BEA Tuxedo®

CORBA 技術情報

BEA Tuxedo 8.1
2003 年 1 月

Copyright

Restricted Rights Legend

Trademarks or Service Marks

目次

4. Client Data Caching デザイン・パターン

このマニュアルについて

このマニュアルでは、以下の技術情報について説明します。

- 「第 1 章 CORBA プログラミング・モデル」では、CORBA プログラミング・モデルについて説明します。
- 「第 2 章 CORBA オブジェクト」では、CORBA とは何かについて、また BEA Tuxedo の情報で使用されるオブジェクト用語について説明します。
- 「第 3 章 Process-Entity デザイン・パターン」では、Process-Entity デザイン・パターンについて説明します。
- 「第 4 章 Client Data Caching デザイン・パターン」では、CORBA Client Data Caching デザイン・パターンについて説明します。

対象読者

このマニュアルは、安全でスケーラブルなトランザクション・ベース処理を企業からイントラネットおよびインターネットへ拡張することに関心を持ち、CORBA プログラミングの基礎知識を必要としているシステム構築担当者およびプログラマを対象としています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、Web サイト上にあります。また、マニュアルの CD-ROM にも収められています。BEA Tuxedo この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo マニュアルのホーム・ページにある [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe 社の Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

CORBA、BEA Tuxedo、分散オブジェクト・コンピューティング、トランザクション処理、C++ プログラミング、および Java プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで docsupport-jp@bea.com までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、BEA Tuxedo 8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。
イタリック体	強調またはマニュアルのタイトルを示します。

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力するテキストも等幅テキストで表示します。
	例 :
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な語を示します。 例 : `void` **`commit`** `( )`
斜体の等幅テキスト	コード内の変数を示します。 例 : `String` *`expr`*

規則	項目
大文字のテキスト	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
{ }	構文の行で、選択肢の組み合わせを示します。かっこは入力しません。
[ ]	構文の行で、オプション項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	構文の行で、相互に排他的な選択肢の区切りとして使います。記号は入力しません。
...	コマンド・ラインで、以下のいずれかの場合を示します。 ■ コマンド・ラインで、同じ引数を繰り返し使用できることを示します。 ■ 文で、追加のオプション引数が省略されていることを示します。 ■ 追加のパラメータ、値、またはその他の情報を入力できることを示します。 記号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。記号は入力しません。

1 CORBA プログラミング・モデル

CORBA は、オブジェクト・ベースの分散アプリケーションを作成するための仕様です。CORBA のアーキテクチャと仕様は、Object Management Group (OMG) によって開発されました。OMG は、数百社の情報システム・ベンダで構成されるコンソーシアムです。CORBA の目的は、分散ソフトウェア・アプリケーションをビルドおよび統合するオブジェクト指向の手法を促進することです。

CORBA 仕様では、分散アプリケーションをビルドするためのオープンで一貫性のあるモデルが提供されます。そのために、CORBA 仕様では次の事項が定義されています。

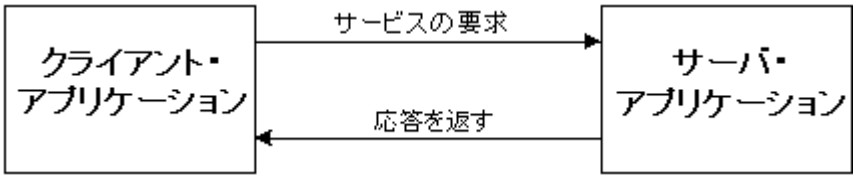
- 分散アプリケーションをビルドするためのオブジェクト・モデル
- クライアント・アプリケーションおよびサーバ・アプリケーションで使用される共通のアプリケーション・プログラミング・オブジェクトのセット
- 分散アプリケーションの開発で使用されるオブジェクトのインターフェイスを記述する構文
- 複数のプログラミング言語で記述されたアプリケーションによる使用のサポート

CORBA 仕様では、CORBA のインプリメンテーションを開発する方法が説明されています。また、アプリケーションの開発で使用するプログラミング言語のバインディングも記述されています。

CORBA アーキテクチャを使用する利点を説明するために、この節では初期のクライアント / サーバ・アプリケーション開発手法と CORBA の開発手法を比較します。

クライアント / サーバ開発の CORBA 以前の手法

クライアント / サーバ・コンピューティングは、ネットワーク接続された複数のマシンで処理を分散し、マシンのリソースをより効率的に利用できるようにするアプリケーション開発方法です。クライアント / サーバ・コンピューティングのアプリケーションは、クライアント・アプリケーションとサーバ・アプリケーションの 2 つのパートで構成されます。通常、それらの 2 つのアプリケーションは、次の図で示されているようにネットワークで接続された異なるマシンで動作します。



クライアント・アプリケーションは情報またはサービスを要求し、通常は結果を表示する手段をユーザに提供します。サーバ・アプリケーションは、1 つまたは複数のクライアント・アプリケーションの要求を満たし、通常は計算集中型の機能を実行します。

クライアント / サーバ・モデルの主要な利点は次のとおりです。

- 計算機能が最も適切なマシン・システムで実行されます。
- 複数のサーバにアプリケーションの処理負荷を分散させることができます。
- 多くのクライアント・アプリケーションでサーバ・アプリケーションを共有できます。

たとえば、デスクトップ・システムは、情報を表示するための使いやすいグラフィカルな環境を多くのビジネス・ユーザに提供します。しかし、デスクトップ・システムはディスク領域とメモリが制限される場合があり、シングル・ユーザ・システムであるのが普通です。より大きく、より強力なマシン・システムの方が、計算集中型の機能を実行し、マルチ・ユーザ・アクセスと共有データベース・アクセスを実現するには適しています。

