BEA Tuxedo®

BEA Jolt

BEA Tuxedo 8.1
2003 年 1 月

Copyright

Restricted Rights Legend

Trademarks or Service Marks

目次

3. BEA Jolt システムの設定

4. Jolt リポジトリ・エディタを使う

5. Jolt クラス・ライブラリを使う

6. JoltBeans を使う

7. Servlet Connectivity for BEA Tuxedo を使う

8. Jolt ASP Connectivity for BEA Tuxedo を使う

A. BEA Jolt の例外

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 入門」では、BEA Jolt のコンポーネントと主な機能について説明します。
- 第 2 章「大量の BEA Tuxedo サービス定義をロードする」では、Jolt バルク・ローダについて説明します。バルク・ローダは、複数の Tuxedo サービスを Jolt リポジトリ・データベースに一度にロードするためのコマンド・ユーティリティです。
- 第 3 章「BEA Jolt システムの設定」では、BEA Jolt の設定方法について説明します。
- 第 4 章「Jolt リポジトリ・エディタを使う」では、Jolt リポジトリ・エディタを使用して、Jolt リポジトリの BEA Tuxedo サービスを追加、変更、テスト、エクスポート、および削除する方法について説明します。
- 第 5 章「Jolt クラス・ライブラリを使う」では、BEA Jolt クラス・ライブラリについて説明します。このクラス・ライブラリは、Java アプレットから BEA Tuxedo サービスにアクセスするための、オブジェクト指向の Java 言語クラスを開発者に提供します。
- 第 6 章「JoltBeans を使う」では、JoltBeans を使用した BEA Jolt クライアントの開発方法について説明します。JoltBeans には、BEA Jolt 用の JavaBeans 対応インターフェイスが用意されています。
- 第 7 章「Servlet Connectivity for BEA Tuxedo を使う」では、BEA Jolt のサーブレット・コネクティビティについて説明します。この機能を利用すると、HTTP サーブレットを使用して、HTTP リクエストに対するサーバ・サイドの Java タスクを実行できます。この機能により、さまざ

まな HTML クライアントがリモートの BEA Tuxedo サービスのリクエストを作成できるようになります。

- 第 8 章「Jolt ASP Connectivity for BEA Tuxedo を使う」では、Jolt ASP (Active Server Pages) Connectivity for BEA Tuxedo について説明します。この機能は、動的な HTML ページを処理および生成して Tuxedo サービスにアクセスするための使いやすいインターフェイスを提供します。
- 付録 A「BEA Jolt の例外」では、BEA Jolt の実行中に発生する可能性のある BEA Jolt および BEA Tuxedo の例外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 は、Adobe 社の Web サイト (`http://www.adobe.co.jp/`) から入手できます。

関連情報

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

- 『BEA Tuxedo システム入門』
- 『Windows での BEA Tuxedo の使用』
- 『BEA Tuxedo CORBA アプリケーション入門』

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 入門

BEA Jolt は、BEA Tuxedo システム用の Java ベースのインターフェイスです。既存の BEA Tuxedo アプリケーションの機能が拡張されており、イントラネットやインターネットにも対応できます。Jolt を使用すると、通常の Web ブラウザから BEA Tuxedo サービスにアクセスできるように BEA Tuxedo アプリケーションを変更することができ、さらにこれを簡単に行うことができます。また、Jolt は既存および新規の BEA Tuxedo アプリケーションや BEA Tuxedo サービスを扱い、クライアント / サーバ間のイントラネット・トランザクションおよびインターネット・トランザクションを安全かつスケーラブルに処理します。さらに、Jolt を使用して、既存の BEA Tuxedo サービスをリモートから呼び出すことができるクライアント・アプリケーションやアプレットを作成し、アプリケーション・メッセージング機能、コンポーネント管理、および分散トランザクション処理を実現することができます。

アプリケーションの開発は、BEA Tuxedo と Java プログラミング言語を使用する Jolt API および Jolt リポジトリ・エディタで行います。したがって、このマニュアルは、BEA Tuxedo と Java のプログラミングに精通した読者を対象としています。このマニュアルは、システム管理者、ネットワーク管理者および開発者を対象としています。

ここでは、次の内容について説明します。

- BEA Jolt コンポーネント
- 主な特徴
- BEA Jolt の仕組み
- Jolt クライアントを作成して BEA Tuxedo アプリケーションにアクセスする

BEA Jolt コンポーネント

BEA Jolt は、リモートの Java クライアントが BEA Tuxedo にアクセスするためのインターフェイスを提供する Java クラス・ライブラリと API です。BEA Jolt は、BEA Tuxedo サービスにアクセスするための Java ベースのクライアント・プログラムを作成する以下のコンポーネントで構成されています。

- Jolt サーバークライアントからのネットワーク接続の受け付け、Jolt メッセージの変換、複数のクライアントの単一プロセスへの多重化、1 つまたは複数の BEA Tuxedo サーバで実行される BEA Tuxedo アプリケーションとの間でのリクエストの送受信を行います。
- Jolt クラス・ライブラリー Jolt クラス・ライブラリは、Jolt API を実装するクラス・ファイルを収めた Java パッケージです。これらのクラスによって、Java アプリケーションおよびアプレットは BEA Tuxedo サービスを呼び出すことができるようになります。Jolt クラス・ライブラリには、通信属性、通知、ネットワーク接続、トランザクション、およびサービスについての設定、検索、管理および呼び出しを行う機能が含まれています。
- JoltBeans ー BEA JoltBeans には、BEA Jolt 用の JavaBeans 対応インターフェイスが用意されています。JoltBeans は、BEA Jolt クライアントを作成するため、JavaBeans 対応の統合開発環境 (IDE : Integrated Development Environment) で使用される Beans コンポーネントです。JoltBeans は、JoltBeans ツールキットと Jolt GUI Bean の 2 種類の Java Beans のセットで構成されています。JoltBeans ツールキットは BEA Jolt 用の JavaBeans 対応インターフェイスであり、JoltServiceBean、JoltSessionBean、JoltUserEventBean が含まれています。Jolt GUI Bean は、Jolt 対応 AWT (Abstract Window Toolkit) Bean と Jolt 対応 Swing Bean で構成されています。
- Jolt リポジトリーリポジトリには BEA Tuxedo サービスの定義が収められています。これらのサービス定義情報は、Jolt が BEA Tuxedo サービスにアクセスする時点で利用されます。また、Jolt クライアント・アプリケーションへサービスをエクスポートしたり、Jolt クライアントから

定義を見えなくすることによってサービスをアンエクスポートすることができます。リポジトリ・エディタを使うと、クライアント・アプリケーションからは独立して、新規または既存の BEA Tuxedo サービスをテストできます。

- Jolt インターネット・リレー― Jolt インターネット・リレーは Jolt クライアントからのメッセージを Jolt サーバ・リスナ (JSL) または Jolt サーバ・ハンドラ (JSH) にルーティングするコンポーネントです。このコンポーネントにより、JSH と BEA Tuxedo を Web サーバと同じマシンで実行する必要がなくなります。Jolt インターネット・リレーは Jolt リレー (JRLY) と Jolt リレー・アダプタ (JRAD) から構成されます。

主な特徴

BEA Jolt により、既存の BEA Tuxedo サービスを利用して、トランザクション環境を企業のイントラネットや世界中のインターネットにまで拡張できます。Jolt アーキテクチャの特徴は、その簡便さにあります。Jolt を使うと、インターネット上で動く、堅牢でモジュール化され、スケーラブルな電子商業システムを構築、運用、管理することができます。

BEA Jolt には以下のような特徴があります。

- 開発を容易にする Java に基づく API ― Java に基づいた API により、優れた設計のオブジェクト・インターフェイスを提供することで、BEA Jolt はアプリケーションの設計を簡単にします。Jolt は SDK (Java 2 Software Development Kit) をサポートし、Java スレッドとは完全に互換性があります。Jolt により、Java プログラマは、詳細なトランザクション情報を理解したり、既存の BEA Tuxedo アプリケーションを書き直したりしなくても、BEA Tuxedo のアプリケーション・サービスやトランザクション・サービスを使用するグラフィカルなフロントエンドを構築することができます。
- 純粋な Java クライアントの開発― Java 対応ブラウザであればどんなブラウザでも実行できる純粋な Java クライアントを構築することができます。Jolt は、Java のデータ型を BEA Tuxedo のネイティブなデータ型およびバッファに自動的に変換します。また、その逆の変換も自動的に行

います。純粋な Java クライアントであるので、作成されたアプレットまたはアプリケーションはクライアント側に常駐のライブラリやインストールを必要としません。そのため、クライアント・アプリケーションはネットワーク経由でダウンロードできます。

- Jolt リポジトリによる BEA Tuxedo サービスへの容易なアクセスー BEA Jolt のリポジトリにより、Java クライアント側から利用する BEA Tuxedo サービスの定義が管理されるため、Java アプリケーションの開発が容易になります。Jolt リポジトリのバルク・ロード・ユーティリティを使うと、既存の BEA Tuxedo サービスを Jolt 開発環境にすばやく統合することができます。Jolt と BEA Tuxedo は、ネットワークとアプリケーションの拡張性の確保を簡単にする一方、アプリケーション・コンポーネントの再利用を促進します。
- GUI ベースでの BEA Tuxedo サービスの管理と配布ー Jolt リポジトリ・エディタを使って、サービス名や入出力などの BEA Tuxedo サービスの定義を管理することができます。Jolt リポジトリ・エディタでは、Jolt リポジトリで定義されたサービスに対する異なる入出力名の使用をサポートします。
- トランザクション処理を安全に行うための暗号化ー BEA Jolt では、Jolt クライアントと JSL/JSH の間で送受信されるデータを暗号化することができます。Jolt の暗号化により、インターネット・トランザクション処理の安全性が向上します。
- インターネット・リレーによるセキュリティ機能の追加ー BEA Jolt で提供されているインターネット・リレーを使うと、ネットワーク管理者は Web サーバと BEA Tuxedo アプリケーション・サーバを分離することができます。Web サーバは、企業のファイアウォール外にあることが多いため、一般的には安全性が低いと考えられています。Jolt インターネット・リレーにより、Jolt クライアント側からのトランザクション処理をインターネット上で実現しつつ、BEA Tuxedo サーバをネットワーク上の安全な場所や環境に配置できるようになりました。
- イベント・サブスクリプションのサポートー Jolt のイベント・サブスクリプションを使って、BEA Tuxedo サービスや他の BEA Tuxedo クライアントからイベント通知を受け取ることができます。Jolt イベント・サブスクリプションで扱うことのできる BEA Tuxedo アプリケーション・イベントは次の 2 種類です。

- 任意通知型イベント通知ー Jolt クライアントがこれらの通知を受信するのは、Jolt クライアントが任意通知型メッセージ・イベントをサブスクライブしており、BEA Tuxedo クライアントがブロードキャスト・メッセージまたは直接 Jolt クライアントに宛てたメッセージを送信するときです。
- ブローカ経由のイベント通知ー Jolt クライアントは、BEA Tuxedo イベント・ブローカ経由でこれらの通知を受信します。Jolt クライアントがこれらの通知を受信するのは、それがあるイベントをサブスクライブしており、任意の BEA Tuxedo クライアントまたはサーバがイベントを通知するときに限られます。

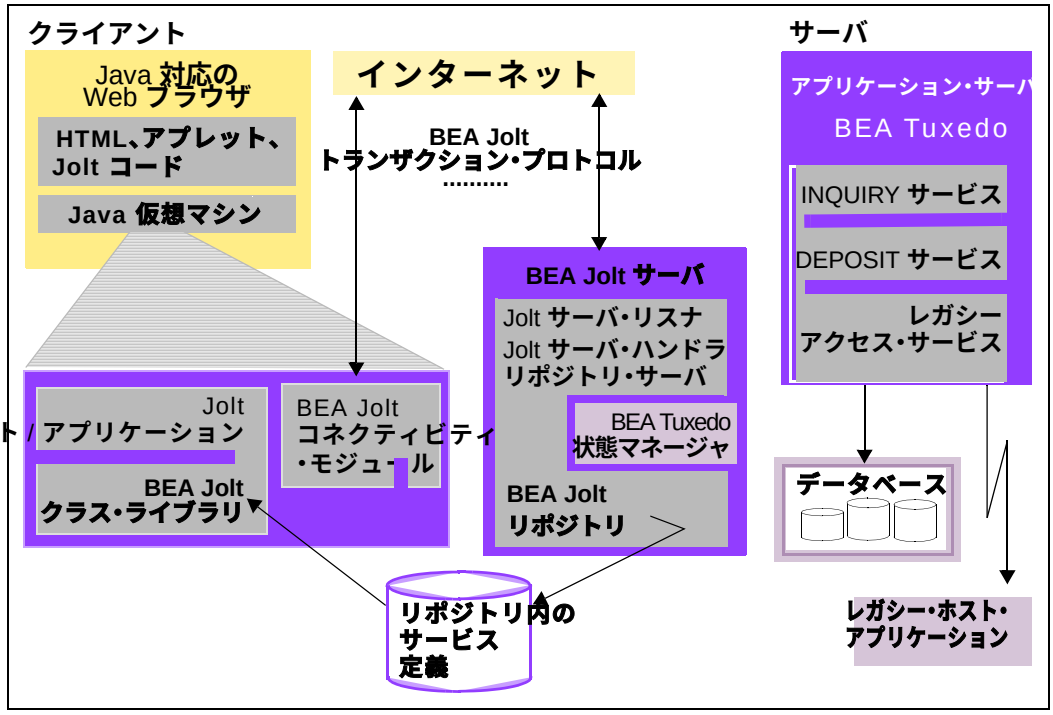
BEA Jolt の仕組み

BEA Jolt は、Java クライアントを BEA Tuxedo を使って構築されたアプリケーションに接続します。BEA Tuxedo はモジュール化されたサービス一式を提供し、各サービスはアプリケーション全体に関連した特定の機能を提供します。

1-7 ページの「BEA Jolt のアーキテクチャ」は、BEA Jolt のアーキテクチャ全体の様子および関連する BEA Tuxedo コンポーネントとそれらの相互作用を表示したものです。

この図が示す簡単な銀行業務アプリケーションには、照会、引き出し、振り替え、預け入れなどのサービスがあります。サービス・リクエストは通常、プログラム・ライブラリを呼び出すプログラムとして C 言語か COBOL で記述されます。ネイティブなプログラムからライブラリにアクセスすることは、特定の CPU とオペレーティング・システムのリリースの組み合わせに対応するライブラリをインストールすることを意味しますが、これは Java が、その設計上避けようとしているような状況です。Jolt サーバの実装は Jolt クライアントのプロキシとして動作し、Jolt クライアントに代わって BEA Tuxedo サービスを呼び出します。BEA Jolt サーバは、リクエストを Jolt クライアントから受信し、それらのリクエストを BEA Tuxedo サービス・リクエストにマップします。

図 1-1 BEA Jolt のアーキテクチャ



Jolt サーバ

次のような Jolt サーバ・コンポーネントは協調して、Jolt クライントのトランザクション処理リクエストを BEA Tuxedo アプリケーションに渡します。

- Jolt サーバ・リスナ (JSL)

  JSL は、Jolt クライアントからの最初の接続リクエストを処理し、Jolt クライアントを Jolt サーバ・ハンドラに割り当てます。

- Jolt サーバ・ハンドラ (JSH)

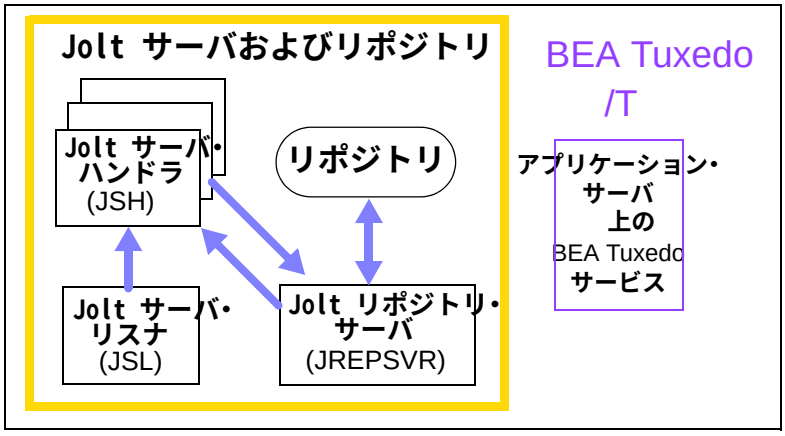
JSH は、ネットワーク接続の管理、クライアントの代理としてのサービス・リクエストの実行、BEA Tuxedo バッファ型と Jolt バッファ型の相互変換を行います。

- Jolt リポジトリ・サーバ (JREPSVR)

  JREPSVR は Jolt サービス定義を Jolt リポジトリから取り出し、そのサービス定義を JSH に返します。また、Jolt サービス定義の更新や追加も行います。

図 1-2 に Jolt サーバ・コンポーネントと Jolt リポジトリ・コンポーネントを示します。

図 1-2 Jolt サーバ・コンポーネントと Jolt リポジトリ・コンポーネント



Jolt クラス・ライブラリ

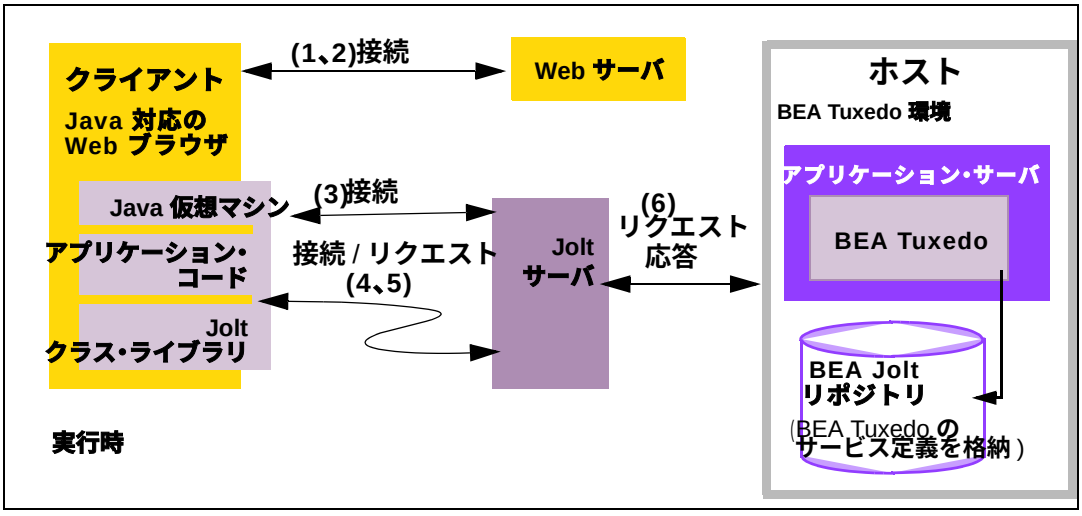
BEA Jolt クラス・ライブラリは一組のクラスであり、これを Java アプリケーションやアプレットで使うことによって、Java 対応のクライアントから BEA Tuxedo にサービスをリクエストできるようになります。これらの Jolt クラス・オブジェクトにより、BEA Tuxedo トランザクション・サービスにアクセスすることができます。

Jolt クライアント・アプリケーションを開発するときに開発者が知っていなければならないのは、Jolt が提供するクラスと Jolt リポジトリによってエクスポートされる BEA Tuxedo サービスのみです。Jolt は背後にあるアプリケーションの詳細を隠します。Jolt と Jolt クラス・ライブラリを使うために、背後で実行されているトランザクションの詳細や、サービスを記述した言語、バッファ操作、サービスの所在、使用されるデータベース名などを知る必要はありません。

Jolt API は Java クラス・ライブラリであり、Java が提供する利点を備えています。つまり、アプレットは動的にダウンロードされ、実行されている間のみ存在します。その結果、クライアント側でインストールしたり、管理したり、バージョンをコントロールする必要がありません。サービスが変更された場合、クライアント・アプリケーションは次に Jolt リポジトリを呼び出すときに、その変更があったことを知ります。

次の図は、Jolt クライアントと BEA Tuxedo システム間の処理の流れを示します。図中の番号は、1-10 ページの「Jolt クラス・ライブラリの使用」のアクションの説明に対応しています。

図 1-3 Jolt クラス・ライブラリを使って BEA Tuxedo サービスにアクセスする



次の表で、「Jolt クラス・ライブラリを使って BEA Tuxedo サービスにアクセスする」に示した Jolt クラス・ライブラリを使用して BEA Tuxedo サービスにアクセスする処理の流れを簡単に説明します。

表 1-1 Jolt クラス・ライブラリの使用

プロセス	手順	アクション
接続	1	Java 対応の Web ブラウザが HTTP プロトコルを使用して HTML ページをダウンロードします。
...	2	Jolt アプレットがダウンロードされ、クライアントの Java 仮想マシンで実行されます。
...	3	Java アプレットの最初の作業は、Jolt サーバに対する別の接続を確立することです。
リクエスト	4	Jolt クライアントはサービスのシグネチャ (名前、パラメータ、型など) を知っているので Jolt クラス定義に基づいてサービス・リクエスト・オブジェクトを構築し、メソッドを呼び出すことができます。
...	5	リクエストは Jolt サーバに送信され、そこで Java に基づくリクエストが BEA Tuxedo リクエストに変換されて BEA Tuxedo 環境に転送されます。
応答	6	BEA Tuxedo がリクエストを処理してその情報を Jolt サーバに返すと、変換し直されて Java アプレットに返されます。

JoltBeans

BEA Jolt クライアントを作成するために Java 対応の統合開発環境 (IDE : Integrated Development Environment) で使用される JoltBeans、Java Beans コンポーネントが、今回 BEA Jolt に含まれるようになりました。JavaBeans を使用すると、クライアント・アプリケーションを Web Gain Visual Café などの JavaBeans 対応開発ツールのグラフィカル機能を使って作成することができます。

BEA JoltBeans には、BEA Jolt への JavaBeans 対応インターフェイスが用意されています。これにより、コードを記述することなく、フル機能を備えた BEA Jolt クライアントを開発することができます。開発環境のコンポーネント・パレットにある JoltBeans を、作成中の Jolt クライアント・アプリケーションの Java フォーム (複数のフォームも可) へドラッグアンドドロップできます。次に、Bean のプロパティを設定し、アプリケーションやアプレットの Bean に対してイベント・ソースとイベント・リスナの関係をグラフィカルに設定します。通常、開発ツールはイベント発生時に起動するコードを生成します。コードが生成されない場合は手動で生成します。JoltBeans を使ったクライアントの開発では、BEA Jolt リポジトリも使用します。BEA Jolt リポジトリを使用すると、利用可能な BEA Tuxedo 機能に簡単にアクセスできます。

ASP Connectivity for BEA Tuxedo

Jolt ASP Connectivity for BEA Tuxedo ツールキットは、Jolt Java クラス・ライブラリに対して拡張された機能です。このツールキットを使うと、Jolt クライアント・クラス・ライブラリを Microsoft IIS (Internet Information Server) などの Web サーバで使用し、HTML クライアントや HTML ブラウザと BEA Tuxedo サービス間のインターフェイスを実現することができます。

Jolt ASP Connectivity for BEA Tuxedo ツールキットは、動的な HTML ページを処理したり、作成したりするための使いやすいインターフェイスを提供します。BEA Tuxedo サービスにアクセスするために Common Gateway Interface (CGI) トランザクション・プログラムの書き方を学ぶ必要はありません。

Jolt サーバと Jolt クライアントの通信

Jolt は Jolt サーバと Jolt クライアント間の通信すべてを BEA Jolt プロトコルで処理します。Jolt サーバと Jolt クライアント・アプレットやアプリケーション間の通信手順は以下のとおりです。

1. BEA Tuxedo サービス・リクエストおよび関連するパラメータがメッセージ・バッファにパッケージ化されて、ネットワーク経由で Jolt サーバに配信されます。
2. Jolt サーバはメッセージの中からデータを入手して、数値形式の変換や文字セットの変換などの必要なデータ変換を行います。
3. Jolt サーバは、Jolt クライアントが要求したアプリケーション・サービスに対する適切なサービス要求を実行します。
4. サービス・リクエストが BEA Tuxedo システムに入力されると、ほかの BEA Tuxedo クライアントによって出されるリクエストとまったく同じように実行されます。
5. その結果は BEA Jolt 1.2 BEA Jolt サーバに返され、そこで結果やエラー情報をメッセージにパッケージ化して Jolt クライアントに送ります。
6. Jolt クライアントはメッセージの中身を様々な Jolt クライアント・インターフェイス・オブジェクトにマップして、リクエストを完了します。

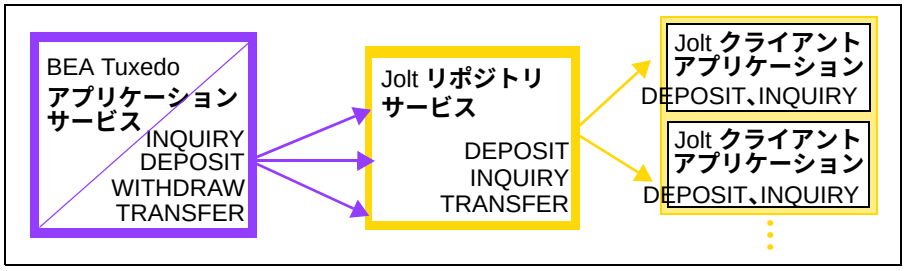
Jolt リポジトリ

Jolt リポジトリは、名前、番号、型、パラメータ・サイズ、パーミッションなどの BEA Tuxedo サービスに関する情報が定義されるデータベースです。リポジトリは BEA Tuxedo サービス定義の中心的データベースとして機能し、新規および既存の BEA Tuxedo アプリケーション・サービスを Jolt クライアント・アプリケーションで利用できるようにします。BEA Tuxedo アプリケーションには、ADD_CUSTOMER、GET_ACCOUNTBALANCE、CHANGE_LOCATION、GET_STATUS などのサービスまたはサービス定義を数多く持たせることができます。これらの定義の全部またはごく一部だけを Jolt リポジトリにエクスポートすることができます。開発者またはシステム管理者は、Jolt リポジトリ・エディタを使って、これらのサービスを Jolt クライアント・アプリケーションにエクスポートすることができます。

1 つのクライアントにエクスポートされたすべてのリポジトリ・サービスは、すべてのクライアントにエクスポートされます。サービスのサブセットが 1 つのクライアントに必要で、他には必要でないようなケースは、BEA Tuxedo に処理させます。

次の図は、Jolt リポジトリが BEA Tuxedo サービスを複数の Jolt クライアント・アプリケーションに仲介する仕組みを示しています。4 種類の BEA Tuxedo サービスが示されていますが、WITHDRAW サービスはリポジトリに定義されておらず、また TRANSFER サービスは定義されていますがエクスポートされていません。

図 1-4 Jolt 経由で BEA Tuxedo サービスを配信する



Jolt リポジトリ・エディタ

Jolt リポジトリ・エディタは、アプリケーション管理者が個々の BEA Tuxedo サービスにアクセスできるようにする、Java の GUI 管理ツールです。Jolt リポジトリ・エディタを使って、サービスの定義、テストおよび Jolt クライアントに対するエクスポートを行うことができます。

Note: Jolt リポジトリ・エディタは、Jolt クライアント・アプリケーションに対するサービスだけを管理します。BEA Tuxedo アプリケーション自体の変更を行うためには使用できません。

Jolt リポジトリ・エディタによって、コードを大量に修正することなく、BEA Tuxedo サービスを Jolt クライアントに拡張および配信することができます。また、BEA Tuxedo サービスのパラメータを修正し、BEA Tuxedo サービスをパッケージに論理的にグループ化し、作成されたパッケージからサービスを取り除くことができます。また、サービスをエクスポートすることで、ブラウザ上の Jolt アプレットまたは Jolt アプリケーションから利用できるようにすることができます。

Jolt インターネット・リレー

Jolt インターネット・リレーは、Jolt クライアントからのメッセージを Jolt サーバにルーティングするコンポーネントです。Jolt インターネット・リレーは Jolt リレー (JRLY) と Jolt リレー・アダプタ (JRAD) から構成されます。JRLY は、Jolt メッセージを Jolt リレー・アダプタにルーティングするスタンドアロンのソフトウェア・コンポーネントです。Jolt リレーは Jolt クライアントと通信するための最低限の設定しか必要とせず、これによって BEA Tuxedo は Web サーバと同じマシン上で実行されなくてもよくなります。

JRAD は、BEA Tuxedo システム・サーバですが、これには BEA Tuxedo サービスはまったく含まれていません。これを JSH および BEA Tuxedo とともに動かすためにはコマンド・ライン引数が必要です。JRAD は JRLY からクライアント・リクエストを受信し、このリクエストを適切な JSH に転送します。JSH からの応答は JRAD に返され、JRAD は JRLY にその応答を返します。1 つの Jolt インターネット・リレー (JRLY/JRAD のペア) は複数のクライアントを同時に並行して処理します。

Jolt クライアントを作成して BEA Tuxedo アプリケーションにアクセスする

Jolt クライアントを作成および運用する主な手順を、以下の手順および 1-17 ページの「Jolt アプリケーションの作成」で説明します。

1. BEA Tuxedo システム・アプリケーションを作成したことを確認します。

   BEA Tuxedo のインストール方法と BEA Tuxedo アプリケーションの作成方法の詳細については、『BEA Tuxedo システムのインストール』および『BEA Tuxedo アプリケーションの設定』を参照してください。

2. Jolt をインストールします。

   『BEA Tuxedo システムのインストール』を参照してください。

3. バルク・ロード・ユーティリティを使って、Tuxedo サービスを Jolt リポジトリ・データベースにロードします。

   このユーティリティの使用方法の詳細については、「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。

4. Jolt リポジトリ・エディタを使用してサービスを設定し定義します。

   Jolt リポジトリ・エディタの設定方法と BEA Tuxedo サービスを Jolt で利用する方法の詳細については、第 4 章「Jolt リポジトリ・エディタを使う」を参照してください。

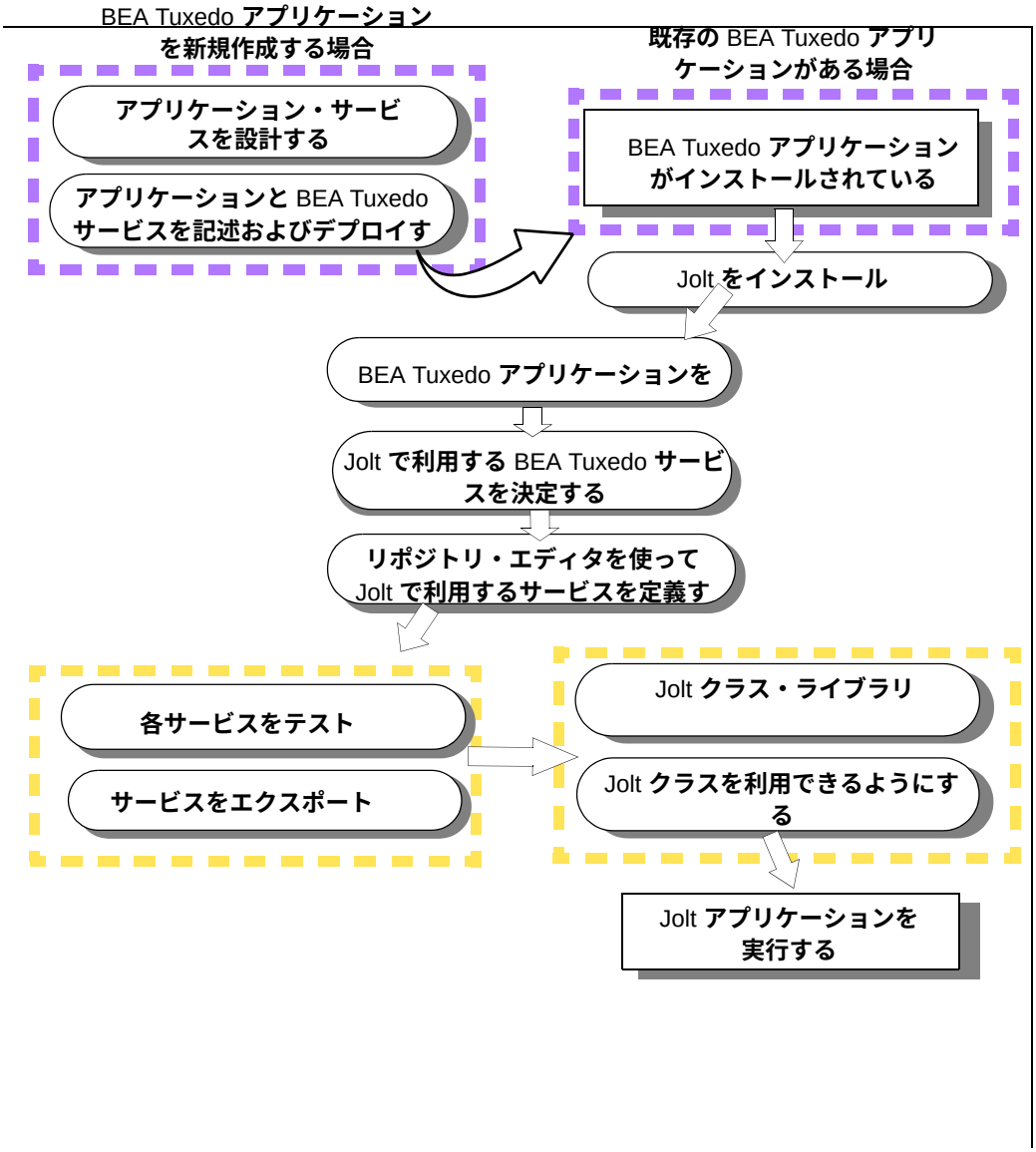
5. Jolt クラス・ライブラリを使用してクライアント・アプリケーションを作成します。

   Jolt クラス・ライブラリを使用してクライアント・アプリケーションをプログラミングする方法については、以下を参照して下さい。

   - 第 5 章「Jolt クラス・ライブラリを使う」
   - 『BEA Jolt API リファレンス』

6. Jolt に基づくクライアント・アプレットやアプリケーションを実行します。

図 1-5 Jolt アプリケーションの作成



2 大量の BEA Tuxedo サービス定義をロードする

既存の BEA Tuxedo アプリケーションに複数の BEA Tuxedo サービスがある場合、これらのサービスを手動で定義してリポジトリ・データベースに登録すると、大幅に時間がかかります。Jolt バルク・ローダは、複数の定義済み BEA Tuxedo サービスを一度に Jolt リポジトリ・データベースにロードするためのコマンド・ユーティリティです。`jbld` プログラムを実行すると、バルク・ローダ・ユーティリティは BEA Tuxedo サービスの定義が記述されたテキスト・ファイルを読み込み、その内容を Jolt リポジトリにロードします。複数のサービス定義情報は、1 回の 「バルク・ロード」でリポジトリ・データベースにロードされます。サービスが Jolt リポジトリにロードされると、Jolt リポジトリ・エディタを使って、サービスの編集、新規作成、およびグループ化を行うことができます。

ここでは、次の内容について説明します。

- バルク・ローダの使い方
- バルク・ローダ・データ・ファイルの構文
- トラブルシューティング
- バルク・ロード・データのサンプル

バルク・ローダの使い方

jbld プログラムは Java アプリケーションです。そのため、jbld コマンドを実行する前に、Jolt クラス・ディレクトリ (jolt.jar や joltadmin.jar) の場所を指すように CLASSPATH 環境変数 (または、それに相当するもの) を設定してください。CLASSPATH 変数が設定されていないと、Java 仮想マシン (JVM) は Jolt クラスを見つけることができません。

セキュリティ上の理由により、jbld でユーザ認証情報 (ユーザ・パスワードまたはアプリケーション・パスワード) を指定する場合はコマンド行引数を使用しません。サーバのセキュリティ・レベルに応じて、jbld はパスワードの入力をユーザに自動的に求めます。

バルク・ローダ・ユーティリティは、コマンド行引数と入力ファイルから入力情報を取得します。

バルク・ローダを起動する

1. プロンプトで、適切なオプションを指定して以下のように入力します。

   ```
   java bea.jolt.admin.jbld [-n][-p package][-u usrname][-r
   usrrole] //host:port filename
   ```

2. 次の表を使用して、適切なコマンド行オプションを指定してください。

コマンド行オプション

表 2-1 バルク・ローダのコマンド行オプション

オプション	説明
-u *usrname*	ユーザ名 (デフォルトはアカウント名) を指定します。(セキュリティの設定で必要とされていれば必須)

表 2-1 バルク・ローダのコマンド行オプション (続き)

オプション	説明
`-r usrrole`	ユーザ・ロール (デフォルトは admin) を指定します。(セキュリティの設定で必要とされていれば必須)
`-n`	現在使用しているリポジトリに対して、入力ファイルが有効であるかを検証します。リポジトリは更新されません。(オプション)
`-p package`	リポジトリのパッケージ名。(デフォルトは BULKPKG)
`//host:port`	JRLY または JSL アドレス (ホスト名と IP ポート番号) を指定します。(必須)
`filename`	サービス定義を含むファイルを指定します。(必須)

バルク・ロード・ファイル

バルク・ロード・ファイルは、サービスおよびそれに関するパラメータを定義するテキスト・ファイルです。バルク・ローダは、このバルク・ロード・ファイルに定義されたサービスを Jolt リポジトリにロードします。このとき、デフォルトで 「BULKPKG」というパッケージ名が使用されます。-p コマンドを使って任意のパッケージ名を指定し、デフォルトのパッケージ名を上書きすることもできます。-p オプションを指定してバルク・ロード・ファイルから再度ロードを行うと、元のパッケージ内のサービスはすべて削除され、新しいバルク・ロード・ファイルを作成すると、新しいパッケージが作成されます。

-p オプションで指定していない名前がパッケージに付いていると、コンフリクトの発生がバルク・ローダから報告され、バルク・ロード・ファイルからリポジトリへのサービスのロードは行われません。リポジトリ・エディタを使って重複するサービスを削除し、再度バルク・ロード・ファイルをロードしてください。その他の情報については、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

バルク・ローダ・データ・ファイルの構文

各サービス定義は、サービス・プロパティと、特定の数のパラメータ・プロパティを持つパラメータで構成されます。各プロパティは、キーワードと値で表されます。

キーワードは、次の 2 つのレベルに分類されます。

- サービス・レベル
- パラメータ・レベル

キーワードの使用に関するガイドライン

jbld プログラムはテキスト・ファイルからサービス定義を読み取ります。キーワードを使用する際は、次の表のガイドラインに従ってください。

表 2-2 キーワードの使用に関するガイドライン

ガイドライン	例
各キーワードの後には等号 (=) と値を指定する。	正：`type=string` 誤：`type`
1 行につき 1 つのキーワードを指定する。	正：`type=string` 誤：`type=string access=out`
等号 (=) のない行は無視される。	正：`type=string` 誤：`type string`
明確に定義された値しか受け付けないキーワードがある。	キーワード access が受け付ける値は、in、out、inout、noaccess の 4 つです。

表 2-2 キーワードの使用に関するガイドライン (続き)

ガイドライン	例
入力ファイルには複数のサービス定義を指定できる。	service=INQUIRY <service keywords and values> service=DEPOSIT <service keywords and values> service=WITHDRAWAL <service keywords and values> service=TRANSFER <service keywords and values>
各サービス定義には、キーワードと値の組み合わせを複数指定できる。	service=DEPOSIT export=true inbuf=VIEW32 outbuf=VIEW32 inview=INVIEW outview=OUTVIEW

バルク・ローダ・データ・ファイルのキーワードの順序

バルク・ロード時の転送エラーを避けるため、データ・ファイル内のキーワードは指定された順序で並べる必要があります。

まず、バルク・ローダ・データ・ファイルの先頭には、最初のサービスの service=<NAME> キーワードを定義します (「データ・ファイル内のキーワードの順序」を参照)。service=<NAME> キーワードの後には、このサービスに適用されるその他のサービス・キーワードをすべて指定します。これらのサービス・キーワードは、param=<NAME> より前に指定しなければなりません。ただし、サービス・キーワードは、どのような順序で指定してもかまいません。

次に、サービスに関連するパラメータをすべて指定します。param=<NAME> キーワードの後には、このパラメータに適用されるパラメータ・キーワードをすべて指定します。これらのパラメータ・キーワードは、次のパラメータ定義の前に指定しなければなりません。パラメータ・キーワードは、どのよ

うな順序で指定してもかまいません。最初のサービスに関連するすべてのパラメータを定義したら、次のサービスの service=<NAME> キーワードを定義します。

コード リスト 2-1 データ・ファイル内のキーワードの順序

```
service=<NAME>
<service keyword>=<value>
<service keyword>=<value>
<service keyword>=<value>
param=<NAME>
<parameter keyword>=<value>
<parameter keyword>=<value>
param=<NAME>
<parameter keyword>=<value>
<parameter keyword>=<value>
```

サービス・レベルのキーワードと値を使う

サービス定義は service=<NAME> キーワードから始める必要があります。CARRAY、STRING、または XML のバッファ型を使うサービスに指定できるパラメータは 1 つだけです。CARRAY バッファ型を使用するサービスには、パラメータ名として CARRAY、データ型として carray を指定することをお勧めします。STRING バッファ型を使用するサービスの場合は、パラメータ名として STRING、データ型として string を指定することをお勧めします。XML バッファ型を使用するサービスの場合は、パラメータ名として XML、データ型として xml を指定することをお勧めします。

次の表は、サービス・レベルのキーワードと適切な値を使用する際のガイドラインです。

表 2-3 サービス・レベルのキーワードと値

キーワード	値
service	任意の BEA Tuxedo サービス名。

表 2-3 サービス・レベルのキーワードと値 (続き)

キーワード	値
export	true または false (デフォルトは false)。
inbuf/outbuf	次のバッファ型のいずれかを選択します。 FML FML32 VIEW VIEW32 STRING CARRAY XML X_OCTET X_COMMON X_C_TYPE
inview	入力パラメータに対する任意のビュー名。 (VIEW、VIEW32、X_COMMON、X_C_TYPE のいずれかのバッファ型が使用されている場合に限り、キーワードは省略可能です。)
outview	出力パラメータに対する任意のビュー名 (オプション)。

パラメータ・レベルのキーワードと値を使う

パラメータの定義では、まず param=<NAME> キーワードを指定し、続いてパラメータ・キーワードを指定します。パラメータ・キーワードの次は、別の param キーワードか service キーワードが続くか、ファイルの最後になります。param=<NAME> キーワード以降のパラメータ・キーワードは、どのような順序で指定してもかまいません。

次の表は、パラメータ・レベルのキーワードと適切な値を使用する際のガイドラインです。

表 2-4 パラメータ・レベルのキーワードと値

キーワード	値
param	任意のパラメータ名
type	byte short integer float double string carray xml
access	in out inout noaccess
count	オカレンスの最大数 (デフォルトは 1)。無制限のオカレンスには 0 を指定します。リポジトリ・エディタでテスト画面のフォーマット用に使用されます。それ以外には使用されません。

トラブルシューティング

バルク・ローダ・ユーティリティの使用中に問題が発生した場合は、次の表を参照してください。バルク・ローダ・ユーティリティのエラー・メッセージと解決策の総合一覧については、「BEA Jolt システム・メッセージ」を参照してください。

表 2-5 バルク・ローダのトラブルシューティング

状況	解決策
データ・ファイルが見つからない。	パスが正しく設定されているかどうかを確認します。
キーワードが無効である。	パッケージ、サービスまたはパラメータのキーワードが有効であるかどうかを確認します。
キーワードの値がヌルである。	キーワードの値を入力します。
値が無効である。	パラメータの値がパラメータに割り当てられた範囲内にあるかどうかを確認します。
データ型が無効である。	パラメータで有効なデータ型が使用されていることを確認します。

バルク・ロード・データのサンプル

次の一覧は、UNIX コマンドの cat servicefile を使って表示した、正しい形式のサンプル・データ・ファイルです。この例では、TRANSFER、LOGIN、PAYROLL サービス定義が BULKPKG にロードされます。

コード リスト 2-2 バルク・ロード・データのサンプル

```
service=TRANSFER
export=true
inbuf=FML
outbuf=FML
param=ACCOUNT_ID
type=integer
access=in
count=2
param=SAMOUNT
type=string
access=in
param=SBALANCE
type=string
access=out
count=2
param=STATLIN
type=string
access=out

service=LOGIN
inbuf=VIEW
inview=LOGINS
outview=LOGINR
export=true
param=user
type=string
access=in
param=passwd
type=string
access=in
param=token
type=integer
access=out
```

```
service=PAYROLL
inbuf=FML
outbuf=FML
param=EMPLOYEE_NUM
type=integer
access=in
param=SALARY
type=float
access=inout
param=HIRE_DATE
type=string
access=inout
```

3 BEA Jolt システムの設定

この章では、BEA Jolt の設定方法を説明します。Jolt を既に使用したことがある場合は「簡易設定」をご覧ください。それ以外の節では、さらに詳しい内容を説明しています。この章は、BEA Jolt のインストール先のオペレーティング・システムやワークステーション・プラットフォームでの作業を経験したことのあるシステム管理者またはアプリケーション開発者を対象としています。

ここでは、次の内容について説明します。

- 簡易設定
- Jolt に関する背景情報
- Jolt リレー
- Jolt リレー・アダプタ
- Jolt リポジトリ
- イベント・サブスクリプション
- BEA Tuxedo に関する背景情報
- BEA Jolt オンライン・マニュアルのサンプル・アプリケーション

簡易設定

BEA Jolt および BEA Tuxedo を既に使用したことがある場合は、この「簡易設定」で説明する、BEA Jolt のコンフィギュレーション方法に関するガイドラインを参照してください。Jolt を初めて使用する場合は、3-14 ページの「Jolt に関する背景情報」を読んでから設定を始めてください。

簡易設定では、Jolt サーバ・リスナ (JSL) を BEA Tuxedo に設定するために必要な次の手順を説明します。

- UBBCONFIG ファイルを編集する
- Jolt リポジトリを設定する
- BEA Tuxedo およびリポジトリ・エディタを使用するサービスを初期化する
- リポジトリ・エディタにログオンする
- リポジトリ・エディタを終了する
- イベント・サブスクリプション用に BEA Tuxedo の TMUSREVT サーバを設定する
- Jolt リレーを設定する

UBBCONFIG ファイルを編集する

1. `MACHINES` セクションで、`MAXWSCLIENTS=`*`number`* を指定します (必須)。

注記　`MAXWSCLIENTS` が設定されていないと、JSL は起動しません。

2. `GROUPS` セクションで、`GROUPNAME` に必須パラメータとオプション・パラメータを設定します。
3. `SERVERS` セクションを設定します (必須)。

   このセクション内にある各行の形式は次のとおりです。

JSL *必須パラメータ* [*オプション・パラメータ*]

JSL は、tmboot(1) で実行されるファイル (*string_value*) を指定します。

4. JSL の必須パラメータを設定します。

   以下のパラメータは必須です。

   SVRGRP=*string_value*

   SRVID=*number*

   CLOPT="-A...-n...//host port"

5. JSL にその他のパラメータを設定します。

   JSL には次のパラメータを使用できますが、これらのパラメータを設定するとアプリケーションにどのような影響が及ぶかを考慮してください。詳細については、3-51 ページの「JSL で使用できるパラメータ」を参照してください。

   MAX # of JSHs

   MIN # of JSHs

Jolt リポジトリを設定する

Groups セクションの設定

1. MACHINES セクションの LMID パラメータに指定されている値と同じ識別子を指定します。
2. GRPNO に 1 ～ 30,000 の範囲の値を指定します。

Servers セクションの設定

BEA Jolt リポジトリ・サーバ (JREPSVR) には、リポジトリにアクセスしたり、リポジトリを編集するためのサービスが格納されています。JREPSVR インスタンスが複数ある場合は、共有ファイルにより、リポジトリの情報が共有されます。UBBCONFIG ファイルの SERVERS セクションには、JREPSVR を指定してください。

1. SRVID パラメータを使用して、新しいサーバの識別子を指定します。
2. JREPSVR に -W フラグを指定し、リポジトリを編集可能にしておきます (この設定は 1 つの JREPSVR だけに対して行います)。このフラグを設定しないと、リポジトリは読み取り専用になります。
3. -P フラグを指定して、リポジトリ・ファイルへのパスを設定します。-P フラグに対する引数が指定されていないと、BEA Tuxedo の ULOG ファイルにエラー・メッセージが表示されます。
4. リポジトリ・ファイルのファイル・パス名を追加します (例 : /app/jrepository)。
5. tmloadcf コマンドと tmboot コマンドを使用して、BEA Tuxedo システムを起動します。

BEA Tuxedo およびリポジトリ・エディタを使用するサービスを初期化する

BEA Tuxedo および BEA Jolt を使用する BEA Tuxedo サービスを定義し、クライアントが Jolt サービスを利用できるようにします。

1. サービスを格納した BEA Tuxedo サーバを作成します。
2. BEA Jolt リポジトリ・エディタにアクセスします。

リポジトリ・エディタを使用する

リポジトリ・エディタを起動する前に、必要な BEA Jolt ソフトウェアがすべてインストールされているかどうかを確認してください。

注記　JREPSVR と JSL が実行されていないと、リポジトリ・エディタは使用できません。

リポジトリ・エディタを使用するには、次の手順に従います。

1. リポジトリ・エディタを起動します。

リポジトリ・エディタは、JavaSoft `appletviewer` または Web ブラウザから起動します。これらの手順は、次の節でさらに詳しく説明されます。

2. リポジトリ・エディタにログオンします。

Java Applet Viewer を使用してリポジトリ・エディタを起動する

1. `CLASSPATH` に、Jolt クラスのディレクトリか、または `*.jar` ファイルが置かれたディレクトリを指定します。
2. アプレットをローカル・ディスクからロードする場合は、URL に次のように入力してください。

   `appletviewer` full-pathname`/RE.html`

   Web サーバからアプレットをロードする場合は、URL に次のように入力してください。

   `http://`*`www.server/URL path`*`/RE.html`

3. Enter キーを押します。

   3-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」に示すようなウィンドウが表示されます。

Web ブラウザを使用してリポジトリ・エディタを起動する

次のいずれかの方法を使用して、Web ブラウザからリポジトリ・エディタを起動してください。

ローカル・ファイルからリポジトリ・エディタを起動する場合

1. `CLASSPATH` に Jolt クラスのディレクトリを指定します。
2. 次を入力します。

   `file:`full-pathname`/RE.html`

3. Enter キーを押します。

   3-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」に示すようなウィンドウが表示されます。

Web サーバからリポジトリ・エディタを起動する場合

1. CLASSPATH に Jolt クラスのディレクトリが含まれていないことを確認します。
2. CLASSPATH から Jolt クラスを削除します。
3. 次を入力します。

   `http://www.server/URL path/RE.html`

   注記　jolt.jar および admin.jar が RE.html と同じディレクトリに置かれている場合は、Web サーバ側がクラスを作成します。RE.html とは異なるディレクトリに置かれている場合は、アプレットのコードを変更します。
4. Enter キーを押します。

   3-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」に示すようなリポジトリ・エディタのログオン・ウィンドウが表示されます。

リポジトリ・エディタにログオンする

Jolt リポジトリ・エディタを起動したら、以下の手順に従ってログオンします。

注記　ログオンする前に、3-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」が表示されている必要があります。この画面が表示されたことを確認してから以下の手順に進んでください。

1. [Logon] ウィンドウに、BEA Tuxedo アプリケーションへの「アクセス・ポイント」として指定されているサーバ・マシン名を入力し、Tab キーを押します。
2. ポート番号を入力して Enter キーを押します。

   システムにより、サーバとポートの情報が検証されます。

注記　Jolt リレー経由でログオンしない限り、ここで指定したポート番号と同じ番号が Jolt リスナの設定時に使用されます。詳細については、`UBBCONFIG` ファイルを参照してください。

3. BEA Tuxedo のアプリケーション・パスワードを入力して Enter キーを押します。

   認証レベルに応じて、手順 5 と 手順 6 を実行します。

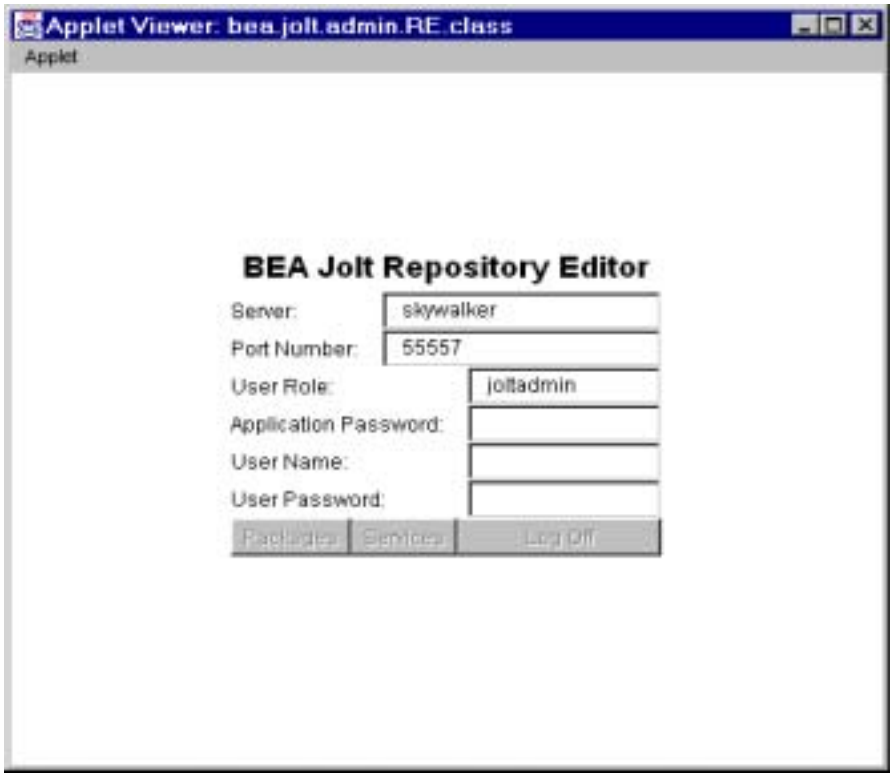
4. BEA Tuxedo ユーザ名を入力して Tab キーを押します。

5. BEA Tuxedo ユーザ・パスワードを入力して Enter キーを押します。

   [Packages] ボタンと [Services] ボタンが有効になります。

   注記　BEA Jolt のリポジトリ・エディタでは、ユーザ・ロールにハードコーディングされた `joltadmin` が使用されます。

図 3-1BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ



次の「リポジトリ・エディタの [Logon] ウィンドウの説明」では、このウィンドウにあるフィールドとボタンについて説明します。

リポジトリ・エディタの [Logon] ウィンドウの説明

表 3-1 リポジトリ・エディタの [Logon] ウィンドウの説明

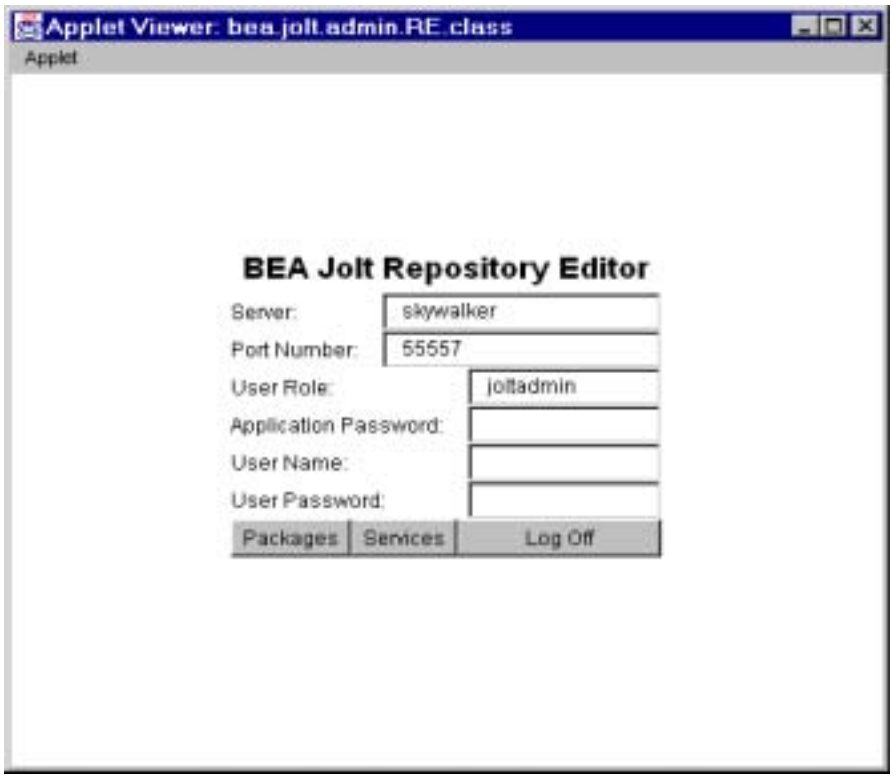
オプション	説明
Server	サーバ名を入力します。

Port Number	10 進値でポート番号を入力します。 注記　サーバ名とポート番号が入力されると、[User Name] フィールドと [User Password] フィールドがアクティブになります。これらのフィールドがアクティブになるかどうかは BEA Tuxedo アプリケーションの認証レベルに基づきます。
User Role	BEA Tuxedo ユーザ・ロールを入力します。BEA Tuxedo の認証レベルが USER_AUTH 以上の場合にのみ入力します。
Application Password	BEA Tuxedo の管理パスワードを入力します。
User Name	BEA Tuxedo ユーザを識別するための名前を入力します。名前の先頭には英字を指定します。
User Password	BEA Tuxedo のパスワードを入力します。
Packages	[Packages] ウィンドウにアクセスします (ログオンすると利用可能になります)。
Services	[Services] ウィンドウにアクセスします (ログオンすると利用可能になります)。
Log Off	サーバとの接続を切断します。

