ＳＴＥＰ１０

Ｓｐｒｉｎｇフレームワーク入門

コードで理解するＳｐｒｉｎｇの世界

ログイン認証とトランザクション管理

目次

[1 ログイン認証を実装する 1](#_Toc415584448)

[1.1 Spring Security ライブラリ登録 （Maven） 1](#_Toc415584450)

[1.2 Spring Security定義ファイル （Spring Bean 定義ファイル）作成 2](#_Toc415584451)

[1.3 サーブレット設定ファイル web.xml から Spring Security定義ファイルをロード 3](#_Toc415584452)

[1.4 ログインフォーム画面 と ログイン失敗画面 を用意する 4](#_Toc415584453)

[1.5 画面表示テスト 6](#_Toc415584454)

[2 トランザクション管理を実装する 9](#_Toc415584455)

[2.1 Spring 定義ファイルにトランザクションマネージャの定義を追加 9](#_Toc415584457)

[2.2 サービスクラスでトランザクション管理できるようにする 10](#_Toc415584458)

[2.3 トランザクションのテスト 11](#_Toc415584459)

# ログイン認証を実装する



## Spring Security ライブラリ登録 （Maven）

１）Maven設定ファイルに必要なライブラリを追記

今回のテストでは、Spring フレームワークが提供する Spring Securityライブラリを使用するため、pom.xmlの2箇所に以下の定義を追加する。

|  |
| --- |
| /pom.xml |
| ● 9～13行目付近 |
| <properties>  <org.springframework-version>4.2.5.RELEASE</org.springframework-version>  <org.springfsecurity-version>4.0.4.RELEASE</org.springfsecurity-version>  <org.slf4j-version>1.7.16</org.slf4j-version>  </properties> |
| ● 77行目付近 |
| <!-- Validation -->  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-validator</artifactId>  <version>5.1.3.Final</version>  </dependency>  83    <!-- Spring Security -->  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-core</artifactId>  <version>${org.springfsecurity-version}</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-web</artifactId>  <version>${org.springfsecurity-version}</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-config</artifactId>  <version>${org.springfsecurity-version}</version>  </dependency>    <!-- Logging -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${org.slf4j-version}</version>  </dependency> |

## Maven設定ファイルを更新するとプロジェクトがリビルドする。

## ビルドの進捗は右下に地味に表示されているだけなので見落とさないこと

## ビルドに数秒～数分がかかる場合があるが、終わるまで待っているほうが吉。

## Spring Security定義ファイル （Spring Bean 定義ファイル）作成

１）以下の名前でSpring Security用のBean定義ファイルを新規作成する。

|  |
| --- |
| /src/main/webapp/WEB-INF/spring/application-context-security.xml （新規作成） |

２）Spring Security用のBean定義ファイルを以下の通り定義する。

|  |
| --- |
| /src/main/webapp/WEB-INF/spring/application-context-security.xml （新規作成） |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:sec="http://www.springframework.org/schema/security"  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.xsd  http://www.springframework.org/schema/security  http://www.springframework.org/schema/security/spring-security.xsd">  <!-- 以下のURLに対してはSpring Securityの対象外とする-->  <sec:http pattern="/loginform" security="none" />  <sec:http pattern="/autherror" security="none" />  <sec:http pattern="/resources/\*\*" security="none" />  <sec:http pattern="/css/\*\*" security="none" />  <!-- HTTP-Security auto-configをtrueにしてデフォルト設定を有効とする -->  <sec:http auto-config="true">  <!-- アクセス権限がない場合のエラーハンドラ -->  <sec:access-denied-handler error-page="/autherror" />    <!-- Cross-Site Request Forgery protection を無効にしておく -->  <sec:csrf disabled="true"/>  <!-- 全てのBookmgrサービスには ROLE\_USER 権限を必要とする-->  <sec:intercept-url pattern="/\*\*" access="hasRole('ROLE\_USER')" />    <!-- ログイン画面を独自の画面に切り替える -->  <sec:form-login login-page="/loginform"  username-parameter="j\_username"  password-parameter="j\_password"  login-processing-url="/j\_spring\_security\_check"  default-target-url="/main"  authentication-failure-url="/loginform?iserror=1"/>    <!-- ログアウト処理の設定 -->  <sec:logout logout-url="/logout" logout-success-url="/loginform"  invalidate-session="true"/>  </sec:http>  <!-- ユーザ認証の設定 -->  <sec:authentication-manager>  <sec:authentication-provider>  <sec:user-service>  <!-- ここでは簡単に認証情報を定義する -->  <!-- DBにアクセスして認証することも、LDAPとかを利用することも可能とのこと -->  <sec:user name="user1" password="password1" authorities="ROLE\_USER" />  <sec:user name="user2" password="password2" authorities="ROLE\_USER" />  <sec:user name="user3" password="password3" authorities="ROLE\_USER" />  </sec:user-service>  </sec:authentication-provider>  </sec:authentication-manager>  </beans> |