したがって、通常はより規模の大きいシステムでアプリケーションのサーバ部分が実行されます。このようにして、分散したデスクトップ・システムとネットワーク接続されたサーバにより、分散クライアント / サーバ・アプリケーションをデプロイする完璧なコンピューティング環境が実現されます。

CORBA 以外のクライアント / サーバ手法でも異機種間ネットワークで処理を分散する手段が提供されますが、その手法には次の欠点があります。

- 通信については、クライアント・アプリケーションで、必要なネットワーク・プロトコル情報を含めてサーバ・アプリケーションへのアクセス方法が理解されていなければなりません。

  クライアント / サーバ・アプリケーションでは、1 つのネットワーク・プロトコルを使用する場合と、複数の異なるプロトコルを使用する場合があります。複数のプロトコルが使用される場合、アプリケーションではネットワークごとにプロトコル固有のコードを論理的に反復しなければなりません。

- アプリケーションでは、異なるデータ形式が使用されるマシンと統合された場合に、データ形式の変換を処理しなければなりません。

  たとえば、一部のマシンでは整数値が最下位バイト・アドレスから最上位へ向かって読み取られ (リトル・エンディアン)、ほかのマシンでは逆に最上位バイトから読み取られます (ビッグ・エンディアン)。また、一部のマシン・システムでは、浮動小数点数またはテキスト文字列で異なる形式が使用される場合もあります。アプリケーションから異なるデータ形式が使用されるマシンにデータが送信される場合で、そのアプリケーションによってデータが変換されないときには、データは間違って解釈されます。

  ネットワーク経由でデータを転送し、そのデータをターゲット・システムの適切な表現に変換することを、データ・マーシャリング**と呼びます。**多くの非 CORBA クライアント / サーバ・モデルでは、アプリケーションですべてのデータ・マーシャリングを実行しなければなりません。データ・マーシャリングでは、アプリケーションでネットワークとオペ

レーティング・システムの機能を利用してデータをマシン間で移動する必要があります。また、データ形式の変換を実行して、送信時と同じようにデータが読み取られるようにすることも必要です。

- アプリケーションの拡張に柔軟性がありません。

  CORBA 以外のクライアント / サーバ手法では、クライアント・アプリケーションとサーバ・アプリケーションが強く結び付けられます。したがって、クライアント・アプリケーションかサーバ・アプリケーションで変更があった場合には、プログラマはインターフェイス、ネットワーク・アドレス、およびネットワーク・トランスポートを変更する必要があります。また、クライアント・アプリケーションとサーバ・アプリケーションを別のネットワーク・インターフェイスをサポートするマシンに移植する場合、プログラマはそれらのアプリケーションの新しいネットワーク・インターフェイスを作成しなければなりません。

クライアント / サーバ開発の CORBA 手法

CORBA モデルでは、分散アプリケーションをもっと柔軟に開発することができます。

- アプリケーションのクライアント部分とサーバ部分がはっきりと分離されます。

  CORBA クライアント・アプリケーションでは処理を要求する方法のみが理解され、CORBA サーバ・アプリケーションではクライアントから要求されたタスクを遂行する方法のみが理解されています。このような分離のおかげで、クライアント・アプリケーションがサーバ・アプリケーションにタスクの遂行を要求する方法に影響を与えることなく、サーバでタスクを遂行する方法を変更できます。

- アプリケーションが、オペレーションと呼ばれる特定のタスクを実行できる複数のオブジェクトに論理的に分割されます。

  CORBA は、分散オブジェクト・コンピューティング・モデルに基づいています。このモデルでは、分散コンピューティング (クライアントと

サーバ) の概念とオブジェクト指向コンピューティング (オブジェクトとオペレーションをベースとする) の概念が結合されます。

オブジェクト指向コンピューティングにおいて、オブジェクトはアプリケーションを構成するエンティティであり、オペレーションはそれらのオブジェクトでサーバが実行できるタスクです。たとえば銀行業務アプリケーションでは、顧客口座のオブジェクトと、預け入れ、引き出し、および残高照会のオペレーションを利用できます。

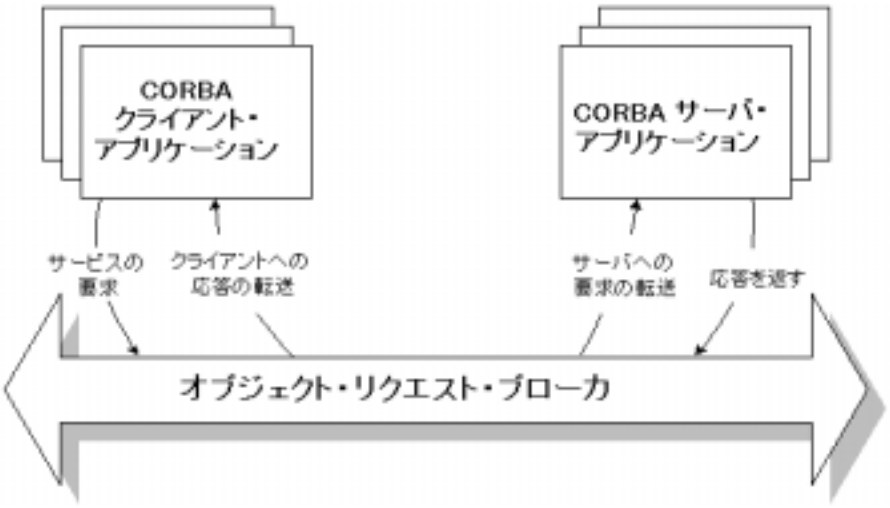
- リモート・マシンまたはローカル・マシンのアプリケーションとデータをやり取りするためのデータ・マーシャリングが提供されます。

