BEA Tuxedo®

BEA WebLogic Server での BEA Jolt の使用

BEA Tuxedo 8.1
2003 年 1 月

Copyright

Restricted Rights Legend

Trademarks or Service Marks

目次

このマニュアルについて

このマニュアルでは、以下の内容について説明します。

- 第 1 章「BEA Jolt for BEA WebLogic Server の概要」では、BEA WebLogic Server で使用できる BEA Jolt の主な機能について説明します。
- 第 2 章「Jolt for WebLogic Server の設定」では、Tuxedo と WebLogic Server 間の Jolt セッション・プール接続の設定方法について説明します。
- 第 3 章「Jolt for WebLogic のインプリメント」では、WebLogic アプリケーションまたはサーブレットから Tuxedo に接続するための Jolt の設定方法について説明します。
- 付録 A 「クラス階層」では、BEA Jolt for WebLogic API のクラス階層を示します。
- 付録 B 「簡単なサーブレット・サンプル・プログラム」では、BEA Jolt を使用して WebLogic サーブレットから BEA Tuxedo に接続する方法を示します。
- 付録 C 「Servlet with Enterprise JavaBean サンプル・プログラム」では、Jolt を使用して、Tuxedo サーバに対する EJBean ステートフル・セッションを設定および実行する方法について説明します。

対象読者

このマニュアルは、BEA Tuxedo 製品について理解する必要のあるユーザを対象としています。

e-docs Web サイト

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

マニュアルの印刷方法

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

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

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

関連情報

次の BEA Tuxedo ドキュメントには、BEA Jolt 製品に関連する情報が記載されています。

- *『製品の概要』*

- 『BEA Tuxedo システム入門』
- 『BEA Jolt』

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

サポート情報

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

電子メール メッセージには、BEA Tuxedo 8.1 リリースのマニュアルを使用していることを明記してください。

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

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

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

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

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

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

等幅太字	コード内の重要な単語を示します。 例 : `void **commit** ( )`
等幅イタリック体	コード内の変数を示します。 例 : `String *expr*`

規則	項目
大文字	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
{ }	構文の行で選択肢を示します。かっこは入力しません。
[ ]	構文の行で省略可能な項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	構文の行で、相互に排他的な選択肢を分離します。記号は入力しません。
...	コマンド行で次のいずれかを意味します。 ■ コマンド行で同じ引数を繰り返し指定できること ■ 省略可能な引数が文で省略されていること ■ 追加のパラメータ、値、その他の情報を入力できること 省略符号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。省略符号は入力しません。

1 BEA Jolt for BEA WebLogic Server の概要

BEA Jolt for BEA WebLogic Server を使用すると、Web 対応の BEA Tuxedo サービスを有効にし、BEA WebLogic Server をフロントエンド HTTP およびアプリケーション・サーバとして使用することができます。

BEA Jolt は Java ベースのクライアント API であり、Tuxedo サーバ上で実行されている Jolt サービス・リスナ (JSL: Jolt Service Listener) 経由で、BEA Tuxedo サービスに対する要求を管理します。Jolt API は WebLogic API 内に埋め込まれており、サーブレットまたはほかの BEA WebLogic アプリケーションからアクセス可能です。

BEA Jolt for BEA WebLogic Server は Jolt Java クラス・ライブラリを拡張しているため、WebLogic Server 内で実行される HTTP サーブレットでは、Jolt Java クライアントのクラス・ライブラリを使用することができます。また BEA Jolt for BEA WebLogic Server は、Java HTTP サーブレットを使用して、HTML ブラウザ・クライアントと BEA Tuxedo サービスとの間のインターフェイスを提供します。

以降の説明では、読みやすさを考慮して、BEA Tuxedo を「Tuxedo」、BEA Jolt を「Jolt」、BEA WebLogic を「WebLogic」と表記します。

ここでは、次の内容について説明します。

- 主な特長

- Jolt for WebLogic のしくみ
- サンプル・パッケージの使用方法

主な特長

BEA Jolt for BEA WebLogic Server のアーキテクチャには主に次の特長があります。

- Java HTTP サーブレットを有効にし、Tuxedo アプリケーションに対する動的な HTML フロントエンドを提供
- Tuxedo のリソースを有効活用するためのセッション・プール機能を用意
- トランザクションをサポート
- セッション・プール管理の機能を WebLogic コンソールに統合

注記　Jolt for WebLogic では、非同期の Tuxedo イベント通知へのアクセスは提供されません。

Jolt for WebLogic のしくみ

この節では、Jolt 内での通信で使用する主なコンポーネントと BEA Jolt for BEA WebLogic Server のしくみについて、次の内容を説明します。

- サーバ起動時の接続の初期化方法
- 次の各コンポーネントにおける情報の流れ
  - エンドユーザの Web ブラウザ
  - WebLogic Server
  - Tuxedo トランザクション処理システム

Jolt for WebLogic と Tuxedo の関係

BEA Jolt for BEA WebLogic Server を使用すると、基盤となる Tuxedo システムに Web からアクセスできます。Web アクセスが可能なので、Tuxedo ドメイン内の他のシステムやデータベースと通信できる Web 対応アプリケーションを作成することができます。

ここで説明するシステムには、標準の Web ブラウザからアクセスします。この Web ブラウザに対し、WebLogic Server は、カスタマイズした Java HTTP サーブレットを使用して、ブラウザのインタラクティブな HTTP 要求を処理します。HTTP サーブレットは、受け取った HTTP 要求を処理し、それに対して HTTP 応答を送信する Java クラスです。カスタマイズした HTTP サーブレットは、Jolt for WebLogic API を使用して、リモート・マシン上またはセキュリティ・ファイアフォールの背後に配置された Jolt Server とも通信します。

Jolt Server は Tuxedo ドメイン内に存在するサーバであり、各クライアントに対してアクセスを許可する Tuxedo サービスを決定します。Jolt Server は要求された Tuxedo サービスを呼び出し、結果を WebLogic Server に返します。返された結果は、サーブレットが生成した Web ページにコンパイルし、ブラウザに送信することができます。その際、インターネットまたはイントラネット上のどこからでも Tuxedo サービスにアクセスできる使いやすいインターフェイスを作成します。

Jolt アーキテクチャの主要コンポーネント

WebLogic Java HTTP サーブレットから Jolt Server へ、また、Jolt Server から Tuxedo への通信接続を維持する基本的なオブジェクト・タイプは、以下のとおりです。

- `Session`

  セッション・オブジェクトは、Tuxedo システムとの物理的接続を表します。

- `SessionPool`

セッション・プールには、1 つまたは複数のセッションが含まれています。セッション・プール内のセッションは、効率を高めるために再利用されます。WebLogic サーブレットは、セッションを使用し、セッション・プールのメソッドによって Tuxedo 内のサービスを呼び出します。セッション・プールは、起動時に WebLogic サーバによって初期化され、`config.xml` ファイルの属性によって設定されます。

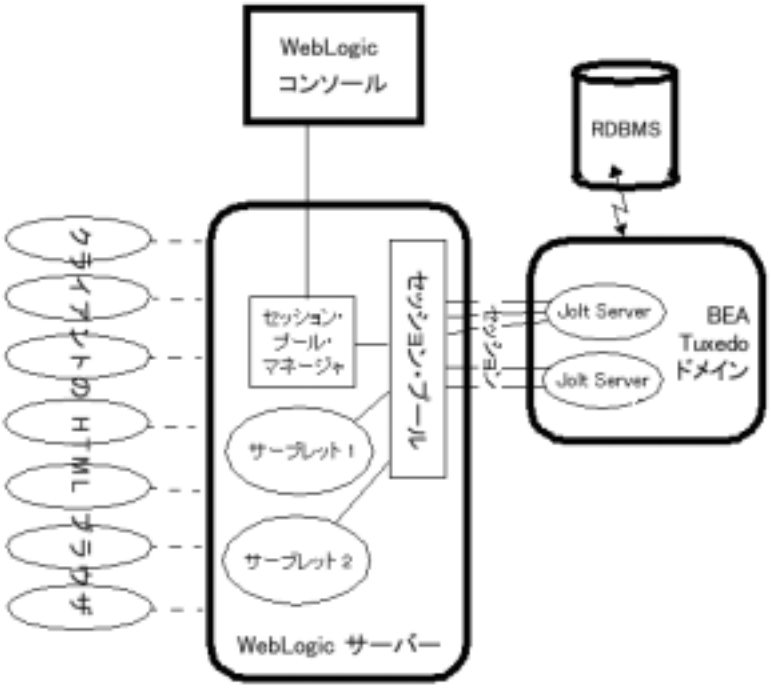
注記　BEA WebLogic Server 6.0 以降では、xml ベースの `config.xml` 環境設定ファイルが `weblogic.properties` ファイルの代わりに使用されます。`config.xml` ファイルの詳細については、『BEA WebLogic Server 管理者ガイド』を参照してください。

