AxiZ
Academy eXtend IT skill Zero

実践 Spring Framework + MyBatis

Spring Framework – Spring Web MVC 編

ver. 1.2.2

TC. Techno-core Corporation 株式会社テクノコア
http://www.techno-core.jp

はじめに

Spring Web MVC フレームワークとは?

Spring Web MVC(以降Spring MVCと記す)フレームワークとは、MVCアーキテクチャでWebアプリケーションの開発を行うために Spring Framework が提供している、プレゼンテーション層をサポートするフレームワークです。

MVCアーキテクチャでプレゼンテーション層をサポートするフレームワークとしては Struts が有名ですが、Spring Framework も Spring MVC によってプレゼンテーション層をサポートすることが可能となっています。

また、仕組みは Struts 1.x系と非常によく似ているため、比較的違和感なく受け入れられると思います。
そして、Spring MVC の方が後発であり、また同じSpring Framework の DIコンテナと組み合わせることでよりシンプルな実装が可能となっています。

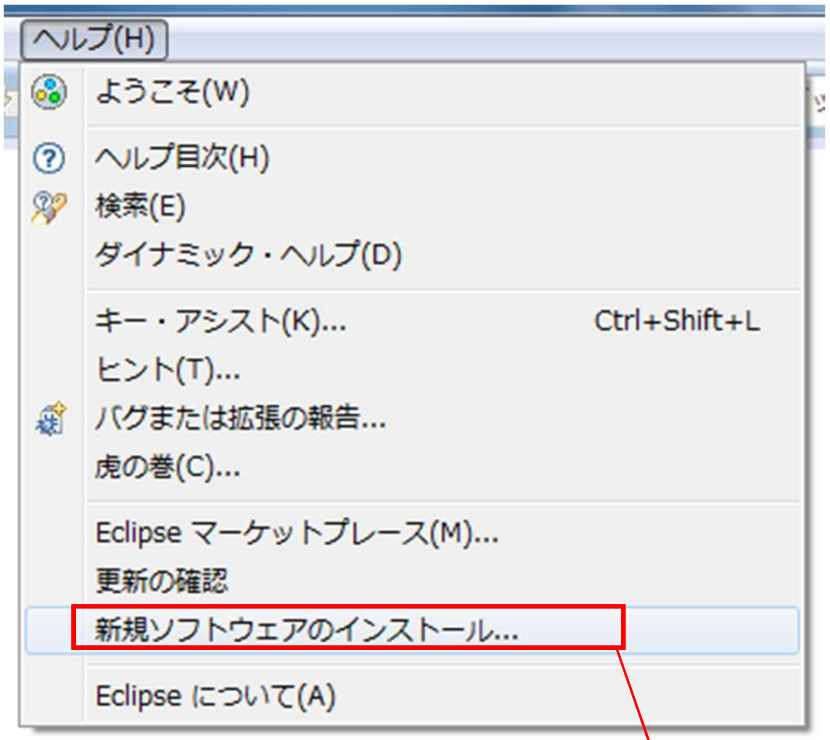
今回は、APサーバとして Apache Tomcat を採用し、また開発ツールには STS(Springsource-Tool-Suite)を使用して実装を進めていきます。
STS を Eclipse に組み込むことで、非常に効率的な開発が可能となります。

