネージャは、ストア API によりデータストアから抽出されます。したがって、ログをさまざまな方法で実装したり ( 請求情報のログを別の物理レポジトリに格納するなど )、特定のログタイプ (サードパーティの監視システムに渡されるイベントを生成するエラーなど ) を実装したりすることができます。

# 監視ログ

監視ログは、iPlanet Trustbase Transaction Manager プラットフォームのライフタイムにおける、次のような特定の時点で生成されます。

- サービスの起動イベント
- サービスの終了イベント ( 通常の終了と、異常な終了の両方 )
- すべての構成の変更
- すべてのセキュリティドメインの変更

これらの標準的な iPlanet Trustbase Transaction Manager 監視ログタイプは、uk.co.jcp.tbaseimpl.log.audit.type パッケージ内で定義されています。

## イベントの監視ログ

iPlanet Trustbase Transaction Manager 監視ログにエントリが生成されるようにするには、まずログをとる AuditObject タイプの新規インスタンスを作成します。このインスタンスは、uk.co.jcp.tbaseimpl.log.audit.AuditObject のサブタイプであることが必要です。次にこのインスタンスを静的メソッド AuditLog.log(...) に渡します。

ログされる監視の例を次に示します。

```
package com.iplanet.trustbase.app;
public class DummyService
{
......
private static final String SERVICE_STARTED = "START";

// Audit the occurrence of this event in the Audit Log
AuditLog.log(new OperationBeginAudit( this.getClass(),
SERVICE_STARTED, new String[]{ serviceName.toString() });
......
}
```

すべての AuditObject には、次の 3 つのパラメータを使用します。

- this.getClass()
  クラスオブジェクト。この名前がメッセージの国際化バンドルキーの最初の部分に使われる。これは、通常は監視のログをとるクラスで、監視メッセージとパッケージの関連付けを容易にする。ただし、ほかの任意のクラスオブジェクトを使用することも可能
- SERVICE_STARTED
  任意の文字列。クラス名と連結して、監視目的でメッセージのテキストを見つけるための固有のバンドルキーを提供する。AuditBundle キーとその値については、後述の説明を参照

- new String[] { serviceName.toString() }
  文字列の配列。標準の java.text.MessageFormat クラスを使って最終的な監視メッセージを提供するために、AuditBundle メッセージとともに使用されるパラメータ

上に述べたように、クラスパスに AuditBundle ファイルがあり、実際の監視メッセージを検索できるようにバンドルマネージャに登録されていることが必要です (「新規監視タイプの定義」を参照 )。前述の例では、.../com/iplanet/trustbase/app に AuditBundle.properties ファイルが作成されます。このファイルには次の行が含まれます。

```
com.iplanet.trustbase.app.DummyService_START=The service {0} has begun.
```

これは「this.getClass()」パラメータによって提供され、文字「_」と文字列「SERVICE_STARTED」が後についた、完全指定のクラス名です。

この監視に関連付けられるメッセージは「The service {0} has begun.」になります。「{0}」の部分は、java.text.MessageFormat クラスにより、監視コンストラクタが受け取る文字列配列のエントリ 0 に置き換えられます。

---

**注** 監視ログに関連付けられる API クラスは、uk.co.jcp.tbaseimpl.log.audit.AuditLog、uk.co.jcp.tbaseimpl.log.audit.AuditObject、および uk.co.jcp.tbaseimpl.log.audit.type 内の多くの標準的な具体監視クラスです。

---

# 新規監視タイプの定義

開発者は、新規監視ログタイプを定義して、アプリケーションやドメインに固有のメッセージを使用することができます。これらのログタイプは、uk.co.jcp.tbaseimpl.log.audit.AuditObject クラスのサブクラスであることが必要です。

これらの拡張監視タイプは、iPlanet Trustbase Transaction Manager サービスによって処理されたメッセージに関する情報のログをとる場合に、もっとも一般的に使用されます。

新規監視タイプを実装したら、次の手順に従って、標準の iPlanet Trustbase Transaction Manager 監視ログにこのタイプが表示されるようにします。

- 新規監視クラスを作成し、この監視を利用するサービスの JAR ファイルに入れます。iPlanet Trustbase Transaction Manager サービスの配置については、後の章で詳しく説明します。
- 監視メッセージに対し、適切なリソースバンドルを作成します。メッセージをさまざまな言語で簡単に地域対応化するには、iPlanet Trustbase Transaction Manager 監視ログに表示されるすべての監視タイプ文字列をリソースバンドル内で定義します。これらのバンドルは、新規 AuditType クラスと同じパッケージ ( 同じディレクトリ ) に入れます。

リソースバンドルの名前の形式は、標準的な Java の java.util.ResourceBundle クラスで定義されているものと同じにします。バンドルの形式は、名前と値の組み合わせのリストになります。名前の部分は「完全指定の監視タイプクラス名 _bundleKey」の形で構成され、値は監視ログの AuditType フィールドに入れられる文字列になります。

- tbase.properties にエントリを作成することによって、新規監視リソースバンドルを登録します。このためには、新規監視リソースバンドルの完全指定の名前を参照する [BundleProviderManager/Audit] セクションにエントリを追加します。
- 最後に、tbase.properties で新規監視タイプを有効にします。このためには、サービスで定義される各新規監視クラスに対し、[uk.co.jcp.tbaseimpl.log.present.audit.AuditLogPresentationConfigService] セクションにエントリを追加します。
- iPlanet Trustbase Transaction Manager を再起動して、新規監視タイプをアクティブにします。これは新規サービスを有効にするために必要な作業であり、通常は新規監視タイプを作成するたびに必要な作業ではありません。

AuditObject の実装例を次に示します。

```
package uk.co.jcp.tbaseimpl.log.audit.type;
import uk.co.jcp.tbaseimpl.log.audit.*;

/** This class represents an audit object for recording that an
operation has begun
*/
public class OperationBeginAudit extends AuditObject
{
        public OperationBeginAudit ( Class auditClass , String
bundleKey , String [] params )
        {
        super ( auditClass , bundleKey , params );
        }
        public static String getAuditTypeString()
        {
        // The bundleKey is added to this class name to determine the
        // actual string to be writteninto the AuditType field of the
log
        String bundleKey = "OPERATION_BEGIN";
        return
uk.co.jcp.tbaseimpl.log.audit.AuditObject.getAuditTypeString (
uk.co.jcp.tbaseimpl.log.audit.type.OperationBeginAudit.class,
bundleKey );
        }
}
```

リソースバンドルを定義するファイルを次に示します。このファイルは、パッケージ階層の OperationBeginAudit クラスと同じ場所に入れます。ファイル名は TheNewAuditBundle_en.properties です。

```
uk.co.jcp.tbaseimpl.log.audit.type.OperationBeginAudit_OPERATION_BEGIN =
OPERATION_BEGIN
```

新規バンドルを登録するために tbase.properties に追加するエントリを次に示します。ロケールを表す拡張子「_en」および .properties 拡張子は必要ありません。これは標準的な Java の java.util.ResourceBundle クラスと同様です。

```
[BundleProviderManager/Audit]
bundle=uk.co.jcp.tbaseimpl.log.audit.type.TheNewAuditBundle
```

新規監視タイプを有効にするために tbase.properties に追加するエントリを次に示します。

```
[uk.co.jcp.tbaseimpl.log.present.audit.AuditLogPresentationConfig
Service]
auditlog.type.enabled=uk.co.jcp.tbaseimpl.log.audit.type.Operation
BeginAudit
```

# エラーの処理とログ

iPlanet Trustbase Transaction Manager プラットフォームで発生するエラーは、次の 2 つのカテゴリに分類できます。

- 論理エラー - データ、または処理に不適切な部分があり、リクエストをていねいに拒否する必要がある場合
- 例外 - 予期しない状態が発生し、処理ロジックが中断された場合

どちらの場合も、iPlanet Trustbase Transaction Manager のログマネージャによってエラーが発生したことが記録されるため、分析を行って適切な処置を取ることができます。

## エラーのログ

iPlanet Trustbase Transaction Manager では、重要度を取る単独のエラーログクラス、エラーを定義するオブジェクトのクラス、およびプログラマによって定義されたメッセージに加え、メッセージに代入できる任意の数の文字列の引数を使用できます。デフォルトのエラーログの実装（uk.co.jcp.tbaseimpl.log.error.ErrorLog）では、さまざまな重要度を示す 4 つの定数が定義されています。

**表 6–1** エラーの重要度のタイプ

| 定数 | 説明 |
|---|---|
| INFORMATION | 情報イベントのログに使用。たとえば、コード内の関連のないセクションが実行されたなど、特にエラーとはいえないもの。あまり多用しないようにする。この定数は値「0」を持つ |
| WARNING | 予期された処理可能なものであるが、動作分析のためにログが必要なエラーに使用。この定数は値「1」を持つ |
| ERROR | システムや、システム内の情報に本質的な問題があることを示す、重大なエラーに使用。ただし、このエラーでは処理の続行や再試行が可能。この定数は値「2」を持つ |
| FATAL | 処理を回復できない、致命的なエラーに使用。これらのエラーが発生すると、処理が放棄される。この定数は値「3」を持つ |

**注** エラーログに関係する API クラスは、uk.co.jcp.tbaseimpl.log.error.ErrorLog および uk.co.jcp.tbaseimpl.log.error.ErrorObject です。

iPlanet Trustbase Transaction Manager でエラーのログをとるには、ErrorObject のインスタンスを含む ErrorLog.log( .... ) を呼び出します。ErrorObject のコンストラクタは多数ありますが、これらのコンストラクタはすべて、このエラー固有のエラーコードを識別する文字列パラメータを少なくとも 1 つ含んでいます。

iPlanet Trustbase Transaction Manager のエラーログメカニズムでは、発生するエラーそれぞれに対し、iPlanet Trustbase Transaction Manager 全体で固有なコードが与えられていることが必要です。iPlanet Trustbase Transaction Manager のエラーログサブシステムのエラー情報はすべて、次に示す各データベーステーブルに格納されます。

表 6-2 に、固有なエラーコードの詳細をまとめます。

**表 6–2 Error_codes**

| **error_codes** テーブル | |
|---|---|
| errorcode | エラーを識別する、固有な errorcode 文字列。長さは必ず 7 文字であること。通常の形式は「XXXnnnn」であり、「XXX」はサービスまたはサブシステムの 3 文字のコードを、「nnnn」は固有の番号を表す。たとえば、「IPH0009」は Identrus プロトコルハンドラのエラー |
| classname | このエラーのログをとるクラス。これにより、各エラーコードは 1 つの場所からのみ使用可能という制限が追加される |
| severity | 前に述べた、エラーの重要度。各重要度の定数は、uk.co.jcp.tbaseimpl.log.error.ErrorLog クラスにある |
| message | エラーログに表示される、地域対応化されたエラーメッセージ。標準的な Java クラスである java.text.MessageFormat によって記述されているため、このメッセージではパラメータを使用可能。これらのパラメータに入れる値は、ErrorObject コンストラクタの 1 つが受け入れる文字列の配列として渡される |

