WebOTX 運用編(アプリケーション配備)

WebOTX 運用編

バージョン: 7.1

版数: 第 5 版

リリース: 2011 年 2 月

目次

1 はじめに

本書は WebOTX 実行環境を運用するための運用操作法について概要や具体的な設定項目や設定方法について記載しています。

対象読者

このマニュアルは WebOTX Application Server Web Edition、Standard-J Edition、Standard Edition、Enterprise Edition を使って運用環境を構築するシステムエンジニア、日々の運用を行うオペレータを対象としています。

表記について

パス名表記

本書ではパス名の表記については特に OS を限定しない限りセパレータはスラッシュ'/'で統一しています。Windows 環境においては'¥'に置き換えてください。

環境変数表記

インストールディレクトリやドメインルートディレクトリなど環境によって値の異なるものについては環境変数を用いて表します。

${*env*} または $(*env*)で表しています。

例)
${AS_INSTALL} : インストールディレクトリ
${INSTANCE_ROOT}: ドメインルートディレクトリ

コマンド操作について

本書中では運用操作に用いるコマンドの詳細についての説明は省略しています。

コマンドの詳細は「運用管理コマンド」、「運用管理コマンドリファレンス」を参照してください。

2 概要

2.1 アプリケーションの種類

WebOTX Application Server は以下に示す 8 種類のアプリケーションをサポートします。サポートするアプリケーションの種類は WebOTX Application Server のエディションにより異なります。アプリケーションは種類ごとにアーカイブの形式や拡張子が仕様で定められています。

Web アプリケーション アーカイブ (J2EE)

Web アプリケーション アーカイブは Web アプリケーションのためのコンテンツを格納します。静的な HTML を提供するだけでなく、サーブレットや JSP を利用して動的にコンテンツを生成することもできます。Web アプリケーション アーカイブのファイル名の拡張子は「.war」です。WebOTX Application Server Web Edition は、Web アプリケーション アーカイブのみサポートします。

EJB JAR (J2EE)

EJB JAR には EJB (Enterprise JavaBeans)を格納します。また EJB が利用するユーティリティ クラスを含めることもできます。EJB JAR のファイル名の拡張子は「.jar」です。

J2EE アプリケーション クライアント (J2EE)

J2EE アプリケーション クライアントはクライアント コンテナ上で動作するアプリケーションです。J2EE アプリケーション クライアントは RMI-IIOP 経由で EJB 等のサーバ サイド アプリケーションにアクセスします。J2EE アプリケーション クライアントのファイル名の拡張子は「.jar」です。

リソース アダプタ アーカイブ (J2EE)

リソース アダプタ アーカイブにはリソース アダプタを格納します。リソースア ダプタとは、EJB や Web アプリケーションが外部のエンタープライズ システムにアクセスできるようにするためのコンポーネントです。リソース アダプタ アーカイブのファイル名の拡張子は「.rar」です。

SIP アプリケーション (J2EE)

SIP と呼ばれる通信制御プロトコルに特化したアプリケーションです。SIP アプリケーションのファイル名の拡張子は「.sar」です。SIP サーブレットは WebOTX SIP Application Server だけがサポートします。

エンタープライズ アプリケーション アーカイブ (J2EE)

エンタープライズ アプリケーション アーカイブは、上記の J2EE のアプリケーションを 1 つ以上格納した統合アプリケーションです。エンタープライズ アプリケーション アーカイブのファイル名の拡張子は「.ear」です。

CORBA アプリケーション

OMG が標準化を行っている CORBA という仕様に準拠したアプリケーションです。Java だけではなく、C/C++や COBOL で実装することもできます。CORBA アプリケーションの形式には R5 形式と R4 形式が存在します(「アプリケーション開発ガイド」参照)。CORBA アプリケーションのファイル名の拡張子は「.cpk」です。CORBA アプリケーションは Standard Edition と Enterprise Edition でサポート

しています。

共有コンポーネント

プロセスグループ間で共有するライブラリです。共有コンポーネントのファイル名の拡張子は「.spk」です。共有コンポーネントは Standard Edition と Enterprise Edition でサポートしています。

J2EE のアプリケーションを提供する場合、1 つの EAR でアプリケーションを組み立てることもできますし、また個別に WAR や EJB JAR などを配備してアプリケーションを組み立てても構いません。さらには各アプリケーションを異なるサーバ上に配備することでアプリケーションを分散することができます。

2.2 アプリケーションに対する運用操作

作成したアプリケーションを WebOTX Application Server 上で実行する場合、アプリケーションに対して次に示す運用操作を行います。

パッケージング

開発したアプリケーションを、各アプリケーションが準拠している仕様に基づきアーカイブします。

配備

作成したアプリケーションを WebOTX の管理下におく作業です。アプリケーションを配備するためには、ドメインが起動している必要があります。配備するアプリケーションは、あらかじめ ear や war 等の形式でパッケージ化しておきます。

開始と停止

デフォルトでは配備後のアプリケーションは開始した状態にあります。開始した状態とは、配備されたアプリケーションが動作しており、クライアントからの要求を受け付けることができる状態です。また、停止した状態とは、アプリケーションが動作していない状態のことで、クライアントからの要求は受け付けることができません。停止した状態のアプリケーションは依然として WebOTX の管理下にあります。簡単な運用操作によって、アプリケーションの開始状態と停止状態を切り替えることができます。

再配備

配備済みのアプリケーションを置換する作業を再配備と呼びます。再配備はアプリケーションの開発時に頻繁に行われる作業です。

配備解除

アプリケーションを WebOTX の管理下から除外する作業です。

3 アプリケーションの動作プロセス

WebOTX Application Server に配備するアプリケーションは、エディションにより動作するプロセスが異なります。この章では、配備の観点からアプリケーションが動作するプロセスについての解説を行います。

3.1 配備先の種類

アプリケーションが動作するプロセスは大きく分けて「エージェント プロセス」と「プロセス グループ」の二種類存在します。この節では、これらの違いについて解説します。

3.1.1 エージェントプロセス

エージェントプロセスとは、WebOTX Application Server が動作している、コアとなるプロセスのことです。Web Edition と Standard-J Edition と WebOTX SIP Application Server Standard Edition では、配備したアプリケーションは全てエージェントプロセス上で動作します。これらのエディションでは、配備時には配備先の指定は必要ありません。

3.1.2 アプリケーショングループとプロセスグループ

Standard Edition と Enterprise Edition では、配備したアプリケーションは WebOTX Application Server のコアのプロセスから分離された個別のプロセス上で動作します。このプロセスは多重化することができ、多重化したプロセス群のことを「プロセスグループ」と呼んでいます。

プロセスグループは、更に業務単位にまとめることができます。業務単位でまとめたプロセスグループ群を「アプリケーショングループ」と呼びます。

Standard Edition と Enterprise Edition で以下のアプリケーションを配備する際には、どのアプリケーショングループおよびプロセスグループに対しての配備なのかを指定する必要があります。

- Web アプリケーションアーカイブ
- EJB JAR
- エンタープライズアプリケーションアーカイブ（ただし、Web アプリケーションアーカイブか EJB JAR を含む場合）
- CORBA アプリケーション

アプリケーショングループとプロセスグループの詳細については、「運用編（TPモニタの運用操作）」を参照してください。

3.2 Webコンテナの動作モード

Standard Edition および Enterprise Edition における Web コンテナには、「マルチプロセスモード」と「シングルプロセスモード」という 2 つの動作モードが存在します。「マルチプロセスモード」とは、Web コンテナが動作するプロセスを WebOTX Application Server のコアのプロセスから分離したモードです。「シングルプロセスモード」とは WebOTX Application Server のコアのプロセス内で Web コンテナを動作させるモードで、旧互換のために存在します。

「マルチプロセスモード」の場合は、配備時にアプリケーショングループとプロセスグループの指定が必要になります。「シングルプロセスモード」の場合はこれらの指定は不要です。

「マルチプロセスモード」と「シングルプロセスモード」の選択は WebOTX Application Server のインストール

時に行い、インストール後に変更することはできません。

3.3 アプリケーショングループとプロセスグループの作成方法

この節では、各ツールを使ってアプリケーショングループとプロセスグループを作成する方法を説明します。

3.3.1 コマンドを利用した作成方法

運用管理コマンドを利用してアプリケーショングループとプロセスグループを作成する手順を説明します。なお運用管理コマンドの詳細については、運用管理コマンドリファレンスマニュアルの「otxadmin」を参照してください。

プロセスグループを作成する前に、それが属するアプリケーショングループを作成する必要があります。アプリケーショングループを作成するには、次に示すオペレーション create-apg を実行します。

```
otxadmin> create-apg APG 名
```

具体例を挙げると、「myapg」という名前のアプリケーショングループを作成するには、次のようにします。

```
otxadmin> create-apg myapg
```

次にプロセスグループを作成します。プロセスグループを作成するには、次に示すオペレーション create-pg を実行します。

```
otxadmin> create-pg --apgroup  所属 APG 名  --kind [種別] --version [バージョン] PG 名
```

具体例を挙げると、「j2eepg」という名前の、「myapg」というアプリケーショングループに属する J2EE 用のプロセスグループを作成するには、次のようにします。

```
otxadmin> create-pg --apgroup myapg --kind j2ee --version 7 j2eepg
```

別の例を挙げると、「corba」という名前の、「yourapg」というアプリケーショングループに属する、Visual C++ 2005 で実装した CORBA アプリケーションのためのプロセスグループを作成するには、次のようにします。

```
otxadmin> create-pg --apgroup yourapg --kind vc2005 --version 7 corba
```

3.3.2 配備ツールを利用した作成方法

配備ツールを利用してアプリケーショングループとプロセスグループを作成する手順を説明します。

(1) 配備ツールを起動し、画面右側のツリーのドメイン名を選択し、右クリックします。表示されたメニューの「アプリケーション／プロセスグループの作成・削除」を選択します。

(2) まずアプリケーショングループを作成します。表示されたダイアログの「アプリケーショングループ名」に作成するアプリケーショングループ名を入力し、「作成」ボタンを押下します。

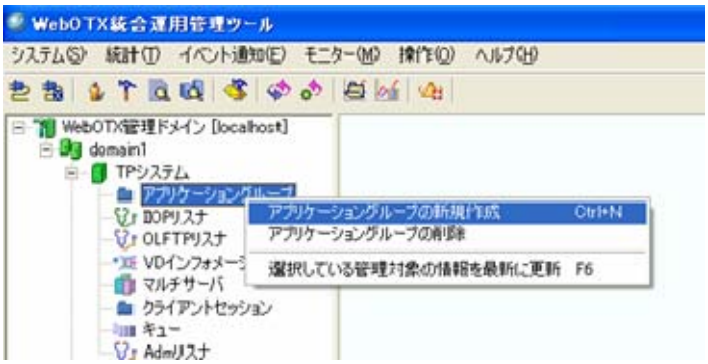
(3) 次にプロセスグループを作成します。ツリーからアプリケーショングループを選択します。そして「プロセスグループ名」に作成するプロセスグループ名を入力し、「作成」ボタンを押下します。

(4) プロセスグループの作成が完了すると、以下のような画面になります。

3.3.3 統合運用管理ツールを利用した作成方法

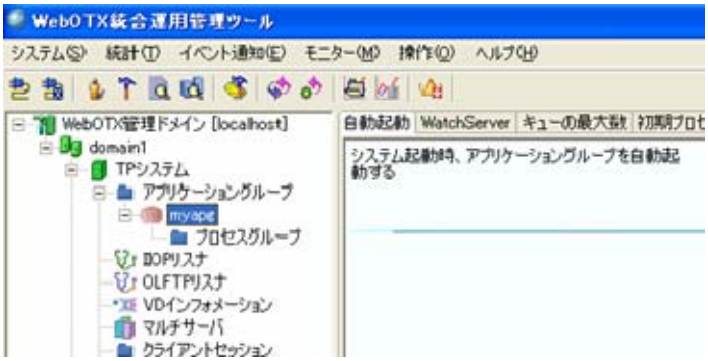
統合運用管理ツールを利用して、アプリケーショングループとプロセスグループを作成する手順を説明します。

(1) 統合運用管理ツールで WebOTX Application Server に接続します。画面右側のツリーの「アプリケーショングループ」を選択し、右クリックします。表示されたメニューから「アプリケーショングループの新規作成」を選択します。

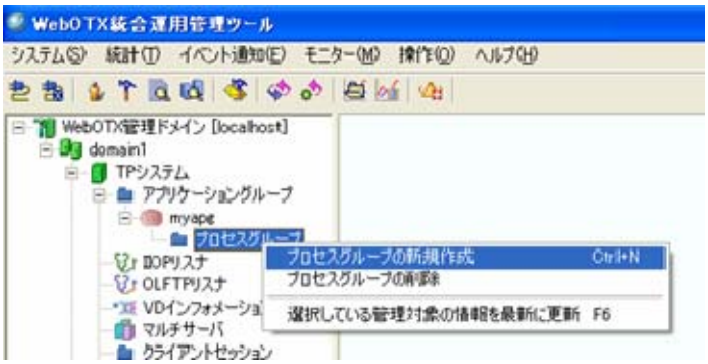


(2) アプリケーショングループを作成するためのダイアログに必要な情報を入力します。「アプリケーショングループ」に作成するアプリケーショングループの名前を入力し、「実行」ボタンを押下します。

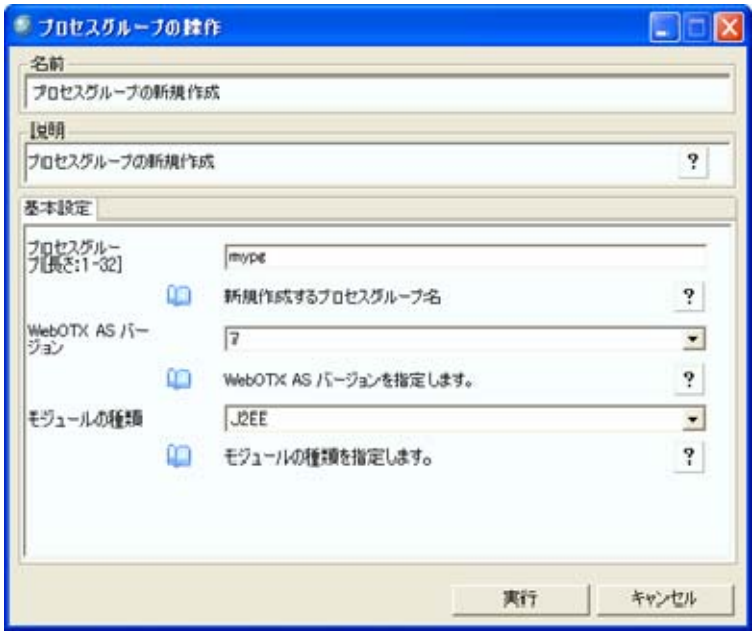
(3) アプリケーショングループの作成に成功すると、以下のように「アプリケーショングループ」の下に作成したアプリケーショングループが現れます。



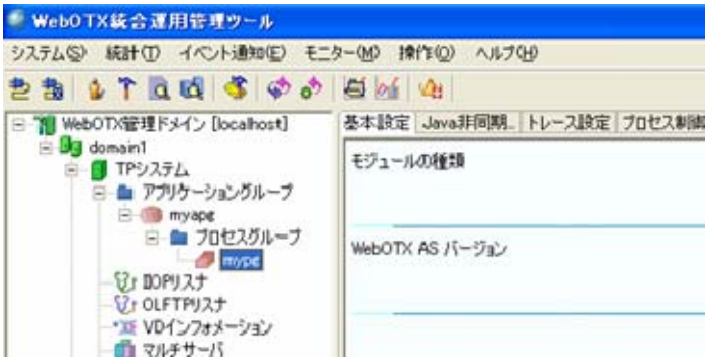
(4) 次にプロセスグループを作成します。作成したアプリケーショングループの下の「プロセスグループ」を選択し、右クリックします。表示されたメニューから「プロセスグループの新規作成」を選択します。



(5) プロセスグループを作成するためのダイアログに必要な情報を入力します。「プロセスグループ」に作成するプロセスグループ名を入力します。また「WebOTX AS バージョン」と「モジュールの種類」を適切に設定してください。必要な情報の入力の後、「実行」ボタンを押下します。



(6) プロセスグループの作成に成功すると、以下のように「プロセスグループ」の下に作成したプロセスグループグループが現れます。



3.4 注意事項

- 配備ツールは J2EE に特化したツールであるため、CORBA アプリケーション用のプロセスグループは作成できません。

4 パッケージング

作成したアプリケーションを配備する前に、アプリケーションを配備可能な形式にパッケージングします。アーカイブ内のディレクトリ構造やファイルの配置方法は、アプリケーションの種類ごとに定められています。この章では、アプリケーションのアーカイブの方法について解説します。

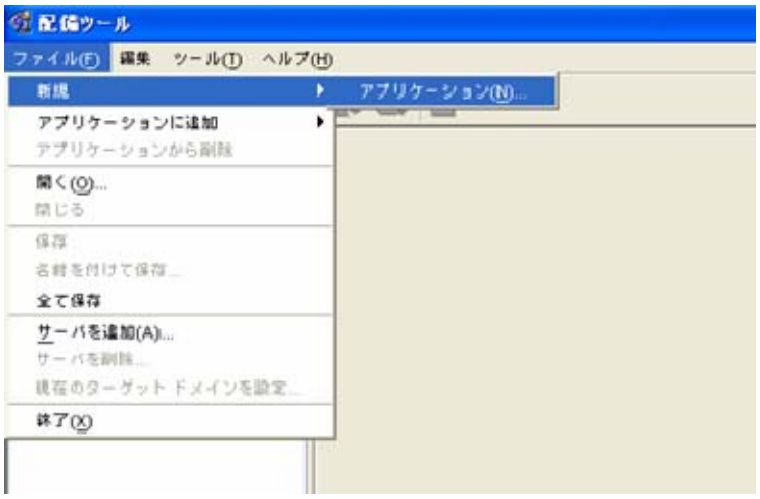
4.1 J2EEのアプリケーションのパッケージング

Web アプリケーションアーカイブや EJB JAR などの J2EE のアプリケーションを配備する際には、標準仕様(JSR)に準拠した形式でパッケージングを行う必要があります。パッケージングの形式が標準化されているため、ポータビリティの高いアプリケーションの作成が可能になっています。

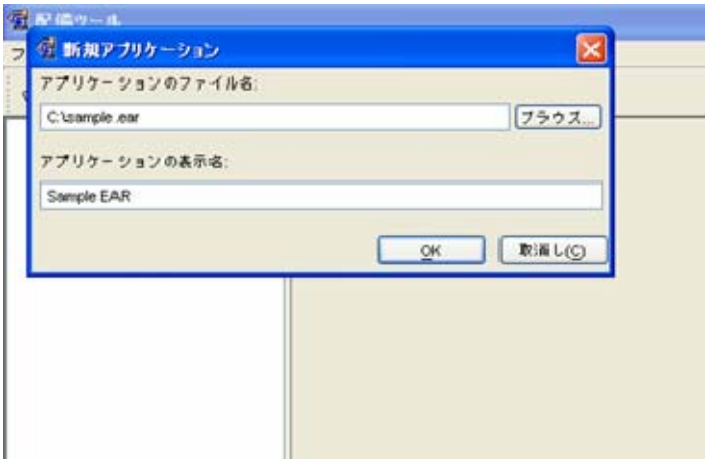
4.1.1 配備ツールを利用したパッケージング

配備ツールの主要機能は、配備記述子を編集する機能と、アプリケーションを WebOTX Application Server に配備する機能です。これらの機能に加え、配備ツールは簡易的なパッケージング機能も提供しています。配備ツールはエンタープライズアプリケーションアーカイブのパッケージングのみサポートします。以下でエンタープライズアプリケーションアーカイブのパッケージング方法を解説します。

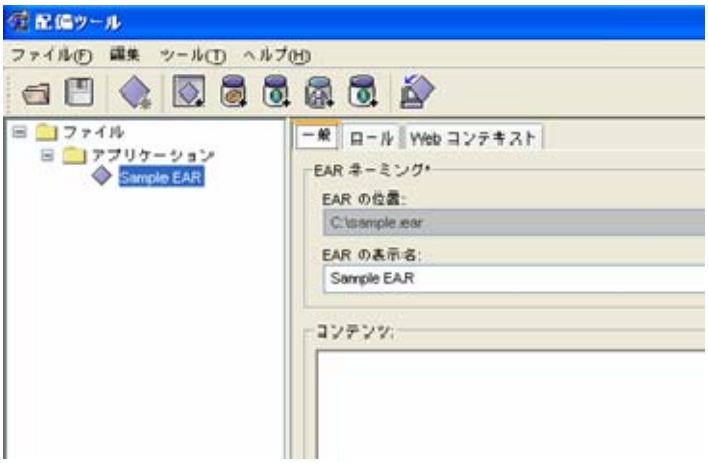
(1) 配備ツールを起動し、メニューから「ファイル」→「新規」→「アプリケーション(N)...」を選択します。



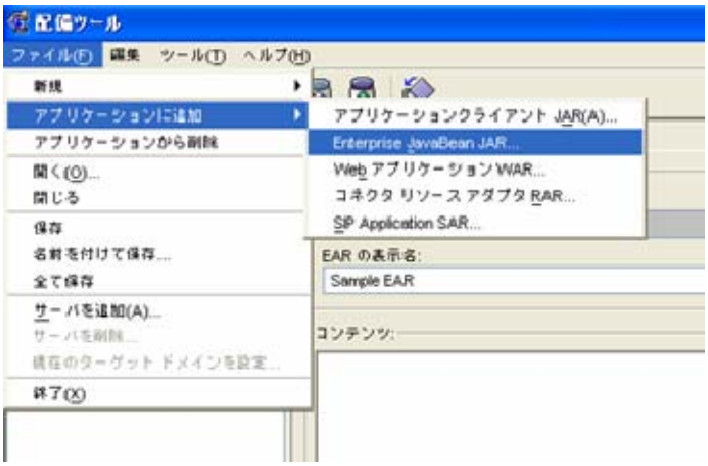
(2) 表示された「新規アプリケーション」ダイアログに、新規に作成するエンタープライズアプリケーションアーカイブのパスと表示名を入力し、「OK」ボタンを押下します。