- `SessionPoolManager`

  セッション・プール・マネージャを使用して、セッション・プールへの参照を取得し、セッション・プールの作成、管理、および削除を行います。セッション・プール・マネージャは、WebLogic Server が最初のセッション・プールを初期化する直前に作成されます。

図 1-1 は、BEA Jolt for BEA WebLogic Server のアーキテクチャを示します。

図 1-1BEA Jolt for BEA WebLogic Server **のアーキテクチャ**



WebLogic Server の起動

WebLogic 標準に基づくピュア Java アプリケーション・サーバは、分散型 Java アプリケーションのアセンブリ、デプロイ、および管理を行います。このサーバは、Enterprise JavaBeans、Remote Method Invocation (RMI)、分散型 JavaBeans、および Java Database Connect (JDBC) など、分散型コンポーネント・サービスおよびエンタプライズ・データベース・アクセスをサポートしています。

WebLogic Server の管理サーバには、JavaBean に似たオブジェクトである、Sun Microsystem の Java Management Extension (JMX) 標準が設定されています。これらのオブジェクトにより、ドメインの資源に対して管理アクセスを実行できます。

管理サーバには、コンフィグレーション MBean と実行時 MBean があります。MBean は、コンフィグレーション属性への SET (書き込み) アクセスと GET (読み取り) アクセスの両方を提供します。実行時 MBean は、現在の HTTP セッションや JDBC セッション・プールのロードなど、ドメインの資源に関する情報のスナップショットを提供します。ドメイン内の特定の資源 (Jolt 接続プールなど) がインスタンス化されると、その資源に関する情報を収集するための MBean が作成されます。

注記　コンフィグレーション MBean および実行時 MBean の詳細については、『BEA WebLogic Server 管理者ガイド』を参照してください。

WebLogic Server は、起動時に `config.xml` ファイルによって、セッション・プールを初期化するように設定されます。特殊なスタートアップ・クラス `PoolManagerStartUp` が、いくつかのパラメータを設定した WebLogic Server によって呼び出されます。このクラスには、次の機能があります。

- セッション・プール・マネージャがない場合はそれを作成する
- 指定されたパラメータに従ってセッション・プールを作成する
- プール・マネージャに新しいセッション・プールを追加する

注記　Jolt サーバは、セッション・プールを作成する前に起動してください。作成後に起動すると、スタートアップ・クラスは異常終了し、再コミットは行われません。

作成されるセッション・プールの数は、`config.xml` ファイルで設定されている `JoltConnectionPools` の数によって決まります。

クライアント・ブラウザから WebLogic サーバへの接続

WebLogic Server は、別の Java サービスを提供するほか、Java HTTP サーブレットをサポートする本格的な HTTP サーバです。一般に、サーブレットはそれぞれ仮想名で `config.xml` ファイルに登録する必要があります。

サーブレットは、直接呼び出して実際に HTML をブラウザに表示する場合もあれば、ユーザがフォームを送信するときなど、HTML フォームから間接的に呼び出す場合もあります。WebLogic Server は、登録されているサーブレットの仮想名を含む要求を受け取ると、該当するサーブレットの `service()` メソッドを呼び出します。HTTP サーブレットの詳細については、『WebLogic HTTP サーブレット プログラマーズ ガイド S』を参照してください。

HTTP サーブレットの `service()` メソッド (コンテキストに応じて、サーブレットの `doPost()` または `doGet()` メソッドのいずれかを呼び出す) が呼び出されると、ブラウザから送られた HTTP データを含む `HttpServletRequest` オブジェクトが渡されます。1-9 ページの「サンプル・パッケージの使用方法」で説明するサンプル・パッケージでは、Tuxedo に対するトランザクション呼び出しでクライアントのクエリ・データを使用し、応答は新しい HTML ページに組み込まれます。

サーブレットから Tuxedo への接続

まず、サーブレットは、WebLogic Server が起動時に作成し、初期化したセッション・プール・マネージャに対する参照を取得します。セッション・プール・マネージャは、`config.xml` ファイルで設定されたセッション・プールを取得するために使用されます。このセッション・プールは、Tuxedo ドメイン内の適切な Jolt Server を参照します。サーブレットは、セッション・プールを使用して特定の Tuxedo サービスを呼び出します。

Tuxedo サービスが記述され、Jolt リポジトリの Jolt サーバにエクスポート (アクセス可能を宣言) されます。Jolt リポジトリでは、サービス側で予想する入力パラメータと出力パラメータのタイプが宣言されています。サーブ

レット側では、必要な入力パラメータを提供しなければなりません。BEA Jolt for BEA WebLogic Server は、`HttpServletRequest` オブジェクトから直接入力を受け取ることができる特殊な `ServletSessionPool` オブジェクトを使用します。出力データは、`ServletResult` オブジェクトに返されます。

要求が異常終了した場合の処理

セッション・プールは、プール内のセッションに均等に要求を分散します。また、未処理の要求が最も少ないセッションを選択して Tuxedo サービスを呼び出します。選択したセッションが、Tuxedo サービスが呼び出される前に終了した場合、セッション・プールはサービスの呼び出しを別のセッションにリダイレクトし、切断されたセッションに代わる新しいセッションを確立します。セッション・プールはラウンドロビン・アルゴリズムを使用して、プライマリ Jolt サーバへの接続を選択し、確立します。プライマリ Jolt サーバから応答がない場合、セッション・プールはフェイルオーバー・サーバに接続します。

セッション・プールに使用可能なセッションがない場合、またはセッション・プールが中断されている場合は、`SessionPoolException` がスローされます。

複数の要求は、1 つのトランザクションにグループ化されます。トランザクションが異常終了すると、`TransactionException` がスローされます。この例外はサーブレットによってキャッチされ、適切に処理されます (通常、サーブレットはロールバックを実行します)。

クライアント・ブラウザへの応答

サービスの呼び出しが正常に終了すると、次のイベントが発生します。

- 求めていた結果が、`ServletResult` オブジェクトから取り出されます。
- 結果はサーブレットによって処理され、ユーザのブラウザに表示するための HTML ページに組み込まれます。HTML ページは、次の 2 つのうちいずれかの方法で作成できます。

- 標準の HTML ページに Java を埋め込める、WebLogic の使いやすい Java Server Pages (JSP) サービスを使用する
- WebLogic `htmlKona` による、より高度なプログラミング方法を使用する

■ WebLogic Server は、`HttpServletResponse` オブジェクト経由でクライアントに HTML ページを返します。

Jolt サーバとの接続の切断

WebLogic Server は、`config.xml` ファイルで、Tuxedo への既存のセッション・プール接続をシャットダウンするようにも設定されています。

`PoolManagerShutDown` クラスを登録し、WebLogic Server がシャットダウンしたときに、Jolt セッション・プールが正しくクリーンナップされるようにしてください。`PoolManagerShutDown` は、`config.xml` ファイルの属性を必要としません。

サンプル・パッケージの使用方法

BEA Jolt for BEA WebLogic Server には、2 つのサンプル・パッケージが付属しています。これらのパッケージについては、付録 B 「簡単なサーブレット・サンプル・プログラム」および 付録 C「Servlet with Enterprise JavaBean サンプル・プログラム」で説明します。サンプル・パッケージでは、WebLogic サーブレットで Jolt を使用して Tuxedo サービスにアクセスする方法を示します。これらのサンプルを作成、実行、および検証することにより、WebLogic を使用して Tuxedo サービスをインターネットに拡張する方法を判断することができます。

■ 簡単なサーブレット・サンプル・プログラム

FORM ベースの HTML フロントエンドが、HTTP サーブレットに文字列を送信します。サーブレットは、受け取った文字列を Tuxedo サービスに

送信します。返されたデータは、動的に生成された HTML ファイルにコンパイルされ、クライアント・ブラウザに送信されます。

- Servlet with Enterprise JavaBean サンプル・プログラム

  Enterprise JavaBean (EJBean) サンプル・パッケージには、Jolt を使用している Tuxedo サーバへの EJBean ステートフル・セッションを設定し実行するために必要なクラスおよびその他のファイルが含まれています。

2 Jolt for WebLogic Server の設定

Tuxedo と WebLogic Server の間で Jolt セッション・プール接続を設定するには、次の 2 つの手順が必要です。

- Jolt for Tuxedo を設定する
- Jolt for WebLogic Server を設定する

Jolt for Tuxedo の設定

Tuxedo 内で Jolt サービス・リスナ (JSL) を設定する手順については、『BEA Jolt』を参照してください。『BEA Jolt』では、Tuxedo ドメイン内で既に JSL サービスが設定されていることを前提としています。そこでは、WebLogic Server からこれらのサービスへのセッション・プール接続を確立する方法のみを説明します。

Jolt for WebLogic Server の設定

この節では、WebLogic Server と Tuxedo ドメインの JSL の間に BEA Jolt 接続プールを設定する方法について説明します。使用中の WebLogic Server から、JSL を実行しているホストにアクセスしている必要があります。