  たとえば、CORBA モデルでは必要に応じてビッグ・エンディアンまたはリトル・エンディアン用に自動的に形式変換が行われます。データ・マーシャリングの説明については、前節を参照してください。

- ネットワーク・プロトコル・インターフェイスがアプリケーションから隠蔽されます。

  CORBA モデルではすべてのネットワーク・インターフェイスが処理されます。アプリケーションで認識されるのはオブジェクトだけです。アプリケーションは複数の異なるマシンで動作できます。その理由は、すべてのネットワーク・インターフェイス・コードが ORB によって処理され、異なるネットワーク・プロトコルをサポートするマシンに後からデプロイされる場合でもアプリケーションではネットワーク関連のいかなる変更も必要としないからです。

CORBA モデルのクライアント・アプリケーションでは、情報の供給源やその位置について直接知らなくても、サーバ・アプリケーションに要求を送信して応答を受信することができます。CORBA 環境では、通信のためのネットワークやオペレーティング・システムの情報はアプリケーションで必要としません。代わりに、クライアント・アプリケーションとサーバ・アプリケーションはオブジェクト・リクエスト・ブローカ (ORB) と通信します。次の図は、クライアント / サーバ環境の ORB を示しています。



CORBA では、クライアント・アプリケーションとサーバ・アプリケーションの媒介として ORB が定義されています。ORB では、クライアントの要求が適切なサーバ・アプリケーションに転送され、サーバの応答が要求元のクライアント・アプリケーションに返されます。ORB を使用することで、クライアント・アプリケーションではサーバ・アプリケーションの位置またはサーバが要求を満たす方法を知らなくてもサービスを要求できます。

CORBA モデルのクライアント・アプリケーションでは、どのような要求が可能なのか、およびどのように要求を行うことができるのかを知っているだけで十分です。サーバまたはデータ形式の細かなインプリメンテーションは必要ありません。サーバ・アプリケーションは、要求を満たすことができれば十分です。クライアント・アプリケーションにデータを返す方法について知っている必要はありません。

つまり、プログラマは、クライアント・アプリケーションがサーバ・アプリケーションにタスクの遂行を要求する方法に影響を与えることなく、サーバでタスクを遂行する方法を変更できます。たとえば、クライアント・アプリケーションとサーバ・アプリケーションのインターフェイスが変更されない限り、クライアント・アプリケーションに変更を加えることなくサーバ・アプリケーションのインプリメンテーションを拡張したり、新しいインプリメンテーションを開発したりできます。また、サーバ・アプリケーションはそのままに新しいクライアント・アプリケーションを作成することも可能です。

2 CORBA オブジェクト

CORBA プログラミングの説明を有意義にするには、まず、CORBA とは何なのかということと、BEA Tuxedo の情報で使われるオブジェクト用語を理解しておく必要があります。

ここでは、以下の内容について説明します。

- CORBA オブジェクトの定義
- CORBA オブジェクトの実体化
- CORBA オブジェクトの構成要素
- オペレーションの格納位置
- オブジェクト呼び出しのしくみ

オブジェクトの定義は、アーキテクチャやプログラミング言語によってさまざまです。たとえば C++ オブジェクトの概念は、CORBA オブジェクトの概念とは少し異なります。また、Component Object Model (COM) オブジェクトの概念も CORBA オブジェクトの概念とはかなり異なります。

最も重要なことは、この章の CORBA オブジェクトの概念が Object Management Group (OMG) によって提示されている定義と一致しているということです。OMG からは、オブジェクトの仕様やオブジェクトの細部を規定するほかの文書が多く発行されています。

CORBA オブジェクトの定義

CORBA オブジェクトは、単独では存在し得ないという意味において仮想のエンティティですが、その CORBA オブジェクトのリファレンスを利用し、クライアント・アプリケーションによってそのオブジェクトのオペレーションが要求された時点で実体化されます。CORBA オブジェクトのリファレンスは、オブジェクト・リファレンスと呼ばれます。オブジェクト・リファレンスは、BEA Tuxedo システムで CORBA オブジェクトを指定し操作できるようにする唯一の手段です。オブジェクト・リファレンスの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

クライアント・アプリケーションまたはサーバ・アプリケーションによってオブジェクト・リファレンスを使用してオブジェクトに対する要求が発行されると、そのオブジェクトがまだメモリでアクティブになっていない場合は、BEA Tuxedo サーバ・アプリケーションでオブジェクト・リファレンスで指定されたオブジェクトのインスタンス化が行われます (要求は常にオブジェクトの特定のオペレーション呼び出しにマッピングされる)。

通常、オブジェクトをインスタンス化する過程では、サーバ・アプリケーションによってオブジェクトの状態が初期化されます。その際、オブジェクトの状態はデータベースなどの永続ストレージから読み込まれる場合もあります。

オブジェクトには、以下のために必要なすべてのデータが格納されます。

- オブジェクトのオペレーションを実行する。
- オブジェクトが不要になったときに永続ストレージにオブジェクトの状態を格納する。

CORBA オブジェクトの実体化

CORBA オブジェクトを構成するデータは、データベースのレコードをベースとする場合があります。そのデータベースのレコードは、永続的なオブジェクトの状態です。このレコードは、次のように操作が行われたときに BEA Tuxedo ドメインの CORBA オブジェクトからアクセスできるようになります。