リポジトリ・エディタを終了する

パッケージ、サービス、およびパラメータの追加、編集、テスト、削除が完了したらリポジトリ・エディタを終了します。終了する前に、3-10 ページの「BEA Jolt リポジトリ・エディタを終了する前の [Logon] ウィンドウ」に示すようなウィンドウが表示されます。

図 3-2BEA Jolt リポジトリ・エディタを終了する前の [Logon] ウィンドウ



使用できるのは、[Packages]、[Services]、および [Log Off] のみです。文字入力フィールドは利用できません。

リポジトリ・エディタを終了するには、以下の手順に従います。

1. 前のウィンドウで [Back] をクリックして、リポジトリ・エディタの [Logon] ウィンドウに戻ります。
2. [Log Off] をクリックしてサーバとの接続を切断します。

   リポジトリ・エディタの [Logon] ウィンドウ内のフィールドが入力できない状態になります。
3. ブラウザのメニューの [Close] をクリックしてウィンドウを閉じます。

イベント・サブスクリプション用に BEA Tuxedo の TMUSREVT サーバを設定する

Jolt のイベント・サブスクリプション機能では、BEA Tuxedo サービスまたは別の BEA Tuxedo クライアントからイベント通知を受信します。BEA Tuxedo の TMUSREVT サーバを設定し、アプリケーションの UBBCONFIG ファイルを編集してください。次の「UBBCONFIG ファイル内の TMUSREVT パラメータ」は、UBBCONFIG ファイル内の TMUSREVT パラメータを示しています。

コード リスト 3-1　UBBCONFIG **ファイル内の** TMUSREVT **パラメータ**

```
TMUSREVT        SRVGRP=EVBGRP1  SRVID=40        GRACE=3600
                ENVFILE="/usr/tuxedo/bankapp/TMUSREVT.ENV"
                CLOPT="-e tmusrevt.out -o tmusrevt.out -A --
                -f /usr/tuxedo/bankapp/tmusrevt.dat"
                SEQUENCE=11
```

UBBCONFIG ファイルの SERVERS セクションで、SRVGRP と SRVID を指定してください。

Jolt リレーを設定する

UNIX の場合

システム・プロンプトで次のコマンドを入力し、UNIX での JRLY プロセスを開始します。

```
jrly -f <config_file_path>
```

コンフィギュレーション・ファイルが存在しないか開けない場合、JRLY は標準エラーにメッセージを書き込み、起動時のエラーをエラー・ログに記録してから終了します。

UNIX および Windows 2000 の場合

コンフィギュレーション・ファイルは、「タグ = 値」の形式で記述されています。空白行または「#」で始まる行は無視されます。次の「正式なコンフィギュレーション・ファイルの形式」の例を参照してください。

コード リスト 3-2 正式なコンフィギュレーション・ファイルの形式

```
LOGDIR=<LOG_DIRECTORY_PATH>
ACCESS_LOG=<ACCESS_FILE_NAME in LOGDIR>
ERROR_LOG=<ERROR_FILE_NAME in LOGDIR>
LISTEN=<IP:Port combination where JRLY will accept
comma-separated connections>
CONNECT=<IP:Port1, IP:Port2...IP:PortN:Port(List of IP:Port
combinations associated with JRADs:can be 1...N)>
```

Windows 2000 のみの場合 (省略可能)

SOCKETTIMEOUT は、JRLY の Windows 2000 サービスが、ネットワーク・アクティビティ (新しい接続、読み込み対象のデータ、クローズされた接続など) を実現するためにソケット接続をブロックする期間を示す時間 (秒単位) を指定します。SOCKETTIMEOUT の値はサービス・コントロール・マネージャ (SCM: Service Control Manager) にも影響します。サービス・コントロール・マネージャは、Windows 2000 サービスの停止を要求する場合には少なくとも SOCKETTIMEOUT で指定した秒数を待つ必要があります。

注記　ディレクトリ名とファイル名の形式は、オペレーティング・システムによって異なります。UNIX システムではスラッシュ (/) を使用します。Windows 2000 システムでは円記号 (\) を使用します。LOGDIR、ACCESS_LOG、または ERROR_LOG で指定されたファイルを開けない場合、JRLY は stderr にエラー・メッセージを記録してから終了します。

次の表は、ホスト名とポート番号の形式を示しています。

表 3-2 ホスト名とポート番号の形式

ホスト名 / ポート番号	説明
//Hostname:Port	Hostname は文字列で指定し、Port は 10 進数で指定します。
IP:Port	IP にはドット区切りの IP アドレスを指定し、Port には 10 進数値を指定します。

Jolt リレー・アダプタ (JRAD: Jolt Relay Adapter) を起動する

1. tmloadcf -y <UBBFILE> と入力します。
2. tmboot と入力します。

Jolt リレー・アダプタを設定する

1 つの JRLY に接続できる JRAD プロセスは 1 つだけです。JRAD は、1 つの JSL および関連する JSH とだけ通信するように設定できます。複数の JRAD を 1 つの JSL と通信するように設定することもできます。UBBCONFIG ファイルには、BEA Tuxedo サービス用の CLOPT パラメータを含める必要があります。

1. -l *hexadecimal format* (JRLY がクライアントの代わりに接続する JSL ポート) と入力します。
2. -c *hexadecimal format* (JRAD が接続する JSL のアドレス) と入力します。

   注記　形式は、「0x0002PPPNNN」、またはドット区切りの「100.100.10.100」です。

3. ネットワーク接続されたコンポーネントを設定します。

   これで、Jolt が設定されました。

Jolt に関する背景情報

この節では、Jolt コンポーネントに関するその他の情報を説明します。

Jolt サーバ

Jolt サーバは、1 つまたは複数のハンドラを扱うリスナです。

Jolt サーバ・リスナ (JSL: Jolt Server Listener) ー JSL は、IP/ ポートの組み合わせで設定し、クライアントをサポートします。JSL は、Jolt サーバ・ハンドラ (JSH: Jolt Server Handler) と動作して、クライアントが BEA Jolt システムのバックエンドへ接続できるようにします。JSL は BEA Tuxedo サーバとして実行されます。

Jolt サーバ・ハンドラ (JSH: Jolt Server Handler) ー Tuxedo サーバ・マシンで実行されるプログラム。リモート・クライアント用のネットワーク接続ポイントを提供します。JSH は、JSL と動作し、クライアントが BEA Jolt システムのバックエンドに接続できるようにします。JSL に対し、最大 32,767 までの JSH を利用できます。詳細については、3-17 ページの「JSL のコマンド行オプション」の `-M` コマンド行オプションの説明を参照してください。

システム管理者の作業ーシステム管理者は、BEA Jolt のサーバ・コンポーネントに関して、以下の作業を行う必要があります。

- JSL のネットワーク・アドレスを決定する。
- サービスの対象とする Jolt クライアントの数を決定する。サービスの対象とするクライアントの数は、`UBB` の `MAXWSCLIENTS` で制限されています。
- JSH の最小数と最大数を決定する。

JSL を起動する

UBBCONFIG ファイルにあるすべての管理プロセスとサーバ・プロセスを開始するには、次の手順に従います。

1. tmloadcf と入力します。

   このコマンドにより、コンフィギュレーション・ファイルが解析され、バイナリ形式のコンフィギュレーション・ファイルがロードされます。

2. tmboot -y と入力します。

   このコマンドにより、コンフィギュレーション・ファイルで指定されたアプリケーションがアクティブになります。

   オプションを指定しない場合は、TUXCONFIG ファイルを上書きしてもよいかどうかを確認するメッセージが表示されます。

tmloadcf と tmboot については、『BEA Tuxedo アプリケーション実行時の管理』または 『BEA Tuxedo コマンド・リファレンス』を参照してください。

JSL をシャットダウンする

Jolt サーバに対してシャットダウンを要求するには、BEA Tuxedo の次のコマンドを入力します。

```
tmshutdown -y
```

シャットダウン中には次の制約があります。

- クライアント接続を新たに確立することはできません。
- 確立中のすべてのクライアント接続は切断されます。実行中のトランザクションは BEA Tuxedo によってロールバックされます。各クライアントには、サービスが利用不可能であることを通知するエラー・メッセージが送信されます。

JSL を再起動する

BEA Tuxedo は JSL を監視し、障害が発生した場合は JSL を再起動します。BEA Tuxedo がリスナ・プロセスを再起動すると、次のイベントが発生します。

- リスナに接続しようとするクライアントは、再接続する必要があります。ハンドラに接続しようとするクライアントでは、タイムアウトまたは遅延が発生します。
- ハンドラに接続中のクライアントは切断されます (対応する JSL が正常に終了すると、JSH は終了)。

JSL を設定する

JSL は、Jolt から JSH へ接続要求を分散する BEA Tuxedo サーバです。BEA Tuxedo は、JSL と JREPSVR が置かれているホスト・マシンで実行されていなければなりません。

注記　JSH に対する JSL のポートの選択方法は、BEA Tuxedo ワークステーション・リスナ (WSL: Workstation Server Listener) の場合のプロセスとは異なります。JSL ポートを正しく設定する方法については、3-46 ページの「UBBCONFIG ファイルを作成する」の「SERVERS セクション」を参照してください。

JSL のコマンド行オプション

サーバ側では、コマンド行からの情報の取得が必要な場合があります。CLOPT パラメータを使用すると、コマンド行オプションを指定して、サーバに設定されたデフォルト値を変更することができます。次の表では、JSL のコマンド行オプションを説明します。

表 3-3JSL のコマンド行オプション

オプション	説明
[-a]	Jolt 接続プールのセキュリティ・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、このオプションを設定して 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。-a オプションを設定しないと、SecurityContext が有効の場合、JSH はこのリクエストを受け付けません。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 の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
オプション	**説明**
[-c *compression_threshold*]	Jolt クライアントと Jolt サーバ (JSH) の間で送受信されるアプリケーション・データがネットワーク上で転送されるときに、圧縮されるようにします。 *compression_threshold* には、バイト数 (0 ～ 2,147,483,647) を指定します。指定されたしきい値を超えるメッセージは、転送前に圧縮されます。 デフォルトでは圧縮は行われません。つまり、圧縮のしきい値が指定されていないため、BEA Jolt によるクライアントまたはサーバ上のメッセージの圧縮は行われません。
[-d *device_name*]	トランスポート層インターフェイスを使用するプラットフォームのデバイスを指定します。デフォルト値はありません。必須のコマンド行オプションです (ただし、ソケットの場合は省略可能)。

表 3-3JSL のコマンド行オプション (続き)

[-H *external netaddr*]	ネットワーク・アドレスの変換が行われる場合に、Jolt クライアントがアプリケーションに接続するために使用するネットワーク・アドレス・マスクを指定します。JSL プロセスでは、このアドレスを使用して、このアドレスで接続しようとするクライアントをリッスンします。外部アドレス・マスクが 0x0002MMMMdddddddd で、JSH のネットワーク・アドレスが 0x00021111ffffffff の場合、結果 (外部) のネットワーク・アドレスは 0x00021111dddddddd になります。先頭に「//」が付いたネットワーク・アドレスは、IP ベースであることを示し、JSH ネットワーク・アドレスから TCP/IP ポート番号がコピーされて、新しいネットワーク・アドレスを構成します。 外部 IP アドレス・マスクは、次の形式で指定されます。 -H //external ip address:MMMM
[-I *init-timeout*]	Jolt クライアントが JSH を介して初期化を完了するまでの時間 (秒単位) を指定します。この時間を過ぎると、JSL によるタイムアウトが発生します。デフォルト値は 60 秒です。(省略可能)。

表 3-3JSL のコマンド行オプション (続き)

オプション	説明
[-j *connection_mode*]	コネクション・モードの種類は次のとおりです。 RETAINED －セッションの確立中、完全にネットワーク接続は保持されます。 RECONNECT －クライアントが接続を確立した後でアイドル状態が続き、タイムアウトが発生したために接続が切断された場合に、複数の要求を受け取るとセッション中に再接続を行います。 ANY －サーバは、クライアントが RETAINED または RECONNECT の接続をセッションで確立することを許可します。 デフォルトは ANY です。つまり、オプションが何も指定されていない場合、サーバはクライアントが RETAINED または RECONNECT の接続を要求することを許可します (省略可能)。
[-m *minh*]	一度に JSL と共に利用できる JSH の最小数を指定します。このパラメータに指定できる値の範囲は 0 ～ 255 です。デフォルトは 0 です (省略可能)。
[-M *maxh*]	一度に JSL と共に利用できる JSH の最大数を指定します。このオプションを指定しない場合、このパラメータにはデフォルト値として、MAXWSCLIENTS を -x 多重係数 (小数点以下切り上げ) で割った値が指定されます。オプションを指定する場合、-M オプションには 1 ～ 32,767 の値を指定できます (省略可能)。

表 3-3JSL のコマンド行オプション (続き)

オプション	説明
[-n *netaddr*]	BEA Tuxedo 6.4、BEA Tuxedo 6.5、および WebLogic Enterprise 4.2 で BEA Jolt リスナが使用するネットワーク・アドレスを指定します。 TCP/IP アドレスは、次のいずれかの形式で指定されます。 "*//host.name:port_number*" "*//#.#.#.#:port_number*" 最初の形式の場合、ドメインはローカル名を解決する手法 (通常は DNS) を使用して *hostname* のアドレスを検索します。*hostname* にはローカル・マシン名を指定し、名前解決の機能でローカル・マシンのアドレスに明確に解決されなければなりません。 2 番目の形式の場合、「#.#.#.#」にはドット区切りの 10 進数を指定します。ドットで区切った 10 進数の形式では、各 # は 0 から 255 までの数でなければなりません。このドットで区切った 10 進数は、ローカル・
[-T *Client-timeout*]	クライアントがアイドル状態でいられる期間 (分単位) を指定します。ここで指定した期間内にクライアントから要求が発行されないと、JSH によりクライアントの接続は切断され、セッションは終了します。引数が指定されていない場合、セッションでタイムアウトは発生しません。 -j ANY オプションまたは -j RECONNECT オプションを使用する場合は、常に -T でアイドル・タイムアウト値を指定します。-T が指定されず、接続が一時停止になっている場合、JSH が自動的にセッションを終了することはありません。クライアントがセッションを異常終了すると、セッションは完全に終了しません。 パラメータを指定しないと、デフォルトではタイムアウトの指定なしになります (省略可能)。
[-w *JSH*]	このコマンド行オプションは、Jolt サーバ・ハンドラを示します。デフォルトは JSH です (省略可能)。

表 3-3JSL のコマンド行オプション (続き)

オプション	説明
[-x *mpx-factor*]	1 つの JSH で扱えるクライアントの数を指定します。このパラメータを使用して、各 JSH プロセスでの多重化のレベルを制御してください。UNIX および Windows 2000 の場合、このパラメータには 1 ～ 32767 の範囲の値を指定できます。デフォルト値は 10 です (省略可能)。
[-Z 0|56|128]	Jolt クライアントと JSH の間にネットワーク・リンクが確立されている場合、このオプションを使用して特定のレベルまでの暗号化を行うことができます。初期値の 0 は、DH ノードの暗号化も RC4 形式の暗号化も行わないことを示します。56 および 128 は、暗号化キーの長さ (ビット単位) を示します。キーを生成するには、DH 形式による鍵暗号が必要です。セッション・キーは、ネットワーク経由で転送されません。デフォルト値は 0 です。

セキュリティ機能と暗号化

認証データと鍵暗号データは、Diffie-Hellman (DH) 方式の鍵暗号機能で暗号化され、Jolt クライアントと JSL/JSH の間で送受信されます。以降のすべての鍵暗号は、RC4 形式で暗号化されます。国際版のパッケージでは、DES 形式の鍵暗号と 128 ビットの暗号化キーを使用します。128 ビットのうち、40 ビットが暗号化され、88 ビットがエクスポーズされます。

128 ビットの暗号化を使ったプログラムは、米国政府の正式な承認なく米国以外の国へ輸出することはできません。社内のイントラネットが米国以外の国にも拡張されており、海外にも社内クライアントがいる場合は、この暗号化を使用することはできません。

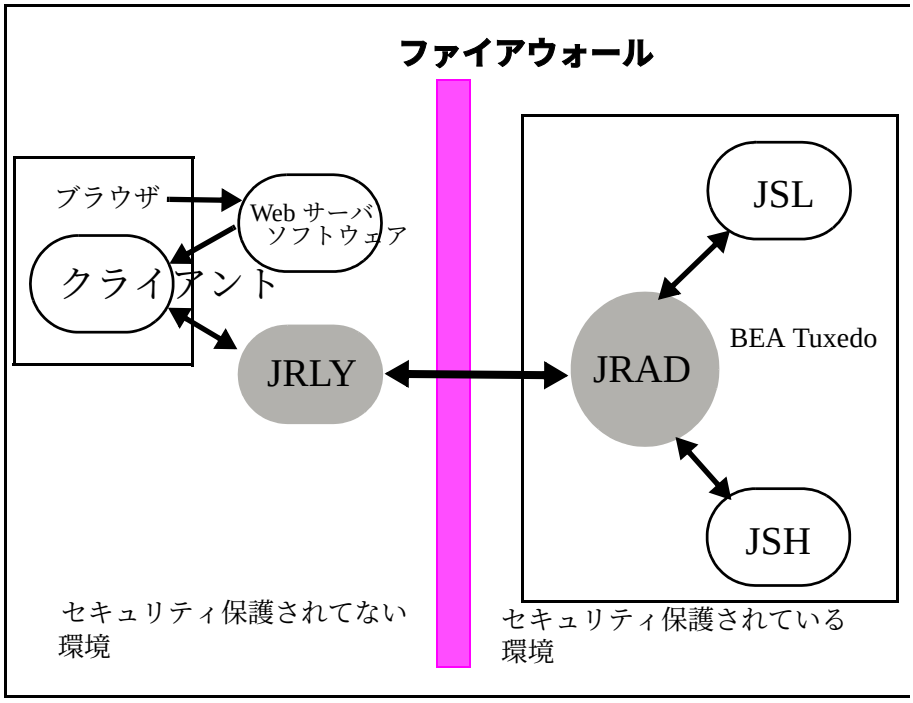
Jolt リレー

Jolt リレー (JRLY) と Jolt リレー・アダプタ (JRAD) 間の動作を、通常インターネット・リレーと呼びます。Jolt リレーは、Jolt クライアントから JSL や JSH にメッセージをルーティングします。これにより、JSH と BEA Tuxedo を、(安全性に問題があると見なされている) Web サーバと同じマシンで実行する必要がなくなります。Jolt リレーは、3-23 ページの「Jolt インターネット・リレー」で示すように 2 つのコンポーネントで構成されています。

- Jolt リレー (JRLY: Jolt Relay) ― JRLY は、Jolt リレーのフロント・エンドです。JRLY は BEA Tuxedo のクライアントでもサーバでもなく、BEA Tuxedo のバージョンにも依存しません。スタンドアロンのソフトウェア・コンポーネントです。Jolt クライアントと共に動作させるためのコンフィギュレーションは最小限で済みます。
- Jolt リレー・アダプタ (JRAD: Jolt Relay Adapter) ― JRAD は、Jolt リレーのバックエンドです。これは BEA Tuxedo システム・サーバですが、BEA Tuxedo サービスは含まれていません。これを JSL および BEA Tuxedo とともに動かすためにはコマンド・ライン引数が必要です。

注記　Jolt リレーは、Jolt クライアントおよび Jolt サーバからは見えません。Jolt サーバは、複数のイントラネット・クライアントに同時に直接接続したり、Jolt リレーを介してインターネット・クライアントに接続できます。

図 3-3Jolt インターネット・リレー



この図では、ブラウザから Web サーバ・ソフトウェアに接続し、BEA Jolt アプレットをダウンロードする様子を示しています。まず、Jolt のアプレットまたはクライアントは、Web サーバ・マシン上の JRLY に接続します。次に、JRLY は、ファイアウォールを越えて Jolt メッセージを JRAD に転送します。さらに、JRAD はメッセージを JSL または適切な JSH に転送します。

Jolt リレーのフェイルオーバー

JRLY のフェイルオーバーには、次の 2 つがあります。

- Jolt クライアントから JRLY への接続時に発生するフェイルオーバー
- JRLY から JRAD への接続時に発生するフェイルオーバー

Jolt クライアントから JRLY への接続時に発生するフェイルオーバー

1 つのサーバ・アドレスがセッションで失敗すると、フェイルオーバー機能がはたらきます。この機能により、Jolt クライアントの API は、次に接続可能な (まだ接続されていない) JRLY を API の引数一覧から検索して接続します。Windows 2000 環境でこのフェイルオーバー機能を有効にするには、複数の Windows 2000 JRLY サービスを実行します。Windows 2000 以外の環境では、JRLY プロセスが複数実行されています。各 JRLY (サービスまたはプロセス) には、固有のコンフィギュレーション・ファイルが用意されています。この種のフェイルオーバー処理は、BEA Jolt のクライアント API 機能により行われ、ユーザは Jolt サーバ・アドレス (JSL または JRLY) の一覧を指定することができます。

JRLY から JRAD アダプタへの接続時に発生するフェイルオーバー

各 JRLY のコンフィギュレーション・ファイルには、JRAD アドレスの一覧が用意されています。JRAD が利用できない場合、JRLY は次に利用可能な (接続されていない) JRAD をラウンド・ロビン方式で検索して接続しようとします。2 つの JRLY から同じ JRAD に接続することはできません。これらの条件を利用し、JRAD アドレスの順序を変えて効率的に接続を確立することができます。つまり、予備の JRAD を待機させておき、JRLY から JRAD への最初の接続が切断されたら、予備の JRAD に接続されるようにします。この種のフェイルオーバー処理は、JRLY のみで行われます。

JRLY の起動時に一覧内の JRAD がどれも実行されていない場合、最初の接続は失敗します。Jolt クライアントが JRLY に接続しようとすると、JRLY は、再び JRAD に接続しようとします。

フェイルオーバー機能を有効にするには、`UBBCONFIG` ファイルで JRAD を設定し、複数の JRAD を起動する必要があります。

Jolt リレーのプロセス

JRLY (フロントエンド・リレー) のプロセスは、JRAD の起動前または起動後のどちらかの時点で始まります。JRLY の起動時に JRAD が利用できない場合、JRLY はクライアントからの要求を受信する時点で JRAD に接続を試みます。クライアントからの要求受信時にも JRAD に接続できない場合、クライアントはアクセスを拒否され、JRLY のエラー・ログ・ファイルに警告メッセージが書き込まれます。

UNIX で JRLY を起動する

システム・プロンプトで次のコマンドを入力し、JRLY プロセスを開始します。

```
jrly -f config_file_path
```

コンフィギュレーション・ファイルが存在しないか、または開けない場合、JRLY はエラー・メッセージを出力します。

JRLY を起動できない場合、JRLY は標準エラーにメッセージを書き込み、起動時のエラーをエラー・ログに記録してから終了します。

JRLY コマンド行オプション (Windows 2000)

この節では、`JRLY.exe` の Windows 2000 バージョンで利用できるコマンド行オプションについて説明します。次のような制限事項があります。

- Windows サービスとしての JRLY は、Windows 2000 に対してのみ利用可能です。
- 表示接尾辞をオプションで指定できる場合 ([*display_suffix*] が表示されている場合)、すべての操作はデフォルトの JRLY Windows 2000 サービス・インスタンスで実行されます。
- 別の JRLY サービスを手動でインストールする場合、接尾辞 (任意の文字列) が必要です。オプションの文字列の接尾辞を省略して、デフォルトのサービスを手動でインストールすることもできます。
- JRLY Windows 2000 サービスの各インスタンスでは、同じバイナリ形式の実行可能ファイルが使用されます。
- JRLY Windows 2000 サービスのインスタンスごとに新しいプロセスが開始されます。
- コマンド行オプションの構文は、`jrly -command` です。
- 角かっこ ([ ]) で囲まれた文字列はオプションです。

- 次の表にあるコマンド行オプションのうち、`-start` と `-stop` 以外は、Windows 2000 レジストリに対する書き込み権が必要です。
- `-start` と `-stop` を使用する場合は、Windows 2000 サービスの制御権が必要です。これらの制限は、Windows 2000 でのユーザ制限に基づいています。

JRLY のコマンド行オプションを次の表で詳しく説明します。

表 3-4JRLY コマンド行オプション (Windows 2000)

オプション	説明

`jrly -install` [display_suffix]	`jrly` を Windows 2000 サービスとしてインストールします。
	
	例 1
	`jrly -install`
	このコマンドを実行すると、デフォルトの JRLY が Windows 2000 サービスとしてインストールされ、サービス・コントロール・マネージャ (SCM: Service Control Manager) に Jolt リレーとして表示されます。
	
	例 2
	`jrly -install MASTER`
	このコマンドを実行すると、JRLY のインスタンスが Windows 2000 サービスとしてインストールされ、SCM に Jolt Relay_MASTER として表示されます。接尾辞の MASTER は、さまざまな JRLY のインスタンスを一意に識別するために使用されており、特に意味はありません。
	
	この時点の JRLY のインスタンスは、まだ開始できません。コンフィギュレーション・ファイルを割り当て (`set` コマンドの説明を参照)、接続指示を受け付ける TCP/IP ポート、JSH の接続を行う TCP/IP ポート、ログ・ファイル、および `sockettimeout` を指定する必要があります。JRLY の複数のインスタンスがコンフィギュレーション・ファイルを共有することはできません。

表 3-4JRLY コマンド行オプション (Windows 2000) (続き)

オプション	説明

`jrly -remove [display_suffix] | -all`	Windows 2000 サービスから 1 つまたはすべての JRLY のインスタンスを削除します。
	[*display_suffix*] を指定すると、指定された JRLY サービスが削除されます。
	[*display_suffix*] を指定しないと、Windows 2000 サービスとしてのデフォルトの JRLY が削除されます。
	`-all` オプションを指定すると、すべての JRLY Windows 2000 サービスが削除されます。
	`HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\BEA JoltRelay`
	および
	`HKEY_LOCAL_MACHINE\Software\BEA Systems\Jolt\x.x`
	に置かれている Windows 2000 の関連レジストリは削除されます。

表 3-4JRLY コマンド行オプション (Windows 2000) (続き)

オプション	説明

`jrly -set [-d display_suffix] -f config_file`	新しいコンフィギュレーション・ファイルの絶対パスを指定して、レジストリを更新します。
	
	例 1
	`jrly -set -f c:\tux71\udataobj\jolt\jrly.con`
	このコマンドを実行すると、デフォルトの JRLY Windows 2000 サービス (Jolt リレー) に `jrly.con` という名前のコンフィギュレーション・ファイルが割り当てられ、`c:\tuxdir\udataobj\jolt` ディレクトリに置かれます。
	
	例 2
	`jrly -set -d MASTER -f c:\tuxdir\udataobj\jolt\master.con`
	
	このコマンドを実行すると、Jolt Relay_MASTER という JRLY Windows 2000 サービスのインスタンスに `jrly_master.con` というコンフィギュレーション・ファイルが割り当てられ、`c:\tuxdir\udataobj\jolt` ディレクトリに置かれます。

`jrly -manual [display_suffix]`	処理の開始または終了を手動で行うように設定します。 このコマンドを実行すると、コマンド行オプションまたは SCM を使用して、指定された JRLY インスタンスを手動で制御できるように設定されます。
`jrly -auto [display_suffix]`	処理の開始または終了を自動で行うように設定します。 このコマンドを実行すると、OS の起動時およびシャットダウン時に、指定した Windows 2000 サービスに対するすべての操作が自動的に開始するように設定されます。

表 3-4JRLY コマンド行オプション (Windows 2000) (続き)

オプション	説明
`jrly -start` [display_suffix]	指定した JRLY を開始します。
`jrly -stop` [display_suffix]	指定した JRLY を停止します。
`jryl -version`	JRLY バイナリの現在のバージョンを出力します。
`jrly -help`	コマンド行オプション (簡単な説明付き) を出力します。

JRLY コマンド行オプション (UNIX)

UNIX 用の JRLY のコマンド行オプションは、次の 1 つだけです。

表 3-5JRLY コマンド行オプション (UNIX)

オプション	説明
`jrly -f` *`config_file_path`*	JRLY プロセスを開始します。 このコマンドを実行すると、JRLY プロセスが開始します。コンフィギュレーション・ファイルが存在しないか、または開けない場合、JRLY はエラー・メッセージを出力します。JRLY を起動できない場合、JRLY は標準エラーにメッセージを書き込み、起動時のエラーをエラー・ログに記録してから終了します。

JRLY コンフィギュレーション・ファイル

コンフィギュレーション・ファイルは、「タグ = 値」の形式で記述されています。空白行または「#」で始まる行は無視されます。次のリストは、正式なコンフィギュレーション・ファイルの形式を示しています。

コード リスト 3-3 コンフィギュレーション・ファイルの内容

```
LOGDIR=<LOG_DIRECTORY_PATH>
ACCESS_LOG=<ACCESS_FILE_NAME in LOGDIR>
ERROR_LOG=<ERROR_FILE_NAME in LOGDIR>
LISTEN=<IP:Port combination where JRLY will accept connections>
CONNECT=<IP:Port combination associated with JRAD>
SOCKETTIMEOUT=<Seconds for socket accept()function>
```

注記　`SOCKETTIMEOUT` は、リレー Windows 2000 サービスがネットワーク・アクティビティ (新しい接続、読み込み対象のデータ、クローズされた接続など) を実現するために、新しいソケット接続の確立をブロッ

クする期間を示す時間 (秒単位) です。これは、Windows 2000 マシンでのみ有効です。SOCKETTIMEOUT の値は、SCM にも影響します。SCM からサービスの停止が要求されると、SCM は少なくとも SOCKETTIMEOUT で指定した秒数を待つ必要があります。

次のリストは、JRLY のコンフィギュレーション・ファイルの例です。CONNECT で始まる行は、JRAD マシンの IP アドレスとポート番号を指定します。

コード リスト 3-4 JRLY **コンフィギュレーション・ファイルの例**

```
LOGDIR=/usr/log/relay
ACCESS_LOG=access_log
ERROR_LOG=errorlog
# jrly will listen on port 4444
LISTEN=200.100.10.100:4444
CONNECT=machine1:port1
CONNECT=machine2:port2

SOCKETTIMEOUT=30              // リストの下のテキストを参照
```

ディレクトリ名とファイル名の形式は、オペレーティング・システムによって異なります。UNIX システムではスラッシュ (/) を使用します。Windows 2000 システムでは円記号 (\) を使用します。LOGDIR、ACCESS_LOG、または ERROR_LOG で指定されたファイルを開けない場合、JRLY は stderr にエラー・メッセージを記録してから終了します。

次の表は、ホスト名とポート番号の形式を示しています。

表 3-6 ホスト名とポート番号の形式

ホスト名 / ポート番号	説明
Hostname:Port	*Hostname* は文字列で指定し、*Port* は 10 進数で指定します。
//Hostname:Port	*Hostname* は文字列で指定し、*Port* は 10 進数で指定します。
IP:Port	*IP* にはドット区切りの IP アドレス表記を指定し、*Port* には 10 進数を指定します。

Jolt リレー・アダプタ

Jolt リレー・サーバ・アダプタ (バックエンド・リレー) は、BEA Tuxedo システム・サーバです。Jolt リレー・アダプタ (JRAD) の配置場所は、JSL サーバの接続先である BEA Tuxedo のホスト・マシン (シングル・ホスト・モード (SHM)) およびサーバ・グループでなくてもかまいません。

JRAD は、関連する JRLY とは独立して起動できます。JRAD は、起動およびシャットダウンのアクティビティを BEA Tuxedo ログ・ファイルからトラッキングします。

JRAD の設定

1 つの JRLY に接続できる JRAD プロセスは 1 つだけです。JRAD は、1 つの JSL および関連する JSH とだけ通信するように設定できます。複数の JRAD を 1 つの JSL と通信するように設定することもできます。`UBBCONFIG` ファイルには、BEA Tuxedo サーバ用の `CLOPT` パラメータを含める必要があります。3-37 ページの「UBBCONFIG ファイル内の JRAD エントリの例」のコンフィギュレーション・ファイルの例を参照してください。

次の表では、`CLOPT` パラメータに関する追加情報を示します。

表 3-7JRAD の CLOPT パラメータの説明

CLOPT パラメータ	説明
`-l` *`netaddr`*	クライアントの代わりに接続を行う JRLY をリッスンするポートを指定します。
`-c` *`netaddr`*	JRAD の接続先 JSL のアドレスを指定します。

表 3-7JRAD の CLOPT パラメータの説明 (続き)

CLOPT パラメータ	説明
`-H` *`netaddr`*	外部プロキシの接続指示受け付けアドレスを指定します。外部プロキシはクライアントのホストで実行するプロキシです。このプロキシは HTTP プロトコルやその他のプロトコルを処理します。プロキシの他方の側は JRLY に接続し、JRLY は JSL/JSH に接続します。 プロキシが Jolt クライアント (特に、JRLY に接続するアプレット) の代わりに動作するように、JRAD は `-H` 引数をアプレットに渡し、JRLY アドレスではなくプロキシ・アドレスに接続するよう指示します。 注記　JSL の `-H` オプションと異なり、JRAD の `-H` オプションは、ネットワーク・アドレスの変換には使用されず、またアドレス・マスクとしても使用されません。

JRAD CLOPT パラメータのアドレスは、次のいずれかの形式で指定できます。

`//`hostname:port

`0x0002ppppphhhhhhhh`
`pppp` はポート番号、`hhhhhhhh` は 16 進数の IP アドレスを示します。

コード リスト 3-5　UBBCONFIG **ファイル内の** JRAD **エントリの例**

```
# JRAD host 200.100.100.10 listens at port 2000, connects to JSL
port 8000 on the same host

JRAD    SRVGRP=JSLGRP   SRVID=60
        CLOPT="-A -- -l 0x000207D0C864640A –c 0x00021f40C864640A"
```

ネットワーク・アドレスの設定

Jolt インターネット・リレーを設定するには、ネットワーク接続されたいくつかのコンポーネントを動作させる必要があります。設定を行う前に次の表に示す条件を確認し、情報を書き留め、間違った設定を行わないようにしてください。

表 3-8Jolt インターネット・リレーのネットワーク・アドレスの設定条件

JRLY	JRAD	JSL
LISTEN: クライアントの接続先を指定します。 CONNECT:JRAD の場所を指定します。JRAD の -l パラメータと一致していなければなりません。	-l: リスナが JRLY に接続する場所を指定します。 -c:JSL の場所を指定します。JSL の -n パラメータと一致していなければなりません。	-n:JSL の場所を指定します。JRAD の -c パラメータと一致していなければなりません。

Jolt リポジトリ

Jolt リポジトリには、BEA Tuxedo サービスの定義が格納されています。この定義情報により、Jolt クライアントは BEA Tuxedo サービスにアクセスすることができます。インストール時にダウンロードされる Jolt リポジトリのファイル群には、BEA Jolt で内部的に使用されるサービス定義が含まれています。アプリケーション・サービスに定義を追加する方法の詳細については、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

Jolt リポジトリを設定する

BEA Jolt リポジトリを設定するには、アプリケーションの UBBCONFIG ファイルを編集してください。UBBCONFIG ファイルは、ASCII 形式の BEA Tuxedo コンフィギュレーション・ファイルです。アプリケーションごとに別の UBBCONFIG ファイルを作成してください。ファイル内のエントリの構文については、『BEA Tuxedo コマンド・リファレンス』を参照してください。次のリストは、UBBCONFIG ファイルの一部を示しています。

コード リスト 3-6 UBBCONFIG **ファイルの例**

```
*GROUPS
JREPGRP         GRPNO=94 LMID=SITE1
*SERVERS
JREPSVR SRVGRP=JREPGRP SRVID=98
RESTART=Y GRACE=0 CLOPT="-A -- -W -P /app/jrepository"
JREPSVR SRVGRP=JREPGRP SRVID=97
RESTART=Y RQADDR=JREPQ GRACE=0 CLOPT="-A -- -P /app/jrepository"
JREPSVR SRVGRP=JREPGRP SRVID=96
RESTART=Y RQADDR=JREPQ REPLYQ=Y GRACE=0 CLOPT="-A -- -P
/app/jrepository"
```

注記　UNIX システムで jrepository ファイルのパスを設定する場合は、スラッシュ (/) を使用してください (例 : app/repository)。Windows 2000 システムの場合は円記号 (\) を使用し、ドライブ名を指定してください (例 : c:\app\repository)。

次の表に従って、UBBCONFIG ファイル内のセクションを変更してください。

表 3-9UBBCONFIG ファイル

セクション	指定するパラメータ
GROUPS	LMID、GRPNO
SERVERS	SRVGRP、SRVID

GROUPS セクション

GROUPS エントリを使用して、BEA Jolt リポジトリを含むグループを設定します。グループ名のパラメータには、アプリケーション側で選択された名前が使用されます。

1. MACHINES セクションの LMID パラメータに指定されている値と同じ識別子を指定します。
2. GROUPS セクションの GRPNO に 1 ～ 30,000 の範囲の値を指定します。

SERVERS セクション

Jolt リポジトリ・サーバである JREPSVR には、リポジトリにアクセスしたり、リポジトリを編集するためのサービスが格納されています。JREPSVR インスタンスが複数ある場合は、共有ファイルにより、リポジトリの情報が共有されます。UBBCONFIG ファイルの SERVERS セクションには、JREPSVR を指定してください。

1. SRVID パラメータに新しいサーバの識別子を指定します (例 : 98 など)。
2. 1 つの JREPSVR に対して -W フラグを設定し、リポジトリを編集可能にします。このフラグを設定しないと、リポジトリは読み取り専用になります。

注記　インストールする必要があるのは、書き込み可能な 1 つの JREPSVR (-W フラグを指定した JREPSVR) だけです。読み取り専用に設定された複数の JREPSVR を同じホストにインストールすることもできます。

3. -P フラグを指定して、リポジトリ・ファイルへのパスを設定します。-P フラグに対する引数が指定されていないと、BEA Tuxedo の ULOG ファイルにエラー・メッセージが表示されます。
4. リポジトリ・ファイルのファイル・パス名を追加します (例 : /app/jrepository)。
5. tmloadcf コマンド (例 : tmloadcf -y ubbconfig) や tmboot コマンドを使用して、BEA Tuxedo システムを起動します。tmloadcf および tmboot については、『BEA Tuxedo アプリケーション実行時の管理』を参照してください。

リポジトリ・ファイル

リポジトリ・ファイル jrepository は、BEA Jolt で利用可能です。このファイルには bankapp のサービスとリポジトリ・サービスが含まれています。これらのサービスは、リポジトリ・エディタを使って編集したり、テストを行ったり、削除することができます。

注記　BEA Jolt バージョン 1.x からアップグレードする場合、現在のバージョンとの互換性を保つために、バルク・ローダを使って jrepository ファイルを作成し直す必要があります。

BEA Jolt で bankapp アプリケーションのテストを行わない場合も、インストール時に提供される jrepository ファイルから始めてください。bankapp のパッケージまたはサービスが必要でない場合は削除してください。

ファイルのパス名は、-P オプションの引数と一致していなければなりません。

警告 :　リポジトリ・ファイルを手動で変更しないでください。手動で変更すると、リポジトリ・エディタを使用できなくなります。jrepository ファイルの変更はどんなテキスト・エディタでも行うことができますが、BEA Jolt システムでは正当性チェック (

ファイルが正しい形式かどうかを確認) は行われません。`jrepository` ファイルが手動で変更されたかどうかは、実行時までわかりません。その他の情報については、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

BEA Tuxedo およびリポジトリ・エディタを使用してサービスを初期化する

BEA Tuxedo および BEA Jolt リポジトリ・エディタを使用して次の手順で BEA Tuxedo サービスを定義し、クライアントが Jolt サービスを利用できるようにします。

1. サービスを格納した BEA Tuxedo サーバを作成します。以下の情報については、『BEA Tuxedo アプリケーション実行時の管理』または『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』を参照してください。
   - BEA Tuxedo アプリケーション・サーバを作成する
   - `UBBCONFIG` ファイルを編集する
   - `TUXCONFIG` ファイルを更新する
   - `tmboot` コマンドを使用する
2. BEA Jolt リポジトリ・エディタにアクセスします。以下に関する情報については、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。
   - サービスを追加する
   - 作業を保存する
   - サービスをテストする
   - サービスのエクスポートとアンエクスポート

イベント・サブスクリプション

Jolt のイベント・サブスクリプション機能では、BEA Tuxedo サービスまたは別の BEA Tuxedo クライアントからイベント通知を受信します。

- 任意通知型イベント通知ー Jolt クライアントがこれらの通知を受信するのは、BEA Tuxedo のクライアントまたはサービスが任意通知型メッセージ・イベントをサブスクライブしており、BEA Tuxedo クライアントがブロードキャストを発行するとき (`tpbroadcast()` を使用するか、または `tpnotify()` ATMI 呼び出しを使用して直接目的のメッセージを送信) です。任意通知型通知には、`TMUSREVT` サーバは不要です。
- ブローカ経由のイベント通知ー Jolt クライアントは、BEA Tuxedo のイベント・ブローカ経由でこれらの通知を受信します。これらの通知は、両方のクライアントが同じイベントをサブスクライブしており、任意の BEA Tuxedo クライアントまたはサーバが `tppost()` を使用してイベントをポストするときにのみ受信されます。ブローカ経由のイベント通知には、`TMUSREVT` サーバが必要です。

イベント・サブスクリプションを設定する

BEA Tuxedo の `TMUSREVT` サーバを設定し、アプリケーションの `UBBCONFIG` ファイルを編集してください。次のリストは、`UBBCONFIG` ファイル内の `TMUSREVT` パラメータの内容を示しています。コンフィギュレーション・ファイル内のエントリの構文については、『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』を参照してください。

コード リスト 3-7　UBBCONFIG **ファイル**

```
TMUSREVT        SRVGRP=EVBGRP1  SRVID=40        GRACE=3600
                ENVFILE="/usr/tuxedo/bankapp/TMUSREVT.ENV"
                CLOPT="-e tmusrevt.out -o tmusrevt.out -A --
                -f /usr/tuxedo/bankapp/tmusrevt.dat"
                SEQUENCE=11
```

UBBCONFIG ファイルの SERVERS セクションで、SRVGRP パラメータと SRVID パラメータを必要に応じて変更してください。

BEA Tuxedo の FML バッファまたは VIEW バッファをフィルタ処理する

フィルタ処理を行うと、サブスクリプションをカスタマイズできます。BEA Tuxedo のイベント・ブローカ、イベントのサブスクライブ方法、またはフィルタ処理に関する追加情報については、『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』を参照してください。

BEA Tuxedo の FML バッファまたは VIEW バッファをフィルタ処理するには、BEA Tuxedo の実行時にフィールド定義ファイルが必要です。

注記　STRING バッファをフィルタ処理する場合の条件は特にありません。

バッファ型

表 3-10BEA Tuxedo のバッファ型

バッファ型	説明
FML	属性と値の組み合わせ。明示的です。
VIEW	C 構造体。詳細なオフセットです。暗黙的です。

表 3-10BEA Tuxedo のバッファ型

バッファ型	説明
STRING	長さとオフセットは異なる値です。すべて読み取り可能です。
CARRAY	文字配列。BLOB のバイナリ・データです。クライアントとサーバ側だけが認識している情報であり、JSL 側は認識していません。
X_C_TYPE	VIEW と同じです。
X_COMMON	VIEW と同じですが、COBOL と C の両方で使用されます。
X_OCTET	CARRAY と同じです。
XML	形式の整った XML 文書。CARRAY に似ています。

FML バッファの例

3-45 ページの「TMUSREVT.ENV ファイルの FIELDTBLS 変数」は、FML バッファの使用例を示しています。FIELDTBLS 変数と FLDTBLDIR 変数を設定することにより、BEA Tuxedo で FML フィールドの定義テーブルを利用できるようになります。

my.flds ファイルのフィールドをフィルタ処理するには、次の手順に従います。

1. my.flds ファイルを /usr/me/bankapp ディレクトリにコピーします。
2. 次のリストに示すように、TMUSREVT.ENV ファイルの FIELDTBLS 変数に my.flds を追加します。

コード リスト 3-8　TMUSREVT.ENV **ファイルの** FIELDTBLS **変数**

```
FIELDTBLS=Usysflds,bank.flds,credit.flds,event.flds,my.flds
FLDTBLDIR=/usr/tuxedo/me/T6.2/udataobj:/usr/me/bankapp
```

UBBCONFIG ファイルで ENVFILE="/usr/me/bankapp/TMUSREVT.ENV" と定義されている場合 (3-43 ページの「UBBCONFIG ファイル」を参照)、FIELDTBLS と FLDTBLDIR の定義は、設定されている環境変数の代わりに TMUSREVT.ENV ファイルから取得されます。

ENVFILE="/usr/me/bankapp/TMUSREVT.ENV" の定義を削除すると、FIELDTBLS と FLDTBLDIR の定義は、設定されている環境変数から取得されます。BEA Tuxedo システムを起動する前には、FIELDTBLS と FLDTBLDIR の定義に適切な値が設定されていなければなりません。

イベント・サブスクリプションと BEA Jolt クラス・ライブラリの詳細については、第 5 章「Jolt クラス・ライブラリを使う」を参照してください。

BEA Tuxedo に関する背景情報

次の節では、コンフィギュレーションに関する情報を詳しく説明します。BEA Tuxedo について理解している場合でも、Jolt サービス・ハンドラ (JSL: Jolt Service Handler) の設定についてこの節で確認してください。

コンフィギュレーション・ファイル

BEA Tuxedo のコンフィギュレーション・ファイルには、ASCII 形式の UBBCONFIG と、コンパイル済みの TUXCONFIG の 2 種類があります。TUXCONFIG ファイルを作成したら、UBBCONFIG はバックアップとして保存してください。

UBBCONFIG ファイルは、使い慣れたテキスト・エディタを使用して変更することができます。MASTER マシンにログインしているときにアプリケーションが動作しなくなったら、tmloadcf(1) を実行して TUXCONFIG をコンパイルし直します。BEA Tuxedo からは、TUXCONFIG ファイルを上書きしてもよいかどうかを確認するプロンプトが表示されます (-y オプションを指定してコマンドを実行すると、このプロンプトは表示されません)。

UBBCONFIG ファイルを作成する

バイナリ形式のコンフィギュレーション・ファイル、TUXCONFIG には、tmboot(1) の実行時に使用される情報が含まれています。これにより、サーバの起動と BEA Tuxedo システムの掲示板の初期化が順番に行われます。バイナリ形式の TUXCONFIG ファイルを直接作成することはできません。まず、UBBCONFIG ファイルを作成する必要があります。tmloadcf(1) を実行すると、このファイルが解析され、TUXCONFIG に読み込まれます。次に、tmadmin(1) により、コンフィギュレーション・ファイルまたはそのコピーを使ったシステムの監視が行われます。tmshutdown(1) は、アプリケーションのシャットダウン時に必要な情報をコンフィギュレーション・ファイルから参照します。

コンフィギュレーション・ファイルの形式

UBBCONFIG ファイルには、最大 9 つまでのセクションを指定することができます。セクションは、アスタリスク (*) が先頭に付いた行から始まります。アスタリスク (*) の直後にはセクション名が表示されます。使用可能なセクションは、RESOURCES、MACHINES、GROUPS、NETGROUPS、NETWORK、SERVERS、SERVICES、INTERFACES、および ROUTING です。

注記　RESOURCES セクション (使用する場合) と MACHINES セクションは、この順序で最初に指定しなければなりません。GROUPS セクションは、SERVERS セクション、SERVICES セクション、および ROUTING セクションより前に指定しなければなりません。

JSL を設定するには、UBBCONFIG ファイルを変更する必要があります。BEA Tuxedo の設定に関するさらに詳しい情報については、『BEA Tuxedo アプリケーション実行時の管理』を参照してください。

次のリストは、UBBCONFIG ファイルの一部を示しています。

コード リスト 3-9　UBBCONFIG **ファイル**

```
*MACHINES
MACH1  LMID=SITE1
       MAXWSCLIENTS=40
*GROUPS
JSLGRP        GRPNO=95   LMID=SITE1
*SERVERS
JSL SRVGRP=JSLGRP SRVID=30 CLOPT= “ -- -n 0x0002PPPPNNNNNNNN -d
/dev/tcp -m2 -M4 -x10”
```

次の表では、Jolt サーバ・グループと Jolt サーバに対して指定できるパラメータを示しています。これら以外のパラメータを指定する必要はありません。

次の表に従って、UBBCONFIG ファイル内のセクションを変更してください。

表 3-11UBBCONFIG ファイル内のセクション

セクション	指定するパラメータ
MACHINES	MAXWSCLIENTS
GROUPS	GRPNO、LMID
SERVERS	SRVGRP、SRVID、CLOPT

MACHINES セクション

MACHINES セクションでは、物理マシンの論理名を指定します。また、このセクションではマシン固有のパラメータも指定します。MACHINES セクションには、アプリケーションで使用される物理プロセッサごとのエントリが必要です。エントリの形式は次のとおりです。

ADDRESS または *NAME* 必須パラメータ [オプション・パラメータ]

ADDRESS はプロセッサの物理名です。たとえば、UNIX システムの uname -n コマンドの実行結果として返される値などです。

LMID=*string_value*

このパラメータは、ADDRESS のシンボリック名として、ほかのセクションで *string_value* が使用されることを指定します。この名前にはカンマを指定できません。名前は 30 文字以内で指定します。このパラメータは必須です。コンフィギュレーションで使用されるすべてのマシンには、LMID 行を指定する必要があります。

MAXWSCLIENTS=*number*

コンフィギュレーション・ファイルの MACHINES セクションには、MAXWSCLIENTS パラメータを指定する必要があります。これは、プロセッサにおけるアクセサ数を指定する、Jolt クライアントと Workstation クライアント専用のパラメータです。このパラメータには、0 ～ 32,768 の範囲の値を指定します。

Jolt サーバと Workstation では、同じ要領で MAXWSCLIENTS が使用されます。たとえば、MAXWSCLIENTS に 200 スロットが設定されると、Jolt と Workstation で使用されるリモート・クライアントの総数が決まります。

コンフィギュレーション・ファイルの MAXWSCLIENTS は必ず指定してください。指定しない場合は、デフォルトで 0 が設定されます。

注記　MAXWSCLIENTS が設定されていないと、JSL は起動しません。

GROUPS セクション

このセクションでは、サーバ・グループに関する情報を定義します。サーバ・グループは少なくとも 1 つ定義しなければなりません。サーバ・グループのエントリには、サーバ群およびマシン上のサービス群に対して、論理名を指定します。論理名は、SERVERS セクションの SRVGRP パラメータの値に使用されます。この値により、サーバはグループ内のサーバとして識別されます。SRVGRP は、SERVICES セクションで、グループ内の特定のサービス・インスタンスのオカレンスを識別する場合にも使用されます。GROUPS セクションのその他のパラメータは、このグループを特定のリソース管理インスタンスに関連付けます (社員データベースなど)。GROUPS セクション内にある各行の形式は次のとおりです。

GROUPNAME 必須パラメータ [オプション・パラメータ]

GROUPNAME は、グループの論理名 (string_value) を指定します。グループ名は、GROUPS セクションのグループ名と MACHINES セクションの LMID の中で一意でなければなりません。このグループ名には、アスタリスク (*)、カンマ (,)、またはコロン (:) を指定できません。名前は 30 文字以内で指定します。

Jolt サーバ・リスナ (JSL) を含むグループには、GROUPS エントリを指定する必要があります。次の手順に従って、GROUPS エントリを作成します。

1. アプリケーション側でグループ名が選択されます (例 :JSLGRP、JREPGRP)。
2. MACHINES セクションの LMID パラメータに指定されている値と同じ識別子を指定します。
3. *GROUPS セクションの GRPNO に 1 ～ 30,000 の範囲の値を指定します。

注記　UBBCONFIG ファイルの GROUPS セクションに指定されているすべてのグループに対して、デフォルトでリソース・マネージャが割り当てられないようにしてください。デフォルト値として指定されたリ

ソース・マネージャは、JSL に割り当てられ、tmboot の実行時にエラーが発生します。SERVERS セクションの RESTART、MAXGEN、その他に指定されているデフォルト値は、JSL に対して有効です。

SERVERS セクション

このセクションでは、システムで起動されるサーバの初期状態に関する情報を定義します。常時実行中の状態にあり、受信したサーバ・グループのサービス要求を処理するのがサーバである、という捉え方は、特定のリモート環境には当てはまらない場合があります。ほとんどの環境では、オペレーティング・システムまたはリモート・ゲートウェイは単にサービスの送信を行っています。このような場合は、リモート・プログラムのエントリ・ポイントに対して、SERVER テーブルのエントリではなく SERVICE エントリ・ポイントを指定するだけで十分です。BEA Tuxedo システムのゲートウェイ・サーバは、リモート・ドメインのサービス要求を宣言し、キューに入れます。ホスト固有のリファレンス・ページでは、UBBCONFIG のサーバ・テーブル・エントリが特定の環境に適応しているかどうかを示し、適応している場合は対応するセマンティクスを明記する必要があります。SERVERS セクション内にある各行の形式は次のとおりです。

AOUT 必須パラメータ [オプション・パラメータ]

AOUT は、tmboot(1) によって実行されるファイル (string_value) を指定します。tmboot は、サーバ・グループで指定されたマシン上で AOUT を実行します。tmboot はターゲット・マシンで AOUT ファイルを検索するため、AOUT はそのマシンのファイルシステム内になければなりません (AOUT のパスには、ほかのマシン上にあるファイルシステムへの RFS 接続を含めることができます)。サーバの相対パス名が指定されている場合、AOUT の検索は、APPDIR、TUXDIR/bin、/bin、*path*、の順で行われます。<path> は、マシンの環境設定ファイル (ある場合) の最後の行 (PATH=) に指定されている値です。APPDIR および TUXDIR の値は、TUXCONFIG ファイル内の適切なマシン・エントリから取得されます。

クライアントは、Jolt サーバ・リスナ (JSL) を介して BEA Jolt アプリケーションに接続します。サービスは、Jolt サーバ・ハンドラ (JSH) を介してアクセスされます。JSL は複数のクライアントを扱うことができます。これらのクライアントは 唯一の通信ポイントである JSL を経由して特定のネットワーク・アドレス (JSL コマンド行で指定) のアプリケーションに接続しま

す。JSL は、ハンドラ・プロセスをスケジューリングします。ハンドラ・プロセスは、アプリケーションの管理ドメインの範囲内にあるリモート・ワークステーションで、クライアントの代わりに動作します。ハンドラは、1 つのポートで同時に複数のクライアントを扱うために、多重化スキーマを使用します。

JSL に指定されたネットワーク・アドレスは、JSL および JSL に関連付けられた JSH プロセスの TCP/IP アドレスを決定します。ネットワーク・アドレスにより決定されたポート番号は、JSL が新しい接続を受け付けるポート番号を指定します。JSL に関連付けられた各 JSH は、同じ TCP/IP アドレスで連続するポート番号を使用します。たとえば、JSL の最初のポート番号が 8000 であり、最大 3 つの JSH プロセスがある場合、これらの JSH プロセスは 8001、8002、8003 のポートを使用します。

注記　後続の JSL を誤って設定すると、ポート番号の衝突が発生します。

JSL で使用できるパラメータ

これまでに説明したセクションのパラメータのほか、JSL では次のパラメータを指定することができます。ただし、これらのパラメータを設定すると、アプリケーションにどのような影響が及ぶかを考慮してください。

`SVRGRP=`*`string_value`*

このパラメータは、実行するサーバが含まれるサーバ・グループの名前を指定します。*`string_value`* は、`*GROUPS` セクションのサーバ・グループを示す論理名でなければならず、名前は 30 文字以内で指定します。`*GROUPS` セクションのエントリと関連付けるということは、`LMID` が指定されたサーバ・グループ内のマシンで `AOUT` が実行されることを意味します。また、この関連付けにより、サーバ・グループの `GRPNO` と、関連するリソース・マネージャがオープンされている場合に受け付けられるパラメータが指定されます。すべてのサーバ・エントリには、サーバ・グループのパラメータが指定されていなければなりません。

`SRVID=`*`number`*

このパラメータには、グループ内の特定のサーバを示す識別子 (1 ～ 30,000 の範囲の値) を指定します。このパラメータは、すべてのサーバ・エントリに必要です (サーバ・グループ内のサーバが 1 つの場合も必要) 。複数の

サーバのオカレンスを設定する場合は、連続する SRVID を指定しないでください。MAX で指定された数までのサーバ用に、SRVID を残しておいてください。

オプション・パラメータ

SERVERS セクションのオプション・パラメータには、ブート・パラメータとランタイム・パラメータがあります。

ブート・パラメータ

ブート・パラメータは、tmboot によってサーバが実行されるときに使用されるパラメータです。いったん実行されると、サーバはコンフィギュレーション・ファイルからエントリを読み込み、ランタイム・オプションを決定します。正しいエントリが検索されるようにするため、一意なサーバ識別子が使用されます。次のパラメータがブート・パラメータです。

CLOPT=*string_value*

CLOPT パラメータは、起動時に AOUT に渡すコマンド行オプションの文字列を指定します。『BEA Tuxedo のファイル形式とデータ記述方法』の servopts(5) ページには、有効なパラメータが一覧表示されています。

開発中のサーバに適用されるオプションもあります。たとえば、-r オプションは、サービス要求が開始または終了するたびに、標準のエラー・ファイルにレコードを書き込むようサーバに指示します。