Jolt スタートアップ・クラスおよび接続プール

WebLogic Server は、起動時または再起動時に必ず `PoolManagerStartUp` クラスを呼び出すように設定する必要があります。この呼び出しにより、次の例に示すように、`config.xml` ファイルを基に Tuxedo へのプール接続が確立されます。

注記　WebLogic Server 6.0 以降の場合、Jolt スタートアップ・クラスおよび接続プールの属性は、Administration Console のコンフィグレーション MBean を介して設定されます。コンフィグレーション MBean および実行時 MBean の詳細については、『BEA WebLogic Server 管理者ガイド』を参照してください。

```
<StartupClass
  ClassName="bea.jolt.pool.servlet.weblogic.PoolManagerStartUp"
  FailureIsFatal="false"
  Name="MyStartup Class"
  Targets="myserver"
/>
<JoltConnectionPool
   ApplicationPassword="tuxedo"
   MaximumPoolSize="5"
   MinimumPoolSize="3"
   Name="MyJolt Connection Pool"
   PrimaryAddresses="//TUXSERVER:6309"
   RecvTimeout="300"
   SecurityContextEnabled="true"
   Targets="myserver"
   UserName="joltuser"
   UserPassword="jolttest"
   UserRole="clt"
/>
```

この例のスタートアップ・クラスでは、WebLogic Server に対し、起動時に PoolManagerStartUp クラスを呼び出すように指示しています。JoltConnectionPool は、PoolManagerStartUp クラスに渡す初期化引数を指定します。

Jolt 接続プールの属性

Jolt 接続プールの属性は、以下のように定義されます。

Application Password	(オプション) Tuxedo アプリケーションのパスワード。この属性は、Tuxedo の認証レベルが USER_AUTH または APP_PW の場合のみ必要です。
MininumPoolSize	(必須) セッション・プール作成時の初期セッション・プール・サイズを指定します。
MaximumPoolSize	(必須) セッション・プールの最大サイズを指定します。プール内の各セッションは、一度に最大 50 の要求を処理できます。
Name	(オプション) ほかのセッション・プールの名前と重複しない、このセッション・プールの名前を定義します。この属性はオプションですが、混乱を避けるため、使用することをお勧めします。SessionPoolManager では、名前のないセッション・プールを 1 つだけ使用することができます。アプリケーションからこの名前のないセッション・プールにアクセスするには、getSessionPool() メソッドの poolname 文字列引数の代わりに null を指定します。 注記　すべてのセッション・プールに名前を付けることをお勧めします。

`PrimaryAdresses`	(必須) Tuxedo システムのプライマリ Jolt サーバ・リスナ (JSL) のアドレス・リストを定義します。このリストは、次の形式で定義されます。 `//hostname:port` *`hostname`* は JSL が実行されているサーバの名前、*`port`* は、JSL が要求をリッスンするように設定されているポートです。セミコロンで区切られたリストを使用して、複数のアドレスを指定することができます。 注記　プライマリ JSL *`hostname:port`* アドレスを少なくとも 1 つ指定する必要があります。
`Failover Addresses`	(オプション) 上記の `appaddrlist` と同じ形式で、フェイルオーバーの Jolt サーバ・リスナのリストを指定できます。上記のプライマリ JSL が異常終了した場合、Jolt はここで指定したフェイルオーバー用 JSL を使用します。これらの JSL は、プライマリ JSL と同じホスト上に存在している必要があります。
`RecvTimeout`	(必須) クライアントが応答を待機する時間を指定します。この時間を過ぎると、タイムアウトが発生します。
`SecurityContext Enabled`	(オプション) この接続プールに対するセキュリティ・コンテキストを有効または無効にします。WebLogic Server と Jolt の間で認証の伝播をインプリメントする場合は、このオプションを有効にしてください。ID の伝播をインプリメントするには、`-a` オプションを指定して、Jolt サービス・ハンドラ (JSH: Jolt Service Handler) を起動する必要があります。このオプションを設定しないと、SecurityContext が有効の場合、JSH はこの要求を受け付けません。 SecurityContext 属性が有効の場合、Jolt クライアントは呼び出し元のユーザ名を JSH に渡します。 JSH は、呼び出し元の ID が付いたメッセージを取得すると、`impersonate_user()` を呼び出してそのユーザの appkey を取得します。JSH は appkey をキャッシュし、呼び出し元が次に要求したときに、appkey をキャッシュから取り出してリクエストがサーバに転送されるようにします。キャッシュは JSH ごとに維持されます。つまり、同じ JSH に接続されたすべてのセッション・プールに対して 1 つのキャッシュが維持されます。

`Targets`	(必須) 接続プールのターゲット・サーバを指定します。
`UserName`	(オプション) Tuxedo ユーザ名。この属性は、Tuxedo の認証レベルが `USER_AUTH` の場合のみ必要です。
`UserPassword`	(オプション) Tuxedo ユーザ・パスワード。この属性は、Tuxedo の認証レベルが `USER_AUTH` の場合のみ必要です。
`UserRole`	(オプション) Tuxedo ユーザ・ロール。この属性は、Tuxedo の認証レベルが `USER_AUTH` または `APP_PW` の場合のみ必要です。

WebLogic Server 上で実行している各アプリケーションに対し、それぞれ 1 つの Jolt セッション・プールを設定することをお勧めします。

Jolt シャットダウン・クラス

WebLogic Server のシャットダウン時に Jolt セッション・プールが Tuxedo から切断されるようにするには、WebLogic Server の `config.xml` ファイルに以下の行を追加します。

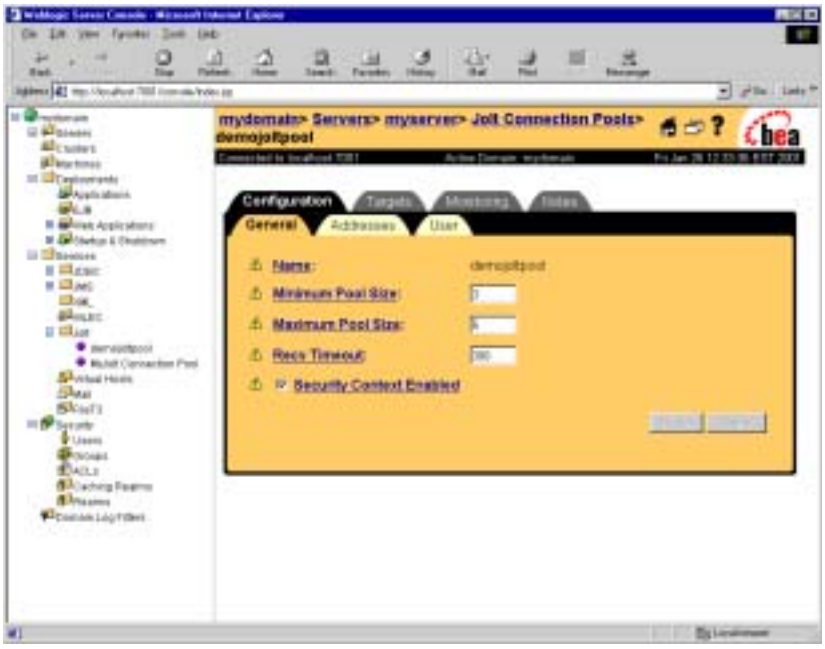
```
<ShutdownClass
  ClassName=”bea.jolt.pool.servlet.weblogic.PoolManager ShutDown.”
/>
```

シャットダウン・クラスは、WebLogic Server がシャットダウンされると `PoolManagerShutDown` クラスを呼び出すように指示します。

WebLogic Administration Console での Jolt の表示

Jolt が正しくインストールされ、設定されている WebLogic Server に接続している場合、Administration Console を起動すると、図 2-1 に示すように、Jolt 接続プールのコンフィグレーション MBean が Administration Console に表示されます。

図 2-1WebLogic Server のコンソールに表示された Jolt 接続プール



各 Jolt 接続プールには、それぞれの `MBean` があり、プール名、最大接続数、プールの状態、および接続ステータスに関する統計情報が表示されます。

注記　MBeans の詳細については、『WebLogic Server 管理者ガイド』を参照してください。

Jolt 接続プールのリセット

WebLogic Server を再起動せずに、Jolt 接続プールをリセットできます。`resetConnectionPool()` メソッドは、`SessionPoolManager.stopSessionPool()` メソッドを呼び出して、プール内

のすべての接続をシャットダウンします。次に、`SessionPoolManager.createSessionPool()` メソッドを呼び出して、接続プールを再起動します。

コマンド行を使用する方法

`resetConnectionPool` メソッドを Admininstration Console コマンド行インターフェイスから呼び出すには、次のコマンドを使用します。

```
java weblogic.Admin -url t3://localhost:7001 -username system
-password gumby1234 -invoke -mbean
mydomain:Name=myserver.jolt.demojoltpool,Type=JoltConnectionPoolR
untime,Location=myserver -method resetConnectionPool
```

