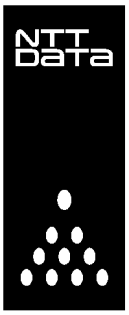


TERASOLUNA Server Framework for Java

Spring 版 チュートリアル

株式会社ＮＴＴデータ

第1章
概要

1.1 本書の目的

「TERASOLUNA Server Framework for Java チュートリアル」（以下、「本書」という。）は、TERASOLUNA Server Framework for Java (以下、Spring 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、Struts フレームワーク、Spring フレームワーク、Spring 版の説明など。
- 第 2 章 Web ブラウザ対応版 チュートリアル
  Web ブラウザ対応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、3 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、Spring 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 Spring 版について説明していく。

■　Spring 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 Web ブラウザ対応版に Struts 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、Struts フレームワークの概略、Spring フレームワークの概略、Spring 版の概略について次節以降、説明していく。

1.5 Struts フレームワーク概略

Struts フレームワークは Web アプリケーションにおいて、主に画面遷移を制御するためのフレームワークである。本節では Struts フレームワークの概略について説明する。

■ Struts フレームワーク 概略

下図の各層は、アプリケーション設計構造の大枠となる論理的なレイヤである。本書では、以下の 4 層に区分する。

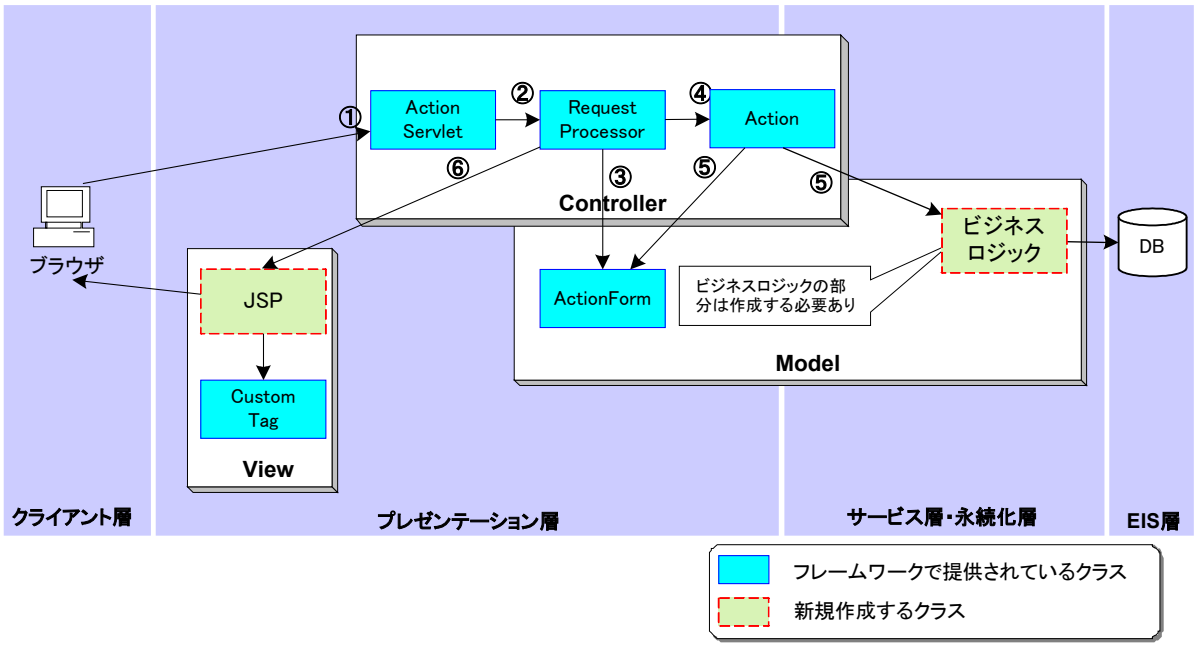


図1 Struts フレームワークのクラス構造

以下の各層は J2EE のアーキテクチャにもとづいたものである。

表1. 各層の説明

クライアント層	：ブラウザが配置される。JavaScript が実行される場合もある。
プレゼンテーション層	：動的 Web ページ表示・入出力処理が配置される。
サービス層	：業務処理が配置される。
永続化層	：業務データ（主にデータベース）と連携した処理が配置される。
EIS(Enterprise Information System)層	：業務データ（主にデータベース）が配置される。

Struts は、Model-View-Controller（MVC）アーキテクチャを採用しており、仕様のライフサイクルが異なるビジネスロジック（Model）と処理制御（Controller）、画面（View）を分離することにより、画面などの仕様変更に柔軟に対応できる構成となっている。

■ **リクエスト処理フロー**

図 2 の番号にそってリクエスト処理フローを解説する。

① ActionServlet クラスが、クライアントからのリクエストを受け付け、現在のリクエストに対応するモジュールを選択する。

② ActionServlet クラスは、現在のリクエストとレスポンスオブジェクトを引数として RequestProcessor クラスの process メソッドを呼び出し、RequestProcessor クラスはリクエストに対応したマッピング情報を検索し、そのマッピング情報を持つ ActionMapping インスタンスを取得する。

③ RequestProcessor クラスは、（手順②で取得した）マッピング情報をもとに、ActionForm インスタンスを取得する。ActionForm インスタンスが生成されていない場合はインスタンスを生成し、生成済みの場合は再利用する。ActionForm インスタンス取得後、RequestProcessor クラスは、ActionForm インスタンスに対して、リクエストデータを設定する。設定後、データ検証が行われる。

④ RequestProcessor クラスは、（手順②で取得した）マッピング情報をもとに、リクエストのパスに対応した Action インスタンスを取得する。Action インスタンスが生成されていない場合はインスタンスを生成し、生成済みの場合は再利用する。

⑤ Action インスタンス取得後、RequestProcessor クラスは（手順③で生成した)ActionForm インスタンス、（手順②で取得した）ActionMapping インスタンスなどを引数として Action クラスの execute メソッドを呼び出す。execute メソッドにはリクエストに対応した処理を記述し、ビジネスロジックを呼び出す。ビジネスロジック実行結果として返された遷移先論理名を ActionForward インスタンスとして RequestProcessor クラスに返す。

⑥ RequestProcessor クラスは、ActionForward インスタンスとして取得した実行結果の遷移先論理名と、（手順②で取得した）マッピング情報をもとに遷移先画面のパスを取得し、画面をフォワードまたはリダイレクトを行い、クライアントに返す画面を生成、表示する。

■ (参考)フレームワークで規定しているクラス構造

Struts では、データベースアクセスを制御するクラスや業務ロジックを定義するクラスは存在しない。これらのクラスは、JavaBeans や EJB などの技術を利用して、独自に定義する必要がある。Spring 版を始めとする Struts 拡張フレームワークの多くは、この点を補っているものが多い。
以下に、Struts で提供されている主なクラスを説明する。

(1) ActionServlet

クライアントから来るリクエストの受付窓口であり、Java Servlet として（詳細には HttpServlet クラスを継承して）実装される。このクラスでは RequestProcessor の初期化や、リクエストを適切な RequestProcessor に振り分ける処理、各種リソースの制御などを行っている。

(2) RequestProcessor

RequestProcessor はリクエスト処理の中心となるクラスで、リクエストで指定された URL と struts 設定ファイル（struts-config.xml）の設定内容に従って、ActionForm への入力データの格納、Action の呼び出し、JSP への遷移（フォワード、リダイレクト）などを行っている。また、モジュール化を実現するためのクラスでもある。モジュール化の概念を以下に示す。

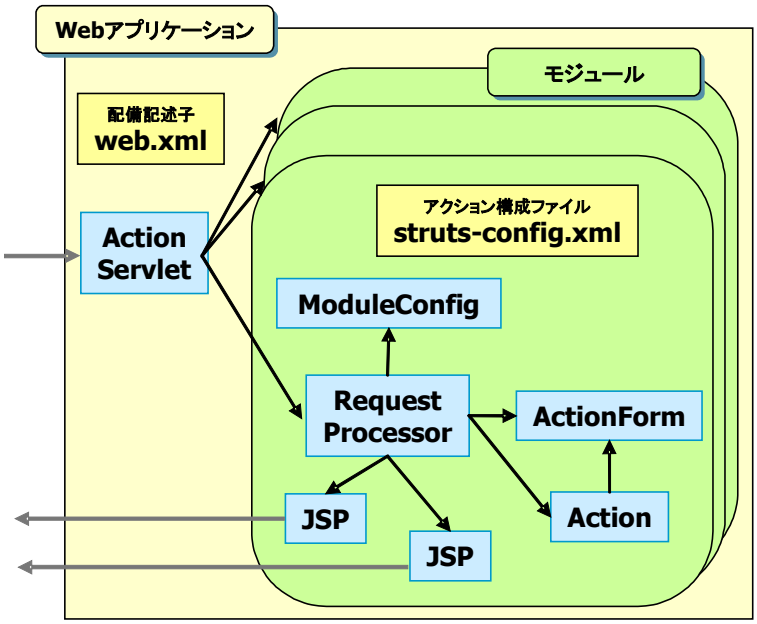


図2 モジュール化の仕組み

モジュール化の核になるクラスで、各モジュールが RequestProcessor のインスタンスを 1 つ保有し、このインスタンスがモジュール内の処理制御を行う。モジュール化により、分散開発が容易となった。

(3) ModuleConfig

struts 設定ファイル（struts-config.xml）の定義情報のうち、モジュールに関する定義情報を保持するためのクラス。ActionServlet の初期化時に struts 設定ファイル（struts-config.xml）を読み込み、このクラスのインスタンスに定義情報を格納する。

(4) ActionMapping

struts 設定ファイル（struts-config.xml）の定義情報のうち、Action に関する定義情報を保持するためのクラス。RequestProcessor の初期化時（ActionServlet の初期化時に実行される）に struts 設定ファイル（struts-config.xml）を読み込み、このクラスのインスタンスに情報を格納する。
ActionMapping のインスタンスは ModuleConfig 内の HashMap で管理され、リクエストで指定された URL の “xxxx.do”の部分をキーに、HashMap から ActionMapping のインスタンスを検索し、そのインスタンスに格納されている情報をもとに処理の振り分けや遷移先の指定が行われる。

(5) ActionForm

HTML のフォームに入力されたパラメータを格納するクラスである。入力パラメータに対して、入力チェックを行うことができる。JavaBean として実装される。Struts1.0 系では入力されるデータの種類や組み合わせに合わせて、あらかじめ ActionForm を継承してデータを格納するクラスを定義しておく必要があったが、Struts1.1 では入力されるデータの種類と組み合わせを struts 設定ファイル（struts-config.xml）に定義しておくことで、クラスをあらかじめ定義しなくても入力データを格納できる機能が提供された。

(6) Action

業務ロジックを呼び出すクラスである。開発者はこのクラスを継承して各 Action クラスを設計する必要がある。Action クラス自身に業務ロジックを持つことも可能であるが、Struts では推奨されておらず、通常は業務ロジック用のクラスを定義して、そのインスタンスを呼び出す。業務ロジックを実行した戻り値を受け取って、遷移先情報として ActionForward クラスのインスタンスを返却する。Action クラスの実装においては、スレッドセーフとなるよう注意する必要がある。

(7) ActionForward

遷移先と遷移の手順（フォワード、またはリダイレクト）の情報を格納するクラス。Action クラスのインスタンスを呼び出した際の戻り値として使用され、RequestProcessor は ActionForward に格納されている情報にもとづいて画面遷移を実行する。

(8) JSP

動的に HTML を生成するための仕組み。JSP は最終的には Servlet クラスに変換され、コンパイルされることで動的に HTML を生成する。

(9) Custom Tag

JSP 内に HTML とプログラムが混在するとコードが複雑化し可読性も落ちるので、表の表示など共通的に使うものはタグハンドラクラスとして切り出して設計する。アプリケーション開発者は必要に応じて拡張することができる。Struts では様々な種類のタグライブラリを以下の 5 種類に分けて提供している。

- struts-bean タグライブラリ
  JavaBeans の定義、取得、内容表示機能を提供するタグライブラリ。
- struts-html タグライブラリ
  HTML フォームと Struts の ActionForm とを結びつける機能を提供するタグライブラリ。
- struts-logic タグライブラリ
  比較、存在チェック、繰り返し制御機能を提供するタグライブラリ。
- struts-nested タグライブラリ
  階層化されたプロパティに効率的にアクセスする機能を提供するタグライブラリ。
- struts-tiles タグライブラリ
  レイアウトとコンテンツを分離するための枠組みを提供するタグライブラリ。

1.6　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。

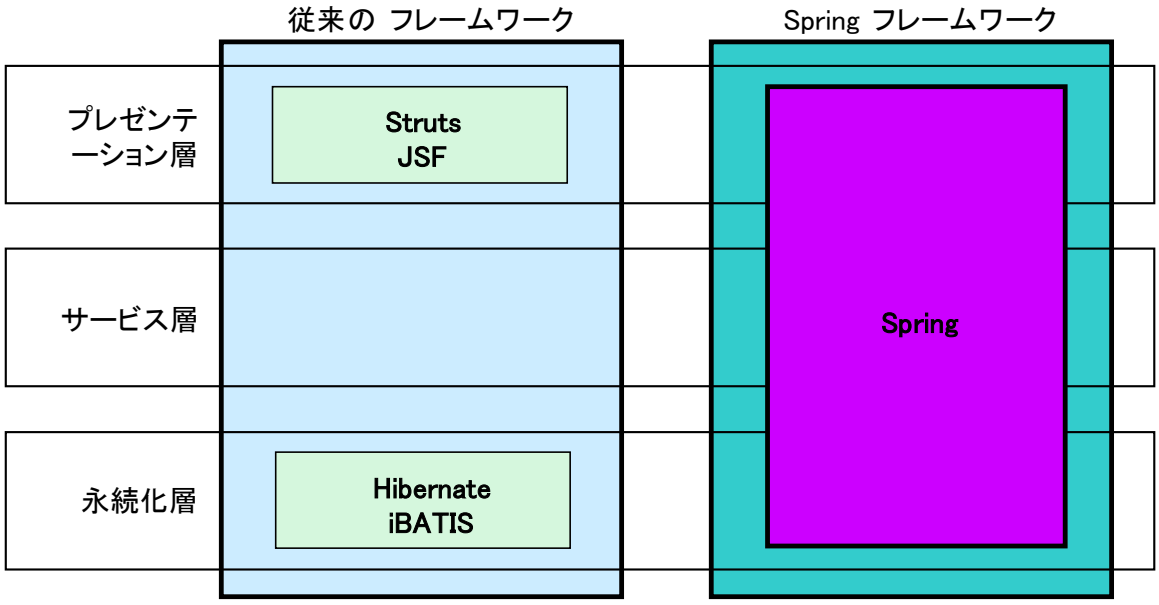


図3　Spring フレームワーク概略

(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

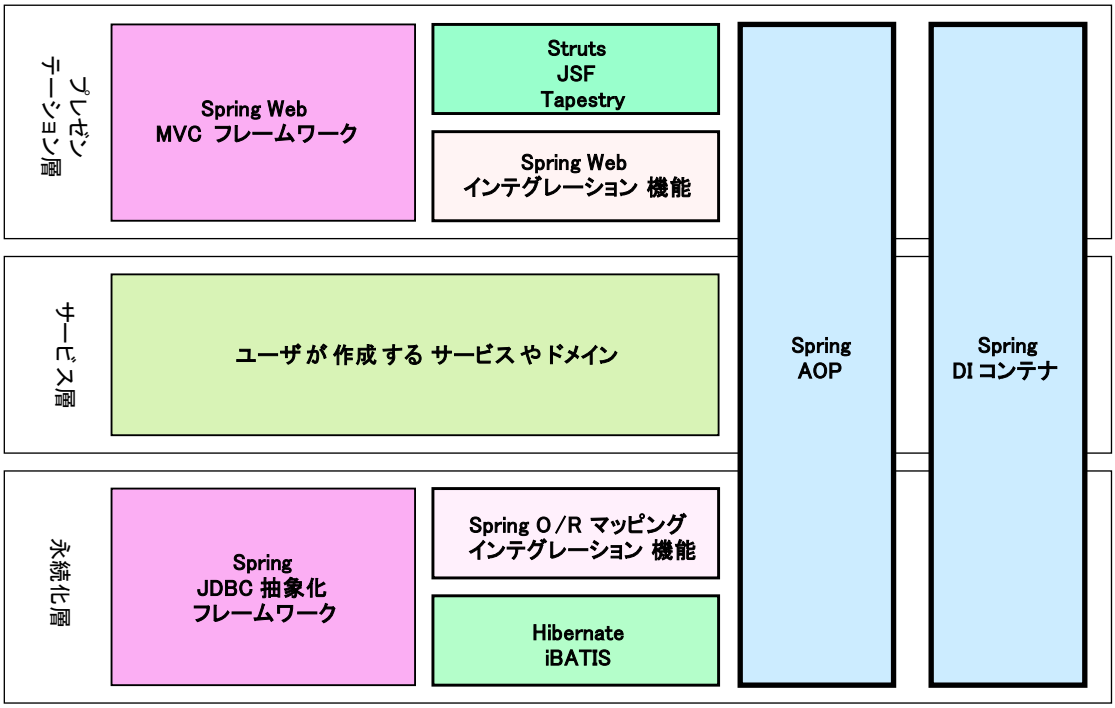


図4 Spring フレームワークの機能

(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
- 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（Web ブラウザ対応版では、プレゼンテーション層のフレームワークとして Struts を用いている）。

(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。

1.7 Spring 版概略

1.7.1 Web ブラウザ対応版

本書で採用する J2EE フレームワークである Web ブラウザ対応版について説明する。Web ブラウザ対応版は Spring フレームワークおよび Struts フレームワークを機能拡張して提供している。

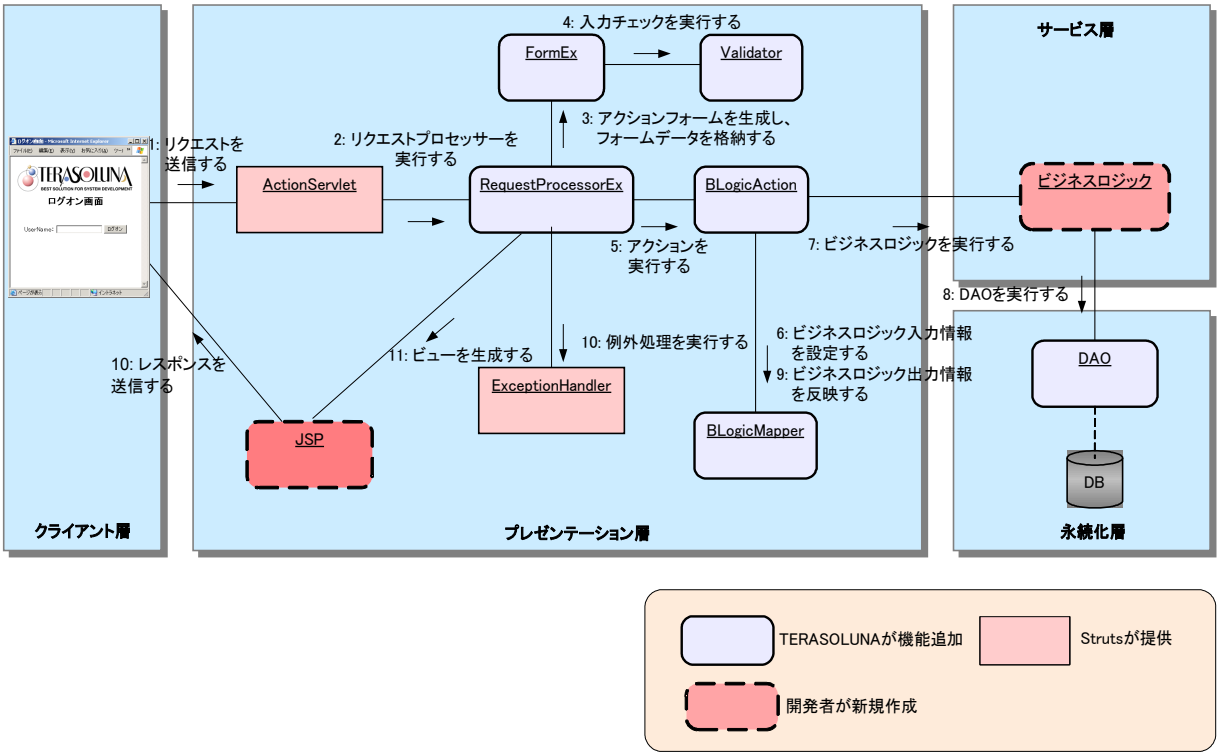


図5 Web ブラウザ対応版のクラス構造

■ フレームワークが規定しているクラス構造

Web ブラウザ対応版 Spring 版では、Spring フレームワークや Sturts 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
- サーバ閉塞機能
- 業務閉塞機能
- POJO 及び定型インタフェースを持つ業務ロジックを起動する仕組み
- J2EE/JDBC の知識がなくともデータベースアクセスが可能な仕組み
- 一覧表示画面でのページ操作等をサポートするカスタムタグ等

次に、Web ブラウザ対応版が提供するクラスについて説明していく。

(1) RequestProcessorEx

"_"で始まるアクションフォーム名をセッション内で唯一性を保証する機能など TERASOLUNA フレームワークで新規追加されたアクションマッピングのプロパティに対応した処理や、アクションでのエラー発生時にログ出力を行う処理などを実現している。

(2) Validator

ビジネスロジックの入力となる ActionForm 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。

(3) ActionEx

Action クラスを継承した抽象基底クラスである。ログ出力機能・トランザクショントークンチェック機能を追加している。

- AbstractBLogicAction
  ActionEx を継承した抽象クラスである。ビジネスロジックの起動を行うアクションクラスに共通する機能を集約した抽象クラスである。POJO の場合は AbstractBLogicAciton を継承してアクションを作成する。web 層のオブジェクト（Request、Session、ActionForm）とビジネスロジックの入出力となる JavaBean の相互変換を行う（実際の変換処理は、BLogicMapper を利用している）。
- BLogicAction
  AbstractBLogicAction クラスを継承したビジネスロジック起動クラスである。

(4) BLogicMapper

ビジネスロジックの入出力処理を行うクラスである。設定ファイル（blogic-io.xml）をもとに以下の処理を行う。

- Web 層のオブジェクト（Form、Session、Request、ServletContext）よりビジネスロジックの入力となる JavaBean を生成する。
- ビジネスロジックの実行結果として返却される JavaBean を、Web 層のオブジェクト（Form、Session、Request）に反映させる。

(5) BLogicResult

ビジネスロジックの出力情報クラスである。ビジネスロジックの結果オブジェクトや遷移先情報、エラーメッセージを保持する。

(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である。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。

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
  Web ブラウザ対応版が提供するビジネスロジックのインタフェース。BLogicAction 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、Stored ProcedureDAO）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) JSP

Struts フレームワークを利用した実装の場合と同様に主に JSP で実装する。

(9) Custom Tag

Web ブラウザ対応版では Struts フレームワークが標準で提供しているカスタムタグに加え、一覧表示画面でのページ操作をサポートするタグなどを新たにカスタムタグとして提供している。

第2章
Web ブラウザ対応版
チュートリアル

2.1　チュートリアル概要

本書では、Web ブラウザ対応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2. 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 画面遷移	: 基本的な画面遷移を行う方法を学ぶ。
2.4 一覧表示	: 入力したデータを一覧表示する方法を学ぶ。
2.5 ログオン	: ログオン処理の方法を学ぶ。
2.6 データベースアクセス	: データベースからデータを取得する方法を学ぶ。
2.7 登録	: データベースにデータを登録する方法を学ぶ。
2.8 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。
2.9 例外処理	: 例外発生時の対処方法を学ぶ。
2.10 アクセス制御	: 特定のパスへアクセス制御する方法を学ぶ

