移行ガイド

iPlanet™ Application Server

Version 6.5

2002 年 2 月

目 次

はじめに

このマニュアルでは、Netscape Application Server バージョン 2.1、4.0 および iPlanet™ Application Server 6.0 SPx から iPlanet Application Server 6.5 にアプリケーションを移行する方法について説明します。さらに、このマニュアルには Net Dynamics アプリケーションを移行するための情報も記載されています。

この章には次の節があります。

- マニュアルの使用法
- このマニュアルの構成
- マニュアルの表記規則
- 関連情報

マニュアルの使用法

次の表は、iPlanet Application Server のマニュアル、および『リリースノート』に記述されているタスクと概念を示しています。特定のタスクを行う場合や特定の概念について調べる場合は、該当するマニュアルを参照してください。

注 : 印刷版マニュアルは、`http://docs.iplanet.com/docs/manuals/ias.html` から PDF および HTML 形式で入手できます。

情報の内容	参照するマニュアル	添付されている製品
ソフトウェアおよびマニュアルの最新情報	リリースノート	iPlanet のマニュアルは、次の Web サイトから入手可能です。 http://docs.iplanet.com

情報の内容	参照するマニュアル	添付されている製品
iPlanet Application Server およびそのコンポーネント (Web コネクタプラグイン、iPlanet Application Server Administrator) のインストールと、サンプルアプリケーションの設定	インストールガイド	iPlanet Application Server 6.5
次のタスクを行うための、iPlanet Application Server Administrator Tool による 1 台または複数台のアプリケーションサーバの管理 • サーバの稼動状況の監視およびログ記録 • iPlanet Application Server へのセキュリティの実装 • サーバリソースの高利用度の実現 • Web コネクタプラグインの設定 • データベース接続の管理 • トランザクションの管理 • 複数のサーバの設定 • 複数のサーバでのアプリケーションの管理 • サーバのロードバランス • 分散データ同期の管理 • 開発用の iPlanet Application Server のセットアップ	管理者ガイド	iPlanet Application Server 6.5
iPlanet Application Server に付属するオンラインバンクアプリケーションの移行サンプルを含む、Netscape Application Server バージョン 2.1 から新しい iPlanet Application Server 6.5 プログラミングモデルへのアプリケーションの移行	移行ガイド	iPlanet Application Server 6.5

情報の内容	参照するマニュアル	添付されている製品
次のタスクによる、オープン Java 標準モデル (Servlet、EJB、JSP、および JDBC) に準拠した iPlanet Application Server 6.5 アプリケーションの作成 • アプリケーションのプレゼンテーション層および実行層の作成 • EJB (Enterprise JavaBeans) コンポーネントへのビジネスロジックの個別部分およびエンティティの配置 • JDBC を使ったデータベースとの通信 • 反復テスト、デバッグなどアプリケーションの調整機能を使用した、正確かつ高速に動作するアプリケーションの生成	開発者ガイド	iPlanet Application Server 6.5
Java アプリケーションを作成する場合の iPlanet Application Server クラスライブラリの共有クラスとインターフェイス、およびそれらのメソッドの使用	Server Foundation Class Reference (Java)	iPlanet Application Server 6.5
C++ アプリケーションを作成する場合の iPlanet Application Server クラスライブラリの共有クラスとインターフェイス、およびそれらのメソッドの使用	Server Foundation Class Reference (C++)	別注文

このマニュアルの構成

このマニュアルは次の 5 つの章に分かれています。

- 第 1 章「移行の概要」
- 第 2 章「アプリケーションの移行」
- 第 3 章「NAS 2.1 アプリケーションの実行」
- 第 4 章「NAS 2.1 アプリケーションの移行」
- 第 5 章「NAS 4.0 アプリケーションの実行」
- 第 6 章「NetDynamics アプリケーションの実行」

さらに、オンラインバンクのサンプルアプリケーションを使って、NAS 4.0 から iPlanet Application Server 6.0 への移行例のコードを解説します。このコード例もオンラインで入手可能です。移行の詳細は、iPlanet Web サイトのサポートページをご覧ください。

マニュアルの表記規則

ファイルとディレクトリのパスは、Windows の形式で表記されます (ディレクトリ名を円記号で区切って表記)。Unix バージョンではディレクトリパスについては Windows と同じですが、ディレクトリの区切りには円記号ではなくスラッシュが使われます。

このマニュアルでは次のように URL 形式を使います。

http://server.domain/path/file.html

URL で、*server* はアプリケーションを実行するサーバの名前です。*domain* はユーザのインターネットのドメイン名、*path* はサーバ上のディレクトリの構造、*file* は個別のファイル名です。URL の斜体文字の部分はプレースホルダです。

このマニュアルでは、フォントについて次の規則を採用しています。

- モノスペースフォントは、サンプルコード、コードの一覧表示、API および言語要素 (関数名、クラス名など)、ファイル名、パス名、ディレクトリ名、および HTML タグに使います。
- イタリックフォントは、マニュアル名、変数とプレースホルダ、およびリテラルに使われる文字に使います。

関連情報

iPlanet Application Server 6.5 プログラミングモデルに関連する仕様書は、インストール CD の docs ディレクトリに収録されています。ただし、製品受領後に更新されている場合があるので、常にオンラインマニュアルを先に参照してください。

公式の仕様書は次の URL で管理されています。iPlanet Application Server でサポートされているバージョンの仕様書がこれらのサイトにない場合もありますので注意してください。

情報の内容	参照するマニュアル
Servlet	`http://java.sun.com/products/servlet`
JavaServer Pages (JSP)	`http://java.sun.com/products/jsp`
Enterprise JavaBeans (EJB)	`http://java.sun.com/products/ejb`
Java Naming and Directory Interface (JNDI)	`http://java.sun.com/products/jndi`
Java Database Connectivity (JDBC)	`http://java.sun.com/products/jdbc`

また、次のリソースも参考にしてください。

Servlet および *JSP* を使ったプログラミング

- 『Java Servlet Programming』、Jason Hunter、William Crawford 共著、O'Reilly 発行
- 『Java Threads, 2nd Edition』、Scott Oaks、Henry Wong 共著、O'Reilly 発行
- Web サイト *http://www.servletcentral.com*

EJB を使ったプログラミング

- 『Enterprise JavaBeans』、Richard Monson-Haefel 著、O'Reilly 発行
- Web サイト *http://ejbhome.iona.com*

JDBC を使ったプログラミング

- 『Database Programming with JDBC and Java』、George Reese 著、O'Reilly 発行
- 『JDBC』、Graham Hamilton、Rick Cattell、Maydene Fisher 共著

第 1 章

移行の概要

この章では、iPlanet™ Application Server 6.5 プログラミングモデルを紹介し、NAS 4.0 プログラミングモデルおよび NAS 2.1 プログラミングモデルの両方と比較します。

新しい iPlanet Application Server 6.5 プログラミングモデルは、Java アプリケーション専用です。C++ アプリケーションでは、引き続き、NAS 2.1 モデルが使われます。次の互換性に関する事項に注意してください。

- iPlanet Application Server 6.5 をビルドするためにコンパイラを変更するには、Solaris 上のアプリケーションをベースにした AppLogics が、iPlanet Application Server 6.5 で実行できるようにコンパイルし直す必要があります。
- iPlanet Application Server 6.5 は、NAS 2.1 アプリケーションと後方互換性があります。NAS 2.1 アプリケーションは、コード変更しなくても iPlanet Application Server 6.5 上で実行できます。
- iPlanet Application Server 6.0 は、J2EE 標準に変換することによって、NAS 4.0 アプリケーションとの互換性を維持します。また、NAS 4.0 アプリケーションを一部変換する必要があります。

この章では、次のトピックについて説明します。

- 新しい J2EE プログラミングモデル

新しい J2EE プログラミングモデル

iPlanet Application Server 6.5 は、Java 2 Platform、Enterprise Edition (J2EE) Specification Version 1.2 に準拠しており、Java コミュニティによって開発された標準、つまり、Servlet、JavaServer Pages、および Enterprise JavaBeans をベースとしています。この点で、NAS 2.1 で使われている専用の AppLogic ベースのプログラミングモデルとは異なります。NAS 4.0 は J2EE プログラミングモデルをベースにしていますが、J2EE 標準の以前のバージョンを採用しています。

iPlanet Application Server 6.5 モデルと iPlanet Application Server 4.0 および 2.1 モデルのアプリケーションフローは似ています。すなわち、入力の処理、ビジネスロジック関数の実行、データベースとの対話、入力に応答した出力ページの提供、および次のユーザ対話の設定を行う 1 つまたは複数のアプリケーションコンポーネントによって各ユーザ対話が処理されます。6.5 モデルは、4.0 モデルと同様にモジュール化が進んでおり、アクティビティをより多くの個々のコンポーネントに分割できます。

新しいプログラミングモデルでは、それぞれが一連のコンポーネントまたは API によって表された、3 層のアプリケーションロジックが記述されます。次の表で、これらの層について説明します。

プログラミング層	NAS 2.1 コンポーネント	NAS 4.0 コンポーネント	iPlanet Application Server 6.5 コンポーネント	説明
プレゼンテーションロジック	AppLogic	Java Servlet と専用標準	Java Servlet	リクエストを処理し、それに応答するコンテンツを生成してフォーマット化し、そのコンテンツをユーザに返信することによってアプリケーションとユーザとのインターフェイスを制御する。6.5 では、Servlet が受信リクエストを処理して、レスポンスを作成する。ビジネスロジックは通常、EJB にオフロードされ、出力は JSP にオフロードされる
プレゼンテーションレイアウト (プレゼンテーションロジックの一部)	HTML テンプレート	JavaServer Page (JSP) と専用標準	JavaServer Page (JSP)	各ページの外観を制御する。通常は JavaServer Pages によって処理されるプレゼンテーションロジックの一部。JSP は、Java が組み込まれている HTML ページであるため、2.1 HTML テンプレートよりも汎用性が高く、より強力
ビジネスロジック	AppLogic	Enterprise JavaBeans (EJB) と専用標準	Enterprise JavaBeans (EJB)	ビジネスロジックを制御する。EJB によって、ビジネスロジックは、複数の呼び出しにわたって持続するので、キャッシュ効率も向上する。また、EJB は、データベーストランザクション用の JDBC と緊密に連携するように設計されている

プログラミング層	NAS 2.1 コンポーネント	NAS 4.0 コンポーネント	iPlanet Application Server 6.5 コンポーネント	説明
データアクセスロジック	DAE	JDBC と専用標準	JDBC	データベースの保存および検索を制御する。6.5 モデルでは、データベーストランザクションは通常、EJB によって制御されるが、すべての API と同様に、JDBC API は、すべての Java コンポーネントに利用できる

この節には、次の追加のトピックがあります。

- コンポーネントのモジュール性と柔軟性
- プレゼンテーションロジックとレイアウト
- ビジネスロジック
- データアクセスロジック

コンポーネントのモジュール性と柔軟性

このマニュアルや『開発者ガイド』には、iPlanet Application Server 6.5 コンポーネントの役割に関して、「通常は」や「普通は」という言葉が頻繁に使われています。Servlet、JSP、および EJB はすべて同じ仮想マシン内にある Java オブジェクトであるため、各タスクを複数のコンポーネントで処理できる柔軟性を持っています。どのタスクがどのコンポーネントに適しているかを指定する厳密なルールはありません。たとえば、JSP または Servlet だけを使って、複雑なアプリケーション全体を記述できます。

ただし、コンポーネントはモジュール形式で連動しながら各コンポーネントの長所を活かすように設計されています。たとえば、Servlet でレイアウトタスクを実行することは困難ですが、JSP (HTML ページ) はレイアウトタスクに非常に適しています。一方、プレゼンテーションロジックは、Servlet の方が簡潔に実行できます。

コンポーネントを分割して整理することによって、分散環境で効率よく動作する強力なアプリケーションモデルを記述できます。この章で説明するプログラミング層をガイドラインとして使い、必要なタスクを実行するコンポーネントを選択してください。

プレゼンテーションロジックとレイアウト

プレゼンテーション ロジックとは、リクエストの処理やそれに続くコンテンツの生成および配信といった、各ユーザ対話の観点からのアプリケーションフローを表します。プレゼンテーションロジックの目的は、リクエストに対する論理的なレスポンスの生成と、ほかのリクエストを促すプロンプトの表示です。プレゼンテーションレイアウトの目的は、あらかじめ決められた書式でレスポンスコンテンツを表示することです。ユーザセッション、セキュリティおよびユーザ認証、入力確認などのアプリケーション機能も、プレゼンテーションロジックによって処理されます。

つまり、プレゼンテーションロジックには、アプリケーションとユーザ間のインターフェイスに関するすべてが含まれます。

NAS 2.1 プログラミングモデルでは、プレゼンテーションロジックは AppLogic クラスによって制御される一方、レイアウトは HTML テンプレートによって処理されていました。実行時に、AppLogic オブジェクトがデータを出力し、テンプレートに入力していました。

iPlanet Application Server 6.5 プログラミングモデルでは通常、プレゼンテーションロジックは Java Servlet によって処理され、レイアウトは JSP によって処理されます。実行時に、Servlet は JSP を使って、ビジネスロジックによって生成されるコンテンツをフォーマット化します。