(3) エンタープライズアプリケーションアーカイブが作成されると、以下のような画面になります。



(4) メニューから「アプリケーションに追加」を選択し、続いて表示されるサブメニューに従い、エンタープライズアプリケーションアーカイブに追加するアプリケーションを選択してください。



(5) エンタープライズアプリケーションアーカイブにアプリケーションが追加されると、次のような画面になります。

(6) アプリケーションの追加と配備記述子の編集が完了したら、メニューから「ファイル」→「保存」を選択し、エンタープライズアプリケーションアーカイブを保存します。

4.2 CORBAアプリケーションのパッケージング

J2EE の場合とは異なり、CORBA アプリケーションにはパッケージングに関する標準仕様はありません。ここで示す CORBA のパッケージングは WebOTX Application Server に固有の形式です。

CORBA アプリケーションをパッケージングするためには、最低限次の 2 つのファイルが必要です。

- CORBA アプリケーションそのものである、モジュールファイル（.jar / .dll / .so / .sl）
- インタフェースリポジトリに登録するインタフェースが定義してある、if ファイル

また、必要であれば次のファイルを準備してください。

- 各種設定情報を記述した、プロパティファイル

プロパティファイルにパッケージングコマンド(makecpk)等に渡す情報を予め準備しておくことで、作業を効率化できます。プロパティファイルは Java のクラス java.util.Properties が読み込める形式である必要があります。プロパティファイルで設定できる項目を次に示します。

プロパティ	説明
component.initfunc	コンポーネント初期化ファイル名を指定します。
component.id	コンポーネント ID を指定します。
component.useshareif	共有コンポーネントの if ファイルを利用する場合 true を利用しない場合 false を指定します(既定値:false)。
component.shareif	共有コンポーネントの if ファイルを利用する場合その if ファイル名を指定します。
component.sharedComponentList	利用する共有コンポーネントのアプリケーション名を記述します。複数ある場合はカンマ「,」で区切ります。
sharedcomponent.allModuleUse	共有コンポーネントを全てのアプリケーションから利用できるようにします。

なお、統合運用管理ツールにはパッケージングされていない CORBA アプリケーションを配備する機能がありますので、CORBA アプリケーションのパッケージングは必須ではありません。

4.2.1 CORBAアプリケーションの形式

CORBA アプリケーションの形式には R5 形式と R4 形式があります。CORBA アプリケーションの形式についての詳細は、「アプリケーション開発ガイド 7.1 WebOTX AS CORBA アプリケーション」を参照してください。

4.2.2 コマンドを利用したパッケージング

コマンドラインからCORBAアプリケーションをパッケージングするにはmakecpkコマンドを利用します。makecpkの詳細なリファレンスについては運用管理コマンドリファレンスマニュアルの「makecpk」を参照してください。

例 1: 既定値で CORBA アプリケーションをパッケージング

```
> makecpk corbaap.cpk module=corbaap.jar if=corbaap.if
```

例 2: プロパティファイルを指定しての CORBA アプリケーションをパッケージング

```
> makecpk corbaap.cpk module=corbaap.jar if=corbaap.if prop=corbaap.properties
```

例 3: オプションを指定しての CORBA アプリケーションのパッケージング

```
> makecpk corbaap.cpk module=corbaap.jar if=shareap.if id=AA initfunc=corbaap.create
useshareif=true
```

4.3 共有コンポーネントのパッケージング

共有コンポーネントは WebOTX Application Server 固有のコンポーネントです。共有コンポーネントをパッケージングするためには、最低限次のファイルが必要です。

- 共有コンポーネントそのものである、モジュールファイル（.jar / .dll / .so / .sl）

また共有コンポーネントが CORBA インタフェースを持つ場合、次のファイルも準備してください。

- インタフェースリポジトリ登録するインタフェースが定義してある、if ファイル

なお、統合運用管理ツールにはパッケージングされていない共有コンポーネントを配備する機能がありますので、共有コンポーネントのパッケージングは必須ではありません。

4.3.1 コマンドを利用したパッケージング

コマンドラインから共有コンポーネントをパッケージングするにはmakecpkコマンドを利用します。makecpkの詳細なリファレンスについては運用管理コマンドリファレンスマニュアルの「makecpk」を参照してください。

例 1: 共有コンポーネントのパッケージング

```
> makecpk shareap.spk module=shareap.jar
```

例 2: CORBA インタフェースを持つ共有コンポーネントのパッケージング

```
> makecpk shareap.spk module=shareap.jar if=shareap.if
```

4.4 注意事項

- 配備ツールは J2EE に特化したツールであるため、CORBA アプリケーションおよび共有コンポーネントのパッケージングはできません。

5 配備と再配備

配備とは、作成したアプリケーションを WebOTX Application Server の管理下におく作業です。また再配備とは、WebOTX Application Server の管理下にあるアプリケーションを、新しいアプリケーションで置換することです。この章では、配備および再配備について解説します。

5.1 アプリケーション名の規約

WebOTX Application Server がアプリケーションを管理するために、配備の際にはアプリケーションに名前をつける必要があります。アプリケーションの名前は、WebOTX Application Server のドメイン内で一意でなければなりません。

アプリケーション名は、デフォルトではアーカイブのファイル名から拡張子を取り除いたものになります。例えば、アーカイブのファイル名が「sample.war」の場合、「sample」というアプリケーション名で WebOTX Application Server に配備されます。また、配備をサポートする各ツールは、アプリケーション名を明示的に指定する方法を提供しています。

5.2 アプリケーショングループとプロセスグループ

§ 3.1.2で示したとおり、Standard EditionおよびEnterprise Editionでは、配備の際にアプリケーショングループとプロセスグループの指定が必要な場合があります。

再配備時には、アプリケーショングループとプロセスグループの指定があっても、それらは無視されます。新しいアプリケーションは、古いアプリケーションが配備されていたアプリケーショングループおよびプロセスグループに配備されます。

5.3 J2EEのアプリケーションの配備/再配備

この節では、J2EE のアプリケーションの配備/再配備について説明します。特に断りがない限り、説明は J2EE のアプリケーションの種類すべてに共通しているものとします。

5.3.1 コマンドを利用した配備（deploy）

運用管理コマンドのdeployオペレーションを利用してJ2EEのアプリケーションを配備/再配備する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照してください。

配備するには、J2EE のアプリケーションのアーカイブを指定してオペレーション deploy を実行します。再配備するには、更に--force オプションを付加します。また Standard Edition および Enterprise Edition の場合は、配備先のアプリケーショングループ名とプロセスグループ名も必要となる場合があります。

```
otxadmin> deploy [--apgroup APG 名] [--pgroup PG 名] アーカイブ名
otxadmin> deploy [--apgroup APG 名] [--pgroup PG 名] --force アーカイブ名
```

具体例を挙げると、Web Edition においてアーカイブ名が「hello.war」である Web アプリケーションを配備し、その後再配備するには、次のようにします。

```
otxadmin> deploy hello.war
otxadmin> deploy --force hello.war
```

別の例を挙げると、Enterprise Edition においてアーカイブ名が「sample.jar」である EJB JAR を配備し、その後再配備するには、次のようにします。配備先のアプリケーショングループ名を apg で、プロセスグループ名を pg としています。

```
otxadmin> deploy --apgroup apg --pgroup pg sample.jar
otxadmin> deploy --force sample.jar
```

5.3.2 コマンドを利用した配備（deploydir）

運用管理コマンドのオペレーションdeploydirを利用してJ2EEのアプリケーションを配備/再配備する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照してください。

通常の配備では、アプリケーションをパッケージングし、そのアーカイブを配備します。オペレーション deploydir を利用すると、パッケージング前のディレクトリ構造そのものを配備することができます。

配備するには、J2EE のアプリケーションが格納されているディレクトリのフルパスを指定してオペレーション deploydir を実行します。配備対象のディレクトリのフルパスは、WebOTX Application Server が動作しているホストにおけるフルパスです。アプリケーションをクライアントマシンからサーバマシンへ転送する機能はありません。再配備するには、更に--force オプションを付加します。また Standard Edition および Enterprise Edition の場合は、配備先のアプリケーショングループ名とプロセスグループ名も必要となる場合があります。

```
otxadmin> deploydir [--apgroup APG 名] [--pgroup PG 名] ディレクトリのフルパス
otxadmin> deploydir [--apgroup APG 名] [--pgroup PG 名] --force ディレクトリのフルパス
```

具体例を挙げると、Web Edition においてアプリケーションのディレクトリのフルパスが「C:¥hello」である Web アプリケーションアーカイブを配備し、その後再配備するには、次のようにします。

```
otxadmin> deploydir C:¥hello
otxadmin> deploydir --force C:¥hello
```

別の例を挙げると、Enterprise Edition において、サーバ上でのディレクトリのフルパスが「C:¥sample」である Web アプリケーションを配備し、その後再配備するには次のようにします。配備先のアプリケーショングループ名を apg で、プロセスグループ名を pg としています。

```
otxadmin> deploydir --apgroup apg --pgroup pg C:¥sample
otxadmin> deploydir --force C:¥sample
```

オペレーション deploydir は特殊なオペレーションであるため、開発時のみ利用し、本番環境では利用しないでください。オペレーション deploydir で配備できる J2EE のアプリケーションは Web アプリケーションアーカイブと EJB JAR と SIP アプリケーションに制限されています。オペレーション deploydir は配備するディレクトリの内容を書き換えるため、配備するディレクトリのバックアップを取ってから実行してください。

deploydir では、JAR ファイルが解体された状態のディレクトリ配下の中にある配備記述子 XML ファイルを直接変更できます。「オートデプロイ」の項で説明した構成情報ファイル「`domain.xml`」の同じ「`das-config`」要素内の中にある下記の動的再ロードに関する属性が有効になっていると、サーバは自動的に変更を検知し、アプリケーションを再ロードします。

属性	既定値	説明
dynamic-reload-enabled	false	true の場合、動的再ロードを起こすために、すべてのモジュールやアプリケーションのディレクトリ単位レベルで「.reload」ファイルのタイムスタンプをチェックする。
dynamic-reload-poll-interval-in-seconds	2	動的再ロードのポーリング頻度を制御する。

「dynamic-reload-enabled」の設定を、「true」に変更します。それに加えて、新しいファイルのロードや既存のファイルを再ロードさせるために、次の操作を行います。

- 配備済みアプリケーションのルートディレクトリの「.reload」という名前の空ファイルを生成する。

  モジュール（WAR、EJB JAR、アプリケーションクライアント JAR):
  ${INSTANCE_ROOT}/applications/j2ee-modules/<モジュール名>/.reload

- 変更がある度に、明示的に「.reload」ファイルのタイムスタンプを更新する。(UNIX の場合は、touch .reload を実行する)

5.3.3 配備ツールを利用した配備

ここでは、配備ツールを利用して J2EE のアプリケーションを配備する方法について説明します。

(1) アプリケーションを配備するために、メニューの「ツール(T)」 → 「配備(D)…」を選択してください。

(2) Standard Edition および Enterprise Edition の場合は、配備先のプロセスグループを指定するためのダイアログが表示されます。配備先のプロセスグループを指定して「OK」ボタンを押下すると、配備が開始します。Standard Edition でも Enterprise Edition ではない場合、このダイアログは表示されずに、すぐに配備が始まります。

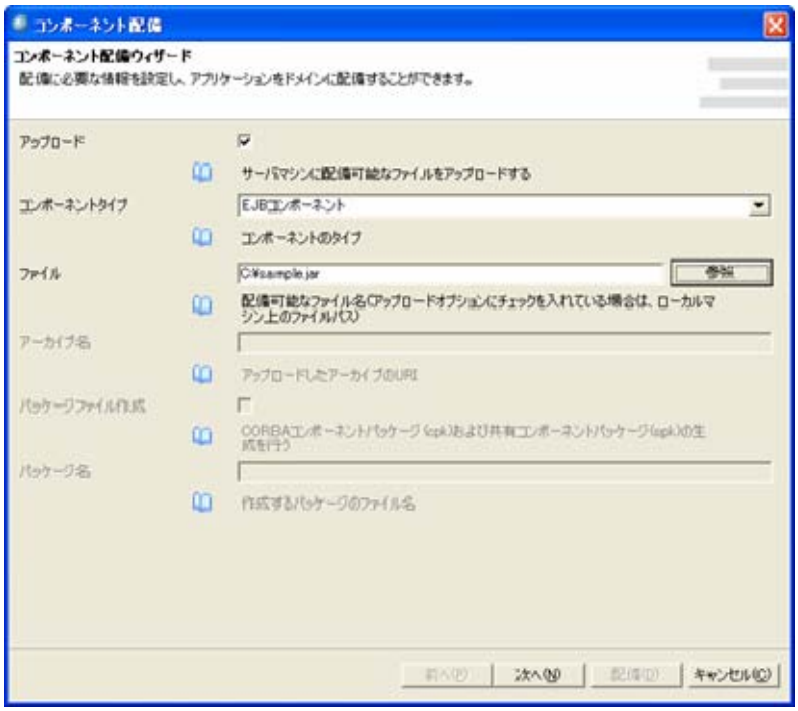
(3) 配備が正常に完了すると、以下の画面のように、「配備済みオブジェクト」のリストに配備したアプリケーションが表示されます。

5.3.4 統合運用管理ツールを利用した配備

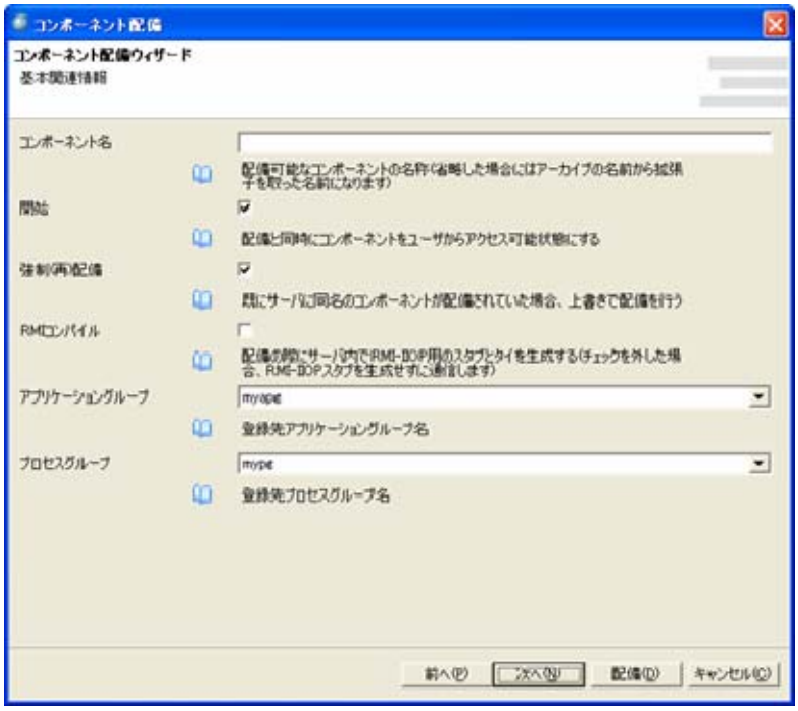
統合運用管理ツールを利用して、J2EE のアプリケーションを配備する手順を説明します。

(1) 統合運用管理ツールを起動し、ドメインに接続します。そして画面右側のツリーの「アプリケーション」を選択し、右クリックします。表示されたメニューから「コンポーネントの配備」を選択します。

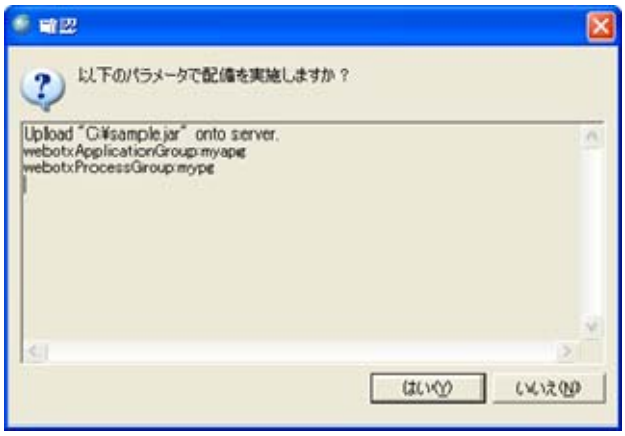
(2) コンポーネントタイプを適切に選択してください。この例では「EJB コンポーネント」を選択しています。また配備するアーカイブのパスを「ファイル」に記述します。「参照」ボタンを押下することでダイアログからパスを選択できます。入力が完了したら「次へ(N)」ボタンを押下します。



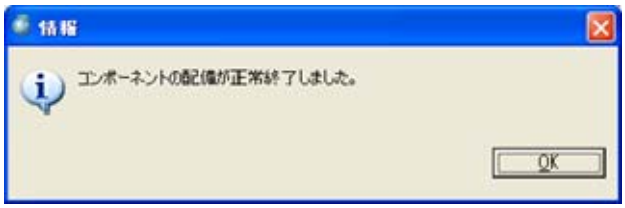
(3) Standard Edition および Enterprise Edition の場合は、配備先のアプリケーショングループとプロセスグループ指定します。そして「配備(D)」ボタンを押下します。



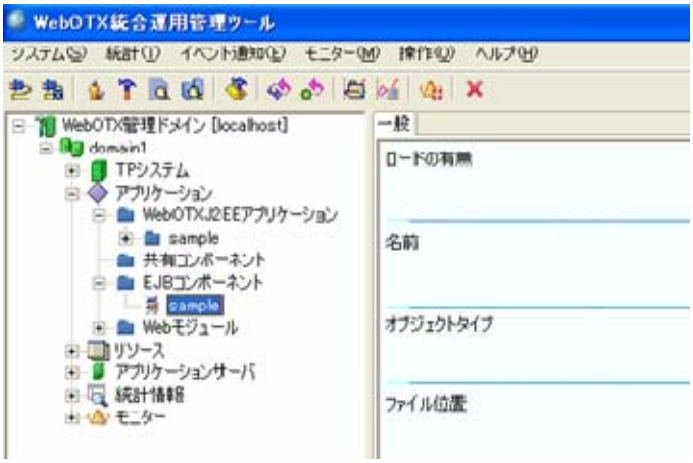
(4) 配備実行の確認のダイアログが表示されますので「はい(Y)」ボタンを押下してください。



(5) 配備の進行状況が表示された後、終了を示すダイアログが表示されます。



(6) 正常終了した場合、画面右側のツリーにアプリケーション名が表示されます。以下の画面の例では、「EJB コンポーネント」の子要素として「sample」というアプリケーションが追加されています。アプリケーションがどの要素の子要素になるのかは、アプリケーションの種類によって異なります。さらに、Standard Edition および Enterprise Edition では、「WebOTXJ2EE アプリケーション」の子要素としても追加されます。



5.3.5 オートデプロイ機能を利用した配備

オートデプロイ機能とは、アプリケーションのアーカイブファイルを次のディレクトリにコピーすることで自動的に配備を行う機能です。

```
${INSTANCE_ROOT}/autodeploy
```

例えば Windows でドメイン domain1 を利用している場合、「hello.war」というファイル名のアプリケーションを C:¥WebOTX¥domains¥domain1¥autodeploy にコピーするだけで、自動で配備を実行できます。

配備に成功した場合には、ディレクトリ autodeploy に、[アーカイブファイル名]_deployed というファイルが作成されます。また配備に失敗した場合は、[アーカイブファイル名]_deployFailed というファイルが作成されます。これらのファイルの存在を確認することで、配備が正常に終了したかどうかを調べることができます。

再配備を行う場合は、再配備するアーカイブをディレクトリ autodeploy に上書きでコピーしてください。上書きするアーカイブのタイムスタンプは、上書きされるアーカイブのタイムスタンプよりも新しいものである必要があります。古いタイムスタンプのアーカイブで上書きした場合、配備処理は実行されません。

Standard Edition および Enterprise Edition では、配備時にアプリケーショングループとプロセスグループを指定する必要があります。オートデプロイ機能で配備先のアプリケーショングループとプロセスグループを指定するには、ディレクトリ autodeploy に、プロパティファイル autodeploy.ini を作成し、内容を適切に設定してください。プロパティファイル autodeploy.ini は、java.util.Properties が読み込める形式でなければいけません。プロパティファイル autodeploy.ini はドメイン起動時に読み込まれ、その後変更するたびに再読み込みされます。プロパティファイル autodeploy.ini に記述するプロパティの詳細を以下に示します。

キー	値の説明
apg	登録先アプリケーショングループ名。
pg	登録先プロセスグループ名。

オートデプロイ機能の各種設定は、${INSTANCE_ROOT}/config/domain.xml に記述してあります。この domain.xml ファイルの要素<das-config>の属性の値を編集することで、設定を変更することができます。domain.xml の編集はドメインを停止してから行ってください。属性の詳細は以下の通りです。

属性	既定値	説明
autodeploy-enabled	false	true の場合、オートデプロイ機能が有効になります。false の場合は、オートデプロイ機能は無効になり、動作しません。
autodeploy-poll-interval-in-seconds	2	オートデプロイのためのディレクトリを監視する間隔(ポーリング間隔)です。単位は秒です。
autodeploy-dir	autodeploy	オートデプロイのためのディレクトリを設定します。このディレクトリは、絶対パスかドメインのディレクトリからの相対パスです。
autodeploy-verifier-enabled	false	true の場合、配備を実行する前に、配備記述子の妥当性検証(ベリファイ)を行います。妥当性検証に失敗した場合は、配備は実行しません。false の場合は、妥当性検証は実行しません。
autodeploy-jsp-precompilation-enabled	false	true の場合、配備中に JSP をコンパイルします。false の場合、JSP のコンパイルは行いません。この場合は JSP の初回アクセス時に動的にコンパイルを実行します。

