Sun Java System Application Server Enterprise Edition 8.2 アップグレードと移行

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–1612

070620@17466

目次

はじめに

このマニュアルでは、アプリケーションを Sun Java System Application Server 8.2 Enterprise Edition にアップグレードおよび移行する方法について説明します。

この『Sun Java System Application Server Enterprise Edition 8.2 アップグレードと移行』では、以前のバージョンの Application Server から現在のバージョンにアップグレードする方法を説明します。また、Java™ 2 Platform, Enterprise Edition (J2EE™ プラットフォーム) アプリケーションを、以前のバージョンの Sun Java SystemApplication Server や他社のアプリケーションサーバーから Sun Java SystemApplication Server 8.2 Enterprise Edition に移行する方法についても説明します。

さらに、Sun Java System Application Server 8.2 と以前のリリースの Application Server との相違点についても説明します。

ここでは、Sun Java System Application Server のマニュアルセット全体に関する情報を提供し、表記規則について示します。

Application Server のマニュアルセット

Application Server のマニュアルセットは、配備の計画とシステムのインストールについて説明しています。スタントアロンの Application Server のマニュアルの URL は `http://docs.sun.com/app/docs/coll/1310.4` です。Sun Java Enterprise System (Java ES) Application Server マニュアルの URL は `http://docs.sun.com/app/docs/coll/1659.1` です。Application Server の概要については、次の表に示すマニュアルを順番に参照してください。

表 P–1　Application Server のマニュアルセットの内容

マニュアル名	説明
『リリースノート』	ソフトウェアとマニュアルに関する最新情報。サポートされているハードウェア、オペレーティングシステム、Java Development Kit (JDK™)、データベースドライバに関する包括的な説明が表にまとめられています。
『Quick Start Guide』	Application Server 製品の使用を開始するための手順。
『Installation Guide』	ソフトウェアとそのコンポーネントのインストール。

表 P–1　Application Server のマニュアルセットの内容　　(続き)

マニュアル名	説明
『配備計画ガイド』	最適な方法で確実に Application Server を導入するための、システムニーズや企業ニーズの分析。サーバーを配備する際に注意すべき一般的な問題や懸案事項についても説明しています。
『Developer's Guide』	J2EE コンポーネントおよび API 用のオープン Java 標準モデルに従い、Application Server 上で実行することを目的とする Java 2 Platform, Enterprise Edition (J2EE プラットフォーム) アプリケーションの作成と実装。開発ツール、セキュリティー、デバッグ、配備、ライフサイクルモジュールの作成などについての情報も提供します。
『J2EE 1.4 Tutorial』	J2EE アプリケーションの開発のための J2EE 1.4 プラットフォーム技術および API の使用。
『管理ガイド』	管理コンソールからの、Application Server サブシステムおよびコンポーネントの設定、管理、および配備。
『高可用性 (HA) 管理ガイド』	高可用性データベースのための、インストール後の設定と管理に関する解説。
『Administration Reference』	Application Server 設定ファイル `domain.xml` の編集。
『アップグレードと移行』	新しい Application Server プログラミングモデルへのアプリケーションの移行 (特に Application Server 6.x または 7 からの移行)。このマニュアルでは、製品仕様の非互換性をもたらす可能性のある、隣接した製品リリース間の相違点や設定オプションについても説明しています。
『パフォーマンスチューニングガイド』	パフォーマンスを向上させるための Application Server の調整。
『トラブルシューティングガイド』	Application Server の問題の解決。
『Error Message Reference』	Application Server のエラーメッセージの解決。
『Reference Manual』	Application Server で使用できるユーティリティーコマンド。マニュアルページのスタイルで記述されています。`asadmin` コマンド行インタフェースも含みます。

関連マニュアル

Application Server は、単体で購入することが可能です。あるいは、ネットワークまたはインターネット環境にわたって分散しているエンタープライズアプリケーションをサポートするソフトウェアインフラストラクチャーである Java ES のコンポーネントとして購入することもできます。Application Server を Java ES のコンポーネントとして購入した場合は、`http://docs.sun.com/coll/1657.1` にあるシステムマニュアルをよく読むことをお勧めします。Java ES およびそのコンポーネントに関するすべてのマニュアルの URL は `http://docs.sun.com/prod/entsys.5` です。

その他の Sun Java System サーバーのマニュアルとしては、次のマニュアルを参照してください。

- Message Queue のマニュアル
- Directory Server のマニュアル
- Web Server のマニュアル

さらに、次のリソースが役立つことがあります。

- J2EE 1.4 Specifications (`http://java.sun.com/j2ee/1.4/docs/index.html`)
- J2EE 1.4 Tutorial (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`)
- J2EE Blueprints (`http://java.sun.com/reference/blueprints/index.html`)

デフォルトのパスとファイル名

次の表は、このマニュアルで使用するデフォルトのパスやファイル名について説明したものです。

表 P–2 デフォルトのパスとファイル名

プレースホルダ	説明	デフォルト値

install-dir	Application Server のベースインストールディレクトリを表します。	Solaris™ プラットフォームへの Sun Java Enterprise System (Java ES) インストールの場合:
		`/opt/SUNWappserver/appserver`
		Linux プラットフォームへの Java ES インストールの場合:
		`/opt/sun/appserver/`
		Solaris および Linux プラットフォームへのインストールで、ルートユーザーでない場合:
		ユーザーのホームディレクトリ `/SUNWappserver`
		Solaris および Linux プラットフォームへのインストールで、ルートユーザーである場合:
		`/opt/SUNWappserver`
		Windows のすべてのインストールの場合:
		SystemDrive`:\Sun\AppServer`

表 P–2 デフォルトのパスとファイル名 *(続き)*

プレースホルダ	説明	デフォルト値

domain-root-dir	すべてのドメインを含むディレクトリを表します。	Solaris プラットフォームへの Java ES インストールの場合:
		`/var/opt/SUNWappserver/domains/`
		Linux プラットフォームへの Java ES インストールの場合:
		`/var/opt/sun/appserver/domains/`
		そのほかのすべてのインストールの場合:
		install-dir`/domains/`

domain-dir	ドメインのディレクトリを表します。 設定ファイルには、次のように表される *domain-dir* があります。 `${com.sun.aas.instanceRoot}`	*domain-root-dir*/*domain-dir*
instance-dir	サーバーインスタンスのディレクトリを表します。	*domain-dir*/*instance-dir*

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–3 表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `machine_name% you have mail.`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	`machine_name%` **`su`** `Password:`
aabbcc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、`rm` *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャー・ユーザーズガイド』を参照してください。

表 P–3 表記上の規則 (続き)

字体または記号	意味	例
「」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

コード例は次のように表示されます。

- C シェル

```
machine_name% command y|n [filename]
```

- C シェルのスーパーユーザー

```
machine_name# command y|n [filename]
```

- Bourne シェルおよび Korn シェル

```
$ command y|n [filename]
```

- Bourne シェルおよび Korn シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

記号の規則

次の表は、この用語集で使用される記号の一覧です。

表 P–4 記号の規則

記号	説明	例	意味
[ ]	省略可能な引数やコマンドオプションが含まれます。	ls [-l]	-l オプションは必須ではありません。
{ | }	必須のコマンドオプションの選択肢を囲みます。	-d {y|n}	-d オプションには y 引数か n 引数のいずれかを使用する必要があります。
${ }	変数参照を示します。	${com.sun.javaRoot}	com.sun.javaRoot 変数の値を参照します。
-	同時に押すキーを示します。	Control-A	Control キーを押しながら A キーを押します。
+	順番に押すキーを示します。	Ctrl+A+N	Control キーを押してから放し、それに続くキーを押します。
→	グラフィカルユーザーインタフェースでのメニュー項目の選択順序を示します。	「ファイル」→「新規」→「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから「テンプレート」を選択します。

マニュアル、サポート、およびトレーニング

Sun のサービス	URL	内容
マニュアル	http://jp.sun.com/documentation/	PDF 文書および HTML 文書をダウンロードできます。
サポートおよびトレーニング	http://jp.sun.com/supportraining/	技術サポート、パッチのダウンロード、および Sun のトレーニングコース情報を提供します。

◆ ◆ ◆ 第 1 章

Application Server の互換性の問題

Application Server 8.2 Enterprise Edition は、Application Server 8.1、8.0、および 7.x とバイナリ互換性があります。バージョン 8.1、8.0、および 7.x で実行される J2EE アプリケーションも Application Server8.2 で動作しますが、この章で説明する部分については互換性がありません。

この章では、次の領域における非互換性について説明します。

HTTP ファイルキャッシュ

Application Server 8.1 Enterprise Edition に存在した HTTP ファイルキャッシュは、Application Server 8.2 では提供されていません。

domain.xml の要素

サーバーインスタンスに対するメッセージレベルのセキュリティープロバイダが設定されていない場合、Application Server 8.1 では Domain Administration Server (DAS) によるデフォルト設定が適用されます。Application Server 8.2 ではデフォルト設定は適用されません。メッセージレベルのセキュリティーを使用する各サーバーインスタンスに対して、メッセージレベルのセキュリティープロバイダ (`ClientProvider` および `ServerProvider`) を手動で導入する必要があります。以前のバージョンから Application Server 8.2 にアップグレードする場合、アップグレードツールでは、これらの不足している要素は `domain.xml` ファイルに追加されません。

システムプロパティー

Application Server 8.2 のデフォルトのセキュリティーポリシーでは、一部のシステムプロパティーの変更が許可されていません。たとえば、Application Server 7 では、`java.util.PropertyPermission` プロパティーの読み取り権/書き込みの権限は `"*", "read,write";` でしたが、Application Server 8.2 の `java.util.PropertyPermission` プロパティーの読み取り権/書き込みの権限は `"*", "read";` です。

暗黙の URL 書き換え

Application Server 6.x では、暗黙の URL 書き換えがサポートされていました。この機能では、Web コネクタプラグインがブラウザに送信された HTML ストリームを解析し、`href=` および `frame=` などの属性にセッション ID を追加していました。Application Server 7、8、および Application Server 8.2 ではこの機能を使用できません。ユーザーは自分のアプリケーションを見直して、Cookie をサポートしていないクライアント (携帯電話など) にアプリケーションが提示するすべての URL で `encodeURL` および `encodeRedirectURL` を使用する必要があります。

Web サーバーの機能

次に示す Web サーバー固有の機能は、Application Server 8.2 ではサポートされていません。

- `cgi-bin`、`shtml`
- Simple Network Management Protocol (SNMP) のサポート
- Netscape API (NSAPI) プラグイン API
- ネイティブのコンテンツ処理機能
- Web サーバーツール (`flexanlg`、`htpasswd`)
- HTTP QoS
- Web サーバーの設定ファイル (`*.conf`、 `*.acl`、`mime.types`)
- Web サーバー固有のログローテーション機能
- ウォッチドッグ処理 (`appserv-wdog`)

レルム

アップグレードツールは、レルムおよびロールマッピング設定、任意のカスタムレルムクラス、および各ドメインのファイルベースのユーザーキーファイルを移動します。XML タグ、`security-service` によって、レルムとロールマッピング設定が定義されます。このタグは `sun-server_1_0.dtd` および `sun-domain_1_0.dtd` で定義されます。Application Server 7 の場合、タグのデータは `server.xml` にあり、Application Server 8.2 の場合は `domain.xml` にあります。

アップグレードツールは、カスタムレルム用に定義されたクラスファイルを配置し、これを Application Server 8.2 環境で使用できるようにします。カスタムレルムクラスは、タグ `auth-realm` のクラス名属性で定義されます。`security-service` タグでは、`default-realm` 属性はサーバーの使用しているレルムを指しています。これが、設定済みの `auth-realm` 名のいずれかを指すようにします。デフォルトのレルムはファイルです。`default-realm` のクラス名が見つからない場合、アップグレードツールはこれをエラーとしてログします。

セキュリティーレルム実装のパッケージ名は、Application Server 7 では `com.iplanet.ias.security.auth.realm` でしたが、Application Server 8.2 では `com.sun.enterprise.security.auth.realm` に名前変更されました。したがって、`com.iplanet.*` クラスを使用して作成したカスタムレルムは変更する必要があります。

`com.sun.enterprise.security.AuthenticationStatus` クラスは削除されました。

`com.sun.enterprise.security.auth.login.PasswordLoginModule` `authenticate` メソッドの実装は、次のように変更されました。

```
/**
    * Perform authentication decision.
    * <P> Note: AuthenticationStatus and AuthenticationStatusImpl
```

```
     * classes have been removed.
     * Method returns silently on success and returns a LoginException
     * on failure.
     *
     * @return void authenticate returns silently on
     *               successful authentication.
     * @throws LoginException on authentication failure.
     *
     */
abstract protected void authenticate()
    throws LoginException;
```

Sun 配備記述子: sun-web.xml

Application Server 7 では、オプション属性 delegate のデフォルト値は false でした。Application Server 8.2 では、この属性のデフォルト値は true に設定されています。この変更は、デフォルトでは Web アプリケーションのクラスローダーが、それ自身でクラスをロードする前に、まず親のクラスローダーに委任するようになったことを意味しています。詳細については、70 ページの「delegate 属性」を参照してください。

encodeCookies プロパティー

sun-web.xml ファイルにある sun-web-app 要素の encodeCookies プロパティーが true に設定されている場合、Cookie の URL エンコーディングが実行されます。Application Server 7 では、encodeCookies プロパティーのデフォルト値は true でした。このプロパティーは Application Server 8 では存在していませんでした。Application Server 8.2 では、デフォルト値は false です。

Cookie の URL エンコーディングは必要ありません。このプロパティーは true に設定しないようにしてください。このプロパティーが必要になるのは、Application Server 7 でこの動作に依存していた非常に少数のアプリケーションのみです。

CORBA パフォーマンスオプション

Application Server 7 では、ユーザーは、必要に応じていくつかの Object Request Broker (ORB) パフォーマンスの最適化をオンにするために、次のシステムプロパティーを指定することができました。

```
-Djavax.rmi.CORBA.UtilClass=com.iplanet.ias.util.orbutil.IasUtilDelegate
```

Application Server 8.2 では、デフォルトで ORB パフォーマンスの最適化がオンになっています。以前のシステムのシステムプロパティー参照を使用している場合は、デフォルトの最適化を干渉しないようにするために、これを削除してください。

ファイル形式

Application Server 8.2 では、`domain.xml` がメインのサーバー設定ファイルです。Application Server 7 では、メインのサーバー設定ファイルは `server.xml` でした。`domain.xml` の DTD ファイルは `lib/dtds/sun-domain_1_1.dtd` にあります。Application Server 8.2 に含まれるアップグレードツールを使用すると、Application Server 7 の `server.xml` から Application Server 8.2 の `domain.xml` に移動できます。

Application Server 8.2 の `lib/dtds/sun-domain_1_1.dtd` ファイルは、Application Server 8 における対応ファイル (`sun-domain_1_0.dtd`) に対して完全な下位互換を保っています。

通常、設定ファイルの形式は下位互換性がありません。次の設定ファイルはサポートされていません。

- `*.conf`
- `*.acl`
- `mime.types`
- `server.xml` (`domain.xml` に置き換え)

クラスタのスクリプト

Application Server 7 における `clsetup` および `cladmin` スクリプトは、Application Server 8.2 ではサポートされていません。Application Server 8.2 では、 `clsetup` スクリプトは `asadmin configure-ha-cluster` コマンドに置き換わり、`cladmin` スクリプトによってサポートされていたコマンドは、クラスタを操作する `asadmin` コマンドに置き換わっています。`asadmin` コマンドの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Reference Manual 』を参照してください。

主キーの属性値

Application Server 7 では、管理コンソールで任意のフィールド、コマンド行インタフェース (CLI) で任意の属性を変更できました。Application Server 8.2 では、項目の主キーのフィールドや属性を変更することはできません。ただし、項目を削除して、新しい主キーの値で再作成することはできます。ほとんどの場合、主キーは名前、ID、参照、または JNDI 名のいずれかです。次の表は、変更することのできない主キーの一覧です。

注 – domain.xml ファイルでは、フィールドまたは属性は *attribute*、項目は *element* と呼ばれています。domain.xml の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Administration Reference 』を参照してください。

表 1–1 主キーの属性

項目	主キーのフィールドまたは属性
admin-object-resource	jndi-name
alert-subscription	name
appclient-module	name
application-ref	ref
audit-module	name
auth-realm	name
cluster-ref	ref
cluster	name
config	name
connector-connection-pool	name
connector-module	name
connector-resource	jndi-name
custom-resource	jndi-name
ejb-module	name
external-jndi-resource	jndi-name
http-listener	id
iiop-listener	id
j2ee-application	name
jacc-provider	name
jdbc-connection-pool	name
jdbc-resource	jndi-name
jms-host	name
jmx-connector	name

表 1–1 主キーの属性 *(続き)*

項目	主キーのフィールドまたは属性
lb-config	name
lifecycle-module	name
mail-resource	jndi-name
message-security-config	auth-layer
node-agent	name
profiler	name
element-property	name
provider-config	provider-id
resource-adapter-config	resource-adapter-name
resource-ref	ref
security-map	name
server	name
server-ref	ref
system-property	name
thread-pool	thread-pool-id
virtual-server	id
web-module	name
persistence-manager-factory-resource	jndi-name

コマンド行インタフェース: hadbm

次の表は、コマンド行ユーティリティー hadbm のサポートされなくなったオプションの一覧です。hadbm コマンドの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Reference Manual』を参照してください。

表 1–2 サポートされていない hadbm オプション

オプション	サポート対象でなくなったサブコマンド
--inetdsetup	addnodes サブコマンドに対してサポートされなくなりました。

表 1–2 サポートされていない hadbm オプション (続き)

オプション	サポート対象でなくなったサブコマンド
--inetd	create サブコマンドに対してサポートされなくなりました。
--inetdsetupdir	create サブコマンドに対してサポートされなくなりました。
--configpath	create サブコマンドに対してサポートされなくなりました。
--set managementProtocol	create サブコマンドに対してサポートされなくなりました。
--set DataDeviceSize --set TotalDatadeviceSizePerNode	create または set サブコマンドに対してサポートされなくなりました。

コマンド行インタフェース: start-appserv および stop-appserv

start-appserv および stop-appserv コマンドは非推奨です。これらのコマンドを使用すると警告が表示されます。asadmin start-domain および asadmin stop-domain を代わりに使用してください。

Application Server 8.2 では、管理コンソールから「標準エラーへのログメッセージ」フィールドが削除されています。domain.xml ファイル内の log-to-console 属性は非推奨で無視されます。asadmin set コマンドは、 log-to-console 属性に対して無効です。asadmin start-domain コマンドの ---verbose オプションを使用して、asadmin start-domain コマンドを実行したウィンドウに対するメッセージを印刷します。このオプションが機能するのは、起動中のドメインのあるマシン上で asadmin start-domain コマンドを実行した場合のみです。

コマンド行インタフェース: asadmin

ここでは、コマンド行ユーティリティー asadmin に対する変更内容について説明します。

asadmin コマンドの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Reference Manual』を参照してください。

asadmin のサブコマンド

サブコマンドは、次の示すものを除いて下位互換が保たれています。

reconfig サブコマンドは非推奨で、無視されます。

次のサブコマンドは Application Server 8.2 ではサポートされていません。

- show-instance-status (list-instances を使用してください)
- restart-instance (stop-instance の後に start-instance を続けて使用してください)
- configure-session-persistence (configure-ha-persistence に名前変更されました)
- create-session-store (create-ha-store に名前変更されました)
- clear-session-store (clear-ha-store に名前変更されました)

次のサブコマンドは Application Server 8.2 ではサポートされなくなりました。ソフトウェアライセンスキーと Web コアが削除され、Application Server 8.2 では、Web サーバー機能からの制御関数はサポートされなくなりました。

- install-license
- display-license
- create-http-qos
- delete-http-qos
- create-mime
- delete-mime
- list-mime
- create-authdb
- delete-authdb
- list-authdbs
- create-acl
- delete-acl
- list-acls

Start および Stop サブコマンドに対するエラーコード

Application Server 7 では、asadmin コマンドの start および stop サブコマンドに対するエラーコードは、必要とされる END 状態に基づいていました。たとえば asadmin start-domain の場合、ドメインがすでに実行中であれば、終了コードは 0 (成功) でした。ドメインの起動に失敗すれば、終了コードは 1 (エラー) でした。

