SpringSecurity入门版本

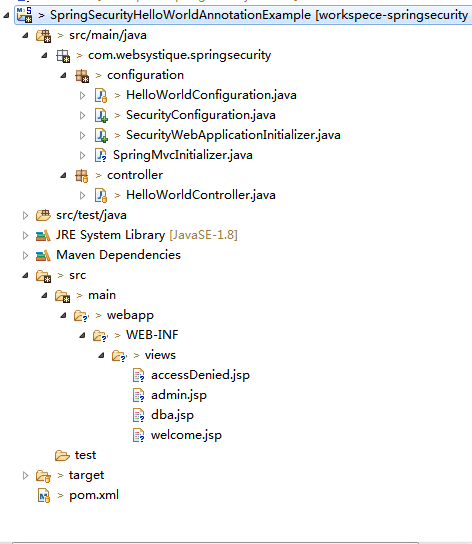
纯java配置版本

这个项目是根据网上的例子来照着做的。使用的SpringMVC整合SpringSecurity4的例子。

参考资料：<http://websystique.com/>

本文使用基于Servlet3.0容器的Spring注解(因此没有web.xml文件)。同样也会给出基于Security 配置的xml配置。

先上一个项目的结构图



## 首先

使用maven建立一个web项目（虽然我们这里使用的是java配置，没有用到web.xml，但后面的页面还是放在webapp这一web层中），并命名为 SpringSecurityHelloWorldAnnotationExample。

然后再pom.xml中引入下面的jar包（也可以全部复制过去）

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.me</groupId>

<artifactId>SpringSecurityHelloWorldAnnotationExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>SpringSecurityHelloWorldAnnotationExample</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<springframework.version>4.3.8.RELEASE</springframework.version>

<springsecurity.version>4.0.1.RELEASE</springsecurity.version>

</properties>

<dependencies>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${springframework.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${springframework.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${springframework.version}</version>

</dependency>

<!-- Spring Security -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>${springsecurity.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>${springsecurity.version}</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>javax.servlet.jsp-api</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.2</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

<warName>SpringSecurityHelloWorldAnnotationExample</warName>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

<finalName>SpringSecurityHelloWorldAnnotationExample</finalName>

</build>

</project>

需要注意的是maven-war-plugin 的声明。鉴于我们使用纯注解，甚至都没用web.xml。因此我们需配置此插件防止maven创建war包失败。

我们使用的是Spring 和 Spring Security(在本文发表时)最新版本。与此同时，由于我们将使用servlet api和jstl在我们界面中，我们也添加了JSP/Servlet/Jstl的依赖。

一般来说，容器也许已经包含了这些库，所以我们在pom.xml文件中，可以设置他们的scope 为provided（意味着打包的时候可以不用包进去，别的设施(Web Container)会提供）。

#### **第一步：**通过war注册springSecurityFilter（spring安全过滤器）

下面是定制初始化war包中的springSecurityFilter（第二步中的）注册类。

com.websystique.springsecurity.configuration.SecurityWebApplicationInitializer

package com.websystique.springsecurity.configuration;

import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {

}

#### **第二步：**添加 Spring Security 配置类

这个配置创建一个叫springSecurityFilterChain的Servlet过滤器，来对我们应用中所有的安全相关的事项（保护应用的所有url，验证用户名密码，表单重定向等）负责。

com.websystique.springsecurity.configuration.SecurityConfiguration

package com.websystique.springsecurity.configuration;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableWebSecurity

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired

    public void configureGlobalSecurity(AuthenticationManagerBuilder auth) throws Exception {

        auth.inMemoryAuthentication().withUser("bill").password("abc123").roles("USER");

        auth.inMemoryAuthentication().withUser("admin").password("root123").roles("ADMIN");

        auth.inMemoryAuthentication().withUser("dba").password("root123").roles("ADMIN","DBA");//dba have two roles.

    }

    @Override

    protected void configure(HttpSecurity http) throws Exception {

      http.authorizeRequests()

        .antMatchers("/", "/home").permitAll()

        .antMatchers("/admin/\*\*").access("hasRole('ADMIN')")

        .antMatchers("/db/\*\*").access("hasRole('ADMIN') and hasRole('DBA')")

        .and().formLogin()

        .and().exceptionHandling().accessDeniedPage("/Access\_Denied");

