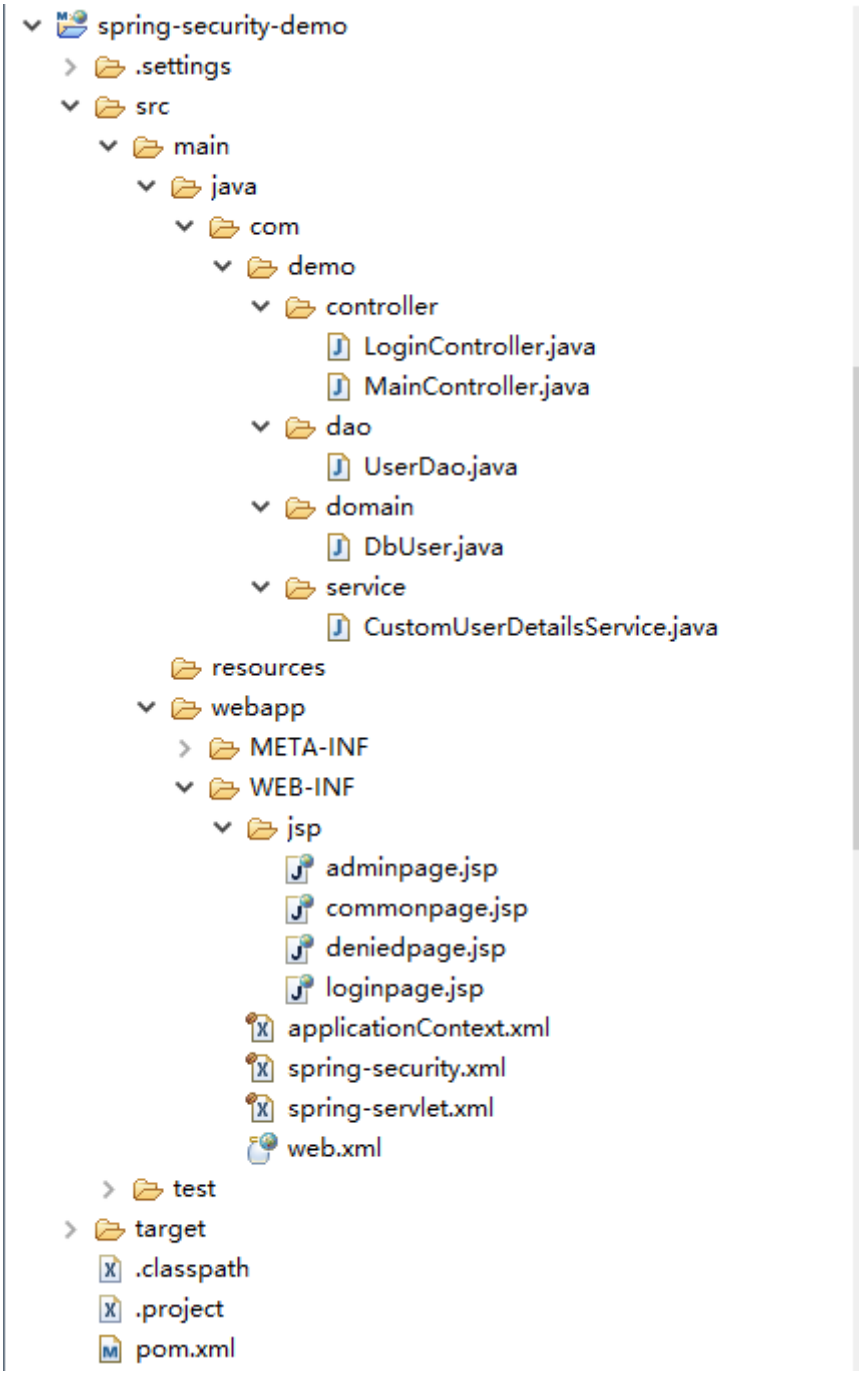
一、Spring Security简介

SpringSecurity，这是一种基于Spring AOP和Servlet过滤器的安全框架。它提供全面的安全性解决方案，同时在Web请求级和方法调用级处理身份确认和授权。在Spring Framework基础上，Spring Security充分利用了依赖注入（DI，Dependency Injection）和面向切面技术。

二、建立工程

参考http://blog.csdn.net/haishu_zheng/article/details/51490299，用第二种方法创建名为spring-security-demo的Maven工程。