■　画面遷移図

本章で作成するアプリケーションの画面遷移図は下図のようになる。ただし、例外発生時の画面遷移は除いている。

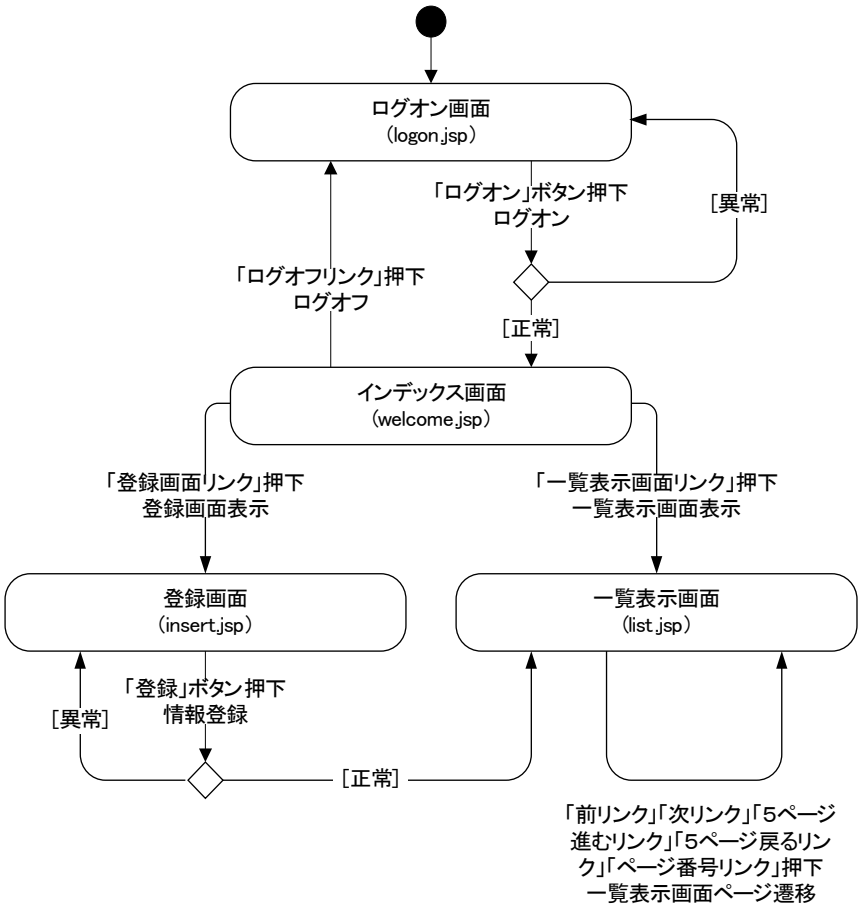


図6　画面遷移図

■ 各節の概要図

本書のチュートリアルでは、節ごとに実装する範囲の概要図を載せている。図は以下の凡例と記述説明に沿って記述している。

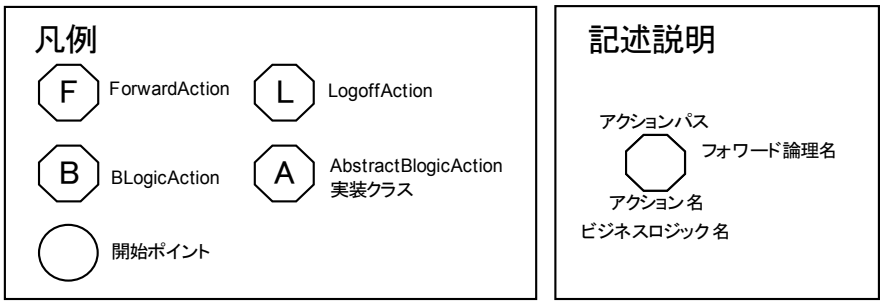


図7 各節概要図凡例

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：J2SDK1.5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.x (x はバージョン番号)
- データベース：HSQLDB 1.8.0.7
- ビルドツール：Apache Ant 1.6.5 以上
- 総合開発環境：Eclipse SDK 3.2.x
- Eclipse plugin ：WTP 1.5.x

■ 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。

- J2SDK 1.5.0_0x (x はバージョン番号)
- Tomcat 5.5.x (x はバージョン番号)
- HSQLDB 1.8.0.7
- Eclipse SDK 3.2.x
- WTP 1.5.x

(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3. インストールディレクトリ

J2SDK 1.5.0_0x (x はバージョン番号)	C:\Program Files\java\jdk1.5.0_0x (x はバージョン番号)
Eclipse 3.2.x+プラグイン	C:\Eclipse
Tomcat 5.5.x (x はバージョン番号)	C:\Program Files\Apache Software Foundation\Tomcat 5.5
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 Spring 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-spring-thin-blank.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-thin-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-thin-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。

下図にインポートした"terasoluna-spring-thin-blank”のプロジェクト構成を示す。

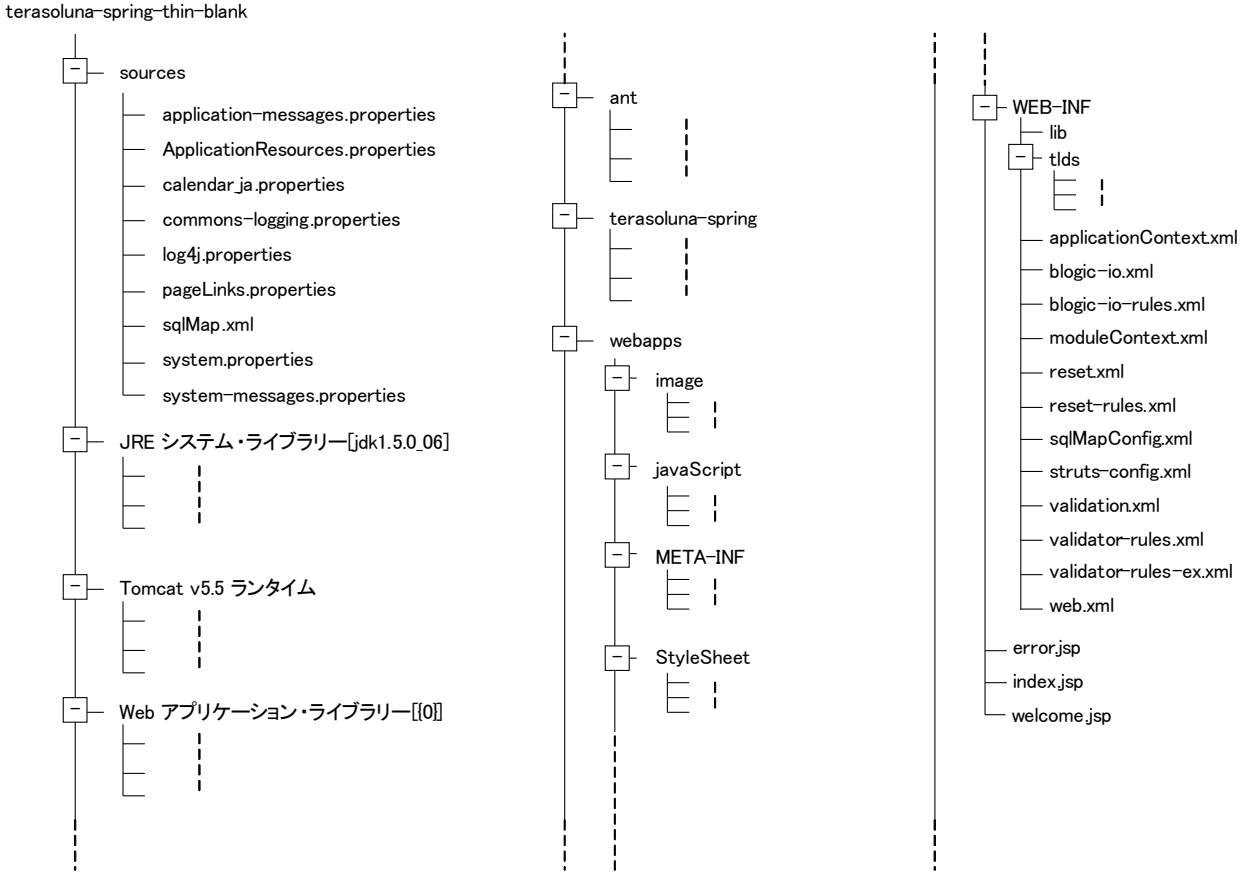


図8 プロジェクト構成

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4. 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。
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。データソース、トランザクション、DAO、iBATIS の設定を行う。
blogic-io.xml	TERASOLUNA の入出力設定ファイル。アクションフォームとビジネスロジック間の値の入出力を設定する。
moduleContext.xml	Spring 設定ファイル。モジュール固有の Bean 定義ファイル。アクション、ビジネスロジックの設定を行う。
struts-config.xml	Struts 設定ファイル。アクションフォームやアクションパスの設定を行う。
validation.xml	バリデーション設定ファイル。
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。

(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-thin”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“USERLIST”、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　画面遷移

本節では、Web ブラウザ対応版が提供するフォワード機能を利用し、画面遷移を行う方法について学習する。

■ **概要**

下図のようにログオン画面(初期表示用画面)を表示する機能を実装する。Spring 版では、原則としてブラウザから拡張子が“jsp”である URI へのアドレス直接指定によるアクセスを禁止している。そのため、画面遷移にはフォワード機能を利用する。

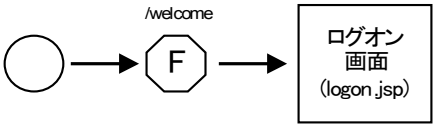


図9　画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
2. ログオン画面へ遷移する。

このとき、以下の作業が必要となる。

- JSP ファイルの作成
- “struts-config.xml”ファイルの変更
- “moduleContext.xml”ファイルの変更
- エンコーディングフィルタの作成
- “web.xml”ファイルの設定

■ **手順**

● **JSP ファイルの作成**

① デフォルトページの設定は“web.xml”に記述されている。ここでは“terasoluna-spring-thin-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。

```
<?xml version="1.0" encoding="UTF-8" ?>
                                          :
                                         略
                                          :
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
                                          :
                                         略
                                          :
</web-app>
```

② デフォルトページである index.jsp はブランクプロジェクトに作成済みである。ここでは“terasoluna-spring-thin-blank/webapps/index.jsp”ファイルが以下のようになっていることを確認する。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>-</title>
</head>
<body>

<logic:redirect forward="welcome"/>

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

③ 本節で表示させるログオン画面の logon.jsp を作成する。
“terasoluna-spring-thin-blank/webapps/logon.jsp”ファイルを以下のように作成する。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>ログオン画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                ログオン画面
            </h2>
        </td>
    </tr>
</table>
</body>

</html:html>
```

- **“struts-config.xml”ファイルの設定**

struts-config.xml にページ遷移の設定を行う。
“terasoluna-spring-thin-blank/webapps/WEB-INF/struts-config.xml”ファイルを以下のように変更する。(網掛け部分を変更する)。

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

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>

<!-- ========================================== Form Bean Definitions -->
  <form-beans>

  </form-beans>

<!-- ========================================== Global Exception Definitions -->


<!-- ========================================== Global Forward Definitions -->
  <global-forwards>
    <forward name="welcome"
             path="/welcome.do" redirect="true"/>
  </global-forwards>

<!-- ========================================== Global Forward Definitions -->
  <action-mappings type="jp.terasoluna.fw.web.struts.action.ActionMappingEx">

    <action path="/welcome" parameter="/logon.jsp"/>

  </action-mappings>

<!-- ========================================== Controller Definitions -->
  <controller processorClass="jp.terasoluna.fw.web.struts.action.RequestProcessorEx"/>

<!-- ========================================== PlugIn Definitions -->
                                          :
                                         略
                                          :

</struts-config>
```

- **“moduleContext.xml”ファイルの作成**

画面遷移のアクションを定義する“moduleContext.xml”は以下のように記述する。moduleContext.xml で設定するアクション定義の“bean name”は struts‐config.xml で設定した“action‐path”と一致させる必要がある。ここでは“/welcome”で一致させている。なお、ブランクプロジェクトでは今回の設定が作成済みのため、“terasoluna-spring-thin-blank/webapps/WEB-INF/moduleContext.xml”ファイルが以下のように設定されていることを確認する。

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

<!-- モジュール固有の Bean 定義 -->
<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/sp
ring-util-2.0.xsd">

    <!-- 共通定義のインポート -->
    <import resource="commonContext.xml"/>

    <!-- アクション定義 -->
    <bean name="/welcome" scope="prototype"
          class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

</beans>
```

● エンコーディングフィルタの作成

エンコーディングフィルタを作成、設定することにより日本語の文字化けを防ぐことが出来るので、この段階で作成しておく。
リクエストパラメータのエンコーディング設定を行うためには以下の作業が必要となる。

- エンコーディングフィルタクラスの作成
- “web.xml”の変更

“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web/SetCharacterEncodingFilter.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.thin.tutorial.web;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

/**
 * リクエストパラメータのエンコーディング設定を行う。
 * <p/>
 * ServletAPI バージョン 2.3 以上において有効。<br>
 */
public class SetCharacterEncodingFilter implements Filter {

    /**
     * エンコーディング。
     */
    protected String encoding = null;

    /**
     * フィルターコンフィグ。
     */
    protected FilterConfig filterConfig = null;

    /**
     * クライアントで指定されたエンコーディングを無視するかどうかのフラグ。
     */
    protected boolean ignore = true;

    /**
     * 終了時処理。
     */
    public void destroy() {
        this.encoding = null;
        this.filterConfig = null;
    }

    /**
     * フィルター処理。
     * リクエストパラメータを指定されたエンコーディングに設定する。
     *
     * @param request リクエスト
     * @param response レスポンス
     * @param chain フィルターチェーン
     * @exception IOException IO 例外
     * @exception ServletException 例外
     */
    public void doFilter(ServletRequest request, ServletResponse response,
```

```
        FilterChain chain) throws IOException, ServletException {

            if (ignore || (request.getCharacterEncoding() == null)) {
            String encoding = selectEncoding(request);
            if (encoding != null) {
                request.setCharacterEncoding(encoding);
            }
        }
        chain.doFilter(request, response);
    }

    /**
     * 初期化処理。
     *
     * @param filterConfig フィルター設定
     * @exception ServletException 例外
     */
    public void init(FilterConfig filterConfig) throws ServletException {

        this.filterConfig = filterConfig;
        this.encoding = filterConfig.getInitParameter("encoding");

        String value = filterConfig.getInitParameter("ignore");
        if (value == null) {
            this.ignore = true;
        } else if (value.equalsIgnoreCase("true")) {
            this.ignore = true;
        } else if (value.equalsIgnoreCase("yes")) {
            this.ignore = true;
        } else {
            this.ignore = false;
        }
    }
    /**
     * エンコーディング選択。
     *
     * @param request リクエスト
     * @return エンコーディング
     */
    protected String selectEncoding(ServletRequest request) {
        return this.encoding;
    }

}
```

“terasoluna-spring-thin-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/we
b-app_2_4.xsd"
    version="2.4">

<!-- listener setting -->

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>

<!-- filter setting -->

  <filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>
      jp.terasoluna.thin.tutorial.web.SetCharacterEncodingFilter
    </filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>Windows-31J</param-value>
    </init-param>
  </filter>

<!-- filter mapping -->

 <filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
                                                    :
                                                   略
                                                    :

</web-app>
```

■ 確認

実際にログオン画面が表示出来ることを確認する。

(1) 作成・更新したファイルを全て保存し、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。

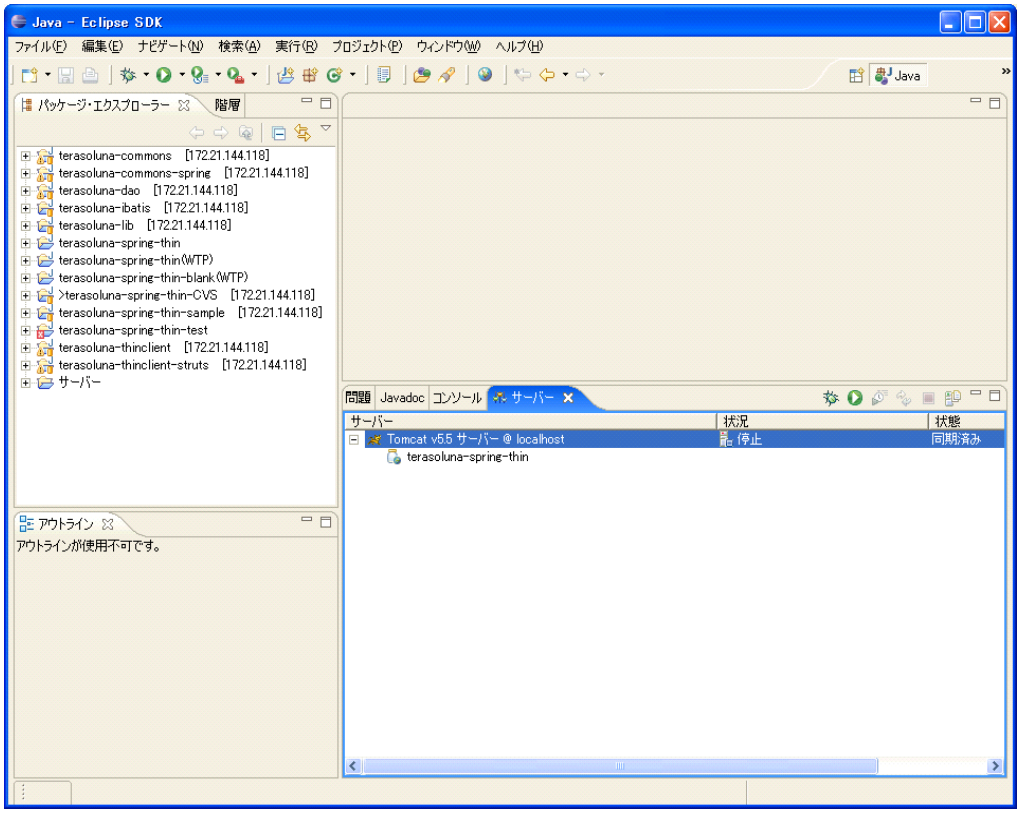


図10 Eclipse サーバ起動ボタン

※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) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
(3) 下図のように、ログオン画面が表示されることを確認する。下図のように表示されない場合はもう一度設定方法を確認する。

図11 ログオン画面表

■ **参考資料**

- Spring 版機能説明書
  『WE-03 フォーワード機能』

2.4　ログオン

本節では、Web ブラウザ対応版が提供する「ビジネスロジック実行機能」、「ビジネスロジック入出力機能」、「ユーザ情報保持機能」、「アクションフォーム拡張機能」、「ログオフ機能」を利用し、ログオン、ログオフ処理を行う方法に関して学習する。

■　概要

下図のようにログオン、ログオフ行う方法について学習する。ログオン画面では UserName を入力し、ログオン処理を行う。LogonBLogic を実行した結果、ログオンに成功すれば、フォワードアクションでインデックス画面に遷移する機能を実装する。

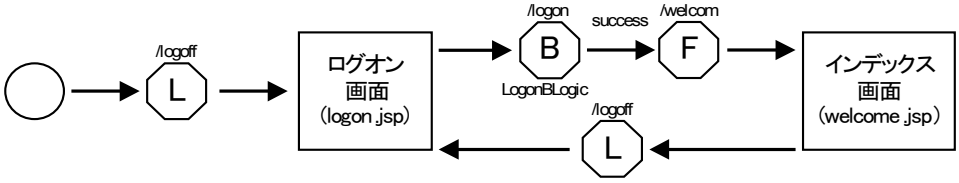


図12　ログオン画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスし、ログイン画面に遷移する。
2. UserName を入力し、ログオンボタンを押下する。
3. インデックス画面が表示され、UserName が表示される。

このとき、以下の作業が必要となる。

- JSP ファイルの変更
- アクションフォームの作成
- ユーザバリューオブジェクトクラスの作成
- “ApplicationResources.properties”の変更
- “blogic-io”ファイルの変更
- ビジネスロジック実装クラスの作成
- “struts-config.xml”ファイルの変更
- “moduleContext.xml”の変更
- “application-messages.properties” の変更

■　手順

- **JSP ファイルの変更**

① ログオン画面に「UserName」を入力するテキストボックスと、「ログオン」ボタンを作成する。
“terasoluna-spring-thin-blank/webapps/logon.jsp”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)。
なお、ここでは TERASOLUNA - Spring 独自の<ts:>タグを使用している。詳細については Spring 版機能説明書『WJ-01～WK-06 画面表示機能』参照のこと。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>ログオン画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                ログオン画面
            </h2>
        </td>
    </tr>
</table>

<p>

<ts:form action="/logon">
<table border="0" align="center">
  <tr>
    <td>
      UserName :
    </td>
    <td>
      <html:text property="userId"/>
    </td>
    <td>
      <html:submit value="ログオン"/>
    </td>
  </tr>
</table>
</ts:form>

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

② ログオン後表示されるインデックス画面を作成する。
“terasoluna-spring-thin-blank/webapps/welcome.jsp”ファイルを以下のように変更する。
※初期状態では、ブランクプロジェクト適用の確認用ページなので、内容を全て変更する。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>インデックス画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                インデックス画面
            </h2>
            <html:messages id="message" message="true">
              <bean:write name="message"/>
            </html:messages>
        </td>
    </tr>
</table>

<p>

<table border="0">
  <tr>
    <td>
      <html:link action="/logoff.do" >
        ログオフ
      </html:link>
    </td>
  </tr>

</table>

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

- **アクションフォームクラスの作成**

静的なアクションフォームである ValidatorActionFormEx を拡張してログオン処理で扱う「userId」を定義する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web.form/LogonForm.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.thin.tutorial.web.form;

import jp.terasoluna.fw.web.struts.form.ValidatorActionFormEx;

/**
 * ログオン時に使用するアクションフォーム。
 *
 */
public class LogonForm extends ValidatorActionFormEx {

    /**
     * ユーザ ID。
     */
    private String userId = null;

    /**
     * ユーザ ID を返却する。
     *
     * @return ユーザ ID
     */
    public String getUserId() {
        return userId;
    }

    /**
     * ユーザ ID を設定する。
     *
     * @param userId ユーザ ID
     */
    public void setUserId(String userId) {
        this.userId = userId;
    }

}
```

- **ユーザバリューオブジェクトクラスの作成**

ログオン中のユーザ情報を保持するユーザバリューオブジェクトクラスを作成する。ここでは「userId」を保持する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web/SampleUVO.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.thin.tutorial.web;

import jp.terasoluna.fw.web.UserValueObject;

/**
 * ユーザーバリューオブジェクト実装クラス。
 *
 */
public class SampleUVO extends UserValueObject {

    /**
     * ユーザ ID。
     */
    private String userId = null;

    /**
     * ユーザ ID を返却する。
     *
     * @return 保持するユーザ ID
     */
    public String getUserId() {
        return userId;
    }

    /**
     * ユーザ ID を設定する。
     *
     * @param userId ユーザ ID
     */
    public void setUserId(String userId) {
        this.userId = userId;
    }

}
```

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

ユーティリティメソッドを使用してユーザ情報取得クラスのインスタンスを作成するために実装クラス名をプロパティファイルに記述する。ここでは“terasoluna-spring-thin-blank/sources/ApplicationResources.properties”ファイルに以下の内容を追記する。