Administration Console を使用する方法

Jolt 接続プールは、以下の方法で GUI コンソールからリセットすることもできます。

1. 左側のフレームの [サービス] の下にある Jolt サービス・フォルダをクリックします。
2. 設定済み Jolt 接続プールのうち、監視対象の接続プールをクリックします。
3. 右側のフレームで [モニタ] タブをクリックし、[すべてのアクティブなプールのモニタ] リンクをクリックします。コンソールに、設定済みの接続プールがすべて表示されます。
4. 監視する Jolt 接続プールの横にある [すべてのインスタンスのモニタ] リンクをクリックします。コンソールに、アクティブな Jolt 接続プールが表示されます。

5. 行の最後にある [このプールをリセット] アイコンをクリックし、接続プールをリセットします。

3 Jolt for WebLogic のインプリメント

WebLogic アプリケーションまたはサーブレットから Tuxedo に接続するように Jolt を設定するには、次の手順を実行する必要があります。

- パッケージをインポートする
- セッション・プールを設定する
- サーブレット・セッション・プールにアクセスする
- サーブレット・セッション・プールを使用する
- Jolt を使用して Tuxedo サービスにアクセスする
- Java のデータ型を Tuxedo のデータ型に変換する
- サービスから結果を受信する
- トランザクションを使用する
- 例外の処理

付録 B の 1 ページには、HTTP サーブレットから接続を確立し、Tuxedo サービスにアクセスする方法を示す簡単なサンプル・プログラムがありますので参考にしてください。

パッケージをインポートする

Jolt Java クラス・パッケージは、Jolt for WebLogic Server のインストール時に自動的にインストールされます。BEA Jolt for BEA WebLogic Server を使用するには、Jolt と共にインストールされた次のクラス・パッケージをサーブレットにインポートします。

```
bea.jolt.pool.*
```

```
bea.jolt.pool.servlet.*
```

このほかにも、サーブレットにインポートする必要のあるクラスがあります。Java サーブレットの作成方法については、『WebLogic HTTP サーブレット プログラマーズ ガイド』を参照してください。

セッション・プールを設定する

セッション・プールには `SessionPoolManager` クラスからアクセスできます。WebLogic Server では、サーブレット・セッション・プールと呼ばれるセッション・プールを使用します。サーブレット・セッション・プールによって、HTTP サーブレット内で使用できるさまざまな機能を追加することができます。

WebLogic Administration Console を使用してサーブレット・セッション・プールを設定すると、次の情報が `config.xml` コンフィギュレーション・ファイルに追加されます。

```
<StartupClass
  ClassName="bea.jolt.pool.servlet.weblogic.PoolManagerStartUp"
  FailureIsFatal="false"
  Name="MyStartup Class"
  Targets="myserver"
/>
<JoltConnectionPool
   ApplicationPassword="tuxedo"
   MaximumPoolSize="5"
```

```
   MinimumPoolSize="3"
   Name="MyJolt Connection Pool"
   PrimaryAddresses="//TUXSERVER:6309"
   RecvTimeout="300"
   SecurityContextEnabled="true"
   Targets="myserver"
   UserName="joltuser"
   UserPassword="jolttest"
   UserRole="clt"
/>
```

WebLogic を起動 (または再起動) すると、PoolManagerStartUp クラスと関連する startupArgs が呼び出されます。初回起動時には、PoolManagerStartUp クラスによって ServletSessionPoolManager オブジェクトが作成されます。このオブジェクトには、config.xml コンフィギュレーション・ファイルで設定された ServletSessionPool が含まれます。

以降の呼び出しでは、同じ ServletSessionPoolManager に別の ServletSessionPool が追加されます。前の例に示すように、セッション・プールごとに一意の仮想名をバインディングしてエントリを追加する必要があります。WebLogic Server は、config.xml ファイルの定義に応じて新しい ServletSessionPool を作成します。

プロパティ設定と定義の一覧については、2-2 ページの「Jolt スタートアップ・クラスおよび接続プール」を参照してください。

サーブレット・セッション・プールにアクセスする

WebLogic Server の起動時に Jolt セッション・プールを設定するように指定すると、Java アプリケーションまたはサーブレットから Jolt セッション・プールにアクセスして使用することができます。既に説明したとおり、WebLogic Server ではすべての ServletSessionPool オブジェクトが同じ ServletSessionPoolManager によって管理されます。

```
ServletSessionPoolManager poolMgr = (ServletSessionPoolManager)
  SessionPoolManager.poolmanager;
```

WebLogic Server は、SessionPoolManager から派生した ServletSessionPoolManager クラスを使用します。ServletSessionPoolManager は ServletSessionPool オブジェクトを管理します。このオブジェクトは追加の HTTP サーブレット・メソッドを提供します。

SessionPoolManager は、セッション・プールを管理するためのいくつかのメソッドを提供します。次の例では、SessionPoolManager を使用して、*joltpoolname* という名前の SessionPool を取得します。

```
SessionPool sPool = poolMgr.getSessionPool("joltpoolname");
```

WebLogic Server はサブクラス ServletSessionPoolManager を使用するので、上の例では、SessionPool の代わりに実際には ServletSessionPool オブジェクトが返されます。

次のコード例に示すように、SessionPool を ServletSessionPool にキャストする必要があります。

```
ServletSessionPool ssPool =
  (ServletSessionPool) poolMgr.getSessionPool("joltpoolname");
```

WebLogic Server が ServletSessionPoolManager を作成して設定するので、大部分の場合はこのメソッドだけを使用しますが、ほかの SessionPoolManager メソッドを使用して、個々のセッション・プールまたはすべてのセッション・プールを、作成、中断、停止、またはシャットダウンすることができます。WebLogic の config.xml コンフィギュレーション・ファイルを使用してセッション・プールを設定および管理して、これらの管理タスクを WebLogic Server で処理することをお勧めします。

サーブレット・セッション・プールを使用する

プール・マネージャから指定した ServletSessionPool への参照は、Tuxedo の Jolt Server に対するセッション (または接続) のプールを表します。このプールのサイズと接続先の Tuxedo システムはアプリケーション・コードから抽出され、WebLogic の config.xml コンフィギュレーション・ファイルで定義されます。要求が発行されると、SessionPool は使用可能な接続で、最も負荷の低い接続を使用します。

Tuxedo サービスを呼び出す

Jolt 要求は、通常 SessionPool の call() メソッドを使用する Tuxedo サービスに対する 1 つの呼び出しで構成されています。Tuxedo サービスに名前を付けて、call() メソッドに一連のパラメータを設定すると、Tuxedo サービスから結果が返されます。1 つのトランザクションで複数の呼び出しを行うことにより、サーブレットが Tuxedo アプリケーションのトランザクション要求に対応したり、データベース間の整合性を維持することができます。このトランザクションの詳細については、3-9 ページの「トランザクションを使用する」で説明します。

ServletDataSet を送信する

ServletSessionPool では、オーバーロードした call() メソッドを HTTP サーブレット内で使用できます。これらのメソッドは、HttpServletRequest オブジェクトによって入力パラメータを受け取るので、HTTP サーブレットの doPost() または doGet() メソッドに渡された HttpServletRequest オブジェクトと同じオブジェクトを受け取ることができます。ただし、この場合、HTTP でポストされた *name=value* の組み合わせが、Tuxedo サービスが要求する名前と一致している必要があります。こ

れらのデータは最終的に Java Hashtable に変換されるので、名前の順序は識別されません。HttpServletRequest に含まれるその他のデータによって Tuxedo サービスが中断されることはありません。

Tuxedo サービスは、次のメソッドを使用して HTTP サーブレットから呼び出されます。

```
ssPool.call("serviceName", request);
```

ここで、ssPool は ServletSessionPool に対する参照、*"serviceName"* は呼び出す Tuxedo サービスの名前、*request* 引数はサーブレットに関連付けられた HttpServletRequest オブジェクトを示します。

ServletSessionPool.call() メソッドは、HttpServletRequest を内部で ServletDataSet (通常の SessionPool に送信可能) に変換します。

データ・セットにパラメータを追加する

Tuxedo サービスを呼び出す前に、パラメータ・セットにデータを追加することができます。たとえば、要求の日時を表すパラメータを追加したい場合、このパラメータは HttpServletRequest の FORM データからは取得できません。このような場合、日時を表すパラメータをサーブレットに追加して、追加されたデータ・セットを Tuxedo サービスに送信します。次の例は、この手順を示しています。

```
// 新しいデータ・セットの作成
ServletDataSet dataset = new ServletDataSet();
// HttpServletRequest をデータ・セットにインポート
dataset.importRequest(request);
// 追加のパラメータをデータ・セットに挿入
dataset.setValue("REQUEST_TIME", (new Date()).toString());
// データ・セットを指定したサービスに送信
ssPool.call("service_name", dataset, null);
```