5.4 CORBAアプリケーションの配備

ここでは CORBA アプリケーションの配備について説明します。Standard Edition および Enterprise Edition が CORBA アプリケーションをサポートしています。

5.4.1 動的配備と静的配備

CORBA アプリケーションの形式が R4 形式なのか R5 形式なのかによって、配備を行うための条件が異なります。アプリケーショングループが起動状態で配備を行うことを、動的配備と呼びます。アプリケーショングループが停止状態で配備を行うことを、静的配備と呼びます。

R4 形式の CORBA アプリケーションの場合、静的配備のみをサポートします。配備を行うにはその CORBA アプリケーションが属しているアプリケーショングループを停止してください。アプリケーショングループが起動している場合は配備できません。

R5 形式の CORBA アプリケーションの場合は、動的配備も静的配備もサポートしています。ただし、動的配備を行う場合は以下の条件を満たす必要があります。

- プロセスグループのバージョンが 6 以上である。

バージョン 5 のプロセスグループは動的配備をサポートしていませんので、アプリケーショングループを停止し

て静的配備を行ってください。

5.4.2 動的再配備と静的再配備

CORBA アプリケーションの形式が R4 形式なのか R5 形式なのかによって、再配備を行うための条件が異なります。アプリケーショングループが起動状態で再配備を行うことを、動的再配備と呼びます。アプリケーショングループが停止状態で再配備を行うことを、静的再配備と呼びます。

R4 形式の CORBA アプリケーションの場合、静的再配備のみをサポートします。再配備を行うにはその CORBA アプリケーションが属しているアプリケーショングループを停止してください。アプリケーショングループが起動している場合は再配備できません。

R5 形式の CORBA アプリケーションの場合は、動的再配備も静的再配備もサポートしています。ただし、動的再配備を行う場合は以下の条件を満たす必要があります。

- プロセスグループのバージョンが 6 以上である。
- 該当 CORBA アプリケーションが停止状態である。

バージョン 5 のプロセスグループは動的再配備をサポートしていませんので、アプリケーショングループを停止して静的再配備を行ってください。またCORBAアプリケーションが停止状態でない場合は、再配備ができません。CORBAアプリケーションを停止状態にすることで、再配備を行うことができます。CORBAアプリケーションを停止する際、実行中のオペレーションがエラーとなる可能性があります。安全に停止するためには § 7.2.2 を参照してください。

5.4.3 業務の停止を伴わないアプリケーションの置換

動的配備と動的配備解除を利用することにより、業務を停止せずにアプリケーションを置換することができます。手順を以下に示します。

(1) 新しいアプリケーションを作成する際に、モジュール名を古いアプリケーションと異なるものにします。これにより新旧アプリケーションを同時に配備することができるようになります。

(2) 新しいアプリケーションを古いアプリケーションとは異なるアプリケーション名で動的配備します。これにより新旧アプリケーションが同時に起動している状態になります。

(3) 名前サーバの登録情報を、新しいアプリケーションの情報で上書きします。これにより新規に接続するクライアントは新しいアプリケーションを参照するようになります。

(4) 古いアプリケーションを動的配備解除します。詳細は § 6.2.1を参照してください。

5.4.4 ファクトリインタフェースのオペレーション

WebOTX V6.5 より、オペレーション数削減のために、業務実行において不要な CORBA オペレーションを表示しないようにしています。具体的には、R4 形式で作成された CORBA アプリケーションにおける、自動生成されたファクトリインタフェースのオペレーションのうち、CreateServerObject2 と ReleaseServerObject2 の登録を行わないようにしています。これらは開発環境の IDL コンパイラ(woigenxx, woigenj)で作成したときに自動付加されます。WebOTX V6.4 以前と同じように表示させたい場合は、以下の方法でドメインの Java VM オプションにシステムプロパティを設定してから配備しなおす必要があります。

設定するシステムプロパティは以下の通りです。

プロパティ：com.nec.webotx.analyzeFactory

値：true/false (既定値:false　CreateServerObject2, ReleaseServerObject2 を表示しない)

システムプロパティを Java VM オプションに追加します。設定手順は以下の通りです。

- 統合運用管理ツールから行う場合

  以下のツリーの[一般]画面内の JVM オプションに「-Dcom.nec.webotx.analyzeFactory=true」を追加する
  ドメイン名
  - アプリケーションサーバ
  - JVM 構成 --> [一般]タグ

- 運用管理コマンドから行う場合

  以下のコマンドで Java VM オプションの追加を行う（--user などのオプションは省いています）
  設定
  otxadmin> create-jvm-options -- -Dcom.nec.webotx.analyzeFactory=true

  Java VM オプション確認
  otxadmin> get server.java-config.jvm-options

この設定は、ドメイン再起動以降有効になります。配備時のみ影響するものであり、配備済みの CORBA アプリケーションには影響ありません。また、R4 形式以外のアプリケーションにも影響はありません。なお、表示する設定にした場合、1 ドメイン内に登録するオペレーション数が増加します。TP システムの属性である「最大オペレーション数」を超えないように、事前の調整を行う必要があります。

5.4.5 コマンドを利用した配備

運用管理コマンドを利用してCORBAアプリケーションを配備/再配備する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照してください。

配備するには、CORBA アプリケーションのアーカイブを指定して deploy オペレーションを実行します。deploy オペレーションを実行する際には、配備先のアプリケーショングループ名とプロセスグループ名も指定する必要があります。再配備するには、更に--force オプションを付加します。

```
otxadmin> deploy --apgroup APG 名 --pgroup PG 名 アーカイブ名
otxadmin> deploy --apgroup APG 名 --pgroup PG 名 --force アーカイブ名
```

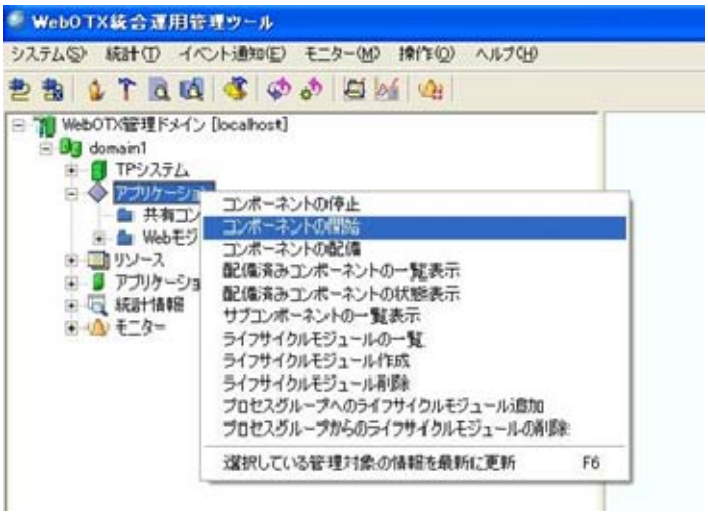
例えば、アーカイブ名が「corba.cpk」である CORBA アプリケーションを配備し、その後再配備するには、次のようにします。配備先のアプリケーショングループ名を apg で、プロセスグループ名を pg としています。

```
otxadmin> deploy --apgroup apg --pgroup pg corba.cpk
otxadmin> deploy --apgroup apg --pgroup pg --force corba.cpk
```

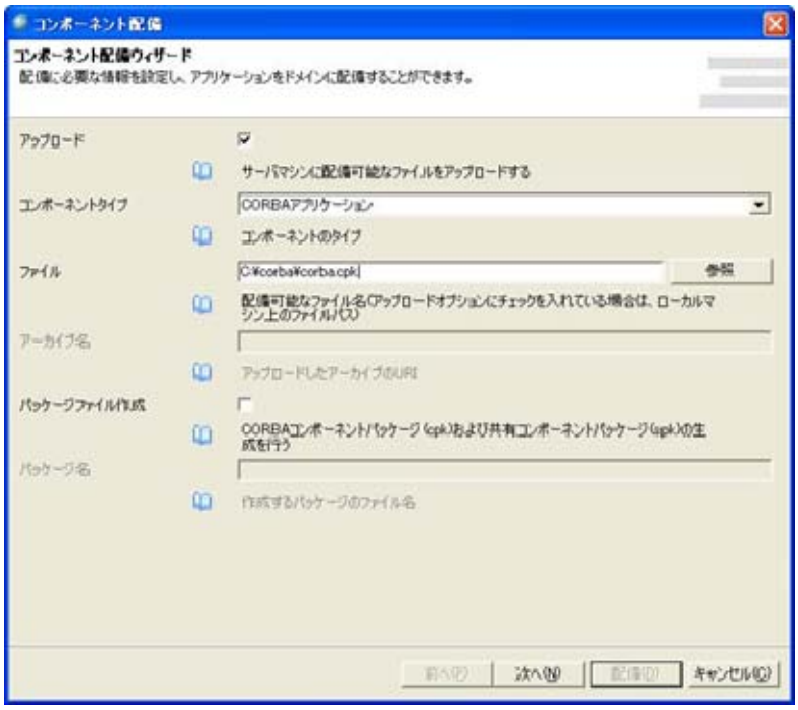
5.4.6 統合運用管理ツールを利用した配備

統合運用管理ツールを利用して、CORBAアプリケーションを配備する手順を説明します。CORBAアプリケーションの配備および再配備の条件については§5.4.1と§5.4.2を参照してください。

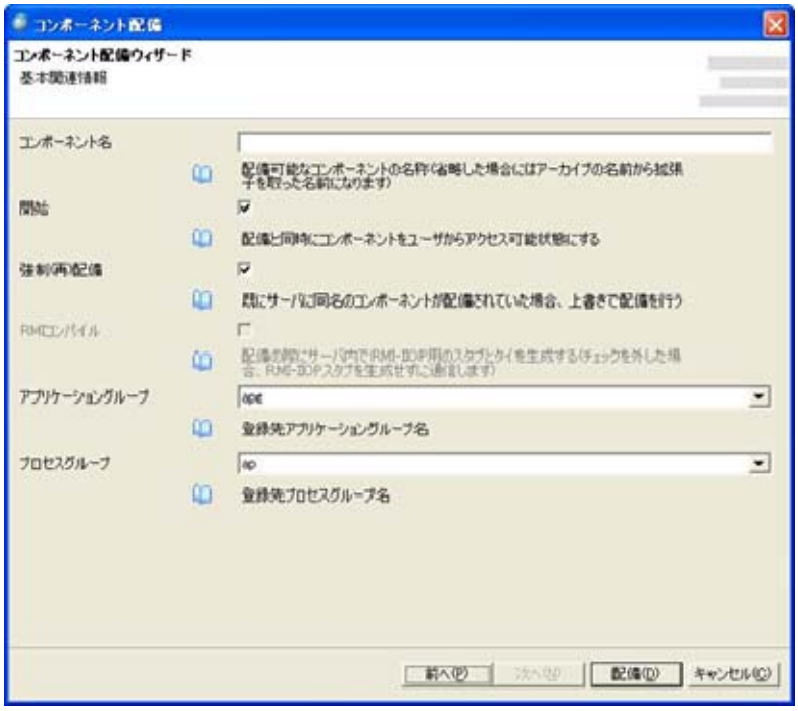
(1) 統合運用管理ツールの画面右側のツリーの「アプリケーション」を選択し、右クリックします。表示されたメニューから「コンポーネントの配備」を選択します。



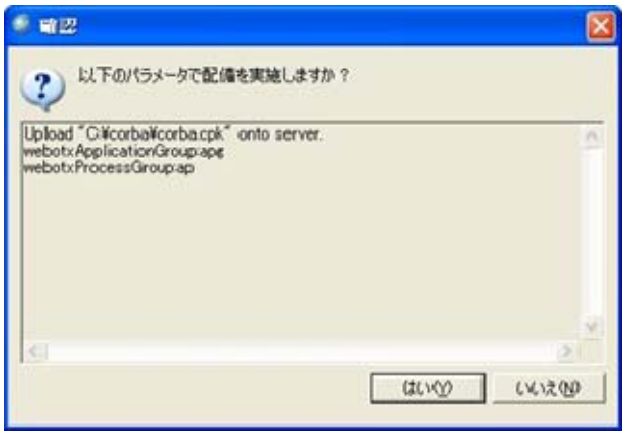
(2) 「コンポーネントタイプ」を「CORBA アプリケーション」に設定します。また「ファイル」に配備する CORBA アプリケーションのアーカイブ(.cpk)を設定します。入力が完了したら、「次へ(N)」ボタンを押下します。



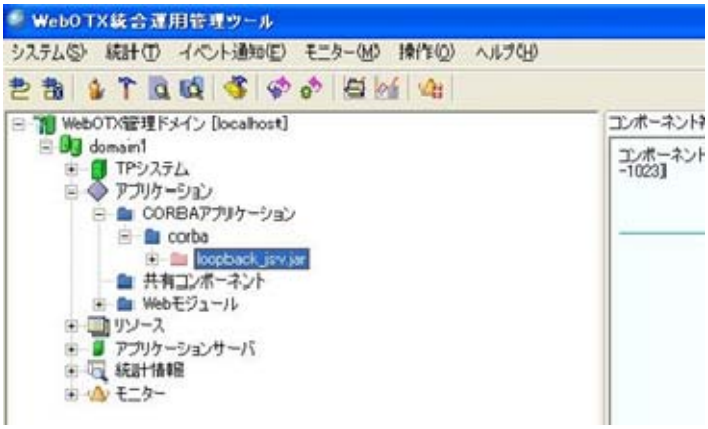
(3) 「アプリケーショングループ」と「プロセスグループ」を適切に設定してください。また再配備を行う場合は、「強制(再)配備」にチェックを入れてください。入力が完了したら「配備(D)」ボタンを押下してください。



(4) 配備実行の確認画面が出てきますので、「はい(Y)」ボタンを押下してください。



(5) 配備が正常に完了すると、以下の画面のように、画面右側のツリーの「CORBA アプリケーション」の下に配備したアプリケーション名が表示されます。



5.5 共有コンポーネントの配備

ここでは、共有コンポーネントの配備について説明します。Standard Edition および Enterprise Edition が共有コンポーネントをサポートしています。

5.5.1 コマンドを利用した配備

運用管理コマンドを利用して共有コンポーネントを配備/再配備する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照してください。

配備する場合は、共有コンポーネントのアーカイブを指定して deploy オペレーションを実行します。再配備するには、更に--force オプションを付加します。

```
otxadmin> deploy アーカイブ名
otxadmin> deploy --force アーカイブ名
```

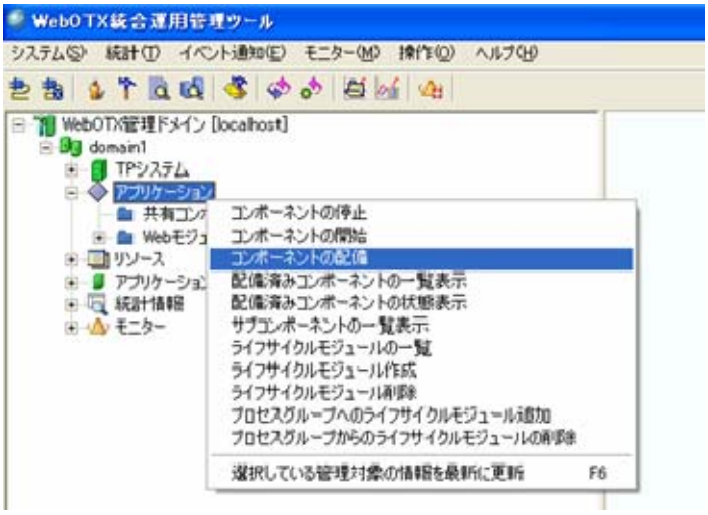
具体例を挙げると、アーカイブ名が「shared.spk」である共有コンポーネントを配備し、その後再配備するには、次のようにします。

```
otxadmin> deploy shared.spk
otxadmin> deploy --force shared.spk
```

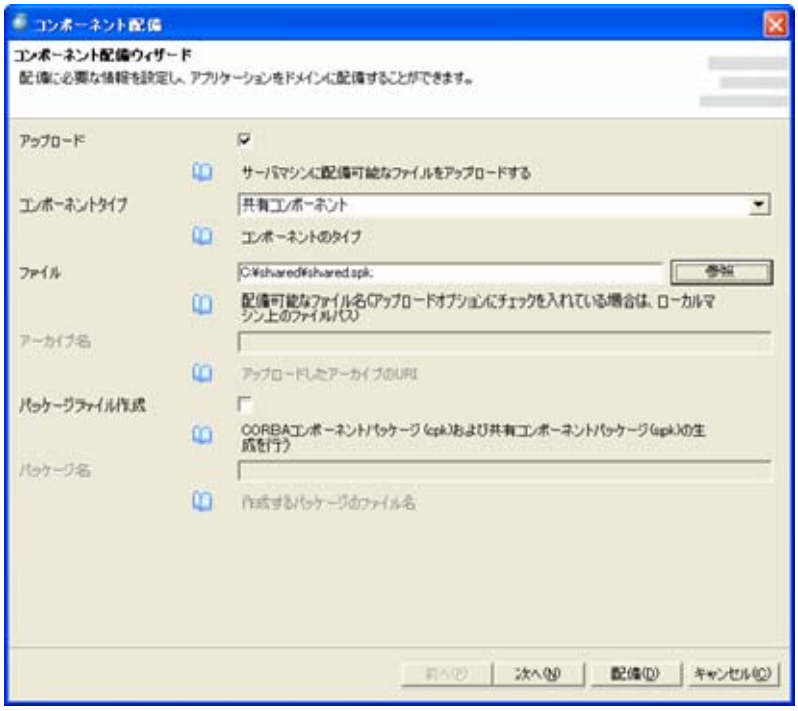
5.5.2 統合運用管理ツールを利用した配備

統合運用管理ツールを利用して、共有コンポーネントを配備する手順を説明します。

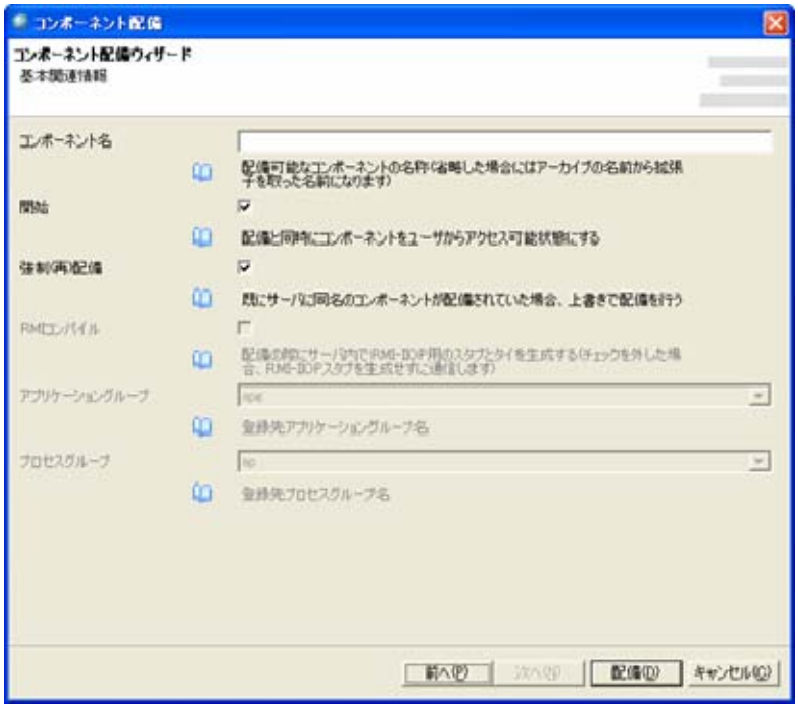
(1) 統合運用管理ツールの画面右側のツリーの「アプリケーション」を選択し、右クリックします。表示されたメニューから「コンポーネントの配備」を選択します。



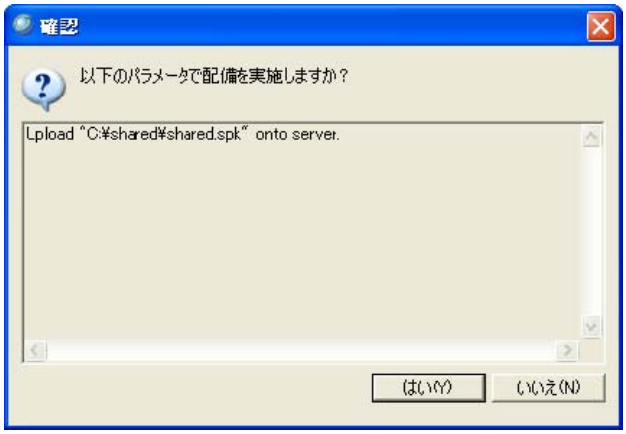
(2) 「コンポーネントタイプ」を「共有コンポーネント」に設定します。また「ファイル」に配備する共有コンポーネントのアーカイブ(.cpk)を設定します。入力が完了したら、「次へ(N)」ボタンを押下します。



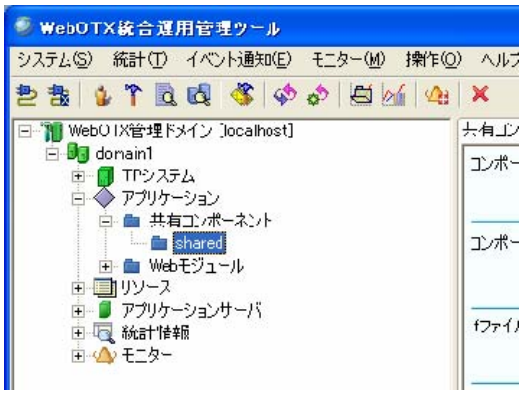
(3) 再配備を行う場合は、「強制(再)配備」にチェックを入れてください。入力が完了したら「配備(D)」ボタンを押下してください。