Application Server 8.2 の場合、終了コードはコマンドが期待どおりに実行されるかどうかに基づいています。たとえば、asadmin start-domain コマンドは、ドメインがすでに実行中であるか、ドメインの起動に失敗すれば、終了コード 1 を返します。同じように、asadmin stop-domain は、ドメインがすでに停止されたか、停止できなければ、終了コード 1 を返します。

非推奨およびサポートされていないオプション

次の表のオプションは、非推奨であるか、サポートされなくなったものです。

表 1–3 非推奨およびサポートされていない asadmin オプション

オプション	サブコマンドで非推奨またはサポートされない
--acceptlang	create-virtual-server サブコマンドに対して非推奨です。
--acls	create-virtual-server サブコマンドに対して非推奨です。
--adminpassword	関連するすべてのサブコマンドに対して非推奨です。--passwordfile を代わりに使用します。
--blockingenabled	create-http-listener サブコマンドに対して非推奨です。
--configfile	create-virtual-server サブコマンドに対して非推奨です。
--defaultobj	create-virtual-server サブコマンドに対して非推奨です。
--domain	stop-domain サブコマンドに対して非推奨です。
--family	create-http-listener サブコマンドに対して非推奨です。
--instance	すべてのリモートサブコマンドに対して非推奨です。--target を代わりに使用します。
--mime	create-virtual-server サブコマンドに対して非推奨です。
--optionsfile	すべてのコマンドに対してサポートされなくなりました。
--password	すべてのリモートサブコマンドに対して非推奨です。--passwordfile を代わりに使用します。
--path	create-domain サブコマンドに対して非推奨です。 --domaindir を代わりに使用します。
--resourcetype	関連するすべてのサブコマンドに対して非推奨です。--restype を代わりに使用します。
--storeurl	すべてのコマンドに対してサポートされなくなりました。
--target	すべての jdbc-connection-pool、connector-connection-pool 、connector-security-map、および resource-adapter-config サブコマンドに対して非推奨です。

表 1–3 非推奨およびサポートされていない asadmin オプション (続き)

オプション	サブコマンドで非推奨またはサポートされない
--type	関連するすべてのサブコマンドに対して非推奨です。

ドット表記名

asadmin get および set サブコマンドにおけるドット表記名の使用は、次の部分で下位互換ありません。

- デフォルトサーバー名は、server1 ではなく server です。
- server_instance.resource は domain.resources.resource になります。
- server_instance.app-module は domain.applications.app-module になります。
- 属性名の形式が異なっています。たとえば、poolResizeQuantity は pool-resize-quantity になりました。
- Application Server 7 でサポートされていたエイリアスの一部が Application Server 8.2 ではサポートされていません。

Application Server 8.2 では、asadmin コマンドの ---passwordfile オプションでは password.conf ファイルを読み取らないので、アップグレードツールによってこのファイルはアップグレードされません。Application Server 8.2 でパスワードファイルを作成する方法については、『Sun Java System Application Server Enterprise Edition 8.2 管理ガイド』を参照してください。

下の表は、Application Server 7 と 8.2 の間で互換性のないドット表記名を 1 対 1 で示したものです。この表には互換性のあるドット表記名は表示されていません。

表 1–4 バージョン間で互換性のないドット表記名

Application Server 7 のドット表記名	Application Server 8.2 のドット表記名
server_instance.http-listener.*listener_id* *server_instance*.http-service.http-listener. *listener_id*	*server_instance*.http-service.http-listener.*listener_id* *config_name*.http-service.http-listener.*listener_id*
server_instance.orb *server_instance*.iiop-service	*server_instance*.iiop-service *config_name*.iiop-service
server_instance.orblistener *server_instance*.iiop-listener	*server_instance*.iiop-service.iiop-listener.*listener_id* *config_name*.iiop-service.iiop-listener.*listener_id*
server_instance.jdbc-resource. *jndi_name*	*server_instance*.resources.jdbc-resource.*jndi_name* domain.resources.jdbc-resource.*jndi_name*

表 1–4 バージョン間で互換性のないドット表記名 (続き)

Application Server 7 のドット表記名	Application Server 8.2 のドット表記名
server_instance.jdbc-connection-pool. *pool_id*	*server_instance*.resources.jdbc-connection-pool. *pool_id*domain.resources.jdbc-connection-pool. *pool_id*
server_instance.external-jndi-resource. *jndi_nameserver_instance*.jndi-resource. *jndi_name*	*server_instance*.resources.external-jndi-resource. *jndi_name*domain.resources.external.jndi-resource. *jndi_name*
server_instance.custom-resource. *jndi_name*	*server_instance*.resources.custom-resource. *jndi_name*domain.resources.custom-resource. *jndi_name*
server_instance.web-container.logLevel (最後の注を参照)	*server_instance*.log-service.module-log-levels.web-container *config_name* .log-service.module-log-levels.web-container
server_instance.web-container.monitoringEnabled (最後の注を参照)	*server_instance*.monitoring-service.module-monitoring -levels.web-container *config_name* .monitoring-service.module-monitoring-levels.web-container
server_instance.j2ee-application. *application_nameserver_instance*.application. *application_name*	*server_instance*.applications.j2ee-application. *application_name* domain.applications.j2ee-application. *application_name*
server_instance.ejb-module. *ejb-module_name*	*server_instance*.applications.ejb-module. *ejb-module_name*domain.applications.ejb-module. *ejb-module_name*
server_instance.web-module. *web-module_name*	*server_instance*.applications.web-module. *web-module_name*domain.applications.web-module. *web-module_name*
server_instance.connector-module. *connector_module_name*	*server_instance*.applications.connector-module. *connector_module_name* domain.applications.connector-module. *connector_module_name*
server_instance.lifecycle-module. *lifecycle_module_name*	*server_instance*.applications.lifecycle-module. *lifecycle_module_name* domain.application.lifecycle-module. *lifecycle_module_name*
server_instance.virtual-server-class	N/A*
server_instance.virtual-server. *virtual-server_id*	*server_instance*.http-service.virtual-server.virtual-server_id *config_name*.http-service.virtual-server.virtual-server_id
server_instance.mime. *mime_id*	N/A*

表 1–4 バージョン間で互換性のないドット表記名 *(続き)*

Application Server 7 のドット表記名	Application Server 8.2 のドット表記名
server_instance.acl. *acl_id*	N/A*
server_instance.virtual-server .*virtual-server_id*.auth-db .*auth-db_id*	N/A*
server_instance.authrealm. *realm_idserver_instance*.security-service.authrealm .*realm_id*	*server_instance*.security-service.auth-realm. *realm_idconfig_name* .security-service-auth-realm.*realm_id*
server_instance.persistence-manager-factory-resource. *jndi_nameserver_instance* .resources.persistence-manager-factory-resource. *jndi_name*	*server_instance*.resources.persistence-manager-factory-resource. *jndi_name*domain.resources.persistence-manager-factory-resource. *jndi_name*
server_instance.http-service.acl. *acl_id*	N/A*
server_instance.mail-resource. *jndi_name*	*server_instance*.resources.mail-resource. *jndi_name*domain.resources.mail-resource. *jndi_name*
server_instance.profiler	*server_instance*.java-config.profiler *config_name* .java-config.profiler

* — Application Server 7 におけるこれらの属性名は、Application Server 8.2 のドット表記名に直接は対応していません。

属性値のトークン

Application Server 8 では `asadmin get` コマンドによって解決済みの値が表示されますが、Application Server 8.2 では未処理の値が表示されます。これらの未処理の値はトークンの場合があります。たとえば、次のコマンドを実行します。

```
asadmin get domain.log-root
```

上記のコマンドによって次の値が表示されます。

```
${com.sun.aas.instanceRoot}/logs
```

属性値の NULL

Application Server 8 では、値のない属性値には NULL が格納されていました。これが原因で、パスを指定する属性に問題が発生していました。Application Server 8.2 では、Application Server 7 の場合と同じように、値のない属性には空の文字列が格納されるようになりました。

◆ ◆ ◆ 第 2 章

Application Server インストールのアップグレード

Application Server 8.2 に付属するアップグレードツールは、ターゲットインストールに以前にインストールされたサーバーの設定をレプリケートします。アップグレードツールは、Application Server の設定、アプリケーション、および認証データを以前のバージョンから Application Server 8.2 にアップグレードするのに役立ちます。アップグレードできる Application Server の旧バージョンの一覧については、表 2–1 を参照してください。

次の内容について説明します。

アップグレードの概要

次の表に、サポートされる Sun Java System Application Server のアップグレードを示します。この表では、PE は Platform Edition、EE は Enterprise Edition を表します。

表 2–1 サポートされるアップグレードパス

元のインストール	8.2 Platform Edition	8.2 Enterprise Edition
7.X PE	サポートされていない	O
7.XSE	-	O

表 2–1 サポートされるアップグレードパス (続き)

元のインストール	8.2 Platform Edition	8.2 Enterprise Edition
7.XEE	-	O
8.0PE	O	O
8.1PE	O	O
8.1EE	-	O
8.2PE	-	O

注 – 8.1 SE から 8.2 EE へのアップグレードでは、HADB パッケージのインストールのみを行います。アップグレードツールを使用する必要はありません。

アップグレードツールを起動するには、`asupgrade` コマンドを発行するか、Application Server インストーラの「アップグレード」オプションを選択します。

アップグレードツールのインタフェース

このツールは、コマンド行インタフェース (CLI) または GUI を介して使用できます。

アップグレードツールを GUI モードで使用するには、オプションを付けずに `asupgrade` コマンドを発行します。

アップグレードツールを CLI モードで実行するには、`--c`/`--console` オプションを指定して `asupgrade` コマンドを起動します。アップグレード CLI は、対話型モードまたは非対話型モードで実行できます。コンソールで `asupgrade` を起動するときに必要な引数をすべて指定した場合は、非対話型モードでアップグレードが実行され、それ以上の入力は必要ありません。`asupgrade` のオプションの完全なリストについては、表 2–2 を参照してください。`--c`/`--console` オプションのみを指定してツールを起動すると、ツールは対話型 CLI モードになり、ユーザーは一連の入力を求められます。

アップグレードの用語

アップグレード処理に関する重要な用語を次に示します。

- ソースサーバー: 新しいバージョンにアップグレードする元のインストール。
- ターゲットサーバー: アップグレード先のインストール。
- ドメインルート: ドメインが作成されるディレクトリ。このディレクトリは、デフォルトでは `asenv.conf` に `AS_DEF_DOMAINS_PATH` として指定されている場所です。

- ドメインディレクトリまたは *domain-dir*: 特定のドメインに対応する (ドメインルート内の) ディレクトリ。ドメインに関係するすべての設定データやその他のデータは、このディレクトリ内にあります。
- インストールルート: Application Server がインストールされているディレクトリ。
- 管理ユーザー名: サーバーを管理するユーザー名。この用語は、アップグレードする Application Server インストールの管理ユーザーを指します。
- パスワード: アップグレードする Application Server インストールのドメイン管理サーバー (DAS) にアクセスするための、管理ユーザーのパスワード (8 文字以上)。
- マスターパスワード: ドメイン管理サーバーの起動などの操作に使用される SSL 認証データベースのパスワード。この用語は、アップグレードする Application Server インストールのマスターパスワードを指します。

アップグレードツールの機能

アップグレードツールは、設定、配備されたアプリケーション、および認証データベースを Application Server の以前のバージョンから最新バージョンに移行します。アップグレードツールでは、Application Server のバイナリはアップグレードされません。バイナリのアップグレードはインストーラで実行します。データベースの移行や変換もこのアップグレード処理の範囲外です。

Sun Java System Web Server 固有の機能を使用しないインスタンスのみがシームレスにアップグレードされます。HTTP パス、CGI バイナリ、SHTML、および NSAPI プラグインに関する設定ファイルは、アップグレードされません。

注 – アップグレード処理を開始する前に、ソースサーバー (アップグレードする元のサーバー) とターゲットサーバー (アップグレードする先のサーバー) の両方が停止していることを確認します。

配備されたアプリケーションの移行

Application Server 7.x/8.0 環境に配備されているアプリケーションアーカイブ (EAR ファイル) とコンポーネントアーカイブ (JAR ファイル、WAR ファイル、および RAR ファイル) は、何も変更せずに Application Server 8.2 上で実行できます。

ソースサーバーに配備されているアプリケーションとコンポーネントは、アップグレード時にターゲットサーバーに配備されます。ターゲットサーバーに正常に配備されないアプリケーションは、Migration Tool または `asmigrate` コマンドを使用して移行し、手動で再配備する必要があります。移行ツールを使用したアプリケーションの移行については、第 6 章を参照してください。

配備されているアプリケーションに関する情報がドメインに含まれており、インストール済みのアプリケーションコンポーネントがその設定情報と一致しない場合、不正な設定は再設定されずにそのまま移行されます。

クラスタのアップグレード

Application Server 7.x でクラスタを含む設定をアップグレードするときは、1 つ以上のクラスタファイルまたは `clinstance.conf` ファイルを指定します。Application Server 8.x では、クラスタが `domain.xml` ファイルで定義されるため、個々のクラスタを指定する必要はありません。もう 1 つの大きな違いとして、Application Server 8.x では、クラスタ内のすべてのインスタンスが同じドメイン内にあり、したがって同じ `domain.xml` ファイル内にあります。Application Server 7.x では、1 つのクラスタを形成するインスタンスが複数のドメインにまたがっている可能性があります。

証明書とレルムファイルの転送

アップグレードツールは、ソース証明書データベースからターゲットに証明書を転送します。このツールは、JKS 証明書から NSS 証明書への変換をサポートします。このツールは、セキュリティーポリシー、標準のファイルベースレルムのパスワードファイル、およびカスタムレルムクラスを転送します。認証データベースのパスワードを指定する際の具体的な要件については、31 ページの「アップグレードの前に」を参照してください。

アップグレードログ

アップグレードログには、アップグレードのアクティビティーが記録されます。アップグレードログファイルは、`upgrade.log` という名前で、アップグレードが行われるドメインのルートに作成されます。

アップグレードのロールバック

実行中のアップグレードが取り消された場合は、アップグレード開始前の設定が復元されます。

アップグレードシナリオ

アップグレード実行の 3 つのシナリオを次に示します。

- サイドバイサイドアップグレード: ソースサーバーとターゲットサーバーが、同じコンピュータ上の異なるインストール場所にインストールされます。これらのインストールに対応する設定を同じコンピュータ上の異なる場所に配置したい場合は、このタイプのアップグレードを実行できます。
- インプレースアップグレード: ターゲットサーバーが、ソースサーバーと同じインストール場所にインストールされます。設定 (つまり、ドメイン) を以前と同じ場所にインストールしたい場合は、このタイプのアップグレードを実行できま

す。このシナリオでは、インストーラを使用して、既存のバイナリと同じ場所にバイナリをインストールします。インストール時に「アップグレード」を選択した場合は、インストール後にインストーラがソースドメインディレクトリとターゲットサーバーディレクトリを設定してからアップグレードツールを起動します。このタイプのアップグレードでは、ユーザーがソースドメインルートを指定し、そのドメインルートの下にあるすべてのドメインがアップグレードされます。複数のドメインルートを作成してカスタマイズした場合は、その下のドメインをアップグレードするために、各ドメインを指定する必要があります。

アップグレードの前に

次の各シナリオでは、アップグレードの前に次の手順を実行する必要があります。

- Application Server 7.x からのアップグレード: アプリケーションサーバーのソースインストールとターゲットインストールを同じコンピュータ上の異なるインストール場所にインストールするサイドバイサイドアップグレードのみを実行できます。アップグレードの前に、次の作業を行う必要があります。
  1. すべての Application Server 7.x ドメインに同じ管理者資格があることを確認します。
  2. Application Server 8.2 EE のインストール中にドメインを作成する場合は、Application Server 7.x と同じ管理者資格を使用します。
- Application Server 8.x EE からのアップグレード: サイドバイサイドアップグレードまたはインプレースアップグレードを実行できます。アップグレードの前に、キーストアや認証データベースに `changeit` 以外のパスワードがあるかどうかをチェックする必要があります。そのようなパスワードは、すべて `changeit` に変更する必要があります。JKS データベースには `keytool` ユーティリティーを使用し、NSS データベースには `certutil` ユーティリティーを使用します。

注意 – キーストアや認証データベースのパスワードを変更していない場合は、アップグレードツールによる証明書転送処理が異常終了することがあります。

アップグレード後に、キーストアや認証データベースのパスワードの値を復元したい場合は、`keytool` ユーティリティーや `certutil` ユーティリティーを使用して手動で値を復元できます。

コマンド行からのアップグレード

アップグレードユーティリティーは、次の構文を使用してコマンド行から実行します。

```
asupgrade
[--console ]
[--version ]
[--help ]
[--source applicationserver_7.x/8.x_installation]
[--target applicationserver_8.2_installation]
[--domain domain_name]
[--adminuser admin_user]
[--adminpassword admin_password]
[--masterpassword master_password]
[--targetnsspwdfile targetNSS_password_filepath]
[--nsspwdfile NSS_password_filepath]
[--jkspwdfile JKS_password_filepath]
[--capwdfile CA_password_filepath]
[--clinstancefiles file1 [, file2, file3, ... filen]]
```

次の表に、コマンドオプションの詳細 (短い書式、長い書式、および説明) を示します。

表 **2–2** asupgrade ユーティリティーのコマンドオプション

短い書式	長い書式	説明
-c	--console	アップグレードコマンド行ユーティリティーを起動します。
-V	--version	アップグレードツールのバージョン。
-h or -?	--help	アップグレードユーティリティーを起動する際の引数を表示します。
-t	--target	Application Server 8.2 EE インストールのドメインルートディレクトリ。
-s	--source	ソースが Application Server 7.x の場合は、インストールディレクトリ。ソースが Application Server 8.x で、インプレースアップグレードが行われた場合は、ドメインルートディレクトリ。ソースが Application Server 8.x で、サイドバイサイドアップグレードが行われた場合は、ドメインディレクトリ *domain-dir*。
-d	--domain	移行する証明書のドメイン名。
-a	--adminuser	管理者ユーザーのユーザー名
-w	--adminpassword	管理者パスワード

表 2–2 asupgrade ユーティリティーのコマンドオプション (続き)

短い書式	長い書式	説明
-m	--masterpassword	マスターパスワード
-n	--nsspwdfile	NSS パスワードファイルへのパス。このファイルは、パスワードのみを格納するプレーンテキストファイルです。
-e	--targetnsspwdfile	ターゲットの NSS パスワードファイルへのパス。このファイルは、パスワードのみを格納するプレーンテキストファイルです。
-j	--jkspwdfile	JKS パスワードファイルへのパス。このファイルは、パスワードのみを格納するプレーンテキストファイルです。
-p	--capwdfile	CA 証明書パスワードファイルへのパス。このファイルは、パスワードのみを格納するプレーンテキストファイルです。
-i	--clinstancefiles	ソースインストールが Application Server 7.x の場合は、クラスタファイルへのパス。デフォルトのファイル名は `$AS_INSTALL/conf/clinstance.conf` です。

次の例は、`asupgrade` コマンド行ユーティリティーを使用して既存のアプリケーションサーバーインストールを Application Server 8.2 にアップグレードする方法を示しています。

例 1: 証明書移行の確認を含む Application Server 7 インストールの Application Server 8.2 EE へのアップグレード

この例は、Sun Java SystemApplication Server 7 インストールの Sun Java System Application Server 8.2 へのサイドバイサイドアップグレードの実行方法を示しています。このコマンドは、証明書を移行するかどうかをユーザーに確認します。移行しないことを選択した場合、証明書は移行されません。

```
asupgrade --source /home/sunas7 --target /home/sjsas8.2/domains
```

例 2: クラスタと NSS 証明書を含む Application Server 7.1 EE インストールの Application Server 8.2 EE へのアップグレード

この例は、クラスタを含む Sun Java System Application Server 7.1 EE インストールの Sun Java System Application Server 8.2 EE へのサイドバイサイドアップグレードの実行方法を示しています。NSS 証明書は、`clinstance.conf` クラスタファイルと同じように移行されます。

```
asupgrade --source /home/sjsas7.1 --target /home/sjsas8.2/domains --domain domain1
--nsspwdfile /home/sjsas7.1/nsspassword.txt
--targetnsspwdfile /home/sjsas8.2/nsspassword.txt
--clinstancefiles /home/sjsas7.1/config/clinstance.conf
```

アップグレード後、すべてのリモートインスタンスのノードエージェントがターゲットの DAS 上に作成されます。これらのノードエージェントは、それぞれのホストシステムにコピーして起動する必要があります。詳細は、38 ページの「Application Server 7.x EE からノードエージェントをアップグレードする」を参照してください。