```
#UVO
user.value.object=jp.terasoluna.thin.tutorial.web.SampleUVO
```

- **“blogic-io.xml”ファイルの変更**

アクションフォームとビジネスロジック間の値の入出力を定義する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/blogic-io.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" ?>

<!DOCTYPE blogic-io PUBLIC "-//NTTDATA//DTD TERASOLUNA for Spring blogic-io 1.0//JA"
                           "dtd/blogic-io.dtd">


<blogic-io>

   <action path="/logon">
     <blogic-params bean-name="java.util.HashMap">
       <set-property property="userId"
                     source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="USER_VALUE_OBJECT" blogic-property="uvo"
                     dest="session" />
     </blogic-result>
   </action>

</blogic-io>
```

● ビジネスロジック実装クラスの作成

ログオン処理を行うビジネスロジッククラスを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.blogic/LogonBLogic.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.thin.tutorial.service.blogic;

import java.util.HashMap;
import java.util.Map;
import jp.terasoluna.fw.service.thin.BLogic;
import jp.terasoluna.fw.service.thin.BLogicMessage;
import jp.terasoluna.fw.service.thin.BLogicMessages;
import jp.terasoluna.fw.service.thin.BLogicResult;
import jp.terasoluna.fw.web.UserValueObject;
import jp.terasoluna.thin.tutorial.web.SampleUVO;

/**
 * ログオン処理を行うビジネスロジック。
 */
public class LogonBLogic implements BLogic<Map<String, String>> {

    /**
     * ログオン処理を行う。
     *
     * @param params 入力されたユーザIDを保持したMap
     * @return ログオン処理結果を保持したBLogicResult
     */
    public BLogicResult execute(Map<String, String> params) {

        //入力されたユーザIDの取得
        String userId = params.get("userId");

        //UVOの生成
        SampleUVO uvo = (SampleUVO) UserValueObject.createUserValueObject();

        //ユーザIDをUVOに設定する。
        uvo.setUserId(userId);

        //UVOを返却用のMapに保持させる。
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("uvo", uvo);

        //BLogicResultの生成、設定
        BLogicResult result = new BLogicResult();
        result.setResultObject(map);
        result.setResultString("success");

        //表示するメッセージの生成、設定
        BLogicMessages messages = new BLogicMessages();
        messages.add("logon.message",
                new BLogicMessage("logon.message", userId));
        result.setMessages(messages);

        return result;
    }
}
```

- **“struts-config.xml”ファイルの変更**

ログオン画面で用いるアクションフォームとアクションパスを定義する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/struts-config.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" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>

<!-- ========================================== Form Bean Definitions -->
  <form-beans>
    <!-- ログオン用 -->
     <form-bean name="logonForm"
                type="jp.terasoluna.thin.tutorial.web.form.LogonForm" />

  </form-beans>


<!-- ========================================== Global Exception Definitions -->
  <global-exceptions>

  </global-exceptions>


<!-- ========================================== Global Forward Definitions -->
  <global-forwards>
    <forward name="welcome"
             path="/logoff.do" redirect="true"/>
  </global-forwards>


<!-- ========================================== Global Forward Definitions -->
  <action-mappings type="jp.terasoluna.fw.web.struts.action.ActionMappingEx">

    <action path="/welcome"
            parameter="/welcome.jsp"/>

    <action path="/logon"
            name="logonForm" scope="request">
        <forward name="success" path="/welcome.do"/>
    </action>

    <action path="/logoff"
            parameter="/logon.jsp"/>

  </action-mappings>

<!-- ========================================== Controller Definitions -->

                                          :
                                         略
                                          :

</struts-config>
```

- **“moduleContext.xml”ファイルの変更**

アクション定義とビジネスロジックを定義し、オブジェクト間の依存関係を設定する。ここでは、アクション定義に Spring 版が提供する BLogicAction を設定する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/moduleContext.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" ?>

<!-- モジュール固有の Bean 定義 -->
<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/sp
ring-util-2.0.xsd">

<!-- 共通定義のインポート -->
<import resource="/commonContext.xml"/>

<!-- アクション定義 -->
<bean name="/welcome" scope="prototype"
        class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

<bean name="/logon"
        class="jp.terasoluna.fw.web.struts.actions.BLogicAction"
        scope="prototype">
    <property name="businessLogic">
      <ref bean="logonBLogic"/>
    </property>
  </bean>

  <bean name="/logoff" scope="prototype"
        class="jp.terasoluna.fw.web.struts.actions.LogoffAction" />

<!-- ビジネスロジック定義 -->

<bean id="logonBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.LogonBLogic" />

</beans>
```

- **“application-messages.properties” の変更**

画面に出力するメッセージを“application-messages.properties”に設定する。
“terasoluna-spring-thin-blank/sources/application-messages.properties”ファイルに以下を追記する。

```
logon.message=ようこそ{0}さん！
logon.userId=UserName
```

■ 確認

UserName に名前を入力し、インデックス画面で入力した名前が表示されることを確認する。

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

(2) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスし、
ログオン画面が表示されたら、下図のように UserName を入力し「ログオン」ボタンを押下する。

図13 ログオン画面(UserName 入力)

(3) ログオンボタン押下後、下図のようにインデックス画面が表示されることを確認する。

図14 インデックス画面(UserName 表示)

■ 参考資料

- Spring 版機能説明書
  『WH-01 ビジネスロジック実行機能』
  『WH-02 ビジネスロジック入出力機能』
  『WB-01 ユーザ情報保持機能』
  『WB-02 アクションフォーム拡張機能』
  『WE-07 ログオフ機能』

2.5 一覧表示

本節では Web ブラウザ対応版を利用し、TERASOLUNA が提供する一覧表示機能、リンク機能を用いて一覧表示を行う方法について学習する。なお、本節ではデータベースは使用しない。データベースを用いたデータ取得については次節で説明する。

■ 概要

下図のように一覧を表示させるプログラムを実装する。

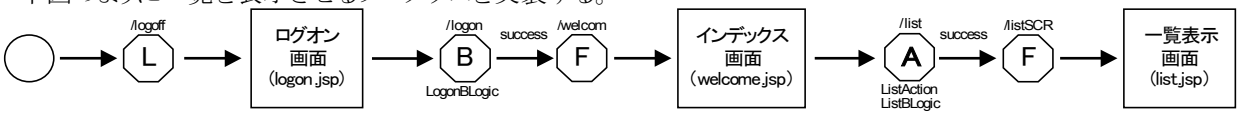


図15 一覧表示画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
2. ログオン画面へ遷移する。
3. UserName を入力し、ログオンボタンを押下してインデックス画面へ遷移する。
4. 一覧表示リンクを押下して一覧表示画面へ遷移する。
5. 次リンクを押下し、ページリンク機能を確認する。

このとき、以下の作業が必要となる。

- JSP ファイルの作成・変更
- JavaBean の作成
- ビジネスロジックのインタフェース作成
- ビジネスロジックの作成
- アクションの作成
- “blogic-io.xml”ファイルの変更
- “struts-config.xml”ファイルの変更
- “QueryDAO”の Stub の作成
- “moduleContext.xml”ファイルの変更
- “applicationContext.xml”ファイルの変更
- “pageLinks.properties”ファイルの作成
- “ApplicationResources.properties”ファイルの変更

■ 手順

● JSP ファイルの作成・変更

① インデックス画面に一覧表示画面へのリンクを作成する。
“terasoluna-spring-thin-blank/webapps/welcome.jsp”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)。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>インデックス画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                インデックス画面
            </h2>
            <html:messages id="message" message="true">
              <bean:write name="message"/>
            </html:messages>
        </td>
    </tr>
</table>

<p>

<table border="0">
  <tr>
    <td>
      <html:link page="/list.do">
        一覧表示画面
      </html:link>
    </td>
  </tr>
  <tr>
    <td>
      <html:link action="/logoff.do" >
        ログオフ
      </html:link>
    </td>
  </tr>

</table>

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

② 一覧表示画面を作成する。“terasoluna-spring-thin-blank/webapps/list.jsp”ファイルを以下のように作成する。ここで row は 1 ページ表示件数、startIndex は表示開始インデックス、totalCount は一覧情報全件数をそれぞれ意味している。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>一覧表示画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
```

```
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                一覧表示画面
            </h2>
        </td>
    </tr>
</table>

<html:messages id="message" message="true">
    <bean:write name="message"/>
</html:messages>

<ts:pageLinks id="userListPageLinks"
              action="/list" name="dynaFormBean" rowProperty="row"
              totalProperty="totalCount" indexProperty="startIndex"
              currentPageIndex="now" totalPageCount="total"/>

<center>

<h5>

<bean:write name="now"/>／<bean:write name="total"/> (<bean:write name="dynaFormBean" pro
perty="totalCount"/>)


<bean:write name="userListPageLinks" filter="false"/>

<table border="1" frame="box">
    <tr>
        <td width="30"  align="center"><b>ID</b></td>
        <td width="100" align="center"><b>NAME</b></td>
        <td width="30"  align="center"><b>AGE</b></td>
        <td width="80"  align="center"><b>BIRTH</b></td>
    </tr>
    <logic:iterate id="userBean" name="userBeans" scope="session">
    <tr>
        <td align="center">
          <bean:write name="userBean" property="id"/>
        </td>
        <td>
          <bean:write name="userBean" property="name"/>
        </td>
        <td align="center">
          <bean:write name="userBean" property="age"/>
        </td>
        <td align="right">
          <bean:write name="userBean" property="birth"/>
        </td>
    </tr>
    </logic:iterate>
</table>


<bean:write name="userListPageLinks" filter="false"/>

</h5>

</center>

<hr>

<ts:link page="/welcome.do">インデックス</ts:link>

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

- **JavaBean の作成**

値を保持する JavaBean を作成する。項目としては「ユーザ ID」、「ユーザ名」、「年齢」、「生年月日」を作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.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.thin.tutorial.service.bean;

import java.io.Serializable;

/**
 * ユーザ情報保持 Bean。
 *
 */
public class UserBean implements Serializable {

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

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

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

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

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

    /**
     * ユーザ ID を設定する。
     *
     * @param id ユーザ ID
     */
    public void setId(Long 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 String getAge() {
        return age;
    }

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

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

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

● ビジネスロジックのインタフェース作成

一覧情報取得ビジネスロジックのインタフェースを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.blogic/ListBLogic.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.thin.tutorial.service.blogic;

import java.util.Map;

/**
 * 一覧情報取得ビジネスロジックのインタフェース。
 *
 */
public interface ListBLogic {

    /**
     *
     * @param map 一覧情報取得に必要な値を保持したMap
     * @return 結果を保持したMap
     */
    public Map<String, Object> getUserList(Map<String, String> map);

}
```

● ビジネスロジックの作成

一覧情報取得ビジネスロジックを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.blogic/ListBLogicImpl.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.thin.tutorial.service.blogic;

import java.util.HashMap;
import java.util.Map;
import jp.terasoluna.fw.dao.QueryDAO;
import jp.terasoluna.thin.tutorial.service.bean.UserBean;

/**
 * 一覧情報取得ビジネスロジック。
 *
 * DAOのStubから画面に表示する件数のみ一覧情報を取得する。
 * また、一覧情報の全件数も取得する。
 *
 */
public class ListBLogicImpl implements ListBLogic{

    /**
     * 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 dao = null;
    /**
     * QueryDAOを返却する。
     *
```

```
     * @return 保持するQueryDAO
     */
    public QueryDAO getDao() {
        return dao;
    }

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

    /**
     * DAOのStubから一覧情報を取得する。
     * 引数から1ページの表示件数と開始インデックスを取得し、QueryDAOに渡す。
     * また、一覧情報の全件数も取得し返却する。
     *
     * @param map 表示件数、開始インデックスを保持したMap
     * @return 一覧情報、全件数を保持したMap
     */
    public Map<String, Object> getUserList(Map<String, String> map) {

        String strIndex = map.get("startIndex");
        String strRow = map.get("row");

        int startIndex = 0;
        int row = 10;
        if (strIndex != null) {
            try {
                startIndex = Integer.parseInt(strIndex);
            } catch (NumberFormatException e) {
                startIndex = 0;
            }
        }
        if (strRow != null) {
            try {
                row = Integer.parseInt(strRow);
            } catch (NumberFormatException e) {
                row = 10;
            }
        }

        UserBean[] bean = dao.executeForObjectArray("getUserList", null,
                UserBean.class, startIndex, row);

        Map<String, Object> retMap = new HashMap<String, Object>();
        retMap.put("totalCount", getUserCount());
        retMap.put("userBeans", bean);

        return retMap;
    }

    /**
     * 一覧情報の全件数を取得して返却する。
     *
     * @return 取得した件数文字列
     */
    public String getUserCount() {

         String count
            = dao.executeForObject("getUserCount", null, String.class);
        if (count == null || "".equals(count)) {
            count = "0";
        }
        return count;
    }
}
```

- **アクションの作成**

一覧情報取得ビジネスロジックを呼び、一覧情報を取得する一覧情報取得アクションを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web.action/ListAction.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.thin.tutorial.web.action;

import java.util.Map;
import jp.terasoluna.fw.service.thin.BLogicResult;
import jp.terasoluna.fw.web.struts.actions.AbstractBLogicAction;
import jp.terasoluna.thin.tutorial.service.blogic.ListBLogic;

/**
 * 一覧情報取得アクション。
 *
 * 一覧情報取得ビジネスロジックを呼び、一覧情報を取得する。
 *
 */
public class ListAction extends AbstractBLogicAction<Map<String, String>> {

    /**
     * 実行するビジネスロジッククラス。
     */
    private ListBLogic listBLogic = null;

    /**
     * ビジネスロジックを返却する。
     *
     * @return 保持するビジネスロジック
     */
    public ListBLogic getListBLogic() {
        return listBLogic;
    }

    /**
     * ビジネスロジックを設定する。
     *
     * @param listBLogic ビジネスロジック
     */
    public void setListBLogic(ListBLogic listBLogic) {
        this.listBLogic = listBLogic;
    }

    /**
     * ビジネスロジックを実行し、一覧情報を取得、返却する。
     *
     * @param map 一覧情報取得のための変数を保持したMap
     * @return 一覧情報を保持したBLogicResult
     * @throws Exception 例外
     */
    @Override
    public BLogicResult doExecuteBLogic(Map<String, String> map)
        throws Exception {

        //ビジネスロジックの実行、結果の取得
        Map<String, Object> retMap = listBLogic.getUserList(map);

        //BLogicResultの生成、結果の設定
        BLogicResult result = new BLogicResult();
        result.setResultString("success");
        result.setResultObject(retMap);

        return result;
    }
}
```

- **“blogic-io.xml”ファイルの変更**

アクションフォームとビジネスロジック間の値の入出力を定義する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/blogic-io.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" ?>

<!DOCTYPE blogic-io PUBLIC "-//NTTDATA//DTD TERASOLUNA for Spring blogic-io 1.0//JA"
                           "dtd/blogic-io.dtd">

<blogic-io>

   <action path="/logon">
     <blogic-params bean-name="java.util.HashMap">
       <set-property property="userId"
                     source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="USER_VALUE_OBJECT" blogic-property="uvo"
                     dest="session" />
     </blogic-result>
   </action>

   <action path="/list">
     <blogic-params bean-name="java.util.HashMap">
       <set-property property="startIndex" source="form" />
       <set-property property="row"        source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="userBeans"  dest="session" />
       <set-property property="totalCount" dest="form" />
     </blogic-result>
   </action>

</blogic-io>
```

- **“struts-config.xml”ファイルの変更**

一覧表示で使用するアクションフォームの設定を行う。
“terasoluna-spring-thin-blank/webapps/WEB-INF/struts-config.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" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>

<!-- ========================================== Form Bean Definitions -->
  <form-beans>

    <!-- ログオン用 -->
    <form-bean name="logonForm"
               type="jp.terasoluna.thin.tutorial.web.form.LogonForm" />

    <form-bean name="dynaFormBean"
        type="jp.terasoluna.fw.web.struts.form.DynaValidatorActionFormEx" >

      <!-- 一覧表示用 -->
      <form-property name="row"        type="java.lang.String" initial="10"/>
      <form-property name="startIndex" type="java.lang.String" initial="0"/>
      <form-property name="totalCount" type="java.lang.String"/>

    </form-bean>

  </form-beans>
<!-- ========================================== Global Exception Definitions -->
  <global-exceptions>

  </global-exceptions>
<!-- ========================================== Global Forward Definitions -->
  <global-forwards>
    <forward name="welcome"
             path="/logoff.do" redirect="true"/>
  </global-forwards>
<!-- ========================================== Global Forward Definitions -->
  <action-mappings type="jp.terasoluna.fw.web.struts.action.ActionMappingEx">

    <action path="/welcome"
            parameter="/welcome.jsp"/>

    <action path="/logon"
            name="logonForm" scope="request">
        <forward name="success" path="/welcome.do"/>
    </action>

    <action path="/logoff"
            parameter="/logon.jsp"/>

    <action path="/list"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/listSCR.do"/>
    </action>

    <action path="/listSCR"
            name="dynaFormBean" scope="request"
            parameter="/list.jsp" />

  </action-mappings>
                                          :
                                        略
                                          :
</struts-config>
```

- **“QueryDAO”の Stub 作成**

ここではデータベースを使用せず、QueryDAO の Stub を作成し、値を取得する。

“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.stub/QueryDAOStub.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.thin.tutorial.service.stub;
import java.util.List;
import java.util.Map;
import jp.terasoluna.fw.dao.QueryDAO;
import jp.terasoluna.thin.tutorial.service.bean.UserBean;

/**
 * QueryDAO の Stub。
 *
 * 一覧表示の情報を取得する。
 */
public class QueryDAOStub implements QueryDAO {

    /**
     * totalCount 40 を返す
     */
    public <E> E executeForObject(String arg0, Object arg1, Class arg2) {
        return (E) "40";
    }

    /**
     * UserBean のテストデータを作成する。
     *
     * @param row
     *            作成する件数
     * @param startIndex
     *            開始番号
     * @return
     */
    public <E> E[] executeForObjectArray(String arg0, Object arg1, Class arg2,
            int startIndex, int row) {
        UserBean[] beans = new UserBean[row];
        for (int i = 0; i < row; i++) {
            UserBean userBean = new UserBean();
            userBean.setId(new Long(++startIndex));
            userBean.setName("user");
            userBean.setAge("xx");
            userBean.setBirth("xxxx/xx/xx");
            beans[i] = userBean;
        }
        E[] result = (E[]) beans;
        return result;
    }

    /**
     * 使用しない
     */
    public Map<String, Object> executeForMap(String arg0, Object arg1) {
        return null;
    }

    /**
     * 使用しない
     */
    public <E> E[] executeForObjectArray(String arg0, Object arg1, Class arg2) {
        return null;
    }
```

```
    /**
     * 使用しない
     */
    public Map<String, Object>[] executeForMapArray(String arg0, Object arg1) {
        return null;
    }

    /**
     * 使用しない
     */
    public Map<String, Object>[] executeForMapArray(String arg0, Object arg1,
            int arg2, int arg3) {
        return null;
    }

    /**
     * 使用しない
     */
    public List<Map<String, Object>> executeForMapList(String sqlID,
            Object bindParams) {
        return null;
    }

    /**
     * 使用しない
     */
    public List<Map<String, Object>> executeForMapList(String sqlID,
            Object bindParams, int beginIndex, int maxCount) {
        return null;
    }

    /**
     * 使用しない
     */
    public <E> List<E> executeForObjectList(String sqlID, Object bindParams) {
        return null;
    }

    /**
     * 使用しない
     */
    public <E> List<E> executeForObjectList(String sqlID, Object bindParams,
            int beginIndex, int maxCount) {
        return null;
    }
}
```

- **“moduleContext.xml”ファイルの変更**

一覧表示で使用するアクションとビジネスロジックを定義する。

“terasoluna-spring-thin-blank/webapps/WEB-INF/moduleContext.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" ?>

<!-- モジュール固有の Bean 定義 -->
<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/sp
ring-util-2.0.xsd">

<!-- 共通定義のインポート -->
  <import resource="/commonContext.xml"/>

<!-- アクション定義 -->
  <bean name="/welcome" scope="prototype"
```

```
        class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <bean name="/logon"
        class="jp.terasoluna.fw.web.struts.actions.BLogicAction"
        scope="prototype">
    <property name="businessLogic">
      <ref bean="logonBLogic"/>
    </property>
  </bean>

  <bean name="/logoff" scope="prototype"
        class="jp.terasoluna.fw.web.struts.actions.LogoffAction" />

  <bean name="/list" scope="prototype"
        class="jp.terasoluna.thin.tutorial.web.action.ListAction">
    <property name="listBLogic">
      <ref local="listBLogic"/>
    </property>
  </bean>

  <bean name="/listSCR" scope="prototype"
        class="org.apache.struts.actions.ForwardAction"/>

<!-- ビジネスロジック定義 -->
  <bean id="listBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.ListBLogicImpl">
    <property name="dao"><ref bean="queryDAOStub"/></property>
  </bean>

  <bean id="logonBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.LogonBLogic" />

</beans>
```

- **“applicationContext.xml”ファイルの変更**

アプリケーション全体の Bean 定義ファイルにテスト用に作成した QueryDAO の Stub を設定する。
“terasoluna-spring-thin-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" ?>

<!-- アプリケーション全体の Bean 定義 -->
<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/sp
ring-util-2.0.xsd">

<!-- DAO 定義 -->
  <bean id="queryDAOStub"
        class="jp.terasoluna.thin.tutorial.service.stub.QueryDAOStub">
  </bean>

</beans>
```

- **“pageLinks.properties”の変更**

出力するリンクのフォーマットを“pageLinks.properties”に設定する。
“terasoluna-spring-thin-blank/sources/pageLinks.properties”ファイルを以下のように変更する(網掛け部分を追加・変更する)。

```
pageLinks.maxDirectLinkCount=10
pageLinks.prev5.char=5ﾍﾟｰｼﾞ戻る
pageLinks.prev1.char=前
pageLinks.next1.char=次
pageLinks.next5.char=5ﾍﾟｰｼﾞ進む
```

- **“ApplicationResources.properties”の変更**

ページ遷移リンク機能で使用するプロパティファイルを読込む設定を追加する。
“terasoluna-spring-thin-blank/sources/ApplicationResources.properties”ファイルを以下のように変更する(網掛け部分を追加・変更する)。