(4) 配備実行の確認画面が出てきますので、「はい(Y)」ボタンを押下してください。



(5) 配備が正常に完了すると、以下の画面のように、画面右側のツリーの「共有コンポーネント」の下に配備したアプリケーション名が表示されます。



5.6 注意事項

- 再配備時、置換するアプリケーションと置換されるアプリケーションの種類は同じである必要があります。異なる場合は配備に失敗します。
- WebOTX SIP Application Server において、SIP アプリケーションのみの(すなわち Web との統合型ではない).sar ファイルを含んだ.ear ファイルはサポートしません。このような.ear ファイルを配備した場合、配備は失敗します。
- 共有コンポーネントが起動中のプロセスグループで利用されている場合、再配備ができません。再配備するには、関連するプロセスグループを停止してください。
- 共有コンポーネントのアプリケーション名は、コンポーネントファイル名自身(拡張子つきのファイル名)は指定できません。拡張子を除いた名前にしてください。
- コンテキストパスに ″/″ を含めると一部機能が利用できなくなります。詳しくは［注意制限事項 ＞ 3.Web コンテナ ＞ 3.1.Web コンテナの注意事項］を参照してください。

6 配備解除

配備解除とは、WebOTX Application Server の管理下からアプリケーションを除外することです。この章では配備解除について解説します。

6.1 J2EEのアプリケーションの配備解除

この節では、J2EE のアプリケーションの配備解除について説明します。特に断りがない限り、説明は J2EE のアプリケーションの種類すべてに共通しているものとします。

6.1.1 コマンドを利用した配備解除

運用管理コマンドを利用してJ2EEのアプリケーションを配備解除する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照ください。

配備解除するには、アプリケーション名を指定して undeploy オペレーションを実行します。

```
otxadmin> undeploy  アプリケーション名
```

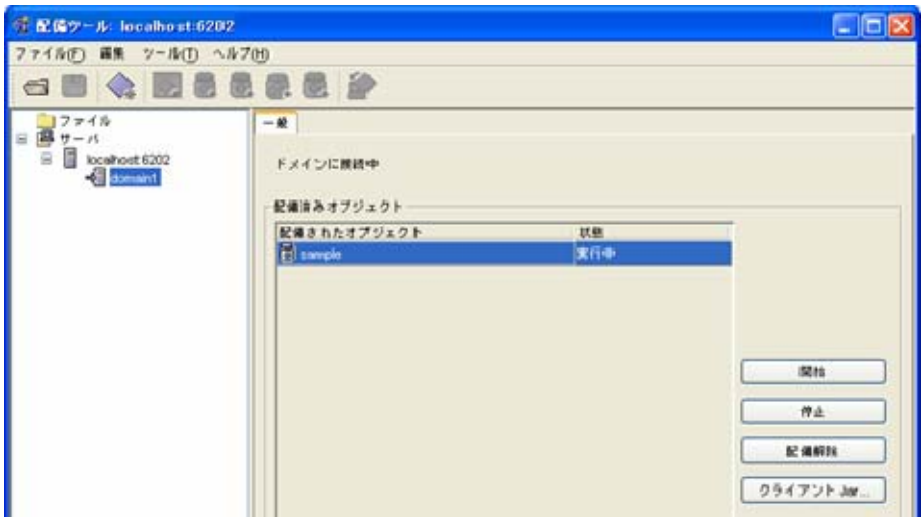
具体例を挙げると、アプリケーション名が「hello」であるアプリケーションを配備解除するには、次のようにします。

```
otxadmin> undeploy hello
```

6.1.2 配備ツールを利用した配備解除

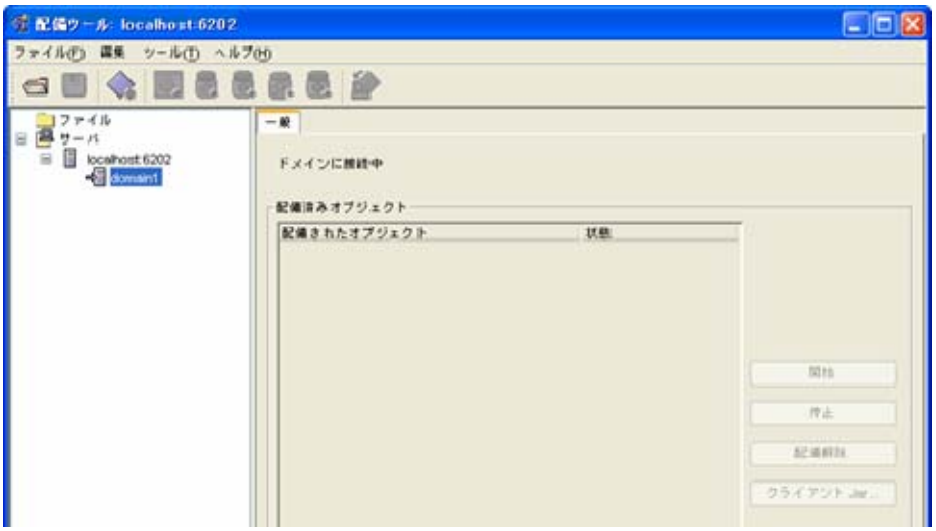
配備ツールを利用して、J2EE アプリケーションの配備解除を行う手順について説明します。

(1) 配備ツールの画面右側のツリーから配備解除するアプリケーションが配備されているドメインを選択します。「配備済みオブジェクト」に表示されている、配備解除するアプリケーションを選択し、「配備解除」ボタンを押下します。



(2) 配備解除の確認のダイアログが表示されます。「はい(Y)」ボタンを押下してください。

(3) 配備解除が成功すると、以下の画面のように「配備済みオブジェクト」から該当アプリケーションが削除されます。

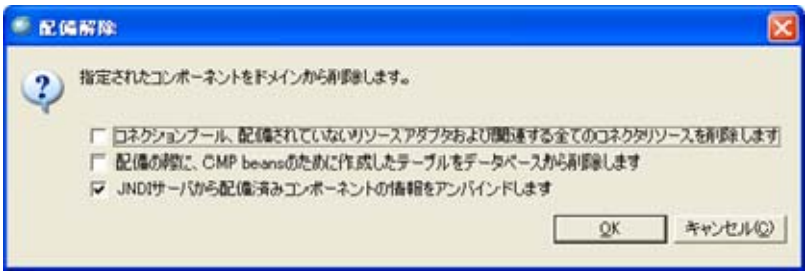


6.1.3 統合運用管理ツールを利用した配備解除

統合運用管理ツールを利用して、J2EE のアプリケーションを配備解除する手順を説明します。

(1) 統合運用管理ツールを起動し、ドメインに接続します。そして画面右側のツリーから配備解除するアプリケーションを選択し、右クリックします。表示されたメニューから「配備解除」を選択します。

(2) 配備解除のオプションを選択するダイアログが表示されますので、必要な情報を設定して、「OK」ボタンを押下します。



(3) 配備解除に成功すると、以下のような画面が表示されます。

6.1.4 オートデプロイ機能による配備解除

オートデプロイ機能での配備で利用した以下のディレクトリから、該当アーカイブを削除することで、自動的に配備解除が開始します。

```
${INSTANCE_ROOT}/autodeploy
```

6.2 CORBAアプリケーションの配備解除

ここでは CORBA アプリケーションの配備解除について説明します。

6.2.1 動的配備解除と静的配備解除

CORBA アプリケーションの形式が R4 形式なのか R5 形式なのかによって、配備解除を行うための条件が異なります。アプリケーショングループが起動状態で配備解除を行うことを、動的配備解除と呼びます。アプリケーショングループが停止状態で配備解除を行うことを、静的配備解除と呼びます。

R4 形式の CORBA アプリケーションの場合、静的配備解除のみをサポートします。配備解除を行うにはその CORBA アプリケーションが属しているアプリケーショングループを停止してください。アプリケーショングループが起動している場合は配備解除できません。

R5 形式の CORBA アプリケーションの場合は、動的配備解除も静的配備解除もサポートしています。ただし、動的配備解除を行う場合は以下の条件を満たす必要があります。

- プロセスグループのバージョンが 6 以上である。
- 該当 CORBA アプリケーションが停止状態である。

バージョン 5 のプロセスグループは動的配備解除をサポートしていませんので、アプリケーショングループを停止して静的配備解除を行ってください。またCORBAアプリケーションが停止状態でない場合は、配備解除ができません。CORBAアプリケーションを停止状態にすることで、配備解除を行うことができます。ただし停止状態にする際、現在実行中のオペレーションがエラーになる可能性があります。安全な停止方法については§7.2.2を参照してください。

6.2.2 コマンドを利用した配備解除

運用管理コマンドを利用してCORBAアプリケーションを配備解除する方法について説明します。CORBAアプリケーションを配備解除するための条件については§6.2.1を参照してください。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照ください。

配備解除するには、アプリケーション名を指定して undeploy オペレーションを実行します。

```
otxadmin> undeploy  アプリケーション名
```

例えば、アプリケーション名が「corba」である CORBA アプリケーションを配備解除するには、次のようにします。

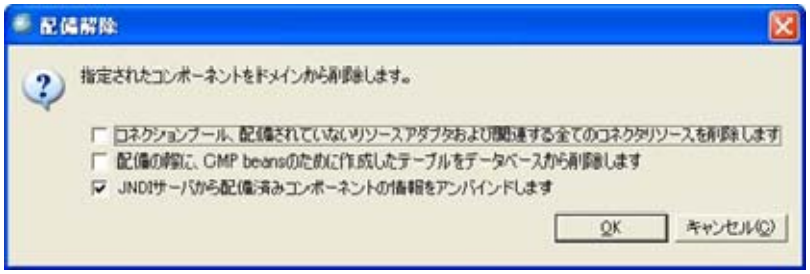
```
otxadmin> undeploy corba
```

6.2.3 統合運用管理ツールを利用した配備解除

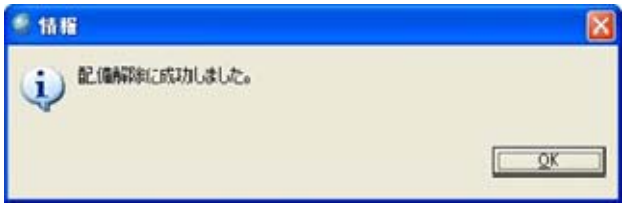
ここでは統合運用管理ツールを利用してCORBAアプリケーションを配備解除する手順を説明します。CORBAアプリケーションを配備解除するための条件については§6.2.1を参照してください。

(1) 統合運用管理ツールの画面右側のツリーから、配備解除する CORBA アプリケーションの名前を選択し、右クリックします。表示されたメニューの中から「配備解除」を選択します。

(2) 配備解除の各種設定を行った後、「OK」ボタンを押下します。



(3) 配備解除に成功すると、以下のような画面が表示されます。



6.3 共有コンポーネントの配備解除

ここでは、共有コンポーネントの配備解除について説明します。

6.3.1 コマンドを利用した配備解除

運用管理コマンドを利用して共有コンポーネントを配備解除する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照ください。

配備解除するには、アプリケーション名を指定して undeploy オペレーションを実行します。

```
otxadmin> undeploy  アプリケーション名
```

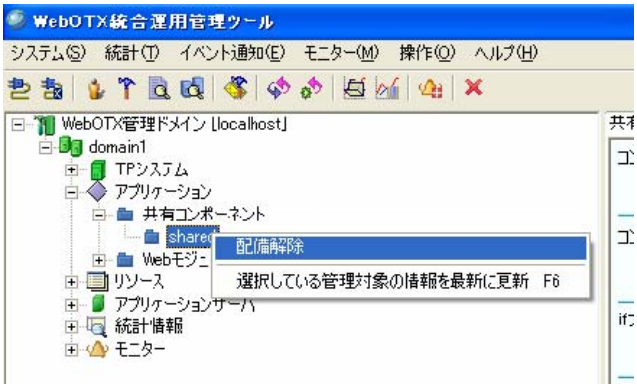
例えば、アプリケーション名が「shared」である共有コンポーネントを配備解除するには、次のようにします。

```
otxadmin> undeploy shared
```

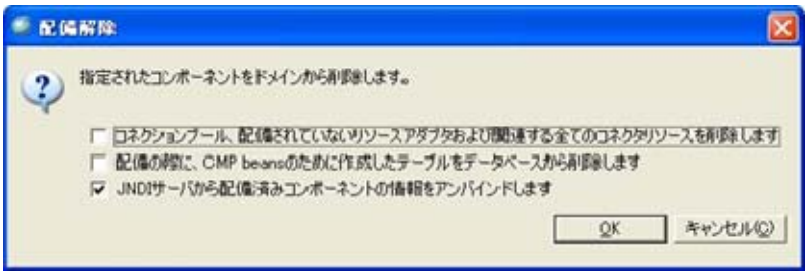
6.3.2 統合運用管理ツールを利用した配備解除

ここでは、統合運用管理ツールを利用して共有コンポーネントを配備解除する手順について説明します。

(1) 統合運用管理ツールの画面右側のツリーから、配備解除する共有コンポーネントの名前を選択し、右クリックします。表示されたメニューの中から「配備解除」を選択します。



(2) 配備解除の各種設定を行った後、「OK」ボタンを押下します。



(3) 配備解除に成功すると、以下のような画面が表示されます。

6.4 注意事項

- 共有コンポーネントが起動中のプロセスグループで利用されている場合、配備解除ができません。配備解除するには、関連するプロセスグループは停止してください。
- 共有コンポーネントを配備解除しても、アプリケーションが利用する共有コンポーネントの設定は削除されません。共有コンポーネントを配備解除した後にアプリケーションの設定も見直してください。

7 開始と停止

アプリケーションが開始した状態とは、配備されたアプリケーションが動作しており、クライアントからの要求を受け付けることができる状態です。また、アプリケーションが停止した状態とは、アプリケーションが動作していない状態のことで、クライアントからの要求は受け付けることができませんが、アプリケーションは依然としてWebOTX の管理下にあります。この章ではアプリケーションの開始と停止について解説します。

7.1 J2EEのアプリケーションの開始と停止

この節では、J2EE のアプリケーションの開始と停止について説明します。特に断りがない限り、説明は J2EE のアプリケーションの種類すべてに共通しているものとします。

7.1.1 コマンドを利用した開始と停止

運用管理コマンドを利用してJ2EEのアプリケーションを開始/停止する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照してください。

アプリケーションを開始するにはオペレーション enable を、停止するにはオペレーション disable を実行します。

```
otxadmin> enable  アプリケーション名
otxadmin> disable  アプリケーション名
```

具体例を挙げると、アプリケーション名が「myapp」であるアプリケーションを開始し、その後停止するには、次のようにします。

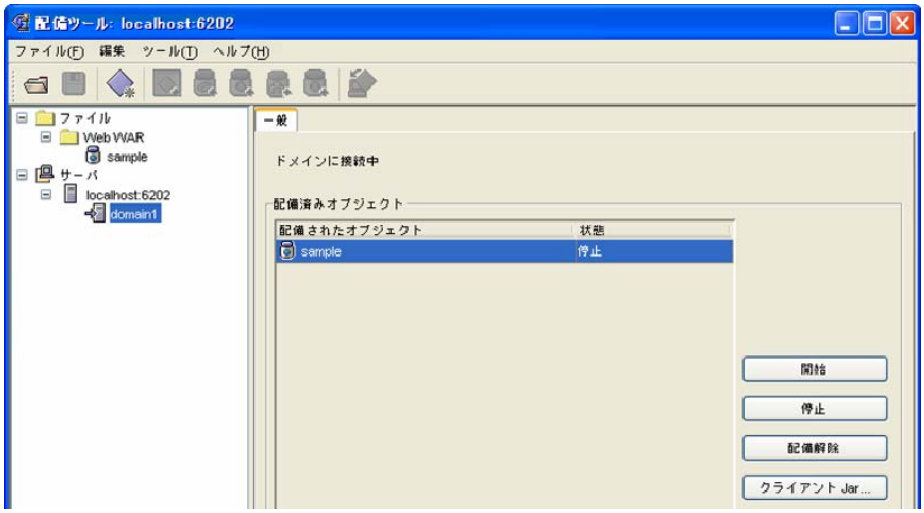
```
otxadmin> enable myapp
otxadmin> disable myapp
```

7.1.2 配備ツールを利用した開始と停止

配備ツールを利用して、J2EE アプリケーションの配備解除を行う手順について説明します。

まず、開始の手順から説明します。

(1) 配備ツールの画面右側のツリーから開始するアプリケーションが配備されているドメインを選択します。「配備済みオブジェクト」に表示されているアプリケーションから、開始するアプリケーションを選択し、「開始」ボタンを押下します。

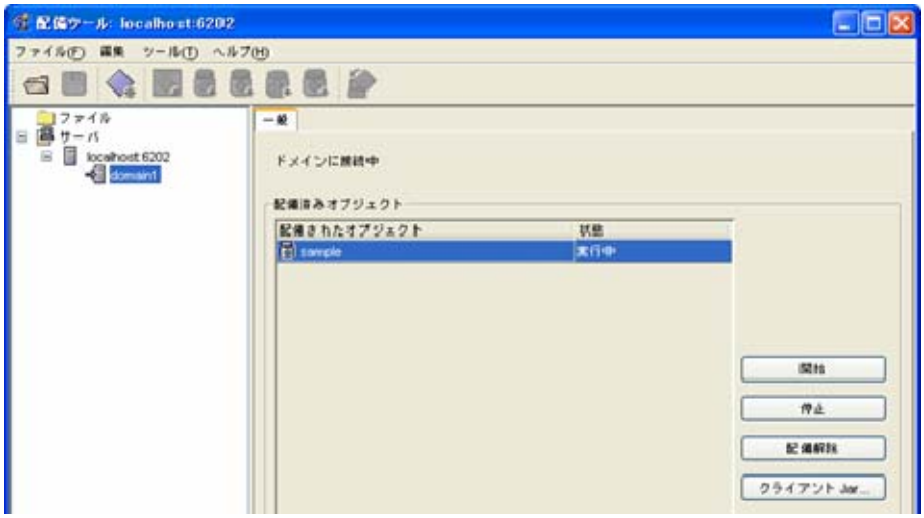


(2) 開始の確認のダイアログが表示されます。「はい(Y)」ボタンを押下してください。

(3) 処理が正常に完了すると、「配備済みオブジェクト」に表示されている該当アプリケーションの状態が「実行中」になります。

次に、停止の手順を説明します。

(1) 配備ツールの画面右側のツリーから停止するアプリケーションが配備されているドメインを選択します。「配備済みオブジェクト」に表示されているアプリケーションから、停止するアプリケーションを選択し、「停止」ボタンを押下します。



(2) 停止の確認のダイアログが表示されます。「はい(Y)」ボタンを押下してください。

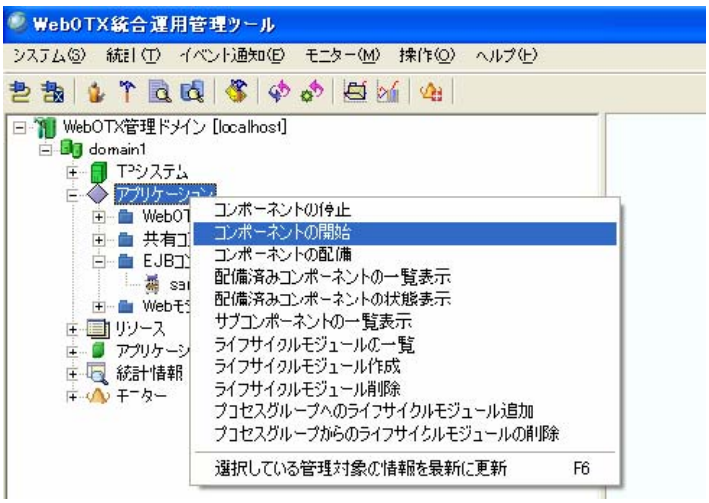
(3) 処理が正常に完了すると、「配備済みオブジェクト」に表示されている該当アプリケーションの状態が「停止」になります。

7.1.3 統合運用管理ツールを利用した開始と停止

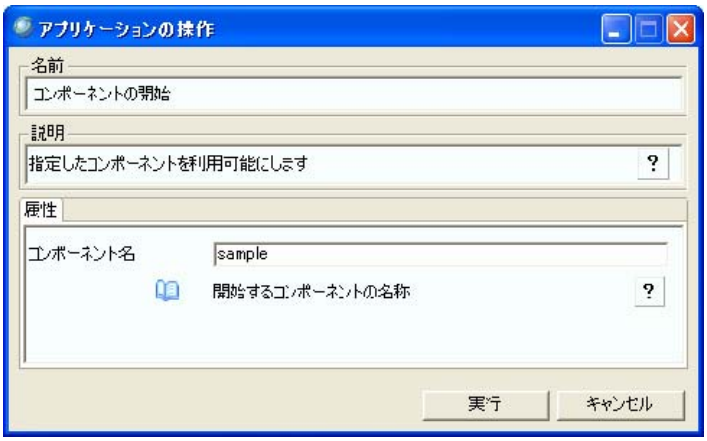
ここでは統合運用管理ツールを利用して、J2EE アプリケーションを開始/停止する手順を説明します。

まず、アプリケーションの開始から説明します。

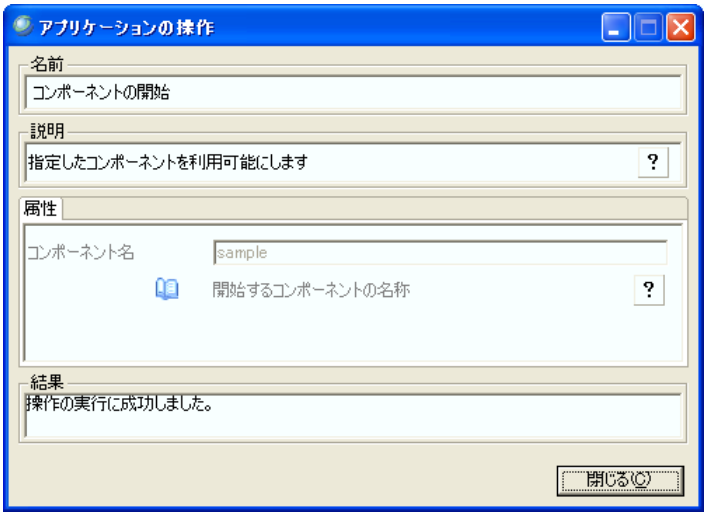
(1) 統合運用管理ツールの画面右側のツリーから「アプリケーション」を選択し、右クリックします。表示されたメニューから「コンポーネントの開始」をクリックします。