工程的最终目录结构为



三、源代码

1 pom.xml里引入所需要的包

```
<project xmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/
  <modelVersion>4.0.0</modelVersion>
  <groupId>spring-security-demo</groupId>
  <artifactId>spring-security-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>spring-security-demo</name>
  <description/>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>7.0</version>
            <scope>provided</scope>
        </dependency>
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>3.2.9.RELEASE</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>3.2.9.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>3.1.6.RELEASE</version>
            <type>jar</type>
```

```
            <scope>compile</scope>
                                    </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
            <version>3.1.6.RELEASE</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.15</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
    </dependencies>

</project>
```

2 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.j
  <display-name>spring-security-demo</display-name>
        <filter>
                <filter-name>springSecurityFilterChain</filter-name>
                <filter-class>org.springframework.web.filter.DelegatingFilterProxy</
        </filter>
        <filter-mapping>
                <filter-name>springSecurityFilterChain</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>

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

```
        <servlet>
                        <servlet-name>spring</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</se
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>spring</servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>

        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListene
        </listener>
</web-app>
```

这里两处关于springsecurity的配置表示项目中所有路径的资源都要经过Spring Security。

注意：最好是将DelegatingFilterProxy写在DispatcherServlet之前，否则Spring Security可能不会正常工作。

3 spring-servlet.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:p="http://www.spri
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                    http://www.springframework.org/schema/beans/spring-beans-3.0.x
            <!-- 定义一个视图解析器 -->
        <bean id="viewResolver"

             class="org.springframework.web.servlet.view.InternalResourceViewResolv
                p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />    </beans>
```

这个XML配置声明一个视图解析器.在控制器中会根据JSP名映射到/WEB-INF/jsp中相应的位置。

4 applicationContext.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:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="http://www.springframework.org/schema/beans

                    http://www.springframework.org/schema/beans/spring-beans-3.0.x
                       http://www.springframework.org/schema/context
                    http://www.springframework.org/schema/context/spring-context-3
                       http://www.springframework.org/schema/mvc
                    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
            <!-- 激活spring的注解. -->
        <context:annotation-config />

        <!-- 扫描注解组件并且自动的注入spring beans中.

     例如,他会扫描@Controller 和@Service下的文件.所以确保此base-package设置正确. -->
        <context:component-scan base-package="com.demo" />

     <!-- 配置注解驱动的Spring MVC Controller 的编程模型.注:次标签只在 Servlet MVC工作!
        <mvc:annotation-driven />
</beans>
```

5 spring-security.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:security="http://www.springframework.org/schema/security"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                       http://www.springframework.org/schema/beans/spring-beans-3.0
                       http://www.springframework.org/schema/security
                       http://www.springframework.org/schema/security/spring-securi

        <!--  Spring-Security 的配置 -->
        <!-- 注意use-expressions=true.表示开启表达式,否则表达式将不可用.
        see:http://www.family168.com/tutorial/springsecurity3/html/el-access.html
         -->
        <security:http auto-config="true" use-expressions="true" access-denied-page=

                <security:intercept-url pattern="/auth/login" access="permitAll"/>
```

```
                <security:intercept-url pattern="/main/admin" access="hasRole('ROLE_
                <security:intercept-url pattern="/main/common" access="hasRole('ROLE

                <security:form-login
                                login-page="/auth/login"
                                authentication-failure-url="/auth/login?error=true"
                                default-target-url="/main/common"/>

                <security:logout
                                invalidate-session="true"
                                logout-success-url="/auth/login"
                                logout-url="/auth/logout"/>

        </security:http>

        <!-- 指定一个自定义的authentication-manager :customUserDetailsService -->
        <security:authentication-manager>
                <security:authentication-provider user-service-ref="customUserDetail
                                <security:password-encoder ref="passwordEncoder"/>
                </security:authentication-provider>
        </security:authentication-manager>

        <!-- 对密码进行MD5编码 -->
        <bean class="org.springframework.security.authentication.encoding.Md5Passwor

        <!--
                通过 customUserDetailsService,Spring会自动的用户的访问级别.
                也可以理解成:以后我们和数据库操作就是通过customUserDetailsService来进行
         -->
        <bean id="customUserDetailsService" class="com.demo.service.CustomUserDetail

</beans>
```

分析：

（一）

这里/auth/login的权限为permitAll，表示所有人都可以访问此页面；/main/admin的权限为ROLE_ADMIN，表示属于ROLE_ADMIN角色的用户才有权访问此页面；/main/common的权限为ROLE_USER，表示属于ROLE_USER的用户才有权访问此页面。

需要注意的是我们使用了SpringEL表达式来指定角色的访问.

以下是表达式对应的用法：

hasRole([role])返回 true 如果当前主体拥有特定角色。

hasAnyRole([role1,role2])返回 true 如果当前主体拥有任何一个提供的角色（使用逗号分隔的字符串队列）

principal 允许直接访问主体对象，表示当前用户

authentication允许直接访问当前 Authentication对象 从SecurityContext中获得

permitAll 一直返回true

denyAll 一直返回false

isAnonymous()如果用户是一个匿名登录的用户 就会返回 true

isRememberMe()如果用户是通过remember-me 登录的用户 就会返回true

isAuthenticated()如果用户不是匿名用户就会返回true

isFullyAuthenticated()如果用户不是通过匿名也不是通过remember-me登录的用户时， 就会返回true。

（二）