```
#ページ遷移リンク機能にて使用するプロパティファイルを読み込む。
add.property.file.2=pageLinks.properties
```

■ **確認**

一覧表示画面に遷移し、ビジネスロジック内で作成した情報が一覧表示されることを確認する。
またページリンク機能が動作することも確認する。

(1) 作成・変更したファイルをすべて保存し、コンパイルを行ったのちに Tomcat を起動する。
(2) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスし、ログオン画面が表示されたら下図のように UserName を入力しログオンボタンを押下する。

図16 ログオン画面

(3) ログオンボタン押下後、下図のようにインデックス画面に遷移する。

図17 インデックス画面

(4) 一覧表示画面リンクを押下し、一覧表示画面へ遷移する。遷移後、データの一覧が表示されているか確認する。

図18　一覧表示画面

(5) 次リンクを押下すると画面が遷移することを確認し、ページリンク機能の動作確認をする。

図19　一覧表示画面(ページリンク)

■ 参考資料

- Spring 版機能説明書
  『WI-01 一覧表示機能』
  『WB-02 アクションフォーム拡張機能』

2.6　データベースアクセス

本節では Web ブラウザ対応版が提供するデータベースアクセス機能を利用し、データベースにアクセスして一覧表示とデータ削除を行う方法について学習する。

■ **概要**

下図のように一覧表示させ、データを削除出来るプログラムを実装する。

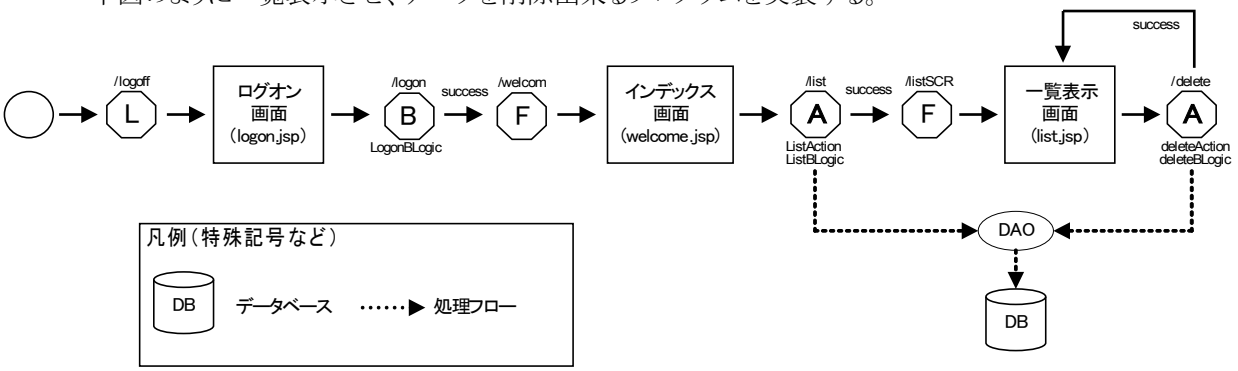


図20　データベースアクセス画面遷移

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
2. ログオン画面へ遷移する。
3. UserName を入力し、ログオンボタンを押下してインデックス画面へ遷移する。
4. インデックス画面から一覧表示リンクを押下して一覧表示画面へ遷移する。
5. 削除ボタンを押下し、データを削除する。

このとき、以下の作業が必要となる。

- JSP ファイルの変更
- ビジネスロジックのインタフェース作成
- ビジネスロジックの作成
- アクションの作成
- “blogic-io.xml”ファイルの変更
- “struts-config.xml”ファイルの変更
- “moduleContext.xml”の変更
- “applicationContext.xml”の変更
- “sqlMap.xml”の変更
- “jdbc.properties”ファイルの作成
- jar ファイルのビルドパスへの追加

■ **手順**

● **JSP ファイルの変更**

① 一覧画面に削除ボタンを追加する。
“terasoluna-spring-thin-blank/webapps/list.jsp”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)。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>一覧表示画面</title>
<script type="text/javascript">
<!--
function setDeleteSubmit(param) {
    document.forms[0].deleteId.value=param;
    document.forms[0].submit();
}
-->
</script>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                一覧表示画面
            </h2>
        </td>
    </tr>
</table>

<html:messages id="message" message="true">
    <bean:write name="message"/>
</html:messages>

<ts:pageLinks id="userListPageLinks"
              action="/list" name="dynaFormBean" rowProperty="row"
              totalProperty="totalCount" indexProperty="startIndex"
              currentPageIndex="now" totalPageCount="total"/>

<ts:form action="/delete">
<center>
<h5>

<bean:write name="now"/>／<bean:write name="total"/> (<bean:write name="dynaFormBean" pro
perty="totalCount"/>)

<p/>

<bean:write name="userListPageLinks" filter="false"/>

<p/>
<table border="1" frame="box">
    <tr>
        <td width="30"  align="center"><b>ID</b></td>
        <td width="100" align="center"><b>NAME</b></td>
        <td width="30"  align="center"><b>AGE</b></td>
        <td width="80"  align="center"><b>BIRTH</b></td>
        <td>-</td>
    </tr>
    <logic:iterate id="userBean" name="userBeans" scope="session">
    <tr>
        <td align="center">
          <bean:write name="userBean" property="id"/>
        </td>
        <td>
          <bean:write name="userBean" property="name"/>
        </td>
        <td align="center">
          <bean:write name="userBean" property="age"/>
        </td>
```

```
        <td align="right">
          <bean:write name="userBean" property="birth"/>
        </td>
        <td>
          <bean:define id="id" name="userBean" property="id"/>
      <input type="button" value="削除" onclick="setDeleteSubmit('<%=id%>');"/>
        </td>
    </tr>
    </logic:iterate>
</table>
<p/>
<input type="hidden" name="deleteId" value="" />

<bean:write name="userListPageLinks" filter="false"/>

</h5>

</center>

<hr>

<ts:link page="/welcome.do">インデックス</ts:link>

</ts:form>
</body>
</html:html>
```

- **ビジネスロジックのインタフェース作成**

① 情報削除ビジネスロジックのインターフェースを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.blogic/DeleteBLogic.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.thin.tutorial.service.blogic;

import jp.terasoluna.thin.tutorial.service.bean.UserBean;

/**
 * 情報削除ビジネスロジックのインタフェース。
 *
 */
public interface DeleteBLogic {

    /**
     * データベースから情報を削除する。
     * 登録に成功した件数を返却する。
     *
     * @param bean 削除に必要な値を保持する Bean
     * @return 登録結果件数
     */
    public int delete(UserBean bean);

}
```

- **ビジネスロジックの作成**

情報削除ビジネスロジックを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.blogic/DeleteBLogicImpl.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.thin.tutorial.service.blogic;

import jp.terasoluna.fw.dao.UpdateDAO;
import jp.terasoluna.thin.tutorial.service.bean.UserBean;

/**
 * 情報削除ビジネスロジック。
 *
 * データベースからユーザ情報を削除する。
 *
 */
public class DeleteBLogicImpl implements DeleteBLogic{

     /**
      * 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 dao = null;

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

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

     /**
      * データベースからユーザ情報を削除する。
      * 登録に成功した件数を返却する。
      *
      * @param bean 画面にて入力された値を保持する Bean
      * @return 登録結果件数
      */
     public int delete(UserBean bean) {

         return dao.execute("deleteUser", bean);
     }
}
```

- **アクションの作成**

情報削除ビジネスロジックを呼び、データベースの情報を削除する情報削除アクションを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web.action/DeleteAction.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.thin.tutorial.web.action;

import jp.terasoluna.fw.service.thin.BLogicMessage;
import jp.terasoluna.fw.service.thin.BLogicMessages;
import jp.terasoluna.fw.service.thin.BLogicResult;
import jp.terasoluna.fw.web.struts.actions.AbstractBLogicAction;
import jp.terasoluna.thin.tutorial.service.bean.UserBean;
import jp.terasoluna.thin.tutorial.service.blogic.DeleteBLogic;
import org.apache.struts.Globals;

/**
 * 情報削除アクション。
 *
 * 削除を行うビジネスロジックを呼び、データベースの情報を削除する。
 *
 */
public class DeleteAction extends AbstractBLogicAction<UserBean> {

    /**
     * 実行するビジネスロジッククラス。
     */
    private DeleteBLogic deleteBLogic = null;

    /**
     * ビジネスロジックを返却する。
     *
     * @return 保持するビジネスロジック
     */
    public DeleteBLogic getDeleteBLogic() {
        return deleteBLogic;
    }

    /**
     * ビジネスロジックを設定する。
     *
     * @param deleteBLogic ビジネスロジック
     */
    public void setDeleteBLogic(DeleteBLogic deleteBLogic) {
        this.deleteBLogic = deleteBLogic;
    }

    /**
     * ビジネスロジックを実行し、結果を返却する。
     *
     * @param bean 削除対象のユーザ ID の値を保持した bean
     * @return 登録結果を保持した BLogicResult
     * @throws Exception 例外
     */
    @Override
    public BLogicResult doExecuteBLogic(UserBean bean) throws Exception {

        //ビジネスロジックの実行、結果の取得
        int i = deleteBLogic.delete(bean);

        //BLogicResult の生成、結果の設定
        BLogicResult result = new BLogicResult();
        result.setResultString("success");
```

```
        //結果件数が 0 以下の場合は削除できなかったとして、
        //エラーメッセージを設定する。
        if (i <= 0) {
            BLogicMessages errors = result.getErrors();
            if (errors == null) {
                errors = new BLogicMessages();
            }
            errors.add(Globals.ERROR_KEY, new BLogicMessage("error.no.delete"));
            result.setErrors(errors);
        }

        return result;
    }

}
```

- **“blogic-io.xml”ファイルの変更**

アクションフォームとビジネスロジック間の値の入出力を定義する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/blogic-io.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" ?>

<!DOCTYPE blogic-io PUBLIC "-//NTTDATA//DTD TERASOLUNA for Spring blogic-io 1.0//JA"
                           "dtd/blogic-io.dtd">

<blogic-io>

   <action path="/logon">
     <blogic-params bean-name="java.util.HashMap">
       <set-property property="userId"
                     source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="USER_VALUE_OBJECT" blogic-property="uvo"
                     dest="session" />
     </blogic-result>
   </action>

  <action path="/list">
     <blogic-params bean-name="java.util.HashMap">
       <set-property property="startIndex" source="form" />
       <set-property property="row"        source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="userBeans"  dest="session" />
       <set-property property="totalCount" dest="form" />
     </blogic-result>
   </action>

   <action path="/delete">
     <blogic-params
       bean-name="jp.terasoluna.thin.tutorial.service.bean.UserBean">
       <set-property property="deleteId" blogic-property="id" source="form" />
     </blogic-params>
     <blogic-result>
     </blogic-result>
   </action>

</blogic-io>
```

- **“struts-config.xml”ファイルの変更**

削除用のアクションフォームとアクションパスを定義する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/struts-config.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" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>

<!-- ========================================== Form Bean Definitions -->
  <form-beans>

    <!-- ログオン用 -->
    <form-bean name="logonForm"
               type="jp.terasoluna.thin.tutorial.web.form.LogonForm" />
```

```
    <form-bean name="dynaFormBean"
        type="jp.terasoluna.fw.web.struts.form.DynaValidatorActionFormEx" >

      <!-- 一覧表示用 -->
      <form-property name="row"        type="java.lang.String" initial="10"/>
      <form-property name="startIndex" type="java.lang.String" initial="0"/>
      <form-property name="totalCount" type="java.lang.String"/>

      <!-- 削除用 -->
      <form-property name="deleteId"   type="java.lang.Long"/>
    </form-bean>

  </form-beans>

<!-- ========================================== Global Exception Definitions -->
  <global-exceptions>

  </global-exceptions>

<!-- ========================================== Global Forward Definitions -->
  <global-forwards>
    <forward name="welcome"
             path="/logoff.do" redirect="true"/>
  </global-forwards>

<!-- ========================================== Global Forward Definitions -->
  <action-mappings type="jp.terasoluna.fw.web.struts.action.ActionMappingEx">

    <action path="/welcome"
            parameter="/welcome.jsp"/>

    <action path="/logon"
            name="logonForm" scope="request">
        <forward name="success" path="/welcome.do"/>
    </action>

    <action path="/logoff"
            parameter="/logon.jsp"/>

    <action path="/list"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/listSCR.do"/>
    </action>

    <action path="/listSCR"
            name="dynaFormBean" scope="request"
            parameter="/list.jsp" />

    <action path="/delete"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/list.do"/>
    </action>

  </action-mappings>
                                          :
                                        略
                                          :

</struts-config>
```

- **“moduleContext.xml”ファイルの変更**

削除用のアクションとビジネスロジックを定義する。listBLogic に dao プロパティの設定を追加する。

“terasoluna-spring-thin-blank/webapps/WEB-INF/moduleContext.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" ?>
```

```
<!-- モジュール固有の Bean 定義 -->
<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/sp
ring-util-2.0.xsd">

<!-- 共通定義のインポート -->
  <import resource="/commonContext.xml"/>

<!-- アクション定義 -->
  <bean name="/welcome" scope="prototype"
        class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <bean name="/logon"
        class="jp.terasoluna.fw.web.struts.actions.BLogicAction"
        scope="prototype">
    <property name="businessLogic">
      <ref bean="logonBLogic"/>
    </property>
  </bean>

  <bean name="/logoff" scope="prototype"
        class="jp.terasoluna.fw.web.struts.actions.LogoffAction" />

  <bean name="/list" scope="prototype"
        class="jp.terasoluna.thin.tutorial.web.action.ListAction">
    <property name="listBLogic">
      <ref local="listBLogic"/>
    </property>
  </bean>

  <bean name="/listSCR" scope="prototype"
        class="org.apache.struts.actions.ForwardAction"/>

  <bean name="/delete" scope="prototype"
        class="jp.terasoluna.thin.tutorial.web.action.DeleteAction">
    <property name="deleteBLogic">
      <ref local="deleteBLogic"/>
    </property>
  </bean>

<!-- ビジネスロジック定義 -->

  <bean id="listBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.ListBLogicImpl">
  <property name="dao"><ref bean="queryDAOStub"/></property>
  </bean>

  <bean id="deleteBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.DeleteBLogicImpl">
    <property name="dao"><ref bean="updateDAO"/></property>
  </bean>

  <bean id="logonBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.LogonBLogic" />

</beans>
```

- **“applicationContext.xml”ファイルの変更**

データソース、トランザクション、DAO の定義を行う。“terasoluna-spring-thin-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)。なお、ここ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 Spring 版機能説明書『CC-01 JNDI アクセス機能』参照のこと。

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

<!-- アプリケーション全体の Bean 定義 -->
<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/sp
ring-util-2.0.xsd">

<!-- データソース設定 -->
  <bean id="propertyConfigurer"
 class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location">
      <value>/WEB-INF/jdbc.properties</value>
    </property>
  </bean>
  <bean id="TerasolunaDataSource"
        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>

<!-- トランザクション設定 -->
  <bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource">
      <ref bean="TerasolunaDataSource"/>
    </property>
  </bean>

  <bean id="attrSource"
      class="org.springframework.transaction.interceptor.NameMatchTransactionAttributeSo
urce">
    <property name="properties">
      <props>
        <prop key="*">PROPAGATION_REQUIRED,-java.lang.Exception</prop>
      </props>
    </property>
  </bean>

  <!-- トランザクション処理（インタセプタの定義） -->
  <bean id="transactionInterceptor"
      class="org.springframework.transaction.interceptor.TransactionInterceptor">
    <property name="transactionManager">
      <ref bean="transactionManager"/>
    </property>
    <property name="transactionAttributeSource">
      <ref local="attrSource"/>
    </property>
  </bean>

  <bean id="autoProxy"
      class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <!-- ここに指定したインタセプタを適用する -->
    <property name="interceptorNames">
      <list>
        <idref bean="transactionInterceptor"/>
      </list>
    </property>
    <!-- ここに指定した Bean 定義ファイルの ID に対して、インタセプタを適用する -->
    <property name="beanNames">
      <list>
        <value>*DAO</value>
      </list>
    </property>
  </bean>
```

```
<!-- iBATIS 定義 -->
  <bean id="sqlMapClient"
        class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
    <property name="configLocation" value="WEB-INF/sqlMapConfig.xml"/>
    <property name="dataSource"><ref bean="TerasolunaDataSource"/></property>
  </bean>

<!-- DAO 定義 -->

  <bean id="queryDAOStub"
        class="jp.terasoluna.thin.tutorial.service.stub.QueryDAOStub">
  </bean>

  <bean id="queryDAO"
        class="jp.terasoluna.fw.dao.ibatis.QueryDAOiBatisImpl">
    <property name="sqlMapClient"><ref local="sqlMapClient"/></property>
  </bean>

  <bean id="updateDAO"
        class="jp.terasoluna.fw.dao.ibatis.UpdateDAOiBatisImpl">
    <property name="sqlMapClient"><ref local="sqlMapClient"/></property>
  </bean>
</beans>
```

- **sqlMap.xml ファイルの変更**

実行する SQL 文を sqlMap.xml に設定する。

“terasoluna-spring-thin-blank/sources/sqlMap.xml”ファイルを以下ように変更する(網掛け部分を追加する)。

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

<sqlMap namespace="user">

    <select id="getUser"
            resultClass="java.util.HashMap">
        SELECT ID, NAME, AGE, BIRTH FROM USERLIST WHERE ID = #ID#
    </select>

    <select id="getUserCount"
            resultClass="java.lang.String"
            resultSetType="SCROLL_INSENSITIVE">
        SELECT
            COUNT(ID)
        FROM
            USERLIST
    </select>

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

    <delete id="deleteUser"
        parameterClass="jp.terasoluna.thin.tutorial.service.bean.UserBean">
        DELETE FROM USERLIST
        WHERE ID = #id#
    </delete>

</sqlMap>
```

- **“jdbc.properties”ファイルの作成**

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

```
#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=
```

- **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チュートリアルフォルダにある“\webapps\WEB-INF\lib”の中から、以下のファイルを“C:\Eclipse\workspace\terasoluna-spring-thin-blank\webapps\WEB-INF\lib\”フォルダにコピーする。
- commons-collections-3.2.jar
- commons-dbcp-1.2.2.jar
- commons-pool-1.3.jar
- hsqldb.jar

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“commons-collections-3.2.jar”、“commons-dbcp-1.2.2.jar”、“commons-pool-1.3.jar”、“hsqldb.jar”が含まれていることを確認する。

■ 確認

一覧表示画面に遷移し、データベースから値を取得して一覧表示されることを確認する。
また、削除ボタンを押下すると、データが削除されることも確認する。

(1) 作成・変更したファイルをすべて保存し、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) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスし、ログオン画面が表示されたら下図のように UserName を入力しログオンボタンを押下する。

図21 ログオン画面

(4) ログオンボタン押下後、下図のようにインデックス画面に遷移する。

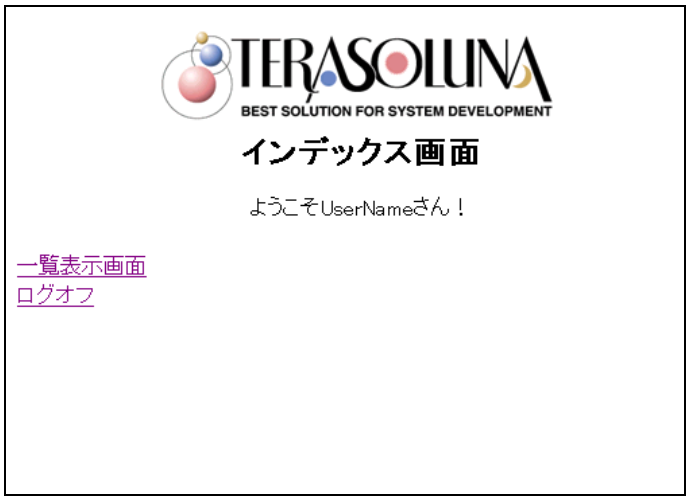


図22 インデックス画面

(5) 一覧表示画面リンクを押下し、一覧表示画面へ遷移する。遷移後、データベースから取得したデータの一覧が表示されているか確認する。

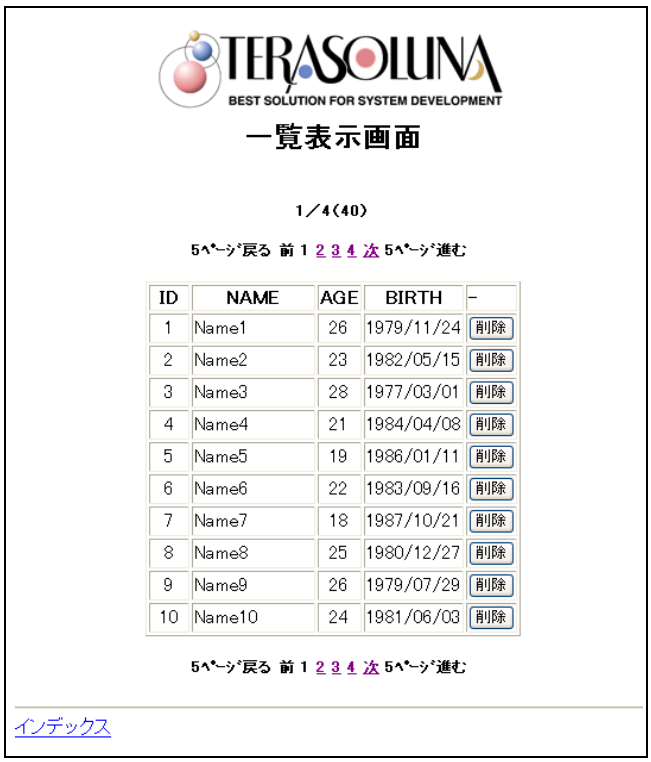


図23 一覧表示画面

(6) 削除ボタン押下後、データが削除されることを確認する。

図24 一覧表示画面(データ削除後)

■ 参考資料

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

2.7　登録

本節では Web ブラウザ対応版を利用し、データベースにアクセスして画面から入力した値をデータベースに登録する方法を学ぶ。

■ **概要**

下図のようにデータを入力する方法について学習する。

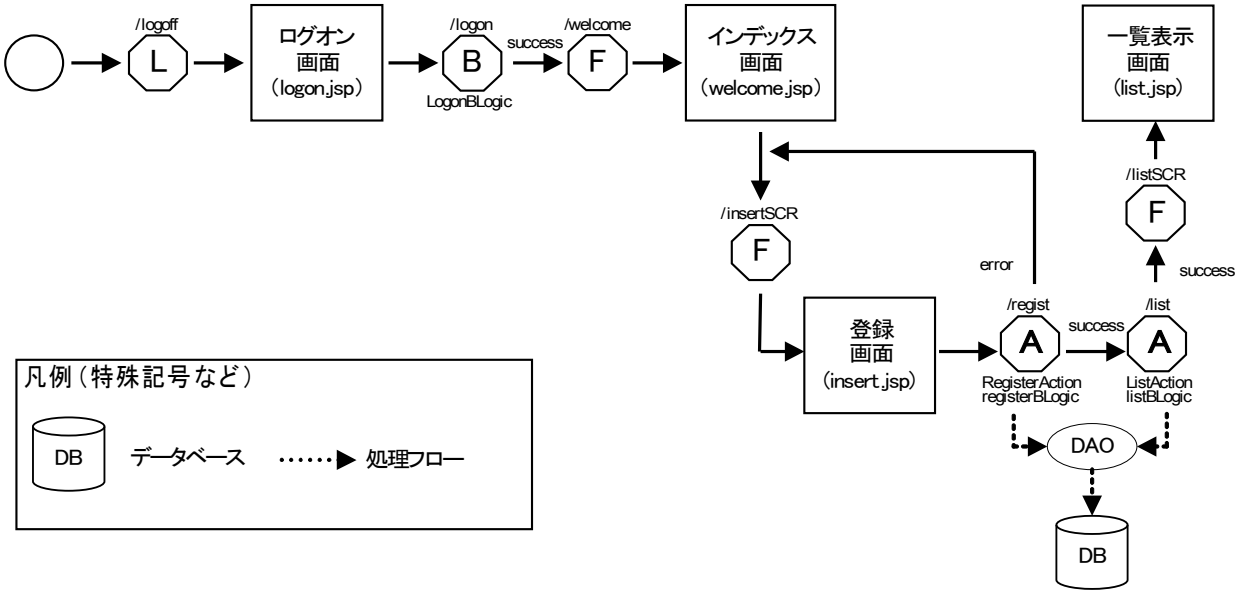


図25　登録画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
2. ログオン画面へ遷移する。
3. UserName を入力し、ログインボタンを押下いてインデックス画面へ遷移する。
4. インデックス画面から登録画面リンクを押下して登録画面へ遷移する。
5. 削除ボタンを押下し、データを削除する。

このとき、以下の作業が必要となる。

- JSP ファイルの作成・変更
- アクションフォームの作成
- ビジネスロジックのインタフェース作成
- ビジネスロジックの作成
- “blogic-io.xml”ファイルの変更
- “struts-config.xml”ファイルの変更
- “moduleContext.xml”の変更
- “calendar_ja.properties”ファイルの変更
- “sqlMap.xml”ファイルの変更
- “application-messages.properties”の変更