(2)　表示されたダイアログに、開始するアプリケーションの名前を入力します。入力が完了したら「実行」ボタンを押下します。

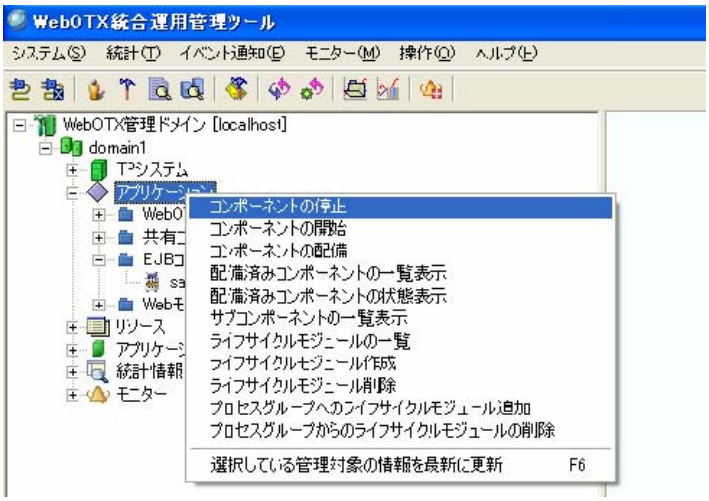


(3)　開始に成功すると、以下の画面のように成功を示すメッセージが表示されます。

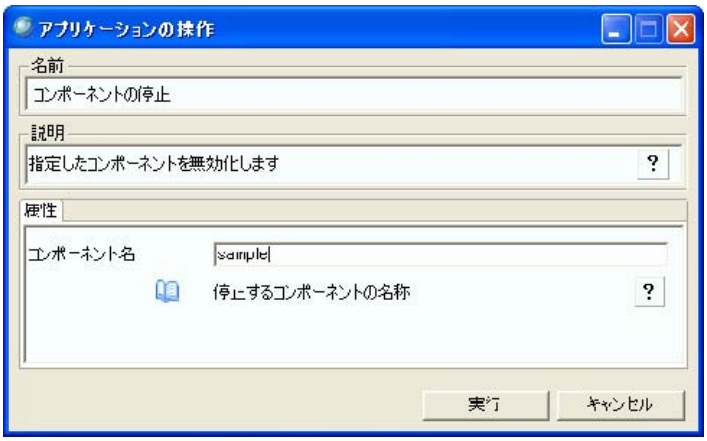


次に、アプリケーションの停止の手順について説明します。

(1) 統合運用管理ツールの画面右側のツリーから「アプリケーション」を選択し、右クリックします。表示されたメニューから「コンポーネントの停止」をクリックします。



(2) 表示されたダイアログに、停止するアプリケーションの名前を入力します。入力が完了したら「実行」ボタンを押下します。



(3) 停止に成功すると、以下の画面のように成功を示すメッセージが表示されます。

7.2 CORBAアプリケーションの開始と停止

ここでは、CORBA アプリケーションの開始と停止について説明します。

7.2.1 CORBAアプリケーションの形式

CORBA アプリケーションの形式が R4 形式なのか R5 形式なのかによって、開始/停止の方法が異なります。

R4 形式の CORBA アプリケーションの場合、アプリケーション単位の開始/停止はできません。プロセスグループの開始/停止、あるいはアプリケーショングループの開始/停止により、該当 CORBA アプリケーションを開始/停止してください。

R5 形式の CORBA アプリケーションの場合は、アプリケーション単位の開始/停止をサポートしています。ただし、アプリケーション単位の開始/停止を行う場合は以下の条件を満たす必要があります。

- プロセスグループのバージョンが 6 以上である。

ただし、アプリケーション単位の開始が成功するためには、そのアプリケーションが属しているプロセスグループが開始している必要があります。

バージョン 5 のプロセスグループはアプリケーション単位の開始/停止をサポートしていませんので、プロセスグループの開始/停止、あるいはアプリケーショングループの開始/停止により、該当 CORBA アプリケーションを開始/停止してください。

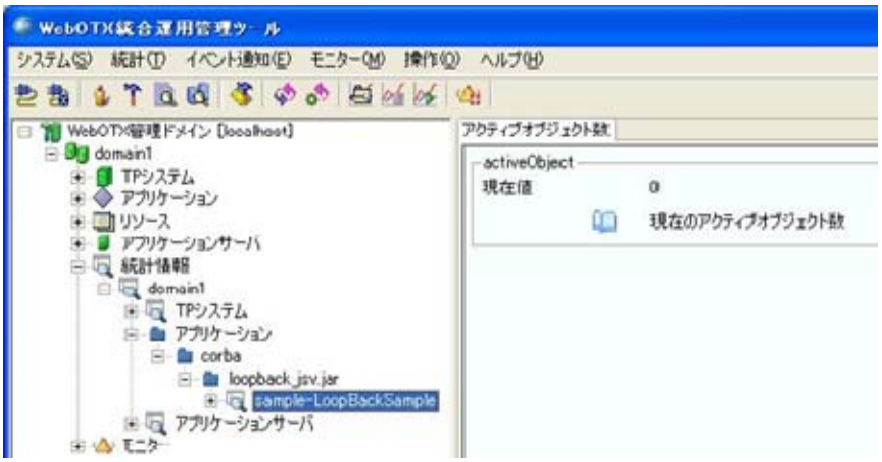
7.2.2 安全な停止

使用中の CORBA アプリケーションを停止してしまうと、以後はその CORBA アプリケーションに対する呼び出しは失敗します。安全に CORBA アプリケーションの停止を行うためには、その CORBA アプリケーションがクライアントから利用されていないか確認する必要があります。CORBA アプリケーションの利用状況ついては、以下の情報が参考になります。

- モジュールアイドル時間が十分に長い
- モジュールアクティブオブジェクト数が 0（ステートフルの場合）

モジュールアイドル時間が十分に長い場合、そのアプリケーションは利用されていないと判断できます。モジュールアイドル時間が-1 の時は、オペレーションが実行中であることを示しています。。

統合運用管理ツールを利用する場合、モジュールアイドル時間とモジュールアクティブオブジェクト数は以下の場所で確認できます。



また、運用管理コマンドを利用する場合は、以下のようにします。

otxadmin> get applications.corba-applications.[アプリケーション名].[モジュール名].idleTime

otxadmin> get -m applications.[アプリケーション名].[モジュール名].
[インタフェース名].activeObject-Current

具体例を挙げると、アプリケーション名が「myapp」、モジュール名が「corba.jar」、インタフェース名が「loopback」の場合、次のようになります。モジュール名に含まれるピリオド「.」は円マーク「¥」でエスケープする必要がある点に注意してください。

otxadmin> get applications.corba-applications.myapp.corba¥.jar.idleTime

otxadmin> get -m applications.myapp.corba¥.jar.loopback.activeObject-Current

7.2.3 コマンドを利用した開始と停止

運用管理コマンドを利用してCORBAアプリケーションを開始/停止する方法について説明します。なお、運用管理コマンドの詳細については運用管理コマンドリファレンスマニュアルの「otxadmin」を参照してください。

アプリケーション単位の開始/停止が可能な場合には、開始するにはオペレーション enable を、停止するにはオペレーション disable を実行します。

otxadmin> enable アプリケーション名

otxadmin> disable アプリケーション名

具体例を挙げると、アプリケーション名が「myapp」である CORBA アプリケーションを開始し、その後停止するには、次のようにします。

```
otxadmin> enable myapp
otxadmin> disable myapp
```

アプリケーション単位の開始/停止がサポートされていない場合には、プロセスグループあるいはアプリケーショングループの開始/停止を行うことにより、該当 CORBA アプリケーションを開始/停止します。プロセスグループの開始/停止にはオペレーション start-pg/stop-pg を、アプリケーショングループの開始/停止にはオペレーション start-apg/stop-apg を利用します。

```
otxadmin> start-pg プロセスグループ名
otxadmin> stop-pg プロセスグループ名
otxadmin> start-apg アプリケーショングループ名
otxadmin> stop-apg アプリケーショングループ名
```

具体例を挙げます。アプリケーショングループ「apg」、プロセスグループ「pg」に配備されている CORBA アプリケーション「myapp」があるとします。アプリケーショングループを開始することによりアプリケーションを開始し、プロセスグループを停止することによりアプリケーションを停止するには、以下のようにします。

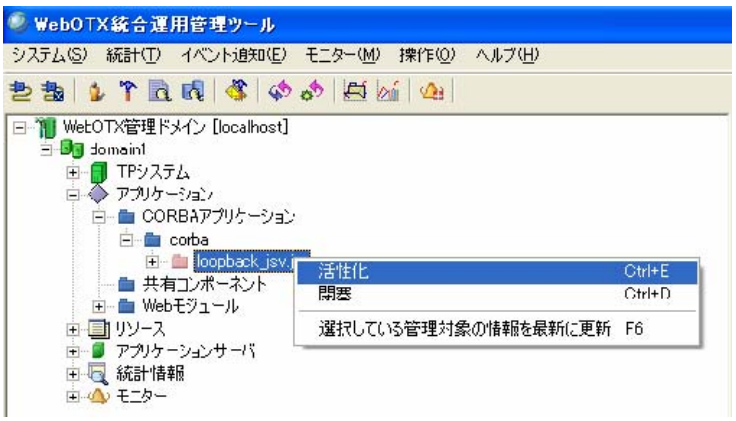
```
otxadmin> start-apg apg
otxadmin> stop-pg pg
```

7.2.4 統合運用管理ツールを利用した開始と停止

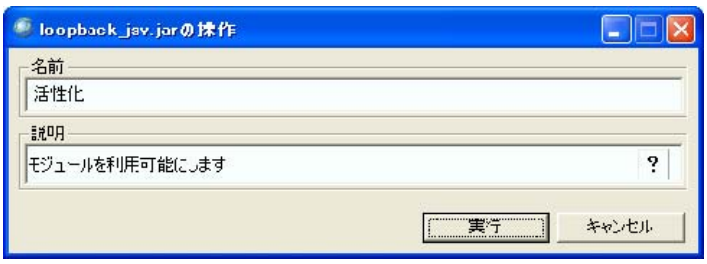
アプリケーション単位の開始/停止が可能な場合について解説します。アプリケーショングループとプロセスグループの開始/停止の手順については「運用編（TPモニタの運用操作）」を参照してください。

まず CORBA アプリケーションを開始する手順を示します。

(1) 統合運用管理ツールの画面右側の、CORBA アプリケーションのモジュール名を選択し、右クリックします。表示されたメニューの中から「活性化」を選択します。

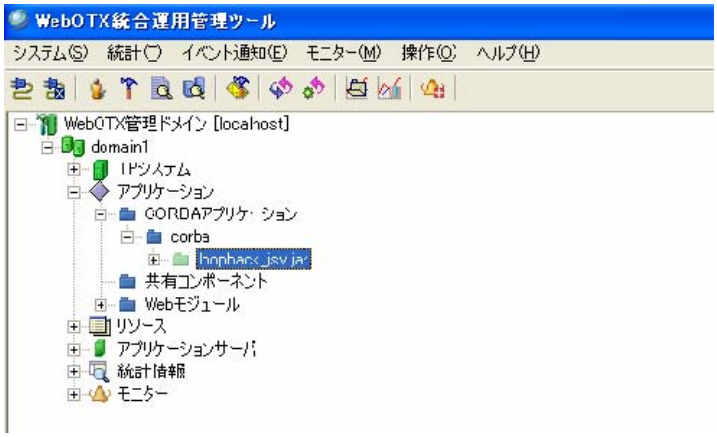


(2) 開始のための確認のダイアログが表示されますので、「実行」ボタンを押下してください。



(3) 開始処理が完了したことを示すメッセージが表示されますので、「閉じる」ボタンを押下して下さい。

(4) 正常に開始処理が完了すると、CORBA アプリケーションのモジュールを示すノードのアイコンが緑色になります。



次に、CORBA アプリケーションを停止する手順について解説します。

7.3 注意事項

- 配備ツールは J2EE に特化したツールであるため、CORBA アプリケーションの開始/停止はできません。
- 共有コンポーネントには「開始」や「停止」はありません。

8 ライフサイクルモジュール

ライフサイクルモジュールについて説明します。

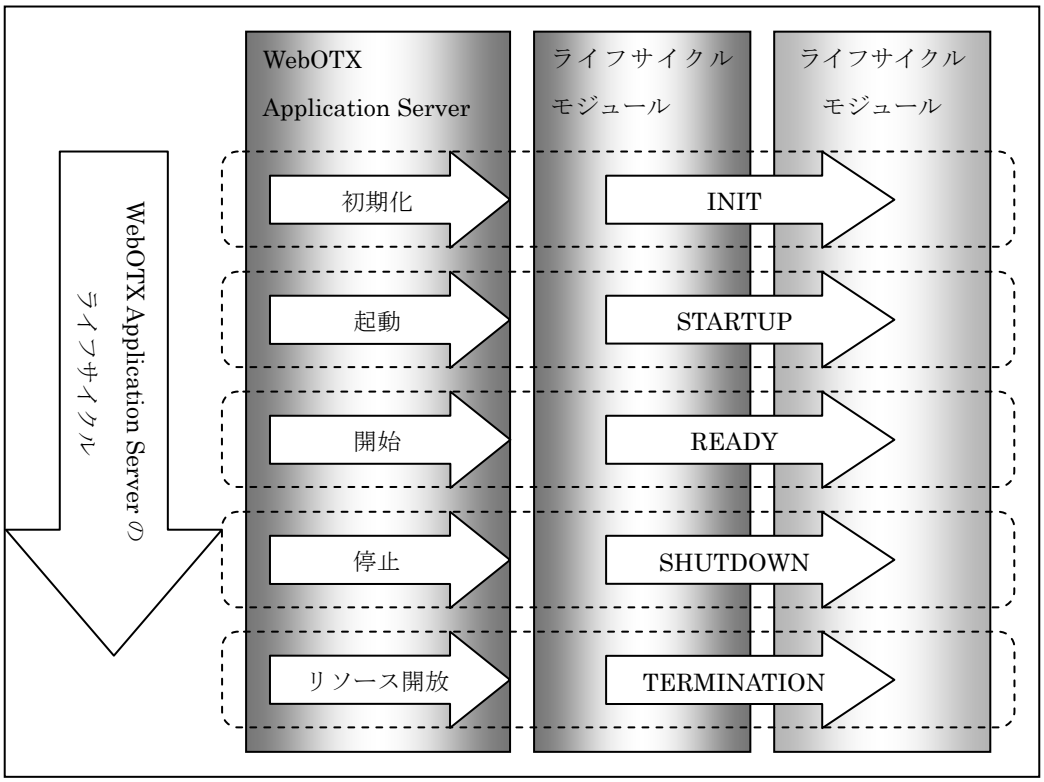
8.1 ライフサイクルモジュールとは

ライフサイクルモジュールとは、WebOTX Application Server の起動・停止のライフサイクルの状態に合わせ実行される Java クラスまたはモジュールであり、ユーザが作成した任意のクラスまたはモジュールをライフサイクルモジュールとすることが可能です。

WebOTX Application Server ではライフサイクルの状態管理のために以下の状態を定義しています。ライフサイクルモジュールはこの状態により管理され、WebOTX Application Server のライフサイクルに同期した処理を実行します。

状態	説明
INIT	初期化処理中であることを表します
STARTUP	起動処理中であることを表します
READY	サービス開始処理中であることを表します
SHUTDOWN	停止処理中であることを表します
TERMINATION	リソース開放処理中であることを表します

ライフサイクルモジュールが複数登録されている場合は、各ライフサイクルモジュールは並列に処理されます。たとえば、WebOTX Application Server が初期化の状態であるとき、全てのライフサイクルモジュールの状態は INIT であり、全てのライクサイクルモジュールの INIT イベントによる処理が終了するまで、次の状態には移行しません。



8.2 ライフサイクルモジュール登録の流れ

ライフサイクルモジュールを登録する場合の流れについて説明します。

(1) ライフサイクルモジュールの作成

登録するコンポーネントと LifecycleListener 実装クラスを作成します。登録に必要なファイルは以下のとおりです。

コンポーネントと LifecycleListener 実装クラスをまとめた jar ファイル

(2) ライフサイクルモジュールの登録

ライフサイクルモジュールをWebOTX Application Serverに登録します。登録は統合運用管理ツールおよび運用管理コマンドで行なうことができます。

(3) ライフサイクルモジュールの設定

ライフサイクルモジュールの設定を変更します。この設定はライフサイクルモジュールの登録時にも行うことができます。登録後に設定の変更が必要な場合に行ってください。変更は統合運用管理ツールおよび運用管理コマンドで行なうことができます。

8.3 ライフサイクルモジュールの作成

WebOTX Application Server へのライフサイクルモジュールの登録には WebOTX Application Server とライフサイクルモジュールのインタフェースとして、LifecycleListener インタフェースを実装したクラスを作成する必要があります。

LifecycleListener インタフェースは Public メソッドとして handleEvent を持ちます。WebOTX Application Server はこのメソッドを呼び出し、LifecycleLisner 実装クラスに状態情報を保持した LifecycleEvent クラスのオブジェクトを渡します。

LifecycleEvent クラスは状態情報として以下の public かつ static なフィールドを持ちます。

フィールド名	説明
INIT_EVENT	ライフサイクルモジュールの初期化処理中であることを表します
STARTUP_EVENT	ライフサイクルモジュールの起動処理中であることを表します
READY_EVENT	ライフサイクルモジュールのサービス開始処理中であることを表します
SHUTDOWN_EVENT	ライフサイクルモジュールの停止処理中であることを表します
TERMINATION_EVENT	ライフサイクルモジュールのリソース開放処理中であることを表します

LifecycleListener 実装クラスでは、この状態情報に応じた処理を実装する必要があります。

LifecycleListener 実装クラスの実装方法については「API リファレンスマニュアル」の「6.ライフサイクル・リスナ」を参照してください。

8.4 統合運用管理ツールによる登録・登録削除および一覧の取得

統合運用管理ツールよりライフサイクルモジュールの登録・登録削除および一覧を取得する方法について説明します。なお、ここでは以下の条件でライフサイクルモジュールの登録を行っています。

LifecycleListener 実装クラス名 ：sample.SampleLifecycleModule

モジュールと LifecycleListener 実装クラスを含む jar ファイル名 ：sample.jar

jar ファイル配置位置 ：C ドライブ直下

ライフサイクルモジュールの登録名 ：Sample

アプリケーショングループ名 ：apg

プロセスグループ名 ：pg

8.4.1 登録

以下の手順で登録します。なお、ここではライフサイクルモジュール名に加え、クラスパスと必須入力項目であるクラス名のみ指定しています。必要に応じて他の設定項目を入力して実行してください。

(1) 統合運用管理ツールよりドメインと接続します。

(2) ツリービューより[アプリケーション]を選択します。

(3) [操作]-[ライフサイクルモジュールの作成]を実行します。

(4) ライフサイクルモジュール名、クラスパス、クラス名を入力し、「OK」ボタンを押します。

Standard Edition および Enterprise Edition において、ライフサイクルモジュールをプロセスグループ上で動作させる場合は、追加で以下の作業も必要になります。

(5) ツリービューより[アプリケーション]を選択します。

(6) [操作]-[プロセスグループへのライフサイクルモジュールを追加]を実行します。

(7) 登録するライフサイクルモジュール名、アプリケーショングループ名、プロセスグループ名を入力し、「OK」ボタンを押します。

8.4.2 登録削除

ドメインからライフサイクルモジュールを登録削除する手順を示します。Standard Edition または Enterprise Edition で、登録削除対象のライフサイクルモジュールをプロセスグループへも登録している場合は、先にプロセスグループからライフサイクルモジュールを登録削除する必要がある点に注意してください。

(1) ツリービューより[アプリケーション]を選択します。

(2) [操作]-[ライフサイクルモジュールの削除]を実行します。

(3) 登録削除するライフサイクルモジュール名を入力し、「OK」ボタンを押します。

次に、プロセスグループからライフサイクルモジュールを登録削除する手順を示します。

(1) ツリービューより[アプリケーション]を選択します。

(2) [操作]-[プロセスグループからのライフサイクルモジュールの削除]を実行します。

(3) 登録削除するライフサイクルモジュール名、アプリケーショングループ名、プロセスグループ名を入力し、「OK」ボタンを押します。

8.4.3 一覧の取得

まず、ドメインに登録されたライフサイクルモジュールの一覧を取得する手順を以下に示します。

(1) ツリービューより[アプリケーション]を選択します。

(2) [操作]-[ライフサイクルモジュールの一覧]を実行します。

(3) 「OK」ボタンを押します。Standard Edition, Enterprise Edition の場合はアプリケーショングループ名とプロセスグループ名の入力欄が表示されますが、何も入力せずに「OK」ボタンを押してください。

次に、プロセスグループに登録されたライフサイクルモジュールの一覧を取得する手順を以下に示します。こちらは Standard Edition および Enterprise Edition でサポートしています。

(1) ツリービューより[アプリケーション]を選択します。

(2) [操作]-[ライフサイクルモジュールの一覧]を実行します。

(3) 一覧を取得したいプロセスグループのアプリケーショングループ名、プロセスグループ名を入力し、「OK」ボタンを押します。

8.5 コマンドによる登録・登録削除および一覧の取得

運用管理コマンドよりライフサイクルモジュールコンポーネントの登録および登録削除する方法について説明します。なお、ここでは以下の条件でライフサイクルモジュールの登録を行っています。

