1. Shiro同Web集成

本章主要介绍Shiro安全框架同Web（Servlet）应用的简单集成。主要包含如下几个方面：

* 了解配置内容和步骤
* 理解Shiro安全框架是基于Filter进行web集成的

目录

[9 Shiro同Web集成 1](#_Toc494349604)

[9.1 Shiro同Web集成简介 2](#_Toc494349605)

[9.2 Shiro同Web集成 3](#_Toc494349606)

[9.3 使用authcBasic拦截器验证身份 12](#_Toc494349607)

[9.4 总结 12](#_Toc494349608)

通过前面章节的研究，我们基本上理解Shiro安全框架的运行原理，知道基于Shiro安全框架进行身份认证和授权的基本流程，并且能在Java 应用程序中集成使用。因为现在基于Java开发的大部分是Web应用，而且基本上都是基于框架（Struts+Spring+Hibernate， SpringMVC+Spring+MyBatis）。这章简单介绍Shiro安全框架怎么样集成到基本的web应用（jsp/servlet）中，理解Shiro安全框架同Web应用集成的基本思路和实施方式，为后面同流行框架集成做好准备。

* 1. Shiro同Web集成简介

在Web应用中集成使用Shiro安全框架的目的，同在Java 应用程序中集成使用Shiro安全框架的目的一样，都是将身份认证和授权功能，都委托给Shiro安全框架实现，而不需要我们的应用再单独实现。

当然，Web应用中集成Shiro安全框架同Java应用程序中集成Shiro安全框架还是会有独特的需求，比如记住我功能、Web容器中的会话管理等等，这些我们在这里暂时不研究。

在Java应用程序中，我们是通过自己从SecurityManager中获取Subject对象，然后调用Subject的相关方法，实现Subject的身份认证和授权操作。在Web应用中也是要做同样的功能，不同的是Java应用我们是在main方法中，或main方法调用的其他方法中自己编写的代码，而Web应用是由Web容器管理的，所有应该有Web容器启动，主动调用相关的组件完成Subject的身份认证和授权操作。这些操作，都是由Shiro定义好的Filter完成（我们理解这些Filter后，可以根据需求自定义更合适的Filter）。

Shiro安全框架里面默认实现了如下几个Filter，并且在Web服务器启动后，自动初始化好，直接服务在Shiro安全框架中。

* org.apache.shiro.web.filter.authc.AnonymousFilter

默认名为anon，匿名拦截器。不需要登录即可访问；一般用于静态资源过滤

* org.apache.shiro.web.filter.authc.FormAuthenticationFilter

默认名称为authc，基于表单拦截器。连接指定的请求，先验证是否有登录，有正常登录，执行后续功能。没有登陆，如果是get请求，转向loginUrl，显示登录页面；如果是post请求，进入FormAuthenticationFilter的身份认证接口进行身份认证，认证成功，转向successUrl，否则转向登录页面。

常用的属性和意义如下：

1. usernameParam：登录的用户名参数名称
2. passwordParam：登录的密码参数名称
3. rememberParam：登录的记住我参数名称
4. loginUrl：登录页面url
5. successUrl：登录成功后的地址
6. failureKeyAttribute：登录失败后错误信息存储key，默认是shiroLoginFailure

* org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter

默认名称为authcBasic，基本的Http身份验证拦截器。同authc类似，不同的是如果没有登录，会弹出系统自己的登录验证框登录。主要属性是applicationName，应用名称，默认是application

* org.apache.shiro.web.filter.authc.LogoutFilter

默认名称是logout，退出拦截器。退出用户登录，自动清空session相关信息。主要属性是redirectUrl，退出后重定向页面。

* org.apache.shiro.web.filter.session.NoSessionCreationFilter

默认名称是noSessionCreation，不创建会话拦截器。代码中调用 subject.getSes  
sion(false)不会有什么问题，但是如果 subject.getSession(true)将抛出 DisabledSessionException异常；