☆今回の開発環境は、以下の構成で構築します。

【APサーバ】 Tomcat7.0.30
【Eclipseプラグイン】 Springsource-Tool-Suite(STS) 3.2.0

開発環境構築

STS(Springsource-Tool-Suite) の導入

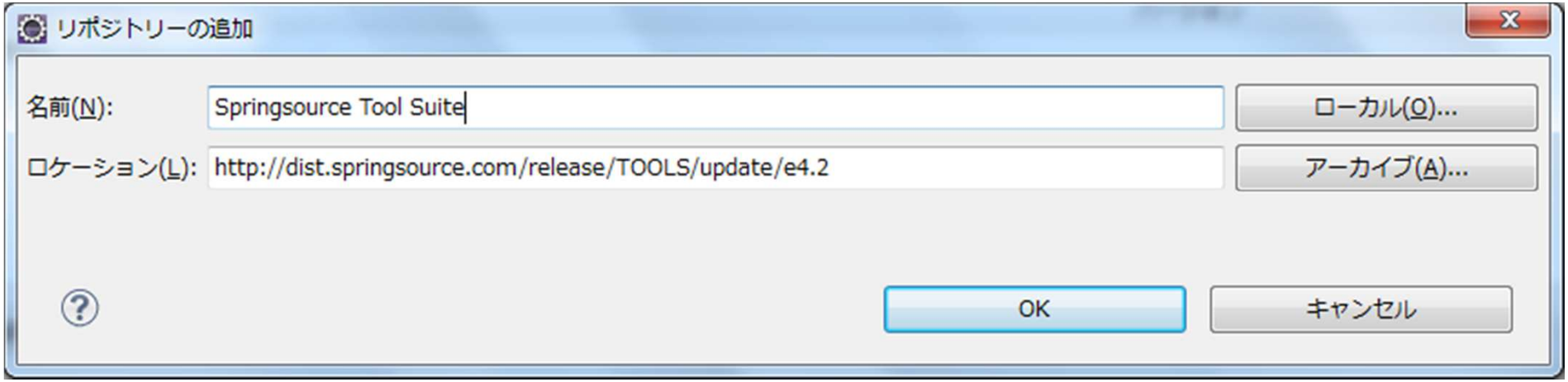


ヘルプメニューから新規ソフトウェアのインストールを選択

開発環境構築

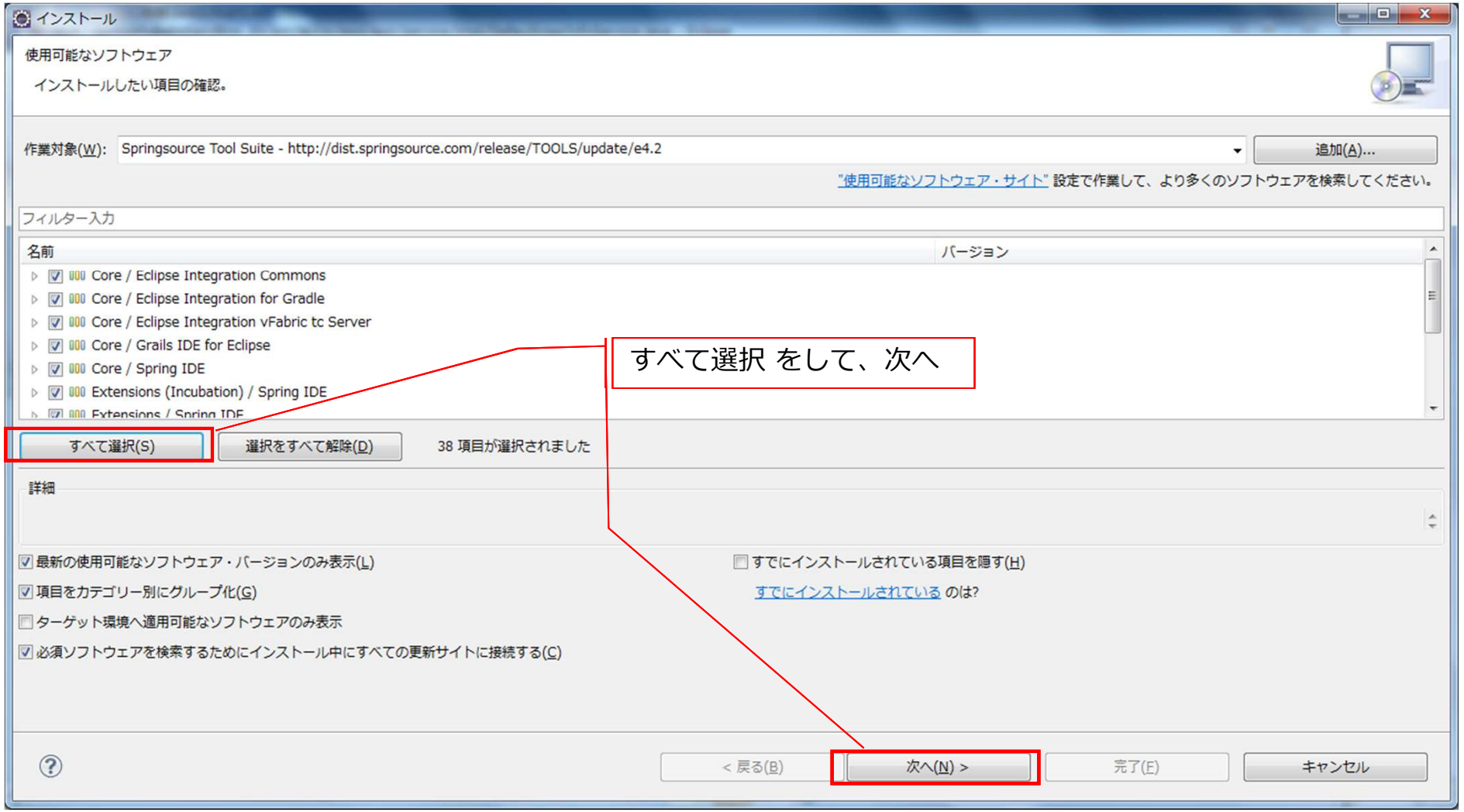
STS(Springsource-Tool-Suite) の導入

作業対象を新しく追加。名前は任意の名前を入力する。
ロケーションは http://dist.springsource.com/release/TOOLS/update/e4.2