LifecycleListener 実装クラス名 ：sample.SampleLifecycleModule

モジュールと LifecycleListener 実装クラスを含む jar ファイル名 ：sample.jar

jar ファイル配置位置 ：C ドライブ直下

ライフサイクルモジュールの登録名 ：Sample

アプリケーショングループ名 ：apg

プロセスグループ名 : pg

8.5.1 登録

ライフサイクルモジュールをドメインに登録するには、登録するコンポーネントとライフサイクルモジュールの初期設定情報を指定して、create-lifecycle-module コマンドを実行します。なお、ここではコマンドのオプションとして classpath と必須オプションである classname のみ指定しています。必要に応じて他のオプションを指定して実行してください。

```
otxadmin> create-lifecycle-module --classname sample.SampleLifecycleModuleClass --classpath
C:/Sample.jar Sample
```

Standard Edition および Enterprise Edition において、ライフサイクルモジュールをプロセスグループに登録するには、登録するライフサイクルモジュール名、アプリケーショングループ名、プロセスグループ名を指定して、add-pg-lifecycle-module コマンドを実行します。

```
otxadmin> add-pg-lifecycle-module --apgroup apg --pgroup pg Sample
```

8.5.2 登録削除

ライフサイクルモジュールをドメインから登録削除するには、登録削除するライフサイクルモジュール名を指定して、delete-lifecycle-module コマンドを実行します。ただし、Standard Edition または Enterprise Edition をお使いで、登録削除対象のライフサイクルモジュールをプロセスグループへも登録している場合は、先にプロセスグループからの登録削除を行う必要があります。

```
otxadmin> delete-lifecycle-module Sample
```

Standard Edition および Enterprise Edition において、ライフサイクルモジュールをプロセスグループから登録削除するには、登録削除するライフサイクルモジュール名、アプリケーショングループ名、プロセスグループ名を指定して、remove-pg-lifecycle-module を実行します。

```
otxadmin> remove-pg-lifecycle-module --apgroup apg --pgroup pg Sample
```

8.5.3 一覧の取得

ドメインに登録されたライフサイクルモジュールの一覧を取得するには、list-lifecycle-modules コマンドをオプション指定なしで実行します。

```
otxadmin> list-lifecycle-modules
```

Standard Edition および Enterprise Edition において、プロセスグループに登録されたライフサイクルモジュールの一覧を取得するには、オプションで一覧を取得したいプロセスグループのアプリケーショングループ名、プロセスグループ名を指定し、list-lifecycle-modules コマンドを実行します。

```
otxadmin> list-lifecycle-modules --apgroup apg --pgroup pg
```

8.6 ライフサイクルモジュールの設定

ライフサイクルモジュールは JMX 仕様に基づき、WebOTX Application Server 内で管理対象オブジェクト(Managed Object : MO)として登録され、管理されます。

登録されたライフサイクルモジュールは統合運用管理ツールまたは運用管理コマンドからその設定を変更することが可能です。ただし設定の変更を反映させるには、そのライフサイクルモジュールの登録先(ドメイン又はプロセスグループ)を再起動する必要があります。

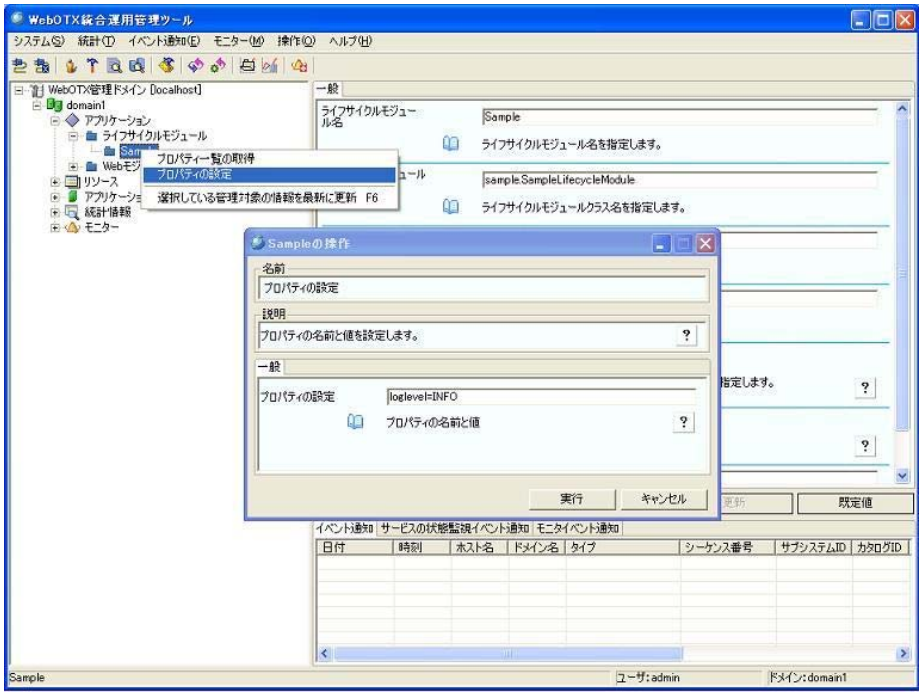
8.6.1 ライフサイクルモジュールMOの設定項目

ライフサイクルモジュールの設定項目についての一覧を以下に示します。なお、MO の詳細については運用編「MO 定義リファレンス」を参照してください。また、設定方法については運用編「統合運用管理ツール」、運用編「運用管理コマンド」を参照してください。

属性名	値	統合運用管理ツールでの表記	Dotted Name	説明
name	文字列	ライフサイクルモジュール名	server.applications.lifecycle-module.*ライフサイクルモジュール名*.name	ライフサイクルモジュール名です。ここで指定した名前でライフサイクルモジュールをドメインに登録します。ただし、文字列にスペースを混入させることはできません。
class-name	文字列	ライフサイクルモジュールクラス名	server.applications.lifecycle-module.*ライフサイクルモジュール名*.class-name	LifecycelLisner 実装クラスのクラス名を、パッケージ名を含めて指定します。
classpath	文字列	クラスパス	server.applications.lifecycle-module.*ライフサイクルモジュール名*.classpath	ライフサイクルモジュールとして登録するコンポーネントのクラスパスです。クラスパスを複数指定する場合は各パスを Windows OS では「;」、UNIX 系 OS では「:」で区切ってください。また、クラスパスの通っているフォルダ/ディレクトリに格納する場合はクラスパスの指定は不要になります。(例. <INSTANCE_ROOT>/lib など)
load-order	数値	サービス実行順番	server.applications.lifecycle-module.*ライフサイクルモジュール名*.load-order	サーバ起動時に、この起動順序番号に従って各モジュールが順次起動されます。既定ではドメインに登録された順序に従って起動されます。ライフサイクルモジュールの起動順序に依存関係がある場合はこの値を指定する必要があります。
is-failure-fatal	true/false	サービス起動失	server.applications.lif	既定値 false を選択す

		敗時動作	ecycle-module.*ライフサイクルモジュール名*.is-failure-fatal	るとドメイン起動時にライフサイクルモジュールの起動処理に失敗した場合でもドメインの起動を継続します。
enabled	true/false	サービス起動設定	server.applications.lifecycle-module.*ライフサイクルモジュール名*.enabled	ライフサイクルモジュールの有効の可否を決定します。
description	文字列	説明	server.applications.lifecycle-module.*ライフサイクルモジュール名*.description	登録したライフサイクルモジュールに関する説明書き欄です。
property	文字列	（以下で解説）	server.applications.lifecycle-module.*ライフサイクルモジュール名*.property.*プロパティ名*	ユーザにより定義可能なライフサイクルモジュールの任意のプロパティです。「プロパティ名＝値」の形式で指定してください。運用管理コマンドから複数指定する場合はプロパティの組を「:」で繋げてください。

属性 property はライフサイクルモジュールの「一般」タグ内には表示されません。統合運用管理ツールから property を取得・設定するにはツール上のライフサイクルモジュールの右クリックから「プロパティ一覧の取得」または「プロパティの設定」を実行してください。

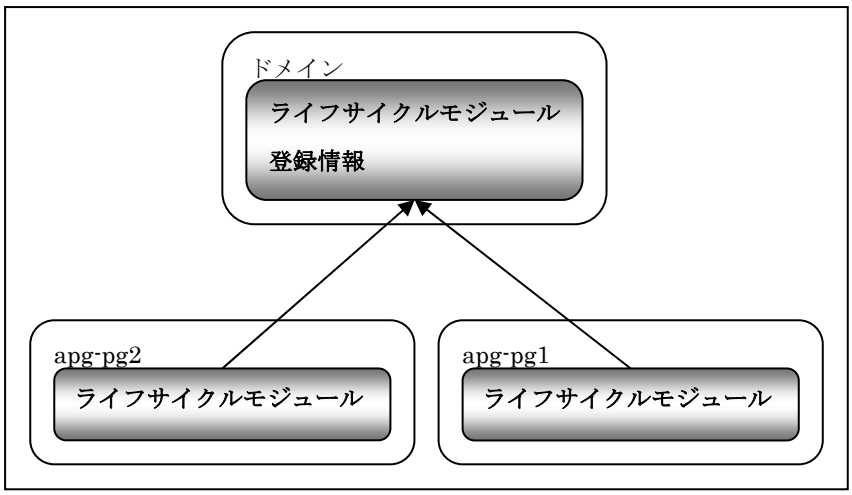


なお、コマンドから property を取得・設定するにはその他の設定項目同様に get・set コマンドを使用してください。

8.6.2 各プロセス上での有効・無効設定

WebOTX Application Server では複数プロセスグループ上でライフサイクルモジュールの機能を提供するため、各プロセスグループ上からはドメインに登録されたライフサイクルモジュールの情報を参照し、設定情報を取得しています。したがって、ドメインに登録された情報への参照の有効・無効を設定することで、プロセスグループごとにライフサイクルモジュールの有効・無効を設定することができます。

以下の図はドメイン上のライフサイクルモジュールの情報を各プロセスグループから参照する場合のイメージです。ただし、ここでは各プロセスグループを表すのにアプリケーショングループ名とプロセスグループ名を-(ハイフン)でつないだサーバ名で表しています。



参照の有効・無効設定を行うには、運用管理コマンドから以下の手順で変更します。

otxadmin> set *サーバ名*.application-ref.ライフサイクルモジュール名.enabled=true

または

otxadmin> set *サーバ名*.application-ref.ライフサイクルモジュール名.enabled=false

9 配備チューニング

この章では主に配備処理を高速化する方法について解説します。アプリケーションの開発時には、配備／配備解除が頻繁に繰り返されるため、配備処理の高速化を意識することで開発効率が向上します。

9.1 EJBのメソッド識別情報の割り当て抑止(Standard/Enterprise Edition)

TP モニタでは、オペレーション毎にトランザクション等の実行時情報を管理する機能を備えています。あるモジュールが配備されると、TP モニタでは個々のオペレーションを識別するために番号を付与して、内部管理テーブルと状態情報との関連付けを行います。EJB モジュールの場合には、オペレーションがリモートインタフェースのメソッドに該当します。

メソッド識別情報の割り当て範囲を限定することにより、配備処理中に行われる識別情報の生成数を削減し、配備処理にかかる時間の短縮およびメモリ消費量の削減が可能です。

9.1.1 オプション

割り当て方法は以下の中から選択できます。

- 全てのリモートメソッドに対して割り当てる
  全てのリモートメソッドに割り当てます。
- ビジネスメソッドのみ割り当てる（既定値）
  おもにユーザ定義のメソッドについて割り当てます。リモートインタフェース、ホームインタフェースのメソッドの割り当てについては EJB 種別によって異なります。下記表を参照してください。
- 割り当てない
  リモートメソッドに対しては割り当てません。

メソッド名	ステートレス Session Bean	ステートフル Session Bean	Entity Bean	メッセージドリブン Bean
リモートインタフェース				
getEJBHome()	×	×	×	―
getPrimaryKey()	×	×	×	―
getHandle()	×	×	×	―
isIdentical(javax.ejb.EJBObject)	×	×	×	―
remove()	×	○	○	―
ホームインタフェース				
getEJBMetaData()	×	×	×	―
getHomeHandle()	×	×	×	―
remove(java.lang.Object)	×	×	○	―
remove(javax.ejb.Handle)	×	○	○	―
[EJB 生成メソッド] (例：create()メソッド)	×	○	○	―

[ファインダメソッド] (例:findByPrimaryKey(String)メソッド)	ー	ー	○	ー

凡例

○ 割り当てます
× 割り当てられません
ー 存在しません

(注:メッセージドリブン Bean の onMessage()メソッド、タイマーBean の ejbTimeout()メソッドについてはリモートメソッドではありませんが、どのオプションでも割り当てられます。)

識別情報を割り当てなかったメソッドに対しては以下の機能が使用できなくなります。

- 実行時間上限監視
- メソッド呼び出し時のイベントジャーナルへのメソッド名の出力
- オペレーション優先度の設定
- オペレーションの統計情報取得
- 運用アシスタント機能
- contps DI X TR コマンド
- オペレーションジャーナル
- スレッド情報取得時の、実行中オペレーション情報採取

しかしながら、割り当てるメソッド数を減らすことにより、モジュールの配備時間が向上し、消費メモリ量が削減されます。

注意:「割り当てない」オプションを使用すると障害発生時の原因解析が困難になる場合があります。頻繁に配備/配備解除を繰り返す開発時には有用ですが、本番環境では「ビジネスメソッドのみ割り当てる」もしくは「全てのリモートメソッドに対して割り当てる」を使用することを強く推奨します。

9.1.2 指定方法

これらのオプションの指定は、EJB モジュールを配備操作する時に行います。以下では、配備操作を支援する機能毎に説明します。

配備ツール

「編集」メニューの「設定の変更」のダイアログ画面から「EJB メソッド識別情報の割り当て方法」で設定します。

統合運用管理ツール

コンポーネント配備ダイアログの「EJB メソッド識別情報の割り当て方法」で設定します。

運用管理コマンド

deploy サブコマンドに「--methodid」オプションを指定します。

--methodid all : 全てのリモートメソッドに対して割り当てる

--methodid limit : ビジネスメソッドのみ割り当てる (既定値)

--methodid none : 割り当てない

オートデプロイ (自動配備)

ドメインの autodeploy ディレクトリ下の autodeploy.ini ファイルで以下の指定をします。

methodid=all : 全てのリモートメソッドに対して割り当てる

methodid=limit : ビジネスメソッドのみ割り当てる (既定値)

methodid=none : 割り当てない

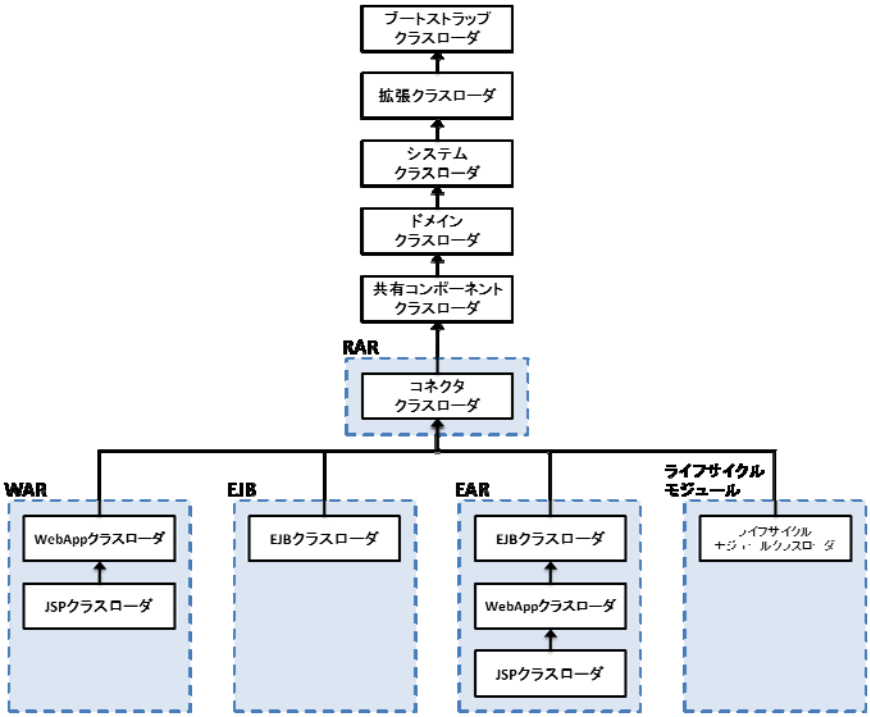
10 クラスローダ

この章では、WebOTX AS が利用するクラスローダについて解説します。クラスローダについての知識は、アプリケーションが共通に利用するクラスファイルや Jar ライブラリの配置場所を決定する際に役立ちます。

10.1 クラスローダの階層

クラスローダは階層構造になっており、この階層構造に従ってクラスをロードします。Java VM 等からクラスのロードの要求があった場合、クラスローダはまず親のクラスローダにクラスのロード処理を委譲します。親のクラスローダでクラスがロードできなかった場合、そのクラスローダ自身でクラスをロードしようとします。それでもクラスが見つからない場合、`NoClassDefFoundError` または `ClassNotFoundException` がスローされます。

WebOTX AS で利用しているクラスローダの階層を以下の図に示します。



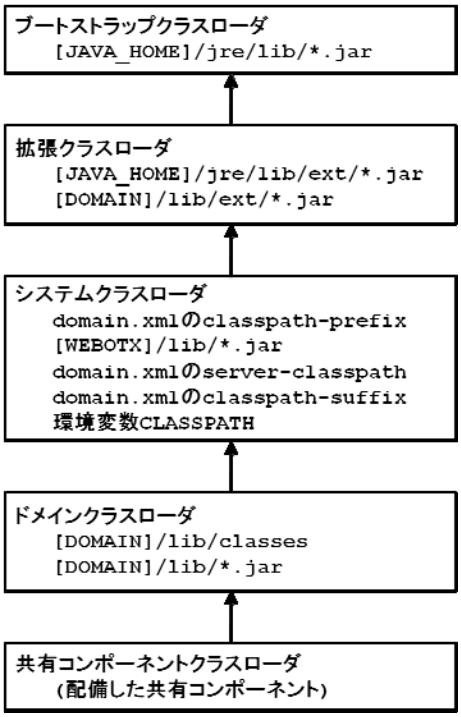
ブートストラップクラスローダからコネクタクラスローダまでは、同一 Java VM 内で共通に利用するクラスローダです。破線で囲まれたクラスローダは、アプリケーションをロードするクラスローダです。WAR、EJB、EAR、ライフサイクルモジュールはクラスローダによって分離されているため、お互い干渉することなく動作することができます。RAR は他のアプリケーションから利用されるという性質上、コネクタクラスローダはドメイン内で共通に利用されます。また、共有コンポーネントクラスローダは、WebOTX AS Standard Edition および Enterprise Edition のみで利用されます。

一般に、下位のクラスローダでロードしたクラスは上位のクラスローダでロードしたクラスを参照できますが、その逆はできません。つまり、上位のクラスローダでロードしたクラスは、下位のクラスローダでロードしたクラスを参照できず、`NoClassDefFoundError` または `ClassNotFoundException` の原因となります。アプリケーションを構築する上で、この原則に従うようにライブラリを配置する必要があります。

10.2 同一Java VM内で共通に利用するクラスローダ

ここでは、ブートストラップクラスローダから共有コンポーネントクラスローダまでを解説します。同一 Java

VM 内で動作するすべてのアプリケーションは、これらのクラスローダを共有します。これらのクラスローダがロードする対象についての概要図を以下の図に示します。



図中の鍵括弧で囲まれた部分は、以下のパスを示しています。

[JAVA_HOME]

JDK のインストール先を示しています。

[WEBOTX]

WebOTX AS のインストール先を示しています。デフォルトのインストール先は、以下の通りです。

Windows	C:¥WebOTX
Unix 系の OS	/opt/WebOTX

[DOMAIN]

WebOTX AS で利用するドメインのパスを示しています。デフォルトのドメインを利用する場合は、、以下のパスになります。

Windows	C:¥WebOTX¥domains¥domain1
Unix 系の OS	/opt/WebOTX/domains/domain1

以降では、これらのクラスローダの詳細を解説します。

10.2.1 ブートストラップクラスローダ

このクラスローダは、Java の標準のクラスライブラリをロードするクラスローダです。ブートストラップクラスローダに親のクラスローダはありません。ブートストラップクラスローダは、以下の Jar ファイルから標準のクラス

ライブラリを読み込みます。

```
[JAVA_HOME]/jre/lib/*.jar
```

このパスは JDK 依存のパスであり、JDK のバージョンや JDK のベンダによって異なる可能性があります。

10.2.2 拡張クラスローダ

このクラスローダは、Java の拡張機能を読み込むクラスローダです。拡張クラスローダの親クラスローダはブートストラップクラスローダです。拡張クラスローダは、以下の Jar ファイルから拡張機能を実装しているクラスライブラリを読み込みます。

```
[JAVA_HOME]/jre/lib/ext/*.jar
[DOMAIN]/lib/ext/*.jar
```

[JAVA_HOME]/jre/lib/ext には、JDK が標準で提供する拡張機能の Jar ライブラリが配置されています。このパスは JDK 依存のパスであり、JDK のバージョンや JDK のベンダによって異なる可能性があります。

[DOMAIN]/lib/ext に Jar ライブラリを配置すると、WebOTX AS に対して拡張機能を提供することができます。アプリケーションで JDBC ドライバを利用する場合、このディレクトリに JDBC ドライバを格納した Jar ライブラリを配置する必要があります。

拡張クラスローダの詳細については、以下の JDK のマニュアルを参照してください。

The Java Extension Mechanism (英語)

```
http://java.sun.com/j2se/1.4.2/docs/guide/extensions
```

Java 拡張機能機構 (日本語)

```
http://sdc.sun.co.jp/java/docs/j2se/1.4/ja/docs/ja/guide/extensions
```