この基本モデルに代わる主な 2 つの方法を次に示します。

- 指定された対話のすべてのプレゼンテーションロジックおよびレイアウトを JSP で処理します。ビジネスロジックがなく、前の対話からの処理がほとんどない対話については、この方法で簡単に制御できます。たとえば、多くの場合、アプリケーションの「表紙」には処理が不要なのでこの方法が適しています。
- すべてのプレゼンテーションロジックおよびレイアウトを Servlet で処理します。この方法は、レイアウトをほとんど持たない対話に適しています。たとえば、簡単なデータベースレポートでは、データベースクエリから取得した行を一覧表示するだけです。Servlet からページが出力されるだけの場合は、JSP 呼び出しのオーバーヘッドが発生することはありません。

ビジネスロジック

ビジネスロジックは、データの保存および取得とそのデータに関する計算の実行といった、特定のコンテンツの生成を伴う作業を表します。ビジネスロジックの目的は、プレゼンテーションロジックによって提示された質問に対してレスポンスを生成または決定するアクティビティを実行することです。

つまり、ビジネスロジックには、アプリケーションによって提供されるコンテンツとアプリケーションに対して生成されるコンテンツが含まれます。

NAS 2.1 プログラミングモデルでは、指定されたユーザ対話のプレゼンテーションロジックを処理した AppLogic によってビジネスロジックが制御されていました。

iPlanet Application Server 6.5 プログラミングモデルでは通常、1 つまたは複数の Enterprise JavaBeans (EJB) によってビジネスロジックが処理されます。EJB はデータベーストランザクションを制御し、その結果をカプセル化します。EJB は強力で再利用可能なコンポーネントであり、アプリケーションの柔軟性を大幅に向上させます。これは、EJB が、どのオブジェクトからでも起動または検査が可能であり、継続させることができるためです。

このモデルに代わる 1 つの方法は、AppLogic によるビジネスロジックの処理とほぼ同じ方法で、Servlet や JSP などのプレゼンテーションロジックでビジネスロジックを処理することです。この方法は、特定のディレクトリリクエストなど目的が明確で比較的短いビジネスイベントには効果的ですが、EJB がプログラミングモデルにもたらすような柔軟性とパワーはありません。

データアクセスロジック

データアクセスロジックとは、データベースまたはディレクトリサーバとのトランザクションを表します。データアクセスロジックの目的は、アプリケーションと関連データセット間のインターフェイスを提供することです。通常、データアクセスはビジネスロジックの一機能として実行されます。

つまり、データアクセスロジックには、ビジネスロジックによって収集または生成されたコンテンツの保存および取得が含まれます。

NAS 2.1 プログラミングモデルでは、一部のクラスおよびインターフェイスの API を使った AppLogic からの呼び出しによって、データアクセスロジックが制御されていました。これらのクラスおよびインターフェイスには、`DataSet`、`DBDataSet`、および `DBStoredProcedure` クラスと、`ICallableStmt`、`IColumn`、`IDataConn`、`IDataConnSet`、`IHierQuery`、`IHierResultSet`、`IListDataSet`、`IPreparedQuery`、`IQuery`、`IResultSet`、`ITable`、`ITrans`、および `IValList` インターフェイスが含まれます。

iPlanet Application Server 6.5 プログラミングモデルでは、JDBC 標準の API セットによってデータアクセスロジックが制御されます。iPlanet Application Server 6.5 では、以前の API はすべて廃止されています。

第 2 章

アプリケーションの移行

この章では、iPlanet™ Application Server 6.0 SPx アプリケーションを iPlanet Application Server 6.5 に配置し直すために必要な手順を説明します。

iPlanet Application Server 6.5 プログラミングモデルは、Java アプリケーション専用です。C++ アプリケーションでは、引き続き、NAS 2.1 モデルが使われます。次の互換性に関する事項に注意してください。

- iPlanet Application Server 6.5 をビルドするためにコンパイラを変更するには、Solaris® 上のアプリケーションをベースにした AppLogics が、iPlanet Application Server 6.5 で実行できるようにコンパイルし直す必要があります。
- iPlanet Application Server 6.5 は、NAS 2.1 アプリケーションと後方互換性があります。NAS 2.1 アプリケーションは、コード変更しなくても iPlanet Application Server 6.5 上で実行できます。
- iPlanet Application Server 6.5 は、J2EE 標準に変換することによって、NAS 4.0 アプリケーションとの互換性を維持します。また、NAS 4.0 アプリケーションを一部変換する必要があります。

この章では、次のトピックについて説明します。

- 6.0 SPx アプリケーションを 6.5 に移行する
- NAS 4.0 アプリケーションから iPlanet Application Server 6.5 への移行
- NAS 4.0 アプリケーションから iPlanet Application Server 6.5 への移行
- NAS 2.1 アプリケーションから iPlanet Application Server 6.5 への移行

6.0 SPx アプリケーションを 6.5 に移行する

この節では、iPlanet Application Server 6.0 SPx を iPlanet Application Server 6.5 に移行するために必要な手順を説明します。

この節には次のトピックがあります。

アップグレードを始める前に

移行プロセスを開始する前に、次のことを確認してください。

- 『iPlanet Application Server 6.5 リリースノート』で説明するように、追加の OS/JDK/Compiler パッチをインストールします。

  『リリースノート』は http://docs.iplanet.com/docs/manuals/ias.html から入手可能です。

- 『iPlanet Application Server インストールガイド』に示した、ソフトウェアおよびハードウェアの要件を満たしている必要があります。

- 『iPlanet Application Server インストールガイド』に示した、iPlanet Application Server のインストールのすべての前提条件を満たしていることを確認します。

- 配置したアプリケーション (.ear、.war、.jar、および .xml ファイルや Applogic) をバックアップします。配置したアプリケーションをテンポラリディレクトリにコピーしてください。

  通常、配置されたすべての .ear あるいは .war ファイルは *iASInstallDir*`/ias/JAR` ディレクトリの下に置かれます。

- 6.0 SPx アプリケーションサーバのスクリプトファイルおよびバッチファイルに加えた変更を書きとめておいてください。

  この変更は 6.5 にもレプリケートする場合に必要です。6.0 SP1 と SP2 ユーザの場合、スクリプトファイル `iasenv.ksh` に環境パラメータの大部分が保存されていることを認識しておく必要があります。

  `iasenv.ksh` スクリプトは、*iASInstallDir*`/ias/env` ディレクトリにあります。kjs、kxs のようなほかのスクリプトに加えた変更は `iasenv.ksh` へレプリケートする必要があります。

- 配置済みアプリケーションのユーザを作成するために、LDIF ファイルをバックアップします。

  LDIF ファイルが使用できない場合は、ディレクトリサーバで ldif ファイルから db ファイルにエクスポートします。これは、バンドルされているディレクトリサーバに設定とユーザデータの両方を保存する場合に必要です。

インストール時

「アップグレードを始める前に」の作業を行ってから、iPlanet Application Server の前バージョンをアンインストールし、マシンを再起動します (Windows のみ)。

iPlanet Application Server 6.5 のインストール時に、以前のインストールと同じ設定情報を指定する必要があります。たとえば、エンジン、ポート、データベースドライバ、クラスタ情報などを同数に指定します。これにより、以前のアプリケーションサーバの設定が維持されます。

インストール手順の詳細は、『iPlanet Application Server インストールガイド』を参照してください。

インストール後のタスク

インストールが終わったら、アプリケーションを配置するため次の手順を実行します。手順は J2EE アプリケーションと Applogic アプリケーションに共通です。

- iPlanet Native (Type 2) ドライバのトランザクションサポートは使用不可能です。

  したがって、トランザクションサポートにはサードパーティドライバを使用しなければなりません。Applogic (Java/C++) には影響しません。

- グローバルトランザクションの場合、データベースをアップグレードする必要があります。Oracle を使用している場合には、Oracle 8.1.7 以降が必要です。

  『iPlanet Application Server インストールガイド』を参照してください。

- リッチクライアントサイドで JDK1.2 を使用する場合、JDK1.3 に移行する必要があります。

  以前の ORB への参照をすべて削除しなければなりません。それには、クラスパス、あるいは `ext` ディレクトリから、`iioport.jar`、`rmiorb.jar` などを削除します。

  また、`j2eeorb.jar` を ext ディレクトリにコピーするか、またはクラスパス内にコピーする必要があります。以前の iasclient.jar や orb.properties は、iPlanet Application Server 6.5 に対応したものに置き換えられます。

  詳細については、『iPlanet Application Server 開発者ガイド』を参照してください。

次の節では、各アプリケーション (J2EE と Applogic) での手順について説明します。

- J2EE アプリケーションの展開
- Applogic アプリケーションの展開

J2EE アプリケーションの展開

次の手順を実行して iPlanet Application 6.5 に J2EE アプリケーションを配置します。

- コマンドラインオプション (`deploycmd`)、または配置ツールを使用して、J2EE アプリケーションのスタブとスケルトンを再生成し、`.ear`、`.war`、および `.jar` ファイルにパッケージ化し直します。
- iPlanet Application Server 6.5 にアプリケーションを配置します。
- 正しいデータベースドライバがインストールされていることを確認します。

  インストールされていない場合は、Administration Tool を使用してデータベースドライバを作成します。詳細は『iPlanet Application Server 管理者ガイド』を参照してください。
- `iasdeploy -regdatasource` オプションを使用して、XML のデータソースを登録します。
- ディレクトリサーバに、配置されたアプリケーションのユーザを作成します。この手順は、バンドルされているディレクトリサーバを設定とユーザデータの管理に使用している場合に必要です。
- Administration Tool を使用するか、またはレジストリを編集して、iPlanet Application Server 6.5 を再設定します。

  この手順は、以前のインストールの設定を変更していて、新しいインストールにその設定をレプリケートしたい場合に必要です (オプション)。
- 必要に応じて `iasenv.ksh` (Solaris の場合) を変更するか、あるいは固有のバッチファイル (Windows の場合) を変更します (オプション)。

Applogic アプリケーションの展開

次の手順を実行して iPlanet Application 6.5 に Applogic アプリケーションを配置します。

- Solaris 上の C++ アプリケーションをコンパイルし直します。

  コンパイルエラーが発生する場合は、C++ コードが ANSI/ISO C++ 標準に従うようにアプリケーションを作り直します。Windows では、コンパイラに変更がないためコンパイルし直す必要はありません。

NAS 4.0 アプリケーションから iPlanet Application Server 6.5 への移行

NAS 4.0 では Netscape および以前の Java 標準を使っています。これらは iPlanet Application Server 6.5 では J2EE 1.3 標準に置き換えられています。廃止されたメソッドを置き換え、新しい XML 記述子を使ってアプリケーションを再配置する必要があります。このプロセスについては、便利なツールが用意されています。詳細は、第 5 章「NAS 4.0 アプリケーションの実行」を参照してください。

NAS 2.1 アプリケーションから iPlanet Application Server 6.5 への移行

移行するには、NAS 2.1 プログラミングモデル用に記述されたアプリケーションを変更して、iPlanet Application Server 6.5 プログラミングモデルに適合させる必要があります。このプロセスには 3 つの方法があります。このマニュアルでは、それぞれの方法が説明されています。

- 移行しない **:** この方法では、開発者は何も行う必要がなく、サーバが後方互換性をサポートしているかどうかに依存します。新しいプログラミングモデルの柔軟性および性能を利用しない場合は、この方法で間に合わせることができます。ただし、NAS 2.1 でサポートされていた API の多くは廃止されており、今後のリリースでもサポートされない可能性があります。

  後方互換性については、第 3 章「NAS 2.1 アプリケーションの実行」を参照してください。

- 部分的移行 **:** この方法では、アプリケーションの一部を新しいプログラミングモデルに適合させます。残りの部分は後方互換性に依存します。この方法では、開発者は、アプリケーションの判明している部分およびテスト済みの部分を維持する一方で、アプリケーションを一部分ずつ (たとえば、1 つのユーザ対話レベルまたは 1 プログラミング層ずつ) 移行できます。

  iPlanet Application Server 6.5 は、以前のコンポーネントと新しいコンポーネント間の「接着手段」を提供することによって、部分的移行をサポートしています。このサポートについては、第 4 章「NAS 2.1 アプリケーションの移行」の「コンポーネントの部分的移行」を参照してください。

- 新しいプログラミングモデルへの全面的移行 **:** この方法では、多くの開発リソースと全体的な設計の見直しが必要になりますが、アプリケーションは新しいプログラミングモデルの機能をフルに活用できるようになります。