例 3: クラスタと NSS 証明書を含む Application Server 8.1 EE インストールの Application Server 8.2 EE へのアップグレード (インプレース)

この例は、クラスタを含む Sun Java System Application Server 8.1 EE インストールの Sun Java System Application Server 8.2 EE へのインプレースアップグレードの実行方法を示しています。NSS 証明書は、`clinstance.conf` クラスタファイルと同じように移行されます。

```
asupgrade --source /home/sjsas8.1/domains
--target /home/sjsas8.2/domains
--domain domain1
--nsspwdfile /home/sjsas8.1/nsspassword.txt
--targetnsspwdfile /home/sjsas8.2/nsspassword.txt
--clinstancefiles /home/sjsas8.1/config/clinstance.conf
```

アップグレード後、すべてのリモートインスタンスのノードエージェントがターゲットの DAS 上に作成されます。これらのノードエージェントは、それぞれのホストシステムにコピーして起動する必要があります。詳細は、39 ページの「Application Server 8.1 EE からノードエージェントをアップグレードする」を参照してください。

例 4: クラスタと NSS 証明書を含む Application Server 8.1 EE インストールの Application Server 8.2 EE へのアップグレード (サイドバイサイド)

この例は、クラスタを含む Sun Java System Application Server 8.1 EE インストールの Sun Java System Application Server 8.2 EE へのサイドバイサイドアップグレードの実行方法を示しています。NSS 証明書は、`clinstance.conf` クラスタファイルと同じように移行されます。

```
asupgrade --source /home/sjsas8.1/domains/domain1 --target /home/sjsas8.2/domains
--domain domain1 --nsspwdfile /home/sjsas8.1/nsspassword.txt
--targetnsspwdfile /home/sjsas8.2/nsspassword.txt
--clinstancefiles /home/sjsas8.1/config/clinstance.conf
```

ウィザードによるアップグレード

GUI モードのアップグレードウィザードは、コマンド行またはデスクトップから起動できます。

ウィザードを起動するには、次の手順に従います。

- UNIX では、`<install_dir>/bin` ディレクトリに移動し、`asupgrade` と入力します。

- Windows では、`<install_dir>/bin` ディレクトリの `asupgrade` アイコンをダブルクリックします。

Application Server のインストール時に「アップグレード」チェックボックスを選択した場合は、インストールの完了後に自動的にアップグレードウィザード画面が表示されます。

▼ アップグレードウィザードを使用する

1 「ソースインストールディレクトリ」フィールドに、設定のインポート元となる既存のインストールの場所を入力します。

「ソースインストールディレクトリ」フィールドには、Application Server の古いインストールのバージョンと実行するアップグレードのタイプに応じて適切な値を入力します。有効な値は次のとおりです。

- Application Server 7.x からアップグレードする場合は、Application Server 7.x のインストールルートを入力します。たとえば、Solaris/Linux ユーザーの場合は `/home/sunappserver7.1`、Windows ユーザーの場合は `C:\ProgramFiles\SunAppserver7.1` などです。Application Server 7.x からアップグレードする場合は、サイドバイサイドアップグレードのみを実行できます。
- Application Server 8.x からのインプレースアップグレードを実行する場合は、ドメインルートを入力します。たとえば、Solaris/Linux ユーザーの場合は `/home/sunappserver7.1/domains`、Windows ユーザーの場合は `C:\ProgramFiles\SunAppserver7.1\domains` などです。
- Application Server 8.x からのサイドバイサイドアップグレードを実行する場合は、ドメインディレクトリを入力します。たとえば、Solaris/Linux ユーザーの場合は `/home/sunappserver7.1/domains/domain1`、Windows ユーザーの場合は `C:\ProgramFiles\SunAppserver7.1\domains\domain1` などです。

2 「ターゲットインストールディレクトリ」フィールドに、設定のインポート先となる **Application Server** インストールの場所を入力します。

アップグレードウィザードをインストールから起動した (Application Server のインストール時に「旧バージョンからアップグレード」チェックボックスを選択した) 場合、このフィールドのデフォルト値は Application Server ソフトウェアをインストール

したディレクトリになります。アップグレードツールがインストーラ経由で起動されなかった場合や、Application server 7.x からアップグレードする場合は、Application Server 8.2 EE インストールのドメインルートをこのフィールドに入力する必要があります。たとえば、Solaris/Linux ユーザーの場合は `/home/sunappserver8.2/domains`、Windows ユーザーの場合は `C:\ProgramFiles\SunAppserver8.2\domains` などです。

3 **Application Server 7.x** からアップグレードする場合は、**Application Server 7.x** のインストールで使用した管理者ユーザー名、管理パスワード、およびマスターパスワードを入力する必要があります。ターゲットサーバー **(Application Server 8.2)** のインストール時にドメインを作成した場合は、**Application Server 7.x** で使用したのと同じ管理者資格を使用していることが必要です。**Application Server 7.x** ドメインが複数ある場合は、すべてのドメインに同じ管理者資格が必要です。**Application Server 8.x** をアップグレードする場合は、管理ユーザー名として `admin` を、管理パスワードとして `adminadmin` を、マスターパスワードとして `changeit` をそれぞれ入力する必要があります。ほかの値を入力すると、アップグレードツールはそれらの値を無視し、管理者資格にデフォルト値を割り当てます。

注 – アップグレードの前に必要な作業については、31 ページの「アップグレードの前に」を参照してください。

---

4 クラスタがあってセキュリティー証明書がない **Application Server 7.x Enterprise Edition** インストールをアップグレードする場合は、「次へ」ボタンをクリックして手順 **10** に進み、クラスタファイルの情報を入力します。**Application Server 8.x Enterprise Edition** を **Application Server 8.2 Enterprise Edition** にアップグレードする場合は、「次へ」ボタンをクリックしてアップグレードを続行します。アップグレードツールがソースインストール内のクラスタを自動的に検出します。セキュリティー証明書を変換する必要がある場合は、手順 **5** に進みます。

5 ソースインストールに変換が必要なセキュリティー証明書がある場合は、「セキュリティー証明書を転送」チェックボックスを選択して「次へ」ボタンをクリックします。

「セキュリティー証明書を転送」画面が表示されます。

6 「セキュリティー証明書を転送」画面で、「ドメインを追加」ボタンをクリックし、転送する証明書を含むドメインを追加します。

「ドメインを追加」ダイアログが表示されます。

7 「ドメインを追加」ダイアログで、移行するセキュリティー証明書を含むドメイン名を選択し、適切なパスワードを入力します。

8 作業が完了したら、「了解」ボタンをクリックします。

「セキュリティー証明書を転送」画面がもう一度表示されます。

9 手順 **6** と **7** を繰り返して、転送する証明書があるドメインをすべて追加します。作業が完了したら、「次へ」ボタンをクリックします。

10 クラスタがある **Sun Java System Application Server 7.1 Enterprise Edition** インストールを **Sun Java System Application Server 8.2 Enterprise Edition** にアップグレードする場合は、「クラスタ設定を転送」画面が表示されます。「クラスタを追加」ボタンをクリックします。

「clinstance.conf ファイルの選択」ダイアログボックスが表示されます。clinstance.conf ファイルを選択し、「開く」ボタンをクリックします。clinstance.conf ファイルがリストに追加されます。

11 手順 **10** を繰り返して、移行する必要があるクラスタ設定ファイルをすべて追加します。「次へ」ボタンをクリックします。

この処理を繰り返して、移行する必要があるクラスタ設定ファイルをすべて追加し、「次へ」ボタンをクリックします。

12 アップグレード操作のステータスを示す「アップグレードの結果」パネルが表示されます。

13 アップグレード処理が完了したら、「完了」ボタンをクリックしてアップグレードツールを閉じます。

クラスタのアップグレード

Application Server のアップグレードツールは、クラスタの詳細情報をクラスタ設定ファイル clinstance.conf から取得します。Application Server 7.x に対して 2 つ以上のクラスタが定義されていた場合、アップグレード前に複数の .conf ファイルが存在している可能性があります。設定ファイルには任意の名前をつけることができますが、その拡張子は常に .conf になります。クラスタがアップグレードに含まれる場合、clinstance.conf ファイルの定義時に次の点を考慮してください。clinstance.conf ファイル内のインスタンス名は一意である必要があります。たとえば、Application Server 7.x において、マシン A 上に特定のクラスタに属する server1 と server2 が存在しているとします。また、マシン B 上にも、それと同じクラスタに属する server1 が存在しているとします。通常、clinstance.conf ファイルにはマシン A の server1 と server2、およびマシン B の server1 が含まれます。Application Server 8.1 では、クラスタ内のインスタンス名が一意である必要があります。このため、アップグレードする前に、clinstance.conf ファイル内のマシン B の名前 server1 を server3 や server1ofmachineB のような一意の名前に変更する必要があります。

注 – マシン B 内の server1 インスタンス自体の名前を変更する必要はありません。変更する必要があるのは、clinstance.conf ファイル内のサーバー名だけです。

なお、クラスタ内のインスタンスは「同種」であることが期待されますが、それは、同種のリソースを備え、同一のアプリケーションが配備されている、という意味においてです。アップグレード処理を実行すると、マスターインスタンスとしてマークされたインスタンスが、設定転送用として選択されます。マスターインスタンスとしてマークされたインスタンスが存在しない場合は、インスタンスのうちの 1 つがランダムに選択され、それが設定転送用として使用されます。クラスタは、clinstance.conf ファイル内に定義されているインスタンスとともに、DAS 内で作成されます。このクラスタに属するインスタンスはすべて、cluster_name-config という名前の同一の設定を共有します。ここで、*cluster_name* は、最初のクラスタでは cluster_0、次のクラスタでは cluster_1、といった具合になります。クラスタ内の各インスタンスのシステムプロパティー内には、HTTP ポートと IIOP ポートが設定されます。HTTP ポートは、clinstance.conf ファイル内でインスタンスポートとして定義されていたポートになります。IIOP ポートは、server.xml ファイル内の iiop-cluster 設定から選択されます。

Application Server 8.x EE を Application Server 8.2 EE にアップグレードする場合は、アップグレードツールがソースインストール上のクラスタを (存在する場合は) 自動的に検出します。この場合、設定ファイルを指定する必要はありません。

▼ Application Server 7.x EE からノードエージェントをアップグレードする

クラスタに属しているが、DAS が実行されていないマシン上で実行されるサーバーインスタンスは、<*host-name*>-<*domain-name*> と言う名前のノードエージェントを使用して作成されます。ここで、*host-name* は clinstance.conf ファイル内でその特定のインスタンス用に設定された名前であり、*domain-name* は、このクラスタが属するドメインの名前です。

DAS 上でのアップグレード処理が完了したら、クラスタ化されたインスタンスを実行する必要があるほかのマシン上に Application Server 8.2 をインストールします。

1 install-dir/nodeagents/ の下にあるノードエージェントのディレクトリを、**DAS** マシンからクライアントマシンへコピーします。たとえば、**DAS** が HostA 上にインストールされており、クライアントマシンの名前が HostB である場合、アップグレード処理を実行すると、「HostB_ <*domain_name*>」という名前のノードエージェントが、HostB のノードエージェントとして作成されます。したがって、HostB_<*domain_name*> を HostA<*AS82_install_dir*>/nodeagents/HostB_<*domain_name*>

ディレクトリから HostB<*AS82_install_dir*>/nodeagents にコピーします。コピーし終わったら、そのコピーしたノードエージェントのディレクトリを HostA から削除します。

2 **DAS** を起動します。

3 HostB 上の HostB_<*domain_name*> という名前のノードエージェントを起動します。**DAS** およびリモートインスタンスとのランデブーを行うノードエージェントが HostB に作成され、配備されたアプリケーションがコピーされます。

▼ Application Server 8.1 EE からノードエージェントをアップグレードする

インプレースアップグレードを実行する場合は、ノードエージェントをアップグレードする必要はありません。更新されたバイナリでも、同じノードエージェントディレクトリを使用できます。サイドバイサイドアップグレードを実行する場合は、次の手順を実行します。

1 **Application Server 8.2 EE** をデフォルトノードエージェントとともに Machine B の /opt/SUNWappserver8.2 にインストールします。

2 **Application Server 8.1 EE** のインストール場所からサイドバイサイドアップグレードを実行します。

3 machine A 上の **DAS** を参照するリモートインスタンスを含むノードエージェントがある Machine B 上に **Application Server 8.2 EE** をインストールします。Machine B にはノードエージェントのみをインストールしてください。

4 アップグレード後、Machine A および Machine B の nodeagent.properties ファイルで、agent.adminPort プロパティーを確認します。このファイルには、domain.xml ファイルの該当する node-agent 要素内の jmx-connector ポートと同じ値を反映する必要があります。そうでない場合は、nodeagent.properties ファイルを適宜編集します。

5 Machine A 上の **DAS** を起動します。

6 Machine A 上のデフォルトノードエージェントを起動します。ノードエージェントが、instance1 というインスタンスとともに起動します。この手順のあとで、cluster1 クラスタが部分的に起動します。

7 Machine B で、そのリモートインスタンスのデフォルトノードエージェントを起動します。

8 asadmin のコマンド list-node-agents(1)、list-clusters(1)、list-instances(1) を実行することにより、アップグレードされた要素をチェックできます。

PE および EE のアップグレード時に発生する可能性がある問題の修正

この節では、Application Server 8.2 へのアップグレード時に発生する可能性がある次の各問題に対する解決方法を示します。

古いドメイン上での --domaindir オプションの実行

Application Server 8.2 と古い Application Server 8.1 が 2 つの異なる場所にインストールされている場合は、ドメインを実際には最新バージョンにアップグレードせずに、旧バージョンの Application Server で作成されたドメイン上で --domaindir オプションを使用することもできます。このシナリオでは、ドメインが必ず Application Server の最新のバイナリを使用するように、startserv スクリプトと stopserv スクリプトを更新する必要があります。最新の場所にある asenv.conf ファイルを指し示すようにスクリプトを変更してください。

▼ ソースサーバー上で定義された追加 HTTP リスナーをターゲット PE サーバーに移行する

PE ソースサーバーで追加の HTTP リスナーが定義されていた場合、アップグレード後にそれらのリスナーを PE ターゲットサーバーに追加する必要があります。

1 管理コンソールを起動します。

2 「設定」を展開します。

3 「**HTTP** サービス」を展開します。

4 「仮想サーバー」を展開します。

5 **<server>** を選択します。

6 右側の区画で、追加の **HTTP** リスナー名を「**HTTP** リスナー」フィールドに追加します。

7 設定が完了したら「保存」をクリックします。

▼ ソースサーバー上で定義された追加 HTTP リスナーと IIOP リスナーをターゲット EE サーバーに移行する

ソースサーバーで追加の HTTP リスナーまたは IIOP リスナーが定義されていた場合、クラスタ化されたインスタンスを起動する前に、ターゲット EE サーバーの IIOP ポートを手動で更新しておく必要があります。たとえば、クラスタに属する `server1` で、追加 HTTP リスナーとして `MyHttpListener` が定義されていたとします。その場合、クラスタ内のサーバーインスタンスは互いに対称的になっているため、クラスタ内のほかのインスタンスでも同じ HTTP リスナーが定義されています。ターゲットの *<cluster_name>*`-config` という名前の設定内で、このリスナーを追加し、そのポートを特定のシステムプロパティー `{myHttpListener_HTTP_LISTENER_PORT}` に設定する必要があります。ターゲットサーバーでは、この設定を使用するこのクラスタ内のサーバーインスタンスのそれぞれが、`myHttpListener_HTTP_LISTENER_PORT` という名前のシステムプロパティーを持つことになります。このプロパティーの値は、どのサーバーインスタンスの場合も、ソースサーバーである `server1` のポート値に設定されます。サーバーを起動する前に、これらのサーバーインスタンスのシステムプロパティーの値を競合しないポート番号に手動で更新する必要があります。

ソースサーバーで追加の HTTP リスナーが定義されていた場合、アップグレード後にそれらのリスナーをターゲットサーバーに追加する必要があります。

1 管理コンソールを起動します。

2 「設定」を展開し、対象の *<server>*`-config` 設定を選択します。

3 「**HTTP** サービス」を展開します。

4 「仮想サーバー」を展開します。

5 *<server>* を選択します。

6 右側の区画で、追加の **HTTP** リスナー名 (複数可) を「**HTTP** リスナー」フィールドに追加します。

7 設定が完了したら「保存」をクリックします。

ポートの競合問題の解消

ソースサーバーを Application Server 8.1 EE にアップグレードし終わったら、ドメインを起動します。ノードエージェントを起動します。すると、デフォルトでサーバーインスタンスが起動します。管理コンソールを起動し、これらのサーバーが起動されていることを確認します。実行されていないサーバーが見つかった場合、*install_dir*/nodeagents/*node-agent-name*/*server_name*/logs/server.log ファイル内で、ポートの競合によるエラーが発生していないか確認します。ポートの競合によるエラーが発生していた場合、管理コンソールを使って競合が解消されるようにポート番号を変更します。ノードエージェントとサーバーをいったん停止し、再起動します。

Application Server 7.x SE または EE からのアップグレード時に、7.x サーバー内のいずれかのインスタンスが Application Serve 8.2 インストールによって作成されたデフォルトドメインに割り当てられたデフォルトポートと同じである場合は、ポートの競合が発生します。Application Server 8.2 EE でドメインが作成された場合にデフォルトで割り当てられるポートの値については、次のリストを参照してください。このような場合は、アップグレード後に管理コンソールを起動し、server-config のリスナーのポートを競合しないポート番号に変更してください。

注 – Application Server 8.2 EE のデフォルトポートは、次のとおりです。

- HTTP インスタンス (DAS インスタンス) の場合は 8080
- JMS の場合は 7676
- IIOP の場合は 3700
- HTTP_SSL の場合は 8181
- IIOP_SSL の場合は 3820
- IIOP_MUTUALAUTH の場合は 3920
- JMX_ADMIN の場合は 8686

単一ドメイン内に複数の証明書データベースパスワードが存在する場合に発生する問題の解消

アップグレードに証明書が含まれる場合、移行対象の証明書を格納するドメインごとに、ソース PKCS12 ファイルとターゲット JKS キーファイルに対するパスワードを入力します。Application Server 7 が使用する証明書ストア形式 (NSS) は Application Server 8 PE が使用する形式 (JSSE) とは異なるため、移行対象の鍵と証明書はその新し

い形式に変換されます。1 つのドメインでサポートされる証明書データベースパスワードは、1 つだけです。単一ドメイン内で複数の証明書データベースパスワードが使用されている場合、アップグレードを開始する前に、それらのパスワードをすべて同一のものに変更します。アップグレードの終了後にパスワードをリセットします。

サイドバイサイドアップグレード時のロードバランサプラグインに関する問題の解決

ロードバランサプラグインがほかのアプリケーションサーバーコンポーネントと同じシステムに配置されている場合は、Application Server 7.1 EE から Application Server 8.2 EE へのアップグレードでサイドバイサイドアップグレードを実行中に、新しい 8.2 のロードバランサプラグインが古い 7.1 の Web サーバーインストールを指し示すことができません。Web サーバーを再インストールし、8.2 のロードバランサプラグインインストールが新しいインストールに属するインスタンスを指し示すように設定する必要があります。

サイドバイサイドアップグレード時の共有コンポーネントに関する問題の解決

MQ と HADB を含む Application Server 7.x から Application Server 8.2 EE へのサイドバイサイドアップグレードを実行した場合は、旧バージョン (Application Server 7.x) をアンインストールしないでください。アンインストール処理によって、JAF ライブラリ、JavaMail ライブラリ、MQ ライブラリなどの共有コンポーネントが削除されます。共有コンポーネントがないと、Application Server 8.2 EE インストールが正常に機能しません。Application Server 7.x をアンインストールする場合は、`pkgrm` コマンドを実行して Application Server 7.x に属する `SUNWas*` パッケージを削除し、アンインストールスクリプトは実行しないようにしてください。アンインストールスクリプトを使用して Application Server 7.x をすでに削除した場合は、`pkgadd` コマンドを実行して、手動で共有コンポーネントをコピーしてください。

バイナリとリモートアップグレード

このツールでは、サーバーの実行時バイナリはアップグレードされません。アップグレードツールは、以前にインストールされたサーバーの設定情報と配備されたアプリケーションをアップグレードします。サーバーのバイナリパッケージをインストールするには、Application Server インストーラを使用する必要があります。アップグレード処理では、最初にインストーラを使用してターゲットサーバーのバイナリをインストールします。