10.2.3 システムクラスローダ

システムクラスローダは、WebOTX AS 本体のクラスをロードします。システムクラスローダの親クラスローダは拡張クラスローダです。WebOTX AS のクラスは以下の場所の Jar ファイルに格納されています。

```
[WEBOTX]/lib/*.jar
```

エージェントプロセスでは、domain.xml の設定を変更することで、ユーザ定義のクラスをシステムクラスローダで読み込ませることができます。ただしこの設定を正しく行うためには WebOTX への深い知識が必要になりますので、通常はドメインクラスローダもしくは拡張クラスローダを利用してユーザ定義のクラスライブラリをロードします。以下に、システムクラスローダでユーザ定義のクラスライブラリをロードするための設定の詳細を解説します。

classpath-prefix

domain.xml 中の要素<java-config>の属性 classpath-prefix で指定したクラスパスは、[WEBOTX]/lib/*.jar よりも先に読み込まれます。そのため WebOTX AS の動作に影響を与える危険性があります。デフォルトではこの属性には何も指定されていません。

server-classpath

domain.xml 中の要素<java-config>の属性 server-classpath で指定したクラスパスは、[WEBOTX]/lib/*.jar の後に読み込まれます。デフォルトではこの属性には何も指定されていません。

classpath-suffix

domain.xml 中の要素<java-config>の属性 classpath-suffix で指定したクラスパスは、要素<java-config>の属性 server-classpath で指定したクラスパスの後に読み込まれます。デフォルトではこの属性には何も指定されていません。

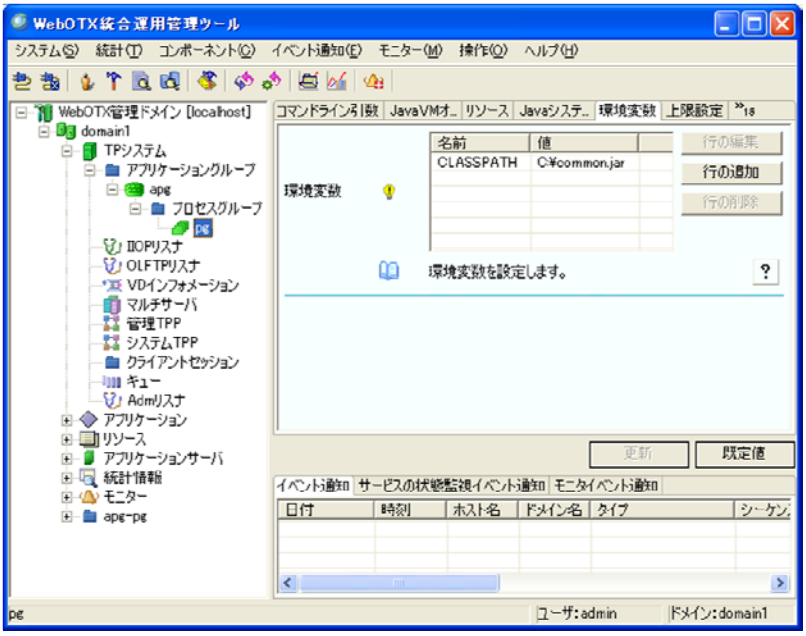
環境変数 **CLASSPATH**

domain.xml 中の要素<java-config>で属性 env-classpath-ignored="false"の指定があった場合、環境変数 CLASSPATH で指定されたクラスパスを読み込みます。環境変数 CLASSPATH は、要素<java-config>の属性 classpath-suffix で指定したクラスパスの後に読み込まれます。デフォルトでは属性 env-classpath-ignored="true"が設定されており、環境変数 CLASSPATH は読み込まれません。

まとめると、エージェントプロセスのシステムクラスローダでは以下の順でクラスパスを検索します。

1. domain.xml 中で属性 classpath-prefix に指定したクラスパス
2. [WEBOTX]/lib/*.jar（WebOTX 本体のライブラリ）
3. domain.xml 中で属性 server-classpath に指定したクラスパス
4. domain.xml 中で属性 classpath-suffix に指定したクラスパス
5. 環境変数 CLASSPATH（env-classpath-ignored="false"の場合）

一方プロセスグループにおいては、domain.xml 中の classpath-suffix 等の設定は反映されません。これらのクラスパスに関する設定は、エージェントプロセスに対するものだからです。プロセスグループでは、そのプロセスグループに対して環境変数 CLASSPATH を設定することで、システムクラスローダにクラスパスを追加することができます。設定の反映には、アプリケーショングループの再起動が必要になります。統合運用管理ツールでの設定場所を、以下の図に示します。



まとめると、プロセスグループのシステムクラスローダは、以下の順でクラスパスを検索します。

1. `[WEBOTX]/lib/*.jar` (WebOTX 本体のライブラリ)
2. 環境変数 CLASSPATH (もし設定されていれば)

10.2.4 ドメインクラスローダ

ドメインクラスローダは、ドメイン内で動作するアプリケーションで共通に利用するクラスをロードします。ドメインクラスローダの親クラスローダはシステムクラスローダです。ドメインクラスローダは、以下の場所からクラスライブラリを読み込みます。

```
[DOMAIN]/lib/classes
[DOMAIN]/lib/*.jar
```

`[DOMAIN]/lib/classes` には、ドメイン内で共通に利用するクラスのクラスファイルを配置します。
`[DOMAIN]/lib` には、ドメイン内で共通に利用するクラスが格納されている Jar ファイルを配置します。

上記の位置に複数のアプリケーションで共通に利用するライブラリを配置すると、WAR ファイルや EAR ファイルの中にライブラリを含める必要がなくなります。ただし、共通するライブラリをバージョンアップ等の理由で置換すると、そのライブラリを利用する全てのアプリケーションが影響を受けます。

10.2.5 共有コンポーネントクラスローダ

このクラスローダは、WebOTX AS Standard Edition および Enterprise Edition でのみ利用可能であり、配備した共有コンポーネントをロードするクラスローダです。共有コンポーネントクラスローダの親クラスローダはドメインクラスローダです。

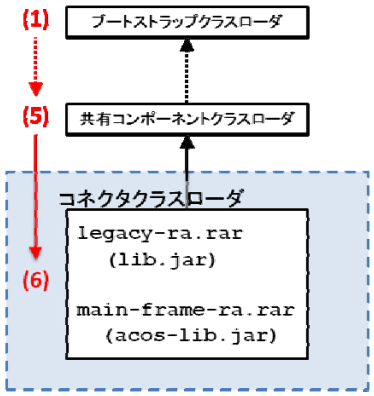
共有コンポーネントの更新を行なう場合は、その共有コンポーネントを利用している全てのアプリケーションを停止すればよく、プロセスグループの再起動は必要ありません。そのため、その他の動作中のアプリケーシ

ョンに影響を及ぼすことなく、ライブラリの更新が可能です。

10.3 コネクタクラスローダ

コネクタクラスローダは JCA に準拠したリソースアダプタを読み込むためのクラスローダです。コネクタクラスローダの親クラスは共有コンポーネントクラスローダです。EAR に含まれるかどうかを問わず、リソースアダプタはコネクタクラスローダでロードされます。リソースアダプタは他のアプリケーションから利用されるという性質上、コネクタクラスローダはドメイン内で共有されます。

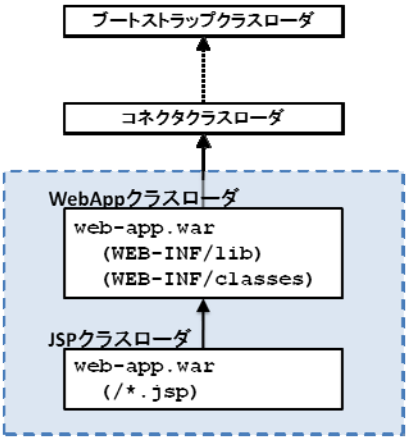
以下の図では、例として二つのリソースアダプタ `legacy-ra.rar` と `main-frame-ra.rar` を配備した状態のコネクタクラスローダの概要図を示します。



最上位のブートストラップクラスローダから最下位のコネクタクラスローダまで、6 つのクラスローダが関係しています。クラスをロードする優先順位は 1 位のブートストラップクラスローダから始まり、コネクタクラスローダは最後の 6 位になります。コネクタクラスローダは `legacy-ra.rar` と `main-frame-ra.rar` を読み込みますが、これらのファイルに含まれる Jar ライブラリもロードの対象になります。

10.4 スタンドアロンのWAR

スタンドアロンの WAR(すなわち EAR に含まれていない WAR)のクラスローダの構成を、下の図に示します。



WAR のクラスローダは、Servlet 用の WebApp クラスローダと JSP 用の JSP クラスローダに分かれています。JSP は実行時に動的に Servlet に変換されるために、このような構造になっています。最上位のブートストラップクラスローダから最下位の JSP クラスローダまで、8 つのクラスローダが関係しています。

Servlet を読み込む WebApp クラスローダは、`/WEB-INF/classes` に配置したクラスファイルと `/WEB-INF/lib` に配置した Jar ライブラリを読み込みます。一方 JSP を読み込む JSP クラスローダは、`/WEB-INF` および`/META-INF` 以外のディレクトリに配置されている JSP を読み込みます。

10.4.1 クラスのロード処理の優先順位の制御

WebApp クラスローダは、クラスのロード処理を親クラスローダに委譲するタイミングを、以下の 2 つから選択できます。

(1) 親クラスローダ優先。まず親クラスローダにクラスのロードを委譲し、見つからなかった場合に WebApp クラスローダでロード。

(2) WebApp クラスローダ優先。まず WebApp クラスローダでクラスのロードを試みて、見つからなかった場合に親クラスローダにクラスのロードを委譲。

「(1)親クラスローダ優先」は他のクラスローダでも採用されている一般的なクラスの探索方法であり、親のクラスローダでロードするクラスを優先的に利用します。(2)は WebApp クラスローダ特有の探索方法であり、WebApp クラスローダがロードするクラスを優先的に利用します。すなわち(2)を採用した場合、Servlet は上位のクラスローダがロードするクラスを WEB-INF/classes または WEB-INF/lib にあるクラスで置換することができます。

ロード処理の優先順位の制御は、WEB-INF/nec-web.xml 中の要素<class-loader>の属性 delegate で指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<nec-web-app>
  <context-root>myapp</context-root>
  <class-loader delegate="true"/>
</nec-web-app>
```

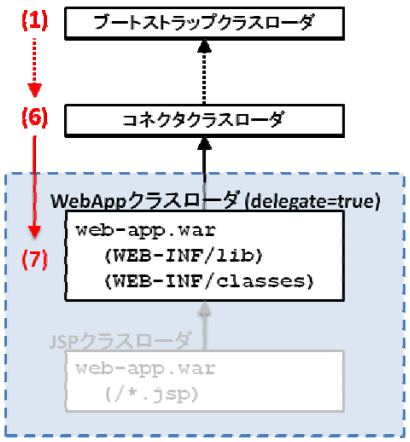
属性 delegate="true"の場合は、「(1) 親クラスローダ優先」となり、属性 delegate="false"の場合は、「(2) WebApp クラスローダ優先」となります。要素<class-loader>を省略した場合や、WEB-INF/nec-web.xml そのものを省略した場合の属性 delegate のデフォルト値は、Servlet のバージョンによって異なります。Servlet 2.4 では属性 delegate のデフォルト値は"true"であり、Servlet 2.3 以前では"false"です。

以降では、属性 delegate の値別に、クラスのロードの具体的な動作を解説します。

10.4.2 属性delegate="true"の場合のクラスローダの動作

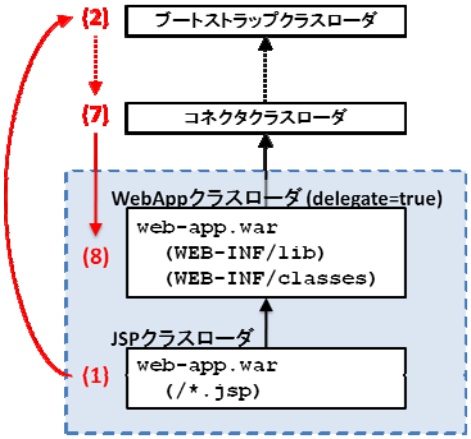
ここでは、属性 delegate="true"の場合のクラスローダの動作について説明します。

まず Servlet の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。Servlet の視点では、JSP クラスローダはクラスのロード処理に関与しない点に注意してください。



WebApp クラスローダが Servlet 等のクラスをロードする際、まず親クラスローダであるコネクタクラスローダにクラスのロード処理を委譲します。コネクタクラスローダを含む上位のクラスローダも、通常どおり先に親クラスローダに処理を委譲します。そのため優先順位の最も高いクラスローダはブートストラップクラスローダであり、最も優先順位の低いクラスローダは WebApp クラスローダで、優先順位 7 位になります。

次に JSP の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。

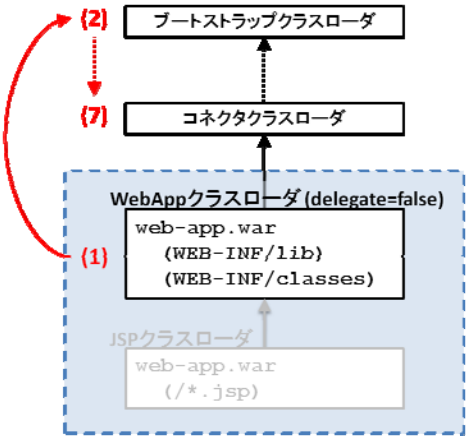


JSP クラスローダは JSP のみをロードする特別なクラスローダです。JSP クラスローダに対してロード要求があった場合、まず JSP クラスローダ自身で JSP のロードを試みます。従って、JSP クラスローダが優先順位 1 位になります。ロード対象が JSP ではない場合、親クラスローダである WebApp クラスローダにクラスのロード処理を委譲します。WebApp クラスローダを含む上位のクラスローダも、通常どおり先に親クラスローダにロード処理を委譲します。そのためブートストラップクラスローダが優先順位 2 位です。最も優先順位の低いクラスローダは優先順位 8 位の WebApp クラスローダです。

10.4.3 属性delegate="false"の場合のクラスローダの動作

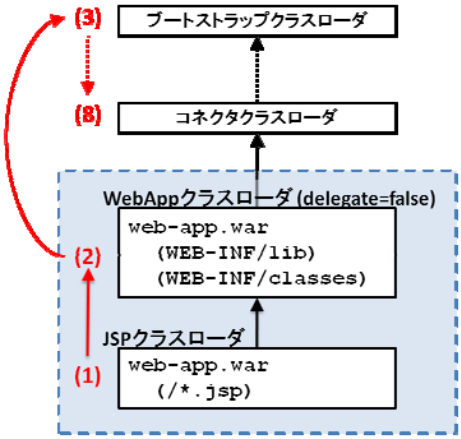
ここでは、属性 delegate="false"の場合のクラスローダの動作について説明します。

まず Servlet の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。Servlet の視点では、JSP クラスローダは無関係である点に注意してください。



WebApp クラスローダが Servlet 等のクラスをロードする際、まず WebApp クラスローダ自身でクラスのロードを試みます。従って、WebApp クラスローダは優先順位 1 位になります。WebApp クラスローダでクラスをロードできなかった場合は、親クラスローダであるコネクタクラスローダにクラスのロード処理を委譲します。コネクタクラスローダを含む上位のクラスローダは、通常どおり先に親クラスローダに処理を委譲します。そのためブートストラップクラスローダが優先順位 2 位となり、コネクタクラスローダは優先順位 7 位になります。

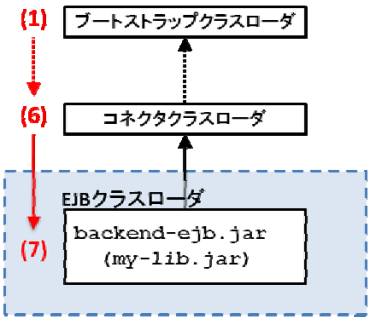
次に JSP の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。



JSP クラスローダは JSP のみをロードする特別なクラスローダです。JSP クラスローダに対してロード要求があった場合、まず JSP クラスローダ自身で JSP のロードを試みます。従って、JSP クラスローダが優先順位 1 位になります。ロード対象が JSP ではない場合、親クラスローダである WebApp クラスローダにクラスのロード処理を委譲します。WebApp クラスローダは親クラスローダにロード処理を委譲する前に、自身でクラスのロードを試みます。従って、WebApp クラスローダは優先順位 2 位になります。WebApp クラスローダでクラスをロードできなかった場合は、親クラスローダであるコネクタクラスローダにクラスのロード処理を委譲します。コネクタクラスローダを含む上位のクラスローダは、通常どおり先に親クラスローダに処理を委譲します。そのためブートストラップクラスローダが優先順位 3 位となり、コネクタクラスローダは優先順位 8 位になります。

10.5 スタンドアロンのEJB

スタンドアロンの EJB(すなわち EAR に含まれていない EJB)のクラスローダの構成を、以下の図に示します。以下の図は、例として `backend-ejb.jar` を配備したときの EJB クラスローダの概要図です。



最上位のブートストラップクラスローダから最下位の EJB クラスローダまで、7 つのクラスローダが関係しています。EJB クラスローダを利用してクラスをロードする場合、まず親クラスローダであるコネクタクラスローダにロード処理を委譲します。コネクタクラスローダを含む上位のクラスローダも、通常どおり先に親クラスローダに処理を委譲します。そのため優先順位の最も高いクラスローダはブートストラップクラスローダであり、最も優先順位の低いクラスローダは EJB クラスローダで、優先順位 7 位になります。

図中の backend-ejb.jar の中には Jar ライブラリ my-lib.jar が含まれています。WebOTX AS の拡張仕様として、EJB のアーカイブが含む Jar ライブラリもロードの対象になります。移植性を高めるためには、EJB のアーカイブの中には Jar ライブラリを含めず、[WEBOTX]/lib などに配置してください。

10.6 EARのクラスローダ

EAR は WAR、EJB、RAR を含むことのできる、「アプリケーションのとりまとめ」のような役割を担っています。そのため、クラスローダの構成も、他のアプリケーションと比べると複雑になっています。ここでは以下のような EAR(app.ear)を例に挙げ、解説を行います。

EAR (app.ear) の構成

1 つの配備記述子(META-INF/application.xml)

2 つの WAR (web-app.war と other-web-app.war)

1 つの EJB (backend-ejb.jar)

1 つの RAR (legacy-ra.rar)

2 つのライブラリ (lib/common.jar と lib/utilities.jar)

クラスローダの委譲の設定 (WAR の/META-INF/nec-web.xml の属性 delegate)

web-app.war は delegate="true"に設定

other-web-app.war は delegate="false"に設定

マニフェストファイルの属性 Class-Path:

web-app.war は、Class-Path: lib/common.jar と設定

backend-ejb.jar は、Class-Path: lib/utilities.jar と設定

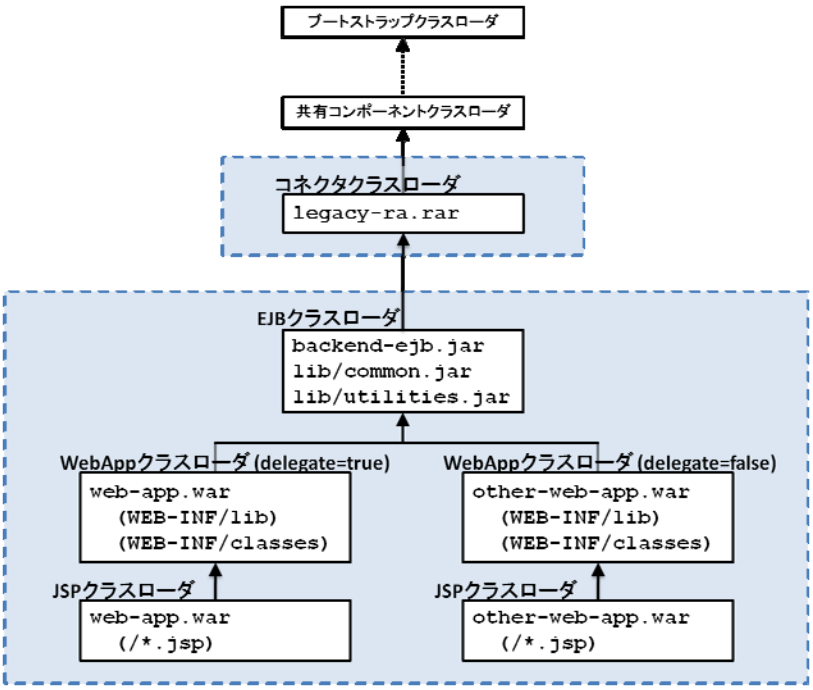
まとめると、app.ear は以下のような構成になります。

```
app.ear
  META-INF/application.xml
  web-app.war              (delegate="true"、Class-Path: lib/common.jar)
```