  この方法については、第 4 章「NAS 2.1 アプリケーションの移行」を参照してください。

第 3 章

NAS 2.1 アプリケーションの実行

この章では、ソースレベルの変更を行わずに iPlanet™ Application Server 6.0 上で NAS 2.1 アプリケーションを実行する方法について説明します。オンラインバンクサンプルのセットアップについては、この章の最後に説明します。iPlanet Application Server 6.5 上でユーザのアプリケーションを実行する前に、バンクのサンプルをセットアップすることをお勧めします。バンクのサンプルは移行プロセスを順を追って最後まで理解するのに役立ちます。

iPlanet Application Server 6.5 は NAS 2.1 と後方互換性があります。つまり、コードを変更しなくても以前の NAS 2.1 アプリケーションを iPlanet Application Server 6.5 上に配置できます。ただし、実装には特定の手順が必要です。たとえば、新しいサーバに配置する前に C++ アプリケーションおよびエクステンションをコンパイルし直す必要があります。28 ページの「C++ アプリケーションおよびエクステンション」を参照してください。また、使用する JDBC のバージョンへの正しいクラスパスも必要です。

この章には次の節があります。

- NAS 2.1 のアプリケーションコンポーネント
- NAS 2.1 アプリケーションの配置
- Java エクステンション
- C++ アプリケーションおよびエクステンション
- 移行プロセスの開始
- サンプルアプリケーションの移行

NAS 2.1 のアプリケーションコンポーネント

この節では、2.1 プログラミングモデルの主な各コンポーネントタイプに対する iPlanet Application Server 6.5 でのサポートについて説明します。このサポートについては、次の節で説明します。

- HTML テンプレート
- AppLogics
- データベースロジック : DAE と JDBC
- NAS レジストリ

HTML テンプレート

プレゼンテーションレイアウトでは、GX タグを含む NAS 2.1 スタイルの HTML テンプレートが全面的にサポートされており、NAS テンプレートエンジンによる変更の必要はありません。ただし、テンプレートが Servlet によって呼び出されると、テンプレートは JSP としてコンパイルされます。JSP では、階層型クエリを除く GX タグがサポートされています。

AppLogics

Java 標準が新しいプログラミングモデルのベースとなり、NAS 2.1 で採用されていた専用の API の多くは廃止されましたが、iPlanet Application Server 6.5 でも AppLogic フレームワークは全面的にサポートされています。詳細は、『iPlanet Application Server Foundation Class Reference』を参照してください。

データベースロジック : DAE と JDBC

NAS 2.1 のデータベースアクセスクラスおよびインターフェイスは廃止され、Java 標準のデータベースコネクション API である JDBC が採用されています。iPlanet Application Server 6.5 では、NAS 2.1 のデータベースコネクションおよびクエリメソッドを使用するコードがサポートされていますが、今後のリリースではサポートされない可能性があります。

新しい JDBC レイヤは以前の 2.1 JDBC レイヤと同じ機能を持っており、多くの新規メソッドがサポートされています。その結果、一部の AppLogic コードを変更して、回避策を削除したり新規 JDBC 呼び出しを追加したりできます。

AppLogic では、次のいずれかの JDBC レイヤを使う必要があります。

- 新しい JDBC レイヤ
- 以前の JDBC レイヤのどちらかを使いますが、両方は使えません。Servlet および EJB では、新しい JDBC レイヤのみを使います。各バージョンの JDBC 呼び出しの併用はサポートされていません。

iPlanet Application Server 6.5 の JDBC レイヤの代わりに NAS 2.1 の JDBC AppLogic を使えますが、JDBC 1.2 ではなく JDBC 2.0 のインターフェイスが JVM に読み込まれていることを確認する必要があります。たとえば、次のようなログメッセージが返された場合、CLASSPATH 内の JDBC 2.0 インターフェイスの前に JDBC 1.2 インターフェイスがある可能性があります。

```
[01/05/99 11:25:51:0] error:APPLOGIC-caught_exception:Caught
Exception:
java.lang.NoSuchMethodError:java.sql.Statement:method
addBatch(Ljava/lang/String;)V not found
```

NAS レジストリ

大部分の NAS レジストリの場所は変わっていませんが、一部は LDAP ディレクトリ内にあります。詳細は、『管理および配置ガイド』を参照してください。

NAS 2.1 アプリケーションの配置

すべてのアプリケーションを iPlanet Application Server に配置するには、iPlanet Application Server Administrator Tool を使います。詳細は、『管理および配置ガイド』を参照してください。

Java エクステンション

Java NAS 2.1 エクステンションを iPlanet Application Server 6.5 に移行するには、次の手順を実行します。

1. iPlanet Extension Builder 6.5 に IDL コードを読み込み、新規生成コードを作成します。
2. 以前のエクステンションに対して行った変更を新規生成コードにマージします。
3. NMI に対するすべてのリファレンスを JNI に変換します (該当する場合のみ)。
4. ほかのすべての Java コードを JDK 1.30.1 に変更します。
5. すべてのコードをコンパイルし直します。

C++ アプリケーションおよびエクステンション

iPlanet Application Server 6.5 には新バージョンの必須 C++ ヘッダファイルが用意されています。このため、新しいヘッダファイルを使って C++ アプリケーションおよびエクステンションをコンパイルし直す必要があります。iPlanet Application Server 6.5 ライブラリに対するエクステンションを再度コンパイルし、リンクする必要もあります。

iPlanet Application Server には、MQSeries、TUXEDO、CICS を含む一部のレガシーシステム用のビルド済みエクステンションが用意されています。これらのエクステンションは iPlanet Application Server 6.5 をサポートするために再度リリースされましたが、NAS 2.1 プログラミングモデルも引き続きサポートされます。

移行プロセスの開始

アプリケーションを新しいモデルに移行する場合、アプリケーションの再設計と移行計画の作成から開始するのが最も簡単な方法です。多くの場合、アプリケーション全体の大規模な移行計画を立てるよりも、アプリケーションを一部ずつ新しいプログラミングモデルに移行する方が簡単です。また、アプリケーションコンポーネントごとでなく、プレゼンテーションレイアウトやビジネスロジックなどのプログラミング層ごとに段階的に移行することもできます。

2.1 コンポーネント (AppLogic など) と 6.5 コンポーネント (Servlet や EJB) 間の対話を可能にする部分的移行については、第 4 章「NAS 2.1 アプリケーションの移行」の「コンポーネントの部分的移行」を参照してください。

サンプルアプリケーションの移行

この節では、次のサンプルアプリケーションを移行する方法について説明します。

- OnlineBank のサンプル
- US Population Java サンプルアプリケーション
- US Population C++ サンプルアプリケーション

OnlineBank のサンプル

このサンプルは次の節に分かれています。

- NAS 2.1 から iPlanet Application Server 6.5 への移行 - Solaris のみ
- C++ COnlineBank (NAS2.1) サンプルアプリケーション

NAS 2.1 から iPlanet Application Server 6.5 への移行 - Solaris のみ

iPlanet Application Server 6.5 には配布物の一部として JDK 1.3.1_02 が含まれています。JDK 1.1 では classes.zip でしたが、新しい JDK 1.2 では、`java.io` や `java.lang` などの Java のすべてのコアパッケージが `rt.jar` に含まれています。`rt.jar` は次の場所にあります。

```
/iAS6.5-install-directory/ias/usr/java/jre/lib
```

java クラスをコンパイルする場合、`rt.jar` を `CLASSPATH` に置いて、/iAS6.0-install-directory/ias/usr/java/bin ディレクトリにある `javac` を使ってコンパイルする必要があります。

NAS 2.1 から iPlanet Application Server 6.5 (Solaris プラットフォーム) に移行するには、次の手順を実行します。

1. OnlineBank java Application パッケージを iPlanet Application Server 6.5 がインストールされているマシンにコピーします。これは、次のように `GXAPP` の下に新しい OnlineBank ディレクトリを作成して行います。

   ```
   /iAS6.5-install-directory/ias/APPS/GXApp/OnlineBank
   ```

2. Web サーバをインストールしたディレクトリ内の docs/GXApp の下に OnlineBank という新しいディレクトリを作成します。

   ```
   /Netscape/Suitespot/docs/GXApp/OnlineBank
   ```

   さらに、NAB 2.1 Web サーバの docs ディレクトリにある OnlineBank のすべての HTML ファイルをこの OnlineBank ディレクトリにコピーします。

3. `tnsnames.ora` に `ksample` のエントリを作成します。`ksample` が、8.1.7 で動作する Oracle データベースを指定していることを確認します。

4. `kreg` ユーティリティを使って登録します。

   ディレクトリを `/iAS6.5-install-directory/ias/APPS/GXApp/OnlineBank` に変更し、`/iAS6.5-install-directory/ias/bin/kreg OnlineBank.gxr` を実行します。

   以前の 2.1 バージョンではなく、iPlanet Application Server 6.5 kreg を実行していることを確認します。

5. サンプルアプリケーション
http://hostname/GXApp/OnlineBank/OBLogin.html を実行し、アカウントバランスおよびその他のオプションを確認します。

C++ COnlineBank (NAS2.1) サンプルアプリケーション

C++ COnlineBank (NAS2.1) サンプルアプリケーションを実行するには、次の手順に従います。

1. COnlineBank Application を iPlanet Application Server 6.5 がインストールされているマシンにコピーします。これは、GXApp の下に新しいディレクトリ COnlineBank を作成して行います。

   ```
   /iAS6.5-install-directory/ias/APPS/GXApp/ConlineBank
   ```

2. Web サーバをインストールしたディレクトリ内の docs/GXApp の下に COnlineBank という新しいディレクトリを作成します。

   /Netscape/Suitespot/docs/GXApp/COnlineBank

   NAB 2.1 Web サーバの docs ディレクトリにある COnlineBank のすべての HTML ファイルを、作成した COnlineBank ディレクトリにコピーします。

3. 次の 2 つの環境変数を設定します。

   setenv GX_ROOTDIR /iAS6.5-install-directory/ias

   setenv GX_ROOT /iAS6.5-install-directory/ias

4. `tnsnames.ora` に `ksample` のエントリを作成します。`ksample` が、8.1.7 で動作する Oracle データベースを指定していることを確認します。

5. 提供されている makefile で make を実行します。

   /usr/ccs/bin/make -f makefile

   ここで /usr/ccs/bin は makefile が存在するディレクトリです。このアクションによって、生成された libCOnlineBank.so ファイルが /iAS6.5-install-directory/ias/gxlib ディレクトリにコピーされます。

6. `kreg` ユーティリティを使って登録します。

   ディレクトリを `/iAS6.5-install-directory/ias/APPS/GXApp/COnlineBank` に変更し、`/iAS6.5-install-directory/ias/bin/kreg COnlineBank.gxr` を実行します。

   以前の NAS 2.1 バージョンではなく、新しい iPlanet Application Server 6.5 kreg を実行していることを確認します。

7. サンプルアプリケーション
http://hostname/GXApp/COnlineBank/COBLogin.html を実行し、アカウントバランスおよびその他のオプションを確認します。

US Population Java サンプルアプリケーション

US Population Java サンプルを NAS 2.1 から iPlanet Application Server 6.5 に移行するには、次の手順を実行します。

1. US Population Java アプリケーションパッケージを iPlanet Application Server 6.5 インストール済みのマシンの States という新規ディレクトリにコピーします。

   GXAPP/IAS6.5-install-directory/ias/APPS/GXApp/States

2. Web サーバをインストールしたディレクトリにある docs/GXApp の下に States という新しいディレクトリを作成します。

   /Netscape/Suitespot/docs/GXApp/States

   さらに、この States という名前のディレクトリに NAB 2.1 Web サーバの docs ディレクトリにある US Population アプリケーションからすべての HTML ファイルをコピーします。

3. `tnsnames.ora` に `ksample` のエントリを作成します。`ksample` が、8.1.7 で動作する Oracle データベースを指定していることを確認します。

4. kreg ユーティリティを使って登録します。

   ディレクトリを /IAS6.5-install-directory/ias/APPS/GXApp/States に変更し、/IAS6.5-install-directory/ias/bin/kreg `states.gxr` を実行します。

   以前の 2.1 バージョンではなく、iPlanet Application Server 6.5 `kreg` を実行していることを確認します。

5. US Population アプリケーションを実行します。

   http://hostname/GXApp/States/index.html

   人口統計を表示するには、`RunRegionReport` をクリックします。

US Population C++ サンプルアプリケーション

US Population C++ サンプルを NAS 2.1 から iPlanet Application Server 6.5 に移行するには、次の手順を実行します。

1. US Population C++ アプリケーションを iPlanet Application Server 6.5 インストール済みのマシンにコピーします。GXAPP の下の CStates という新規ディレクトリにコピーします。

   /IAS6.5-install-directory/ias/APPS/GXApp/Cstates

2. Web サーバをインストールしたディレクトリ内の docs/GXApp の下に CStates という新しいディレクトリを作成します。

   /Netscape/Suitespot/docs/GXApp/CStates

   さらに、NAB 2.1 Web サーバのマニュアルのディレクトリにある US Population のすべての HTML ファイルを、作成した CStates ディレクトリにコピーします。

3. `tnsnames.ora` に `ksample` のエントリを作成します。`ksample` が、8.1.7 で動作する Oracle データベースを指定していることを確認します。

