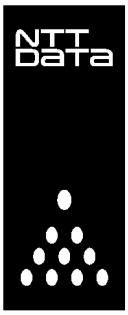


TERASOLUNA Server Framework for Java（Rich 版）

チュートリアル

第 2.0.0.2 版

株式会社ＮＴＴデータ

第1章 概要.....3

1.1 本書の目的.....5
1.2 本書の構成.....5
1.3 対象者.....5
1.4 はじめに.....6
1.5 Spring フレームワーク概略.....7
1.6 Rich 版概略.....11
1.6.1 Rich 版.....11

第2章 Rich 版チュートリアル.....15

2.1 チュートリアル概要.....17
2.2 チュートリアル学習環境の整備.....18
2.3 単純なロジック.....24
2.4 データベースアクセス.....38
2.5 入力チェック.....51
2.5.1 電文形式チェック.....51
2.5.2 単項目入力チェック.....60
2.5.3 相関入力チェック.....68
2.6 例外処理.....77
2.7 アクセス制御.....83

第3章 Appendix.....89

3.1 チュートリアル学習環境の整備(Oracle).....91
3.2 チュートリアル学習環境の整備(PostgreSQL).....93
3.3 チュートリアル学習環境の整備(Tomcat-JNDI).....96
3.4 チュートリアル学習環境の整備(WebLogic).....100
3.5 チュートリアル学習環境の整備(WebSphere).....104
3.6 チュートリアル学習環境の整備(WebLogic-WTP).....109
3.7 WTP プロジェクトを非 WTP 環境へ移行する手順.....112
3.8 JDK のバージョンを変更する手順.....115
3.9 チュートリアル学習環境の整備(Cosminexus).....120

第1章

概要

1.1 本書の目的

「TERASOLUNA Server Framework for Java（Rich 版）チュートリアル」（以下、「本書」という。）は、TERASOLUNA Server Framework for Java（Rich 版）(以下、Rich 版）をソフトウェアアーキテクチャとして採用した場合の Web アプリケーションの実装方法をチュートリアル形式で学んでいくものである。プログラマは実際に手を動かして理解してもらいたい。

1.2 本書の構成

本書は、以下のような章立てで構成される。

- 第 1 章 概要
  本書の位置付け、Spring フレームワーク、Rich 版の説明など。
- 第 2 章 Rich 版 チュートリアル
  Rich 版を利用したコーディングの仕方を、ポイントごとにチュートリアル形式で説明する。
- 第 3 章 Appendix
  2 章と異なる環境でチュートリアルを実施する方法について説明する。

1.3 対象者

本書は、Rich 版を用いた J2EE アプリケーション開発に携わるプロジェクトメンバのうち、特に以下の方を対象とする。

- 業務ロジックの設計担当者
- 業務ロジックのプログラマ

本書を読むための前提はオブジェクト指向、J2SE、J2EE、Web Application Server、データベースの知識をある程度持っている作業者を対象としている。

1.4 はじめに

■ TERASOLUNA とは

世界デファクト技術に立脚した Web アプリケーション開発の総合的ソリューションであり、そのプロダクト構成の一部分として J2EE と.NET プラットフォームのそれぞれに対応したフレームワークを提供している。本書では、J2EE プラットフォームに対応した Rich 版について説明していく。

■ Rich 版とは

Spring フレームワークをベースとした Java/J2EE アプリケーション向けフレームワークである。Web アプリケーションの全ての層をカバーしている。プレゼンテーション層では Spring フレームワークが提供している Spring Web MVC フレームワークを採用して拡張している。また、永続化層では O/R マッピングツールの iBATIS を採用して拡張している。O/R マッピングツールとは、オブジェクトとリレーショナルデータベースのデータについてそれらのギャップを吸収し、相互変換を可能にしたツールである。

本書では、Spring フレームワークの概略、Rich 版の概略について次節以降、説明していく。

1.5 Spring フレームワーク概略

本節では Spring フレームワークの概略について説明する。

■ **Spring フレームワーク 概略**

- DI コンテナをベースにした Java/J2EE アプリケーション向けのフレームワーク。
- AOP/JDBC/MVC フレームワークなどの様々な機能が備わっている。
- プレゼンテーション層・サービス層・永続化層など全ての層をカバーする。
- 個々のモジュールは独立で利用可能なため、必要なモジュールのみ導入可能。
- 既存技術との親和性が高い。

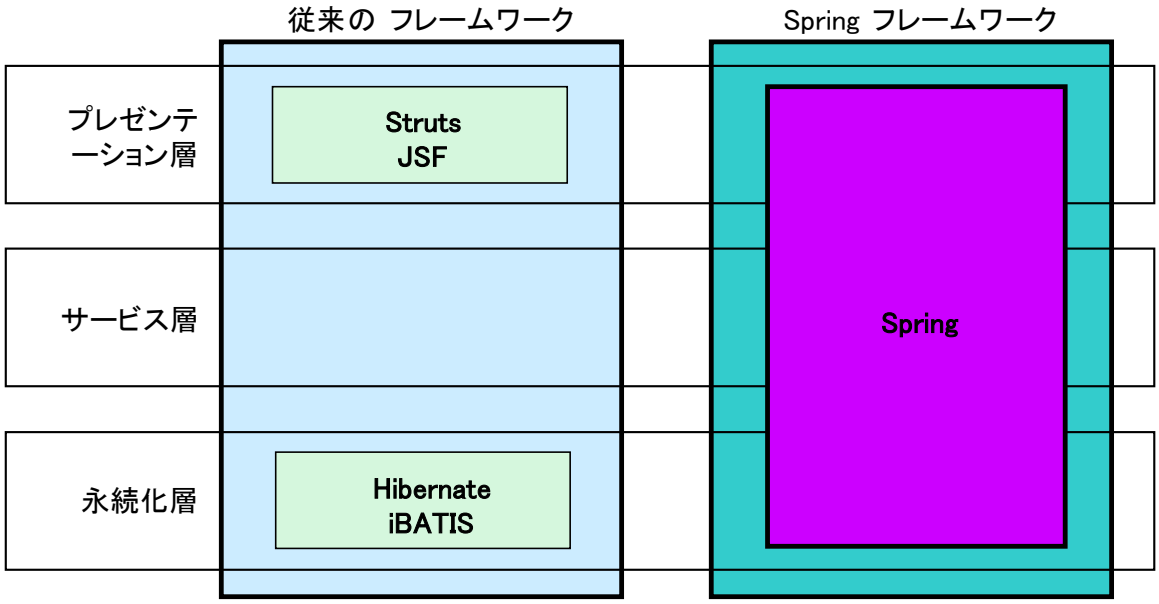


図1 Spring フレームワーク概略

(1) DI コンテナとは

依存性の注入（Dependency Injection）と呼ばれる技法で、オブジェクトのライフサイクルや依存関係の管理機能を備えた実行基盤（コンテナ）のことを表す。

- DI コンテナの役割
  - オブジェクトのインスタンス生成
    オブジェクト A がオブジェクト B を使用する際にインスタンスを new する必要がなく、DI コンテナがオブジェクト B のインスタンスの生成を行い、そのインスタンスを使用するオブジェクト A のセッターに設定してくれる。
  - オブジェクトのシングルトン管理
    Bean 定義ファイルでオブジェクトに対してシングルトンの宣言を行うだけで、コンテナはそのオブジェクトをシングルトンとして管理する。
  - ライフサイクル管理機能
    オブジェクトの生成・破棄のタイミングでメソッドを呼び出し、処理を実行させることができる。
  - オブジェクト間の関連制御機能

- DI コンテナのメリット
  DI コンテナを使用することで、クラス間を疎結合に保つことが可能となり、以下のメリットが生まれる。
  - メンテナンス性の向上
    オブジェクト同士はインタフェースで関連付けられ、Bean 定義ファイルで実装クラスを切り替えることができる。実装クラス同士の依存度が低下するため、ロジック変更やコンポーネントの差し替えに対してその影響範囲を極小化できる。
  - テストの容易性
    モックを利用したテストが行えるため、単体試験でのテスタビリティが向上する。
  - 再利用性の向上
    ・SpringAPI に対して依存の少ないオブジェクト設計が可能。
    ・コンポーネントは POJO（通常の Java クラス）であり、EJB などの特定の API に依存しないため再利用できる機会が増える。

(2) AOP（アスペクト指向プログラミング）とは

オブジェクト指向プログラミング（OOP）を補完する技術として生まれたものです。
AOP では、ログ出力のような、あらゆるモジュールに横断的に散在する処理を分離することができる。

- OOP の問題点
  オブジェクト指向プログラミング（OOP）での問題点として、各モジュール内に別モジュールを呼び出す処理が散在してしまうことがあげられる。
  AOP を使用することで、ソースコードに手を加えることなく、任意の処理を実行時またはコンパイル時に組み込むことができる。
- AOP の利用例
  代表的な例として、以下の処理が挙げられる。
  - ロギング処理
    プログラムの欠陥の原因を突き止めるため、処理の経過を記録（ロギング）する。
    ロギングするということは、プログラムの複数箇所（各クラスの各メソッド）にその命令を追加する必要があり、またプログラムの書き換えは最終的に手作業となる。これにより、同じような処理が複数箇所に分散する。後になって集めたい情報が増えたり、処理が不要になったりすると、それに合わせてすべての場所を変更する必要があった。
  - トランザクション管理
    EJB の代表的な特徴である宣言的なトランザクション制御を、AOP を利用することで可能とする。
    宣言的トランザクションを行うことで、開発者からトランザクションの開始、終了を隠蔽することができる（ビジネスロジック開始時にトランザクションを開始し、終了時にコミットさせ、例外時はロールバックを行うような処理を、容易にフレームワークに任せることができる）。

  他にも AOP の使用が有効だと思われる例として、以下の処理が挙げられる。
  - 例外処理
  - メソッドの実行時間が閾値を超えたらログに書き出す
  - セキュリティ管理
  - 永続化
  - モック
  - メソッドの呼び出しをリモート化
  - メソッドの呼び出しを同期化(synchronized)
  - メソッドの呼び出しを非同期化
  - メソッド呼び出しの委譲

(3) 既存技術との親和性

個々のモジュールは独立で利用可能なため、必要なモジュールのみ導入可能。
例えば、MVC フレームワーク部分を Spring Web MVC フレームワークを利用せず、Struts を利用することが可能。ただし DI コンテナは必要(プログラムの実行基盤であるため)。

■ Spring が各層で提供する機能

- DI コンテナ
- AOP フレームワーク
- MVC フレームワーク
- Web インテグレーション機能
- JDBC 抽象化フレームワーク
- O/R マッピングインテグレーション機能

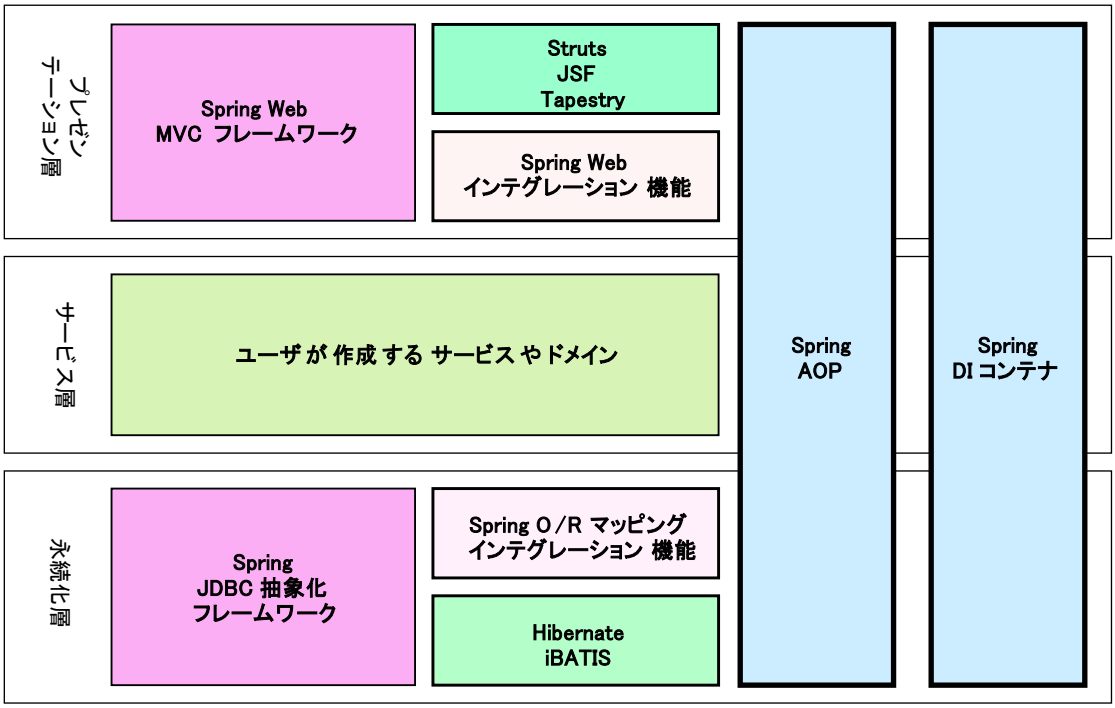


図2 Spring フレームワークの機能

(1) プレゼンテーション層

① Spring Web MVC フレームワーク

- Spring が提供するプレゼンテーション層のフレームワーク(Rich 版では、プレゼンテーション層のフレームワークとして Spring Web MVC フレームワークを用いている)。
- アプリケーションコントローラとして以下のもので構成。
  - アプリケーションコントローラであるコントローラサーブレット
  - コントローラサーブレットから呼ばれるコントローラクラス
  - 動作を制御する定義ファイル
- MVC の View として以下のものが利用可能。
  - JSP&JSTL
  - Velocity
  - XSLT
  - PDF/Excel

- コントローラクラスはユーザからの入力データを Web コンテナに依存しない POJO で取得でき、Web コンテナから分離された入力データの検証クラスを利用することが可能。

② Spring Web インテグレーション機能

- Spring を利用してプレゼンテーション層の設計・実装をする場合、必ずしも Spring Web MVC フレームワークを使う必要はない。
- Struts、JSF などを Web プレゼンテーション層として利用することが可能。

(2) サービス層

① DI コンテナとサービス、ドメイン

- サービスおよびドメインは POJO で実装される。
- コントローラは必ず DI コンテナを通してサービスクラスをインタフェースとして取得し、サービスロジックを利用する。
- サービスクラスからデータアクセスの利用も DI コンテナからインタフェースを取得して行う。

② AOP とトランザクション管理

- Spring は AOP 機能を使用して POJO にトランザクション機能を追加する。

(3) 永続化層

① Spring JDBC 抽象化フレームワーク

- Spring が提供する JDBC 抽象化フレームワーク
  - JDBC を直接扱わない。
  - SQL 文を利用するタイプのデータアクセスフレームワーク。

② Spring O/R マッピングインテグレーション機能

- Spirng が提供する O/R マッピングインテグレーション機能
  - Hibernate、iBATIS などの O/R マッピングツールが利用できる。

1.6 Rich 版概略

1.6.1 Rich 版

次に、本書で採用する J2EE フレームワークである Rich 版を説明する。
Rich 版は Spring フレームワーク及び Spring Web MVC フレームワークを機能拡張して提供している。

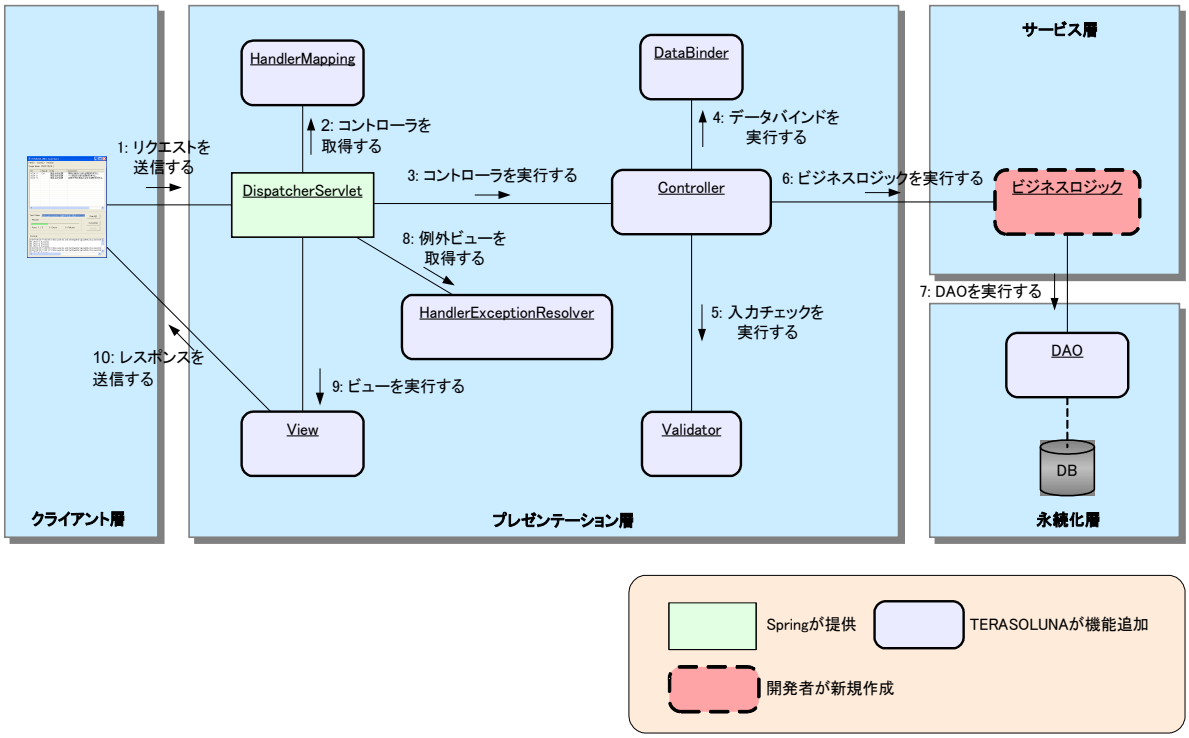


図3 Rich 版のクラス構造

■ フレームワークが規定しているクラス構造

Rich 版では、Spring フレームワークを拡張し、以下の機能や仕組みを提供する。

- アクセス制御
- XML 形式のリクエストデータを JavaBean にバインドする機能
- JavaBean の値をチェックする機能
- POJO 及び定型インタフェースを持つ業務ロジッククラスを起動する仕組み
- J2EE/JDBC の詳細な知識がなくともデータベースアクセスが可能な仕組み
- リクエスト処理で使用する共通の情報を保持する機能

次に、Rich 版が提供するクラスについて説明していく。

(1) DispatcherServlet

Spring-MVC で使用される Servlet。リクエスト処理フロー全体を司る。

(2) HandlerMapping

サーバに送られたリクエストが、どの Controller を使用するか判断するインタフェース。

- SimpleHandlerMappingEx
  HandlerMapping の実装クラス。リクエストヘッダに設定されたリクエスト名をキーにして、(Bean 定義ファイル内の)コントローラ ID と対応付けを行う。

(3) Controller

リクエスト処理フローを司るインタフェース。DispatcherServlet から呼び出される。
ビジネスロジックの入力となる JavaBean の生成および入力値検証、ビジネスロジックの実行、業務出力 JavaBean 及びビューの返却を行う。

- TerasolunaController
  Rich 版で拡張した Controller の実装クラス。
  本クラスは抽象クラスとして定義しているため、実際に使用するためにはサブクラスを生成する必要がある。ビジネスロジックが BLogic インタフェースを実装する場合は、BLogicController を使用する。
  POJO の場合は、コントローラを個別に生成する必要がある。
- BLogicController
  TerasolunaController の継承クラス。BLogic インタフェースを実装したビジネスロジックを実行する。

(4) DataBinder

リクエストデータを業務入力 JavaBean にバインドするクラス。

- ServletRequestDataBinder
  Binder の実装クラス。クエリ形式のリクエストデータを業務入力 JavaBean にバインドするクラス。
- XMLServletRequestDataBinder
  ServletRequestDataBinder の継承クラス。XML 形式のリクエストデータを業務入力 JavaBean にバインドするクラス。

(5) Validator

業務入力 JavaBean に対して入力値検証を実行するインタフェース。デフォルトでは、Commons Validator を用いた設定ファイルベースの入力チェックを提供している。

(6) ビジネスロジック

ビジネスロジックを実行するクラス。ビジネスロジックは POJO と BLogic の 2 種類の実行方式から選択することができる。
入出力クラスは、POJO と BLogic の2種類とも JavaBean になる。
ビジネスロジックのポータビリティを保つ場合はインタフェースが自由な POJO で実装を行い、ビジネスロジックのインタフェースを固定する場合はビジネスロジックに BLogic インタフェースを実装させる。

- BLogic
  Rich 版が提供するビジネスロジックのインタフェース。BLogicController から呼び出される。

(7) DAO

データベースにアクセスするオブジェクトのインタフェースが3種類
（QueryDAO、UpdateDAO、StoredProcedureDAO）用意されている。
データベース接続等の JDBC リソースの取得と開放はフレームワーク側が行うため、フレームワーク利用者が意識する必要はない。

- QueryDAO
  参照系のデータベースアクセスを行う DAO インタフェース。
- UpdateDAO
  更新系のデータベースアクセスを行う DAO インタフェース。
- StoredProcedureDAO
  ストアドプロシージャを実行する DAO インタフェース。

(8) HandlerExceptionResolver

コントローラで発生した例外のハンドリングを行うインタフェース。例外の型とビューの対応付けを行う。

- SimpleMappingExceptionResolverEx
  HandlerExceptionResolver の実装クラス。例外の型とエラーコード及びエラー種別の対応付けを行う。

(9) View

ビジネスロジックの実行結果として返却された JavaBean からレスポンスを生成し、クライアントに送信するインタフェース。

- CastorView
  オブジェクト-XML 変換ツール Castor を利用した、View 実装クラス。業務出力 JavaBean から XML 形式のレスポンスを生成する。
- VelocityView
  テンプレートエンジン Velocity を利用した View 実装クラス。テンプレートファイルをもとに、業務出力 JavaBean から XML 形式のレスポンスを生成する。

第2章

Rich 版チュートリアル

2.1 チュートリアル概要

本書では、Rich 版を理解するために、チュートリアル形式で簡単な Web アプリケーションを構築する。各節の内容を以下に示す。

表1. 各節の内容

2.2 チュートリアル学習環境の整備	: Web アプリケーションを構築するための環境を準備する。
2.3 単純なロジック	: クライアントからのリクエスト処理を行う方法を学ぶ。
2.4 データベースアクセス	: データベースからのデータ取得やデータ更新の方法を学ぶ。
2.5 入力チェック	: 入力したデータをチェックする方法を学ぶ。
2.6 例外処理	: 例外発生時の対処方法を学ぶ。
2.7 アクセス制御	: 特定のパスへのアクセス制御の方法を学ぶ。

2.2　チュートリアル学習環境の整備

本節では、チュートリアルを学習するための環境整備について説明する。
※ 環境整備を Oracle で行いたい場合は「3.1 チュートリアル学習環境の整備(Oracle 版)」を参照のこと。

■ 想定環境

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.23
- データベース：HSQLDB 1.8.0.4
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：WTP 1.5.5

■ インストール/開発環境の整備

(1) アプリケーションの用意

本書で必要となるアプリケーションを以下に用意する。

- Java SE5.0
- Tomcat 5.5.23
- HSQLDB 1.8.0.4
- Eclipse SDK 3.2.2
- WTP 1.5.5

(2) アプリケーションのインストール

用意した各アプリケーションをインストールする。本書ではアプリケーションをそれぞれ以下のディレクトリにインストールすると想定して記述している。

表2. インストールディレクトリ

Java SE5.0	C:\Program Files\Java\jdk1.5.0_0x (x はバージョン番号)
Eclipse 3.2.2+プラグイン	C:\Eclipse
Tomcat 5.5.23	C:\Program Files\Apache Software Foundation\apache-tomcat-5.5.23
Workspace ディレクトリ	C:\Eclipse\workspace

- Java のホームディレクトリの設定

ここでは以下のように環境変数を設定する。

- JAVA_HOME：C:\Program Files\Java\jdk1.5.0_0x (x はバージョン番号)
- path：%JAVA_HOME%\bin;

- Eclipse の設定

1. インストール済み JRE の設定

Eclipse の「ウィンド(W)」-「設定(P)」メニューを選択する。
「java」-「インストール済みのJRE」で、「jdk1.5.0_0x (x はバージョン番号)」にチェックがついていることを確認する。
続けて「java」-「コンパイラー」で、「JDK 準拠」のコンパイラ準拠レベルが“5.0”になるように設定する。

2. Tomcat の設定

Eclipse の「ファイル(F)」-「新規(O)」-「その他(N)」メニューを選択する。
ウィザードの選択から「サーバー」-「サーバー」を選択し次へを押下する。新規サーバーから「Apache」-「Tomcat v5.5 サーバー」を選択して終了する。

3. 構文チェックの設定

Eclipse の「ウィンドウ(W)」-「設定(P)」メニューを選択する。
「妥当性検証」を選択し、「JSP 構文バリデーター」のチェックを外し、「OK」を押下する。

(3) プロジェクトの準備

本書で使用するチュートリアルアプリケーションでは Rich 版で提供しているブランクプロジェクトを使用する。
ここでは“terasoluna-server4jrich-blank_2.0.0.2.zip”ファイルを“C:\Eclipse\workspace”直下に展開する。

(4) プロジェクトのインポート

「(3)プロジェクトの準備」で作成した“C:\Eclipse\workspace\terasoluna-spring-rich-blank”を Eclipse で編集出来るようにインポートする。

① Eclipse を起動する。
② 「ファイル(F)」-「インポート(I)」を選択する。
③ 選択画面では「既存のプロジェクトをワークスペースへ」を選択して、次へを押下する。
④ 「プロジェクトのインポート」画面ではルートディレクトリの選択欄に“C:\Eclipse\workspace\terasoluna-spring-rich-blank”を指定し、終了を押下する。

※インポートと同時に Eclipse が XML ファイル検証のためにインターネットへ接続する。その際に、ネットワーク環境によっては ID とパスワードの入力のウィンドウが出力されるが、入力に失敗するとリトライを内部で繰り返し ID をロックされる可能性がある。入力の際は十分注意が必要である。なお出力したウィンドウをキャンセルで閉じることもできるが何回も繰り返し要求されるので、最初は入力することをお勧めする。

⑤ 「パッケージ・エクスプローラ」よりインポートしたプロジェクトを開き、Web アプリケーション・ライブラリー[{0}]内に jar ファイルが存在しているかを確認する。存在しない場合は、プロジェクトを右クリックして、「プロジェクトを閉じる」を選択し、再度プロジェクトを選択し「プロジェクトを開く」を選択する。それでも表示されない場合は、Eclipse を再起動する。
⑥ 「パッケージ・エクスプローラ」よりインポートしたプロジェクトを開き、「サーバー・クラスパス・コンテナー」と表示されたものがないことを確認する。ある場合は、「(2)アプリケーションのインストール」の Tomcat の設定作業を行っていない可能性があるので再度行い、その後、「サーバー・クラスパス・コンテナー」を選択し、右クリックメニューより「構成」を選択し、「Apache Tomcat v5.5」を選択し、「終了」を押下する。その後、Eclipse を再起動する。
⑦ Eclipse の「プロジェクト(P)」-「クリーン(N)」を選択し、すべてのプロジェクトをクリーンにチェックを入れて「OK」を押下する。