次に、実際のエラーログテーブルについて説明します。通常、管理者がこのテーブルを直接表示することはありません。代わりに、ログされたエラーを目的に応じて表示できる、errorview という Oracle のビューを使用します。

**表 6–3 エラー**

| エラーテーブル | |
|---|---|
| errorid | エラーログエントリの固有 ID。順番に増加するシーケンスから生成される。つまり、このフィールドを使って、エラーメッセージをエラーがログされた順番に正確に並べ替えることが可能 |
| errorcode | ログされるエラーのエラーコード。「error_codes テーブル」を参照 |

| エラーテーブル | |
|---|---|
| message | error_codes テーブルのメッセージ文字列と、ランタイムシステムによって提供された変数パラメータを組み合わせて生成された、最終的なメッセージ文字列 |
| timestamp | エラーがログされた日時を識別する ORACLE の DateTime フィールド |
| severity | このエラーの error_codes エントリから取得された、重要度を表す整数 |
| classname | エラーをログしたクラス名 |
| machineid | エラーをログした iPlanet Trustbase Transaction Manager の IP アドレスを示す文字列。マルチノードの IAS インストールでは異なる場合もある |
| contextid | 将来の拡張用のコンテキスト ID |

エラーがログされる場合、自由形式の文字列データが付属していることが多くあります。これらの文字列データは、エラーが発生したコンテキストを示し、診断の助けとなります。このようなデータのもっとも一般的な例としては、例外のスタックトレースがあります。

**表 6–4** エラーサポート

| **error_support** テーブル | |
|---|---|
| errorid | このエントリを「エラーテーブル」のエントリにリンク |
| datatype | 任意の文字列識別子。データフィールド内のデータを分類する。iPlanet Trustbase Transaction Manager で定義されているこのフィールドの唯一の値は、データフィールドの内容が Java の例外のスタックトレースであることを示す「STACKTRACE」 |
| data | 自由形式の文字列データ |

ここで説明したテーブルでは、iPlanet Trustbase Transaction Manager でサポートされているデータ指向型のエラーログメカニズムが完全に取り込まれています。

# 新規エラーの定義

新規エラーの定義はとても簡単です。定義を行うには、error_codes テーブルに新規エントリを作成します。現在のところ、この操作をサポートするツールはありませんが、基本的な SQL を使って実行できます。唯一気をつけなければならない点として、error_code フィールドは固有でなければならないことがあげられます。これは ORACLE によるテーブルに対する制限で、新規コードをエラーなしに挿入するには、そのコードが固有である必要があります。

次に、SQL を使ってエラーコードを挿入する例をあげます。

```
INSERT INTO error_codes values
(
'TST0001',
'com.iplanet.trustbase.sample.service',
'1',
'This is the only exception in the test service'
);
```

このエラーのログをとるには、次のコードを使います。

```
.....
}
catch (Exception exc)
{
        ErrorLog.log( new ErrorObject( "TST0001", exc );
}
.....
```

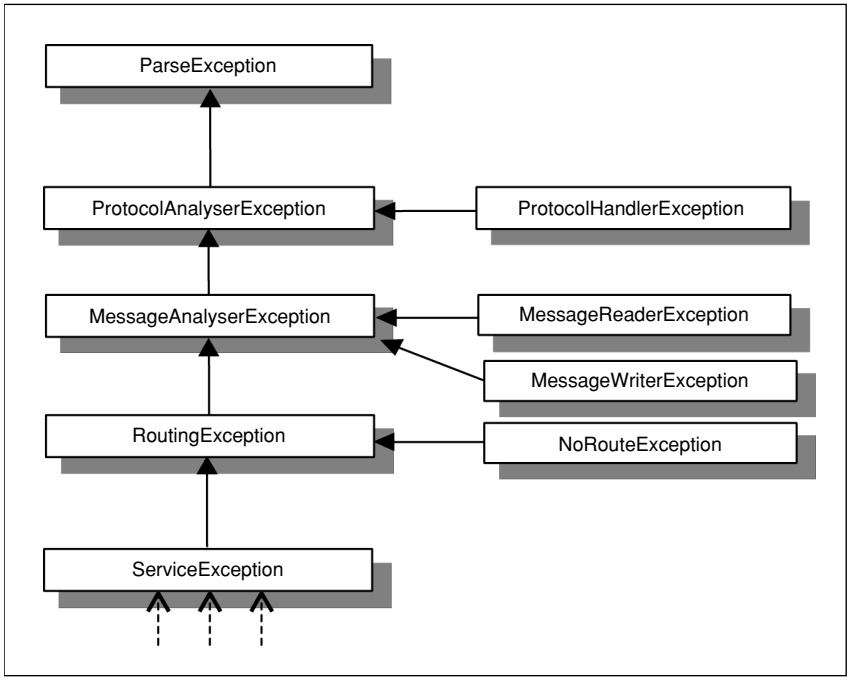
これにより、エラーテーブルにエントリが作成され、関連するスタックトレースが error_support テーブルにログされます。

# 例外の処理

論理エラーと同様、例外の処理でも情報のログをとり、システムのユーザや開発者が例外の発生した状況を分析できるようにします。

iPlanet Trustbase Transaction Manager の例外階層は、呼び出しを行ったコンポーネントが明示的に処理コードを提供しなくても、さまざまなコンポーネントが呼び出しを行ったコンポーネントに対して例外を返せるように設計されています。これは、例外の連続的な各「レベル」( それぞれコードの連続的な「レベル」に関連 ) が、前のレベルから派生しているためです。たとえば、ServiceException から派生する例外をスローするサービスを考えてみましょう。サービスによって明示的に例外の処理が行われない場合、例外はルータ ( 呼び出しを行ったコンポーネント ) に戻されます。ServiceException は RoutingException から派生しているので、ルータは例外を適切に処理できると想定できます。

図　6–2　　iPlanet Trustbase Transaction Manager の例外階層




現在 iPlanet Trustbase Transaction Manager システムに内蔵されている処理コードが適切に機能するためには、システムの拡張機能の開発者がこの階層によるアプローチを維持することが非常に重要です。

この階層状の構造は、例外が発生した場所によって異なる処理を行うことを可能にします。コンポーネントおよび関連するアクションを表　6–5 に示します。

表 6–5　例外

| コンポーネント | 基本的な例外 | アクション |
|---|---|---|
| サーブレット | TbaseRuntimeException | TbaseRuntimeExceptions をキャッチし、適切な HTTP エラーを生成 |
| プロトコル アナライザ | ProtocolAnalyserException | サーブレットで HTTP エラーとなる ParseException をスローする場合がある |
| メッセージ アナライザ | MessageAnalyserException | RemoteException が発生した際に生成される。サーブレットで HTTP エラーを引き起こす |
| メッセージ リーダ | MessageReaderException | 埋め込みメッセージを含む場合、メッセージアナライザでキャッチされ、そのメッセージが処理される。その他の場合、例外はサーブレットに渡される |
| メッセージ ライタ | MessageWriterException | メッセージアナライザを通して、HTTP エラーを返すサーブレットに渡される |
| ルータ | RoutingException | メッセージアナライザを通して、HTTP エラーを返すサーブレットに渡される |
| サービス | ServiceException | メッセージを含む場合と、含まない場合がある。ルータを通して、メッセージアナライザに返され、MessageReaderException と同様に処理される |

第 7 章

# Identrus ログ

Identrus トランザクションコーディネータの仕様では、次の 2 つのログアクションが指定されています。

- トランザクションコーディネータによって送受信されたすべてのメッセージのログ (原初ログ )
- 請求用のデータの生成

# 概要

iPlanet Trustbase Transaction Manager は、デフォルトの Identrus メッセージ処理アクションによってこれら 2 つの条件を満たしています。データはインストール時に指定した RDBMS に格納され、開発者は標準の JDBC を介してテーブルにアクセスし、この情報を使うサービスを提供できます。この章では、RDBMS に格納されるテーブルを定義し、各テーブル間の関係を説明します。iPlanet Trustbase Transaction Manager では、次に説明するテーブルすべてを、すべての Identrus メッセージに使います。開発者がこれらのテーブルを作成する必要はありません。

# データの定義

## 接続情報

SSL プロクシと SMTP メールリスナーは、どちらも行った接続に関するログをとります。表　7–1 に、SSL プロクシのログの各列についてまとめます。

表　7–1　　**SSL** 接続

| ssl_connection テーブル | |
|---|---|
| ConnectionId | 固有の接続識別子 |
| ClientCertIssuerDN | 接続元クライアントの証明書発行元 DN |
| ClientCertSerialNumber | 接続元クライアントの証明書のシリアル番号 |
| CipherSuite | SSL セッションに使用される符号化 |
| ConnectTime | 接続を確立した時刻。ORACLE の DateTime フィールド |
| TimeStampType | タイムスタンプのタイプ |
| ConnectIPAddr | 接続元クライアントの IP アドレス |
| ConnectionFailed | 接続の成否を示す整数値。値が「1」の場合は失敗 |
| ConnectionFailedReason | 失敗した場合の、SSL エラーコード |

次の表では、SMTP/SMIME 接続ログの各列について説明します。表　7–2 のデータは、SMIME v2 におけるメッセージの署名の本文部分から抽出されるものです。

表　7–2　　**SMIME** トランスポート

| smime_transport テーブル | |
|---|---|
| ConnectionId | 「smtp_message テーブル」へ戻るリンクを提供 |
| peer_issuer_dn | メッセージの確認に使用した証明書の発行元 DN |
| peer_cert_serial_number | メッセージの確認に使用した証明書のシリアル番号 |
| message_protection | メッセージに使用した保護のタイプ |
| time_stamp_type | タイムスタンプのタイプ。LOCAL または NETWORK |
| time_stamp | エントリが作成された日時 |

表 7–3 **SMTP** 接続

| smtp_connection テーブル | |
|---|---|
| stream_id | 「smtp_message テーブル」へ戻るリンクを提供 |
| peer_ip_addr | 送信元 SMTP エージェントの IP アドレス |
| timestamptype | タイムスタンプのタイプ。LOCAL または NETWORK |
| timestamp | エントリが作成された日時 |