■ **手順**

● **JSP ファイルの作成・変更**

① インデックス画面に登録画面へのリンクを作成する。
“terasoluna-spring-thin-blank/webapps/welcome.jsp”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)。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>インデックス画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                インデックス画面
            </h2>
            <html:messages id="message" message="true">
              <bean:write name="message"/>
            </html:messages>
        </td>
    </tr>
</table>

<p>

<table border="0">
  <tr>
    <td>
      <html:link page="/list.do">
        一覧表示画面
      </html:link>
    </td>
  </tr>
   <tr>
    <td>
      <html:link page="/insertSCR.do">
        登録画面
      </html:link>
    </td>
  </tr>
  <tr>
    <td>
      <html:link action="/logoff.do" >
        ログオフ
      </html:link>
    </td>
  </tr>

</table>

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

② 一覧画面に登録画面へのリンクを作成する。
“terasoluna-spring-thin-blank/webapps/list.jsp”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)。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>一覧表示画面</title>
<script type="text/javascript">
<!--
function setDeleteSubmit(param) {
    document.forms[0].deleteId.value=param;
    document.forms[0].submit();
}
-->
</script>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                一覧表示画面
            </h2>
        </td>
    </tr>
</table>

<html:messages id="message" message="true">
    <bean:write name="message"/>
</html:messages>

<ts:pageLinks id="userListPageLinks"
              action="/list" name="dynaFormBean" rowProperty="row"
              totalProperty="totalCount" indexProperty="startIndex"
              currentPageIndex="now" totalPageCount="total"/>

<center>
 <!-- <ts:form action="/delete">-->
<h5>

<bean:write name="now"/>／<bean:write name="total"/> (<bean:write name="dynaFormBean" pro
perty="totalCount"/>)

<p/>

<bean:write name="userListPageLinks" filter="false"/>

<p/>
<table border="1" frame="box">
    <tr>
        <td width="30"  align="center"><b>ID</b></td>
        <td width="100" align="center"><b>NAME</b></td>
        <td width="30"  align="center"><b>AGE</b></td>
        <td width="80"  align="center"><b>BIRTH</b></td>
  <td>-</td>
    </tr>
    <logic:iterate id="userBean" name="userBeans" scope="session">
```

```
    <tr>
        <td align="center">
          <bean:write name="userBean" property="id"/>
        </td>
        <td>
          <bean:write name="userBean" property="name"/>
        </td>
        <td align="center">
          <bean:write name="userBean" property="age"/>
        </td>
        <td align="right">
          <bean:write name="userBean" property="birth"/>
        </td>
        <td>
          <bean:define id="id" name="userBean" property="id"/>
     <input type="button" value="削除" onclick="setDeleteSubmit(<'id');"/>
        </td>
    </tr>
    </logic:iterate>
</table>
<p/>
<input type="hidden" name="deleteId" value="" />

 <bean:write name="userListPageLinks" filter="false"/>

</h5>

</center>

<hr>

<ts:link page="/insertSCR.do">登録画面</ts:link>

<ts:link page="/welcome.do">インデックス</ts:link>

</ts:form>
</body>
</html:html>
```

③ 登録画面を作成する。
“terasoluna-spring-thin-blank/webapps/insert.jsp”ファイルを以下のように作成する。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>登録画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                登録画面
            </h2>
        </td>
    </tr>
</table>

<html:errors />
<ts:form action="/regist">

<fieldset style="border:2pt solid black;padding:20px;width:100%;">
   <legend style="font-weight:bold;">
     登録情報入力
   </legend><p>
   <table border="0" frame="box">
       <tr>
           <td valign="top">
               <table frame="box">
                   <tr>
                       <td align="center" width="50">
                           ID
                       </td>
                       <td width="200">
                           <html:text name="dynaFormBean" property="id" maxlength="10" />
                           <br><font size="1" color="red">(必須入力,数字のみ 10 桁以内)</font>
                       </td>
                   </tr>
                   <tr>
                       <td align="center" width="50">
                           NAME
                       </td>
                       <td width="200">
                          <html:text name="dynaFormBean" property="name" maxlength="20"/>
                          <br><font size="1" color="red">(必須入力,20 桁以内)</font>
                       </td>
                   </tr>
                   <tr>
                       <td align="center" width="50">
                           AGE
                       </td>
                       <td width="200">
                           <html:text name="dynaFormBean" property="age" maxlength="3" />
                           <br><font size="1" color="red">(必須入力,数字のみ 3 桁以内)</font>
                       </td>
                  </tr>
```

```
                    <tr>
                       <td align="center" width="50">
                            BIRTH
                       </td>
                       <td width="200">
                          <html:text name="dynaFormBean" property="birth" maxlength="10" />
                           <t:inputCalendar for="birth" formatKey="calendar.format.key"/>
                           <br><font size="1" color="red">(yyyy/MM/dd形式)</font>
                          </td>
                      </tr>
                </table>
            </td>
        </tr>
        <tr>
            <td>
               <ts:submit value="登録" />
            </td>
        </tr>
    </table>
</fieldset>

<hr>

<ts:link page="/list.do">一覧表示画面</ts:link>
<ts:link page="/welcome.do">インデックス</ts:link>

</ts:form>
</body>
</html:html>
```

- **ビジネスロジックのインタフェース作成**

情報登録ビジネスロジックのインターフェースを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.blogic/RegisterBLogic.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.thin.tutorial.service.blogic;

import jp.terasoluna.thin.tutorial.service.bean.UserBean;

/**
 * 登録ビジネスロジックのインタフェース。
 *
 */
public interface RegisterBLogic {

    /**
     * 入力された情報をデータベースに登録する。
     * 登録に成功した件数を返却する。
     *
     * @param bean 画面にて入力された値を保持するBean
     * @return 登録結果
     */
    public boolean register(UserBean bean);

}
```

● ビジネスロジックの作成

情報登録ビジネスロジックを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.service.blogic/RegisterBLogicImpl.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.thin.tutorial.service.blogic;

import java.util.HashMap;
import java.util.Map;

import jp.terasoluna.fw.dao.QueryDAO;
import jp.terasoluna.fw.dao.UpdateDAO;
import jp.terasoluna.thin.tutorial.service.bean.UserBean;

/**
 * 入力情報登録ビジネスロジック。
 *
 * 画面から入力された値をデータベースに登録する。
 *
 */
public class RegisterBLogicImpl implements RegisterBLogic{

    /**
     * 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;
    }

    /**
     * 入力された情報をデータベースに登録する。
     * 登録に成功した件数を返却する。
     *
     * @param bean 画面にて入力された値を保持するBean
     * @return 登録結果
     */
    public boolean register(UserBean bean) {

        //重複チェック実行
        //エラーの場合は、falseを返却する。
        if (!check(bean)) {
            return false;
        }

        //登録後、結果件数を返却する。
        updateDAO.execute("insertUser", bean);

        return true;
    }

    /**
     * 入力されたIDが重複しているかどう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 bean 画面にて入力された値を保持するBean
     * @return チェック結果
     */
    protected boolean check(UserBean bean) {

        //IDからデータを検索する。
        Map<String, Long> params = new HashMap<String, Long>();
        params.put("ID", bean.getId());
        Map<String, Object>[] result
            = queryDAO.executeForMapArray("getUser", params);

        //IDで検索し、データが取得できた場合は、重複エラーとする。
        if (result != null && result.length > 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;
    }
}
```

- **アクションの作成**

情報登録ビジネスロジックを呼び、データベースへ情報を登録する情報登録アクションを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web.action/RegisterAction.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.thin.tutorial.web.action;
```

```
import jp.terasoluna.fw.service.thin.BLogicMessage;
import jp.terasoluna.fw.service.thin.BLogicMessages;
import jp.terasoluna.fw.service.thin.BLogicResult;
import jp.terasoluna.fw.web.struts.actions.AbstractBLogicAction;
import jp.terasoluna.thin.tutorial.service.bean.UserBean;
import jp.terasoluna.thin.tutorial.service.blogic.RegisterBLogic;

import org.apache.struts.Globals;

/**
 * 入力情報登録アクション。
 *
 * 登録を行うビジネスロジックを呼び、入力された値をデータベースに登録する。
 *
 */
public class RegisterAction extends AbstractBLogicAction<UserBean> {

    /**
     * 入力情報登録ビジネスロジッククラス。
     */
    private RegisterBLogic registBLogic = null;

    /**
     * 入力情報登録ビジネスロジックを返却する。
     *
     * @return 保持する入力情報登録ビジネスロジック
     */
    public RegisterBLogic getRegistBLogic() {
        return registBLogic;
    }

    /**
     * 入力情報登録ビジネスロジックを設定する。
     *
     * @param registBLogic 入力情報登録ビジネスロジック
     */
    public void setRegistBLogic(RegisterBLogic registBLogic) {
        this.registBLogic = registBLogic;
    }

    /**
     * ビジネスロジックを実行し、結果を返却する。
     *
     * @param bean 入力された値を保持したbean
     * @return 登録結果を保持したBLogicResult
     * @throws Exception 例外
     */
    @Override
    public BLogicResult doExecuteBLogic(UserBean bean) throws Exception {

        //BLogicResultの生成、結果の設定
        BLogicResult result = new BLogicResult();

        //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 (!registBLogic.register(bean)) {

            //重複エラー時の処理
            setErrors(result, "errors.input.id.repeat", "error");
            result.setResultObject(bean);
            return result;
        }

        //正常な場合は、結果文字列を設定して返却する。
        result.setResultString("success");
        result.setResultObject(new UserBean());
        return result;
    }

    /**
     * 引数のBLogicResultオブジェクトに
     * エラーメッセージおよび結果文字列を設定する。
```

```
     *
     * @param result BLogicResultオブジェクト
     * @param errorKey 設定するエラーキー
     * @param resultString 結果文字列
     */
    private void setErrors(BLogicResult result, String errorKey,
            String resultString) {
        BLogicMessages errors = result.getErrors();
        if (errors == null) {
            errors = new BLogicMessages();
        }
        errors.add(Globals.ERROR_KEY,
                new BLogicMessage(errorKey));
        result.setErrors(errors);
        result.setResultString(resultString);
    }
}
```

- **“blogic-io.xml”ファイルの変更**

アクションフォームとビジネスロジック間の値の入出力を定義する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/blogic-io.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" ?>

<!DOCTYPE blogic-io PUBLIC "-//NTTDATA//DTD TERASOLUNA for Spring blogic-io 1.0//JA"
                           "dtd/blogic-io.dtd">

<blogic-io>

   <action path="/logon">
     <blogic-params bean-name="java.util.HashMap">
       <set-property property="userId"
                     source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="USER_VALUE_OBJECT" blogic-property="uvo"
                     dest="session" />
     </blogic-result>
   </action>

   <action path="/list">
     <blogic-params bean-name="java.util.HashMap">
       <set-property property="startIndex" source="form" />
       <set-property property="row"        source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="userBeans"  dest="session" />
       <set-property property="totalCount" dest="form" />
     </blogic-result>
   </action>

   <action path="/regist">
     <blogic-params
       bean-name="jp.terasoluna.thin.tutorial.service.bean.UserBean">
       <set-property property="id"     source="form" />
       <set-property property="name"   source="form" />
       <set-property property="age"    source="form" />
       <set-property property="birth"  source="form" />
     </blogic-params>
     <blogic-result>
       <set-property property="id"     source="form" />
       <set-property property="name"   source="form" />
       <set-property property="age"    source="form" />
       <set-property property="birth"  source="form" />
     </blogic-result>
   </action>
```

```
    <action path="/delete">
      <blogic-params
        bean-name="jp.terasoluna.thin.tutorial.service.bean.UserBean">
        <set-property property="deleteId" blogic-property="id" source="form" />
      </blogic-params>
      <blogic-result>
      </blogic-result>
    </action>

</blogic-io>
```

- **“struts-config.xml”ファイルの変更**

登録用のアクションフォームとアクションパスを設定する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/struts-config.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" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>

<!-- ======================================== Form Bean Definitions -->
  <form-beans>

    <!-- ログオン用 -->
    <form-bean name="logonForm"
               type="jp.terasoluna.thin.tutorial.web.form.LogonForm" />

    <form-bean name="dynaFormBean"
        type="jp.terasoluna.fw.web.struts.form.DynaValidatorActionFormEx" >

      <!-- 一覧表示用 -->
      <form-property name="row"        type="java.lang.String" initial="10"/>
      <form-property name="startIndex" type="java.lang.String" initial="0"/>
      <form-property name="totalCount" type="java.lang.String"/>

      <!-- 登録用 -->
      <form-property name="id"         type="java.lang.Long"/>
      <form-property name="name"       type="java.lang.String"/>
      <form-property name="age"        type="java.lang.String"/>
      <form-property name="birth"      type="java.lang.String"/>

      <!-- 削除用 -->
      <form-property name="deleteId"   type="java.lang.Long"/>
    </form-bean>

  </form-beans>

<!-- ======================================== Global Exception Definitions -->
  <global-exceptions>

  </global-exceptions>


<!-- ======================================== Global Forward Definitions -->
  <global-forwards>
    <forward name="welcome"
             path="/logoff.do" redirect="true"/>
  </global-forwards>


<!-- ======================================== Global Forward Definitions -->
  <action-mappings type="jp.terasoluna.fw.web.struts.action.ActionMappingEx">

    <action path="/welcome"
```

```
            parameter="/welcome.jsp"/>

    <action path="/logon"
            name="logonForm" scope="request">
        <forward name="success" path="/welcome.do"/>
    </action>

    <action path="/logoff"
            parameter="/logon.jsp"/>

    <action path="/list"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/listSCR.do"/>
    </action>

    <action path="/listSCR"
            name="dynaFormBean" scope="request"
            parameter="/list.jsp" />

    <action path="/regist"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/list.do"/>
        <forward name="error"   path="/insertSCR.do"/>
    </action>

    <action path="/insertSCR"
            name="dynaFormBean" scope="request"
            parameter="/insert.jsp"/>

    <action path="/delete"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/list.do"/>
    </action>
  </action-mappings>


                                          :
                                        略
                                          :

</struts-config>
```

- **“moduleContext.xml”ファイルの変更**

登録用のアクションとビジネスロジックを登録する。
“terasoluna-spring-thin-blank/webapps/WEB-INF/moduleContext.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" ?>

<!-- モジュール固有の Bean 定義 -->
<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/sp
ring-util-2.0.xsd">

<!-- 共通定義のインポート -->
  <import resource="/commonContext.xml"/>

<!-- アクション定義 -->
  <bean name="/welcome" scope="prototype"
        class="jp.terasoluna.fw.web.struts.actions.ForwardAction" />

  <bean name="/logon"
```

```
        class="jp.terasoluna.fw.web.struts.actions.BLogicAction"
        scope="prototype">
    <property name="businessLogic">
      <ref bean="logonBLogic"/>
    </property>
  </bean>

  <bean name="/logoff" scope="prototype"
        class="jp.terasoluna.fw.web.struts.actions.LogoffAction" />

   <bean name="/list" scope="prototype"
        class="jp.terasoluna.thin.tutorial.web.action.ListAction">
    <property name="listBLogic">
      <ref local="listBLogic"/>
    </property>
  </bean>

  <bean name="/listSCR" scope="prototype"
        class="org.apache.struts.actions.ForwardAction"/>

   <bean name="/regist" scope="prototype"
        class="jp.terasoluna.thin.tutorial.web.action.RegisterAction">
    <property name="registBLogic">
      <ref local="registBLogic"/>
    </property>
  </bean>

  <bean name="/insertSCR" scope="prototype"
        class="org.apache.struts.actions.ForwardAction"/>

  <bean name="/delete" scope="prototype"
        class="jp.terasoluna.thin.tutorial.web.action.DeleteAction">
    <property name="deleteBLogic">
      <ref local="deleteBLogic"/>
    </property>
  </bean>

<!-- ビジネスロジック定義 -->
  <bean id="listBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.ListBLogicImpl">
         <property name="dao"><ref bean="queryDAO"/></property>
   </bean>

  <bean id="registBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.RegisterBLogicImpl">
    <property name="updateDAO"><ref bean="updateDAO"/></property>
    <property name="queryDAO" ><ref bean="queryDAO" /></property>
  </bean>

 <bean id="deleteBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.DeleteBLogicImpl">
    <property name="dao"><ref bean="updateDAO"/></property>
  </bean>

  <bean id="logonBLogic" scope="prototype"
        class="jp.terasoluna.thin.tutorial.service.blogic.LogonBLogic" />

</beans>
```

- **“calendar_ja.properties”ファイルの変更**

“terasoluna-spring-thin-blank/sources/calendar_ja.properties”ファイルを以下のように作成する。

```
################################################################################
##
## カレンダー入力機能(InputCalendarTag)で読み込むリソースファイル。
##
## カレンダー入力機能が取得するリソースバンドルのファイルで、
## 以下の値を変更することで、画面に出力するカレンダーの表示等を変更できる。
##
## ※このファイルはサンプルとして提供するものであるため、
##    業務にて自由に変更を行って使用してください。##
################################################################################
#画像保存場所のディレクトリ取得キー。
calendar.img.dir=image/calendar/

#現在日付付与文字列の取得キー。
calendar.today.string=今日は

#ボタン表示文字列の取得キー。
calendar.button.value=カレンダー

#スタイルシートプレフィックスの取得キー。
calendar.style.themeprefix=BlueStyle

#スタイルシート保存場所の取得キー。
calendar.stylesheet.dir=stylesheet/

#カレンダー外部JavaScript保存場所の取得キー。
calendar.javascript.dir=javascript/

#休日定義。
calendar.holiday.1=0,1,1,元旦
calendar.holiday.2=0,2,11,建国記念日
calendar.holiday.3=0,4,29,みどりの日
calendar.holiday.4=0,5,3,憲法記念日
calendar.holiday.5=0,5,4,国民の休日
calendar.holiday.6=0,5,5,こどもの日
calendar.holiday.7=0,11,3,文化の日
calendar.holiday.8=0,11,23,勤労感謝の日
calendar.holiday.9=0,12,23,天皇誕生日
```

- **“sqlMap.xml”ファイルの変更**

登録で使用するsqlを設定する。

“terasoluna-spring-thin-blank/sources/sqlMap.xml”ファイルを以下ように変更する(網掛け部分を追加する)。

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

<sqlMap namespace="user">

    <select id="getUser"
            resultClass="java.util.HashMap">
        SELECT ID, NAME, AGE, BIRTH FROM USERLIST WHERE ID = #ID#
    </select>

    <select id="getUserCount"
            resultClass="java.lang.String"
            resultSetType="SCROLL_INSENSITIVE">
        SELECT
            COUNT(ID)
        FROM
            USERLIST
    </select>
```

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

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

    <delete id="deleteUser"
        parameterClass="jp.terasoluna.thin.tutorial.service.bean.UserBean">
        DELETE FROM USERLIST
        WHERE ID = #id#
    </delete>

</sqlMap>
```

- **“application-messages.properties”の変更**

重複した ID を登録しようとした時に出るエラーメッセージを設定する。
“terasoluna-spring-thin-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)

```
errors.prefix=●
errors.suffix=<br>
errors.headeerrors
errors.header=<font color=red><h3>以下のエラーが発生しました。</h3>
errors.input.id.repeat=入力された ID は重複しています。別 ID を指定してください。
```

■ 確認

登録画面に遷移し、入力した値がデータベースに登録され、一覧表示に追加されることを確認する。
また、重複した ID で登録するとエラーメッセージが出ることも確認する。

(1) 作成・変更したファイルをすべて保存し、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) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスし、ログオン画面が表示されたら下図のように UserName を入力しログオンボタンを押下する。

図26 ログオン画面

(4) ログオンボタン押下後、下図のようにインデックス画面に遷移する。

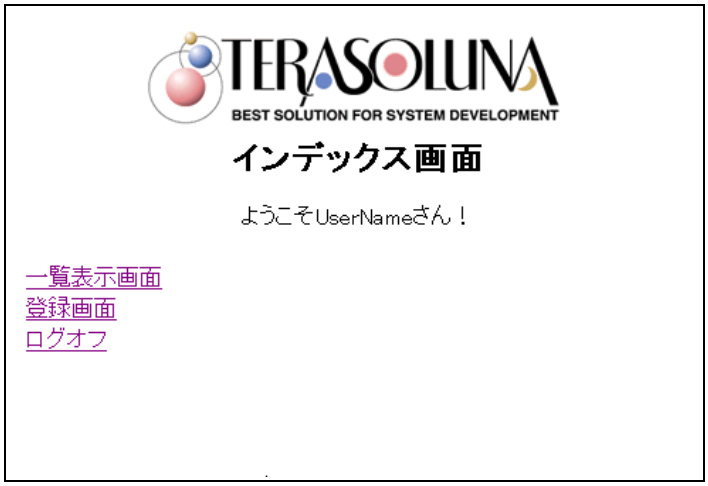


図27 インデックス画面

(5) インデックス画面遷移後、登録リンクを押下する。リンク後、以下のように登録情報を入力し、登録ボタンを押下する。

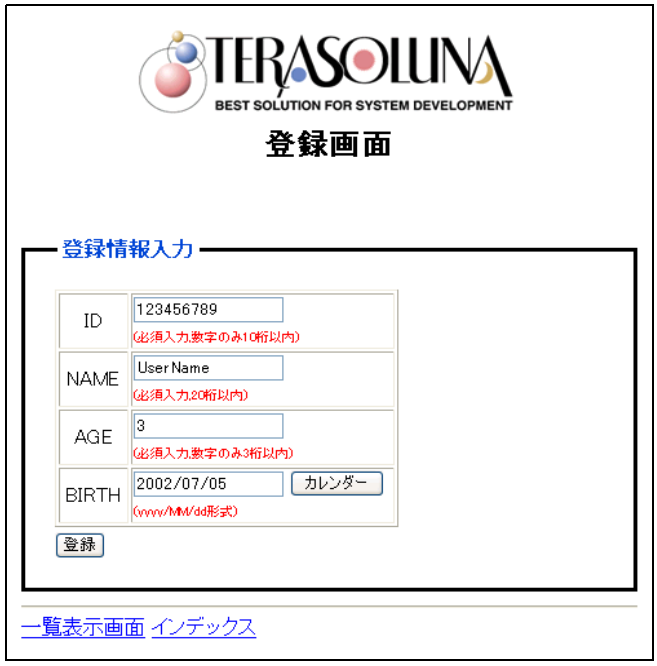


図28　登録画面

(6) 登録ボタン押下後、一覧表示画面面に遷移するので、登録画面で登録した内容が一覧に追加されていることを確認する。

図29　一覧表示画面(登録後)

(7) また、重複した ID を登録する時にエラーメッセージが表示されることを確認する。

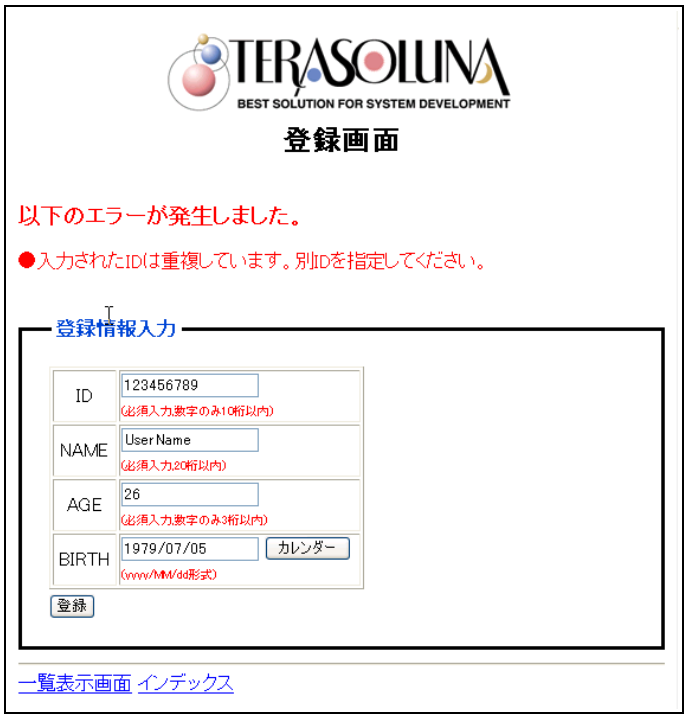


図30 登録画面(ID 重複)