    }

}

上面这个类的configureGlobalSecurity方法为 AuthenticationManagerBuilder配置用户授权和角色信息 。

此AuthenticationManagerBuilder （权限管理器创建器）创建负责所有权限请求的AuthenticationManager（权限管理器）。

注意：在上面例子中，我们使用的是 基于内存的权限认证，当然你也可以自由选择JDBC,LDAP或者基于其他技术的权限认证。

重写Configure方法，来配置HttpSecurity 来配置基于特定http请求的安全认证。

它默认是实用所有请求的，但是也可以通过requestMatcher(RequestMatcher)/antMathchers 或者其他类似的方法进行限定。

在上述配置中，我们可以看到‘/’ & ‘/home’这种Url配置是不安全的，任何人都可以访问。

只有具有ADMIN权限的用户才可以访问符合‘/admin/\*\*’的url。只能够同时具有ADMIN 和 DBA权限的人才可以访问符合‘/db/\*\*’ 的Url 。

formLogin 方法提供了基于表单的权限验证，将会产生一个默认的对用户的表单请求。

你也可以自定义登录表单。在接下来的文章里面，你可以看到类似的例子。

我们也会使用exceptionHandling().accessDeniedPage() ，在本例中它将获取所有的403（http访问拒绝）异常然后显示我们的用户定义的HTTP403页面（虽然也没有太大益处）。

上面的安全配置 XML 配置形式如下:

<beans:beans xmlns="http://www.springframework.org/schema/security"

xmlns:beans="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-4.1.xsd

http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-4.0.xsd">

<http auto-config="true" >

<intercept-url pattern="/" access="permitAll" />

<intercept-url pattern="/home" access="permitAll" />

<intercept-url pattern="/admin\*\*" access="hasRole('ADMIN')" />

<intercept-url pattern="/dba\*\*" access="hasRole('ADMIN') and hasRole('DBA')" />

<form-login authentication-failure-url="/Access\_Denied" />

</http>

<authentication-manager >

<authentication-provider>

<user-service>

<user name="bill" password="abc123" authorities="ROLE\_USER" />

<user name="admin" password="root123" authorities="ROLE\_ADMIN" />

<user name="dba" password="root123" authorities="ROLE\_ADMIN,ROLE\_DBA" />

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

#### 第三步：添加 Controller（控制器）

com.websystique.springsecurity.controller.HelloWorldController

package com.websystique.springsecurity.controller;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

public class HelloWorldController {

@RequestMapping(value = { "/", "/home" }, method = RequestMethod.GET)

public String homePage(ModelMap model) {

model.addAttribute("greeting", "Hi, Welcome to mysite. ");

return "welcome";

}

@RequestMapping(value = "/admin", method = RequestMethod.GET)

public String adminPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

return "admin";

}

@RequestMapping(value = "/db", method = RequestMethod.GET)

public String dbaPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

return "dba";

}

@RequestMapping(value="/logout", method = RequestMethod.GET)

public String logoutPage (HttpServletRequest request, HttpServletResponse response) {

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

if (auth != null){

new SecurityContextLogoutHandler().logout(request, response, auth);

}

return "welcome";

}

@RequestMapping(value = "/Access\_Denied", method = RequestMethod.GET)

public String accessDeniedPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

return "accessDenied";

}

private String getPrincipal(){

String userName = null;

Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (principal instanceof UserDetails) {

userName = ((UserDetails)principal).getUsername();

} else {

userName = principal.toString();

}

return userName;

}

}

controller类中的方法比较繁琐.getPrincipal 方法返回从Spring SecurityContext中记录的登录的用户。

logoutPage 方法简单调用 SecurityContextLogoutHandler().logout(request, response, auth)方法  
来处理退出操作。

它很巧妙而且将你从不容易管理的jsp页面退出逻辑中解放出来。

你也许注意到上面没有出现 /login’，因为Spring Security默认会产生和处理。

#### 第四步: 添加 SpringMVC 配置类

com.websystique.springsecurity.configuration.HelloWorldConfiguration

package com.websystique.springsecurity.configuration;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.ViewResolver;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

import org.springframework.web.servlet.view.JstlView;

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = "com.websystique.springsecurity")

public class HelloWorldConfiguration {

@Bean

public ViewResolver viewResolver() {

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.class);

viewResolver.setPrefix("/WEB-INF/views/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

}

#### 第五步: 添加Initializer（初始化器）类

com.websystique.springsecurity.configuration.HelloWorldConfiguration

package com.websystique.springsecurity.configuration;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class SpringMvcInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class<?>[] getRootConfigClasses() {

return new Class[] { HelloWorldConfiguration.class };

}

@Override

protected Class<?>[] getServletConfigClasses() {

return null;

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

注意上面的初始化器继承自AbstractAnnotationConfigDispatcherServletInitializer ，它是所有WebApplicationInitializer 实现的基类.

在Servlet 3.0 环境下，通过实现WebApplicationInitializer 来配置ServletContext 。这意味着我们将不使用web.xml而且将在支持servlet3.0容器下发布此应用。

#### 第六步: 添加Views（视图）

welcome.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>HelloWorld page</title>

</head>

<body>

Greeting : ${greeting}

This is a welcome page.