1. サーバ・アプリケーションのファクトリでオブジェクトのリファレンスが作成されます。オブジェクト・リファレンスには、データベースのレコードを見つける方法についての情報が含まれています。
2. ファクトリで作成されたオブジェクト・リファレンスを使用し、クライアント・アプリケーションでオブジェクトに対する要求が発行されます。
3. オブジェクトがインスタンス化されます。オブジェクトは、TP フレームワークにより `Server::create_servant` メソッド (Server オブジェクトにある) を呼び出すことによってインスタンス化されます。
4. BEA Tuxedo ドメインによって、そのオブジェクトで `activate_object` オペレーションが呼び出されます。その結果、状態を格納しているレコードがメモリに読み込まれます。

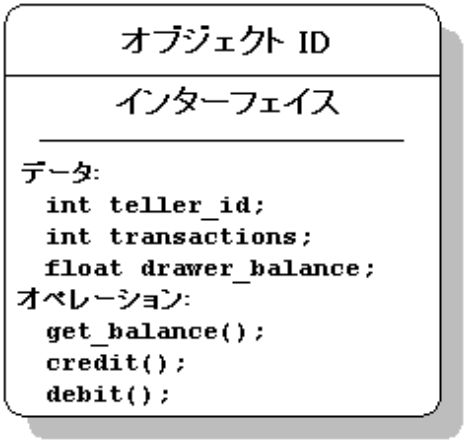
言語オブジェクトはアプリケーションの実行の境界内にのみ存在しますが、CORBA オブジェクトはプロセスやマシン・システムをまたがって存在できます。BEA Tuxedo システムには、オブジェクトを作成し、そのオブジェクトをアプリケーションからアクセス可能にするためのメカニズムがあります。

BEA Tuxedo CORBA サーバ・アプリケーションのプログラマは、オブジェクトの状態を初期化するコードとオブジェクトがアクティブでなくなったときにそのオブジェクトの状態を処理するコードを記述する必要があります。オブジェクトのデータが永続ストレージにある場合、このコードには永続ストレージのデータを読み書きするオペレーションが含まれます。サーバ・アプリケーション開発の詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

CORBA オブジェクトの構成要素

通常の CORBA オブジェクトの構成要素は次のとおりです (図も参照)。

- ID (オブジェクト ID または OID とも呼ぶ)
- インターフェイス (CORBA オブジェクトのデータとオペレーションを指定する)



以降の節では、オブジェクトの各構成要素を詳しく説明します。

オブジェクト ID

オブジェクト ID (OID) は、オブジェクトとその状態 (データベース・レコードなど) を関連付け、オブジェクトのインスタンスを識別します。ファクトリでオブジェクト・リファレンスが作成されると、そのオブジェクト・リファレンスの要求でファクトリに渡されるパラメータに基づいた OID が割り当てられます。

注記　サーバ・アプリケーションのプログラマは、BEA Tuxedo クライアント / サーバ・アプリケーションで使用されるファクトリを作成する必要があります。プログラマは、OID を割り当てるコードを記述しなければなりません。ファクトリの説明およびファクトリ作成の例については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

BEA Tuxedo システムでは、次の情報を利用してオブジェクトをインスタンス化する方法を指定できます。

- OID
- オブジェクト・リファレンスのアドレス指定データ
- オブジェクト・リファレンスのグループ ID

オブジェクト・インターフェイス

オブジェクトのインターフェイス (アプリケーションの OMG IDL ファイルで記述) は、データおよびオブジェクトに対して実行できるオペレーションを識別します。たとえば、University 窓口オブジェクトのインターフェイスでは次のものが識別されます。

- オブジェクトと関連付けられているデータ型 (窓口 ID や窓口の引き出しにある現金など)、およびオブジェクトによって管理されるデータ (口座など)
- オブジェクトで実行できるオペレーション (口座の残高照会や口座振替など)

CORBA オブジェクトの特性の 1 つは、インターフェイスの定義が実行時にそのデータとオペレーションから切り離されることです。CORBA システムでは、CORBA オブジェクトのインターフェイス定義をインターフェイス・リポジトリというコンポーネントに配置できます。データとオペレーションはインターフェイス定義によって指定されますが、オブジェクトがアクティブになったときにそれらはサーバ・アプリケーションのプロセスに存在します。

オブジェクトのデータ

オブジェクトのデータには、オブジェクト・クラスまたはオブジェクト・インスタンスに固有の情報がすべて含まれます。たとえば、University アプリケーションのコンテキストの典型的なオブジェクトに窓口があります。窓口のデータとしては次のものが考えられます。

- ID
- 窓口の引き出しにある現金の額
- 特定の期間 (日単位や月単位) でその窓口で処理された取引の数

オブジェクトのデータは、属性を使用してアクセス可能な構造体に結合したりすることで効率的にカプセル化できます。属性は、オブジェクトのデータをオペレーションから切り離す典型的な手段です。

オブジェクトのオペレーション

オブジェクトのオペレーションは、オブジェクトのデータを使用して処理を実行できるルーチンのセットです。たとえば、窓口オブジェクトを使用して機能を実行するオペレーションとしては次のものが考えられます。

- `get_balance()`
- `credit()`
- `debit()`

CORBA システムで、オブジェクトのオペレーションとして記述するコードはオブジェクト・インプリメンテーション (次節参照) と呼ばれることがあります。

オペレーションの格納位置

前節で説明したように、CORBA オブジェクトを構成するデータはデータベースのレコードに格納されている場合があります。言い換えると、CORBA オブジェクトのデータはオブジェクトがメモリでアクティブになっ

たときのみ確立できます。この節では、CORBA オブジェクトのオペレーションを記述する方法、およびそのオペレーションをオブジェクトの一部とする方法を説明します。

特定の CORBA オブジェクトで記述されるオペレーションは、オブジェクトのインプリメンテーションとも呼ばれます。インプリメンテーションは、オブジェクトの振る舞いを実現するコードと見なすことができます。BEA Tuxedo CORBA クライアント / サーバ・アプリケーションを作成する過程では、アプリケーションの OMG IDL ファイルをコンパイルします。OMG IDL ファイルには、アプリケーションのインターフェイスおよびそのインターフェイスで実行できるオペレーションを記述する文が格納されます。