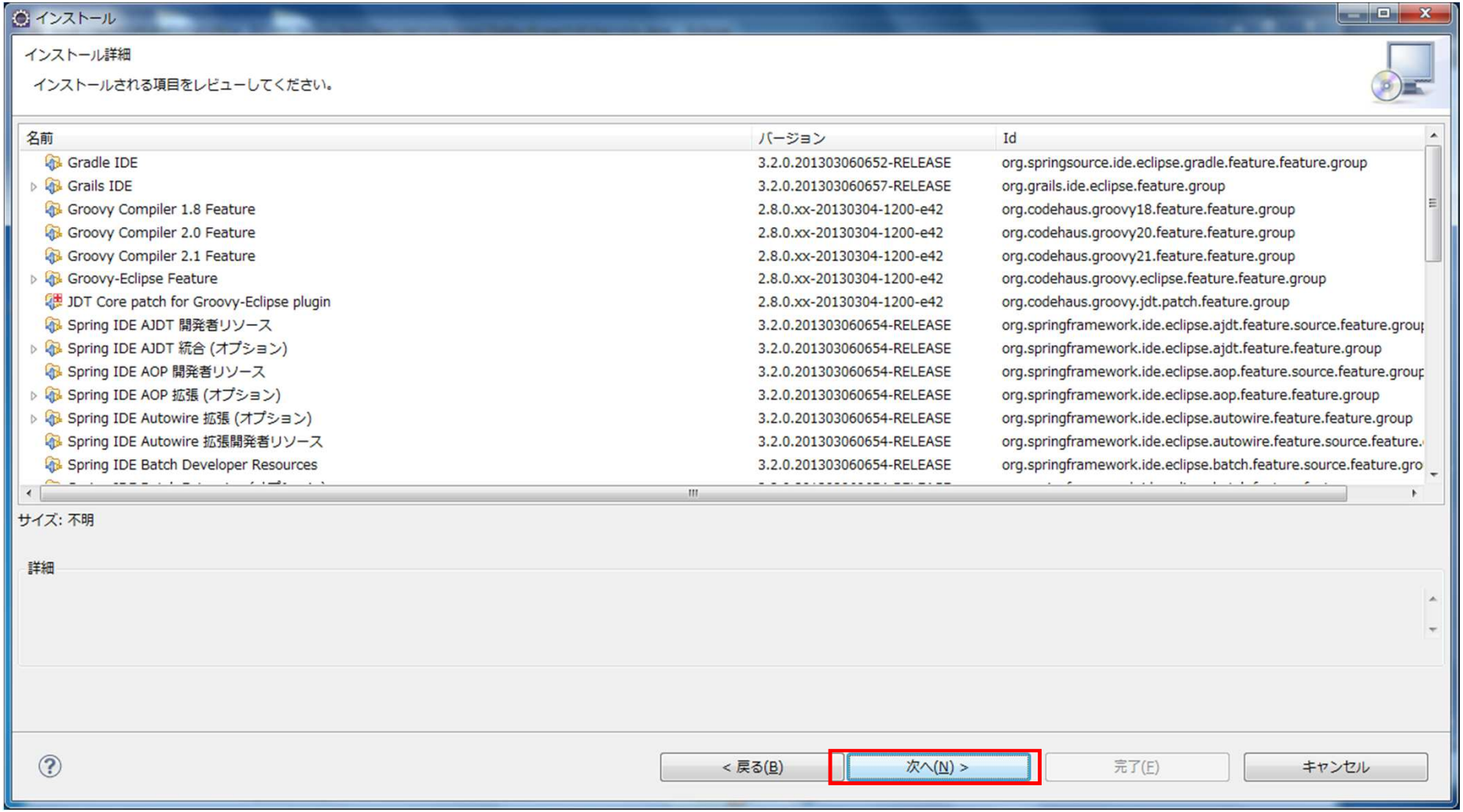
開発環境構築

STS(Springsource-Tool-Suite) の導入



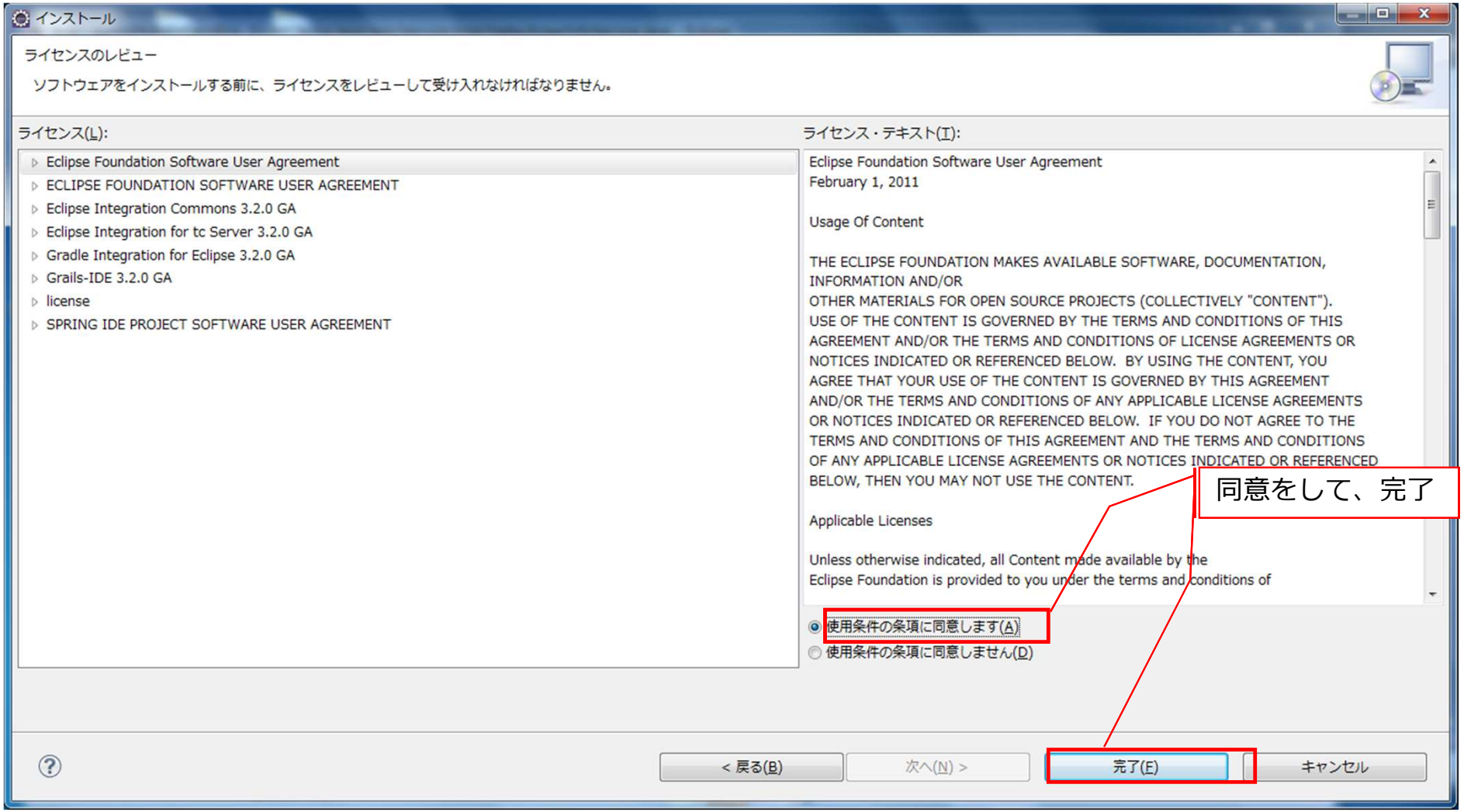
開発環境構築

STS(Springsource-Tool-Suite) の導入



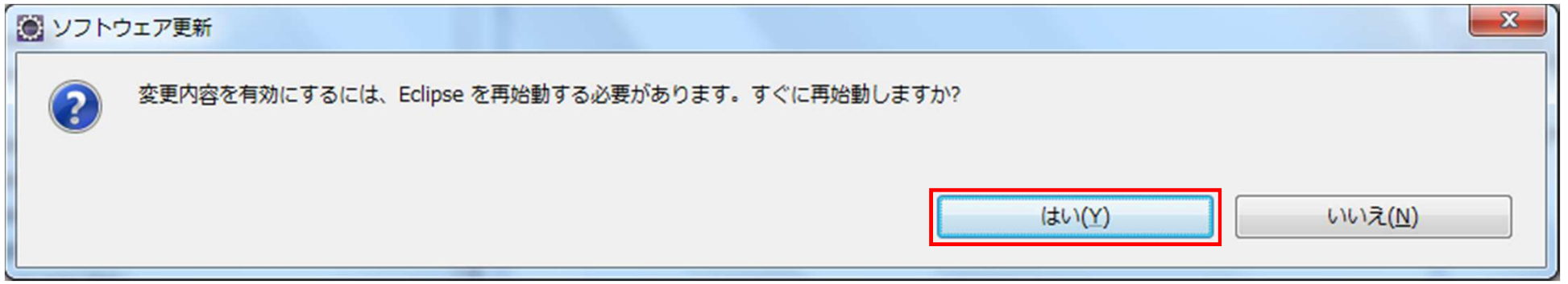
開発環境構築

STS(Springsource-Tool-Suite) の導入



開発環境構築

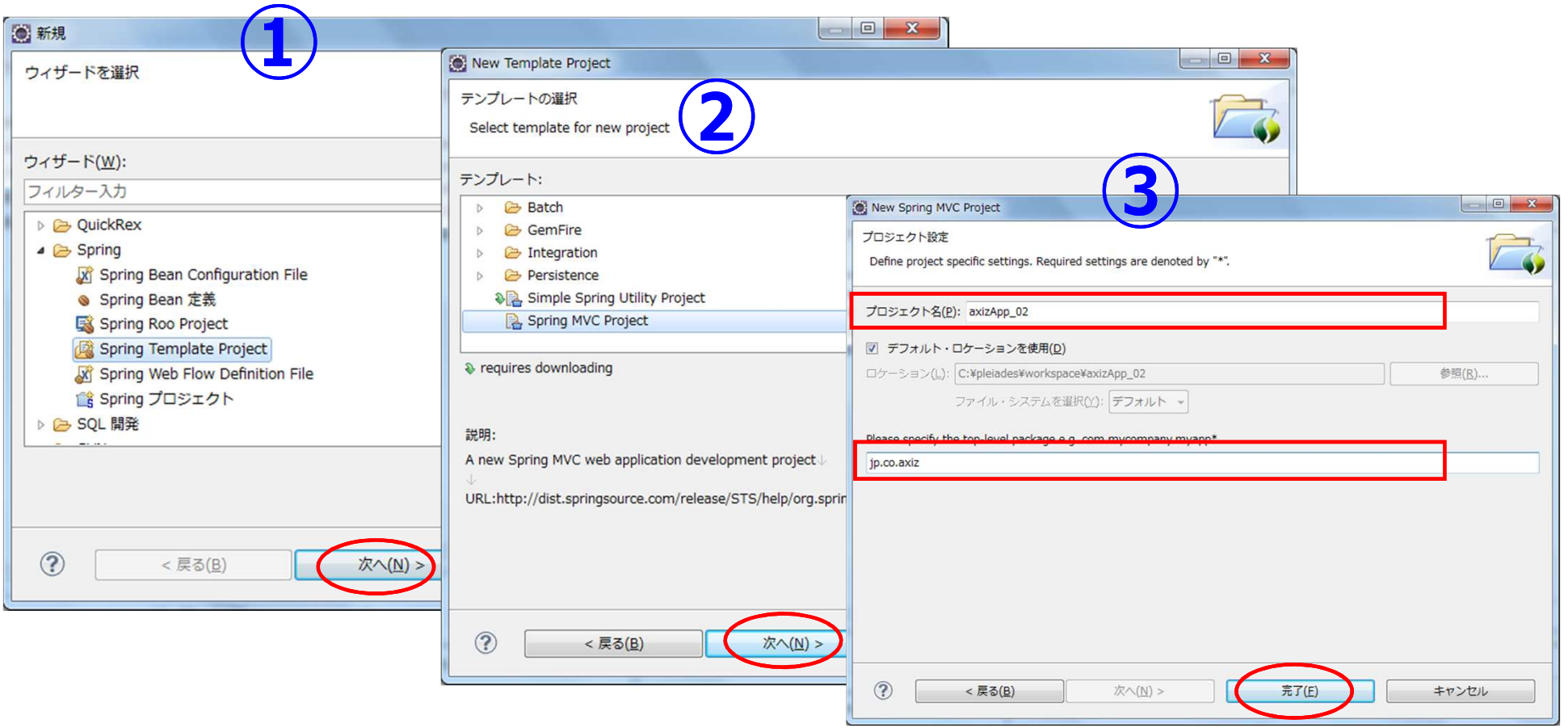
STS(Springsource-Tool-Suite) の導入



開発環境構築

開発用プロジェクト作成 1/4

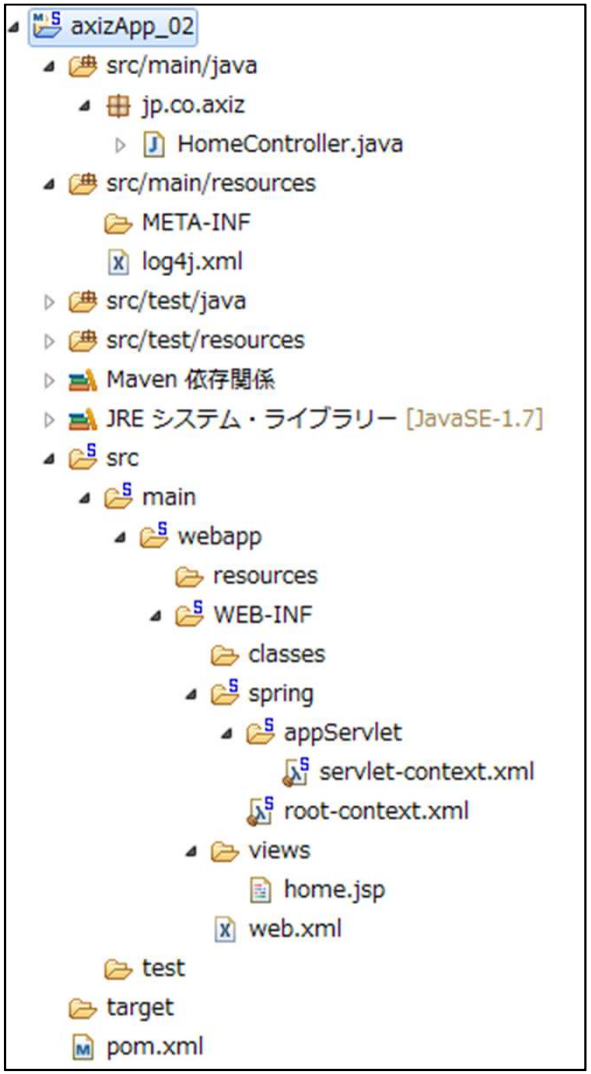
Eclipseのパッケージ･エクスプローラビュー内で右クリック -> 新規 -> プロジェクト を選択し、以下のウィザードにてプロジェクトを作成します。ここではプロジェクト名をaxizApp_02、パッケージをjp.co.axizとしています。