下図に"terasoluna-spring-rich-blank”のプロジェクト構成を示す。

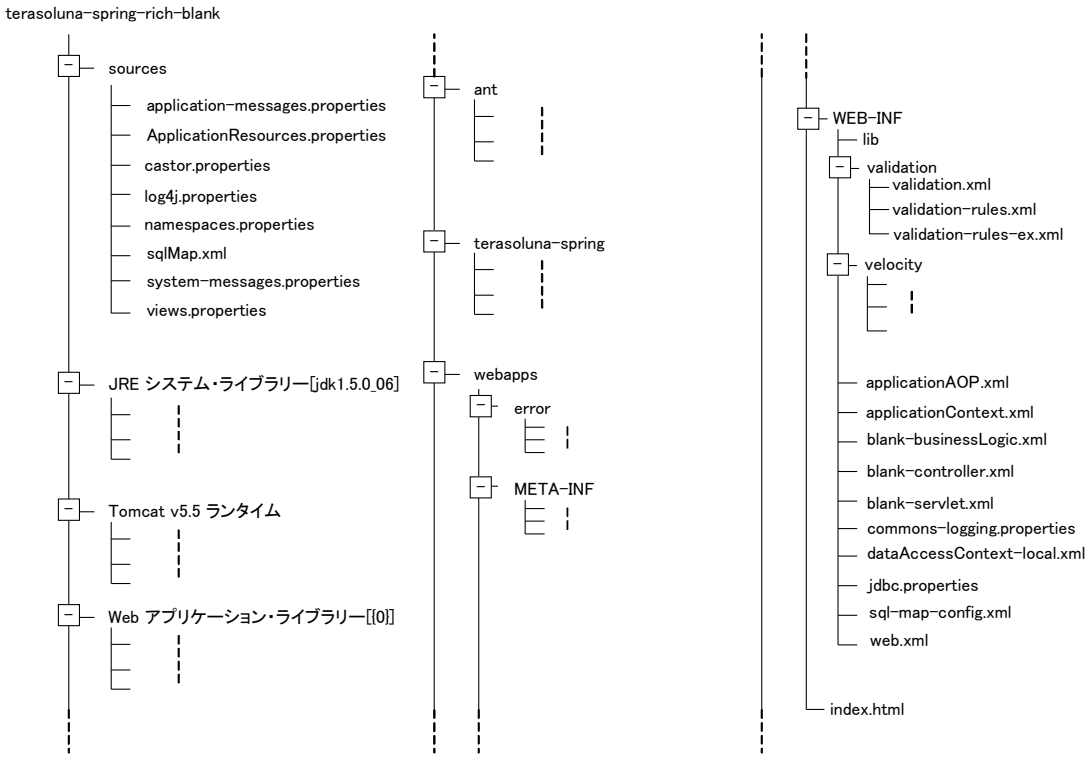


図4 プロジェクト構成

また、本書で使用する各設定ファイルの解説を下表にしめす。

表3. 設定ファイル解説

ファイル名	説明
sqlMap.xml	iBATIS 設定ファイル。SQL 文を設定する。
validation.xml	バリデーション設定ファイル。
applicationAOP.xml	Spring 設定ファイル。AOP の Bean 定義ファイル。トランザクション、ログの設定を行う。
applicationContext.xml	Spring 設定ファイル。アプリケーション全体の Bean 定義ファイル。メッセージリソース、入力チェッククラスの設定を行う。
blank-businessLogic.xml	Spring 設定ファイル。モジュール固有の Bean 定義ファイル。ビジネスロジックの設定を行う。
blank-controller.xml	Spring 設定ファイル。モジュール固有の Bean 定義ファイル。コントローラの設定を行う。
blank-servlet.xml	Spring 設定ファイル。プレゼンテーション層共通の Bean 定義ファイル。例外ハンドラ、ビュー、コントローラの設定を行う。
dataAccessContext-local.xml	Spring 設定ファイル。データアクセスの Bean 定義ファイル。データソース、トランザクション、DAO、iBATIS の設定を行う。
web.xml	Web アプリケーション設定ファイル。フィルタの設定を行う。

⑧ ファイル名のリネーム
“terasoluna-spring-rich-blank/webapps/WEB-INF/”フォルダ配下にある3つのファイル名を以下の表のようにリネームする。

表4. リネーム対応表

リネーム前	リネーム後
blank-businessLogic.xml	tutorial-businessLogic.xml
blank-controller.xml	tutorial-controller.xml
blank-servlet.xml	tutorial-servlet.xml

⑨ “web.xml”ファイルの変更
“terasoluna-spring-rich-blank/webapps/WEB-INF/web.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web
-app_2_4.xsd"
  version="2.4">

 <display-name>TERASOLUNA for Spring Rich 版 チュートリアルプロジェクト</display-name>
 <description>TERASOLUNA for Spring Rich 版 チュートリアルプロジェクト</description>

 <!--
  - ルートアプリケーションコンテキストを定義する XML ファイルの場所
  - ContextLoaderServlet で用いられる。
 -->
 <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/dataAccessContext-local.xml /WEB-INF/applicationContext.xml /WEB-INF/tutoria
l-businessLogic.xml /WEB-INF/applicationAOP.xml
  </param-value>
 </context-param>
                                          :
                                         略
                                          :

 <!--
  - リッチ版 TERASOLUNA でリクエスト処理を行う Spring Web MVC Servlet の設定。
  - Web アプリケーションごとに Application context を保持する。
  - デフォルトで"{servlet-name}-servlet.xml"という名前の設定ファイルが使用される。
  - このファイルでは"tutorial-servlet.xml"という名前の設定ファイルになる。
  -->
 <servlet>
  <servlet-name>tutorial</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/tutorial-controller.xml,/WEB-INF/tutorial-servlet.xml</param-val
ue>
  </init-param>
  <load-on-startup>2</load-on-startup>
 </servlet>
```

```
<!-- Rich 版 TERASOLUNA の Servlet を使用するリクエストの URL パターン -->
<servlet-mapping>
 <servlet-name>tutorial</servlet-name>
 <url-pattern>*.do</url-pattern>
</servlet-mapping>
                                        :
                                       略
                                        :
</web-app>
```

(5) Tomcat へのプロジェクト追加

サーバービューから「(2)アプリケーションのインストール」の Eclipse の設定で追加したサーバーを選択して右クリックする。右クリックメニューの中から「プロジェクトの追加と除去」を選択する。使用可能プロジェクトの中に「(4)プロジェクトのインポート」でインポートしたプロジェクトがあるので、構成プロジェクトに追加する。

(6) データベースの設定

① ブランクプロジェクトとは別で用意されたチュートリアルプロジェクト“tutorial-rich”の中に“hsqldb.zip”がある。ここでは“hsqldb.zip”を“C:\”直下に展開する。

② データベースの起動“C:\hsqldb\terasoluna\startDB.bat”を実行する。

③ データベースへの接続
②のデータベースが起動した状態で、“C:\hsqldb\terasoluna\startDBManager.bat”を実行する。DBManager が起動し、connect 画面が表示される。ここでは、以下のように入力する。 なお、この DBManager によるデータベースへの接続はデータ確認のためなので、データベース起動の度に実行する必要はない。

- Recent：Recent settings...
- Setting Name：なし
- Type：HSQL Database Engine Server
- Driver：org.hsqldb.jdbcDriver
- URL：jdbc:hsqldb:hsql://localhost/terasoluna
- User：sa
- Password：なし

④ データの確認
③の接続後、画面左にテーブル“USERTABLE”、カラム“ID”、“NAME”、“AGE”、“BIRTH”があることを確認する。画面右のコンソール部分に SQL を記述し、“Execute”ボタンを押下するとデータが表示されることを確認する。

2.3　単純なロジック

本節では、Rich 版を利用し、単純なロジックを行う方法について学習する。

■ 概要

図のように単純なロジックを実装する。ビジネスロジックの機能としては、リクエストデータをそのままレスポンスデータに格納して総件数を設定して返す機能とする。

- リクエストデータ・・・ユーザ ID、ユーザ名、年齢、生年月日

```
<?xml version="1.0" encoding="UTF-8"?>
<UserBean>
  <id>1</id>
  <name>テラソルナユーザ１</name>
  <age>28</age>
  <birth>1978-01-14T12:34:56</birth>
</UserBean>
```

- レスポンスデータ・・・総件数、ユーザ ID、ユーザ名、年齢、生年月日

```
<?xml version="1.0" encoding="UTF-8"?>
<resultData>
    <totalCount>1</totalCount>
    <userBean>
        <id>1</id>
        <name>テラソルナユーザ１</name>
        <age>28</age>
        <birth>1978-01-14T12:34:56.000+09:00</birth>
    </userBean>
</resultData>
```

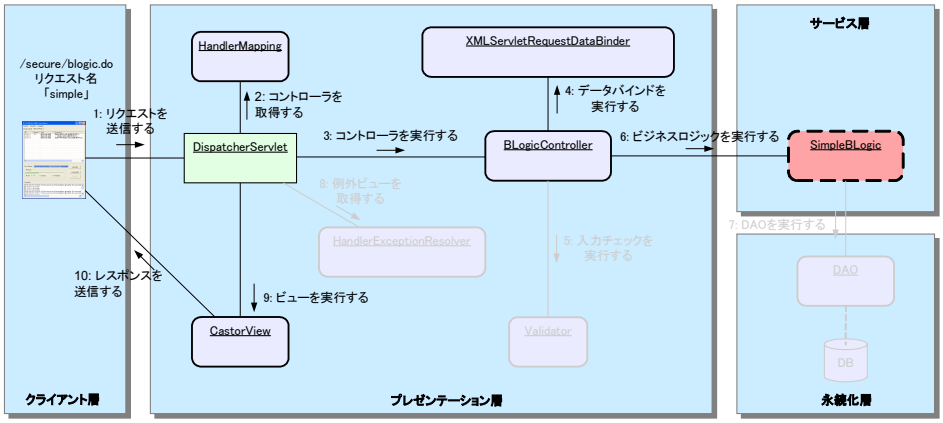


図5　単純なロジック画面遷移図

1. “http://localhost:8080/terasoluna-spring-rich-blank”にアクセスし、テストクライアント画面を表示する。リクエストデータを作成して、リクエスト名“simple”で、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。
2. リクエスト名“simple”と“tutorial-controller.xml”ファイルをもとにコントローラを実行する。
3. コントローラでは、リクエストデータ、スキーマ定義ファイルをもとに XMLServletRequestDataBinder を使ってビジネスロジックの入力となる JavaBean へ変換し、“tutorial-businessLogic.xml”ファイルをもとにビジネスロジックを実行する。
4. ビジネスロジックでは、入力された JavaBean の属性の値及び総件数を実行結果用の JavaBean に格納して返却する。
5. ビジネスロジックから返却された JavaBean、“namespaces.properties”ファイル、マッピング定義ファイルをもとにレスポンスデータを生成し、CastorView を使ってクライアントに送信する。
6. テストクライアント画面に応答電文が正しく表示される。（画面下部に応答電文が表示される）。

このとき、以下の作業が必要となる。

- ビジネスロジックの入力となる JavaBean の作成
- ビジネスロジックの実行結果を格納する JavaBean の作成
- スキーマ定義ファイルの作成
- “tutorial-servlet.xml”ファイルの確認及び変更
- ビジネスロジック実装クラスの作成
- “tutorial-businessLogic.xml”ファイルの変更
- “tutorial-controller.xml”ファイルの変更

■ **手順**

● **ビジネスロジックの入力となる JavaBean の作成**

ビジネスロジックの入力となる JavaBean を作成する。この JavaBean はリクエストデータからマッピングされた値を保持する。項目としては「ユーザ ID」、「ユーザ名」、「年齢」、「生年月日」を定義する。“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.bean/UserBean.java”ファイルを以下のように作成する。

```
/*
 * $Id$
 *
 * Copyright (c) 2007 NTT DATA Corporation
 *
 */
package jp.terasoluna.rich.tutorial.service.bean;

import java.io.Serializable;
import java.util.Date;

/**
 * ユーザ情報保持 Bean。
 *
 * 入力クラス・データベースからの取得時に使用される。
 *
 */
public class UserBean implements Serializable {

    /**
     * ユーザ ID。
     */
    private Integer id = null;

    /**
     * ユーザ名。
     */
    private String name = null;

    /**
     * 年齢。
     */
    private Integer age = null;

    /**
     * 生年月日。
     */
    private Date birth = null;

    /**
     * ユーザ ID を返却する。
     *
     * @return 保持するユーザ ID
     */
    public Integer getId() {
        return id;
    }

    /**
     * ユーザ ID を設定する。
     *
     * @param id ユーザ ID
     */
    public void setId(Integer id) {
        this.id = id;
    }

    /**
     * ユーザ名を返却する。
     *
```

```
     * @return 保持するユーザ名
     */
    public String getName() {
        return name;
    }

    /**
     * ユーザ名を設定する。
     *
     * @param name ユーザ名
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * 年齢を返却する。
     *
     * @return 保持する年齢
     */
    public Integer getAge() {
        return age;
    }

    /**
     * 年齢を設定する。
     *
     * @param age 年齢
     */
    public void setAge(Integer age) {
        this.age = age;
    }

    /**
     * 生年月日を返却する。
     *
     * @return 保持する生年月日
     */
    public Date getBirth() {
        return birth;
    }

    /**
     * 生年月日を設定する。
     *
     * @param birth 生年月日
     */
    public void setBirth(Date birth) {
        this.birth = birth;
    }
}
```

● ビジネスロジックの実行結果を格納する JavaBean の作成

ビジネスロジックの実行結果を格納する JavaBean を作成する。この JavaBean はレスポンスデータへマッピングする値を保持する。項目としては「ユーザ情報保持 Bean」、「総件数」を定義する。“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.bean/ResultData.java”ファイルを以下のように作成する。

```
/*
 * $Id$
 *
 * Copyright (c) 2007 NTT DATA Corporation
 *
 */
package jp.terasoluna.rich.tutorial.service.bean;

import java.io.Serializable;
import java.util.List;
```

```
/**
 * 結果クラス
 *
 */
public class ResultData implements Serializable {

    /**
     * ユーザ情報保持 Bean リスト。
     */
    private List<UserBean> userBean = null;

    /**
     * 総件数。
     */
    private Integer totalCount = null;

    /**
     * ユーザ情報保持 Bean を返却する。
     *
     * @return 保持するユーザ情報保持 Bean
     */
    public List<UserBean> getUserBean() {
        return userBean;
    }

    /**
     * ユーザ情報保持 Bean リストを設定する。
     *
     * @param userBean ユーザ情報保持 Bean リスト
     */
    public void setUserBean(List<UserBean> userBean) {
        this.userBean = userBean;
    }

    /**
     * ユーザ情報保持 Bean をリストに追加する。
     *
     * @param userBean ユーザ情報保持 Bean
     */
    public void addUserBean(UserBean userBean) {
        this.userBean.add(userBean);
    }

    /**
     * 総件数を返却する。
     *
     * @return 保持する総件数
     */
    public Integer getTotalCount() {
        return totalCount;
    }

    /**
     * 総件数を設定する。
     *
     * @param totalCount 総件数
     */
    public void setTotalCount(Integer totalCount) {
        this.totalCount = totalCount;
    }

}
```

● **スキーマ定義ファイルの作成**

リクエストデータに対し、電文形式チェックを行うために必要なスキーマ定義ファイルの作成を行う。(電文形式チェックに関しては、次節以降で説明する)。

“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.bean/UserBean.xsd”ファイルを以下のように作成する。

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="UserBean" type="UserBean-Type"/>

  <xs:complexType name="UserBean-Type">
    <xs:sequence>
      <xs:element name="id" type="xs:int" />
      <xs:element name="name" type="xs:string" />
      <xs:element name="age" type="xs:int" />
      <xs:element name="birth" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

● **“tutorial-servlet.xml”ファイルの確認及び変更**

電文形式チェック機能を実装するためにコントローラ定義を確認及び変更する。

① “tutorial-controller.xml”ファイルで Bean 定義したコントローラの抽象クラスが xmlDataBinderCreator を設定(DI)していることを確認する。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルが以下のようになっていることを確認する。(網掛け部分を確認する)。

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の共通 Bean 定義 -->

                                          :
                                         略
                                          :

    <!-- コントローラの抽象 Bean 定義
         ( ビジネスロジック：BLogic、受信リクエスト：XML 形式 )
    -->
    <bean id="blogicXmlRequestController" abstract="true" parent="blogicController">
        <property name="dataBinderCreator" ref="xmlDataBinderCreator"/>
    </bean>

    <!-- コントローラの抽象 Bean 定義
( ビジネスロジック：BLogic、受信リクエスト：XML 形式、ビュー：Castor )
            -->
            <bean id="blogicXmlRequestCastorViewController" abstract="true" parent="blogi
cXmlRequestController"/>

                                          :
                                         略
                                          :

</beans>
```

② xmlDataBinderCreator の Bean 定義が schemaValidator を設定（DI）していることを確認する。
ここの XMLServletRequestDataBinderCreator で電文形式チェックを行っている。
チュートリアルでは名前空間を意識した XML の送受信に対応するために、schemaValidator の Bean 定義を次の手順で変更する。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルが以下のようになっていることを確認する。

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の共通 Bean 定義 -->
                                          :
                                         略
                                          :

  <bean id="xmlDataBinderCreator"
      class="jp.terasoluna.fw.web.rich.springmvc.bind.creator.XMLServletRequestDataBinder
Creator">
    <property name="oxmapper" ref="oxmapper"/>
    <property name="schemaValidator" ref="schemaValidator"/>
  </bean>

                                          :
                                         略
                                          :

</beans>
```

③ SchemaValidator の Bean 定義に名前空間の設定をする。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルを以下のように変更する。

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の共通 Bean 定義 -->
                                          :
                                         略
                                          :


  <bean id="schemaValidator"
      class="jp.terasoluna.fw.oxm.xsd.xerces.SchemaValidatorImpl" />

                                          :
                                         略
                                          :


</beans>
```

④ oxmapper の bean 定義にレスポンス XML の整形を有効にするために、preserveWhitespaceAtMarshal をに設定する。
※FW ではデフォルト(true)を推奨している。本チュートリアルでテスト時に、目視確認しやすいためにする。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルを以下のように変更する。

(網掛け部分を追加する)。

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の共通 Bean 定義 -->
                                                :
                                               略
                                                :

  <bean id="oxmapper"
      class="jp.terasoluna.fw.oxm.mapper.castor.CastorOXMapperImpl">
          <property name="preserveWhitespaceAtMarshal" value="false"/>
  </bean>
                                                :
                                               略
                                                :

</beans>
```

● ビジネスロジック実装クラスの作成

チュートリアルとしては BLogic インタフェースを実装したビジネスロジックを使う[1]。

ビジネスロジックとして以下の機能を実装する。
（1） UserBean クラスの属性の値を ResultData クラスに格納する。
（2） 総件数を ResultData クラスに設定する。
（3） ResultData クラスを返却する。

“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.blogic/SimpleBLogic.java”ファイルを以下のように作成する。

```
/*
 * $Id$
 *
 * Copyright (c) 2007 NTT DATA Corporation
 *
 */
package jp.terasoluna.rich.tutorial.service.blogic;

import java.util.ArrayList;
import java.util.List;
import jp.terasoluna.rich.tutorial.service.bean.UserBean;
import jp.terasoluna.rich.tutorial.service.bean.ResultData;
import jp.terasoluna.fw.service.rich.BLogic;

/**
 * 入力クラスを結果クラスに格納する BLogic クラス
 *
 */
public class SimpleBLogic
        implements BLogic<UserBean, ResultData> {
```

1 コントローラを実装する手間を省くことができるため。

```
    /**
     * 入力クラスを結果クラスに格納する<br>
     * １．UserBean クラスの属性の値を ResultData クラスに格納する。<br>
     * ２．総件数を ResultData クラスに設定する。<br>
     * ３．ResultData クラスを返却する。
     *
     * @param params 入力クラス
     * @return 結果クラス
     */
    public ResultData execute(UserBean params) {

        ResultData result = new ResultData();
        List<UserBean> userBeanList = new ArrayList<UserBean>();

        UserBean userBean = new UserBean();
        userBean.setId(params.getId());
        userBean.setName(params.getName());
        userBean.setAge(params.getAge());
        userBean.setBirth(params.getBirth());
        userBeanList.add(userBean);

        result.setUserBean(userBeanList);
        result.setTotalCount(userBeanList.size());
        return result;
    }

}
```

- **“tutorial-businessLogic.xml”ファイルの変更**

ビジネスロジック（SimpleBLogic）の Bean 定義情報の追加を行う。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-businessLogic.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。
※初期の“tutorial-businessLogic.xml”ファイルには設定例がコメントで記述してあるが、削除するかそのままにしておくこと。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- サービス層の Bean 定義 -->
<!--                       -->
<!-- サービス層共通の Bean 定義は applicationContext.xml に記述する。 -->
  <!-- ======= ビジネスロジックの定義 ======= -->

  <!-- サービスの設定例 -->
  <!--
  <bean id="sumService"
      class="jp.terasoluna.rich.functionsample.controllerex.service.SumServiceImpl"
      scope="prototype"/>
  -->

  <!-- 2.3 単純なロジック simpleService 定義 -->
  <bean id="simpleService"
      class="jp.terasoluna.rich.tutorial.service.blogic.SimpleBLogic"
      scope="prototype">
  </bean>

</beans>
```

- **“tutorial-controller.xml”ファイルの変更**

ビジネスロジック（SimpleBLogic）を呼び出すコントローラの定義を追加する。

- parent・・・“blogicXmlRequestCastorViewController”とする。（これによりコントローラを実装しなくてもBLogicContorller がコントローラの役割を担い、さらに XML と JavaBean の変換を CastorView を用いて行えるようになる）。

“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-controller.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。
※初期の“tutorial-controller.xml”ファイルには設定例がコメントで記述してあるが、削除するかそのままにしておくこと。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の Bean 定義 -->
```

```
<!--                                                                    -->
<!-- コントローラや相関入力チェッククラスの Bean 定義などを記述する。 -->
<!-- プレゼンテーション層共通の Bean 定義は blank-servlet.xml に記述する。 -->
  <!-- ======= コントローラ定義 ======= -->

  <!-- 動作確認用コントローラの設定 -->
  <bean name="/testController"
      class="org.springframework.web.servlet.mvc.SimpleFormController">
    <property name="commandClass" value="java.lang.Object"/>
    <property name="successView" value="success"/>
  </bean>

  <!-- コントローラの設定例 -->
  <!--
  <bean name="/xmlPOJOCastorController"
      class="jp.terasoluna.rich.functionsample.controllerex.controller.
             ControllerExController"
      parent="pojoXmlRequestCastorViewController" scope ="prototype">
    <property name="sumService" ref="sumService"/>
  </bean>
  -->

  <!-- 2.3 単純なロジック xml-BLogic-castor 定義 -->
  <bean name="/simpleController"
      parent="blogicXmlRequestCastorViewController" scope ="prototype">
    <property name="blogic" ref="simpleService"/>
  </bean>

</beans>
```

■ **確認**

実際にリクエストデータの内容を含んだ応答電文が受信できることを確認する。

（1） 作成・更新したファイルを全て保存し、コンパイルを行ったのちに Tomcat を起動する。
Tomcat の起動は Eclipse のサーバから“サーバを起動”ボタンを押下する。ボタンは下図の四角で囲まれた位置にある。

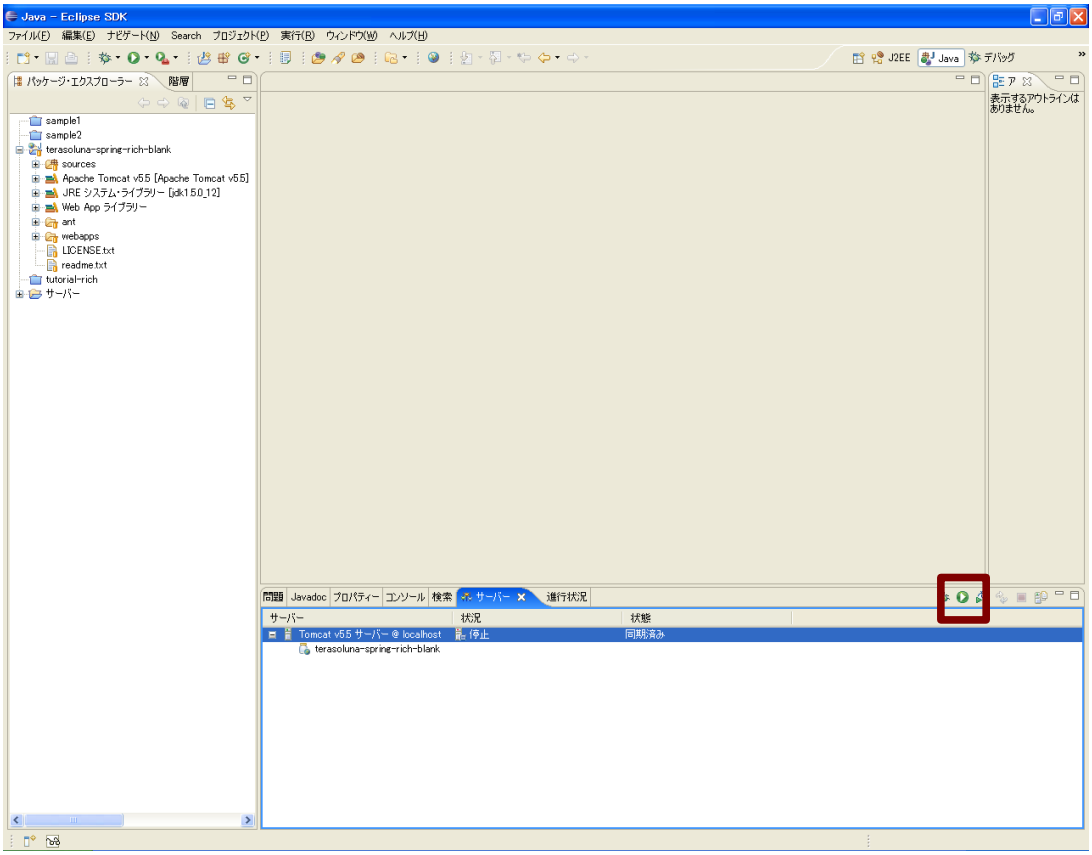
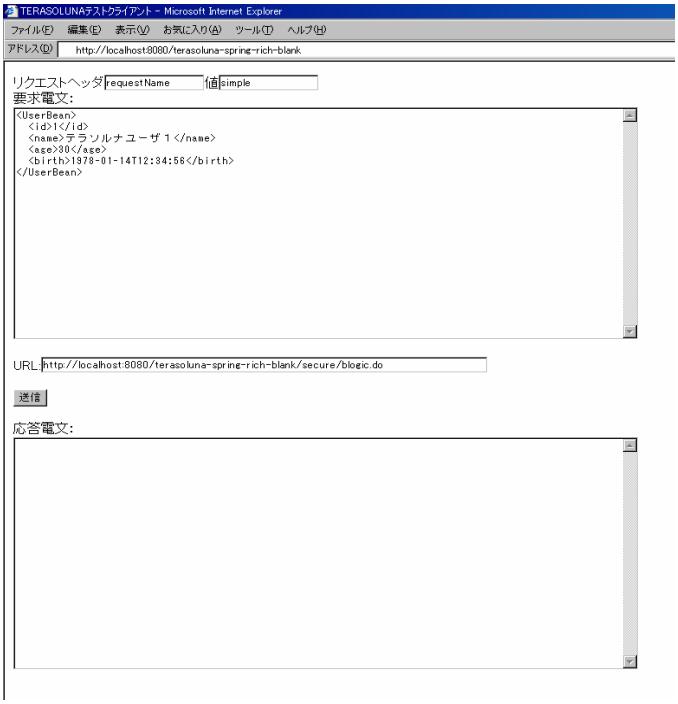


図6 Eclipse サーバ起動ボタン

※ここで、「Publishing failed」のウィンドウが表示された場合は、「OK」ボタンでウィンドウを閉じプロジェクトを選択し、右クリックメニューより更新を選択する。その後再度「サーバを起動」ボタンを押下する。

（2） ブラウザを開き、”http://localhost:8080/terasoluna-spring-rich-blank” にアクセスし、テストクライアント画面を表示する。画面上部のリクエストヘッダに“requestname”、値に“simple”と入力する。続いて、要求電文の欄に以下の XML を入力し、URL 欄に“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”と入力し、「送信ボタン」をクリックする。
※birth の項目は xs:dateTime 型のため、「yyyy-mm-ddThh:mm:ss」のデータフォーマットで登録すること。



(3) 要求電文に対するレスポンスが、応答電文として正しく表示されることを確認する。
(画面下部の応答電文に表示される)。
birth の項目は、リクエスト時とデータ形式が異なっているが問題ない。

この応答電文ではタグが改行やインデントされた形で表示されている。
改行やインデントするスペースなどの無駄なパケットを送信せずパフォーマンスを向上したい場合は、“castor.properties”というファイル名で以下の内容を記述し、クラスパス上に配置することでインデントされない形で応答電文が送信される。

- “castor.properties”ファイル