別のコマンド行オプションを使用して、サーバの標準出力 (stdout) や標準エラー (stderr) を特定のファイルに書き込んだり、サーバの起動時に利用可能なサービスの種類を最初に宣言するように指定できます。

CLOPT パラメータのデフォルト値は -A であり、サーバの起動時に、利用可能なサービスがすべて宣言されることを示します。

CLOPT パラメータには最大 256 文字まで指定できます。パラメータは二重引用符で囲む必要があります。

SEQUENCE=*number*

このパラメータは、ほかのサーバとの関係において、いつサーバをシャットダウンまたは起動するかを指定します。SEQUENCE が指定されていない場合、サーバは SERVERS セクションで指定された順序で起動し、逆の順序でシャットダウンされます。シーケンス番号が指定されているサーバとそうでないサーバがある場合、シーケンス番号が指定されたサーバが低い番号から順に起動します。次に、シーケンス番号が指定されていないサーバが、コンフィギュレーション・ファイルに表示されている順序で起動します。シーケンス番号には 1 ～ 9999 の値を指定します。複数のサーバに同じシーケンス番号が割り当てられると、tmboot の実行時にこれらのサーバが同時に起動します。

MIN=*number*

MIN パラメータは、tmboot で起動されるサーバのオカレンスの最小数を指定します。RQADDR が指定されており、MIN が 1 より大きい数の場合、サーバは MSSQ (複数サーバ、単一キュー) になります。サーバの識別子は SRVID で指定します。SRVID の最大値は SRVID + (MAX -1) です。サーバのすべてのオカレンスには、同じサーバ・パラメータのほか、同じシーケンス番号が付きます。MIN には 0 ～ 1000 までの範囲の値を指定できます。MIN を指定しないと、デフォルトで 1 が設定されます。

MAX=*number*

MAX パラメータは、起動するサーバのオカレンスの最大数を指定します。tmboot が実行されると、MIN で指定した数のサーバが起動します。次に、tmboot の -i オプションを使用して関連するサーバ識別子を指定し、その他のサーバ (MAX で指定した数まで) を起動します。MAX には 0 ～ 1000 の範囲の値を指定できます。MAX を指定しないと、デフォルトで MIN と同じ値かまたは 1 が設定されます。

- tmboot を実行すると、MIN で指定した数のサーバが起動します。これ以外のサーバを起動するには、tmboot の -i SRVID オプションを使用して明示的に呼び出します。
- RQADDR が指定されており、MIN が 1 より大きい数の場合、MSSQ セットが作成されます。
- MIN を指定しない場合は、デフォルトで 1 が設定されます。
- MAX を指定しない場合は、デフォルトで MIN と同じ値が設定されます。

- 必要に応じて自動的に生成される会話型サーバでは、MAX を指定しておくことが特に重要です。

ランタイム・パラメータ

tmboot によって起動したサーバは、ランタイム・パラメータを使用します。既に説明したとおり、tmboot は、サーバの起動時に MACHINES セクションに対して TUXDIR、APPDIR、および ENVFILE の各パラメータの値を使用します。さらに、サーバの PATH を次のパスに設定します。

"APPDIR:TUXDIR/bin:/bin:*path*"

path は、ENVFILE ファイルの最後の行 (PATH=) に指定されている値です。次のパラメータはランタイム・パラメータです。

ENVFILE=*string_value*

ENVFILE パラメータを使用して、サーバの初期化時に、tmboot によって作成された環境に対して値を追加することができます。tmboot の MACHINES ENVFILE で変数を指定した後、オプションとして、SERVERS ENVFILE パラメータで指定されたファイルの変数を設定することもできます。これらのファイルを使用して TUXDIR、APDIR、TUXCONFIG、または TUSOFFSET を上書きすることはできません。最も良い方法は、アプリケーションを正しく実行するために必要な変数だけをサーバの ENVFILE に設定しておくことです。

サーバ側では、サーバの起動後に ENVFILE ファイルが処理されます。したがって、サーバの実行に必要な実行可能ファイルまたは動的にロードされたファイルを検索するためのパス名を、このファイルに設定することはできません。これらのタスクを実行する必要がある場合は、代わりにマシンの ENVFILE を使用してください。

ENVFILE では、各行を次の形式で指定する必要があります。

VARIABLE =*string*

VARIABLE の先頭には、下線 (_) または英文字を指定する必要があります。また、下線と英数字だけで構成することもできます。サーバ・グループに属するサーバが別のマシンに移行される場合、ENVFILE はどちらのマシンでも同じ場所になければなりません。

CONV={Y | N}

CONV は、サーバが会話型サーバであるかどうかを指定します。会話型サーバが定義されている場合、CONV の値は Y になります。接続は会話型サーバに対してのみ行うことができます。tpacall(3c) または tpcall(3c) を使った rpc 要求は非会話型サーバに対してのみ行うことができます。受け取った要求に対して応答を行うサーバの場合は、CONV=N (デフォルト) を設定するか、またはパラメータを省略します。

RQADDR=*string_value*

RQADDR は、このサーバの要求キューにシンボリック名を割り当てます。複数のサーバに対して同じシンボリック名を使用し、MSSQ セットを作成します (1 より大きい MIN の値を指定)。MSSQ セットに属するすべてのメンバは、同じサービスのセットを提供し、同じサーバ・グループに属していなければなりません。

RQADDR を指定しないと、このサーバのキュー・アドレスとなる一意なキーが割り当てられます。ただし、キューにシンボリック名が設定されている場合は、キュー・アドレスを引数として使用する tmadmin コマンドを使う方が簡単です。

RQPERM=*number*

このサーバの要求キューに UNIX 形式でパーミッションを割り当てる場合は、RQPERM パラメータを使用してください。*number* には、0001 ～ 0777 の範囲の値を指定します。パラメータが何も指定されていない場合は、掲示板に設定されたパーミッションの値 (RESOURCES セクションの PERM で指定) が使用されます。そこでもパーミッションが設定されていない場合は、デフォルトの 0666 が指定されます。ただし、この値が設定された状態では、システムにログインしたユーザであれば誰でもアプリケーションを使用できるため、注意が必要です。

REPLYQ={ Y | N }

REPLYQ パラメータは、応答キュー (要求キューとは別) を AOUT に対して作成する必要があるかどうかを指定します。N が指定されると、AOUT と同じ LMID に応答キューが作成されます。要求キューを使用するサーバが 1 つの場合、要求キューから応答を取り出す操作は問題なく行われます。しかし、サーバが MSSQ セットのメンバであり、応答メッセージを受信するようにプログラミングされているサービスを含んでいる場合、REPLYQ を Y に設定して、このサーバに対して応答キューが個別に作成されるようにする必要があ

ります。N に設定されると、応答は MSSQ セット内の全サーバが共有する要求キューに送信されてしまい、応答が要求元のサーバに返されるかどうかは保証されません。

応答を必ず受信するには、常に MSSQ セット内の全サーバに REPLYQ=Y を設定するべきです。MSSQ セット内のサーバでは、同じサービスが提供されなければなりません。つまり、セット内のあるサーバが応答を待機している場合は、セット内のほかのサーバでも応答を待機できます。

RPPERM=*number*

応答キューにパーミッションを割り当てるには、RPPERM パラメータを使用してください。*number* は、通常の UNIX 形式 (例 : 0600) で指定されます。指定できる値は、0001 ～ 0777 までの数値です。RPPERM を指定しない場合は、デフォルトで 0666 が指定されます。このパラメータは、REPLYQ=Y の場合のみ有効です。要求と応答が同じキューから読み出される場合、必要なのは RQPERM のみで、RPPERM は無視されます。

RESTART={ Y | N }

RESTART パラメータは、AOUT を再起動できるかどうかを示し、Y または N を指定できます。デフォルトは N です。移行可能なサーバ・グループにサーバが属している場合、RESTART には Y を指定しなければなりません。SIGTERM シグナルを指定して起動したサーバは再起動できません。このサーバはリブートする必要があります。

サーバの再起動に関する方針は、サーバの状態、つまりサーバが開発中であるかどうかによって異なります。アプリケーションがテスト過程の段階では、サーバに繰り返し障害が発生することも考えられます。しかし、アプリケーションがプロダクション段階に進んだら、サーバは、ほとんど障害が発生しない状態でなければなりません。アプリケーションがプロダクション段階に進んだら、サーバの再起動に関してさらに厳しい条件のパラメータを設定することもできます。

RESTART に関連するパラメータ

RCMD=*string_value*

AOUT が再起動可能な場合、このパラメータは、AOUT が異常終了した場合に実行するコマンドを指定します。最初の空白またはタブまでの文字列には、実行可能な UNIX ファイルの名前 (絶対パス名または APPDIR への相対パス

名) を指定します。コマンドの先頭に Shell の変数を設定しないでください。オプションで、コマンド名の後にコマンド行引数を指定することもできます。コマンド行には、サーバの再起動条件を指定する GRPNO と SRVID の 2 つの引数を追加できます。*string_value* は、サーバの再起動と並行して実行されます。

RCMD パラメータを使用すると、サーバの再起動時に並行して実行されるコマンドを指定することができます。このコマンドは、サーバの PATH のディレクトリにある実行可能な UNIX のシステム・ファイルでなければなりません。たとえば、カスタマイズされたメッセージを userlog に送信し、再起動するサーバにマークを付けるコマンドです。

MAXGEN=*number*

このパラメータは、AOUT が再起動可能な場合、GRACE で指定された期間内に、最大「number - 1」回まで再起動されることを指定します。指定できる値は 0 より大きく、256 より小さい数値です。この値を指定しないと、デフォルトの 1 (サーバは一度起動できるが、再起動はできない) が設定されます。サーバが再起動可能な場合、MAXGEN には 2 以上の値を指定する必要があります。RESTART に Y を指定しないと、MAXGEN の値は無視されます。

GRACE=*number*

RESTART が Y の場合、GRACE パラメータを使用して、サーバの再起動を行える期間 (秒単位) を指定することができます。再起動は MAXGEN - 1 回行うことができます。秒数として 0 以上 2,147,483,648 未満 (または 68 年強) の値を指定します。GRACE を指定しない場合は、デフォルトの 86,400 秒 (24 時間) が指定されます。GRACE を 0 に設定すると、すべての制限が解除されます。つまり、サーバの再起動回数が制限されなくなります。

パラメータを入力する

JSL には、BEA Tuxedo のパラメータである RESTART、RQADDR、および REPLYQ を使用できます。ランタイム・パラメータの詳細については、『BEA Tuxedo アプリケーション実行時の管理』を参照してください。パラメータは次の要領で指定してください。

1. SRVGRP パラメータを指定するには、GROUPS セクションにある、定義済みのサーバ・グループ名を示す値を入力します。

2. SRVID を指定するには、1 ～ 30,000 の数値を使用してグループ内のサーバを示す識別子を入力します。
3. CLOPT パラメータが以下の構文になっていることを確認します。

   ```
   CLOPT= “-- -n 0x0002PPPPNNNNNNNN -d /dev/tcp -m2 -M4 -x10”
   ```

   注記　CLOPT パラメータは数種類あります。該当するコマンド行の説明については、3-17 ページの「JSL のコマンド行オプション」の表を参照してください。
4. 以下の説明に従い、必要に応じてオプション・パラメータを入力します。
   - サーバを起動する順序を決定するには、SEQUENCE パラメータを入力します。
   - サーバの再起動を許可するには、RESTART パラメータに Y を指定します。
   - サーバを再起動する回数を制限しない場合は、GRACE パラメータに 0 を指定します。

BEA Jolt オンライン・マニュアルのサンプル・アプリケーション

以下の BEA Jolt 製品の Web ページには、サンプル・コードが用意されています。このサンプル・コードは BEA Jolt を使用して変更することができます。

`http://www.bea.com/products/jolt/index.htm`

これらのサンプルを使用すると、BEA Jolt の機能を実際に試すことができます。

以下の Java 関連の Web サイトも参照してください。

- Javasoft ホームページ (`http://www.java.sun.com/`)
- comp.lang.java 階層のニュースグループ。この階層内のニュースグループには、Java に関する過去の記事やメールなど、貴重な資料が保存されています。

4 Jolt リポジトリ・エディタを使う

Jolt リポジトリ・エディタを使用すると、BEA Tuxedo のコンフィギュレーション・ファイルから利用できる情報に基づき、リポジトリ内の BEA Tuxedo サービスの定義を追加、変更、テスト、エクスポート、および削除することができます。Jolt リポジトリ・エディタでは、パッケージ名、サービス名、パラメータ名などの BEA Tuxedo サービスの定義を扱うことができます。

ここでは、次の内容について説明します。

- リポジトリ・エディタの紹介
- はじめに
- リポジトリ・エディタの主なコンポーネント
- パラメータを表示する手順
- パッケージ・オーガナイザを使ってサービスをグループ化する
- パッケージ、サービス、パラメータを変更する
- Jolt クライアントからサービスを利用可能にする
- サービスをテストする
- リポジトリ・エディタのトラブルシューティング

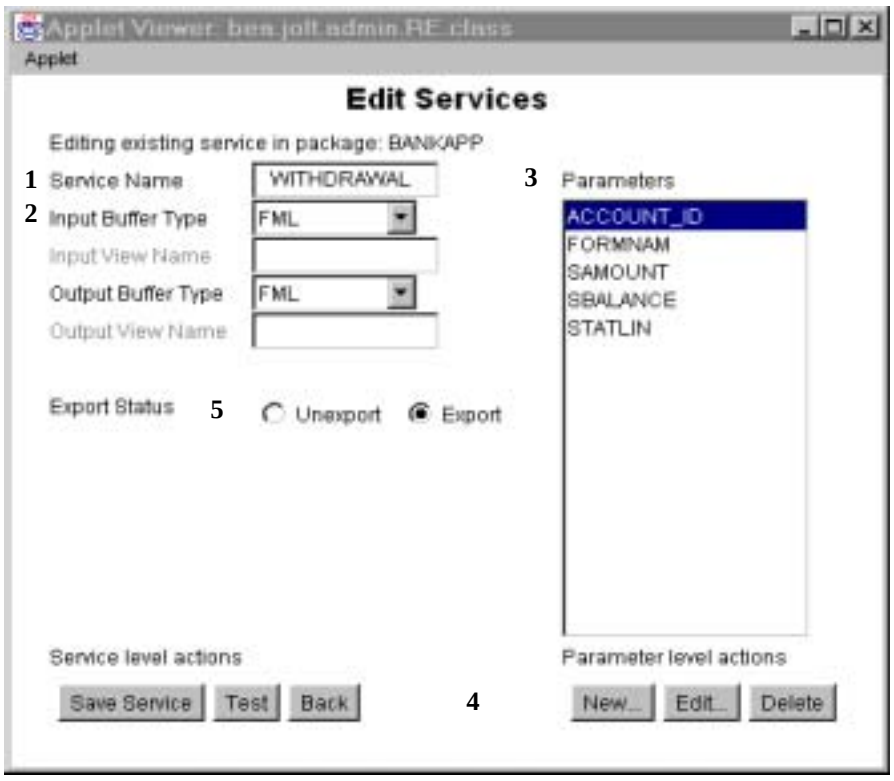
リポジトリ・エディタの紹介

Jolt リポジトリは、Java パラメータを BEA Tuxedo の型付きバッファに変換するために Jolt 内部で使用されます。リポジトリ・エディタはダウンロード可能な Java アプレットとして利用できます。BEA Tuxedo サービスをリポジトリに追加する場合は、Jolt サーバにエクスポートして、Jolt クライアントからクライアント・リクエストを発行できるようにする必要があります。

[Repository Editor] ウィンドウ

[Repository Editor] ウィンドウには、入力フィールド、選択ポップアップ・メニュー、コマンド・ボタン、ステータス、およびラジオ・ボタンがあります。4-3 ページの「[Repository Editor] ウィンドウの例」は、[Repository Editor] ウィンドウの各構成要素を示しています。4-4 ページの「[Repository Editor] ウィンドウの構成要素」では、各構成要素の詳細について説明します。

図 4-1[Repository Editor] ウィンドウの例



[Repository Editor] ウィンドウの説明

次の表は、前の図に示した [Repository Editor] ウィンドウの各構成要素の説明です。

表 4-1[Repository Editor] ウィンドウの構成要素

構成要素	関数
1 テキスト・ボックス	サービス名、入力 VIEW 名、サーバ名、ポート番号などの文字、番号、および英数字を入力します。前の図では、[Service Name] ボックスがこれに当たります。
2 選択ポップアップ・メニュー	画面に表示されていない項目の一覧は矢印ボタンを使用して表示します。前の図では、[Input Buffer Type] または [Output Buffer Type] がこれに当たります。
3 リストボックス	定義済みの項目の一覧から必要な項目を選択したり ([Parameters] リストボックス)、ユーザ定義による項目の一覧から選択します。
4 コマンド・ボタン	[Packages] ウィンドウ、[Services] ウィンドウ、[Package Organizer] の表示などの操作を実行します。前の図では、[Save Service] ボタン、[Test] ボタン、[Back] ボタン、[New] ボタン、[Edit] ボタン、[Delete] ボタンなどがこれに当ります。
5 ラジオ・ボタン	複数のオプションのうち、いずれか 1 つを選択します。一度に選択できるラジオ・ボタンは 1 つだけです。たとえば、Export Status では、[Unexport] または [Export] のどちらか一方しか選択できません。

はじめに

リポジトリ・エディタを起動する前に、最低限必要なコンポーネントがインストールされていることを確認してください。Jolt サーバと Jolt クライアントは必須です。

リポジトリ・エディタを使用するには、次の手順に従います。

1. リポジトリ・エディタを起動します。

   リポジトリ・エディタは、JavaSoft `appletviewer` または Web ブラウザから起動します。これらの手順は、次の節でさらに詳しく説明されます。

2. リポジトリ・エディタにログオンします。

注記　情報の入力を完了した後でリポジトリ・エディタを終了する方法については、4-10 ページの「リポジトリ・エディタを終了する」を参照してください。

Java Applet Viewer を使ってリポジトリ・エディタを起動する

1. `CLASSPATH` に Jolt クラスのディレクトリを指定します。
2. アプレットをローカル・ディスクからロードする場合は、URL に次のように入力してください。

   ```
   appletviewer <full-pathname>/RE.html
   ```

   Web サーバからアプレットをロードする場合は、URL に次のように入力してください。

   ```
   appletviewer http://<www.server>/<URL path>/RE.html
   ```

3. Enter キーを押します。

   4-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」に示すようなウィンドウが表示されます。

Web ブラウザを使ってリポジトリ・エディタを起動する

次のいずれかの方法を使って、Web ブラウザからリポジトリ・エディタを起動してください。

ローカル・ディスク上のリポジトリ・エディタを起動する場合

1. CLASSPATH に Jolt クラスのディレクトリを指定します。
2. 次を入力します。

   ```
   file:<full-pathname>/RE.html
   ```

3. Enter キーを押します。

   4-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」に示すようなリポジトリ・エディタが表示されます。

Web サーバからリポジトリ・エディタを起動する場合

1. CLASSPATH に Jolt クラスのディレクトリが含まれていないことを確認します。
2. CLASSPATH の設定を解除します。
3. 次を入力します。

   ```
   http://<www.server>/<URL path>/RE.html
   ```

   注記　ファイルを開く前に、RE.html の applet codebase パラメータを変更して Jolt の Java クラス・ディレクトリに一致させてください。

4. Enter キーを押します。

   4-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」に示すようなリポジトリ・エディタが表示されます。

リポジトリ・エディタにログオンする

注記　JDK 1.3 appletviewer を使用して Jolt リポジトリ・エディタを起動する場合、リモート・マシンに接続することはできません。ローカル・ホストの JSL にのみ接続できます。これは JDK 1.3 appletviewer におけるセキュリティの制限によるものです。また、JDK 1.2 では、リモート・マシンの JSL に接続するには appletviewer の `-nosecurity` オプションを使用しなければなりません。

1. リポジトリ・エディタを起動するのに必要な手順を実行します。

   手順 2 に進む前に 4-8 ページの「BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ」に示したようなログオン画面が表示されます。この画面が表示されたことを確認してから次の手順に進んでください。

2. BEA Tuxedo アプリケーションへの「アクセス・ポイント」として指定されているサーバ・マシン名を入力し、Tab キーを押します。

3. ポート番号を入力して Enter キーを押します。

   システムにより、サーバとポートの情報が検証されます。

   注記　Jolt リレー経由でログオンしない限り、ここで指定したポート番号と同じ番号が Jolt リスナの設定時に使用されます。詳細については、`UBBCONFIG` ファイルを参照してください。

4. BEA Tuxedo のアプリケーション・パスワードを入力して Enter キーを押します。

   認証レベルに応じて、手順 5 と 手順 6 を実行します。

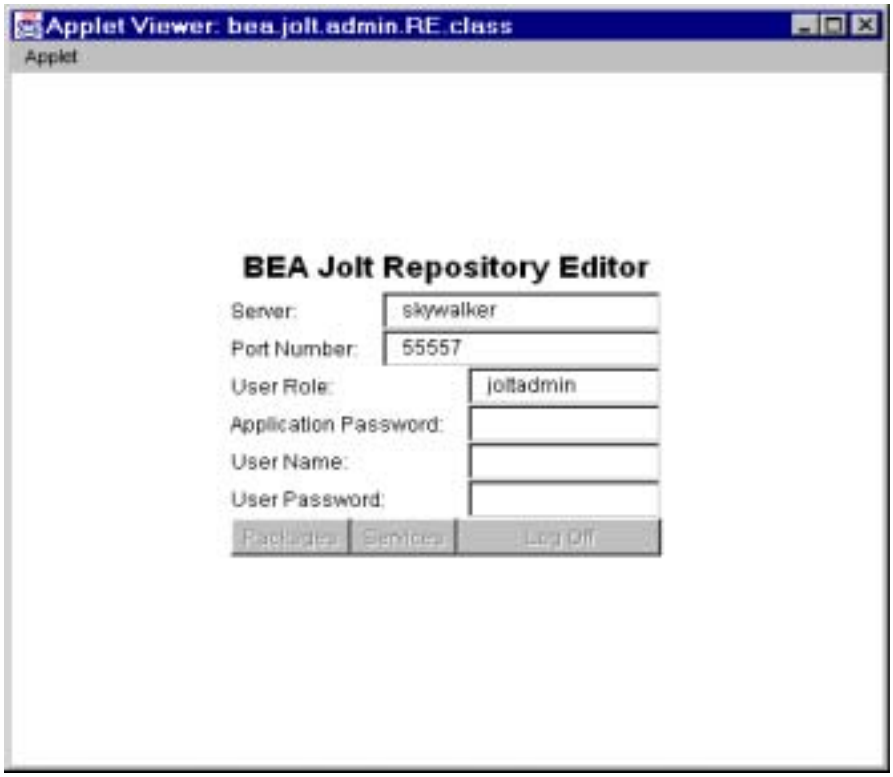
5. BEA Tuxedo ユーザ名を入力して Tab キーを押します。

6. BEA Tuxedo ユーザ・パスワードを入力して Enter キーを押します。

   [Packages] ボタンと [Services] ボタンが有効になります。

   注記　詳細については、「`JoltSessionClass`」を参照してください。

図 4-2BEA Jolt リポジトリ・エディタの [Logon] ウィンドウ



次の「リポジトリ・エディタの [Logon] ウィンドウの説明」では、リポジトリ・エディタの [Logon] ウィンドウの構成要素について説明します。

リポジトリ・エディタの [Logon] ウィンドウの説明

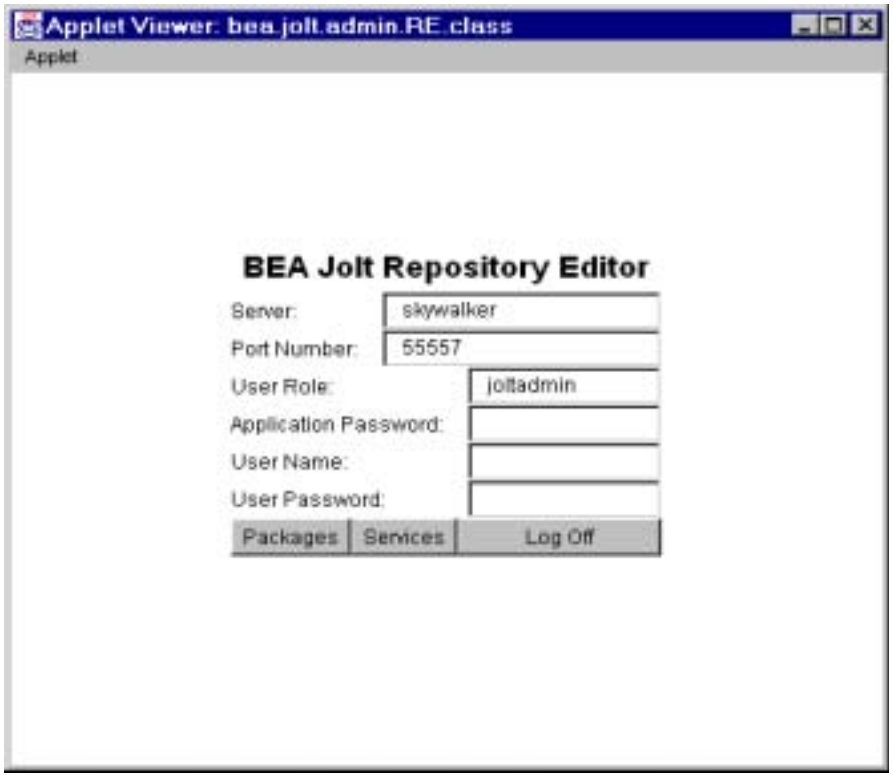
表 4-2 リポジトリ・エディタの [Logon] ウィンドウの説明

オプション	説明
Server	サーバ名を入力します。
Port Number	10 進値でポート番号を入力します。 注記　サーバ名とポート番号が入力されると、[User Name] フィールドと [User Password] フィールドがアクティブになります。これらのフィールドがアクティブになるかどうかは BEA Tuxedo アプリケーションの認証レベルに基づきます。
User Role	BEA Tuxedo ユーザ・ロールを入力します。BEA Tuxedo の認証レベルが USER_AUTH 以上の場合にのみ入力します。
Application Password	BEA Tuxedo の管理パスワードを入力します。
User Name	BEA Tuxedo ユーザを識別するための名前を入力します。名前の先頭には英字を指定します。
User Password	BEA Tuxedo のパスワードを入力します。
Packages	[Packages] ウィンドウにアクセスします (ログオンすると利用可能になります)。
Services	[Services] ウィンドウにアクセスします (ログオンすると利用可能になります)。
Log Off	サーバとの接続を切断します。

リポジトリ・エディタを終了する

パッケージ、サービス、およびパラメータの追加、編集、テスト、削除が完了したらリポジトリ・エディタを終了します。終了する前に、4-10 ページの「BEA Jolt リポジトリ・エディタを終了する前の [Logon] ウィンドウ」に示すようなウィンドウが表示されます。

図 4-3BEA Jolt リポジトリ・エディタを終了する前の [Logon] ウィンドウ



使用できるのは、[Packages]、[Services]、および [Log Off] のみです。文字入力フィールドは利用できません。

リポジトリ・エディタを終了するには、次の手順に従います。

1. [Back] をクリックして、リポジトリ・エディタの [Logon] ウィンドウに戻ります。
2. [Log Off] をクリックしてサーバとの接続を切断します。

   リポジトリ・エディタの [Logon] ウィンドウは表示されたままですが、いくつかのフィールドは入力できなくなります。
3. ブラウザのメニューから [Close] を選択して画面上のウィンドウを終了します。

リポジトリ・エディタの主なコンポーネント

リポジトリ・エディタを使って、以下のコンポーネントの追加、変更、または削除を行うことができます。

- パッケージ
- サービス

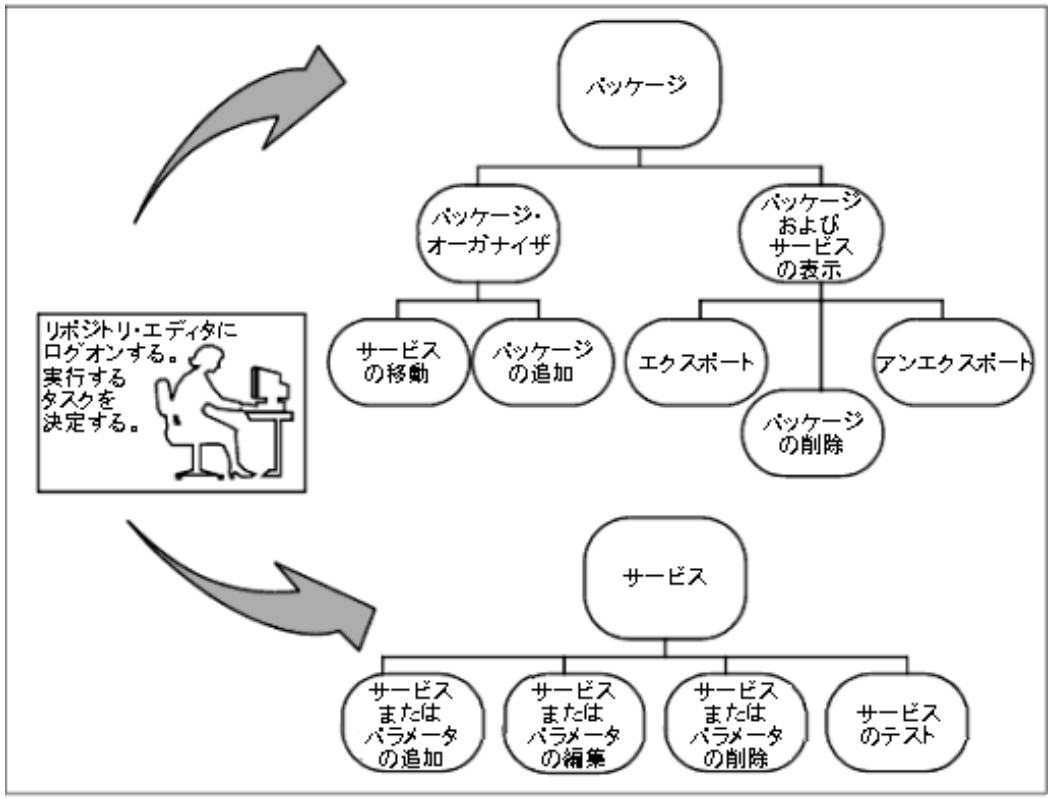
  サービスのテストやグループ化を行うこともできます。
- パラメータ

リポジトリ・エディタの操作手順

リポジトリ・エディタにログオンすると、[Packages] ボタンと [Services] ボタンの 2 つのボタンが使用可能になります。

次に、リポジトリ・エディタの流れ図を示します。2 つのボタンのどちらを選択するかを決める際に参考にしてください。

図 4-4 リポジトリ・エディタの流れ図



[Packages] ウィンドウを開き、以下の機能を実行する場合は、[Packages] を選択します。

- パッケージとサービスを表示する
  - [Export] や [Unexport] を使ってサービスを利用可能にする
  - 削除するパッケージを選択する
- パッケージ・オーガナイザにアクセスして次の操作を行う
  - パッケージ間でサービスを移動する
  - 新規パッケージを作成する

詳細については、4-14 ページの「パッケージとは」を参照してください。

[Services] ウィンドウを開き、以下の機能を実行する場合は、[Services] を選択します。

- サービス定義の作成、追加、編集、または削除を行う
- パラメータの作成、追加、編集、または削除を行う
- サービスとパラメータのテストを行う

詳細については、4-17 ページの「サービスとは」を参照してください。

パッケージとは

Jolt を管理する際には、パッケージを使ってサービスをグループ化すると便利です。サービスは、複数のパラメータ (個人識別番号、口座番号、支払、レート、期間、年齢、社会保障番号など) から構成されます。

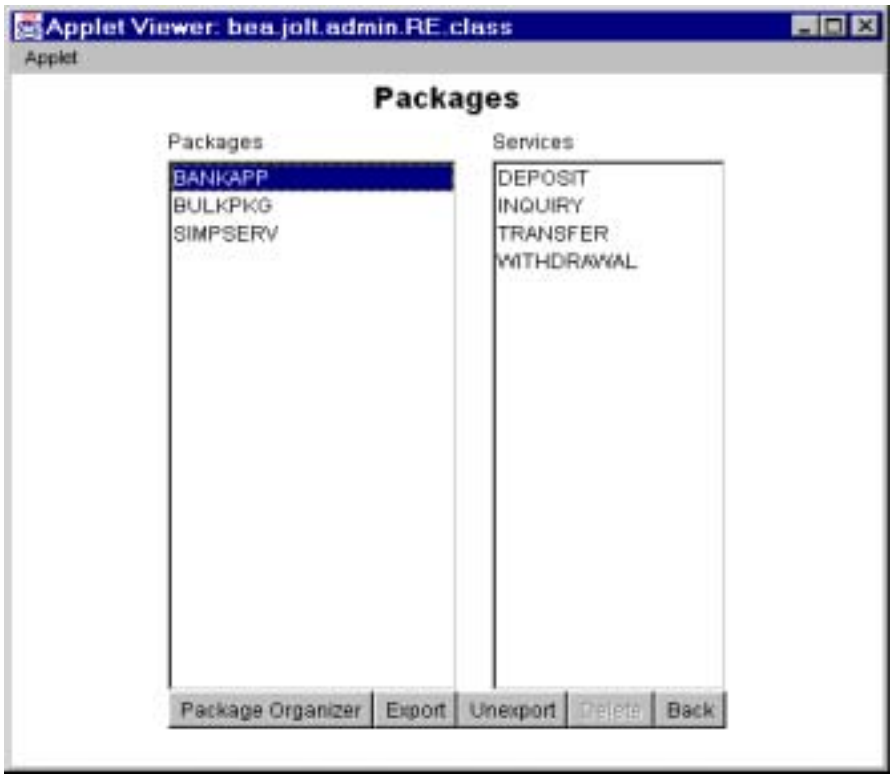
以下の機能を実行する場合は、[Packages] を選択します。

- パッケージとサービスを表示する
- サービスのエクスポートまたはアンエクスポートを行う
- パッケージを削除する
- パッケージ・オーガナイザにアクセスして次の操作を行う
  - サービスを移動する
  - 新規パッケージを作成する

利用可能なパッケージを表示するには、リポジトリ・エディタの [Logon] ウィンドウの [Packages] ボタンをクリックします。表示一覧からパッケージを選択すると、パッケージ内に含まれるサービスが表示されます。

次の図は、[Packages] ウィンドウの例です。BANKAPP パッケージが選択されると、BANKAPP パッケージ内に含まれるサービスが表示されます。

図 4-5[Packages] ウィンドウの例



[Packages] ウィンドウの説明

表 4-3[Packages] ウィンドウの説明

オプション	説明
Packages	利用可能なパッケージを一覧表示します。
Services	選択されたパッケージに含まれるサービスのうち、利用可能なサービスを一覧表示します。

表 4-3[Packages] ウィンドウの説明 (続き)

Package Organizer	[Package Organizer] ウィンドウにアクセスして、利用可能なパッケージとサービスを検討します。このウィンドウを使って、パッケージ間でサービスを移動したり、新しいパッケージを追加します。
オプション	**説明**
Export	クライアントが最新のサービスを利用できるようにします。このオプションはパッケージが選択されると利用可能になります。
Unexport	既存のサービスをテストする前にこのオプションを選択してください。このオプションはパッケージが選択されると利用可能になります。
Delete	パッケージを削除します。このオプションは選択されたパッケージが空の場合 (パッケージ内にサービスが 1 つもない) に利用可能になります。
Back	前のウィンドウに戻ります。

パッケージを表示する手順

1. リポジトリ・エディタの [Logon] ウィンドウの [Packages] ボタンをクリックします。

   [Packages] ウィンドウが表示され、使用可能なパッケージの一覧が表示されます。

   4-15 ページの「[Packages] ウィンドウの例」では、`BANKAPP`、`BULKPKG`、および `SIMPSERV` が利用可能なパッケージです。

2. 詳細については、4-21 ページの「パラメータを表示する手順」を参照してください。

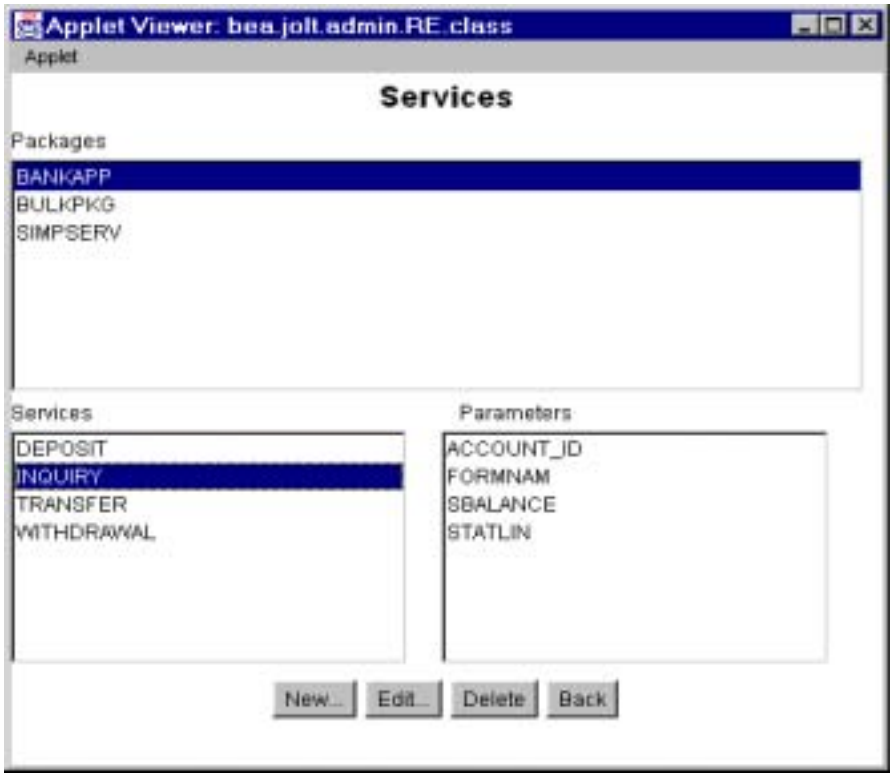
サービスとは

サービスは、利用可能な BEA Tuxedo サービスの定義です。サービスは、複数のパラメータ (個人識別番号、口座番号、支払、レートなど) から構成されます。Jolt サービスを追加したり編集しても、既存の BEA Tuxedo サービスには影響ありません。

[Services] ウィンドウを使用して、サービスの追加、編集、または削除を行います。

次に、BANKAPP パッケージを選択した場合の [Services] ウィンドウの例を示します。このパッケージで利用可能なサービスおよびパラメータが一覧表示されます (パラメータの詳細については後述します)。

図 4-6[Services] ウィンドウの例



[Services] ウィンドウの説明

表 4-4[Services] ウィンドウの説明

オプション	説明
Packages	利用可能なパッケージを一覧表示します。
Services	選択したパッケージ内のサービスを一覧表示して、編集や削除を行います。サービスを選択するとそのサービス内のパラメータが表示されます。
Parameters	選択されたサービスのパラメータを表示します。
New	[Edit Services] ウィンドウを表示して新しいサービスを追加します。
Edit	[Edit Services] ウィンドウを表示して既存のサービスを編集します。このボタンはサービスが選択された場合のみ利用可能になります。
Delete	サービスを削除します。このボタンはサービスが選択された場合のみ利用可能になります。
Back	前のウィンドウに戻ります。

サービスを表示する手順

1. リポジトリ・エディタの [Logon] ウィンドウで [Services] を選択します。

   [Services] ウィンドウが開かれ、利用可能なパッケージの一覧が表示されます。

2. パッケージを選択します。

   選択したパッケージで利用可能なサービスを一覧表示します。

   図 4-18 ページの「[Services] ウィンドウの例」では、`BANKAPP` が選択されたパッケージです。`DEPOSIT`、`INQUIRY`、`TRANSFER`、および `WITHDRAWAL` が `BANKAPP` で利用可能なサービスです。

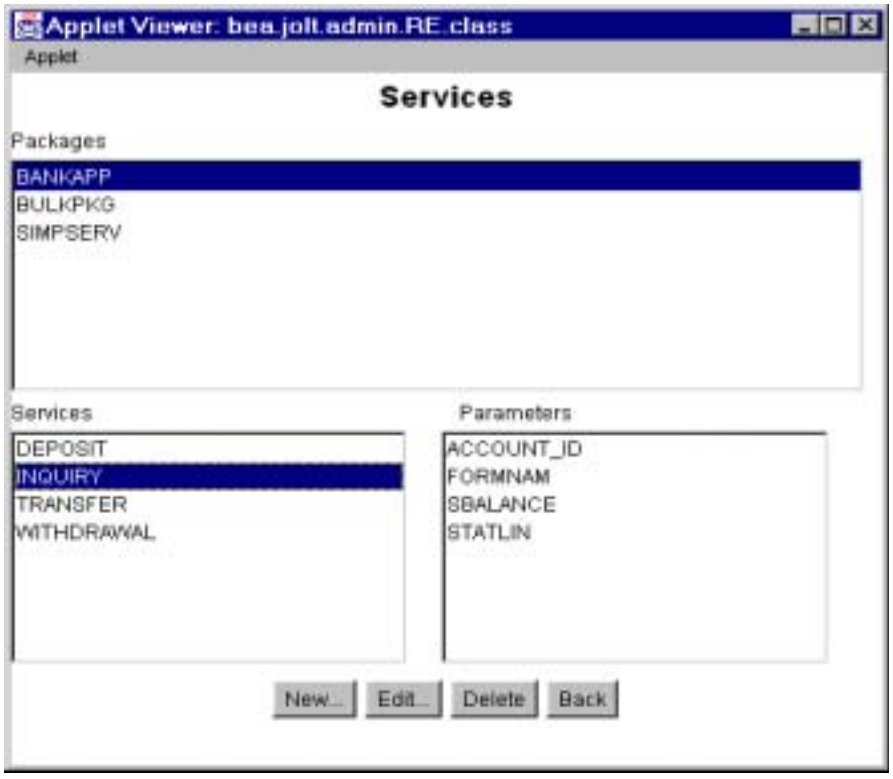
3. 詳細については、4-21 ページの「パラメータを表示する手順」を参照してください。

パラメータの機能

サービスは、複数のパラメータ (ピン番号、口座番号、支払、レート、期間、年齢、社会保障番号など) から構成されます。次の図は、サービスとそのパラメータを選択した [Services] ウィンドウを示しています。

注記　パラメータを追加または編集しても、既存の BEA Tuxedo サービスは変更されません。

図 4-7 パラメーター覧が表示された [Services] ウィンドウ



パラメータを表示する手順

1. リポジトリ・エディタの [Logon] ウィンドウで [Services] を選択します。

   [Services] ウィンドウが開かれ、利用可能なパッケージの一覧が表示されます。

2. パッケージを選択します。

   選択したパッケージで利用可能なサービスを一覧表示します。

   次の図では、BANKAPP が選択されたパッケージです。

3. サービスを選択します。

   選択したサービスで利用可能なパラメータを一覧表示します。

   前の図では、INQUIRY が選択されたサービスです。

4. 選択されたサービスのパラメータを [Parameters] リストボックスに表示します。

   前の図では、ACCOUNT_ID、FORMNAM、SBALANCE、および STATLIN が INQUIRY サービスで利用可能なパラメータです。

5. 詳細については、4-29 ページの「パラメータを追加する」を参照してください。

パッケージとサービスを設定する

この節では、パッケージとサービスの設定に必要な手順を説明します。

- 作業を保存する
- パッケージを追加する
- サービスを追加する
- パラメータを追加する

作業を保存する

サービスやパラメータを作成したり編集する場合は、入力した情報を誤って失うことがないよう定期的に情報を保存することが大切です。[Edit Services] ウィンドウの [Save Service] をクリックして情報を保存しておくと、システム・エラーが発生しても情報を再入力せずに済みます。

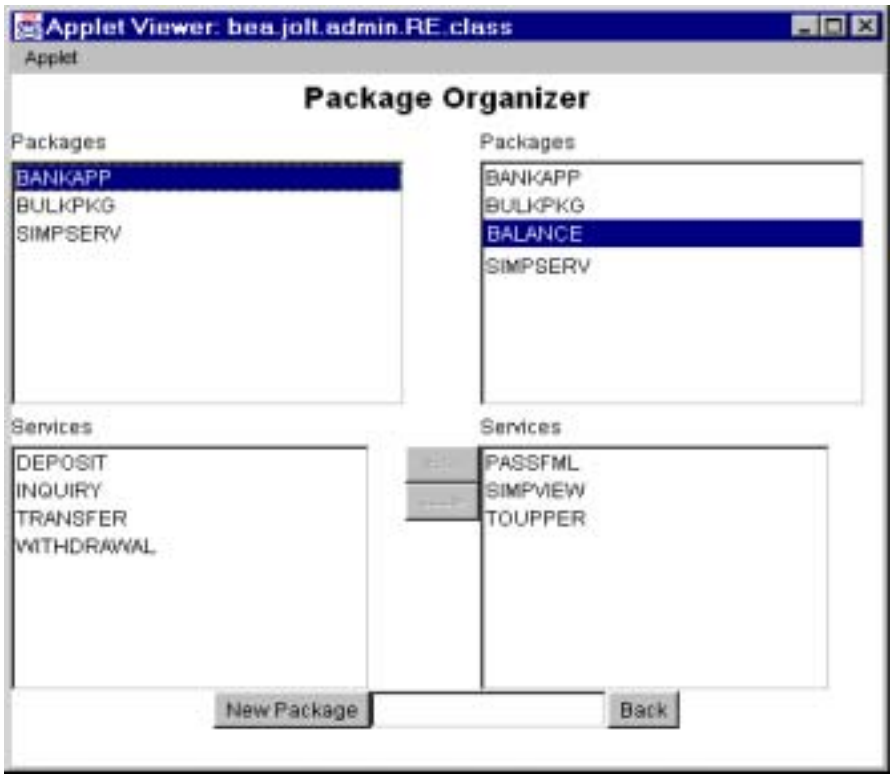
注記　サービスのパラメータを追加したり編集する場合は、[Add] を選択してから [Edit Parameters] ウィンドウの [Back] を選択し、[Edit Service] ウィンドウに戻ってください。

[Edit Service] ウィンドウで新規サービスを追加したり、既存のサービスを変更する場合は、[Back] を選択する前に [Save Service] が選択されたことを確認してください。変更された情報が保存される前に [Back] が選択されると、画面の下のステータス行に簡単な警告メッセージが表示されます。

パッケージを追加する

サービスの新規グループを追加する必要がある場合は、サービスを追加する前に新規パッケージを作成しておく必要があります。4-23 ページの「[Package Organizer] ウィンドウ」は、新規パッケージの BALANCE を [Packages] リストボックスに追加する手順を示しています。

図 4-8 [Package Organizer] ウィンドウ



パッケージを追加する手順

1. リポジトリ・エディタの [Logon] ウィンドウの [Packages] をクリックして、[Packages] ウィンドウを表示します。
2. [Package Organizer] を選択すると、4-23 ページの「[Package Organizer] ウィンドウ」に示すような [Package Organizer] ウィンドウが表示されます。

   このウィンドウの詳細については、この章の 4-36 ページの「[パッケージ・オーガナイザ] ウィンドウの説明」を参照してください。
3. [Package Organizer] ウィンドウで [New Package] をクリックします。

   テキスト・フィールドが入力可能になります。
4. 新規パッケージの名前 (32 文字以内) を入力し、Enter キーを押します。

   新しい名前 (前の図では BALANCE) が [Packages] リストボックス内に順不同で表示されます。

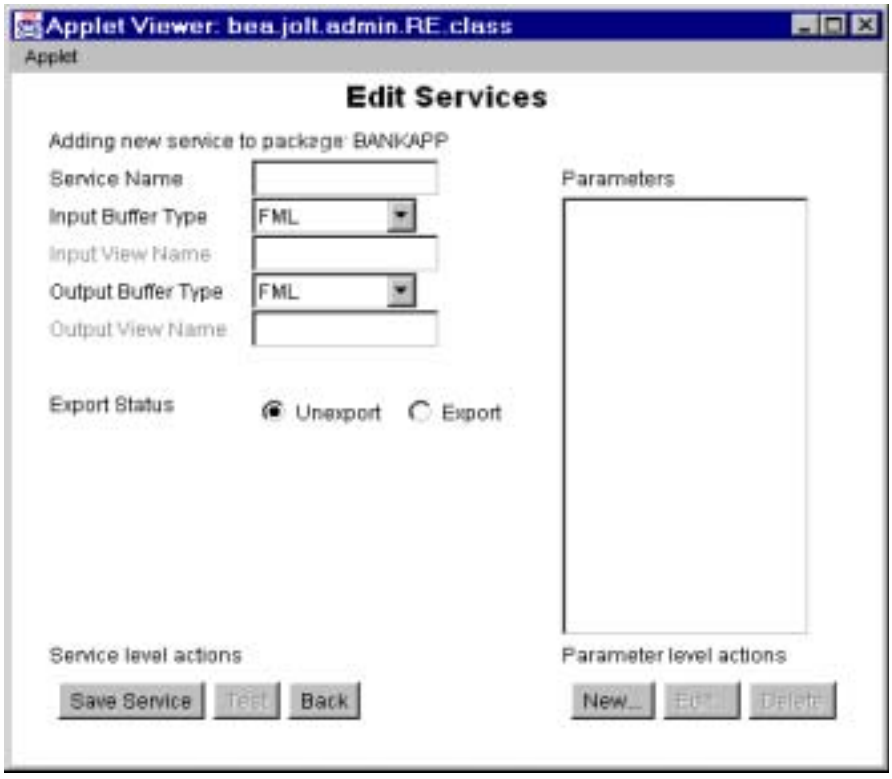
サービスを追加する

サービスは利用可能な BEA Tuxedo サービスの定義で、Jolt パッケージの一部に過ぎません。新規または既存のパッケージの一部としてサービスを作成する必要があります。

リポジトリ・エディタに入力された新しいサービス名は、入力されたとおりに登録されます。たとえば、すべて大文字で入力された名前、略語、スペルミスなども、そのまま登録されます。サービス名は 30 文字以内で指定します。

次に示すような [Edit New Services] ウィンドウを使って、新しいサービスを追加します。

図 4-9[Edit Services] ウィンドウ : 新規パッケージの追加



サービスを追加するウィンドウの説明

次の表は、[package] ウィンドウ内のパッケージにサービスを追加するオプションの説明です。

	オプション	説明
[Edit Services] ウィンドウのオプション	**Service Name**	新しいサービスの名前を Jolt リポジトリに追加します。
	Input Buffer Type/Output Buffer Type	VIEW － C 構造体と 16 ビット整数フィールド。特定の構造体を持つサブタイプを含みます。X_C_TYPE と X_COMMON は同等です。X_COMMON は COBOL と C で使用されます。 VIEW32 － VIEW に似ていますが、32 ビット・フィールド ID が VIEW32 構造体要素と関連付けられている点が異なります。 CARRAY －転送時に符号化や復号化の行われない、連続したバイナリ・データ配列。ヌル文字が含まれる場合があります。X_OCTET は同等です。 XML －ウェルフォームド XML 文書。CARRAY に似ています。 FML －各フィールドが固有の定義を持つ型。 FML32 － FML に似ていますが、ID フィールドと長さフィールドの長さが 32 ビットである点が異なります。 STRING －符号化、または復号化されるヌル文字によって終了する文字配列。
	Input View Name/Output View Name	入力 VIEW バッファと出力 VIEW バッファに割り当てられる一意の名前。これらのフィールドが利用できるのは、選択されたバッファ型が VIEW または VIEW32 であるときに限られます。
Export Status	**Unexport Export**	サービスの現在の状態を設定するラジオ・ボタンです。ステータス ([Export] または [Unexport]) が選択されます。デフォルトのステータスは [UNEXPORT] です。

Service level actions	**Save Service**	新しく作成されたサービスをリポジトリに保存します。
	Test	サービスをテストします。このコマンド・ボタンは、新規サービスが作成されるか、または既存のサービスに対して行った編集内容が保存されるまで使用できません。
	Back	前のウィンドウに戻ります。
Parameter	**Parameters**	編集または削除するサービス・パラメータを一覧表示します。
Parameter level actions	**New**	新規パラメータをサービスに追加します。
	Edit	既存のパラメータを編集する場合に使用します。このコマンド・ボタンはパラメータが選択されるまで使用できません。
	Delete	パラメータを削除します。このオプションはパラメータが選択されるまで使用できません。

サービスを追加する手順

1. リポジトリ・エディタの [Logon] ウィンドウで [Services] を選択します。

   4-18 ページの「[Services] ウィンドウの例」に示すような [Services] ウィンドウが表示されます。

2. サービスを追加するパッケージを選択します。

   サービスを追加するパッケージを後で決める場合は、パッケージ・オーガナイザを使ってサービスを別のパッケージに移動します。(その他の情報については、4-34 ページの「パッケージ・オーガナイザを使ってサービスをグループ化する」を参照してください。)

3. [Services] ウィンドウで [New] を選択して、4-25 ページの「[Edit Services] ウィンドウ : 新規パッケージの追加」に示すような [Edit Services] を表示します。

4. [Service Name] テキスト・フィールドを選択して入力可能な状態にします。

5. 追加する新しいサービスの名前を入力します。

6. 入力バッファ型を選択します。

   出力バッファ用のバッファ型には、入力バッファ用に選択されたバッファ型が自動的に選択されます。ただし、このバッファ型は別のバッファ型に変更することができます。

   - VIEW または VIEW32 が選択されたら、入力 VIEW 名と出力 VIEW 名をテキスト・フィールドに入力します。
   - 別のバッファ型を選択した場合は、入力 VIEW 名と出力 VIEW 名のテキスト・フィールドへの入力はできません。
   - CARRAY または STRING を選択した場合は、4-28 ページの「CARRAY または STRING をサービス・バッファ型として選択する」を参照してください。

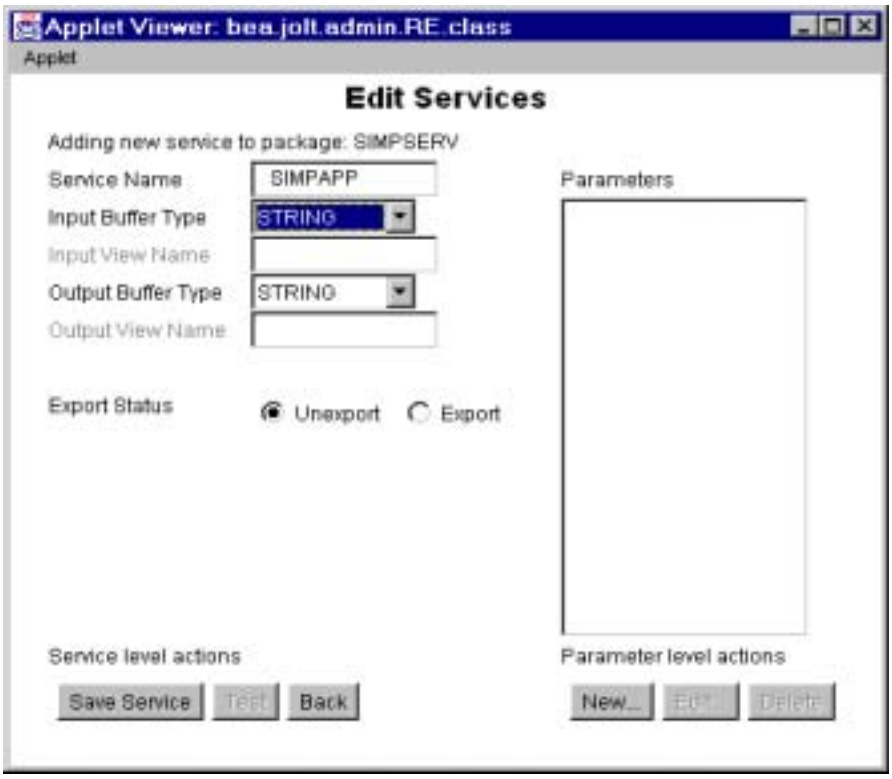
7. [Save Service] を選択して、新規に作成したサービスを保存します。

CARRAY または STRING をサービス・バッファ型として選択する

新規サービスのバッファ型として CARRAY または STRING を選択すると、サービスのパラメータに追加できるデータ型は、CARRAY または STRING に限定されます。この章の 4-29 ページの「パラメータを追加する」および 4-32 ページの「パラメータのデータ型として CARRAY または STRING を選択する」を参照してください。詳細については、第 5 章「Jolt クラス・ライブラリを使う」を参照してください。