C++ でサーバ・アプリケーションをインプリメントする場合、IDL コンパイラで必要に応じて生成されるファイルの 1 つにインプリメンテーション・ファイルのテンプレートがあります。インプリメンテーション・ファイルのテンプレートの内容は、アプリケーションのオブジェクトのデフォルトのコンストラクタとメソッド・シグニチャです。オブジェクトをインプリメントするコードを記述する場所は、このインプリメンテーション・ファイルです。つまり、このファイルには特定のインターフェイスのオペレーションのビジネス・ロジックが格納されます。

BEA Tuxedo システムでは、インターフェイスを CORBA オブジェクトとしてインプリメントします。IDL コンパイラでは、BEA Tuxedo CORBA クライアント / サーバ・アプリケーションに組み込まれ、特定のオブジェクトに記述したインプリメンテーションが実行時に適切なオブジェクト・データに接続されるようにするほかのファイルも生成されます。

ここでサーバントの概念が登場します。サーバントとは、オブジェクト・クラスのインスタンスのことです。つまり、サーバントはインプリメンテーション・ファイルの各オペレーションで記述したメソッド・コードのインスタンスです。BEA Tuxedo CORBA クライアント / サーバ・アプリケーションの動作中に、アクティブではない (メモリにない) オブジェクトに対するクライアントの要求がサーバ・アプリケーションに届くと、以下のようにイベントが発生します。

1. 必要なオブジェクトで利用可能なサーバントがない場合、BEA Tuxedo システムでは Server オブジェクトの `Server::create_servant` メソッドを呼び出します。

   `Server::create_servant` メソッドは、ユーザがすべてを記述します。`Server::create_servant` メソッドで記述するコードでは、必要なサーバ

ントをインスタンス化します。コードでは、インターフェイス名 (`Server::create_servant` メソッドにパラメータとして渡される) を使用して、BEA Tuxedo ドメインによって作成されるサーバントの型を指定できます。

BEA Tuxedo ドメインで作成されるサーバントは、特殊なサーバント・オブジェクト・インスタンスです (CORBA オブジェクトではない)。このサーバントには、必要な CORBA オブジェクトをインプリメントする、以前に記述したオペレーションの実行可能バージョンが格納されます。

2. BEA Tuxedo ドメインでは、制御をサーバントに渡し、必要に応じてサーバントの `activate_object` メソッドを呼び出します (メソッドがインプリメントされている場合)。次のように、`activate_object` メソッドを呼び出すと、CORBA オブジェクトがアクティブになります。

   a. `activate_object` メソッドのコードを記述します。`activate_object` メソッドに渡すパラメータは、アクティブにするオブジェクトのオブジェクト ID の文字列値です。オブジェクト ID は、オブジェクトを初期化する方法を指定するために使用できます。

   b. CORBA オブジェクトのデータを初期化します。その際には、永続ストレージ (データベースのレコードなど) から状態データを読み込む場合もあります。

   c. サーバントのオペレーションがデータにバインドされ、そのオペレーションとデータの結合によってアクティブな CORBA オブジェクトが確立されます。

   上記の a、b、および c が終わると、CORBA オブジェクトはアクティブと判断されます。

   `activate_object` メソッドは、必ずしもインプリメントする必要はありません。どのような場合にインプリメントする必要があるのかについては、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

注記　サーバントは CORBA オブジェクトではありません。実際上、サーバントは言語オブジェクトとして表されます。サーバでは、サーバントを通じてオブジェクトでオペレーションを実行します。

オブジェクト・インプリメンテーション作成の詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

オブジェクト呼び出しのしくみ

CORBA オブジェクトは分散環境で機能するように意図されているので、OMG ではオブジェクト呼び出しのしくみに関するアーキテクチャが定義されています。CORBA オブジェクトは、次の 2 とおりの方法で呼び出すことができます。

- 生成されたクライアント・スタブおよびスケルトンの使用 (スタブ・スタイル起動とも呼ばれる)
- 動的起動インターフェイスの使用 (動的起動と呼ばれる)

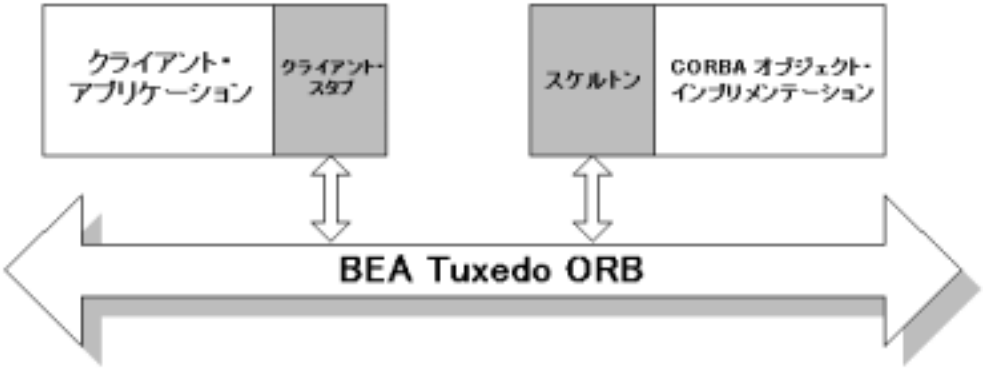
動的起動のしくみについては、『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』を参照してください。この節では、動的起動よりも使いやすいスタブ・スタイル起動について説明します。