```
org.exolab.castor.indent=false
org.exolab.castor.xml.saveMapKeys=false
org.exolab.castor.xml.naming=mixed
```

-

■ **参考資料**

Rich 版機能説明書
『RA-01 リクエスト・コントローラマッピング機能』
『RA-02 コントローラ拡張機能』
『RB-01 リクエストデータ解析機能』
『RB-02 レスポンスデータ生成機能』
『RB-03 XML-Object 変換機能』
『RC-01 ビジネスロジック実行機能』

2.4 データベースアクセス

本節では、Rich 版を利用し、データベースから値を取得する方法とデータベースの値を更新する方法について学習する。

■ 概要

図のようにリクエストデータをデータベースに登録した後に、データベースのレコードを全件返す機能を実装する。ただし、本節ではデータベースアクセスに関する部分のみを作成し、エラー処理などは次節以降で作成する。

- リクエストデータ･･･ユーザ ID、ユーザ名、年齢、生年月日
- レスポンスデータ･･･総件数、ユーザ ID、ユーザ名、年齢、生年月日

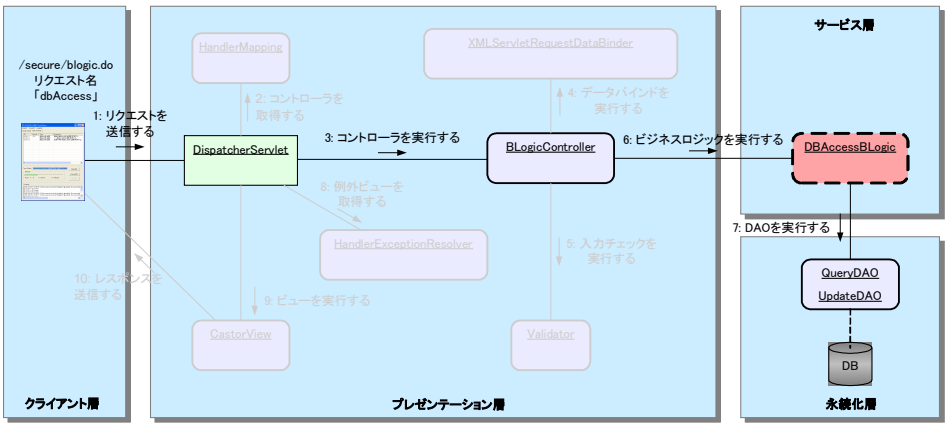


図7 データベースアクセス画面遷移図

1. “http://localhost:8080/terasoluna-spring-rich-blank”にアクセスし、テストクライアント画面を表示する。リクエストデータを作成して、リクエスト名“dbAccess”で、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。
2. リクエスト名“dbAccess”と“tutorial-controller.xml”ファイルをもとにコントローラを実行する。
3. コントローラでは、リクエストデータをビジネスロジックの入力クラスに変換し、“tutorial-businessLogic.xml”ファイルをもとにビジネスロジックを実行する。
4. ビジネスロジックでは、入力クラスのユーザ ID でデータベースを検索し、ユーザが存在するか確認する。
5. ユーザが存在しない場合には、入力クラスの値をデータベースに登録した後に、データベースのレコードを全件取得し出力クラスに格納して返却する。ユーザが存在する場合はここでは何もしない。
6. ビジネスロジックから返却されたクラスの値をクライアントに送信する。
7. テストクライアント画面に応答電文が正しく表示される。(画面下部の応答電文に表示される)。

このとき、以下の作業が必要となる。

- DAO クラスを利用したビジネスロジック実装クラスの作成
- “tutorial-businessLogic.xml”ファイルの変更
- “tutorial-controller.xml”ファイルの変更
- “sqlMap.xml”ファイルの変更
- “applicationAOP.xml”ファイルの変更
- “dataAccessContext-local.xml”ファイルの変更
- “jdbc.properties”ファイルの変更
- “hsqldb.jar”ファイルのビルドパスへの追加

■ **手順**

● **DAO クラスを利用したビジネスロジック実装クラスの作成**

ビジネスロジックとして以下の機能を実装する。

(1) UserBean クラスのユーザ ID でデータベースを検索し、存在するかチェックする。
(2) UserBean クラスのデータをデータベースに登録する。
(3) データベースのレコードを全件取得する。
(4) ResultData クラスに結果を格納し、返却する。

- DAO の getter/setter・・・TERASOLUNA Server Framework for Java 提供の DAO が Spring により設定（DI）されるため記述すること。

“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.blogic/DBAccessBLogic.java”ファイルを以下のように作成する。

```
/*
 * $Id$
 *
 * Copyright (c) 2007 NTT DATA Corporation
 *
 */
package jp.terasoluna.rich.tutorial.service.blogic;

import java.util.List;

import jp.terasoluna.rich.tutorial.service.bean.ResultData;
import jp.terasoluna.rich.tutorial.service.bean.UserBean;
import jp.terasoluna.fw.dao.QueryDAO;
import jp.terasoluna.fw.dao.UpdateDAO;
import jp.terasoluna.fw.service.rich.BLogic;

/**
 * データベースアクセスの BLogic クラス
 *
 */
public class DBAccessBLogic
        implements BLogic<UserBean, ResultData> {
    /**
     * DAO クラス。
     * Spring によりインスタンス生成され設定される。
     */
    private UpdateDAO updateDAO = null;

    /**
     * DAO クラス。
     * Spring によりインスタンス生成され設定される。
     */
    private QueryDAO queryDAO = null;

    /**
     * UpdateDAO を返却する。
     *
     * @return 保持する UpdateDAO
     */
    public UpdateDAO getUpdateDAO() {
        return updateDAO;
    }

    /**
     * UpdateDAO を設定する。
     *
     * @param updateDAO UpdateDAO
     */
```

```
    public void setUpdateDAO(UpdateDAO updateDAO) {
        this.updateDAO = updateDAO;
    }

    /**
     * QueryDAO を返却する。
     *
     * @return 保持する QueryDAO
     */
    public QueryDAO getQueryDAO() {
        return queryDAO;
    }

    /**
     * QueryDAO を設定する。
     *
     * @param queryDAO QueryDAO
     */
    public void setQueryDAO(QueryDAO queryDAO) {
        this.queryDAO = queryDAO;
    }

    /**
     * 入力クラスの引数のデータを登録して検索した結果を戻す。<br>
     * １．UserBean クラスのユーザ ID でデータベースを検索し、存在するかチェックする。<br>
     * ２．UserBean クラスのデータをデータベースに登録する。<br>
     * ３．データベースのレコードを全件取得する。<br>
     * ４．ResultData クラスに結果を格納し、返却する。
     *
     * @param inputData 入力クラス
     * @return 結果クラス
     */
    public ResultData execute(UserBean inputData) {

        //結果クラスの生成、結果の設定
        ResultData result = new ResultData();

        //入力情報ビジネスロジックの実行
        //返却値が、false の場合は重複エラー。
        //true の場合は登録処理が正常に終了。
        if (!check(inputData)) {

            //重複エラー時の処理
            return result;
        }

        //ユーザ情報を登録する。
        updateDAO.execute("insertUser", inputData);

        // ユーザ情報を保持するための List 型を宣言。
        List<UserBean> userBeanList = null;

        //ユーザ情報を全件取得する。
        userBeanList = queryDAO.executeForObjectList("getUserList",
                       null);
        result.setUserBean(userBeanList );
        result.setTotalCount(Integer.valueOf(userBeanList .size()));

        return result;
    }

    /**
     * 入力された ID が重複か否かをデータベースから取得して判定する。
     * 重複している場合は、false を返却する。
     * 重複していない場合は、true を返却する。
     *
     * @param inputData 入力された値を保持する JavaBean
     * @return チェック結果
     */
    private boolean check(UserBean inputData) {

        //ID からデータを検索する。
        Integer count
```

```
            = queryDAO.executeForObject("getUserCount", inputData, Integer.class);
        if (count != null && count.intValue() > 0 ) {
            return false;
        }

        //データが取得できなかった場合は、重複していないとして
        //true を返却する。
        return true;
    }

}
```

- **“tutorial-businessLogic.xml”ファイルの変更**

ビジネスロジック（DBAccessBLogic）の Bean 定義情報の追加を行う。

- DAO の Bean 定義・・・“dataAccessContext-local.xml”ファイルで定義している TERASOLUNA Server Framework for Java 提供の DAO をビジネスロジック（DBAccessBLogic）で使えるように定義する。

“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-businessLogic.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- サービス層の Bean 定義 -->
<!--                    -->
<!-- サービス層共通の Bean 定義は applicationContext.xml に記述する。 -->
  <!-- ======= ビジネスロジックの定義 ======= -->

  <!-- サービスの設定例 -->
  <!--
  <bean id="sumService"
      class="jp.terasoluna.rich.functionsample.controllerex.service.SumServiceImpl"
      scope="prototype"/>
  -->

  <!-- 2.3 単純なロジック simpleService 定義 -->
  <bean id="simpleService"
      class="jp.terasoluna.rich.tutorial.service.blogic.SimpleBLogic"
      scope="prototype">
  </bean>

  <!-- 2.4 データベースアクセス dbAccessService 定義 -->
  <bean id="dbAccessService"
      class="jp.terasoluna.rich.tutorial.service.blogic.DBAccessBLogic"
      scope="prototype">
    <property name="updateDAO" ref="updateDAO"/>
    <property name="queryDAO"  ref="queryDAO" />
  </bean>

</beans>
```

- **“tutorial-controller.xml”ファイルの変更**

ビジネスロジック（DBAccessBLogic）を呼び出すコントローラの定義を追加する。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-controller.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の Bean 定義 -->
<!--                                                                    -->
<!-- コントローラや相関入力チェッククラスの Bean 定義などを記述する。 -->
<!-- プレゼンテーション層共通の Bean 定義は blank-servlet.xml に記述する。 -->
  <!-- ======= コントローラ定義 ======= -->

  <!-- 動作確認用コントローラの設定 -->
  <bean name="/testController"
      class="org.springframework.web.servlet.mvc.SimpleFormController">
    <property name="commandClass" value="java.lang.Object"/>
    <property name="successView" value="success"/>
  </bean>

  <!-- コントローラの設定例 -->
  <!--
  <bean name="/xmlPOJOCastorController"
      class="jp.terasoluna.rich.functionsample.controllerex.controller.
             ControllerExController"
      parent="pojoXmlRequestCastorViewController" scope ="prototype">
    <property name="sumService" ref="sumService"/>
  </bean>
  -->

  <!-- 2.3 単純なロジック xml-BLogic-castor 定義 -->
  <bean name="/simpleController"
      parent="blogicXmlRequestCastorViewController" scope ="prototype">
    <property name="blogic" ref="simpleService"/>
  </bean>

  <!-- 2.4 データベースアクセス xml-BLogic-castor 定義 -->
  <bean name="/dbAccessController"
      parent="blogicXmlRequestCastorViewController" scope ="prototype">
    <property name="blogic" ref="dbAccessService"/>
  </bean>

</beans>
```

- **“sqlMap.xml”ファイルの変更**

① UserBean クラスのユーザ ID が USERTABLE に登録されている件数を取得する SQL を追加する。
② UserBean クラスの属性を USERTABLE テーブルに登録する SQL を追加する。
③ USERTABLE テーブルのレコードを全件取得する SQL を追加する。
“terasoluna-spring-rich-blank/sources/sqlMap.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。
※初期の“sqlMap.xml”ファイルには設定例がコメントで記述してあるが、削除するかそのままにしておくこと。

```
<?xml version="1.0" encoding="Windows-31J" ?>
<!DOCTYPE sqlMap
            PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
            "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="user">

<!-- SQLMap の設定例
  <select id="selectUserAuthenticated"
      parameterClass="jp.hotelsample.logon.bean.LogonInput"
      resultClass="jp.hotelsample.logon.bean.LogonOutput">
```

```
    SELECT USER_CODE AS userID, USER_NAME AS userName, USER_DIVISION AS auth FROM USERKA
RI WHERE USER_CODE = #userID# AND USER_PASSWORD = #password#
  </select>
-->

    <select id="getUserCount"
            parameterClass="jp.terasoluna.rich.tutorial.service.bean.UserBean"
            resultClass="java.lang.Integer">
        SELECT
            COUNT(ID)
        FROM
            USERTABLE
        WHERE
            ID = #id#
    </select>

    <insert id="insertUser"
        parameterClass="jp.terasoluna.rich.tutorial.service.bean.UserBean">
        INSERT INTO USERTABLE (
        ID,
        NAME,
        AGE,
        BIRTH
        )VALUES(
        #id#,
        #name#,
        #age#,
        #birth#)
    </insert>

    <select id="getUserList"
            resultClass="jp.terasoluna.rich.tutorial.service.bean.UserBean"
            resultSetType="SCROLL_INSENSITIVE">
        SELECT
            ID,
            NAME,
            AGE,
            BIRTH
        FROM
            USERTABLE
        ORDER BY
            ID
    </select>

</sqlMap>
```

- **“applicationAOP.xml”ファイルの変更**

トランザクション設定を有効にする。ここで、AOP でトランザクションをかける対象とする Bean の Bean 定義 ID を指定する。ここでは、Bean 定義 ID の接尾辞が「BLogic」「Service」の Bean に対して適用している。
“terasoluna-spring-rich-blank/webapps/WEB-INF/applicationAOP.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- AOP 設定 Bean 定義 -->

  <!-- トランザクション用オートプロキシ設定（Bean 定義ファイルの ID） -->
<!--
  <bean id="transactionAutoProxy"
      class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="interceptorNames">
      <list>
        <idref bean="transactionInterceptor"/>
      </list>
    </property>
    <property name="beanNames">
      <list>
        <value>*BLogic</value>
        <value>*Service</value>
      </list>
    </property>
  </bean>
-->
</beans>
```

- **“dataAccessContext-local.xml”ファイルの変更**

トランザクション設定情報を変更する。ここで、AOP の対象となるメソッド名を指定する。ここでは、どのメソッド名でも対象となるように「＊」としている。
“terasoluna-spring-rich-blank/webapps/WEB-INF/dataAccessContext-local.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">


<!--
    - データアクセス層のアプリケーションコンテキスト定義。
           - "applicationContext.xml"で定義されているビジネス層のオブジェクトからアクセスされる。
           - (web.xml の"contextConfigLocation"参照)。
           -->
                                          :
                                        略
                                          :
  <!-- =============== トランザクションの定義 =============== -->
  <!-- トランザクション定義情報 -->
  <bean id="attrSource"
      class="org.springframework.transaction.interceptor.NameMatchTransactionAttributeSou
rce">
    <property name="properties">
      <props>
        <prop key="insert*">PROPAGATION_REQUIRED,-java.lang.Exception</prop>
        <prop key="update*">PROPAGATION_REQUIRED,-java.lang.Exception</prop>
        <prop key="*">PROPAGATION_REQUIRED,readOnly,-java.lang.Exception</prop>
        <prop key="*">PROPAGATION_REQUIRED,-java.lang.Exception</prop>
      </props>
    </property>
  </bean>
                                          :
                                        略
                                          :

</beans>
```

- **“jdbc.properties”ファイルの変更**

インポートしたプロジェクトを hsqldb で使えるように設定する。
“terasoluna-spring-rich-blank/webapps/WEB-INF/jdbc.properties”を以下のように修正する。(網掛け部分を追加する)。

```
#ドライバー
jdbc.driverClassName=org.hsqldb.jdbcDriver

#URL
jdbc.url=jdbc:hsqldb:hsql://127.0.0.1:9001/terasoluna

#ユーザー名
jdbc.username=sa

#パスワード
jdbc.password=
```

なお、チュートリアルではベーシックなデータソースを設定しているが商用では AP サーバの JNDI リソースを利用すること。JNDI の使用方法は Rich 版機能説明書『CC-01 JNDI アクセス機能』参照のこと。

- **“hsqldb.jar”ファイルのビルドパスへの追加**

① エクスプローラにて“C:\hsqldb\lib\hsqldb.jar”ファイルを“C:\Eclipse\workspace\terasoluna-spring-rich-blank\webapps\WEB-INF\lib\”フォルダにコピーする。
② Eclipse 上でプロジェクトを選択し、右クリックメニューで更新を選択する。
③ Web アプリケーション・ライブラリに“hsqldb.jar”が含まれていることを確認する。

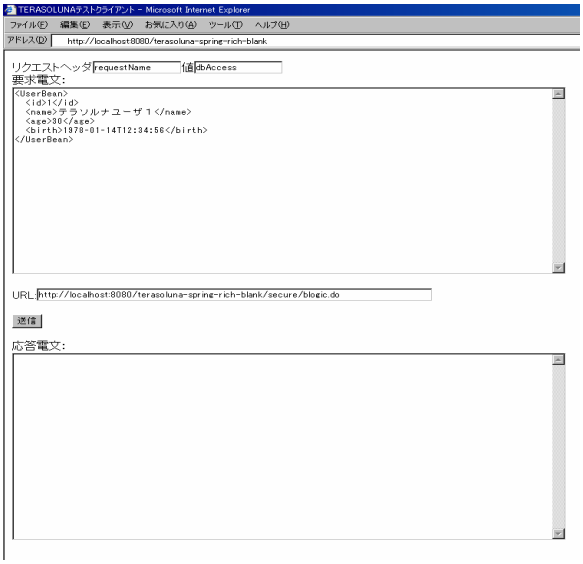
■ 確認

データベースから取得したユーザ情報を含んだ応答電文が正しく表示されることを確認する。

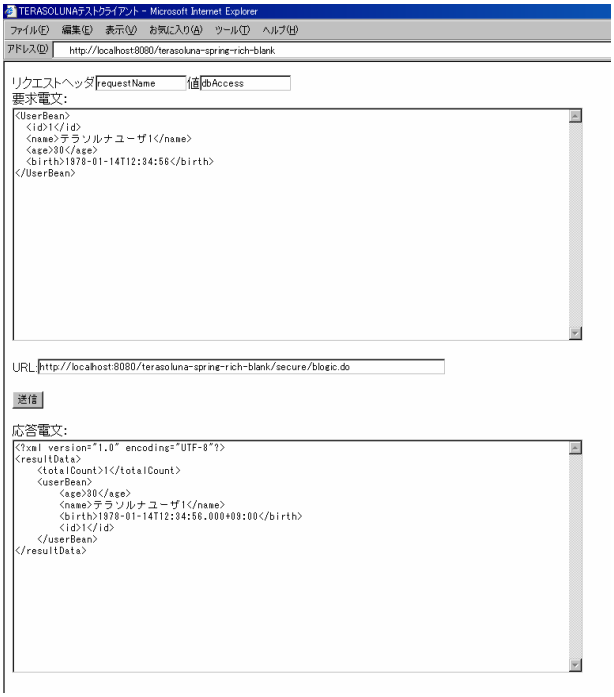
(1) 作成・更新したファイルを全て保存し、コンパイルを行ったのちに Tomcat を起動する。

(2) データベースを起動させるために“C:\hsqldb\terasoluna\startDB.bat”を実行する。

(3) まず、データベースに登録したデータが応答電文に表示されることの確認を行う。“http://localhost:8080/terasoluna-spring-rich-blank”**にアクセスし**、テストクライアント画面を表示する。画面上部のリクエストヘッダに“requestname”、値に“dbAccess”と入力する。続いて、要求電文の欄に以下の XML を入力し、URL 欄に“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”と入力し、「送信ボタン」をクリックする。

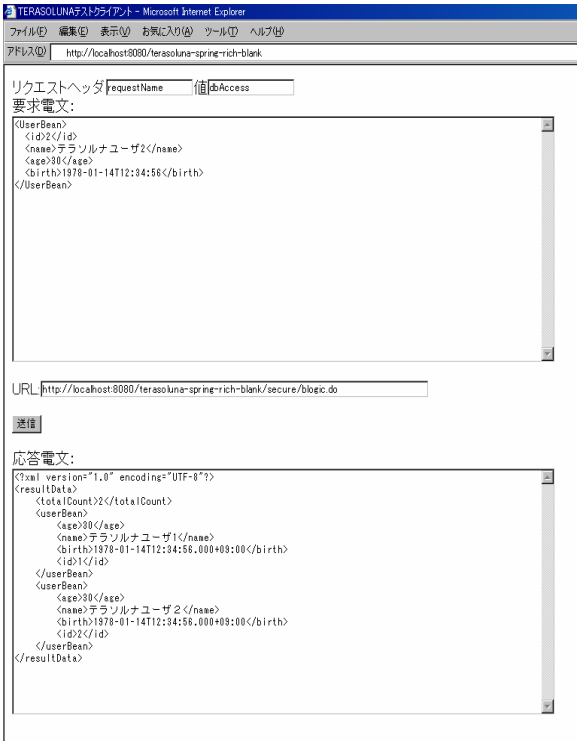


(4) データベースから取得したユーザ情報を含んだ応答電文が表示されることを確認する。



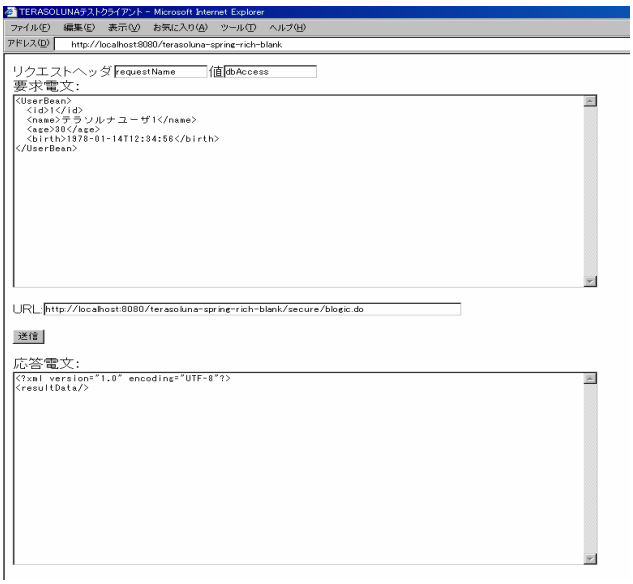
(5) 次に、データベースの全件数が応答電文に表示されることの確認を行う。テストクライアント画面より、以下の XML を入力し、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。

(6) データベースから取得した全ユーザ情報を含んだ応答電文が表示されることを確認する。



(7) 最後に、ユーザが存在する場合の応答電文の確認をする。テストクライアント画面より、(4)と同じXML を入力し、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。

(8) データベースにすでに同じユーザが存在するため、データ項目がない状態の応答電文が表示されることを確認する。



■ 参考資料

- Rich 版機能説明書
  『CA-01 トランザクション管理機能』
  『CB-01 データベースアクセス機能』

2.5 入力チェック

本節では、Rich 版を利用し、入力チェックについて学習する。
本節で説明する入力チェックには以下の種類がある。

(1) 電文形式チェック:リクエストデータの電文自体の妥当性チェックや型チェックを行う。
(2) 単項目チェック:commons-validator によって、必須チェックや範囲チェック、半角チェックなどを行う。
(3) 相関チェック:MultiFieldValidator を実装したクラスを使用し、複数項目の相関関係チェックを行う。
(4) DB 整合性チェック:ビジネスロジックで実装する。

Rich 版では番号順に入力チェックが実行される。
なお、本書では電文形式チェックと単項目チェックと相関チェックを実装する。

2.5.1 電文形式チェック

■ 概要

電文形式チェック機能についての動きは図のようになる。入力値にエラーがあった場合にはエラーメッセージを含んだレスポンスをテストクライアント画面に表示する。

- リクエストデータ・・・ユーザ ID、ユーザ名、年齢、生年月日
- レスポンスデータ・・・総件数、ユーザ ID、ユーザ名、年齢、生年月日

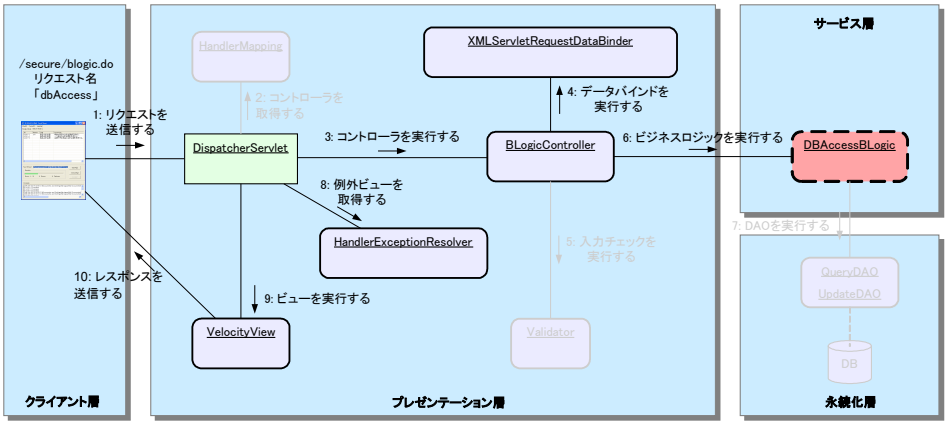


図8 電文形式チェック画面遷移図

1. “http://localhost:8080/terasoluna-spring-rich-blank” にアクセスし、テストクライアント画面を表示する。リクエストデータを作成して、リクエスト名“dbAccess”で、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。
2. リクエスト名“dbAccess”と“tutorial-controller.xml”ファイルをもとにコントローラを実行する。
3. コントローラでは、リクエストデータを入力クラスに変換する際に、スキーマ定義ファイルをもとに次のチェックを行う。
   - ユーザ ID・・・未入力、もしくは入力された値が半角数字ではない場合
   - ユーザ名・・・条件なし（文字列に対する電文形式チェックはチェック条件がないため。）
   - 年齢・・・未入力、もしくは入力された値が半角数字ではない場合
   - 生年月日・・・未入力、もしくは日付形式ではない場合
4. エラーが検出された場合、BindException が生成され、検出されたエラーに対応するエラーメッセージを VelocityView の“bindException.vm”テンプレートファイルを使ってクライアントに送信する。
5. 上記のチェックを全てクリアしている場合には、ビジネスロジックを実行し、データベースから取得したユーザの情報を含んだ応答電文がテストクライアント画面に正しく表示される。（画面下部の応答電文に表示される）。

電文形式チェック機能については、前節の単純なロジックの段階で既に実装された機能である。ここでは、

もう一度電文形式チェック機能を実装する手順を確認する。必要な手順は以下のようになる。

- スキーマ定義ファイルの確認
- “tutorial-servlet.xml”ファイルの確認

■ **手順**

● **スキーマ定義ファイルの作成**

リクエストデータに対し、電文形式チェックを行うために必要なスキーマ定義ファイルを確認する。

“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.bean/UserBean.xsd”ファイルが以下のようになっていることを確認する。

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="UserBean" type="UserBean-Type"/>

  <xs:complexType name="UserBean-Type">
    <xs:sequence>
      <xs:element name="id" type="xs:int" />
      <xs:element name="name" type="xs:string" />
      <xs:element name="age" type="xs:int" />
      <xs:element name="birth" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

● **“tutorial-servlet.xml”ファイルの確認及び変更**

電文形式チェック機能を実装するためにコントローラ定義を確認する。

⑤ “tutorial-controller.xml”ファイルで Bean 定義したコントローラの抽象クラスが xmlDataBinderCreator を設定（DI）していることを確認する。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルが以下のようになっていることを確認する。(網掛け部分を確認する)。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">


<!-- プレゼンテーション層の共通 Bean 定義 -->
                                                    :
                                                   略
                                                    :

    <!-- コントローラの抽象 Bean 定義
         ( ビジネスロジック：BLogic、受信リクエスト：XML 形式 )
    -->
    <bean id="blogicXmlRequestController" abstract="true" parent="blogicController">
        <property name="dataBinderCreator" ref="xmlDataBinderCreator"/>
    </bean>

    <!-- コントローラの抽象 Bean 定義
( ビジネスロジック：BLogic、受信リクエスト：XML 形式、ビュー：Castor )
            -->
            <bean id="blogicXmlRequestCastorViewController" abstract="true" parent="blogi
cXmlRequestController"/>
                                                    :
                                                   略
                                                    :

</beans>
```

⑥ xmlDataBinderCreator の Bean 定義が schemaValidator を設定（DI）していることを確認する。
ここの XMLServletRequestDataBinderCreator で電文形式チェックを行っている。
チュートリアルでは名前空間を意識した XML の送受信に対応するために、schemaValidator の Bean 定義を次の手順で確認する。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルが以下のようになっていることを確認する。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の共通 Bean 定義 -->

                                        :
                                       略
                                        :

  <bean id="xmlDataBinderCreator"
      class="jp.terasoluna.fw.web.rich.springmvc.bind.creator.XMLServletRequestDataBinder
Creator">
    <property name="oxmapper" ref="oxmapper"/>
    <property name="schemaValidator" ref="schemaValidator"/>
  </bean>

                                        :
                                       略
                                        :

</beans>
```

⑦ SchemaValidator の Bean 定義に名前空間の設定がされていることを確認する。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルが以下のようになっていることを確認する。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の共通 Bean 定義 -->

                                        :
                                       略
                                        :

  <bean id="schemaValidator"
      class="jp.terasoluna.fw.oxm.xsd.xerces.SchemaValidatorImpl">
  </bean>

                                        :
                                       略
                                        :

</beans>
```

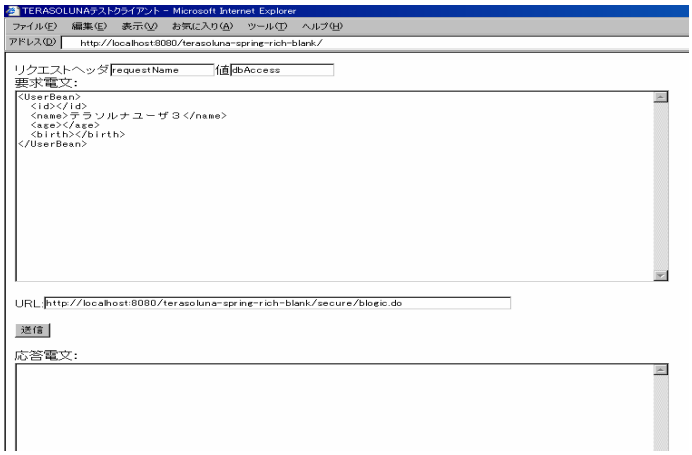
■ 確認

テストクライアント画面より、電文形式チェックでエラーになった場合の応答電文が正しく表示されることを確認する。

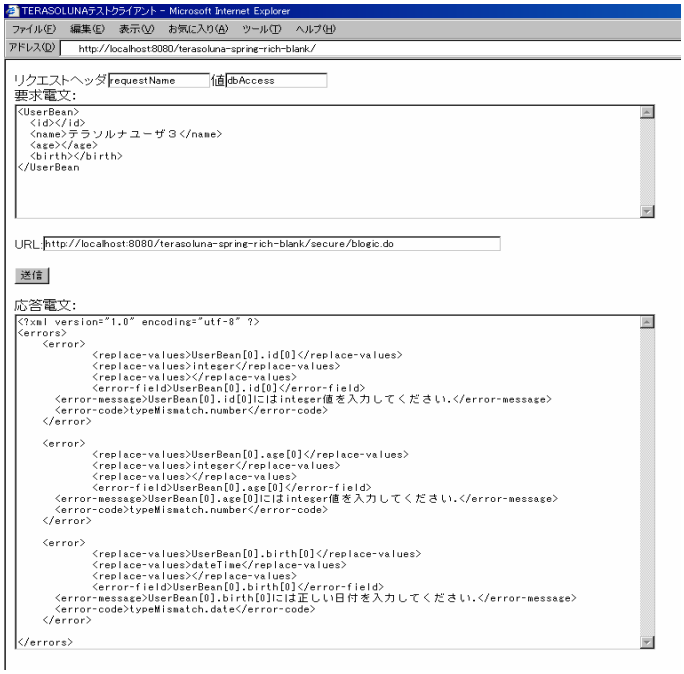
(1) 作成・更新したファイルを全て保存し、コンパイルを行ったのちに Tomcat を起動する。