■ 参考資料

- Spring 版機能説明書
  『CB-01 データベースアクセス機能』
  『WJ-01～WK-06 画面表示機能』

2.8　入力チェック

本節では、画面上に入力された値に対する入力チェックの方法に関して学習する。
本節で説明する入力チェックには以下の種類がある。

- 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
- 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。
- DB 整合性チェック　:ビジネスロジックで実装する

なお、本書では単項目チェックと相関チェックを実装する。

2.8.1　単項目チェック

本節では、画面上に入力された値に対して入力チェックを行い、エラーとなった場合には画面にメッセージを出力する方法に関して学習する。

■ 概要

下図のように、前節までに作成したログオン機能及び登録機能に入力チェック処理を追加する。各機能で入力値にエラーがあった場合には値を入力した画面に戻り、その際のエラーメッセージを画面上に表示する。ただし、登録画面での必須チェック及び日付書式チェックについては、ダイアログでエラーメッセージを表示する。

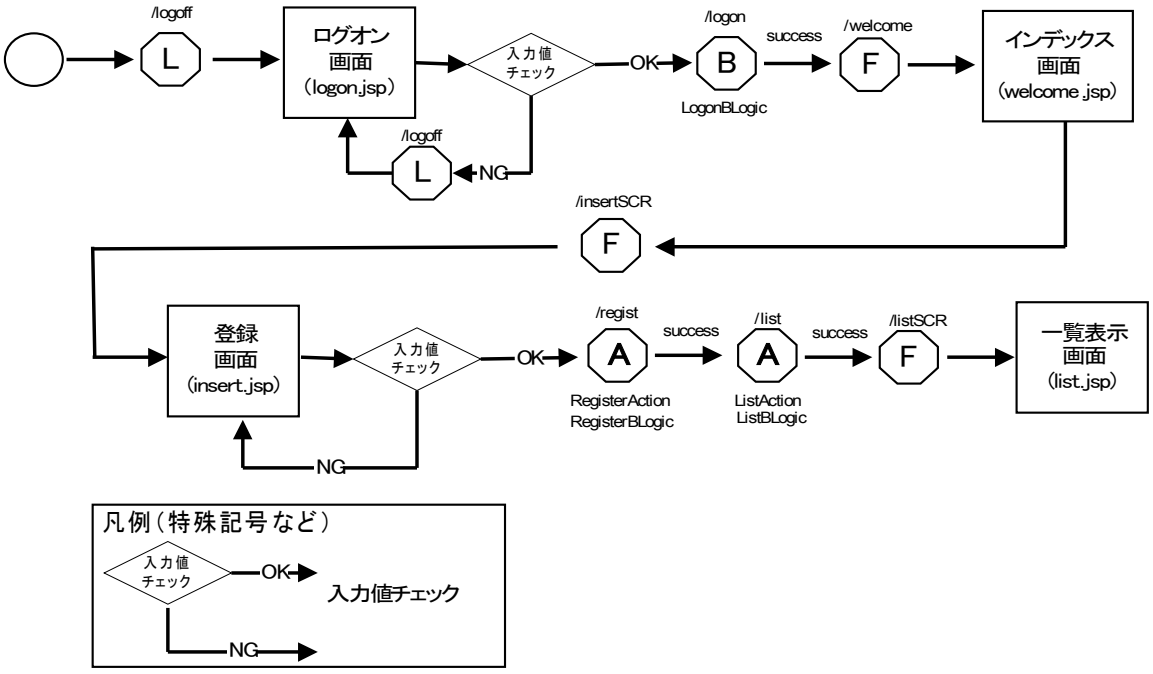


図31　単項目チェック画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
2. ログオン画面へ遷移する。
3. UserName に不適切な値を入力し、ログオンボタンを押下してエラーが表示される。
4. UserName に適切な値を入力し、ログオンボタンを押下してインデックス画面へ遷移する。
5. インデックス画面から登録画面リンクを押下して登録画面へ遷移する。
6. ID、NAME、AGE、BIRTH に不適切な値を入力し、登録ボタンを押下してエラーが表示される。
7. ID、NAME、AGE、BIRTH に適切な値を入力し、登録ボタンを押下して一覧表示画面へ遷移する。

このとき、以下の作業が必要となる。

- JSP ファイルの変更

- “validation.xml”ファイルの変更
- “struts-config.xml”ファイルの変更
  - “application-messages.properties”の変更

■ **手順**

● **JSP ファイルの変更**

① ログオン画面に、エラーメッセージを出力するためのタグを記述する。
“terasoluna-spring-thin-blank/webapps/logon.jsp”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。)

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>ログオン画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                ログオン画面
            </h2>
        </td>
    </tr>
</table>

<p>
<html:errors />
<ts:form action="/logon">
<table border="0" align="center">
  <tr>
    <td>
      UserName :
    </td>
    <td>
      <html:text property="userId"/>
    </td>
    <td>
      <html:submit value="ログオン"/>
    </td>
  </tr>
</table>
</ts:form>
</body>
</html:html>
```

② 登録画面で、一部の入力チェックについてクライアント側でチェックを行うための記述を行なう。
“terasoluna-spring-thin-blank/webapps/insert.jsp”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。)

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>登録画面</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h2>
                登録画面
            </h2>
        </td>
    </tr>
</table>

<html:errors />
<html:javascript formName="/regist" method="validateRegist"/>
<ts:form action="/regist" onsubmit="return validateRegist(this);">

<fieldset style="border:2pt solid black;padding:20px;width:100%;">
    <legend style="font-weight:bold;">
      登録情報入力
    </legend><p>
    <table border="0" frame="box">
        <tr>
            <td valign="top">
                <table frame="box">
                    <tr>
                        <td align="center" width="50">
                            ID
                        </td>
                        <td width="200">
                           <html:text name="dynaFormBean" property="id" maxlength="10" />
                           <br><font size="1" color="red">(必須入力,数字のみ10桁以内)</font>
                        </td>
                    </tr>
                    <tr>
                        <td align="center" width="50">
                            NAME
                        </td>
                        <td width="200">
                         <html:text name="dynaFormBean" property="name" maxlength="20" />
                         <br><font size="1" color="red">(必須入力,20桁以内)</font>
                        </td>
                    </tr>
                    <tr>
                        <td align="center" width="50">
                            AGE
                        </td>
                        <td width="200">
                           <html:text name="dynaFormBean" property="age" maxlength="3" />
                            <br><font size="1" color="red">(必須入力,数字のみ3桁以内)</font>
                        </td>
                    </tr>
                    <tr>
                        <td align="center" width="50">
                            BIRTH
```

```
                        </td>
                        <td width="200">
                        <html:text name="dynaFormBean" property="birth" maxlength="10" />
                        <t:inputCalendar for="birth" formatKey="calendar.format.key"/>
                            <br><font size="1" color="red">(yyyy/MM/dd形式)</font>
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
        <tr>
            <td>
                <ts:submit value="登録" />
            </td>
        </tr>
    </table>
</fieldset>

<hr>

<ts:link page="/list.do">一覧表示画面</ts:link>

<ts:link page="/welcome.do">インデックス</ts:link>

</ts:form>
</body>
</html:html>
```

- **“validation.xml”ファイルの変更**

ログオン及び登録時のアクションパスに対して、入力チェックするための設定を行なう。“terasoluna-spring-thin-blank/webapps/WEB-INF/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。
ここでは各項目に対してそれぞれ以下のような入力チェックを行っている。

- UserId：必須入力、半角英数字文字列のみ、最大入力桁 10 桁
- id：必須入力、数字文字列のみ、最大入力桁 10 桁
- name：必須入力、最大入力桁 20 桁
- age ：必須入力、数字文字列のみ、最大入力桁 3 桁
- birth：必須入力、date 型形式 のみ

```
<?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>

      <!-- ログオン機能確認 -->
      <form name="/logon">
        <field property="userId"
            depends="required,alphaNumericString,maxlength">
          <arg key="logon.userId" position="0"/>
          <arg key="${var:maxlength}" name="maxlength"
               resource="false" position="1"/>
          <var>
            <var-name>maxlength</var-name>
            <var-value>10</var-value>
          </var>
        </field>
      </form>

      <!-- 登録情報確認 -->
      <form name="/regist">
```

```
        <field property="id"
            depends="required,numericString,maxlength">
          <arg key="insert.id" position="0"/>
          <arg key="${var:maxlength}" resource="false" position="1"/>
          <var>
            <var-name>maxlength</var-name>
            <var-value>10</var-value>
          </var>
        </field>
        <field property="name"
            depends="required,maxlength">
          <arg key="insert.name" position="0"/>
          <arg key="${var:maxlength}" resource="false" position="1"/>
          <var>
            <var-name>maxlength</var-name>
            <var-value>20</var-value>
          </var>
        </field>
        <field property="age"
            depends="required,numericString,maxlength">
          <arg key="insert.age" position="0"/>
          <arg key="${var:maxlength}" resource="false" position="1"/>
          <var>
            <var-name>maxlength</var-name>
            <var-value>3</var-value>
          </var>
        </field>

        <field property="birth"
            depends="date,maxlength">
          <arg key="insert.birth" position="0"/>
          <arg key="${var:maxlength}" name="maxlength" resource="false" position="1"/>
          <arg key="${var:datePattern}" name="date" resource="false" position="1"/>
          <var>
            <var-name>datePattern</var-name>
            <var-value>yyyy/MM/dd</var-value>
          </var>
          <var>
            <var-name>maxlength</var-name>
            <var-value>10</var-value>
          </var>
        </field>

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

- **“struts-config.xml”ファイルの変更**

入力チェックエラー時の遷移先の設定を行なう。
“terasoluna-spring-thin-blank/webapps/WEB-INF/struts-config.xml”ファイルを以下のように変更する。(網掛け部分を変更する。)

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

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>
                                          :
                                        略
                                          :
  <action-mappings type="jp.terasoluna.fw.web.struts.action.ActionMappingEx">

    <action path="/welcome"
            parameter="/welcome.jsp"/>
```

```
    <action path="/logon"
            name="logonForm" scope="request"
            validate="true" input="/logoff.do">
        <forward name="success" path="/welcome.do"/>
    </action>

    <action path="/logoff"
            parameter="/logon.jsp"/>

    <action path="/list"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/listSCR.do"/>
    </action>

    <action path="/listSCR"
            name="dynaFormBean" scope="request"
            parameter="/list.jsp" />

    <action path="/regist"
            name="dynaFormBean" scope="request"
            validate="true" input="/insert.jsp">
        <forward name="success" path="/list.do"/>
        <forward name="error"   path="/insertSCR.do"/>
    </action>

    <action path="/insertSCR"
            name="dynaFormBean" scope="request"
            parameter="/insert.jsp"/>

    <action path="/delete"
            name="dynaFormBean" scope="request">
        <forward name="success" path="/list.do"/>
    </action>

    <action path="/error"
            parameter="/error.jsp"/>

  </action-mappings>
                                          :
                                        略
                                          :
```

- **“application-messages.properties”の変更**

必須入力のエラーメッセージを設定する。
“terasoluna-spring-thin-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)

```
insert.id="ID"
insert.name="NAME"
insert.age="AGE"
insert.birth="BIRTH"
```

■ **確認**

入力チェックでエラーとなる値を画面で入力した場合、エラーメッセージが出力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、コンパイルを行なったのちに 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) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。ログオン画面へ遷移する。

(4) 下図のようにログオン画面が表示されるので、最初に、UserName の必須チェックの動作確認を行なう。UserName 入力欄に何も入力せずに「ログオン」ボタンを押下する。

図32 ログオン画面

(5) 下図のように、「UserName は入力必須項目です」というエラーメッセージが、赤字でログオン画面に表示されることを確認する。

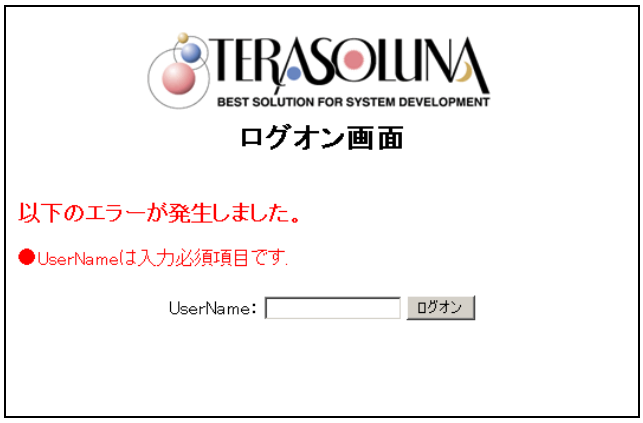


図33 ログオン画面（エラー）

(6) 次に、UserName の半角英数字文字列チェックの動作確認を行なう。引き続き、ログオン画面で、UserName 入力欄に"あいうえお"と入力し、「ログオン」ボタンを押下する。

(7) 下図のように、「UserName には半角英数字で入力してください」というエラーメッセージが、赤字でログオン画面に表示されることを確認する。

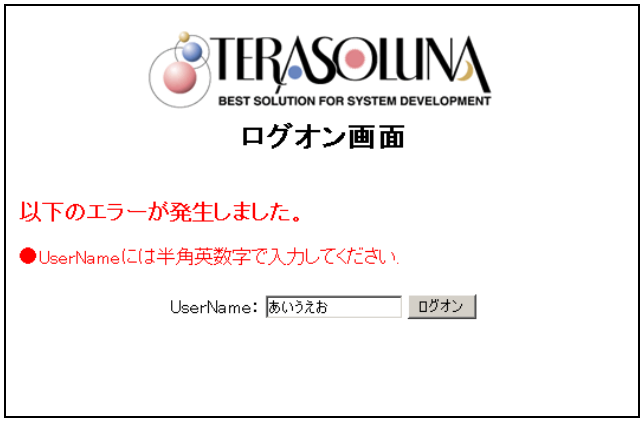


図34 ログオン画面（エラー）

(8) 次に、UserName の文字列長チェックの動作確認を行なう。引き続き、ログオン画面で、UserName 入力欄に"12345678901"と入力し、「ログオン」ボタンを押下する。

(9) 下図のように、「UserName には 10 文字以下にしてください」というエラーメッセージが、赤字でログオン画面に表示されることを確認する。

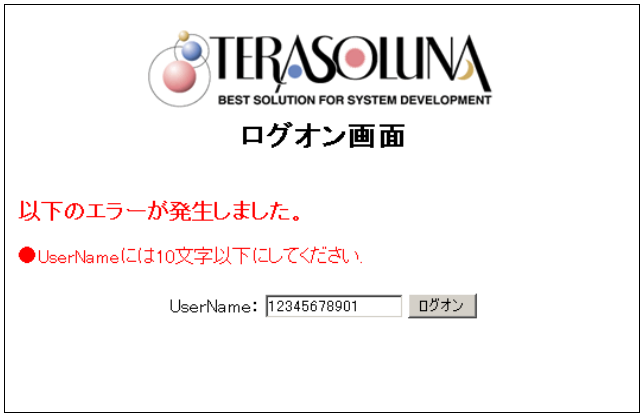


図35 ログオン画面（エラー）

(10) 入力チェックエラーがなかった場合の動作確認を行なう。引き続き、ログオン画面で、UserName 入力欄に"12345"と入力し、「ログオン」ボタンを押下する。

(11)　下図のように、インデックス画面に遷移することを確認する。

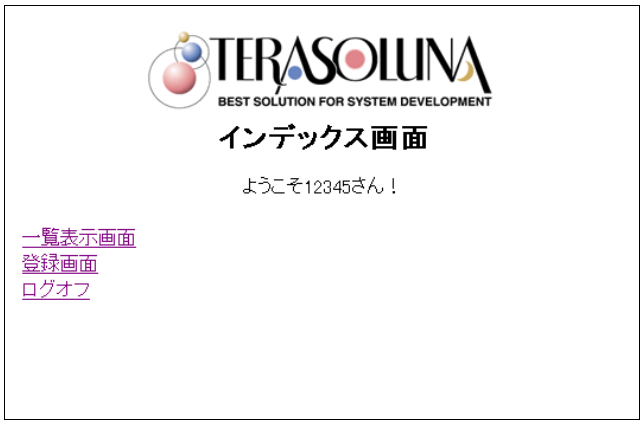


図36　インデックス画面

(12)　「登録画面」を押下し、登録画面に遷移することを確認する。

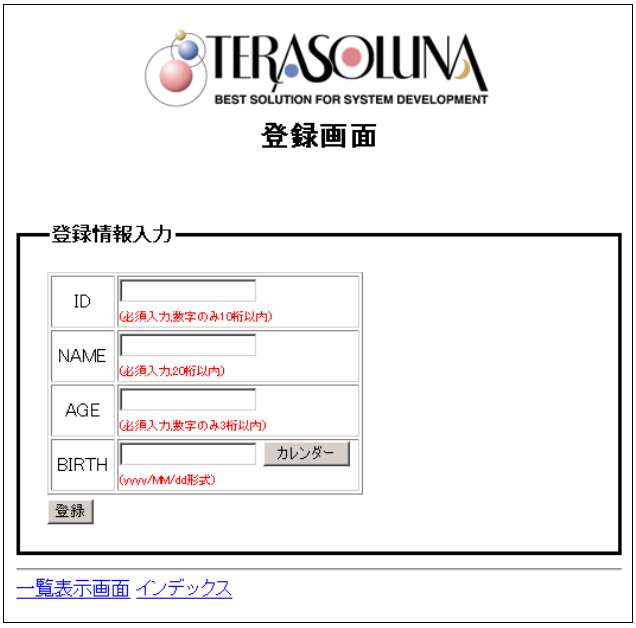


図37　登録画面

(13)　次に、ID、NAME、AGE の必須チェックの動作確認を行なう。登録画面で、ID、NAME、AGE 欄に何も入力せずに「登録」ボタンを押下する。

(14) 下図のように、「"ID"は入力必須項目です」「"NAME"は入力必須項目です」「"AGE"は入力必須項目です」というエラーメッセージが、ダイアログで表示されることを確認する。

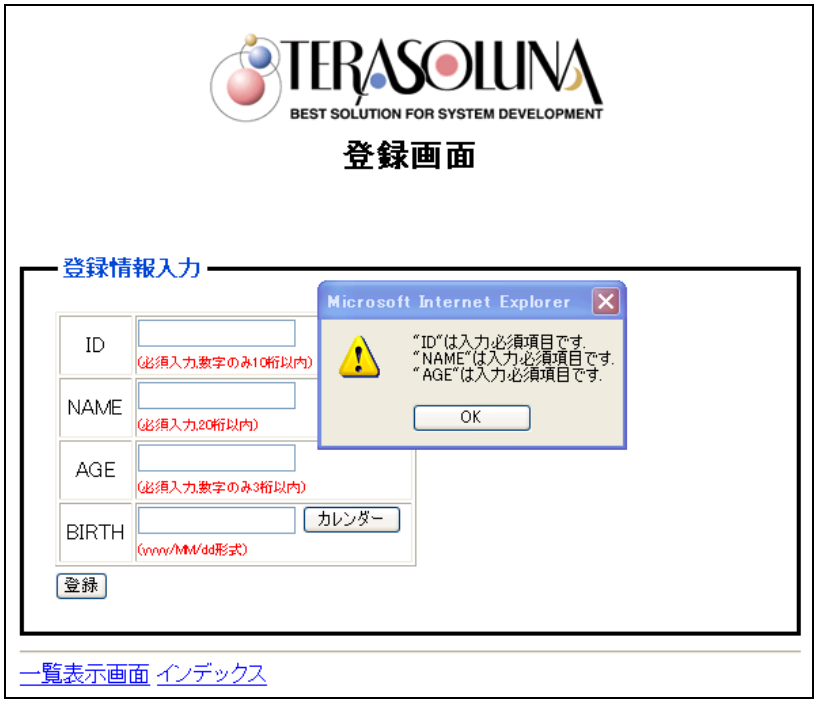


図38 登録画面(エラーダイアログ)

(15) 次に、ID、AGE の数字文字列チェックを行う。引き続き、登録画面で、ID、NAME、AGE 欄に"あいう"と入力し、「登録」ボタンを押下する。(NAME 欄に入力を行わないと、NAME の必須チェックが先に動作し、数字文字列チェックが動作しないので、NAME 欄にも入力する。)

(16) 下図のように、「"ID"には数字を入力してください」「"AGE"には数字を入力してください」というエラーメッセージが、ダイアログで表示されることを確認する。

図39 登録画面(エラー)

(17) 次に、BIRTH の日付書式チェックを行う。引き続き、登録画面で、ID 欄に"12345"、NAME 欄に"あいうえお"、AGE 欄に"67"、BIRTH 欄に"2222/22/22"と入力し、「登録」ボタンを押下する。

(18) 下図のように、「"BIRTH"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が、ダイアログで表示されることを確認する。

図40 登録画面（エラー）

(19) 最後に、エラーがなかった場合の動作確認を行なう。引き続き、登録画面で、ID 欄に"12345"、NAME 欄に"あいうえお"、AGE 欄に"67"、BIRTH 欄に"2000/01/31"と入力し、「登録」ボタンを押下する。

(20)　下図のように、一覧表示画面に遷移することを確認する。

図41　一覧表示画面

(21)　次に、「5」ページリンクを押下し、下図のように結果が正しく画面に表示されることを確認する。

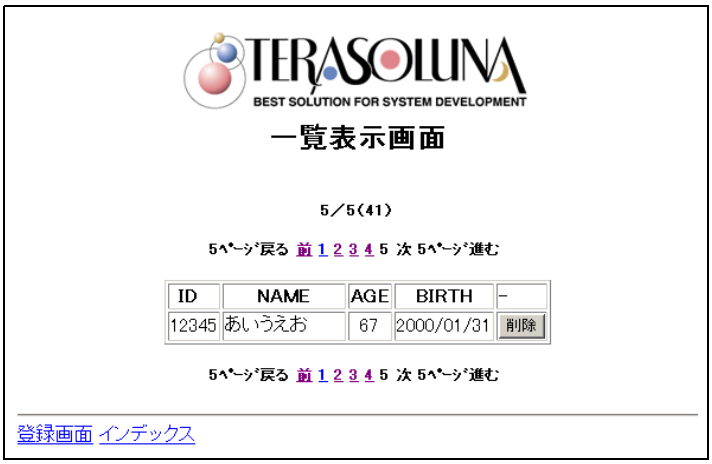


図42　一覧表示画面

■　参考資料

- Spring 版機能説明書
  『WF-01 拡張入力チェック機能』

2.8.2 相関チェック

本節では、画面上に入力された複数の値に対して相関チェックし、エラーとなった場合に、画面にメッセージを出力する方法に関して学習する。

■ **概要**

下図のように、前節までに作成したログオン機能及び登録機能に入力チェック処理（相関チェック）を追加する。相関チェックでエラーの場合には値を入力した画面に戻り、その際のエラーメッセージを画面上に表示する。