ソースサーバーとターゲットサーバーのファイルシステム (特に、ドメインルートのファイルシステム) に同じマシンからアクセスできない場合は、アップグレードを実行できません。現在、アップグレードの大部分はファイルベースで行われます。アップグレードを実行するには、アップグレードを実行するユーザーがソースディレクトリとターゲットディレクトリの読み取りアクセス権、およびターゲットディレクトリの書き込みアクセス権を持っている必要があります。

◆ ◆ ◆ 第 3 章

J2EE アプリケーションの移行

他社のアプリケーションサーバーからアプリケーションを移行するには、移行ツール (http://www.java.sun.com/j2ee/tools/migration/)または asmigrate コマンドを使用します。このツールは、旧バージョンの Sun Java SystemApplication Server からのアップグレード後に正しく配備できていないアプリケーションを移行する場合にも使用します。このツールは、入力アーカイブまたはソースコード上で機能して、実行時の配備記述子をソースアプリケーションのサーバー形式から変換し、最新バージョンに準拠した実行時の配備記述子を生成します。また、JSP および Java ソースコードファイル (ソースコード入力の場合) を解析し、特定のカスタム JSP タグやユーザー独自の API を実行時にサポートします。

この章の内容は、次のとおりです。

移行について

この節では、J2EE アプリケーションの移行の必要性と、移行する必要のある特定のファイルについて説明します。移行が成功すると、J2EE アプリケーションが Application Server に再配備されます。

ここでは、次の内容について説明します。

J2EE コンポーネントと標準

Sun Java System Application Server 8.2 (以後 Application Server と呼びます) は、J2EE v1.4 に準拠したサーバーで、Java コミュニティーによって開発されたコンポーネント標準に基づいています。これとは対照的に、Sun Java SystemApplication Server 7 (Application Server 7) は J2EE v1.3 準拠のサーバーで、Sun ONE Application Server 6.x (Application Server 6.x) は J2EE v1.2 準拠のサーバーです。これら 3 つの J2EE バージョン間には、J2EE アプリケーションのコンポーネント API に大きな違いがあります。

次の表は、J2EE v1.4 準拠の Sun Java System Application Server 8.2、J2EE v1.3 準拠の Sun ONE Application Server 7、および J2EE v1.2 準拠の Sun ONE Application Server 6.x によって使用されるコンポーネント API の違いを示したものです。

表 3–1　J2EE コンポーネントの API に関する Application Server バージョン間の比較

コンポーネント API	Sun ONE Application Server 6.x	Sun Java System Application Server 7	Sun Java System Application Server 8.2
JDK	1.2.2	1.4	1.4
サーブレット	2.2	2.3	2.4
JSP	1.1	1.2	2.0
JDBC	2.0	2.0	2.1、3.0
「EJB」	1.1	2.0	2.0
JNDI	1.2	1.2	1.2.1
JMS	1.0	1.1	1.1
JTA	1.0	1.01	1.01

J2EE アプリケーションコンポーネント

J2EE は、標準化されたモジュール化コンポーネントに基づき、これらのコンポーネントに対するサービスをすべて提供し、多数の詳細なアプリケーション動作を複雑なプログラミングなしで自動的に処理することで、エンタープライズアプリケーションの開発を簡略化します。J2EE v1.4 アーキテクチャーには、多くのコンポーネント API が含まれています。主な J2EE コンポーネントには次のようなものがあります。

- クライアントアプリケーション
- Web アプリケーション
- EJB (Enterprise Java Beans)
- コネクタ
- EAR (Enterprise Application Archive)

J2EE コンポーネントは別個にパッケージ化されており、配備用の J2EE アプリケーションにバンドルされています。各コンポーネント、その関連ファイル (GIF および HTML ファイル、またはサーバー側ユーティリティークラス)、および配備記述子は、モジュール内に組み込まれて J2EE アプリケーションに追加されています。J2EE アプリケーションは、1 つ以上のエンタープライズ Bean、Web、またはアプリケーションクライアントコンポーネントモジュールによって構成されています。最終的なエンタープライズソリューションは、設計要件に従って、J2EE アプリケーションを 1 つだけ使用することも、複数で構成することもできます。

J2EE アプリケーションと各モジュールには、それぞれ独自の配備記述子があります。配備記述子は .xml という拡張子の付いた XML ドキュメントで、ここにはコンポーネントの配備設定が記述されています。

J2EE アプリケーションとそのすべてのモジュールは、EAR (Enterprise Application Archive) ファイルで提供されます。EAR ファイルは、.ear という拡張子の付いた標準の JAR (Java Archive) ファイルです。ear extension. EAR ファイルには EJB JAR ファイル、アプリケーションクライアント JAR ファイル、WAR (Web Archive) ファイルのいずれか、またはすべてが格納されています。

J2EE の詳細については、次の内容を参照してください。

- J2EE 1.4 Tutorial
- J2EE overview
- J2EE website

移行が必要な理由

J2EE 仕様はアプリケーションの要件を広く網羅してはいますが、現在も進化の途上にある標準です。アプリケーションの一部が含まれていなかったり、実装の詳細がアプリケーションプロバイダの裁量に委ねられている部分もあります。

これによって、アプリケーションサーバーの実装に違いが生じたり、アプリケーションサーバー上の J2EE コンポーネントの配備方法にも違いが生じています。特定のアプリケーションサーバー製品で使用可能な一連の設定および配備ツールによっても、製品の実装方法の違いが発生します。

仕様そのものが絶え間なく進化するため、アプリケーションプロバイダーは困難に直面します。また、それぞれのコンポーネント API も進化の途上にあります。このため、製品によって準拠の度合いにばらつきが発生しています。特に、Application Server などの新しい製品では、他の確立されたアプリケーションサーバープラットフォーム上に配備された J2EE アプリケーションコンポーネント、モジュール、およびファイル間の相違点に取り組む必要があります。こうした相違点に対応するには、以前の J2EE 標準の実装詳細 (ファイル命名規則やメッセージ構文) 間のマッピングが必要になります。

さらに、製品プロバイダは通常、追加の機能やサービスを製品とバンドルします。これらの機能は、カスタム JSP タグまたは独自の Java API ライブラリとして提供されます。残念ながら、これらの独自機能を搭載すると、アプリケーションの移植性は失われてしまします。

何を移行する必要があるのか

J2EE アプリケーションは、移行の対象となる次のファイルカテゴリで構成されています。

- 配備記述子 (XML ファイル)
- 独自の API を含む JSP ソースファイル
- 独自の API を含む Java ソースファイル

配備記述子 (XML ファイル)

配備を実行するには、スタンドアロンのエンタープライズ Bean (EJB、JAR ファイル)、フロントエンドの Web コンポーネント (WAR ファイル)、およびエンタープライズアプリケーション (EAR ファイル) の配備記述子を指定します。配備記述子は、J2EE コンポーネントまたはアプリケーションのすべての外部依存を解決するために使用されます。配備記述子の J2EE 仕様 は、すべてのアプリケーションサーバー製品間で共通です。ただしこの仕様では、アプリケーションに関係するコンポーネントの配備状況のいくつかは、製品の実装に依存したままになっています。

JSP ソースファイル

J2EE では、特別なカスタムタグを追加することで JSP の拡張方法を指定します。製品ベンダーは、開発者の作業の一部を簡略化するために、製品内にいくつかのカスタム JSP 拡張を含めています。しかしこうした独自のカスタムタグを使用すると、JSP ファイルの移植性が失われてしまいます。さらに JSP は、他の Java ソースファイル内で定義されたメソッドも呼び出すことができます。独自の API を格納する JSP は、移植する前に書き換える必要があります。

Java ソースファイル

Java ソースファイルは、EJB、サーブレット、またはその他のヘルパークラスの場合があります。EJB とサーブレットは、標準の J2EE サービスを直接呼び出すことができます。ヘルパークラス内で定義されたメソッドを呼び出すこともできます。Java ソースファイルは、アプリケーションのビジネスレイヤー (EJB など) をエンコードするために使用されます。ベンダーは、いくつかのサービスと独自の Java API を製品にバンドルしています。独自の Java API を使用することは、アプリケーションの移植性を失う大きな原因となります。J2EE は進化の途上にある標準なので、さまざまな製品がさまざまなバージョンの J2EE コンポーネント API をサポートする可能性があります。

Migration Tool とその他のリソース

Migration Tool for Sun Java System Application Server 8.2 (以後 Migration Tool と呼びます) は J2EE アプリケーションを他のサーバープラットフォームから Sun Java System Application Server 8.2 に移行します。

Sun Java System Application Server 8.2 では、次のソースプラットフォームがサポートされています。

- Sun ONE Application Server 6.*x*
- Sun Java System Application Server 7
- Sun Java System Application Server 8.0/8.1
- J2EE Reference Implementation Application Server (RI) 1.3、 1.4 Beta1
- WebLogic Application Server (WLS) 5.1、 6.0、 6.1、 8.1
- WebSphere Application Server (WAS) 4.0、 5.x
- Sun ONE Web Server 6.0
- JBoss Application Server 3.0
- TomCat Web Server 4.1

Migration Tool を使用すると、ソースコードを大きく変更することなしに、J2EE アプリケーションの Sun Java System Application Server 8.2 への移行を自動化できます。

このツールの主要な機能は次のとおりです。

- アプリケーションサーバー固有の配備記述子の移行
- 選択したカスタム JSP (Java Server Pages) タグと独自 API の実行時サポート
- 選択した設定パラメータの Application Server における同等の機能による変換
- 移行されたアプリケーションをターゲットサーバーである Application Server に構築および配備するための、Ant ベースのスクリプトの自動生成
- 移行完了後の包括的な移行レポートの生成

Migration Tool は次の場所からダウンロードします。

http://java.sun.com/j2ee/tools/migration/index.html (`http://java.sun.com/j2ee/tools/migration/index.html`)。

Java Application Verification Kit (AVK) for the Enterprise を使用すると、アプリケーションを構築し、そのアプリケーションが J2EE API を正しく使用していることを確認するためのテストを行ったり、特定のガイドラインや規則を使用して他の J2EE 互換アプリケーションサーバーに移行することができます。

Java Application Verification Kit (AVK) は次の場所からダウンロードします。

http://java.sun.com/j2ee/verified/ (`http://java.sun.com/j2ee/verified/`)。

移行されたアプリケーションの配備

ここでの配備とは、これまで旧バージョンの Sun の Application Server またはサードパーティーのアプリケーションサーバープラットフォームに配備されていた状態から移行されたアプリケーションの配備を指しています。

移行されたアプリケーションの配備については、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide』を参照してください。Migration Tool for Sun Java System Application Server 8.2 (J2EE アプリケーション用) を使用してアプリケーションを移行したら、移行後または配備前のタスクを実行する必要があります。ここではその内容について説明します。また、移行されたアプリケーションを実行するために Application Server の準備を行う必要もあります。詳細については、http://java.sun.com/j2ee/tools/migration/doc/postinstall.htmlを参照してください。

アプリケーションを移行する前に

Migration Tool による独自 API やカスタム JSP タグに対するサポートは限定されています。サポートされていない残りの API の使用状況が移行レポートに表示されるので、アプリケーションを配備する前に手動で修正する必要があります。したがって、移行を開始する前に、Application Verification Kit (http://java.sun.com/j2ee/avk/) (AVK) を実行して、アプリケーションの J2EE への準拠の度合いを測定しておくことをお勧めします。

移行プロセスを実際に開始する前に、http://java.sun.com/j2ee/tools/migration/doc/prerun.html を参照して、移行のためのアプリケーションの準備方法を確認してください。

Migration Tool を使用したアプリケーションの移行

Migration Tool は GUI またはコマンド行モードで実行できます。

Solaris および Linux 上で Migration Tool を GUI モードで起動するには、シェルプロンプトで次のコマンドを入力します。

```
<install-dir>/bin/asmigrate.sh -u
```

Solaris および Linux 上で Migration Tool をコマンド行モードで起動するには、シェルプロンプトで次のコマンドを入力します。

```
<install-dir>/bin/asmigrate.sh -c -S <sourceserver> -t <targetdirectory> -T <targetserver> <operand>
または
install-dir>/bin/asmigrate.sh -c -S <sourceserver> -s <sourcedirectory> -t <targetdirectory> -T <targetserver>
```

Windows 上で Migration Tool を起動するには、次の手順を実行します。

1. DOS コマンドプロンプトのウィンドウを開きます。
2. Application Server 8.2 のインストール先のディレクトリに、ディレクトリを変更します。
3. Migration Tool を GUI モードで起動する場合は、DOS プロンプトで次のコマンドを入力します。

   ```
   asmigrate.cmd -u
   ```

   また、Migration Tool を GUI モードで起動する場合は、DOS プロンプトで次のコマンドを入力します。

```
asmigrate.cmd -S <sourceserver> -t <targetdirectory> -T <targetserver> <operand>
or
asmigrate.cmd -c -S <sourceserver> -s <sourcedirectory> -t <targetdirectory> -T <targetserver>
```

`asmigrate` コマンドまたは Migration Tool のコマンド行オプションについては、`http://java.sun.com/j2ee/tools/migration/doc/run.html#CLI` を参照してください。

このツールを使用してアプリケーションを移行するための手順については、`http://java.sun.com/j2ee/tools/migration/doc/StepByStep.html` または Migration Tool のオンラインヘルプを参照してください。

アプリケーションの移行後

Migration Tool によって J2EE アプリケーションを移行したあとは、移行レポートを分析して、生成された出力アプリケーションに対して必要とされる追加変更を確認します。これらの変更を行ってから、移行されたアプリケーションを配備します。このタスクの実行方法については、`http://java.sun.com/j2ee/tools/migration/doc/postrun.html` を参照してください。

◆ ◆ ◆ 第 4 章

EJB 1.1 から EJB 2.0 への移行

Sun Java System Application Server 8.2 では EJB 1.1 仕様も引き続きサポートされますが、拡張機能を活用できるように EJB 2.0 アーキテクチャーを使用することをお勧めします。

EJB 1.1 から EJB 2.0 に移行するには多少の変更が必要ですが、そのうちのいくつかはコンポーネントのソースコード内の変更です。

必要となる変更は、基本的に EJB 1.1 と EJB 2.0 の違いに関連しています。これらすべてについて、次のトピックで説明します。

EJB QL (EJB Query Language)

EJB 1.1 仕様では、個々のアプリケーションサーバーに対する検索メソッド用のクエリーを形成および表現するための方法や言語が残されています。多くのアプリケーションサーバーベンダーでは、開発者が SQL を使用してクエリーを作成することを可能にしていますが、特定のアプリケーションサーバー製品に固有の独自言語を使用している場合もあります。このようにクエリーの実装が混在しているため、アプリケーションサーバー間の不整合が生じています。

EJB 2.0 仕様では、こうした不整合や欠点の多くを修正するために、*EJB Query Language*、つまり *EJB QL* と呼ばれるクエリー言語が導入されています。EJB QL は SQL92 に基づいています。EJB QL は、特にコンテナ管理による持続性を備えたエンティティー Bean に対して、検索メソッドと選択メソッドの両方の形式で、クエリーメソッドを定義します。EJB QL が SQL より優れている主な点は、EJB コンテナにまたがる移植性と、エンティティー Bean の関係をナビゲートする機能です。

ローカルインタフェース

EJB 1.1 アーキテクチャーでは、セッションおよびエンティティー Beans に 1 つのインタフェースのタイプ、つまりリモートインタフェースが備わっています。このインタフェースを通じて、クライアントやその他のアプリケーションコンポーネントからのアクセスが可能になります。リモートインタフェースは、Bean インスタンスがリモート機能を持つように設計されています。Bean は RMI を継承しており、ネットワークを通じて分散クライアントと対話します。

EJB 2.0 では、セッション Bean とエンティティー Bean は、クライアントに対して、リモートインタフェースとローカルインタフェースという 2 種類のインタフェースを通して、メソッドを公開することができます。2.0 のリモートインタフェースは、1.1 で使用されているリモートインタフェースと同じものです。したがって、Bean は RMI を継承しており、ネットワーク層全体にメソッドを公開し、分散クライアントとの対話機能も同じです。

ただし、セッションおよびエンティティー Bean のローカルインタフェースは、ローカルクライアントである (つまり同じ EJB コンテナ内に共存している) EJB からの軽量アクセスをサポートしています。さらに EJB 2.0 仕様では、ローカルインタフェースを使用する EJB が同じアプリケーション内に存在する必要があります。つまり、ローカルインタフェースを使用するアプリケーションの EJB の配備記述子を、1 つの `ejb-jar` ファイル内に格納する必要があります。

ローカルインタフェースは標準の Java インタフェースです。これは RMI を継承していません。エンタープライズ Bean は、ローカルインタフェースを使用して、同じコンテナ内に存在するその他の Bean に対してメソッドを公開します。ローカルインタフェースを使用することで、Bean はもっと緊密にクライアントと連携し、リモートメソッド呼び出しによるオーバーヘッドなしで直接アクセスすることが可能になります。

さらに、ローカルインタフェースでは、参照セマンティクスを使用して Bean 間で値を渡すことが許可されます。オブジェクトそのものではなく、オブジェクトへの参照を渡すようになるので、大量のデータを含むオブジェクトを渡す際に発生するオーバーヘッドが軽減され、パフォーマンスが向上します。

EJB 2.0 のコンテナ管理による持続性 (CMP)

EJB 2.0 仕様では CMP が強化され、複数のエンティティー Bean 間で関係を持たせることが可能になりました。これは、コンテナ管理による関係 (CMR) と呼ばれます。コンテナによって、関係および関係の参照整合性を管理します。

EJB 1.1 仕様で提供されていた CMP モデルにはより多くの制限がありました。EJB 1.1 アーキテクチャーにおける CMP は、データベースやリソースマネージャーのタイプに依存しないデータアクセスに制限されていました。リモートインタフェースを通

して公開できるのは、エンティティー Bean のインスタンス状態のみで、Bean の関係を公開する方法はありませんでした。EJB 1.1 バージョンの CMP は、エンティティー Bean クラスのインスタンス変数をデータベースまたはリソースマネージャー内でこれらの状態を表すデータ項目へマッピングすることに依存しています。CMP のインスタンスフィールドは配備記述子内で指定されます。Bean が配備されると、デプロイヤはツールを使用して、インスタンスフィールドのデータ項目に対するマッピングを実装するコードを生成します。

Bean の実装クラスのコーディング方法を変更する必要もあります。EJB 2.0 仕様では、CMP を使用するエンティティー Bean の実装クラスは、抽象クラスとして定義されるようになります。

この節では、次の項目について説明します。

- 55 ページの「持続フィールドの定義」
- 55 ページの「エンティティー Bean の関係の定義」
- 56 ページの「メッセージ駆動型 Bean」

持続フィールドの定義

EJB 2.0 仕様では、エンティティー Bean のインスタンス変数を CMP フィールドまたは CMR フィールドとして指定することが可能です。これらのフィールドは配備記述子内に定義します。CMP フィールドは `cmp-field` という要素でマークされ、コンテナ管理による関係フィールドは `cmr-field` という要素でマークされます。

実装クラスでは、CMP および CMR フィールドを public 変数として宣言しないように注意してください。その代わりに、これらの CMP および CMR フィールドの値を取得および設定するには、エンティティー Bean 内に `get` および `set` メソッドを定義します。この意味で、2.0 CMP を使用する Bean は JavaBeans モデルに従っていると言えます。クライアントは、インスタンス変数に直接アクセスするのではなく、エンティティー Bean の `get` および `set` メソッドを使用して、これらのインスタンス変数を取得および設定しているからです。`get` および `set` メソッドが関連するのは、CMP または CMR フィールドに指定された変数のみであることに注意してください。

エンティティー **Bean** の関係の定義

前述のとおり、EJB 1.1 アーキテクチャーはエンティティー Bean 間の CMR をサポートしていません。EJB 2.0 アーキテクチャーでは、1 対 1 と 1 対多の両方の CMR がサポートされています。これらの関係は CMR フィールドによって表され、これらのフィールドは配備記述子内で関係としてマークされます。配備記述子内の CMR フィールドは、各アプリケーションサーバーに適した配備ツールを使用して設定します。

CMP フィールドと同じように、Bean では CMR フィールドをインスタンス変数として宣言しません。その代わり、Bean ではこれらのフィールドに対して get および set メソッドが提供されています。