このコード例は、`HttpServletRequest` オブジェクトを `ServletDataSet` オブジェクトに手動で変換する方法を示しています。この新しい形式では、`setValue()` メソッドを使用してパラメータを追加することができます。新しい値は、文字列で表されるキーに関連付けられています。次に、`SessionPool` から継承される `call()` メソッドが呼び出されます。このメソッドは `ServletDataSet` クラスを受け付けますが、トランザクションで使用するその他の引数を要求します。この最後のパラメータに `null` を指定すると、トランザクションを使用して複数のセッション呼び出しをグループ化していないことを示します。詳細については、3-11 ページの「トランザクションを使用する」を参照してください。

Jolt を使用して Tuxedo サービスにアクセスする

Jolt を使用して既存の Tuxedo サービスにアクセスするには、Jolt リポジトリでサービスを定義してエクスポートする必要があります。詳細については、『BEA Jolt』の「Jolt リポジトリ・エディタを使う」および「大量の BEA Tuxedo サービス定義をロードする」を参照してください。Jolt サービス定義では、Tuxedo アプリケーション・サービスによって要求されるパラメータを指定します。

Java のデータ型を Tuxedo のデータ型に変換する

次の表に、Java のデータ型と、Tuxedo サービスで要求されるパラメータの型の対応を示します。`DataSet` または `ServletDataSet` の値には、適切な Java のデータ型を使用してください。Java の `String` 型としてパラメータを指定すると、Jolt リポジトリのサービス定義に応じて適切なデータ型に自動的に変換されます。

要求に関連付けられているすべてのパラメータは文字列形式で表されるので、この機能を使用して、`HttpServletRequest` オブジェクト内のすべてのデータを変換することもできます。それ以外の場合は、下の表に示すデータ型を使用します。適切なデータ型を使用すると、文字列を変換するためのルックアップが不要になるので、効率が向上します。

BEA Tuxedo のデータ型	Java **のデータ型**
char	Byte
short	Short
long	Integer
float	Float
double	Double
char*	String
CARRAY	byte[ ]
XML	byte[ ]

Tuxedo の `CARRAY` 型は、Java 文字列では各バイトが 2 桁の 16 進数で表されます。これらの 16 進数を連結して、複数のバイトを指定します。たとえば、文字列 `FF0A20` は、Tuxedo のデータ型では `CARRAY { 255, 10, 32 }` **で表されます。**

サービスから結果を受信する

ServletSessionPool.call() メソッドは、Tuxedo サービスの結果を含む ServletResult オブジェクトを返します。サービス呼び出しが失敗すると、例外がスローされます。このような例外を迅速に把握し、適切な処理を行う必要があります。発生する可能性のある例外については、『BEA Jolt』の「BEA Jolt の例外」を参照してください。

次の例では、HTTP サーブレットの ServletSessionPool.call() メソッドを使用して ServletResult オブジェクトを取得します。

```
ServletResult sResult = ssPool.call("service_name", request);
```

この例では、ssPool は ServletSessionPool、*request* は HttpServletRequest になります。

ServletSessionPool.call() メソッドによって Result オブジェクトが返されます。このオブジェクトは、ServletResult オブジェクトとしてキャストする必要があります。ServletResult オブジェクトは、Java 文字列でデータを取得するメソッドを提供します。

呼び出しが成功したら、さまざまな形式の getValue() メソッドを使用して Result または ServletResult オブジェクトから個々のパラメータを取り出すことができます。

Result.getValue() メソッドを使用する

Jolt リポジトリの定義に応じて、Tuxedo サービスのパラメータに対応するキーを指定することにより、ServletResult からデータが取り出されます。適切な getValue() メソッドにキーを指定すると、対応する値のオブジェクトが返されます。

また、Result.getValue() メソッドは、キーのルックアップが失敗すると、デフォルト値のオブジェクトを返します。返されたオブジェクトは、Tuxedo サービスの定義に応じて適切なデータ型にキャストする必要があります。たとえば、次のコードでは、

```
Integer answer = (Integer) resultSet.getValue("Age", null);
```

キー "Age" を指定して ServletResult で返された値に整数 answer を設定するか、ServletResult にこのキーが存在しない場合は null を返します。Tuxedo と Java のデータ型の対応については、3-8 ページの「Java のデータ型を Tuxedo のデータ型に変換する」の表を参照してください。

キーに関連付けられた値の配列を指定することができます。この場合、getValue() メソッドによって、そのインスタンスにある配列の最初の要素が返されます。次のコードでは、

```
public Object getValue(String name, int index, Object defVal)
```

このメソッドのシグニチャを使用して、配列内の特定のインデックス指定要素を参照します。

ServletResult.getStringValue() メソッドを使用する

ServletResult は Result の機能を拡張し、追加のメソッドを提供します。

```
public String getStringValue(String name,
                             int index,
                             String defVal)


public String getStringValue(String name,
                             String defVal)
```

これらのメソッドは、Result クラスの getValue() メソッドと同様に動作しますが、該当する値のオブジェクトを Java 文字列型で返します。CARRAY は、3-8 ページの「Java のデータ型を Tuxedo のデータ型に変換する」で説明したように、2 桁の 10 進数の文字列に変換されます。

トランザクションを使用する

トランザクション・オブジェクトを使用して、複数のサービス呼び出しを 1 つのアクションにグループ化し、アプリケーション・ロジック内でデータの整合性を維持することができます。次のメソッドを使用してセッション・プールからトランザクションを取得します。

```
Transaction trans = ssPool.startTransaction(timeout);
```

トランザクション・オブジェクト trans は、トランザクションに対する参照を保持します。ssPool は SessionPool または ServletSessionPool オブジェクトになり、トランザクションの timeout 引数は秒単位で指定します。

トランザクションによってセッションが取得されると、そのトランザクションがコミットまたはアボートされるか、タイムアウトになるまで、ほかのトランザクションはそのセッションを使用できません。ただし、トランザクションには含まれない単独の要求によって使用することはできます。トランザクションがプールからセッションを取得できなかった場合、このメソッドは bea.jolt.pool.TransactionException をスローします。セッション・プールが中断されると、メソッドは bea.jolt.pool.SessionPoolException **をスローします。**

アプリケーションで call() メソッドを使用するごとに、最後のパラメータでトランザクション・オブジェクトを指定する必要があります。次に例を示します。

```
ssPool.call("svcName", request, trans);
```

1 つのトランザクションで複数の呼び出しを行うことができます。この呼び出しは、トランザクション・オブジェクトのメソッドを使用してトランザクションをコミットまたはロールバックするまで完了しません。trans.commit() メソッドによってトランザクションが完了します。このメソッドは、コミットが成功した場合は 0 を返し、失敗した場合は TransactionException をスローします。

トランザクションをアボートする必要がある場合は、Transaction.rollback() メソッドを使用します。このメソッドは、トランザクションのアボートを試みます。アボートが成功した場合は 0 を返し、失敗した場合は TransactionException をスローします。

例外の処理

Jolt による Tuxedo サービスの呼び出し開始時にエラーや障害が発生した場合、Java の例外を使用してアプリケーションに報告されます。常に `try / catch` ブロックに `call()` メソッドを入れ、例外が発生した場合は適切な処理を行う必要があります。`call()` メソッドがスローする可能性のある例外とその理由を以下に示します。

- `bea.jolt.pool.ApplicationException`

  Tuxedo サービスのロジックでエラーが発生すると、この例外がスローされます。たとえば、クライアントが withdrawal サービスを使用して、現在の残高を超える金額を不正に引き出そうとした場合などに発生します。Tuxedo サービスが `TPESVCFAIL` を返すと、`ApplicationException` がスローされます。サービス呼び出し時に返される `Result` オブジェクトに、アプリケーション固有のエラーに関する情報を含めることができます。`Result` オブジェクトには、`ApplicationException.getResult()` メソッドを使用してアクセスできます。

  Jolt では `bea.jolt.ApplicationException` というフル・パッケージ名で別の例外が定義されているので、この例外ではフル・パッケージ名 `bea.jolt.pool.ApplicationException` を使用してください。

- `bea.jolt.JoltException`

  `JoltException` は、以下に示すすべての例外のスーパー・クラスです。これらの例外はすべて、アプリケーション・ロジックに違反するシステム・エラーが発生したことを示します。`JoltException` については、『BEA Jolt』の「BEA Jolt の例外」を参照してください。

- `bea.jolt.pool.SessionPoolException`

  Jolt のセッション・プールで例外が発生すると、この例外がスローされます。たとえば、すべてのセッションがビジー状態の場合や、セッション・プールが中断された場合などに発生します。

- `bea.jolt.ServiceException`

  アプリケーションを含む Tuxedo サービスの起動に関するエラーが発生すると、この例外がスローされます。たとえば、サービスのタイムアウトや、存在しないサービスが呼び出された場合などに発生します。