```
        <security:form-login
                login-page="/auth/login"
                authentication-failure-url="/auth/login?error=true"
                default-target-url="/main/common"/>
```

表示通过 /auth/login这个映射进行登录.
如果验证失败则返回一个URL:/auth/login?error=true
如果登录成功则默认指向:/main/common

（三）

```
<security:logout
                invalidate-session="true"
                logout-success-url="/auth/login"
```

```
                logout-url="/auth/logout"/>
```

这里我们开启了session失效功能。注销URL为:/auth/logout；注销成功后转向:/auth/login。

（四）

```
    <bean id="customUserDetailsService" class="com.demo.service.CustomUserDetailsService
```

一个自定义的CustomUserDetailsService,是实现SpringSecurity的UserDetailsService接口,但我们重写了他即便于我们进行数据库操作.

6 loginpage.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>

<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/htm
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>

        <h1>Login</h1>

        <div id="login-error">${error}</div>

        <form action="../j_spring_security_check" method="post">

                <p>
                        <label for="j_username">Username</label> <input id="j_username"
                                name="j_username" type="text" />
                </p>

                <p>
```

```
                        <label for="j_password">Password</label> <input id="j_password"
                                name="j_password" type="password" />
                </p>

                <input type="submit" value="Login" />

        </form>

</body>
</html>
```

7 commonpage.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/htm
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
        <h1>Common Page</h1>
        <p>每个人都能访问的页面.</p>
        <a href="/spring-security-demo/main/admin"> Go AdminPage </a>
        <br />
        <a href="/spring-security-demo/auth/login">退出登录</a>

</body>
</html>
```

8 adminpage.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/htm
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
        <h1>Admin Page</h1>
        <p>管理员页面</p>
        <a href="/spring-security-demo/auth/login">退出登录</a>
</body>
</html>
```

9 deniedpage.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/htm
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
        <h1>你的权限不够!</h1>
        <p>只有拥有Admin权限才能访问!</p>
        <a href="/spring-security-demo/auth/login">退出登录</a>
</body>
</html>
```

10 数据模型DbUser.java

```
package com.demo.domain;

public class DbUser {

        private String username;
        private String password;
        private Integer access;

        public String getUsername() {
```

```
                return username;
                                        }

        public void setUsername(String username) {
                this.username = username;
        }

        public String getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
        }

        public Integer getAccess() {
                return access;
        }

        public void setAccess(Integer access) {
                this.access = access;
        }

}
```

11 UserDao.java，通过一个初始化的List来模拟数据库操作

```
package com.demo.dao;

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

import org.apache.log4j.Logger;
import com.demo.domain.DbUser;

public class UserDao {

        protected static Logger logger = Logger.getLogger("dao");

        public DbUser getDatabase(String username) {

                List<DbUser> users = internalDatabase();
```

```
                for (DbUser dbUser : users) {
                    if (dbUser.getUsername().equals(username) == true) {
                            logger.debug("User found");
                            return dbUser;                        }
                }
                logger.error("User does not exist!");
                throw new RuntimeException("User does not exist!");

        }

        /**
         * 初始化数据
         */
        private List<DbUser> internalDatabase() {

                List<DbUser> users = new ArrayList<DbUser>();
                DbUser user = null;

                user = new DbUser();
                user.setUsername("admin");

                // "admin"经过MD5加密后
                user.setPassword("21232f297a57a5a743894a0e4a801fc3");
                user.setAccess(1);

                users.add(user);

                user = new DbUser();
                user.setUsername("user");

                // "user"经过MD5加密后
                user.setPassword("ee11cbb19052e40b07aac0ca060c23ee");
                user.setAccess(2);

                users.add(user);

                return users;

        }
}
```

12 CustomUserDetailsService.java，自定义UserDetailsService,可以通过继承UserDetailsService来达到灵活的自定义UserDetailsService

```
package com.demo.service;

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

import org.apache.log4j.Logger;
import com.demo.dao.UserDao;
import com.demo.domain.DbUser;
import org.springframework.dao.DataAccessException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.GrantedAuthorityImpl;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