## サーブレット設定ファイル web.xml から Spring Security定義ファイルをロード

１）web.xmlを以下のように修正。

|  |
| --- |
| /src/main/webapp/WEB-INF/web.xml |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"  version="3.0">  <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>  /WEB-INF/spring/application-context-biz.xml  /WEB-INF/spring/application-context-security.xml  </param-value>  </context-param>  <!-- Creates the Spring Container shared by all Servlets and Filters -->  <listener>  <listener-class>  org.springframework.web.context.ContextLoaderListener  </listener-class>  </listener>  <!-- Processes application requests -->  <servlet>  <servlet-name>appServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/spring/application-context-web.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>appServlet</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  <!-- HTML ESCAPEの設定 -->  <context-param>  <param-name>defaultHTMLEscape</param-name>  <param-value>true</param-value>  </context-param>    <!-- 日本語文字化け対策の設定 -->  <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>  <!-- SpringSecurityフィルタの設定 -->  <filter>  <filter-name>springSecurityFilterChain</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  </filter>  <filter-mapping>  <filter-name>springSecurityFilterChain</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  </web-app> |

## ログインフォーム画面 と ログイン失敗画面 を用意する

１）ログインフォーム画面とログイン処理のコントロールメソッドをMainControllerに追加

|  |
| --- |
| /src/main/java/jp.sample.bookmgr.web.controller.MainController.java |
| ● 13行目付近 |
| ・・・  ・・・ import 文省略  ・・・  @Controller  public class MainController {    /\*\*  \* ロガー  \*/  private static final Logger logger = LoggerFactory.getLogger(MainController.class);    /\*\*  \* ログインフォーム画面コントローラ  \*  \* @return 画面JSP名  \*/  @RequestMapping(value = "/loginform", method = RequestMethod.GET)  public String loginForm() {  // 画面表示にloginform.jsp を呼び出す  return "loginform";  }    /\*\*  \* 権限エラー画面コントローラ  \*  \* @return 画面JSP名  \*/  @RequestMapping(value = "/autherror", method = RequestMethod.GET)  public String authError() {  return "error/autherror";  }  /\*\*  \* 書籍管理メイン画面コントローラ  \*  \* @return 画面JSP名  \*/  @RequestMapping(value = "/main", method = RequestMethod.GET)  public String main() throws Exception {    logger.debug("main() start");    // 画面表示に main.jsp を呼び出す  return "main";  }  } |

２）ログインフォーム画面View（loginform.jsp）新規作成

|  |
| --- |
| /src/main/webapp/WEB-INF/views/loginform.jsp |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <link rel="stylesheet" type="text/css" href="css/bookmgr.css" />  <title>書籍管理 ログイン画面</title>  </head>  <body>  <h1>書籍管理 ログイン画面</h1>  <hr />  <form name="login\_form" action="j\_spring\_security\_check" method="POST">  <table>  <tr>  <th>ログインID</th>  <td><input type="text" id="j\_username" name="j\_username"/></td>  </tr>  <tr>  <th>パスワード</th>  <td><input type="password" id="j\_password" name="j\_password"/></td>  </tr>  </table>  <br>  <input type="submit" value="ログイン" />  </form>  <c:if test="${param.iserror=='1'}">  <p style="color:red">ログイン認証に失敗しました。</p>  </c:if>  <hr />  </body>  </html> |