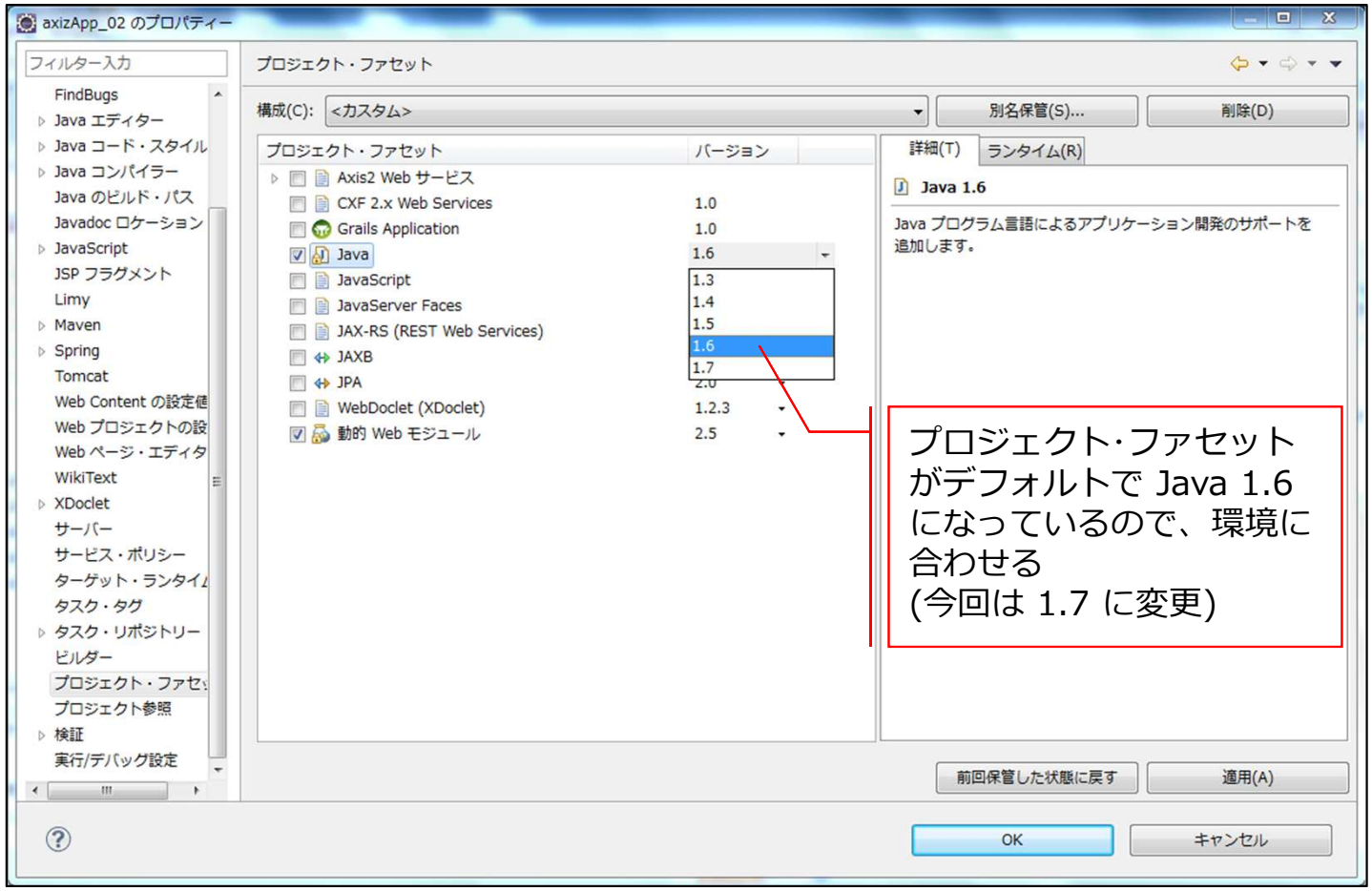
開発環境構築

開発用プロジェクト作成 2/4

以下の構成にてSpring MVC用プロジェクトが生成されます。



次に、プロジェクトを右クリック -> プロパティ を選択します。



開発環境構築

開発用プロジェクト作成 2/4

また、バージョン違いによりエラーが出ていると思いますので、pom.xmlを修正します。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>jp.co</groupId>
    <artifactId>axiz</artifactId>
    <name>axizApp_02</name>
    <packaging>war</packaging>
    <version>1.0.0-BUILD-SNAPSHOT</version>
    <properties>
        <java-version>1.7</java-version>
        <org.springframework-version>3.2.2.RELEASE</org.springframework-version>
        <org.aspectj-version>1.6.10</org.aspectj-version>
        <org.slf4j-version>1.6.6</org.slf4j-version>
    </properties>

    ..
```

開発環境構築

開発用プロジェクト作成 3/4

ライブラリの依存関係は Maven2 を使用しています。
今まで使用してきた環境とライブラリのバージョンを合わせるため、pom.xml を修正します。
修正点を朱書きで記します。
pom.xml を保存すると、必要なライブラリがダウンロードされ、自動でビルド･パスに追加されます。

【pom.xml】

```
<properties>
    <java-version>1.7</java-version>
    <org.springframework-version>3.2.2.RELEASE</org.springframework-version>
    <org.aspectj-version>1.6.10</org.aspectj-version>
    <org.slf4j-version>1.6.6</org.slf4j-version>
    <org.mybatis-version>3.2.2</org.mybatis-version>
    <org.mybatis.spring-version>1.2.0</org.mybatis.spring-version>
    <org.aspectj.weaver-version>1.6.12</org.aspectj.weaver-version>
    <mysql-connector-java-version>5.1.24</mysql-connector-java-version>
</properties>
```

開発環境構築

開発用プロジェクト作成 3/4

```
<!-- Spring -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${org.springframework-version}</version>
    <exclusions>
        <!-- Exclude Commons Logging in favor of SLF4j -->
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
         </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

```
<!-- MyBatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>${org.mybatis-version}</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>${org.mybatis.spring-version}</version>
</dependency>

<!-- MySQL -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql-connector-java-version}</version>
</dependency>

<!-- AspectJ -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${org.aspectj.weaver-version}</version>
</dependency>
```

開発環境構築

開発用プロジェクト作成 4/4

・src/main/resourcesソースフォルダ直下に、
「MyBatis−Spring Framework連携編」で使用したlog4j.xmlを配置します

Spring Web MVC

View (ログイン画面)

では、実際にログインモジュールを例にしてSpring MVCでの実装をしていきます。

まずは、コンフィギュレーション(設定)をベースとした実装手法から紹介します。

最初に、ログイン画面をJSPで用意します。

(ソースは次のページです)

配置場所は

src/main/webapp/WEB-INF/views/login.jsp

とします。

☆配置後は以下となります。

Spring Web MVC

View (ログイン画面)