4. states.mak ファイルに対して `nmake` (`nmake -f states.mak`) を実行します。

5. 生成された states.dll を /IAS6.5-install-directory/ias/bin ディレクトリにコピーします。

6. `kreg` ユーティリティを使って登録します。

   ディレクトリを /IAS6.5-install-directory/ias/APPS/GXApp/CStates に変更し、/IAS6.5-install-directory/ias/bin/kreg `states.gxr` を実行します。

   以前の 2.1 バージョンではなく、iPlanet Application Server 6.5 `kreg` を実行していることを確認します。

7. US Population アプリケーションを実行します。

   http://hostname/GXApp/CStates/index.html

   人口統計を表示するには、`RunRegionReport` をクリックします。

第 4 章

NAS 2.1 アプリケーションの移行

この章では、NAS 2.1 アプリケーションを変更して、iPlanet Application Server 6.5 プログラミングモデルに適合させる方法について説明します。

この章には次の節があります。

- アプリケーションの再設計
- プレゼンテーションロジックの移行
- ビジネスロジックの移行
- データアクセスロジックの移行
- コンポーネントの部分的移行

アプリケーションの再設計

既存のアプリケーションを設計し直す場合は、ある部分に対して行った変更によってほかの部分も影響を受けることを念頭におくことが重要です。

アプリケーションを次のいずれかのモデルとして考えると有益です。

- 目的を達成するための一連のユーザ対話。たとえば、オンライン調査または標準化されたテストなど
- 中心となるフロントページを持つ業務情報センター。たとえば、複数の業務 (引き出しや振り込みなど) に通じている中心のページを持つオンラインバンクなど

実際には、アプリケーションはこれらの 2 つの組み合わせです。たとえば、オンラインバンクは中央業務情報センターであり、そこからの各経路は目的を達成するための一連のユーザ対話に通じています。

ただし、アプリケーションが細分化されている場合でも、一部分ずつ移行することが多くの場合で最適な方法です。詳細については、38 ページの「コンポーネントの部分的移行」を参照してください。

プレゼンテーションロジックの移行

この節では、次の概念について説明します。

- Servlet としての AppLogic の再作成
- プレゼンテーションレイアウトの再作成
- セッションとセキュリティの再作成

Servlet としての AppLogic の再作成

AppLogic は Servlet に直接マッピングされます。AppLogic と Servlet はともに、URL によって呼び出されます。また、入力の処理および出力の生成を行うメカニズムを持っています。コード自体のレイアウトを除く主な相違点は、AppLogic がビジネスロジックを実行するのに対して通常、Servlet は実行しないという点です。ビジネスロジックは EJB で処理され、Servlet によって参照されます。同様に、プレゼンテーションレイアウトは JSP で処理され、Servlet によって参照されます。つまり、Servlet とは、別個のエンティティにビジネスロジックが再実装された AppLogic のようなものです。

Servlet については、『開発者ガイド』の第 3 章「Servlet によるアプリケーションの制御」を参照してください。

Servlet は `service()` メソッドを含んでいる必要があります。HTTP Servlet では、HTTP 転送方式に応じて `doGet()`、`doPost()` などとして実装されていることもあります。このメソッドは、AppLogic の `execute()` メソッドと論理的にほぼ同じです。これはコンポーネントの主な実行フローです。

さらに、AppLogic では、受信データを格納する `IValList` メンバー変数が iPlanet Application Server によって作成されるのに対し、Servlet では、iPlanet Application Server によってリクエストオブジェクトが作成され、パラメータとして Servlet に渡されます。同様に、AppLogic では `IValList` を使って出力するのに対し、Servlet ではレスポンスオブジェクトを使って出力し、パラメータとして Servlet に渡します。次のコードサンプルは両方のケースを示しています。

AppLogic

```
public class MyAppLogic extends AppLogic {
    public void execute () throws IOException {
    ...
    String lastName = valIn.getValString("lastName");
    ...
    return result ("<html><body>¥n"
```

```
                + "<p>Your last name is " + lastName + ".¥n"
                + "</body></html>¥n");
    }
}
```

Servlet

```
public class myServlet extends HttpServlet {
public void service (HttpServletRequest req,
                     HttpServletResponse res)
            throws IOException, ServletException
    {
    ...
    res.setContentType("text/html");
    String lastName = req.getParameter("lastName");
    ...
    PrintWriter output = res.getWriter();
    output.println("<html><body>¥n");
                + "<p>Your last name is " + lastName + ".¥n"
                + "</body></html>¥n");
    }
}
```

JSP および Servlet は観点が異なるほぼ同じエンティティであるため、AppLogic を JSP として再実装することもできます。次のようにします。

```
<html><body>
<p>Your last name is <display property="request:params:lastName">.
</body></html>
```

Servlet については、『開発者ガイド』の第 4 章「JavaServer Pages (JSP) によるアプリケーションページの表示」を参照してください。

プレゼンテーションレイアウトの再作成

ある意味では、2.1 HTML テンプレートはすでに移行されています。iPlanet Application Server 6.5 テンプレートエンジンは、このテンプレートが JSP であるかのように簡単にコンパイルします。新しいテンプレートエンジンには互換性があり、階層型クエリを除く GX タグをサポートしています。

ただし、JSP では GX タグのサポートが廃止されているため、これらのテンプレートを変換して標準の JSP タグおよびシンタックスを使う必要があります。JSP は Beans を使って出力パラメータをカプセル化するため、任意の Java オブジェクトにもアクセスできます。JSP から EJB に直接アクセスすることもできます。ただし、通常は、Servlet の実行時にリクエストオブジェクト内の属性を設定し、JSP 内でそれらの属性を呼び出します。

例を含む JSP の詳細については、『開発者ガイド』の第 4 章「JavaServer Pages (JSP) によるアプリケーションページの表示」を参照してください。

セッションとセキュリティの再作成

iPlanet Application Server 6.5 セッションでは `HttpSession` インターフェイスが使われます。API は異なりますが、その概念は NAS 2.1 セッションと似ています。Servlet (または AppLogic) によってセッションが作成され、これによってユーザセッションが存在する限り持続するセッションオブジェクトがインスタンス化されます。セッション cookie はクライアントに返され、次にクライアントと対話するときに再度読み込まれます。セッションが存在すれば、そのセッションにオブジェクトをバインドできます。

Servlet のセキュリティは変更されています。詳細については、『開発者ガイド』を参照してください。

ビジネスロジックの移行

iPlanet Application Server 6.5 では、AppLogic ではなく、Enterprise JavaBeans (EJB) によってビジネスロジックが処理されます。AppLogic と EJB の重要な相違点は、EJB はユーザとの「セッション」の間持続させることができ、セッション Beans の場合はユーザのセッションとは別に指定されるという点です。エンティティ Beans はユーザと無関係に存在するため、サーバが存在する限り持続する可能性があります。

別々のタスクを実行するこれらの EJB を記述し、Servlet からそれらの Beans に接続します。たとえば、電子ショッピングカートを利用する場合にこの方法を使います。

JDBC とトランザクションサポートの詳細については、第 7 章「EJB のトランザクション処理」、および『iPlanet Application Server 開発者ガイド』の第 8 章「JDBC を使ったデータベースアクセス」を参照してください。

データアクセスロジックの移行

この節では、JDBC API を使ったデータベース呼び出しの再配置について説明します。

iPlanet Application Server 6.5 の JDBC レイヤは、JDBC 2.0 の仕様および標準エクステンションのすべてをサポートしています。

JDBC とトランザクションサポートの詳細については、第 8 章「EJB のトランザクション処理」、および『開発者ガイド』の第 9 章「JDBC を使ったデータベースアクセス」を参照してください。

`$GX_ROOTDIR/solarisdbg/JDK_1.1/java` (または同様のディレクトリ) の JDBC 2.0 インタフェースは、`CLASSPATH` のほかの JDBC インタフェースよりも前に記述されている必要があります。iPlanet Application Server 6.5 は `$JAVA_HOME/lib/rt.jar` に JDBC 2.0 インタフェースがある JDK 1.1 とともに動作するので、この `rt.jar` は iPlanet Application Server のクラスよりも後になるようにします。次のようになります。

```
setenv CLASSPATH
:$GX_ROOTDIR/solarisdbg/JDK_1.1:...:$JAVA_HOME/lib/rt.jar:...
```

不適合エラー

次のようなログメッセージが返される場合、CLASSPATH 内の JDBC 2.0 インターフェイスの前に JDBC 1.2 インターフェイスがある可能性があります。

```
[01/05/99 11:25:51:0] error:APPLOGIC-caught_exception:Caught
Exception:
java.lang.NoSuchMethodError:java.sql.Statement:method
addBatch(Ljava/lang/String;)V not found
```

コンポーネントの部分的移行

この節では、以前のコンポーネント (Java および C++ AppLogic) を新しいコンポーネント (Servlet および EJB) とともに使う方法について説明します。次の 4 つの組み合わせがサポートされています。

- Java AppLogic からの EJB の呼び出し
- Java AppLogic からの Servlet の呼び出し
- Servlet からの Java AppLogic の呼び出し
- Servlet からの C++ AppLogic の呼び出し

Java AppLogic からの EJB の呼び出し

Servlet と EJB 間で共有される特別なコンテキストはないため、Servlet から呼び出す場合と同じ方法で AppLogic から EJB を呼び出します。

次の例は、ShoppingCart という EJB にアクセスする AppLogic を示しています。AppLogic は、カートのリモートインターフェイスをインポートしてからユーザのセッション ID を ShoppingCart に割り当てることによって、カートに対するハンドルを作成します。カートはユーザのセッション内に保存されます。

```
import cart.ShoppingCart;
% // ユーザのセッションおよびショッピングカートを取得します。
    // まずセッションを作成します。
    ISession2 sess = createSession(GXSESSION.GXSESSION_DISTRIB,
                                   0,           //no timeout
                                   "callEjb",    // アプリケーション名
                                   null,        // システム生成 ID
                                   null);

    //IValList を作成してセッションのショッピングカートに保存します。
    IValList ival = sess.getSessionData();

    ShoppingCart cart = (ShoppingCart)ival.getVal("shoppingCart");

    // ユーザがカートを持っていない場合は新規に作成します。
    if (cart == null) {
        cart = new ShoppingCart();
        ival.setVal("shoppingCart", cart);
    }
```

Java Naming Directory Interface (JNDI) を使って EJB に対するハンドルまたはプロキシを確立することによって、EJB にアクセスできます。その後は EJB を正規オブジェクトとして参照でき、すべてのオーバーヘッドはビーンのコンテナによって管理されます。

次の例は、ショッピングカートのプロキシを検索するための JNDI の使用法を示しています。

```
String jndiNm = "Bookstore/cart/ShoppingCart";
javax.naming.Context initCtx;
Object home;
    try {
         initCtx = new javax.naming.InitialContext(env);
    } catch (Exception ex) {
          return null;
    }
    try {
          java.util.Properties props = null;
          home = initCtx.lookup(jndiNm);
    }
    catch(javax.naming.NameNotFoundException e)
    {
          return null;
    }
    catch(javax.naming.NamingException e)
    {
          return null;
    }
    try {
          IShoppingCart cart = ((IShoppingCartHome) home).create();
          ...
} catch (...) {...}
```

Java AppLogic からの Servlet の呼び出し

AppLogic から JSP を呼び出す場合などに、`GXContext.NewRequest()` または `GXContext.NewRequestAsync()` を使って Java AppLogic から Servlet を呼び出すことができます。`NewRequest()` の詳細および具体例については、『iPlanet Application Server Foundation Reference』の `GXContext` クラスの説明を参照してください。

インスタンス化すると、JSP および Servlet は同じタイプのオブジェクトであるため、AppLogic から JSP を呼び出すこともできます。

次の例は、Servlet の Servlet エンジン (`ServletRunner` という AppLogic) と同じプロセス呼び出しを使って AppLogic から Servlet を呼び出す方法を示しています。

```
class SomeApplogic extends Applogic {
   int execute() {
      valIn.setValString("appName","nsOnlineBank");
      valIn.setValString("servletName","Login");
      valIn.setValString("SCRIPT_NAME","nsOnlineBank/Login");
      com.netscape.servlet.servletrunner.ServletRunner sr =
         new
com.netscape.server.servlet.servletrunner.ServletRunner();
      sr.valIn = valIn;
      sr.valOut = valOut;
      sr.context = context;
      sr.stream = this.stream;
      sr.ticket = this.ticket;
      sr.request = this.request;
      sr.COMSet(COMGet());
      sr.COMAddRef();
      sr.execute();
      ...
   }
}
```

次の例は、NewRequest() を使って新しいプロセス内で AppLogic から Servlet を呼び出す方法を示しています。

```
class SomeApplogic extends Applogic {
    int execute() {
        valIn.setValString("appName","nsFortune");
        valIn.setValString("servletName","fortune");
        valIn.setValString("SCRIPT_NAME","nsOnlineBank/Login");
        retValue = GXContext.NewRequest(m_Context,
                                    "ApplogicServlet_nsFortune_fortu
ne",
                                    valIn,valOut,host,port,0);
        ...
    }
}
```