* org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter

默认名称perms，权限授权拦截器。验证用户是否拥有所有权限，主要属性

1. loginUrl：登录url
2. unauthorizedUrl：登录了，但是没有授权url

* org.apache.shiro.web.filter.authz.PortFilter

默认名称是port，端口拦截器。验证发送请求的端口，如果不是指定端口，自动改成指定端口。主要属性：

1. port：端口，默认是80

* org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter

默认名称是rest，rest风格拦截器。自动根据请求名称构建权限字符串（GET=read, POST=create,PUT=update,DELETE=delete,HEAD=read,TRACE=read,OPTIONS=read, MKCOL=create）构建权限字符串；示例“/users=rest[user]”，会自动拼出“user:read,user:create,user:update,user:delete”权限字符串进行权限匹配（所有都得匹配，isPermittedAll）

* org.apache.shiro.web.filter.authz.RolesAuthorizationFilter

默认名称是roles，角色授权拦截器。验证用户是否拥有所有角色。主要属性：

1. loginUrl：登录页面地址（/login.jsp）；
2. unauthorizedUrl：未授权后重定向的地址

* org.apache.shiro.web.filter.authz.SslFilter

默认名称是ssl，SSL拦截器。只有请求协议是https才能通过；否则自动跳转会https端口（443）；其他和port拦截器一样

* org.apache.shiro.web.filter.authc.UserFilter

默认名称是user，用户拦截器。用户已经身份验证/记住我登录的都可

这些拦截器大概可以分成三类：

1. 身份验证相关拦截器，包括authc、authcBasic、logout、user和anon
2. 授权相关拦截器，包括roles、perms、port、rest和ssl
3. 其他拦截器，包括noSessionCreation

这些拦截器在Shiro的DefaultFilter中有定义，如下是DefaultFilter类的样例代码

package org.apache.shiro.web.filter.mgt;

... ...

public enum DefaultFilter {

… …

anon(AnonymousFilter.class),

authc(FormAuthenticationFilter.class),

authcBasic(BasicHttpAuthenticationFilter.class),

logout(LogoutFilter.class),

noSessionCreation(NoSessionCreationFilter.class),

perms(PermissionsAuthorizationFilter.class),

port(PortFilter.class),

rest(HttpMethodPermissionFilter.class),

roles(RolesAuthorizationFilter.class),

ssl(SslFilter.class),

user(UserFilter.class);

… …

}

* 1. Shiro同Web集成

前面简单介绍了Shiro在Web应用中进行安全拦截的拦截器，接下来介绍Web Shiro的应用。

1. 创建Web工程

基于Maven创建web工程ShiroTP09，我们这里就不介绍了，详细内容请查看随书代码[code/ShiroTP09](../code/ShiroTP09)。

1. 配置pom.xml
2. 继承ShiroPOM

在pom.xml添加parent元素，样例代码块如下：

<parent>

<groupId>ShiroBook</groupId>

<artifactId>ShiroPOM</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

1. 添加shiro-web依赖

样例代码块如下：

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

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

</dependency>

1. 添加servlet依赖

样例代码块如下：

<dependency>

<groupId>javax.servlet</groupId>

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

<version>3.0.1</version>

<scope>provided</scope>

</dependency>

<dependency>

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

<artifactId>jsp-api</artifactId>

<version>2.1</version>

<scope>provided</scope>

</dependency>

1. 添加war打包插件

在ShiroPOM的pom.xml的pluginManagement中添加maven-war-plugin，样例代码块如下，详细内容请查看随书代码[code/ShiroPOM/pom.xml](../code/ShiroPOM/pom.xml)。

<plugin>

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

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

<version>2.1.1</version>

</plugin>

在我们的ShiroTP09工程的pom.xml的plugins元素下，添加maven-war-plugin插件，样例代码块如下：

<plugin>

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

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

<configuration>

<encoding>${project.build.sourceEncoding}</encoding>

<warName>ShiroTP09</warName>

<webappDirectory>${project.build.directory}/ShiroTP09</webappDirectory>

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

</configuration>

</plugin>

1. 添加tomcat7插件

在ShiroPOM的pom.xml的pluginManagement中添加tomcat7-maven-plugin，样例代码块如下，详细内容请查看随书代码[code/ShiroPOM/pom.xml](../code/ShiroPOM/pom.xml)。