表 7–4 **SMTP** メッセージ

| smtp_message テーブル | |
|---|---|
| stream_id | smime_transport の固有の ID |
| connection_id | SMTP 接続の固有 ID |
| recipients | このメッセージの受取人 |
| sender | このメッセージの差出人 |
| timestamptype | タイムスタンプのタイプ。LOCAL または NETWORK |
| message_valid | メッセージが有効かどうか。「1」の場合は有効 |
| message_invalid_reason | メッセージが無効な理由 |
| timestamp | エントリが作成された日時 |

ssl_connection テーブル と smtp_message テーブルには、connection_id フィールドがあり、このフィールドはアプリケーションサーバで実行中の iPlanet Trustbase Transaction Manager に渡されます。この connection_id は原初ログテーブルに格納されるため、接続元の情報を実際のリクエストと関連付けてクエリすることができます。

表 7–5 **OCSP**

| ocsp_data テーブル | |
|---|---|
| ocspid | レコードの固有の識別子 |
| type | OCSPREQUEST または OCSPRESPONSE |
| message | リクエストまたは応答の内容のテキストによる要約 |
| machine | リクエストの送信先または応答の送信元の URL |

| ocsp_data テーブル | |
|---|---|
| timestamp | エントリが作成された日時 |
| data | リクエストまたは応答の Base64 エンコード |

# 原初ログテーブル

Identrus メッセージのデフォルトのプレゼンテーションハンドラは、送受信した各メッセージについて次のデータを記録します。

表 7–6　原初ログ

| raw_data テーブル | |
|---|---|
| Sessionid | このレコードを作成した原初ログセッションの ID |
| Logconnectionid | セッション内の接続 ID |
| Recordid | セッション内のレコードの ID |
| msggrpid | メッセージの NIB からの Identrus MsgGrpId |
| msgid | メッセージの NIB からの Identrus MsgId |
| doctype | メッセージの DOCTYPE。CSCRequest や PingRequest など |
| recordmarker | 順番に増加する固有の識別子 |
| connectionid | このレコードを SSL または SMIME の接続ログにリンクする接続 ID |
| protocoltype | メッセージの受信に使用されたプロトコル。HTTP または SMTP |
| input | メッセージが iPlanet Trustbase Transaction Manager で送信されたか、受信されたか。「1」は受信 |
| timestamp | レコードがログされた時点での UNIX の時刻を示す整数 |
| rawdata | Identrus メッセージ XML (CertBundle フィールドなし )。バンドルからの証明書は、「cert_data テーブル」に別に記録される |
| digestofrecord | このレコードの SHA-1 ダイジェスト |
| signeddigestofcalculation | このレコードの RSA 署名と、前のレコードからのデータ |
| servercertissuerdn | 署名の確認に使用した証明書の発行元 DN |
| servercertserialnumber | 署名の確認に使用した証明書のシリアル番号 |

各 Identrus メッセージに関するログデータの量を抑えるために、メッセージヘッダに含まれる証明書は省略され、証明書テーブルに格納されます。iPlanet Trustbase Transaction Manager で、すでに特定の証明書がテーブルにログされている場合、同じ証明書が再びログされることはありません。テーブルに格納される情報は次のとおりです。

表 7–7　証明書データ

| cert_data テーブル | |
|---|---|
| IssuerDN | 証明書の発行元の識別名。RFC 2253 形式の文字列 |
| SerialNumber | 証明書のシリアル番号 |
| CertData | Base64 の証明書データ |

このデータは、不正な編集を検知できるように設計されており、サービスによって原初ログテーブルやタンパテーブルのデータが編集されることは決してありません。不正な編集を検知するために、連続のハッシュが生成され、各レコードとともに格納されます。また、現在のハッシュは、別のタンパテーブルの署名済みレコード内に格納されます。タンパテーブルのフィールドについては、ここでは説明しません。原初ログ内のレコードで不正編集が行われていないかどうかを確認する方法については、『構成およびインストールガイド』を参照してください。

# 請求レコード

請求レコードは、原初メッセージログ内の情報のサブセットであり、各トランザクションを誰が行ったかを見極めるのに十分な情報が記録されています。これらのテーブルは、カスタマ用の実際の請求書を生成する、サードパーティのツールで使用するように設計されています。請求テーブルの各列について、次の表にまとめます。

表 7–8　請求データ

| bill_data テーブル | |
|---|---|
| RawRecordId | 関連する原初ログテーブルレコードの RawRecordId |
| SubjectDN | 必須の Identrus レベル 1 メッセージ署名から抽出された、送信元の識別名。これにより、請求先が決定される |
| IssuerDN | 必須の Identrus レベル 1 メッセージ署名から抽出された、発行元の識別名。このメッセージの署名に使用された実際のキーを識別するために、次のシリアル番号のフィールドとともに使用される |
| SerialNumber | メッセージの署名に使用された実際のキーの識別に使用できる、送信元の証明書のシリアル番号。発行元の識別名とともに使用される |

第 8 章

# Identrus ソリューションの構築

iPlanet Trustbase Transaction Manager プラットフォームと、関連する開発ツールは、基盤となる DTD から、ランタイム環境で使用するサービスのインストールまで、Identrus アプリケーションを開発する手段を提供しています。

この章では、Ping と呼ばれる Identrus アプリケーションの生成について、段階を追って説明します。このアプリケーションの設計は、次の点に焦点を当てています。

- 開発プロセス全体
- iPlanet Trustbase Transaction Manager 開発ツールの使用
- 構成オブジェクトの使用
- ログマネージャの使用

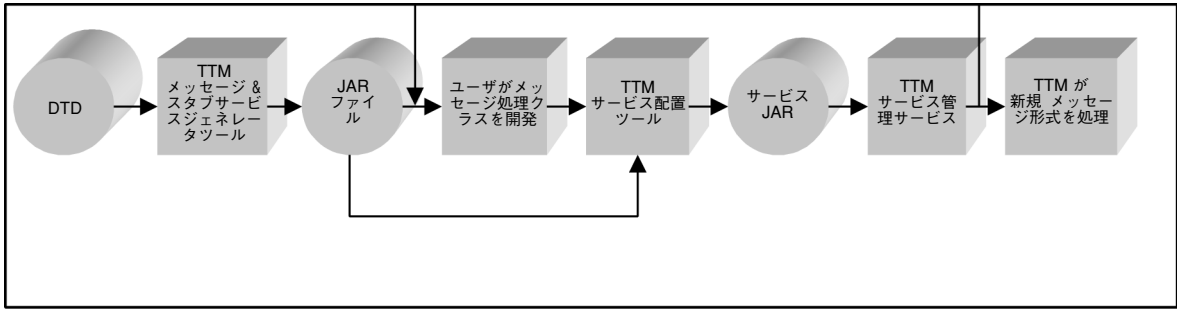
# 方法論

## 開発プロセス

アプリケーションの開発には、基盤となる Identrus 準拠の DTD の入力が必要です。これは一連のスタブ規則や JAR ファイル内のサービスの定義とともに、DTD のすべての要素に対する Java クラスを生成するクラスジェネレータを通じて受け渡されます。次に開発者は、適切なビジネスロジックを、適切なアクションを実行するサービススタブによって呼び出される一連の Java クラスとして実行し、この基盤データを自由に拡張できます。

一連の生成されたファイルと、関連する開発者のファイルは、1 つの JAR ファイルにまとめて iPlanet Trustbase Transaction Manager のディレクトリ構造内に配置します。iPlanet Trustbase Transaction Manager を再起動すると、JAR ファイル内のパッケージ記述子が検索され、この新しいサービスを実行環境内でアクティブ化できるようになります。全体的なプロセスを、図 8–1 にまとめます。

図 8–1 開発プロセス




## クラスの生成

iPlanet Trustbase Transaction Manager のクラス生成ツールは、メッセージ処理パイプラインの 2 つの場所で使用できる、一連のクラスを作成するように設計されています。2 つの場所とは、次のとおりです。

プレゼンテーション - iPlanet Trustbase Transaction Manager XML パーサでは、クラスを使用して受信 XML が DTD と整合性がとれているか確認

サービス - メッセージの要素にアクセスするビジネスロジック

クラス生成ツールにアクセスするには、JDK 1.2.x 以降がインストールされていることが必要です。JRE には、iPlanet Trustbase Transaction Manager サービスを構築するコンパイラは含まれていません。また、クラスパスが適切に設定されていなければなりません。iPlanet Trustbase Transaction Manager のインストール時に、次のコマンドを実行してください。

```
cd …….../Trustbase/TTM/Scripts
. ./setcp
```

これにより、CLASSPATH 環境変数が iPlanet Trustbase Transaction Manager ツールに適切な値に設定されます。

クラス生成ツールはコマンドラインオプションからアクセスでき、次のように呼び出します。

```
java com.iplanet.trustbase.app.classgen.ClassGen [- オプション ] < パブリック ID> <DTD ファイル名 >
```

オプションには、次のものがあります。

```
-help                このヘルプ画面を表示
-d < ディレクトリ >    出力ディレクトリの指定
-o <jar 名 >          出力 jar ファイルの指定
-v                   詳細出力
-r                   含まれる DTD すべてに対し、クラスの生成を繰り返す
-root < 要素名 >      DTD の各ルート要素に対して指定
-stub < クラス名 >    作成するスタブサービスの完全指定のクラス名
```

パブリック ID は、生成するサービススタブの名前です。これは Identrus DTD に必須で、標準的な形式は次のようになります。

```
-//IDENTRUS//service_name//EN
```

「service_name」の部分には、DTD で宣言される Identrus サービスを指定します。パブリック ID では大文字と小文字が区別されます。これは生成されるクラスのパッケージ名の一部として使われるので、注意して入力してください。

ツールで生成されるのは、クラスパス上に使用できるクラスの存在しない DTD 要素のクラスのみです。コア Identrus クラスはすでに Trustbase の jar にあるので、新規 Identrus サービスを作成するたびに再生成されることはありません。

「-root」エントリは、メッセージのルート XML 要素となり得る DTD の各要素に対して必ず作成しなければなりません。これは、iPlanet Trustbase Transaction Manager のデフォルトプレゼンテーション層で Identrus メッセージを自動処理するために必要です。

新規メッセージタイプのスタブ iPlanet Trustbase Transaction Manager サービスの生成にツールが必要な場合は、必ず「-stub」オプションを使用します。このスタブクラスの Java ソースファイルは、ツールが終了したときに現在のディレクトリに配置されます。いったん生成が完了したら、スタブクラスの再パッケージは行わないでください。生成された jar ファイルの tbasesvc.properties ファイルに、スタブクラスの完全指定名が記録されています。

クラスジェネレータでは、参照された DTD に対して、読み込みが繰り返されます。参照には、次のものを使用できます。

- 不完全指定 - DTD がクラスジェネレータを実行しているディレクトリにあることが必要
- ローカル - file:/// 修飾子を使用
- 完全指定 - HTTP://www 修飾子を使用

Classgen ツールを呼び出すと、次の出力が生成されます。

- 次のものを含む zip
  - 一連の生成されたクラス (DTD 内の各エンティティに対して 1 つずつ )
  - クラスの生成で実際に XML パーサによって使用された入力を表す、一連の DTD
  - iPlanet Trustbase Transaction Manager によってサービスの登録に使用された tbasesvc.properties ファイル
- 現在のディレクトリ内のサービススタブ (「-stub」オプションを使用した場合のみ )