- `bea.jolt.TransactionException`

トランザクションの起動、コミット、またはアボートに失敗すると、この例外がスローされます。

A クラス階層

BEA WebLogic Server API の BEA Jolt クラス階層

BEA Jolt for BEA WebLogic Server API のクラス階層構造を以下に示します。各クラスとメソッドの詳細については、Java のマニュアルを参照してください。

```
bea.jolt.pool パッケージ
bea.jolt.pool.servlet パッケージ
bea.jolt.pool.servlet.weblogic パッケージ

java.lang.Object クラス
    bea.jolt.pool.Connection クラス
    java.util.Dictionary クラス
        java.util.Hashtable クラス
        (implements java.lang.Cloneable, java.io.Serializable)
          bea.jolt.pool.DataSet クラス
            bea.jolt.pool.Result クラス
              bea.jolt.pool.servlet.ServletResult クラス
             bea.jolt.pool.servlet.ServletDataSet クラス
           bea.jolt.pool.SessionPoolManager クラス
             bea.jolt.pool.servlet.ServletSessionPoolManager クラス
    bea.jolt.pool.Factory クラス
        bea.jolt.pool.SessionPool クラス
          bea.jolt.pool.servlet.ServletSessionPool クラス
    java.lang.Throwable クラス
     (implements java.io.Serializable)
        java.lang.Exception クラス
          java.lang.RuntimeException クラス
```

```
        bea.jolt.pool.ApplicationException クラス
bea.jolt.pool.Transaction クラス
bea.jolt.pool.UserInfo クラス
```

B 簡単なサーブレット・サンプル・プログラム

この例では、BEA Jolt を使用して WebLogic サーブレットから BEA Tuxedo に接続する方法を示します。また、WebLogic Server を使用して、標準的な Web ブラウザに HTML フォームのフロント・エンドを表示します。

ユーザがフォームに入力したテキストは、登録済みの WebLogic HTTP サーブレットで処理される HTTP POST メソッドによって WebLogic Server に返されます。このサーブレットは、BEA Jolt を使用して Tuxedo サービスを呼び出します。サーブレットが受信したテキストは Tuxedo サービスに送信され、サーブレットに返される前に大文字に変換されます。フォームは、サーブレットによって動的に生成される HTML ページにコンパイルされ、Web ブラウザに返されて、オリジナルのテキストが大文字で表示されます。

ここでは、次の内容について説明します。

- サンプル・プログラムの構成要素と前提条件
- サンプル・プログラムを使用する

サンプル・プログラムの構成要素と前提条件

Jolt for WebLogic Server のサンプル・プログラム simpapp は、次の 2 つの要素で構成されています。

- BEA Tuxedo のインストール先ディレクトリである samples ディレクトリのサンプルに組み込まれている HTTP サーブレット。
- BEA Tuxedo と共にインストールされる Tuxedo サンプルに組み込まれた Tuxedo サービス・アプリケーション。Tuxedo の simpapp サーバには、入力された文字列を大文字に変換する *TOUPPER* サービスが含まれます。

Jolt サーブレットのサンプル・プログラム simpapp のソース・コードは、Tuxedo の /samples/jolt/wls/servlet/ ディレクトリにあります。

simpapp サンプル・ディレクトリには、次のファイルが置かれています。

ファイル名	説明
SimpAppServlet.java	Tuxedo に対する呼び出しを発行し、結果の HTML ページを返すサンプル・ソース・コード
simpapp.html	ユーザ入力用の HTML フォーム
simpapp.rep	リポジトリ・バルク・ロード用の REP ファイル
web.xml	Web アプリケーション用のコンフィギュレーション XML ファイル

simpapp アプリケーション・サービスの Tuxedo サーバ側のソース・コードの一覧は、UNIX システムでは $TUXDIR/samples/atmi/simpapp、Windows 2000 システムでは %TUXDIR%\samples\atmi\simpapp (*TUXDIR* は Tuxedo のホーム・ディレクトリ) にあります。

このサンプル・プログラムを実行するには、以下の事項に精通している必要があります。

- BEA Tuxedo のアーキテクチャと simpapp アプリケーション
- BEA Jolt
- HTML
- Java 言語とサーブレット API
- WebLogic Server HTTP サーブレット

サンプル・プログラムを使用する

サンプル・プログラム simpapp は、WebLogic Server から simpapp.html ページを起動するだけで、簡単に実行できます。simpapp.html ページを起動すると、データ入力用のテキスト・フィールドを含む HTML フォームがロードされます。文字列を入力して [Post] ボタンをクリックすると、ポスト要求として文字列が送信されます。SimpAppServlet は、ユーザが入力した文字列を Jolt for WebLogic クラス・ライブラリで使用可能な形式に変換し、Tuxedo の TOUPPER サービスに対する要求をディスパッチします。このサービスは、文字列を大文字に変換してブラウザに表示します。

サーブレット・サンプル・プログラム simpapp を設定するには、次の手順に従います。

- 手順 1. 準備作業を行う
- 手順 2. WebLogic Server を起動する
- 手順 3. WebLogic Server でサーブレットを設定する
- 手順 4. WebLogic Server を停止して再起動する
- 手順 5. サーブレットをコンパイルする
- 手順 6. simpapp.html フォームを表示する
- 手順 7. フォームのデータをブラウザからポストする
- 手順 8. 要求を処理する

- 手順 9. クライアントに結果を返す

手順 1. 準備作業を行う

1. クライアント・マシンにインストールされているブラウザが以下のいずれかであることを確認します。
   - Netscape Communicator 4.7 以上
   - Internet Explorer 5.0 以上
2. クライアント・マシンは、Tuxedo 環境への接続に使用する WebLogic Server にネットワーク接続されている必要があります。
3. Tuxedo とサンプル・プログラム `simpapp` を設定して起動します。
4. サーバ側の simpapp アプリケーションを起動する方法については、Tuxedo のマニュアルを参照してください。`TOUPPER` サービスが使用可能であることを確認します。
5. Jolt サーバ を設定します。Jolt サーバの構成方法に関する情報については、『BEA Jolt』を参照してください。
   - Jolt サーバ・リスナ (JSL) に関連付けられたホスト名とポート番号を書き留めます。
   - Jolt リポジトリのバルク・ローダ・ファイルを使用して、Jolt リポジトリで `TOUPPER` サービスが定義されていることを確認します。

     `simpapp` サンプル・ディレクトリには、`TOUPPER` サービス定義を含む `simpapp.rep` ファイルがあります。システム管理者は、Jolt リポジトリのバルク・ローダを使用して、このサービス定義を Tuxedo サーバ上の既存の Jolt リポジトリに追加する必要があります。Jolt リポジトリのバルク・ローダ・パッケージは、Tuxedo の Jolt コンポーネントに同梱されています。インストール方法の詳細については、『BEA Jolt』を参照してください。

     次のコード例では、Tuxedo サーバ上で Jolt のバルク・ローダを使用して `TOUPPER` サービス定義を追加します。

```
$ java bea.joltadm.jbld //host:port simpapp.rep
```

host と *port* は、Jolt サーバ・リスナ (JSL) のホスト名とポート番号です。simpapp.rep は、BEA Jolt によって提供されるバルク・ローダ・ファイルで、次のいずれかの場所にあります。

$TUXDIR/samples/jolt/wls/servlet/ (UNIX の場合)

%TUXDIR%\samples\jolt\wls\servlet\ (Windows 2000 の場合)

6. インストール時に CLASSPATH が適切に設定されていることを確認します。WebLogic Server のクラス・ライブラリには、このサンプル・プログラムを実行するために必要な次の 3 つの .jar ファイルがあります。
   - jolt.jar
   - joltjse.jar
   - joltwls.jar.

手順 2. WebLogic Server を起動する

Windows 2000 システムを使用している場合、[スタート] メニューから WebLogic Server を起動できます。それ以外の場合は、WebLogic Server のルート・ディレクトリのコマンド行で startWebLogic スクリプトを使用します。

WebLogic Server の起動方法の詳細については、『BEA WebLogic Server 管理者ガイド』の「WebLogic Server の起動と停止」を参照してください。

手順 3. WebLogic Server でサーブレットを設定する

Jolt 接続プールと、WebLogic Server 6.0 以降のスタートアップ・クラスの設定は、Administration Console で行います。

1. WebLogic ドキュメントのルート・ディレクトリに `simpapp.html` ページをコピーします。

   デフォルトでは、WebLogic Server の `\config\mydomain\applications\simpapp` ディレクトリになります。WebLogic に組み込まれた HTTP サーバは、このディレクトリで HTML ページとその他の MIME の種類を検索します。

2. ブラウザに次のアドレスを入力して、WebLogic Server の Administration Console を起動します。

   ```
   http://hostname:listenport#/console
   ```

3. コンソールの左側のフレームで [サービス] フォルダを開き、Jolt フォルダをクリックします。右側のフレームに、ドメイン内で定義されたすべての Jolt 接続プールを示す Jolt [接続プール] テーブルが表示されます。

4. [新しい Jolt Connection Pool のコンフィグレーション] をクリックします。右側のフレームに、新しい接続プールを設定するためのタブ付きのページが表示されます。