次の図は、STRING をサービス SIMPAPP のバッファ型に選択した [Edit Services] ウィンドウの例です。

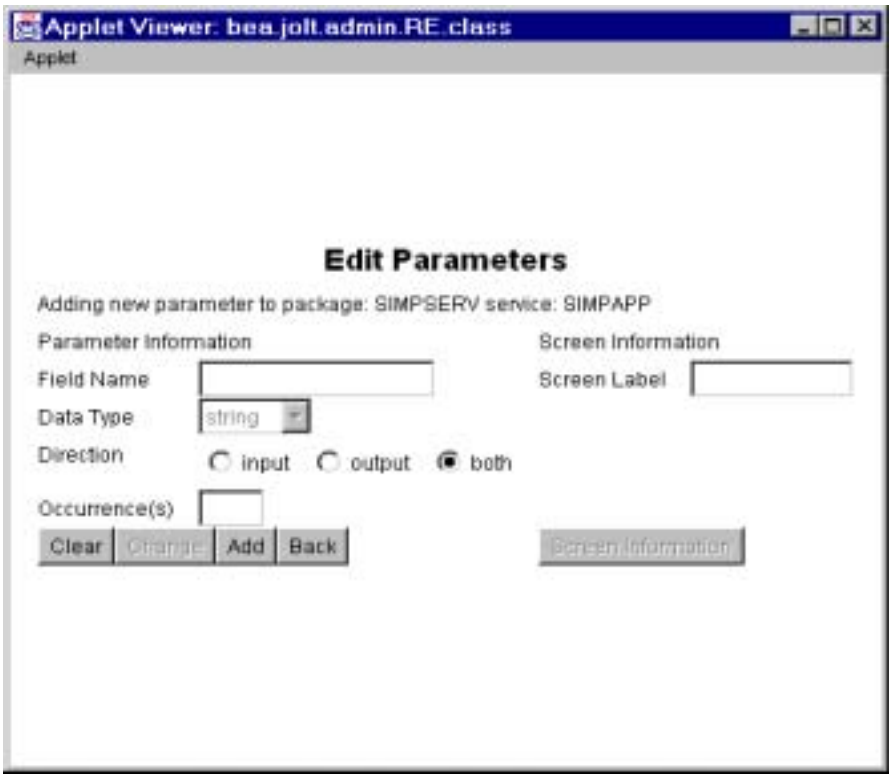
図 4-10[Edit Services] ウィンドウ :STRING バッファ型を選択



パラメータを追加する

[Edit Services] ウィンドウの [Parameter level actions] で [New] をクリックすると、[Edit Parameters] ウィンドウが表示されます。次の図の機能を確認してください。このウィンドウを使って、サービスのパラメータや画面に関する情報を入力します。

図 4-11[Edit Parameters] ウィンドウ : パラメータを追加する



パラメータを追加するウィンドウの説明

オプション	説明
Field Name	フィールド名を追加します (asset、inventory など)。

オプション	説明
Data Type	データ型の種類を表示します。 byte － 8 ビット short － 16 ビット integer － 32 ビット float － 32 ビット double － 64 ビット string －ヌルで終了する文字配列 carray －可変長 8 ビット文字配列
Direction	クライアント / サーバ間での情報の流れを選択するラジオ・ボタンです。 Input －情報はクライアントからサーバに送られます。 Output －情報はサーバからクライアントに送られます。 Both －情報は、クライアントからサーバ、サーバからクライアント、の両方で送られます。
Occurrence(s)	同一のフィールド名が使われる回数。0 は、フィールド名の使用回数が制限されていないことを示します。Occurrence は、テスト画面を作成するために Jolt で使用されます。BEA Tuxedo の送信情報や検索情報を制限するためではありません。
Screen Information	このボタンはウィンドウの起動時は使用できません。
Clear	ウィンドウのフィールドをクリアします。
Change	新規パラメータが追加される間は使用できません。
Add	新規パラメータをサービスに追加します。サービスが保存されるとそのパラメータは保存されます。
Back	前のウィンドウに戻ります。

パラメータを追加する手順

1. [Field Name] を選択してフィールドを入力可能にし、フィールド名を入力します。

   注記　バッファ型が FML または VIEW の場合、フィールド名は FML か VIEW の対応するパラメータのフィールド名に一致している必要があります。

2. データ型を選択します。
3. [input]、[output]、[both] のいずれかのラジオ・ボタンを選択して方向を指定します。
4. [Occurrences] テキスト・フィールドを選択して入力可能にしてから、オカレンスの回数を入力します。
5. [Add] を選択して、情報を追加します。[Add] を実行してもパラメータは保存されません。
6. [Edit Services] ウィンドウで [Save Service] をクリックして、そのパラメータをサービスの一部として保存します。

警告 :　[Back] をクリックする前に [Save Service] をクリックしないと、パラメータはサービスの一部として保存されません。

7. [Back] をクリックして [Edit Services] ウィンドウに戻ります。

パラメータのデータ型として CARRAY または STRING を選択する

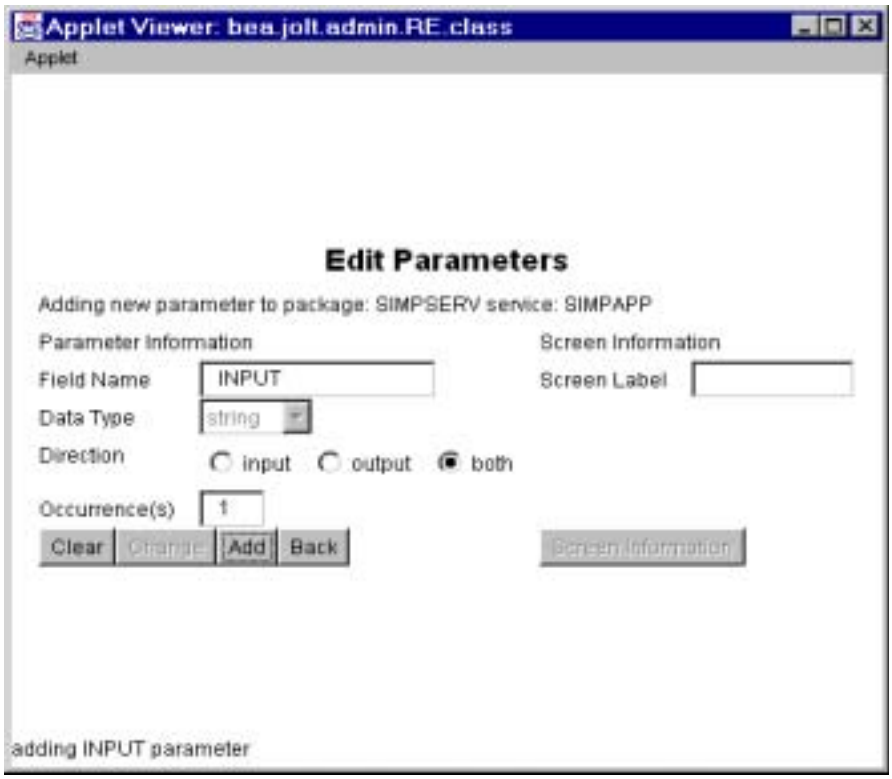
新規サービスのバッファ型として CARRAY または STRING が選択されると、パラメータに追加できるデータ型は、CARRAY または STRING に限定されます。

この場合、1 つのパラメータしか追加できません。CARRAY バッファ型のパラメータ名には「CARRAY」、STRING バッファ型のパラメータ名には「STRING」を指定することをお勧めします。

この章の 4-27 ページの「サービスを追加する手順」および 4-28 ページの「CARRAY または STRING をサービス・バッファ型として選択する」を参照してください。詳細については、第 5 章「Jolt クラス・ライブラリを使う」を参照してください。

次の図は、パラメータのデータ型として String が設定された [Edit Parameters] ウィンドウの例です。[Data Type] はデフォルトで String になります。このデータ型を変更することはできません。[Field Name] には、任意の名前を指定できます。

図 4-12[Edit Parameters] ウィンドウ : string データ型



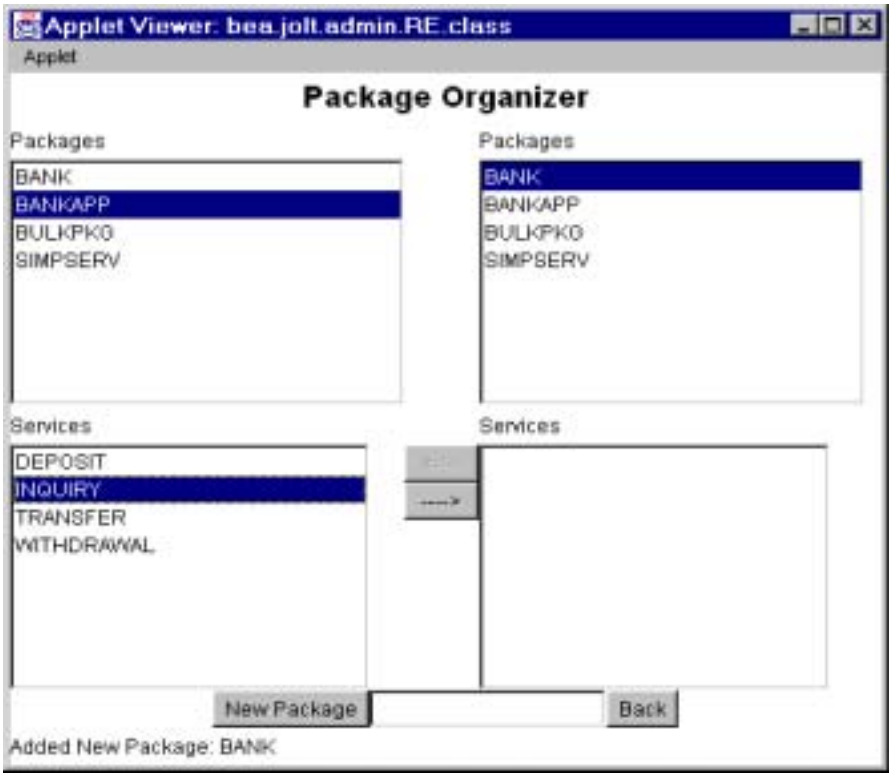
パッケージ・オーガナイザを使ってサービスをグループ化する

パッケージ・オーガナイザを使うと、パッケージ間でサービスを移動することができます。また、関連するサービスをパッケージにグループ化することもできます。たとえば、毎日指定された時刻にエクスポートされる WITHDRAWAL サービスを 1 つのパッケージにグループ化することができます。

パッケージ・オーガナイザの 2 つの矢印ボタンを使うと、サービスを別のパッケージに移動することができます。これらのボタンは、パッケージ間で複数のサービスを移動したい場合に便利です。また、パッケージとサービスのリストボックスから、特定のパッケージ内のサービスを確認することができます。

次の図は、別のパッケージに転送するサービスを選択した [Package Organizer] ウィンドウの例です。

図 4-13[Package Organizer] ウィンドウ



[パッケージ・オーガナイザ] ウィンドウの説明

オプション	説明
Packages (左側のリストボックス)	選択されたパッケージに含まれるサービスを一覧表示します。
Packages (右側のリストボックス)	サービスの移動先として利用できるパッケージを一覧表示します。
Services (左側のリストボックス)	選択されたパッケージに含まれるサービスのうち、利用可能なサービスを一覧表示します。
Services (右側のリストボックス)	反転表示されたパッケージ内のサービスのうち、移動できるサービスを一覧表示します。
Left arrow	左側のパッケージのサービス一覧に移動します (一度に選択できるサービスは 1 つだけです)。
Right arrow	右側のパッケージのサービス一覧に移動します (一度に選択できるサービスは 1 つだけです)。
New Package	新規パッケージの名前を追加します。
Back	前のウィンドウに戻ります。

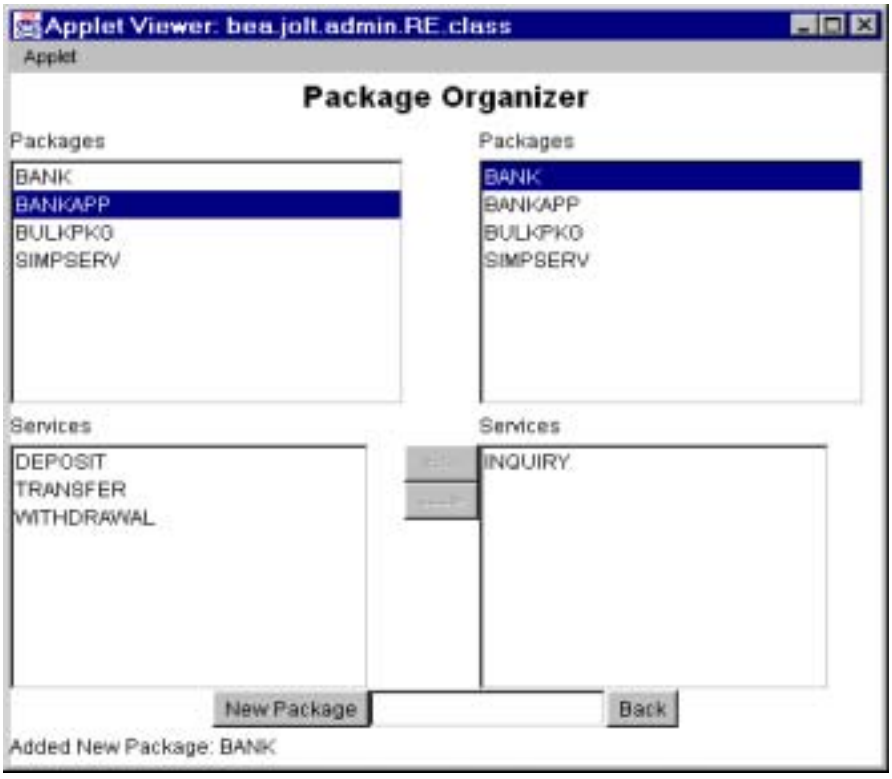
パッケージ・オーガナイザでサービスをグループ化する手順

1. [Packages] ウィンドウで [Package Organizer] をクリックします。
2. [Package Organizer] ウィンドウで、左側の [Packages] リストボックスから移動するサービスを含むパッケージを選択します。
3. 左側の [Services] リストボックスから移動するサービスを選択します。

   前の図では、`INQUIRY` が `BANKAPP` パッケージで選択されたサービスです。
4. 右側の [Packages] リストボックスからのサービスを受け取るパッケージを選択します。

前の図は、移動するサービス INQUIRY と INQUIRY サービスの移動先パッケージ BANK が選択された様子を示しています。

図 4-14 サービスの移動例



5. パッケージ間でサービスを移動するには、左矢印ボタン (←) または右矢印ボタン (→) を使用します。

   これらのボタンは、パッケージ (両側) およびサービスの両方が選択された場合のみアクティブになります。サービスの移動先方向の矢印ボタンのみがアクティブになります。「サービスの移動例」では、INQUIRY サービスが右側の BANK パッケージに移動されています。

   注記　左側と右側のリストボックスで同じパッケージを選択することはできません。

パッケージ、サービス、パラメータを変更する

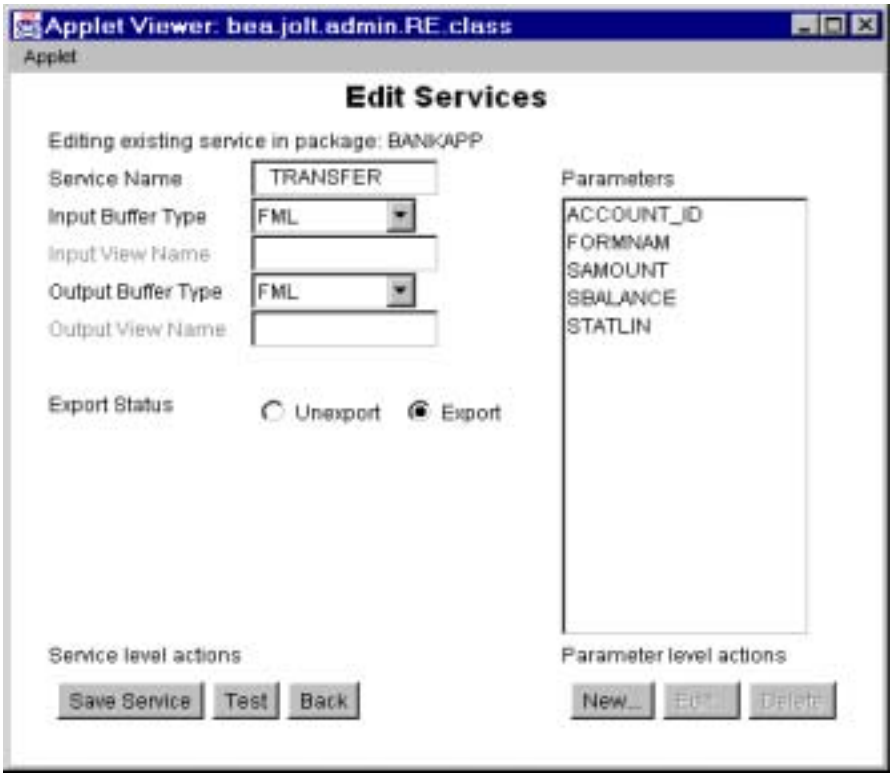
パッケージ、サービス、またはパラメータに対しては、次の変更を行うことができます。

- サービスを編集する
- パラメータを編集する
- パラメータ、サービス、およびパッケージを削除する

サービスを編集する

既存のサービス名やサービス情報を変更したり、既存のサービスに新しいパラメータを追加するためのウィンドウにアクセスすることができます。[Edit Services] ウィンドウの詳細については、4-26 ページの「サービスを追加するウィンドウの説明」を参照してください。次の図は、[Edit Services] ウィンドウの例です。

図 4-15[Edit Services] ウィンドウ



サービスを編集する手順

サービスを編集するには、次の手順に従います。

1. [Services] ウィンドウで、編集が必要なサービスが含まれるパッケージを選択します。

   選択されたパッケージに含まれるサービスのうち、利用可能なサービスを一覧表示します。

2. 編集するサービスを選択します。

   選択されたサービスに含まれるパラメータのうち、利用可能なパラメータを一覧表示します。

3. [Edit] をクリックすると、前の図のような [Edit Services] ウィンドウが表示されます。

4. 新しい情報を入力または選択して [Save Service] をクリックします。

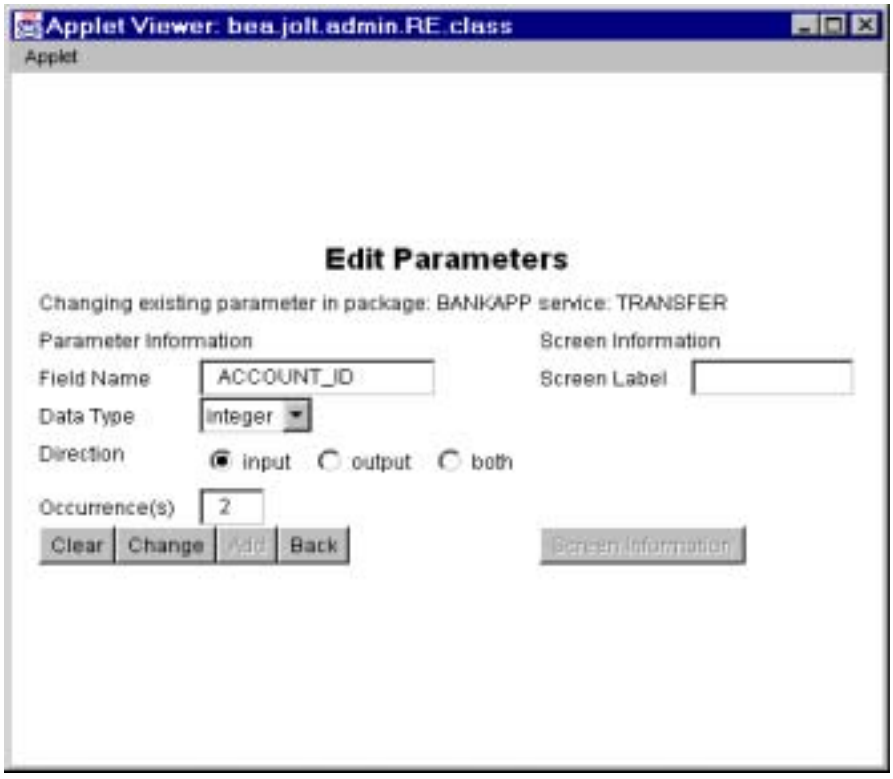
パラメータを編集する

パラメータ名を含むすべてのパラメータの要素は、変更することができます。

警告 : 既存の名前を使って新規パラメータを作成すると、既存のパラメータが上書きされます。

次の図は、[Edit Parameters] ウィンドウの例です。

図 4-16[Edit Parameters] ウィンドウ



パラメータを編集する手順

パラメータを変更するには、次の手順に従います。

1. [Services] ウィンドウで、変更したいパラメータが含まれるパッケージおよびサービスを選択します (「パラメーター一覧が表示された [Services] ウィンドウ」を参照)。
2. [Edit] をクリックし [Edit Services] ウィンドウを表示します。
3. [Parameters] リストボックス内から編集したいパラメータを選択して [Edit] をクリックします。

前の図に示すような [Edit Parameters] ウィンドウが表示されます。

4. 新しい情報を入力して、[Change] をクリックします。
5. [Back] をクリックして前のウィンドウに戻ります。

パラメータ、サービス、およびパッケージを削除する

この節では、パッケージを削除する方法を説明します。パッケージを削除する前に、そのパッケージからすべてのサービスを削除しておくことが必要です。パッケージからサービスなどのコンポーネントがすべて削除されるまで、[Delete] オプションは使用できません。

警告 : 項目を削除するかどうかを確認するメッセージは表示されません。[Delete] を選択する前には、パラメータ、サービス、パッケージが削除される予定になっているかどうか、または既に別の場所に移動されているかどうかを確認してください。

パラメータを削除する

削除するパッケージを決定し、次の手順に従ってください。

1. [Logon] ウィンドウで [Services] を選択して [Services] ウィンドウを表示します。
2. [Services] ウィンドウで、削除したいパラメータが含まれるパッケージおよびサービスを選択します。
3. [Edit] を選択して [Edit Services] ウィンドウを表示します。
4. [Parameters] リストボックス内から削除したいパラメータを選択します。
5. [Parameter level actions] で [Delete] をクリックします。

サービスを削除する

削除するサービスを決定し、次の手順に従ってください。

注記　このオプションを選択する前に、サービス内のすべてのパラメータが削除されていることを確認してください。

1. 削除するサービスが含まれるパッケージを選択します。
2. 削除するサービスを選択します。

   [Delete] が使用可能になります。
3. [Delete] をクリックします。サービスが削除されます。

パッケージを削除する

削除するパッケージを決定し、次の手順に従ってください。このオプションを選択する前に、パッケージ内のすべてのサービスが削除されていること、または別のパッケージに移動されたことを確認してください。

1. リポジトリ・エディタの [Logon] ウィンドウの [Packages] をクリックして、[Packages] ウィンドウを表示します。
2. パッケージを選択します。
3. [Delete] をクリックします。

   パッケージが削除されます。

Jolt クライアントからサービスを利用可能にする

Jolt クライアントでサービスを利用できるようにするには、エクスポートが必要です。パッケージ内のすべてのサービスはグループとしてエクスポートまたはアンエクスポートされなければなりません。[Export] ボタンと [Unexport] ボタンを使用してサービスが利用可能になります。

ここでは、次の内容について説明します。

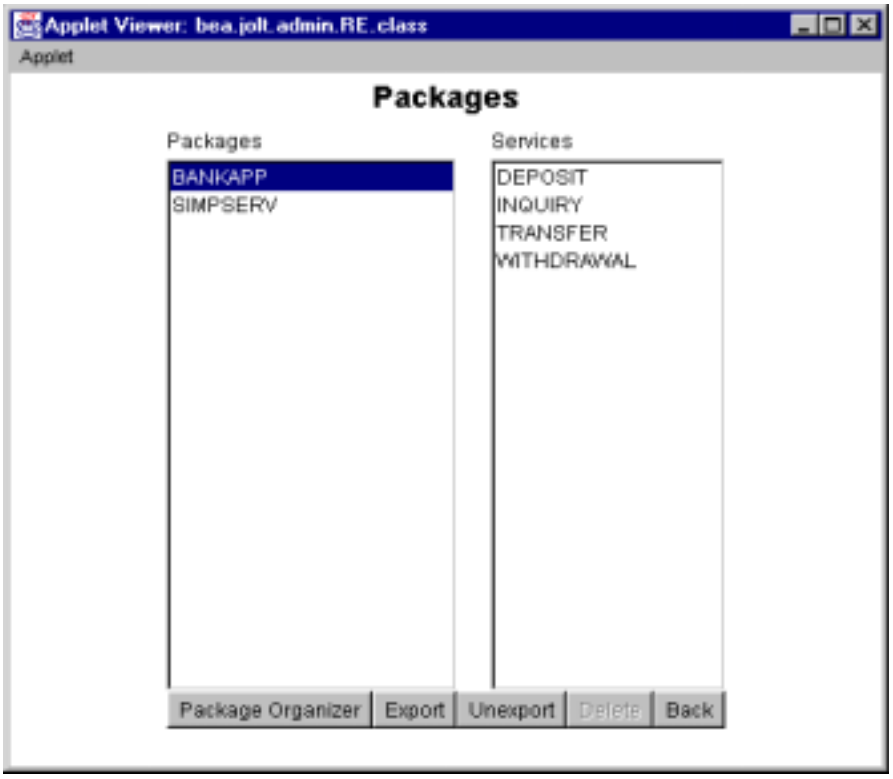
- サービスのエクスポートとアンエクスポート
- エクスポートとアンエクスポートの状態を確認する

サービスのエクスポートとアンエクスポート

Jolt クライアントで使用できるサービスと使用できないサービスを判別します。サービスをエクスポートするのは、Jolt クライアントが Jolt サーバからの最新のサービス定義にアクセスできるようにするためです。

次の図に示すような [Packages] ウィンドウから、サービスをエクスポートしたりアンエクスポートすることができます。

図 4-17[Packages] ウィンドウ : エクスポート・ボタンとアンエクスポート・ボタン



サービスをエクスポートまたはアンエクスポートするには、次の手順に従います。

1. リポジトリ・エディタの [Logon] ウィンドウの [Packages] を選択して、[Packages] ウィンドウを表示します。
2. パッケージを選択します。

   [Export] ボタンと [Unexport] ボタンが有効になります。
3. サービスを利用可能にするには、[Export] ラジオ・ボタンをクリックします。

   サービスを利用不可にするには、[Unexport] ラジオ・ボタンを選択します。

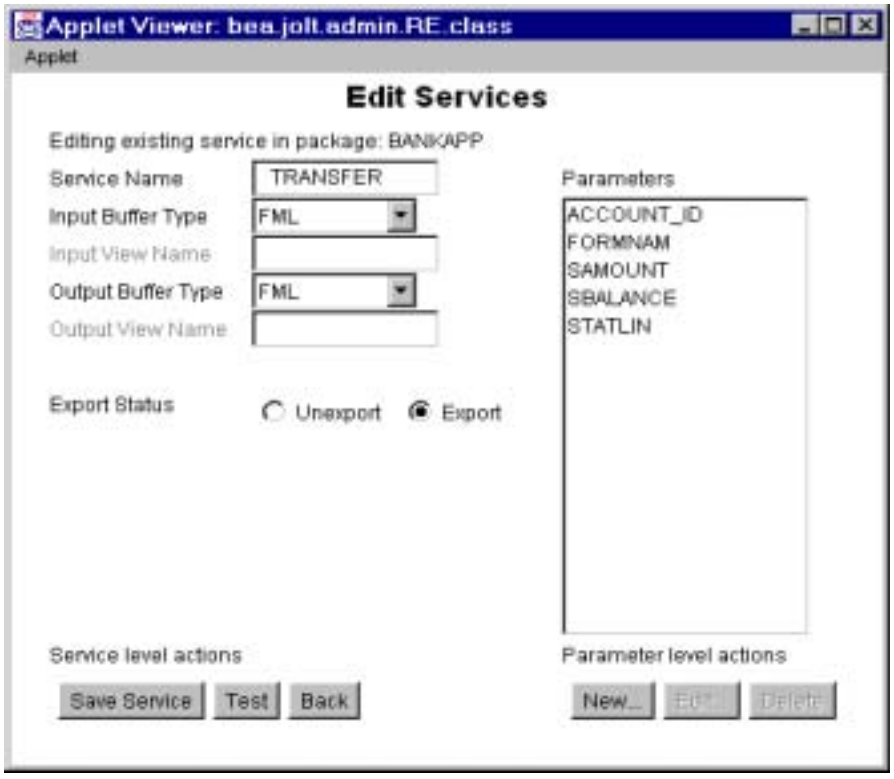
注記　サービスがエクスポートあるいはアンエクスポートされたことを通知するメッセージは表示されません。詳細については、「エクスポートとアンエクスポートの状態を確認する」を参照してください。

エクスポートとアンエクスポートの状態を確認する

サービスがエクスポートあるいはアンエクスポートされると、[Edit Services] ウィンドウでそのステータスを確認できます。

次の図では、[Export Status] の [Export] ラジオ・ボタンがアクティブなので、TRANSFER サービスがエクスポートされます。

図 4-18 エクスポートのステータス



現在のサービスのステータス (エクスポートまたはアンエクスポート) を確認するには、次の手順に従います。

1. リポジトリ・エディタの [Logon] ウィンドウで [Services] を選択すると、4-18 ページの「[Services] ウィンドウの例」に示すような [Services] ウィンドウが表示されます。
2. [Package] リストボックスからパッケージを選択します。

   [Services] リストボックスに選択したパッケージで利用可能なサービスが一覧表示されます。
3. 確認するサービスを選択します。
4. [Edit] をクリックします。

   4-39 ページの「[Edit Services] ウィンドウ」に示すような [Edit Services] ウィンドウが表示されます。

   [Export Status] の横に表示されている [Unexport] または [Export] ラジオ・ボタンは、サービスの現在の状態を示します。

サービスをテストする

Jolt クライアントでサービスおよびそのパラメータを利用可能にする前に、サービスおよびパラメータをテストしてください。サービスやパラメータを変更せずに、現在利用可能なサービスをテストできます。

注記　Jolt のリポジトリ・エディタを使うと、Java コードをまったく記述せずに Jolt で既存の BEA Tuxedo サービスをテストすることができます。

サービスは、エクスポート、アンエクスポートのどちらの状態でもテストできます。サービスやパラメータの変更が必要な場合は、編集する前にそのサービスをアンエクスポートしてください。

ここでは、次の内容について説明します。

- Jolt リポジトリ・エディタの [Service Test] ウィンドウ
- サービスをテストする

Jolt リポジトリ・エディタの [Service Test] ウィンドウ

パラメータ情報が正確かどうかを確認するため、[Run] ボタンを使ってサービスをテストします。サービスのテストは、テスト対象のサービスがある BEA Tuxedo サーバが実行されている場合にのみ可能です。

[Edit Services] ウィンドウの [Test] ボタンは、パラメータがサービスに追加されていなくても使用できますが、その場合、[Service Test] ウィンドウのパラメータ・フィールドが unused と表示され、使用できない状態になります。パラメータ・フィールドの unused の例については、4-50 ページの「[Service Test] ウィンドウの例」を参照してください。

注記　サービス・テスト・ウィンドウは複数オカレンスのパラメータを最大 20 項目まで表示します。20 番目以降の項目は、テストの対象になりません。

次の図は、書き込み可能なテキスト・フィールドと読み取り専用のテキスト・フィールドを持つ [Service Test] ウィンドウの例です。

図 4-19[Service Test] ウィンドウの例

[Service Test] ウィンドウの説明

次の表は、[Service Test] ウィンドウのオプションの説明です。

注記　CARRAY データ・フィールドの各バイトに対して 2 桁の 16 進文字 (0-9、a-f、A-F) を入力することができます。たとえば、10 進数 1234 を 16 進数で表すと 0422 となります。

オプション	説明
Service	テストするサービス名を表示します (読み取り専用)。
表示されているパラメータ	ウィンドウに表示されたパラメータを追跡します (読み取り専用)。
パラメータのテキスト・フィールド	パラメータの情報を入力するテキスト・フィールド。これらのフィールドは、書き込み可能または読み取り専用です。読み取り専用の場合は入力できません。
RUN	入力したデータでテストを実行します。
Clear	テキスト入力フィールドをクリアします。
Next	必要に応じて、別のパラメータ・フィールドを表示します。
Prev	前のパラメータ・フィールドがある場合、それを表示します。
Back	[Edit Services] ウィンドウに戻ります。

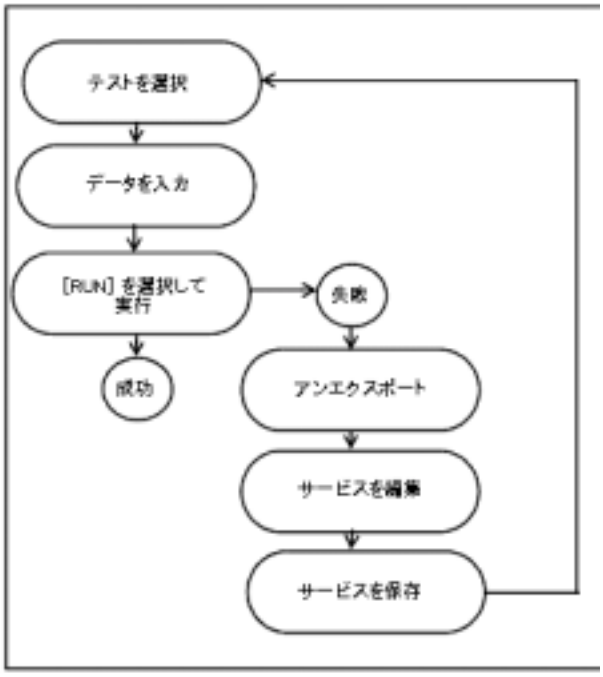
サービスをテストする

サービスやパラメータを変更せずにサービスをテストできます。サービスやパラメータを編集した後、サービスをテストすることもできます。

サービスのテスト手順の流れ

次の図に、リポジトリ・エディタによるサービスのテストの典型的な流れを示します。

図 4-20 サービスのテストの流れ



サービスのテストの手順

サービスをテストするには、次の手順に従います。トラブルシューティングの情報については、リポジトリ・エディタのトラブルシューティングの最初の 2 つの項目を参照してください。

1. リポジトリ・エディタの [Logon] ウィンドウで [Services] を選択します。

   [Services] ウィンドウが表示されます。

2. テストするパッケージとサービスを選択します。
3. [Edit] をクリックして、[Edit Services] ウィンドウにアクセスします。
4. [Test] をクリックして、[Service Test] ウィンドウにアクセスします。
5. [Service Test] ウィンドウのパラメータ・テキスト・フィールドにデータを入力します。
6. [RUN] をクリックします。

   ステータス行に次のような結果が表示されます。

   - テストが成功した場合 : &dlq;`Run C“ted OK”`
   - テストが失敗した場合 : &dlq;`Call “d”`

     リポジトリ・エディタのトラブルシューティング情報については、4-54 ページの「リポジトリ・エディタのトラブルシューティング」を参照してください。

テストの後でサービスを編集する

テストを成功させるためにサービスを編集する場合は、次の手順に従います。

1. リポジトリ・エディタの [Logon] ウィンドウに戻り、[Packages] をクリックします。
2. サービスを再テストするパッケージを選択します。
3. [Unexport] をクリックします。

4. [Back] をクリックして、リポジトリ・エディタの [Logon] ウィンドウに戻ります。
5. [Services] をクリックして [Services] ウィンドウを表示します。
6. パッケージと、編集の必要なサービスを選択して [Edit] をクリックします。
7. [Edit Services] ウィンドウでサービスを編集します。
8. サービスを保存し、[Test] をクリックして、「サービスのテストの手順」の手順 5 と 6 を繰り返します。

リポジトリ・エディタのトラブルシューティング

リポジトリ・エディタの使用中に問題が発生した場合は、以下の表を参照してください。

表 4-5 リポジトリ・エディタのトラブルシューティング

状況	結果
パラメータが不正である	サービスを編集します。
Jolt サーバがダウンしている	サーバを確認してください。BEA Tuxedo サービスがダウンしています。サービスを編集する必要はありません。

表 4-5 リポジトリ・エディタのトラブルシューティング (続き)

状況	結果
エラー・メッセージが返された	使用するブラウザが Java 対応かどうかを確認します。 ■ Netscape ブラウザの場合は、[編集] メニューの [設定] を選択し、[詳細] ウィンドウの [Java を有効にする] および [JavaScript を有効にする] がチェックされていることを確認します。次に、[Communicator] メニューの [Java Console] を選択します。メニューに [Java Console] がない場合、ブラウザは通常 Java をサポートしません。 ■ Internet Explorer の場合は、バージョンが 3.0 以上かどうかを確認します。 ■ Netscape Navigator を使用しているときに発生するエラー・メッセージについては、Java コンソールを参照してください。 ■ `appletviewer` を使用している場合は、システム・コンソール (または、`appletviewer` を開始したウィンドウ) を確認してください。
[Server] と [Port Number] を入力した後で Jolt サーバに接続できない	次の事項を確認してください。 ■ サーバ名が正しいこと (およびマシンからアクセス可能であること)。ポート番号が正しいポートであることを確認してください。そのポートで接続を受け付けるよう設定された JSL または JRLY が必要です。 ■ Jolt サーバが起動し、実行中であることを確認してください。認証機能が有効な場合は、ユーザ名とパスワードが正確に入力されているかどうかを確認してください。 ■ アプレットが HTTP 経由でロードされた場合は、Web サーバ、JRLY および Jolt サーバが同じマシン上で実行されていることを確認してください。つまり、アプレットをダウンロードするときに URL で使用したサーバ名をリポジトリ・エディタに入力します。

表 4-5 リポジトリ・エディタのトラブルシューティング (続き)

状況	結果
リポジトリ・エディタを起動できない	ブラウザでエディタを実行し、http 経由でリポジトリ・エディタのアプレットをダウンロードしている場合は、以下を確認してください。 ■ ブラウザが Java 対応であること。 ■ Web サーバが起動していてアクセス可能であること。 ■ Web サーバから `RE.html` ファイルにアクセスできること。 ■ `RE.html` ファイルに正しい `<codebase>` パラメータが含まれていること。Codebase には、Jolt クラス・ファイルの場所が格納されています。 ブラウザでエディタ (または、`appletviewer`) を実行し、ディスクからアプレットをロードする場合は、以下を確認してください。 ■ ブラウザが Java 対応であること。 ■ `RE.html` ファイルがあり、読み出せること。 ■ `RE.html` ファイルで Java が利用できること。 ■ `RE.html` ファイルに正しい `<codebase>` パラメータ (ローカル・ディスクにインストールされた Jolt クラス・ファイルを格納) があること。 ■ CLASSPATH が設定され、Jolt クラスのディレクトリを指していること。
存在するはずのパッケージまたはサービスが表示されない	■ Jolt リポジトリ・サーバ (JREPSVR) が実行中であることを確認してください。 ■ JREPSVR からリポジトリ・ファイルにアクセスできることを確認してください。 ■ JREPSVR の設定を確認してください。CLOPT パラメータを確認し、`jrep.f16` (FML 定義ファイル) がインストールされ、アクセス可能であることを確認してください。詳細については、インストール関連マニュアルを参照してください。
リポジトリ・エディタの変更を保存できない	リポジトリ・ファイルのパーミッションを確認してください。このファイルの書き込み権は、JREPSVR を起動したユーザに対して与えられていなければなりません。

表 4-5 リポジトリ・エディタのトラブルシューティング (続き)

状況	結果
サービスのテストを実行できない	■ サービスが利用可能かどうかを確認してください。 ■ サービス定義とサービスが一致していることを確認してください。 ■ BEA Tuxedo の認証機能が有効な場合は、サービスの実行に必要なパーミッションを保持しているかどうかを確認してください。 ■ アプリケーション・ファイル (FML または VIEW) が ENVFILE の変数 (FIELDTBLS または VIEWFILES) で正確に指定されているかどうかを確認してください。すべてのアプリケーションの FML フィールド・テーブルまたは VIEW ファイルは、ENVFILE の環境変数である FIELDTBLS と VIEWFILES で指定されていなければなりません。これらのファイルが指定されていないと、JSH によるデータ変換が行われず、「ServiceException: TPEJOLT data conversion failed. (データの変換に失敗しました)」というメッセージが返されます。 ■ さらに、ULOG ファイルを参照して、その他の診断メッセージがないかどうかを確認してください。

5 Jolt クラス・ライブラリを使う

BEA Jolt クラス・ライブラリは、BEA Tuxedo サービスにアクセスするための、開発者向けのオブジェクト指向の Java 言語クラス・セットです。クラス・ライブラリには、Jolt API をインプリメントするクラス・ファイルがあります。これらのクラスを使うと、アプリケーションを拡張してインターネットやイントラネットでトランザクション処理ができるようになります。Jolt クラス・ライブラリを使用して、Java アプレットから BEA Tuxedo サービスへのアクセスをカスタマイズすることもできます。

ここでは、次の内容について説明します。

- クラス・ライブラリの機能の概要
- Jolt オブジェクト間の関係
- Jolt クラス・ライブラリの使い方
- BEA Tuxedo のバッファ型を Jolt で使用する
- マルチスレッド・アプリケーション
- イベント・サブスクリプションおよびイベント通知
- パラメータ値をクリアする
- オブジェクトを再利用する
- Jolt アプレットの配置とローカライズ

以降の節の情報を活用するには、Java 言語によるプログラミングおよびオブジェクト指向プログラミングに関する全般的な知識が必要です。この章で取り上げるプログラムの例はすべて Java コードで記述されています。

注記　プログラムの例では、Jolt の機能を示す部分を抜粋して示しています。そのままコンパイルしたり、実行しないでください。これらのプログラム例を実際に実行するには、コードを追加する必要があります。

クラス・ライブラリの機能の概要

Jolt クラス・ライブラリには、クライアント・サイドのアプリケーションやアプレット (独立した Java アプリケーションとして実行するか、または Java 対応の Web ブラウザで実行) を開発するための BEA Tuxedo アプリケーション開発者向けのツールが用意されています。`bea.jolt` パッケージには Jolt クラス・ライブラリが用意されています。Jolt クラス・ライブラリを使用するには、クライアント・プログラムやアプレットでこのパッケージをインポートする必要があります。`bea.jolt` パッケージをインポートする方法については、5-14 ページの「Jolt のファンド転送の例 (SimXfer.java)」を参照してください。

Java アプリケーションと Java アプレット

ブラウザで実行する Java プログラムは「アプレット」と呼ばれます。アプレットとは、アプリケーションの一部として特定の機能を実行し、簡単にダウンロードできるサイズの小さいプログラムのことです。多くのブラウザでは、高度なセキュリティを実現するため、Java アプレットの機能に制限が設けられています。以下は、アプレットに対するいくつかの制限です。

- アプレットは通常、ホスト・システム上のファイルに対する読み取りや書き込みを行うことができません。
- アプレットは、アプレットを実行するホスト (クライアント) 上のプログラムを起動できません。

- アプレットは、ダウンロード元のホストに対してのみネットワーク接続を行うことができます。それ以外のホスト (クライアント・マシンを含む) に対してネットワーク接続を行うことはできません。

Java アプレットの制限には、大抵の場合プログラミング上の対応策があります。ブラウザでサポートされるアプレットの機能またはアプレットの制限については、お使いのブラウザの Web サイト (www.netscape.com、www.microsoft.com など) や開発者用マニュアルを参照してください。また、Jolt リレーを使うと、ネットワーク接続に関するいくつかの制限を回避することができます。

一方、Java アプリケーションはブラウザ上では実行されず、上記のような制限を受けることはありません。たとえば、Java アプリケーションは、実行中のホスト・マシン上で別のアプリケーションを起動できます。アプレットがユーザ・インターフェイスとしてブラウザや appletviewer のウィンドウ環境に依存するのに対し、Java アプリケーションでは独自のユーザ・インターフェイスを作成する必要があります。また、アプレットが可搬性のある小さいプログラムとして設計されるのに対し、Java アプリケーションは Java 以外の言語で作成されたプログラムとほとんど同じように操作することができます。Java アプリケーションと Java アプレットの最大の違いは、アプレットに対して設けられているブラウザでのセキュリティの制限およびプログラムの適応範囲です。

Jolt クラス・ライブラリの特徴

Jolt クラス・ライブラリには次の特徴があります。

- クラスは完全にスレッド・セーフです。
- 典型的なトランザクション処理 (ログオン、同期型の呼び出し、トランザクションの開始、トランザクションのコミット、トランザクションのロールバック、ログオフなど) を Java オブジェクトにカプセル化します。
- 連続的または断続的なクライアント・ネットワーク接続に対してアイドル・タイムアウト値を設定するメソッドが用意されています。

- Jolt クライアントがイベントに基づいた通知をサブスクライブし、受信することができるメソッドが用意されています。

エラーと例外処理

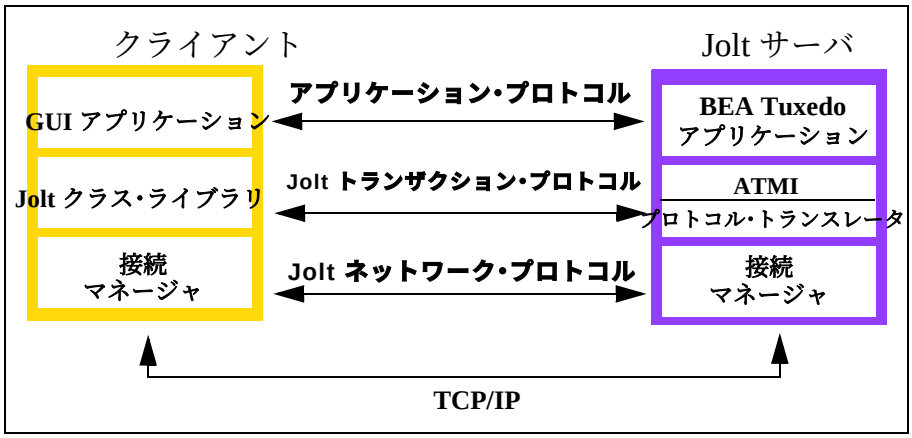
Jolt クラス・ライブラリは、Jolt インタープリタのエラーおよび BEA Tuxedo のエラーを例外として返します。「Jolt Class Library Reference」では、Jolt クラスと、各クラスのエラーや例外を一覧表示します。『BEA Jolt API リファレンス』には、エラーと例外のクラス・リファレンスが含まれています。

Jolt でのクライアント / サーバ関係

BEA Jolt は、分散型のクライアント / サーバ環境で動作し、Java クライアントを BEA Tuxedo をベースとするアプリケーションに接続します。

次の図は、Jolt プログラムと Jolt サーバの間のクライアント / サーバ関係を示しています。

図 5-1Jolt でのクライアント / サーバ関係



図に示すように、Jolt サーバはネイティブな BEA Tuxedo クライアントのプロキシとして動作し、ネイティブな BEA Tuxedo クライアントで利用できる機能をインプリメントします。BEA Jolt サーバは、BEA Jolt クライアントからのリクエストを受け取ると、BEA Tuxedo ATMI インターフェイスを介してこれらのリクエストを BEA Tuxedo サービス要求にマッピングします。リクエストおよび関連付けられたパラメータは、メッセージ・バッファにパッケージされ、ネットワーク経由で BEA Jolt サーバに配信されます。BEA Jolt サーバと BEA Jolt アプレットの間のすべての通信は、BEA Jolt 接続マネージャにより、BEA Jolt トランザクション・プロトコルを使用して処理されます。BEA Jolt サーバは、メッセージからデータを取り出し、必要に応じてデータ変換 (数値形式や文字セットなど) を行ってから、メッセージで指定された適切なサービスを BEA Tuxedo に要求します。

BEA Tuxedo システムが受け取ったサービス要求は、ほかの BEA Tuxedo のリクエストとまったく同じ方法で処理されます。結果は ATMI インターフェイスを通して BEA Jolt サーバに返され、そこで結果とエラー情報がメッセージにパッケージ化されると、BEA Jolt のクライアント・アプレットに送信されます。BEA Jolt クライアントは、メッセージの内容をさまざまな BEA Jolt クライアントのインターフェイス・オブジェクトにマッピングして、リクエストの処理を完了します。

クライアント側では、ユーザ・プログラムにクライアント・アプリケーション・コードが含まれています。Jolt クラス・ライブラリが提供する JoltSession と JoltTransaction がサービス要求を処理します。

次の表は、クライアント側から送信されるリクエストと、それに対する Jolt サーバ側のアクションを簡単なプログラムの例で示したものです。

表 5-1Jolt クライアント / サーバの相互作用

Jolt クライアント	Jolt サーバ
1 `attr=new JoltSessionAttributes();` `attr.setString(attr.APPADDRESS, “//myhost:8000”);`	クライアントを BEA Tuxedo 環境にバインドします。
2 `session=new JoltSession(attr, username, userRole, userPassword, appPassword);`	クライアントを BEA Tuxedo にログオンさせます。
3 `withdrawal=new JoltRemoteService( servname, session );`	リポジトリ内のサービス属性を検索します。
4 `withdrawal.addString(“accountnumber”, “123”);` `withdrawal.addFloat(“amount”, (float) 100.00);`	クライアントに変数を設定します (Jolt サーバ側の動作ではありません)。
5 `trans=new JoltTransaction( time-out, session);`	新しい Tuxedo トランザクションを開始します。
6 `withdrawal.call(trans);`	BEA Tuxedo サービスを実行します。

表 5-1Jolt クライアント / サーバの相互作用

Jolt クライアント	Jolt サーバ
7 `trans.commit() or trans.rollback();`	トランザクションを完了またはロールバックします。
8 `balance=withdrawal.getFloatDef("balance,&drq; (float) 0.0);`	結果を取り出します (Jolt サーバ側の動作ではありません)。
9 `session.endSession();`	クライアントを BEA Tuxedo からログオフさせます。

「Jolt クライアント / サーバの相互作用」をまとめると、次のようになります。

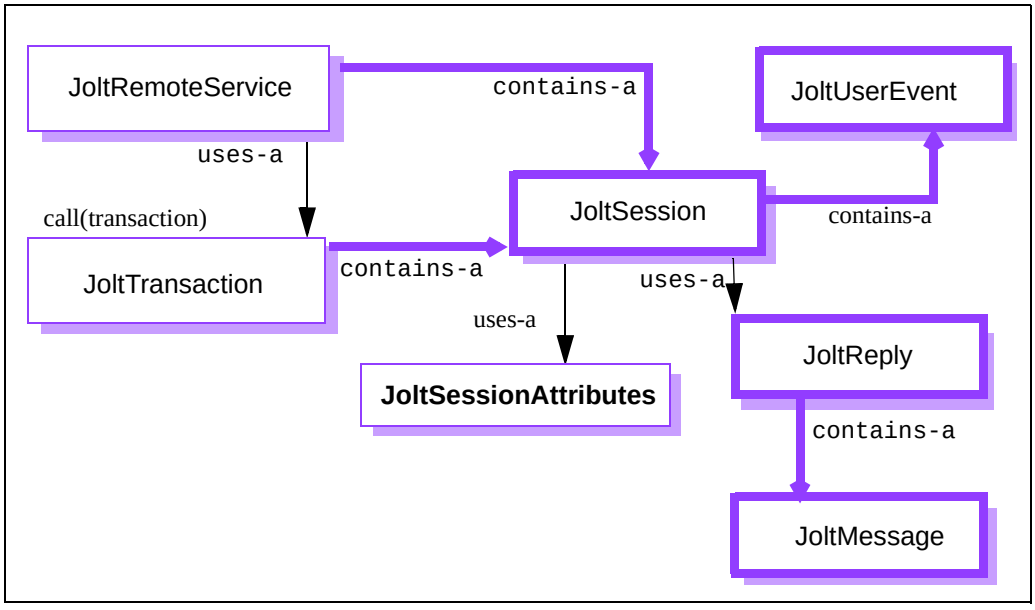
1. `JoltSessionAttributes` クラスを使用してクライアントを BEA Tuxedo 環境にバインドします。
2. セッションを確立します。
3. 変数を設定します。
4. 必要なトランザクション処理を行います。
5. クライアントを BEA Tuxedo からログオフさせます。

上記のアクティビティは、Jolt クラス・ライブラリのクラスによって処理されます。これらのクラスには、データの設定や消去を行ったり、リモート・サービスのアクションを処理するメソッドが用意されています。Jolt クラス・ライブラリの詳細については、5-8 ページの「Jolt オブジェクト間の関係」を参照してください。

Jolt オブジェクト間の関係

次の図は、Jolt クラス・ライブラリのクラスをインスタンス化したオブジェクトの関係を示しています。

図 5-2Jolt オブジェクト間の関係



Jolt クラスは、オブジェクトとしてさまざまな関係で相互作用します。上の図は、3 種類の基本的な関係を示しています。

- contains-a 関係 6 クラス・レベルでは、1 つのオブジェクトに別のオブジェクトを含めることができます。たとえば、JoltTransaction は JoltSessionJoltSession オブジェクトを格納します (または含みます)。
- is-a 関係― is-a 関係は通常、クラス・インスタンス・レベルまたはサブ・オブジェクト・レベルで成立し、オブジェクトが特定のオブジェクトのインスタンスであることを示します。

- uses-a 関係－オブジェクトが別のオブジェクトを格納することなく使用できることを示します。たとえば JoltSession は JoltSessionAttributes オブジェクトを使用して、ホストとポート情報を入手できます。

Jolt クラス・ライブラリの使い方

Jolt クラスを使用すると、ログオン / ログオフ、同期型のサービス呼び出し、トランザクションの開始、トランザクションのコミット、トランザクションのロールバックなどの基本的なトランザクション処理を実行できます。次の節では、これらの機能を実行するための Jolt クラスの使い方を説明します。

- ログオン / ログオフ
- 同期型のサービス呼び出し
- トランザクションの開始、コミット、およびロールバック

Jolt クラス・ライブラリを使用して、マルチスレッド化されたアプリケーションを開発したり、Tuxedo 固有のバッファ型を定義したり、イベントや任意通知型メッセージのサブスクライブを実行することもできます。これらの機能については、後の節で説明します。

ログオン / ログオフ

クライアント・アプリケーションには、トランザクション処理を行う前に BEA Tuxedo 環境にログオンしておく必要があります。Jolt クラス・ライブラリには、BEA Tuxedo システムとの接続を確立するための JoltSessionAttributes クラスと JoltSession クラスが用意されています。

JoltSessionAttributes クラスには、さまざまなプロパティのほか、Jolt および BEA Tuxedo システムへの接続プロパティが格納されます。接続を確立するには、クライアント・アプリケーション側で JoltSession クラスのインスタンスを作成する必要があります。このインスタンスが JoltSession オブジェクトです。開発者が Jolt セッションおよび BEA Tuxedo オブジェクトをインスタンス化すると、Jolt および BEA Tuxedo のログオン機能が有効になります。endSession メソッドを呼び出すと、セッションは終了し、ユーザのログオフが可能になります。

同期型のサービス呼び出し

要求や応答などのトランザクションは、JoltRemoteService オブジェクト (JoltRemoteService クラスのインスタンス) を使用して処理されます 。各 JoltRemoteService オブジェクトは、エクスポートされた BEA Tuxedo の要求 / 応答サービスを参照します。JoltRemoteService オブジェクトを使用する前には、サービス名と JoltSession オブジェクトを指定して、JoltRemoteService オブジェクトをインスタンス化しておく必要があります。

JoltRemoteService オブジェクトを使用するには、次の手順に従います。

1. 入力パラメータを設定します。
2. サービスを呼び出します。
3. 出力パラメータを検証します。

効率を高めるため、Jolt では入力パラメータ・オブジェクトはコピーされません。オブジェクトへの参照 (文字列とバイト配列) だけが保存されます。JoltRemoteService オブジェクトは、状態を持つオブジェクトです。そのため、このオブジェクトの入力パラメータとリクエスト属性は、オブジェクトの寿命がある限り維持されます。JoltRemoteService オブジェクトを再利用する前に、`clear()` メソッドを使用して、属性や入力パラメータをリセットできます。

Jolt はマルチスレッド環境用に設計されているので、Java のマルチスレッド機能を使用して複数の JoltRemoteService オブジェクトを同時に呼び出すことができます。詳細については、5-46 ページの「マルチスレッド・アプリケーション」を参照してください。

トランザクションの開始、コミット、およびロールバック

Jolt では、トランザクションは JoltTransaction クラスのオブジェクトとして表されます。トランザクションは、トランザクション・オブジェクトがインスタンス化されると開始されます。次に示すように、トランザクション・オブジェクトの作成時には、タイムアウト値と JoltSession オブジェクトのパラメータが設定されます。

```
trans = new JoltTransaction(timeout, session)
```

Jolt では、トランザクションに関与するすべてのサービスに対して明示的なトランザクション・モデルを使用します。トランザクション・サービスを呼び出すには、JoltTransaction オブジェクトがパラメータとして必要です。また、サービスとトランザクションが同じセッションに属していることも必要です。Jolt では、同じセッションにバインドされていないサービスとトランザクションを使用することはできません。

5-14 ページの「Jolt のファンド転送の例 (SimXfer.java)」のコード例は、Jolt クラス・ライブラリの使い方を示しています。ここでは、JoltSessionAttributes、JoltSession、JoltRemoteService、および JoltTransaction クラスを使用しています。

例では、ユーザ定義された 2 つの BEA Tuxedo サービス (WITHDRAWAL と DEPOSIT) をバインドして、TRANSFER トランザクションを擬似的に実行します。WITHDRAWAL 操作が失敗した場合は、ロールバックが実行されます。それ以外の場合は DEPOSIT が実行され、commit によりトランザクションが完了します。

5-14 ページの「Jolt のファンド転送の例 (SimXfer.java)」のコード例に示すトランザクション・プロセスのプログラミング手順は、以下のとおりです。

1. *`hostname`* や *`portnumber`* などの接続属性を JoltSessionAttribute オブジェクトに設定します。

   コード・リストで次の行を参照してください。

   ```
   sattr = new JoltSessionAttributes();
   ```

2. `sattr.checkAuthenticationLevel()` を使うと、サーバへのログオンに必要なセキュリティ・レベルをアプリケーション側で決定できます。