生成された各クラスでは、TbaseElement インターフェイスが実装されます。このインターフェイスには、次の 2 つの役割があります。

- パーサ - iPlanet Trustbase Transaction Manager の解釈メカニズムで受信する XML メッセージを確認できるようにする
- サービス - 受信メッセージからデータを取得し、XML を明示的に生成することなく有効な応答を構築する、標準的な手段を提供する

TbaseIdentifiedElement という TbaseElement のサブクラスは、「id」という属性のタイプが「ID」である要素を表します。これにより、XML DSIG 署名を生成または確認する際に、タイプにかかわらず ID の適合性を確認できる方法が提供されます。

TbaseElement のサービス役割は、次の機能を提供します。

- XML ドキュメント階層の構築
- 要素属性を名前によって読み取り / 書き込み
- XML ドキュメントのオブジェクトによる表現を確認
- オブジェクト階層によって記述された XML ドキュメントの文字列による表現を生成

クラスの生成時に「-root」オプションによって識別されたすべての要素は、ベースクラスである com.iplanet.trustbase.identrus.message.IdentrusMessage を拡張することになります。これにより、メッセージが「Identrus Core Network Messaging Definition」に定義されている有効な構造を持ち、フレームワークで必須のメッセージ処理を実行できるようになります。このインターフェイスの使用例は、このマニュアルの最後にあるサンプルのセクションを参照してください。

---

注　　com.iplanet.trustbase.identrus.message.IdentrusMessage、com.iplanet.trustbase.xml.message.TbaseElement および com.iplanet.trustbase.xml.message.TbaseIdentifiedElement の API クラスでは、それぞれの機能についてより詳細な情報が提供されています。

---

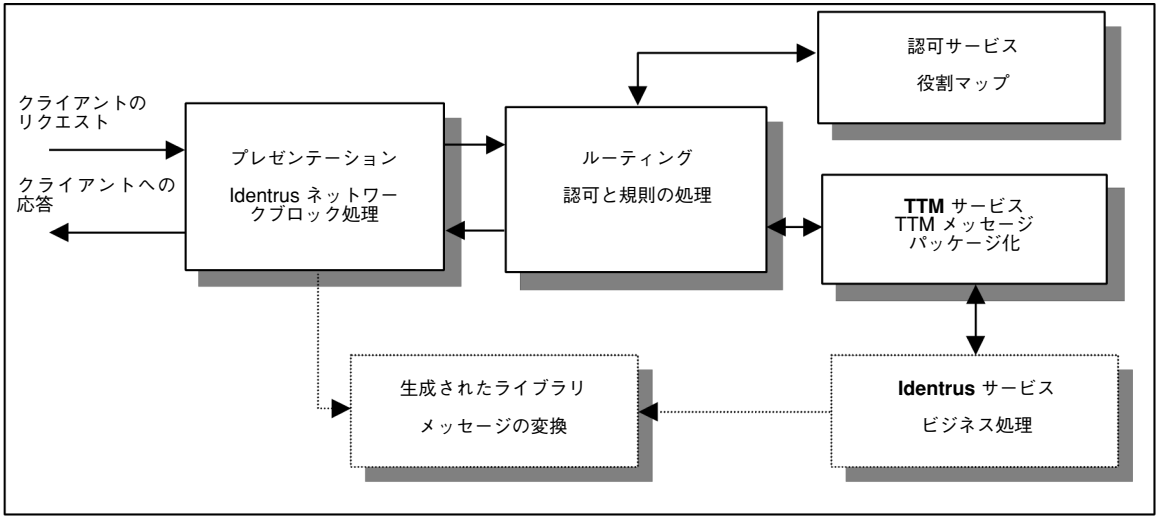
# サービスの開発

サービスを開発するには、サービススタブを機能ビジネスコンポーネントに取り込みます。

各サービスは、プラットフォームで複製できるように、状態がないことが必要です。つまり、前のセッションステータストランザクションに関する情報を保持することなく、異なるトランザクションやユーザからのメッセージを処理できる必要があります。サービスでセッションやトランザクションのステータスに関する情報を保持する必要がある場合は、データベース内で永続化するようにします。そうでないと、トランザクションの次のメッセージが、クラスタ内の同じホストマシンに送られない場合があります。

すべての Identrus ネットワークメッセージ処理は、クラス生成ツールによって生成されたサービススタブクラスの ProcessIdentrusMessage メソッドの呼び出し前後に、iPlanet Trustbase Transaction Manager プラットフォームで実行されます。図 8-2 に、開発者によって作成された Identrus サービスを配置した場合の、メッセージ処理の流れを示します。この Identrus サービスは、生成されたメッセージライブラリを使用し、iPlanet Trustbase Transaction Manager メッセージ処理スタブによってプロクシされています。

図 8–2 メッセージ処理の流れ




つまり、ほとんどの場合、Identrus サービスを開発するために必要な作業は、受信メッセージのビジネス処理を実装するために、生成されたスタブクラスを編集することと、送信メッセージのコアの内容を生成することだけです。

ビジネス処理部分では、メッセージに対する必須の署名確認や生成などの、必須のメッセージ処理を行う必要はありません。これらのコアの処理タスクは、iPlanet Trustbase Transaction Manager のプレゼンテーション層で実行されます。

サービスを開発するには、生成された jar ファイルを、コア iPlanet Trustbase Transaction Manager ライブラリとともに、開発時にクラスパスに配置します。

## サービスの構築

メッセージクラスを生成し、スタブサービスのメッセージ処理ロジック拡張機能を開発したら、次に iPlanet Trustbase Transaction Manager サービス jar を構築します。

iPlanet Trustbase Transaction Manager サービス jar を構築する手順は、次のとおりです。

- 標準的な JDK の jar ツールを使用して、生成した jar ファイルをディレクトリに解凍します。

```
mkdir DumpDirectory
cd DumpDirectory
jar -xvf ../jarfile.jar
cd ..
```

- 「DumpDirectory」に、メッセージの処理に必要なすべてのクラスをコピーします。これらのクラスすべてで、適切なパッケージ構造が維持されていることを確認します。特に、サービススタブクラスは、クラス生成ツールによって作成されたので、パッケージし直すことはできないことに注意してください。
- 標準的な JDK の jar ツールを使って、アーカイブを再度作成します。
- これで jar ファイルを配置することができます。

## サービスの配置

サービス jar ファイルを構築したら、iPlanet Trustbase Transaction Manager に配置します。手順は次のとおりです。

- jar ファイルを ...../Trustbase/TTM/current/deploy ディレクトリにコピーします。
- 管理コンソールを使って、実行中の iPlanet Trustbase Transaction Manager にサービスを配置します。
- iPlanet Trustbase Transaction Manager を再起動して、サービスをアクティブ化します。

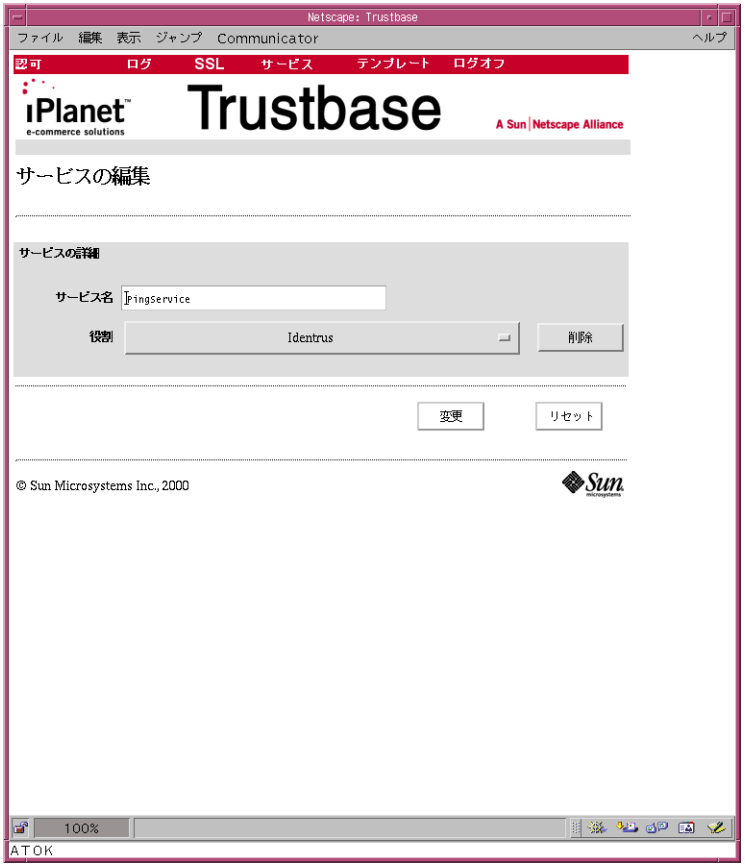
サービスを配置するには、iPlanet Trustbase Transaction Manager に「administrator」としてログインし、「サービス」メニューから「配置」オプションを選択します。サービス jar を deploy ディレクトリに適切にコピーしていれば、次の図に示すように、新規サービスが「サービス配置」ページに表示されます。

- を選択してから 更新 を選択することによって、サービスを配置します。

図 8–3 iPlanet Trustbase Transaction Manager での PingService の配置

サービスを認可する必要がある場合は、「認可」-「サービスの追加」を選択します。次に示すように、サービスには役割を割り当てる必要があります。

図　8–4　　サービスへの役割の割り当て




注　　使用できる役割がない場合は、役割を定義する必要があります。詳細は、『構成およびインストールガイド』を参照してください。

第 9 章

# Ping の例

この章では、Identrus メッセージングの仕様で定義されている Ping DTD を使用して、前の章で説明した開発サイクルについて手順を追って説明します。サービスを配置する場合には、まず次の手順に従って、サービスを開発します。

1. システム内に送り出すメッセージの構文構造を指定する DTD 定義を作成します。
2. Classgen com.iplanet.trustbase.app.classgen.ClassGen を使って、DTD 定義から java クラスを生成します。
3. Identrus API を使って、サービスの Java コードを書きます。Identrus API は、メッセージ、証明書、鍵、およびデジタル署名の Identrus による処理および確認を支援するためのものです。
4. JDK の JAR ツールを使って、最終的な iPlanet Trustbase Transaction Manager サービスの jar ファイルを構築します。
5. 後述の手順に従って、関連する構成オプションを選択して iPlanet Trustbase Transaction Manager 内にサービスを配置します。
6. iPlanet Trustbase Transaction Manager にサービスを配置したら、サービスを実行することができます。