(2) データベースを起動させるために“C:\hsqldb\terasoluna\startDB.bat”を実行する。

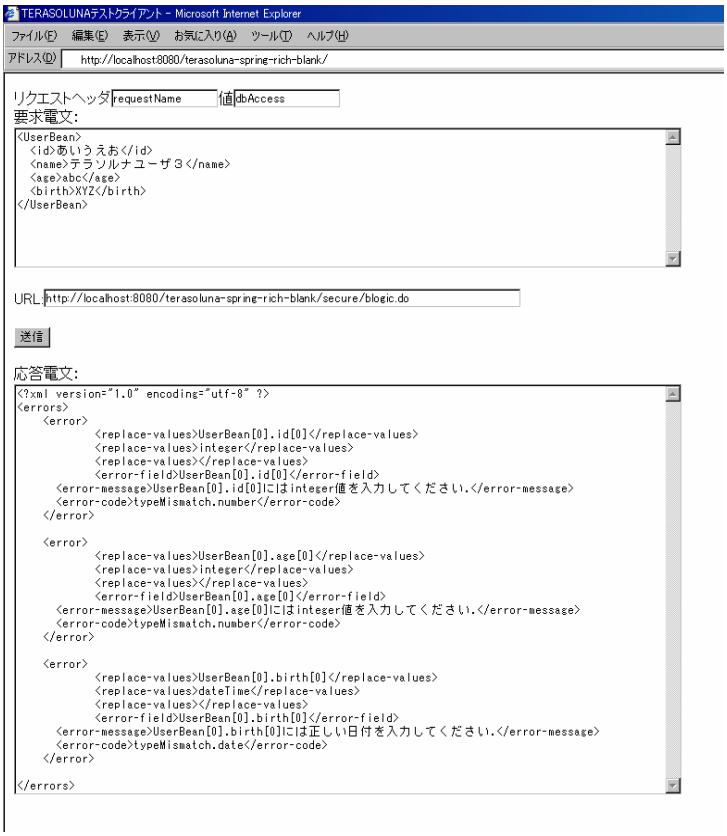
(3) まず、ユーザ ID、年齢、生年月日の必須チェックの動作確認を行う。“http://localhost:8080/terasoluna-spring-rich-blank”**にアクセスし**、テストクライアント画面を表示する。画面上部のリクエストヘッダに“requestname”、値に“dbAccess”と入力する。続いて、要求電文の欄に以下の XML のようにユーザ ID、年齢、生年月日を空文字で入力し、URL 欄に“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”と入力し、「送信ボタン」をクリックする。



(4) 「UserBean[0].id[0]には integer 値を入力してください.」「UserBean[0].age[0]には integer 値を入力してください.」「UserBean[0].birth[0]には正しい日付を入力してください.」というエラーメッセージを含む応答電文が、テストクライアント画面に表示されることを確認する。



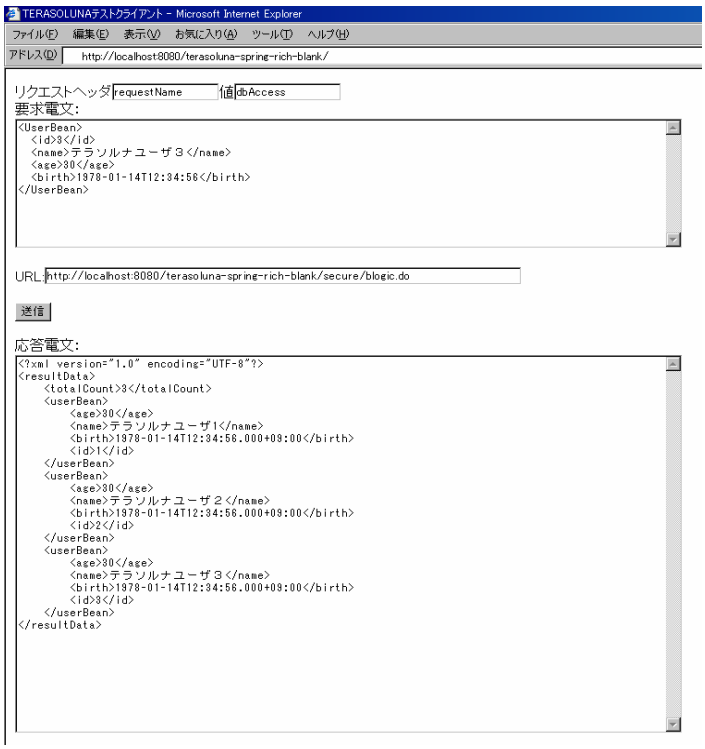
(5) 次に、ユーザ ID、年齢の数字文字列チェック及び生年月日の日付チェックの動作確認を行う。テストクライアント画面より、以下の XML のようにユーザ ID に“あいうえお”、年齢に“abc”、生年月日に“XYZ”と入力し、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。



(6) 「UserBean[0].id[0]には integer 値を入力してください.」「UserBean[0].age[0]には integer 値を入力してください.」「UserBean[0].birth[0]には正しい日付を入力してください.」というエラーメッセージを含む応答電文が、テストクライアント画面に表示されることを確認する。

(7) 最後に、エラーがなかった場合の動作確認を行う。テストクライアント画面より、以下の XML を入力し、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。

(8) テストクライアント画面にデータベースから取得したユーザの情報を含んだ応答電文が正しく表示されることを確認する。



■ 参考資料

- Rich 版機能説明書
  『RF-01 形式チェック機能』
  『RD-01 例外ハンドリング機能』

2.5.2 単項目入力チェック

■ **概要**

図のように、単項目入力チェック機能を追加する。入力値にエラーがあった場合にはエラーメッセージを含んだレスポンスをテストクライアント画面に表示する。

- リクエストデータ・・・ユーザ ID、ユーザ名、年齢、生年月日
- レスポンスデータ・・・総件数、ユーザ ID、ユーザ名、年齢、生年月日

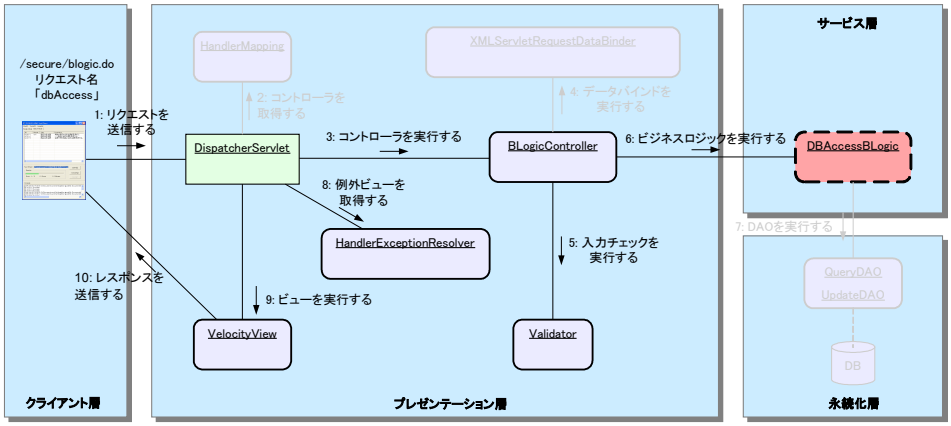


図9 単項目入力チェック画面遷移図

1. “http://localhost:8080/terasoluna-spring-rich-blank” **にアクセスし**、テストクライアント画面を表示する。リクエストデータを作成して、リクエスト名“dbAccess”で、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。
2. リクエスト名“dbAccess”と“tutorial-controller.xml”ファイルをもとにコントローラを実行する。
3. コントローラでは、リクエストデータをビジネスロジックの入力クラスに変換する。
4. 変換した入力クラスに対して“validation.xml”ファイルをもとに次のチェックを行う。
   - ユーザ ID・・・5 文字を超える場合
   - ユーザ名・・・未入力、もしくは 20 文字を超える場合
   - 年齢・・・3 文字を超える場合
   - 生年月日・・・条件なし（Date 型の日付形式チェックは電文形式チェックで行われているため。）
5. エラーが検出された場合、BindException が生成され、検出されたエラーに対応するエラーメッセージを VelocityView の“bindException.vm”テンプレートファイルを使ってクライアントに送信する。
6. 上記のチェックを全てクリアしている場合には、ビジネスロジックを実行し、データベースから取得したユーザの情報を含んだ応答電文がテストクライアント画面に正しく表示される。（画面下部の応答電文に表示される）。

このとき、以下の作業が必要となる。

- “validation.xml”ファイルの変更
- “applicationContext.xml”ファイルの確認
- “tutorial-servlet.xml”ファイルの変更

■ **手順**

● **“validation.xml”ファイルの変更**

単項目入力チェックとして以下のチェック条件を実装する。
① ユーザ ID・・・最大桁数が 5 であること。
② ユーザ名・・・必須項目であること。最大桁数が 20 であること。
③ 年齢・・・最大桁数が 3 であること。

“terasoluna-spring-rich-blank/webapps/WEB-INF/validation/validation.xml”ファイルを変更する。(網掛け部分を追加する)。
※初期の“validation.xml”ファイルには設定例がコメントで記述してあるが、削除するかそのままにしておくこと。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE form-validation PUBLIC
     "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"
     "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">

<form-validation>
  <formset>
    <!-- 単項目の入力チェック -->
    <form name="userBean">
        <field property="id"
            depends="maxLength">
          <arg key="id" position="0"/>
          <arg key="${var:maxlength}" resource="false" position="1"/>
          <var>
            <var-name>maxlength</var-name>
            <var-value>5</var-value>
          </var>
        </field>
        <field property="name"
            depends="required,maxLength">
          <arg key="name" position="0"/>
          <arg key="${var:maxlength}" resource="false" position="1"/>
          <var>
            <var-name>maxlength</var-name>
            <var-value>20</var-value>
          </var>
        </field>
        <field property="age"
            depends="maxLength">
          <arg key="age" position="0"/>
          <arg key="${var:maxlength}" resource="false" position="1"/>
          <var>
            <var-name>maxlength</var-name>
            <var-value>3</var-value>
          </var>
        </field>
    </form>

  </formset>
</form-validation>
```

- **“applicationContext.xml”ファイルの確認**

“validation.xml”ファイルが設定されているところを確認する。ここで単項目入力チェック機能が定義されている。
“terasoluna-spring-rich-blank/webapps/WEB-INF/applicationContext.xml”ファイルが以下のようになっていることを確認する。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- アプリケーション全体の Bean 定義 -->
  <!-- ========================= 一般的な定義  ========================= -->
                                          :
                                         略
                                          :
  <!-- ==================== 入力チェックの定義 ==================== -->

  <!-- 入力チェッククラスを生成するファクトリ -->
  <bean id="validatorFactory"
      class="jp.terasoluna.fw.validation.springmodules.DefaultValidatorFactoryEx">
    <property name="validationConfigLocations">
      <list>
        <value>/WEB-INF/validation/validator-rules.xml</value>
        <value>/WEB-INF/validation/validator-rules-ex.xml</value>
        <value>/WEB-INF/validation/validation.xml</value>
      </list>
    </property>
  </bean>

  <!-- デフォルト入力チェッククラス クラス名を Form 名として扱う -->
  <bean id="beanValidator"
      class="org.springmodules.commons.validator.DefaultBeanValidator">
    <property name="validatorFactory" ref="validatorFactory"/>
  </bean>

</beans>
```

※ validation.xml の form タグの name 属性に絶対パスで記述する場合は、DefaultBeanValidator クラスの Bean 定義にプロパティ useFullQualifiedClassName を追加し、ture を設定する。

- **“tutorial-servlet.xml”ファイルの変更**

先程確認した単項目入力チェック機能をコントローラ定義の抽象定義 BLogicController に設定（DI）する。これにより、データベースアクセス機能に単項目入力チェック機能が追加される。
“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-servlet.xml”ファイルを変更する。（網掛け部分を追加する）。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の共通 Bean 定義 -->
                                                :
                                               略
                                                :
    <!-- コントローラ抽象 Bean 定義
         ( ビジネスロジック : BLogic )
    -->
    <bean id="blogicController" abstract="true" class="jp.terasoluna.fw.web.rich.springmv
c.controller.BLogicController">
        <property name="ctxSupport" ref="ctxSupport"/>
        <property name="validator" ref="beanValidator" />
    </bean>
                                                :
                                               略
                                                :
</beans>
```

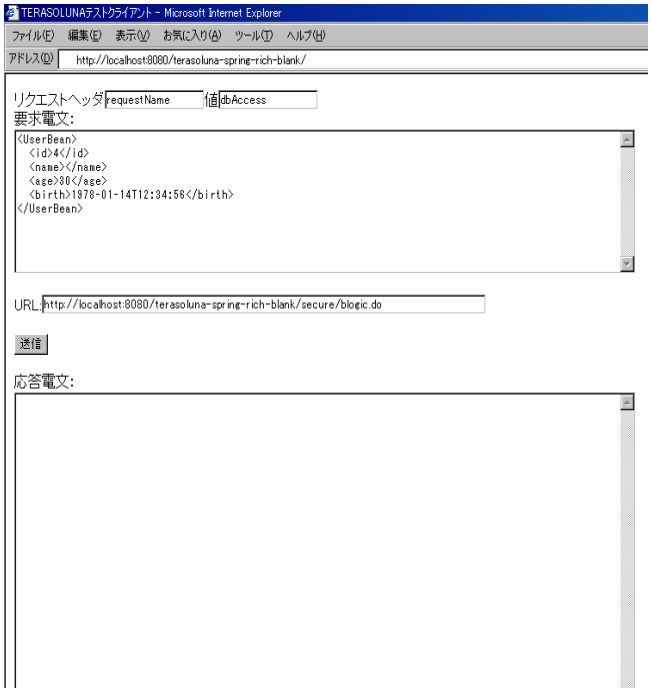
■ **確認**

テストクライアント画面より、単項目入力チェックでエラーになった場合の応答電文が正しく表示されることを確認する。

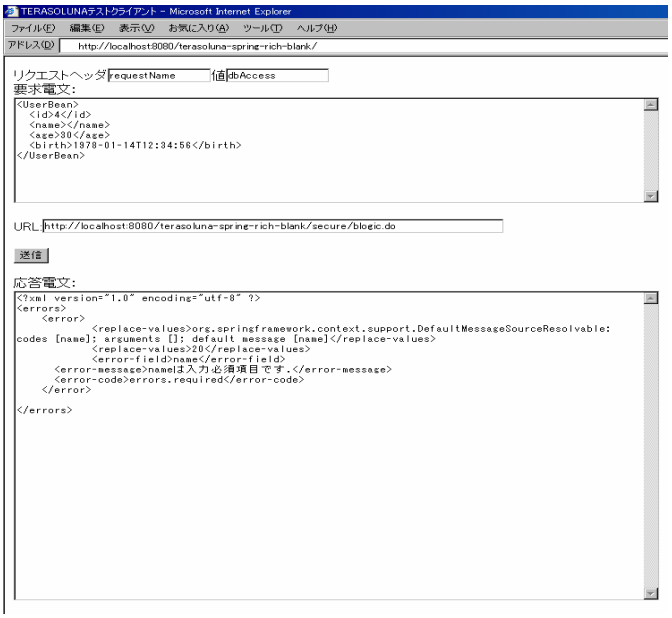
(1) 作成・更新したファイルを全て保存し、コンパイルを行ったのちに Tomcat を起動する。

(2) データベースを起動させるために“C:\hsqldb\terasoluna\startDB.bat”を実行する。

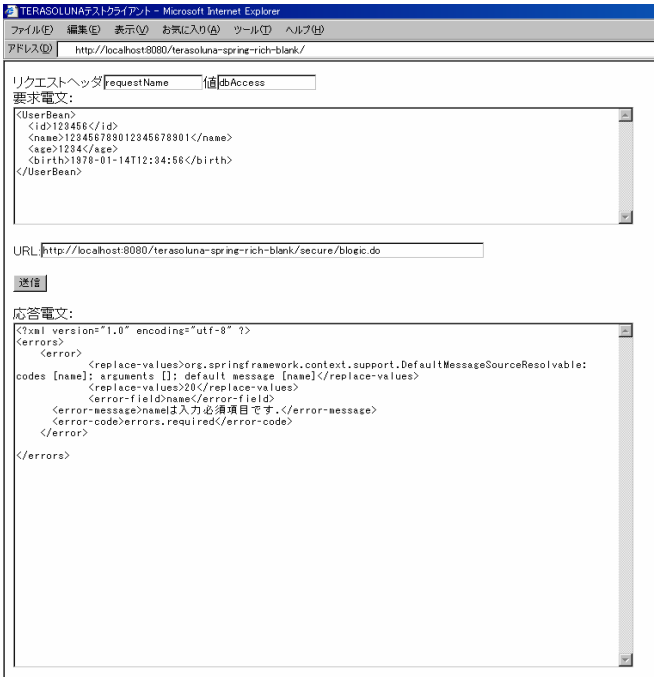
(3) まず、ユーザ名の必須チェックの動作確認を行う。“http://localhost:8080/terasoluna-spring-rich-blank”にアクセスし、テストクライアント画面を表示する。画面上部のリクエストヘッダに“requestname”、値に“dbAccess”と入力する。続いて、要求電文に以下の XML のようにユーザ名を空文字で入力し、URL 欄に“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”と入力し、「送信ボタン」をクリックする。



(4) 「name は入力必須項目です.」というエラーメッセージを含む応答電文が、テストクライアント画面に表示されることを確認する。



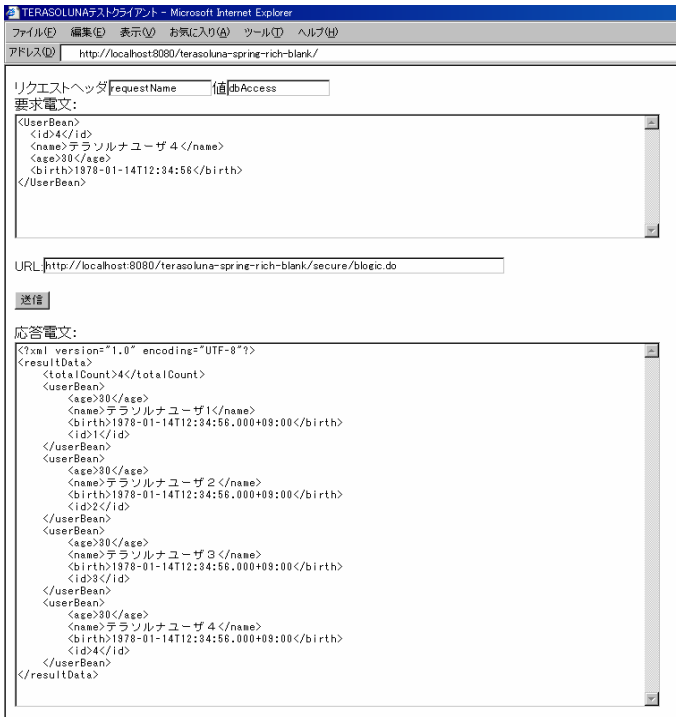
(5) 次に、ユーザ ID、ユーザ名、年齢の文字数チェックの動作確認を行う。テストクライアント画面より、以下の XML のようにユーザ ID に“123456”、ユーザ名に“123456789012345678901”、年齢に“1234”と入力し、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。



(6) 「id には 5 文字以下にしてください.」「name には 20 文字以下にしてください.」「age には 3 文字以下にしてください.」というエラーメッセージを含む応答電文が、テストクライアント画面に表示されることを確認する。

(7) 最後に、エラーがなかった場合の動作確認を行う。テストクライアント画面より、以下の XML を入力し、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。

(8) テストクライアント画面にデータベースから取得したユーザの情報を含んだ応答電文が正しく表示されることを確認する。



■ 参考資料

- Rich 版機能説明書
  『RF-02 入力チェック機能』
  『RD-01 例外ハンドリング機能』

2.5.3 相関入力チェック

■ **概要**

図のように、相関入力チェック機能を追加する。入力値にエラーがあった場合にはエラーメッセージを含んだレスポンスをテストクライアント画面に表示する。

- リクエストデータ・・・ユーザ ID、ユーザ名、年齢、生年月日
- レスポンスデータ・・・総件数、ユーザ ID、ユーザ名、年齢、生年月日

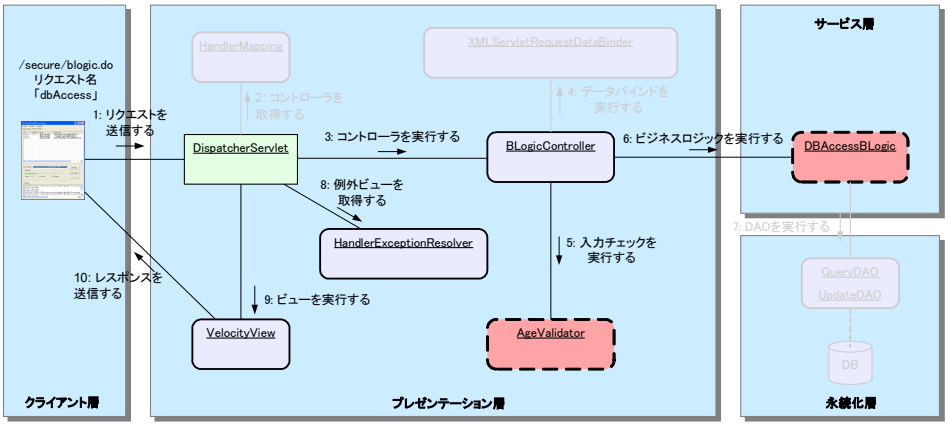


図10 相関入力チェック画面遷移図

1. “http://localhost:8080/terasoluna-spring-rich-blank” にアクセスし、テストクライアント画面を表示する。リクエストデータを作成して、リクエスト名“dbAccess”で、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。
2. リクエスト名“dbAccess”と“tutorial-controller.xml”ファイルをもとにコントローラを実行する。
3. コントローラでは、リクエストデータをビジネスロジックの入力クラスに変換する。
4. 変換した入力クラスに対して、相関チェッククラスが次のチェックを行う。
   - 年齢と生年月日・・・入力された生年月日と年齢が現在日付から計算して誤っている場合
5. エラーが検出された場合、BindException が生成され、検出されたエラーに対応するエラーメッセージを VelocityView の“bindException.vm”テンプレートファイルを使ってクライアントに送信する。
6. 上記のチェックを全てクリアしている場合には、ビジネスロジックを実行し、データベースから取得したユーザの情報を含んだ応答電文がテストクライアント画面に正しく表示される。（画面下部の応答電文に表示される）。

このとき、以下の作業が必要となる。

- 相関チェッククラスの作成
- “tutorial-controller.xml”ファイルの変更
- “applicationContext.xml”ファイルの変更
- “application-messages.properties”ファイルの変更

■ **手順**

● **相関チェッククラスの作成**

相関チェッククラスとして BaseMultiFieldValidator を実装した AgeValidator を作成し、以下の機能を実装する。

(1) 相関チェック対象のオブジェクトが UserBean クラスかの判定をする。
(2) 入力された生年月日と年齢と現在日付の相関チェックをする。
(3) 結果値を戻す。

- errors.rejectValue・・・エラーを追加するメソッド。このメソッドを用いて相関チェックエラーを追加する。void rejectValue(String field, String errorCode, Object[] errorArgs,String defaultMessage)。field に JavaBean のプロパティ名、errorCode にはリソースバンドルのキー、errorArgs は置換文字列、defaultMessage はデフォルトメッセージを指定する。

“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.validation/AgeValidator.java”ファイルを以下のように作成する。

```
/*
 * $Id$
 *
 * Copyright (c) 2007 NTT DATA Corporation
 *
 */
package jp.terasoluna.rich.tutorial.service.validation;

import java.util.Calendar;
import java.util.Date;

import org.springframework.validation.Errors;

import jp.terasoluna.fw.util.DateUtil;
import jp.terasoluna.fw.validation.springmodules.BaseMultiFieldValidator;
import jp.terasoluna.rich.tutorial.service.bean.UserBean;


/**
 * 入力された年齢と、生年月日の相関入力チェックを行う。
 *
 */
public class AgeValidator extends BaseMultiFieldValidator  {

    /**
     * 入力された年齢と、生年月日の相関入力チェックを行う。
     *
     * @param obj 検査対象の JavaBean ( UserBean )
     * @param errors エラー
     */
    @Override
    protected void validateMultiField(Object obj, Errors errors) {

        // キャスト可能かのチェック
        if (!(obj instanceof UserBean)) {
            errors.reject("error.common.00001");
            return;
        }

        // キャスト
        UserBean userBean = (UserBean) obj;

        // 年齢と日付の取得
        int age = userBean.getAge();
        Date birth = userBean.getBirth();
```

```
        // 入力された年齢と、生年月日の相関入力チェック
        if (!validate(birth,age)){
            String[] msg = {"age","birth"};
            errors.rejectValue("birth",
                    "error.multiField", msg, null);
        }

    }

    /**
     * 入力された年齢と、生年月日の相関入力チェックを行う。
     *
     * @param birth 生年月日
     * @param age 年齢
     * @return チェック結果
     */
    private boolean validate(Date birth, int age) {
        // 生年月日＋年齢の Calendar を生成
        Calendar calendarBirth = Calendar.getInstance();
        calendarBirth.setTime(birth);
        calendarBirth.add(Calendar.YEAR,age);

        // 生年月日＋年齢＋１年の Calendar を生成
        Calendar calendarNextYear = Calendar.getInstance();
        calendarNextYear.setTime(birth);
        calendarNextYear.add(Calendar.YEAR,age+1);

        // 現在日付の Calendar を生成
        Calendar calendarSystemTime = Calendar.getInstance();
        calendarSystemTime.setTime(DateUtil.getSystemTime());

        if ( calendarBirth.before(calendarSystemTime)
            && calendarNextYear.after(calendarSystemTime) ){
            return true;
        } else {
            return false;
        }
    }

}
```

● **“tutorial-controller.xml”ファイルの変更**

データベースアクセス機能に相関入力チェック機能を加えるために定義情報の変更を行う。

- validator・・・“ageValidator”とする。(これとあわせて、後述の“applicationContext.xml”ファイルの修正を行うことで相関入力チェック機能が設定(DI)される)。

“terasoluna-spring-rich-blank/webapps/WEB-INF/tutorial-controller.xml”ファイルを以下のように変更する。(網掛け部分を追加する)。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- プレゼンテーション層の Bean 定義 -->
<!--                                                                    -->
<!-- コントローラや相関入力チェッククラスの Bean 定義などを記述する。 -->
<!-- プレゼンテーション層共通の Bean 定義は blank-servlet.xml に記述する。 -->
  <!-- ======= コントローラ定義 ======= -->

  <!-- 動作確認用コントローラの設定 -->
  <bean name="/testController"
      class="org.springframework.web.servlet.mvc.SimpleFormController">
    <property name="commandClass" value="java.lang.Object" />
    <property name="successView" value="success />
  </bean>

  <!-- コントローラの設定例 -->
  <!--
  <bean name="/xmlPOJOCastorController"
      class="jp.terasoluna.rich.functionsample.controllerex.controller.
             ControllerExController"
      parent="pojoXmlRequestCastorViewController" scope ="prototype">
    <property name="sumService" ref="sumService"/>
  </bean>
  -->

  <!-- 2.3 単純なロジック xml-BLogic-castor 定義 -->
  <bean name="/simpleController"
      parent="blogicXmlRequestCastorViewController" scope ="prototype">
    <property name="blogic" ref="simpleService"/>
  </bean>

  <!-- 2.4 データベースアクセス xml-BLogic-castor 定義 -->
  <bean name="/dbAccessController"
      parent="blogicXmlRequestCastorViewController" scope ="prototype">
    <property name="blogic" ref="dbAccessService"/>
    <property name="validator" ref="ageValidator"/>
  </bean>

</beans>
```

- **“applicationContext.xml”ファイルの変更**

相関入力チェッククラスの定義情報の追加を行う。

- ageValidator の validatorFactory・・・“validatorFactory”とする。（これにより、相関入力チェック機能だけでなく単項目入力チェック機能も実装される）。

“terasoluna-spring-rich-blank/webapps/WEB-INF/applicationContext.xml”ファイルを以下のように変更する。（網掛け部分を追加する）。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
pring-beans-2.0.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spr
ing-util-2.0.xsd">