5. [一般] タブに次の情報を入力します。

   a. [名前]、[最小プールサイズ]、[最大プールサイズ]、および [タイムアウト] 属性フィールドに値を入力します。

   b. [セキュリティ コンテキストを有効化] チェックボックスをオンにして、セキュリティ・コンテキストを有効にします。これにより、セキュリティ情報が WebLogic Server 環境から Tuxedo 環境に複製転送されます。

   c. [作成] をクリックすると、[名前] フィールドに入力した名前で接続プールのインスタンスが作成されます。左側のフレームの Jolt ノードの下に新しいインスタンスが追加されます。

6. [アドレス] タブと [ユーザ] タブをクリックして確認し、フィールドの値を変更するか、デフォルト値を使用して、[適用] を クリックします。変更内容が保存されます。

7. [対象] タブをクリックして、Jolt 接続プールを起動する使用可能なサーバを選択します。

8. 左側のフレームで、[デプロイメント] フォルダの下にある [起動と停止] フォルダをクリックします。右側のフレームに [起動と停止] テーブルが表示され、ドメイン内で定義されているすべてのスタートアップ・クラスが示されます。
9. [新しい Startup Class のコンフィグレーション] をクリックします。右側のフレームにタブ付きのダイアログ・ボックスが表示されます。次の手順を実行して新しいスタートアップ・クラスを設定します。
   a. [名前]、[クラス名]、および [引数] 属性フィールドに値を入力します。
   b. 障害が発生した場合に WebLogic Server を起動しないようにするには、[失敗したらサーバを起動しない] チェックボックスをオンにします。
   c. [クラス名] に次の名前を入力します。
      `bea.jolt.pool.servlet.weblogic.PoolManagerStartUp`
      このスタートアップ・クラスには引数はありません。
   d. [作成] をクリックすると、[名前] フィールドに入力した名前でスタートアップ・クラスのインスタンスが作成されます。左側のフレームの [起動と停止] フォルダに新しいインスタンスが追加されます。
10. 次の手順を実行して、`simpapp` サーブレットを Web アプリケーションとして登録します。
    a. コンソールの左側のフレームで [デプロイメント] フォルダを開き、[Web アプリケーション] アイコンをクリックします。
    b. [新しい Web アプリケーションをインストール] をクリックし、[アプリケーションのアップロードとインストール] ページを表示します。
    c. 手順 1 で、`simpapp` サーブレットのインストール先としてデフォルトのディレクトリを使用するか、別のディレクトリを選択します。
    d. 手順 2 で、`simpapp` サーブレットへのパスを入力するか、[参照] を使用して選択し、[Upload] ボタンをクリックします。

       `simpapp` サーブレットが WebLogic に Web アプリケーションとして登録され、Deployments\Web Applications フォルダの下にアイコンが表示されます。

手順 4. WebLogic Server を停止して再起動する

Jolt セッション・プールを起動するには、WebLogic Server をシャットダウンして再起動する必要があります。WebLogic Server の再起動方法については、『WebLogic Server 管理者ガイド』の「WebLogic Server の起動と停止」を参照してください。

手順 5. サーブレットをコンパイルする

WebLogic Server を再起動したら、次の手順を実行して `SimpAppServlet` ファイルをコンパイルします。

1. WebLogic ドキュメントのルート・ディレクトリ `\config\mydomain\applications\simpapp` に、新しい `WEB-INF` ディレクトリを作成します。
2. Tuxedo のインストール・ディレクトリ `\samples\jolt\wls\servlet\` から新しい `WEB-INF` ディレクトリに `web.xml` ファイルをコピーします。
3. 次のコードで `SimpAppServlet.java` ファイルをコンパイルします。

```
javac -d %WL.HOME%\config\mydomain\applications\simpapp\WEB-INF\classes
SimpAppServlet.java
```

これにより、必要な Java クラスも `WEB-INF\classes` ディレクトリにコピーされます。

手順 6. simpapp.html フォームを表示する

1. ブラウザを開きます。
2. `simpapp.html` ファイルの URL を入力します。たとえば、デフォルトの URL は次のとおりです。

   `http://`*`localhost:port`*`/simpapp/simpapp.html`

localhost は WebLogic Server のホスト名、*port* は WebLogic Server がログイン要求をリッスンするポートです。

図 B-1 に示すようなページが表示されます。

図 C-1simpapp.html の例

フォームが正しく表示されない場合は、simpapp.html ファイルが WebLogic ドキュメントのルート・ディレクトリにあることを確認してください。

手順 7. フォームのデータをブラウザからポストする

HTML ページのテキスト・フィールドにデータを入力したら、[POST] ボタンをクリックして送信します。入力したデータとともに、その他のパラメータも WebLogic Server で実行されている simpapp サーブレット・クラスに送信されます。

simpapp.html ファイルの HTML フォームに関する部分を以下に示します。

```
<form name="simpapp" action="simpapp" method="post">
<input type="hidden" name="SVCNAME" value="TOUPPER">

<table bgcolor=#dddddd border=1>
<tr>
<td>Type some text here and click the Post button:
<input type="text" name="string">
</td></tr>
```

```
<tr>
<td align=center><input type="submit" value="Post!">
</td></tr>
</table>
</form>
```

この HTML フォームでは、ユーザ入力フィールドと非表示フィールドの 2 つの入力フィールドが指定されています。この例では、非表示フィールドの値により、起動する Tuxedo サービスの名前が指定されます。Tuxedo サービスの名前を HTML ページに入れておくと便利ですが、セキュリティ上の理由から実運用環境では推奨できません。この HTML ページでは、非表示フィールドで別のサービス名を指定する HTTP 要求を送信できます。

注記　Tuxedo サービス名の大文字と小文字は区別されます。

WebLogic Server が HTTP フォーム要求を受信すると、WebLogic Server によって `simpapp` サーブレットの `doPost()` メソッドが起動し、フォームのデータが `HttpServletRequest` に渡されます。

手順 8. 要求を処理する

最初の要求を `simpapp` サーブレットに送信する前に、WebLogic が `init()` メソッドを呼び出してサーブレットを初期化します。次の形式で Jolt セッション・プールが確立されます。

```
ServletSessionPoolManager b_mgr =
 (ServletSessionPoolManager).SessionPoolManager.poolmanager;
```

次に、サーブレットの `doPost()` メソッドが実行されます。このメソッドには、WebLogic Server の起動時に作成された `simpapp` セッション・プールからの接続を取得するコードが含まれます。次に示すコードの抜粋は、`simpapp` セッション・プールの取得方法を示しています。

```
// "simpapp" セッション・プールを取得
ServletSessionPool session =
  (ServletSessionPool) b_mgr.getSessionPool("simpapp");
```

呼び出される Tuxedo サービスは非表示フィールドで識別され、要求オブジェクトから取得されます。次のコードでサービス名パラメータを取得します。

```
String svcnm[] = req.getParameterValues("SVCNAME");
```

単一の値を含む文字配列で [SVCNAME] フィールドの値を取得します。配列の最初の要素だけを使用します。フォームの [SVCNAME] 非表示フィールドに設定された値は TOUPPER です。これは、サーブレットが起動する Tuxedo サービスの名前で、次のコードによって call() メソッドに渡されます。

```
// サービスを起動して結果を取得
result = session.call(svcnm[0], req);
```

この例のセッション・オブジェクト ServletSessionPool は、HttpServletRequest オブジェクトを直接受け付けることができますが、内部的には、TOUPPER サービスのパラメータを含む Jolt DataSet オブジェクトにデータが変換されます。

注記　TOUPPER サービスは、"STRING" という名前のパラメータを要求します。このパラメータは大文字と小文字が識別されるので、HTML フォームのテキスト・フィールドには "STRING" が大文字で入力されていなければなりません。*SVCNAME* など、パラメータとして関連付けられていないその他のデータ・フィールドにも注意が必要ですが、それによって Tuxedo サービスが中断されることはありません。

フォーム・パラメータを使用してサービスに名前を付けると、サービス・パラメータとして渡す必要がありません。サービス名は HttpServletRequest オブジェクトに含まれるので、自動的に渡されます。

TOUPPER サービスは、"STRING" パラメータのテキストを大文字に変換し、実行される呼び出しの結果を含む ServletResult オブジェクトを使用してサーブレットに返します。また、サービス呼び出し時に例外がスローされた場合は、例外の詳細情報も送信します。

手順 9. クライアントに結果を返す

最後の手順では、サービス呼び出しの結果を含む HTML ページを作成して送信し、HttpResponse 出力ストリームによってクライアントに返します。大文字で表示される結果は、`result.getValue()` メソッドを使用して `ServletResult` オブジェクトから取得されます。

次の例は、ブラウザで表示可能な HTML ページでデータを返す簡単な方法を示しています。

```
out.println("<p><center><font size=+1><b>"+
            result.getValue("STRING", "")+
            "</b></font></center><p><hr
            width=80%>");
```