# DTD 定義の作成

- この例では、Identrus メッセージジングの仕様で定義されている「Ping DTD」を使用します。このため、ping.dtd にある DTD 定義を作成します。ping.dtd の構造と定義については、IT-TCMPD (Identrus TC Messaging Specification) を参照してください。

```
<!--PUBLIC ID for this DTD is: "-//IDENTRUS//PING DTD//EN"-->

<!ENTITY % CoreNetwork.dtd PUBLIC "-//IDENTRUS//CORE NETWORK
INFRASTRUCTURE DTD//EN" "corenetworkinfrastructure.dtd">
%CoreNetwork.dtd;

<!ELEMENT PingRequest (NIB, Signature, CertBundle, PingData)>
<!ATTLIST PingRequest
        id ID #REQUIRED
>
<!ELEMENT PingResponse (NIB, Signature, CertBundle, PingData)>
<!ATTLIST PingResponse
        id ID #REQUIRED
>
<!ELEMENT PingError (NIB, Signature, CertBundle, ErrorInfo)>
<!ATTLIST PingError
        id ID #REQUIRED
>
<!ELEMENT PingData (#PCDATA)>
<!ATTLIST PingData
        id ID #REQUIRED
>

<!ELEMENT ErrorInfo (VendorData*)>
<!ATTLIST ErrorInfo
        id ID #REQUIRED
        errorCode NMTOKEN #REQUIRED
>
<!ELEMENT VendorData (#PCDATA)>
<!ATTLIST VendorData
        id ID #REQUIRED
        dataType CDATA #REQUIRED
>
```

- これらは、次に示す iPlanet Trustbase Transaction Manager に付属の dtd ファイルにリンクされます。

```
XMLDSIG.dtd
corenetworkinfrastucture.dtd
Foundation.dtd
```

---

**注**　すべての dtd ファイルは < インストールディレクトリ >/Trustbase/TTM/current/apidocs にあります。

---

次の手順に従って、ClassGen ツールの実行の準備をします。

- ディレクトリを ../Trustbase/TTM/Scripts に変更します。
- クラスパスを、ClassGen ツールを実行できるように設定します。このためには、標準のシェルで setcp スクリプト (../setcp) をソースします。
- ping という名前の新規のサブディレクトリを作成し、このディレクトリに移動します。
- この新規ディレクトリに、ping.dtd と、3 つのコア Identrus DTD をコピーします。DTD ファイルの名前が適切であることを確認します。ファイル名では大文字と小文字が区別され、DTD ファイル内で完全に同じ名前が使用されていることが必要です。<!ENTITY ....> タグの最後の属性では、DTD に含まれるローカルファイルの名前が定義されます。
- ClassGen ツールにどのオプションを渡すかを決定します。表　9–1 に、この例で使用する各オプションを示します。

表　9–1　**Classgen** オプション

| Ping DTD の ClassGen オプション | |
|---|---|
| -d . | 現在のディレクトリにファイルを作成 |
| -o ping.jar | ping.jar という名前の最終 jar ファイルを作成 |
| -r | すべてのサブ DTD に対し、クラスパスに存在しないすべてのクラスの生成を繰り返す |
| -root PingRequest | DTD 内の PingRequest 要素は、XML ドキュメントの適切なルート要素 |
| -root PingResponse | DTD 内の PingResponse 要素は、XML ドキュメントの適切なルート要素 |

| Ping DTD の ClassGen オプション | |
|---|---|
| -root PingError | DTD 内の PingError 要素は、XML ドキュメントの適切なルート要素 |
| -stub com.iplanet.trustbase.sample.PingService | スタブ Identrus サービスのソースコードを、出力ディレクトリに作成。ファイルを com.iplanet.trustbase.sample にパッケージ |
| "-//IDENTRUS//PING DTD//EN" | Ping DTD のパブリック ID |
| Ping.dtd | クラス生成を開始する DTD |

これにより、次のコマンドが生成されます。

```
java com.iplanet.trustbase.app.classgen.ClassGen -d . -o ping.jar -r
-root PingRequest -root PingResponse -root PingError -stub
com.iplanet.trustbase.sample.PingService "-//IDENTRUS//PING DTD//EN"
ping.dtd
```

このコマンドを実行すると、次のようなことが起こります。

- 画面に多くの警告が表示される ( これらの警告は、DTD で ping.dtd から参照されている特定の要素に対し、クラスが生成されていないという内容のもの。しかし、これらのクラスは、iPlanet Trustbase Transaction Manager のクラスパスにすでに存在するため、再度生成する必要はない )
- 現在のディレクトリに ping.jar というファイルが作成される ( この jar ファイルには、コンパイルされた ping DTD のメッセージクラスが含まれる )
- src というサブディレクトリが作成される ( このサブディレクトリには、生成されたすべてのクラスファイルのソースコードが入れられる )
- 現在のディレクトリに PingService.java というファイルが作成される ( これはスタブ Identrus サービス。次のステップでメッセージ処理コードによって更新することが必要 )

これで JAR ファイルとスタブサービスが生成され、サービス開発の次のステップに進むことができます。

# API

Java コードを作成する前に、iPlanet Trustbase Transaction Manager に付属する Identrus メッセージ処理用のコア API の一部について理解する必要があります。

```
com.iplanet.trustbase.identrus
```

これは、Identrus メッセージ仕様にアクセスすることを可能にするパッケージです。

```
com.iplanet.trustbase.identrus.message
```

これは、Identrus メッセージ処理に必要なルーチンを提供するパッケージです。

```
com.iplanet.trustbase.identrus.security
```

これは、証明書と鍵を処理するパッケージです。

```
com.iplanet.trustbase.util.tree
```

これは、ツリーを検索するパッケージです。

```
com.iplanet.trustbase.xml.dsig
```

これは、XML デジタル署名を生成および確認するパッケージです。メッセージの署名が必須のレベル 1 署名だけである場合は、このパッケージは必要ありません。必須の署名は、iPlanet Trustbase Transaction Manager によって自動的に確認および生成されます。

```
uk.co.jcp.tbase.config
```

これは、構成オブジェクトとのインターフェイスとして機能するパッケージです。

# PingService ソースコード

PingService.java ( 生成されたサービススタブ ) は、抽象 IdentrusService の実装です。ここでは TTM プラットフォームで受信したメッセージが ProcessIdentrusMessage メソッドのパラメータになります。

メッセージの必須署名の確認と、原初ログへの記録は、メッセージが ProcessIdentrusMessage メソッドに渡される前に行われます。

ProcessIdentrusMessage、受信したリクエストに対する応答メッセージを作成するメソッドです。付属の Identrus API を使えば、ソースコードの記述は簡単です。

```
package com.iplanet.trustbase.sample;
import com.iplanet.trustbase.identrus.message.IdentrusMessage;
import uk.co.jcp.tbase.service.ServiceException;
import com.iplanet.trustbase.generated.IDENTRUS.PING_DTD.*;
import com.iplanet.trustbase.generated.IDENTRUS.CORE_NETWORK_INFRA
STRUCTURE_DTD.*;
import com.iplanet.trustbase.identrus.IdentrusService;
import com.iplanet.trustbase.identrus.message.*;
import uk.co.jcp.tbaseimpl.log.error.*;
import uk.co.jcp.tbase.service.*;
/*** Stub Identrus service implementation. */
public class PingService extends IdentrusService
{
        public IdentrusMessage processIdentrusMessage( IdentrusMessage
message )
        {
        if (message instanceof PingRequest)
        {
        // Handle PingRequest
        PingResponse pr = new PingResponse();
        PingData pd = new PingData();
        pd.setPCDATA( "The ping was successful" );
        pr.setPingData( pd );
        try
        {
        NIBAccessor niba = NIBAccessor.getInstance(
message.getNetworkInfoBlk(), "2" );
        pr.setNetworkInfoBlk( niba );
        }
        catch (NIBAccessorException nie)
        {
        ErrorLog.log( new ErrorObject( "IDT0052", nie ));
        }
        return pr;
        }
        if (message instanceof PingResponse)
        {
        // Handle PingResponse
        }
        if (message instanceof PingError)
        {
        // Handle PingError
        }
        return message;
        }
        }
```

この簡単なサービスを作成したら、コンパイルして、サービス JAR に構築できます。

# Identrus サービス JAR の作成

サービスを iPlanet Trustbase Transaction Manager に配置する前の最後のステップは、統合 JAR ファイルの作成です。この JAR ファイルには、すべての生成されたクラスと、手作業で開発した PingService クラスを入れます。

iPlanet Trustbase Transaction Manager サービス jar を構築する手順は次のとおりです。

- 標準的な JDK の jar ツールを使用して、生成した jar ファイルをディレクトリに解凍します。

```
mkdir DumpDirectory
cd DumpDirectory
jar -xvf ../ping.jar
cd ..
```

- 「DumpDirectory」に、メッセージの処理に必要なすべてのクラスをコピーします。これらのクラスすべてで、適切なパッケージ構造が維持されていることを確認します。特に、サービススタブクラスは、クラス生成ツールによって作成されたので、パッケージし直すことはできないことに注意してください。
- 標準的な JDK の jar ツールを使って、アーカイブを再度作成します。

```
jar -cvf ping.jar -C DumpDirectory DumpDirectory/*
```

- これで jar ファイルを配置することができます。

# iPlanet Trustbase Transaction Manager での ping.jar の配置

- ping.jar を ..../Trustbase/TTM/current/deploy ディレクトリに入れます。
- iPlanet Trustbase Transaction Manager に「Administrator」としてログオンします。
- iPlanet Trustbase Transaction Manager に PingService を配置します。「サービス」-「配置」を選択します。
- Ping.jar　はビルダによって自動的に　deploy　ディレクトリに配置されているため、iPlanet Trustbase Transaction Manager によって関連するすべての情報が取得され、次の図に示す「サービス配置」ページの「使用可能なサービス」セクションに表示されます。