<!-- アプリケーション全体の Bean 定義 -->
  <!-- ========================= 一般的な定義  ========================= -->
                                          :
                                         略
                                          :

  <!-- リソースバンドルの設定（プロパティファイル利用） -->
  <bean id="messageSource"
      class="org.springframework.context.support.ResourceBundleMessageSource">
            <property name="basenames" value="ApplicationResources,system-messages,applic
ation-messages"/>
  </bean>


                                          :
                                         略
                                          :
  <!-- ==================== 入力チェックの定義 ==================== -->

  <!-- 入力チェッククラスを生成するファクトリ -->
  <bean id="validatorFactory"
      class="jp.terasoluna.fw.validation.springmodules.DefaultValidatorFactoryEx">
    <property name="validationConfigLocations">
      <list>
        <value>/WEB-INF/validation/validator-rules.xml</value>
        <value>/WEB-INF/validation/validator-rules-ex.xml</value>
        <value>/WEB-INF/validation/validation.xml</value>
      </list>
    </property>
  </bean>

  <!-- デフォルト入力チェッククラス クラス名を Form 名として扱う -->
  <bean id="beanValidator"
      class="org.springmodules.commons.validator.DefaultBeanValidator">
    <property name="validatorFactory" ref="validatorFactory"/>
  </bean>

  <!-- 相関入力チェッククラス -->
  <bean id="ageValidator"
    class="jp.terasoluna.rich.tutorial.service.validation.AgeValidator">
    <property name="validatorFactory" ref="validatorFactory"/>
  </bean>

</beans>
```

- **“application-messages.properties”ファイルの変更**

相関入力チェッククラスでエラーとなった場合のメッセージの追加を行う。
“terasoluna-spring-rich-blank/sources/application-messages.properties”ファイルを以下のように変更する。
(網掛け部分を追加する)。

```
################################################################################
##
## GlobalMessageResources で読み込むメッセージリソースファイル。
##
## このファイルには、業務共通のメッセージを記述する。
## 詳細は、GlobalMessageResources の JavaDoc を参照のこと。
##
################################################################################

#共通
error.common.00001=システムに例外が発生しました。管理者に問い合わせてください。

#Validation
error.multiField=[{0}]から計算した[{1}]が入力された[{1}]と一致しません。
```

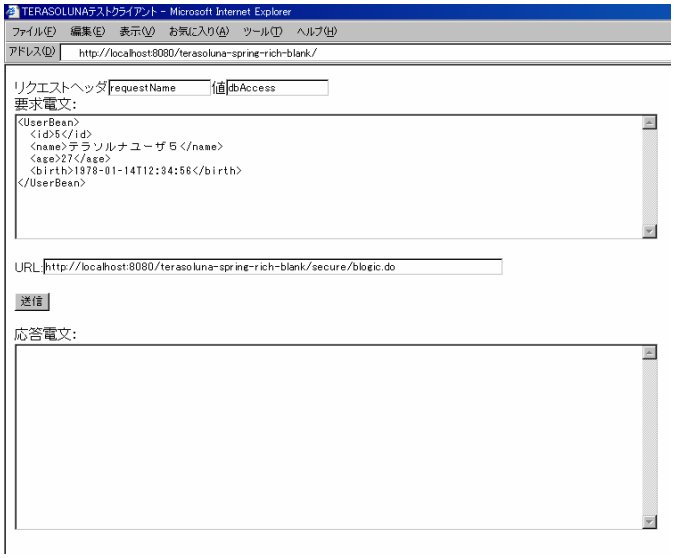
■ **確認**

テストクライアント画面より、相関入力チェックでエラーになった場合の応答電文が正しく表示されることを確認する。

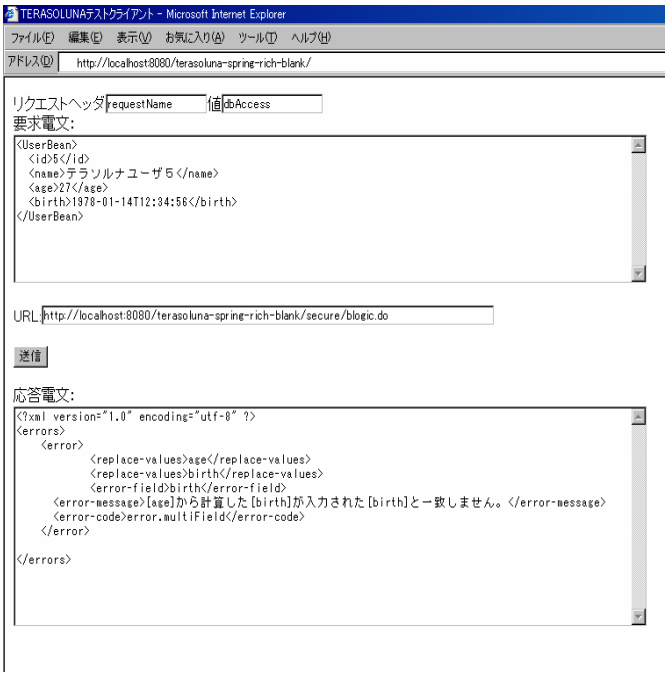
(1) 作成・更新したファイルを全て保存し、コンパイルを行ったのちに Tomcat を起動する。

(2) データベースを起動させるために“C:\hsqldb\terasoluna\startDB.bat”を実行する。

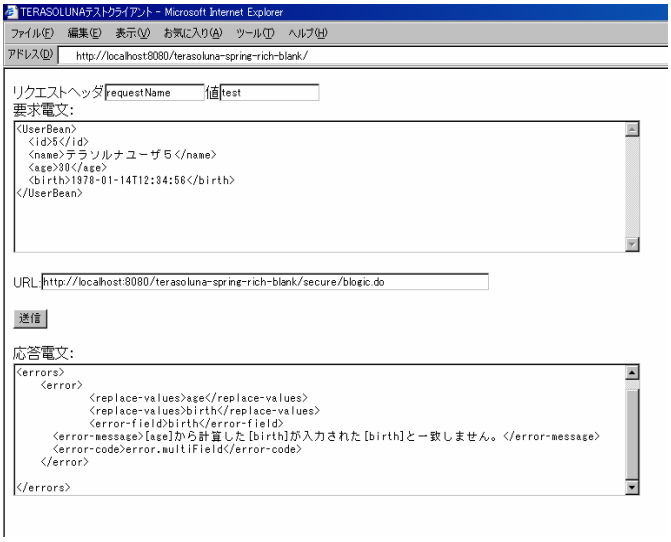
(3) 年齢、生年月日の相関チェックの動作確認を行う。“http://localhost:8080/terasoluna-spring-rich-blank”にアクセスし、テストクライアント画面を表示する。画面上部のリクエストヘッダに“requestname”、値に“dbAccess”と入力する。続いて、要求電文に以下の XML のように年齢に“27”、生年月日に“1978-01-14T12:34:56”と入力し、URL 欄に“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”と入力し、「送信ボタン」をクリックする。



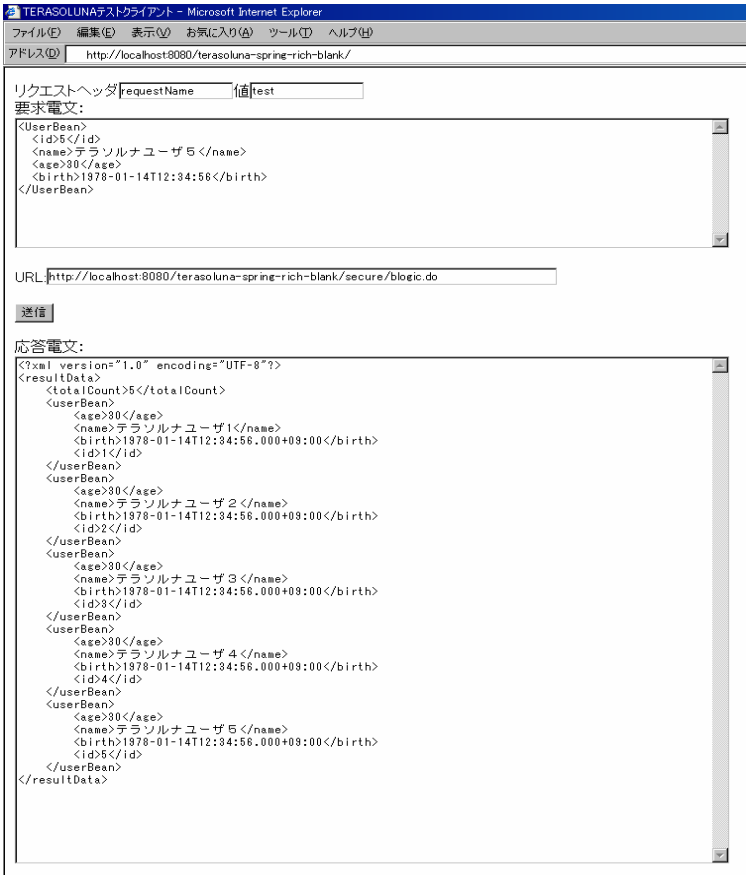
(4) 「[age]から計算した[birth]が入力された[birth]と一致しません。」というエラーメッセージを含む応答電文が、テストクライアント画面に表示されることを確認する。



(5) エラーがなかった場合の動作確認を行う。テストクライアント画面より、以下の XML を入力し、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。
※確認を行う年によって“age”の値を適宜変更すること。age の"30"は 2008 年 2 月時点の例である。



(6) テストクライアント画面にデータベースから取得したユーザの情報を含んだ応答電文が正しく表示されることを確認する。



■ **参考資料**

http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do

- Rich 版機能説明書
  『RF-02 入力チェック機能』
  『RD-01 例外ハンドリング機能』

2.6　例外処理

本節では、アプリケーションで例外を発生させる場合に、Rich 版が提供している例外（ServiceException）クラスを利用し、例外処理を行う方法に関して学習する。

■ 概要

図のように、前節までに作成したデータベースアクセス機能に例外処理を追加する。例外が発生した場合にはエラーメッセージを含んだレスポンスをテストクライアント画面に表示する

- リクエストデータ・・・ユーザ ID、ユーザ名、年齢、生年月日
- レスポンスデータ・・・総件数、ユーザ ID、ユーザ名、年齢、生年月日

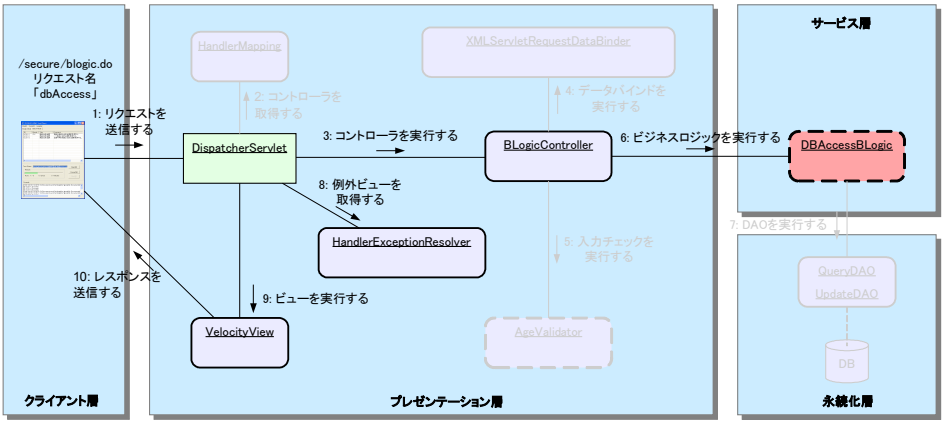


図11　例外処理画面遷移図

1. “http://localhost:8080/terasoluna-spring-rich-blank” にアクセスし、テストクライアント画面を表示する。リクエストデータを作成して、リクエスト名“dbAccess”で、“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”へ送信する。
2. リクエスト名“dbAccess”と“tutorial-controller.xml”ファイルをもとにコントローラを実行する。
3. コントローラでは、リクエストデータをビジネスロジックの入力クラスに変換し、“tutorial-businessLogic.xml”ファイルをもとにビジネスロジックを実行する。
4. ビジネスロジックでは、入力クラスのユーザ ID でデータベースを検索し、ユーザが存在するため、ServiceException を throw する。
5. ServiceException のエラーコードをもとに、VelocityView の“systemException.vm”テンプレートファイルを使ってクライアントに送信する。
6. テストクライアント画面にエラーメッセージを含んだ応答電文が正しく表示される。（画面下部の応答電文に表示される）。

このとき、以下の作業が必要となる。

- ビジネスロジック実装クラスの変更
- “application-messages.properties”ファイルの変更

■ **手順**

● **ビジネスロジック実装クラスの変更**

ユーザ ID でデータベースを検索し、ユーザが存在する場合の処理に ServiceException を throw する処理を追加する。

“terasoluna-spring-rich-blank/sources/jp.terasoluna.rich.tutorial.service.blogic/DBAccessBLogic.java”ファイルを以下のように変更する。（網掛け部分を追加する）。

```
/*
 * $Id$
 *
 * Copyright (c) 2007 NTT DATA Corporation
 *
 */
package jp.terasoluna.rich.tutorial.service.blogic;

import java.util.List;

import jp.terasoluna.rich.tutorial.service.bean.ResultData;
import jp.terasoluna.rich.tutorial.service.bean.UserBean;
import jp.terasoluna.fw.dao.QueryDAO;
import jp.terasoluna.fw.dao.UpdateDAO;
import jp.terasoluna.fw.service.rich.BLogic;
import jp.terasoluna.fw.service.rich.exception.ServiceException;

/**
 * データベースアクセスの BLogic クラス
 *
 */
public class DBAccessBLogic
        implements BLogic<UserBean, ResultData> {
    /**
     * DAO クラス。
     * Spring によりインスタンス生成され設定される。
     */
    private UpdateDAO updateDAO = null;

    /**
     * DAO クラス。
     * Spring によりインスタンス生成され設定される。
     */
    private QueryDAO queryDAO = null;

    /**
     * UpdateDAO を返却する。
     *
     * @return 保持する UpdateDAO
     */
    public UpdateDAO getUpdateDAO() {
        return updateDAO;
    }

    /**
     * UpdateDAO を設定する。
     *
     * @param updateDAO UpdateDAO
     */
    public void setUpdateDAO(UpdateDAO updateDAO) {
        this.updateDAO = updateDAO;
    }

    /**
     * QueryDAO を返却する。
     *
     * @return 保持する QueryDAO
```

```
     */
    public QueryDAO getQueryDAO() {
        return queryDAO;
    }

    /**
     * QueryDAO を設定する。
     *
     * @param queryDAO QueryDAO
     */
    public void setQueryDAO(QueryDAO queryDAO) {
        this.queryDAO = queryDAO;
    }

    /**
     * 入力クラスの引数のデータを登録して検索した結果を戻す。<br>
     * １．パラメータのユーザ ID が存在するかチェックする。<br>
     * ２．パラメータのデータをデータベースに登録する。<br>
     * ３．パラメータのデータをデータベースに登録する。<br>
     * ４．結果値を戻す。
     *
     * @param inputData 入力クラス
     * @return 結果クラス
     */
    public ResultData execute(UserBean inputData) {

        //結果クラスの生成、結果の設定
            ResultData result = new ResultData();

        //入力情報ビジネスロジックの実行
        //返却値が、false の場合は重複エラー。
        //true の場合は登録処理が正常に終了。
        if (!check(inputData)) {

            //重複エラー時の処理
            return result;
            throw new ServiceException("AP.00001");
        }

        //ユーザ情報を登録する。
        updateDAO.execute("insertUser", inputData);

        // ユーザ情報を保持するための List 型を宣言。
        List<UserBean> userBeanList = null;

        //ユーザ情報を全件取得する。
        userBeanList = queryDAO.executeForObjectList("getUserList",
                        null);
        result.setUserBean(userBeanList );
        result.setTotalCount(Integer.valueOf(userBeanList.size()));

        return result;
    }

    /**
     * 入力された ID が重複か否かをデータベースから取得して判定する。
     * 重複している場合は、false を返却する。
     * 重複していない場合は、true を返却する。
     *
     * @param inputData 入力された値を保持する Bean
     * @return チェック結果
     */
    private boolean check(UserBean inputData) {

        //ID からデータを検索する。
        Integer count
            = queryDAO.executeForObject("getUserCount", inputData, Integer.class);
        if (count != null && count.intValue() > 0 ) {
            return false;
        }

        //データが取得できなかった場合は、重複していないとして
        //true を返却する。
```

```
        return true;
    }

}
```

- **“application-messages.properties”ファイルの変更**

例外制御で必要となるメッセージの追加を行う。
“terasoluna-spring-rich-blank/sources/application-messages.properties”ファイルを以下のように追加する。
（網掛け部分を追加する）。

```
################################################################################
##
## GlobalMessageResources で読み込むメッセージリソースファイル。
##
## このファイルには、業務共通のメッセージを記述する。
## 詳細は、GlobalMessageResources の JavaDoc を参照のこと。
##
################################################################################

#共通
error.common.00001=システムに例外が発生しました。管理者に問い合わせてください。

#Validation
error.multiField=[{0}]から計算した[{1}]が入力された[{1}]と一致しません。

#例外制御(AP 例外)で使うメッセージ
AP.00001=入力された ID は既に登録されています。
```

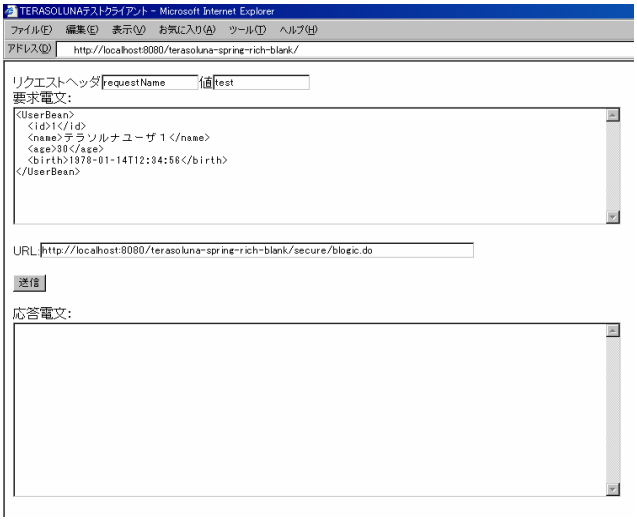
■ 確認

テストクライアントより、ビジネスロジックで例外が発生した場合の応答電文が正しく表示されることを確認する。

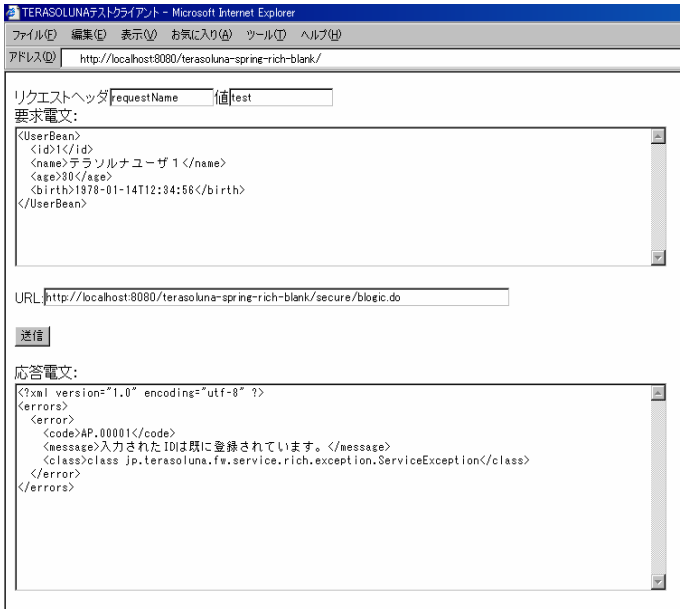
(1) 作成・更新したファイルを全て保存し、コンパイルを行ったのちに Tomcat を起動する。

(2) データベースを起動させるために“C:\hsqldb\terasoluna\startDB.bat”を実行する。

(3) “http://localhost:8080/terasoluna-spring-rich-blank”にアクセスし、テストクライアント画面を表示する。画面上部のリクエストヘッダに“requestname”、値に“dbAccess”と入力する。続いて、要求電文に以下の XML のように入力し、URL 欄に“http://localhost:8080/terasoluna-spring-rich-blank/secure/blogic.do”と入力し、「送信ボタン」をクリックする。



(4) 「入力された ID は既に登録されています。」のメッセージが出力され、AP 例外が発生した際のエラーメッセージが応答電文に表示されることを確認する。



■ 参考資料

- Rich 版機能説明書
  『RD-01 例外ハンドリング機能』

2.7　アクセス制御

本節では、特定のパスへのアクセス制限を行なう方法について学習する。

■ 概要

アクセス制御機能についての動きは図のようになる。許可されていないURLでアクセスしてきた場合にはエラーメッセージを含んだレスポンスをテストクライアント画面に表示する。

- リクエストデータ･･･ユーザ ID、ユーザ名、年齢、生年月日
- レスポンスデータ･･･総件数、ユーザ ID、ユーザ名、年齢、生年月日

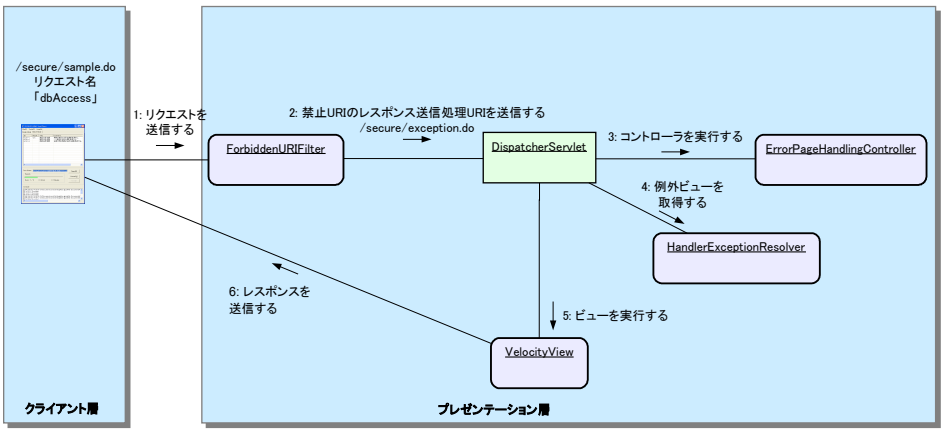


図12　アクセス制御画面遷移図

1. “http://localhost:8080/terasoluna-spring-rich-blank” にアクセスし、テストクライアント画面を表示する。リクエストデータを作成して、リクエスト名“dbAccess”で、“http://localhost:8080/terasoluna-spring-rich-blank/secure/sample.do”へ送信する。
2. サーブレットのフィルタが URL をみて許可されていないと判断し、“/secure/exception.do”を DispatcherServlet に送信する。
3. エラーページ用のコントローラを実行し、VelocityView の“exception.vm”テンプレートファイルを使ってクライアントに送信する。
4. テストクライアント画面にエラーメッセージを含んだ応答電文が正しく表示される。(画面下部の応答電文に表示される)。

アクセス制御機能については、ブランクプロジェクトの段階から実装された機能であり、デフォルトでは、“/secure/blogic.do”のみを許可している。ブランクプロジェクトがどのようにアクセス制御機能を実装しているかの手順を確認する。必要な手順は以下のようになる。

- “web.xml”ファイルの確認
- “applicationContext.xml”ファイルの確認

■ **手順**

● **“web.xml”ファイルの確認**

J2EE のサーブレットフィルタ機能を使い全てのアクセスに対してフィルタリングをかける。

“terasoluna-spring-rich-blank/webapps/WEB-INF/web.xml”ファイルが以下のようになっていることを確認する。(網掛け部分を確認する)。

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/w
eb-app_2_4.xsd"
    version="2.4">
                                          :
                                         略
                                          :
  <!-- フィルタの設定 -->

  <!-- 禁止 URL の設定 -->
  <filter>
    <filter-name>forbiddenURIFilter</filter-name>
    <filter-class>jp.terasoluna.fw.web.rich.ForbiddenURIFilter</filter-class>
  </filter>

  <!-- コンテキスト生成クラスの設定 -->
  <!-- TERASOLUNA Rich 版のハンドラマッピングを使用するときは必須となる。 -->
  <filter>
    <filter-name>requestContextHandlingFilter</filter-name>
    <filter-class>jp.terasoluna.fw.web.rich.RequestContextHandlingFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>forbiddenURIFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>requestContextHandlingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
                                          :
                                         略
                                          :
</web-app>
```

● **“applicationContext.xml”ファイルの確認**

許可する URI をここで設定している。ここで設定しなかった URI は全て禁止 URI とする。

“terasoluna-spring-rich-blank/webapps/WEB-INF/applicationContext.xml”ファイルが以下のようになっていることを確認する。(網掛け部分を確認する)。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/
spring-beans-2.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-2.0.xsd">


                                          :
                                         略
                                          :

  <!-- 禁止 URI チェッカー -->
  <bean id="forbiddenURIChecker"
              class="jp.terasoluna.fw.web.rich.ForbiddenURICheckerImpl">
    <property name="allowedURISet">
      <set>
        <value>/secure/blogic.do</value>
        <value>/</value>
        <value>/index.html</value>
      </set>
    </property>
  </bean>

                                          :
                                         略
                                          :

</beans>
```

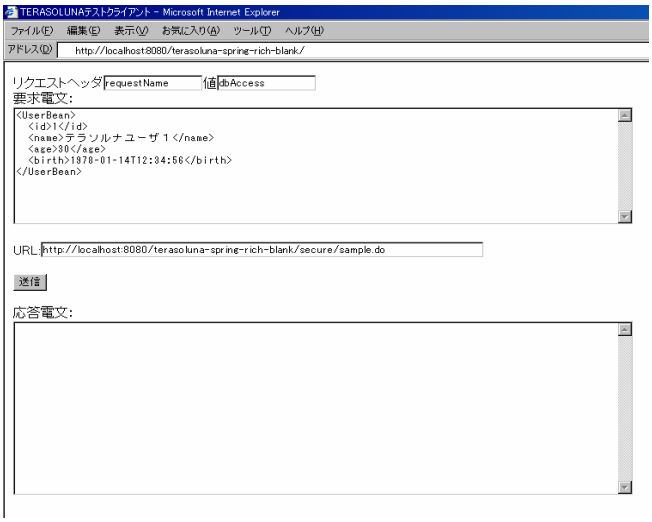
■ **確認**

テストクライアント画面より、禁止URLへアクセスした場合の応答電文が正しく表示されることを確認する。

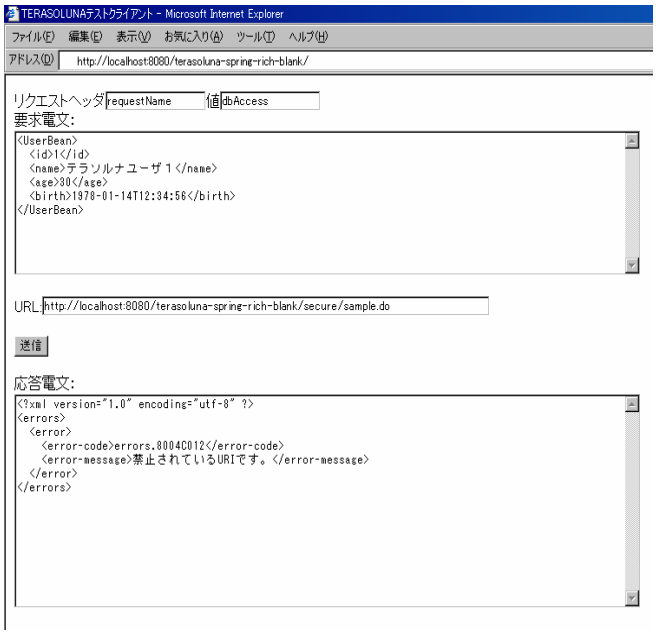
(1) 作成・更新したファイルを全て保存し、コンパイルを行ったのちに Tomcat を起動する。

(2) データベースを起動させるために“C:\hsqldb\terasoluna\startDB.bat”を実行する。

(3) “http://localhost:8080/terasoluna-spring-rich-blank”にアクセスし、テストクライアント画面を表示する。画面上部のリクエストヘッダに“requestname”、値に“dbAccess”と入力する。続いて、要求電文に以下の XML を入力し、URL 欄に“http://localhost:8080/terasoluna-spring-rich-blank/secure/sample.do”と入力し、「送信ボタン」をクリックする。



(4) 「禁止されている URI です。」のメッセージが出力され、禁止URLへアクセスした際のエラーメッセージが応答電文に表示されることを確認する。



■ 参考資料

- Rich 版機能説明書
  『RE-01 URI アクセス制御機能』
  『RD-01 例外ハンドリング機能』

第3章
Appendix

3.1　チュートリアル学習環境の整備(Oracle)

本節では、チュートリアルを学習するための環境整備（Oracle）を説明する。

■ 想定環境

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.23
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：WTP 1.5.5

■ インストール/開発環境の整備

(1) アプリケーションの用意

「2．2 チュートリアル学習環境の整備」以外で必要となるアプリケーションを以下に用意する。

- Oracle Database 10g for Windows（以下、Oracle）

(2) アプリケーションのインストール

Oracle 以外は「2．2 チュートリアル学習環境の整備」を参照のこと。

(3) プロジェクトの準備

(4) プロジェクトのインポート

(5) Tomcat へのプロジェクト追加

（3）～（5）の内容は、Rich 版「2．2 チュートリアル学習環境の整備」を参照のこと。

(6) データベースの設定

① Oracle を任意のディレクトリにインストールする。本チュートリアルでは、“C:\Oracle\product\10.1.0”というディレクトリにインストールしたと仮定する。

② SQL*Plus や Enterprise Manager などを利用し、ユーザの作成、ユーザへの表領域の割り当て、ユーザの権限設定などを行い、ユーザが Oracle データベースへのテーブル作成、データの挿入、選択、削除が出来るようにする。ここでは、データベースサーバの IP アドレスを“192.168.0.100”、ポートを“1521”、データベース名を“TERASOLUNA”、SID 名を“ORCL”、ユーザ名、パスワードを共に“tutorial4”とする。詳細は Oracle のマニュアルを参照のこと。

③ Web アプリケーションからデータベースへアクセスするために使用する Oracle の JDBC ドライバ“ojdbc14.jar”を Oracle 社の WEB サイト“http://otn.oracle.co.jp/software/tech/java/jdbc/index.html”から入手し、Tomcat インストールディレクトリ“C:\Program Files\Apache Software Foundation\apache-tomcat-5.5.23\common\lib”配下にコピーする。

(7) テーブル構築及びデータの作成

以下の SQL 文を実行する。

① USERLIST テーブルの作成