次の例のように、ほとんど同じ方法で JSP を呼び出すことができます。

```
public class SomeApplogic extends Applogic {
    int execute() {
        valIn.setValString("appName","System");
        valIn.setValString("servletName","JSPRunner");
        valIn.setValString("JSP","nsOnlineBank/jsp/abc.jsp");
        valIn.setValString("SCRIPT_NAME","nsOnlineBank/Login");
        retValue =
            GXContext.NewRequest(m_Context,
                                "Applogic
Servlet_System_JSPrunner",
```

```
                                        valIn,valOut,host,port,0);
    ...
    }
}
```

次の例は、GUID を使って Servlet を呼び出す方法を示しています。

```
public class SomeApplogic extends Applogic {
    int execute() {
        valIn.setValString("appName","nsFortune");
        valIn.setValString("servletName","fortune");
        newRequest("{6F3547D0-FDCB-1687-B323-080020A16896}",
                   valIn,valOut,0);
    }
}
```

Servlet からの Java AppLogic の呼び出し

`GXContext.NewRequest()` または `GXContext.NewRequestAsync()` を使って、Servlet から AppLogic を呼び出すことができます。詳細および具体例については、『iPlanet Application Server Foundation Class Reference』の `GXContext` クラスの説明を参照してください。

`NewRequest()` を使って AppLogic を呼び出すには、サーバのコンテキストを `IContext` オブジェクトに割り当ててから AppLogic の入出力 `IValList` オブジェクトを設定します。

次の例は、`IContext` オブジェクトを取得して、AppLogic のパラメータを設定してから、最後に `NewRequest()` を使って AppLogic を呼び出す方法を示しています。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.kivasoft.applogic.*;
import com.kivasoft.types.*;
import com.netscape.server.servlet.extension.*;

public class callAnAppLogic extends HttpServlet {

    public void service(HttpServletRequest req,
                        HttpServletResponse res)
        throws ServletException, IOException
    {
        // IContext のハンドルである ic をまず設定します。
        ServletContext sctx = getServletContext();
        com.netscape.server.IServerContext isc;
```

```
        isc = (com.netscape.server.IServerContext) sctx;
        com.kivasoft.IContext ic = isc.getContext();

        // IValLists と GUID を設定します。
        IValList vi = GX.CreateValList(); // valIn
        valIn.setValString("randomParameter", "Cirdan the
Shipwright");

        IValList vo = GX.CreateValList(); // valOut

        String al = req.getParameter("AppLogicToCall");
                    // AppLogicToCall のリクエストを待ちます。

        //最後に AppLogic を呼び出します。
        GXContext.NewRequest(ic, al, vi, vo, 0);
    }
}
```

Servlet の AppLogic へのアクセス

各 Servlet は AppLogic 内に含まれています。iPlanet Application Server 機能インターフェイス `HttpServletRequest2` でメソッド `getAppLogic()` を使って、Servlet を制御する AppLogic インスタンスにアクセスすることができます。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.kivasoft.applogic.*;
import com.kivasoft.types.*;
import com.netscape.server.servlet.extension.*;7

public class callAnAppLogic extends HttpServlet {

    public void service(HttpServletRequest req,
                        HttpServletResponse res)
        throws ServletException, IOException
    {
        HttpServletRequest2 req2 = (HttpServletRequest2)req;
        AppLogic al = req2.getAppLogic();
        //al はスーパークラスへのハンドルです。
        ...
    }
}
```

Servlet からの C++ AppLogic の呼び出し

39 ページの「Java AppLogic からの Servlet の呼び出し」で説明したように、メソッド `GXContext.NewRequest()` は GUID によって AppLogic を呼び出し、オブジェクトに対するハンドルを入出力パラメータとして提供します。AppLogic は Java 特有のハンドルによって呼び出されるのではなく、特定の名前または GUID によって呼び出されるため、このメソッドは Java AppLogic だけでなく C++ AppLogic も呼び出します。「Servlet からの Java AppLogic の呼び出し」で示した例を参照してください。

部分的に移行したアプリケーションのセッション

`HttpSession2` インターフェイスは、セッションオブジェクトに直接アクセスできるようにする追加のセッションインターフェイスです。このインターフェイスを使うと、AppLogic と Servlet 間でセッション (および結果的にデータ) を共有できます。

Servlet では、セッションは HttpSession のインスタンスです。これに対し、AppLogic では、セッションデータは `IValList` オブジェクトです。AppLogic では Integer、String、および blobs (バイト配列) がセッションに保存されるのに対し、Servlet では直列化可能なオブジェクトがセッションに保存されます。その結果、AppLogic でセッションに保存されるものと Servlet でセッションに保存されるものとの間に直接のマッピングはありません (文字列を除く)。

`HttpSession2` インターフェイスによって、セッションデータ共有の問題が解決されます。`HttpSession2` には Integer、String、blobs、およびユーザログインデータの保存および取得を行うメソッドがあります。これらのメソッドは AppLogic 開発者が使うものと類似するメソッドです。このように、`HttpSession2` を使うと、セッションは AppLogic と Servlet 間で交互に動作させることができます。

`HttpSession2` の `loginSession( )` および `logoutSession( )` によって、Servlet で AppLogic セッション API を共有することができます。これらのメソッドは iPlanet Application Server 6.5 では廃止されているものです。AppLogic の場合と同様に、これらの 2 つのメソッドは通常、`isAuthorized( )` とともに使われます。また、Servlet はアクセス制御リストに登録されるため、AppLogic で確立された安全なセッションを Servlet で使うことができ、その逆も可能です。

詳細については、『開発者ガイド』の第 12 章「安全なアプリケーションの作成」を参照してください。

セッションの表示

セッションが cookie によって制御されるため、デフォルトでは AppLogic で作成されたセッションは Servlet に表示されません。これは cookie がドメインおよび URI に依存しており、Servlet の URI が AppLogic の URI と異なるためです。この問題を回避するには、AppLogic でセッションを作成するとき、`saveSession()` を呼び出す前に `setSessionVisibility()` を呼び出します。

セッションの保存によってもセッション cookie が作成されるため、`saveSession()` を呼び出す前に `setSessionVisibility()` を呼び出すことが重要です。

たとえば、AppLogic では次の例について検討します。

```
domain=".mydomain.com";
path="/"; // すべてのドメインに表示します
isSecure=true;
if ( setSessionVisiblity(domain, path, isSecure) == GXE.SUCCESS )
    { // セッションはすべてのドメインに表示されています }
```

セッションの詳細については、『開発者ガイド』の第 11 章「ユーザセッションの作成と管理」を参照してください。

ITemplateData から ResultSet への変換

NAS 2.1 では、HTML テンプレート処理に使われる階層型データソースを表現するための ITemplateData というインターフェイスが用意されていました。NAS 2.1 の `ITemplateData` (Java) は、一組のメモリベースの階層型データ内で行を繰り返したり、列の値の取得を行うメソッドを提供しています。iPlanet Application Server 6.5 ではこの機能はサポートされていませんが、グループ名はサポートされています (必須)。

iPlanet Application Server 6.5 では `ITemplateData` 機能が JDBC ResultSet オブジェクトに置き換えられています。`BaseUtils` クラスからメソッド `convertITemplateDataToResultSet()` を使って `ITemplateData` オブジェクトを `ResultSet` オブジェクトに変換できます。具体的な使用法については、『iPlanet Application Server Foundation Class Reference』の `BaseUtils` クラスの説明を参照してください。次の例は、AppLogic における `ITemplateData` の `ResultSet` への変換を示しています。変換メソッドにパラメータとしてデータグループ名を渡す必要があります。

```
ITemplateData itd = GX.CreateTemplateDataBasic("myTemplateData");
... // myTemplateData を設定します。
...
ResultSet rs =
BaseUtils.convertITemplateDataToResultSet("dataGroup1",itd);
```

第 5 章

NAS 4.0 アプリケーションの実行

この章では、Netscape Application Server 4.0 アプリケーションを iPlanet Application Server 6.5 で実行するための基本的な移行手順について説明します。

この章には次の節があります。

- 概要
- NAS 4.0 と iPlanet Application Server 6.5 の相違点
- NAS 4.0 コンポーネントの移行
- 「Bank」の移行例

概要

iPlanet Application Server 6.5 は、J2EE 1.2 (Java 2 Platform, Enterprise Edition Specification Version 1.2) への準拠が確認されています。iPlanet Application Server 6.5 のアーキテクチャが NAS 4.0 と同じである一方で、J2EE 標準に 100% 準拠ということは、アプリケーションを実行するには、アプリケーションが J2EE 1.2 に準拠する必要があることを意味します。

移行作業は、廃止された J2EE および NAS 独自のメソッドにアプリケーションがどの程度依存しているかに左右されます。配置および JavaServer Pages には変換プロシージャが必要です。このプロシージャのためのツールが用意されています。一般に、ここでの移行作業は廃止されたメソッドを置き換え、変換および再配置することです。アプリケーションに廃止されたメソッドがあるかどうかを確認する 1 つの方法として、アプリケーションをコンパイルし直す方法があります。

この章で説明する手順に従って、アプリケーションの移行を開始します。iPlanet Web サイトにあるオンラインで利用可能な「Bank」移行例を実行してみると、役に立つ場合があります。詳細は、http://www.iPlanet.com/support にアクセスしてください。NAS 4.0 アプリケーションを iPlanet Application Server 6.5 で実行するための段階的な移行手順が記載されています。

NAS 4.0 と iPlanet Application Server 6.5 の相違点

次の表は、NAS 4.0 と iPlanet Application Server 6.5 のコンポーネントの主な相違点を示しています。これらの各コンポーネントの相違点および移行プロシージャについては、次の表に従ってください。

コンポーネント	NAS 4.0	移行方法	作業レベル	iPlanet Application Server 6.5
JDK	JDK 1.1.7	「JDK の移行」の節を参照	中	JDK 1.30.1
Servlet	Servlet 2.1	「Servlet の移行」の節を参照	低 / なし	Servlet 2.2
Servlet の配置	NTV (Name Type Value) 配置記述子を使用	ツールを使って XML に変換「Servlet の配置」の節を参照	中	XML 記述子を使用
JSP	JSP 0.92	「JSP の移行」の節を参照	中	JSP 1.1
EJB	EJB 1.0	「EJB の移行」の節を参照		EJB 1.1
EJB の配置	プロパティファイル配置記述子を使用	ツールを使って XML に変換「EJB の配置」の節を参照	中	XML 記述子を使用
JNDI	JNDI 1.1	「JNDI の移行」の節を参照		JNDI 1.2
セキュリティ	ACL チェック	「セキュリティ機能」の節を参照	低	宣言に基づく

NAS 4.0 コンポーネントの移行

この節では、さまざまな NAS 4.0 コンポーネントを iPlanet Application Server 6.5 に移行するための必要条件について説明します。この節には次のトピックがあります。

- 基本的な移行手順
- JDK の移行
- サーブレットの移行
- Servlet の配置
- JSP の移行
- EJB の移行
- EJB の配置
- JNDI の移行
- Java エクステンション
- C++ エクステンション
- リッチクライアントの使用 (ISecurity インターフェイス)

基本的な移行手順

NAS 4.0 コンポーネントを iPlanet Application Server に移行するために行うべき基本手順は次のとおりです。

1. コード内の廃止または変更されたメソッドを探します。
2. この章にある説明に従って、廃止されたメソッドを置き換えます。
3. 指定されたツールを使って JSP を変換します。
4. 指定されたツールを使って Servlet および EJB の記述子を変換します。
5. 『管理および配置ガイド』の第 2 章にある説明に従い、iPlanet Application Server Deployment Tool を使ってアプリケーションを再配置します。

JDK の移行

iPlanet Application Server 6.5 では、Java 2 Development Toolkit バージョン 1.3.1 (JDK 1.3.1) を使います。

iPlanet Application Server の重要な相違点は、NMI などのネイティブインターフェイスが使われている場合、それを JNI (Java Native Interface) に置き換える必要があることです。

JDK 1.1.7 から JDK 1.3.1 への変更についての詳細は、次のサイトにアクセスしてください。
`http://java.sun.com/products/jdk/1.3/docs/relnotes/features.html`

JDK 1.3.1 の詳細については、次のサイトにアクセスしてください。
`http://java.sun.com/products/jdk/1.3/docs/index.html`

JDK の移行手順

NAS 4.0 アプリケーションを JDK 1.1.7 から JDK 1.3.1 に移行するには、次の手順を実行してください。

1. 廃止されたメソッドとその代替メソッドのリストを次のサイトから入手します。

   `http://java.sun.com/j2ee/j2sdkee/techdocs/api/deprecated-list.html`

   特定の非互換メソッドのリストについては、次のサイトを参照してください。

   `http://java.sun.com/products/jdk/1.2/compatibility.html`
2. 廃止されたメソッドを置き換え、アプリケーションをコンパイルし直します。
3. アプリケーションを再配置します。

サーブレットの移行

iPlanet Application Server 6.5 では、Java Servlet 仕様のバージョン 2.2 を使います。仕様の詳細は、http://java.sun.com/products/servlet/ にアクセスしてください。