   コード・リストで次の行を参照してください。

   ```
   switch (sattr.checkAuthenticationLevel())
   ```

3. JoltSession オブジェクトをインスタンス化してログオンします。

   コード・リストで次の行を参照してください。

   ```
   session = new JoltSession (sattr, userName, userRole,
   userPassword, appPassword);
   ```

   この例では、`SessionException` エラーを明示的にキャッチしません。

4. すべての JoltRemoteService 呼び出しでは、サービスの指定と `JoltSession()` から返されたセッション・キーが必要です。

   コード・リストで次の行を参照してください。

   ```
   withdrawal = new JoltRemoteService(“WITHDRAWAL”, session);
   ```

   ```
   deposit = new JoltRemoteService(“DEPOSIT”, session);
   ```

   これらの呼び出しは、Jolt リポジトリに格納されている WITHDRAWAL および DEPOSIT のサービス定義を、withdrawal オブジェクトおよび deposit オブジェクトにそれぞれバインドします。WITHDRAWAL および DEPOSIT サービスが Jolt リポジトリで定義されていないと、ServiceException がスローされます。この例では、ServiceException エラーを明示的にキャッチしません。

5. サービス定義が返されると、アプリケーション固有のフィールド (アカウント番号を示す ACCOUNT_ID や引き出し額を示す SAMOUNT など) に自動的に値が設定されます。

   コード・リストで次の行を参照してください。

   ```
   withdrawal.addInt(“ACCOUNT_ID”, 100000);
   ```

   ```
   withdrawal.addString(“SAMOUNT”, “100.00”);
   ```

   `add*()` メソッドでは、`IllegalAccessError` または `NoSuchFieldError` 例外がスローされる場合があります。

6. JoltTransaction 呼び出しでは、指定した時間内にトランザクションが完了しない場合のタイムアウト値を指定できます。

コード・リストで次の行を参照してください。

```
trans = new JoltTransaction(5,session);
```

7. withdrawal サービスの定義が自動的に設定された後で `withdrawal.call(trans)` メソッドを呼び出すと、withdrawal サービスが呼び出されます。

   コード・リストで次の行を参照してください。

   ```
   withdrawal.call(trans);
   ```

8. 失敗した WITHDRAWAL はロールバックできます。

   コード・リストで次の行を参照してください。

   ```
   trans.rollback();
   ```

9. または、いったん DEPOSIT が実行されると、すべてのトランザクションがコミットされます。コード・リストで次の行を参照してください。

   ```
   deposit.call(trans);

   trans.commit();
   ```

次のリストは、Jolt クラスを使用してファンドを転送する簡単なアプリケーションの例です。

コード リスト 5-1　Jolt **のファンド転送の例** (SimXfer.java)

```
/* Copyright 1999 BEA Systems, Inc.  All Rights Reserved */
import bea.jolt.*;
public class SimXfer
{
    public static void main (String[] args)
    {
        JoltSession session;
        JoltSessionAttributes sattr;
        JoltRemoteService withdrawal;
        JoltRemoteService deposit;
        JoltTransaction trans;
        String userName=null;
        String userPassword=null;
        String appPassword=null;
        String userRole=”myapp”;
```

```
sattr = new JoltSessionAttributes();
sattr.setString(sattr.APPADDRESS, “//bluefish:8501”);

switch (sattr.checkAuthenticationLevel())
{
case JoltSessionAttributes.NOAUTH:
    System.out.println(“NOAUTH\n”);
    break;
case JoltSessionAttributes.APPASSWORD:
    appPassword = “appPassword”;
    break;
case JoltSessionAttributes.USRPASSWORD:
     userName = “myname”;
    userPassword = “mysecret”;
    appPassword = “appPassword”;
    break;
}
sattr.setInt(sattr.IDLETIMEOUT, 300);
session = new JoltSession(sattr, userName, userRole,
userPassword, appPassword);
// 転送をシミュレートする。
withdrawal = new JoltRemoteService(“WITHDRAWAL”, session);
deposit = new JoltRemoteService(“DEPOSIT”, session);

withdrawal.addInt(“ACCOUNT_ID”, 100000);
withdrawal.addString(“SAMOUNT”, “100.00”);

// トランザクションの開始。タイムアウトは 5 秒
trans = new JoltTransaction(5, session);
try
{
    withdrawal.call(trans);
}

catch (ApplicationException e)
{
    e.printStackTrace();
    // このサービスでは STATLIN フィールドを
    // クライアント・アプリケーションのエラー報告に使用する。
  System.err.println(withdrawal.getStringDef(“STATLIN”,”NO
    STATLIN”));
    System.exit(1);
}

String wbal = withdrawal.getStringDef(“SBALANCE”, “$-1.0”);

// 文字列を浮動小数に変換する前に「$」記号を削除する。
float w = Float.valueOf(wbal.substring(1)).floatValue();
if (w < 0.0)
```

```
        {
            System.err.println("Insufficient funds");
            trans.rollback();
            System.exit(1);
        }
        else            // ファンドの預け入れ / 転送をする。
        {
            deposit.addInt("ACCOUNT_ID", 100001);
            deposit.addString("SAMOUNT", "100.00");

            deposit.call(trans);
           String dbal = deposit.getStringDef("SBALANCE", "-1.0");
            trans.commit();

            System.out.println("Successful withdrawal");
            System.out.println("New balance is: " + wbal);

            System.out.println("Successful deposit");
            System.out.println("New balance is: " + dbal);
        }

        session.endSession();
        System.exit(0);
        } // main の終了
    } // SimXfer の終了
```

BEA Tuxedo のバッファ型を Jolt で使用する

Jolt では、次の BEA Tuxedo 組み込みのバッファ型がサポートされています。

- FML、FML32
- VIEW、VIEW32
- X_COMMON
- X_C_TYPE
- CARRAY
- X_OCTET
- STRING
- XML

注記　X_OCTET の使用方法は CARRAY と同じです。
X_COMMON および X_C_TYPE の使用方法は VIEW と同じです。

Jolt アプリケーションのプログラマは、BEA Tuxedo に組み込まれているバッファ型のうち、特に CARRAY (文字配列) および STRING の扱い方を知っておく必要があります。

- CARRAY 型はデータを非透過的に処理するために使用されます (つまり、CARRAY データ型の文字はどんな形式にも解釈されません)。Jolt クライアントと BEA Tuxedo サービス間でのデータ変換は行われません。
- STRING のデータ型は文字であり、CARRAY とは違って、ヌル文字にまでの文字数を指定して転送長を設定できます。そのため、異なる文字セットを使用するマシン間でデータを交換する場合は、データが自動的に変換されます。

BEA Tuxedo の型付きバッファ、データ型、バッファ型に関するすべての情報については、以下のマニュアルを参照してください。

- 『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』
- 『BEA Tuxedo C リファレンス』
- 『BEA Tuxedo FML リファレンス』
- 『BEA Tuxedo のファイル形式とデータ記述方法』

STRING バッファ型を使う

STRING バッファ型は、ヌル以外の文字配列で構成され、最後がヌル文字で終了します。CARRAY とは違って、ヌル文字までの文字数を指定して転送長を設定できます。STRING バッファは、自己記述型です。そのため、異なる文字セットを使用するマシン間でデータを交換する場合は、BEA Tuxedo システムによってデータが自動的に変換されます。

注記　Jolt から STRING へのデータ変換時には、STRING バッファの終端にヌル終結文字が自動的に追加されます。Java 文字列はヌルで終結しないためです。

STRING バッファ型を使うには、次の 2 つの操作が必要です。

1. バッファ型で使用する Tuxedo サービスを定義します。
2. STRING バッファ型を使用するコードを記述します。

次の 2 つの節では、これらの手順について例を挙げて説明します。

5-21 ページの「STRING バッファ型の使い方 (ToUpper.java)」に示すコード `ToUpper` は、STRING バッファ型が設定されたサービスが Jolt でどのように動作するかを示しています。BEA Tuxedo サービス `ToUpper` は、BEA Tuxedo のサンプル `simpapp` で利用できます。

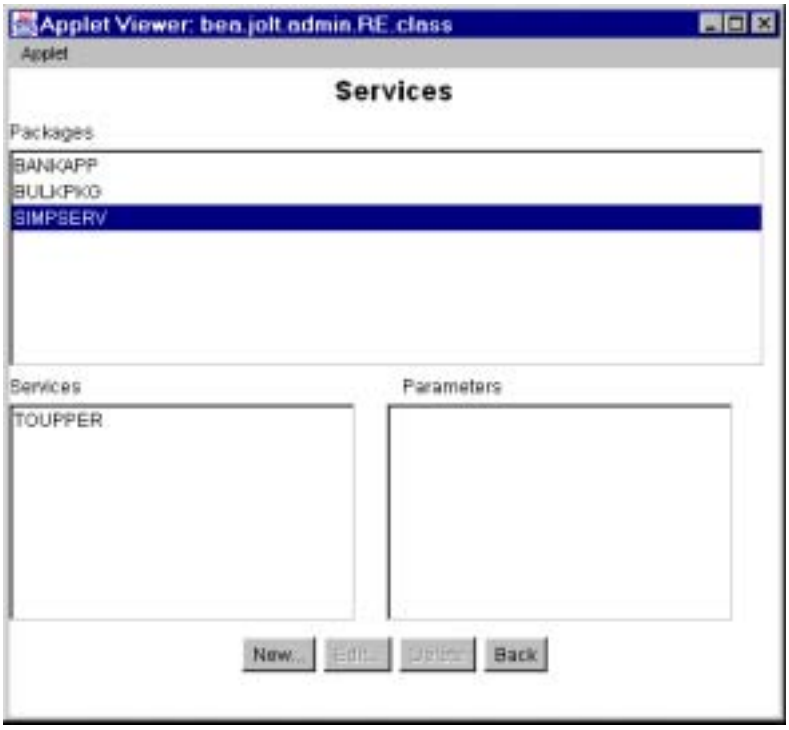
Jolt リポジトリ・エディタで TOUPPER サービスを定義する

`ToUpper.java` を実行する前に、Jolt リポジトリ・エディタを使用して `TOUPPER` サービスを定義する必要があります。

注記　サービスを定義したり、新しいパラメータを追加する方法については、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

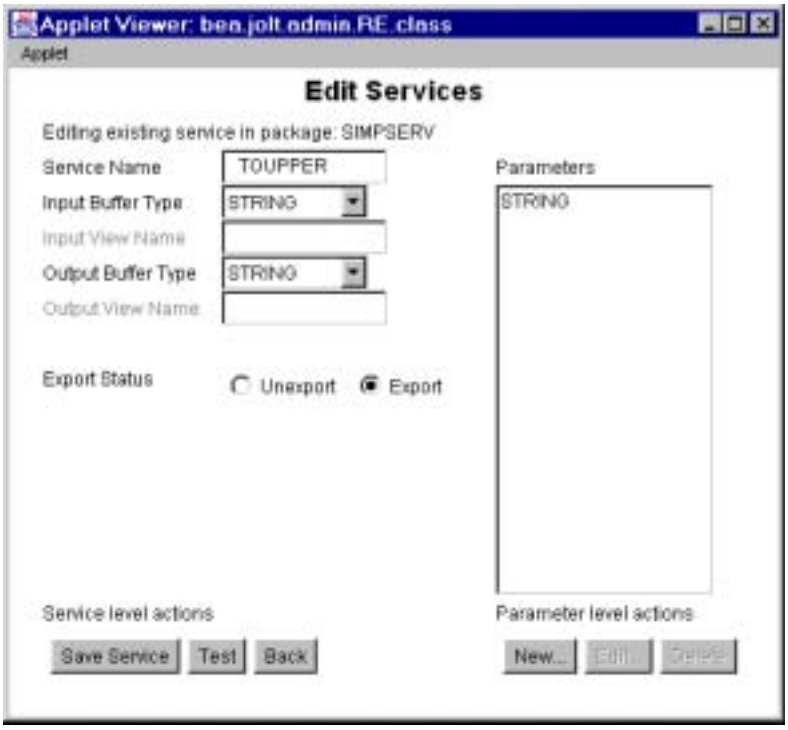
1. リポジトリ・エディタの [Logon] ウィンドウの [Services] をクリックします。

図 5-3TOUPPER サービスを追加する



2. [Services] ウィンドウで SIMPSERV パッケージの TOUPPER サービスを選択します。
3. [Edit] をクリックします。

図 5-4 入力バッファ型と出力バッファ型を STRING に設定する



4. [Edit Services] ウィンドウで、STRING の入力バッファ型と STRING の出力バッファ型を定義します。5-20 ページの「入力バッファ型と出力バッファ型を STRING に設定する」を参照してください。
5. TOUPPER サービスの入出力用のパラメータとして STRING を 1 つだけ定義します。
6. [Save Service] をクリックします。

ToUpper.java クライアント・コード

次のリスト内に示す Java コード `ToUpper.java` は、STRING バッファ型が設定されたサービスが Jolt でどのように動作するかを示しています。この例では、STRING バッファ型を使用する Jolt クライアントがサーバにデータを渡

す様子を示しています。BEA Tuxedo サーバはバッファを受け取り、文字列をすべて大文字に変換し、変換後の文字列をクライアントに返します。次の例では、セッション・オブジェクトが既にインスタンス化されていることを想定しています。

コード リスト 5-2　STRING **バッファ型の使い方** (ToUpper.java)

```
/* Copyright 1996 BEA Systems, Inc.  All Rights Reserved */
import bea.jolt.*;
public class ToUpper
  {
    public static void main (String[] args)
    {
        JoltSession            session;
        JoltSessionAttributes  sattr;
        JoltRemoteService      toupper;
        JoltTransaction        trans;
        String userName=null;
        String userPassword=null;
        String appPassword=null;
        String userRole=”myapp”;
        String outstr;

        sattr = new JoltSessionAttributes();
        sattr.setString(sattr.APPADDRESS, “//myhost:8501”);

        switch (sattr.checkAuthenticationLevel())
        {
        case JoltSessionAttributes.NOAUTH:
            break;
        case JoltSessionAttributes.APPASSWORD:
            appPassword = “appPassword”;
            break;
        case JoltSessionAttributes.USRPASSWORD:
            userName = “myname”;
            userPassword = “mysecret”;
            appPassword = “appPassword”;
            break;
        }
        sattr.setInt(sattr.IDLETIMEOUT, 300);
        session = new JoltSession(sattr, userName, userRole,
        userPassword, appPassword);
        toupper = new JoltRemoteService (“TOUPPER”, session);
        toupper.setString(“STRING”, “hello world”);
        toupper.call(null);
        outstr = toupper.getStringDef(“STRING”, null);
```

```
        if (outstr != null)
            System.out.println(outstr);

        session.endSession();
                  System.exit(0);
        } // main の終了
    } // ToUpper の終了
```

CARRAY バッファ型を使う

CARRAY バッファ型は、BEA Tuxedo システムに組み込まれている文字配列による単純なバッファ型です。CARRAY バッファ型のデータはシステム側で解釈されないため、データ型が明らかであっても Jolt クライアント・アプリケーションでデータの長さを指定する必要があります。このバッファ型を処理するときは、常に Jolt クライアントでデータの長さを指定してください。

たとえば、BEA Tuxedo サービスが CARRAY バッファ型を使用する場合、ユーザが 32 ビットの integer を設定すると (Java では integer はビッグエンディアン・バイト順)、そのデータは変換されずに BEA Tuxedo サービスに送信されます。

CARRAY バッファ型を使用するには、まずバッファ型で使用する Tuxedo サービスを定義します。次に、そのバッファ型を使用するコードを記述します。次の 2 つの節では、これらの手順について説明します。

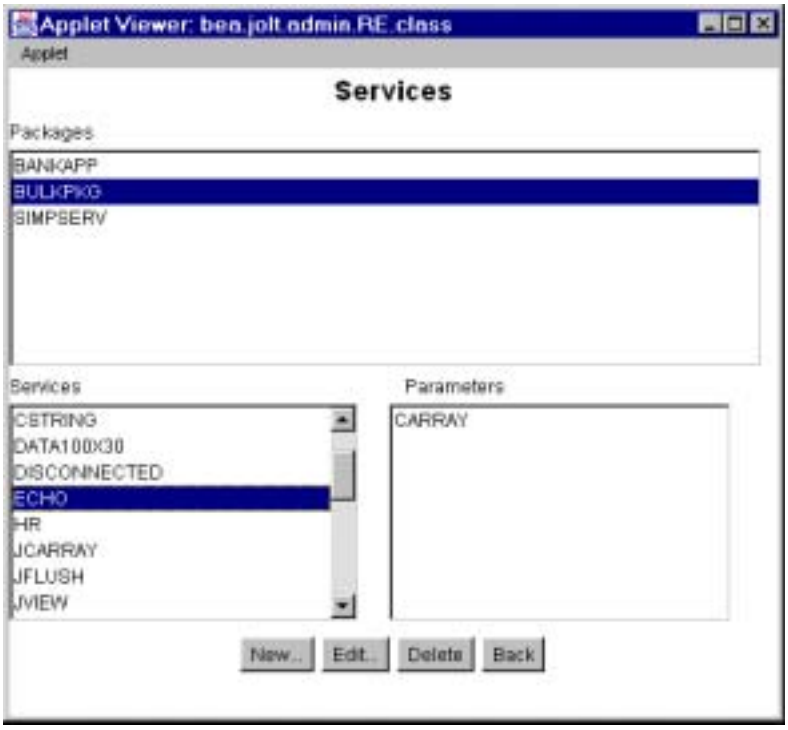
注記　X_OCTET の使用方法は CARRAY と同じです。

リポジトリ・エディタで Tuxedo サービスを定義する

例を実行する前に、BEA Tuxedo の ECHO サービスを作成し、起動する必要があります。ECHO サービスは、バッファを取得し、Jolt クライアントに返します。Jolt リポジトリ・エディタを使用して ECHO サービスを定義します。

注記　サービスを定義したり、新しいパラメータを追加する方法については、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

図 5-5 リポジトリ・エディタ :ECHO サービスを追加する

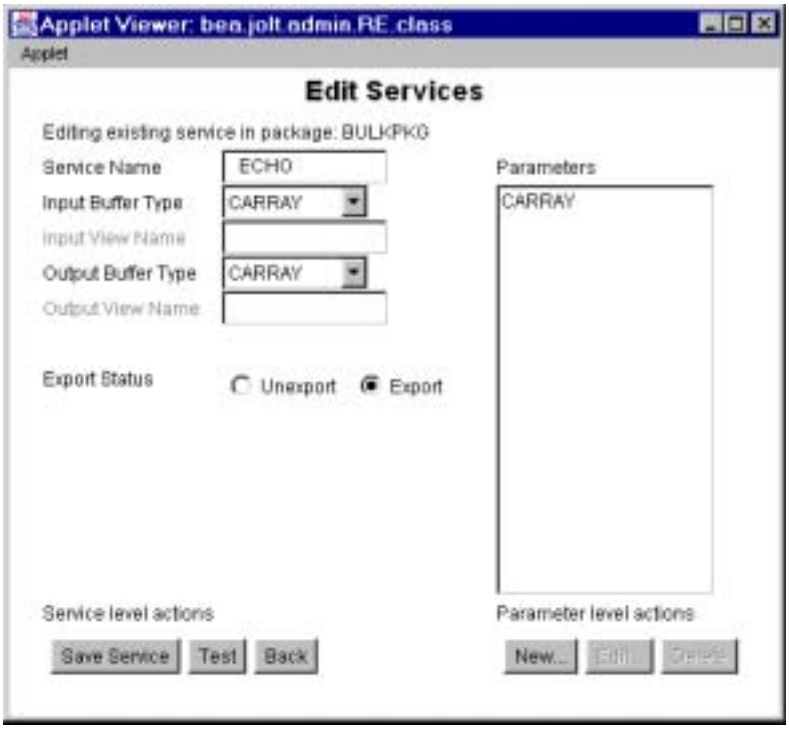


次の手順に従い、リポジトリ・エディタで ECHO サービスを追加します。

1. リポジトリ・エディタで ECHO サービスを追加します。
2. 入出力用のバッファ型として CARRAY を定義します。
3. 入出力用のパラメータとして CARRAY を 1 つだけ定義します

注記　X_OCTET バッファ型を使用する場合は、[Input Buffer Type] フィールドと [Output Buffer Type] フィールドを [X_OCTET] に変更しなければなりません。

図 5-6 リポジトリ・エディタ :ECHO サービスを編集する



tryOnCARRAY.java クライアント・コード

次のリストに示すコードは、CARRAY バッファ型が設定されたサービスが Jolt でどのように動作するかを示しています。Jolt は CARRAY データ・ストリームの内部のデータを調べません。したがって、Jolt クライアントと CARRAY サービスのデータ形式を一致させるのはプログラマ側の役割になります。次の例では、セッション・オブジェクトが既にインスタンス化されていることを想定しています。

コード リスト 5-3　CARRAY **バッファ型の例**

```
/* Copyright 1996 BEA Systems, Inc. All Rights Reserved */

  /* コードからのこの抜粋部分は、バッファ型が CARRAY のサービスに
   * Jolt が対応する様子を示したものです。
   */

import java.io.*;
import bea.jolt.*;
class ...
{
    ...
    public void tryOnCARRAY()
    {
        byte data[];
        JoltRemoteService csvc;
        DataInputStream din;
        DataOutputStream dout;
        ByteArrayInputStream bin;
        ByteArrayOutputStream bout;
        /*
       * java.io.DataOutputStream を使用してデータをバイト配列に出力する。
        */
        bout = new ByteArrayOutputStream(512);
        dout = new DataOutputStream(bout);
        dout.writeInt(100);
        dout.writeFloat((float) 300.00);
        dout.writeUTF("Hello World");
        dout.writeShort((short) 88);
        /*
         * バイト配列を新規バイト配列 "data" にコピーする。
         * 次に Jolt リモート・サービス呼び出しを発行する。
         */
        data = bout.toByteArray();
        csvc = new JoltRemoteService("ECHO", session);
        csvc.setBytes("CARRAY", data, data.length);
        csvc.call(null);
        /*
         * JoltRemoteService オブジェクトから応答を獲得する。
         * java.io.DataInputStream を使用して個々の値をそれぞれ
         * バイト配列から引き出す。
         */
        data = csvc.getBytesDef("CARRAY", null);
        if (data != null)
        {
            bin = new ByteArrayInputStream(data);
```

```
            din = new DataInputStream(bin);
            System.out.println(din.readInt());
            System.out.println(din.readFloat());
            System.out.println(din.readUTF());
            System.out.println(din.readShort());
        }
    }
}
```

FML バッファ型を使う

FML (フィールド操作言語) は、型付きバッファとして使用できる柔軟性のあるデータ構造です。FML は、タグ付きの値を格納するデータ構造です。このタグ付きの値には型が付いており、長さを変更することができ、複数のオカレンスを持つ場合があります。FML では、型付きバッファを抽象的なデータ型として扱います。

FML 操作を用いると、データ構造やデータの格納方法を知らなくても、データに対してアクセスしたり、更新を行うことができます。アプリケーション・プログラムでは、単に識別子を使ってフィールド化されたバッファのフィールドにアクセスしたり更新するだけで済みます。操作の実行時には、FML のランタイム・システムによってフィールドの場所と操作を行うデータ型が決定されます。

FML は、クライアントとサーバがそれぞれ別のコード (Java 言語と C 言語など) で記述されている場合、プラットフォームで扱われるデータ型の仕様が異なる場合、またはクライアント / サーバ間のインターフェイスが頻繁に変わる場合の Jolt クライアントでの使用に特に適しています。

次の `tryOnFml` は、FML バッファ型の使用方法を示しています。この例では、FML バッファを使用する Jolt クライアントがサーバにデータを渡す様子を示しています。サーバはバッファを取得し、データを格納するための新しい FML バッファを作成し、バッファを Jolt クライアントに返します。以下は、コード例の説明です。

- 5-28 ページの「tryOnFml.java のコード例」は、PASSFML サービスを持つ Jolt クライアントの例です。

- 5-28 ページの「tryOnFml.f16 のフィールド定義」は、PASSFML サービスで使用される BEA Tuxedo の FML フィールドの定義テーブルの例です。
- 5-32 ページの「tryOnFml.c のコード例」は、Jolt クライアントから送信されたデータを処理するサーバ・サイドの C コードを含むサーバ・コードの一部です。

tryOnFml.java クライアント・コード

次に示す `tryOnFml.java` の Java コードの抜粋は、バッファ型が FML であるサービスを Jolt から利用する方法を示しています。この例では、セッション・オブジェクトが既にインスタンス化されていることを想定しています。

コード リスト 5-4 tryOnFml.java **のコード例**

```
/* Copyright 1997 BEA Systems, Inc. All Rights Reserved */

import bea.jolt.*;
class ...
{
    ...
    public void tryOnFml ()
    {
        JoltRemoteService passFml;
        String outputString;
        int outputInt;
        float outputFloat;
                ...
        passFml = new JoltRemoteService("PASSFML",session);
        passFml.setString("INPUTSTRING", "John");
        passFml.setInt("INPUTINT", 67);
        passFml.setFloat("INPUTFLOAT", (float)12.0);
        passFml.call(null);
        outputString = passFml.getStringDef("OUTPUTSTRING", null);
        outputInt = passFml.getIntDef("OUTPUTINT", -1);
        outputFloat = passFml.getFloatDef("OUTPUTFLOAT", (float)-1.0);
        System.out.print("String =" + outputString);
        System.out.print(" Int =" + outputInt);
        System.out.println(" Float =" + outputFloat);
        }
}
```

FML フィールド定義

次の「tryOnFml.f16 のフィールド定義」は、「tryOnFml.java のコード例」の FML フィールド定義を示しています。

コード リスト 5-5 tryOnFml.f16 **のフィールド定義**

```
#
# FML field definition table
#
*base    4100
INPUTSTRING   1    string
INPUTINT      2    long
```

```
INPUTFLOAT    3    float
OUTPUTSTRING  4    string
OUTPUTINT     5    long
OUTPUTFLOAT   6    float
```

リポジトリ・エディタで PASSFML を定義する

BULKPKG パッケージには、PASSFML サービスが含まれており、`tryOnFml.java` および `tryOnFml.c` のコードで使用されます。`tryOnFml.java` を使用する前に、Jolt リポジトリ・エディタを使用して PASSFML サービスを変更する必要があります。

注記　サービスの定義に関しては、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

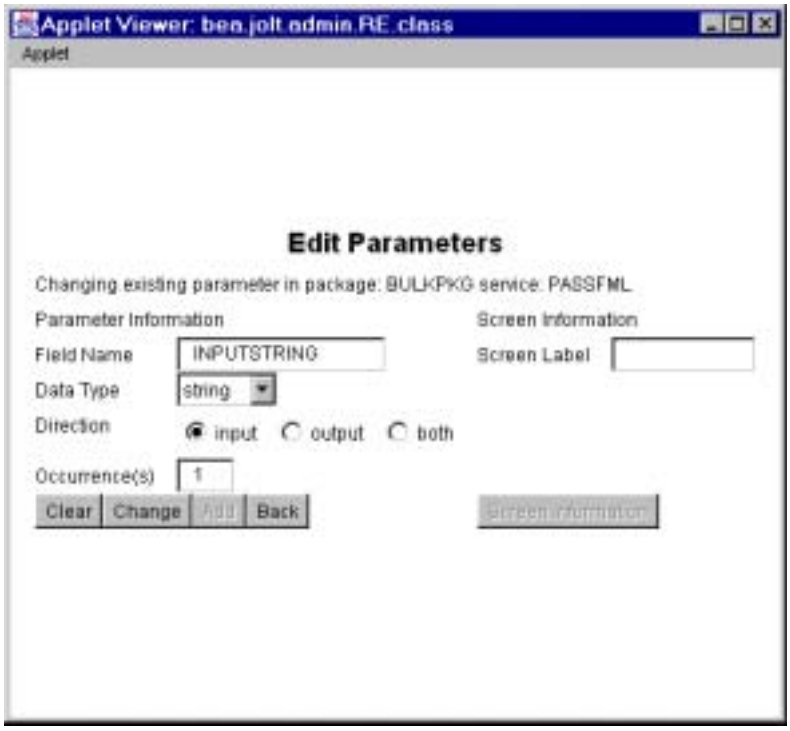
1. Jolt リポジトリ・エディタの [Edit Services] ウィンドウで PASSFML サービスを定義します。入力バッファ型として FML を定義し、出力バッファ型として FML を定義します。

   5-30 ページの「[Repository Editor] ウィンドウ :PASSFML サービスを編集する」は、PASSFML サービスおよび入力バッファ型 FML と出力バッファ型 FML を示しています。

図 5-7[Repository Editor] ウィンドウ :PASSFML サービスを編集する

2. PASSFML サービスの入力バッファ型と出力バッファ型として FML を定義します。
3. Edit] をクリックすると、次の図のような [Edit Parameters] ウィンドウが表示されます。

図 5-8PASSFML パラメータを定義する



4. PASSFML サービスのパラメータを定義します。
5. PASSFML サービスの各パラメータについて、手順 2 から 4 を繰り返します。

tryOnFml.c サーバ・コード

次のリストは、FML バッファ型を使用するためのサーバ・サイドのコードを示しています。PASSFML サービスは、入力 FML バッファを読み込み、FML バッファを出力します。

コード リスト 5-6　tryOnFml.c **のコード例**

```
/*

 * tryOnFml.c
 *
 * Copyright (c) 1997 BEA Systems, Inc. All rights reserved
 *
 * PASSFML BEA Tuxedo サーバが含まれています。
 *
 */
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include <fml.h>
#include <fml32.h>
#include <Usysflds.h>
#include <atmi.h>
#include <userlog.h>
#include "tryOnFml.f16.h"
/*
 * PASSFML サービスは、入力 fml バッファを読み込み、fml バッファを出力します。
 */
void
PASSFML( TPSVCINFO *rqst )
{
FLDLENlen;
FBFR*svcinfo = (FBFR *) rqst->data;
charinputString[256];
longinputInt;
floatinputFloat;
FBFR*fml_ptr;
```

```
intrt;
if (Fget(svcinfo, INPUTSTRING, 0, inputString, &len) < 0) {
(void)userlog("Fget of INPUTSTRING failed %s",
Fstrerror(Ferror));
tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
}
if (Fget(svcinfo, INPUTINT, 0, (char *) &inputInt, &len) < 0) {
(void)userlog("Fget of INPUTINT failed %s",Fstrerror(Ferror));
tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
}
if (Fget(svcinfo, INPUTFLOAT, 0, (char *) &inputFloat, &len) < 0) {
(void)userlog("Fget of INPUTFLOAT failed %s",
Fstrerror(Ferror));
tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
}
/* We could just pass the FML buffer back as is, put lets*/
/* 別の FML バッファに格納して返すことも可能 */
if ((fml_ptr = (FBFR *)tpalloc("FML",NULL,rqst->len))==(FBFR *)NULL) {
(void)userlog("tpalloc failed in PASSFML %s",
tpstrerror(tperrno));
tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
}
if(Fadd(fml_ptr, OUTPUTSTRING, inputString, (FLDLEN)0) == -1) {
userlog("Fadd failed with error: %s", Fstrerror(Ferror));
tpfree((char *)fml_ptr);
tpreturn(TPFAIL, 0, NULL, 0L, 0);
        }
if(Fadd(fml_ptr, OUTPUTINT, (char *)&inputInt, (FLDLEN)0) == -1) {
userlog("Fadd failed with error: %s", Fstrerror(Ferror));
tpfree((char *)fml_ptr);
tpreturn(TPFAIL, 0, NULL, 0L, 0);
        }
if(Fadd(fml_ptr, OUTPUTFLOAT, (char *)&inputFloat, (FLDLEN)0) == -1) {
userlog("Fadd failed with error: %d\n", Fstrerror(Ferror));
tpfree((char *)fml_ptr);
tpreturn(TPFAIL, 0, NULL, 0L, 0);
        }
tpreturn(TPSUCCESS, 0, (char *)fml_ptr, 0L, 0);
}
```

VIEW バッファ型を使う

VIEW は BEA Tuxedo 組み込みのバッファ型です。VIEW は、BEA Tuxedo システムで C 構造体および COBOL レコードを使用するために提供されています。この型付きバッファにより、BEA Tuxedo のランタイム・システムは、実行時に読み込まれる view 記述に基づいて C の構造体や COBOL のレコードのフォーマットを認識します。

VIEW を割り当てると、アプリケーションはバッファ型として VIEW を指定し、view の名前 (view 記述ファイル内の名前) を示すサブタイプを指定します。また、パラメータ名と view 内のフィールド名は一致していなければなりません。BEA Tuxedo のランタイム・システムは構造体のサイズに応じて必要な空間を割り当てるため、アプリケーション側でバッファ長を指定する必要はありません。また、ランタイム・システムは、要求または応答の際に送信されるデータ量の計算や、異なるマシン間で送受信されるメッセージの符号化や復号化を自動的に行います。

次は、Jolt クライアントとサーバ・サイド・アプリケーションがある環境での VIEW バッファ型の使い方の例です。

- 5-37 ページの「simpview.java のコード例」は BEA Tuxedo に接続するためのコードが記述された Jolt クライアント・コードです。バッファ型として VIEW が指定されています。
- 5-38 ページの「simpview.v16 のフィールド定義」には、BEA Tuxedo の VIEW フィールド定義が記述されています。
- 5-39 ページの「simpview.c のコード例」には、Jolt クライアントからの入力情報を処理するサーバ・サイドの C コードが記述されています。

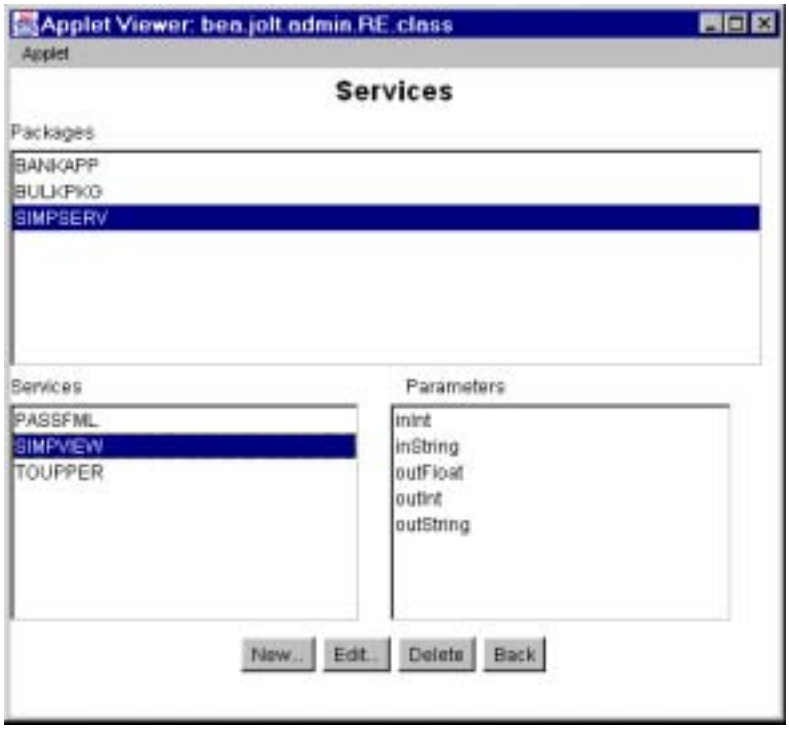
Jolt クライアントでは、VIEW バッファの文字列に含まれるヌル文字は行末文字として扱われ、ヌル文字の後の文字列は切り捨てられます。

リポジトリ・エディタで VIEW を定義する

`simpview.java` および `simpview.c` の例を実行する前に、Jolt リポジトリ・エディタを使用して VIEW サービスを定義する必要があります。

注記　サービスの定義に関しては、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

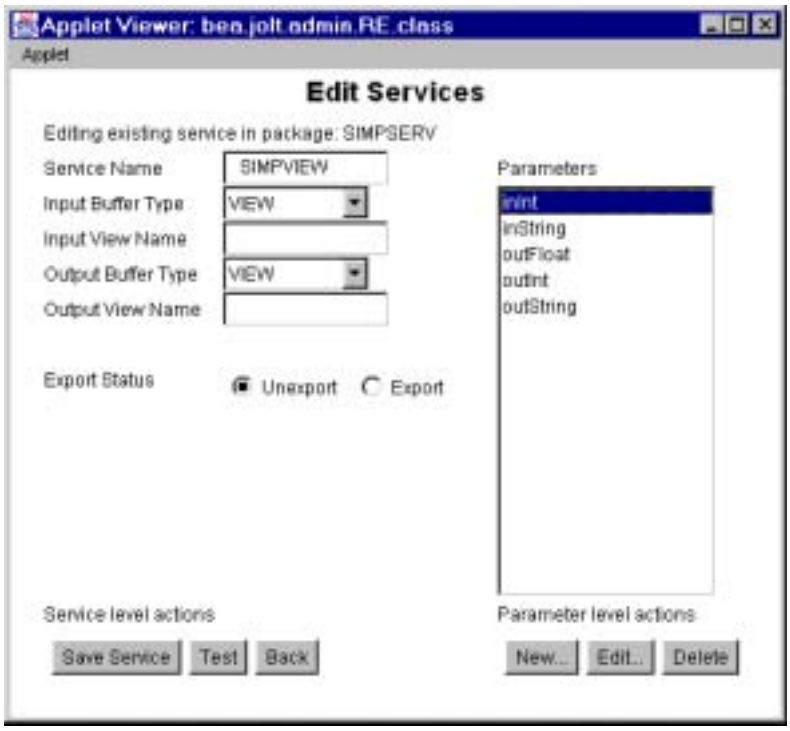
図 5-9 リポジトリ・エディタ :SIMPVIEW サービスを追加する



次の手順に従って、リポジトリ・エディタで VIEW サービスを追加します。

1. SIMPSERV パッケージに SIMPVIEW サービスを追加します。
2. SIMPVIEW サービスを定義します。入力バッファ型として VIEW を定義し、出力バッファ型として VIEW を定義します。

図 5-10 リポジトリ・エディタ : SIMPVIEW サービスを編集する



3. VIEW サービスのパラメータを定義します。この例では、パラメータとして `inInt`、`inString`、`outFloat`、`outInt`、および `outString` があります。

注記　バッファ型として X_COMMON または X_C_TYPE を使用する場合は、[Input Buffer Type] フィールドと [Output Buffer Type] フィールドに正しいバッファ型を指定する必要があります。また、[Input View Name] フィールドと [Output View Name] フィールドにも、対応する名前を指定してください。

simpview.java クライアント・コード

5-37 ページの「simpview.java のコード例」は、VIEW バッファ型が指定されたサービスが Jolt でどのように動作するかを示しています。クライアント・コードは、FML サービスにアクセスするときに使用されるコードと同じです。

注記　次のリスト内のコードでは、例外はキャッチされません。Jolt の例外は、すべて `java.lang.RunTimeException` から派生しています。そのため、アプリケーション側でキャッチされない例外は Java 仮想マシンによってキャッチされます (よりよいアプリケーションでは、これらの例外をキャッチして適切な処理を行うべきです)。

次のリストに示す例を実行する前に、Jolt リポジトリ・エディタを使用して SIMPAPP パッケージに VIEW サービスを追加し、BEA Tuxedo アプリケーション `simpview.c` を記述する必要があります。このサービスは、クライアントの VIEW バッファからデータを取り出し、新しいバッファを作成して、新しい VIEW バッファとしてクライアントに返します。次の例では、セッション・オブジェクトが既にインスタンス化されていることを想定しています。

コード リスト 5-7　simpview.java **のコード例**

```
/* Copyright 1997 BEA Systems, Inc. All Rights Reserved */
/*
 * コードからのこの抜粋部分は、バッファ型が XML のサービスに
 * Jolt が対応する様子を示したものです。
 */
import bea.jolt.*;
class ...
{
...
public void simpview ()
{
JoltRemoteService ViewSvc;
String outString;
int outInt;
float outFloat;
// BEA Tuxedo サービスに対応する Jolt サービス "SIMPVIEW" を作成する。
ViewSvc = new JoltRemoteService("SIMPVIEW",session);
// Set the input parametes required for SIMPVIEW
```

```
ViewSvc.setString("inString", "John");
ViewSvc.setInt("inInt", 10);
ViewSvc.setFloat("inFloat", (float)10.0);
// サービスを呼び出す。トランザクションは不要なため、
// "null" パラメータを渡す。
ViewSvc.call(null);
// 結果を処理し、
outString = ViewSvc.getStringDef("outString", null);
outInt = ViewSvc.getIntDef("outInt", -1);
outFloat = ViewSvc.getFloatDef("outFloat", (float)-1.0);
// 表示する。
System.out.print("outString=" + outString + ",");
System.out.print("outInt=" + outInt + ",");
System.out.println("outFloat=" + outFloat);
}
}
```

VIEW フィールド定義

「simpview.v16 のフィールド定義」は、前のコード例 `simpview.java` の BEA Tuxedo VIEW フィールド定義を示しています。

コード リスト 5-8　simpview.v16 **のフィールド定義**

```
#
# SIMPVIEW の VIEW。これは入出力用に使用されます。
# サービスでは入力と出力で異なる VIEW を使用することもできます。
# 先頭から 3 つは入力用、次の 3 つは出力用のパラメータです。
#
VIEW SimpView
$
#type  cname         fbname count  flag   size   null
string inString      -      1      -      32     -
long   inInt         -      1      -      -      -
float  inFloat       -      1      -      -      -
string outString     -      1      -      32     -
long   outInt        -      1      -      -      -
float  outFloat      -      1      -      -      -
END
```

simpview.c サーバ・コード

次のコード例で使用される入力バッファ型と出力バッファ型は VIEW です。このコードでは、VIEW バッファ型の入力データを受け付け、同じ VIEW バッファ型のデータを出力します。

コード リスト 5-9　simpview.c **のコード例**

```
/*
 * SIMPVIEW.c
 *
 * Copyright (c) 1997 BEA Systems, Inc.  All rights reserved
 *
 * SIMPVIEW BEA Tuxedo サーバが含まれています。
 *
 */
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include <fml.h>
#include <fml32.h>
#include <Usysflds.h>
#include <atmi.h>
#include <userlog.h>
#include "simpview.h"
/*
 * 以下は simpview.h の内容です。
 *
 *struct SimpView {
 *
 *charinString[32];
 *longinInt;
 *floatinFloat;
 *charoutString[32];
 *longoutInt;
 *floatoutFloat;
 *};
 */
```

```
/*
 * サービスは、入力 VIEW バッファを読み込み、VIEW バッファを出力する。
 */
void
SIMPVIEW( TPSVCINFO *rqst )
{
/*
 * TPSVCINFO 構造から構造 (VIEWSVC) を取得する。
 */
struct SimpView*svcinfo = (struct SimpView *) rqst->data;
/*
 * 入力パラメータを UserLog に出力する。ここではエラーは
 * チェックされません。
 * 通常、SERVER は入力の正当性をチェックし、
 * 不正な入力が検出されると TPFAIL を返します。
 */
(void)userlog("SIMPVIEW: InString=%s,InInt=%d,InFloat=%f",
svcinfo->inString, svcinfo->inInt, svcinfo->inFloat);
/*
 * 出力フィールドを設定し、呼び出し側に返す。
 */

strcpy (svcinfo->outString, "Return from SIMPVIEW");
svcinfo->outInt = 100;
svcinfo->outFloat = (float) 100.00;
/*
 * エラーが発生すると、TPFAIL
 * tpreturn(TPFAIL, ErrorCode, (char *)svcinfo, sizeof (*svcinfo), 0); が返される。
 */
tpreturn(TPSUCCESS, 0, (char *)svcinfo, sizeof (*svcinfo), 0);
}
```

XML バッファ型を使う

XML 型バッファを使用すると、BEA Tuxedo アプリケーションで XML を使用して、アプリケーション内やアプリケーション間でデータを交換できるようになります。BEA Tuxedo アプリケーションでは、XML 型バッファの送受信や、それらのバッファを適切なサーバにルーティングできます。解析など、XML 文書のすべての処理ロジックはアプリケーション側にあります。

形式の整った XML 文書は、次の要素から構成されます。

- 表題、タグなどを含む、符号化された文字の並びで構成されるテキスト

- 文書の論理構造の記述と、その構造に関する情報

XML バッファ型を使用するには、まずバッファ型で使用する Tuxedo サービスを定義します。次に、そのバッファ型を使用するコードを記述します。次の 2 つの節では、これらの手順について説明します。

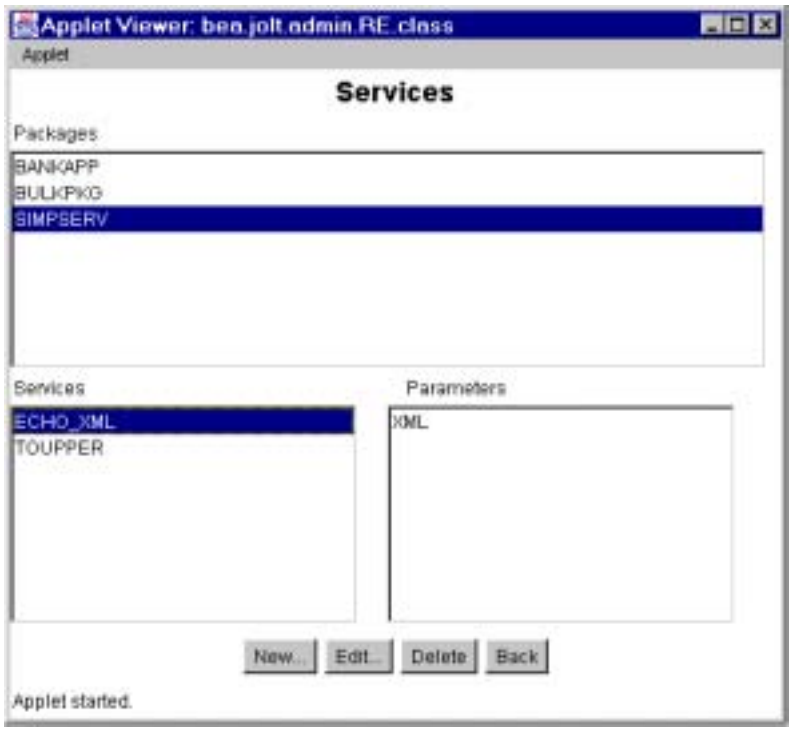
注記　CARRAY 同様に、XML バッファ型は STRING ではなくバイト配列として扱われます。Jolt クライアントと BEA Tuxedo サービス間でのデータ変換は行われません。

リポジトリ・エディタで Tuxedo サービスを定義する

例を実行する前に、BEA Tuxedo の XML サービスを作成し、起動する必要があります。XML サービスは、バッファを取得し、Jolt クライアントに返します。Jolt リポジトリ・エディタを使用して XML サービスを定義します。

注記　サービスを定義したり、新しいパラメータを追加する方法については、4-1 ページの「Jolt リポジトリ・エディタを使う」を参照してください。

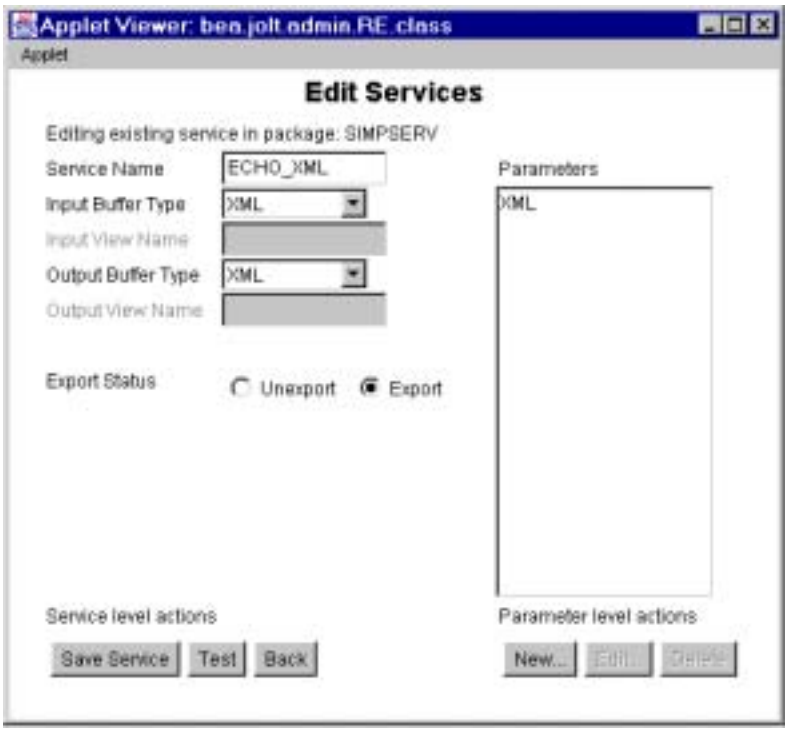
図 5-11 リポジトリ・エディタ :XML サービスを追加する



次の手順に従い、リポジトリ・エディタで XML サービスを追加します。

1. リポジトリ・エディタで ECHO_XML サービスを追加します。
2. ECHO_XML サービスの入出力用のバッファ型として XML を定義します。
3. 入出力用のパラメータとして XML を 1 つだけ指定して ECHO_XML サービスを定義します

図 5-12 リポジトリ・エディタ :XML サービスを編集する



simpxml.java クライアント・コード

次のリストに示すコードは、XML バッファ型が設定されたサービスが Jolt でどのように動作するかを示しています。Jolt は XML データ・ストリームの内部のデータを調べません。したがって、Jolt クライアントと XML サービスのデータ形式を一致させるのはプログラマ側の役割になります。次の例では、セッション・オブジェクトが既にインスタンス化されていることを想定しています。

コード リスト 5-10　XML **バッファ型の例**

```
/* Copyright 2001 BEA Systems, Inc. All Rights Reserved */
/*
 * コードからのこの抜粋部分は、バッファ型が XML のサービスに
 * Jolt が対応する様子を示したものです。
 */

import java.io.*;
import java.lang.*;
import bea.jolt.*;

public class xmldoc {

    public static void main (String[] args) {
        JoltSessionAttributes  sattr;
        JoltSession            session;
        JoltRemoteService      echo_xml;

String inString = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><ORDER><HEADER
DATE=\"05/13/1999\" ORDERNO=\"22345\"/><COMPANY>ACME</COMPANY><LINE><ITEM
MODEL=\"Pabc\" QUANTITY=\"5\">LAPTOP</ITEM></LINE><LINE><ITEM MODEL=\"P500\"
QUANTITY=\"15\">LAPTOP</ITEM></LINE></ORDER>";

        byte data[];
        DataInputStream din;
        DataOutputStream dout;
        ByteArrayInputStream bin;
        ByteArrayOutputStream bout;

        byte odata[];
        String outString = null;
        String appAddress = null;

        //...Create Jolt Session

        try {
            /*
             * java.io.DataOutputStream を使用して
             * データをバイト配列に出力する。
             */
            bout = new ByteArrayOutputStream(inString.length());
            dout = new DataOutputStream(bout);
            dout.writeBytes(inString);
```

```
            /*
                     * バイト配列を新規バイト配列 "data" にコピーする。
                     * 次に Jolt リモート・サービス呼び出しを発行する。
                 */
            data = bout.toByteArray();
        } catch (Exception e) {
            System.out.println("toByteArray error");
            return;
        }

        try {
            echo_xml = new JoltRemoteService("ECHO_XML", session);
            System.out.println("JoltRemoteService Created");
            echo_xml.setBytes("XML", data, data.length);
        } catch (Exception e) {
            System.out.println("RemoteService call error" + e);
            return;
        }

        echo_xml.call(null);
        System.out.println("Service Call Returned");
        odata = echo_xml.getBytesDef("XML", null);

        try {
            System.out.println("Return String is:" + new String(odata));
        } catch (Exception e) {
            System.err.println("getByteDef Error");
        }
    }
}

// end of class
```

マルチスレッド・アプリケーション

Java ベースのクラス・セットである Jolt は、マルチスレッド・アプリケーションをサポートします。ただし、Java 言語のさまざまなインプリメンテーションは、言語や環境ごとの機能によって異なります。Jolt のプログラマは、次の事項を知っておく必要があります。

- Jolt クラス・ライブラリを使ってアプリケーションやアプレットを作成するときのプリエンプティブなスレッドとノンプリエンプティブなスレッドの使い方
- 非同期的な処理を行うためのスレッドの使い方 (BEA Tuxedo の `tpacall()` に類似)

「スレッドの状態の種類」では、さまざまな Java のインプリメンテーションでスレッドを使用する際に起こる問題を説明します。また、Jolt プログラムでスレッドを使う例も示します。

注記　多くの場合、Java のインプリメンテーションでは、ノンプリエンティブではなく、プリエンプティブなスレッドが処理されます。プリエンティブなスレッドとノンプリエンティブなスレッドでは、パフォーマンスやプログラミングの条件が大きく異なります。

スレッドの状態の種類

Java 仮想マシンで同時に実行される個々のタスクをスレッドと呼びます。スレッドの状態には、主に RUNNING、RUNNABLE、および BLOCKED があります。

- RUNNING は、現在実行中のスレッドであることを示します。
- RUNNABLE は、現在のスレッドが CPU を専有しなくなると実行できるスレッドであることを示します。RUNNABLE 状態のスレッドは複数ある場合があります。ただし、RUNNING な状態になれるスレッドは 1 つだけです。スレッドを実行するとは、スレッドの状態を RUNNABLE か

ら RUNNING に変更し、そのスレッドが Java 仮想マシン (VM) を制御できるようになることを意味します。

- BLOCKED は、イベントまたはリソースが利用可能になるまで待機しているスレッドであることを示します。

注記　Java 仮想マシン (VM) は、同じ優先度が設定されたスレッドをラウンド・ロビン方式で実行します。

プリエンプティブなスレッド

プリエンティブなスレッドとノンプリエンティブなスレッドでは、Java 仮想マシン (VM) の制御権を放棄するよう実行中のスレッドに対して通知する方法が異なります。プリエンプティブなスレッド環境では通常、ハードウェアのタイマーを周期的にオフになるように設定します。タイマーがオフになると現在のスレッドの状態が RUNNING から RUNNABLE になり、代わりに別のスレッドが実行されます。

ノンプリエンプティブなスレッド

ノンプリエンプティブなスレッド環境のスレッドは、自発的に CPU の制御権を放棄し、RUNNABLE な状態に移行します。Java 言語クラス内の多くのメソッドには、制御権を自発的に放棄するコードが含まれています。これらのコードは通常、長い処理時間のかかる操作に関連付けられています。たとえば、ネットワークからデータを読み込むと、スレッドは通常パケットの到着を待ちます。イベントやリソースが利用可能になるまで待機するスレッドは BLOCKED の状態です。イベントが発生するか、またはリソースが利用できるようになると、そのスレッドは RUNNABLE になります。

Jolt をノンプリエンプティブなスレッドで使う

ノンプリエンプティブなスレッド環境の仮想マシン (例 : Sun Solaris) で Jolt ベースの Java プログラムを実行する場合、次のいずれかが行われる必要があります。

- スレッドをブロックするメソッドをときどき呼び出す

- `Thread.yield()` メソッドを使用して、明示的に CPU の制御権を放棄する

典型的な方法として、実行するコードが長い部分または時間がかかりそうなループのすべてで次の呼び出しを行います。

```
Thread.currentThread.yield();
```

このメッセージを送信しないと、Jolt ライブラリで使用されるスレッドはスケジューリングされず、Jolt 操作に不具合が生じます。

ノンプリエンプティブなスレッドを使用する仮想マシンとして知られている唯一のマシンは、Sun プラットフォーム用の Java Developer's Kit (JDK) です。アプレットを JDK 1.3 で実行するには、yield メッセージを必ず送信してください。既に述べたように、メソッドの中には yield を含むものがあります。例外は `System.in.read` メソッドです。このメッセージはスレッドの切り替えを行いません。これらのメッセージを使用する代わりに、yield を明示的に使用することをお勧めします。

非同期的な処理を行うためにスレッドを使う

スレッドを使用して、BEA Tuxedo の `tpacall()` に似た非同期的な処理を Jolt で行うことができます。この機能があれば、非同期的にサービスを要求する機能は必要ありません。この機能が使えるのは Jolt がスレッド・セーフであるためです。たとえば、Jolt クライアント・アプリケーションは、BEA Tuxedo サービスにリクエストを送信するスレッドを開始した後で直ちに BEA Tuxedo サービスに別のリクエストを送信する別のスレッドを開始することができます。したがって、Jolt による `tpacall()` の呼び出しが同期的であっても、2 つのスレッドが同時に実行されているため、アプリケーションは非同期的です。

Jolt でスレッドを使用する

Jolt クライアント側のプログラムまたはアプレットは完全にスレッド・セーフです。Jolt がサポートするマルチスレッド・アプリケーションでは、クライアント側に次の特徴があります。

- 1 つのクライアントに対して複数のセッションがある
- 1 つのセッションがマルチスレッドである
- クライアント・アプリケーションは、非同期呼び出しの代わりにスレッドを管理する
- 同期呼び出しを実行する