```
CREATE TABLE USERLIST(ID NUMBER(10) PRIMARY KEY,NAME VARCHAR2(50),AGE VARCHAR2(3),BIRTH V
ARCHAR2(10));
```

② USERLIST テーブルのデータ追加

```
INSERT INTO USERLIST VALUES(1,'Name1','26','1979/11/24');
INSERT INTO USERLIST VALUES(2,'Name2','23','1982/05/15');
INSERT INTO USERLIST VALUES(3,'Name3','28','1977/03/01');
INSERT INTO USERLIST VALUES(4,'Name4','21','1984/04/08');
INSERT INTO USERLIST VALUES(5,'Name5','19','1986/01/11');
INSERT INTO USERLIST VALUES(6,'Name6','22','1983/09/16');
INSERT INTO USERLIST VALUES(7,'Name7','18','1987/10/21');
INSERT INTO USERLIST VALUES(8,'Name8','25','1980/12/27');
INSERT INTO USERLIST VALUES(9,'Name9','26','1979/07/29');
INSERT INTO USERLIST VALUES(10,'Name10','24','1981/06/03');
INSERT INTO USERLIST VALUES(11,'Name11','19','1986/08/09');
INSERT INTO USERLIST VALUES(12,'Name12','28','1977/10/17');
INSERT INTO USERLIST VALUES(13,'Name13','29','1976/12/24');
INSERT INTO USERLIST VALUES(14,'Name14','21','1984/03/28');
INSERT INTO USERLIST VALUES(15,'Name15','27','1978/06/30');
INSERT INTO USERLIST VALUES(16,'Name16','23','1982/09/21');
INSERT INTO USERLIST VALUES(17,'Name17','26','1979/12/06');
INSERT INTO USERLIST VALUES(18,'Name18','25','1980/07/13');
INSERT INTO USERLIST VALUES(19,'Name19','21','1984/02/28');
INSERT INTO USERLIST VALUES(20,'Name20','20','1985/05/02');
INSERT INTO USERLIST VALUES(21,'Name21','27','1978/11/07');
INSERT INTO USERLIST VALUES(22,'Name22','18','1987/01/06');
INSERT INTO USERLIST VALUES(23,'Name23','19','1986/06/26');
INSERT INTO USERLIST VALUES(24,'Name24','20','1985/07/20');
INSERT INTO USERLIST VALUES(25,'Name25','26','1979/12/10');
INSERT INTO USERLIST VALUES(26,'Name26','24','1981/02/17');
INSERT INTO USERLIST VALUES(27,'Name27','23','1982/04/22');
INSERT INTO USERLIST VALUES(28,'Name28','28','1977/03/08');
INSERT INTO USERLIST VALUES(29,'Name29','29','1976/06/29');
INSERT INTO USERLIST VALUES(30,'Name30','18','1987/09/30');
INSERT INTO USERLIST VALUES(31,'Name31','27','1978/11/07');
INSERT INTO USERLIST VALUES(32,'Name32','18','1987/01/06');
INSERT INTO USERLIST VALUES(33,'Name33','19','1986/06/26');
INSERT INTO USERLIST VALUES(34,'Name34','20','1985/07/20');
INSERT INTO USERLIST VALUES(35,'Name35','26','1979/12/10');
INSERT INTO USERLIST VALUES(36,'Name36','24','1981/02/17');
INSERT INTO USERLIST VALUES(37,'Name37','23','1982/04/22');
INSERT INTO USERLIST VALUES(38,'Name38','28','1977/03/08');
INSERT INTO USERLIST VALUES(39,'Name39','29','1976/06/29');
INSERT INTO USERLIST VALUES(40,'Name40','18','1987/09/30');
COMMIT;
```

③ USERTABLE テーブルの作成

```
CREATE TABLE USERTABLE(ID NUMBER(5) PRIMARY KEY,NAME VARCHAR2(20),AGE VARCHAR2(3),BIRTH D
ATE);
```

(8) jdbc.properties の修正

インポートしたプロジェクトを Oracle で使えるように設定する。

“terasoluna-spring-thin-blank/webapps/WEB-INF/jdbc.properties”を以下のように作成する。

```
#ドライバー
jdbc.driverClassName=oracle.jdbc.driver.OracleDriver

#URL
jdbc.url=jdbc:oracle:thin:@192.168.0.100:1521:ORCL

#ユーザー名
jdbc.username=tutorial4

#パスワード
jdbc.password=tutorial4
```

3.2　チュートリアル学習環境の整備(PostgreSQL)

本節では、チュートリアルを学習するための環境整備（PostgreSQL）を説明する。

■ 想定環境

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.23
- データベース：PostgreSQL 8.2.x for Windows
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：WTP 1.5.5

■ インストール/開発環境の整備

(1) アプリケーションの用意

「2．2 チュートリアル学習環境の整備」以外で必要となるアプリケーションを以下に用意する。

- PostgreSQL 8.2.x for Windows（以下、PostgreSQL）

(2) アプリケーションのインストール

PostgreSQL 以外は「2．2 チュートリアル学習環境の整備」を参照のこと。

(3) プロジェクトの準備

(4) プロジェクトのインポート

(5) Tomcat へのプロジェクト追加

（3）～（5）の内容は、Rich 版「2．2 チュートリアル学習環境の整備」を参照のこと。

(6) データベースの設定

① PostgreSQL 8.2.x のインストールを行う。本チュートリアルでは以下の設定のようにインストールしたと仮定する。
ディレクトリ : C:\Program Files\PostgreSQL\8.2（デフォルト）

① Windows のスタートメニューから[プログラム]→[PostgreSQL 8.2]→[pgAdmin III]を起動する。
「PostgreSQL Database Server 8.2(localhost:5432)」を選択し、右クリックメニューから「接続」を選択する。
パスワード入力ウィンドウが表示されたら、「postgres」と入力する。

② 画面左のメニューから「データベース(1)」を選択し、右クリックメニューより「新しいデータベース」を選択し、以下の内容を入力し「OK」を押下する。
名前 : terasoluna

③ 画面左のメニューから「データベース(2)」「terasoluna」を選択すると、×印が消えてデータベースの内容が表示されることを確認する。

④ Web アプリケーションからデータベースへアクセスするために使用する PostgreSQL の JDBC ドライバが“C:\Program Files\PostgreSQL\8.2\jdbc”フォルダにあるので、その中の“postgresql-8.2-405.jdbc3.jar”を、Tomcat インストールディレクトリ“C:\Program Files\Apache Software Foundation\apache-tomcat-5.5.23\common\lib”配下にコピーする。

(7) テーブル構築及びデータの作成

[pgAdmin III]より、画面左のメニューから「データベース(2)」「terasoluna」を選択した状態で、「ツール」

→「クエリーツール」を選択する。上の入力エリアに以下の SQL 文を入力し実行する。

① USERLIST テーブルの作成

```
CREATE TABLE USERLIST(ID BIGINT PRIMARY KEY,NAME VARCHAR(50),AGE VARCHAR(3),BIRTH VARCHAR
(10));
```

② USERLIST テーブルのデータ追加

```
INSERT INTO USERLIST VALUES(1,'Name1','26','1979/11/24');
INSERT INTO USERLIST VALUES(2,'Name2','23','1982/05/15');
INSERT INTO USERLIST VALUES(3,'Name3','28','1977/03/01');
INSERT INTO USERLIST VALUES(4,'Name4','21','1984/04/08');
INSERT INTO USERLIST VALUES(5,'Name5','19','1986/01/11');
INSERT INTO USERLIST VALUES(6,'Name6','22','1983/09/16');
INSERT INTO USERLIST VALUES(7,'Name7','18','1987/10/21');
INSERT INTO USERLIST VALUES(8,'Name8','25','1980/12/27');
INSERT INTO USERLIST VALUES(9,'Name9','26','1979/07/29');
INSERT INTO USERLIST VALUES(10,'Name10','24','1981/06/03');
INSERT INTO USERLIST VALUES(11,'Name11','19','1986/08/09');
INSERT INTO USERLIST VALUES(12,'Name12','28','1977/10/17');
INSERT INTO USERLIST VALUES(13,'Name13','29','1976/12/24');
INSERT INTO USERLIST VALUES(14,'Name14','21','1984/03/28');
INSERT INTO USERLIST VALUES(15,'Name15','27','1978/06/30');
INSERT INTO USERLIST VALUES(16,'Name16','23','1982/09/21');
INSERT INTO USERLIST VALUES(17,'Name17','26','1979/12/06');
INSERT INTO USERLIST VALUES(18,'Name18','25','1980/07/13');
INSERT INTO USERLIST VALUES(19,'Name19','21','1984/02/28');
INSERT INTO USERLIST VALUES(20,'Name20','20','1985/05/02');
INSERT INTO USERLIST VALUES(21,'Name21','27','1978/11/07');
INSERT INTO USERLIST VALUES(22,'Name22','18','1987/01/06');
INSERT INTO USERLIST VALUES(23,'Name23','19','1986/06/26');
INSERT INTO USERLIST VALUES(24,'Name24','20','1985/07/20');
INSERT INTO USERLIST VALUES(25,'Name25','26','1979/12/10');
INSERT INTO USERLIST VALUES(26,'Name26','24','1981/02/17');
INSERT INTO USERLIST VALUES(27,'Name27','23','1982/04/22');
INSERT INTO USERLIST VALUES(28,'Name28','28','1977/03/08');
INSERT INTO USERLIST VALUES(29,'Name29','29','1976/06/29');
INSERT INTO USERLIST VALUES(30,'Name30','18','1987/09/30');
INSERT INTO USERLIST VALUES(31,'Name31','27','1978/11/07');
INSERT INTO USERLIST VALUES(32,'Name32','18','1987/01/06');
INSERT INTO USERLIST VALUES(33,'Name33','19','1986/06/26');
INSERT INTO USERLIST VALUES(34,'Name34','20','1985/07/20');
INSERT INTO USERLIST VALUES(35,'Name35','26','1979/12/10');
INSERT INTO USERLIST VALUES(36,'Name36','24','1981/02/17');
INSERT INTO USERLIST VALUES(37,'Name37','23','1982/04/22');
INSERT INTO USERLIST VALUES(38,'Name38','28','1977/03/08');
INSERT INTO USERLIST VALUES(39,'Name39','29','1976/06/29');
INSERT INTO USERLIST VALUES(40,'Name40','18','1987/09/30');
```

③ USERTABLE テーブルの作成

```
CREATE TABLE USERTABLE(ID BIGINT PRIMARY KEY,NAME VARCHAR(20),AGE VARCHAR(3),BIRTH TIMEST
AMP);
```

(8) jdbc.properties の修正

インポートしたプロジェクトを PostgreSQL で使えるように設定する。

“terasoluna-spring-rich-blank/webapps/WEB-INF/jdbc.properties”を以下のように作成する。

```
#ドライバー
jdbc.driverClassName=org.postgresql.Driver

#URL
jdbc.url=jdbc:postgresql://127.0.0.1:5432/terasoluna

#ユーザー名
jdbc.username=postgres

#パスワード
```

```
jdbc.password=postgres
```

■ PostgreSQL でのストアドプロシージャ使用方法

PostgreSQL でストアドプロシージャを使用する手順を以下に説明する。

Rich 版で提供している DAO の中にストアドプロシージャ用として StoredProcedureDAO を用意している。通常はこの DAO を使う必要があるが、PostgreSQL ではストアドプロシージャは通常の SQL と同様の扱いとなるため、QueryDAO を使用してストアドプロシージャを実行する必要がある。

● PostgreSQL のストアドプロシージャ実装例

① Bean 定義に QueryDAO を設定(DI)する。

```
<bean id="resultReserveForPostgreSQLService" scope="prototype"
  class="jp.hotelsample.register.service.ResultReserveServicePostgreSQLImpl">
  <property name="queryDao" ref="queryDAO" />
</bean>
```

② QueryDAO を使用して、ストアドプロシージャを実行する。

```
    Private QueryDAO queryDAO = null;
        ......Setter は省略

    protected String executeProcedure(InsertReserveParam insertReserveParam) {
        String reserveNumber
            = queryDao.executeForObject(
                  "register.insertReserveProcedurePostgreSQL",
                  insertReserveParam,
                  String.class);
                                          :
                                        略
                                          :
    }
```

③ PostgreSQL 用のビジネスロジックを設定(DI)した Action を実行する。

```
<bean name="/register/resultReserve" scope="prototype"
  class="jp.hotelsample.register.web.action.ResultReserveAction">
  <property name="resultReserveService" ref="resultReserveForPostgreSQLService" />
</bean>
```

■ **参考資料**

● Rich 版機能説明書
  『CB-01 データベースアクセス機能』

3.3　チュートリアル学習環境の整備(Tomcat-JNDI)

本節では、チュートリアルを学習するための環境整備を Tomcat の JNDI データソースを用いた場合について説明する。

■ **想定環境**

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.23
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：WTP 1.5.5

■ インストール/開発環境の整備

(1) アプリケーションの用意
(2) アプリケーションのインストール
(3) プロジェクトの準備
(4) プロジェクトのインポート
(5) Tomcat へのプロジェクト追加
(6) データベースの設定
(7) テーブル構築及びデータの作成

（1）～（7）の内容は、「3．1 チュートリアル学習環境の整備（Oracle）」を参照のこと。

(8) コンテキストの作成

① コマンドプロンプトより以下のコマンドを入力し Tomcat を起動する（Eclipse 上の WTP の Tomcat は停止しておくこと）。

```
set CATALINA_HOME=C:\Program Files\Apache Software Foundation\apache-tomcat-5.5.23
set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_0x (xはバージョン番号)
"%CATALINA_HOME%\bin\startup.bat"
```

② “http://localhost:8080/admin/”にアクセスし、ユーザ名“tomcat”パスワード“tomcat”を入力して Tomcat 管理ツールを起動する。
③ 左メニューの“Tomcat Server\サービス(Catalina)\ホスト(localhost)”を押下し、右に表示された画面から「新しいコンテキストの作成」を選択する。
④ ドキュメントベースに Rich 版は“C:\Eclipse\workspace\terasoluna-spring-rich-blank\webapps”、パスに“/terasoluna-spring-rich-blank”を入力する。コンテキストプロパティの再ロード可能、ネームサービスの利用を「True」にする。
⑤ 「保存」ボタンをクリックし、保存成功と表示されたら、「変更を反映」ボタンをクリックし、一度「ログアウト」する。

(9) データソースの作成

① 再度ログインすると、“Tomcat Server\サービス(Catalina)\ホスト(localhost)”ツリーの中に“コンテキスト(/terasoluna-spring-thin-blank)”または“コンテキスト(/terasoluna-spring-rich-blank)”ができているのでツリーを開く。＞リソース＞データソースを押下する。
② 右側の画面から「新しいデータソースの作成」を選択する。JNDI 名に「TerasolunaDataSource」、データ

ソース URL に“jdbc:oracle:thin:@192.168.0.100:1521:ORCL”、JDBCドライバクラスに“oracle.jdbc.driver.OracleDriver”、ユーザ名に“tutorial4”、パスワードに“tutorial4”を入力する。
※データソース URL、ユーザ名、パスワードは自分の環境を設定すること。
【JNDI 名】
TerasolunaDataSource
【データソース URL】
jdbc:oracle:thin:@192.168.0.100:1521:ORCL
【JDBCドライバクラス】
oracle.jdbc.driver.OracleDriver
【ユーザ名】
tutorial4
【パスワード】
tutorial4

③ 「保存」ボタンを押下し、「変更を反映」「ログアウト」を実行する。
④ コマンドプロンプトより以下のコマンドを入力し Tomcat を停止する。

```
set CATALINA_HOME=C:\Program Files\Apache Software Foundation\apache-tomcat-5.5.23
set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_0x (xはバージョン番号)
"%CATALINA_HOME%\bin\shutdown.bat"
```

(10)データソースの設定を WTP へ反映させる

WTP の Tomcat は設定情報を Eclipse 内部でコピーして管理している。よって、管理ツールで設定したデータソースの設定を WTP へ反映させる必要がある。反映させる方法には以下の2種類がある。

1. WTP で管理している Tomcat の“server.xml”ファイルを手動で変更する方法
2. プロジェクトの META-INF 配下に“context.xml”ファイルを配置する方法

どちらを適用するか手順を参考にして各担当で判断してもらいたい。
META-INF 配下に“context.xml”ファイルが配置してある場合は、“server.xml”ファイルに設定をしても“context.xml”ファイルの設定を優先するので注意が必要である。
商用環境の設定は(9)データソースの設定までででよい。

【server.xml を手動で変更する方法】

① エクスプローラにて、“C:\Program Files\Apache Software Foundation\apache-tomcat-5.5.23\conf\Catalina\localhost”フォルダ配下にある、コンテキスト設定ファイル(Rich 版なら“terasoluna-spring-rich-blank.xml”)をテキストエディタで開き、以下の網掛けの部分をコピーする。

```
<?xml version="1.0" encoding="UTF-8"?>
<Context
    docBase="C:/Eclipse/workspace/terasoluna-spring-rich-blank/webapps"
    reloadable="true">
  <Resource
    name="TerasolunaDataSource"
    type="javax.sql.DataSource"
    password="tutorial4"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    maxIdle="2"
    maxWait="5000"
    username="tutorial4"
    url="jdbc:oracle:thin:@192.168.0.100:1521:ORCL"
    maxActive="4"/>
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>
```

② Eclipse 上で左メニューのパッケージエクスプローラ＞サーバ＞Tomcat v5.5 サーバー @ localhost-config＞server.xml を選択する。右画面に server.xml の内容が表示されたら、先程コピーした内容を Context タグ部分へ貼り付ける。（以下の網掛け部分のようにする。）

```
<?xml version="1.0" encoding="UTF-8"?>
<Server>
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"/>
  <Listener className="org.apache.catalina.storeconfig.StoreConfigLifecycleListener"/>
  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener"/>
  <GlobalNamingResources>
    <Environment name="simpleValue" type="java.lang.Integer" value="30"/>
    <Resource auth="Container" description="User database that can be updated and saved"
factory="org.apache.catalina.users.MemoryUserDatabaseFactory" name="UserDatabase" pathna
me="conf/tomcat-users.xml" type="org.apache.catalina.UserDatabase"/>

  </GlobalNamingResources>
  <Service name="Catalina">
    <Connector connectionTimeout="20000" maxHttpHeaderSize="8192" maxSpareThreads="75" m
axThreads="150" minSpareThreads="25" port="8080" redirectPort="8443">
    </Connector>
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443">
    </Connector>
    <Engine defaultHost="localhost" name="Catalina">
      <Realm className="org.apache.catalina.realm.UserDatabaseRealm"/>
      <Host appBase="webapps" name="localhost">

      <Context docBase="terasoluna-spring-rich-blank" path="/terasoluna-spring-rich-blan
k" reloadable="true" source="org.eclipse.jst.j2ee.server:terasoluna-spring-rich-blank"/>
        <Resource
          name="TerasolunaDataSource"
          type="javax.sql.DataSource"
          password="tutorial4"
          driverClassName="oracle.jdbc.driver.OracleDriver"
          maxIdle="2"
          maxWait="5000"
          username="tutorial4"
          url="jdbc:oracle:thin:@192.168.0.100:1521:ORCL"
          maxActive="4"/>
      </Context>
      </Host>
    </Engine>
  </Service>
</Server>
```

※JNDI データソースを設定したプロジェクトを WTP の Tomcat プロジェクトから除去した場合、同じプロジェクトを Tomcat へ追加した場合に再度この手順が必要となる。

【META-INF 配下に“context.xml”ファイルを配置する方法】

① エクスプローラにて、“C:\Program Files\Apache Software Foundation\apache-tomcat-5.5.23\conf\Catalina\localhost”フォルダ配下にある、コンテキスト設定ファイル（Rich 版なら“terasoluna-spring-rich-blank.xml”）をテキストエディタで開き、以下の網掛けの部分をコピーする。

```
<?xml version="1.0" encoding="UTF-8"?>
<Context
    docBase="C:/Eclipse/workspace/terasoluna-spring-rich-blank/webapps"
    reloadable="true">
  <Resource
    name="TerasolunaDataSource"
    type="javax.sql.DataSource"
    password="tutorial4"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    maxIdle="2"
    maxWait="5000"
```

```
    username="tutorial4"
    url="jdbc:oracle:thin:@192.168.0.100:1521:ORCL"
    maxActive="4"/>
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>
```

② Eclipse にて“プロジェクト名/webapps/META-INF/context.xml”ファイルに、先程コピーした内容を貼り付けて修正する。(以下の網掛け部分のようにする。)

```
<?xml version="1.0" encoding="UTF-8"?>

<Context>
  <Resource
    name="TerasolunaDataSource"
    type="javax.sql.DataSource"
    password="tutorial4"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    maxIdle="2"
    maxWait="5000"
    username="tutorial4"
    url="jdbc:oracle:thin:@192.168.0.100:1521:ORCL"
    maxActive="4"/>
</Context>
```

※Tomcat 以外の AP サーバへ deploy する際には ant の build.xml に“context.xml”ファイルを削除する記述を入れておくこと。
※WTP 環境以外の Tomcat へ deploy する場合は、JNDI の設定の注意点があるので、「3．7　WTP プロジェクトを非 WTP 環境へ移行する手順」の JNDI 設定の注意点を参照のこと。

■ データベース接続設定

データソース用の Bean 定義を JNDI に対応させる必要がある。詳細については、Rich 版機能説明書『CC-01 JNDI アクセス機能』参照のこと。

Rich 版では、「2．4 データベースアクセス」でデータベースに接続する設定を行っているが、“dataAccessContext-local.xml”ファイルの dataSource の Bean 定義については、以下のようにすること。

```
  <bean id="dataSource"
      class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:comp/env/TerasolunaDataSource" />
  </bean>
```

3.4　チュートリアル学習環境の整備(WebLogic)

本節では、チュートリアルを学習するための環境整備を WebLogic を用いた場合について説明する。

■ 想定環境

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：BEA WebLogic Server 9.2 for Windows
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：WTP 1.5.5

■ インストール/開発環境の整備

(1) アプリケーションの用意

「2．2 チュートリアル学習環境の整備」以外で必要となるアプリケーションを以下に用意する。

- BEA WebLogic Server 9.2 for Windows（以下、WebLogic）

(2) アプリケーションのインストール

WebLogic 以外は「2．2 チュートリアル学習環境の整備」を参照のこと。手順内にある Tomcat のサーバ定義も必ず行うこと。

(3) WebLogic のインストール

WebLogic のインストール方法について説明する。

① インストール実行ファイル「server920_ja_win32.exe」を実行するとようこそ画面がでてくるので、「次へ」を押下する。
② 次にライセンス契約の画面となり、「はい」にチェックをして、「次へ」を押下する。
③ ライセンスに同意すると次に Weblogic のホームディレクトリの選択となるので、新しい BEA ホームを作成するを選択し、ディレクトリを入力し、「次へ」を押下する。デフォルト(C:\bea)の場合はそのままでよい。
④ 次にインストールタイプの選択となるので完全インストール(デフォルト)を選択し、「次へ」を押下する。
⑤ 次にオプションツールの選択となるのでチェックを入れて(デフォルト)、「次へ」を押下する。
⑥ 次にライセンス契約の画面となり、「はい」にチェックをして、「次へ」を押下する。
⑦ 次に Weblogic の本体をインストールするディレクトリを選択する。基本的にはデフォルトのままで「次へ」を押下する。
⑧ 次にショートカットの場所の選択の画面となり、デフォルトのままで「次へ」を押下後、インストールが開始される。
⑨ インストールが完了すると、「～のインストールが正常に完了しました。」とメッセージが出力されたウィンドウが表示される。「Quickstart を実行」というチェックボックスがあるが、チェックがついていないことを確認し、「完了」を押下する。

(4) プロジェクトの準備

(5) プロジェクトのインポート

(4)～(5)の内容は、Rich 版「2．2 チュートリアル学習環境の整備」を参照のこと。

(6) データベースの設定

(7) テーブル構築及びデータの作成

(6)～(7)の内容は、「3．1 チュートリアル学習環境の整備(Oracle)」を参照のこと。

(8) ドメインの生成

① Windows のスタートメニューから[プログラム]→[BEA Products]→[Tools]→[Configuration Wizard]を実行する。実行後に出力されたウィンドウで、「新しい WebLogic ドメインの作成」をチェックし、「次へ」を押下する。
② ドメインソースの画面では、デフォルトのまま「次へ」を押下する。
③ 次に作成するドメインの管理者ユーザ名とパスワードの入力画面となるので以下のように入力し「次へ」を押下する。
User Name : weblogic
User Password : weblogic
Confirm User Password : weblogic(パスワードの再入力確認)
Description : (デフォルトのままでよい)
④ 次にサーバの起動モード及び JDK のコンフィグレーション画面では、デフォルトのままで「次へ」を押下する。(「WebLogic ドメインの起動モード」は「開発モード」を選択し、「JDK の選択」は「BEA 提供の JDK」の「Sun SDK 1.5.0_0x (x はバージョン番号)」を選択する。)
⑤ カスタマイズの選択画面では、「いいえ」を選択する。
⑥ ドメインの名前と場所の入力画面では、以下のように入力し「作成」を押下する。
ドメイン名 : tutorial1
ドメインの場所 : C:\bea\user_projects\domains(デフォルト)
⑦ 「ドメインの作成が完了しました。」と画面に出力されたら、「管理サーバの起動」にチェックがついていないことを確認し、「完了」を押下して、ドメインの作成ウィザードを終了する。

(9) データソースの生成

WebLogic のインストールおよびドメインの作成が完了したら、OracleDB へ接続するためのデータソースの作成を以下の手順にそって行う。

① コマンドプロンプトを開き、前節にて作成したドメインのフォルダ\bin へ移動する。
移動先の例:「C:\bea\user_projects\domains\tutorial1\bin」
② コマンドプロンプトから「setDomainEnv.cmd」を実行する。続けて「startWebLogic.cmd」を実行すると、WebLogic サーバが起動する。エラーが発生せずに、以下のようなメッセージが出力されれば正常に起動している。

```
<2007/06/20 16時16分04秒 JST> <Notice> <WebLogicServer> <BEA-000360> <サーバが R
UNNING モードで起動しました。>
```

③ サーバー起動後、InternetExplorer で「http://localhost:7001/console/」を開き、ユーザ名に「weblogic」、パスワードに「weblogic」を入力し、ログインボタンを押下して、ログインする。
④ ログイン後、画面左のメニューから「tutorial1」「サービス」「JDBC」の順に階層を開き「データソース」を選択する。選択後、画面右に JDBC データソースの画面が出力されることを確認し、左上の「ロックして編集」を押下し、「新規作成」を押下する。
⑤ JDBC データソースのプロパティの入力画面では以下のように入力し、「次へ」を押下する。
名前 : TerasolunaDataSource
JNDI 名 : TerasolunaDataSource
データベースの種類 : Oracle
データベースドライバ : *Oracle's Driver (Thin) Versions:9.0.1,9.2.0,10
⑥ トランザクションオプションの入力画面ではデフォルトのまま、「次へ」を押下する。
⑦ 接続プロパティの入力画面では以下の記述に従って入力し、「次へ」を押下する。
データベース名 :(作成したデータベース名) 例 ORCL

ホスト名：（DB のある PC のホスト名、もしくは IP アドレス）例 192.168.0.100
ポート：（Oracle のポート）例 1521
データベースユーザ名：（データベースにログインするユーザ名）例 tutorial4
パスワード：（データベースにログインするパスワード）例 tutorial4

⑧ データベース接続のテスト画面となるので、入力内容を確認し、問題なければ「コンフィグレーションのテスト」ボタンを押下する。
⑨ 設定が正常ならばテストが成功し、成功した場合は「次へ」を押下する。失敗した場合は⑦にて入力した内容を再度確認する。
⑩ 対象の選択画面となるので、AdminServer にチェックを入れて「完了」を押下する。
⑪ 左上の「変更のアクティブ化」を押下する。

(10)プロジェクトを WebLogic 対応にする

① 「/ant/build.properties」を以下のように修正する。なお、以下はデフォルトのフォルダへインストールすることを想定しているため適宜自分の環境に合わせて修正すること。
webapsvr.home=C:/bea/weblogic92
webapsvr.lib.dir=C:/bea/weblogic92/server/lib
deploy.dir=C:/
jdbc.driver=C:/bea/weblogic92/server/lib/ojdbc14.jar

② 「/ant/build.xml」を以下のように修正する。
${webapsvr.lib.dir}/servlet-api.jar; → ${webapsvr.lib.dir}/weblogic.jar
${webapsvr.lib.dir}/jsp-api.jar; → 削除

③ データソース用の Bean 定義を JNDI 用に修正する。詳細については、Rich **版機能説明書**『CC-01 JNDI アクセス機能』参照のこと。

Rich 版では、「2．4 データベースアクセス」でデータベースに接続する設定を行っているが、“dataAccessContext-local.xml”ファイルの dataSource の Bean 定義については、以下のようにすること。