３）ログイン失敗画面View（error/autherror.jsp）新規作成

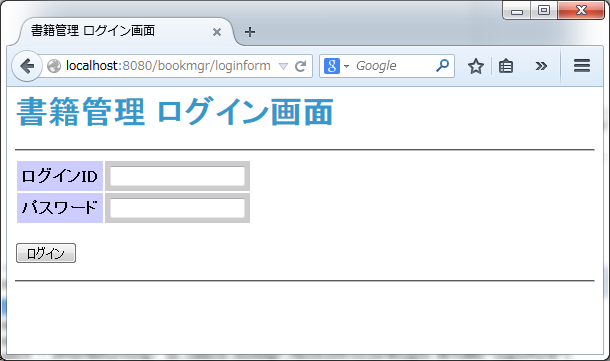
|  |
| --- |
| /src/main/webapp/WEB-INF/views/error/autherror.jsp |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <link rel="stylesheet" type="text/css" href="css/bookmgr.css" />  <title>権限エラー画面</title>  </head>  <body>  <h1>権限エラー</h1>  <hr />  <p style="color:red">アクセス権限がありません。</p>  <hr />  <a href="main">書籍管理メイン画面</a>  </body>  </html> |

４）main 画面View（main.jsp）にログアウトのリンクを追加

|  |
| --- |
| /src/main/webapp/WEB-INF/views/main.jsp |
| ● 14行目付近 |
| <a href="listbook">書籍一覧画面</a>  <br />  <a href="addbookform">書籍登録画面</a>  <br />  <a href="logout">ログアウト</a>  </body> |

## 画面表示テスト

１）プログラムを起動すると、ログイン画面が表示されることを確認。



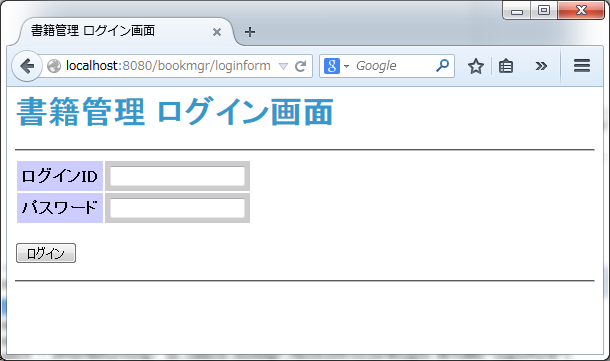
２）ログインIDに user1／パスワードに password1 を入力るとHome画面に遷移することを確認。

　同様に、user2/password2 user3/password3 でもログインできることを確認する。

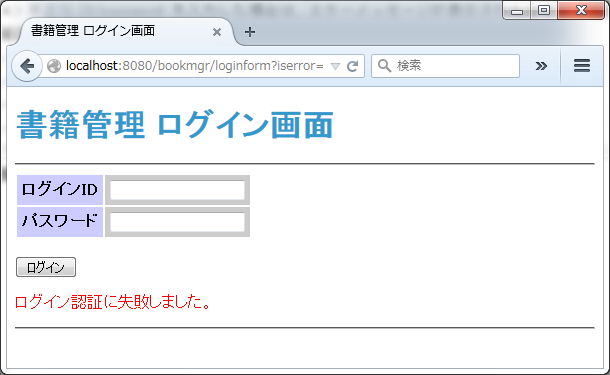


３）正常にログインした後のmain画面でログアウト・リンクをクリックすると、ログイン画面に戻ることを確認。

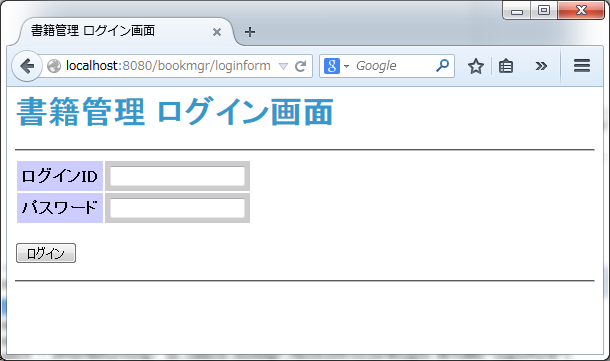




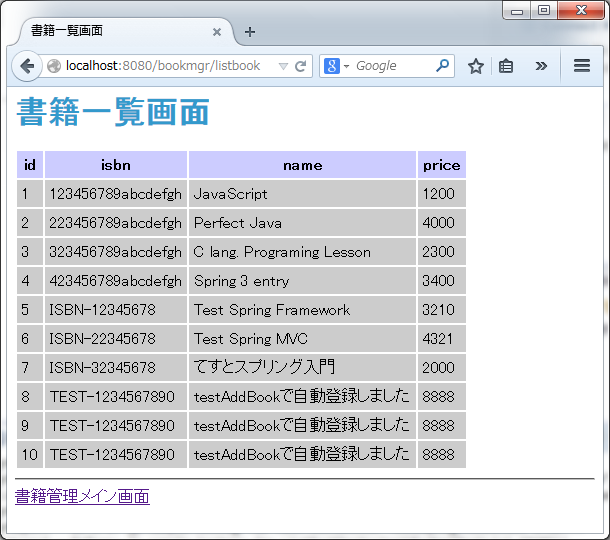
４）不正なID/password を入力した場合は、エラーメッセージが表示されたログイン画面に戻ることを確認する。