【login.jsp】

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c"    uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8" />
    <title>Login</title>
  </head>
  <body>
    <h1>ログイン画面</h1>
    <%-- ログイン失敗メッセージ --%>
    <c:if test="${not empty resultMsg}">
      <p><c:out value="${resultMsg}" /></p>
    </c:if>
    <form:form modelAttribute="loginForm" action="doLogin" method="post">
      <%-- ユーザID&パスワード入力エリア --%>
      <table border="0">
        <tr>
          <td align="right">ユーザID:</td>
          <td><form:input path="userId" maxlength="12" /></td>
        </tr>
        <tr>
          <td align="right">パスワード:</td>
          <td><form:password path="userPwd" maxlength="12" /></td>
        </tr>
        <tr>
          <td align="center" colspan="2">
          <%-- ボタンエリア --%>
          <table style="margin-top: 10px" border="0">
            <tr><td><form:button>ログイン</form:button></td></tr>
          </table>
          </td>
        </tr>
      </table>
    </form:form>
  </body>
</html>
```

Form (ログイン画面入力値)

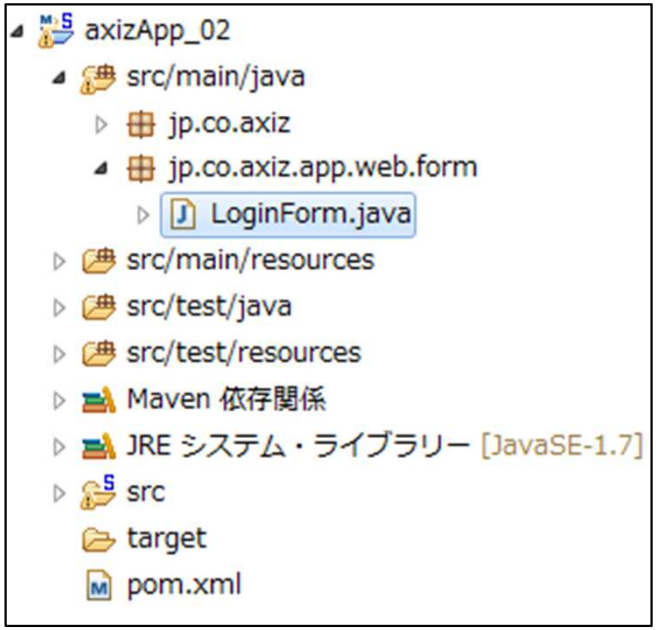
次に、ログイン画面の入力値を保持するFormクラスを用意します。

配置場所は
src/main/java 配下の
jp.co.axiz.app.web.form パッケージとします。

【LoginForm.java】

```
package jp.co.axiz.app.web.form;

/**
 * ログイン画面入力値を保持するクラス。
 *
 * @author {@code AxiZ} t.matsumoto
 */
public class LoginForm {

    /** ユーザ{@code ID}入力値を保持します。 */
    private String userId;
    /** パスワード入力値を保持します。 */
    private String userPwd;

    // TODO: 各アクセサを実装すること.
}
```

☆配置後は以下となります。



Spring Web MVC

Controller (ログイン画面初期表示)

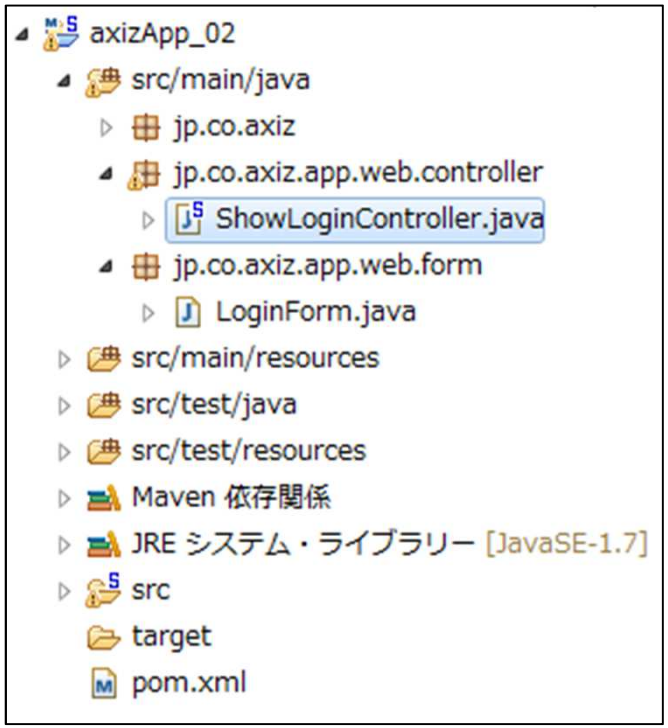
ログイン画面(JSP)と入力値格納Formのペアを作成しました。
では、このログイン画面を表示するControllerクラスを実装します。
＊ org.springframework.web.servlet.mvc.Controller の実装クラスとします

配置場所は src/main/java 配下の jp.co.axiz.app.web.controller パッケージとします。

【ShowLoginController.java】

```
package jp.co.axiz.app.web.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jp.co.axiz.app.web.form.LoginForm;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

/**
 * ログイン画面表示{@code Controller}クラス。
 * @author {@code AxiZ} t.matsumoto
 * @see Controller
 */
public class ShowLoginController implements Controller {

    /**
     * @see Controller#handleRequest(HttpServletRequest, HttpServletResponse)
     */
    @Override
    public ModelAndView handleRequest(HttpServletRequest req, HttpServletResponse res) throws Exception {
        final ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("login");
        modelAndView.addObject("loginForm", new LoginForm());
        return modelAndView;
    }
}
```

☆配置後は以下となります。



Spring Web MVC

Controller (ログイン処理 & 結果に基づいた画面遷移)

次に、ログインボタン押下時にログイン処理を行い、次画面に遷移させる Controllerクラスを実装します。

今回の処理ではログイン画面入力値(LoginForm)を受け取るため、org.springframework.web.servlet.mvc.SimpleFormController のサブクラスとします。

配置場所は、同じく jp.co.axiz.app.web.controller パッケージとします。

【DoLoginController.java】

```
package jp.co.axiz.app.web.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jp.co.axiz.app.web.form.LoginForm;

import org.springframework.util.StringUtils;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;

/**
 * ログイン処理{@code Controller}クラス。
 *
 * @author {@code AxiZ} t.matsumoto
 * @see SimpleFormController
 */
public class DoLoginController extends SimpleFormController {

    /**
     * @see SimpleFormController#onSubmit(HttpServletRequest, HttpServletResponse, Object, BindException)
     */
    @Override
    public ModelAndView onSubmit(HttpServletRequest req, HttpServletResponse res,
                                 Object form, BindException errors) throws Exception {
        final LoginForm loginForm = (LoginForm) form;
        final ModelAndView modelAndView = new ModelAndView();

        // ユーザIDとパスワードが入力されていればログイン可とする.
        if (StringUtils.hasLength(loginForm.getUserId()) &&
            StringUtils.hasLength(loginForm.getUserPwd())) {
            modelAndView.addObject("resultMsg", loginForm.getUserId() + " さんがログインしました.");
            modelAndView.setViewName("top");
        } else {
            modelAndView.addObject("resultMsg", "ログインできません.");
            modelAndView.addObject("loginForm", loginForm);
            modelAndView.setViewName("login");
        }

        return modelAndView;
    }
}
```

Spring Web MVC

View (トップ画面)

最後に、ログイン成功後に表示するトップ画面を用意します。

配置場所はログイン画面同様
src/main/webapp/WEB-INF/views/top.jsp
とします。

【top.jsp】

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8" />
    <title>Top</title>
  </head>
  <body>
    <h1>トップ画面</h1>
    <%-- ログイン結果メッセージ --%>
    <c:out value="${resultMsg}" />
  </body>
</html>
```

☆配置後は以下となります。

Spring Web MVC

サーブレット設定ファイル(servlet-context.xml)

ここまでで、ログイン画面 -> ログイン処理 -> トップ画面と遷移するログインモジュールを実装しました(現状では DoLoginController内の処理だけで可否判定している簡易版ですが)。このログイン関連のソースを関連付けてSpring MVCで駆動させてみます。