<plugin>

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

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.1</version>

</plugin>

在我们的ShiroTP09工程的pom.xml的plugins元素下，添加tomcat7-maven-plugin插件，样例代码块如下：

<plugin>

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

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<charset>${project.build.sourceEncoding}</charset>

<url>http://localhost:9090/manager/text</url>

<server>tomcat7</server>

<path>/ShiroTP09</path>

<port>9090</port>

<warFile>${project.build.directory}/ShiroTP09.war</warFile>

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

<uriEncoding>${project.build.sourceEncoding}</uriEncoding>

</configuration>

</plugin>

详细内容请查看随书代码[code/ShiroTP09/pom.xml](../code/ShiroTP09/pom.xml)。

1. 编写测试代码
2. AdminServlet

@WebServlet("/admin/adminServlet")

public class AdminServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

String currUser = (String)SecurityUtils.getSubject().getPrincipal();

PrintWriter out = response.getWriter();

out.print(currUser + " 你好，欢迎访问AdminServlet，TKS!<br/>");

out.print("<a href='../welcome.jsp'>点击这里</a>返回到导航页面");

out.flush();

out.close();

}

}

1. CreateUserServlet

@WebServlet("/user/createUserServlet")

public class CreateUserServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

String currUser = (String)SecurityUtils.getSubject().getPrincipal();

PrintWriter out = response.getWriter();

out.print(currUser + " 你好，欢迎访问CreateUserServlet，TKS!<br/>");

out.print("<a href='../welcome.jsp'>点击这里</a>返回到导航页面");

out.flush();

out.close();

}

}

1. DeleteUserServlet

@WebServlet("/user/deleteUserServlet")

public class DeleteUserServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

String currUser = (String)SecurityUtils.getSubject().getPrincipal();

PrintWriter out = response.getWriter();

out.print(currUser + " 你好，欢迎访问DeleteUserServlet，TKS!<br/>");

out.print("<a href='../welcome.jsp'>点击这里</a>返回到导航页面");

out.flush();

out.close();

}

}

1. LoginServlet

@WebServlet("/login")

public class LoginServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

request.getRequestDispatcher("index.jsp").forward(request, response);

}

}

1. QueryUserServlet

@WebServlet("/user/queryUserServlet")

public class QueryUserServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

String currUser = (String)SecurityUtils.getSubject().getPrincipal();

PrintWriter out = response.getWriter();

out.print(currUser + " 你好，欢迎访问QueryUserServlet，TKS!<br/>");

out.print("<a href='../welcome.jsp'>点击这里</a>返回到导航页面");

out.flush();

out.close();

}

}

1. UpdateUserServlet

@WebServlet("/user/updateUserServlet")

public class UpdateUserServlet extends HttpServlet {

protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

String currUser = (String)SecurityUtils.getSubject().getPrincipal();

PrintWriter out = response.getWriter();

out.print(currUser + " 你好，欢迎访问UpdateUserServlet，TKS!<br/>");

out.print("<a href='../welcome.jsp'>点击这里</a>返回到导航页面");

out.flush();

out.close();

}

}

如上代码，请查看随书代码目录[code/ShiroTP09/src/main/java/cn/com/shiro/book/tp09  
/servlet](../code/ShiroTP09/src/main/java/cn/com/shiro/book/tp09)。

1. index.jsp

<%@page contentType="text/html" pageEncoding="UTF-8"%>

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

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>JSP Page</title>

</head>

<body>

<h1>我自己的登录页面</h1>

<form action="login" method="post">

UserName:<input type='text' name="userName"/><br/>

Password:<input type='password' name="password"/><br/>

<input type="submit" value="登录"/>

</form>

</body>

</html>

1. unauthorized.jsp

<%@ page language="java" contentType="text/html; charset=utf-8"

pageEncoding="utf-8"%>

<%@taglib uri="http://shiro.apache.org/tags" prefix="shiro"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Insert title here</title>