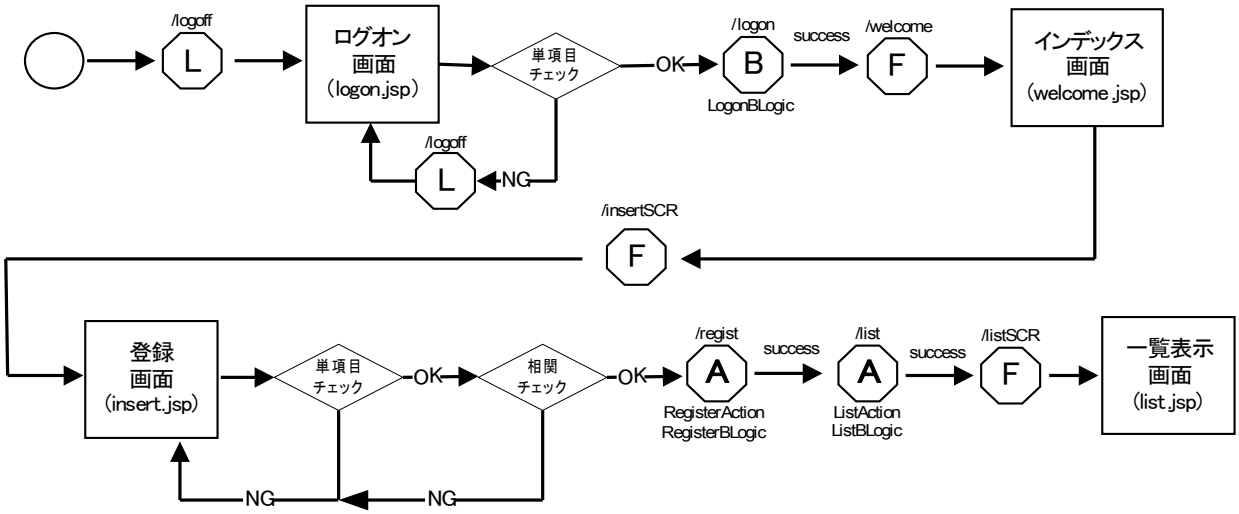


図43 相関チェック画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
2. ログオン画面へ遷移する。
3. UserName に適切な値を入力し、ログオンボタンを押下してインデックス画面へ遷移する。
4. インデックス画面から登録画面リンクを押下して登録画面へ遷移する。
5. ID、NAME に不適切な値を、AGE、BIRTH に相関チェックエラーとなる値を入力し、登録ボタンを押下してエラーが表示される。
6. ID、NAME、AGE、BIRTH に適切な値を入力し、登録ボタンを押下して一覧表示画面へ遷移する。

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

■ 手順

● 相関チェッククラスの作成

相関チェックを行うために、MultiFieldValidator を実装した AgeValidator を作成する。ここでは入力された年齢と生年月日から計算された年齢との相関チェックを行うクラスを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web.form/AgeValidator.java”ファイルを作成する。

```
/*
 * $Id$
 *
 * Copyright (c) 2007 NTT DATA Corporation
 *
 */
package jp.terasoluna.thin.tutorial.web.form;

import java.text.SimpleDateFormat;
import jp.terasoluna.fw.util.DateUtil;
import jp.terasoluna.fw.web.struts.form.MultiFieldValidator;

/**
 * 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 implements MultiFieldValidator {

    /**
     * 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 fields 関連する入力フィールド値
     * @return チェック結果
     */
    public boolean validate(String birth, String[] fields) {
        String age = fields[0];

        if (birth == null || "".equals(birth)) {
            return true;
        }

        return checkBirthAndAge(birth, Integer.parseInt(age));
    }

    /**
     * 入力された生年月日から計算した年齢と入力された年齢を比較する。
     * 同一の場合はtrueを返却、異なっている場合は、falseを返却する。
     *
     * @param birth 入力された誕生日
     * @param inputage 入力された年齢
     * @return 比較結果
     */
    private boolean checkBirthAndAge(String birth, int inputage) {

        //現在日付の取得
        SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd");
        String temp = format.format(DateUtil.getSystemTime());

        //年月日を配列に変換。
        String[] old = birth.split("/");
        String[] now = temp.split("/");

        //単純に現在の年から、入力された年を引く。
        int age = Integer.parseInt(now[0]) - Integer.parseInt(old[0]);

        //入力された月が現在の月以下の場合、且つ
        //入力された日が現在の日よりも小さい場合は、マイナス1する。
        if ((Integer.parseInt(now[1]) < Integer.parseInt(old[1]))
         || (Integer.parseInt(now[1]) == Integer.parseInt(old[1])
         && Integer.parseInt(now[2]) < Integer.parseInt(old[2]))) {
            age--;
```

```
        }

        //入力された年齢と比較する。
        if (inputage == age) {
            return true;
        }
        return false;
    }
}
```

- **“validation.xml”ファイルの変更**

登録時のアクションパスに対して、相関チェックするための設定を行なう。
“terasoluna-spring-thin-blank/webapps/WEB-INF/validation.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" ?>

<!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>
                                          :
                                        略
                                          :

        <field property="birth"
            depends="multiField">
          <msg key="errors.multiField" name="multiField"/>
          <arg key="insert.birth" position="0" />
          <arg key="insert.age"   position="1" />
          <var>
            <var-name>fields</var-name>
            <var-value>age</var-value>
          </var>
          <var>
            <var-name>multiFieldValidator</var-name>
            <var-value>
              jp.terasoluna.thin.tutorial.web.form.AgeValidator
            </var-value>
          </var>
        </field>

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

- **“application-messages.properties”の変更**

入力した年齢と生年月日から計算された年齢が異なる時に出力されるメッセージを設定する。
“terasoluna-spring-thin-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)

```
errors.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

相関チェックでエラーとなる値を画面で入力した場合、エラーメッセージが出力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、コンパイルを行なったのちに 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) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスし、ログオン画面が表示されたら UserName に"12345"と入力しログオンボタンを押下する。

図44 ログオン画面

(4) ログオンボタン押下後、下図のようにインデックス画面に遷移する。

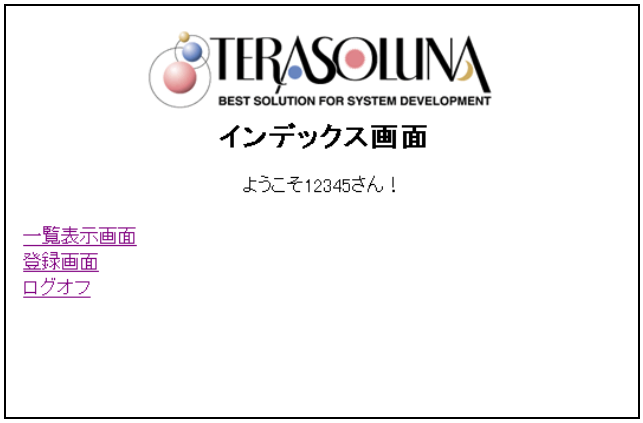


図45 インデックス画面

(5)　インデックス画面で、登録画面リンクを押下し、登録画面に遷移する。

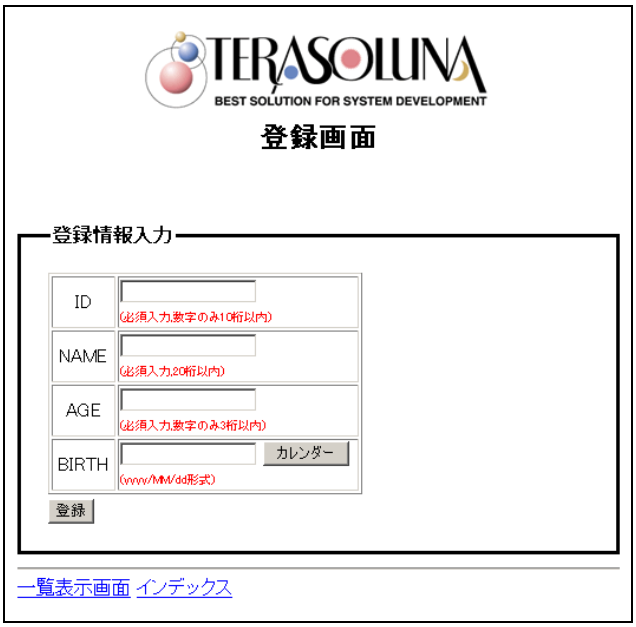


図46　登録画面

(6)　相関チェックの動作確認を行なう。登録画面で、ID 欄に"67890"、NAME 欄に"あいうえお"、AGE 欄に"12"、BIRTH 欄に"2000/01/31"と入力し、「登録」ボタンを押下する。

(7)　下図のように、「["BIRTH"]から計算した["AGE"]が入力された["AGE"]と一致しません」というエラーメッセージが、登録画面に表示されることを確認する。

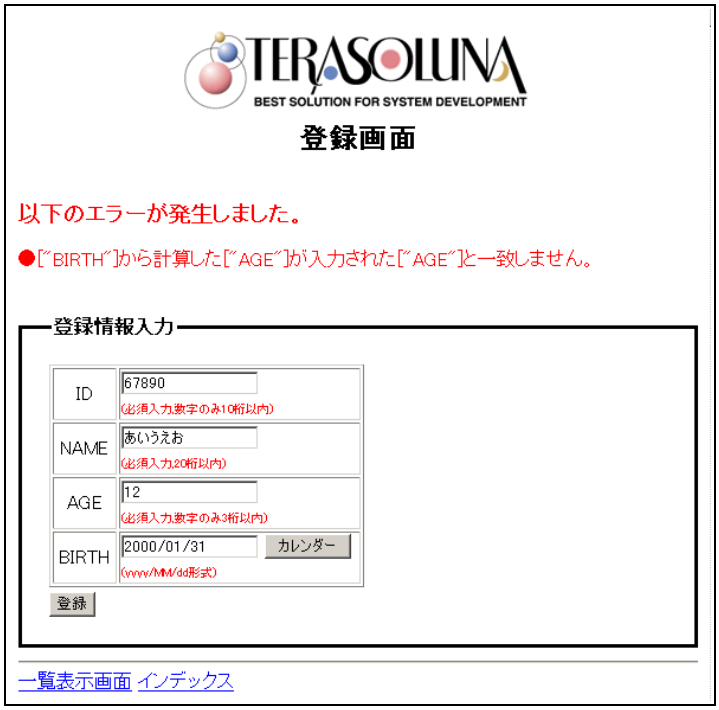


図47　登録画面（エラー）

(8) 最後に、エラーがなかった場合の動作確認を行なう。引き続き、登録画面で、ID 欄に"67890"、NAME 欄に"あいうえお"、AGE 欄に"12"、BIRTH 欄に"1995/01/31"と入力し、「登録」ボタンを押下する。

(9) 下図のように、一覧表示画面に遷移することを確認する。

図48 一覧表示画面

(10) 次に、「5」ページリンクを押下し、下図のように結果が正しく画面に表示されることを確認する。

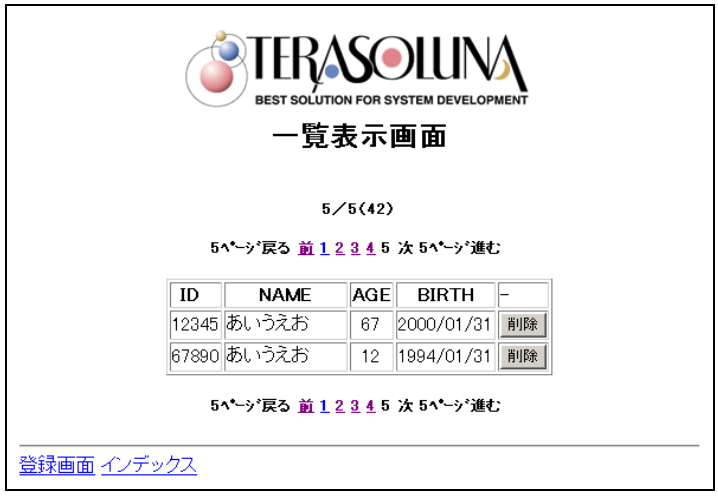


図49 一覧表示画面

■ 参考資料

- Spring 版機能説明書
  『WF-01 拡張入力チェック機能』

2.9　例外処理

本節では、アプリケーションで発生した例外を、共通的に扱ってエラーページに遷移させる方法について学習する。

■ **概要**

下図のように、前節までに作成した DB アクセス機能に例外処理を追加し、Action 実行時に DataAccessException が発生した場合、エラー画面に遷移させる。（ここでは確認のために、事前に SQL 文を変更し、DAO で DataAccessException が発生する状態にしておく。）

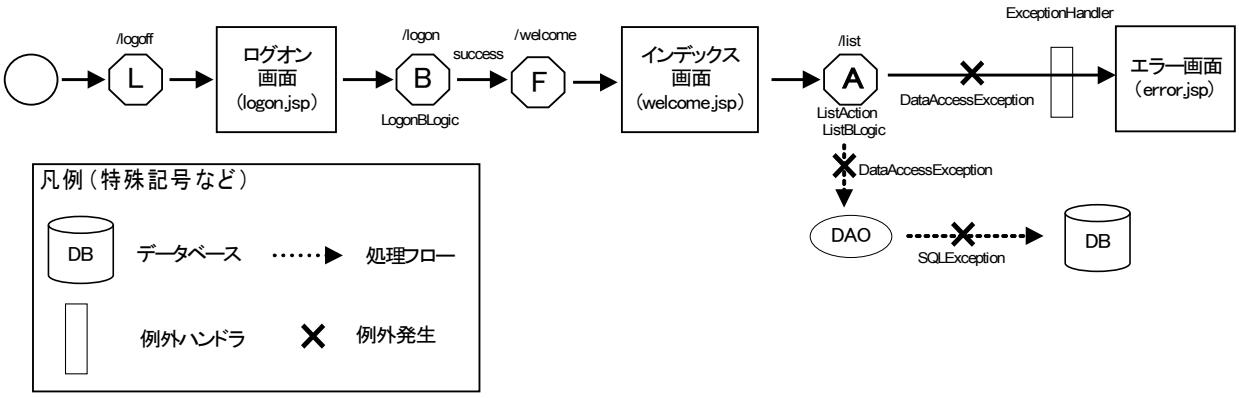


図50　例外処理画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスする。
2. ログオン画面へ遷移する。
3. UserName を入力し、ログオンボタンを押下してインデックス画面へ遷移する。
4. インデックス画面から一覧表示リンクを押下する。
5. DataAccessException が発生するので、エラー画面に遷移する。

このとき、以下の作業が必要となる。

- JSP ファイルの作成
- “application-messages.properties”の変更
- “sqlMap.xml”ファイルの変更
- “struts-config.xml”ファイルの変更

■ **手順**

● **JSP ファイルの作成**

エラー時の遷移先となるページを作成する。
ここでは既に用意されているので、“terasoluna-spring-thin-blank/webapps/error.jsp”ファイルが以下のように作成されていることを確認する。

```
<%@ page contentType="text/html; charset=Windows-31J"%>
<%@ taglib uri="/struts-html" prefix="html" %>
<%@ taglib uri="/struts-bean" prefix="bean" %>
<%@ taglib uri="/struts-logic" prefix="logic" %>
<%@ taglib uri="/terasoluna-struts" prefix="ts" %>
<%@ taglib uri="/terasoluna" prefix="t" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html:html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
<title>Error Page</title>
</head>

<body>
<table border="0" style="width:100%;">
    <tr>
        <td align="center">
            <html:img src="image/terasoluna_logo.gif" />
        </td>
    </tr>
    <tr>
        <td align="center">
            <h1>
                <font color="red"><strong>
                    !! ERROR !!
                </strong></font>
            </h1>
        </td>
    </tr>
</table>

<p>
<html:errors />
</body>
</html:html>
```

- **“application-messages.properties”の変更**

エラーメッセージをメッセージリソースに追加する。
“terasoluna-spring-thin-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)

```
error.system.error=システムエラーが発生しました。
error.data.access.error=データベースのアクセス時に例外が発生しました。
```

- **“sqlMap.xml”ファイルの変更**

DataAccessException を発生させるため、SQL 文を変更する。ここではテーブル名に存在しないテーブル名“EXCEPTIONLIST”を設定し、エラーを発生させる。
“terasoluna-spring-thin-blank/sources/sqlMap.xml”ファイルを以下のように変更する。(網掛け部分を変更する。)

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

- **“struts-config.xml”ファイルの変更**

発生した例外を共通的に扱う設定を行なう。
“terasoluna-spring-thin-blank/webapps/WEB-INF/struts-config.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。)
ここでは、DataAccessException クラス以外に SystemException クラスと Exception クラスについても、あわせて設定している。

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

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>
                                          :
                                         略
                                          :
<!-- ========================================== Global Exception Definitions -->
  <global-exceptions>
    <exception path="/error.jsp"
               key="error.system.error"
               className="jp.terasoluna.fw.web.struts.action.ExceptionConfigEx"
               handler="jp.terasoluna.fw.web.struts.action.SystemExceptionHandler"
               type="jp.terasoluna.fw.exception.SystemException"/>
    <exception path="/error.jsp"
               key="error.data.access.error"
               className="jp.terasoluna.fw.web.struts.action.ExceptionConfigEx"
               type="org.springframework.dao.DataAccessException"/>
    <exception path="/error.jsp"
               key="error.system.error"
               className="jp.terasoluna.fw.web.struts.action.ExceptionConfigEx"
               type="java.lang.Exception"/>
  </global-exceptions>
                                          :
                                         略
                                          :
```

■ 確認

指定した例外が発生した場合は、設定にしたがって処理が行なわ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、コンパイルを行なったのちに 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) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/”にアクセスし、ログオン画面が表示されたら、UserName に"12345"を入力し、ログオンボタンを押下する。

図51 ログオン画面

(4) ログオンボタン押下後、下図のようにインデックス画面に遷移する。

図52 インデックス画面

(5) 一覧表示画面リンクを押下する。

(6) 下図のように、システムエラー画面に遷移し、「データベースのアクセス時に例外が発生しました」というエラーメッセージが表示されることを確認する。

図53 システムエラー画面

(7) 確認後、DataAccessException を発生させるために変更した“sqlMap.xml”ファイルの中の EXCEPTIONLIST を USERLIST に戻す。

■ **参考資料**

- Spring 版機能説明書
  『WC-01 Action 実行システム例外処理機能』

2.10　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

下図のように、前節までに作成したログオン機能に次に示すようなアクセス制限を施す。

- ログオンしていない場合に特定のパス(本節では、登録画面)へのアクセスの禁止。

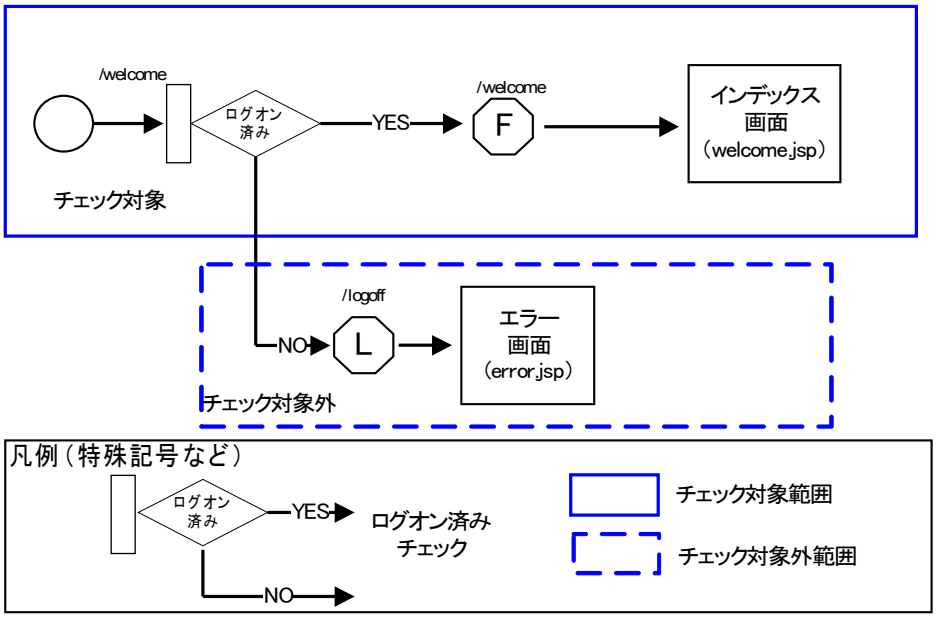


図54　アクセス制御画面遷移図

1. ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/welcome.do”にアクセスする。
2. ログオンしていないので、インデックス画面へは遷移せず、ログオン画面へ遷移する。
3. UserName を入力し、ログオンボタンを押下する。
4. ログオンしているので、インデックス画面へ遷移する。

このとき、以下の作業が必要となる。

- アクセスコントローラクラスの作成
- “applicationContext.xml”ファイルの変更
- “ApplicationResources.properties”ファイルの変更
- web.xml の変更

■ 手順

● アクセスコントローラクラスの作成

認証チェックを行うクラスを作成する。
“terasoluna-spring-thin-blank/sources/jp.terasoluna.thin.tutorial.web/SampleAuthController.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.thin.tutorial.web;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletRequest;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import jp.terasoluna.fw.util.PropertyUtil;
import jp.terasoluna.fw.web.RequestUtil;
import jp.terasoluna.fw.web.thin.AuthenticationController;

/**
 * 認証チェックを行う。
 */
public class SampleAuthController implements AuthenticationController {

    /**
     *  認証チェックを行わないパス情報リストを取得キー。
     */
    private static final String AUTHENTICATED_NOCHECK_PATH_PREFIX
                                = "access.control.authenticated.escape.";

    /**
     *  認証チェックを行わないパス情報リストを取得キー。
     */
    private List<String> noCheckList = null;

    /**
     * リクエストのパス情報に対して、指定された HTTP セッションが
     * 認証済みであるかどうかを判定する。
     *
     * @param pathInfo パス情報
     * @param req HTTP リクエスト
     *
     * @return 認証に成功すれば <code>true</code>
     */
    public boolean isAuthenticated(String pathInfo, ServletRequest req) {

         //セッションから UVO を取得する。
        HttpSession session = ((HttpServletRequest) req).getSession();
        SampleUVO uvo = (SampleUVO) session.getAttribute("USER_VALUE_OBJECT");

        //UVO、または UVO に登録されている ID が null の場合は false を返却する。
        if (uvo != null && uvo.getUserId() != null) {
            return true;
        }
        return false;
    }

    /**
     * パスがチェック対象か否か判定する。
     *
     * @param req 判定対象となる <code>ServletRequest</code> インスタンス
     *
     * @return チェック対象の場合は<code>true</code>
     */
    public boolean isCheckRequired(ServletRequest req) {

        //パス情報を取得する。
        String pathInfo = RequestUtil.getPathInfo(req);

        if (noCheckList == null) {
            noCheckList = new ArrayList<String>();
            for (int i = 1; ; i++) {
                String path = PropertyUtil.getProperty(
                        AUTHENTICATED_NOCHECK_PATH_PREFIX + i);
                if (path == null) {
                    break;
                }
```

```
                noCheckList.add(path);
            }
        }
        for (int cntX = 0; cntX < noCheckList.size(); cntX++) {
            String path = (String) noCheckList.get(cntX);
            if (pathInfo.startsWith(path) || "/".equals(pathInfo)) {
                return false;
            }
        }

        return true;
    }

}
```

- **“applicationContext.xml”ファイルの変更**

アクセスコントローラクラスの Bean 定義を追加する。
“terasoluna-spring-thin-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。)

```
                                        :
                                       略
                                        :
<!-- DAO 定義 -->
  <bean id="queryDAO"
        class="jp.terasoluna.fw.dao.ibatis.QueryDAOiBatisImpl">
    <property name="sqlMapClient"><ref local="sqlMapClient"/></property>
  </bean>
  <bean id="updateDAO"
        class="jp.terasoluna.fw.dao.ibatis.UpdateDAOiBatisImpl">
    <property name="sqlMapClient"><ref local="sqlMapClient"/></property>
  </bean>

<!-- フィルター -->
  <bean id="sampleAuthenticationController"
                class="jp.terasoluna.thin.tutorial.web.SampleAuthController"/>
```

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

認証の対象外とするパスを指定する。
“terasoluna-spring-thin-blank/sources/ApplicationResources.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。)