次のコード例は、Jolt アプリケーションで 2 つのスレッドを使用する方法を示したものです。

コード リスト 5-11　Jolt **でマルチスレッドを使用する** (ThreadBank.java)

```
/* Copyright 1996 BEA Systems, Inc.  All Rights Reserved */
import bea.jolt.*;
public class ThreadBank
{
   public static void main (String [] args)
   {
      JoltSession session;
      try
      {
         JoltSessionAttributes dattr;
         String userName = null;
         String userPasswd = null;
         String appPasswd = null;
         String userRole = null;

         // 必要な属性を設定
         dattr = new JoltSessionAttributes();
         dattr.setString(dattr.APPADDRESS,"//bluefish:8501");

         // ドメインをインスタンス化する
         // 認証レベルを調べる
         switch (dattr.checkAuthenticationLevel())
         {

         case JoltSessionAttributes.NOAUTH:
              System.out.println("NOAUTH\n");
              break;
         case JoltSessionAttributes.APPASSWORD:
              appPasswd = "myAppPasswd";
              break;
         case JoltSessionAttributes.USRPASSWORD:
              userName = "myName";
```

```
                userPasswd = “mySecret”;
                appPasswd = “myAppPasswd”;
                break;
        }
        dattr.setInt(dattr.IDLETIMEOUT, 60);
        session = new JoltSession (dattr, userName, userRole,
                                   userPasswd, appPasswd);
        T1 t1 = new T1 (session);
        T2 t2 = new T2 (session);

        t1.start();
        t2.start();

        Thread.currentThread().yield();
        try
        {
           while (t1.isAlive() && t2.isAlive())
           {
              Thread.currentThread().sleep(1000);

           }
        }
        catch (InterruptedException e)
        {
           System.err.println(e);
           if (t2.isAlive())
           {
              System.out.println(“job 2 is still alive”);
              try
              {
                 Thread.currentThread().sleep(1000);
              }
              catch (InterruptedException e1)
              {
                 System.err.println(e1);
              }
           }
           else if (t1.isAlive())
           {   System.out.println(“job1 is still alive”);
              try
              {
                 Thread.currentThread().sleep(1000);
              }
              catch (InterruptedException e1)
              {
                 System.err.println(e1);
              }
           }
        }
```

```
            session.endSession();
         }
         catch (SessionException e)
         {
            System.err.println(e);
         }
         finally
         {
            System.out.println("normal ThreadBank term");
         }
      }
   }


   class T1 extends Thread
   {
      JoltSession j_session;
      JoltRemoteService j_withdrawal;

      public T1 (JoltSession session)
      {
         j_session=session;
         j_withdrawal= new JoltRemoteService("WITHDRAWAL",j_session);
      }
      public void run()
      {
         j_withdrawal.addInt("ACCOUNT_ID",10001);
         j_withdrawal.addString("SAMOUNT","100.00");
         try
         {
            System.out.println("Initiating Withdrawal from account
   10001");
            j_withdrawal.call(null);
            String W = j_withdrawal.getStringDef("SBALANCE","-1.0");
            System.out.println("-->Withdrawal Balance: " + W);
         }
         catch (ApplicationException e)
         {
            e.printStackTrace();
            System.err.println(e);
         }
      }
   }

   class T2 extends Thread
   {
      JoltSession j_session;
      JoltRemoteService j_deposit;
```

```
   public T2 (JoltSession session)
   {
      j_session=session;
      j_deposit= new JoltRemoteService("DEPOSIT",j_session);
   }
   public void run()
   {
      j_deposit.addInt("ACCOUNT_ID",10000);
      j_deposit.addString("SAMOUNT","100.00");
      try
      {
       System.out.println("Initiating Deposit from account 10000");
         j_deposit.call(null);
         String D = j_deposit.getStringDef("SBALANCE","-1.0");
         System.out.println("-->Deposit Balance: " + D);
      }
      catch (ApplicationException e)
      {
         e.printStackTrace();
         System.err.println(e);
      }
   }
}
```

イベント・サブスクリプションおよびイベント通知

Jolt を使用してクライアント・アプリケーションを開発するプログラマは、BEA Tuxedo サービスまたはほかの BEA Tuxedo クライアントからイベント通知を受信することができます。Jolt クラス・ライブラリには、イベントに基づく通信を処理するため、次の BEA Tuxedo 通知をサポートするクラスが用意されています。

- 任意通知型イベント通知－これは、BEA Tuxedo のクライアントかサービスが、`tpbroadcast()` を呼び出してブロードキャストを発行するか、または ATMI 呼び出しの `tpnotify()` を使用して Jolt クライアント宛てに直接メッセージを発行した結果、Jolt クライアントが受信する通知です。
- ブローカ経由のイベント通知－ Jolt クライアントが BEA Tuxedo イベント・ブローカ経由で受信する通知です。これらの通知は、Jolt クライアントがイベントをサブスクライブしており、かつ任意の BEA Tuxedo クライアントかサーバがシステム通知イベントを発行するか、または `tppost()` 呼び出しを発行する場合にのみ受信されます。

イベント・サブスクリプション用のクラス

Jolt クラス・ライブラリには、Jolt クライアント・アプリケーション用の非同期通知メカニズムをインプリメントするための 4 つのクラスが用意されています。

- JoltSession － JoltSession クラスには、通知およびメッセージを受信する `onReply()` メソッドが用意されています。
- JoltReply － JoltReply クラスを使うと、クライアント・アプリケーションからイベントや通知と共に受信したメッセージにアクセスすることができます。

- JoltMessage − JoltMessage クラスには、通知またはイベントに関する情報を取得するための `get` メソッドが用意されています。
- JoltUserEvent − JoltUserEvent クラスは、任意通知型メッセージとイベント通知の両方のサブスクリプションをサポートします。

これらのクラスについては、『BEA Jolt API リファレンス』を参照してください。

通知イベント・ハンドラ

Jolt クライアント・アプリケーションでは、任意通知型メッセージとブローカ経由のイベント通知の両方に対してイベント・ハンドラ・ルーチン (通知の受信時に呼び出される) が必要になります。Jolt では、1 セッションでサポートされるハンドラは 1 つです。BEA Tuxedo では、通知の生成元であるイベントを特定することはできません。したがって、特定のイベントが発生したときに、そのイベント特有のハンドラを呼び出すことはできません。

クライアント・アプリケーション側では、セッションごとに 1 つのハンドラを用意 (`onReply()` メソッドをオーバーライド) し、そのセッションでクライアントが受信するすべての通知に対して呼び出されるようにしなければなりません。単一ハンドラによるコールバック関数は、任意通知型とイベント通知型の両方で使用されます。ハンドラ呼び出しの原因となったイベントを特定し、適切な措置を取ることは (ユーザ定義の) ハンドラ・ルーチンの役割です。ユーザがセッション・ハンドラをオーバーライドしないと、通知メッセージはデフォルトのハンドラによって暗黙のうちに廃棄されます。

Jolt クライアントは、JoltSession クラスのサブクラスを作成し、`onReply()` メソッドをユーザ定義された `onReply()` メソッドで上書きすることによって、コールバック関数を提供します。

BEA Tuxedo/ATMI クライアントでは、ハンドラ・コールバック関数内では、ATMI 呼び出しのサブセットしか使うことができません。この制約は、Jolt クライアントには適用されません。別々のスレッドを使用して、通知を監視し、イベント・ハンドラ・メソッドを実行します。Jolt がサポートするすべての機能をハンドラ内から実行することが可能です。Jolt クライアント・プログラムに適用される通常の規則 (1 セッションにつき 1 トランザクションなど) は、すべてハンドラにも適用されます。

ハンドラ・メソッドの呼び出しは、独立したスレッド内で行われます。別々のスレッドが `onReply()` メソッドを同時に実行しないようにするため、アプリケーション開発者は必ずそのメソッドに synchronized キーワードを付けるか、またはそのメソッドがスレッド・セーフになるように記述します。

Jolt では、ハンドラのルーティングを有効にする暗黙的なモデルが使用されます。クライアントがイベントをサブスクライブすると、Jolt はそのクライアントのハンドラを内部で有効にしますが、その結果、任意通知型メッセージの受信も有効になります。Jolt クライアントがイベント通知をサブスクライブする場合は、同時に必ず任意通知型メッセージも受信します。さらに、これらの 2 種類の通知に対して、単一の `onReply()` メソッドが呼び出されます。

コネクション・モード

Jolt は、コネクション・モード (RETAINED) またはコネクションレス・モード (RECONNECT) で稼動しているクライアントの通知受信をサポートします。コネクション・モード (RETAINED) のクライアントは、すべての通知を受信します。コネクションレス・モード (RECONNECT) モードで動作する Jolt クライアントは、Jolt セッション・ハンドラ (JSH) へのネットワーク接続がアクティブな間は通知を受信します。ネットワーク接続が切断されている場合、JSH はクライアント宛ての通知のログを記録し、その通知を廃棄します。コネクションレス・モード (RECONNECT) で稼動している Jolt クライアントは、ネットワーク接続がアクティブでない間は任意通知型メッセージまたはイベント通知を受信しません。この間に受信されたメッセージは、JSH によって記録され、廃棄されます。

コネクション・モードでの通知処理には、BEA Tuxedo 環境での Jolt クライアントへの確認付きメッセージ通知の処理も含まれます。JSH がクライアントへの確認付きメッセージ通知を受信し、そのクライアントに対してアクティブなネットワーク接続が確立されていない場合、JSH はエラーのログを記録し、通知メッセージに対して不成功を示す確認を返します。

通知データ・バッファ

クライアントが通知を受信する場合、通知にはデータ・バッファが付随しています。データ・バッファはどの BEA Tuxedo データ・バッファ型でもかまいません。Jolt クライアント (ハンドラ) はこれらのバッファを `JoltMessage` オブジェクトとして受信し、適切な `JoltMessage` クラスの `get*()` メソッドを使用してこのオブジェクトからデータを取り出します。

通知に使用するバッファの定義を Jolt リポジトリに格納する必要はありません。ただし、Jolt クライアント・アプリケーションのプログラマは、フィールド名を知っておく必要があります。

Jolt システムには、BEA Tuxedo の `tptypes()` と同様の機能はありません。FML および VIEW のバッファ型の場合は、適切なフィールド名を指定して `get*()` メソッドを使用してデータにアクセスします。たとえば、次のように入力します。

```
getIntDef ("ACCOUNT_ID", -1);
```

STRING および CARRAY のバッファ型の場合は、バッファ型と同じ名前でデータにアクセスします。たとえば、次のように入力します。

```
getStringDef ("STRING", null);
getBytesDef ("CARRAY", null);
```

STRING と CARRAY のバッファ型には、単一のデータ・エレメントしか含まれていません。このエレメント全体は、`get*()` メソッドで返されます。

BEA Tuxedo のイベント・サブスクリプション

BEA Tuxedo のブローカ経由のイベント通知により、BEA Tuxedo プログラムは、どのプログラムがイベントの発生通知を受信するかを気にしないで、イベントをポストすることができます。また、Jolt のイベント通知機能により、Jolt クライアント・アプリケーションは、BEA Tuxedo の `tpnotify()` または `tpbroadcast()` を呼び出してブロードキャストまたはポストされる BEA Tuxedo イベントをサブスクライブすることができます。

Jolt クライアントがサブスクライブできるのは、BEA Tuxedo の他のコンポーネントによって生成されたイベントまたはメッセージ通知 (BEA Tuxedo のサービスまたはクライアントなど) のみです。Jolt クライアントは、イベントまたはメッセージ通知を送信することはできません。

サポートされているサブスクリプションの種類

Jolt では、メッセージ通知型のサブスクリプションしかサポートされません。サブスクリプションが実行されると `onReply()` メソッドが呼び出されます。Jolt の API では、メッセージ通知の受信時にサービス・ルーチンをディスパッチしたり、アプリケーション・キューにメッセージを入れる機能はサポートされていません。

通知をサブスクライブする

Jolt クライアントが単一のイベント通知をサブスクライブする場合、クライアントは、任意通知型メッセージとイベント通知を両方とも受信します。イベントをサブスクライブすることにより、暗黙的に任意通知型メッセージも利用できるようになります。つまり、たとえばアプリケーションがイベント X に対して JoltUserEvent オブジェクトを作成した場合に `tpnotify()` または `tpbroadcast()` を呼び出すと、クライアントは自分宛てのメッセージを自動的に受信します。

注記　単一のイベント通知をサブスクライブするために任意通知型メッセージを用いることはお勧めしません。任意通知型メッセージを受信したい場合は、アプリケーション側で明示的に指定してください (JoltUserEvent クラスで説明)。次の節では、通知のアンサブスクライブについて説明します。

通知をアンサブスクライブする

イベント通知または任意通知型メッセージのいずれか、または両方のサブスクライブを停止するには、JoltUserEvent の unsubscribe メソッドを使う必要があります。Jolt では、unsubscribe メソッドを使用して任意通知型メッセージの受信を停止しても、すべてのサブスクリプション通知が停止されるわけ

ではありません。この点が BEA Tuxedo と異なります。BEA Tuxedo では、NULL ハンドラを指定して `tpsetunsol()` を呼び出すと、すべてのサブスクリプション通知が停止します。

イベントの通知をアンサブスクライブする場合、次の点に注意してください。

- クライアントが単一のイベントをサブスクライブする場合に通知をアンサブスクライブすると、イベント通知と任意通知型メッセージが両方共受信されなくなります。
- クライアントが複数のイベントをサブスクライブしている場合は、アンサブスクライブ対象のサブスクリプションだけが利用できなくなります。任意通知型メッセージは、引き続き受信されます。最後のサブスクリプションが停止されると、任意通知型メッセージの受信も停止されます。
- クライアントが任意通知型メッセージとイベント通知の両方をサブスクライブしている場合は、任意通知型メッセージだけをアンサブスクライブしても、どちらの通知も停止されません。さらに、このアンサブスクライブでは、例外がスローされません。ただし、Jolt API 側では任意通知型メッセージのアンサブスクライブを記憶しているため、残りのイベント通知をアンサブスクライブすると、イベント通知と任意通知型メッセージの両方が利用できなくなります。

クライアント・アプリケーションで任意通知型メッセージの受信を停止したい場合は、すべてのイベントをアンサブスクライブする必要があります。

Jolt API を使用して BEA Tuxedo からの通知を受信する

「非同期型通知」のコード例は、Jolt クラス・ライブラリを使用して通知を受信する方法を示しています。ここでは、JoltSession クラス、JoltReply クラス、JoltMessage 、および JoltUserEvent クラスが使用されています。

コード リスト 5-12 **非同期型通知**

```
class EventSession extends JoltSession
{
    public EventSession( JoltSessionAttributes attr, String user,
                         String role, String upass, String apass )
    {
           super(attr, user, role, upass, apass);
    }
    /**
    * Override the default unsolicited message handler.
    * @param reply a place holder for the unsolicited message
    * @see bea.jolt.JoltReply
    */
    public void onReply( JoltReply reply )
    {
        // 1 つのフィールドしか含まない STRING バッファ型のメッセージを
        // 印刷します。フィールド名は、"STRING" でなければなりません。
        // メッセージに CARRAY バッファ型を使用する場合、フィールド名は
        //"CARRAY" でなければなりません。その他のバッファの場合は、フィール
ド名は
        // FML または VIEW の要素に従っていなければなりません。

        JoltMessage msg = (JoltMessage) reply.getMessage();
        System.out.println(msg.getStringDef("STRING", "No Msg"));
    }
    public static void main( Strings args[] )
    {
        JoltUserEvent  unsolEvent;
        JoltUserEvent   helloEvent;
        EventSession    session;
        ...

        // 任意通知型メッセージを印刷できる独自のセッション・オブジェクトを
        // インスタンス化します。次に、メッセージに STRING バッファ型を使用す
```

る、

```
        // HELLO イベントと任意通知型メッセージを
        // サブスクライブします。

        session = new EventSession(...);

        helloEvent = new JoltUserEvent("HELLO", null, session);
       unsolEvent = new JoltUserEvent(JoltUserEvent.UNSOLMSG, null,
                                      session);
        ...
        // HELLO イベントと任意通知型メッセージのサブスクライブを停止します。
        helloEvent.unsubscribe();
        unsolEvent.unsubscribe();
    }
}
```

パラメータ値をクリアする

Jolt クラス・ライブラリには、オブジェクトの既存の属性値を消去し、事実上オブジェクトの再利用ができるようにする clear() メソッドが用意されています。「Jolt オブジェクトの再利用 (reuseSample.java)」は、clear() メソッドを使用してパラメータ値をクリアする方法と JoltRemoteService のパラメータ値を再利用する方法を示しています。この例では、再利用するためにサービスを破棄する必要がないことを示しています。代わりに、svc.clear();ステートメントを使用して既存の入力パラメータ値を廃棄してから addString() メソッドを再度使用しています。

コード リスト 5-13　Jolt **オブジェクトの再利用** (reuseSample.java)

```
/* Copyright 1999 BEA Systems, Inc. All Rights Reserved */
import java.net.*;
import java.io.*;
import bea.jolt.*;
/*
 * これは、各呼び出しの後に JoltRemoteService を再利用する
 * 方法を示したサンプル・プログラムです。
 */
class reuseSample
{
    private static JoltSession s_session;
    static void init( String host, short port )
    {
/* Tuxedo ドメインに接続する準備をする。 */
        JoltSessionAttributes attr = new JoltSessionAttributes();
        attr.setString(attr.APPADDRESS,"//"+ host+":"+ port);

        String username = null;
        String userrole = "sw-developer";
        String applpasswd = null;
        String userpasswd = null;

        /* 設定された認証レベルを調べる。*/
        switch (attr.checkAuthenticationLevel())
        {
           case JoltSessionAttributes.NOAUTH:
               break;
           case JoltSessionAttributes.APPASSWORD:
```

```
                applpasswd = “secret8”;
                break;
            case JoltSessionAttributes.USRPASSWORD:
                username = “myName”;
                userpasswd = “BEA#1”;
                applpasswd = “secret8”;
                break;
        }

        /* アイドル・タイムアウトなし (0) でログオンする。*/
        /* ログオフするまでネットワーク接続を維持する。 */
        attr.setInt(attr.IDLETIMEOUT, 0);
        s_session = new JoltSession(attr, username, userrole,
        userpasswd, applpasswd);
    }

    public static void main( String args[] )
    {
        String host;
        short  port;
        JoltRemoteService svc;

        if (args.length != 2)
        {
            System.err.println(“Usage: reuseSample host port”);
            System.exit(1);
        }

        /* 初期化用にホスト名とポート番号を取得する。*/
        host = args[0];
        port = (short)Integer.parseInt(args[1]);

        init(host, port);

        /* DELREC サービスのオブジェクト・リファレンスを取得する。
         * このサービスには出力パラメータはなく、入力
         * パラメータは 1 つのみ。
         */
        svc = new JoltRemoteService(“DELREC”, s_session);
        try
        {
            /* 入力パラメータ REPNAME を設定する。*/
            svc.addString(“REPNAME”, “Record1”);
            svc.call(null);
            /* 再利用する前に入力パラメータを変更する。*/
            svc.setString(“REPNAME”, “Record2”);
            svc.call(null);
```

```
            /* 入力パラメータを単に廃棄する。*/
            svc.clear();
            svc.addString("REPNAME", "Record3");
            svc.call(null);
        }
        catch (ApplicationException e)
        {
            System.err.println("Service DELREC failed: "+
            e.getMessage()+" "+ svc.getStringDef("MESSAGE", null));
        }

        /* ログオフし、オブジェクトを廃棄する。*/
        s_session.endSession();
    }
}
```

オブジェクトを再利用する

次の「Jolt リモート・サービスを拡張する (extendSample.java)」は、JoltRemoteService クラスをサブクラス化する 1 つの方法を示しています。この例では、JoltRemoteService クラスをサブクラス化して TransferService クラスを作成します。TransferService クラスは JoltRemoteService クラスを拡張し、BEA Tuxedo の BANKAPP の TRANSFER サービスを利用する Transfer 機能を追加しています。

次のコード例では、Java 言語の `extends` キーワードが使用されています。`extends` キーワードは、Java でベース・クラス (親クラス) をサブクラス化するのに使用されます。次のコードは、JoltRemoteService を拡張する方法のうちの 1 つです。

コード リスト 5-14 Jolt **リモート・サービスを拡張する**
(extendSample.java)

```
/* Copyright 1999 BEA Systems, Inc. All Rights Reserved */

import java.net.*;
import java.io.*;
import bea.jolt.*;

/*
 * この Jolt サンプル・コードからの抜粋部分は、
 * JoltRemoteService をカスタマイズする方法を示したものです。It uses the
Java language “extends” mechanism
 */
class TransferService extends JoltRemoteService
{
    public String  fromBal;
    public String  toBal;

    public TransferService( JoltSession session )
    {
        super(“TRANSFER”, session);
    }

    public String doxfer( int fromAcctNum, int toAcctNum, String
amount )
    {
/* 以前の入力パラメータすべてを消去する。*/
        this.clear();

        /* 入力パラメータを設定する。*/
        this.setIntItem(“ACCOUNT_ID”, 0, fromAcctNum);
        this.setIntItem(“ACCOUNT_ID”, 1, toAcctNum);
        this.setString(“SAMOUNT”, amount );

        try
        {
            /* 転送サービスを呼び出す。*/
            this.call(null);

            /* 出力パラメータを取得する。*/
            fromBal = this.getStringItemDef(“SBALANCE”, 0, null);
            if (fromBal == null)
                return “No balance from Account “ +
                fromAcctNum;
            toBal = this.getStringItemDef(“SBALANCE”, 1, null);
            if (toBal == null)
                return “No balance from Account “ + toAcctNum;
```

```
            return null;
        }
        catch (ApplicationException e)
        {
            /* トランザクション失敗。理由を返す。*/
            return this.getStringDef("STATLIN", "Unknown reason");
        }
    }
}

class extendSample
{
    public static void main( String args[] )
    {
        JoltSession s_session;
        String   host;
        short    port;
        TransferService xfer;
        String   failure;

        if (args.length != 2)
        {
            System.err.println("Usage: reuseSample host port");
        System.exit(1);
        }

        /* 初期化用にホスト名とポート番号を取得する。*/
        host = args[0];
        port = (short)Integer.parseInt(args[1]);

        /* Tuxedo ドメインに接続する準備をする。 */
        JoltSessionAttributes attr = new JoltSessionAttributes();
        attr.setString(attr.APPADDRESS,"//"+ host+":"+ port);

        String username = null;
        String userrole = "sw-developer";
        String applpasswd = null;
        String userpasswd = null;

        /* 設定された認証レベルを調べる。*/
        switch (attr.checkAuthenticationLevel())
        {
            case JoltSessionAttributes.NOAUTH:
                break;
            case JoltSessionAttributes.APPASSWORD:
                applpasswd = "secret8";
                break;
            case JoltSessionAttributes.USRPASSWORD:
                username = "myName";
                userpasswd = "BEA#1";
```

```
                applplasswd = “secret8”;
                break;
        }

        /* アイドル・タイムアウトなし (0) でログオンする。*/
        /* ログオフするまでネットワーク接続を維持する。*/
        attr.setInt(attr.IDLETIMEOUT, 0);
        s_session = new JoltSession(attr, username, userrole,
        userpasswd, applplasswd);

        /*
         * TransferService は JoltRemoteService を継承し、
         * 標準的な BEA Tuxedo の BankApp の TRANSFER サービスを使用します。このサービス
         * は異なるパラメータで 2 回呼び出されます。
 * s_session が初期化済みであると仮定していることに注意してください。
         */

        xfer = new TransferService(s_session);
       if ((failure = xfer.doxfer(10000, 10001, “500.00”)) != null)
            System.err.println(“Tranasaction failed: “ + failure);
        else
        {
            System.out.println(“Transaction is done.”);
            System.out.println(“From Acct Balance: “+xfer.fromBal);
            System.out.println(“  To Acct Balance: “+xfer.toBal);
        }

      if ((failure = xfer.doxfer(51334, 40343, “$123.25”)) != null)
            System.err.println(“Tranasaction failed: “ + failure);
        else
        {
            System.out.println(“Transaction is done.”);
            System.out.println(“From Acct Balance: “+xfer.fromBal);
            System.out.println(“  To Acct Balance: “+xfer.toBal);
        }

    }
}
```

Jolt アプレットの配置とローカライズ

Jolt クラス・ライブラリを使うと、クライアントの Web ブラウザから実行する Java アプリケーションを作成することができます。この種のアプリケーションを作成するには、次のアプリケーション開発作業を行います。

- Jolt アプレットを HTML ページ内に配置する
- Jolt アプレットを適切な言語と文字セットにローカライズする

次の節では、アプリケーションを開発する際のこれらの注意点について説明します。

Jolt アプレットの配置

Jolt アプレットを配置するときは、次の要件を検討してください。

- BEA Tuxedo サーバおよび Jolt サーバのインストールおよび設定の要件
- クライアント側でのアプレットの実行
- Java アプレットをダウンロードする Web サーバの要件

Jolt アプリケーションで BEA Tuxedo サーバと Jolt サーバを使うためのコンフィギュレーションについては『BEA Tuxedo システムのインストール』を参照してください。次の節では、Jolt アプレットを配置する際に一般的にクライアントや Web サーバで考慮すべき点について説明します。

クライアント側の注意事項

Jolt クラスを使用して Java アプレットを記述すると、HTML ページで動作する Java アプレットと同じように機能します。Jolt アプレットは HTML アプレットのタグを使用して HTML ページに埋め込むことができます。

```
<applet code="applet_name.class"> </applet>
```

Jolt アプレットを HTML ページに埋め込むと、HTML ページのロード時にアプレットがダウンロードされます。アプレットは、ダウンロード直後に実行されるように記述することができます。また、ユーザによる操作の実行時、タイムアウト値の到達時、または指定した間隔で実行されるようにコードを記述することもできます。さらに、アプレットのダウンロード時に別のウィンドウが開かれるように設定したり、指定した間隔で周期的に音楽を鳴らすこともできます。プログラマは、アプレットを最初にコーディングする際に、さまざまな設定を行うことができます。

注記　新しい HTML ページをブラウザにロードすると、アプレットの実行が停止します。

Web サーバに関する注意事項

Java アプレットで Jolt クラスを使用する場合、Web サーバに Jolt リレーがインストールされていない限り、Jolt サーバと Web サーバ (Java アプレットのダウンロード元の Web サーバ) は同じマシンで実行する必要があります。

Web 管理者が Web サーバをセットアップすると、すべての HTML ファイルの格納先ディレクトリが指定されます。指定されたディレクトリに「classes」という名前のサブディレクトリを作成し、Java クラスのすべてのファイルとパッケージを格納してください。次に例を示します。

```
<html-dir>/classes/bea/jolt
```

または、すべての Jolt クラスを含む `jolt.jar` ファイルを指すように `CLASSPATH` を設定することもできます。

注記　Jolt クラスのサブディレクトリは、任意の場所に置くことができます。アクセスしやすいように HTML ファイルと同じディレクトリに置くと便利です。Jolt クラスのサブディレクトリに関する唯一の条件は、Web サーバで利用できることです。

Jolt アプレットの HTML ファイルは、`jolt.jar` ファイルまたは `classes` ディレクトリを参照している必要があります。次に例を示します。

```
/export/html/
    |___ classes/
    |    |_____ bea/
    |    |    |______ jolt/
    |    |          |_____ JoltSessionAttributes.class
    |    |          |_____ JoltRemoteServices.class
    |    |          |_____ ...
    |    |_____ mycompany/
    |         |________ app.class
    |___ ex1.html
    |___ ex2.html
```

Web 管理者は、次のように `ex1.html` に「app」アプレットを指定することができます。

```
<applet codebase="classes" code=mycompany.app.class width=400
height=200>
```

Jolt アプレットをローカライズする

Jolt アプリケーションを多言後で使用する場合は、ローカライズに関わる問題を考慮する必要があります。クライアント側の Web ブラウザで実行するアプリケーション、および Web ブラウザ環境以外で実行するよう設計されたアプリケーションの両方でこの問題を考慮する必要があります。ローカライズの作業は次の 2 つです。

- アプリケーションを元の言語から希望する別の言語に変更します。
- 文字列を希望する言語に翻訳します。このとき、元の言語とは異なるアルファベットや文字セットを指定しなければならない場合があります。

ローカライズの際、Jolt クラス・ライブラリのパッケージは、Java 言語および BEA Tuxedo システムの仕様に従います。Jolt は Java の 16 ビット Unicode 文字を JSH に転送します。JSH には、Unicode をローカルな文字セットに変換するメカニズムが組み込まれています。

Unicode の Java インプリメンテーションおよび文字エスケープの詳細については、Java Development Kit (JDK) のマニュアルを参照してください。

6 JoltBeans を使う

BEA Jolt には、以前アドオンとして用意されていた JoltBeans が組み込まれています。JoltBeans は JavaBeans と同様、簡単に使用できます。JoltBeans は、Jolt クライアントを作成するために Java の開発環境で使用される JavaBeans コンポーネントです。クライアント・アプリケーションは、WebGain Visual Café などの Java 対応開発ツールのグラフィカル機能を使って作成することができます。JoltBeans には、BEA Jolt 用の JavaBeans 対応インターフェイスが用意されています。これにより、コードを記述せずに、フル機能を備えた BEA Jolt クライアントを開発することができます。

ここでは、次の内容について説明します。

- Jolt Beans の概要
- JoltBeans の基本的な使用方法
- JavaBeans イベントと BEA Tuxedo イベント
- JoltBeans における JavaBeans イベント
- JoltBeans ツールキット
- Jolt 対応 GUI Beans
- プロパティ・リストとプロパティ・エディタを使った JoltBeans プロパティの変更
- JoltBeans クラス・ライブラリの使い方
- Jolt リポジトリの使用とプロパティ値の設定
- JoltBeans のプログラミング

Jolt Beans の概要

JoltBeans は、2 種類の Java Beans のセットで構成されています。1 つは Jolt API の Beans バージョンである JoltBeans ツールキットです。2 つ目は、Jolt 対応 AWT Beans と Jolt 対応 Swing Beans を含む GUI Bean です。これらの GUI コンポーネントは、Java AWT 標準コンポーネントおよび Swing 標準コンポーネントを Jolt 対応にしたものです。これらの Jolt 対応 GUI コンポーネントを使用すると、最小限のコーディング、またはコーディングをしなくても Jolt クライアント GUI を作成できます。

開発環境のコンポーネント・パレットにある JoltBeans を、作成中の Jolt クライアント・アプリケーションの Java フォーム (複数のフォームも可) へドラッグアンドドロップできます。次に、Bean のプロパティを設定し、アプリケーションやアプレットの Bean に対してイベント・ソースとイベント・リスナの関係をグラフィカルに設定します。通常、開発ツールはイベント発生時に起動するコードを生成します。コードが生成されない場合は手動で生成します。JoltBeans を使ったクライアントの開発では、BEA Jolt リポジトリも使用します。BEA Jolt リポジトリを使用すると、BEA Tuxedo サービスに簡単にアクセスできます。

注記　現在、BEA 社が認可する JoltBeans 用の統合開発環境は、WebGain Visual Café 3.0 だけです。ただし、JoltBeans は、別の Java 開発環境 (Visual Age など) とも互換性があります。

JoltBeans ツールキットを使用する場合は、JavaBeans 対応の統合開発環境 (IDE: integrated development environment) に慣れておくことをお勧めします。この章の手順説明では、WebGain 社の Visual Caf&233; 3.0 の統合至使用して、アプレットのサンプルを作成する方法を示しています。

JoltBeans の用語

JoltBeans を使用する際には、以下の用語を参照してください。

JavaBeans
移植性があり、プラットフォームに依存しない再利用可能なソフトウェア・コンポーネント。開発環境にグラフィカルに表示されます。

JoltBeans
JoltBeans ツールキットおよび Jolt 対応 GUI Bean の 2 種類の Java Beans のセット。

カスタム GUI コンポーネント
JoltBeans と通信する Java GUI クラス。通信は、JoltBeans が提供する JavaBeans のイベント、メソッド、またはプロパティを介して行われます。

Jolt 対応 Bean
JoltInputEvents のソース、JoltOutputEvents のリスナ、またはその両方である Bean。Jolt 対応 Beans は、Bean のガイドラインに従った、カスタム GUI コンポーネントのサブセットです。

Jolt 対応 GUI Beans
2 種類の GUI コンポーネントのパッケージ AWT (Abstract Window Toolkit) および Swing。どちらにも JoltList、JoltCheckBox、JoltTextField、JoltLabel、および JoltChoice コンポーネントがあります。

JoltBeans ツールキット
BEA Jolt 用の JavaBeans 対応インターフェイス。JoltServiceBean、JoltSessionBean、JoltUserEventBean が含まれます。

関連付け
ある Bean を別の Bean からのイベントのリスナとして登録し、Bean どうしを関連付けること。

JoltBeans を Java 開発環境に追加する

JoltBeans を使用する前には、Java の開発環境に JoltBeans を追加する必要があります。

- Jolt のすべてのクラスが含まれるように、開発環境の CLASSPATH を設定します。
- お使いの開発環境のコンポーネント・ライブラリに JoltBeans を追加します。

CLASSPATH の設定方法は、使用している開発環境によって異なります。

JoltBeans には、.jar という拡張子のファイル群が用意されており、この中にはすべての JoltBeans が含まれています。これらの .jar ファイルをお使いの Java 開発環境に追加し、JoltBeans を Java ツールのコンポーネント・ライブラリで使えるようにすることができます。たとえば、WebGain 社の Visual Café を使う場合は、CLASSPATH を指定することにより、[Component Library] に ..jar ファイルを表示することができます。開発環境では、.jar ファイルの CLASSPATH の設定は一度だけです。.jar ファイルを開発環境の CLASSPATH に設定したら、[Component Library] に JoltBeans を追加できます。これで、Jolt クライアント・アプリケーションを開発する Java フォームに JoltBean を直接ドラッグアンドドロップすることができます。

Java 開発環境の CLASSPATH を設定するには、お使いの開発環境の製品マニュアルに掲載されている説明に従ってください。お使いの統合開発環境から、jolt.jar ファイルが置かれているディレクトリに移動します。jolt.jar ファイルは、通常 %TUXDIR%\udatadoj\jolt というディレクトリに格納されています。jolt.jar ファイルには、Jolt の主要なクラスが含まれています。これらのクラスが含まれるように CLASSPATH を設定してください。JoltBean .jar ファイルを CLASSPATH に追加する必要はありません。お使いの統合開発環境にコンポーネントとして追加するだけで、これらのファイルを使用することができます。

Jolt のクラスが含まれるように CLASSPATH を設定すると、お使いの開発環境のコンポーネント・ライブラリに JoltBeans を追加することができます。コンポーネント・ライブラリの設定方法については、お使いの開発環境のマニュアルを参照してください。

JoltBeans をコンポーネント・ライブラリに追加する準備ができたら、JoltBeans の開発用のバージョンのみを追加してください。詳細については、「開発用およびランタイム用の JoltBeans」を参照してください。

開発用およびランタイム用の JoltBeans

`.jar` ファイルに含まれる JoltBeans には、開発用とランタイム用の 2 つのバージョンがあります。開発バージョンの JoltBean の名前には、拡張子として `Dev` が付きます。一方、各クラスのランタイム・バージョンには、拡張子として `Rt` が付きます。たとえば、JoltBean クラスの場合、開発バージョンは `JoltBeanDev`、ランタイム・バージョンは `JoltBeanRt` になります。

開発時には、開発バージョンの JoltBeans を使用してください。開発バージョンの JoltBeans には、IDE でグラフィカルに開発を行うための別のプロパティが用意されています。たとえば、JoltBeans のグラフィックス用プロパティ (bean information) を使うと、開発環境で Bean をグラフィカルなアイコンで表示することができます。

一方、ランタイム・バージョンの JoltBeans には、このようなプロパティはありません。ランタイムでは、開発用のプロパティは必要ないためです。ランタイム・バージョンの Bean は、開発バージョンの JoltBeans の簡易版です。

お使いの開発環境でアプリケーションをコンパイルする場合は、開発バージョンの Bean が使用されます。ただし、使用中の開発環境以外のコマンド行からコンパイルを実行する場合は、`CLASSPATH` を設定することをお勧めします。これで、ランタイム・バージョンの Bean によってアプリケーションがコンパイルされます。

JoltBeans の基本的な使用方法

JoltBeans は、次のような基本的な手順で使用します。

1. お使いの開発環境のコンポーネント・ライブラリに開発用バージョンの JoltBeans を追加します。「JoltBeans を Java 開発環境に追加する」を参照してください。
2. まず、Jolt クライアント・アプリケーションまたはアプレット用に、お使いの開発環境の JoltBeans コンポーネント・パレットから Java フォーム・デザイナへ Bean をドラッグします。
3. 次に、Bean のプロパティを設定し、アプリケーションやアプレットの Bean に対してイベント・ソースとイベント・リスナの関係を設定します。つまり、Bean の「関連付け」を行います。開発ツールは、イベント発生時に起動するコードを生成します。
4. イベント・コールバックにアプリケーション・ロジックを追加します。

以上の手順は、後の節で詳しく説明します。JoltBeans の使い方に関する節では、例を示しながらこれらの手順を説明します。

JavaBeans イベントと BEA Tuxedo イベント

JavaBeans はイベントを介して通信します。BEA Tuxedo システムと JavaBeans 環境では、イベントの概念が異なります。BEA Tuxedo アプリケーションでは、イベントはアプリケーションのある部分で引き起こされ、同一アプリケーションのほかの部分に通知されます。JoltBeans イベントは複数の Bean 間で通信するイベントです。

JoltBeans で BEA Tuxedo のイベント・サブスクリプションとイベント通知を使用する

BEA Tuxedo は、仲介イベント通知と任意通知型イベント通知をサポートします。Jolt は、Jolt クライアントが BEA Tuxedo イベントを受信するメカニズムを備えています。JoltBeans も同様の機能を備えています。

注記　BEA Tuxedo のイベント・サブスクリプションとイベント通知は JavaBeans のイベントとは異なります。

以下の手順は、BEA Tuxedo の非同期通知機能が JoltBeans のアプリケーションでどのように使用されるかを示しています。

1. `setEventName()` の `setFilter()` メソッドと setFilter() メソッドを使用して、サブスクライブしたい BEA Tuxedo イベントを指定します。
2. イベント通知を受信するコンポーネントは、自身を JoltSessionBean に対する JoltOutputListener として登録します。
3. JoltUserEventBean で `subscribe()` メソッドが呼び出されます。
4. 実際に BEA Tuxedo イベント通知が到着すると、JoltSessionBean は `serviceReturned()` を呼び出し、リスナに JoltOutputEvent を送信します。JoltOutputEvent オブジェクトには BEA Tuxedo イベントのデータが含まれています。

クライアントがイベントの受信を必要としなくなったら、JoltUserEventBean で `unsubscribe()` を呼び出します。

注記　クライアントが任意通知型イベントだけを必要とする場合は、`setEventName ("\\.UNSOLMSG")` を使用します。これはプロパティ・シートを使用して設定できます。`EventName` と `Filter` は、JoltUserEventBean のプロパティです。

JoltBeans における JavaBeans イベント

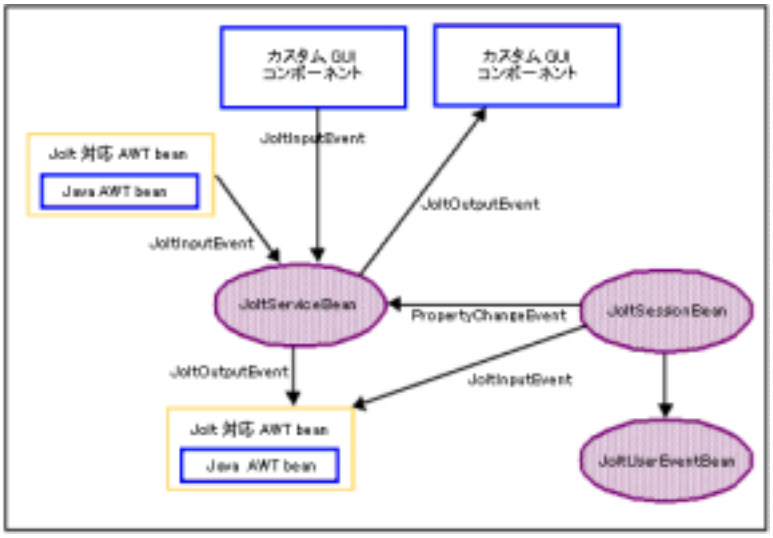
JoltBeans を使用して作成された Jolt クライアントのアプレットやアプリケーションは、通常 Jolt 対応 GUI Beans (JoltTextField や JoltList) と JoltBeans (JoltServiceBean や JoltSessionBean) で構成されています。Bean は主に JavaBeans イベントを介して通信します。

Jolt 対応 Bean は、JoltInputEvent のソース、JoltOutputEvent のリスナ、またはその両方です。JoltServiceBean は、JoltOutputEvent のソースと JoltInputEvent のリスナです。

Jolt 対応 GUI Beans はプロパティとメソッドをエクスポーズしているため、それを BEA Tuxedo サービス (JoltServiceBean として表される) のパラメータに直接リンクさせることができます。Jolt 対応 Beans は、内容が変更された場合、JoltInputEvent を介して JoltServiceBean に通知します。JoltServiceBean は、サービス・コールの後で応答データが入手できるようになると、登録されているすべての Jolt 対応 Beans に JoltOutputEvent を送信します。Jolt 対応 GUI Beans には、対応するサービスの出力パラメータで内容を更新するロジックが入っています。

次の図は、JoltBeans の関係をグラフィカルに示しています。

図 6-1JoltBeans の関係



JoltBeans ツールキット

JoltBeans ツールキットには、以下の Bean が含まれています。

- JoltSessionBean
- JoltServiceBean
- JoltUserEventBean

これらのコンポーネントは、典型的な JavaBean の機能 (再利用が簡単、開発をグラフィカルに行うことができる、など) をすべて備えた状態で Jolt クラス・ライブラリ全体を Bean のコンポーネントに変換します。

JoltBeans のクラス、コンストラクタ、およびメソッドの詳細な説明については、オンライン版の 『BEA Jolt API リファレンス』を参照してください。

以降の節では、各 Bean のプロパティについて説明します。

JoltSessionBean

JoltSessionBean は BEA Tuxedo のセッションを表し、JoltSession クラス、JoltSessionAttribute クラス、および JoltTransaction クラスの機能をカプセル化します。JoltSessionBean は、BEA Tuxedo セッションをオープンまたはクローズするメソッドに加え、タイムアウトや BEA Tuxedo ユーザ名などのセッション属性やセキュリティ属性を設定するプロパティを提供します。

JoltSessionBean は、BEA Tuxedo セッションが確立されるとき、またはクローズされるときに PropertyChange イベントを送信します。PropertyChange は、`java.beans` パッケージで定義される標準的な Bean のイベントです。このイベントの目的は、イベント・ソース Bean のプロパティ値の変更をほかの Bean に通知することです。この場合、ソースは JoltSessionBean で、ターゲットは JoltServiceBean または JoltUserEventBean です。変更されるプロパティは、JoltSessionBean の LoggedOn プロパティです。ログオンが成功してセッションが確立されると、LoggedOn は `true` に設定されます。ログオフが成功してセッションがクローズすると、LoggedOn プロパティは `false` に設定されます。

JoltSessionBean には、`beginTransaction()`、`commitTransaction()`、および `rollbackTransaction()` など、トランザクションをコントロールするメソッドがあります。

次の表は JoltSessionBean のプロパティの一覧とその説明です。

表 6-1JoltSessionBean のプロパティと説明

プロパティ	説明
AppAddress	JSL または Jolt リレーの IP アドレス (ホスト名とポート番号) を設定します。フォーマットは `//host:port number` です。 (例 : `myhost:7000`)
AppPassword	必要な場合に、ログオン時に使用する BEA Tuxedo アプリケーション・パスワードを設定します。
IdleTimeOut	IDLETIMEOUT 値を設定します。

表 6-1JoltSessionBean のプロパティと説明（続き）

プロパティ	説明
inTransaction	トランザクションが開始され、コミットもアボートもされていない場合は `true`、それ以外の場合は `false` を示します。
LoggedOn	BEA Tuxedo セッションが存在する場合は `true` 、存在しない場合は `false` を示します。
ReceiveTimeOut	RECVTIMEOUT 値を設定します。
SendTimeOut	SENDTIMEOUT 値を設定します。
SessionTimeOut	SESSIONTIMEOUT 値を設定します。
UserName	必要な場合、BEA Tuxedo ユーザ名を示します。
UserPassword	必要な場合、BEA Tuxedo ユーザ・パスワードを示します。
UserRole	必要な場合、BEA Tuxedo ユーザ・ロールを示します。

JoltServiceBean

JoltServiceBean はリモートの BEA Tuxedo サービスを表します。このサービス名は、JoltServiceBean のプロパティで設定されます。JoltServiceBean は、ほかの Bean からの JoltInputEvent をリッスンし、入力バッファを設定します。JoltServiceBean には、サービスを呼び出す `callService()` メソッドがあります。JoltServiceBean は、サービスの出力情報を伝達する JoltOutputEvent のイベント・ソースです。`callService()` が成功すると、応答メッセージを伝達する JoltOutputEvent を介してイベント・リスナ Bean に通知されます。

JoltServiceBean のメッセージ・バッファを変更および照会する主な方法はイベントを介する方法です。また、JoltServiceBean には、`setInputValue(...)` や `getOutputValue(...)` など、メッセージ・バッファに直接アクセスできるメソッドも用意されています。

次の表は JoltServiceBean のプロパティの一覧とその説明です。

表 6-2JoltServiceBean のプロパティと説明

プロパティ	説明
ServiceName	JoltServiceBean によって表される BEA Tuxedo サービスの名前。
Session	この Bean に関連付けられた JoltSessionBean であり、BEA Tuxedo クライアント・セッションへのアクセスを許可します。
Transactional	JoltServiceBean が、対応する JoltSessionBean によって開始されたトランザクションに含まれる場合は、`true` に設定されます。

JoltUserEventBean

JoltUserEventBean は、BEA Tuxedo イベントへのアクセスを提供します。この Bean のプロパティ (イベント名およびイベント・フィルタ) を設定して、BEA Tuxedo イベントのサブスクライブまたは非サブスクライブが定義されます。実際のイベント通知は、JoltSessionBean から JoltOutputEvent 形式で通知されます。

次の表は JoltUserEventBean のプロパティの一覧とその説明です。

表 6-3 JoltUserEventBean のプロパティと説明

プロパティ	説明
EventName	Bean によって表されるユーザ・イベントの名前を設定します。
Filter	イベント・フィルタを設定します。
Session	この Bean に関連付けられた JoltSessionBean であり、BEA Tuxedo クライアント・セッションへのアクセスを許可します。

Jolt 対応 GUI Beans

Jolt 対応 GUI Beans は Java AWT Beans と Swing Beans で構成されており、Java Abstract Windowing Toolkit を継承しています。以下の Bean で構成されています。

- `JoltTextField`
- `JoltLabel`
- `JoltList`
- `JoltCheckbox`
- `JoltChoice`

注記　コンパイル時のエラーを避けるため、AWT Bean どうし、または Swing Bean どうしで使用し、これらの 2 種類の Bean を組み合わせないことをお勧めします。

JoltTextField

これは、`java.awt.TextField` および Swing `JTextfield` を拡張したものです。JoltTextField にはサービスへの入力の一部が入っています。JoltServiceBean は JoltTextField によって引き起こされたイベントをリッスンできます。JoltTextField は、内容が変更されるとリスナ (通常は JoltServiceBean) に JoltInputEvent を送信します。

JoltTextField はサービスからの出力を表示します。この場合、JoltTextField は JoltServiceBean から JoltOutputEvent をリッスンし、リンクされたフィールドのオカレンスに従って内容を更新します。

JoltLabel

JoltLabel は、`java.awt.Label` および Swing `JLabel` を Jolt 対応させた拡張クラスであり、JoltFieldName プロパティによって Jolt 出力バッファの特定のフィールドにリンクされています。フィールドが複数のオカレンスを持つ場合、この textfield がリンクされるオカレンスは、この Bean の occurrenceIndex プロパティによって指定されます。JoltLabel を JoltServiceBean に接続して、サービスからの出力を表示することができます。JoltLabel は JoltServiceBean から JoltOutputEvent をリッスンし、リンクされたフィールドのオカレンスに従ってその内容を更新します。

JoltList

JoltList は、`java.awt.List` および Swing `Jlist` を Jolt 対応させた拡張クラスであり、JoltFieldName プロパティによって Jolt の入力バッファまたは出力バッファの特定の Jolt フィールドにリンクされています。Jolt 入力バッファ内でフィールドが複数のオカレンスを持つ場合、リストがリンクされるオカレンスは、この Bean の occurrenceIndex プロパティによって指定されます。JoltList は 次に示す 2 通りの方法で JoltServiceBean に接続できます。

- JoltList にはサービスへの入力の一部が入っています。JoltServiceBean は JoltList によって引き起こされたイベントをリッスンします。JoltList は listbox の選択内容が変わると、リスナに JoltInputEvent を送信します。この場合、JoltInputEvent は選択された項目の単一の値で設定されます。
- JoltList はサービスからの出力を表示します。サービスの出力表示に JoltList が使用されると、JoltList は JoltServiceBean から JoltOutputEvent をリッスンし、リンクされたフィールドのすべてのオカレンスに従って内容を更新します。

JoltCheckbox

JoltCheckbox は、`java.awt.Checkbox` および Swing `JCheckBox` を Jolt 対応させた拡張クラスであり、JoltFieldName プロパティによって Jolt 入力バッファの特定のフィールドにリンクされています。フィールドが複数のオカレンスを持つ場合、checkbox がリンクされるオカレンスは、この Bean の occurrenceIndex プロパティによって指定されます。

JoltCheckbox を JoltServiceBean と接続して、サービスの入力の一部を入れることができます。JoltServiceBean は JoltCheckbox によって引き起こされたイベントをリッスンできます。checkbox の選択内容が変わると、JoltCheckbox はリスナ (通常は JoltServiceBean) に JoltInputEvent を送信します。この場合、JoltInputEvent は、チェックボックスが選択されているときは String データ型の `TrueValue` プロパティで設定され、チェックボックスが選択されていないときは `FalseValue` プロパティで設定されます。

JoltChoice

JoltChoice は `java.awt.Choice` および Swing `JChoice` を Jolt 対応させた拡張クラスであり、JoltFieldName プロパティによって Jolt 入力バッファの特定のフィールドにリンクされています。フィールドが複数のオカレンスを持つ場合、この選択がリンクされるオカレンスは、この Bean の occurrenceIndex プロパティによって指定されます。

JoltChoice を JoltServiceBean と接続して、サービスの入力の一部を入れることができます。JoltServiceBean は JoltChoice によって引き起こされたイベントをリッスンできます。choicebox の選択内容が変わると、JoltChoice はリスナ (通常は JoltServiceBean) に JoltInputEvent を送信します。この場合、JoltInputEvent は選択された項目の単一の値で設定されます。

注記　クラスの詳細については、『BEA Jolt API リファレンス』を参照してください。

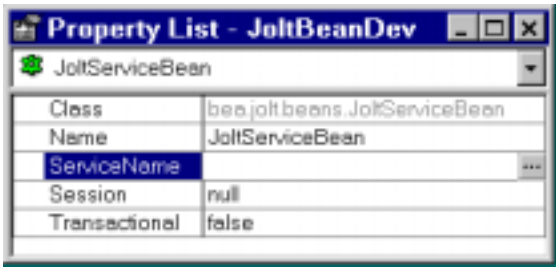
プロパティ・リストとプロパティ・エディタを使った JoltBeans プロパティの変更

JoltBeans のほとんどのプロパティ値は、次の「プロパティ・リストと [...] ボタン」に示す統合開発環境 (Visual Caf&233; など) 祉 eィ・リストで、右側の列を編集するだけで変更できます。

JoltBeans の一部のプロパティについては、カスタム・プロパティ・エディタが用意されています。

プロパティ・リストからアクセスできるカスタム・プロパティ・エディタには、プロパティ値を変更するときに使用するダイアログ・ボックスがあります。対応するプロパティ値の隣にある [...] ボタンをクリックすると、プロパティ・リストからカスタム・プロパティ・エディタを呼び出すことができます。

図 6-2 プロパティ・リストと [...] ボタン



[...] ボタンを選択すると、次の図に示すようなプロパティ・エディタが表示されます。

図 6-3 カスタム・プロパティ・エディタ・ダイアログ・ボックス

JoltBeans のカスタム・プロパティ・エディタはキャッシュされた情報を読み取ります。最初はキャッシュされた情報がないため、プロパティ・エディタを初めて使用するときは、このダイアログ・ボックスには何も表示されません。Jolt リポジトリにログオンして、そのリポジトリからプロパティ・エディタ・キャッシュをロードします。

プロパティ・リストとプロパティ・エディタを使ったログオンの詳細については、6-49 ページの「Jolt リポジトリの使用とプロパティ値の設定」を参照してください。

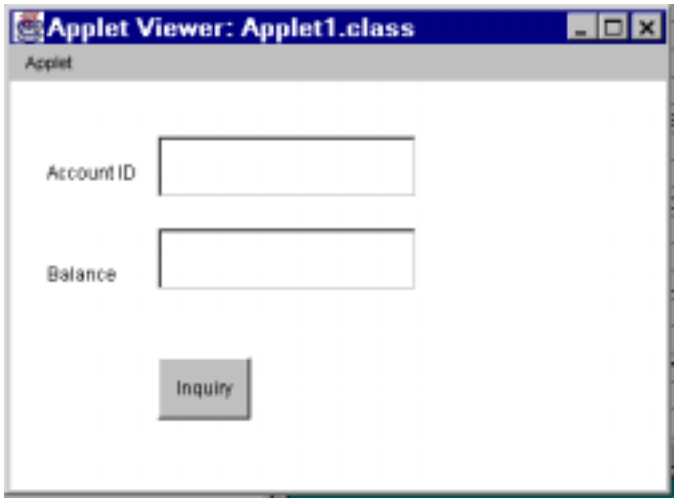
JoltBeans クラス・ライブラリの使い方

ここでは、以下の操作を行うためのアプレットの作成方法を説明します。

- [Account ID] を入力する。
- [Inquiry] ボタンをクリックする。
- 口座残高 (Balance) を表示する (次の図を参照)。

次の図は、JoltBeans を含む完全な Java フォームの例です。このアプレットは BANKAPP の INQUIRY サービスのクライアント機能を実装しています。このサンプルを実行するには、BEA Tuxedo サーバが実行されていなければなりません。

図 6-4 照会アプレットの例



Java フォームで必要となる各項目の例については、6-22 ページの「Visual Café 3.0 のフォーム・デザイナ」を参照してください。図の各項目については、次の「フォームに必要なコンポーネント」で説明します。

表 6-4 フォームに必要なコンポーネント

コンポーネント	目的
アプレット (JFC アプレットが選択された場合は JApplet)	開発環境で Bean をペイントするときに使用するフォーム。
JoltSessionBean	BEA Tuxedo セッションにログオンします。
JoltTextField	ユーザから入力を取得します (この場合は ACCOUT_ID)。
JoltTextField	結果を表示します (この場合は SBALANCE)。
JoltServiceBean	BEA Tuxedo サービスにアクセスします (この場合は BANKAPP からの INQUIRY)。
ボタン	アクションを開始します。
Label	アプレット上のフィールドを説明します。

サンプル・フォームの作成

この例は、Visual Café 統合開発環境で作成されたサンプル・フォームです。この例では、口座 ID (Account ID) を入力し、BEA Tuxedo サービスを使って口座残高を表示するためのアプレットの作成方法を説明します。

このサンプルを作成するには、次のような基本的な手順に従います。

1. Visual Café で、[File] の [New Project] を選択し、JFC アプレットまたは AWT アプリケーションを選択します。この手順により、JoltBeans のドロップ先のフォーム・デザイナが決まります。
2. アプレットで使用したいすべての JoltBeans をコンポーネント・ライブラリからフォーム・デザイナにドラッグ・アンドドロップします。

3. プロパティ・リストまたはカスタム・プロパティ・エディタを使用して、Bean を変更またはカスタマイズします。
4. [Interaction Wizard] を使用して Bean を関連付けます。
5. アプレットをコンパイルします。

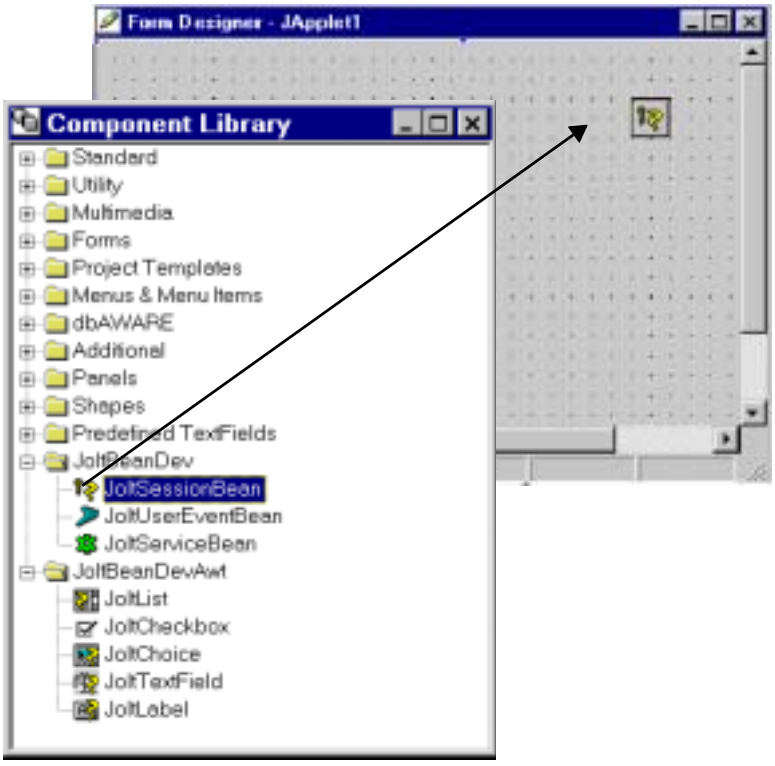
これらの手順は、次の節でさらに詳しく説明されます。

注記　以前の Visual Café に用意されていたグラフィカル・インターフェイスは、Visual Café のイ祉イスとは異なります。このサンプル・アプレットは、以前のバージョンの Visual Café を使って作成することもできますが、[Interaction Wizard] での手順が若干異なります。