アプリケーションの OMG IDL ファイルをコンパイルするときには、クライアント・スタブというソース・ファイルがコンパイラによって生成されます。クライアント・スタブでは、オブジェクト型の OMG IDL オペレーション定義が、要求を満たすために BEA Tuxedo システムによって呼び出される CORBA サーバ・アプリケーションのオペレーションにマッピングされます。クライアント・スタブには、サーバ・アプリケーションに要求を送信するときに使用される、クライアント・アプリケーションのビルド・プロセスで生成されたコードが格納されます。クライアント・スタブのコードは絶対に修正しないでください。

IDL コンパイラでは、スケルトンというソース・ファイルも生成されます。スケルトンには、OMG IDL ファイルで指定された各インターフェイスのオペレーション呼び出しで使用されるコードが格納されます。スケルトンは、クライアントの要求を満たすことができる CORBA オブジェクト・インプリ

メンテーションの適切なコードを指し示すマップです。スケルトンは、オブジェクト・インプリメンテーションと BEA Tuxedo オブジェクト・リクエスト・ブローカの両方に接続されます。

次の図は、クライアント・アプリケーション、クライアント・スタブ、スケルトン、および CORBA オブジェクト・インプリメンテーションを示しています。



クライアント・アプリケーションから要求が送信されると、その要求はクライアント・スタブのオペレーションとしてインプリメントされます。クライアント・スタブで要求を受信すると、クライアント・スタブではその要求をオブジェクト・リクエスト・ブローカ (ORB) に送信します。ORB では、受信した要求を BEA Tuxedo システムを通じてスケルトンに送信します。ORB は、TP フレームワークおよびポータブル・オブジェクト・アダプタ (POA) と連携して適切なスケルトンとオブジェクト・インプリメンテーションを見つけます。

クライアント・スタブおよびスケルトンの生成については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』および『BEA Tuxedo C リファレンス』を参照してください。

3 Process-Entity デザイン・パターン

ここでは、以下の内容について説明します。

- Process-Entity デザイン・パターンについて
- スケーラビリティとリソース利用率の向上
- 適用可能性
- 構成要素
- 留意事項
- 関連する概念

Process-Entity デザイン・パターンについて

Process-Entity デザイン・パターンは、データベース・レコード (エンティティ) とのクライアント・アプリケーションのすべてのやり取りをサーバ・マシン上の単一プロセス・オブジェクトで処理する設計ソリューションです。このデザイン・パターンは、クライアントの CORBA または EJB アプリケーションでリモート・データベースとの複数のやり取りが普通に実行される状況で効果的です。

データベースの細かなすべてのデータを表すサーバ・マシン上の単一の CORBA オブジェクトまたは EJB を設計することで、次のような性能上の利点をもたらす BEA Tuxedo CORBA クライアント / サーバ・アプリケーションをビルドできます。

- クライアントがデータベースと複数のやり取りを行うのではなく、データベースとのやり取りを求めるすべてのクライアント要求をサーバ・マシン上の単一プロセス・オブジェクトで処理することで、ネットワーク・トラフィックを軽減できます。
- プロセス・オブジェクトでは、データ・フィールドを取捨選択してクライアントに渡すことができます。したがって、データベース・レコード全体ではなく必要なデータのみが転送されるので、ネットワークで送信されるデータ量が減り、性能が向上します。
- プロセス・オブジェクトで、データベースへのアクセスがカプセル化されます。クライアントでオブジェクトが呼び出され、そのオブジェクトによってデータベースへのアクセスが行われます。

スケーラビリティとリソース利用率の向上

ここでは、以下の内容について説明します。

- 2 層システムの制限
- Process-Entity デザイン・パターンの利点

2 層システムの制限

データベース層を共有データのセットとして提供する従来の 2 層システムにおいて、純粋なオブジェクト指向手法とは、データベース・レコードを 共有 CORBA オブジェクト (CORBA アプリケーションの場合) またはエンティティ Bean (EJB アプリケーションの場合) として表現することです。ただし、この手法には次のような制限があります。

- 規模が適切に調整されません。クライアント数が劇的に増加した場合、サーバ・マシンで数千 (あるいは数百万) のデータベース・オブジェクトを管理しなければならなくなる可能性があります (各オブジェクトではそれ独自のトランザクション・コンテキストが必要)。
- ネットワーク・リソースが効率的に利用されません。データベース・オブジェクトがサーバ・マシンのメモリでインスタンス化される際、クライアント・アプリケーションでそのオブジェクトのどのくらいの量のデータが必要とされているのかに関係なくデータベース・オブジェクト全体がメモリで読み書きされます。
- EJB アプリケーションでは、通常はエンティティ Bean を使用してデータベースにアクセスします。その際、エンティティ Bean はデータベース・テーブルの行を表します。ただし、エンティティ Bean にアクセスするために、クライアント・アプリケーションでは 2 回の呼び出しを行わなければなりません。最初の呼び出しではエンティティ Bean のオブジェクト・リファレンスを取得し、2 回目の呼び出しでそのエンティティ

Bean のメソッドを呼び出します。オブジェクト・リファレンスを取得することは、特に大規模なエンタープライズ・アプリケーションではクライアントにとってコストのかかる操作となります。

Process-Entity デザイン・パターンの利点

しかし、クライアントの代わりにデータベースとやり取りするサーバ・マシン上のプロセス・オブジェクトのクラスを設計すれば、次のような利点によって上記の制限を克服できます。

- サーバ・マシンで管理する必要のある CORBA オブジェクトまたは EJB の数が減ります。
- メッセージ・トラフィックが減少します。
- EJB アプリケーションでは、クライアント・アプリケーションでエンティティ Bean のオブジェクト・リファレンスを取得する必要がなくなり、また細かな (単一行の) エンティティ Bean を使用しなくて良くなります。