```
  <bean id="dataSource"
      class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="TerasolunaDataSource" />
  </bean>

  <!--
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="
close">
            <property name="driverClassName" value="${jdbc.driverClassName}"/>
            <property name="url" value="${jdbc.url}"/>
            <property name="username" value="${jdbc.username}"/>
            <property name="password" value="${jdbc.password}"/>
  </bean>
  -->
```

④ weblogic.xml を以下のように作成し、/webapps/WEB-INF/配下に配置すること。

Rich 版

```
<?xml version="1.0" encoding="UTF-8"?>

<weblogic-web-app>
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
    <default-mime-type>text/xml;charset=UTF-8</default-mime-type>
  </container-descriptor>
</weblogic-web-app>
```

⑤ web.xml ファイルの mime-mapping の設定を削除する。（Rich 版のみ）
以下の網掛け部分のように修正する。

```
                                        :
                                       略
                                        :
    <mime-mapping>
        <extension>xml</extension>
        <mime-type>text/xml;charset=UTF-8</mime-type>
    </mime-mapping>
                                        :
                                       略
                                        :
```

(11)アプリケーションの起動

① Eclipse 上で「/ant/build.xml」を右クリックし、「Ant の実行」より「deploy」を実行する。C ドライブ直下に <context.name>.war ファイルができていることを確認する。
② InternetExplorer で「http://localhost:7001/console/」を開き、ユーザ名に「weblogic」、パスワードに「weblogic」を入力し、ログインボタンを押下して、ログインする。
③ ログイン後、画面左のメニューから「tutorial1」「デプロイメント」を選択する。選択後、画面右にデプロイメントの画面が出力されることを確認し、左上の「ロックして編集」を押下し、「インストール」を押下する。
④ 場所から C ドライブを選択し、<context.name>.war ファイルにチェックを入れて「次へ」を押下する。
⑤ 対象指定スタイルの選択画面では、デフォルトのまま「次へ」を押下する。
⑥ 省略可能な設定画面では、「ソースのアクセス可能性」で「デプロイメントを次の場所からアクセス可能にする」にチェックを入れて「次へ」を押下する。
⑦ 選択項目を確認して「完了」を押下する。
⑧ 左上の「変更のアクティブ化」を押下する。
⑨ 再度、画面左のメニューから「tutorial1」「デプロイメント」を選択する。選択後、画面右にデプロイメントの画面が出力されることを確認し、<context.name>にチェックを入れて「起動」「すべての要求を処理」を押下する。
⑩ デプロイメントの起動画面で「はい」を押下する。
⑪ 「選択したデプロイメントに開始要求が送信されました。」が表示されればデプロイ終了。
⑫ Rich 版は、テストクライアント画面より http://localhost:7001/<context.name>/secure/blogic.do に電文を送信し確認する。

■ 参考資料

今回説明した以外にも WebLogic を使う上で注意点がいくつかあるので、以下の資料を参照すること。

- Rich 版機能説明書
  『CA-01 トランザクション管理機能』
  『CC-01 JNDI アクセス機能』

3.5　チュートリアル学習環境の整備(WebSphere)

本節では、チュートリアルを学習するための環境整備を WebSphere を用いた場合について説明する。

■ 想定環境

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional SP2 以上
- JDK：Java SE5.0
- Web アプリケーションサーバ：IBM Application Server Network Development V6.1.0.0
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：IBM WebSphere Application Server Toolkit V6.1

■ インストール/開発環境の整備

(1) アプリケーションの用意

「2．2 チュートリアル学習環境の整備」以外で必要となるアプリケーションを以下に用意する。

- IBM Application Server Network Development V6.1J（以下、WebSphere）
- IBM Websphere Application Server Toolkit V6.1（以下、AST）

(2) アプリケーションのインストール

WebSphere 以外は「2．2 チュートリアル学習環境の整備」を参照のこと。手順内にある Tomcat のサーバ定義も必ず行うこと。

(3) WebSphere のインストール

WebSphere のインストール方法について説明する。

① インストール実行ファイル「launchpad.exe」を実行するとようこそ画面がでてくるので、左メニューから「Application Server Network Development のインストール」を押下し、「WebSphere Application Server Network Deployment のインストール・ウィザードの起動」を押下する。
② インストール・ウィザードへようこそ画面となり、「次へ」を押下する。
③ ライセンス契約の画面となり、同意にチェックをして、「次へ」を押下する。
④ システムの前提条件の検査画面となり、「次へ」を押下する。
⑤ サンプル・アプリケーションのインストール画面となり、「サンプルアプリケーションをインストールする」チェックをせずに「次へ」を押下する。
⑥ インストール・ディレクトリー画面となり、製品のインストール・ロケーションはデフォルトで「次へ」を押下する。
⑦ WebSphere Application Server 環境画面となり、環境から「アプリケーション・サーバー」を選択し「次へ」を押下する。
⑧ 管理セキュリティーを有効にする画面となり、「管理セキュリティーを有効にする」をチェックし、ユーザ名は「terasoluna」、パスワードは「password」を入力し「次へ」を押下する。
注：ユーザ名とパスワードは任意でよい。
⑨ インストールの要約画面となり、「次へ」を押下する。
⑩ インストールが完了すると、インストール結果画面となり、「完了」を押下するとファースト・ステップが起動される。「インストール検査」を押下し、AppSrv01 が正常にインストールされていることを確認し、ファースト・ステップの終了を押下する。

(4) AST のインストール

AST のインストール方法について説明する。

① インストール実行ファイル「launchpad.exe」を実行するとようこそ画面がでてくるので、左メニューから「Application Server Toolkit のインストール」を押下し、「Application Server Toolkit のインストール・ウィザードの起動」を押下する。
② インストール・ウィザードへようこそ画面となり、「次へ」を押下する。
③ 次にライセンス契約の画面となり、同意にチェックをして、「次へ」を押下する。
④ インストール先画面となり、インストール・ロケーションはデフォルトで「次へ」を押下する。
⑤ インストールの要約画面となり、「次へ」を押下する。
⑥ インストールが終了すると「完了」を押下する。

(5) プロジェクトの準備

AST の設定方法について説明する。

① ウィンドウズのスタートメニューから[プログラム]→[IBM WebSphere]→[Application Server Toolkit V6.1]→[Application Server Toolkit]を実行し AST を起動するとデフォルトとして「J2EE」パースペクティブが表示される。
② メニューの「ウィンドウ」→「設定」を選択し、「Java」→「コンパイラ」→「コンパイラ準拠レベル」を「5.0」に選択する。
③ 「サーバー」→「WebSphere」で WebSphere プロファイルが「AppSrv01」になっていることを確認する。
④ 「サーバー」→「インストール済みランタイム」に WebSphere Application Server v6.1 がない場合、「新規」を押下し、「IBM」→「WebSphere Application Server v6.1」を選択し、「次へ」を押下する。名前は「WebSphere Application Server v6.1」、インストール・ディレクトリは WebSphere がインストールがインストールされたディレクトリを入力する。（C:\Program Files\IBM\WebSphere\AppServer）ランタイム環境に追加されたら「WebSphere Application Server v6.1」をチェックし「OK」を押下する。
⑤ 画面下段のビューから「サーバー」を表示し、「WebSphere Server v6.1@localhost」が選択されていることを確認する。サーバーがない場合はマウス右クリック→「新規」→「サーバー」で WebSphere Application Server v6.1 を追加する。

(6) プロジェクトのインポート

AST の新規プロジェクト作成とインポート方法について説明する。

① 「プロジェクト・エクスプローラー」から「動的 Web プロジェクト」を選択し、マウス右クリック→「新規」→「動的 Web プロジェクト」を選択する。
② 新規動的 Web プロジェクトウィザードが表示される。「プロジェクト名」は任意で入力し、ターゲット・ランタイムが「WebSphere Application Server v6.1」になっていることを確認し、「次へ」を押下する。
③ プロジェクト・ファセット画面となり、デフォルトのままにしておき、「次へ」を押下する。
④ Web モジュール画面となりコンテンツディレクトリは「webapps」、Java ソースディレクトリは「sources」に修正した後、「完了」を押下する。
注：インポートの際 web.xml の上書きを聞かれれるが、「はい」を押下する。（上書き）
⑤ 「プロジェクト・エクスプローラー」から「動的 Web プロジェクト」を展開し、プロジェクトが正常に作成されていることを確認する。
⑥ zip ファイルのプロジェクトを予め解答しておき、作成されたプロジェクトからマウス右クリック→「インポート」→「ファイルシステム」選択→「次へ」→「ソース・ディレクトリ」に zip を解答したディレクトリを選択し、「.settings」と「.classpath」、「.project」以外を全選択し「完了」を押下する。
⑦ プロジェクトからマウス右クリック→「プロパティー」→「Java ビルド・パス」→「デフォルト出力フォルダー」のパスを「<context.name/webapps/WEB-INF/classes>」に変更し、「OK」を押下する。

（5）～（6）の内容は、Rich 版「2．2 チュートリアル学習環境の整備」を参照のこと。

(7) データベースの設定

(8) テーブル構築及びデータの作成

（7）～（8）の内容は、「3．1 チュートリアル学習環境の整備（Oracle）」を参照のこと。

(9) サーバの設定

① ASTのサーバービューのWebSphere v6.1 Server@localをダブルクリックし、サーバーの概要画面を表示する。
② サーバーの概要画面の「サーバー」のWebSphereプロファイル名がAppSrv01に選択されていることを確認する。選択されていない場合は選択する。
③ 「セキュリティ」の「このサーバー上でセキュリティを有効にする」をチックし、ユーザIDとパスワードはWebSphereインストール時に入力した値を入力し（terasoluna/password）保存する。
④ ASTのサーバービューからWebSphere v6.1 Server@localのマウス右クリック→「始動」でWebSphereを起動する。
⑤ **[プログラム]→[IBM WebSphere]→[Application Server Network Development]→[プロファイル]→[AppSrv01]→[管理コンソール]を実行し、WebSphereインストール時に入力した値（terasoluna/password）でログインする。**
⑥ **メニューの「環境」→「ネーミング」→「CORBA ネーミングサービスグループ」を選択し、表示されたリストから「EVERYONE」をクリックする。ロールを「COS ネーミングの削除」を選択し、「OK」を押下する。設定を変更した後は必ず「保管」をクリックし、変更を保存する。**
注：この設定はサーバの再起動が必要である。

(10)データソースの生成

OracleDBへ接続するためのデータソースの作成を以下の手順にそって行う。

① Oracle JDBCドライバを入手し、WebSphereインストールディレクトリ（C:\Program Files\IBM\WebSphere\AppServer）のlib\extにコピーする。
② 管理コンソールの「セキュリティー」→「管理、アプリケーション、およびインフラストラクチャーの保護」をクリックし、画面右側の「認証」→「Java 認証・承認サービス」→「J2C 認証データ」をクリックする。
③ 「新規作成」ボタンを押下し、「一般プロバイダー」のユーザIDとパスワード欄にデータソース取得の際に使用するIDとパスワードを入力する。別名は任意でよい。
④ 「OK」押下後、「保管」をクリックし設定を保存する。
⑤ 管理コンソールの「リソース」→「JDBC」→「JDBC プロバイダー」をクリックするとJDBC プロバイダー画面が表示される。「有効範囲」を「ノード=xxxNode01,サーバー=server1」を選択し、「設定」の「新規作成」ボタンを押下すると新規JDBC プロバイダーの作成画面が表示される。
⑥ データベースタイプ：Oracle
プロバイダータイプ：Oracle JDBC Driver
実装タイプ：接続プールデータソース
名前：任意（Oracle JDBC Driver）
として選択・入力し「次へ」を押下するとデータベース・クラスパス情報入力画面が表示される。
⑦ JDBCドライバ（ojdbc14.jar）のパス（C:\Program Files\IBM\WebSphere\AppServer\lib\ext）を入力し、「次へ」を押下すると要約画面が表示される。
⑧ 「終了」ボタン押下後、「保管」をクリックして設定を保存する。
⑨ 管理コンソールのガイド付きアクティビティーでデータソースを作成する。
⑩ 管理コンソールの「リソース」→「JDBC」→「データソース」をクリックするとデータソース画面が表示される。「有効範囲」を「ノード=xxxNode01,サーバー=server1」を選択し、「設定」の「新規作成」ボタンを押下するとデータソース作成画面が表示される。
⑪ データソース名：任意
JNDI名：データソースの取得に使用されるJNDI名
コンポーネント管理認証別名とXAリカバリーの認証別名：②～④で作成したJ2C認証データを選択し、「次へ」を押下する。

⑫ JDBC プロバイダー選択画面が表示されたら「既存 JDBC プロバイダーを選択」を選択し、⑤～⑧で作成した JDBC プロバイダーを選択し「次へ」を押下する。
⑬ URL:データベースの URL
データストアのヘルパークラス名:使用する Oracle バージョン
選択後、「次へ」を押下する。
⑭ 「終了」ボタン押下後、「保管」をクリックして設定を保存する。
⑮ データソース画面のリストから作成したデータソースをチェックし、「テスト接続」ボタンを押下し接続テストを行う。

(11)プロジェクトを WebSphere 対応にする

① 「/ant/build.properties」を以下のように修正する。なお、以下はデフォルトのフォルダへインストールすることを想定しているため適宜自分の環境に合わせて修正すること。
webapsvr.home=C:/Program Files/IBM/WebSphere/AppServer
webapsvr.lib.dir=C:/Program Files/IBM/WebSphere/AppServer/lib
deploy.dir=C:/
jdbc.driver=C:/Program Files/IBM/WebSphere/AppServer/lib/ext/ojdbc14.jar
② 「/ant/build.xml」を以下のように修正する。
${webapsvr.lib.dir}/servlet-api.jar; → ${webapsvr.lib.dir}/j2ee.jar
${webapsvr.lib.dir}/jsp-api.jar; → 削除
③ データソース用の Bean 定義を JNDI 用に修正する。詳細については、Rich **版機能説明書**『CC-01 JNDI アクセス機能』参照のこと。

Rich 版では、「2．4 データベースアクセス」でデータベースに接続する設定を行っているが、
“dataAccessContext-local.xml”ファイルの dataSource の Bean 定義については、以下のようにすること。

```
  <bean id="dataSource"
      class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="TerasolunaDataSource" />
  </bean>

  <!--
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="
close">
            <property name="driverClassName" value="${jdbc.driverClassName}"/>
            <property name="url" value="${jdbc.url}"/>
            <property name="username" value="${jdbc.username}"/>
            <property name="password" value="${jdbc.password}"/>
  </bean>
  -->
```

④ web.xml ファイルの mime-mapping の設定を削除する。（Rich 版のみ）
以下の網掛け部分のように mime-type の設定から charset の指定を削除する。

```
                                        :
                                       略
                                        :
   <mime-mapping>
        <extension>xml</extension>
        <mime-type>text/xml;charset=UTF-8</mime-type>
   </mime-mapping>

   <locale-encoding-mapping-list>
        <locale-encoding-mapping>
             <locale>ja</locale>
             <encoding>UTF-8</encoding>
        </locale-encoding-mapping>
   </locale-encoding-mapping-list>
```

```
：
略
：
```

※WebSphere の仕様では<mime-type> タグに、「charset=UTF-8」といった文字コードセットの指定を含めることはできない。

(12)アプリケーションの起動

① AST 上で「/ant/build.xml」を右クリックし、「Ant の実行」より「deploy」を実行する。Cドライブ直下に <context.name>.war ファイルができていることを確認する。
② 管理コンソールを開き、左メニューの「アプリケーション」→「エンタープライズ・アプリケーション」をクリックするとエンタープライズアプリケーション画面が表示される。
③ 「インストール」ボタンを押下し、新規アプリケーションへのパスの「ローカル・ファイル・システム」→「絶対パス」に①でデプロイした war ファイルを選択する。「コンテキストルートには」/ant/build.properties の context.name と一致させる。（例：/terasoluna-spring-thin-blank）
④ 「次へ」→「次へ」→「次へ」→「終了」後「保管」をクリックし設定を保存する。
⑤ 左メニューの「アプリケーション」→「エンタープライズ・アプリケーション」からインストールしたアプリケーションをチェックし「始動」ボタンを押下する。
⑥ Rich 版は、テストクライアント画面より http://localhost:9080/<context.name>/secure/blogic.do に電文を送信し確認する。

3.6　チュートリアル学習環境の整備(WebLogic-WTP)

本節では、チュートリアルを学習するための環境整備を WebLogic をサーバ定義した形での WTP を用いた場合について説明する。

■ **想定環境**

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：BEA WebLogic Server 9.2 for Windows
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：WTP 1.5.5

また、WebLogic をサーバ定義として動作させるためのコンピュータの推奨動作環境を以下に示す。

- CPU：Intel Pentium 4 以上のプロセッサ
- メモリ：512MB 以上の空きメモリ

■ **インストール/開発環境の整備**

(1) アプリケーションの用意
(2) アプリケーションのインストール
(3) WebLogic のインストール
(4) プロジェクトの準備
(5) プロジェクトのインポート
(6) データベースの設定
(7) テーブル構築及びデータの作成
(8) ドメインの作成
(9) データソースの作成
(10)プロジェクトを WebLogic 対応にする

(1)～(10)の内容は、「3．4 チュートリアル学習環境の整備(WebLogic)」を参照のこと。(2)アプリケーションのインストール作業にある Tomcat のサーバ定義も必ず行うこと。

(11)ドメインの作成及びデータソースの作成で使用した WebLogic サーバを停止する

① コマンドプロンプトを開き、前節にて作成したドメインのフォルダ\bin へ移動する。
　移動先の例：「C:\bea\user_projects\domains\tutorial1\bin」
② コマンドプロンプトから「stopWebLogic.cmd」を実行すると、WebLogic サーバが停止する。

(12)WebLogic のサーバ定義の追加

① Eclipse の「ファイル」－「新規」－「その他」メニューを選択する。
② ウィザードの選択から「サーバー」－「サーバー」を選択し「次へ」を押下する。
③ 新規サーバーから「BEA Systems」－「汎用 BEA WebLogic サーバー 9.0」を選択し「次へ」を押下する。
④ ランタイムの画面では、以下のように入力して「次へ」を押下する。
　JRE : jdk1.5.0_0x (x はバージョン番号)
　アプリケーション・サーバー・ディレクトリー : C:/bea/weblogic92
⑤ サーバーの設定画面では、以下のように入力して「次へ」を押下する。

ドメイン・ディレクトリー : C:/bea/user_projects/domains/tutorial1
自動デプロイ・ディレクトリー : C:/bea/user_projects/domains/tutorial1/autodeploy
開始スクリプト : C:/bea/user_projects/domains/tutorial1/bin/startWebLogic.cmd
停止スクリプト : C:/bea/user_projects/domains/tutorial1/bin/stopWebLogic.cmd
ポート : 7001
デバッグ・ポート : 8453

⑥ 「プロジェクトの追加及び除去」はデフォルトのまま「終了」を押下する。
⑦ Eclipse の「ウィンドウ」－「ビューの表示」－「その他」メニューを選択する。
⑧ 「サーバー」－「サーバー」を選択し「OK」を押下する。
⑨ 画面右下に表示されたサーバービューに「Generic BEA WebLogic Server v9.0（汎用）」があるのを確認する。
⑩ 「Generic BEA WebLogic Server v9.0（汎用）」をダブルクリックし、「Automatic Publishing」を選択し、「デフォルトの公開設定を使用」にチェックが付いている事を確認し、「Edit」を選択する。
⑪ 設定メニューの中の「サーバの始動時に自動的に公開」のチェックを外し、「OK」を押下する。

(13)プロジェクトを WebLogic 用の WTP プロジェクトに移行する

① 「パッケージ・エクスプローラ」よりインポートしたプロジェクトを開き、「Tomcat v5.5 ランタイム」を選択し右クリックメニューより「構成」を選択する。
② 「 汎用 BEA WebLogic サーバー 9.0 （汎用）」を選択し「終了」を押下する。

(14)WebLogic へのプロジェクト追加

① サーバービューから「Generic BEA WebLogic Server v9.0（汎用）」を選択して右クリックメニューの中から「プロジェクトの追加と除去」を選択する。
② 使用可能プロジェクトの中に「（13）プロジェクトを WebLogic 用の WTP プロジェクトに移行する」で移行したプロジェクトがあるので、構成プロジェクトに追加し、終了する。
③ Eclipse のサーバービューより「Generic BEA WebLogic Server v9.0（汎用）」を選択し右クリックメニューより、「始動」を選択する。
④ 再度「Generic BEA WebLogic Server v9.0（汎用）」を選択し右クリックメニューより、「公開」を選択する。

※ここで「Ant パブリッシャー」についてのメッセージダイアログが表示された場合は、プロジェクトを選択し右クリックメニューより更新を実行した後、再度「公開」を行うこと。

(15)アプリケーションの確認

① Rich 版は、テストクライアント画面より http://localhost:7001/<context.name>/secure/blogic.do に電文を送信し確認する。

■ プロジェクトの除去方法

① サーバが始動中にサーバービューから「Generic BEA WebLogic Server v9.0（汎用）」を選択して右クリックメニューの中から「プロジェクトの追加と除去」を選択する。
② 構成プロジェクトより除去したいプロジェクトを選択し、除去を選択し、終了を押下する。
③ 再度「Generic BEA WebLogic Server v9.0（汎用）」を選択して右クリックメニューの中から「公開」を選択する。

※サーバが停止している状態でプロジェクトの除去を行った場合は、サーバ始動後にサーバの右クリックメニューより「公開」を選択し、同期を取ること。

■ サーバ起動時の注意点

WebLogic をサーバ定義した形での WTP 環境は、リソースをかなり必要とします。推奨動作環境を満たした場合であっても、別のアプリケーションを動作しながらサーバを起動した場合は「タイムアウトは Generic BEAWebLogic Server v9.0(凡用）の始動を待機中です。75000 後にサーバーは始動しませんでした。」といったメッセージが表示されタイムアウトする可能性があります。
サーバの起動時は以下の事前準備を行った後で行うこと。

1、別のアプリケーションを動作させない。
2、編集中のプロジェクト以外は「プロジェクトの除去方法」より除去しておく。
　　サーバ起動時に全てのプロジェクトが自動でデプロイされるため、編集中以外のプロジェクトを除去することで起動時間を短縮させる。

3.7　WTP プロジェクトを非 WTP 環境へ移行する手順

本節では、WTP を用いたプロジェクトを WTP を使わない環境で動作させる方法について説明する。

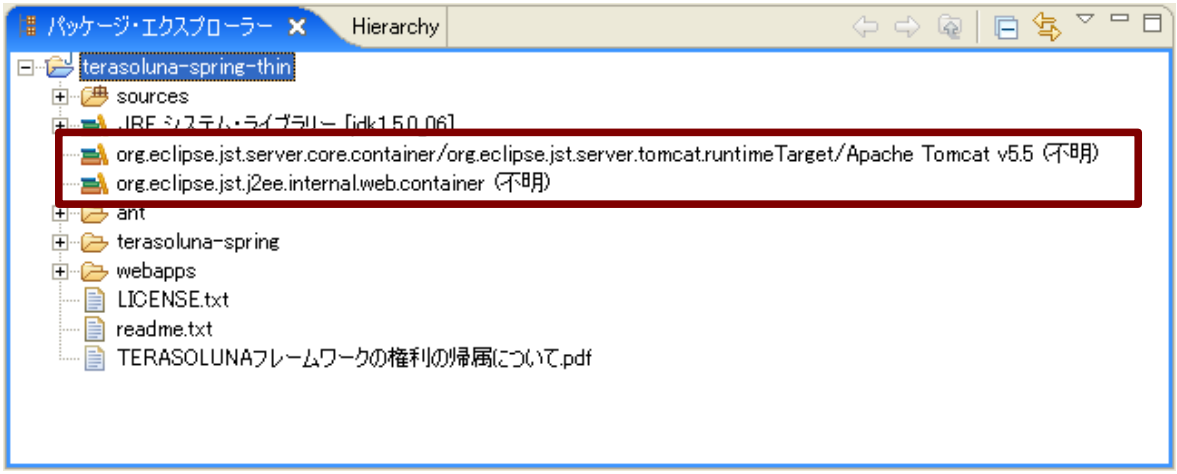
■ **想定環境**

本書では以下の開発環境を想定して、解説している。

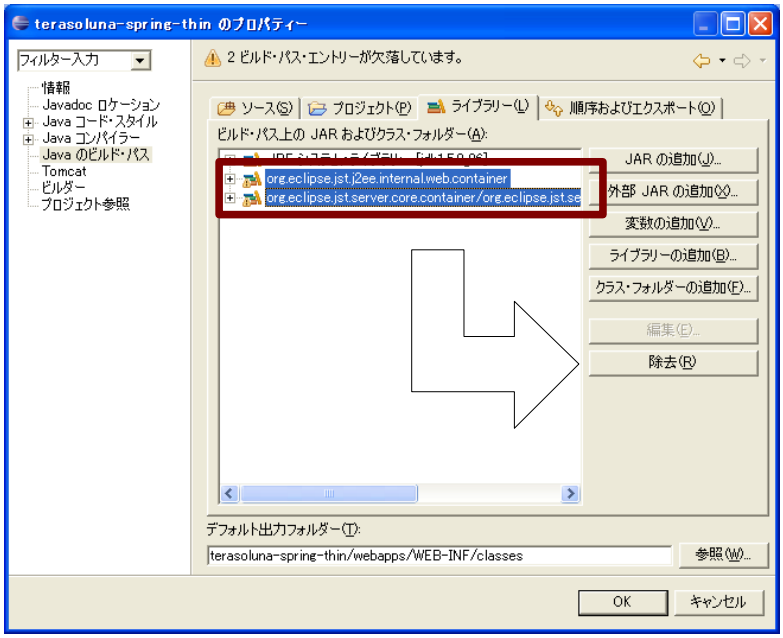
- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.23
- ビルドツール：Apache Ant 1.6.5（Eclipse の Ant プラグインのみでも可）
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：Sysdeo Eclipse Tomcat Launcher plugin 3.1beta

■ 手順

① WTP を用いたプロジェクトを非 WTP 環境へインポートする。すると、ビルドパスの表示部分に（不明）のエラーが表示される。非 WTP 環境のため WTP の設定情報が判別できないため出力されるエラーである。



② （不明）を削除する。プロジェクトを右クリックして「プロパティー」から「Java のビルドパス」のページを開き、「ライブラリー」タブを選択し、エントリを除去する。



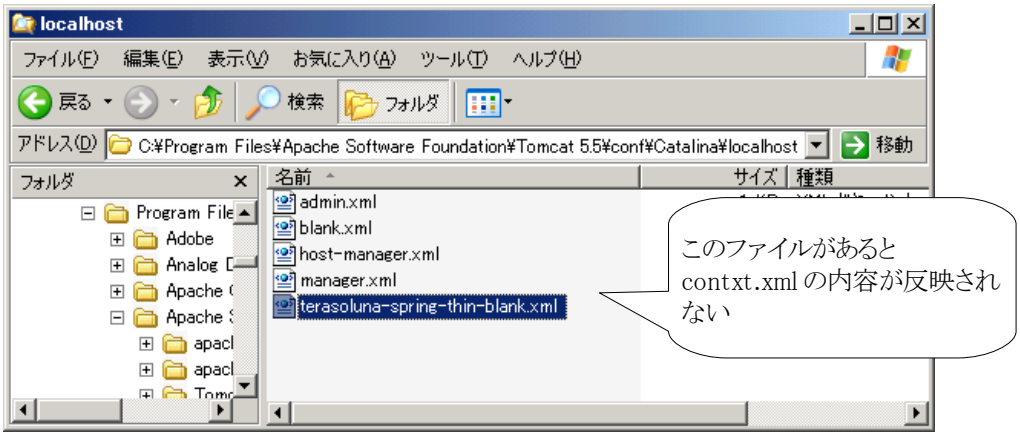
③ 続けて必要なライブラリをビルド・パスに追加する。プロジェクトの WEB-INF/lib に配置した JAR ファイルおよび TOMCAT_HOME/common/lib に存在する JAR ファイルを追加する。

■ デプロイ

/ant フォルダにある build.properties を自分の環境にあうように設定する。その後で build.xml の deploy タスクを実行する。

■ JNDI の設定の注意点

Eclipse プロジェクトの/webapps/META-INF/context.xml ファイルに JNDI の設定を記述してある場合は、TOMCAT_HOME/conf/Catalina/localhost/に<context.name>.xml ファイルが存在しないことを確認する。ある場合は、削除すること。なお、デプロイする度に削除する必要は無く context.xml ファイルを修正した場合のみ必要な作業である。



さらに、TOMCAT_HOME/conf/server.xml ファイルに<context>・・・</context>タグが存在する場合は、削除すること。なお、この作業は一度すればよい。