```
                                        :
                                       略
                                        :
#認証を行わないパスを記述
access.control.authenticated.escape.1=/logon.do
access.control.authenticated.escape.2=/logoff.do
access.control.authenticated.escape.3=/error.do
access.control.authenticated.escape.4=/error.jsp
access.control.authenticated.escape.5=/javascript/InputCalendar.js
access.control.authenticated.escape.6=/stylesheet/BlueStyleInputCalendar.css
access.control.authenticated.escape.7=/stylesheet/WhiteStyleInputCalendar.css
access.control.authenticated.escape.8=/image/terasoluna_logo.gif
access.control.authenticated.escape.9=/image/calendar/BlueStyle/BlueStyle-close.gif
access.control.authenticated.escape.10=/image/calendar/BlueStyle/BlueStyle-drop1.gif
access.control.authenticated.escape.11=/image/calendar/BlueStyle/BlueStyle-drop2.gif
access.control.authenticated.escape.12=/image/calendar/BlueStyle/BlueStyle-left1.gif
access.control.authenticated.escape.13=/image/calendar/BlueStyle/BlueStyle-left2.gif
access.control.authenticated.escape.14=/image/calendar/BlueStyle/BlueStyle-right1.gif
access.control.authenticated.escape.15=/image/calendar/BlueStyle/BlueStyle-right2.gif
access.control.authenticated.escape.16=/image/calendar/WhiteStyle/WhiteStyle-close.gif
access.control.authenticated.escape.17=/image/calendar/WhiteStyle/WhiteStyle-drop1.gif
access.control.authenticated.escape.18=/image/calendar/WhiteStyle/WhiteStyle-drop2.gif
access.control.authenticated.escape.19=/image/calendar/WhiteStyle/WhiteStyle-left1.gif
access.control.authenticated.escape.20=/image/calendar/WhiteStyle/WhiteStyle-left2.gif
access.control.authenticated.escape.21=/image/calendar/WhiteStyle/WhiteStyle-right1.gif
access.control.authenticated.escape.22=/image/calendar/WhiteStyle/WhiteStyle-right2.gif
```

- **“web.xml”の変更**

認証チェックを行う“AuthenticationControlFilter”のマッピングと、認証エラー時の処理を追加する。
"terasoluna-spring-thin-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。)

```
                                        :
                                       略
                                        :
<!-- filter setting -->

  <filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>
      jp.terasoluna.thin.tutorial.web.SetCharacterEncodingFilter
    </filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>Windows-31J</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter>
    <filter-name>AuthenticationControlFilter</filter-name>
    <filter-class>
      jp.terasoluna.fw.web.thin.AuthenticationControlFilter
    </filter-class>
    <init-param>
      <param-name>controller</param-name>
      <param-value>sampleAuthenticationController</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>AuthenticationControlFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

                                        :
                                       略
                                        :
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <error-page>
    <exception-type>
      jp.terasoluna.fw.web.thin.UnauthenticatedException
    </exception-type>
    <location>/logoff.do</location>
  </error-page>

                                        :
                                       略
                                        :
```

■ 確認

実際に、ログオンをせずにチェック対象となるパスにアクセスした場合、認証エラー発生時の処理が行なわ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、コンパイルを行なったのちに Tomcat を起動する。
(2) ブラウザを起動して“http://localhost:8080/terasoluna-spring-thin-blank/welcome.do”(ログオン済みチェック対象のパス)にアクセスする。(前節などで既にブラウザを立ち上げ済みの場合は、ログオフを行ってから上記のパスにアクセスする。)
(3) (2)でアクセスしたパスは、ログオンしていない場合にはアクセスが禁止されているパスなので、ログオン画面へ遷移することを確認する。
(4) ログオン画面を表示するためのパスは、チェック対象外のパスとして指定しているため、下図のようにログオン画面が表示されることを確認する。

図55 ログオン画面表示

(5) ログオン画面では、ユーザ ID 入力欄に“00001”と入力し、ログオンボタンを押下する。
(6) ログオン済みなのでインデックス画面へのアクセスが許可され、インデックス画面へ遷移できることを確認する。

図56 インデックス画面表示

■ 参考資料

- Spring 版機能説明書
  『WA-01 ログオン済みチェック機能』
  『WA-02 アクセス権限チェック機能』

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:J2SDK1.5.0
- Web アプリケーションサーバ:Apache Tomcat 5.5.x (x はバージョン番号)
- データベース:Oracle Database 10g for Windows
- ビルドツール:Apache Ant 1.6.5 以上
- 総合開発環境:Eclipse SDK 3.2.x
- Eclipse plugin :WTP 1.5.x

■ 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、Web ブラウザ対応版は「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\Tomcat 5.5\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：J2SDK1.5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.x (x はバージョン番号)
- データベース：PostgreSQL 8.2.x for Windows
- ビルドツール：Apache Ant 1.6.5 以上総合開発環境：Eclipse SDK 3.2.x
- Eclipse plugin ：WTP 1.5.x

■　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、Web ブラウザ対応版は「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）

② 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。

③ 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

④ 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。

⑤ 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\Tomcat 5.5\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-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=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。

Spring 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

- Spring 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：J2SDK1.5.0
- Webアプリケーションサーバ：Apache Tomcat 5.5.x (xはバージョン番号)
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 以上
- 総合開発環境：Eclipse SDK 3.2.x
- Eclipse plugin ：WTP 1.5.x

■ 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\Tomcat 5.5
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Webブラウザ対応版は“C:\Eclipse\workspace\terasoluna-spring-thin-blank\webapps”、パスに“/terasoluna-spring-thin-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)”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\Tomcat 5.5
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\Tomcat 5.5\conf\Catalina\localhost”フォルダ配下にある、コンテキスト設定ファイル(Web ブラウザ対応版なら“terasoluna-spring-thin-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-thin-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-thin-blank" path="/terasoluna-spring-thin-blan
k" reloadable="true" source="org.eclipse.jst.j2ee.server:terasoluna-spring-thin-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\Tomcat 5.5\conf\Catalina\localhost”フォルダ配下にある、コンテキスト設定ファイル（Web ブラウザ対応版なら“terasoluna-spring-thin-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-thin-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、Spring 版機能説明書『CC-01 JNDI アクセス機能』参照のこと。

Web ブラウザ対応版では、「2．6 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、“applicationContext.xml”ファイルの TerasolunaDataSource の Bean 定義については、以下のようにすること。

```
  <bean id="TerasolunaDataSource"
      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：J2SDK1.5.0
- 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 以上
- 総合開発環境：Eclipse SDK 3.2.x
- Eclipse plugin ：WTP 1.5.x

■ 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、Web ブラウザ対応版は「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、Spring **版機能説明書**『CC-01 JNDI アクセス機能』参照のこと。

Web ブラウザ対応版では、“applicationContext.xml”ファイルの TerasolunaDataSource の Bean 定義については、以下のようにすること。

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

  <!--
  <bean id="TerasolunaDataSource"
        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。

Web ブラウザ対応版

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

<weblogic-web-app>
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>
```

(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 Web ブラウザ対応版はブラウザに http://localhost:7001/<context.name>/を入力し確認する。

■ 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。

- Spring 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：J2SDK1.5.0
- 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 以上
- 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、Web ブラウザ対応版は「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、Spring **版機能説明書**『CC-01 JNDI アクセス機能』参照のこと。

Web ブラウザ対応版では、“applicationContext.xml”ファイルの TerasolunaDataSource の Bean 定義については、以下のようにすること。

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

  <!--
  <bean id="TerasolunaDataSource"
        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>
  -->
```

(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 Web ブラウザ対応版はブラウザに http://localhost:9080/<context.name>/を入力し確認する。

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：J2SDK1.5.0
- 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 以上
- 総合開発環境：Eclipse SDK 3.2.x
- Eclipse plugin ：WTP 1.5.x

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」ー「ビューの表示」ー「その他」メニューを選択する。
⑧ 「サーバー」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 Web ブラウザ対応版はブラウザに http://localhost:7001/<context.name>/を入力し確認する。

■ 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。

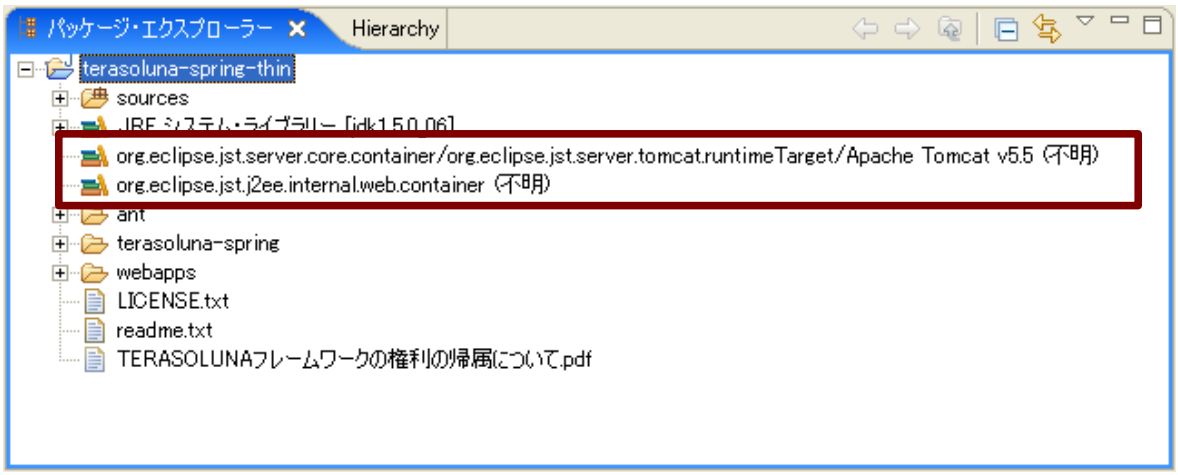
■ 想定環境

本書では以下の開発環境を想定して、解説している。

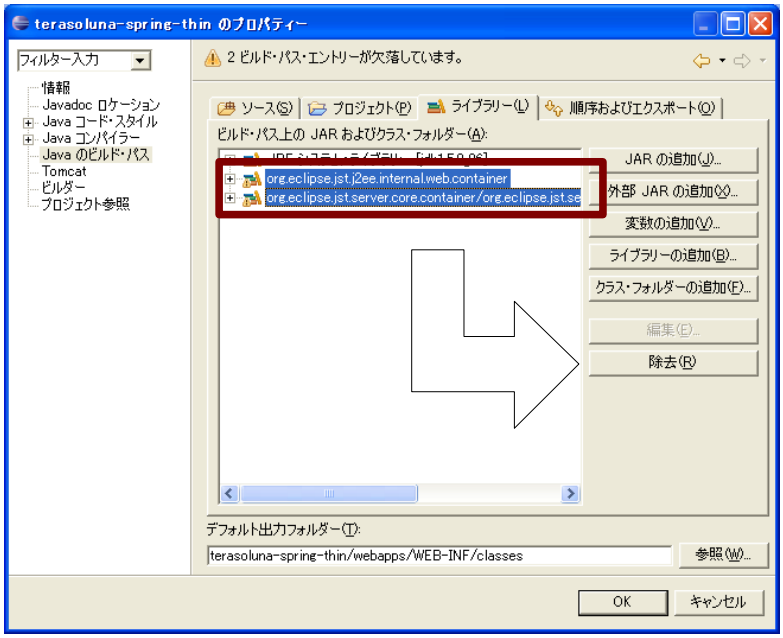
- OS：Microsoft Windows 2000 / XP Professional
- JDK：J2SDK1.5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.x (x はバージョン番号)
- ビルドツール：Apache Ant 1.6.5 以上
- 総合開発環境：Eclipse SDK 3.2.x
- Eclipse plugin ：Sysdeo Eclipse Tomcat Launcher plugin 3.1beta

■ 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。



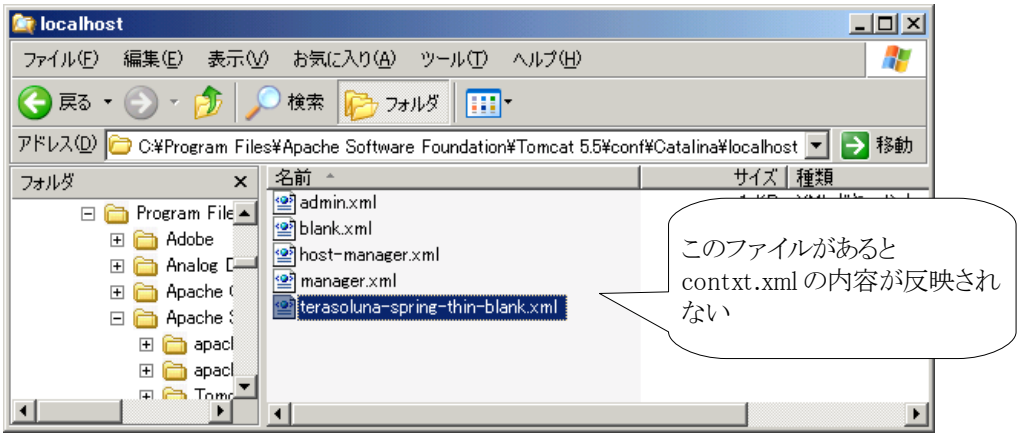
③ 続けて必要なライブラリをビルド・パスに追加する。プロジェクトの WEB-INF/lib に配置した JAR ファイルおよび TOMCAT_HOME/common/lib に存在する JAR ファイルを追加する。

■ 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：J2SDK1.5.0
- Web アプリケーションサーバ：Apache Tomcat 5.5.x (x はバージョン番号)
- データベース：Oracle Database 10g for Windows
- ビルドツール：Apache Ant 1.6.5 以上
- 総合開発環境：Eclipse SDK 3.2.x
- Eclipse plugin ：WTP 1.5.x

■　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。

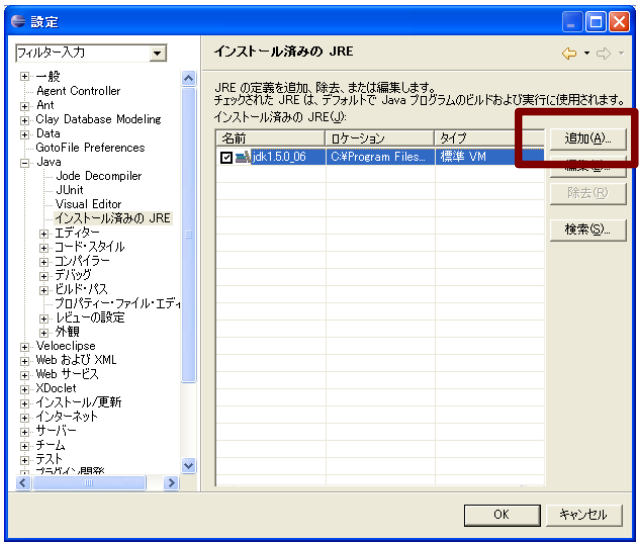
① 「インストール済み JRE」の設定の変更
② 「インストール済みサーバ・ランタイム」の JRE の設定の変更
③ 「Projects Facets」のランタイムの JRE の設定の変更

※手順は上記番号順に行うこと

■　手順

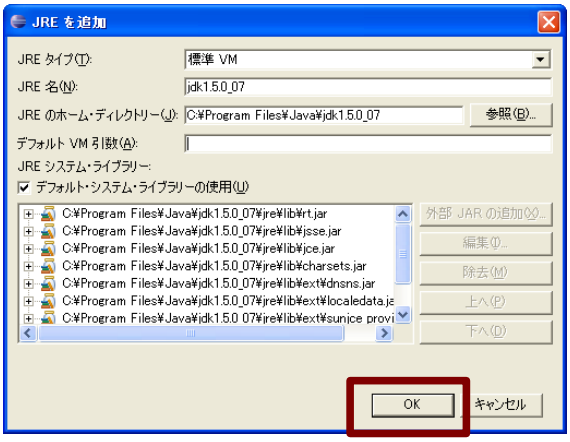
① 「インストール済み JRE」の設定の変更

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。

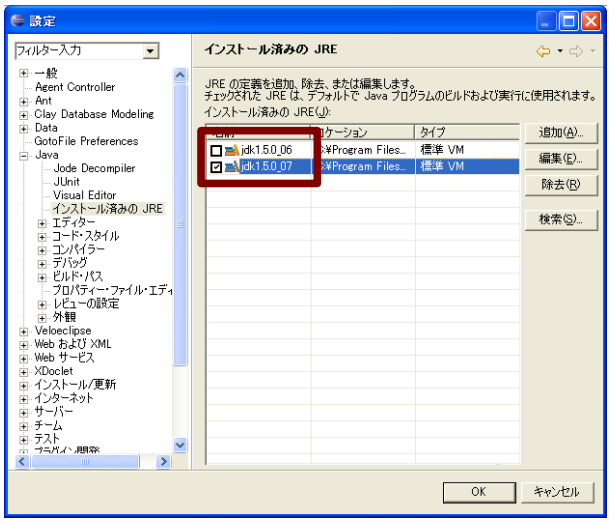


3. 表示された「JRE を追加」の設定ウィンドウより以下の内容を入力し、「OK」ボタンを押下する。

- 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



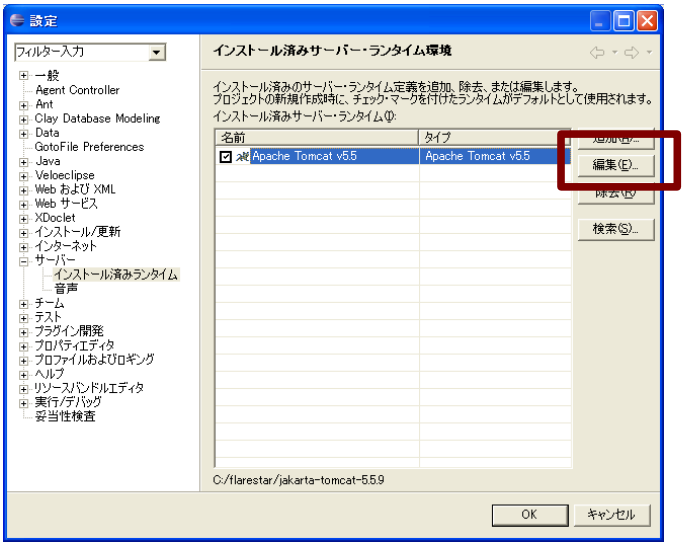
4. 新しく追加した JRE 名にチェックを入れ、「OK」ボタンを押下する。



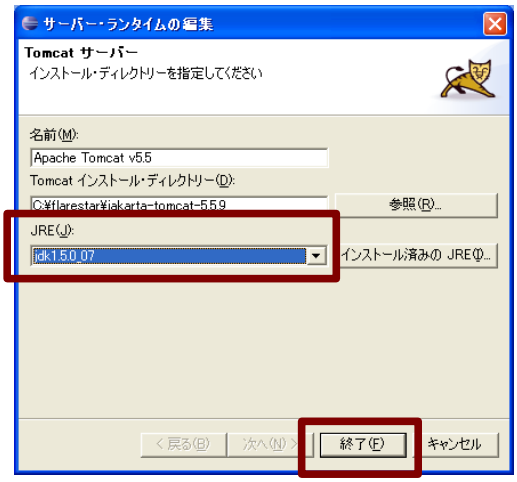
② 「インストール済みサーバ・ランタイム」の JRE の設定の変更

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。

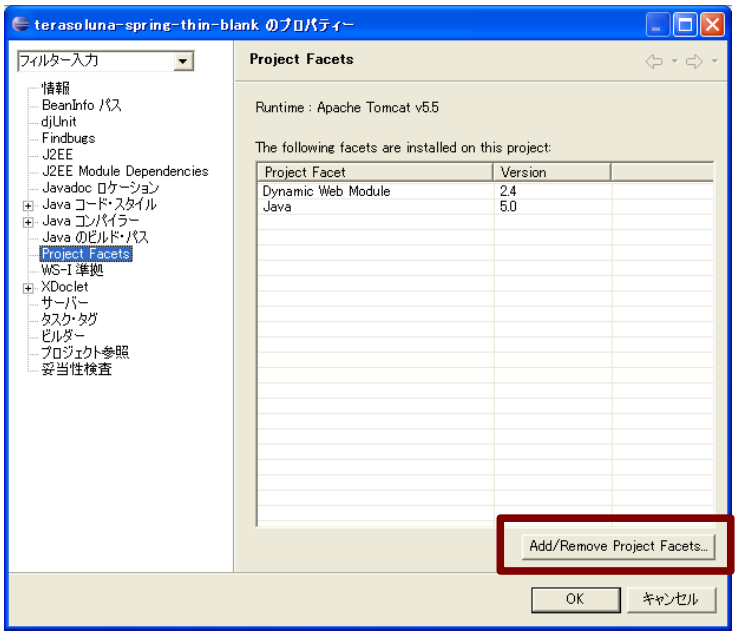


4. 「OK」ボタンを押下し、設定ウィンドウを閉じる。

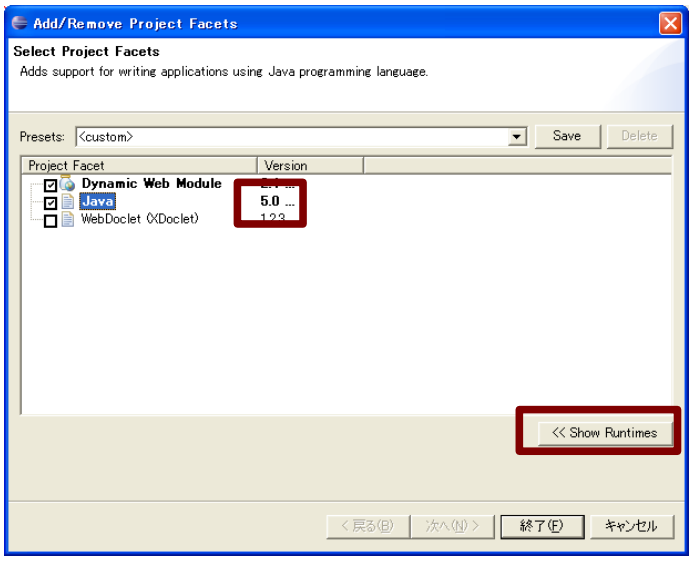
③ 「Projects Facets」のランタイムの JRE の設定の変更

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。

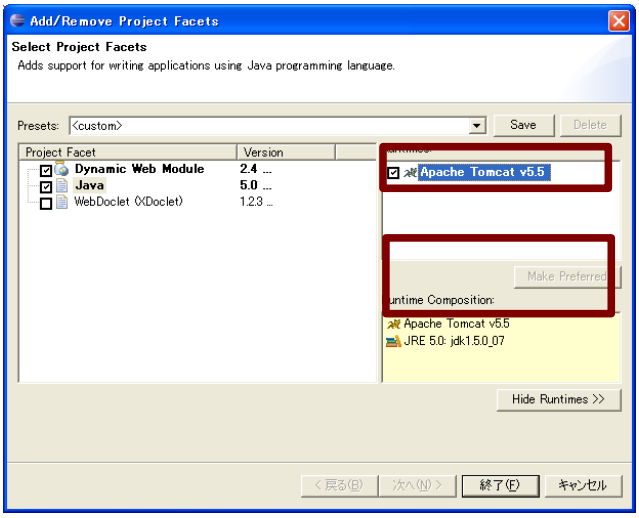
1. 「パッケージ・エクスプローラ」より該当のプロジェクトを選択し、右クリックメニューより「プロパティ」を選択する。
2. 「Projects Facets」を選択し、「Add/Remove Projects Facets」ボタンを押下する。



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。