出力ストリームによって、図 B-2 に示すようなページが生成されます。

図 C-2 出力ストリームの結果例

C Servlet with Enterprise JavaBean サンプル・プログラム

サンプル・プログラムの Servlet with Enterprise JavaBean を使用するには、次の節を参照してください。

- Servlet with JavaBean サンプル・プログラムの概要
- Servlet with JavaBean サンプル・プログラムを使用するための準備を行う
- Servlet with JavaBean サンプル・プログラムを実行する

この Enterprise JavaBean (EJBean) サンプル・パッケージには、Jolt を使用する Tuxedo サーバへの EJBean ステートフル・セッションを設定し、実行するために必要なクラスおよびその他のファイルが含まれています。このパッケージのコンポーネントは以下のとおりです。

- クライアント・アプリケーション (クライアント・アプリケーションのマニュアルとソース・コード)
- デプロイメント
  - `DeploymentDescriptor.txt`
  - `manifest`

- インターフェイス
  - `Teller` (リモート・インターフェイスのマニュアルとソース・コード)
  - `TellerHome` (ホーム・インターフェイスのマニュアルとソース・コード)
  - `TellerResult` (アプリケーション固有のユーティリティのマニュアルとソース・コード)
  - `ProcessingErrorException` (アプリケーション固有の例外処理のマニュアルとソース・コード)
  - `TransactionErrorException` (アプリケーション固有の例外処理のマニュアルとソース・コード)
- サーバ (EJBean)
  - `TellerBean` (EJBean のマニュアルとソース・コード)

Servlet with JavaBean サンプル・プログラムの概要

このサンプル・プログラムでは、Enterprise JavaBean (EJBean) の機能を紹介し、Tuxedo サーバにアクセスする簡単なインターフェイスの例を示します。このサンプル・プログラムのソース・コードは、BEA Tuxedo の `/samples/jolt/wls/ejb/bankapp` ディレクトリに収められています。

独自の EJBean を作成する前にこのサンプル・プログラムを実行する場合は手順が異なります。このサンプル・プログラムは、`TellerBean` という名前のステートフル・セッション EJBean です。TellerBean は Jolt for WebLogic を使用して Tuxedo サーバとコンタクトし、以下に示すトランザクションを実行します。

- Tuxedo サーバとコンタクトしてサービスを呼び出し、返された結果を取得する

- セッション EJBean を使用する
- ステートフル・パーシステンスを使用する
- アプリケーション定義の例外とユーティリティを使用する
- クライアント・ブラウザ・アプリケーションを使用する

クライアント・ブラウザ・アプリケーションは次の手順を実行します。

1. JNDI を通して窓口のホーム ("TellerHome") にコンタクトし、EJBean を検索します。
2. 窓口 ("Terry") を作成します。
3. 作成された窓口に対して、アプリケーションは以下の一連のトランザクションを実行します。
   - 口座 10000 の現在の残高を取得します。
   - トランザクション 1: 口座に 100 ドルを預け入れ、残高を表示します。
   - トランザクション 2: 200 ドルの預け入れを行います (トランザクション限度額の 300 ドルを超えます)。

   注記　トランザクション 1 では、単一の呼び出しが行われ、自動的にコミットされます。トランザクション 2 では、`begin()` と `commit()` で 2 つの要求 (預け入れと引き出し) がまとめて処理されます。

   - 口座の残高より 100 ドル多い金額の引き出しを試みます。
   - ApplicationException をキャッチし、例外に埋め込まれたステータス・メッセージを取得して、トランザクション 2 にロールバックします。
   - その口座の最終的な残高を取得します。
   - 窓口を削除します。

   トランザクション 2 では、トランザクション 1 の終了時点の残高にロール・バックする方法が示されています。

Servlet with JavaBean サンプル・プログラムを使用するための準備を行う

このサンプル・プログラムを効果的に利用するには、まずサンプル・コード・ファイルを通読して内容を確認します。`DeploymentDescriptor.txt` を起動して、EJBean の基本構造と、オブジェクトやインターフェイスの種類に応じて使用されるクラスを識別します。また、`Client.java` でアプリケーションの動作を確認します。

以下の節では、このサンプル・プログラムの使用方法について詳しく説明します。

- 環境を設定する
- サンプル・プログラムを作成する
- Servlet with JavaBean サンプル・プログラムを実行する

環境を設定する

「付録 B 簡単なサーブレット・サンプル・プログラム」の「手順 3. WebLogic Server でサーブレットを設定する」に示す手順に従って、BEA の公開 Tuxedo サーバに接続する Jolt 接続プールを追加する必要があります。処理が完了すると、`config.xml` コンフィギュレーション・ファイルに次のセクションが追加されます。

```
<StartupClass
  ClassName="bea.jolt.pool.servlet.weblogic.PoolManagerStartUp"
  FailureIsFatal="false"
  Name="MyStartup Class"
  Targets="myserver"
/>
<JoltConnectionPool
   ApplicationPassword="tuxedo"
   MaximumPoolSize="5"
   MinimumPoolSize="3"
   Name="MyJolt Connection Pool"
   PrimaryAddresses="//TUXSERVER:6309"
   RecvTimeout="300"
   SecurityContextEnabled="true"
   Targets="myserver"
   UserName="joltuser"
   UserPassword="jolttest"
   UserRole="clt"
/>|
<ShutdownClass
  ClassName=”bea.jolt.pool.servlet.weblogic.PoolManager
  ShutDown.”
/>
```

サンプル・プログラムを作成する

WebLogic Server の開発環境を設定したら、サンプル・プログラムを作成する必要があります。BEA Jolt では、Windows 2000 用と UNIX 用のビルド・スクリプトが用意されています。

- Windows 2000: `%TUXDIR%\samples\jolt\wls\ejb\bankapp\build.cmd`

- UNIX:`$TUXDIR/samples/jolt/wls/ejb/bankapp/build.sh`

これらのスクリプトによって、Windows 2000 用のエントリなど、個々のサンプルが作成されます。

```
$ build
```

Microsoft の JDK for Java で作成するには、次のように指定します。

```
$ build -ms
```

スクリプトによってサンプル・プログラムが作成され、Windows 2000 システムでは以下に示すデフォルトの WebLogic Server ディレクトリにファイルがインストールされます。

- クライアント・ファイル : `d:\bea\wlserver6.0\config\examples`
- EJBean:`d:\bea\wlserver6.0\config\mydomain\applications`

Servlet with JavaBean サンプル・プログラムを実行する

デフォルトの `\config\mydomain` ディレクトリで WebLogic Server を起動すると、サンプル・プログラムの EJBean が `\applications` ディレクトリに自動的に配置されます。

1. `\config\mydomain` ディレクトリで WebLogic Server を起動します。
   EJBean が正しく配置されていることを確認するには、サーバのコマンド行ウィンドウをチェックするか、Console を開いて Deployments の下の EJB を調べます。`ejb.jolt.bankapp` がデプロイされ、そのアクティビティが監視可能になっている必要があります。
2. 別のコマンド行ウィンドウを開き、次のコマンドを入力してクライアントを実行します。

   ```
   $ java examples.jolt.ejb.bankapp.Client
   ```

WebLogic Server をデフォルト設定で実行していない場合、次のコマンド行を使用する必要があります。

```
$ java examples.jolt.ejb.bankapp.Client "t3://WebLogicURL:Port"
```

パラメータの定義は次のとおりです。

- WebLogicURL―WebLogic Server のドメイン・アドレス
- Port―接続をリッスンするポート (`weblogic.system.ListenPort`)

次のオプション・パラメータは、クライアントによって上から順に解釈されます。

- url―サーバの URL (`t3://localhost:7001` など)。
- user―ユーザ名。デフォルトは NULL。
- password―ユーザ・パスワード。デフォルトは NULL。

3. `Client` サンプルを実行している場合、クライアント・アプリケーションから次のような出力が返されます。

```
4.Beginning jolt.bankapp.Client...
5.
6.Created teller Terry
7.
8.Getting current balance of Account 10000 for Erin
9.Balance: 27924.02
10.
11.Start Transaction 1 for Erin
12.
13.  Depositing 100.0 for Erin
14.  Balance: 28024.02
15.
16.End Transaction 1 for Erin
17.
18.Start Transaction 2 for Erin
19.
20.  Depositing 200.0 for Erin
21.  Balance: 28224.02
22.
23.  Withdrawing 28324.02 for Erin
24.  Transaction error:
25.  examples.jolt.ejb.bankapp.TransactionErrorException:Teller
error: application
26. exception:
27.Account Overdraft
28.
```

```
29.  Rolling back transaction for Erin
30.
31.End Transaction 2 for Erin
32.
33.Getting final balance of Account 10000 for Erin
34.Balance: 28024.02
35.
36.Removing teller Terry
37.
End jolt.bankapp.Client...
```

注記　*トランザクション2*が*トランザクション1*の終了時点の残高にロールバックされ、最終的な残高が表示されていることに注意してください。

EJB の詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』を参照してください。BEA Jolt の使用方法については、『BEA Jolt』を参照してください。