```
other-web-app.war        (delegate="false")
backend-ejb.jar          (Class-Path: lib/utilities.jar)
legacy-ra.rar
lib/common.jar
lib/utilities.jar
```

10.6.1 クラスローダの構成

app.ear を配備した際のクラスローダの構成を以下の図に示します。



EAR のクラスローダは、以下のクラスローダから構成されます。

EJB クラスローダ

EJB をロードするクラスローダです。EJB クラスローダの親クラスローダはコネクタクラスローダです。EAR 内に含まれる EJB の数とは関係なく、必ず 1 つだけ存在します。EJB クラスローダは、EJB だけではなく、マニフェストファイルの属性 Class-Path で指定したライブラリもロードの対象とします。

WebApp クラスローダ/JSP クラスローダ

WAR をロードするクラスローダです。EAR 内に含まれる WAR の数だけ存在します。WebApp クラスローダの親クラスローダは EJB クラスローダです。WebApp クラスローダと JSP クラスローダは対になっており、JSP クラスローダの親クラスローダは、それと対の WebApp クラスローダです。

コネクタクラスローダ

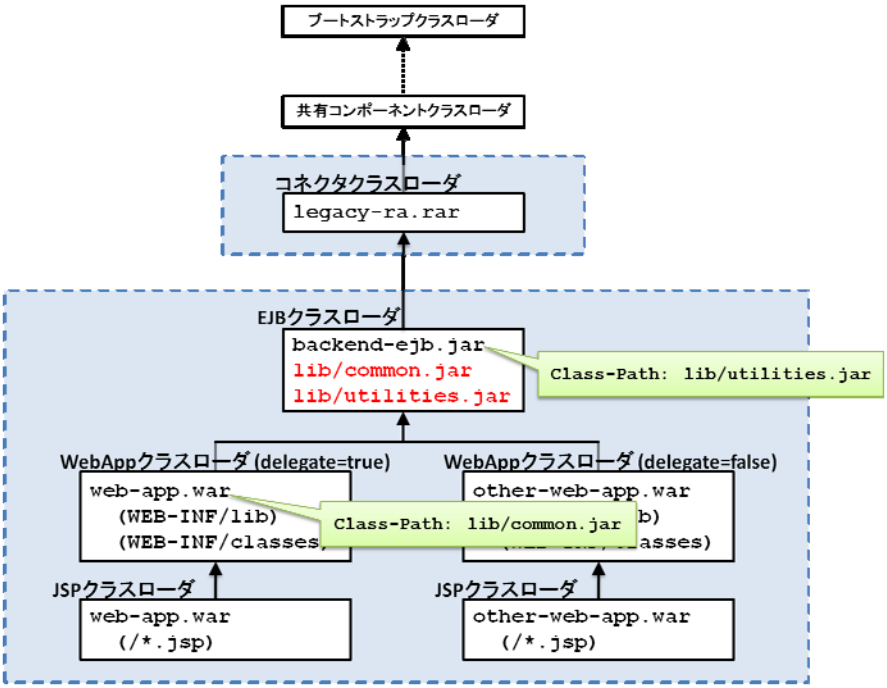
RAR をロードするクラスローダです。ドメイン内で共通のコネクタクラスローダでロードされます。

10.6.2 EAR内で共通に利用するJarライブラリの配置

EAR の中には、WAR や EJB で共通に利用するライブラリを含めることができます。ただし、単に EAR に含めるだけではロードの対象にはなりません。そのライブラリを利用する WAR や EJB のマニフェストファイルの属性 Class-Path にライブラリのパスを記述する必要があります。ライブラリのパスは、EAR ファイルのトップディレクトリからの相対パスでなければいけません。例えば、web-app.war は lib/common.jar を利用するために、web-app.war に次のようなマニフェストファイルを含めます。

```
Manifest-Version: 1.0
Created-By: 1.4.2_18 (Sun Microsystems Inc.)
Class-Path: lib/common.jar
```

マニフェストファイルの属性 Class-Path に着目した app.ear のクラスローダの構成図を以下に示します。web-app.war および backend-ejb.jar のマニフェストファイルに記述したライブラリが EJB クラスローダで読み込まれています。

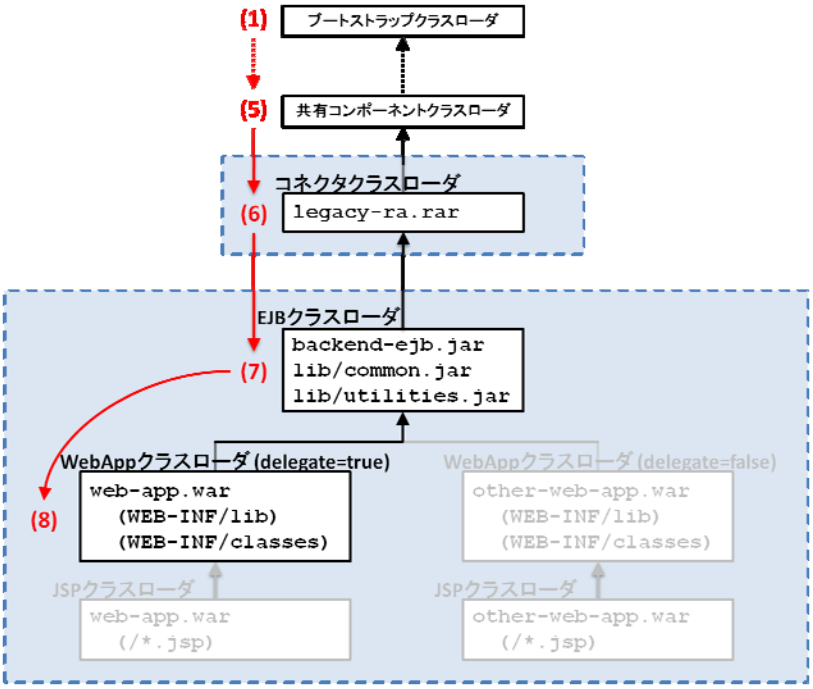


属性 Class-Path の利用には副作用がある点に注意してください。マニフェストファイルの属性 Class-Path に記述したライブラリは EJB クラスローダでロードされるため、その属性を持たないアプリケーションでも利用可能になります。例えば、other-web-app.war はマニフェストファイル中に属性 Class-Path を持ちませんが、親クラスローダである EJB クラスローダが lib/common.jar と lib/utilities.jar をロードしているため、これらのライブラリを利用することができます。

10.6.3 WARからの視点(delegate="true")

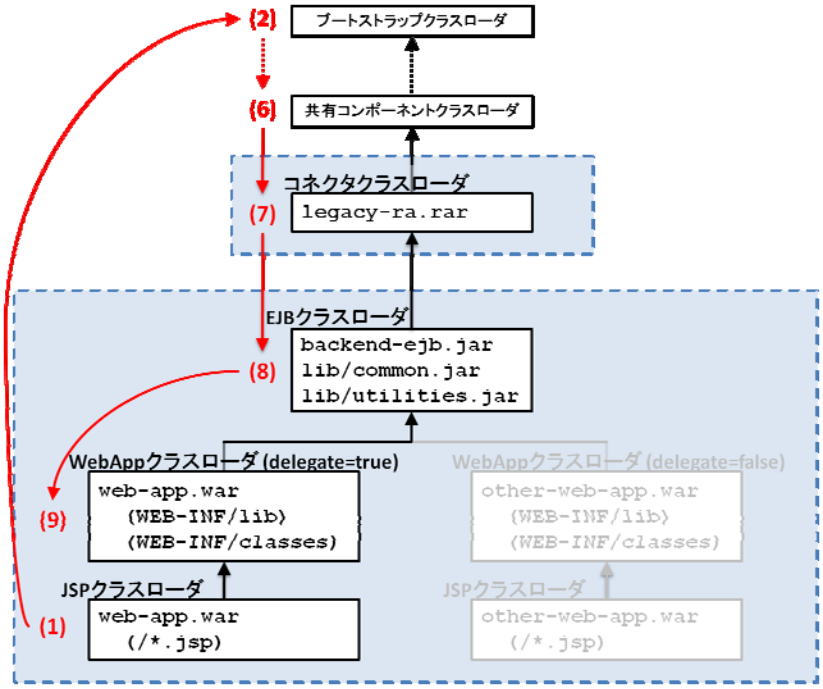
ここでは WEB-INF/nec-web.xml に<class-loader delegate="true"/>を含んでいる web-app.war についてのクラスローダの動作を説明します。すなわち、WebApp クラスローダはクラスのロード処理を親クラスローダに優先的に委譲します。また、もう一つの WAR である other-web-app.war のクラスローダは、クラスのロード処理に関与しない点に注意してください。

まず Servlet の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。Servlet の視点では、JSP クラスローダはクラスのロードに関与しません。



WebApp クラスローダが Servlet 等のクラスをロードする際、まず親クラスローダである EJB クラスローダにクラスのロード処理を委譲します。EJB クラスローダを含む上位のクラスローダも、通常どおり先に親クラスローダに処理を委譲します。そのため優先順位の最も高いクラスローダはブートストラップクラスローダであり、最も優先順位の低いクラスローダは WebApp クラスローダで、優先順位 8 位になります。

次に JSP の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。

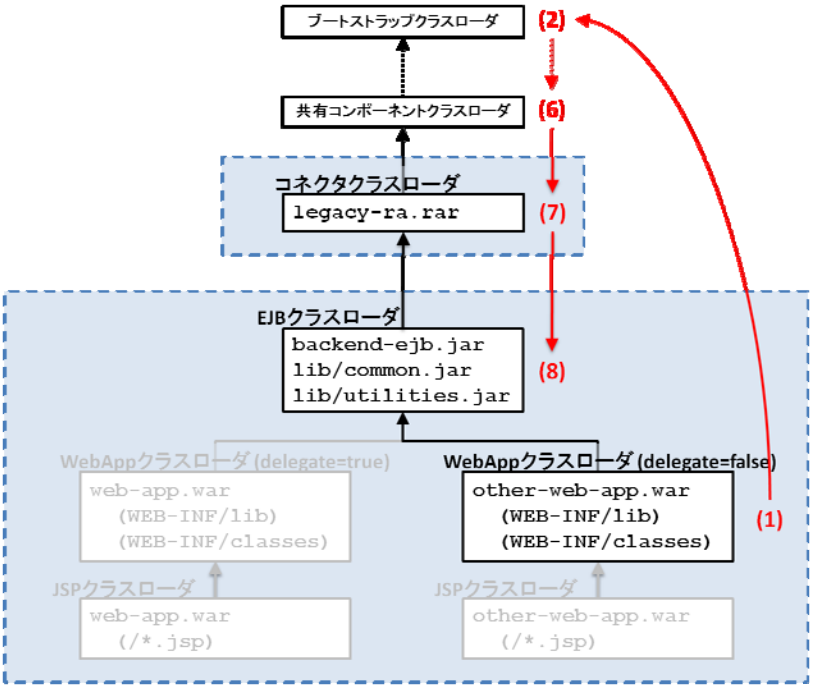


JSP クラスローダは JSP のみをロードする特別なクラスローダです。JSP クラスローダに対してロード要求があった場合、まず JSP クラスローダ自身で JSP のロードを試みます。従って、JSP クラスローダが優先順位 1 位になります。ロード対象が JSP ではない場合、親クラスローダである WebApp クラスローダにクラスのロード処理を委譲します。WebApp クラスローダを含む上位のクラスローダも、通常どおり先に親クラスローダにロード処理を委譲します。そのためブートストラップクラスローダが優先順位 2 位となり、WebApp クラスローダは優先順位 9 位になります。

10.6.4 WARからの視点(delegate="false")

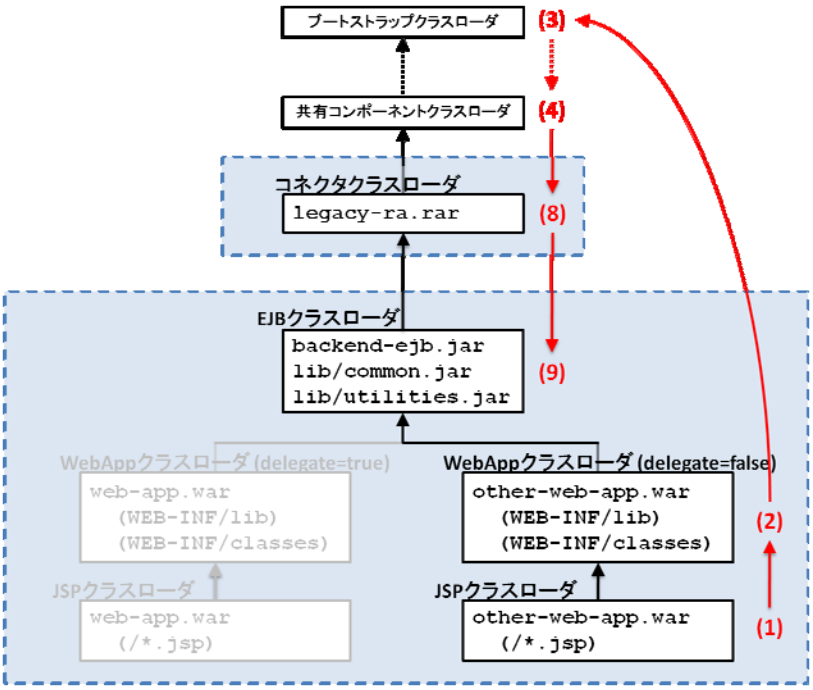
ここでは WEB-INF/nec-web.xml に<class-loader delegate="false"/>を含んでいる other-web-app.war についてのクラスローダの動作を説明します。すなわち、WebApp クラスローダはクラスのロード処理を優先的に自身で行い、クラスが見つからなかった場合に親クラスローダに処理を委譲します。また、もう一つの WAR である web-app.war のクラスローダは、クラスのロード処理に関与しない点に注意してください。

まず Servlet の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。Servlet の視点では、JSP クラスローダはクラスのロードに関与しません。



WebApp クラスローダが Servlet 等のクラスをロードする際、まず WebApp クラスローダ自身でクラスのロードを試みます。従って、WebApp クラスローダは優先順位 1 位になります。WebApp クラスローダでクラスをロードできなかった場合は、親クラスローダである EJB クラスローダにクラスのロード処理を委譲します。EJB クラスローダを含む上位のクラスローダは、通常どおり先に親クラスローダに処理を委譲します。そのためブートストラップクラスローダが優先順位 2 位となり、EJB クラスローダは優先順位 8 位になります。EJB クラスローダでもクラスをロードできなかった場合は WebApp クラスローダに処理が戻りますが、WebApp クラスローダは既にロード処理を試みているために何もせず、従ってクラスがロードできなかったという結果になります。

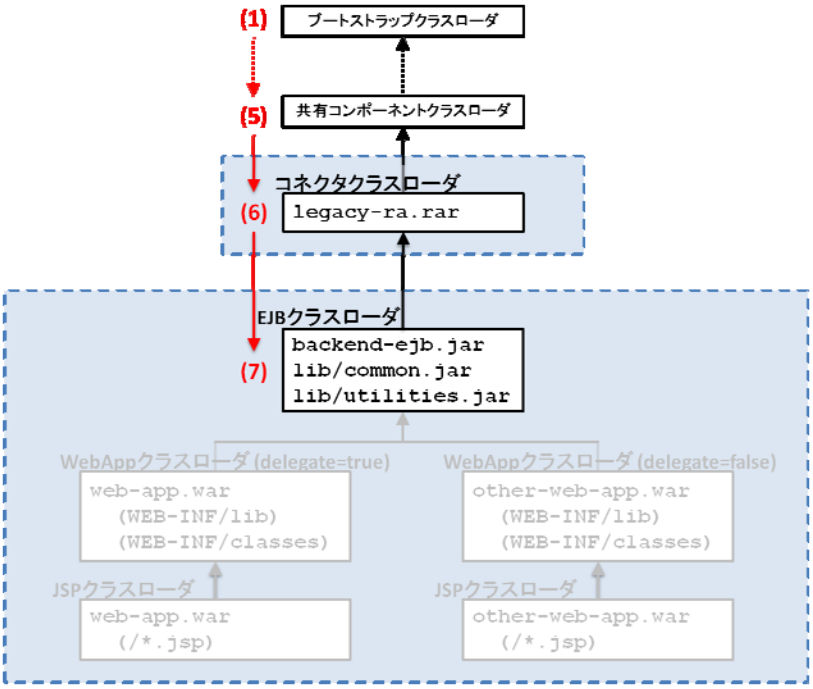
次に JSP の視点から解説します。クラスローダのロード処理順の概要図を以下に示します。



JSP クラスローダは JSP のみをロードする特別なクラスローダです。JSP クラスローダに対してロード要求があった場合、まず JSP クラスローダ自身で JSP のロードを試みます。従って、JSP クラスローダが優先順位 1 位になります。ロード対象が JSP ではない場合、親クラスローダである WebApp クラスローダにクラスのロード処理を委譲します。WebApp クラスローダは親クラスローダにロード処理を委譲する前に、自身でクラスのロードを試みます。従って、WebApp クラスローダは優先順位 2 位になります。WebApp クラスローダでクラスをロードできなかった場合は、親クラスローダである EJB クラスローダにクラスのロード処理を委譲します。EJB クラスローダを含む上位のクラスローダは、通常どおり先に親クラスローダに処理を委譲します。そのためブートストラップクラスローダが優先順位 3 位となり、EJB クラスローダは優先順位 9 位になります。

10.6.5 EJBからの視点

ここでは EJB の視点から EJB クラスローダの動作を説明します EJB クラスローダの動作の概要図を以下に示します。

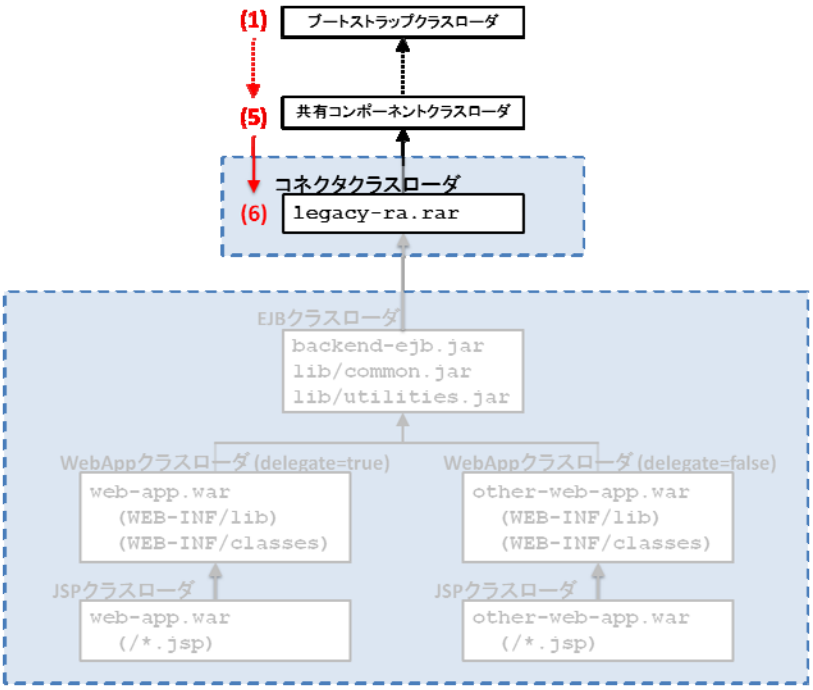


EJB の視点では、WebApp クラスローダと JSP クラスローダはクラスのロード処理に関与しない点に注意してください。EJB クラスローダを利用してクラスをロードする場合、まず親クラスローダであるコネクタクラスローダにロード処理を委譲します。コネクタクラスローダを含む上位のクラスローダも、通常どおり先に親クラスローダに処理を委譲します。そのため優先順位の最も高いクラスローダはブートストラップクラスローダであり、最も優先順位の低いクラスローダは EJB クラスローダで、優先順位 7 位になります。

スタンドアロンの EJB と同様に、EJB のアーカイブの中にある Jar ライブラリも EJB クラスローダでロードされます。これは WebOTX AS の拡張仕様であるため、移植性を高めるためには、EJB のアーカイブの中には Jar ライブラリを含めないでください。

10.6.6 RARからの視点

ここでは RAR の視点からコネクタクラスローダの動作を説明します。コネクタクラスローダの動作の概要図を以下に示します。



RAR の視点では、EJB クラスローダ、WebApp クラスローダ、JSP クラスローダはクラスのロード処理に関与しない点に注意してください。また EJB や WAR の場合とは異なり、RAR はドメイン内で共通に利用されるコネクタクラスローダでロードされます。最上位のブートストラップクラスローダから最下位のコネクタクラスローダまで、6 つのクラスローダが関係しています。クラスをロードする優先順位は 1 位のブートストラップクラスローダから始まり、コネクタクラスローダは最後の 6 位になります。

10.7 ライブラリの配置計画

ドメインクラスローダでロード

最も一般的な、ライブラリの共有方法です。配置したライブラリは、WebOTX AS に配備した全てのアプリケーションで利用できます。ただしライブラリを更新する場合、そのライブラリを利用している全てのアプリケーションが影響を受けるという点に注意する必要があります。ライブラリの置換を行う場合は、WebOTX AS の再起動が必要になります。

アプリケーションのアーカイブに含める

ライブラリを共有せずに、個別にアプリケーションのアーカイブに含めます。他のアプリケーションに影響を与えることなく利用するライブラリを更新できるメリットがあります。ただしライブラリを更新するためには、アプリケーションの再配備が必要になります。

さらにアプリケーションの形式が EAR の場合は、利用するライブラリを EAR 内で共通とするか、WAR 等で個別に持つかを決定する必要があります。EJB の呼び出しで引数や戻り値として利用するクラスは、必ず EAR 内で共通のライブラリとして配置する必要があります。

共有コンポーネントとして配備

WebOTX AS Standard Edition/Enterprise Edition で、プロセスグループ上で動作するアプリケーションで利用可能な方法です。共有コンポーネントは、プロセスグループ上で動作している全てのアプリケーションで利用可能です。共有コンポーネントの更新を行なう場合は、その共有コンポーネントを利用している全てのアプリケーションを停止すればよく、プロセスグループの再起動は必要ありません。

拡張クラスローダでロード

配置したライブラリは、WebOTX AS に配備した全てのアプリケーションで利用できます。拡張クラスローダでライブラリをロードした場合、WebOTX AS 自体のライブラリよりも優先してロードされるため、注意が必要です。JDBC ドライバは拡張クラスローダで読み込む必要があります。拡張クラスローダでロードする Jar ライブラリを更新するためには、ドメインの再起動が必要になります。

システムクラスローダでロード

プロセスグループ毎に環境変数 CLASSPATH でクラスパスを設定可能であるため、プロセスグループ毎に共有するライブラリを分ける場合に利用できます。設定変更後は、アプリケーショングループの再起動が必要になります。

10.7.1 JDBCドライバの配置

JDBCドライバはWebOTX本体をロードしているシステムクラスローダがアクセスできる必要があります。そのため、JDBCドライバは拡張クラスローダかシステムクラスローダで読み込むように配置する必要があります。通常は`[DOMAIN]/lib/ext` に配置します。