/**
 * 一个自定义的service用来和数据库进行操作. 即以后我们要通过数据库保存权限.则需要我们继承User
 *
 */
public class CustomUserDetailsService implements UserDetailsService {

	protected static Logger logger = Logger.getLogger("service");

	private UserDao userDAO = new UserDao();

	public UserDetails loadUserByUsername(String username)
			throws UsernameNotFoundException, DataAccessException {

		UserDetails user = null;

		try {

			// 搜索数据库以匹配用户登录名.
			// 我们可以通过dao使用JDBC来访问数据库
			DbUser dbUser = userDAO.getDatabase(username);


		// Populate the Spring User object with details from the dbUse

		// Here we just pass the username, password, and access level

		// getAuthorities() will translate the access level to the cor
			// role type
		user = new User(dbUser.getUsername(), dbUser.getPassword()
```

```
                                        .toLowerCase(), true, true, true, true,
                                        getAuthorities(dbUser.getAccess()));
                } catch (Exception e) {
                        logger.error("Error in retrieving user");
                        throw new UsernameNotFoundException("Error in retrieving user"
                    }                       return user;
            }

            /**
             * 获得访问角色权限
             *
             * @param access
             * @return
             */
            public Collection<GrantedAuthority> getAuthorities(Integer access) {


                List<GrantedAuthority> authList = new ArrayList<GrantedAuthority>(2);
                        // 所有的用户默认拥有ROLE_USER权限
                    logger.debug("Grant ROLE_USER to this user");
                    authList.add(new GrantedAuthorityImpl("ROLE_USER"));

                    // 如果参数access为1.则拥有ROLE_ADMIN权限
                    if (access.compareTo(1) == 0) {
                            logger.debug("Grant ROLE_ADMIN to this user");
                            authList.add(new GrantedAuthorityImpl("ROLE_ADMIN"));
                    }

                    return authList;
            }
}
```

13 控制器LoginController.java

```
package com.demo.controller;

import org.apache.log4j.Logger;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
```

```
@Controller
@RequestMapping("auth")
public class LoginController {
        protected static Logger logger = Logger.getLogger("controller");

        /**
         * 指向登录页面
         */
        @RequestMapping(value = "/login", method = RequestMethod.GET)
        public String getLoginPage(

                        @RequestParam(value = "error", required = false) boolean error
                           ModelMap model) {
                logger.debug("Received request to show login page");

                if (error == true) {
                        // Assign an error message
                        model.put("error",

                                        "You have entered an invalid username or passw
                } else {                                model.put("error", "");
                }
                return "loginpage";
        }

        /**
         * 指定无访问额权限页面
         *
         * @return
         */
        @RequestMapping(value = "/denied", method = RequestMethod.GET)
        public String getDeniedPage() {

                logger.debug("Received request to show denied page");