```
                                        :
                                       略
                                        :
    <Host>
        <Context path="/terasoluna-spring-thin-sample">・・・</Context>
    </Host>

                                        :
                                       略
                                        :
```

■ サーバの起動・停止

AP サーバの起動は以下のイメージにあるアイコンから行う。

AP サーバの停止は以下のイメージにあるアイコンから行う。

3.8　JDK のバージョンを変更する手順

本節では、WTP を用いたプロジェクトに対して JDK のバージョンを変更した場合の設定方法について説明する。

■ 想定環境

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional
- JDK：Java SE5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.23
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5（Eclipse の Ant プラグインのみでも可）
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：WTP 1.5.5

■ 概要

今回の例では JDK のバージョンを「1.5.0_06」から「1.5.0_07」に変更する場合を例にして説明する。
必要な手順としては以下になる。

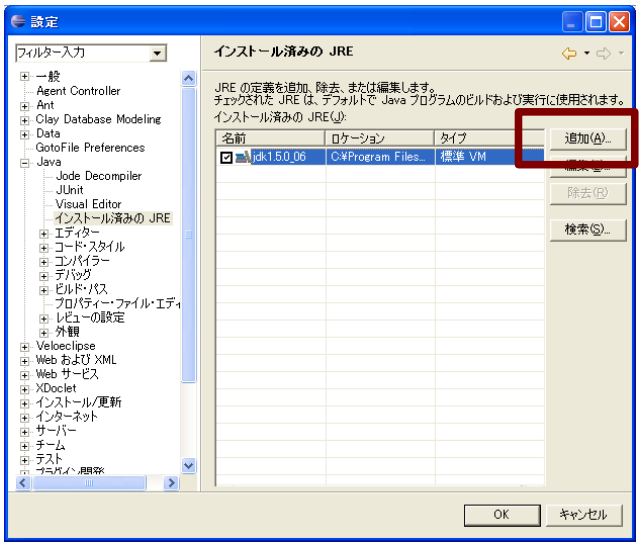
① 「インストール済み JRE」の設定の変更
② 「インストール済みサーバ・ランタイム」の JRE の設定の変更
③ 「Projects Facets」のランタイムの JRE の設定の変更

※手順は上記番号順に行うこと

■ 手順

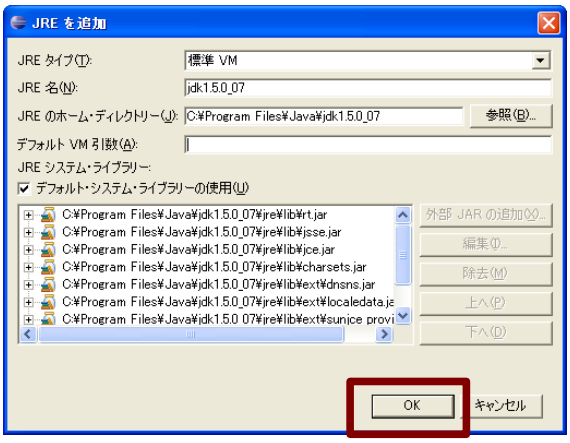
① 「インストール済み JRE」の設定の変更

1. Eclipse のメニューより「ウィンドウ」→「設定」を選択する。
2. 設定ウィンドウより「Java」→「インストール済み JRE」を選択し、「追加」ボタンを押下する。

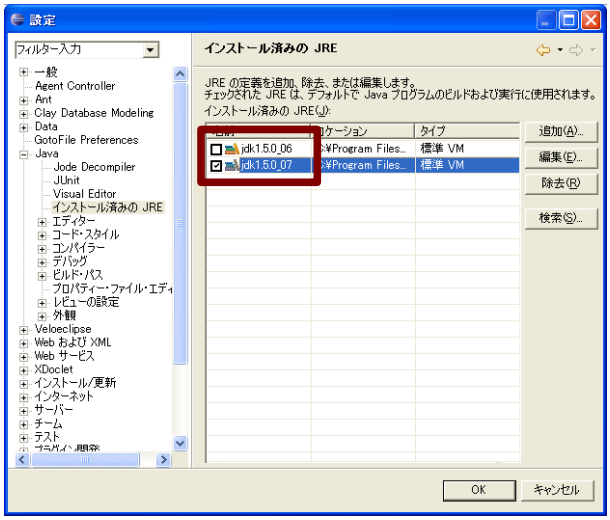


3. 表示された「JRE を追加」の設定ウィンドウより以下の内容を入力し、「OK」ボタンを押下する。

- JRE タイプ：標準 VM
- JRE 名：jdk1.5.0_07 <任意の名前でよいがバージョンを表す文字列とした方がよい>
- JRE のホーム・ディレクトリー：<jdk1.5.0_07 のインストールディレクトリ>
- デフォルト VM 引数：なし
- デフォルト・システム・ライブラリーの使用：チェックあり

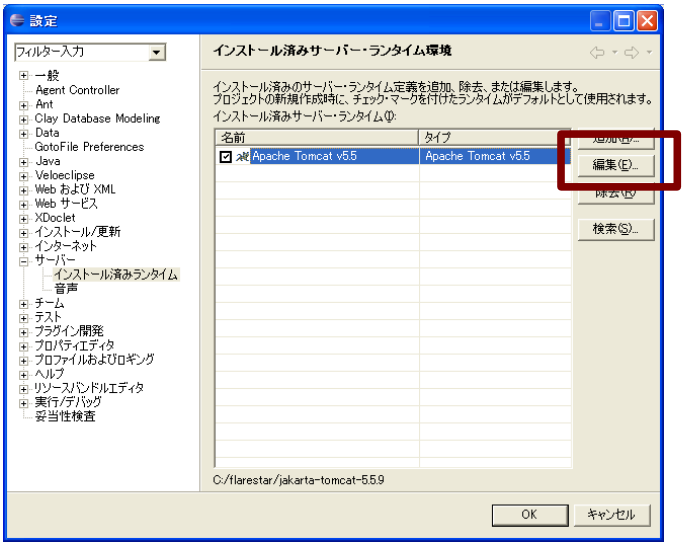


4. 新しく追加した JRE 名にチェックを入れ、「OK」ボタンを押下する。

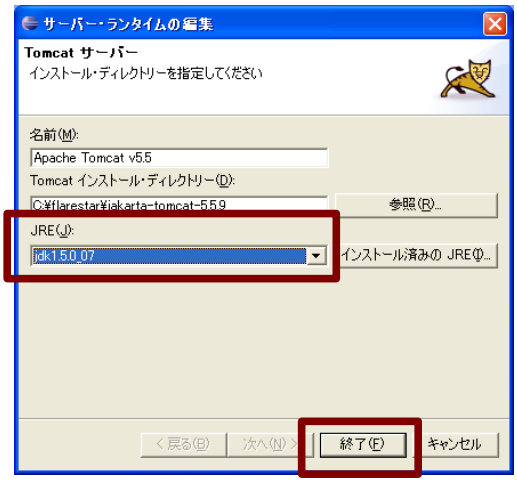


② 「インストール済みサーバ・ランタイム」の JRE の設定の変更
WTP として動作している Eclipse 上の Web アプリケーションサーバのランタイム設定を変更する。

1. Eclipse のメニューより「ウィンドウ」→「設定」を選択する。
2. 設定ウィンドウより「サーバ」→「インストール済みランタイム」を選択し、既に設定されているサーバを選択し、「編集」ボタンを押下する。（サーバが設定されていない場合は、2．2　チュートリアル学習環境の整備を参照すること。）



3. 表示された「サーバー・ランタイムの編集」の JRE のセレクトボックスを新しく追加した JRE 名に変更し、「終了」ボタンを押下する。

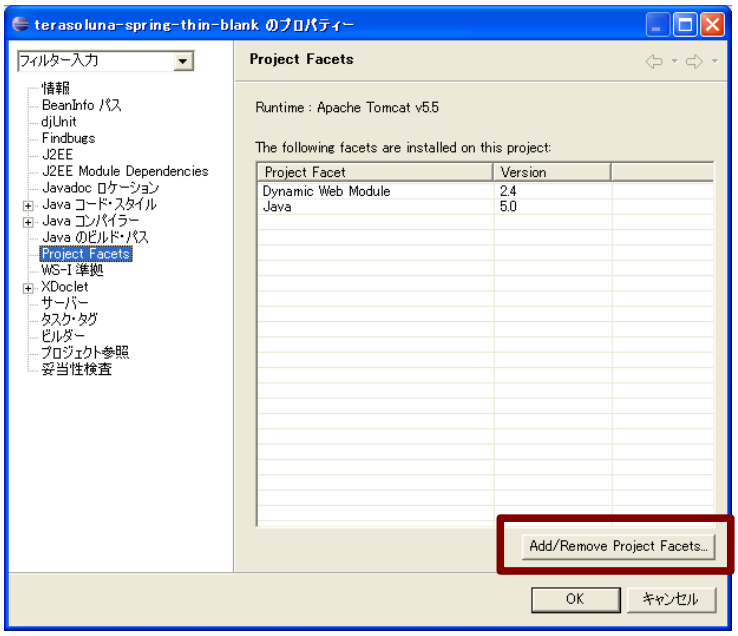


4. 「OK」ボタンを押下し、設定ウィンドウを閉じる。

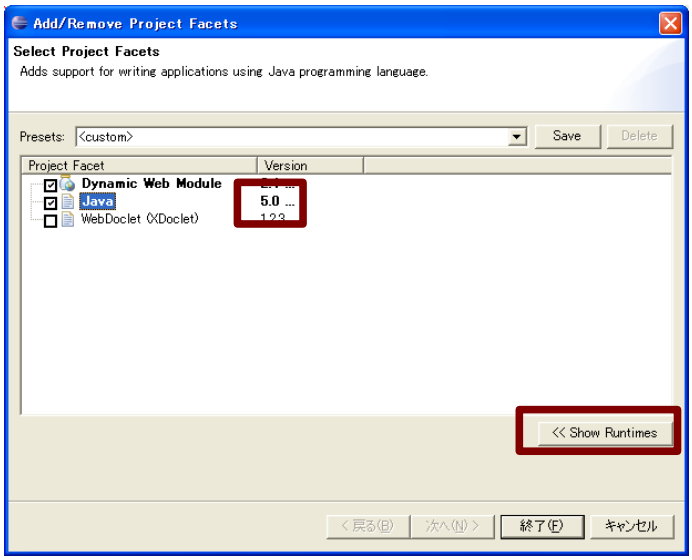
③ 「Projects Facets」のランタイムの JRE の設定の変更

WTP プロジェクトは Eclipse 上では動的 Web プロジェクトとして動いているので、そこの設定を変更する。

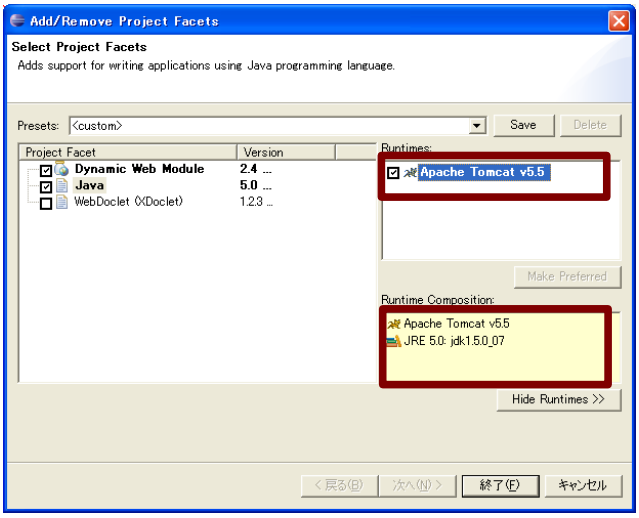
1. 「パッケージ・エクスプローラ」より該当のプロジェクトを選択し、右クリックメニューより「プロパティ」を選択する。
2. 「Projects Facets」を選択し、「Add/Remove Projects Facets」ボタンを押下する。



3. 表示された「Add/Remove Projects Facets」の設定ウィンドウより「Java」のバージョンが「5.0」になっていることを確認し、「<<Show Runtimes」ボタンを押下する。



4. 「Runtimes」の設定内容が表示され、サーバ名を選択し、「Runtime Composition」に新しく追加した JRE 名が表示されることを確認する。



5. 確認したら「終了」ボタンを押下し、「OK」ボタンを押下し、プロパティウィンドウを閉じる。

3.9　チュートリアル学習環境の整備(Cosminexus)

本節では、チュートリアルを学習するための環境整備を Cosminexus をサーバ定義した形での WTP を用いる場合について説明する。

■ 想定環境

本書では以下の開発環境を想定して、解説している。

- OS：Microsoft Windows 2000 / XP Professional Service Pack 2 以上
- JDK：Java SE5.0
- Web アプリケーションサーバ：Hitachi Cosminexus 7.5
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 (Eclipse の Ant プラグインのみでも可)
- 総合開発環境：Eclipse SDK 3.2.2
- Eclipse plugin ：Server Plug-in（リソースアダプタやアプリケーションに対して，プロパティの設定，デプロイ，開始などの操作をする機能）

■ インストール/開発環境整備

(1) アプリケーションの用意

「2.2 チュートリアル学習環境の整備」以外で必要となるアプリケーションを以下に用意する。

- Hitachi Cosminexus 7.5 ・・・以下、Cosminexs

(2) アプリケーションのインストール

Cosminexus 以外は「2.2 チュートリアル学習環境の整備」を参照のこと。

(3) Cosminexus のインストール

Cosminexus のインストール方法について説明する。

① 日立総合インストーラを起動する。(D:\Products\HCD_INST.EXE)
② 日立総合インストーラのメニューが表示されたら、「uCosminexus Developer Professional」を選択し、「インストール実行」を押下する。
③ 確認のダイアログが表示されるので、「OK」を押下する。
④ ライセンスの同意が求められるので、「OK」を押下する。
⑤ セットアッププログラムが起動するので、「次へ」を押下する。
⑥ インストール先のディレクトリを設定する。特に指定しない場合はデフォルト（C:\Program Files\Hitach\Cosminexus）のまま「次へ」を押下する。
⑦ コンポーネントの選択で、「標準」を押下する。
⑧ ユーザ名と会社名の入力画面になるので、適切な項目を入力し、「次へ」を押下する。
※会社名を空欄にすることはできないので、何らかの名称を設定する。
⑨ プログラムフォルダの選択画面となる。基本的にはそのまま「次へ」を押下する。
⑩ インストール内容の確認画面が表示される。設定項目に問題がなければ「次へ」を押下する。
⑪ インストールが開始される。インストールが完了すると、「～のインストールを終了しました。」と表示されるので、「完了」を押下する。
⑫ リブートを行いますかと聞かれるので、「いいえ」を押下する。この後、環境変数を変更した際に再起動を行うためここでは行わない。
⑬ 日立総合インストーラダイアログで「終了ボタン」を押下する。

(4) システム環境変数の設定

Cosminexus で使用する環境変数を以下のように設定する。

	変数名	値
編集	Path	<Cosminexus のインストールディレクトリ>\DB\CLIENT\UTL; <Cosminexus のインストールディレクトリ>\DB\BIN; <Cosminexus のインストールディレクトリ>\DAB\lib; <Cosminexus のインストールディレクトリ>\TPB\bin; **それぞれを先頭に付け足す。**
新規	TZ	JST-9
新規	VBROKER_ADM	<Cosminexus のインストールディレクトリ>\TPB\adm
新規	TPDIR	<Cosminexus のインストールディレクトリ>\TPB

※環境変数の設定が終了したら OS を再起動させる。

(5) 運用管理エージェントの起動停止設定

OS の起動と停止に合わせ、運用管理エージェントを起動・停止する設定を行う。

① コマンドプロンプトで以下を実行。

```
C:\>"<Cosminexus のインストールディレクトリ>\manager\bin\mngautorun" both
```

② 実行結果が以下のようになっていることを確認。

```
KEOS21303-I Settings have been successfully completed.
```

(6) 運用管理エージェントの起動

運用管理エージェントを起動させる。

① コマンドプロンプトで以下を実行。

```
C:\>"<Cosminexus のインストールディレクトリ>manager\bin\adminagentctl" start
```

② 実行結果が以下のようになっていることを確認。

```
KEOS21200-I The administration agent has started
```

(7) Management Server の設定

Management Server をセットアップし、起動させる。

① Management Server をセットアップする。
コマンドプロンプトで以下を実行。

```
C:\>"<Cosminexus のインストールディレクトリ>\manager\bin\mngsvrctl" setup
```

② 実行結果が以下のようになっていることを確認。

```
KDJE41800-I The setup for the Web container server has finished successfully. Server
name = cosmi_m
KEOS10118-I Management server was initialized normally.
```

③ Management Server の起動
コマンドプロンプトで以下を実行。

```
C:\>"<Cosminexus のインストールディレクトリ>\manager\bin\mngsvrctl" start
```

④ 実行結果が以下のようになっていることを確認。

```
KEOS10110-I Management Server service has started from command line.
```

⑤ Management Server の管理者アカウントの設定
コマンドプロンプトで以下を実行。

```
C:\>"<Cosminexus のインストールディレクトリ>\manager\bin\cmx_admin_passwd" -m
localhost -u admin -p admin
```

※-m ホスト（ポート）、-u 任意のユーザ ID、-p 任意のパスワード
例：「"<Cosminexus のインストールディレクトリ>\manager\bin\cmx_admin_passwd"
-m 0.0.0.0:28080 -u <YouserId> -p <YouserPassword>」

⑥ 実行結果が以下のようになっていることを確認。

```
KEOS24117-I The setting of the administrator's user account were changed.
```

⑦ Management Server の管理者アカウントの設定。
Server Plug-in を使用するために、mserver.properties に以下のプロパティを追加する。

<Cosminexus のインストールディレクトリ>\manager\config\mserver.properties

```
com.cosminexus.mngsvr.management.enabled=true
com.cosminexus.mngsvr.management.connector.enabled=true
com.cosminexus.mngsvr.management.port=28099
```

⑧ コントロールパネルの「管理ツール」→「サービス」から「Cosminexus Management Server」を選択し、右クリックして「再起動」を押下する。
⑨ ブラウザに「http://localhost:28080/mngsvr」を入力し、Management Server の画面が表示されていることを確認する。

(8) Web システムの構成を定義

① ブラウザから Management Server を開き、ログインする。（設定した管理ユーザ ID とパスワード）
② 運用管理ポータル画面から「運用管理ドメインの構成定義」を選択する。
③ 「ホストビュー」タグから「運用管理ドメインの構成定義」直下の「ホスト」を選択する。
④ ホスト名を入力（使用している PC のコンピューター名）する。
⑤ 「定義」ボタンを押下する。
⑥ 「サーバビュー」タグの「DefaultDomain」→「論理パフォーマンストレーサ」→「パフォーマンストレーサ」を選択する。
⑦ 「論理サーバ名」に任意のサーバ名を設定する。（以下、<performance>）
⑧ 「追加」ボタンを押下する。
⑨ 「サーバビュー」タグの「DefaultDomain」→「論理 J2EE サーバ」→「J2EE サーバ」を選択する。
⑩ 「論理サーバ名」に任意のサーバ名を設定する。（以下、<j2ee>）
⑪ 「追加」ボタンを押下する。
⑫ 「サーバビュー」タグの「DefaultDomain」→「論理 Web サーバ」→「Web サーバ」を選択する。

⑬「論理サーバ名」に任意のサーバ名を設定する。(以下、<web>)
⑭「追加」ボタンを押下する。
⑮「サーバのセットアップ」を押下する。
⑯「全てセットアップ」ボタンを押下する。

(9)論理サーバの環境設定

① ブラウザから Management Server を開き、ログインする。(設定した管理ユーザ ID とパスワード)
②「運用管理ポータル」画面から「論理サーバの環境設定」を選択する。
③「サーバービュー」タグの「DefaultDomain」→「論理 J2EE サーバ」→「J2EE サーバ」→「<j2ee>」を選択する。
④「基本設定」タグの、「利用する論理サーバ設定」の「利用するパフォーマンストレーサ」項目で、「<performance>」を選択する。
⑤「基本設定」タグの「起動オプションの設定」の「セキュリティマネージャの使用」項目で、「しない」をチェックをする。
⑥「適用」ボタンを押下する。
⑦「コンテナ」タグの「拡張パラメタ」の「有効」項目に、チェックをいれ、「拡張パラメタ」項目に「add.class.path=<ojdbc ドライバの完全修飾名>」を入力する。
※通常は「add.class.path=C:/Program Files/Hitachi/Cosminexus/CC/client/lib/ojdbc14.jar」
⑧「適用」ボタンを押下する。
⑨「サーバービュー」タグの「DefaultDomain」→「論理 Web サーバ」→「Web サーバ」→「<web>」を選択する。
⑩「リダイレクタ」タグの「パフォーマンストレーサに関する設定」の「利用するパフォーマンストレーサ」項目で、「<performance>」を選択する。
⑪「適用」ボタンを押下する。
⑫「マッピング」タグのマッピングに関する設定の「有効」にチェックを入れる。
⑬「URLパターン」項目に「/<任意の名前>」(以下、<test>)を入力し、「論理サーバ名」項目で「<j2ee>」を選択する。
⑭「追加」ボタンを押下する。
⑮「戻る」を押下し、「URLパターン」項目に「/<test>/*」を入力し、それ以外は⑫、⑬、⑭と同様の処理を行う。
⑯ 画面右上の「設定情報の配布」を押下し、「全て配布」を押下する。
⑰ 運用ポータル画面から「論理サーバの起動/停止」を押下する。
⑱「ホストビュー」タグから「ホスト」を選択し、一括起動タグを押下し、「実行」ボタンを押下する。
⑲「ログ」タグから全て起動したことを確認する。(多少時間がかかる)

(10)Server Plug-in の設定

Server Plug-in はリソースアダプタやアプリケーションに対して、プロパティの設定、
デプロイ、開始などの操作をする機能

① Eclipse を起動して、メニューから「ヘルプ」→「ソフトウェア更新」→「構成の管理」を選択する。
② 拡張ロケーションの追加を選択し、
「<Cosminexus のインストールディレクトリ>\plugins\eclipse」を選択し、「OK」を押下する。
③ 再起動のダイアログが表示されるので Eclipse を再起動し、メニューから「ウィンドウ」→「パースペクティブを開く」→「その他」→「Comsminexus Server Plug-in」が選べるようになっていることを確認する。
④ メニューから「ウィンドウ」→「設定」を選択する。
⑤ 左ペインから「Cosminexus Server Plug-in」を選択する。
⑥「接続ホスト」タグで「localhost」を選択し、「編集」ボタンを押下する。
⑦「接続ホストの編集」でポート番号が「28099」になっていることを確認する。
⑧ メニューから「ウィンドウ」→「パースペクティブを開く」→「その他」→

「Comsminexus Server Plug-in」を選択し、「OK」を押下する。

⑨ サーバー・エクスプローラに「localhost:28099(未接続)」があることを確認する。

⑩ RMIレジストリのポート番号を userconf.properties に追加する。

<Cosminexus のインストールディレクトリ>\CC\admin\usrconf\usrconf.properties

```
ejbserver.rmi.naming.port=23152
```

(11)プロジェクトを Cosminexus 対応にする

① プロジェクト内にある「/ant/build.properties」を以下のように修正する。なお、以下はデフォルトのフォルダへインストールすることを想定しているため適宜自分の環境に合わせて修正すること。

```
webapsvr.home=C:/Program Files/Hitachi/Cosminexus
webapsvr.lib.dir=C:/Program Files/Hitachi/Cosminexus/CC/client/lib
deploy.dir=C:/
jdbc.driver=C:/Program Files/Hitachi/Cosminexus/CC/client/lib/ojdbc14.jar
```

② 「/ant/build.xml」を以下のように修正する。

```
${webapsvr.lib.dir}/servlet-api.jar;  →  ${webapsvr.lib.dir}/j2ee.jar
```

③ データソース用の Bean 定義を JNDI 用に修正する。詳細については、Rich **版機能説明書**『CC-01 JNDI アクセス機能』参照のこと。

(12)アプリケーションをデプロイする。

① プロジェクトを Ant 実行により war ファイルにする。

② 以下をコマンドプロンプトで以下を実行。

```
C:\>”<Cosminexus のインストールディレクトリ>\CC\admin\bin\cjimportres” <j2ee>
-type war -f <war ファイルの完全修飾名>
```

※<war ファイルの完全修飾名>は、ここでは C:/<プロジェクト名>.war

③ 実行結果が以下のようになっていることを確認。

```
KDJE37400-I Connecting to <j2ee>...
KDJE37404-I Importing <war ファイル名から自動でつけられる名前>...
KDJE37501-I WAR has bean imported successfully.(name = [<war ファイル名から自
動でつけられる名前>])
```

※<war ファイル名から自動でつけられる名前>は、元の war ファイル名が「tutorial-thin.war」の場合、「tutorial_thin_war」と命名されている。

④ 以下をコマンドプロンプトで以下を実行。

```
<Cosminexus のインストールディレクトリ>\CC\admin\bin\cjaddapp <j2ee> -type war
-name <任意の名前> -resname <war ファイル名から自動でつけられる名前>
```

（<任意の名前>は以下、<project>）

⑤ 実行結果が以下のようになっていることを確認。

```
KDJE37400-I Connecting to <j2ee>...
KDJE37401-I Searching <project>...
KDJE37402-I Creating <project>...
KDJE37406-I Adding <war ファイル名から自動でつけられる名前>...
KDJE37503-I WAR has bean added successfully.(name = [<war ファイル名から自動で
```

つけられる名前>])

(13)アプリケーションの起動

① **メニューから「ウィンドウ」→「ビューの表示」→「その他」→「**Cosminexus Server Plug-in**」→**「サーバ・エクスプローラ」**を選択し、「OK」を押下する。**
② 「localhost:28099(未接続)」を右クリックし、「ログイン」を押下する。
③ Management Server の管理ユーザ ID とパスワードを入力し、「OK」を押下する。
④ 「localhost:28099(未接続)」→「DefaultDomain」を右クリックし、「開始」を押下する。
⑤ 「DefaultDomain」→「コンピューター名」→「<j2ee>」**→「**J2EE リソース」→「リソース・アダプター」を右クリックして、「リソース・アダプター(Connector)のインポート」を押下する。
⑥ ファイルパス「${cosminexus}/CC/DBConnector_Oracle_CP.rar」を選択し、「OK」を押下する。
⑥ 直下に作成された DB コネクタを選択し、右クリックし、「プロパティ」を押下する。
⑦ データベースアクセス情報を記述する。
(DB_Connector_for_Oracle\リソース・アダプター\アウトバウンド・リソース・アダプター\コネクション定義\javax.sql.DataSources)

- コンフィグレーション・プロパティ
  以下を各自の環境にあった値を設定する。
  ◇databaseName(コンフィグレーション・プロパティの値)
  ◇serverName(コンフィグレーション・プロパティの値)
  ◇portNumber(コンフィグレーション・プロパティの値)
- コネクター・ランタイム
  ◇右クリック→「追加」→「リソース外部プロパティ」を押下する。
  ◇作成したリソース外部プロパティの別名に「TerasolunaDataSource」と設定する。
  ◇プロパティを選択し、Oracle にアクセスする「ユーザ名」「パスワード」を設定する。

⑧ サーバ・エクスプローラ内で、「DB_Connector_for_Oracle(設定中)」 を右クリックで「デプロイ」を押下する。
⑨ デプロイした「DB_Connector_for_Oracle」が「J2EE リソースアダプター」配下に作成されている。
⑩ 作成された「DB_Connector_for_Oracle」を右クリックし、「開始」を押下する。
⑪ 「J2EE アプリケーション」直下にあるプロジェクト(<project>)を右クリックし、「プロパティ」を押下する。
⑫ 「WAR」タグを選択し、「WARs」→「WAR」→「<war ファイル名から自動でつけられる名前>」を選択する。
⑬ コンテキストルートを任意の値に変更する。
※(8)論理サーバの環境設定の手順⑬で設定した任意の名前(<test>)に設定する。
ここでは「/<test>」になる。
⑭ 「J2EE アプリケーション」直下にあるプロジェクト(<project>)を右クリックし、「開始」を選択する。
⑮ ブラウザに「http://localhost:8080/<test>/」を入力し、一覧表示・登録・削除が行えることを確認する。
※<test>の後ろに必ず「/」を記述する。