</body>

</html>

admin.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>HelloWorld Admin page</title>

</head>

<body>

Dear <strong>${user}</strong>, Welcome to Admin Page.

<a href="<c:url value="/logout" />">Logout</a>

</body>

</html>

dba.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>DBA page</title>

</head>

<body>

Dear <strong>${user}</strong>, Welcome to DBA Page.

<a href="<c:url value="/logout" />">Logout</a>

</body>

</html>

accessDenied.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>AccessDenied page</title>

</head>

<body>

Dear <strong>${user}</strong>, You are not authorized to access this page

<a href="<c:url value="/logout" />">Logout</a>

</body>

</html>

#### 第七步: 创建和发布应用

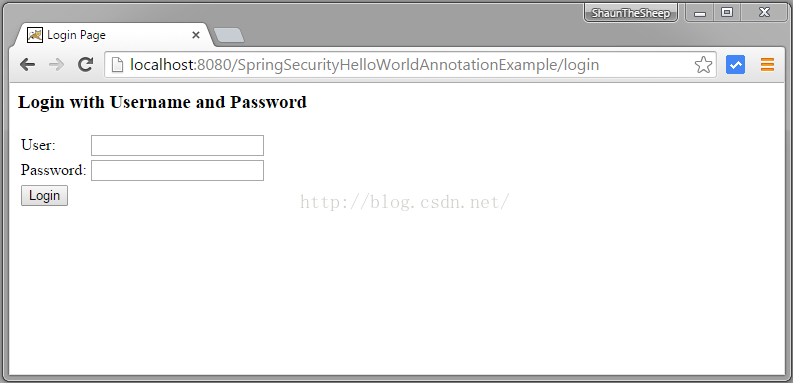
正如第五步提到的, 在我们应用中没有用到web.xml作为ServletContext 来启动程序.

现在构建 war 包(通过eclipse或者myeclipse)或者通过maven 命令行( mvn clean install). 在一个 Servlet 3.0 容器中发布本应用. 在这里我使用的是tomcat, 我将 war 文件放到 tomcat webapps 文件夹然后点击 tomcat安装目录的bin文件夹下的start.bat .

启动应用  
打开浏览器 在地址栏输入 localhost:8080/SpringSecurityHelloWorldAnnotationExample/并回车

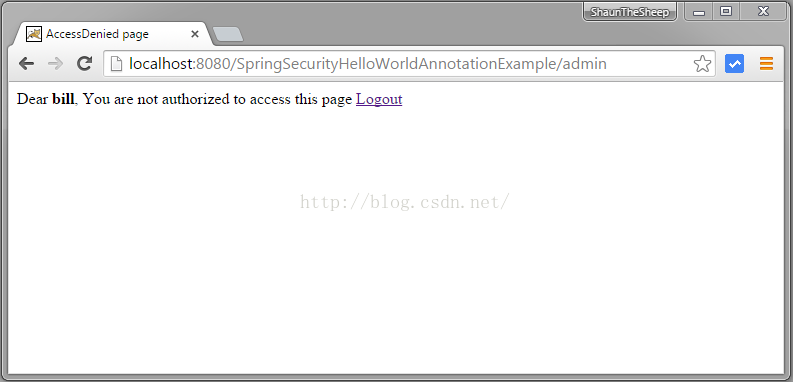


通过 localhost:8080/SpringSecurityHelloWorldAnnotationExample/admin 来访问admin 页面, 你将会被引导到登录页面.