メッセージ駆動型 Bean

メッセージ駆動型 Bean は、EJB 2.0 アーキテクチャーによって導入されたもう 1 つの新機能です。メッセージ駆動型 Bean は、JMS (Java Message Service) を通して配信された非同期メッセージを処理するトランザクション対応のコンポーネントです。JMS API は J2EE 1.3 および J2EE 1.4 プラットフォームに不可欠な部分です。

非同期メッセージではアプリケーションがメッセージ交換によって通信できるため、送信者は受信者と無関係となります。メッセージを送信した送信者は、受信者がそのメッセージを受信または処理するのを待機する必要はありません。ここが同期通信と異なっている点です。同期通信では、別のコンポーネントのメソッドを呼び出しているコンポーネントは、処理が完了し、呼び出し元のコンポーネントにコントロールが戻って来るまで、待機またはブロックする必要があります。

EJB のクライアントアプリケーションの移行

この節では、次の内容について説明します。

JNDI コンテキストにおける EJB の宣言

Sun Java System Application Server 8.2 では、EJB は JNDI サブコンテキスト *ejb/* に体系的にマップしています。JNDI 名 *Account* を EJB に帰属させた場合、Sun Java System Application Server 8.2 によってグローバル JNDI コンテキスト内に、参照 *ejb/Account* が自動的に作成されます。したがって、この EJB のクライアントは、対応するホームインタフェースを取得するために、*ejb/Account* をルックアップする必要があります。

ここで、Sun ONE Application Server 6.x に配備されるサーブレットメソッドのコードを調べてみましょう。

ここに提示されるサーブレットでは、ステートフルセッション Bean の BankTeller を呼び出しています。これは JNDI コンテキストのルートにマップされています。ここでコードを検討するメソッドは、EJB のホームインタフェースを取得して、BankTeller オブジェクトをインスタンス化し、このオブジェクトのリモートインタフェースを取得できるようにします。これによって、ユーザーはこのコンポーネントに対するビジネスメソッド呼び出しを作成できるようになります。

```
/**
   * Look up the BankTellerHome interface using JNDI.
   */
private BankTellerHome lookupBankTellerHome(Context ctx)
      throws NamingException
{
    try
    {
      Object home = (BankTellerHome) ctx.lookup("ejb/BankTeller");
      return (BankTellerHome) PortableRemoteObject.narrow(home,
                  BankTellerHome.class);
    }
    catch (NamingException ne)
    {
      log("lookupBankTellerHome: unable to lookup BankTellerHome" +
          "with JNDI name 'BankTeller': " + ne.getMessage() );
      throw ne;
    }
}
```

このコードでは、ルックアップに対する引数として `ejb/BankTeller` がすでに使用されているので、Sun Java System Application Server 8.2 に配備するためのコード変更を行う必要はありません。

EJB JNDI 参照の使用方法のまとめ

ここでは、EJB JNDI 参照を使用する場合に注意すべき点をまとめています。注意事項の詳細が、特定のソースアプリケーションサーバーのプラットフォームに固有なものである場合は、その旨を記してあります。

JNDI コンテキスト内の EJB 参照の配置

既存の WebLogic アプリケーション内の JNDI コンテキストのルートに EJB がマップされている場合は、前述の JNDI コンテキスト内の EJB 参照の名前を変更する (これらの参照を JNDI コンテキストルートからサブコンテキスト *ejb/* に移動する) 必要があるだけです。

これらの EJB がすでに既存のアプリケーション内の JNDI サブコンテキスト `ejb/` にマップ済みの場合は、何も変更する必要はありません。

ただし、Sun Java Studio IDE 内の配備記述子に EJB の JNDI 名を設定している場合は、EJB の JNDI 名にプレフィックス `ejb/` を含めないようにすることが重要です。これらの EJB 参照は、Sun Java System Application Server 8.2 によって JNDI `ejb/` サブコンテキスト内に自動的に配置されることを忘れないでください。つまり、配備記述子内で EJB に *BankTeller* という JNDI 名が指定されている場合、Sun Java System Application

Server 8.2 によって、この EJB に対する参照は ejb/BankTeller に変換されます。この EJB のクライアントコンポーネントは、ルックアップの実行時に、この JNDI 名を使用する必要があります。

グローバル JNDI コンテキストとローカル JNDI コンテキスト

グローバル JNDI コンテキストを使用した EJB 参照の取得は、Sun Java System Application Server 8.2 に最も有効で適切なアプローチです。しかしながら、J2EE 仕様にできるだけ近い形で、EJB クライアントアプリケーションのローカル JNDI コンテキストを使用して、EJB 参照を取得することをお勧めします。ローカル JNDI コンテキストを使用する場合、まず、クライアント部分の配備記述子内 (Web アプリケーションの場合は web.xml、EJB コンポーネントの場合は ejb-jar.xml) に EJB リソース参照を宣言する必要があります。

CMP エンティティー EJB の移行

ここでは、アプリケーションコンポーネントを EJB 1.1 アーキテクチャーから EJB 2.0 アーキテクチャーに移行する手順について説明します。

CMP 1.1 の Bean を CMP 2.0 に移行するには、まず、特定の Bean が移行可能かどうかを検証する必要があります。この検証作業は次の手順で実行します。

▼ Bean を移行できるかどうか検証する

1. ejb-jar.xml ファイルから <cmp-fields> 名に移動し、オプションのタグ <prim-key-field> が ejb-jar.xml ファイル内に存在していて、指定された値が設定されているかどうかをチェックします。オプションのタグが存在し、値が設定されていれば、次の手順に進みます。

   ejb-jar.xml 内で <prim-key-class> のフィールド名を検索し、クラス名を取得して、そのクラス内で宣言されている public instance variables を取得します。次に、これらの変数の署名 (名前と大文字/小文字の区別) が上記の <cmp-field> 名に一致するかどうか確認します。見つかった変数を分離します。これらの分離されたフィールドで、大文字で始まっているものがないかどうかチェックします。ある場合は移行できません。

2. **Bean** クラスのソースコードを調べて、すべての <cmp-field> 変数の **Java** の型を取得します。

3. すべての <cmp-field> 名を小文字に変更し、これらの名前からアクセサを構築します。たとえば、元のフィールド名が Name で、その **Java** 型が String である場合、アクセス用メソッド署名は次のようになります。

   ```
   Public void setName(String name)Public String getName()
   ```

4 これらのアクセス用メソッド署名を、**Bean** クラス内のメソッド署名と比較します。完全に一致する組み合わせが見つかった場合、移行は実行できません。

5 カスタム検索メソッドの署名とそれに対応する **SQL** を取得します。**SQL** 内に **Join**、**Outer join**、または **OrderBy** が存在するかどうかチェックします。存在する場合は移行できません。**EJB QL** は **Join**、**Outer join**、**orOrderBy** をサポートしていないからです。

6 `java.util.Enumeration` を使用していたすべての **CMP 1.1** ファインダが、`java.util.Collection` を使用する必要があります。これを反映するようにコードを変更してください。**CMP2.0** ファインダは `java.util.Enumeration` を返すことができません。

実際の移行プロセスの実行方法については、59 ページの「Bean クラスの移行」を参照してください。

Bean クラスの移行

ここでは、Bean クラスを Sun Java System Application Server 8.2 に移行するために必要な手順を説明します。

▼ Bean クラスを移行する

1 **Bean** クラス宣言の先頭にキーワード `abstract` を追加します。

たとえば、Bean クラス宣言が次のような場合

```
public class CabinBean implements EntityBean
```

次のように変更します。

```
abstract public class CabinBean implements EntityBean
```

2 アクセサの前に接頭辞としてキーワード `abstract` を付けます。

3 変更後のすべてのアクセサを、**Bean** クラスのソース **(.java)** ファイルにクラスレベルで挿入します。

4 **Bean** クラスのソースファイル内のすべての `cmp` フィールドをコメントにします。

5 **protected** インスタンス変数の宣言を `cmp-field` 名から小文字で構築し、これをクラスレベルで挿入します。

6 すべての `ejbCreate()` メソッド本体を読み込みます。複数の `ejbCreate` が存在する場合もあります。

「`<cmp-field>`=いくつかの値またはローカル変数」というパターンを検索し、「抽象ミュテータメソッド名 (同じ値またはローカル変数)」という表現に置き換えます。

たとえば、移行前の ejbCreate 本体が次のような場合、

```
public MyPK ejbCreate(int id, String name) {
   this.id = 10*id;
   Name = name;   //1
   return null;
}
```

次のように変更します。

```
public MyPK ejbCreate(int id, String name) {
   setId(10*id);
   setName(name);   //1
   return null;
}
```

//1 の抽象アクセサのメソッド署名が、EJB 2.0 仕様によって定められた Camel Case 規約に沿っていることに注目してください。また、キーワード「*this*」が元のソースに存在する場合としない場合がありますが、これは変更後のソースファイルからは削除してください。

7 手順 **5** の ejbPostCreate() メソッドで宣言したすべての **protected** 変数を初期化します。

protected 変数の数は ejbCreate() メソッドの数と同じです。この初期化は、初期化コードを次の方法で挿入することで実行されます。

```
protected String name;  //from step 5
protected int id;  //from step 5
public void ejbPostCreate(int id, String name) {
   name = getName();    /*abstract accessor*/ //inserted in this step
   id  = getId();        /*abstract accessor*/ //inserted in this step
}
```

8 ejbLoad メソッド内部で、**protected** 変数を **Bean** のデータベース状態に設定します。

このためには、次のコード行を挿入します。

```
public void ejbLoad() {
   name = getName();    // inserted in this step
   id = getId();        // inserted in this step
   ...                  // existing code
}
```

9 同じように、データベース状態が更新されるように、ejbStore() 内部の **Bean** の状態を更新します。

ただし、ejbCreate() 外部の主キーに対応する setter を更新することはできないので、このメソッド内に setter を含めないでください。次のコード行を挿入します。

```
public void ejbStore() {
    setName(name);       //inserted in this step
    setId(id);           //Do not insert this if
```

```
                        //it is a part of the
                        //primary key.
    ...                 //already present code
}
```

10 存在するすべての <cmp-field> 変数名を、手順 **5** で宣言した同等の **protected** 変数名に置き換えます。

Bean を移行しない場合、少なくとも <cmp-version>1.*x*</cmp-version> タグを ejb-jar.xml ファイル内の適切な場所に挿入する必要があります。これによって、移行されていない Bean も Sun Java System Application Server 8.2 上で動作し続けます。

ejb-jar.xml の移行

ファイル ejb-jar.xml を Sun Java System Application Server 8.2 に移行するには、次の手順を実行します。

▼ EJB 配備記述子を移行する

EJB 配備記述子ファイル ejb-jar.xml を移行するには、ファイルを編集して次の変更を行います。

1 すべての <cmp-fields> を小文字に変換します。

2 <reentrant> タグの後に <abstract-schema-name> タグを挿入します。

< ejb-name> タグ内と同じように、スキーマ名は Bean の名前になり、ias_ という接頭辞が付きます。

3 次のタグを <primkey-field> タグの後に挿入します。

```
<security-identity>
   <use-caller-identity/>
</security-identity>
```

4 上記で取得した **SQL** を使用して、**SQL** から **EJB QL** を構築します。

5 <query> タグと、そこにネストされたすべての子タグを、すべての必要な情報とともに、<security-identity> タグのすぐ後に挿入します。

カスタム検索メソッド

カスタム検索メソッドは、デフォルトの findByPrimaryKey メソッドではなく、findBy メソッドです。このメソッドはエンティティー Bean のホームインタフェース内で定義することができます。EJB 1.1 仕様では、これらの検索メソッドのロジック

を定義する標準が規定されていないため、EJB サーバーのベンダーは実装を自由に選択できます。この結果、メソッドの定義に使用される手順は、ベンダーの選択するさまざまな実装によって大きく異なっています。

Sun ONE Application Server 6.x では、標準の SQL を使用して検索ロジックを指定します。

この検索メソッドの定義に関する情報は、エンタープライズ Bean の持続性記述子 (`Account-ias-cmp.xml`) 内に、次のように格納されています。

```
<bean-property>
  <property>
    <name>findOrderedAccountsForCustomerSQL</name>
    <type>java.lang.String</type>
    <value>
       SELECT BRANCH_CODE,ACC_NO FROM ACCOUNT where CUST_NO = ?
    </value>
    <delimiter>,</delimiter>
  </property>
</bean-property>
<bean-property>
  <property>
    <name>findOrderedAccountsForCustomerParms</name>
    <type>java.lang.Vector</type>
    <value>CustNo</value>
    <delimiter>,</delimiter>
  </property>
</bean-property>
```

このように、各 `findXXX` 検索メソッドには、配備記述子内に対応する 2 つのエントリ (クエリー用の SQL コードと関連するパラメータ) が存在しています。

Sun Java System Application Server 8.2 ではカスタム検索メソッドのロジックも宣言型ですが、これは EJB QL (EJB query language) に基づいています。

EJB-QL 言語はそれ自身に対して使用することはできません。ファイル `ejb-jar.xml` にある `<ejb-ql>` タグ内で指定する必要があります。このタグは `<query>` タグの内部にあります。これは、EJB 内部でクエリー (検索または選択メソッド) を定義するタグです。EJB コンテナでは、各クエリーを検索または選択メソッドの実装に変換することができます。次に `<ejb-ql>` タグの例を示します。

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>hotelEJB</ejb-name>
      ...
      <abstract-schema-name>TMBankSchemaName</abstract-schema-name>
      <cmp-field>
```

```
      ...
      <query>
        <query-method>
          <method-name>findByCity</method-name>
            <method-params>
              <method-param>java.lang.String</method-param>
            </method-params>
        </query-method>
        <ejb-ql>
          <![CDATA[SELECT OBJECT(t) FROM TMBankSchemaName AS t
                                        WHERE t.city = ?1]]>
        </ejb-ql>
      </query>
    </entity>
    ...
  </enterprise-beans> ...
</ejb-jar>
```

◆ ◆ ◆ 第 5 章

J2EE 1.4 の互換性の問題

次にこの章の内容を示します。

バイナリ互換性

Application Server 8.2 に含まれる Java SDK は J2EE Version 1.4 SDK です。このバージョンの J2EE SDK は J2EE SDK, v1.3 と互換性があります。

ソース互換性

ソースの下位互換性は保たれていません。ソースファイルで新しい J2EE API が使用されている場合、これらは以前のバージョンの J2EE プラットフォームでは使用できません。

一般的なポリシーは次のようになります。

- 保守リリースでは新しい API は導入されないので、相互間のソース互換性は維持されます。ただし、J2EE は J2SE に基づいているので、新しい Application Server のリリースには新しいバージョンの J2SE が含まれている可能性があります。詳細については、次に示す J2SE のマニュアルの互換性の問題に関する説明を参照してください。

  http://java.sun.com/j2se/1.4.2/compatibility.html
  (`http://java.sun.com/j2se/1.4.2/compatibility.html`)

- 機能リリースとメジャーリリースはソースの上位互換性を持っていますが、ソースの下位互換性は持っていません。

非推奨 API は下位互換性のみをサポートするメソッドやクラスであり、このような API を使用していると、`-nowarn` コマンド行オプションを指定していない限り、コンパイラは警告メッセージを生成します。非推奨メソッドやクラスを使用しないようにプログラムを変更することが推奨されますが、現在のところ、このようなメソッドやクラスを完全に削除する計画はありません。

J2EE 1.4 プラットフォーム (J2EE 1.3 リリース以降) との非互換性

Sun Java System Application Server 8.2 リリースは Java 2 Platform, Enterprise Edition, version 1.4 に基づいています。Sun Java System Application Server 7 リリースは Java 2 Platform, Enterprise Edition, version 1.3 に基づいています。

ほとんどの既存のプログラムは、何も変更しなくても Sun Java System Application Server 8.2 リリースで実行できるはずです。しかし、まれな状況や「コーナーケース」で発生するいくつかのマイナーな潜在的な非互換性も存在しているためその状況に対処できるように補足としてここで説明します。

- Java Servlet 仕様 Version 2.4 は Sun Java System Application Server 8.2 リリースに付属しており、次の URL からダウンロードできます。

  http://java.sun.com/products/servlet/ (`http://java.sun.com/products/servlet/`)

  Version 2.3 の仕様は J2EE 1.3 SDK に付属していました。次の各項目は、これらのリリース間の互換性の問題について説明しています。

  - `HttpSessionListenersessionDestroyed` メソッドは、これまではセッションが無効化されたことを通知するために使用されていました。今回のリリースでは、このメソッドはセッションが無効化されようとしていることを通知するために使用されるようになったので、セッションの無効化の前に通知を行います。以前の動作を前提としてコードが作成されている場合は、新しい動作に合うように変更する必要があります。

  - `ServletRequest,getRemotePort,getLocalName,getLocalAddr,getLocalPort`

    今回のバージョンの仕様では、`ServletRequest` インタフェースに次のメソッドが追加されています。この追加によって、開発者が `ServletRequest` インタフェースを実装した場合などに、ソース互換性が失われる場合があるので注意が必要です。このような場合は、次の新しいメソッドがすべて実装されていることを確認します。

    - `public int getRemotePort()` は、クライアントのインターネットプロトコル (IP) のソースポートまたは最後に要求を送信したプロキシを返します。
    - `public java.lang.String getLocalName()` は、要求を受信した IP インタフェースのホスト名を返します。

- `public java.lang.String getLocalAddr()` は、要求を受信したインタフェースの IP アドレスを返します。
- `public int getLocalPort()` は、要求を受信したインタフェースの IP ポート番号を返します。

Java Server Pages (JSP) 仕様 2.0 は Sun Java System Application Server 8.2 リリースに付属しており、次の URL からダウンロードできます。

http://java.sun.com/products/jsp/ (`http://java.sun.com/products/jsp/`)

JSP 仕様 1.2 は J2EE 1.3 SDK に付属していました。JSP 2.0 仕様は、可能な限り JSP 1.2 仕様との完全な下位互換を保つように配慮されています。JSP 1.2 仕様には一部あいまいな部分がありましたが、これらは JSP 2.0 仕様で明確化されています。いくつかの JSP 1.2 コンテナは異なる動作を行うため、コンテナ固有の動作に依存するアプリケーションについては、JSP 2.0 環境で正しく動作するように調整しなければならない場合があります。

JSP に関連する既知の下位互換性の問題を次に示します。

- 名前空間を認識せず、プレフィックスパラメータのみに依存するタグライブラリのバリデータは、いくつかの JSP 2.0 ページの妥当性を正しく検査できない可能性があります。これは、XML ビューでは `jsp:root` 以外の要素にタグライブラリ宣言が格納されていることがあり、同じタグライブラリ宣言が異なるプレフィックスを使用して何度も格納される可能性があるからです。これらの代わりに、タグライブラリのバリデータでは常に uri パラメータを使用するようにしてください。既存のタグライブラリを含む既存の JSP ページでは何も問題は発生しません。
- いくつかのコンテナで、I18N の動作が異なっていることがあります。これは主に JSP 1.2 仕様のあいまいさが原因です。可能な限り、下位互換性への影響を最小限に抑えるための手段が講じられた結果、全体的に I18N のテクノロジ機能は大幅に改善されました。

  JSP 2.0 より前の JSP 仕様のバージョンでは、XML 構文の JSP ページと標準構文の JSP ページは、同じ方法でページエンコーディングを決定していました。つまり、それぞれのページ指令の `pageEncoding` または `contentType` 属性を調べることによってページエンコーディングを決定しており、どちらの属性も存在しない場合は、デフォルトで ISO-8859-1 が設定されていました。

  JSP 仕様 v2.0 からは、JSP ドキュメントのページエンコーディングは、XML 仕様のセクション 4.3.3 および付録 F.1 の説明のとおりに決定されており、これらのページの `pageEncoding` 属性は、XML 仕様のとおりに決定されたページエンコーディングとの整合性を確認するだけのためにチェックされます。

  この変更の結果、`pageEncoding` 属性から決定されるページエンコーディングに依存する JSP ドキュメントは、正しくデコードされなくなりました。したがって、これらの JSP ドキュメントには適切な XML エンコーディング宣言を追加する必要があります。

また、JSP 1.2 仕様のページエンコーディングは変換ユニットごとに決定されますが、JSP 2.0 仕様のページエンコーディングは、ファイルごとに決定されます。したがって、.jsp に b.jsp が静的に含まれており、ページエンコーディングは b.jsp ではなく a.jsp で指定されている場合、JSP 1.2 仕様では、b.jsp に対して a.jsp のエンコーディングが使用されますが、JSP 2.0 仕様では b.jsp に対してデフォルトのエンコーディングが使用されます。

- 型強制規則 (JSP 2.0 仕様の表 JSP.1-11 参照) は、EL 強制規則との整合性がとられました。このため、いくつかの例外条件について、JSP 2.0 仕様では例外が発生しなくなりました。特に、空の文字列を数値タイプの属性に渡した場合、これまでは変換エラーか `NumberFormatException` が発生していましたが、JSP 2.0 仕様では、代わりに 0 が渡されるようになりました。詳細については、JSP 2.0 仕様の 表 JSP.1-11 を参照してください。通常、この調整により問題が発生することはありません。なぜなら、これらは JSP 1.2 仕様の例外条件であり、この仕様ではこれらの例外が変換時または要求時に発生することが許容されていたからです。
- JSP コンテナは `web.xml` を使用して、さまざまなコンテナ機能のデフォルト動作を決定します。次の項目は、JSP 開発者が `web.xml` ファイルをサーブレットバージョン 2.3 仕様からサーブレットバージョン 2.4 仕様にアップグレードするときに注意すべき点を示したものです。
  - EL 式は、JSP 1.2 テクノロジによって作成されたアプリケーションでは、デフォルトで無視されます。Web アプリケーションを JSP 2.0 仕様にアップグレードする場合、EL 式はデフォルトで解釈されます。エスケープシーケンス `\\$` を使用して、コンテナによって解釈されるべきではない EL 式をエスケープすることができます。あるいは、`isELIgnored` ページ指令や、`el-ignored` 設定要素によって、変換ユニット全体に対して EL を無効化することもできます。JSTL 1.0 のユーザーは、`taglib/` インポートを JSTL 1.1 URI にアップグレードするか、`_rt` バージョンのタグを使用する必要があります (たとえば、`c` の代わりに `c_rt` 、`fmt` の代わりに `fmt_rt` を使用)。
  - 拡張子が `.jspx` のファイルは、デフォルトで JSP ドキュメントとして解釈されます。JSP 設定要素 `is-xml` を使用して、`.jspx` ファイルを正規の JSP ページとして処理します。JSP コンテナから `.jspx` の関連付けを解除する方法はありません。
  - エスケープシーケンス `\\$` は、JSP 1.2 仕様では予約されていませんでした。JSP 1.2 仕様では、`\\$` という表示のテンプレート文字や属性値を使用すると `\\$` と出力されていましたが、現在では `$` のみが出力されます。

JAXP と SAX の非互換性

Sun Java System Application Server 8.2 は JAXP 1.3 をサポートしています。つまり SAX 2.0.2 をサポートしていることになります。SAX 2.0.2 では、`DeclHandler.externalEntityDecl` が、`DTDHandler.unparsedEntityDecl` との整合性を保つために、絶対的なシステム識別子を返すようパーサーに要求します。このことによって、SAX 2.0.0 を使用するアプリケーションを移行するときに、いくつかの非互換性が発生します。

これまでの `externalEntityDecl` の動作を変更せずに、SAX 2.0.0 を使用するアプリケーションを SAX 2.0.2 に移行するには、resolve-dtd-uris 機能を false に設定します。次に例を示します。