Java Servlet API 2.2 の新しい情報については、
http:/developer.java.sun.com/developer/technicalArticles/Servlets/servletapi/ にアクセスしてください。

NAS 4.0 からの Servlet がバージョン 2.1 からのインターフェイスだけを使い、廃止されたクラスをまったく使わない場合、その Servlet は iPlanet Application Server 6.5 で現状のまま実行されます。

バージョン 2.1 からの Java Servlet 仕様の主な変更は次のとおりです。

- Java クラスは XML で設定する
- Servlet はコンテナ内に存在する
- Servlet は常にアプリケーションの一部である

- Servlet は .war ファイルにアーカイブされる
- セキュリティ機能を追加
- Web アプリケーション概念の導入
- Web アプリケーションアーカイブファイルの導入
- レスポンスバッファリングの追加
- 分散可能な Servlet の導入
- RequestDispatcher は名前によって取得可能
- RequestDispatcher は相対パスを使って取得可能
- インターナショナル化の改善
- 分散 Servlet エンジンセマンティックの詳しい説明
- Servlet パラメータ確認の動作の変更 (詳細については、『開発者ガイド』を参照)

Servlet API の変更

次の Servlet API の変更が実装されました。

- ServletRequest インターフェイスに getLocale メソッドが追加されて、クライアントがどのロケール内にあるかを調べる際に支援を行います。
- ServletRequest インターフェイスに isSecure メソッドが追加されました。これは、リクエストが HTTPS などの保護されたトランスポートによって送信されたかどうかを示します。
- ServletContext インターフェイスに、getInitParameter および getInitParameterNames メソッドが追加されました。現在、初期化パラメータはアプリケーションレベルで設定され、そのアプリケーションの一部であるすべての Servlet によって共有することが可能です。
- UnavailableException の構築メソッドは、既存のコンストラクタシグネチャとして置き換えられました。これらのコンストラクタはより単純なシグネチャに置き換えられました。
- ServletConfig インターフェイスに getServletName メソッドが追加されました。これにより、Servlet では、システムに知られている名前があればその名前を取得することができます。
- HttpServletRequest インターフェイスに getHeaders メソッドが追加されました。これにより、特定の名前に関連付けられたすべてのヘッダをリクエストから取得することができます。
- HttpServletRequest メソッドに isUserInRole および getUserPrinciple メソッドが追加されました。これにより、Servlet では抽象的なロールに基づく認証を使うことができます。

- HttpServletResponse インターフェイスに addHeader、addIntHeader、および addDateHeader メソッドが追加されました。これにより、同じヘッダ名を使って複数のヘッダを作成することができます。
- HttpSession インターフェイスに getAttribute、getAttributeNames、setAttribute、および removeAttribute メソッドが追加されました。これにより、API の命名規則が改善されました。getValue、getValueNames、setValue、および removeValue メソッドは廃止されました。
- HttpServletRequest インターフェイスに getContextPath メソッドが追加されました。これにより、Web アプリケーションに関連付けられたリクエストパスの一部が得られます。

Servlet の移行手順

移行には 2 つの方法があります。

a. 廃止された NAS 4.0 メソッドおよび J2EE メソッドの両方を置き換えて、アプリケーションを完全に J2EE に準拠させます。

b. 廃止された NAS 4.0 メソッドだけを置き換えます。廃止された J2EE メソッドを使っていても、アプリケーションは iPlanet Application Server 6.5 で実行されます。しかし、廃止されたメソッドは将来使えなくなる可能性があるため、J2EE 1.2 に移行する計画を立てることをお勧めします。

一部のメソッドは HTTPSession2 ですでに廃止されていることに注意してください。

Servlet を NAS 4.0 から完全に J2EE に準拠したアプリケーションへ移行するには、NAS 4.0 独自のメソッドおよび廃止された J2EE メソッドを置き換える必要があります。置き換えを実行するには、オプションステップ 1 に従ってください。

1. 廃止された Servlet 2.2 メソッドを置き換えます (オプション)。

   廃止されたメソッドの一覧については、http://java.sun.com/products/servlet/2.2/javadoc/deprecated-list.html にアクセスしてください。

2. アクセス制御リストに基づくロジックを宣言セキュリティモデルに置き換えます。

   アクセス制御リスト (ACL) ではなく、Servlet 仕様に記述されている、新しい宣言に基づくセキュリティプロシージャを使います。セキュリティは、アプリケーションレベルではなく、XML ファイル内に配置の一部として実装されます。

   a. 次の廃止された HTTPSession2 セキュリティメソッドを削除します。

      ```
      boolean loginSession(String user, String paswd);
      void logoutSession();
      boolean isAuthorized(String target, String permission);
      ```

   b. XML ファイルの auth メソッドタグを使って、認証メソッドを Basic、Certificate、または Form based に設定します。

c. .xml ファイルでは、<security constraint> を使って Servlet を実行できるロールを指定します。<role-name> を使ってロールを作成するか、または <role-link> タグを使って論理ロールを参照します。

d. ACL エントリ .gxr ファイルを削除します。

3. JSP またはほかの Servlet への絶対参照のルートとして AppPath を使う URI ネーミングを置き換えます。

   iPlanet Application Server 6.5 では、アプリケーションコンテキストルートは絶対参照のルートです。Servlet は、次の方法で、その Servlet と同じアプリケーション内にあるほかの JSP に転送します。

```
RequestDispatcher rd = req.getRequestDispatcher("/sample.jsp");
rd.forward(req,res);
```

ここで、sample.jsp は、sample.jsp が含まれている Servlet と同じアプリケーション内にあります。JSP は、NAS 4.0 のように AppPath ではなく、AppPath/ApplicationName の下にあります。

Servlet の配置

Servlet 2.2 では、NAS 4.0 で使われた配置記述子の代わりに XML ファイルの使用を採用しました。NAS 4.0 アプリケーションの NTV 記述子ファイルを、XML ファイルに変換して、Deployment Tool によって作成される Web アプリケーションアーカイブファイルに追加する必要があります。

1. NTV 記述子ファイルを XML ファイルに変換します。

   次のツールを使って NTV ファイルを XML に変換します。

   convertNtv2Xml $path/appInfo.ntv $newpath/myApp.xml

   $path は appInfo.ntv の場所を示します (appInfo.ntv は Serlvet info NTV ファイルの場所を内部的に指定します)。

   変換ツールは、$newpath に myApp.xml および ias-myApp.xml という 2 つの新規ファイルを作成します。これらのファイルは、それぞれ J2EE および iPlanet Application Server 固有の XML を示します。

2. 変換した NTV 記述子ファイルを WAR アーカイブファイルに追加します。次の手順を実行してください。

   a. Servlet ファイルと変換した記述子ファイル .xml を持つ、Servlet 2.2 WAR アーカイブを作成します。

   b. iPlanet Application Server 6.5 の Deployment Tool を起動します。

   c. 「ファイル」メニューの「開く」を選択します。

d. WAR アーカイブに移動し、「OK」を選択します。

e. 継続して Servlet ファイルおよびほかのファイルを Web アプリケーションに追加します。

3. Web アプリケーションを保存し、Deployment Tool で配置します。

JSP の移行

iPlanet Application Server 6.5 では、JSP 仕様バージョン 1.1 を使います。JSP 1.1 仕様は、特にセキュリティ、トランザクション、およびセッションステート概念について、J2EE に統合されます。仕様の詳細については、http://java.sun.com/products/jsp/ にアクセスしてください。

JSP 1.1 仕様では、次の機能強化を組み込むことによって JSP 0.92 から JSP 1.1 へ拡張しています。

- セマンティックの基礎として Servlet 2.2 を使用
- 変換された JSP ページの JSP コンテナへの配布を有効化
- 移植可能な Tag Extension メカニズムを提供

さらに、iPlanet Application Server 6.5 では、JSP に対するキャッシュやロードバランス、およびカスタムタグ拡張を提供します。

GX タグの廃止

GX タグは廃止されました。GX タグのある NAS 4.0 JSP テンプレートを移行してください。iPlanet Application Server は代わりに JSP 拡張タグを使用します。

JSP の移行手順

NAS 4.0 JavaServer Pages を移行するには、次の手順を実行してください。

1. Servlet またはほかの JSP への絶対参照のルートとして `AppPath` を使う URI ネーミングを置き換えます。

2. JavaServer Pages の仕様を 0.92 から 1.1 に変換します。JavaServer Pages を移行する必要があります。`convert2jsp11` ツールを使って JSP 0.92 を JSP 1.1 に変換することができます。詳細については、この章の「JSP 0.92 から JSP 1.1 への変換」を参照してください。

JSP 0.92 から JSP 1.1 への変換

既存の JSP 0.92 ファイルを変換するには、提供されている変換ツール (convert2jsp11) を使います。ツールを使って個々のファイルを変換したり、ディレクトリのツリー全体で、検索されたすべての JSP ファイルを再帰的に変換したりできます。

注 : 変換する前に、必ずファイルのバックアップコピーを作成してください。

変換ツールは、名前を変更せずに、すべての 0.92 JSP ファイルを 1.1 ファイルに変換します。0.92 バージョンのファイルは、拡張子が 0.92 である同じ名前のファイルにコピーされます。たとえば、myApp.jsp というファイルを変換した場合、そのファイルは新しい JSP 1.1 バージョンになり、0.92 バージョンのファイルは myApp.jsp.0.92 というファイルにコピーされます。

指定された変換でいずれかの 0.92 JSP ファイルにエラーがあると、そのファイルの変換に失敗し、空の出力ファイルが作成されます。この場合は、対応する `filename.0.92` バージョンを `filename` にコピーし直し、エラーを修正してから再びそのファイルの変換スクリプトを実行します。

使用法

```
convert2jsp11 [-r] -ap appPath file/directory
```

引数	説明
-r	オプション。指定されたディレクトリ / フォルダと、すべてのサブディレクトリ / サブフォルダ内の JSP を再帰的にすべて変換する。このオプションを指定しない場合は、指定されたファイルだけが変換される
-ap *appPath*	NAS インストール用の `appPath` を指定する (たとえば、Solaris では /export/nas4/nas/APPS、Windows NT では C:¥Netscape¥Server4¥NAS¥APPS です)
file/directory	変換するファイルを指定する。または、`-r` オプションを指定した場合は、すべてのファイルを変換する必要があるディレクトリを指定する。このディレクトリは、`-ap` オプションで指定した `appPath` を基準にする必要がある

例

この例で、`appPath` は、Windows NT の場合、`C:¥Netscape¥Server4¥NAS¥APPS`、Solaris の場合、`/export/nas4/nas/APPS` です。

最初の例では、appPath でルートされる myApplication というディレクトリ内にある、すべての JSP ファイルが変換されます。

Windows NT :	convert2jsp11 -r -ap c:¥netscape¥server4¥nas¥APPS myApplication
Solaris :	convert2jsp11 -r -ap /export/nas4/nas/APPS myApplication

2 番目の例では、appPath でルートされる myApplication というディレクトリ内にある、myJSP.jsp という 1 つの JSP ファイルが変換されます。

Windows NT :	convert2jsp11 -ap c:¥netscape¥server4¥nas¥APPS myApplication¥myJSP.jsp
Solaris :	convert2jsp11 -ap /export/nas4/nas/APPS myApplication/myJSP.jsp

EJB の移行

iPlanet Application Server 6.5 では Enterprise Java Bean (EJB) 仕様バージョン 1.1 を使います。仕様の詳細は、次のサイトにアクセスしてください。

http://java.sun.com/products/ejb/

この EJB 仕様には、EJB 1.0 から EJB 1.1 への次の主な変更が含まれています。

- エンティティ Beans の仕様が強化され、エンティティ Beans のサポートはコンテナプロバイダにとって必須です。変更は主に、トランザクションのサポート、Enterprise JavaBeans 環境、セキュリティおよび配置記述子に影響を及ぼします。ランタイムでは EJB 1.0 アプリケーションへの影響はほとんどありません。EJB コンテナのランタイム API への唯一の変更は、java.security.Identity クラスを java.security.Principal インターフェイスに置き換えたことです。

EJB 1.1 仕様の次の変更は、iPlanet Application Server 6.5 アプリケーションの開発、アプリケーションアセンブリ、および配置についてのサポートを改善するために行われました。

- Enterprise JavaBeans 環境についてのサポートが強化されました。Beans プロバイダは、JNDI ネーミングコンテキストのエントリを使って、Beans の環境依存関係をすべて指定する必要があります。
- 配置記述子でのアプリケーションアセンブリをサポートします。

- Beans プロバイダとアプリケーションアセンブラの責任が明確に分けられました。

EJB の移行手順

NAS 4.0 Enterprise Java Beans を移行するには、次の手順を実行してください。

1. アクセス制御リストに基づくロジックを宣言セキュリティモデルに置き換えます。

   Servlet と同様に、EJB のアクセス制御リストを削除する必要があります。`<method-permission>` タグを使って、EJB メソッドを実行できるルールを指定します。