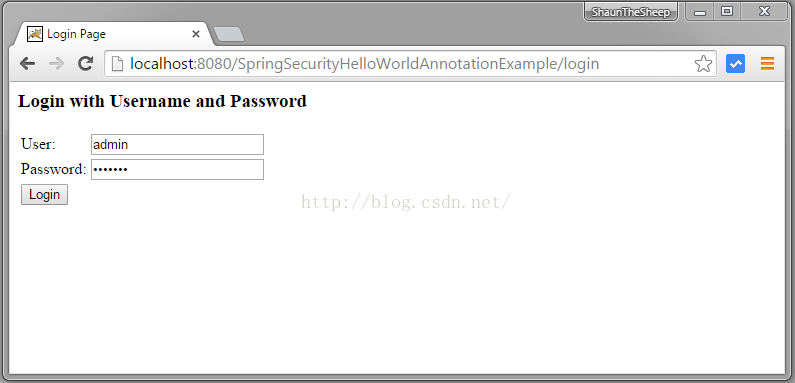


输入一个USER角色的账户

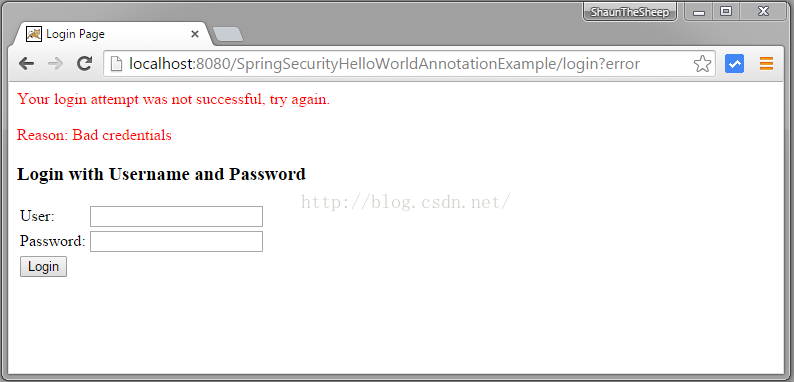
提交表单, 你将看到AccessDenied（访问拒绝）页面



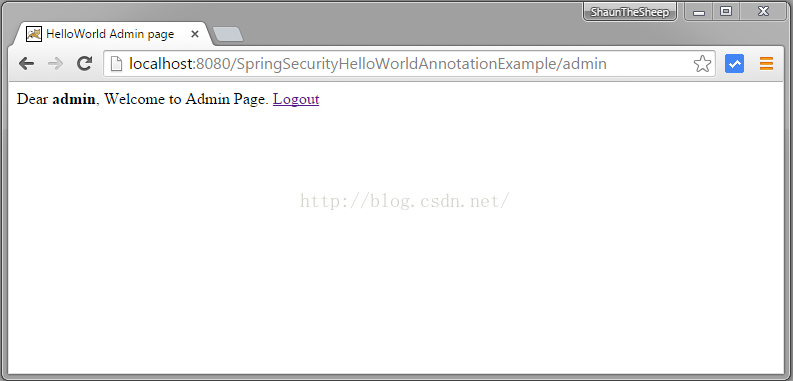
退出然后再次访问admin页面

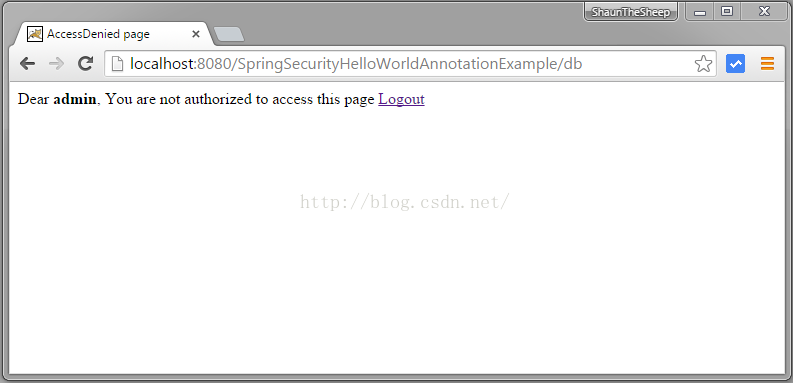


输入错误的password（密码）

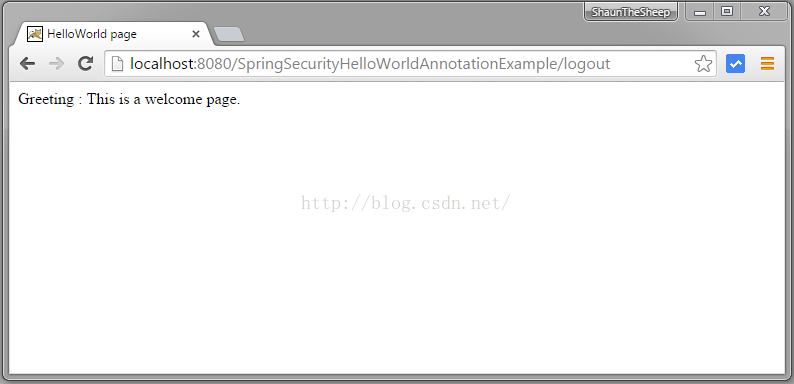


提供正确的admin 权限的账户再次登录

  
现在尝试通过localhost:8080/SpringSecurityHelloWorldAnnotationExample/db  访问 db页面将得到AccessDenied页面



退出, 将回到首页



本文是介绍基本应用. 

下一篇文章将介绍怎么样自定义登录表单来代替spring 自动生成的登录表单。

参考资料源码下载地址：<http://websystique.com/?smd_process_download=1&amp;download_id=1334>