適用可能性

ここでは、以下の内容について説明します。

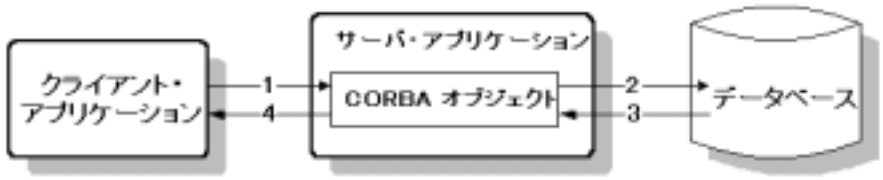
- CORBA アプリケーションでの要求の流れ
- EJB アプリケーションでの要求の流れ

Process-Entity デザイン・パターンは、エンタープライズ・レベルのミッション・クリティカルなアプリケーションでほぼ普遍的に適用できます。このデザイン・パターンは、クライアント・アプリケーションがサーバ・マシン上のデータベース・レコードとやり取りする必要のある状況で使用します。

CORBA アプリケーションでの要求の流れ

図 3-1 は、CORBA アプリケーションの Process-Entity デザイン・パターンの基本設計を示しています。

図 3-1CORBA の Process-Entity デザイン・パターン



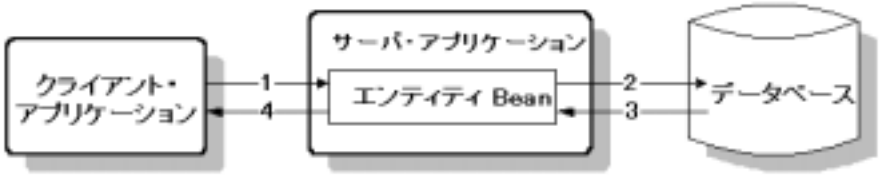
このプロセスは、次の順序で行われます。

1. クライアント・アプリケーションから、データベース・エンティティにアクセスするために CORBA プロセス・オブジェクトに要求が発行されます。
2. CORBA オブジェクトからデータベースに要求が送信されます。
3. データベースから CORBA オブジェクトに応答が返されます。
4. CORBA オブジェクトからクライアントに応答が返されます。応答に含まれているのは、クライアントが必要とするデータベースの情報のみです。

EJB アプリケーションでの要求の流れ

図 3-2 は、EJB アプリケーションの Process-Entity デザイン・パターンの基本設計を示しています。

図 3-2EJB の Process-Entity デザイン・パターン



このプロセスは、次の順序で行われます。

1. クライアント・アプリケーションから、データベース・エンティティにアクセスするために RMI over IIOP を使用してエンティティ Bean に要求が発行されます。
2. エンティティ Bean からデータベースに要求が送信されます。
3. データベースから エンティティ Bean に応答が返されます。
4. エンティティ Bean からクライアントに応答が返されます。応答に含まれているのは、クライアントが必要とするデータベースの情報のみです。

構成要素

クライアント・アプリケーションでは、ファクトリ (CORBA アプリケーションの場合) またはホーム・インターフェイス (EJB アプリケーションの場合) からプロセス・オブジェクトのリファレンスを取得します。プロセス・オブジェクトでは、データベースとのすべてのやり取りがインプリメントされます。データベース・レコード (エンティティ) は、プロセス・オブジェクトのクライアント呼び出しを処理するために必要なときに取り出されます。プロセス・オブジェクトのオペレーションでは、特定のデータ・フィールドがクライアント・アプリケーションに返されます。クライアント・アプリケーションでは、そのデータで必要なすべての処理を実行します。

留意事項

プロセス・オブジェクトは、特定のクライアント要求で実際に必要とされる最小限の情報を渡すように設計します。プロセス・オブジェクトのオペレーションは、可能な限り「密度の高い」処理が実行されるようにインプリメントします。クライアント・アプリケーションは、タスクを遂行するために必要なデータの取得に複数のプロセス・オペレーションを呼び出さないように設計します。

複数のオペレーションを呼び出す必要がある場合は、追加の呼び出しがクライアント・アプリケーションからプロセス・オブジェクトにではなく、プロセス・オブジェクトからデータベースに対して行われるようにプロセス・オブジェクトを設計してください。そのように設計することで、クライアント・アプリケーションからネットワーク経由で送信される呼び出しの数が減ります。クライアント・アプリケーションからプロセス・オブジェクトに連続的に呼び出しを行わなければならない場合は、プロセス・オブジェクトをステートフルにしてください。オブジェクトをステートフルにする方法については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

CORBA アプリケーションでは、OMG IDL で属性を使用しないでください。属性は、ネットワーク経由で取り出すのにコストがかかります。代わりに、クライアント・アプリケーションで必要となりそうなすべての値を格納するデータ構造を返すオペレーションをプロセス・オブジェクトでインプリメントします。

関連する概念

- SmallTalk MVC (モデル / ビュー / コントローラ) デザイン・パターン
- Flyweight デザイン・パターン (『Design Patterns: Elements of Reusable Object-Oriented Software』（Gamma ほか著）を参照)

4 Client Data Caching デザイン・パターン

この章では、CORBA Client Data Caching デザイン・パターンについて説明します。このデザイン・パターンの目的は、サーバからの永続的な状態情報をクライアントがローカルでデータ集中型の処理に利用できるようにすることです。このようにすれば、CORBA クライアント・アプリケーションでデータを取り出すために何度もサーバ・アプリケーションに呼び出しを行う必要がなくなります。