2. 配置記述子を変換します。

   詳細については、「EJB の配置」を参照してください。

3. コードと記述子を変更して廃止されたクラスを除外し、新規メソッドに置き換えます。

EJB 1.0 Enterprise JavaBeans コードは、EJB 1.1 コンテナで実行するために変更したりコンパイルし直したりする必要がありません。ただし、次の節に詳しく説明されている例外があります。EJB 1.1 XML に変換される配置記述子の移行は必須です。

EJB コードを変更または再コンパイルする必要があるインスタンス

次のインスタンスでは、EJB コードを変更または再コンパイルする必要があります。

- `javax.jts.UserTransaction` インターフェイスを使う Beans。`javax.jts` インターフェイスのパッケージ名は `javax.transaction` に変更されました。このインターフェイスのメソッドによってスローされる例外に多少の変更がありました。`javax.jts.UserTransaction` インターフェイスを使うビーンは、`javax.transaction.UserTransaction` という新しい名前を使うために変更する必要があります。

- `javax.ejb.EJBContext` インターフェイスの `getCallerIdentity()` または `isCallerInRole(Identity identity)` メソッドを使う Beans。クラス `java.security.Identity` が Java 2 プラットフォームで廃止されるため、これらのメソッドは EJB 1.1 で廃止されました。EJB 1.0 仕様に記述されているビーンは、すべての EJB 1.1 コンテナで動作するための新規メソッドを使うために変更する必要があります。

- `UserTransaction` インターフェイスを使うエンティティ Beans。EJB 1.1 では、エンティティ Beans の `UserTransaction` インターフェイスの使用が禁止されています。

- `UserTransaction` インターフェイスを使い、同時に `SessionSynchronization` インターフェイスを実装する Beans。これは、EJB 1.1 では許可されていません。

- EJB 1.0 では定義されず、EJB 1.1 で定義されている追加のセマンティック制約のいずれかに違反する Beans。

- 各 ejbCreate() では、一致する ejbPostCreate() が必要です。EJBPostCreate は EJB 1.0 ではオプションでしたが、EJB 1.1 では必須です。

例外処理の変更

EJB 1.1 仕様の例外処理は、EJB 1.0 仕様で定義されたルールを維持しました。ただし、次の例外があります。

- EJB 1.0 では、EJB ビジネス メソッドおよびコンテナを起動するコールバックが、java.rmi.RemoteException を使って非アプリケーション例外をレポートするように指定しました。この方法は、EJB 1.1 では廃止されました。Enterprise JavaBeans メソッドでは、javax.ejb.EJBException またはほかの適切な RuntimeException を使って、非アプリケーション例外をレポートしてください。
- EJB 1.1 では、インスタンスによってスローされるすべての非アプリケーション例外は、そのインスタンスが実行したトランザクションをロールバックし、インスタンスを破棄します。EJB 1.0 では、インスタンスによって java.rmi.RemoteException がスローされた場合、コンテナはトランザクションをロールバックせずにそのインスタンスを破棄します。
- EJB 1.1 では、アプリケーション例外によって、コンテナはトランザクションを自動的にロールバックしません。EJB 1.0 では、アプリケーション例外がコンテナによって開始されたトランザクション境界から渡されたときに、コンテナはトランザクションをロールバックする必要がありました。EJB 1.1 では、インスタンスがその EJBContext object.javax.ejb.ejbex で setRollback-Only() メソッドを起動した場合のみ、コンテナはロールバックを実行します。

EJB の配置

EJB 1.1 では、NAS 4.0 で使われた配置記述子の代わりに XML ファイルの使用を採用しました。NAS 4.0 アプリケーションのプロパティ記述子ファイルは、XML ファイルに変換する必要があります。アプリケーションの登録に加えて、ejbReg を実行する必要があります。

プロパティファイルを XML に変換するには、指定されたツール convertPropsXML を使います。次の手順でその使い方について説明します。

1. プロパティファイルを XML ファイルに変換します。

   次のツールを使って、.props ファイルを XML に変換します。

   ```
   convertProps2Xml $path/foobar.props $newpath/myAppEjb.xml
   ```

   `$path` は `.props` ファイルの場所を示します。ツールによって、`myAppEJB.xml` および `ias-myAppEjb.xml` という 2 つの XML ファイルが生成されます。これらのファイルは、それぞれ J2EE および iPlanet Application Server 固有の XML を示します。

2. NTV 記述子ファイルを変換して、EJB JAR アーカイブファイルに追加します。

   a. 新規 EJB JAR モジュールを作成します。詳細については『管理および配置ガイド』の第 2 章を参照してください。

   b. 「EJB」メニューの「4.0 からインポート」を選択します。

   c. 変換する `.properties` ファイルに移動し、「OK」を選択します。`.properties` ファイルは、EJB JAR モジュールに追加される `.xml` ファイルに変換されます。

   d. 引き続き、`.class` ファイルおよびほかのファイルを EJB JAR モジュールに追加します。

   e. EJB JAR モジュールを保存し配置します。詳細については『管理および配置ガイド』の第 2 章を参照してください。

3. アプリケーションコードを変更して例外およびトランザクションを処理します。

   トランザクションまたは例外を使っている場合は、一部のコードの変更が必要な場合があります。「例外処理の変更」の節を参照してください。

4. アプリケーションを配置します。詳細については『管理および配置ガイド』を参照してください。

JNDI の移行

iPlanet Application Server 6.5 では、JNDI (Java Naming and Directory Interface) 拡張バージョン 1.2 を使います。JNDI は、Java Enterprise API セットの一部として提供されます。

JNDI 1.1 を使う NAS 4.0 アプリケーションは、JNDI 1.2 に移行する必要があります。特殊な非互換性については、次のサイトを参照してください。

http://java.sun.com/products/jndi/1.2/compat.html#incompat

J2EE では、各アプリケーションは、コンポーネントの配置記述子によって指定される独自のネーミング環境を定義します。また、コンポーネントの記述子には、ejb-ref および resource-ref 要素によって検索している、すべての EJB およびデータソースに関する情報が記述されている必要があります。NAS 4.0 アプリケーションの iPlanet Application Server 6.5 への移行には、次の必要条件があります。

- アプリケーションで検索されている Beans/ リソースの使用法を識別
- resource-refs および ejb-refs について、適切な配置記述子エントリを設定
- J2EE 仕様で指定されているように、`java:comp/env/<envionmentEntryName>` パターンによって、確実に環境検索が発生するようにします。

Java エクステンション

NAS 4.0 Java エクステンションを iPlanet Application Server 6.5 に移行するには、次の手順を実行します。

1. iPlanet Extension Builder 6.0 に IDL コードを読み込み、新規生成コードを作成します。
2. 以前のエクステンションに対して行った変更を新規生成コードにマージします。
3. NMI に対するすべてのリファレンスを JNI に変換します (該当する場合のみ)。
4. JDK 1.2.2 についてほかのすべての Java コードの変更を実行します。
5. すべてのコードをコンパイルし直します。

C++ エクステンション

NAS 4.0 C++ エクステンションを iPlanet Application Server 6.5 に移行するには、iPlanet Application Server 6.5 ライブラリに対して NAS 4.0 C++ エクステンションを再度コンパイルしてリンクします。

リッチクライアントの使用 (ISecurity インターフェイス)

`ISecurity` インターフェイスは、リッチクライアントを使うときに実装する必要があります。ISecurity は、リッチクライアントでユーザ名とパスワードを設定します。

セキュリティ機能

iPlanet Application Server 6.5 は、配置時にセキュリティ制約を実装します。標準の宣言アクセス制御ルールは、アプリケーションの配置時に開発者によって定義されます。たとえば、開発者は、管理者、ゲスト、メンバなどいくつかのセキュリティレベルを設定します。次に、セキュアプロシージャにアクセスしようとするユーザのパーミッションレベルを調べるコードを記述します。配置時に、ユーザのグループは、制限されているプロシージャにアクセスする前に、アプリケーションで容易に権限レベルを確認できるように、正しいセキュリティレベルが割り当てられます。

NAS 4.0 では、セキュリティは、特定のユーザおよびグループに与えられる権限を定義するアクセス制御リストを設定して、アプリケーションレベルで実装されました。NAS 4.0 はコードレベルでセキュリティを実装しました。iPlanet Application Server 6.5 では、アプリケーションの配置時にコードとは無関係にセキュリティを実装します。

セキュリティに関するヒントについては、次のサイトにアクセスしてください。

http://www.java.sun.com/security/seccodeguide.html

EJB 関連のセキュリティについては、次のサイトにアクセスしてください。

http://www.java.sun.com/j2ee/j2sdkee/techdocs/guides/ejb/html/Security.fm.html

「Bank」の移行例

この例を使って移行プロセスについて説明します。また、ユーザ独自のアプリケーションを移行する前に、例として NAS 4.0 nsOnline Bank ソースを変更し、そのサンプルを iPlanet Application Server 6.5 に移行することもできます。http://www.iPlanet.com/support/ のコードソースオンラインを参照してください。

この節では、Bank サンプルアプリケーションを NAS 4.0 から iPlanet Application Server 6.5 に移植するためのガイドラインについて説明します。

iPlanet Application Server 6.5 Bank サンプルアプリケーションには次の特徴があります。

- Bank サンプルアプリケーションは、新しいフォームベースのログイン (J2EE スタイル) を使っています。
- `HttpSession2`、`loginSession`、`NASRowSet` などの独自のメソッドは、J2EE のメソッドに置き換えられます。
- EJB 配置記述子は `.xml` ファイルに記述されます。
- EJB 検索は "java:comp/env/"lookupname" のように URL ベースです。

- Servlet の配置は、独自の .ntv ファイルではなく .xml ファイルに記述されます。

iPlanet Application Server 6.5 Bank アプリケーションと NAS 4.0 nsOnlineBank の比較

この節では NAS 4.0 と iPlanet Application Server 6.5 コンポーネントを比較します。

コンポーネント	iPlanet Application Server 6.5	NAS 4.0
Servlet	セッション作成には、HttpSession (標準 Java API) を使用し、ログイン認証にはフォームベースのログインメカニズム (J2EE スタイル) を使用する。Bank では、HttpSession2、loginSession などの iPlanet-API のような独自メソッドを使用しない。iPlanet Application Server 6.5 では、HttpSession2 は、AppLogic と Servlet 間でセッションデータを共有するために使用する	ログイン認証には、HttpSession2 メソッド loginSession API、および AppLogic と Servlet 間の共有セッションを使用する
Servlet の配置	配置手順は XML ファイルに記述される。各 Servlet には J2EE XML と iPlanet Application Server 固有の XML の両方がある。Servlet は webappreg コマンドラインユーティリティで登録する	配置手順は ntv ファイルに記述される
EJB	J2EE 固有の URL スタイル検索 "java:comp/env/< lookup name >" を使用する。この検索を使って DataSource オブジェクトを得ることで、データベースコネクションが作成される。新しいファクトリクラス "com.netscape.server.jndi.RootContextFactory" を使用する	古いスタイル "ejb/<lookup name >" を使用する NAS レジストリから取り込まれた DataSourceName は、データベースコネクションを作成するために NASRowSet メソッドに渡し、ファクトリクラス "com.kivasoft.eb.jndi.GDSInitContextFactory" を使用する

コンポーネント	iPlanet Application Server 6.5	NAS 4.0
EJB の配置	配置記述子は XML に記述される 各 EJB には J2EE XML と iPlanet Application Server 固有の XML の両方がある EJB は `ejbreg` コマンド行ユーティリティから登録する	配置手順はプロパティファイルに記述される
JSP	コンテキストルートは /APPS/App-Name/directory から始まる	コンテキストルートは /APPS/directory から始まる
LDAP	認証ドメインは、"uid=admin、ou=Administrators、ou=TopologyManagement、o=NetscapeRoot"。認証パスワードは、"< インストール時にユーザが選択したパスワード >"	デフォルトの認証ドメイン (uid) "cn=Directory Manager" を使う。認証パスワードは "dmanager"

一般的な移植ガイドライン

この節では、NAS 4.0 アプリケーションコンポーネントを iPlanet Application Server 6.5 に移行するための一般的な移植ガイドラインについて説明します。

- **LDAP :** LDAP コード (.java ファイル) は、インストール時に指定される LDAP サーバポート番号を反映する必要があります。
- **DataBase :** DataSource は、<env entry> タグではなく <resource ref> タグから渡される必要があります。
- **Servlet :** Servlet が EJB を検索している場合、対応する J2EE Servlet XML には `<ejb-ref>` タグが必要です。
- フォームベースのメカニズムの場合 **:** J2EE Servlet XML には `<login-config>` タグが必要です。
- ログインページ **:** フォームベースのログインメカニズムのためには `.jsp` ファイルでなければなりません。
- フォームベースのログインページのテキストフィールド **:** `username` は "`j_username`"、パスワードは "`j_password`" でなければなりません。
- ログインページ **:** `login`、`jsp`、`loginerror.jsp` のようなログインページは、`<install-location>/<app-name>` の下に保管される必要があります。

- **EJB :** DataSource が <resource ref> タグから渡された場合、対応する iPlanet Application Server EJB XML にはデータソース "jdbc/LocalDS" のような <jndi-name> が必要です。