                return "deniedpage";
        }
}
```

该controller有两个mapping映射

main/common

main/admin

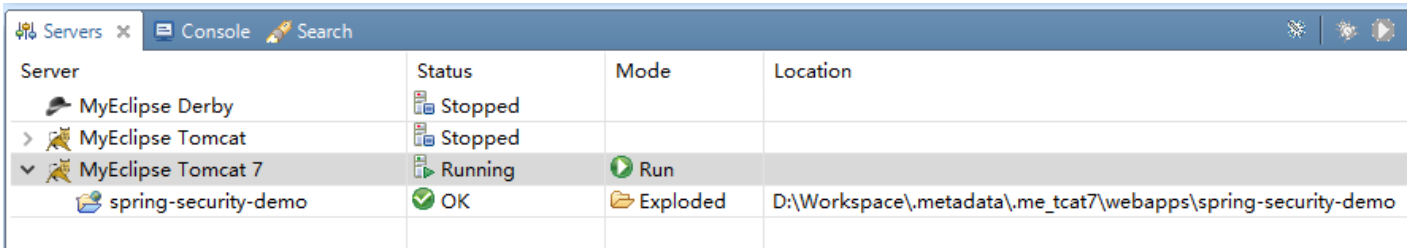
现在我们将同过Spring Security框架实现成功登陆的人都能访问到main/common，但只有拥有admin权限的用户才能访问main/admin。

14 控制器MainController.java

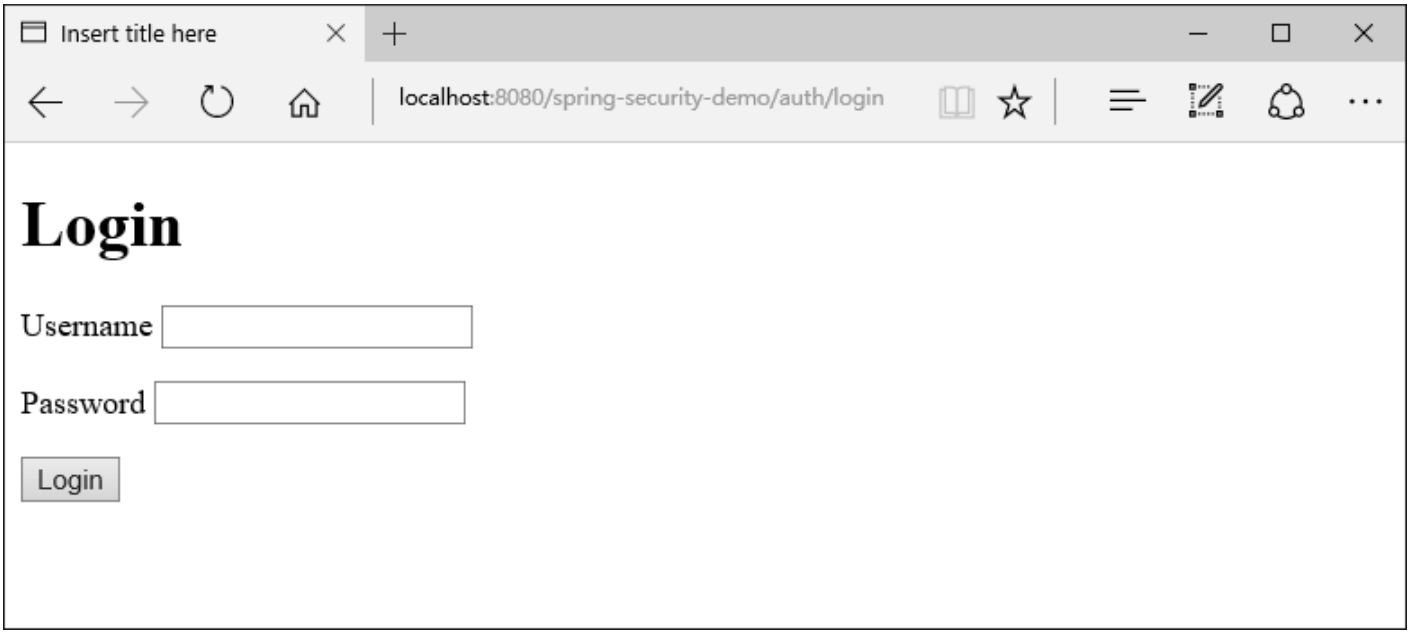
```
package com.demo.controller;

import org.apache.log4j.Logger;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/main")
public class MainController {
        protected static Logger logger = Logger.getLogger("controller");

        /**
         * 跳转到commonpage页面
         *
         * @return
         */
        @RequestMapping(value = "/common", method = RequestMethod.GET)
        public String getCommonPage() {
                logger.debug("Received request to show common page");
                return "commonpage";
        }

        /**
         * 跳转到adminpage页面
         *
         * @return
         */
        @RequestMapping(value = "/admin", method = RequestMethod.GET)
        public String getAadminPage() {
                logger.debug("Received request to show admin page");
                return "adminpage";