JoltBeans をフォーム・デザイナ上に配置する

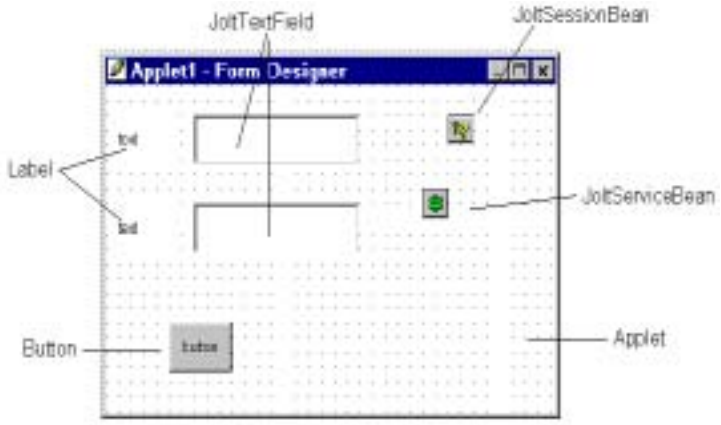
1. [File] の [New Project] を選択し、[JFC Applet] を選択します。
2. 次の図に示すように Bean をコンポーネント・ライブラリから目的の位置へドラッグ・アンド・ドロップします。

図 6-5Visual Café におけるフォーム・デザイナ



「Visual Café 3.0 のフォーム・デザイナ」は、フォーム・デザイナのパレットに配置された JoltBeans がどのように表示されるかを示しています。

図 6-6Visual Café 3.0 のフォーム・デザイナ



3. 各 Bean のプロパティを設定します。ボタン、ラベル、フィールドを変更またはカスタマイズする場合は、プロパティ・リストを使用します。カスタム・プロパティ・エディタを使用する JoltBeans もあります。

   「JoltTextField のプロパティ・リストとカスタム・プロパティ・エディタの例」は、テキスト・フィールドのプロパティ・リストにある JoltFieldName を選択すると、カスタム・プロパティ・エディタがどのように表示されるかを示しています。

4. たとえば、JoltTextField の JoltFieldName プロパティを ACCOUNT_ID に設定します。

注記　JoltBeans のプロパティの設定と変更の詳細については、6-49 ページの「Jolt リポジトリの使用とプロパティ値の設定」を参照してください。

次の図は、設定する必要があるプロパティ値を指定しています。太字で示した値は必須値、普通の文字で示した値は推奨値、イタリック体は設定によって変わる必須値の例です。

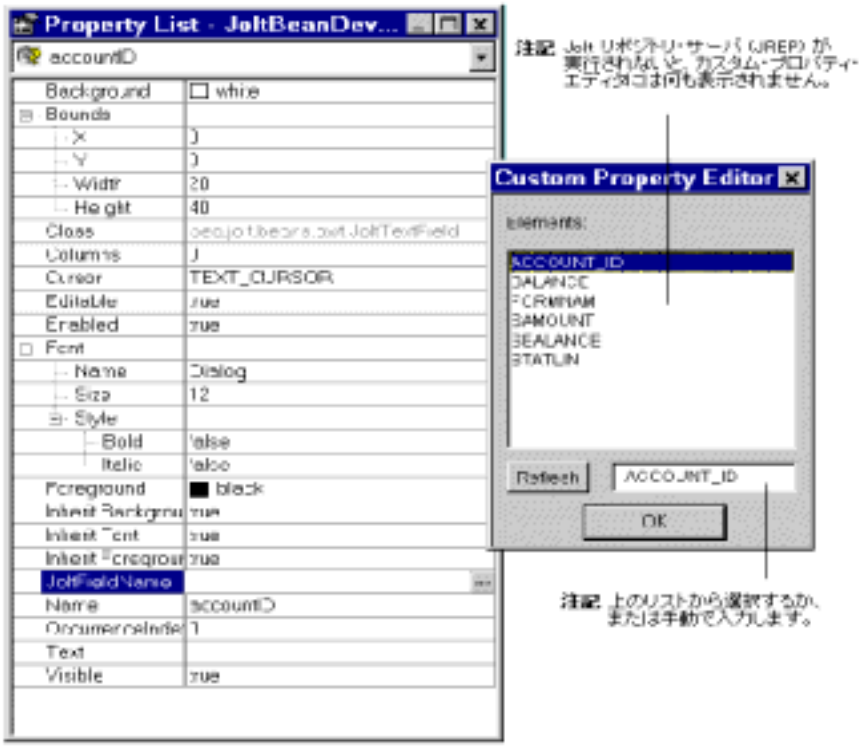
表 6-5 必須および推奨のプロパティ値

Bean	プロパティ	値
label1	Text	Account ID
label2	Text	Balance
JoltTextField1	Name	accountId
JoltTextField1	JoltFieldName	ACCOUNT_ID
JoltTextField2	Name	balance
JoltTextField2	JoltFieldName	SBALANCE
JoltSessionBean1	AppAddress	//tuxserv:2010
JoltServiceBean1	Name	inquiry
JoltServiceBean1	ServiceName	INQUIRY
button1	Label	Inquiry

注記　この例では、occurrenceIndex のデフォルト値 0 は両方の JoltTextField に有効です。

JoltBean のプロパティについての一般的な説明については、次の「JoltTextField のプロパティ・リストとカスタム・プロパティ・エディタの例」および 6-49 ページの「Jolt リポジトリの使用とプロパティ値の設定」を参照してください。

図 6-7JoltTextField のプロパティ・リストとカスタム・プロパティ・エディタの例



5. JoltFieldName のプロパティ値を変更するには、プロパティ・リストの JoltFieldName の [...] ボタンをクリックします。

   カスタム・プロパティ・エディタが表示されます。

6. 新しいフィールド名 (この例では「ACCOUNT_ID」) を選択または入力して、[OK] をクリックします。

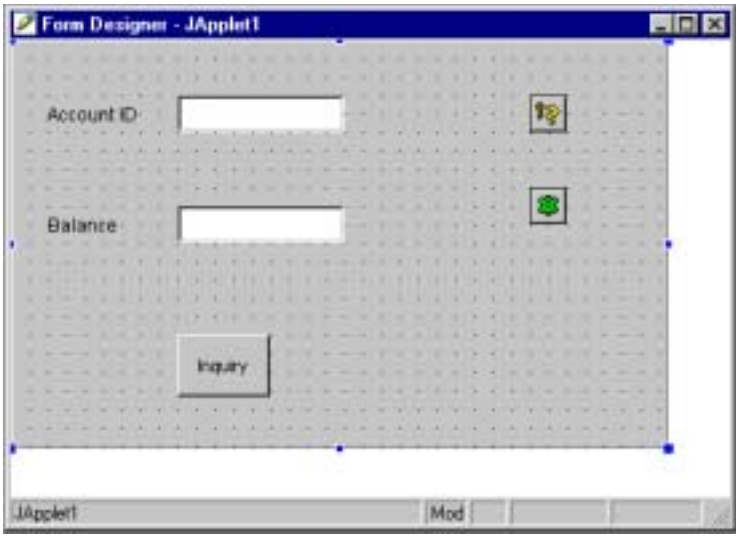
   変更内容は、「JoltFieldName を変更した JoltTextField プロパティ・リスト」のプロパティ・リストと 6-26 ページの「プロパティが変更されたアプレット・フォーム・デザイナの例」のテキスト・フィールドに反映されます。

注記　カスタム・プロパティ・エディタに表示されるプロパティはローカルにキャッシュされます。そのため、元のデータベースが変更された場合は、[Refresh] ボタンを押さないと、最新の使用可能なプロパティが表示されません。

図 6-8JoltFieldName を変更した JoltTextField プロパティ・リスト

次の「プロパティが変更されたアプレット・フォーム・デザイナの例」は、Bean のプロパティ・リストのフィールドにテキストを追加した後、ボタンやテキストフィールドのテキストがどのように変わるかを示します。

図 6-9 プロパティが変更されたアプレット・フォーム・デザイナの例



7. 右の列にプロパティ値を設定した後、Visual Caf&233; の Interacti 姉 ard を使用して Bean を関連付けることにより、Bean のインタラクションの仕方を定義します。詳細については、6-23 ページの「必須および推奨のプロパティ値」を参照してください。詳細については、「JoltBeans の関連付け」を参照してください。

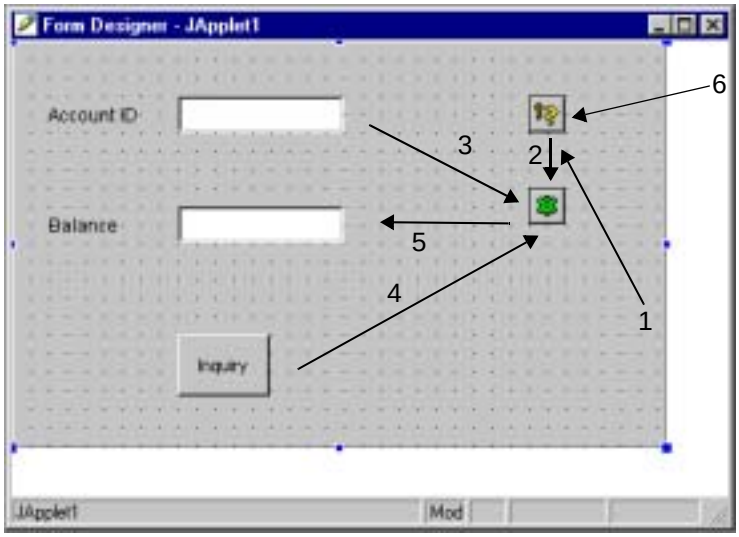
JoltBeans の関連付け

すべての Bean がフォーム上に配置され、プロパティを設定した後、それらのイベントを関連付けます。「JoltBeans を関連付ける順序」は、Bean を関連付ける際の手順の例を示しています。

Bean を関連付けると、さまざまなフォームの Bean 間のイベント・ソースとイベント・リスナの関係を確立できます。たとえば、JoltServiceBean はボタンの ActionEvent のリスナであり、このイベントをリッスンしたときに `callService()` を呼び出します。Visual Caf&233; の 鯖 ction Wizard を使って Bean どうしを関連付けてください。

次の図は、このサンプル・アプレットを作成するための、Bean を関連付ける順序を示しています。図中の数は、次に説明する手順の番号を示しています。

図 6-10JoltBeans を関連付ける順序



以下の手順は、6-27 ページの「JoltBeans を関連付ける順序」に示した番号に対応しています。これらの手順は、次の節でさらに詳しく説明します。

手順 1: JoltSessionBean とログオンを関連付ける

手順 2: propertyChange を使用して JoltSessionBean を JoltServiceBean に関連付ける

手順 3: JoltInputEvent を使用して入力項目 accountID の JoltTextField を JoltServiceBean に関連付ける

手順 4: JoltAction を使用してボタンを JoltServiceBean に関連付ける

手順 5: JoltOutputEvent を使用して JoltServiceBean を Balance の JoltTextField に関連付ける

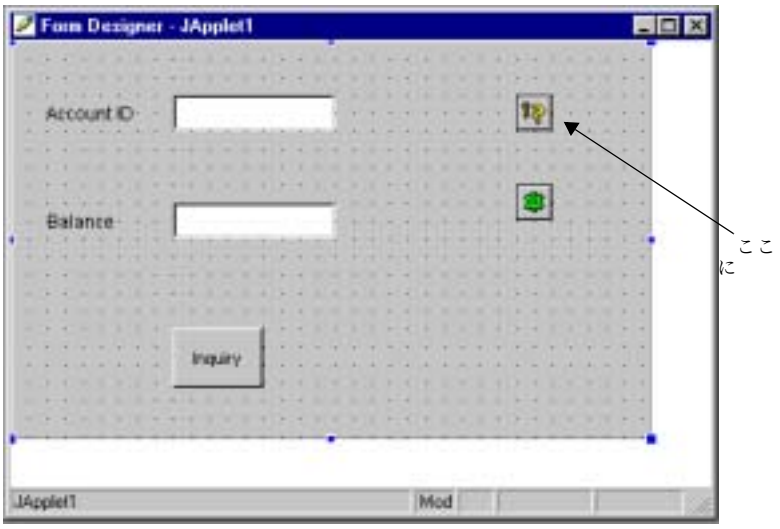
手順 6: JoltSessionBean とログオフを関連付ける

手順 7: アプレットをコンパイルする (図には表示されていません)。

手順 1: JoltSessionBean とログオンを関連付ける

1. [Form Designer] ウィンドウで、[Interaction Wizard] ボタンをクリックします。
2. 次の図に示すようにアプレット・ウィンドウの一点をクリックし、そのまま JoltSessionBean へドラッグします。

図 6-11 アプレットを Jolt Session Bean に関連付ける

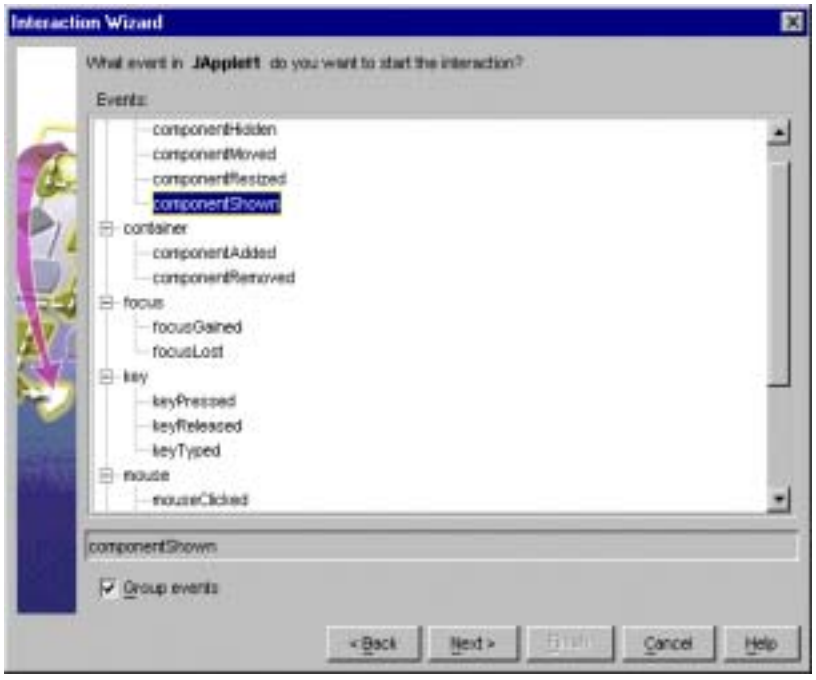


[Interaction Wizard] ウィンドウが表示されます (6-30 ページの「[ComponentShown] イベントを選択する」を参照)。

「What event in JApplet1 do you want to start the interaction?」というメッセージが表示されます。

3. 次の図のように、インタラクションを開始するイベントとして [Interaction Wizard] ウィンドウの [ComponentShown] を選択します。

図 6-12[ComponentShown] イベントを選択する



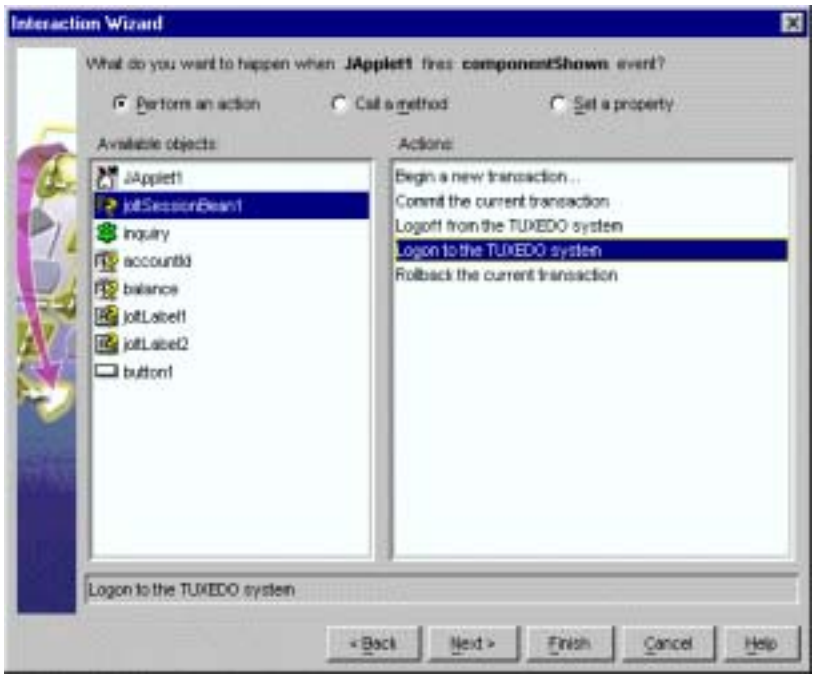
4. [Next] をクリックします。

   [Interaction Wizard] ウィンドウが表示されます (6-31 ページの「[Logon to the TUXEDO System] を選択する」を参照)。

   「What do you want to happen when Japplet1 fires componentShown event?」というメッセージが表示されます。

5. 次の図のように、[Perform an action] ラジオ・ボタンを選択し、[Logon to the Tuxedo system] を選択します。

図 6-13[Logon to the TUXEDO System] を選択する



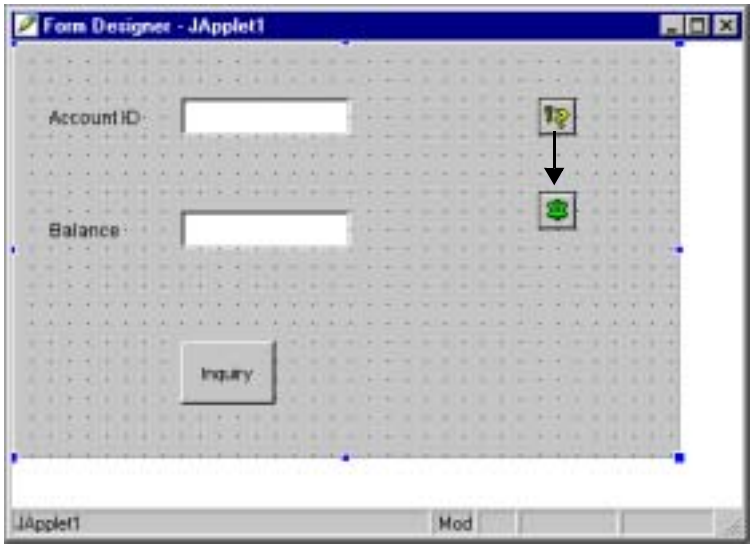
6. [Finish] をクリックします。

「手順 1: JoltSessionBean とログオンを関連付ける」を終了すると、アプレットを最初に開いたときに送信されるイベント (例 : ComponentShown) で、JoltSessionBean の `logon()` メソッドをトリガできるようになります。

手順 2: propertyChange を使用して JoltSessionBean を JoltServiceBean に関連付ける

1. Visual Café の [Form Designer] ウィンドウのツールバーで [Interaction Tool] アイコンをクリックします。
2. 次の図に示すように [JoltSessionBean] をクリックし、そのまま JoltServiceBean へドラッグします。

図 6-14JoltSessionBean を JoltServiceBean に関連付ける

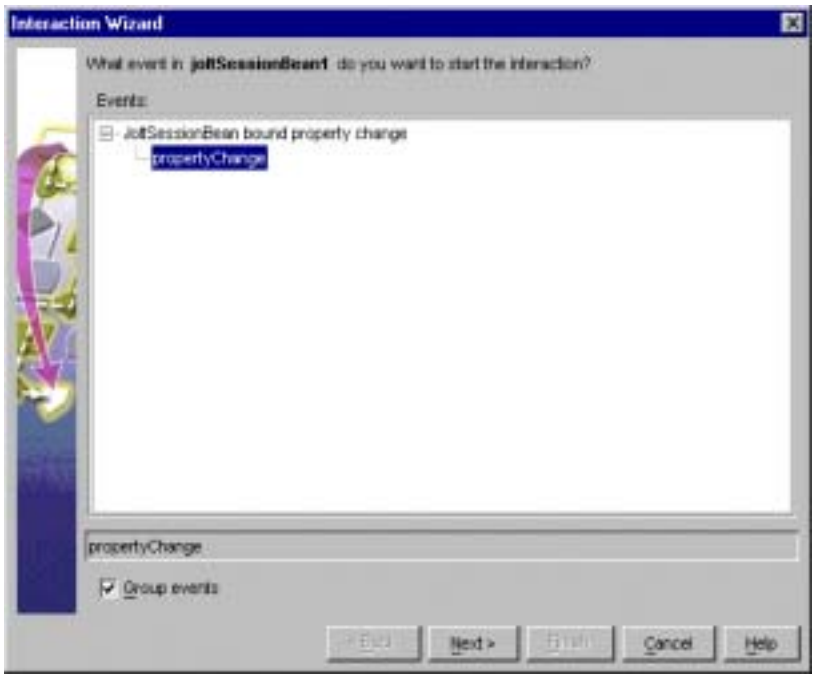


[Interaction Wizard] ウィンドウが表示されます (6-33 ページの「[propertyChange] イベントを選択する」を参照)。

「What event in joltSessionBean1 do you want to start the interaction?」というメッセージが表示されます。

3. インタラクションを開始するイベントとして [propertyChange] を選択します。

図 6-15[propertyChange] イベントを選択する



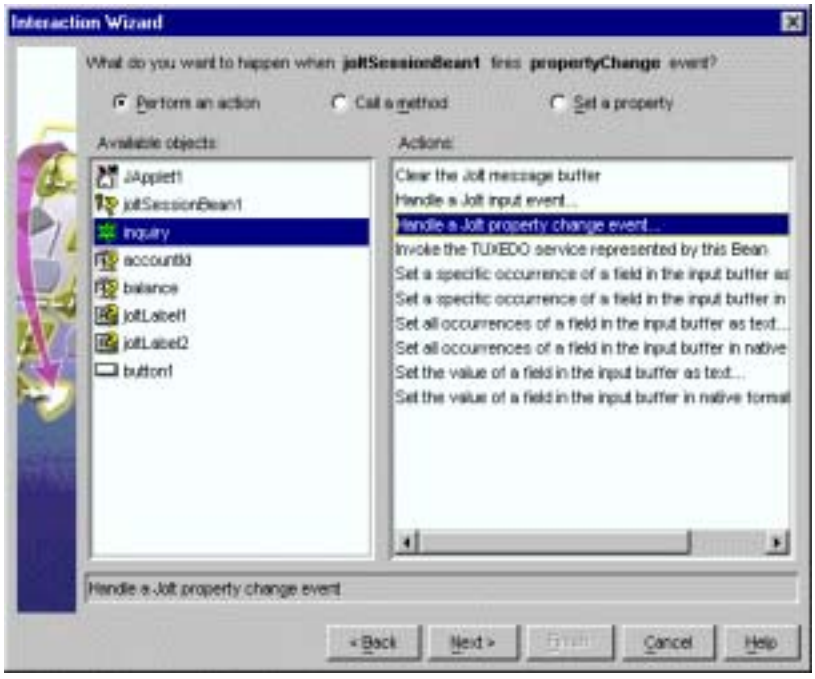
4. **[Next]** をクリックします。

   [Interaction Wizard] ウィンドウが表示されます (6-34 ページの「[Handle a Jolt property change event] を選択する」を参照)。

   「What do you want to happen when joltSessionBean1 fires propertyChange event?」というメッセージが表示されます。

5. 次の図のように、メソッドとして [Handle a Jolt property change event] を選択します。

図 6-16[Handle a Jolt property change event] を選択する



6. [Next] をクリックします。

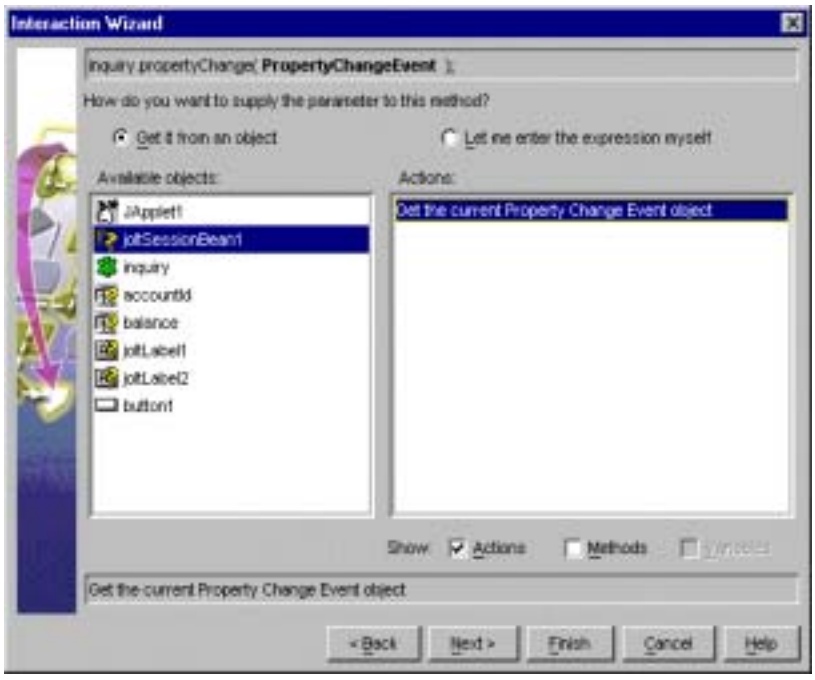
   [Interaction Wizard] ウィンドウが表示されます (6-35 ページの「[joltSesssionBean1] を選択する」を参照)。

   「How do you want to supply the parameter to this method?」というメッセージが表示されます。

   使用可能なオブジェクトとアクションの一覧が表示されます。

7. 次の図のように、アクションを実行するオブジェクトとして [joltSessionBean1] を選択します。
8. アクションとして [Get the current Property Change Event] オブジェクトを選択します。

図 6-17[joltSesssionBean1] を選択する



9. [Finish] をクリックします。

「手順 2: propertyChange を使用して JoltSessionBean を JoltServiceBean に関連付ける」を終了すると、`logon()` 完了時に JoltSessionBean が propertyChange イベントを送信できるようになります。JoltServiceBean はこのイベントをリッスンし、サービスをこのセッションに関連付けます。

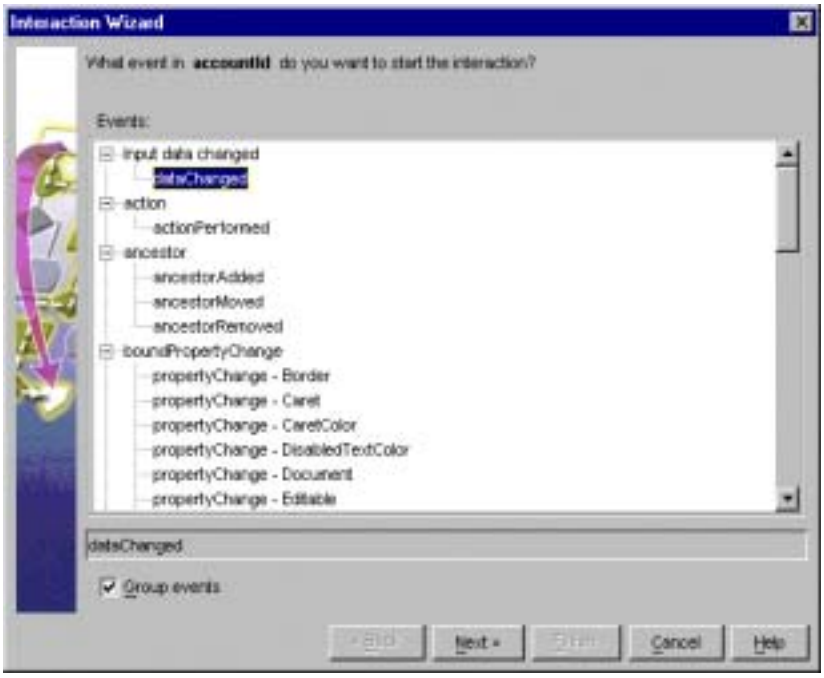
手順 3: JoltInputEvent を使用して入力項目 accountID の JoltTextField を JoltServiceBean に関連付ける

1. Visual Café の [Form Designer] ウィンドウで [Interaction Wizard] アイコンをクリックします。
2. [accountID JoltTextfield Bean] を選択し、そのまま JoltServiceBean へドラッグします。

   次の図のように、[Interaction Wizard] ウィンドウが表示されます。

   「What event in accountId do you want to start the interaction?」というメッセージが表示されます。
3. 次の図のように、イベントとして [dataChanged] を選択します。

図 6-18[dataChanged] イベントを選択する



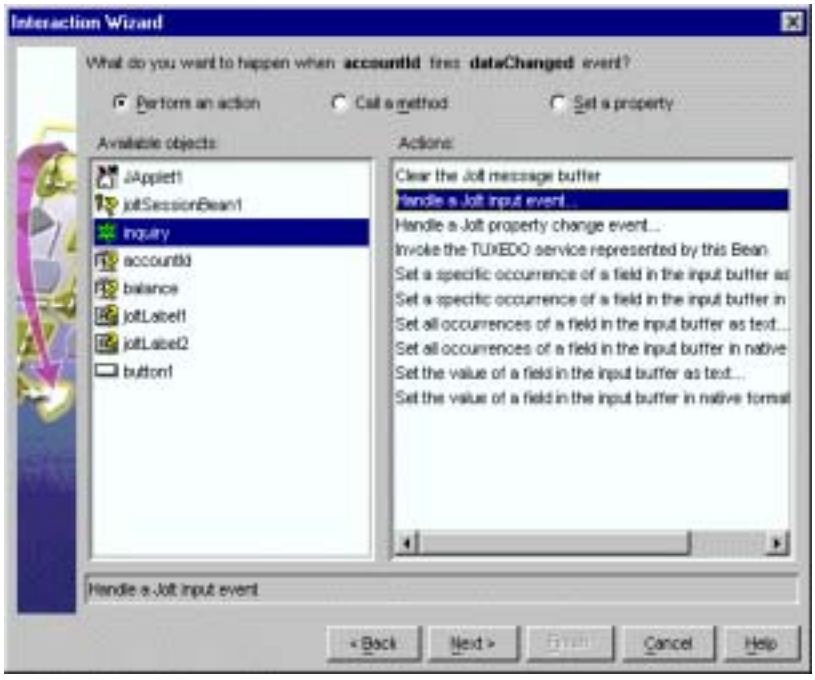
4. [Next] をクリックします。

[Interaction Wizard] ウィンドウが表示されます (6-37 ページの「オブジェクトとして [inquiry]、アクションとして [Handle a Jolt input event] を選択する」を参照)。

「What do you want to happen when accountId fires dataChanged event?」というメッセージが表示されます。

5. 次の図のように、パラメータを実行するオブジェクトとして joltServiceBean の [inquiry] を選択します。
6. アクションとして [Handle a jolt input event] を選択します (次の図を参照)。

図 6-19 オブジェクトとして [inquiry]、アクションとして [Handle a Jolt input event] を選択する



7. [Next] をクリックします。

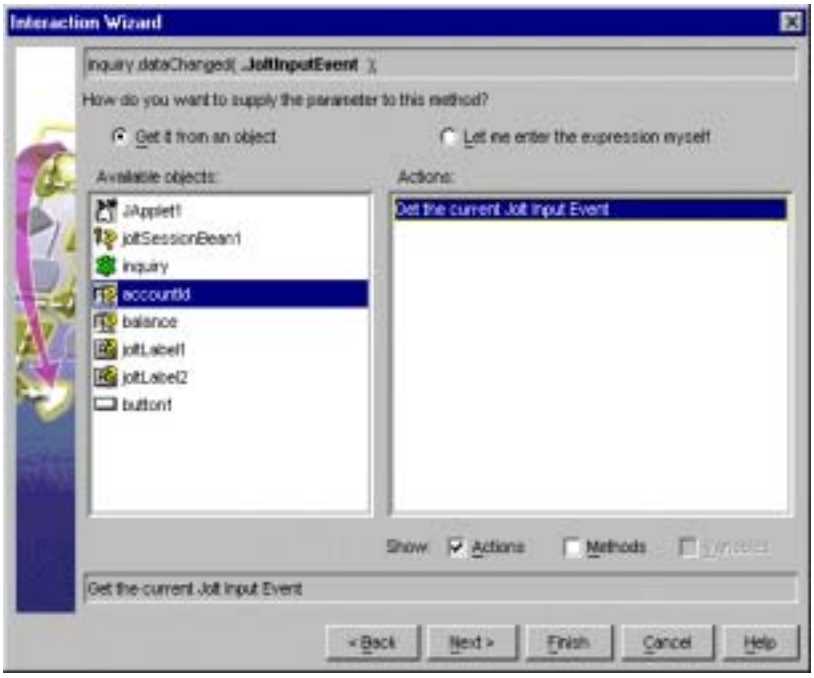
[Interaction Wizard] ウィンドウが表示されます (6-38 ページの「オブジェクトとして [accountId]、アクションとして [Get the current Jolt Input Event] を選択する」を参照)。

「How do you want to supply the parameter to this method?」というメッセージが表示されます。

使用可能なオブジェクトとアクションの一覧が表示されます。

8. 次の図のように、オブジェクトとして [accountId] を選択します。
9. アクションとして [Get the current Jolt Input Event] を選択します。

図 6-20 オブジェクトとして [accountId]、アクションとして [Get the current Jolt Input Event] を選択する



10. [Finish] をクリックします。

「手順 3: JoltInputEvent を使用して入力項目 accountID の JoltTextField を JoltServiceBean に関連付ける」を終了すると、最初のテキスト・フィールドに口座番号を入力できるようになります。JoltTextField の JoltFieldName のプロパティが「ACCOUNT_ID」に設定されます。このテキストボックスの内容が変更されると JoltInputEvent が JoltServiceBean に送信されます。JoltServiceBean は、このテキスト・ボックスから JoltInputEvent をリッスンします。JoltInputEvent オブジェクトには、このフィールドの名前、値、およびオカレンス・インデックスが含まれます。

手順 4: JoltAction を使用してボタンを JoltServiceBean に関連付ける

1. Visual Café の [Form Designer] ウィンドウで [Interaction Wizard] アイコンをクリックします。
2. [Inquiry] ボタンをクリックし、JoltServiceBean へドラッグします。

   次の図のように、[Interaction Wizard] ウィンドウが表示されます。

   「What event in button1 do you want to start the interaction?」というメッセージが表示されます。
3. 次の図のように、イベントとして [actionPerformed] を選択します。

図 6-21 イベントとして [actionPerformed] を選択する

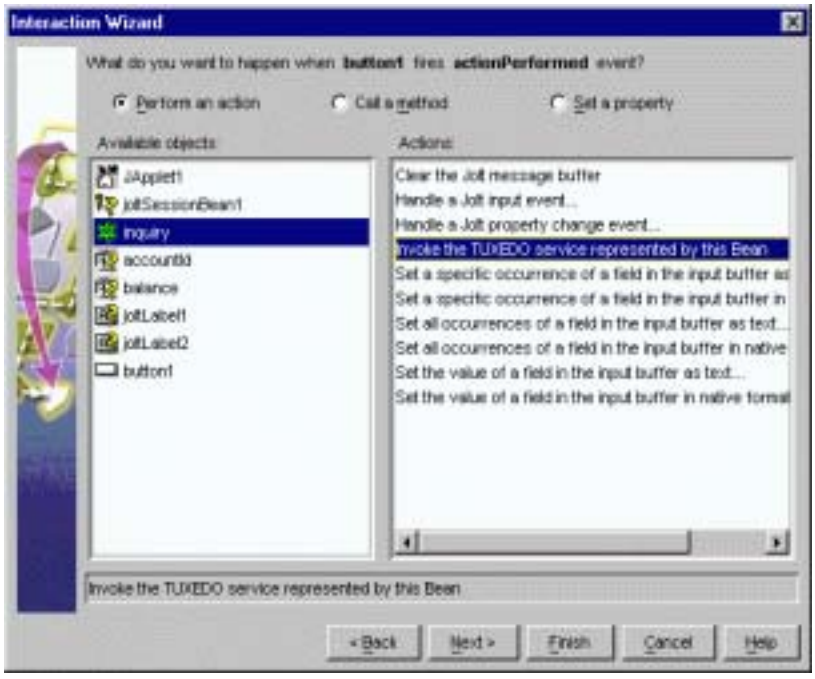
4. [Next] をクリックします。

[Interaction Wizard] ウィンドウが表示されます (6-41 ページの「オブジェクトとして [inquiry] を選択し、アクションとして [Invoke the TUXEDO Service represented by this Bean] を選択する操作」を参照)。

「What do you want to happen when button1 fires actionPerformed event?」というメッセージが表示されます。

5. 次の図のように、オブジェクトとして [inquiry] を選択します。
6. アクションとして [Invoke the TUXEDO Service represented by this Bean] を選択します (次の図を参照)。

図 6-22 オブジェクトとして [inquiry] を選択し、アクションとして [Invoke the TUXEDO Service represented by this Bean] を選択する操作



7. [Finish] をクリックします。

「手順 4: JoltAction を使用してボタンを JoltServiceBean に関連付ける」を終了すると、JoltServiceBean の `callService()` メソッドを、[Inquiry] ボタンからの ActionEvent でトリガできます。

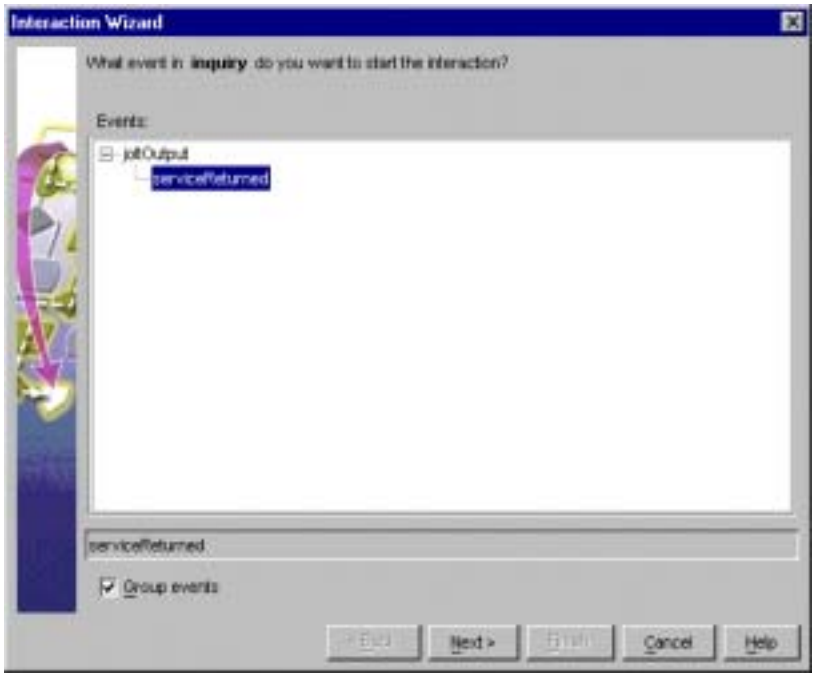
手順 5: JoltOutputEvent を使用して JoltServiceBean を Balance の JoltTextField に関連付ける

1. Visual Café の [Form Designer] ウィンドウで [Interaction Wizard] アイコンをクリックします。
2. [JoltServiceBean] を選択して、[AmountJoltTextFeald Bean] へドラッグします。

   次の図のように [Interaction Wizard] ウィンドウが表示されます。

   「What event in inquiry do you want to start the interaction?」というメッセージが表示されます。
3. 次の図のように、イベントとして [serviceReturned] を選択します。

図 6-23[ServiceReturned] イベントを選択する



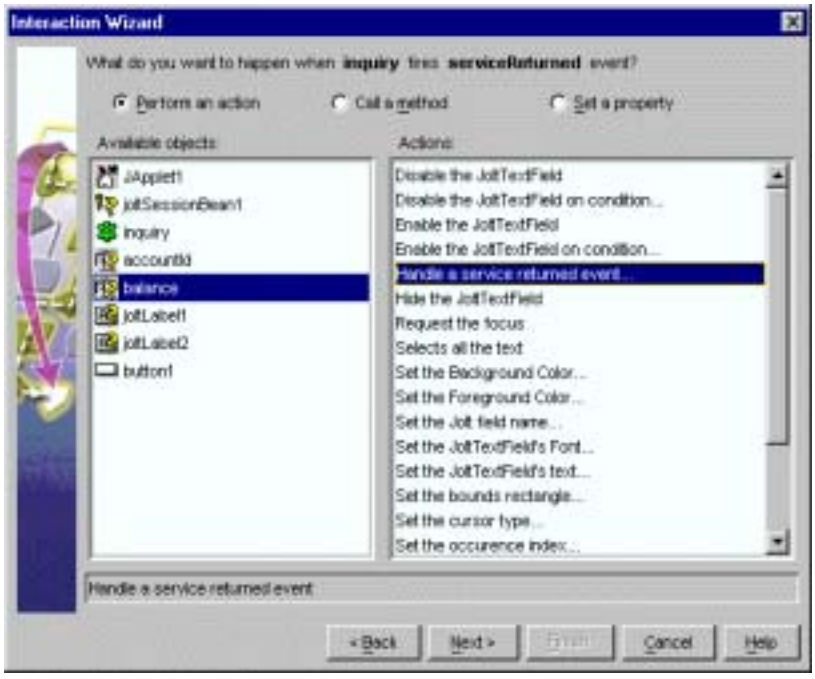
4. [Next] をクリックします。

[Interaction Wizard] ウィンドウが表示されます (6-44 ページの「オブジェクトとして [balance] を選択し、アクションとして [Handle a service returned event] を選択します。」を参照)。

「What do you want to happen when inquiry fires serviceReturned event?」というメッセージが表示されます。

5. 次の図のように、オブジェクトとして [balance] を選択します。
6. アクションとして [Handle a service returned event...] を選択します (次の図を参照)。

図 6-24 オブジェクトとして [balance] を選択し、アクションとして [Handle a service returned event] を選択します。



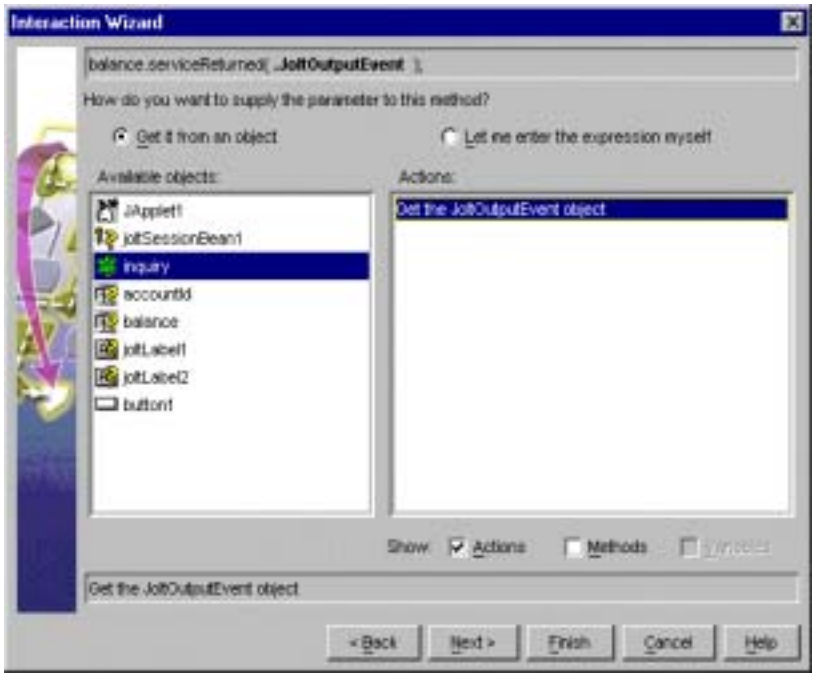
7. [Next] をクリックします。

[Interaction Wizard] ウィンドウが表示されます (6-46 ページの「オブジェクトとして [inquiry]、アクションとして [Get the JoltOutputEvent object] を選択する」を参照)。

「How do you want to supply the parameter to this method?」というメッセージが表示されます。

8. 次の図のように、オブジェクトとして [inquiry] を選択します。
9. アクションとして [Get the JoltOutputEvent] オブジェクトを選択します (次の図を参照)。

図 6-25 オブジェクトとして [inquiry]、アクションとして [Get the JoltOutputEvent object] を選択する



10. [Finish] をクリックします。

「手順 5: JoltOutputEvent を使用して JoltServiceBean を Balance の JoltTextField に関連付ける」を終了すると、JoltServiceBean はリモートのサービスから応答データを受信したときに JoltOutputEvent を送信することができるようになります。JoltOutputEvent オブジェクトには、出力バッファのフィールドにアクセスするためのメソッドが入っています。JoltTextField は INQUIRY サービスの結果を表示します。

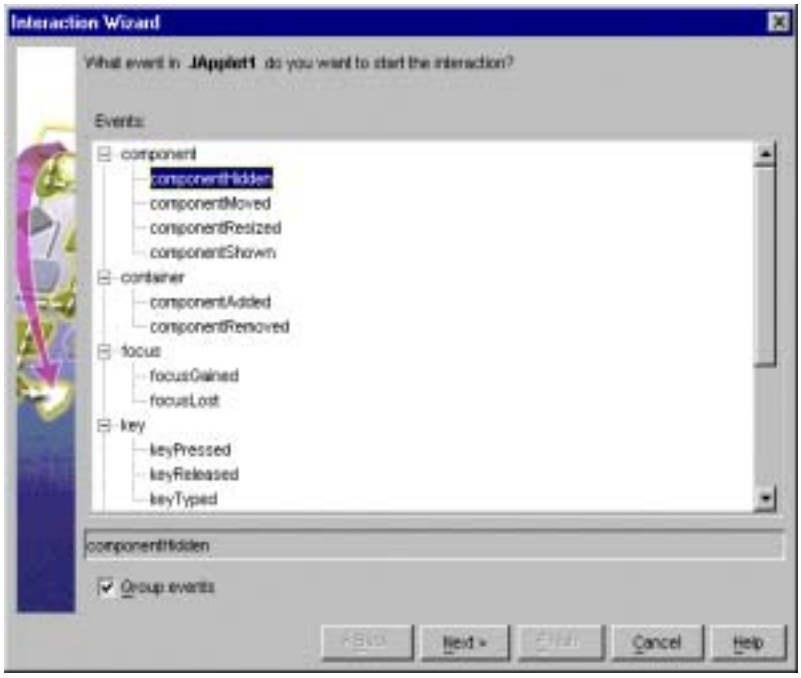
手順 6: JoltSessionBean とログオフを関連付ける

1. Visual Café の [Form Designer] ウィンドウで [Interaction Wizard] アイコンをクリックします。
2. アプレット・ウィンドウの一点をクリックし (Bean はクリックしない)、そのまま JoltSessionBean へドラッグします。

   次の図のように [Interaction Wizard] ウィンドウが表示されます。

   「What event in JApplet1 do you want to start the interaction?」というメッセージが表示されます。
3. 次の図のように、イベントとして [componentHidden] を選択します。

図 6-26 イベントとして [componentHidden] を選択する



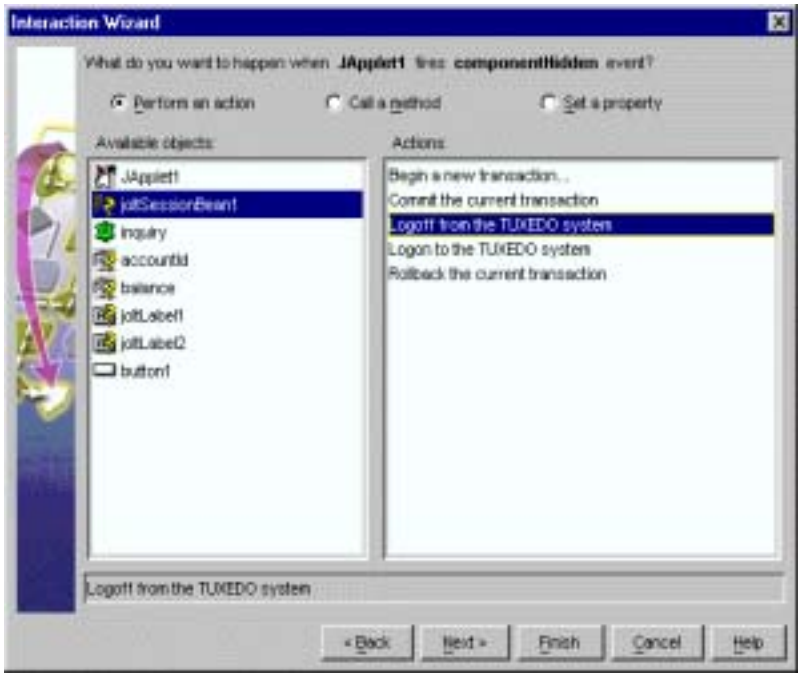
4. [Next] をクリックします。

[Interaction Wizard] ウィンドウが表示されます (6-48 ページの「オブジェクトとして [joltSessionBean1]、アクションとして [Logoff from the TUXEDO system] を選択する」を参照)。

「What do you want to happen when JApplet1 fires componentHidden event?」というメッセージが表示されます。

5. アクションを実行するオブジェクトとして [joltSessionBean1] を選択します。
6. アクションとして [Logoff from the TUXEDO system] を選択します (次の図を参照)。

図 6-27 オブジェクトとして [joltSessionBean1]、アクションとして [Logoff from the TUXEDO system] を選択する



7. [Finish] をクリックします。

「手順 6: JoltSessionBean とログオフを関連付ける」を終了すると、アプレットが非表示になったときに送信されるイベント (例 : componentHidden) が、JoltSessionBean の `logoff()` メソッドをトリガできるようになります。

手順 7: アプレットをコンパイルする

JoltBeans を関連付けたら、アプレットをコンパイルします。例外処理をキャッチ・ブロックに設定します。メッセージ・ウィンドウに、コンパイル・エラーや例外がないかどうかを確認します。

詳細については、次の「Jolt リポジトリの使用とプロパティ値の設定」、6-50 ページの「JoltBean 固有のプロパティ」、および 6-51 ページの「JoltServiceBean のプロパティ・エディタ」を参照してください。

サンプル・アプリケーションの実行

サンプル・アプリケーションを実行するには、BEA Tuxedo サーバが実行されていなければなりません。次に、口座番号を [Account ID] フィールドに入力します。BANKAPP データベース内にある口座番号であれば、どの口座番号を使用してもかまいません。次の 2 つの口座番号は、サンプル・アプリケーションで使用できる口座番号の例です。

- 80001
- 50050

Jolt リポジトリの使用とプロパティ値の設定

カスタム・プロパティ・エディタは以下のプロパティのために用意されています

- JoltFieldName (Jolt 対応 AWT beans)

- serviceName (JoltServiceBean)

プロパティ・リストからアクセスされるプロパティ・エディタには、プロパティを追加または変更するときに使用するダイアログ・ボックスが含まれています。対応するプロパティ値の隣にある [...] ボタンを選択することにより、このダイアログ・ボックスをプロパティ・リストから呼び出すことができます。

JoltBean のなかには、プロパティ・リスト・フィールドへの設定を必要とするものもあります。このような JoltBean を次の表にまとめて示します。

表 6-6JoltBean 固有のプロパティ

JoltBean	プロパティ	入力の説明
JoltSessionBean	appAddress	例： `//host:port`
	userName、Password または AppPassword	BEA Tuxedo ユーザ名とパスワードを入力します。
JoltServiceBean	serviceName	たとえば、INQUIRY を指定します。
	isTransactional	トランザクションの処理中にサービスを実行する必要がある場合は `true` に設定します。サービスがトランザクションを必要としない場合は、isTransactional を `false` に設定します。
JoltUserEventBean	eventName	BEA Tuxedo `tpsubscribe` コールを参照してください。
	filter	
すべての Jolt 対応 GUI Bean	joltFieldName	たとえば、ACCOUNT_ID を指定します。
	occurrenceIndex	同じ名前の複数のフィールド。インデックスは 0 から始まります。
JoltCheckbox	TrueValue と FalseValue	チェックボックスの状態に対応するフィールド値。

プロパティ・エディタは、リポジトリからキャッシュされた情報を読み取り、使用可能なサービス名とリストボックスのデータ要素を返します。ServiceName のプロパティ・エディタの例を次の「JoltServiceBean のプロパティ・エディタ」に示します。

Bean のプロパティを追加するには、以下の手順に従います。

1. 次の図の ServiceName フィールドの [...] ボタンをクリックしてサービス名を選択します。

図 6-28JoltServiceBean のプロパティ・エディタ

次の図に示すような ServiceName のカスタム・プロパティ・エディタが表示されます。

図 6-29ServiceName のカスタム・プロパティ・エディタ

注記　Repository データベースに接続できない、または接続したくない場合は、テキスト・ボックスにサービス名を入力し、手順 7 に進んでください。

2. ログオンしていない場合は、[Logon] を選択します。

   次の図に示すような [JoltBeans Repository Logon] ウィンドウが表示されます。

図 6-30JoltBeans Repository Logon ウィンドウ

3. Server フィールドに、BEA Tuxedo または Jolt リレー・マシンの名前を入力し、Port number フィールドに JSL または Jolt リレーを入力します。

4. パスワード、およびユーザ名情報 (必要な場合) を入力して [Logon] をクリックします。

   カスタム・プロパティ・エディタがリポジトリからキャッシュをロードします (「プロパティ・エディタと選択されたサービス」を参照)。

5. に示すリスト・ボックス内の該当するサービス名を選択します。

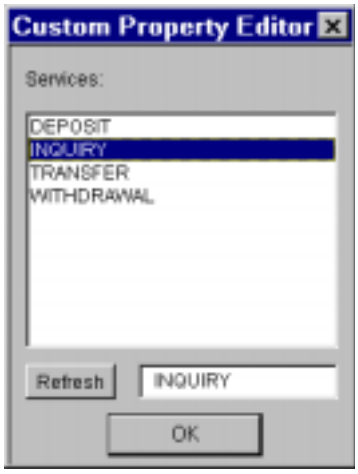
6. プロパティ値 (サービス名またはフィールド名) を直接入力します。

   用意されているテキストボックスに入力してください。

7. カスタム・プロパティ・エディタ・ダイアログ・ボックスの [OK] をクリックします。

   テキストボックスに入力した内容が Bean のプロパティに設定されます。

図 6-31 プロパティ・エディタと選択されたサービス



8. カスタム・プロパティ・エディタ・ダイアログ・ボックスの [OK] をクリックします。

JoltBeans のプログラミング

以下の手順が追加されています。

- JoltBeans によるトランザクションの使用
- JoltServiceBean でカスタム GUI コンポーネントを使用する

JoltBeans によるトランザクションの使用

BEA Tuxedo アプリケーション・サービスが、データベースを更新する機能を持つ場合があります。この場合は、JoltBeans でトランザクションを使用できます。たとえば BANKAPP では、TRANSFER と WITHDRAWAL のサービスが BANKAPP のデータベースを更新します。アプリケーション・サービスが読み取り専用 (例 : INQUIRY) の場合には、トランザクションを使用する必要はありません。

以下は、JoltBeans でトランザクションを使用する例です。

1. isJoltServiceBean クラスの `setTransactional(true)` メソッドを呼び出します。isTransactional は JoltServiceBean の boolean 型のプロパティです。
2. JoltSessionBean クラスの `beginTransaction()` メソッドを呼び出します。
3. JoltServiceBean クラスの `callService()` メソッドを呼び出します。
4. サービス・コールの結果により、JoltSessionBean クラスの `commitTransaction()` メソッドまたは `rollbackTransaction()` メソッドを呼び出します。

JoltServiceBean でカスタム GUI コンポーネントを使用する

JoltBeans には、Jolt 対応の限定された GUI コンポーネントのセットが用意されています。また、Jolt に対応していないコントロールを JoltServiceBean と一緒に使用することもできます。JoltServiceBean にはコントロール (JoltServiceBean によって表されるサービスの出力情報を表示) をリンクすることができます。また、入力情報を表示するコントロールをリンクすることもできます。

たとえば、GUI コンポーネントは JoltOutputListener インターフェイスを実装するアダプタ・クラスを使用して、JoltOutputEvent をリッスンできます。JoltServiceBean は JoltOutputEvent のイベント・ソースとして、JoltOutputEvent を送信するときにアダプタ・クラスの `serviceReturned()` メソッドを呼び出します。`serviceReturned()` の内部では、イベント・オブジェクトからの情報を使用して、コントロールの内部データが更新されます。

開発ツールは、JoltServiceBean と GUI コンポーネントが関連付けられるとアダプタ・クラスを生成します。

2 つ目の例として、GUI コンポーネントは JoltServiceBean の `setInputTextValue()` メソッドを呼び出すことができます。GUI コンポーネントには JoltServiceBean によって表される BEA Tuxedo サービスへの入力データが含まれています。

3 つ目の例として、GUI コンポーネントは必要なメソッド (`addJoltInputListener()` と `removeJoltInputListener()`) を実装でき、JoltInputEvent のイベント・ソースとしての役割を果たします。JoltServiceBean は、これらのイベントのイベント・リスナとしての役割を果たします。コントロールは、状態が変わる場合に JoltInputEvent を送信して、入力情報で JoltServiceBean を更新します。

7 Servlet Connectivity for BEA Tuxedo を使う

BEA Jolt のサーブレット・コネクティビティの機能を利用すると、HTTP リクエストに対する処理を HTTP サーブレットを使用して行い、サーバ・サイドの Java タスクを実行することができます。Jolt では、Java Web Server バージョン 1.1.3 以上のサーブレット・コネクティビティの機能がサポートされており、標準的なサーブレット・エンジンがサポートされています。Jolt のセッション・プール・クラスを使用すると、単純な HTML クライアントからも一般的なサーブレットをサポートする Web サーバに接続することができます。これで、すべての Jolt トランザクションは、クライアント側のアプレットやアプリケーションの代わりに、Web サーバ側のサーブレットで処理されます。