関連付ける設定は、src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml で行います(ファイル名は任意です)。 既存の servlet-context.xml に朱書き部分を追記してください。

【servlet-context.xml】

```
    <context:component-scan base-package="jp.co.axiz" />

    <!-- ========================================================= HandlerMapping -->
     <beans:bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
         <beans:property name="mappings">
             <beans:props>
                 <beans:prop key="/">showLoginController</beans:prop>
                 <beans:prop key="/login">showLoginController</beans:prop>
                 <beans:prop key="/doLogin">doLoginController</beans:prop>
             </beans:props>
         </beans:property>
     </beans:bean>

     <!-- ========================================== Spring MVC Controllers -->

     <!-- ログイン画面表示Controller -->
     <beans:bean id="showLoginController" class="jp.co.axiz.app.web.controller.ShowLoginController" />

     <!-- ログイン処理Controller -->
     <beans:bean id="doLoginController"   class="jp.co.axiz.app.web.controller.DoLoginController">
         <beans:property name="commandClass" value="jp.co.axiz.app.web.form.LoginForm" />
     </beans:bean>

</beans:beans>
```

動作確認

Tomcatが起動したら、 http://localhost:8080/axiz/login にアクセスし、ログイン画面が表示されれば成功です。

【初期表示】

ログイン押下

今回の(簡易)ログインモジュールは

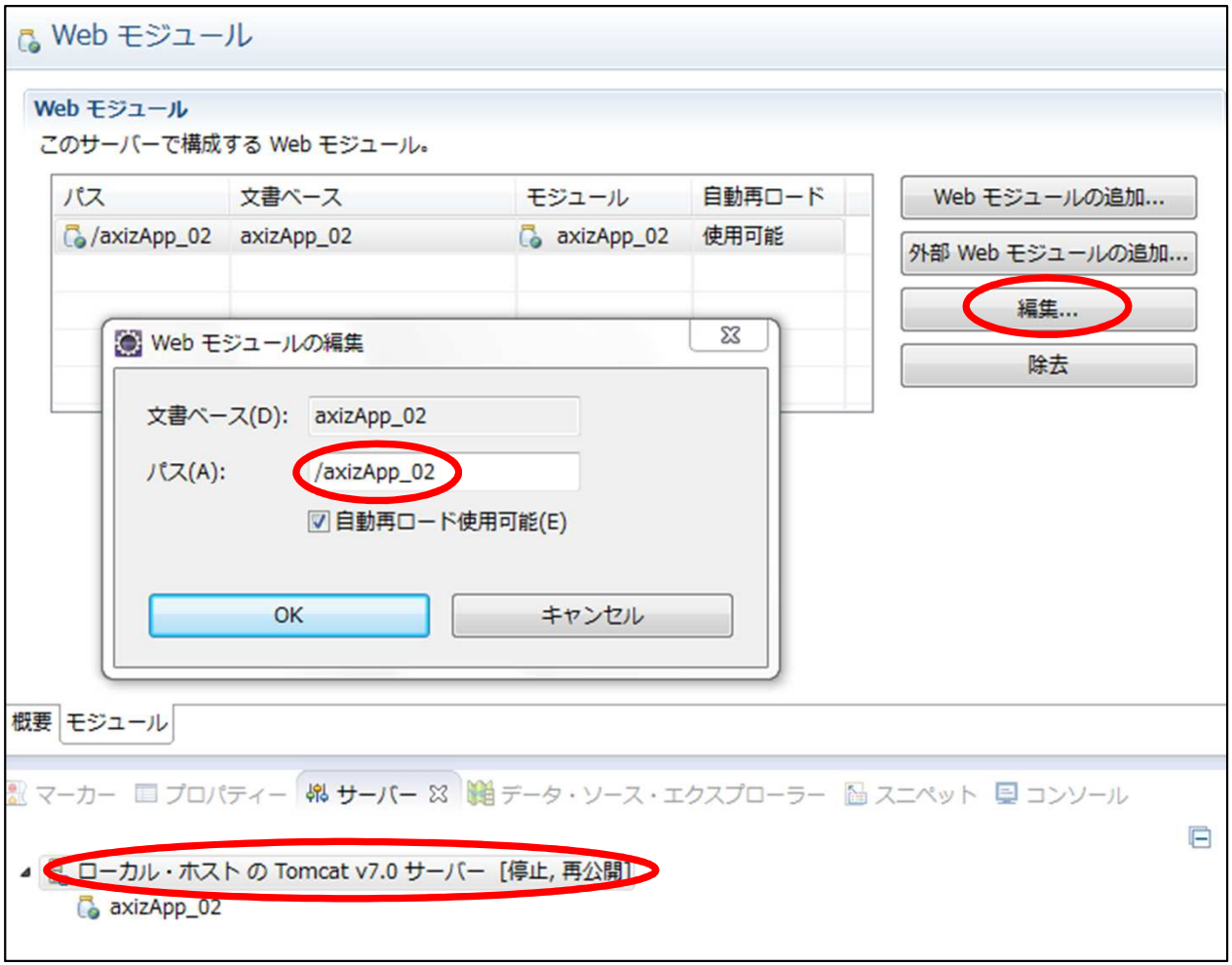
・ユーザIDとパスワードが入力されていればログイン許容
　(入力値はなんでも可)
・どちらか一方でも未入力であればログイン拒否

の実装となっています

ログイン押下

パスの変更

アクセスするURL・パスを変更したい場合には、サーバーのプロパティを開き、変更したいモジュールを選択、編集...ボタンを選択後、パスを書き換えることで、変更することができます。



CharacterEncodingFilter

Webアプリケーションに開発において、常に意識しなければならない問題に「文字化け」が存在します。
今回のログインモジュールでは、この文字化け対策がなされていないため、入力フォームに日本語等のマルチバイト文字が入力されると文字化けが発生します。

Spring Frameworkの CharacterEncodingFilter を利用することで、この問題に対応することができます。
具体的には src/main/webapp/WEB-INF/web.xml に朱書き部分を追記します。

【web.xml】

```
  <!-- ==================================== CharacterEncodingFilter -->
  <filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
      <param-name>forceEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>

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

</web-app>
```

Spring Web MVC

SimpleFormController に関して

まずは、コンフィグレーション(設定)をベースとしたWebアプリケーションの実装を行いました。

Controllerクラスは org.springframework.web.servlet.mvc.Controller の実装クラスに、またはFormを受け取るためには org.springframework.web.servlet.mvc.SimpleFormController のサブクラスにしています。

しかし注意が必要なのは、SimpleFormController(やAbstractFormController) は Spring Framework 3.0から @deprecated(非推奨) となっていることです。

＊いずれライブラリから削除されるかもしれません

Spring MVC はアノテーションをベースとした実装モデルもサポートされています。

次は、このアノテーションによるSpring MVC Webアプリケーションの実装方法を紹介していきます。

Spring Web MVC

ControllerClassNameHandlerMapping Memo

Spring MVC に用意されている HandlerMapping は SimpleUrlHandlerMapping だけではありません。
その他の代表的なものとしては、ControllerClassNameHandlerMapping が挙げられます。
これは、Controllerクラス名を基にした URLパスが、対応する Controllerオブジェクトにマッピングされます。

以下に簡単な実装例を紹介します。

① servlet-context.xml で、ControllerClassNameHandlerMapping を定義します。

```
<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" />
```

② 以下の Controllerクラスが存在するとします。

```
@Controller
public class SampleController {
    @RequestMapping
    public String showFoo() {
        return "foo";
    }
}
```

③ src/main/webapp/WEB-INF/views に、
foo.jsp が存在するとします。

左記 ①～③ により、以下のマッピングが適用されます。

- URLパス「/sample」 と SampleController オブジェクト
- URLパス「/sample/*」 と SampleController オブジェクト

つまり、

http://localhost:8080/axizApp_02/sample
http://localhost:8080/axizApp_02/sample/
http://localhost:8080/axizApp_02/sample/任意の文字列