５）ログインしていない状態で、ダイレクトにメイン画面や書籍一覧画面のURLを入力しても、その画面遷移することなく、ログイン画面が表示されることを確認。



上記の状態でログインすると、ダイレクトに指定したURLに対する画面に遷移することを確認する。



# トランザクション管理を実装する



## Spring 定義ファイルにトランザクションマネージャの定義を追加

１）トランザクションマネージャを定義

|  |
| --- |
| /src/main/webapp/WEB-INF/spring/application-context-biz.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"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="  http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd">    <!-- Defines shared resources visible to all other web components -->  <context:component-scan base-package="jp.sample.bookmgr.biz" />    <!-- Enables the Spring AOP -->  <aop:aspectj-autoproxy />    <!-- データソース用のリソース -->  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  <property name="location" value="classpath:/jdbc.properties" />  </bean>    <!-- データソース設定 (for tomcat dbcp) -->  <bean id="dataSource" class="org.apache.tomcat.jdbc.pool.DataSource" 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>    <!-- JDBC Template Bean -->  <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">  <property name="dataSource" ref="dataSource" />  </bean>  <!-- JDBC Named Template Bean -->  <bean id="namedJdbcTemplate" class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">  <constructor-arg ref="dataSource" />  </bean>    <!-- Message Source -->  <bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">  <property name="basename" value="classpath:/messages" />  </bean>  <!-- Validator -->  <bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">  <property name="validationMessageSource" ref="messageSource" />  </bean>    <!-- Transaction Manager -->  <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource"/>  </bean>  <!-- アノテーションベースのトランザクションを有効にする -->  <tx:annotation-driven />    </beans> |

## Maven設定ファイルを更新するとプロジェクトがリビルドする。

## ビルドの進捗は右下に地味に表示されているだけなので見落とさないこと

## ビルドに数秒～数分がかかる場合があるが、終わるまで待っているほうが吉。

## サービスクラスでトランザクション管理できるようにする

１）書籍一覧サービス

書籍一覧サービス「ListBookServiceImple」は、DBの更新がないので、トランザクション管理を行う必要もないが、一応、readOnly モードで設定しておく。

|  |
| --- |
| /src/main/java/jp.sample.bookmgr.biz.service.ListbookServiceImple.java |
| ● 17行目付近 |
| @Service // サービスクラスとしてDI可能というアノテーションを宣言  public class ListBookServiceImple implements ListBookService {  /\*\*  \* 書籍一覧取得DAO  \*/  @Autowired // ListBookDaoオブジェクトをインジェクション  ListBookDao listBookDao;  /\*\*  \* 書籍一覧取得サービス  \* @return 書籍一覧情報  \*/  @Override  31  @Transactional(readOnly=true)  public List<Book> getBookList() throws Exception {  // 書籍一覧を取得  return listBookDao.getBookList();  }  } |

２）書籍追加サービス

書籍追加サービス「AddBookServiceImple」は、DB更新系なので、競業が発生しないようにReadCommitted で設定する。

|  |
| --- |
| /src/main/java/jp.sample.bookmgr.biz.service.AddbookServiceImple.java |
| ● 15行目付近 |
| @Service // サービスクラスとしてDI可能というアノテーションを宣言  public class AddBookServiceImple implements AddBookService {  /\*\*  \* データベースに書籍登録を行うDAOクラス  \*/  @Autowired // インジェクション  private AddBookDao addBookDao;    /\*\*  \* 書籍登録サービスを実行する  \*  \* @param Book 書籍情報  \*/  @Override  29  @Transactional(  propagation=Propagation.REQUIRED,  isolation=Isolation.READ\_COMMITTED,  timeout=10,  readOnly=false,  rollbackFor=RuntimeException.class)  public void addBook(Book book) throws Exception {  // 書籍登録を行うDAOクラスを使用して書籍情報を永続化する  addBookDao.addBook(book);  }  } |

## トランザクションのテスト

１）コードを追加しないとできないため講師の方でデモを行います。