サーブレット機能により、HTML クライアントは BEA Tuxedo に直接接続せずに BEA Tuxedo サービスを呼び出すことができます。HTML クライアントは、HTTP プロトコルを使用して Web サーバ (BEA Tuxedo サービスに対するリクエストをサーブレットを使って処理するサーバ) に接続します。Web サーバ上のサーブレットは、Jolt セッションを使用して BEA Tuxedo サービスに対するリクエストを管理します。つまり、Jolt サーバ・ハンドラ (JSH: Jolt Server Handler) または Jolt サーバ・リスナ (JSL: Jolt Server Listener) を介して BEA Tuxedo サーバに接続し、BEA Tuxedo サービスに対するリクエストを作成します。

この機能により、さまざまな HTML クライアントがリモートの BEA Tuxedo サービスのリクエストを作成できるようになります。これで、すべての Jolt トランザクションはサーバ側で処理されます。元の HTML クライアントを変更する必要はありません。したがって、単純な構造の HTML クライアントで処理することが可能になり、HTML クライアントの管理も容易になります。

ここでは、次の内容について説明します。

- サーブレットとは
- サーブレットと Jolt の関係
- HTTP サーブレットの作成と登録
- Jolt サーブレット・コネクティビティのサンプル
- サーブレットに関するその他の情報

サーブレットとは

サーブレットとは、クライアントの代わりにサーバ上で呼び出され、実行される Java クラスのことです。アプレットがクライアント側で実行されるのに対し、サーブレットはサーバ側で実行されます。HTTP サーブレットは、受け取った HTTP リクエストを処理し、それに対して HTTP 応答を送信する Java クラスです。HTTP サーブレットは HTTP サーバに常駐しており、サーブレット・クラス JavaSoft javax.servlet.http.Http の拡張である必要があります。これで、サーブレットは、一般的なサーブレット・エンジンのフレームワークで動作します。

HTTP サーブレットには、次のような利点があります。

- 形式の整ったコンパイル済み言語 (Java) で記述されているため、「変換して使用する」スクリプトより強力である。
- 常駐する HTTP サーバの主要な部分を構成している。

- セキュリティに関して安全性が低い CGI とは異なり、常駐するサーバの堅牢なセキュリティ機能によって保護されている。
- うまく開発されたプログラム・インターフェイスを使用して HTTP リクエストを処理するため、プログラムの作成を簡単に行うことができ、エラーの発生を抑えることができる。

サーブレットと Jolt の関係

Jolt のサーブレット・コネクティビティ機能を使用すると、一般的な HTTP サーブレットで Jolt 機能を利用することができます。Jolt サーブレットは、次の Jolt クラスを使用して HTTP リクエストを処理します。

- ServletDataSet
- ServletPoolManagerConfig
- ServletResult
- ServletSessionPool
- ServletSessionPoolManager

Jolt サーブレット・コネクティビティ・クラス

以下は、Jolt サーブレット・コネクティビティ・クラスの説明です。

ServletDataSet

このクラスには、BEA Tuxedo サービスの入力パラメータと出力パラメータを表すデータ・エレメントが格納されています。HTML のフィールド名と値を `javax.servlet.http.HttpServletRequest` オブジェクトからインポートするメソッドが用意されています。

ServletPoolManagerConfig

このクラスは、Jolt セッション・プール・マネージャと、関連する 1 つまたは複数の Jolt セッション・プール用のスタートアップ・クラスです。必要に応じてセッション・プールを作成し、最低限必要な数のセッションを設定してセッション・プールを開始します。Jolt セッション・プール・マネージャは、1 つまたは複数の名前付きセッション・プールを内部的にトラッキングします。

このクラスは `bea.jolt.pool.PoolManagerConfig` から派生しています。呼び出し側は Properties オブジェクトまたは Hashtable オブジェクトを静的な `startup()` メソッドに渡してセッション・プールを作成したり、静的な `getSessionPoolManager()` メソッドを渡して `bea.jolt.pool.servlet.ServletSessionPoolManager` クラスのセッション・プール・マネージャを取得することができます。

ServletResult

このクラスには、ServletResult オブジェクトの各フィールドを String 型で取得するメソッドが用意されています。

ServletSessionPool

このクラスには、Java サーブレットで使用するセッション・プールが用意されています。セッション・プールは、BEA Tuxedo システムに対する 1 つまたは複数の接続 (セッション) を表します。このクラスには、BEA Tuxedo サービスの入力パラメータを `javax.servlet.http.HttpServletRequest` オブジェクトとして受け取る呼び出しメソッドが用意されています。

ServletSessionPoolManager

このクラスは、サーブレットに固有なセッション・プール・マネージャです。`ServletSessionPool` クラスの 1 つまたは複数のセッション・プールの集まりを管理します。このクラスには、ServletSessionPoolManager クラスと、このクラスに含まれるセッション・プールを作成するためのメソッドが用意されています。これらのメソッドは、セッション・プール用の管理 API の一部です。

HTTP サーブレットの作成と登録

HTTP サーブレットを作成と登録を行う前に、まず Jolt サーブレット・コネクティビティをサポートするパッケージ (`jolt.jar`、`joltjse.jar`、`servlet.jar`) をインポートする必要があります。HTTP サーブレットは、javax.servlet.http.HttpServlet の拡張です。作成した HTTP サーブレットは、一般的なサーブレットをサポートする Web サーバに登録します。カスタマイズしたサーブレットは、HTTP 機能を持つ標準の HTTP サーブレットと同様に扱われます。

各 HTTP サーブレットには、特定の URL の形式が登録されます。ある URL が要求されると、対応するサーブレットが呼び出され、要求が処理されます。

サーブレットの登録方法については、お使いの Web サーバのマニュアルを参照してください。

Jolt サーブレット・コネクティビティのサンプル

Jolt ソフトウェアには、Jolt サーブレット・クラスを使用してサーブレット・コネクティビティを解説するサンプル・アプリケーションが用意されています。次の 3 つのサンプルが用意されています。

- サンプル・アプリケーション「SimpApp」
- サンプル・アプリケーション「BankApp」
- サンプル・アプリケーション「Admin」

Jolt サーブレット・クラスの使用方法を学ぶには、これらのサンプルで示されているコード例を参照してください。

サーブレット・アプリケーションのサンプルを表示するには

Jolt サンプル・アプリケーションのコードを表示するには、Jolt API クライアント・クラス (通常は、Jolt のインストール時のオプション項目) をインストールする必要があります。指定したディレクトリに Jolt API クライアント・クラスをインストールしたら、次のディレクトリにあるサンプル・アプリケーション・ファイルを選択してください。

```
<Installation directory>\udataobj\jolt\examples\servlet
```

サンプル・コードを表示するには、Microsoft のメモ帳などのテキスト・エディタを使って、サンプル・アプリケーションごとに Java ファイルを開いてください。

サンプル・アプリケーション「SimpApp」

Jolt には、SimpApp というサンプル・アプリケーションがあります。SimpApp は、サーブレットがどのように Servlet Connectivity for BEA Tuxedo を使用するかを示しています。SimpApp では、以下のサーブレットのタスクが説明されています。

- プロパティ・ファイルを使ってセッション・プールを作成する
- セッション・プール・マネージャを取得する
- 名前を使ってセッション・プールを取得する
- BEA Tuxedo サービスを呼び出す
- 結果セットを処理する

このサンプルでは、サーブレットが BEA Tuxedo に接続し、BEA Tuxedo サービスを呼び出す方法を解説しています。まず、simpapp.html ファイルを呼び出します。サーブレットは初期化時にセッション・プール・マネージャを作成します。作成されたセッション・プール・マネージャは、doPost() メソッドが呼び出されたときに、セッションを取得するために用いられます。このセッションは、ポストされた引数「SVCNAME」で指定された BEA Tuxedo サービスに接続するために使用されます。このサンプルの場合は「TOUPPER」サービスです。TOUPPER サービスは、引数「STRING」に指定された文字列を大文字に変換し、結果を返します。結果は、クライアントのブラウザ内に別画面で表示されます。

注記　このサンプルでは Weblogics Server が使用されます。

SimpApp の実行に必要な条件

SimpApp の実行に必要な条件は、以下のとおりです。

- Servlet JSDK 1.1 以上がインストールされた Web アプリケーション・サーバ
- SimpApp を実行中の BEA Tuxedo 8.0 以上
- BEA Jolt

SimpApp のインストール

1. Web アプリケーション・サーバに Jolt クラス・ライブラリ (`jolt.jar`) と Servlet Connectivity for BEA Tuxedo クラス・ライブラリ (`joltjse.jar`) をインストールします。Web アプリケーション・サーバによっては、必要に応じてクラス・ファイルを解凍します。

2. `SimpAppServlet.java` をコンパイルします。標準の `JDK 1.1.x classes.zip`、JSDK 1.1 クラス、Jolt クラス・ライブラリ、および Servlet Connectivity for BEA Tuxedo クラス・ライブラリが classpath に含まれていることを確認してください。

```
javac -classpath
$(JAVA_HOME)/lib/classes.zip:$(JSDK)/lib/servlet.jar:
      $(JOLTHOME)/jolt.jar:$(JOLTHOME)/joltjse.jar:./classes
            -d ./classes SimpAppServlet.java
```

   注記　SimpAppServlet のパッケージ名は `examples.jolt.servlet.simpapp` です。

3. `simpapp.html` と `simpapp.properties` の 2 ファイルを public HTML ディレクトリに置きます。

4. `simpapp.properties` ファイルを変更します。「`appaddrlist`」と「`failoverlist`」を正しい Jolt サーバのホストとポートに変更します。SimpApp でセキュリティ機能が有効になっている場合は、正しい BEA Tuxedo 認証情報を指定します。次に例を示します。

```
#simpapp
#Fri Apr 16 00:43:30 PDT 1999
poolname=simpapp
appaddrlist=//host:7000,//host:8000
failoverlist=//backup:9000
minpoolsize=1
maxpoolsize=3
userrole=tester
```

```
apppassword=appPass

username=guest

userpassword=myPass
```

5. SimpAppServlet に「Simpapp」を登録します。詳細については、お使いの Web アプリケーション・サーバを参照してください。BEA WebLogic ユーザの場合は、`config.xml` ファイルの次のセクションを追加します。

```
<Application
   Deployed="true"
   Name="simpapp"
   Path=".\config\mydomain\applications"
>
   <WebAppComponent
     Name="simpapp"
     Targets="myserver"
     URI="simpapp"
   />
</Application>
```

6. 次のように入力して、SimpApp の最初のページ「`simpapp.html`」にアクセスします。

```
http://mywebserver:8080/simpapp.html
```

サンプル・アプリケーション「BankApp」

BankApp アプリケーションは、サーブレットが PageCompiledServlet を使い、Servlet Connectivity for BEA Tuxedo を利用してどのように作成されるかを示しています。BankApp では以下の方法を説明します。

- プロパティ・ファイルを使ってセッション・プールを作成する
- セッション・プール・マネージャを取得する
- 名前を使ってセッション・プールを取得する
- BEA Tuxedo サービスを呼び出す
- 結果セットを処理する

BankApp の実行に必要な条件

BankApp の実行に必要な条件は、以下のとおりです。

- Servlet JSDK 1.1 以上がインストールされた Web アプリケーション・サーバ
- BankApp を実行中の BEA Tuxedo 8.0 以上
- BEA Jolt

Admin のインストール

1. Web アプリケーション・サーバに Jolt クラス・ライブラリ (jolt.jar) と Servlet Connectivity for Tuxedo クラス・ライブラリ (joltjse.jar) をインストールします。Web アプリケーション・サーバによっては、必要に応じてクラス・ファイルを解凍します。
2. Web アプリケーション・サーバの public HTML ディレクトリに、HTML、JHTML、bankapp.properties ファイルをすべてコピーします (WebLogic の場合は $WEBLOGIC/myserver/public_html)。

```
bankapp.properties
```

```
tellerForm.html
inquiryForm.html
depositForm.html
withdrawalForm.html
transferForm.html
InquiryServlet.jhtml
DepositServlet.jhtml
WithdrawalServlet.jhtml
TransferServlet.jhtml
```

3. bankapp.properties ファイルを変更します。「appaddrlist」と「failoverlist」を正しい Jolt サーバのホストとポートに変更します。BankApp でセキュリティ機能が有効になっている場合は、正しい BEA Tuxedo 認証情報を指定します。次に例を示します。

```
#bankapp
#Fri Apr 16 00:43:30 PDT 1999
poolname=bankapp
appaddrlist=//host:8000,//host:7000
failoverlist=//backup:9000
minpoolsize=2
maxpoolsize=10
userrole=teller
apppassword=appPass
username=JaneDoe
userpassword=myPass
```

4. 必要に応じて、お使いのサーブレット・エンジンから JHTML の自動コンパイル機能を有効にしておきます。詳細については、お使いの Web アプリケーション・サーバのマニュアルを参照してください。

5. お使いのブラウザで次の URL を指定し、Servlet Connectivity for BEA Tuxedo を使って BankApp にアクセスしてください。

   ```
   http://mywebserver:8080/tellerForm.html
   ```

サンプル・アプリケーション「Admin」

「Admin」サンプル・アプリケーションでは、以下のサーブレットのタスクを説明します。

- 管理用 API を使用してセッション・プールを管理する
- Servlet Connectivity for BEA Tuxedo の PageCompiledServlet を使用して統計情報を取得する

Admin の実行に必要な条件

Admin の実行に必要な条件は、以下のとおりです。

- Servlet JSDK 1.1 以上がインストールされた Web アプリケーション・サーバ
- BEA Jolt

Admin のインストール

1. Web アプリケーション・サーバに Jolt クラス・ライブラリと Servlet Connectivity for BEA Tuxedo クラス・ライブラリをインストールします。
2. すべての JHTML ファイルを public HTML ディレクトリ (WebLogic の場合は `$WEBLOGIC/myserver/public_html for WebLogic`) にコピーします。

   ```
   PoolList.jhtml
   ```

   ```
   PoolAdmin.jhtml
   ```

3. セッション・プールの一覧を表示するには、お使いのブラウザで次の URL を指定してください。

   ```
   http://mywebserver:8080/PoolList.jhtml
   ```

サーブレットに関するその他の情報

サーブレットの作成方法と使用方法の詳細については、以下の Web サイトを参照してください。

BEA WebLogic サーブレットに関するマニュアル

`http://edocs.beasys.co.jp/e-docs/wls60e/adminguide/index.html`

`http://edocs.beasys.co.jp/e-docs/wls60e/servlet/index.html`

`http://e-docs.bea.com/wls/docs60/javadocs/index.html`

Java サーブレット

`http://jserv.java.sun.com/products/java-server/documentation/webserver1.1/index_developer.html`

サーブレット関連団体

`http://servlet-interest@java.sun.com`

8 Jolt ASP Connectivity for BEA Tuxedo を使う

Jolt ASP (Active Server Pages) Connectivity for BEA Tuxedo は、動的な HTML ページを処理したり、作成したりするための使いやすいインターフェイスを提供します。BEA Tuxedo サービスにアクセスするために Common Gateway Interface (CGI) トランザクション・プログラムの書き方を学ぶ必要はありません。

ここでは、次の内容について説明します。

- 主な特徴
- Jolt ASP Connectivity for BEA Tuxedo の仕組み
- ASP Connectivity for BEA Tuxedo ツールキット
- Jolt ASP Connectivity for BEA Tuxedo の使い方
- ASP for BEA Tuxedo の使い方についての概要
- 準備チェックリスト
- TRANSFER サービスの概要
- TRANSFER リクエストの呼び出し方

主な特徴

Jolt ASP Connectivity for BEA Tuxedo は、Jolt クラス・ライブラリを拡張したものです。また、スクリプト言語を使って BEA Tuxedo のサービスやトランザクションを Web サーバから呼び出すことができます。

このアーキテクチャには、以下のような利点があります。

- HTML インターフェイスをそのまま利用できます。
- Java クラス・ファイルをダウンロードする必要がなく、ダウンロードにかかる時間も節約できます。
- セッション・プール機能を使用して、BEA Tuxedo のリソースを効率的に利用できます。
- Jolt ASP Connectivity for Tuxedo では、暗号化の機能を持つ業界標準の HTTP プロトコル、および Web サーバ用のファイアウォール・コンフィギュレーションを活用できます。

注記　ASP Connectivity for BEA Tuxedo では、非同期通知は利用できません。非同期通知をサポートしたい場合は、RETAINED モードの接続を使用する Jolt 対応 Java クライアント (アプレット) を開発することをお勧めします。

Jolt ASP Connectivity for BEA Tuxedo の仕組み

Jolt ASP Connectivity for BEA Tuxedo のアーキテクチャには、セッション、セッション・プール、セッション・プール・マネージャという 3 つの主要なコンポーネントがあります。セッション・オブジェクトは、BEA Tuxedo シ

ステムとの接続を表します。セッション・プールは、Web サーバと BEA Tuxedo システム間の物理的な接続を表します。また、セッション・プールは、セッションと HTTP リクエストを関連付けます。

セッション・プール・マネージャは、セッション・オブジェクトを管理します。各セッション・オブジェクトは、一意なセッション識別子を持ちます。

Jolt ASP Connectivity for Tuxedo は以下のように動作します。

1. Web アプリケーションが初期化されていない場合、Web アプリケーションはセッション・プール・マネージャを初期化し、セッション・プールを生成して、Jolt サーバとのセッション (接続) を確立します。
2. Web アプリケーションは、サービス要求を受け取ると、セッション・プール・マネージャからセッション・プール・オブジェクトを取得します。セッション・プールは、未処理の呼び出し要求の数が最も少ない (least busy) セッションを用いてサービス呼び出しを実行します。
3. 選択したセッションが Jolt サーバによって切断された場合、セッション・プール・オブジェクトは新しいセッションを再び開始するか、またはほかのセッションにリクエストを渡します。セッション・プール・マネージャがセッションを獲得できない場合は、セッション・オブジェクトとしてヌルが返されます。

次の図に、ASP Connectivity for BEA Tuxedo のアーキテクチャを示します。

図 8-1Jolt ASP Connectivity for BEA Tuxedo のアーキテクチャ

`SessionPool` および `SessionPoolManager` クラスの詳細については、オンライン・マニュアルの『BEA Jolt API リファレンス』を参照してください。

ASP Connectivity for BEA Tuxedo ツールキット

ASP Connectivity for BEA Tuxedo ツールキットは、Jolt の Java クラス・ライブラリを拡張したものです。このツールキットにより、Microsoft Active Server などの Web サーバで Jolt クライアント・クラス・ライブラリが使用できるようになります。また、HTML のクライアントまたはブラウザと BEA Tuxedo アプリケーションをつなぐインターフェイスが提供されます。

このソフトウェアに付属するサンプルは、INQUIRY、WITHDRAWAL、DEPOSIT、TRANSFER の 4 つのサービスをサポートします。この節では、BEA Tuxedo BankApp アプリケーションの TRANSFER サービスを利用する場合の HTML クライアント・インターフェイスの使用手順を説明します。TRANSFER サービスは、複数のオカレンスを持つパラメータの使い方を示します。この章では、TRANSFER サービスの使い方のみを説明します。

Jolt ASP Connectivity for BEA Tuxedo の使い方

この章で説明するすべてのサンプルは、Jolt ソフトウェアに含まれています。この節では、これらのサンプルのコードの一部を使用してツールキットの使い方を説明します。

このソフトウェアに付属するサンプルは、INQUIRY、WITHDRAWAL、DEPOSIT、TRANSFER の 4 つのサービスをサポートします。この章では、BEA Tuxedo BankApp アプリケーションの TRANSFER サービスを利用する場合の HTML クライアント・インターフェイスの使用手順を説明します。TRANSFER サービスは、複数のオカレンスを持つパラメータの使い方を示します。この章では、TRANSFER サービスの使い方のみを説明します。

注記　この章では、Microsoft IIS および VBScript と共に ASP Connectivity for BEA Tuxedo を使用する方法について説明します。

以降の節で示す情報を活用するためには、以下の知識が必要です。

- BEA Tuxedo および BEA Tuxedo のサンプル・アプリケーションの BankApp
- BEA Jolt
- ハイパーテキスト・マークアップ言語 (HTML)
- VB (Visual Basic) Script
- オブジェクト指向プログラミングの概念

ASP for BEA Tuxedo の使い方についての概要

ASP Connectivity for BEA Tuxedo の使い方を学ぶには、以下の手順に従ってください。

1. 準備チェックリストの内容を確認します。
2. TRANSFER サービスの概要の内容を確認します。
3. TRANSFER リクエストの呼び出し方の手順に従います。
   - Jolt セッション・プール・マネージャを初期化する
   - クライアントから TRANSFER リクエストを送信する
   - リクエストを処理する
   - クライアントへ結果を返す

準備チェックリスト

「TRANSFER リクエストの呼び出し方」を始める前に、このチェックリストの内容を確認します。

注記　この準備チェックリストは、Microsoft Active Server Pages 用です。また、BEA Tuxedo と Jolt Server がホスト・マシンにインストールされていること、および Microsoft IIS がインストールされているマシンに BEA Jolt クライアントがインストールされていることが前提となっています。

Jolt サーバを実行する Tuxedo ホスト

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サポートされているかどうかを確認します。クライアント・マシンは、BEA Tuxedo 環境へ接続される Web サーバにネットワーク接続されている必要があります。
2. BEA Tuxedo と BEA Tuxedo のサンプル例である BankApp を構成し、ブートします。
   a. TRANSFER サービスが使用可能であることを確認します。
   b. この作業を完了するための情報については、BEA Tuxedo のマニュアルを参照してください。
3. Jolt を起動するように BEA Tuxedo アプリケーションを設定します。Jolt サーバの構成方法については 『BEA Tuxedo システムのインストール』を参照してください。
   a. 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。
   b. TRANSFER サービスが Jolt リポジトリに定義されているかどうかを確認します。

c. Jolt リポジトリ・エディタを使用して、TRANSFER サービスがアクセス可能かどうかをテストします。次に、Jolt リポジトリ・エディタを使用して TRANSFER サービスをエクスポートし、TRANSFER サービスが Jolt クライアントからアクセス可能かどうかをテストします。

Jolt クライアントと Microsoft IIS を実行するマシン

1. Microsoft IIS SDK の Microsoft Java Component Framework がインストールされていない場合は、ここでインストールします。

   Windows NT 4.0 プラットフォームでは、Microsoft NT Option Pack 4.0 の標準のインストールを選択した場合、この SDK はインストールされません。インストーラで [SHOW SUBCOMPONENTS] オプションを選択して、IIS の追加のオプションを表示します。SDK を参照してインストールします。

   Windows NT または 2000 では、Microsoft SDK for Java の Microsoft Java Component Framework クラスを入手することもできます。Microsoft Java Component Framework ファイルをダウンロードするには、次の手順に従います。

   a. Web ブラウザを開いて Microsoft の Web サイトにアクセスします。

      `http://www.microsoft.com`

      「Microsoft SDK for Java」という文字列を検索します。

      または、次のダウンロード・ページに直接移動します。

      `http://www.microsoft.com/java/download/dl_sdk40.htm`

   b. Microsoft SDK for Java 4.0 for Windows をダウンロードして、マシンにインストールします。

2. `java\Trustlib` ディレクトリに新しいディレクトリ `aspcomp` を作成します。`Trustlib` ディレクトリは通常 `%windir%\java\TrustLib` にあります。Framework ファイルはパッケージ `aspcomp` に作成されるので、Java 仮想マシン (JVM) はこのディレクトリを検索します。

3. Microsoft Java Component Framework クラス・ファイルを java\TrustLib\aspcomp ディレクトリにコピーします。*.class ファイルのみをコピーします。次に例を示します。

```
 ...\Program Files\Microsoft SDK for Java 4.0\Samples\ASP\aspcomp> copy *.class
C:\WINNT\java\TrustLib\aspcomp
```

4. jolt.jar ファイルを %windir%\java\Trustlib ディレクトリにコピーします。次に、jar コマンドを使って jolt.jar をアンパックし、Jolt クラス・ファイルを java\TrustLib ディレクトリに作成します。Jolt クラスは名前が BEA で始まるパッケージに作成されるので、JVM は java\Trustlib\bea サブディレクトリを検索します。

5. joltasp.jar ファイルを %windir%\java\Trustlib ディレクトリにコピーします。次に、jar コマンドを使って joltasp.jar をアンパックし、Jolt Pool ASP クラス・ファイルを java\Trustlib ディレクトリに作成します。Jolt Pool ASP クラスは名前が BEA で始まるパッケージに作成されるので、JVM は java\Trustlib\bea\ サブディレクトリを検索します。

6. Jolt のインストール・ディレクトリから wasreg.cmd ファイルを取得します。wasreg.cmd を実行して、BEA Jolt ASP クラスを BEAJOLTPOOL ActiveX コンポーネントとして登録します。この操作によって、BEAJOLTPOOL コンポーネントが Microsoft ASP スクリプトからアクセス可能になります。Java クラスの ActiveX コンポーネントとしての登録を解除するには、BEA Jolt で用意されている wasunreg.cmd スクリプトを実行します。

   javareg/register が実行するのはレジストリ・エントリの作成だけなので、登録は手動またはセットアップ・プログラムでも行えます。

   wasreg.cmd および wasunreg.cmd スクリプトは JavaReg ユーティリティを使用します。このユーティリティは Visual J++ または Microsoft SDK for Java で利用できます。

7. ASP クラスを ActiveX コンポーネントとして登録したら、Jolt ASP Connectivity for BEA Tuxedo ソフトウェアに付属のサンプル・アプリケーションをテストできます。

   a. %tuxdir%\udataobj\jolt\examples\asp\bankapp ディレクトリをデフォルトの Microsoft IIS ディレクトリにコピーします。

b. 実行環境に合わせて `bankapp.properties` ファイルを編集します。

c. ブラウザから次の URL を入力して、アプリケーションを起動します。

```
http://web-server:port/bankapp/tellerForm.asp
```

注記　ポート番号はオプションです。ポートを指定するかどうかは、Web サーバのコンフィギュレーションによって異なります。ほとんどの場合、この URL にポート番号を含める必要はありません。

8. 次の表は、このサンプル・アプリケーションのファイルの一覧です。これらのファイルは「TRANSFER リクエストの呼び出し方」で参照する重要なファイルです。Jolt サンプルのディレクトリ (*TUXDIR*\udataobj\jolt\examples\asp) に格納されています。

表 8-1BankApp のサンプル・ソース・ファイル

ファイル名	説明
tellerForm.asp	Jolt セッション・プール・マネージャを初期化し、使用可能な BankApp サービスを表示します。
transferForm.htm	ユーザ入力用の HTML フォームを表示します。
tlr.asp	HTML フォームを処理し、結果を HTML ページとして返します。
web_admin.inc	Jolt セッション・プール・マネージャを初期化するための VBScript 関数
web_start.inc	
web_templates.inc	HTML テンプレートをキャッシュするための VBScript 関数
templates/*.temp	結果を返すために使用される HTML テンプレート

TRANSFER サービスの概要

BankApp の TRANSFER サービスは、2 つの口座の間で資金を移動します。このサービスは 2 つの口座番号と入力金額をとり、2 つの残高 (各口座に 1 つ) を返します。さらに、アプリケーションまたはシステムのエラーが発生した場合は、エラー・メッセージを返します。

TRANSFER は、WITHDRAWAL と DEPOSIT を単一のトランザクションとして実行します。このトランザクションはサーバ上で生成されるため、クライアントはトランザクションを生成する必要はありません。

クライアント・インターフェイスは、必要なデータ (口座番号とドル金額) を入力するフォームが表示された HTML ページです。このデータは、POST リクエストとして Web サーバに送信されます。

Web サーバでは、このリクエストは VBScript Active Server Page を使用して処理されます。このプログラムは、リクエストから入力データ・フィールドを抽出し、Jolt ASP Connectivity for BEA Tuxedo クラス・ライブラリで使用できるようにフォーマットし、BankApp アプリケーションの TRANSFER

サービスにリクエストを送信します。TRANSFER サービスは、トランザクションの結果を返します。結果はプログラムに返され、動的に作成された HTML ページにマージされます。このページは、Web サーバのインフラストラクチャを介してクライアントに返されます。

この章を読み終えたら、必要な HTML ページとサーバ・サイド VBScript ロジックを実行して TRANSFER を実行してみてください。

TRANSFER リクエストの呼び出し方

この節では、TRANSFER リクエストの実行時の状況について説明します。ただし、必要な手順のみを説明しており、すべての手順は説明していません。

- Jolt セッション・プール・マネージャを初期化する
- クライアントから TRANSFER リクエストを送信する
- リクエストを処理する
- クライアントへ結果を返す

Jolt セッション・プール・マネージャを初期化する

まず、クライアントのブラウザを使用して、Jolt ASP Connectivity for BEA Tuxedo のクラスがインストールされている Web サーバに接続します。ダウンロードする最初のページは、`tellerForm.asp` (次の図の `tellerForm.asp` ページの例を参照) です。8-7 ページの「準備チェックリスト」で、この teller サンプルがすでにインストールされている場合、このページの URL は次のようになっています。

`http://<web-server:port>/teller/tellerForm.asp`

注記　Web サーバの構成によっては、ポート番号の使用は必要ありません。ほとんどの場合、URL に「:port」を追加する必要はありません。

図 8-2tellerForm.asp ページの例

`tellerForm.asp` ページには、Jolt セッション・プール・マネージャを初期化する際に必要な VBScript プロシージャが含まれています。初期化コードは、ASP Script ブロックに入っています。このコードは、このコード・ブロックを、クライアントに送信するのではなくサーバ上で実行するよう Web サーバに命令します。

コード リスト 8-1 tellerForm.asp :Jolt **セッション・プール・マネージャを初期化する**

```
<%
'// テンプレートを初期化しテンプレートをキャッシュします。
Call web_initSessionMgr(Null)
Call web_cacheTemplates()
%>
```

VBScript プロシージャの `web_initSessionMgr()` は、Jolt セッション・プールを確立する別の VBScript プロシージャを呼び出します。この Jolt セッションは、Web サーバ内の Jolt ASP Connectivity for BEA Tuxedo と BEA Tuxedo アプリケーション内に常駐する Jolt サーバの間に確立されます。呼び出されるプロシージャの 1 つは、`web_start()` です。`web_start.inc` ファイルにあるこのプロシージャは、8-7 ページの「準備チェックリスト」の teller アプリケーションのインストールの際に編集されている必要があります。

`web_cacheTemplates()` プロシージャは、さまざまな HTML テンプレートをメモリ・キャッシュに読み込みます。この手順は必須ではありませんが、この操作によってパフォーマンスが向上します。

コード リスト 8-2 tellerForm.asp : **ユーザが** TRANSFER **サービスを選択できるようにする**

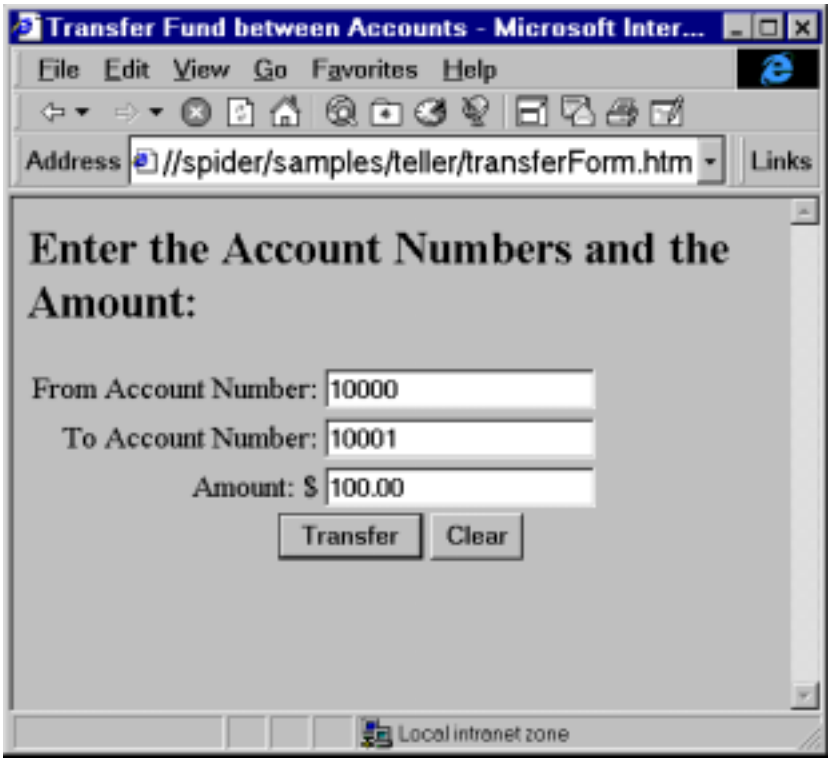
```
<INPUT TYPE="button" VALUE="Transfer"
    onClick="window.location='transferForm.htm'">
```

上記の HTML セグメントは、「Transfer」という名前のボタンを表示します。このボタンが選択されると、ブラウザは `transferForm.htm` ページをロードします。このページは、TRANSFER サービスで必要なデータを入力するためのフォームを表示します。

クライアントから TRANSFER リクエストを送信する

次の図のフォームは、transferForm.htm によって生成されます。このページは、入力用のフォームを表示します。このページは、3 つのテキスト・フィールド (2 つの口座番号の入力フィールドとドル金額の入力フィールド) と、TRANSFER サービスを呼び出すときに使用するボタンで構成されています。

図 8-3transferForm.htm の例



次のコード・セグメントは、このページの主要な HTML エレメントを示しています。次のコード例で強調表示されているエレメントは、8-16 ページの「主要な HTML エレメントとその説明」のエレメントに対応しています。

コード リスト 8-3　transferForm.htm:TRANSFER **フォーム**

```
<FORM NAME="teller" ACTION="tlr.asp" METHOD="POST">
<TABLE>
<TR><TD ALIGN=RIGHT>From Account Number: </TD>
    <TD><INPUT TYPE="text" NAME="ACCOUNT_ID_0"></TD></TR>
<TR><TD ALIGN=RIGHT>To Account Number: </TD>
    <TD><INPUT TYPE="text" NAME="ACCOUNT_ID_1"></TD></TR>
<TR><TD ALIGN=RIGHT>Amount: $</TD>
    <TD><INPUT TYPE="text" NAME="SAMOUNT"></TD></TR>
</TABLE>
<CENTER>
<INPUT TYPE="hidden" NAME="SVCNAME" VALUE="TRANSFER">
<INPUT TYPE="submit" VALUE="Transfer">
<INPUT TYPE="reset" VALUE="Clear">
</CENTER>
</FORM>
```

表 8-2 主要な HTML エレメントとその説明

エレメント	説明
ACTION=”tlr.asp&drq;	[submit] ボタンをクリックすると、このフォームの内容が Web サーバ上の tlr.asp というページに送信され、処理されます。
NAME=”ACCOUNT_ID_0”	複数のオカレンスを持つフィールドの使用を示します。TRANSFER サービスは、2 つの口座番号の入力を待ちます。両者とも「ACCOUNT_ID」というフィールド名です。フィールド名の末尾に 下線 (_) と *オカレンス番号* を付ける規則を使用し (例 : _0、_1)、フィールド名とオカレンスを Web サーバ上のプログラムに渡します。

表 8-2 主要な HTML エレメントとその説明

エレメント	説明
NAME="SAMOUNT"	単一のオカレンスを持つ入力フィールドの使用を示します。この例では、フィールド名の末尾には何も付きません。

この例で使用されている HTML フォームのフィールド名は、TRANSFER サービスで使用される BEA Tuxedo のフィールド名と完全に一致しています。これは必須ではありませんが、このように BEA Tuxedo と同じフィールド名を使用すると、入力項目を BEA Tuxedo フィールド名にマッピングする必要がなくなるため、サーバ上の処理が簡単になります。これは、Jolt ASP Connectivity for BEA Tuxedo クラスによって実行されます。

非表示の SVCNAME フィールドには、「TRANSFER」という値が指定されます。このフィールドはクライアント・フォームには表示されませんが、リクエストの一部として Web サーバに送信されます。VBScript プログラムは、どの BEA Tuxedo サービスを呼び出すか (この場合は TRANSFER サービス) を決定するために、このフィールドの値を取得します。

[From Account Number]、[To Account Number]、および [Amount] フィールドに入力します。BankApp では、10000 および 10001 が口座番号として有効です。[Transfer] ボタンを押します。フォームで入力されたデータは、Web サーバに送信され、このフォームの ACTION フィールドで指定されている tlr.asp で処理されます。

リクエストを処理する

Web サーバは、TRANSFER リクエストを受信すると tlr.asp プログラムを実行します。クライアント・リクエストは Web サーバで Request オブジェクトに変換されます。Request オブジェクトは、このフォームに入力されたすべてのデータと非表示フィールドなどのほかのフォーム・データを含むメンバを持っています。Web サーバは、この Request オブジェクトを実行プログラムで使用できるようにします。

tlr.asp プログラムは、VBScript のみを含みます。このプログラムは、最初に Jolt セッション・プール・マネージャが初期化されているかどうかを確認します。

次のリスト内のコード例は、この初期化チェックの実行と、セッション・プールが初期化されていない場合に HTML のエラー・ページを返すことを示しています。

コード リスト 8-4　tlr.asp:Jolt **セッション・プール・マネージャが初期化されているかどうかを確認する**

```
<%
If Not IsObject(Application("mgr")) Then
%>
    <HTML>
    <HEAD><TITLE>Error</TITLE></HEAD>
    <BODY><CENTER>
    <H2>Session Manager is not initialized</H2>
    <P>Make sure that you access the correct HTML
    </CENTER></BODY>
    </HTML>
<%
End If
%>
```

セッション・プールが初期化されている場合、このプログラムはリクエストの処理を続行します。プログラムは、次のリストで示すように、セッション・プール・マネージャからセッションを探し出します。

コード リスト 8-5　tlr.asp: **セッションを探し出す**

```
Set pool = Application("mgr").getSessionPool(Null)
```

有効なセッションを探し出すと、プログラムは、結果をクライアントに返すために使用する HTML テンプレートを取得します。この例では、テンプレートは初期化セクションでキャッシュされています。取り出されるテンプレートは、呼び出されているサービスの名前である `Request("SVCNAME")` によって識別されます (次のリストを参照)。

コード リスト 8-6 tlr.asp: **キャッシュされている** HTML **テンプレートを取り出す**

```
'// レスポンス・テンプレートを選択します
If IsEmpty(Application("templates")) Then
   Set template = Server.CreateObject("BEAWEB.Template")
Else
   Select Case Request("SVCNAME")
      Case "INQUIRY"
         Set template = Application("templates")(INQUIRY)
      Case "DEPOSIT"
         Set template = Application("templates")(DEPOSIT)
      Case "WITHDRAWAL"
         Set template = Application("templates")(WITHDRAWAL)
      Case "TRANSFER"
         Set template = Application("templates")(TRANSFER)
   End Select
End If
```

次のリストに示すような BEA Tuxedo サービスを呼び出します。この例では、request オブジェクトからの入力データが、セッションの `call()` メソッドに渡されます。`call()` メソッドは、ASP 組み込みの Request オブジェクトを入力として使用します。`call()` メソッドの結果は、`output` オブジェクトと `iodata` 配列に格納されます。

コード リスト 8-7 tlr.asp:BEA Tuxedo **サービスの呼び出し**

```
Set output = pool.call(Request("SVCNAME"), Null, Nothing)
Set iodata(1) = output
```

BEA Tuxedo サービスを呼び出した後、output オブジェクトと iodata 配列の第 2 要素に、このサービス呼び出しの結果が入っています。

注記　フォームで最初に指定されるフィールド名と BEA Tuxedo サービスのパラメータ名が一致しているため、Request オブジェクトをそのまま call() メソッドで使用することができます。これらの名前が一致しない場合は、call() メソッドを呼び出す前に、各サービス・パラメータに対して「名前 = 値」の形式で入力配列を作成します。

クライアントへ結果を返す

この段階では、クライアントに結果は返されていません。最後の手順では、このサービス呼び出しの結果を含む HTML ページをクライアントに返します。この HTML ページは、テンプレートにサービス呼び出しが返すデータをマージしたものです (コード リスト 8-7 を参照)。

テンプレート・ファイルには、呼び出しに固有なさまざまなデータのプレースホルダが入っています。これらのプレースホルダは、<%=NAME%> という特殊なタグによって識別されます。次のリスト内に示すコード例は、インデックスを使用しているパラメータ名のどのオカレンスが使用されているかを示します。たとえば、ACCOUNT_ID[0] は、ACCOUNT_ID フィールドの最初のオカレンスを指定します。

コード リスト 8-8　transfer.temp:TRANSFER **の結果のプレースホルダ**

```
<TABLE BORDER=1>
<TR><TD></TD><TD ALIGN=CENTER><B>Account #</B></TD>
   <TD ALIGN=CENTER><B>Balance</B></TR>
<TR><TD ALIGN=RIGHT><B>From:</B></TD><TD><%=ACCOUNT_ID[0]%></TD>
   <TD><%=SBALANCE[0]%></TR>
<TR><TD ALIGN=RIGHT><B>To:</B></TD><TD><%=ACCOUNT_ID[1]%></TD>
   <TD><%=SBALANCE[1]%></TR>
</TABLE>
```

テンプレートの中のプレースホルダを、サービス呼び出しが返したデータの実際の値に置換する場合は、次のリストに示す Template オブジェクトの `eval()` メソッドを使用します。このメソッドは、テンプレート・ファイルのプレースホルダと結果データの同名のフィールドを比較し、一致するデータでプレースホルダを置換します。結果 (`output` オブジェクト) の有効性チェックは、次のリスト内に示す方法で実行されます。output オブジェクトがない場合は、エラー・テンプレート・ページが返されます。

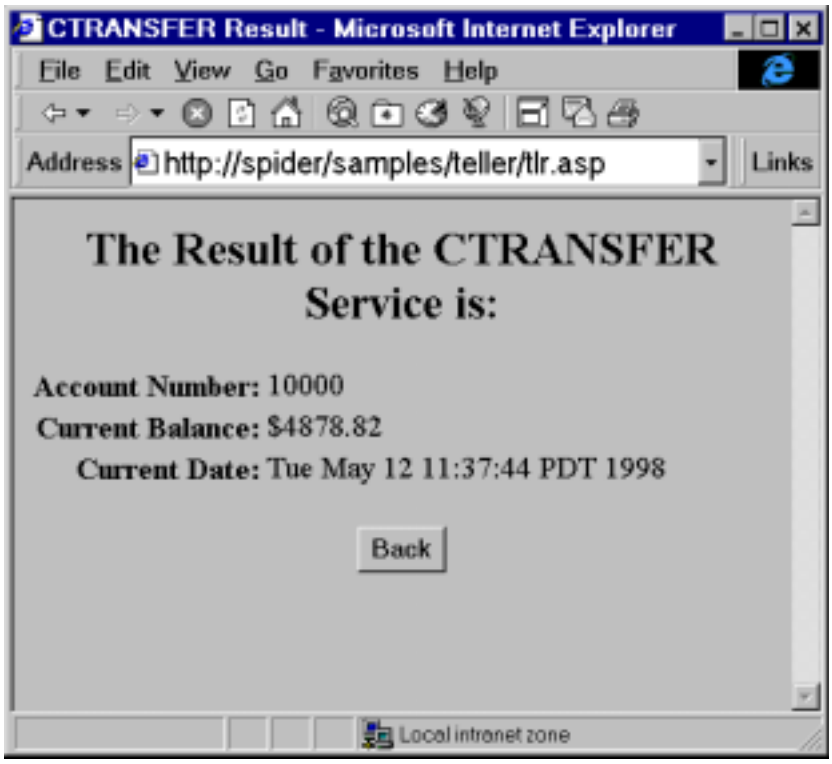
コード リスト 8-9 tlr.asp: **テンプレートの処理**

```
path = Application("templatedir")
If (Not IsObject(output)) Or (output is Nothing) Then
    Call template.evalFile(path & "\nosession.temp", Null)
Elseif output.noError() Then
    Call template.eval(iodata)
Elseif output.applicationError() Then
    Call template.evalFile(path & "\error.temp", iodata)
Else
    '// System error
    Dim errdata(0)
    Set errdata(0) = Server.CreateObject("BEAWEB.TemplateData")
    Call errdata(0).setValue("ERRNO", output.getError())
    Call errdata(0).setValue("ERRMSG", output.getStringError())
    Call template.evalFile(path & "\syserror.temp", errdata)
End If
```

注記　`iodata` 配列には、入力リクエストとサービス呼び出しの結果が含まれています。iodata 配列は、結果ページに入力データを含ませたい場合に便利です。

テンプレートが処理されると、次の図のような HTML の結果ページがクライアントに返されます。

図 8-4tlr.asp 結果ページ



A BEA Jolt の例外

この付録では、発生する可能性のある BEA Jolt の例外について説明します。Jolt クラス・ライブラリは、BEA Jolt と BEA Tuxedo の両方の例外を返します。

BEA Tuxedo の例外の詳細については、次のマニュアルのいずれかを参照してください。

- 『BEA Tuxedo コマンド・リファレンス』
- 『BEA Tuxedo C リファレンス』
- 『BEA Tuxedo COBOL リファレンス』
- 『BEA Tuxedo FML リファレンス』
- 『BEA Tuxedo のファイル形式とデータ記述方法』

また、Jolt クラス・ライブラリの例外は、『BEA Jolt API リファレンス』でクラス、コンストラクタ、およびメソッドごとにリストされています。

以下は、BEA Jolt の実行中に発生する可能性のある、BEA Jolt と BEA Tuxedo の例外の一覧です。各項目では、例外発生の原因と問題を解決するためのアクションを示しています。

1. TPEABORT	トランザクションをコミットできませんでした。	
	原因	サーバ側でトランザクションをコミットできなかった場合に発生します。また、以前のブロッキング状態が原因でタイムアウトになったコミット操作に対して、JSH がメッセージの再送を実行したときに発生する場合もあります。BEA Tuxedo では、未処理の応答が残っているか、または会話型接続がオープンのまま、`tpcommit()` が呼び出されると、この例外が返されます。
	アクション	サーバ側にトランザクションのエラーがないかどうかを確認してください。BEA Jolt クライアントからのリクエストの再送は、サーバ側のトランザクションの問題が解決された後で行ってください。
2. TPEBADDESC	BEA Jolt では、この例外は発生しません。	
	原因	BEA Tuxedo では通常、`tpgetrply()` または `tpsend()` に無効な呼び出し記述子が指定された場合にこの例外が発生します。
	アクション	なし
3. TPEBLOCK	**BEA Tuxedo でブロッキング状態が発生し、`TPNOBLOCK` フラグが指定されます。**	
	原因	サーバがバックアップされているために発生します。
	アクション	負荷が大きい場合は、アプリケーションを再検討し、再構築しなければならない場合があります。

4. TPEINVAL	アプリケーションに対して無効な引数が指定されました。	
	原因	セキュリティ・プロトコルを実行する前に新しい JoltSession クラスが処理されると発生します。Jolt の URL ハンドラ・ルーチンでは、`openConnection()` メソッドで無効な CHALLENGE 応答を受け取るとこの例外が発生します。また、JSL `-H` オプションに指定した 16 進数のアドレスの先頭に「`0x`」を付けなかった場合、または `UBBCONFIG` ファイルに間違ったアドレスを入力した場合にも `TPEINVAL` 例外が発生します。さらに、`REPNAME` がない場合に `JREPSVR` の `GETREC()`、`DELREC()`、および `GETSVC()` サービスから `TPEINVAL` が返される場合もあります。`REPVAL` を指定しないと、`JREPSVR` の `ADDREC()` サービスから `TPEINVAL` が返されます。
	アクション	この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
5. TPELIMIT	未処理のリクエストまたはサブスクリプションが最大数に達しました。	
	原因	未処理のリクエストが最大数に達しています。この例外は、BEA Tuxedo システムのイベント・ブローカのサブスクリプションが最大数 (現在、内部的な定義は 50) に達したことを意味する場合もあります。
	アクション	負荷が大きい場合は、アプリケーションを再検討し、再構築しなければならない場合があります。
6. TPENOENT	**要求されたサービスが利用できません。**	

	原因	通常、要求されたサービスが BEA Tuxedo のサーバ側で起動または宣言されていないことが原因です。要求されたサービスが Jolt リポジトリで定義されていない可能性もあります。また、BEA Tuxedo システムのイベント・ブローカにアクセスできなかったことを示す場合もあります。
	アクション	サーバ側でサービスが起動または宣言されているかどうかを確認してください。または、要求されたサービスが Jolt リポジトリで定義されているかどうかを確認します。サーバ側でサービスを使用できるようになったら、Jolt クライアントからリクエストを再送する必要があります。
7．TPEOS	**オペレーティング・システムの例外が発生しました。**	
	原因	問題の内容は ULOG ファイルに記述されます。通常、この例外はメモリ割り当ての失敗、無効なネットワーク・アドレス、または JSL の掲示板へのアタッチの失敗が原因で発生します。
	アクション	ULOG ファイルに従って問題を修正してください。問題が修正されたら、Jolt クライアントは接続し直してリクエストを再送しなければならない場合があります。
8．TPEPERM	**セッションに参加しようとする際にパーミッションの問題が発生しました。**	

	原因	JoltSession クラスでは、Jolt クライアントがアプリケーションに参加するためのパーミッションを持たない場合にこの例外が発生します。アプリケーション・パスワードが無効な場合、アプリケーション固有の認証にパスしなかった場合、または許可されていないクライアント名を使用している場合は、パーミッションが拒否されます。Jolt の URL ハンドラ・ルーティングでは、`openConnection()` メソッドに対して無効な CHALLENGE 応答を受け取ると、この例外が発生します。Jolt リポジトリが読み取り専用に設定されていると、`ADDREC()` および `DELREC()` サービス、または `JREPSVR` の `GARBAGECOLLECT()` サービスから `TPEPERM` 例外が返されます。
	アクション	この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
9. TPEPROTO	**関数が不適切なコンテキストで呼び出されました。**	
	原因	この例外で不適切なコンテキストとは、`rollback()` または `commit()` メソッドがパーティシパントによって呼び出された場合、「unsubscribe all」の実行中にアンサブスクライブ・イベントが呼び出された場合、または呼び出し元がクライアントでない場合です。
	アクション	この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
10. TPESVCERR	**BEA Tuxedo の `tpreturn()` または `tpforward()` の実行中、**サービス・ルーチンで例外が発生しました。	

	原因	アプリケーション・レベルのエラー、つまり、無効なフラグが指定された `tpreturn()` または `tpforward()` が呼び出された、呼び出し元の記述子が無効である、戻り値が無効である、のいずれかのエラーがサービス・ルーチンから返されます。
	アクション	この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
11. TPESVCFAIL	**呼び出し元の応答を送信するサービス・ルーチンが、`tpreturn()` を呼び出し、`TPFAIL` が返されました。**	
	原因	サービス・ルーチンからアプリケーション・レベルのエラーが返されます。
	アクション	この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
12. TPESYSTEM	**BEA Tuxedo システムの例外が発生しました。**	
	原因	例外の内容は ULOG ファイルに書き込まれます。たとえば、Diffie-Hellman 方式の暗号化を実行するとします。JSH が調整パラメータを送信できない場合、この例外が発生します。JSL は Challenge 呼び出しの応答を Jolt クライアントに送信できません。そのため、Jolt クライアントがタイムスタンプ、暗号ビット、およびチケット値を間違った値で送信したり、再接続プロトコルのタイムスタンプが一致しなくなります。JSL はネットワーク・プロトコル情報を初期化できず、ネットワーク上の接続指示受け付けアドレスを確立することもできません。また、JSH は、不明なコンテキスト付きのネットワーク・メッセージを受信したり、別の接続でのメッセージを受信します。
	アクション	通常、サーバ側で ULOG ファイル内の例外の内容を確認する必要があります。ハードウェア障害やネットワーク障害が原因のとき、ハードウェアやネットワークのフェイルオーバーを実行できる場合は接続し直してください。
13. TPETIME	**トランザクション・タイムアウトが発生しました。**	

	原因	サーバ側でトランザクション・タイムアウトが発生しました。
	アクション	この種の例外は、アプリケーションのサーバ側で処理する必要があります。Jolt クライアントからのリクエストの再送は、サーバ側の問題が解決された後で行ってください。
14. TPETRAN	**要求されたサービスがトランザクションをサポートしないサーバ上にあり、TPNOTRAN が設定されていません。**	
	原因	要求されたサービスに対してトランザクションがサポートされていません。
	アクション	この種の例外は、アプリケーションのサーバ側で処理する必要があります。Jolt クライアントからのリクエストの再送は、サーバ側の問題が解決された後で行ってください。
15. TPGOTSIG	**予期しないシグナルを受信しました。**	
	原因	シグナルを受け取りましたが、TPSIGSTRT フラグが指定されていませんでした。
	アクション	なし
16. TPERMERR	サーバ側でリソース・マネージャがオープンまたはクローズに失敗しました。	
	原因	リソース・マネージャを使用できないか、クローズする前にすべてのリソースを解放またはコミットできません。
	アクション	ULOG ファイルを参照して、リソース・マネージャがサーバ側でオープンまたはクローズに失敗した原因を確認します。
17. TPEITYPE	**JoltRemoteService クラスで、要求された BEA Tuxedo サービスが入力データのタイプおよびサブタイプを認識しません。**	

	原因	入力データのタイプおよびサブタイプが Jolt リポジトリで定義されていません。
	アクション	入力データのタイプとサブタイプを Jolt リポジトリで定義する必要があります。この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
18. TPEOTYPE	**JoltRemoteService クラスで、BEA Tuxedo の呼び出し元が応答データのタイプおよびサブタイプを認識しません。**	
	原因	出力データのタイプおよびサブタイプが Jolt リポジトリで定義されていません。
	アクション	出力データのタイプとサブタイプを Jolt リポジトリで定義する必要があります。この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
19. TPERELEASE	**BEA Jolt では、この例外は発生しません。**	
	原因	通常、TPACK フラグを指定した任意通知型メッセージがサーバから送信され、ターゲットが肯定応答プロトコルをサポートしていない以前のリリースの BEA Jolt クライアントの場合に発生します。
	アクション	ご使用のマシンに正しいバージョンの BEA Jolt がインストールされているかどうかを確認してください。この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。
20. TPEHAZARD	**何らかの障害が発生したため、トランザクションの代わりに行われた作業がヒューリスティックに完了された可能性があります。**	
	原因	サーバ側で ULOG ファイルを参照して、詳しい情報を確認してください。
	アクション	なし
21. TPEHEURISTIC	**ヒューリスティックな決定により、トランザクションの代わりに行われた処理は部分的に完了し、部分的にアボートされました。**	

	原因	サーバ側で ULOG ファイルを参照して、詳しい情報を確認してください。
	アクション	なし
22. TPEEVENT	**BEA Jolt では、この例外は発生しません。**	
	原因	この例外は通常、BEA Tuxedo の会話型接続で、メッセージの送受信中にイベントが発生したことを意味します。ただし、会話型サーバ接続は BEA Jolt では使用できません。
	アクション	なし
23. TPEMATCH	**JoltUserEvent クラスは非同期通知イベントのサブスクリプションをインプリメントしましたが、既存のサブスクリプションと一致するため、サブスクリプションが失敗しました。**	
	原因	BEA Tuxedo システムのイベント・ブローカのリストに既に存在するサブスクリプションと一致するため、サブスクリプションが失敗しました。
	アクション	なし
24. TPEDIAGNOSTIC	**BEA Jolt では、この例外は発生しません。**	
	原因	通常 BEA Tuxedo で、メッセージを指定したキューに登録したり、キューから取り出す操作が失敗すると、この例外が発生します。ただし、キューへのメッセージの登録や取り出しは BEA Jolt では実行できません。
	アクション	なし
25. TPEMIB	**BEA Jolt では、この例外は発生しません。**	
	原因	この例外は通常、BEA Tuxedo で、`tpadmcall()` を使用した管理リクエストが失敗すると発生します。ただし、TMIB 呼び出しは BEA Jolt では使用できません。
	アクション	なし
26. TPEJOLT	**この例外は BEA Jolt で問題が発生していることを示します。**	

原因	TPEJOLT 例外は、次のいずれかの理由で発生します。 ■ JoltSession クラス―セッション・オブジェクトまたはメッセージ ID が無効な場合、send()、recv()、または cancel() メソッドは TPEJOLT をスローします。 ■ JoltSession クラス―TPINIT データ変換が失敗すると TPEJOLT をスローします。 ■ bea.jolt.pool.connection クラス―実行時の例外が発生すると TPEJOLT をスローします。 ■ JoltRemoteService―BEA Jolt と BEA Tuxedo の間のバッファ変換が失敗した場合、要求されたサービスが Jolt リポジトリで定義されていない場合、要求されたサービスが正しいバージョンではない場合、または応答データの変換が失敗した場合に、call() メソッドは TPEJOLT をスローします。 ■ JoltUserEvent クラス―イベント名の変換が失敗した場合、無効なメッセージ ID が検出された場合、または任意通知型メッセージのデータ変換が失敗した場合に TPEJOLT をスローします。
アクション	この種の例外は、アプリケーションの開発時に処理しておく必要があります。この例外が実働環境で発生してはなりません。

索引

M

R

S

T

U

V

W

X

あ

い

お

す

せ

つ

て

と

は

ふ

ま

め

り

れ

ろ