いずれのケースも SampleController#showFoo() メソッドが実行され、
foo.jsp が Webブラウザに画面表示されます。

アノテーションによる開発

開発用プロジェクト(axizApp_03)を新規に作成します。

以下の項目は先程のソースを流用して構いません。

- pom.xml
- log4j.xml
- web.xml
- top.jsp
- login.jsp
- LoginForm.java

テンプレートで生成される HomeController.java と home.jsp は不要なため、削除してください。

login.jspの17行目を下記のように修正してください。

<form:form modelAttribute="loginForm" action="*login*" method="post">

アノテーションによるController 1/3

先程は、「ログイン画面表示Controller」と「ログイン処理Controller」を別クラスとして実装しましたが、アノテーション実装モデルを採用することにより、分けていた2つの Controllerを1つにまとめることができます。

下記の LoginController.javaを jp.co.axiz.app.web.controller パッケージに配置したら、デプロイしてWebブラウザから http://localhost:8080/axizApp_03/login にアクセスしてみてください。

コンフィグレーション(設定)で実装したものと同様のログイン画面と挙動が実現できていれば成功です。

【LoginController.java】

```
package jp.co.axiz.app.web.controller;

/**
 * ログイン処理{@code Controller}クラス。
 *
 * @author {@code AxiZ} t.matsumoto
 * @see Controller
 * @see ModelAttribute
 * @see RequestMapping
 */
@Controller
public class LoginController {

    /** ログ出力に使用する{@link Logger}を表す定数。 */
    private static final Log LOG = LogFactory.getLog(LoginController.class);

    /**
     * ログイン画面入力値格納オブジェクトを返却します。
     * @return ログイン画面入力値オブジェクト
     */
    @ModelAttribute("loginForm")
    public LoginForm loginForm() {
        return new LoginForm();
    }
```

スペースの都合上、
import宣言を省略しています。
適宜追加してください。

アノテーションによるController 1/3

```
    /**
     * パスが{@code /}または{@code /login}で、リクエストが{@code GET}である場合の処理を実行します。
     * @return 遷移先
     */
    @RequestMapping(value = {"/", "/login"}, method = RequestMethod.GET)
    public String showLogin() {
        return "login";
    }

    /**
     * ログイン画面のログインボタンが押下された際の処理を実行します。
     * <p>パスが{@code /login}で、リクエストが{@code POST}である場合</p>
     * @param loginForm ログイン画面入力値格納オブジェクト
     * @return 遷移情報
     */
    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public ModelAndView login(LoginForm loginForm) {
        final ModelAndView modelAndView = new ModelAndView();

        // ユーザIDとパスワードが入力されていればログイン可とする.
        if (StringUtils.hasLength(loginForm.getUserId()) &&
            StringUtils.hasLength(loginForm.getUserPwd())) {
            modelAndView.addObject("resultMsg", loginForm.getUserId() + " さんがログインしました.");
            modelAndView.setViewName("top");
        } else {
            modelAndView.addObject("resultMsg", "ログインできません.");
            modelAndView.addObject("loginForm", loginForm);
            modelAndView.setViewName("login");
        }

        return modelAndView;
    }
}
```

アノテーションによるController 2/3

コンフィグレーション実装モデルでは、サーブレット設定ファイルにHandlerMapping(SimpleUrlHandlerMapping)を定義し、パスと Controller をマッピングしました。

また、Controllerクラスを bean定義して、Controller と Formオブジェクトを関連付けていました。

アノテーション実装モデルでは、これらの設定を Controllerクラスのソースファイル内に記述することが可能です。

そして、Controllerクラスを POJO として実装できるため、メソッド名の制約がなくなります。

さらに、アノテーション@RequestMappingで指定した「URLパス と リクエストメソッド(GET or POST)の組み合わせ」によってコールバックされるメソッドが決定するため、ログイン画面表示Controller とログイン処理Controllerを同一の URLパスにマッピングして1つにまとめることができました(コンフィグレーション実装モデルでも可能ですが GET or POSTを判定するコードの実装が必要となります)。

＊ リクエストメソッドの指定は任意です、指定がない場合は URLパスでメソッドが決定されます

アノテーションによるController 3/3

アノテーション @RequestMapping を付与したメソッドが受け取ることができる代表的な引数の型は、以下のようなものがあります。

javax.servlet.http.HttpServletRequest	リクエスト
javax.servlet.http.HttpServletResponse	レスポンス
javax.servlet.http.HttpSession	セッション
org.springframework.web.context.request.WebRequest	リクエストパラメータへのアクセス用オブジェクト 通常はHttpServletRequestを使用すれば良い
org.springframework.validation.BindingResult	検証結果格納用オブジェクト

ResourceBundleMessageSource 1/2

Webアプリケーション開発では、画面に表示するメッセージやラベル等の文字列リテラルをJSPやソースコードから分離させ、(一般的に)プロパティファイルにて管理します。

Spring Frameworkでは、org.springframework.context.support.ResourceBundleMessageSourceによりメッセージリソースを管理します。

まずは、JSPに直接記述していたラベル(タイトル等の文言)をメッセージリソース管理に移してみます。

以下の messageResource_ja.properties を src/main/java/properties に配置します。

【messageResource_ja.properties】

```
# Window Title
lbl.window.title.login=Login
lbl.window.title.top=Top

# Screen Title
lbl.screen.title.login=ログイン画面
lbl.screen.title.top=トップ画面

# Label
lbl.userId=ユーザID：
lbl.userPwd=パスワード：

# Button
lbl.btn.login=ログイン
```

次に、root-context.xml には右朱書きの定義を追加します。

【root-context.xml】

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

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

  <!-- Root Context: defines shared resources visible to all other web components -->

  <!-- ================================================== MessageResource -->
  <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
      <list>
        <value>properties.messageResource</value>
      </list>
    </property>
  </bean>

</beans>
```

root-context.xmlが格納されている場所は src/main/webapp/WEB-INF/spring です。
この設定ファイルには、アプリケーション共通の定義などを記述します。
appricationContext.xml というファイル名にすることもあります。

ResourceBundleMessageSource 2/2

では、messageResource_ja.properties に定義しているラベルを出力するように次ページのようにJSPを修正します。

朱書きの箇所が、追加/修正した記述です。

実際にログイン画面にアクセスし、正常に表示されれば成功です(設定ファイルを修正しているため、サーバの再起動が必要です)。

<spring:message>タグを使用することで、 メッセージ(ラベル)を画面出力させています。
code属性には、メッセージ･キー(コード)を指定します。

Spring Frameworkでは、その他にもタグが用意されています。

Spring Web MVC

ResourceBundleMessageSource 2/2

【login.jsp】

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c"    uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8" />
    <title><spring:message code="lbl.window.title.login" /></title>
  </head>
  <body>
    <h1><spring:message code="lbl.screen.title.login" /></h1>
    <%-- ログイン失敗メッセージ --%>
    <c:if test="${not empty resultMsg}">
      <p><c:out value="${resultMsg}" /></p>
    </c:if>
    <form:form modelAttribute="loginForm" action="login" method="post">
      <%-- ユーザID&パスワード入力エリア --%>
      <table border="0">
        <tr>
          <td align="right"><spring:message code="lbl.userId" /></td>
          <td><form:input path="userId" maxlength="12" /></td>
        </tr>
        <tr>
          <td align="right"><spring:message code="lbl.userPwd" /></td>
          <td><form:password path="userPwd" maxlength="12" /></td>
        </tr>
        <tr>
          <td align="center" colspan="2">
          <%-- ボタンエリア --%>
          <table style="margin-top: 10px" border="0">
            <tr><td><form:button><spring:message code="lbl.btn.login" /></form:button></td></tr>
          </table>
          </td>
        </tr>
      </table>
    </form:form>
  </body>
</html>
```