```
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setFeature("http://xml.org/sax/features/resolve-dtd-uris",false);
```

SAX 2.0.0 と SAX 2.0.2 との間のその他の非互換性については、『JAXP Compatibility Guide』を参照してください。

pass-by-reference 要素

Sun Java System Application Server 8.2 は、デフォルトで Java 2 Platform, Enterprise Edition 仕様と互換性があります。この場合、移植性のあるすべての J2EE プログラムは、変更なしで Application Server 上で実行できます。ただし、J2EE の互換性要件で許可されている内容に従って、J2EE 仕様と互換性がない Sun Java System Application Server 8.2 の機能を使用するようにアプリケーションを設定することもできます。

`sun-ejb-jar.xml` ファイル内の pass-by-reference 要素は、リモート呼び出しのみに適用されます。EJB 2.0 仕様、セクション 5.4 で定義されているように、ローカルインタフェースへの呼び出しは pass-by-reference (参照渡し) のセマンティクスを使用します。

pass-by-reference 要素のデフォルト値が false に設定されている場合、リモートインタフェースへの呼び出しのセマンティクスを渡すパラメータは、EJB 2.0 仕様、セクション 5.4 に準拠します。true に設定されている場合、この仕様とは逆に、リモート呼び出しでは pass-by-value (値渡し) のセマンティクスではなく pass-by-reference (参照渡し) のセマンティクスが必要になります。

移植性のあるプログラムでは、このような呼び出しのときにオブジェクトのコピーが作成されることは想定できないので、元のオブジェクトを変更しても安全です。コピーが作成されないことも想定できないので、オブジェクトへの変更は、呼び出し元と呼び出し先の両方に表示されます。このフラグが true に設定されている場合、パラメータと戻り値は読み取り専用であると見なされます。このようなパラメータや戻り値を変更するプログラムの動作は定義されていません。

pass-by-reference 要素の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide 』を参照してください。

delegate 属性

sun-web.xml ファイルの class-loader 要素にある delegate 属性のデフォルト値が true に設定されている場合、コンテナ全体のライブラリ JAR ファイルにあるクラスとリソースの方が、WAR ファイル内にパッケージされたクラスやリソースよりも優先して読み込まれます。これは、サーブレット 2.3 仕様、セクション 9.7.2 で推奨されている内容と相反しています。false に設定されている場合、クラスローダーの委任動作は、サーブレット 2.3 仕様、セクション 9.7.2 で推奨されている内容に準拠しています。

delegate 属性の値を true にして使用している移植性のあるプログラムを、J2EE 仕様の一部であるクラスやインタフェースと一緒にパッケージしないでください。このようなクラスまたはインタフェースが WAR ファイルに含まれているプログラムの動作は定義されていません。class-loader 要素の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide 』を参照してください。

◆ ◆ ◆ 第 6 章

Application Server 6.x/7.x からの移行

この章では、J2EE アプリケーションを Application Server Enterprise Edition 8.2 製品ラインに移行するときに必要な注意点と方針について説明します。

以降のセクションでは、一般的な J2EE アプリケーションの主要なコンポーネントを Application Server 6.x/7.x から Application Server Enterprise Edition 8.2 に移行する間に発生する問題について説明します。

この章の内容は次のとおりです。

この章で説明する移行の問題は、擬似的なオンラインバンキングサービスである *iBank* という J2EE アプリケーションに関して実際に行った Application Server 6.x/7.x から Sun Java System Application Server 8.2 への移行に基づいています。このアプリケーションは、従来の J2EE アプリケーションのすべての側面を反映しています。

iBank アプリケーションでは、J2EE 仕様の次の領域が網羅されています。

- サーブレット、特に JSP ページへのリダイレクションを伴うもの (モデルビューコントローラアーキテクチャー)

- JSP ページ、特にページの静的および動的な取り込みを伴うもの
- JSP カスタムタグライブラリ
- HTTP セッションの作成と管理
- JDBC API によるデータベースアクセス
- Enterprise Java Beans (EJB): ステートフルおよびステートレスセッション Bean、CMP および BMP エンティティー Bean。
- J2EE アプリケーションの標準的なパッケージ方法に沿ったアセンブリと配備

Application Server 6.x からの配備記述子の移行

配備記述子には、標準配備記述子と実行時配備記述子の 2 種類があります。標準配備記述子は、J2EE プラットフォームのバージョン間やベンダー間で移植性があり、変更を必要としません。現時点では、標準規格の解釈による例外があります。次の表は、そのような配備記述子の一覧です。

ソース配備記述子	ターゲット配備記述子
`ejb-jar.xml` - 1.1	`ejb-jar.xml` - 2.0
`web.xml`	`web.xml`
`application.xml`	`application.xml`

J2EE の標準配備記述子 `ejb-jar.xml`、`web.xml`、および `application.xml` には、大幅な変更はありません。ただし、アプリケーションを Sun Java System Application Server 8.2 上に配備できるようにするため、`ejb-jar.xml` 配備記述子は EJB 2.0 仕様に準拠するように変更されています。

実行時配備記述子は、ベンダーおよび製品に固有のものであり、その形式が異なるため、アプリケーションサーバー間で移植性がありません。このため、配備記述子の移行が必要です。この節では、実行時配備記述子を手動で作成し、関連する情報を移行する方法について説明します。

次の表に、配備記述子の移行マッピングの概要を示します。

ソース配備記述子	ターゲット配備記述子
`ias-ejb-jar.xml`	`sun-ejb-jar.xml`
`<bean-name>-ias-cmp.xml`	`sun-cmp-mappings.xml`
`ias-web.xml`	`sun-web.xml`

Application Server 8.2 に移行するときは、DTD が適合しないため、Application Server 6.x の標準配備記述子を変更する必要があります。

`sun-ejb-jar.xml` と `sun-web.xml` を作成するために必要な情報の大部分は、それぞれ `ias-ejb-jar.xml` と `ias-web.xml` に由来します。ただし、移行する `sun-ejb-jar.xml` に宣言がある場合は、一部の必要な情報が CMP エンティティー Bean のホームインタフェース (.java ファイル) から抽出されます。これは、その CMP エンティティー Bean のホームインタフェース内の情報を必要とする `sun-ejb-jar.xml` 内の `<query-filter>` コンストラクトを構築するために必要な条件です。移行時にこのソースファイルが見つからない場合は、移行された `sun-ejb-jar.xml` に `<query-filter>` コンストラクトが作成されますが、情報は設定されず、「REPLACE ME」の形で表されます。

さらに、`ias-ejb-jar.xml` に `<message-driven>` 要素が含まれている場合は、この要素内の情報が取得され、`ejb-jar.xml` と `sun-ejb-jar.xml` の両方に情報を設定するために使用されます。また、`ias-ejb-jar.xml` の `<message-driven>` 要素内には、MDB が待機するトピックまたはキューの JNDI 名を保持する `<destination-name>` 要素があります。Application Server 6.5 では、この JNDI 名の命名規則は `cn=<SOME_NAME>` です。この名前を持つ JMS トピックまたはキューは Application Server に配備できないため、アプリケーションサーバーはこれを `<SOME_NAME>` に変更し、この情報を `sun-ejb-jar.xml` に挿入します。この変更を、すべての有効な入力ファイル (すべての `.java` ファイル、`.jsp` ファイル、および `.xml` ファイル) に反映してください。このため、この JNDI 名の変更はアプリケーション全体に伝達されます。この JNDI 名への参照を含むソースファイルがあり、それらを使用できない場合、管理者はアプリケーションを配備できるように手動で変更してください。

Application Server 6.x からの Web アプリケーションの移行

Application Server 6.x は、サーブレット (Servlet API 2.2) および JSP (JSP 1.1) をサポートします。Sun Java System Application Server 8.2 は、Servlet API 2.4 と JSP 2.0 をサポートします。

これらの環境内では、アプリケーションの各種コンポーネント (サーブレット、JSP、HTML ページ、その他のリソース) をまとめて 1 つのアーカイブファイル (J2EE 標準の Web アプリケーションモジュール) にグループ化し、それをアプリケーションサーバーに配備することが不可欠です。

J2EE 仕様によれば、Web アプリケーションは次のような構造を持つアーカイブファイル (WAR ファイル) です。

- アプリケーションの HTML ページ、JSP、イメージ、およびその他の静的リソースを含むルートディレクトリ。
- 使用する SDK のバージョン情報と (オプションで) アーカイブに含まれるファイルのリストを格納したアーカイブマニフェストファイル `MANIFEST.MF` を含む `META-INF/` ディレクトリ。

- アプリケーションの配備記述子 (`web.xml` ファイル) とアプリケーションによって使用されるすべての Java クラスおよびライブラリを含む `WEB-INF/` ディレクトリ。このディレクトリは次のように構成されています。
  - アプリケーションのコンパイル済みクラス (サーブレットや補助クラス) のツリー構造を含み、パッケージに編成された `classes/` サブディレクトリ。
    - アプリケーションによって使用される Java ライブラリ (JAR ファイル) を含む `lib/` ディレクトリ。

Java Server Pages (JSP) と JSP カスタムタグライブラリの移行

Application Server 6.x は JSP 1.1 仕様に準拠し、Application Server 8.2 は JSP 2.0 仕様に準拠します。

JSP 2.0 仕様には、多くの新機能に加えて、JSP 1.1 仕様の更新が含まれています。

これらの変更は機能拡張であり、JSP ページの JSP 1.1 から 2.0 への移行には必要ありません。

Application Server 6.x の JSP カスタムタグライブラリの実装は、J2EE 仕様に準拠します。その結果、JSP カスタムタグライブラリの Application Server Enterprise Edition 8.2 への移行では、特に問題は発生せず、変更も必要ありません。

サーブレットの移行

Application Server 6.x は、Servlet 2.2 API をサポートします。Sun Java System Application Server 8.2 は、Servlet 2.4 API をサポートします。

Servlet API 2.4 では、サーブレットのコアはあまり変更されていません。ほとんどの変更は、コアの外部に追加された新機能に関係しています。

もっとも重要な機能を次に示します。

- サーブレットに JDK 1.2 以降が必要になった
- フィルタ機構が作成された
- アプリケーションライフサイクルイベントが追加された
- 国際化サポートが追加された
- エラー属性とセキュリティー属性が拡張された
- `HttpUtils` クラスが非推奨になった
- いくつかの DTD 動作が拡張および明確化された

これらの変更は機能拡張であり、サーブレットを Servlet API 2.2 から 2.4 に移行する場合には必要ありません。

ただし、アプリケーション内のサーブレットが JNDI を使用して J2EE アプリケーション内のリソース (データソースや EJB など) にアクセスする場合は、ソースファイルや配備記述子にいくつかの変更が必要になることがあります。

これらの変更の詳細については、次の節で説明します。

- 75 ページの「JNDI コンテキストからのデータソースの取得」
- 75 ページの「JNDI コンテキストでの EJB の宣言」

最後のシナリオとして、サーブレットコードの変更が必要になる場合があります。JSP ページに既存の Java クラスと同じ名前が存在する場合は、Application Server 6.x で名前の競合が発生する可能性があります。このような場合は、JSP ページ内の該当する名前を変更することによって、競合を解決してください。さらに、場合によっては、この JSP ページを呼び出すサーブレットのコードも編集する必要があります。この問題は、新しいクラスローダー階層を使用する Application Server では解決されています。新しいバージョンのアプリケーションサーバーでは、特定のアプリケーションに関して、1 つのクラスローダーがすべての EJB モジュールを読み込み、別のクラスローダーが Web モジュールを読み込みます。これら 2 つのローダーどうしは通信しないため、名前の競合は発生しません。

JNDI コンテキストからのデータソースの取得

JNDI コンテキストにバインドされたデータソースへの参照を取得するには、初期コンテキストオブジェクトからデータソースの JNDI 名を検索します。次に、このようにして取得されたオブジェクトを次のように `DataSource` タイプのオブジェクトとしてキャストします。

```
ds = (DataSource)ctx.lookup(JndiDataSourceName);
```

詳細は、「JDBC コードの移行」を参照してください。

JNDI コンテキストでの EJB の宣言

第 4 章の 56 ページの「JNDI コンテキストにおける EJB の宣言」を参照してください。

サーブレットおよび JSP の移行に関する潜在的な問題

サーブレットまたは JSP アプリケーションのコンポーネントを実際に Application Server 6.x から Application Server 8.2 に移行するときは、コンポーネントのコードを変更する必要はありません。

Web アプリケーションがデータソースなどのサーバーリソースを使用する場合、Application Server ではそのリソースを `web.xml` ファイル内で宣言し、`sun-web.xml` ファイル内でも同様に宣言する必要があります。`jdbc/iBank` という名前のデータソースを宣言する場合は、`web.xml` ファイルの `<resource-ref>` タグは次のようになります。

```
<resource-ref>
   <res-ref-name>jdbc/iBank</res-ref-name>
   <res-type>javax.sql.XADataSource</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

これに対応する sun-web.xml ファイル内の宣言は、次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE FIX ME: need confirmation on the DTD to be used for this file
<sun-web-app>
   <resource-ref>
      <res-ref-name>jdbc/iBank</res-ref-name>
      <jndi-name>jdbc/iBank</jndi-name>
   </resource-ref>
 </sun-web-app>
```

Web アプリケーションモジュールの移行

アプリケーションを Application Server 6.x から Sun Java System Application Server 8.2 に移行するときは、Java コードや Java Server Pages を変更する必要はありません。ただし、次のファイルは変更する必要があります。

- web.xml
- ias-web.xml

Application Server は J2EE 1.4 標準規格に準拠しているため、WAR ファイル内の web.xml ファイルは改訂された DTD (http://java.sun.com/dtd/web-app_2_3.dtd) に従う必要があります。この DTD は旧バージョンの DTD のスーパーセットなので、移行する web.xml ファイル内の <! DOCTYPE 定義を変更するだけで済みます。変更された <! DOCTYPE 宣言は、次のようになります。

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//
                        DTD Web Application 2.3//EN"
                        "http://java.sun.com/dtd/web-app_2_3.dtd">
```

Application Server Enterprise Edition 8.2 では、このファイルの名前が sun-web.xml に変更されます。

この XML ファイルでは、Web アプリケーションに必要な Application Server 固有のプロパティーとリソースを宣言する必要があります。

このファイルに追加すべき重要な内容については、75 ページの「サーブレットおよび JSP の移行に関する潜在的な問題」を参照してください。

Application Server 6.5 アプリケーションの ias-web.xml が存在し、Application Server 6.5 固有のプロパティーが宣言されている場合は、このファイルを Application Server の標準規格に移行する必要があります。DTD のファイル名は、sun-web.xml に変更してください。詳細は、URL http://wwws.sun.com/software/dtd/appserver/sun-web-app_2_4-1.dtd を参照してください。

web.xml ファイルと ias-web.xml ファイルにこれらの変更を行ったあとは、Application Server の配備ツール GUI インタフェースまたは asadmin コマンド行ユーティリティーから Web アプリケーション (WAR ファイル) を配備できます。配備コマンドでは、アプリケーションのタイプを Web として指定してください。

asadmin コマンド行ユーティリティーを起動するには、Application Server の bin ディレクトリで asadmin.bat ファイルまたは asadmin.sh スクリプトを実行します。

asadmin プロンプトで入力するコマンドを次に示します。

```
asadmin deploy -u username -w password
-H hostname
-p adminport
--type web
[--contextroot contextroot]
[--force=true]
[--name component-name]
[--upload=true] filepath
```

Application Server 6.x からのエンタープライズ EJB モジュールの移行

Application Server 6.x は EJB 1.1 をサポートし、Application Server は EJB 2.0 をサポートします。このため、どちらも次のものをサポートできます。

- ステートフルまたはステートレスなセッション Bean
- BMP または CMP を持つエンティティー Bean

ただし、EJB 2.0 には MDB と呼ばれる新しいタイプのエンタープライズ Bean が導入されています。

J2EE 1.4 仕様によれば、EJB の各種コンポーネントは次のような構造を持つ JAR ファイルにグループ化する必要があります。

- ejb-jar.xml という名前の XML 配備記述子を含む META-INF/ ディレクトリ
- パッケージ内のホームインタフェースとリモートインタフェース、および Bean の実装クラスと補助クラスに対応する .class ファイル

Application Server 6.x では、このアーカイブ構造が使用されます。ただし、EJB 1.1 仕様では、各 EJB コンテナベンダーに対して、彼らが次のような特定のものを適当と見なした場合には実装させる余地を残しています。

- CMP EJB のデータベースの持続性 (特に、データベース表における Bean の CMP フィールドと列とのマッピング設定)
- CMP Bean に対するカスタム検索メソッドのロジックの実装
- Application Server 6.x と Application Server 8.2 では移行の処理方法が同じでないため、次のようにいくつかの XML ファイルを変更する必要があります。
- 最新の DTD URL を指し示すように `<!DOCTYPE` 定義を変更してください (`ejb-jar.xml` などの J2EE の標準配備記述子の場合)。
- `ias-ejb-jar.xml` ファイルをこのファイルの修正バージョン (たとえば、DTD に従って手動で作成した `sun-ejb-jar.xml`) に置き換えます。詳細は、http://www.sun.com/software/dtd/appserver/sun-ejb-jar_2_1-1.dtd を参照してください。
- すべての *<ejb-name>*`-ias-cmp.xml` ファイルを、手動で作成した 1 つの `sun-cmp-mappings.xml` ファイルに置き換えます。詳細は、http://www.sun.com/software/dtd/appserver/sun-cmp-mapping_1_2.dtd を参照してください。
- CMP エンティティー Bean の場合は、必要に応じて Application Server の `bin` ディレクトリ内の `capture-schema` ユーティリティーを使用して dbschema を生成します。次に、それをエンティティー Bean の `META-INF` ディレクトリより上位に配置します。