</head>

<body>

sorry <shiro:principal/>,您没有权限访问该页面<br/>

<a href="welcome.jsp">请点击这里返回到导航页</a>，继续其他操作，谢谢！

</body>

</html>

1. welcome.jsp

<%@ page language="java" contentType="text/html; charset=utf-8"

pageEncoding="utf-8"%>

<%@taglib uri="http://shiro.apache.org/tags" prefix="shiro"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Insert title here</title>

</head>

<body>

<shiro:principal/>你好，恭喜成功进入本系统！<br>

您可以进行如下操作，请点击它们进入对应操作页面 <a href="logout">退出点这里</a><br/>

<a href="admin/adminServlet">进入AdminServlet页面</a><br/>

<br/>

<a href="user/createUserServlet">进入CreateUserServlet页面</a><br/>

<a href="user/updateUserServlet">进入UpdateUserServlet页面</a><br/>

<a href="user/deleteUserServlet">进入DeleteUserServlet页面</a><br/>

<a href="user/queryUserServlet">进入QueryUserServlet页面</a><br/>

</body>

</html>

如上代码详细内容请查看随书代码目录[code/ShiroTP09/src/main/webapp](../code/ShiroTP09/src/main/webapp)。

1. 配置web.xml

在web.xml中添加初始化Shiro安全框架的监听器，样例代码如下：

<listener>

<listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>

</listener>

在web.xml中添加Shiro安全框架的总入口过滤器，样例代码块如下：

<filter>

<filter-name>ShiroFilter</filter-name>

<filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>ShiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

<dispatcher>INCLUDE</dispatcher>

<dispatcher>ERROR</dispatcher>

</filter-mapping>

在web.xml中添加ini配置文件参数，样例代码如下：

<context-param>

<param-name>shiroConfigLocations</param-name>

<param-value>classpath:shiro\_shiroRealm.ini</param-value>

</context-param>

Shiro\_shiroRealm.ini文件，请查看6)

1. 编写Shiro安全框架的自定义组件。

Shiro安全框架的自定义组件包括ShiroRealm.java, BitPermission.java, BitAndWildcardPermissionResolver.java, ShiroRolePermissionResolver.java。这些代码都是第7章的样例代码，我们这里就不重复列出了，现在已经从ShiroTP07中复制到ShiroTP09，详细代码请查看随书代码目录[code/ShiroTP09/src/main/java/cn/com/  
shiro/book/tp07](../code/ShiroTP09/src/main/java/cn/com/shiro/book/tp07)。

1. 配置shiro的ini文件

这个配置文件是在ShiroTP07中的shiro\_shiroRealm.ini的基础上，添加了web url认证的[urls]部分，样例代码块如下，详细内容请参考随书代码[code/ShiroTP09/src/  
main/resources/shiro\_shiroRealm.ini](../code/ShiroTP09/src/main/resources/shiro_shiroRealm.ini)。

[main]

passwordService=org.apache.shiro.authc.credential.DefaultPasswordService

hashService=org.apache.shiro.crypto.hash.DefaultHashService

passwordService.hashService=$hashService

hashFormat=org.apache.shiro.crypto.hash.format.Shiro1CryptFormat

passwordService.hashFormat=$hashFormat

hashFormatFactory=org.apache.shiro.crypto.hash.format.DefaultHashFormatFactory

passwordService.hashFormatFactory=$hashFormatFactory

passwordMatcher=org.apache.shiro.authc.credential.PasswordMatcher

passwordMatcher.passwordService=$passwordService

#自定义authorizer

authorizer=org.apache.shiro.authz.ModularRealmAuthorizer

#自定义permissionResolver

permissionResolver=cn.com.shiro.book.tp07.authz.permission.BitAndWildcardPermissionResolver

authorizer.permissionResolver=$permissionResolver

#自定义rolePermissionResolver

rolePermissionResolver=cn.com.shiro.book.tp07.authz.permission.ShiroRolePermissionResovler