このデザイン・パターンの理由

このデザイン・パターンでは、分散クライアント / サーバ・アプリケーションのスケーラビリティと性能が改善されます。CORBA オブジェクトの属性を取り出すリモート呼び出しのオーバーヘッドは、システムのロードや他の要素によっては非常に大きくなる場合があります。また、永続的なデータ・レコードを CORBA オブジェクトとしてエクスポーズする場合は、非常に多くの同時アクティブ・オブジェクトをシステムで管理しなければならなくなる可能性があるのでアプリケーションの規模を適切に調整できなくなります。クライアント・アプリケーションのデータが集中する処理、またはユーザ入力を必要とする処理 (フィールドの編集など) は、データの取得に複数のリモート呼び出しを要する場合はクライアント・アプリケーションとシステムの両方の速度を低下させる可能性があります。

適用可能性

このデザイン・パターンは、CORBA プロセス・オブジェクトからクライアント・アプリケーションに大量のデータを渡さなければならない状況で使用するのが適しています。クライアント・アプリケーションのローカル言語オブジェクトはデータのコンテナになり、そのコンストラクタはローカル・オブジェクトの状態を設定するために使用します。

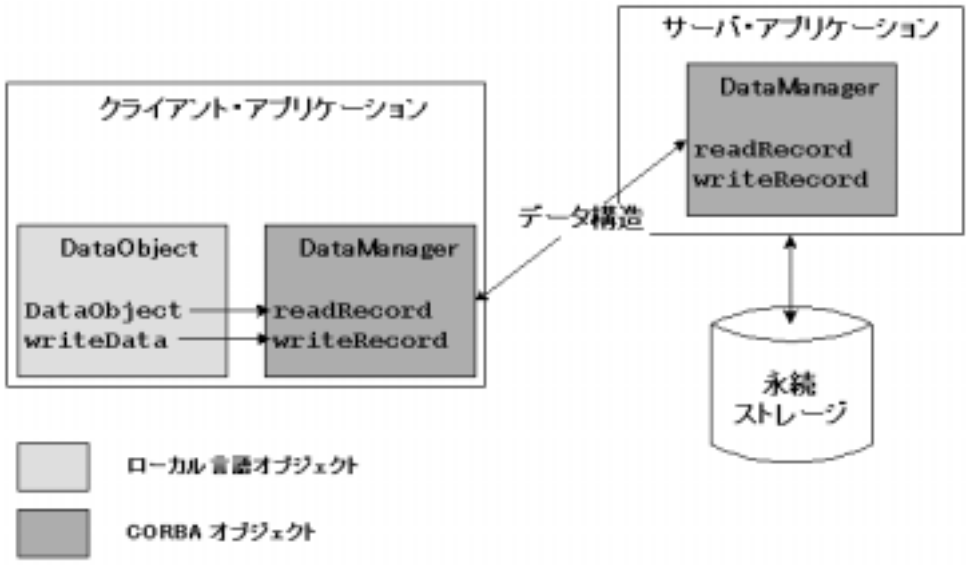
このデザイン・パターンは、ローカル言語オブジェクト (この章では `DataObject` と呼ぶ) を作成するクライアント・アプリケーションでインプリメントします。サーバ・アプリケーションでは、永続ストレージのエンティティとやり取りする CORBA プロセス・オブジェクトをインプリメントします。この CORBA オブジェクトは、この章では `DataManager` オブジェクトと呼びます。

`DataManager` CORBA オブジェクトの OMG IDL では、クライアント・アプリケーションとサーバ・アプリケーションの間でデータを転送する際に使用するデータ構造を定義します。このデザイン・パターンでは、「楽観的ロック」が採用されています。つまり、クライアント・アプリケーションでローカル・コピーが使用されている間はほかのサーバ・プロセスによってデータが変更されることはないと想定し、サーバ・アプリケーションで管理されるデータが更新用にロックされることはありません。

クライアント・アプリケーションでローカルの `DataObject` がインスタンス化されると、そのオブジェクトのコンストラクタから (`DataObject` にデータ構造を渡す) `DataManager` CORBA オブジェクトのオペレーションが呼び出されます。`DataObject` では、渡されたデータを利用してクラス変数を設定します。

クライアント・アプリケーションからサーバ・マシンに変更された状態を渡す必要がある場合、クライアント・アプリケーションでは `DataObject::writeData()` メソッドを呼び出します。このメソッドでは、今度は `DataManager` CORBA オブジェクトの `writeRecord()` オペレーションが呼び出されます。この呼び出しでは、データ構造が `writeRecord()` オペレーションにパラメータとして渡されます。`DataManager` CORBA オブジェクトでは、永続ストレージを適切に更新します。

次の図は、CORBA Client Data Caching デザイン・パターンのしくみを示しています。



次に、この図の意味を説明します。

1. `DataObject` コンストラクタでは、`DataManager` CORBA オブジェクトの `readRecord()` オペレーションが呼び出され、返されたデータ構造を使用してローカルの状態が初期化されます。
2. クライアント・アプリケーションでは、`DataObject` インスタンスのローカルの状態を変更する場合があります。
3. 変更された状態を `DataManager` CORBA オブジェクトに渡すために、クライアント・アプリケーションでは `DataObject::writeData()` オペレーションを呼び出し、変更されたデータを格納するデータ構造を渡します。

構成要素

`DataObject` のメソッドでは、`DataManager` CORBA オブジェクトのオペレーションを呼び出してデータを読み書きします。

留意事項

クライアント・アプリケーションに渡されるデータ構造は、必要最小限のデータを提供するように設計されていなければなりません。大量のデータが伴う場合は、必要なフィールドのサブセットを格納する複数のデータ構造を使用した方が効率的になる場合があります。CORBA プロセス・オブジェクトのオペレーションは、必要とされるデータのサブセットのみが操作に関与するように設計します。そうすれば、ネットワーク・トラフィックが削減されます。