- を選択してから 更新 を選択することによって、サービスを配置します。

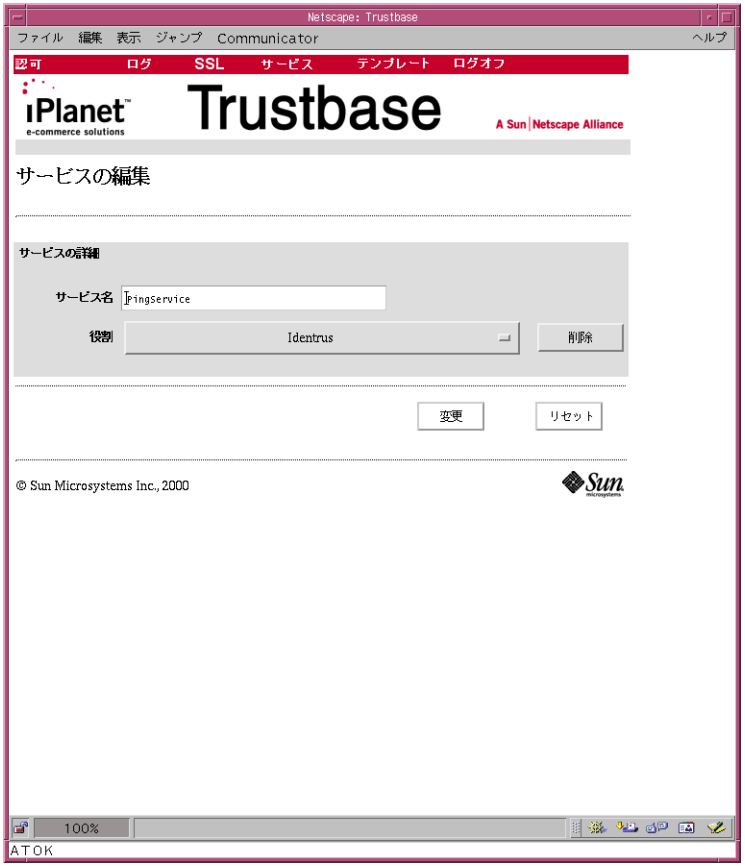
図 9–1 iPlanet Trustbase Transaction Manager での PingService の配置

- サービスは、ビルダツールで jar ファイルを構築する際に、自動的に登録されます。次の図に示すように、構成コンソールから「サービス」－「レジストリ構成」を選択すると、PingService が表示されます。

図　9–2　　サービスレジストリ構成

- サービスを認可する必要がある場合は、「認可」-「サービスの追加」を選択します。次に示すように、サービスには役割を割り当てる必要があります。

図 9–3　　サービスへの役割の割り当て




注　　使用できる役割がない場合は、役割を定義する必要があります。詳細は、『構成およびインストールガイド』を参照してください。

# 用語集および関連サイト

この章には、次の項目があります。

- ソフトウェアプラットフォーム
- プロトコル
  - トランスポートプロトコル
  - セキュリティ関連プロトコル
  - 取引プロトコル
  - メッセージプロトコル
- 用語集
  - セキュリティ関連用語
  - Java 関連用語
  - サーバ定義

# ソフトウェアプラットフォーム

## Solaris 8 および JDK

英語 :

http://www.sun.com/software/solaris/cover/sol8.html

日本語 :

http://www.sun.co.jp/software/solaris/cover/sol8.html

## Java

英語 :

http://www.javasoft.com

日本語 :

http://java.sun.com/products/jdk/1.2/download-ja-docs.html

## iPlanet Application Server 6.0

英語 :

http://www.iplanet.com/products/iplanet_application/home_2_1_1n.html

日本語 :

http://www.iplanet.ne.jp/products/ias6/index.html

## iPlanet Web Server 4.1

英語 :

http://www.iplanet.com/products/iplanet_web_enterprise/home_2_1_1m.html

日本語 :

http://www.iplanet.ne.jp/products/iws4_1/index.html

## Oracle 8i

英語 :

http://www.oracle.com

日本語 :

http://www.oracle.co.jp

## Hardware Security nCipher KeySafe 1.0 および CAFast

英語 :

http://www.ncipher.com

日本語 :

http://www.tel.co.jp

# トランスポートプロトコル

## HTTP

HTTP/1.0 または 1.1 プロトコル :

http://www.w3.org/Protocols/rfc1945/rfc1945.txt

http://www.ietf.org/rfc/rfc1945.txt

## SMTP RFC821

ftp://ftp.isi.edu/in-notes/rfc821.txt http://www.imc.org/ietf-smtp/

# セキュリティ関連プロトコル

## S/MIME バージョン 2 のメッセージ仕様

ftp://ftp.isi.edu/in-notes/rfc2311.txt

http://www.imc.org/ietf-smime

http://www.ietf.org/rfc/rfc2311.txt

## DOMHASH

http://www.ietf.org/rfc/rfc2803.txt

## OCSP

http://www.ietf.org/rfc/rfc2560.txt

## 証明書リクエストと応答

PKCS10 リクエスト RFC2314 :

http://www.ietf.org/rfc.html

PKCS7 応答 RFC2315 :

http://www.ietf.org/rfc.html

# 取引プロトコル

## Identrus

`http://www.identrus.com`

Transaction Coordinator requirements (IT-TCFUNC)

Core messaging specification (IT-TCMPD)

Certificate Status Check Messaging specification (IT-TCCSC)

# メッセージプロトコル

## DOM

http://www.w3.org/TR/REC-DOM-Level-1/

## DTD

http://www.w3.org/XML/1998/06/xmlspec-v20.dtd

## XML

http://www.w3.org/TR/REC-xml

## XML 構文処理の仕様

http://www.w3.org/TR/xmldsig-core

## HTML

HTML 3.2。定義については、以下のサイトを参照

http://www.w3.org/TR/REC-html32.html

# セキュリティ関連用語

| | |
|---|---|
| 3DES | DES に類似。 |
| ASN.1 | 抽象構文記法 (Abstract Syntax Notation One)。 |
| base64 | 65 文字から成る ASCII サブセットによる、デジタル形式の文字表現。 |
| BBS | 乱数発生アルゴリズムの一種。 |
| BER | X509 で使用される基本エンコーディング規則。 |
| CBC モード | ブロック符号化方式によって暗合化される平文の各ブロックと、直前の暗号文ブロックの排他的論理和を次の暗合化の入力とするモード ( 最初の平文ブロックの場合は、初期化ベクトルとの排他的論理和が取られる )。 |
| DER | X509 で使用される特殊なエンコーディング規則。 |
| DH | データを暗号化および複合化するための公開鍵暗合化アルゴリズム。 |
| DSA | デジタル署名アルゴリズム (Digital Signature Algorithm)。 |
| IDEA | Xuejia Lai と James Massey が考案した 64 ビットブロック符号化方式。 |
| MD5 | 任意の長いデータストリームを固定長のダイジェストに変換する、セキュリティ保護されたハッシュ関数。 |
| MIME | 多目的インターネットメール拡張仕様 (MultiPURPOSE Internet Mail Extension)。 |
| OSI | 開放型システム間相互接続 (Open Systems Inter-Connection)。 |
| PBE | パスワードに基づく暗合化 (Password based encryption)。 |
| PEM | プライバシー強化メール (Privacy enhanced mail)。 |
| RC2、RC4 | RSA Data Security, Inc が独自に開発したバルク符号化方式。RC2 はブロック符号化方式、RC4 はストリーム符号化方式。 |
| RFC | 信頼性のある一連の提案書形式のドキュメント（Request for Comments）。 |
| RSA | 広く用いられている公開鍵アルゴリズム。暗合化またはデジタル署名に使用できる。 |
| Salt | エクスポート暗号化鍵が事前計算攻撃を撃退するために使用される秘密ではないランダムデータ。 |
| SHA | セキュアハッシュアルゴリズム (Secure Hash Algorithm)。20 バイトの数を出力する。FIPS PUB 180-1 で定義されている。 |
| SSL | セキュアソケットレイヤ (Secure sockets layer)。 |
| TSL | トランスポートセキュリティレイヤ (Transport Security Layer)。 |
| X509 | ASN.1 BER、ASN.1 DER、および base64 に基づく認証フレームワーク。 |
| X690 | ASN.1 仕様。 |
| アプリケーションプロトコル | 通常はトランスポート層 (TCP/IP など) の上に直接重なっているプロトコル。HTTP、TELNET、FTP、SMTP などがある。 |

| | |
|---|---|
| クライアント | サーバへの接続を開始するアプリケーションエンティティ。 |
| クライアント書き込みキー | クライアントが書き込んだデータを暗号化するための鍵。 |
| クライアント書き込み MAC 秘密 | クライアントが書き込んだデータを認証するための秘密データ。 |
| 公開鍵暗号化法 | 2 つの鍵による符号化方式を用いる暗合化技術。公開鍵を使って暗合化されたメッセージは、その公開鍵と対になった共有鍵だけが復号化できる。逆に、共有鍵を使って暗号化されたメッセージは、公開鍵を使って復号化できる。 |
| 公開鍵インフラストラクチャ (PKI) | オンラインでのやり取りをサポートするためのプロトコルを定義する。 |
| サーバ | クライアントの接続リクエストに応答するアプリケーションエンティティ。受動的であり、クライアントからのリクエストを待つ。 |
| サーバ書き込みキー | サーバが書き込んだデータを暗号化するための鍵。 |
| サーバ書き込み MAC 秘密 | サーバが書き込んだデータを認証するための秘密データ。 |
| 証明書 | X.509 プロトコル (ISO 認証フレームワーク ) の一部。認証局によって割り当てられる。エンティティを識別するためのものであり、そのエンティティの公開鍵を提供することもある。 |
| 証明書破棄リスト (CRL) | 有効期限が切れてはいないが、無効化された証明書のリスト。 |
| 初期化ベクトル (VI) | ブロック符号化方式が CBC モードで使用されるときに、暗号化に優先して最初の平文ブロックとの排他的論理和が取られる。 |
| ストリーム符号化方式 | 鍵を強固に暗号化された鍵ストリームに変換し、平文の排他的論理和を取る暗号化アルゴリズム。 |
| セッション | SSL セッションとはクライアントとサーバのつながりのこと。セッションは、ハンドシェイクプロトコルによって作成され、一連の暗合化セキュリティパラメータを定義する。複数の接続がこれらのパラメータを共有することも可能。セッションの使用によって、接続ごとに新しいセキュリティパラメータをネゴシエートする必要がなくなるため、コストを節約できる。 |
| セッション識別子 | サーバーによって作成される特定のセッションを識別するための値。 |
| 接続 | OSI 階層モデルのトランスポートにあたるもので、適切なタイプのサービスを提供する。SSL ではピアツーピア関係を持つ。接続は一時的なものであり、1 つのセッションにつき 1 つずつ確立される。 |
| 対称符号化方式 | バルク符号化方式を参照。 |
| 単方向ハッシュ関数 | 任意長のデータを固定長のハッシュへ単方向変換する。逆変換や衝突の検知がむずかしい。ハッシュ関数の例には、MD5 や SHA などがある。 |

**データ暗号化規格 (DES)** 広く使用されている対称暗号化アルゴリズム。ブロック符号化方式の一種。

**デジタル署名** 認証可能で偽造または拒否されにくいデータの署名を作成するために、公開鍵暗合化法と単方向ハッシュ関数を利用したもの。

**デジタル署名標準 (DSS)** 米国の国立標準技術研究所 (National Institute of Standards and Technology) が認可した、デジタル署名アルゴリズムを含むデジタル署名の標準。1994 年 5 月に米国商務省が発行した NIST FIPS PUB 186「Digital Signature Standard」で定義されている。