Spring Web MVC

タグライブラリ

ここでは、代表的なタグライブラリを説明します。

<spring:message>	メッセージを表示するタグ
<spring:hasBindErrors>	エラーメッセージの有無を判定するタグ
<form:errors>	エラーメッセージを表示するタグ
<form:form>	<form>(HTML)に対応するタグ
<form:input>	<input>(HTML)に対応するタグ
<form:textarea>	<textarea>(HTML)に対応するタグ
<form:button>	<button>(HTML)に対応するタグ
<form:password>	<input type="password">(HTML)に対応するタグ
<form:checkbox>	<input type="checkbox">(HTML)に対応するタグ
<form:radiobutton>	<input type="radio">(HTML)に対応するタグ
<form:select>	<select>(HTML)に対応するタグ
<form:option>	<option>(HTML)に対応するタグ
<form:options>	セレクトボックスの要素を動的に生成するタグ
<form:hidden>	<input type="hidden">(HTML)に対応するタグ

Spring Web MVC

入力値検証 1/4

Webアプリケーションでは、通常、不正データのDB登録を防止するためにプレゼンテーション層で入力値検証を行います。

ここでは、先程のログイン画面を使用して Spring MVC の入力値検証実装方法を紹介していきます。

では、ログイン画面の「ユーザID」と「パスワード」が必須入力項目であると(便宜上)仮定して進めます。

メッセージリソースに、下記の必須入力エラーメッセージを用意します。

【messageResource_ja.properties】

```
# Error Message
errors.input.required=未入力項目があります.
errors.required={0}を入力してください.<br />
```

入力値検証 2/4

次に、入力値(必須入力)の検証を行う Validatorクラスを実装します。
* org.springframework.validation.Validatorの実装クラスとします

jp.co.axiz.app.web.form.validatorパッケージを作成して、下記の LoginFormValidator.java を配置してください。

【LoginFormValidator.java】

```
package jp.co.axiz.app.web.form.validator;

import org.springframework.context.MessageSource;
import org.springframework.context.support.MessageSourceAccessor;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

import jp.co.axiz.app.web.form.LoginForm;

/**
 * ログイン画面入力値検証クラス。
 *
 * @author {@code AxiZ} t.matsumoto
 * @see Validator
 */
public class LoginFormValidator implements Validator {

    /** {@link MessageSourceAccessor}オブジェクトを保持します。 */
    private MessageSourceAccessor messages;

    public void setMessages(MessageSource messages) {
        this.messages = new MessageSourceAccessor(messages);
    }
```

```
    @Override
    public boolean supports(Class<?> clazz) {
        return LoginForm.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object obj, Errors errors) {

        final LoginForm loginForm = (LoginForm) obj;

        if (!StringUtils.hasLength(loginForm.getUserId())) {
            final String[] bindArgs = { messages.getMessage("lbl.userId") };
            errors.rejectValue("userId", "errors.required", bindArgs, "");
        }
        if (!StringUtils.hasLength(loginForm.getUserPwd())) {
            final String[] bindArgs = { messages.getMessage("lbl.userPwd") };
            errors.rejectValue("userPwd", "errors.required", bindArgs, "");
        }
        // ↓は必要に応じてセットする.
        if (errors.hasFieldErrors()) {
            errors.reject("errors.input.required");
        }
    }
}
```

入力値検証 3/4

実装した LoginFormValidatorクラスはservlet-context.xml に bean定義し、DIコンテナによって MessageSource がインジェクションされるように設定しておきます。

次に、入力値検証処理が実行されるようにLoginControllerクラスを一部修正します。

こちらはアノテーション@Autowiredを指定し、DIコンテナによって LoginFormValidator がインジェクションされるように設定しておきます。

【servlet-context.xml】

```
<!-- Validators -->

<!-- ログイン画面入力値Validator -->
<beans:bean id="loginFormValidator" class="jp.co.axiz.app.web.form.validator.LoginFormValidator">
  <beans:property name="messages" ref="messageSource" />
</beans:bean>
```

入力値検証 3/4

【LoginController.java】

```
public class LoginController {

    /** ログイン{@code Form}バリデータを保持します。 */
    @Autowired
    private Validator loginFormValidator;

    // 中略...

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public ModelAndView login(LoginForm loginForm, BindingResult bindingResult) {
        // 入力検証処理.
        loginFormValidator.validate(loginForm, bindingResult);

        final ModelAndView modelAndView = new ModelAndView();

        // 不正入力が発生している場合.
        if (bindingResult.hasErrors()) {
            modelAndView.setViewName("login");
            modelAndView.addObject("resultMsg", "ログインできません.");
            return modelAndView;
        }

        // ユーザIDとパスワードが入力されていればログイン可とする.
        if (StringUtils.hasLength(loginForm.getUserId()) &&
            StringUtils.hasLength(loginForm.getUserPwd())) {
            modelAndView.addObject("resultMsg", loginForm.getUserId() + " さんがログインしました.");
            modelAndView.setViewName("top");
        } else {
            modelAndView.addObject("resultMsg", "ログインできません.");
            modelAndView.addObject("loginForm", loginForm);
            modelAndView.setViewName("login");
        }

        return modelAndView;
    }
}
```

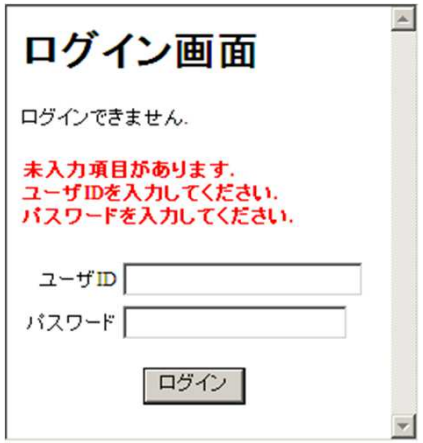
入力値検証 4/4

最後に、エラーメッセージを出力するための記述を login.jsp に追加します。

【login.jsp】

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c"      uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form"   uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<!DOCTYPE html>

<html>
<head>
  <meta charset="utf-8" />
  <title><spring:message code="lbl.window.title.login" /></title>
</head>
<body>
  <h1>
    <spring:message code="lbl.screen.title.login" />
  </h1>
  <%-- ログイン失敗メッセージ --%>
  <c:if test="${not empty resultMsg}">
    <p><c:out value="${resultMsg}" /></p>
  </c:if>
  <form:form modelAttribute="loginForm" action="login" method="post">
    <spring:hasBindErrors name="loginForm">
      <p style="color: red; font-weight: bold;">
      <c:forEach items="${errors.globalErrors}" var="error">
          <spring:message code="${error.code}" /><br />
      </c:forEach>
      <form:errors path="userId"  htmlEscape="false" />
      <form:errors path="userPwd" htmlEscape="false" />
      </p>
    </spring:hasBindErrors>

    <%-- ユーザID&パスワード入力エリア --%>
    <table border="0">
      <%-- 以下略... --%>
```

ログイン画面にアクセスし、正常に入力値検証が機能していれば成功です。



Spring Web MVC

Webアプリケーション レイヤーの結合

この「Spring Web MVC 編」では、Spring Web MVC フレームワークを利用してWebアプリケーションのアプリケーション層・プレゼンテーション層を実装してきました。

これで、以前から実装していたサービス層とDAO層を合わせ、全てのレイヤー(層)が揃いました。

いよいよ、全てを組み合わせでWebアプリケーションとして動かしてみましょう。

「MyBatis-Spring Framework連携 編」で使用していた DIコンテナ設定ファイル(beans.xml)をsrc/main/webapp/WEB-INF/spring に配置します(root-context.xml と同じ階層です)。

次に、src/main/webapp/WEB-INF/web.xml を修正し、配置したDIコンテナ設定ファイル(beans.xml)を読み込むように設定します。

あとは、以前実装したサービスやDAOモジュールの必要なファイル(ソース/設定/プロパティファイル等)を現プロジェクト内に取り込み、必要に応じて適宜 DIコンテナにインジェクションさせる設定を行います。

準備ができたら Controller からサービスを呼び出します。

では、実際に動作確認を行いましょう。

【web.xml】

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/spring/root-context.xml
    /WEB-INF/spring/beans.xml
  </param-value>
</context-param>
```