        }

}
```

四、运行结果

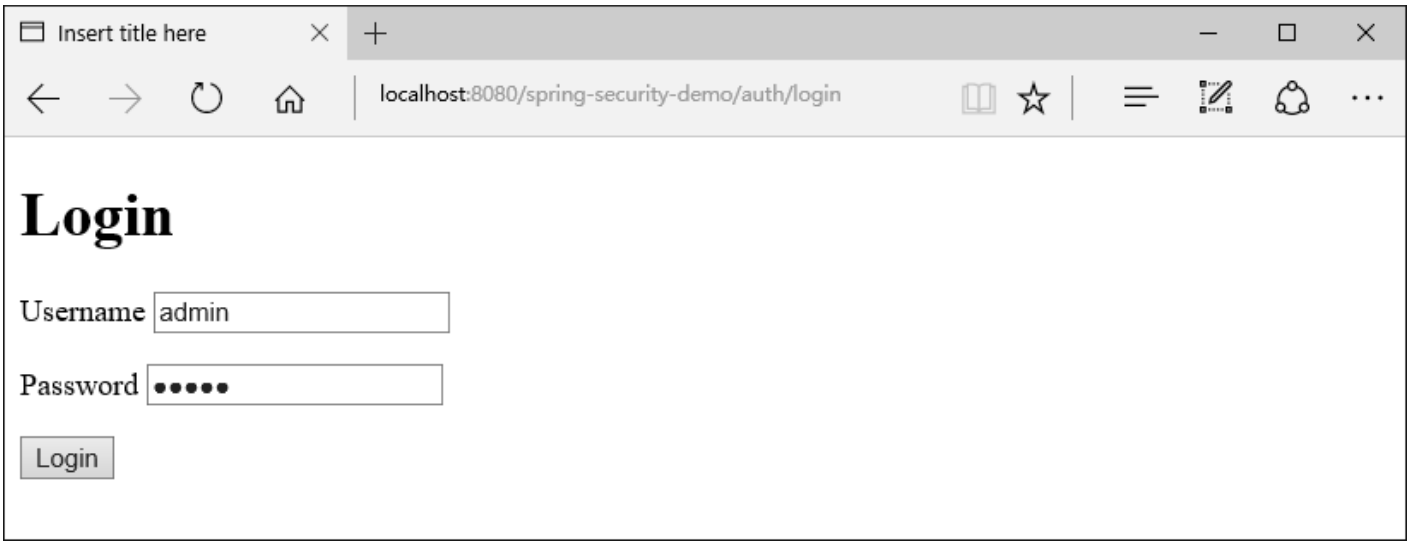
1 启动spring-security-demo程序



2 在浏览器里输入http://localhost:8080/spring-security-demo/auth/login

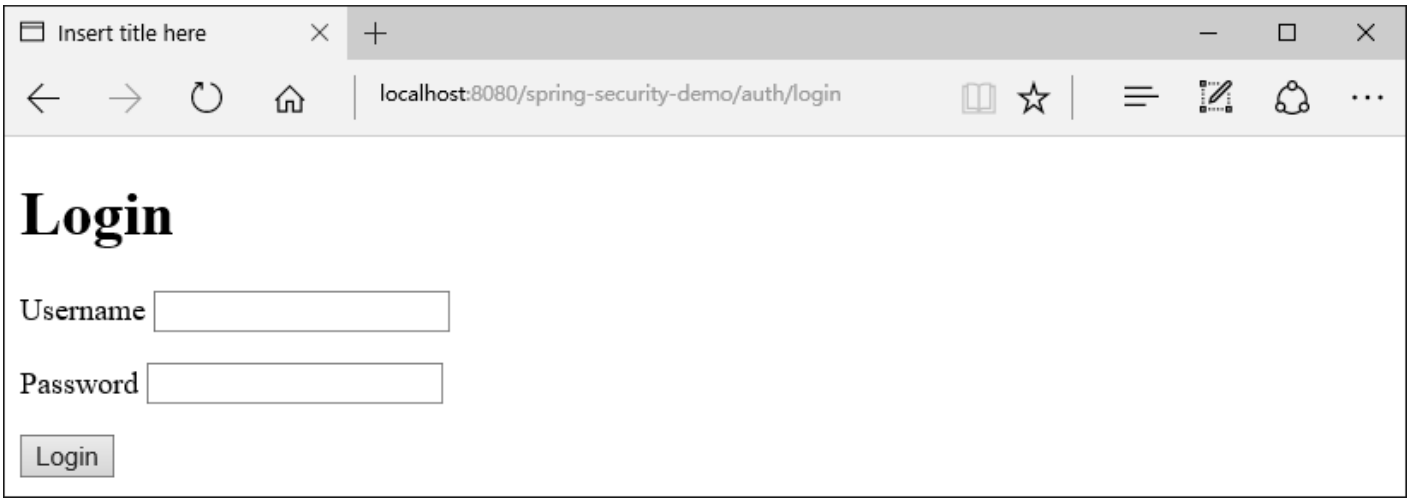


3 输入用户名admin密码admin后，点击“Login”按钮