**認証** あるエンティティが他のエンティティを識別する能力。

**バルク符号化方式** 大量のデータを暗号化するために使用される対称暗号化アルゴリズム。

**ハンドシェイク** トランザクションのパラメータを決定する、クライアントとサーバの初期ネゴシエーション。

**非対称符号化方式** 公開鍵暗号化法を参照。

**ブロック符号化方式** ブロックと呼ばれるデータ単位ごとに平文を処理するアルゴリズム。通常、1 ブロックは 64 ビットである。

**マスターシークレット** 暗号化鍵、MAC 秘密、初期化ベクトルを作成するための安全な秘密データ。

**メッセージダイジェスト** ダイジェストアルゴリズムが、単方向ハッシュ関数を通じて、任意長のデータを短い固定長の一意の表現に変換する。メッセージダイジェストは、デジタル署名の作成やデータの完全性検査に広く用いられている。

**メッセージ認証コード (MAC)** メッセージおよび機密データから計算された単方向ハッシュ。メッセージの変更を検知するために使用される。

# Java 関連用語

**API ( アプリケーション プログラミング インターフェイス )** アプリケーションを作成しているプログラマがクラスおよびオブジェクトの動作および状態にアクセスするための仕様。

**Bean** 再使用可能なソフトウェアコンポーネント。複数を組み合わせてアプリケーションを作成することも可能。

**Classpath** Java 仮想マシンおよびその他の Java アプリケーション (JDK1.1.X\bin ディレクトリ内の Java ツールなど ) に対して、ユーザ定義のクラスライブラリを含むクラスライブラリの呼び出し元を示す環境変数。

**Codebase** <APPLET> タグ内の code 属性とともに、メインアプレットのクラスファイルの呼び出し元を指定する。code はファイル名を指定し、codebase はそのファイルを含むディレクトリの URL を指定する。

**GUI** グラフィカルユーザインターフェイス (Graphical User Interface)。プログラムのインターフェイスにグラフィックを多用し、ユーザがキーボードやマウスの使用によってより簡単にそのプログラムを使用できるようにする手法。

**HotJava™ ブラウザ** Sun Microsystems が開発した Web ブラウザ。Java 言語によって記述されており、簡単にカスタマイズできる。

**HTML ( ハイパーテキスト マークアップ言語 )** インターネット上で使用可能なハイパーテキスト文書を記述するためのファイル形式。SGML を基盤としている。構造が単純で、イメージ、サウンド、ビデオストリーム、およびフォームフィールドを埋め込んだり、単純なテキストレイアウトを指定したりすることができる。他のオブジェクトへのリンクを指定するには、URL を使用する。

**HTTP ( ハイパーテキスト 転送プロトコル )** リモートホストからハイパーテキストオブジェクトを取得するためのインターネットプロトコル。TCP/IP を基盤としている。関連項目「TCP/IP」

**IDL (Java インターフェイス 定義言語 )** CORBA ( 共通オブジェクトリクエストブローカアーキテクチャ、Common Object Request Broker Architecture) との標準ベースの相互運用および接続を可能にする Java API。

**IP ( インターネット プロトコル )** インターネットの基本プロトコル。ホスト間での個々のパケットの配信を可能にする。ただし、パケットが適切に配信されるかどうか、配信にどれほどの時間がかかるか、複数のパケットが送信順序どおりに受信されるかどうか、などを制御する力は持たない。これらの制御を受け持つのは IP の上位層にあるプロトコル。関連項目「TCP/IP」

**JAR ファイル形式** JAR (Java Archive) は、多数のファイルを 1 つにまとめるための、プラットフォームから独立したファイル形式。この形式を使用すると、複数の Java アプレットおよびアプレットに必要なコンポーネント (.class ファイル、イメージ、サウンドなどのリソースファイル ) を 1 つの JAR ファイルにまとめ、1 度の HTTP トランザクションでブラウザにダウンロードできる。また、ファイル圧縮およびデジタル署名もサポートされている。

| | |
|---|---|
| JavaBeans™ | プラットフォームから独立した、移植性のある再使用可能なコンポーネントモデル。 |
| Java Database Connectivity (JDBC™) | Java プログラムがさまざまな種類のデータベースにアクセスするための業界標準。SQL 文を用いてデータベースにアクセスするためのコールレベル API を提供する。 |
| Java™ Development Kit (JDK™) | Java 言語を使用してアプレットやアプリケーションを作成するためのソフトウェア開発環境。 |
| Java™ Foundation Class (JFC) | Abstract Windowing Toolkit (AWT) に GUI クラスライブラリを追加して機能を拡張するための規格。 |
| Java Remote Method Invocation (RMI) | Java アプリケーションが同じホスト上または異なるホスト上の他の Java 仮想マシンからリモート Java オブジェクトのメソッドを呼び出すことが可能な分散オブジェクトモデル。 |
| Java Runtime Environment (JRE) | エンドユーザおよび JRE を再配布したい開発者のための Java™ Development Kit サブセット。Java 仮想マシン、Java コアクラス、およびサポートファイルから構成されている。 |
| JavaScript™ | ブラウザおよび Web サーバで使用される Web スクリプト言語。誤解を招くことが多いが、Java とはあまり関係がない。他のスクリプト言語と同様、この言語を使用する主な目的は、コンポーネント同士を結び付ける、あるいはユーザの入力を受け入れること。 |
| Java™ 仮想マシン (JVM) | Java Runtime Environment の一部。Java バイトコードを解釈する。 |
| Java プラットフォーム | Java™ 仮想マシンおよび Java コアクラスから構成されるプラットフォーム。100% Pure Java プログラムに、配下のオペレーティングシステムの種類を問わず動作できるユニフォームプログラミングインターフェイスを提供する。 |
| JDK™ (Java™ Development Kit) | Java 言語を使用してアプレットやアプリケーションを作成するためのソフトウェア開発環境。 |
| JFC (Java™ Foundation Class) | Abstract Windowing Toolkit (AWT) に GUI クラスライブラリを追加して機能を拡張するための規格。 |
| JRE (Java Runtime Environment) | エンドユーザおよび JRE を再配布したい開発者のための Java™ Development Kit サブセット。Java 仮想マシン、Java コアクラス、およびサポートファイルから構成されている。 |
| JVM (Java 仮想マシン ) | Java Runtime Environment の一部。Java バイトコードを解釈する。 |
| NCSA | National Center for Supercomputer Applications の略。 |

**RPC** リモート手続き呼び出し (Remote Procedure Call)。ネットワークパケットをリモートホストに送信することによって、通常のような手続き呼び出し ( またはメソッド呼び出し ) を実行すること。

**TCP/IP** IP を基盤とする転送制御プロトコル。ホスト間でデータストリームが適切に配信されるようにするためのインターネットプロトコルである。関連項目「IP ( インターネット プロトコル )」

**Unicode** ISO 10646 で定義されている 16 ビット文字セット。Java ソースはすべて Unicode で記述される。

**URL** ユニフォームリソースロケータ (Uniform Resource Locator)。WWW 上の各種情報資源の場所を特定するための標準規格。「プロトコル :// ホスト / ローカル情報」の形式で記述する。「プロトコル」はオブジェクトの取得に使用するプロトコル (HTTP や FTP など ) を示す。「ホスト」はオブジェクトが存在するホストのインターネット名を示す。「ローカル情報」はリモートホスト上のプロトコルハンドラに渡される文字列 ( 主にファイル名 ) を示す。

**アプレット** HotJava™ や Netscape Navigator™ などの Java 対応 Web ブラウザ内で動作できるように Java 言語で記述されたプログラム。

**インスタンス** 特定のクラスのオブジェクト。Java プログラムでは、クラスのインスタンスは新しい演算子の後ろにクラス名を指定することによって生成する。

**インターフェイス** Java では、複数のクラスが実装できるメソッドグループのこと。これらのクラスがクラス階層のどの位置にあるかは無関係。

**拡張する** あるクラス ( クラス X) が別のクラス ( クラス Y) にフィールドまたはメソッドを追加したり、クラス Y のメソッドを無効化したりすることによって、機能を追加すること。インターフェイスの場合は、メソッドを追加することによって他のインターフェイスを拡張する。クラス X はクラス Y のサブクラスと呼ばれる。関連項目「派生する」

**仮想マシン** ソフトウェアまたはハードウェアにさまざまな方法で実装可能な処理デバイスの抽象仕様。仮想マシンの命令セットに対するコンパイルは、マイクロプロセッサの命令セットに対するコンパイルと同様に行う。Java 仮想マシンは、バイトコード命令セット、レジスタ、スタック、ガベージコレクションのヒープ、およびメソッドを保存するための領域から成り立っている。

**危険領域** 複数のスレッドがアクセスできるが、同時に使用することはできないリソース ( 特定のインスタンス変数など ) を含むコードセグメント。

**クラス** Java では、特定の種類のオブジェクトを実装する方法を定義するタイプ。インスタンスや、クラス変数およびメソッドは、クラス定義によって定義される。また、そのクラスが実装するインターフェイスおよびすぐ上のスーパークラスもクラス定義によって指定される。スーパークラスが指定されていない場合は、Object がスーパークラスとして見なされる。

**コアクラス** Java プラットフォームの標準的要素であるパブリッククラス ( またはインターフェイス )。最低限、Java プラットフォームが動作するすべてのオペレーティングシステムで使用可能なように意図されている。100% Pure Java プログラムは、コアクラスにだけ依存するため、オペレーティングシステムの種類に関わらず機能できる。

**サンドボックス** アプリケーションの一部として動作するセキュリティマネージャ、Java 仮想マシンが備えるセキュリティ対策、および Java 言語そのものまでを含む、複数の協働システムコンポーネントから構成されている。危険視されるアプリケーションによるシステムリソースへのアクセスを防止するための機構。

**ジャストインタイム (JIT) コンパイラ** Java プログラム実行時にバイトコードをすべてネイティブマシンコードに置き換えるコンパイラ。JIT を用いると、Java 仮想マシンが解釈するコードの実行速度が向上する。

**シンクライアント** ローカルシステム管理機能がない簡易オペレーティングシステムを使用するシステム。ネットワークを通じて配信される Java アプリケーションを実行する。

**スレッド** プログラム実行の基本単位。プロセスによっては、イベント待ちや、プログラムが次の動作に移るために完了している必要のない時間のかかるジョブなどの、それぞれ異なるジョブを実行している複数のスレッドを同時進行させることもある。スレッドは、ジョブを完了すると、停止または破棄される。関連項目「プロセス」