authorizer.rolePermissionResolver=$rolePermissionResolver

#设置SecurityManager授权器

securityManager.authorizer=$authorizer

#配置shiroRealm对象(realm名称=realm类名)

shiroRealm=cn.com.shiro.book.tp07.realm.ShiroRealm

shiroRealm.passwordService=$passwordService

shiroRealm.credentialsMatcher=$passwordMatcher

#设置shiroRealm对象

securityManager.realms=$shiroRealm

#定义Filter相关的属性

#设置authc(身份认证过滤器)的参数

authc.successUrl=welcome.jsp

authc.loginUrl=/login

authc.passwordParam=password

authc.usernameParam=userName

#设置logout(退出过滤器)的参数

logout.redirectUrl=/login

#设置roles(角色验证过滤器)的参数

roles.unauthorizedUrl=/unauthorized.jsp

#设置perms(权限验证过滤器)的参数

perms.unauthorizedUrl=/unauthorized.jsp

#配置每个url的控制

[urls]

/index.jsp=anon

#/welcome.jsp=authc

#/unauthorized.jsp=authc

/login=authc

/admin/adminServlet=authc,roles[admin]

/user/createUserServlet=authc, roles[role1]

/user/deleteUserServlet=authc, roles[role1,role2]

/user/updateUserServlet=authc, perms[1:4]

/user/queryUserServlet=authc, perms[1:8]

/logout=logout

/\*=authc

其中最重要的就是[urls]部分的配置，其格式是： “url=拦截器[参数]，拦截器[参数]”；即如果当前请求的url匹配[urls]部分的某个url模式，将会执行其配置的拦截器。比如anon拦截器表示匿名访问（即不需要登录即可访问）；authc拦截器表示需要身份认证通过后才能访问；roles[admin]拦截器表示需要有admin角色授权才能访问；而perms["user:create"]拦截器表示需要有“user:create”权限才能访问。

url模式使用Ant风格模式

Ant路径通配符支持?、\*、\*\*，注意通配符匹配不包括目录分隔符“/”：

?：匹配一个字符，如”/admin?”将匹配/admin1，但不匹配/admin或/admin2；

\*：匹配零个或多个字符串，如/admin\*将匹配/admin、/admin123，但不匹配/admin/1；

\*\*：匹配路径中的零个或多个路径，如/admin/\*\*将匹配/admin/a或/admin/a/b。

url模式匹配顺序

url模式匹配顺序是按照在配置中的声明顺序匹配，即从头开始使用第一个匹配的url模式对应的拦截器链。如：

配置代码

/bb/\*\*=filter1

/bb/aa=filter2

/\*\*=filter3

如果请求的url是“/bb/aa”，因为按照声明顺序进行匹配，那么将使用filter1进行拦截。

1. 测试

运行maven命令tomcat7:deploy， tomcat7:undeploy等命令，发布应用到tomcat7，就可以查看web验证效果。

* 1. 使用authcBasic拦截器验证身份

在shiro\_shiroRealm.ini中添加authcBasic的applicationName属性，并且将/user/queryUserServlet的验证改成authcBasic。样例代码如下：

#设置authcBasic的应用名称

authcBasic.applicationName=ShiroBook登录

#配置每个url的控制

[urls]

… …

/user/queryUserServlet=authcBasic, perms[1:8]

重新发布应用，在地址栏中请求user/queryUserServlet就会显示如下图9-1authcBasic身份验证页面。

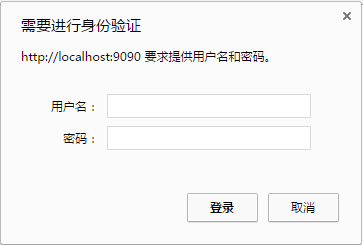


图9-1 authcBasic身份验证

这个页面是Shiro安全框架基于authcBasic身份验证拦截器自己提供的，输入用户名和密码登录，完成身份验证。

* 1. 总结

本章主要介绍了Shiro在Web应用中用来进行安全控制的拦截器，并且基于一个简单应用，配置测试了Shiro在Web应用中的使用。