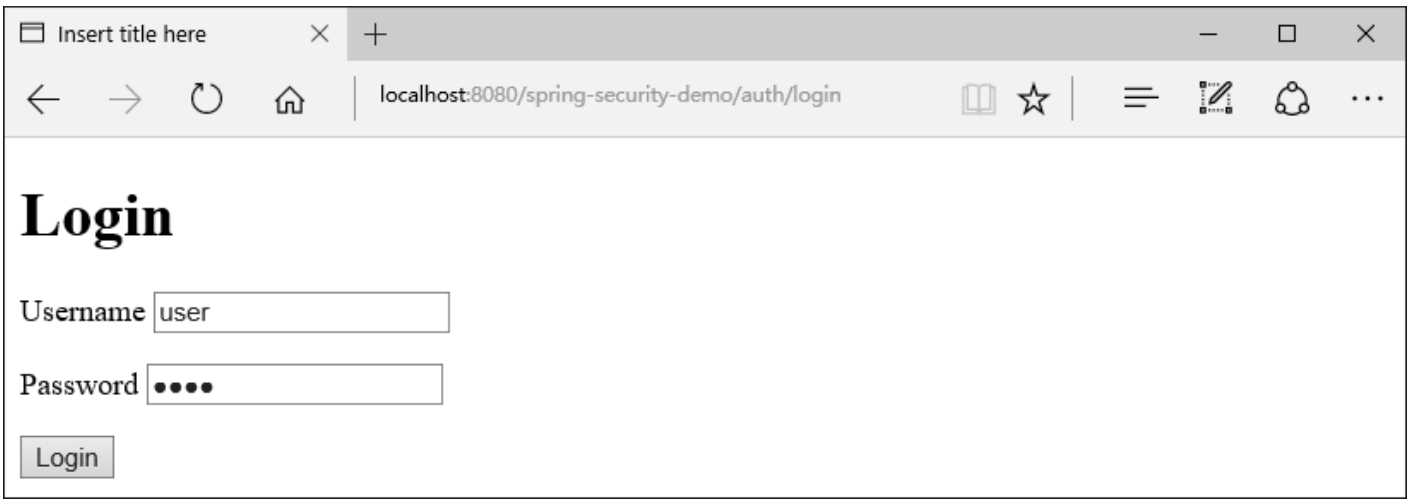


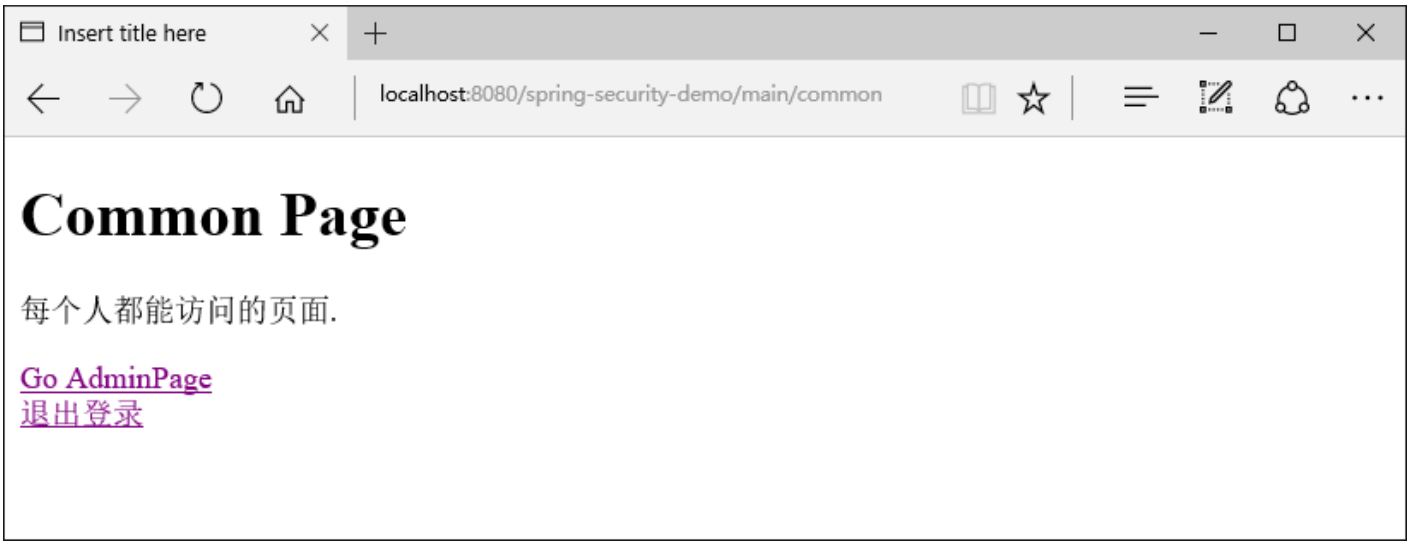
4 点击“Go AdminPage”链接，因为有权限，所以可看到管理员页面

5 点击“退出登录”，返回登录页

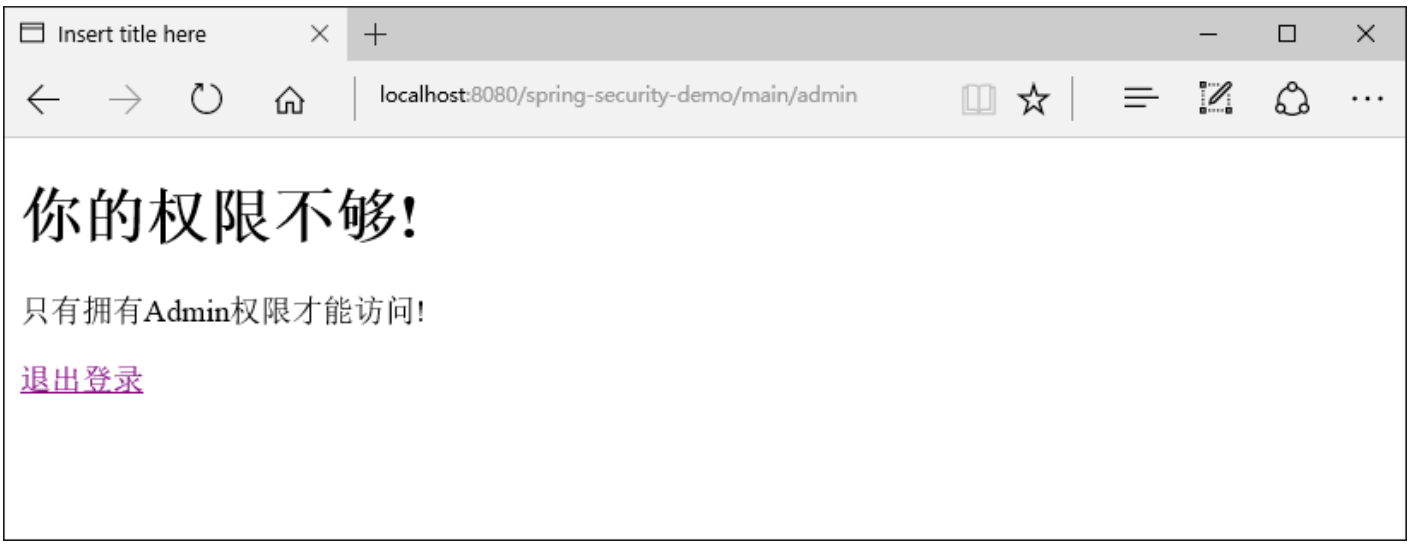


6 输入用户名user密码user并登录