  EJB XML ファイルは、PropToEJB ツールを使って生成されます (変換後に、手作業で追加手順を実行する場合があります)。

参考資料

『Java™ 2 Platform, Enterprise Edition Specification Version 1.2』(Copyright 1999, Sun Microsystems, Inc.) は http://java.sun.com/j2ee/docs.html から入手可能です。

『Java™ 2 Platform, Enterprise Edition Technical Overview (J2EE Overview)』(Copyright 1998, 1999, Sun Microsystems, Inc.) は http://java.sun.com/j2ee/white.html から入手可能です。

『Java™ 2 Platform, Standard Edition, v1.2.2 API Specification (J2SE specification)』(Copyright 1993-99, Sun Microsystems, Inc.) は http://java.sun.com/products/jdk/1.2/docs/api/index.html から入手可能です。

『Enterprise JavaBeans™ Specification, Version 1.1 (EJB specification)』(Copyright1998, 1999, Sun Microsystems, Inc.) は http://java.sun.com/products/ejb から入手可能です。

『Enterprise JavaBeans™ to CORBA Mapping, Version 1.1 (EJB-CORBA mapping)』(Copyright 1998, 1999, Sun Microsystems, Inc.) は http://java.sun.com/products/ejb から入手可能です。

『JavaServer Pages™ Specification, Version 1.1 (JSP specification)』(Copyright 1998, 1999, Sun Microsystems, Inc.) は http://java.sun.com/products/jsp から入手可能です。

『Java™ Servlet Specification, Version 2.2 (Servlet specification)』(Copyright 1998,1999, Sun Microsystems, Inc.) は http://java.sun.com/products/servlet から入手可能です。

『JDBC™ 2.0 API (JDBC specification)』(Copyright 1998, 1999, Sun Microsystems, Inc) は http://java.sun.com/products/jdbc から入手可能です。

『JDBC™ Standard Extension 2.0 API (JDBC extension specification)』(Copyright 1998, 1999, Sun Microsystems, Inc.) は http://java.sun.com/products/jdbc から入手可能です。

『Java™ Naming and Directory Interface 1.2 Specification (JNDI specification)』(Copyright 1998, 1999, Sun Microsystems, Inc.) は `http://java.sun.com/products/jndi` から入手可能です。

『Java™ Message Service, Version 1.0.2 (JMS specification)』(Copyright 1998, Sun Microsystems, Inc.) は `http://java.sun.com/products/jms` から入手可能です。

『Java™ Transaction API, Version 1.0.1 (JTA specification)』(Copyright 1998, 1999, Sun Microsystems, Inc.) は `http://java.sun.com/products/jta` から入手可能です。

『Java™ Transaction Service, Version 0.95 (JTS specification)』(Copyright 1997-1999, Sun Microsystems, Inc.) は `http://java.sun.com/products/jts` から入手可能です。

『JavaMail™ API Specification Version 1.1 (JavaMail specification)』(Copyright 1998, Sun Microsystems, Inc.) は `http://java.sun.com/products/javamail` から入手可能です。

『JavaBeans™ Activation Framework Specification Version 1.0.1 (JAF specification)』(Copyright 1998, Sun Microsystems, Inc.) は `http://java.sun.com/beans/glasgow/jaf.html` から入手可能です。

『The Java™ 2 Platform, Enterprise Edition Application Programming Model』(Copyright 1999, Sun Microsystems, Inc.) は `http://java.sun.com/j2ee/apm` から入手可能です。

第 6 章

NetDynamics アプリケーションの実行

この章は、NetDynamics で作成されたアプリケーションを移行するための計画ガイドです。

この章は次の節に分かれています。

- 概要
- 移行計画の注意事項
- iPlanet Migration Toolbox および J2EE Assisted Take-Off (JATO)

概要

これまでは、NetDynamics 3.x、4.x、および 5.x アプリケーションを iPlanet Application Server 6.5 対応の J2EE に移行するために多数のオプションを用意してきました。オプションによっては、NetDynamics 5.x にアップグレードしてから J2EE にアップグレードするという手順を実行していました。

J2EE が普及し、完成度が高まるにつれて、また、新しく利用可能になった自動ツールを使うと NetDynamics の以前のバージョン (バージョン 3.x および 4.x) で作成されたアプリケーションを簡単に変換できるため、NetDynamics 5.x への移行を経ずに、NetDynamics 3.x および 4.x アプリケーションを J2EE に直接移行することをお勧めします。

この章では、NetDynamics から J2EE への移行作業を計画する際の注意事項について説明します。特に、アプリケーションの移行に必要な作業に影響を与える可能性があるいくつかの問題と、J2EE への全面的な移行を妨げるいくつかの要因について検討します。

さらにこの章では、**iPlanet Migration Toolbox (iMT)** を紹介します。iMT を使うと、NetDynamics アプリケーションを自動的に iPlanet Application Server J2EE 環境へ移行できます。

移行されたアプリケーションでは、**iPlanet J2EE Assisted Take-Off (JATO)** アプリケーションフレームワークを利用します。このフレームワークは NetDynamics アプリケーションフレームワークから J2EE へのアプリケーションの直接的な移行に対応するだけでなく、将来の J2EE 開発のベースとなるスタンドアロンアプリケーションフレームワークとしても機能します。JATO はソースコードの全面的な所有権とともに iMT カスタマに提供されています。

iMT および JATO の包括的な説明については、この章の適用対象外です。iMT に関してより詳しい情報が必要な場合は、御社 iPlanet 担当までお問い合わせください。

移行計画の注意事項

驚くべきことではありませんが、移行作業の範囲を決めるのは非常に厳しいタスクです。NetDynamics 環境は開発において柔軟性に富んでいました。NetDynamics アプリケーションの構成、開発の標準、個々の開発者のスタイル、カスタムエクステンションと機能強化などは、NetDynamics アプリケーションの特定のセットを J2EE に移行するための作業のレベルに大きな影響を与える可能性があります。

iPlanet Migration Toolbox には、移行見積もり作業に取りかかるための基本的なツールがいくつか含まれています。**NDProjectPeeker** ツールでは、NetDynamics プロジェクトオブジェクトの一覧表を生成し、プロジェクトのサイズおよび構成について見当をつけることができます。現在開発中の別のツールでは、NDProjectPeeker ツールの出力を利用して、移行作業の基本的な見積もりを生成します。これらのツールによって収集された情報を利用して最初の見積もりが生成されますが、この章で説明する解析を組み込むことによってその見積もりを修正します。

この節には次のトピックがあります。

- 移行方法
- 移行の計画と見積もり

移行方法

一般には、以前の NetDynamics アプリケーションを iPlanet Application Server 6.5 対応の J2EE に直接移行することをお勧めしますが、NetDynamics 3.x および 4.x アプリケーションを NetDynamics 5.x にアップグレードする方が適切な場合もあります。それには次のような理由があります。

- 特定のアプリケーションについて NetDynamics 5.x へのアップグレードが比較的速やかに実行できると判断され、そのアプリケーションが 2002 年 12 月 31 日の NetDynamics 5.x 製品のサポート終了時までに使われなくなる予定である

さらに、今後 J2EE に移行するのが不可能になる問題がいくつかあります。

- アプリケーションで、NetDynamics 環境でしか動作しないカスタムコンポーネントまたはサードパーティコンポーネントを使っている
- アプリケーションが、NetDynamics だけが提供できる機能に依存している

ただし、一般には、J2EE への直接移行がより望ましいと確信しています。

- アプリケーションの予想耐用年数が 2 年を越える場合は、一般的に J2EE に移行してください。
- NetDynamics 5.x へのアップグレードには多くの作業が必要になる可能性があります。特にテストなどのタスクを考慮すると、以前の NetDynamics アプリケーションから NetDynamics 5.x へのアップグレードに必要な作業は、iMT を有効にして iAS 対応の J2EE に移行する場合と同程度の作業になることがあります。
- NetDynamics のサポートは段階的に廃止されるため、NetDynamics への依存性を解決する必要があります。
- NetDynamics でトレーニングを受けた担当者のチームは今後数年間で縮小される予定です。

移行の計画と見積もり

iPlanet Migration Toolbox 計画ツールによって、NetDynamics から J2EE への移行作業の大きさについて見当をつけることができますが、移行するアプリケーション本体に固有の一連の要因を解析することで、意味のある見積もりを作成することができます。NetDynamics アプリケーションの移行に必要な作業を見積もる際に検討すべきいくつかの要因は次のとおりです。

- 移行チームの持つ技術とアプリケーションに関する知識
  - J2EE
  - NetDynamics
  - 移行するアプリケーションのドメインに関する知識
- 移行する必要がある NetDynamics プロジェクトおよびページの数
- NetDynamics API が使われていた範囲
  - イベントメソッドをコーディングするために NetDynamics API が幅広く使われていたか
  - 既存の Enterprise JavaBeans および他のビジネスロジックは、NetDynamics API に依存しないように記述されていたか
- アプリケーションの構造
  - アプリケーションは表示、ビジネス、およびデータに慎重に分けられているか

    - 移行すべきカスタムクラスがあるか。これらのカスタムクラスが NetDynamics にどの程度依存しているか
- アプリケーションの構成
    - アプリケーションは多数の小さい NetDynamics プロジェクトで構成されているか、あるいは少数の大きいプロジェクトで構成されているか
    - 構成内の異なるアプリケーションが相互依存している度合い

NetDynamics プロジェクトが大きい集合の場合は、解析時にいくつかのタスクを追加することを検討する必要があります。

- アーキテクチャの概要
- カスタムクラスとその使い方についての詳細な解析
- NetDynamics プロジェクトの代表的なサンプルの詳しい調査
- 試験的な移行

iPlanet Migration Toolbox および J2EE Assisted Take-Off (JATO)

この節には次のトピックがあります。

- iPlanet Migration Toolbox
- JATO のアプリケーションフレームワーク
- iMT/JATO コミュニティ

iPlanet Migration Toolbox

iPlanet Migration Toolbox (iMT) には、NetDynamics から J2EE への自動移行を行うためのツールセットが含まれています。

- **NetDynamics Extraction Tool:** このツールは NetDynamics プロジェクトから XML 記述ファイルに宣言情報と Java コードを抽出します。
- **Application Translation Tool:** このツールは XML 記述ファイルを使って JATO フレームワーク内で NetDynamics プロジェクトの J2EE 互換バージョンを構築します。
- その他のツール：iPlanet Migration Toolbox には、変換されたプロジェクトをコンパイルしてまとめるための便利なツールも含まれています。

iMT は、NetDynamics アプリケーションをできるだけ完全に J2EE に移行するように設計されています。その主な機能は、NetDynamics 構成子および機能をサポートする新しいアプリケーション環境に、アプリケーション構成を移行する機能です。さらに、iMT はすべての宣言によるアプリケーション機能をこの新しいアプリケーションフレームワークに移行します。NetDynamics ウィザードによって提供されていたこれらの機能の大部分は J2EE に移行されます。

自動移行プロセスが完了したら、以前のアプリケーションから移行したカスタムコードの適合性を評価する必要があります。NetDynamics アプリケーションのカスタムコードを変換して、J2EE 環境で正しく機能させる確実な方法はありません。手動でのコード変換を簡単に行うには、元のコードのコメントを削除し、適切なモジュールおよびメソッドに移動します。このタスクの量は少なくありませんが、一般に、移行開発者が JATO J2EE コンポーネントのクラスおよびメソッドの知識を持っている場合は、簡単に実行できるはずです。

一般に、移行開発者はアプリケーション上で iMT を試行して、プロジェクトに必要な手動での移行作業のレベルを評価します。iMT では、手動でのコード移植過程で評価および変更が必要となるコードレベルアイテムの一覧表が生成されます。この変更一覧表を使って、移行作業計画を改善してください。

JATO のアプリケーションフレームワーク

J2EE Assisted Take-Off (JATO) は、J2EE アプリケーションを一貫性のある効率的な方法で作成するための基盤を、開発者に提供するように構築されたアプリケーションフレームワークです。JATO では、アプリケーションおよびアプリケーションをサポートする基盤の作成方法を開発しなくても、アプリケーションの作成をすぐに始めるためのプロシージャとサポート基盤が用意されています。つまり、NetDynamics で得られた生産性メリットの一部を再び得ることができます。さらに、標準的なフレームワークが利用できるので、企業は時間が経っても社内で開発の一貫性を確保できるだけでなく、新しいチームメンバが専用の社内 J2EE アプリケーションフレームワークを使うためにトレーニングを受けなければならない可能性も小さくなります。

JATO は NetDynamics から J2EE にジャンプするための単なる踏み台ではなく、NetDynamics アプリケーションを移行したあと、長期間にわたって J2EE アプリケーションを構築できるフレームワークとして設計されています。

iMT/JATO コミュニティ

iPlanet Migration Toolbox チームは、http://www.egroups.com/group/iPlanet-JATO のオンライン iMT/JATO 討論コミュニティのモデレータを務めています。

索引

J

M

N

S

T

U

あ

け