EJB の移行

第 3 章で説明したように、Application Server 6.x は EJB 1.1 仕様をサポートするのに対し、Application Server は EJB 2.0 仕様もサポートします。EJB 2.0 仕様では、次のような新機能がアーキテクチャーに導入されています。

- MDB
- CMP の向上
- CMP を持つエンティティー Bean のコンテナ管理関係
- ローカルインタフェース
- EJB クエリー言語 (EJB QL)

Application Server でも EJB 1.1 仕様は引き続きサポートされますが、EJB 2.0 の拡張機能を利用できるように、EJB 2.0 アーキテクチャーの使用をお勧めします。

EJB 1.1 から EJB 2.0 への移行の詳細については、第 4 章を参照してください。

Application Server Platform Edition 8.2 固有の EJB の変更

Application Server 6.x から Application Server 8.2 に EJB を移行するときは、EJB のコードを変更する必要はありません。ただし、次のような DTD の変更が必要です。

セッション Bean

- `ejb-jar.xml` などの J2EE の標準配備記述子で、最新の DTD を指し示すように `<!DOCTYPE` 定義を変更してください。
- `ias-ejb-jar.xml` ファイルをこのファイルの修正バージョン (配備記述子に従って手動で作成した `sun-ejb-jar.xml` という名前のファイル) に置き換えます。詳細は、 http://wwws.sun.com/software/dtd/appserver/sun-ejb-jar_2_1-1.dtd を参照してください。
- `sun-ejb-jar.xml` ファイルでは、すべての JNDI 名の「`ejb/`」の前に、すべての EJB の JNDI 名を追加する必要があります。その理由は、Application Server 6.5 では EJB の JNDI 名として `ejb/`<*ejb-name*> (<*ejb-name*> は `ejb-jar.xml` ファイル内で宣言された EJB の名前) しか指定できないためです。

  Application Server では、`sun-ejb-jar.xml` に新しいタグが導入されました。これは、EJB の JNDI 名を宣言する場所です。

注 – アプリケーション内の JNDI 名が変更されるのを防ぐため、EJB の JNDI 名を <*jndi-name*> タグ内で `ejb/`<*ejb-name*> として宣言してください。

SFSB フェイルオーバーをサポートする EJB アプリケーションの移行

Sun ONE Application Server 6.5 は、ステートフルセッション Bean のフェイルオーバーをサポートします。6.5 で SFSB フェイルオーバーを利用するには、フェイルオーバーと DSync (Distributed Store) を使用してセッションを設定する必要があります。DSync 機構は、実行中のセッション Bean の対話状態を保存するために使用されます。

注 – Sun ONE Application Server 6.5 は、RMI/IIOP パス上のリッチクライアントに関して、ステートフルセッション Bean のフェイルオーバーをサポートしません。このようなアプリケーションは、Sun Java System Application Server 8.2 の RMI/IIOP パス上の SFSB フェイルオーバーを利用できます。SFSB フェイルオーバーの設定の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「ステートフルセッション Bean のフェイルオーバー」を参照してください。

Sun Java System Application Server 8.2, Enterprise Edition は、ステートフルセッション Bean のフェイルオーバーをサポートします。Application Server 8.2 は、セッション

データの格納に高可用性データベース (HADB) を使用します。SFSB フェイルオーバーのサポートで順守される原則は、SFSB の対話状態をそのライフサイクルのあらかじめ定義された時点で持続的ストアに保存することです。この機構は、チェックポイントと呼ばれます。サーバーがクラッシュした場合は、SFSB のチェックポイント化された状態を持続的ストアから取得できます。セッションデータの格納に HADB を使用するには、HADB を永続的ストアとして設定してください。HTTP セッションとステートフルセッション Bean の基本となるストアは同じであり、永続的ストアの設定はセッションストアの設定とほぼ同じです。

セッションフェイルオーバーのための HADB の設定については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 9 章「高可用性 (HA) セッション持続性とフェイルオーバーの設定」を参照してください。

Sun ONE Application Server 6.5 に配備されたステートフルセッション Bean を Sun Java System Application Server 8.2 に移行するときは、EJB のコードを変更する必要はありません。ただし、次の手順を実行する必要があります。

- `ejb-jar.xml` などの J2EE の標準配備記述子の場合は、最新の DTD URL を指し示すように `<!DOCTYPE` 定義を変更します。
- `ias-ejb-jar.xml` ファイルをこのファイルの修正バージョン (つまり、DTD に従って手動で作成した `sun-ejb-jar.xml`) に置き換えます。
- すべての *<ejb-name>*`-ias-cmp.xml` ファイルを、手動で作成した 1 つの `sun-cmp-mappings.xml` ファイルに置き換えます。
- SFSB 状態のフェイルオーバーのサポートを利用するために、アプリケーションのソースコードを変更する必要はありません。SFSB のチェックポイント化に必要なすべての設定は、Application Server 固有の配備記述子 (`sun-ejb-jar.xml`) またはドメイン設定ファイル (`domain.xml`) に適用されます。

  ただし、サーブレット経由で EJB にアクセスする場合は、EJB のホーム参照とリモート参照をセッションに格納する必要があります。`ejbHome` インタフェースと `ejbRemote` インタフェースをセッションに格納するコード例を次に示します。

  ```
  session.setAttribute("ejbhome", ejbHome);
  session.setAttribute("ejbremote", ejbRemote);
  ```

  次のコード例は、セッションから `ejbHome` と `ejbRemote` を取得する方法を示しています。

  ```
  ejbHome = session.getAttribute("ejbhome");
  ejbRemote = session.getAttribute("ejbremote");
  ```

- `domain.xml` で、`availability-service` 要素の `availability-enabled` 属性が TRUE に設定されていることを確認します。`availability-enabled` 属性が TRUE に設定されている場合は、サーバーインスタンスのレベルでフェイルオーバーが有効になっていることを示します。つまり、サーバーインスタンスが要求を処理できなかった場合は、その要求は使用可能な次のサーバーインスタンスにルーティングされます。

SFSB チェックポイントによって EJB コンテナのパフォーマンスにオーバーヘッドが発生するので、アプリケーションにとって状態のフェイルオーバーが重要である SFSB だけにチェックポイントを制限することもできます。

`sun-ejb-jar.xml` では、チェックポイントをメソッドレベルで有効または無効にできます。詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「チェックポイントを設定するメソッドの指定」を参照してください。

6.5 の SFSB EJB モジュールの配備記述子 (`ias-ejb-jar.xml`) で `session` 要素の `failoverrequired` 属性が TRUE に設定されている場合は、Application Server 8.2 環境でそのような EJB モジュールに対する可用性サービスを有効にすることもできます。

エンティティー Bean

- `ejb-jar.xml` などの J2EE の標準配備記述子を含む最新の DTD を指し示すように `<!DOCTYPE` 定義を変更してください。
- `ejb-jar.xml` ファイル内のすべての CMP について、`<cmp-version>` タグの値を 1.1 に更新します。
- すべての *<ejb-name>*`-ias-cmp.xml` ファイルを、手動で作成した `sun-cmp-mappings.xml` ファイルに置き換えます。詳細は、http://wwws.sun.com/software/dtd/appserver/sun-cmp-mapping_1_2.dtd を参照してください。
- Application Server インストールの bin ディレクトリ内の `capture-schema` ユーティリティーを使用して `dbschema` を生成し、それをエンティティー Bean の `META-INF` フォルダより上位に配置します。
- `ias-ejb-jar.xml` を Application Server の `sun-ejb.jar.xml` に置き換えます。
- Application Server 6.5 では、検索の SQL が *<ejb-name>*`-ias-cmp.xml` に直接埋め込まれていました。Application Server では、数式を使用して各種の検索メソッドの `<query-filter>` が宣言されています。

メッセージ駆動型 Bean

Application Server はメッセージ駆動をシームレスにサポートします。このサポートは、Sun Java System Message Queue と Application Server を密接に統合し、ネイティブの組み込み型 JMS サービスを提供することによって実現されます。

Message Queue をインストールすると、任意の数の Application Server インスタンスをサポートする JMS メッセージングシステムが Application Server に提供されます。各サーバーインスタンスには、デフォルトで、そのインスタンスで実行中のすべての JMS クライアントをサポートする組み込み型 JMS サービスが関連付けられています。

『Enterprise JavaBeans Specification, v2.0』で定義されたコンテナ管理トランザクションと Bean 管理トランザクションの両方がサポートされます。

iPlanet Application Server のメッセージ駆動型 Beans サポートは、開発者に限定されており、古い専用 API を多用していました。メッセージングサービスは、iPlanet Message Queue for Java 2.0 によって提供されていました。`Queue Connection Factory` オブジェクトを設定するには、iPlanet Application Server の下に LDAP ディレクトリも必要でした。

現在は、`QueueConnectionFactory` と Application Server 内でメッセージ駆動型 Beans を設定するために必要なその他の要素を `ejb-jar.xml` ファイルに指定する必要があります。

配備記述子の変更の詳細については、72 ページの「Application Server 6.x からの配備記述子の移行」を参照してください。メッセージ駆動型 Bean については、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide』の「Using Message-Driven Beans」を参照してください。

Application Server 6.x からのエンタープライズアプリケーションの移行

J2EE 仕様によれば、エンタープライズアプリケーションは次のような構造を必要とする EAR ファイルです。

- `application.xml` という名前の J2EE アプリケーションの XML 配備記述子を含む `META-INF/` ディレクトリ
- エンタープライズアプリケーションの EJB モジュール用の .JAR アーカイブファイルと Web モジュール用の .WAR アーカイブファイル

アプリケーション配備記述子では、エンタープライズアプリケーションと Web アプリケーションのコンテキストルートを構成するモジュールが定義されます。

Application server 6.x と Application Server 8.2 は、主に J2EE モデルをサポートします。このモデルでは、アプリケーションがエンタープライズアーカイブ (EAR) ファイル (拡張子は `.ear`) の形式でパッケージ化されます。アプリケーションはさらに J2EE モジュールの集まりに分割され、EJB 用の Java アーカイブ (拡張子が `.jar` の JAR ファイル) とサーブレットおよび JSP 用の Web アーカイブ (拡張子が `.war` の WAR ファイル) にパッケージ化されます。

エンタープライズアプリケーションを配備する前に、次に示す手順を実行することが不可欠です。

▼ EAR ファイルを構築する

1. **EJB** を **1** つ以上の **EJB** モジュールにパッケージ化します。
2. **Web** アプリケーションのコンポーネントを **Web** モジュールにパッケージ化します。
3. **EJB** モジュールと **Web** モジュールを組み合わせて **1** つのエンタープライズアプリケーションモジュールを作成します。
4. エンタープライズアプリケーションのルートコンテキストの名前を定義します。これによって、アプリケーションにアクセスするための **URL** が決まります。

   Application Server では、Application Server 6.x にはなかった新しいクラスローダ階層が使用されます。この新しい方式では、特定のアプリケーションについて、1 つのクラスローダーがすべての EJB モジュールを読み込み、別のクラスローダーが Web モジュールを読み込みます。これら 2 つのモジュールは親子の階層関係を持っており、JAR モジュールクラスローダが WAR モジュールクラスローダの親モジュールになります。JAR クラスローダによって読み込まれたクラスはすべて WAR モジュールでも使用またはアクセスできますが、逆 (WAR クラスローダによって読み込まれたクラスを JAR モジュールで使用またはアクセスすること) はできません。あるクラスが JAR ファイルだけでなく WAR ファイルでも必要な場合は、そのクラスを JAR モジュール内だけでパッケージ化してください。この指針に従わない場合、クラス競合が発生する可能性があります。

アプリケーションルートコンテキストとアクセス URL

Application Server 6.x と Application Server8.2 では、アプリケーションのアクセス URL (アプリケーションの Web モジュールのルートコンテキスト) に関して大きな違いがあります。`hostname` という名前のサーバー上に配備されたアプリケーションのルートコンテキストの名前が `AppName` である場合、このアプリケーションのアクセス URL は、次のように使用されるアプリケーションサーバーによって異なります。

- Application Server 6.x の場合、常に Web フロントエンドと連携して使用されるので、アプリケーションのアクセス URL は次のような形式になります (Web サーバーが標準の HTTP ポート番号 80 で設定されている場合)。

  ```
  http://<hostname>/NASApp/AppName/
  ```

- Application Server8.2 の場合、URL は次のような形式になります。

  ```
  http://<hostname>:<portnumber>/AppName/
  ```

Application Server 8.2 がデフォルトで使用する TCP ポートは、ポート番号 8080 です。

Application Server 6.x と Application Server のアクセス URL の違いはわずかなものに見えますが、絶対 URL 参照を使用するアプリケーションを移行するときに問題になる可能性があります。このような場合は、Application Server 6.x 用の Web サーバープラグインで使用される固有のマーカーが付加されないように、コードを編集して絶対 URL 参照を更新する必要があります。

フォームベース認証を使用するアプリケーション

Application Server 6.5 で開発され、フォームベース認証を使用するアプリケーションは、認証フォームまたはログインページに要求パラメータを渡すことができます。入力パラメータに基づいて認証パラメータを表示するようにログインページをカスタマイズすることもできます。

次に例を示します。

```
http://gatekeeper.uk.sun.com:8690/NASApp/test/secured/page.jsp?
arg1=test&arg2=m
```

Application Server 8.2 は、ログインページ表示中の要求パラメータの引き渡しをサポートしません。フォームベース認証を使用し、要求パラメータを渡すアプリケーションは、Application Server8.2 に移行できません。このようなアプリケーションを Application Server8.2 に移植するには、コードの大幅な変更が必要です。代わりに、要求パラメータの情報をセッションに格納して、それをログインページの表示中に取得できます。

次に、この問題を解決するコード例を示します。

変更前の 6.5 のコードは次のとおりです。

```
---------index-65.jsp -----------
<%@page contentType="text/html"%>
<html>
<head><title>JSP Page</title></head>
<body>
go to the <a href="secured/page.htm">secured a rea</a>
</body>
</html>
----------login-65.jsp--------------
<%@page contentType="text/html"%>
<html>
<head> </head>
<body>
<!-- Print login form -->
<h3>Parameters</h3><br>
out.println("arg1 is " + request.getParameter("arg1"));
out.println("arg2 is " + request.getParameter("arg2"));
```

```
</body>
</html>
```

変更後の Application Server8.2 のコードは次のとおりです。

```
---------index-81.jsp -----------
<%@page contentType="text/html"%>
<html>
<head><title>JSP Page</title></head>
<body>
<%session.setAttribute("arg1","test"); %>
<%session.setAttribute("arg2","me"); %>
go to the <a href="secured/page.htm">secured area</a>
</body>
</html>
```

`index-81.jsp` は、セッションの要求パラメータの格納方法を示しています。

```
----------login-81.jsp--------------
<%@page contentType="text/html"%>
<html>
<head> </head>
<body>
<!-- Print login form -->
<h3>Parameters</h3><br>
<!--retrieving the parameters from the session -->
out.println("arg1 is"+(String)session.getAttribute("arg1"));
out.println("arg2 is" + (String)session.getAttribute("arg2"));
</body>
</html>
```

Application Server 6.x からの固有の拡張の移行

Application Server 6.x 環境独自の多数のクラスがアプリケーション内で使用されている場合があります。Application Server 6.x で使用される固有のパッケージの一部を次に示します。

- `com.iplanet.server.servlet.extension`
- `com.kivasoft.dlm`
- `com.iplanetiplanet.server.jdbc`
- `com.kivasoft.util`
- `com.netscape.server.servlet.extension`
- `com.kivasoft`
- `com.netscape.server`

これらの API は、Application Server8.2 ではサポートされません。前述のパッケージに属するクラスを使用するアプリケーションは、標準の J2EE API を使用するように作

成し直す必要があります。また、カスタム JSP タグや UIF フレームワークを使用するアプリケーションも、標準の J2EE API を使用するように作成し直す必要があります。

Application Server 6.x からの UIF の移行

Application Server 8.2 は、アプリケーションの UIF (Unified Integration Framework) API をサポートしません。代わりに、アプリケーションを統合する JCA (J2EE Connector Adapter) の使用をサポートします。しかし、Application Server 6.5 で開発されたアプリケーションには UIF が使用されています。このようなアプリケーションを Application Server8.2 に配備するには、UIF を J2EE コネクタアーキテクチャーに移行する必要があります。この節では、UIF を使用してアプリケーションを Application Server に移行するための必要条件と手順について説明します。

アプリケーションを移行する前に、Application Server 6.5 に UIF 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るかどうかをチェックするには、次のいずれかの方法に従います。

レジストリファイルのチェック

UIF は、アプリケーションサーバーの拡張セットとしてインストールされます。これらは、インストール時にアプリケーションサーバーのレジストリに登録されます。UIF がインストールされているかどうかをチェックするには、レジストリ内で次の文字列を検索します。

拡張名セット:

- Extension DataObjectExt-cDataObject
- Extension RepositoryExt-cLDAPRepository
- Extension MetadataService-cMetadataService
- Extension RepoValidator-cRepoValidator
- Extension BSPRuntime-cBSPRuntime
- Extension BSPErrorLogExt-cErrorLogMgr
- Extension BSPUserMap-cBSPUserMap

Solaris オペレーティング環境のレジストリファイルは、次の場所にあります。

AS_HOME/AS/registry/reg.dat

インストールディレクトリの UIF バイナリのチェック

UIF インストーラは、アプリケーションサーバーインストールに特定のバイナリファイルをコピーします。次に示すファイルが見つかった場合は、UIF がインストールされています。

Solaris および Windows でのファイルの場所は次のとおりです。

AS_HOME/AS/APPS/bin

Solaris で検索するファイルのリスト:

- libcBSPRlop.so
- libcBSPRuntime.so
- libcBSPUserMap.so
- libcDataObject.so
- libcErrorLogMgr.so
- libcLDAPRepository.so
- libcMetadataService.so
- libcRepoValidator.so
- libjx2cBSPRuntime.so
- libjx2cDataObject.so
- libjx2cLDAPRepository.so
- libjx2cMetadataService.so

Windows で検索するファイルのリスト:

- cBSPRlop.dll
- cBSPRuntime.dll
- cBSPUserMap.dll
- cDataObject.dll
- ErrorLogMgr.dll
- cLDAPRepository.dll
- cMetadataService.dll
- cRepoValidator.dll
- jx2cBSPRuntime.dll
- jx2cDataObject.dll
- jx2cLDAPRepository.dll
- jx2cMetadataService.dll

UIF を Application Server8.2 に移行する前に、アプリケーション内で UIF API が使用されていることを確認してください。使用されているかどうかは、次のようにして確認します。

- Java ソースに netscape.bsp というパッケージ名があるかどうかを確認します。
- ソースに access_cBSPRuntime.getcBSPRuntime というメソッド名があるかどうかを確認します。UIF ランタイムを取得するときに、このメソッドを呼び出してください。

Application Server8.2 への UIF の移行については、appserver-migration@sun.com に問い合わせてください。

Application Server 6.x からの JDBC コードの移行

JDBC API を使用したデータベースアクセスには、次の 2 つの方法があります。

- `DriverManager` インタフェース (JDBC 1.0 API) を介した接続の確立

  特定のドライバを読み込み、接続 URL を提供します。この方法は、ほかのアプリケーションサーバー (IBM の WebSphere 4.0 など) で使用されます。

- JDBC 2.0 データソースの使用

  `DataSource` インタフェース (JDBC 2.0 API) は、設定可能な接続プールを介して使用できます。J2EE 1.2 によれば、データソースは JNDI ネーミングサービスを介してアクセスされます。

注 – Application Server8.2 は、Application Server 6.x に含まれている Native Type 2 JDBC ドライバをサポートしません。Type 2 のドライバを使用して他社製の JDBC ドライバにアクセスするコードは、手動で移行する必要があります。

DriverManager インタフェースを介した接続の確立

このデータベースアクセス方法は古くてあまり効率がよくないため推奨されていませんが、まだこの方法を使用しているアプリケーションが存在する可能性があります。

その場合、アクセスコードは次のようになります。

```
public static final String driver = "oracle.jdbc.driver.OracleDriver";
public static final String url =
  "jdbc:oracle:thin:tmb_user/tmb_user@iben:1521:tmbank";
Class.forName(driver).newInstance();
Properties props = new Properties();
props.setProperty("user", "tmb_user");
props.setProperty("password", "tmb_user");
Connection conn = DriverManager.getConnection(url, props);
```