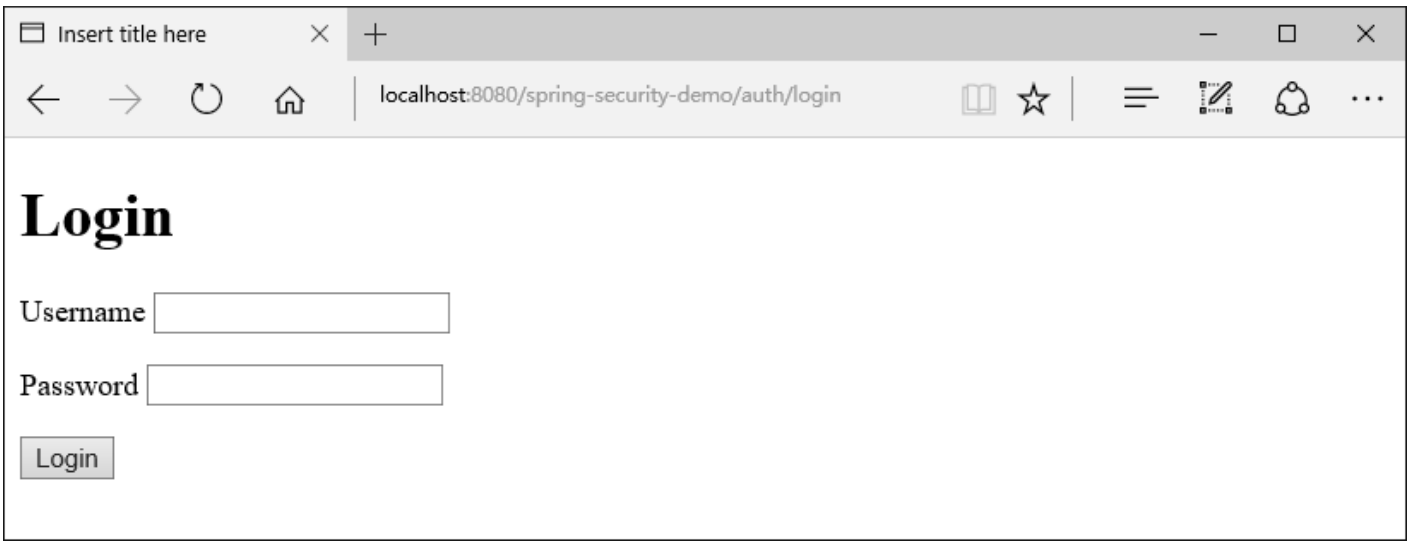




7 点击Go AdminPage链接，因为没有权限，所以看到权限不够的提示



8 退出登录，返回登录页



五、源码下载地址

CSDN: http://download.csdn.net/detail/haishu_zheng/9555916

Github: https://github.com/zhenghaishu/Spring-Security-Demo