**セキュアソケットレイヤ (SSL)** Web ブラウザとサーバの間で送受信するデータを暗号化することによってプライバシーを保護するプロトコル。その他のエンティティに適用することも可能。

**抽象クラス** 1 つ以上の抽象メソッドを含み、それゆえにインスタンス化できないクラス。他のクラスが抽象メソッドを実装することによって拡張および具体化できるように定義されている。

**抽象メソッド** 実装されないメソッド。

**同期** Java では、複数のコードが同時にメソッドやコードブロックを実行しないようにすること。

**派生する** あるクラス ( クラス X ) が別のクラス ( クラス Y) を拡張している場合、「クラス X はクラス Y から派生している」などのように表現する。関連項目「拡張する」

**パッケージ** タイプのグループ。package キーワードによって宣言される。

**反対** もはや使用を推奨されておらず、将来のバージョンでは存在しなくなる可能性があるクラス、インターフェイス、コンストラクタ、メソッド、またはフィールドを指す。

**不可分** どのような場合にも、中断したり不完全な状態で残しておいたりできない処理を指す。

**プロセス** 1 つ以上のスレッドを含む仮想アドレス空間。

**マルチスレッド** コードの各部分が同時に実行されるように設計されたプログラムを指す。関連項目「スレッド」

**ラッパー** カプセル化および委託によって、他のオブジェクトのインターフェイスまたは動作を変化させるオブジェクト。

**例外** プログラム実行中にそのプログラムの正常な動作を妨害するイベント。通常はエラーのこと。Java は、try、catch、および throw キーワードによって例外をサポートする。関連項目「例外ハンドラ」

**例外ハンドラ** 特定のタイプの例外を処理するコードブロック。プログラムが復帰可能なエラーによって例外が発生した場合は、プログラムは例外ハンドラ実行後に動作を再開できる。

# サーバ定義

| | |
|---|---|
| API | アプリケーションプログラミングインターフェイス (Application Programming Interface)。 |
| ASP | Active Server Pages の略。 |
| CORBA | 共通オブジェクトリクエストブローカアーキテクチャ (Common Object Request Broker Architecture)。 |
| CSS | カスケーディングスタイルシート (Cascading Stylesheet)。 |
| DMZ | 非武装地帯 (De-militarised Zone)。 |
| DOM | ドメインオブジェクトモデル (Domain Object Model)。 |
| DTD | データ型定義 (Data Type Definition) または文書型定義 (Document Type Definition)。 |
| EJB | Enterprise Java Bean の略。 |
| HSM | ハードウェアセキュリティモジュール (Hardware Security Module)。 |
| HTML | ハイパーテキストマークアップ言語 (HyperText Markup Language)。 |
| IDL | インターフェイス定義言語 (Interface Definition Language)。 |
| JDBC | Java Database Connectivity の略。 |
| JWS | Java Web Server の略。 |
| lastService | 最後に実行されたサービスの名前を含む属性。 |
| MessageType | メッセージのタイプを保持する属性で、メッセージ内に含まれている。メッセージのタイプは、ユーザによって外部で定義される。 |
| NAS | Netscape Application Server の略。 |
| NSK | Non Stop Kernel の略。 |
| OAS | Oracle Application Server の略。 |
| PKI | 公開鍵インフラストラクチャ (Public Key Infrastructure)。 |
| RMI | リモートメソッド呼び出し (Remote Method Invocation)。 |
| SP | サービスプロバイダ (Service Provider)。 |
| TISS | トランスポート独立スタブサービス (Transport Independent Stub Service)。 |
| URL | ユニフォームリソースロケータ (Uniform Resource Locator)。 |
| X500 | ディレクトリサービスに関する一連の開放標準。ISO および X500 の各サイト (`http://www.iso.ch` および `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x500.html`) にアクセスし、標準として定義されている国名コードなどを比較してみるとよい。 |
| XML | 拡張可能マークアップ言語 (Extensible Markup Language)。 |
| XSL | 拡張可能スタイルシート言語 (Extensible Stylesheet Language)。 |

**環境** 特定のメッセージに関連付けられている一連のコンテキスト。

**規則** 規則は、規則名、前提条件、およびディレクティブの 3 つの要素から成る。前提条件が true の場合はディレクティブが実行される。

**規則セット** 与えられたタスクをアーカイブするために、メッセージを 1 つまたは複数のサービスに配信する規則の集まり。

**規則名** 規則にはそれぞれ名前があり、規則セットのコンテキスト内で名前によって参照される。

**構成オブジェクト** サービスが使用できる永続的な構成データを保持するもの。

**構成可能エンティティ** 構成オブジェクトおよび構成マネージャを使用する、サービスまたはコンポーネント。

**構成サービス** 構成オブジェクトに読み取り / 書き込みインターフェイスを実装するサービス。

**コネクタ** 接続マネージャのメインインターフェイス。iPlanet Trustbase Transaction Manager の外部にリクエストを送る。リクエストを含む iPlanet Trustbase Transaction Manager メッセージと、リクエストの終端を表す目的地オブジェクトを取り込む。

**コンテキスト** トランザクションの現状の記録を維持する。

**コンテキストディレクティブ** 規則セットを構成するアクションコンポーネント。

**サービス** ビジネスロジックを実装するオブジェクト。サービスはユーザによって書き込まれる。

**サービスレジストリ** サービスが登録されているレジストリ。サービス名からそのサービスを実装するクラスをルックアップしたり、クラスからサービス名をルックアップしたりするために使用される。

**状態** ある瞬間にあるタスクに関連付けられている属性の集まり。

**スケルトン** CORBA および RMI 下の分散型オブジェクトのサーバ / リモート部分。スタブによって呼び出される。関連項目：「スタブ」

**スタブ** CORBA や RMI などのメカニズムを使用する分散型オブジェクトのクライアント部分。関連オブジェクトの実装がローカルシステム外にあることを隠すように設計されている。関連項目：「スケルトン」

**セッション** ユーザがある期間を通して実行しているタスクをすべて含むコンテナ。

**接続マネージャ** iPlanet Trustbase Transaction Manager が外部エンティティと通信するときのプロセスのこと。接続マネージャは、プロトコルマップ、プロトコルアナライザ、ハンドラ、メッセージリーダ、およびメッセージライタを使用してこのタスクを行う。

**前提条件** 論理式の一種。前提条件が true の場合に限り、対応するディレクティブが実行される。前提条件は属性およびその値によって表される。前提条件には、特定の名前を持つ属性だけが存在するタイプと、特定の名前と特定の値を持つ属性が存在するタイプがある。

**属性** 次に実行すべきアクションを判断するために、ルータが規則セットに関連して使用する文字列値。メッセージに関する情報を使ってコンテキストを提供する。

**タスク** ビジネスレベルでの作業単位。タスクがどのように実行されるかは、規則セットによって定義される。

**ディレクティブ** 前提条件が true の場合に実行される規則の「アクション」部分。

**ビジネスロジック** システムにおける「ユーザ」コード。口座からの引き落としや残金の算出などのタスクを実行するときに実行されるロジックのこと。

**ホスト環境アダプタ** Web サーバやアプリケーションサーバなどのホストと iPlanet Trustbase Transaction Manager との間にインターフェイスを形成するもの。

**メッセージ** ユーザからのリクエストやサーバからの応答の内部表現。システム内で配信される。

**メッセージアナライザ** トランスポートおよびメッセージ内容の外部形式に基づき、特定のメッセージに対して使用すべきメッセージリーダまたはライタを識別するためのロジックを提供するもの。

**メッセージライタ** 処理されたメッセージオブジェクトを、クライアントが要求するプレゼンテーションプロトコルに変換し、プロトコルアナライザが提供する出力ストリームに書き込むもの。

**メッセージリーダ** InputStream に残っているメッセージの内容を解析し、メッセージの内容フィールドに入れるもの。アプリケーションの一部として特定のメッセージタイプ情報を備えている場合もあれば、一般目的のリーダとして一般的なメッセージ形式情報を備えている場合もある。

**メッセージレジストリ** tbase.properties ファイル内のセクション。メッセージタイプとそのメッセージを処理するためのメッセージリーダ / メッセージライタとの間のマッピングを提供する。

**メッセージロガー** 未処理の原初形式で、着信および送信メッセージのログをとる。ログは、logManager またはバックエンドデータベースエンジンを通じてクエリおよび操作できる。

**メッセージログマネージャ** メッセージロガーへのアクセスをインスタンス化および許可する。アクセスは、メッセージロガーが登録されている MIME タイプに基づいて行われる。

**目的地** コネクタが外部にリクエストを送るときの宛先。目的地の実装、および目的地を ProtocolDescriptor に変換する ProtocolMap は、アプリケーションによって指定される。これにより、コネクタが実際の接続を確立し、管理できるようになる。

**パブリック規則セットリポジトリ** ユーザ定義の規則セットの集まり。ユーザによる構成が可能。このリポジトリ内の規則よりもプライベート規則セットリポジトリ内の規則の方が優先される。

**プライベート規則セットリポジトリ** iPlanet Trustbase Transaction Manager に含まれる規則セットの集まり。このリポジトリによって、構成サービスなどの内蔵サービスが機能する。

**プロトコルアナライザ** 特定のメッセージタイプに対して使用すべきプロトコルハンドラを識別するためのロジックを提供するもの。

**プロトコル記述子** 終端の定義、接続に関する URL 形式のトランスポートプロトコルとプレゼンテーションプロトコル、および送信メッセージの形式を保持するもの。また、MIME タイプとして目的地を実装する。目的地のアドレスを直接指定するために SimpleProtocolMap と併用することも可能。

プロトコルハンドラ　メッセージヘッダからメッセージタイプとコンテキスト ID を抽出するコンポーネント。通常、1 つのメッセージクラス ( 例 : iPlanet Trustbase Transaction Manager メッセージング、OFX など ) にはプロトコルハンドラが 1 つある。プロトコルハンドラによって、適切なプロトコルがメッセージアナライザにルーティングされる。

プロトコルマップ　ProtocolMap 実装は、コネクタが接続の確立に使用できる URL および MIME タイプにアプリケーションの目的地実装をマップするために、アプリケーションによって指定される。

プロトコルマップマネージャ　一連の ProtocolMap 実装を管理するもの。特定の目的地実装を ProtocolDescriptor に変換するために、適切な ProtocolMap を選択する。

役割　システムが認識できる特定の属性の名前。属性セットではない。システムが認識できる属性には、次のようなものがある。

lastService - 最後に実行されたサービスの名前。
messageType - メッセージのタイプ。
role - ユーザがどのような立場でシステムを使用しているかを表す文字列

( 例 : role - operator、role =administrator など )。

ルータ　コードの変更を必要としないセキュリティ保護された方法によって、サービス実行時にそのサービスを構造化および秩序化するためのメカニズムを提供するもの。

# 索引

## 数字



## A



## B



## C

# D



# E



# G



# H



# I



# J

## M



## N



## O



## P



## R



## S

## T



## U



## X



## あ



## い



## え



## か

## き



## く



## け



## こ



## さ

## し



## す



## せ



## そ



## た



## ち



## て

## と



## に



## は



## ひ



## ふ



## へ



## ほ