このコードは、Application Server 6.x から Application Server8.2 に完全に移植できます。ただし、Application Server8.2 が適切な JDBC ドライバを読み込むのに必要なクラスを特定できることが条件です。必要なクラスが Application Server に配備されたアプリケーションにアクセスできるようにするには、Application Server のインストールディレクトリの `/lib` ディレクトリにドライバ実装用のアーカイブ (JAR または ZIP) を置きます。

管理コンソールの GUI を使用してドライバのパスを設定することにより、*CLASSPATH* を変更します。

- サーバーインスタンス server1 をクリックします。
- 右側のペインの「JVM 設定」タブをクリックします。
- 「パス設定」オプションをクリックし、「クラスパスのサフィックス」テキスト入力ボックスにパスを追加します。
- 設定変更を行ったら、「保存」をクリックします。
- 新しい設定を適用します。
- 設定ファイル server.xml を変更するためにサーバーを再起動します。

JDBC 2.0 データソースの使用

データベースへのアクセスに JDBC 2.0 データソースを使用すると、パフォーマンス上の利点 (透過的な接続プーリングなど) が得られるとともに、コードや実装が単純化することで生産性が向上し、コードの移植性が高まります。

Application Server 6.x アプリケーション上に xyz と言う名前のデータソースがあり、JNDI 検索コードに影響を与えたくない場合は、Application Server8.2 のために作成するデータソースの名前に必ず「JDBC」を付けてください。次に例を示します。jdbc/xyz。

JDBC データソースの設定については、『Sun Java System Application Server Enterprise Edition 8.2 管理ガイド』の第 3 章「JDBC リソース」を参照してください。

JNDI 経由でデータソースを検索して接続を取得する

データソースから接続を取得するには、次の手順に従います。

▼ データソースに接続する

1. **初期 JNDI コンテキストを取得します。**

   異なる環境間での移植性を保証するには、(サーブレット、JSP ページ、または EJB 内にある) InitialContext オブジェクトを取得するためのコードを次のようにします。

   ```
   InitialContext ctx = new InitialContext();
   ```

2. **JNDI 検索を使用してデータソースへの参照を取得します。**

   JNDI コンテキストにバインドされたデータソースへの参照を取得するには、初期コンテキストオブジェクトからデータソースの JNDI 名を検索します。この方法で検索されたオブジェクトを次のように DataSource タイプオブジェクトとしてキャストします。

   ```
   ds = (DataSource)ctx.lookup(JndiDataSourceName);
   ```

3 データソースへの参照を使用して接続を取得します。

この操作には、次のようなコード行が必要です。

```
conn = ds.getConnection();
```

Application Server 6.x と Application Server は、どちらもこれらの方法に従ってデータソースから接続を取得します。

Application Server 6.x からのリッチクライアントの移行

この節では、iPlanet Application Server 6.x で開発された RMI/IIOP クライアントや ACC クライアントを Application Server8.2 に移行する手順について説明します。

Application Server 6.x でのクライアントの認証

Application Server 6.x は、アプリケーションがユーザーからユーザー名やパスワードなどの認証データを収集できるようにするクライアント側のコールバック機構を備えています。iPlanet CORBA インフラストラクチャーによって収集された認証データは、IIOP 経由で Application Server に伝達されます。

RMI/IIOP の ORB として ORBIX 2000 を使用している場合は、移植性のあるインターセプタが要求/応答シーケンス内の段階を定義するフック (中断ポイント) を提供することによってセキュリティーを実装します。

Sun Java System Application Server 8.2 でのクライアントの認証

認証は、JAAS (Java Authorization and Authentication System API) に基づいて行われます。クライアントが `CallbackHandler` を提供しない場合、ACC は `LoginModule` と呼ばれるデフォルトの `CallbackHandler` を使用して認証データを取得します。

認証に JAAS を使用する手順の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 管理ガイド』の第 9 章「セキュリティーの設定」を参照してください。

Application Server 6.x および Sun Java System Application Server 8.2 での ACC の使用

Application Server 6.x では、独立した `appclient` スクリプトは提供されません。クラスパスに `iascleint.jar` ファイルではなく `iasacc.jar` ファイルを配置する必要がありま

す。6.x で ACC を使用してアプリケーションクライアントをパッケージ化する唯一の利点は、クライアントアプリケーションに指定された JNDI 名が EJB の絶対 JNDI 名に間接的にマップされる点です。

Application Server 6.x アプリケーションでは、スタンドアロンクライアントは JNDI 検索で EJB の絶対名を使用します。つまり、ACC 外では次のような方法で JNDI が検索されます。

```
initial.lookup(“ejb/ejb-name”);
initial.lookup(“ejb/module-name/ejb-name”);
```

Application Server 6.5 SP3 を使用してアプリケーションを開発した場合は、絶対参照を使用して検索を行うときにプレフィックス `java:comp/env/ejb/` を使用しました。

```
initial.lookup("java:comp/env/ejb/ejb-name");
```

Sun Java System Application Server 8.2 では、EJB の `jndi-name` に対して JNDI 検索を行います。EJB の絶対名を使用しないでください。また、Sun Java System Application Server 8.2 ではプレフィックス `java:comp/env/ejb` はサポートされません。クラスパス内の `iasclient.jar`、`iasacc.jar`、または `javax.jar` の各 JAR ファイルを `appserv-ext.jar` に置き換えてください。

Sun Java System Application Server 8.2 でアプリケーションがロードバランス機能を提供する場合は、クライアント側のコンテキストファクトリである `S1ASCTXFactory` の形式でのみロードバランス機能がサポートされます。このため、`com.sun.appserv.iiop.loadbalancingpolicy` システムプロパティーを次のように設定することにより、クラスタ内の代替のホストとポートを指定します。

```
com.sun.appserv.iiop.loadbalancingpolicy=

roundrobin,host1:port1,host2:port2,...,
```

このプロパティーは、ORB をラウンドロビンさせるためのホストとポートの組み合わせのリストを提供します。これらのホスト名を複数の IP アドレスにマップすることもできます。このプロパティーをシステムプロパティーとして `org.omg.CORBA.ORBInitialHost` および `org.omg.CORBA.ORBInitialPort` とともに使用した場合、ラウンドロビンアルゴリズムは提供されたすべての値をラウンドロビンします。しかし、コード内の環境オブジェクトにホスト名とポート番号を指定した場合は、その値がシステムプロパティーの設定値をオーバーライドします。

Application Server 6.5 でクライアントが接続するプロバイダ URL は、CORBA Executive Engine (CXS エンジン) の IIOP ホストおよびポートになります。Sun Java System Application Server 8.2 の場合は、クライアントがインスタンスの IIOP リスナーホストおよびポート番号を指定する必要があります。Sun Java System Application Server 8.2 には、独立した CXS エンジンは存在しません。

Sun Java System Application Server 8.2 のデフォルトの IIOP ポートは 3700 です。実際の IIOP ポートの値は、`domain.xml` 設定ファイルに指定されています。

ACC クライアントのロードバランス機能とフェイルオーバー機能 (Enterprise Edition)

Sun ONE Application Server 6.5 では、CXS エンジンが登録されている Java エンジンの数に応じてロードバランス機能を暗黙的に処理します。Application Server 8.2 Enterprise Edition でこの機能を使用するには、クライアントの明示的な設定の詳細が必要です。

配備記述子を 6.x から 8.2 に移行したら、ACC クライアントのフェイルオーバー機能を有効にするため、`sun-acc.xml` ファイルに設定の詳細を指定します。配備記述子の移行については、72 ページの「Application Server 6.x からの配備記述子の移行」を参照してください。

可用性の高い ACC クライアントを提供するには、`sun-acc.xml` ファイルにロードバランスプロパティーを定義します。これらのプロパティーは、`sun-acc.xml` ファイル内のプロパティー要素として定義されます。

- `com.sun.appserv.iiop.endpoints`

  このプロパティーは、1 つ以上の IIOP エンドポイントのリストを定義します。エンドポイントは *host*:*port* として指定されます。host は、Application Server 8.2 が実行されているシステムの名前または IP アドレスです。port は、サーバーが IIOP 要求を待機する IIOP ポートです。

- `com.sun.appserv.iiop.loadbalancingpolicy`

  エンドポイントプロパティーが指定された場合は、このプロパティーを使用してロードバランスのポリシーを指定します。このプロパティーの値は、InitialContext ベースである必要があります。

次に例を示します。

```
<client-container>
   <target-server name="qasol-e1" address="qasol-e1" port="3700">
   <property name="com.sun.appserv.iiop.loadbalancingpolicy"
             value="ic-based" />
   <property name="com.sun.appserv.iiop.endpoints"
             value="qasol-e1:3700,qasol-e1:3800" />
</client-container>
```

RMI/IIOP パス上で ACC クライアントをフェイルオーバーするには、RMI/IIOP 要求を継続できるクラスタ内のすべてのエンドポイントに関する情報が使用可能である必要があります。IIOP エンドポイントは、`domain.xml` ファイルで定義されている必要があります。`availability-service` 要素の下にある `iiop-cluster` 要素が IIOP エンドポイントを定義します。

詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 5 章「HTTP 負荷分散の設定」を参照してください。

Application Server 6.x からの HTTP フェイルオーバーをサポートするアプリケーションの移行

Application Server, Enterprise Edition 8.2 は、ロードバランスと HTTP セッション持続をサポートします。ロードバランスの主な目標は、作業負荷を複数のサーバーインスタンスに分散することによって、システムの全体的なスループットを向上させることです。

HTTP セッションフェイルオーバーの設定については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「HTTP セッションフェイルオーバー」を参照してください。

6.x の HTTP アプリケーションを Application Server 8.2 EE 環境に移行してロードバランス機能を有効にするには、次の手順を実行します。アプリケーション内のコードを変更する必要はありません。

▼ 移行してロードバランスを有効にする

1. 少なくとも **2** つのアプリケーションサーバーインスタンスが作成および設定されていることを確認します。

2. `ias-web-app.xml` の名前を `sun-web.xml` に変更します。

   配備記述子の移行の詳細については、72 ページの「Application Server 6.x からの配備記述子の移行」を参照してください。

3. **<DOCTYPE** 定義を次のコードで更新します。

   ```
   <!DOCTYPE web-app PUBLIC
   '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
   'http://java.sun.com/j2ee/dtds/web-app_2_3-1.dtd'>
   ```

4. **Sun ONE Application Server 6.5** では、**HTTP** アプリケーションのフェイルオーバーは **Dsync** 機構に基づいて行われていました。**HTTP** フェイルオーバーの設定は `ias-web-app.xml` ファイルで行われていました。

   `<servlet-info>` 要素の下に定義された `<server-info>` 要素は、サーブレットの保存先となるサーバーが有効になっているかどうかを指定します。

   `<session-info>` 要素は、次の情報を定義します。

   - `dsync-type`: この要素は、値として `dsync-distributed` または `dsync-local` を取ります。

     `dsync-distributed` は、セッションが分散され、すべての設定済みサーバーで使用可能であることを示します。

dsync-local は、セッションがその作成元のサーバー上でのみ使用可能であることを示します。

- impl: この要素は、値として distributed または lite を取ります。

  distributed は、セッションが分散されていることを示します。

  lite は、セッションがその作成元である Java エンジンに対してローカルであることを示します。この値が設定されている場合、dsync-type の設定値は無視されます。

  Sun Java System Application Server 8.2 で HTTP ルート上のアプリケーションのフェイルオーバーを有効にするには、Sun 固有の Web アプリケーション配備記述子ファイルである sun-web.xml に次のプロパティーを定義します。

- persistence-store - このプロパティーは、値として memory、file、または ha を取ります。ただし、6.5 ではメモリーベースの持続性ストアのみがサポートされていました。
- persistence-scope - 持続性のスコープを定義します。
  - session - すべてのセッションで、セッション情報が保存されます。
  - modified-session - 変更されたセッションデータのみが格納されます。
  - modified-attribute - 変更された属性データのみが格納されます。6.5 では、modified-attribute スコープのみがサポートされていました。

  persistenceFrequency - 頻度を Web メソッド単位または時間ベースに設定できます。6.5 では、web-method のみがサポートされていました。

  - web-method - セッション状態は、各 Web 要求の終了時に、クライアントに応答を返信する前に格納されます。このモードでは、障害発生時にセッション状態を完全に更新するための最良の保証が得られます。
  - time-based - セッション状態は、指定された頻度でバックグラウンドで格納されます。このモードでは、セッション状態が必ずしも完全に更新される保証は得られません。ただし、各要求後に状態が格納されないので、パフォーマンスが大幅に向上します。

    sun-web.xml ファイルの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE
sun-web-app PUBLIC '-//Sun Microsystems, Inc.//
            DTD Sun ONE Application Server 7.1 Servlet 2.3//EN'
"http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-1.dtd'>
  <sun-web-app>
    <session-config>
      <seession-manager>
        <manager-properties>
          <property name="persistence-type" value "ha'>
          <property name="persistenceFrequency" value ="web-based">
        </manager-properties>
```

```
        <store-properties>
          <property name="persistenceScope" value="session">
        </store-properties>
      </session-manager>
    </session-config>
</sun-web-app>
```

sun-web.xml 設定ファイルの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide』の「The sun-web.xml File」を参照してください。

5 **Sun Java System Application Server 8.2 で HTTP 要求の負荷を分散し、障害発生時に要求をクラスタ内の使用可能なサーバーインスタンスに継続するには、ロードバランサプラグインをインストールして設定する必要があります。**

ロードバランサの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 5 章「HTTP 負荷分散の設定」を参照してください。

6 load-balancer.xml ファイルで、web-module enabled 要素が **true** に設定されていることを確認します。

```
<loadbalancer>
  <cluster name=cluster1>
  ...
    <web-module context-root="abc" enabled=true>
  </cluster>
  <property name="https-routing" value="true"/>
</loadbalancer>
```

enabled=true は、要求の負荷分散先となる Web モジュールがアクティブである (有効になっている) ことを示します。

7 https-routing プロパティーを定義し、その値を true に設定します。

load-balancer.xml ファイルの編集の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 5 章「HTTP 負荷分散の設定」を参照してください。

ロードバランスに参加しているすべてのサーバーインスタンスにアプリケーションを配備します。

Application Server 7 から Application Server 8.2 へのアプリケーションの移行

7 PE/SE から 8.2 EE へのリッチクライアントの移行

Application Server 7 PE/SE に配備されたリッチクライアントを Application Server 8.2 に移行するのは比較的簡単です。Application Server 7 で使用していた配備記述子は、Application Server 8.2 でもそのまま使用できます。ただし、クライアントアプリケーションでロードバランス機能とフェイルオーバー機能を有効にする場合は、配備記述子でロードバランス機能とフェイルオーバー機能を設定する必要があります。

▼ リッチクライアントを移行する

1 以前にインストールされたコンポーネントを特定します。

2 `asadmin` コマンドまたはディレクトリの一覧表示を使用して、サーバーインスタンスを見つけます。

`asadmin` コマンドを使用する場合は、管理インスタンスが実行されている必要があります。一方、ディレクトリの一覧表示を使用してインスタンスを特定する場合は、管理インスタンスが実行されている必要はありません。

3 **RMI/IIOP** フェイルオーバー機能を有効にするため、`server.xml` ファイルの `jvm-config` 要素の下に次の **jvm-option** を追加します。

```
<jvm-config java-home=path...server-classpath=path>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
com.sun.appserv.ee.iiop.EEORBInitializer
  </jvm-option>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
com.sun.appserv.ee.iiop.EEIORInterceptorInitializer
  </jvm-option>
  <jvm-option>
    Dcom.sun.CORBA.connection.ORBSocketFactoryClass=
com.sun.appserv.enterprise.iiop.EEIIOPSocketFactory
  </jvm-option>
</jvm-config>
```

4 `availability-service` 要素を更新して `availability-enabled` フラグを **true** に設定します。

```
<availability-service availability-enabled="true">
  <persistence-store>
    <property-name="store-pool-jndi-name" value="" />
```

```
    <property-name="cluster-id" value="cluster1" />
  </persistence-store>
</availability-service>
```

5 java-config 要素の下のサーバークラスパスエントリを変更して次のパスを追加します。

install_dir/SUNWhads/ 4.2.2-17/lib/hadbjdbc.jar;

install_dir/lib/appserv-rt-ee.jar

6 java-config 要素の下に次の jvm-option を追加します。

```
<jvm-option>

Dcom.sun.aas.hadbRoot=install-dir/ SUNWhadb/4.2.2-17

</jvm-option>
```

7 次の新しいロードバランスプロパティーを使用して sun-acc.xml を更新します。

```
<property-name="com.sun.appserv.iiop.loadbalancingpolicy"
                    value="ic-based" />
<property name="com.sun.appserv.iiop.endpoints" value=<host>:<port>" />
```

7 EE から 8.2 EE へのリッチクライアントの移行

7 EE のアプリケーションを 8.2 EE に移行するには、次の手順に従います。

▼ 7 EE から 8.2 EE にリッチクライアントを移行する

1 **RMI/IIOP** フェイルオーバー機能を有効にするため、java-config 要素の下に次の jvm-option 要素を追加します。次の jvm-option 要素のクラス名は、ページ幅に合わせて **2** 行に分けて表示しています。プロジェクトに追加するときは、このように **2** 行に分けないでください。

```
<jvm-config java-home=path...server-classpath=path>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
          com.sun.appserv.ee.iiop.EEORBInitializer
  </jvm-option>
  <jvm-option>
    Dorg.omg.PortableInterceptor.ORBInitializerClass.
          com.sun.appserv.ee.iiop.EEIORInterceptorInitializer
  </jvm-option>
  <jvm-option>
    Dcom.sun.CORBA.connection.ORBSocketFactoryClass=
          com.sun.appserv.enterprise.iiop.EEIIOPSocketFactory
  </jvm-option>
</jvm-config>
```

2 **iiop-cluster** を設定するため、`server.xml` に次のエントリを追加します。

```
<iiop-cluster>
     <iiop-server-instance name=<server-name>>
         <iiop-endpoint id=orb-listener-id,
                        host=hostname,
                        port=orb-listener-port/>
    </iiop-server-instance>
</iiop-cluster>
```

3 次の新しいエントリを使用して `sun-acc.xml` を更新します。

```
<property-name=ÂÂ¨com.sun.appserv.iiop.loadbalancingpolicy"
                    value="ic-based" />
<property name="com.sun.appserv.iiop.endpoints"
          value="hostname:port" />
```

SFSB フェイルオーバーをサポートする EJB アプリケーションの移行

Application Server 7 は、ステートフルセッション Bean (SFSB) のフェイルオーバーをサポートしません。Application Server Enterprise Edition 8.2 は、HTTP および RMI/IIOP パス上のステートフルセッション Bean のフェイルオーバーをサポートします。この節では、SFSB 状態のフェイルオーバーをサポートする EJB アプリケーションを Application Server 7 SE/PE/EE から Application Server 8.2 EE に移行する手順について説明します。

7 SE/PE/EE から 8.2 EE への EJB アプリケーションの移行

ステートフルセッション Bean を使用してデータを持続する EJB アプリケーションの高可用性を実現するには、アプリケーションサーバーのクラスタごとに持続的ストアを設定し、個々のアプリケーションサーバーインスタンスの潜在的障害に対してクライアントセッションの情報を管理できるようにする必要があります。また、クラスタ内の各サーバーインスタンスの availability-enabled フラグを有効にする必要があります。

Application Server 8.2 EE は、ステートフルセッション Bean のフェイルオーバーをサポートします。Application Server 8.2 EE に配備された EJB アプリケーションでこの機能を有効にするには、次の手順に従います。

以前にリリースされた Sun の Application Server からエンティティー Bean を移行するには、81 ページの「エンティティー Bean」で説明した手順に従います。

SFSB フェイルオーバーは、同じアプリケーションサーバープロセスで実行されているアプリケーション内の EJB、サーブレット、または Java Server Pages から SFSB にアクセスする場合にサポートされます。SFSB には、ローカルインタフェースまたはリモートインタフェース経由でアクセスできます。

SFSB 状態のフェイルオーバーを利用するために、コードを編集する必要はありません。ただし、SFSB のチェックポイント化に必要なすべての設定パラメータを Sun 固有の配備記述子 (`sun-ejb-jar.xml`) またはサーバー設定ファイルに指定する必要があります。

SFSB フェイルオーバーの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「ステートフルセッション Bean のフェイルオーバー」を参照してください。

索引