[**第七章 与Web集成——《跟我学Shiro》**](http://jinnianshilongnian.iteye.com/blog/2024723)

**博客分类：**

* [跟我学Shiro](http://jinnianshilongnian.iteye.com/category/305053)

[跟我学Shiro](http://www.iteye.com/blogs/tag/%E8%B7%9F%E6%88%91%E5%AD%A6Shiro)

**目录贴：**[**跟我学Shiro目录贴**](http://jinnianshilongnian.iteye.com/blog/2018398)

Shiro提供了与Web集成的支持，其通过一个ShiroFilter入口来拦截需要安全控制的URL，然后进行相应的控制，ShiroFilter类似于如Strut2/SpringMVC这种web框架的前端控制器，其是安全控制的入口点，其负责读取配置（如ini配置文件），然后判断URL是否需要登录/权限等工作。

**7.1 准备环境**

**1、创建webapp应用**

此处我们使用了jetty-maven-plugin和tomcat7-maven-plugin插件；这样可以直接使用“mvn jetty:run”或“mvn tomcat7:run”直接运行webapp了。然后通过URLhttp://localhost:8080/chapter7/访问即可。

**2、依赖**

Servlet3

**Java代码  [收藏代码](javascript:void())**

1. <dependency>
2. <groupId>javax.servlet</groupId>
3. <artifactId>javax.servlet-api</artifactId>
4. <version>3.0.1</version>
5. <scope>provided</scope>
6. </dependency>

Servlet3的知识可以参考<https://github.com/zhangkaitao/servlet3-showcase>及Servlet3规范<http://www.iteye.com/blogs/subjects/Servlet-3-1>。

shiro-web

**Java代码  [收藏代码](javascript:void())**

1. <dependency>
2. <groupId>org.apache.shiro</groupId>
3. <artifactId>shiro-web</artifactId>
4. <version>1.2.2</version>
5. </dependency>

其他依赖请参考源码的pom.xml。

**7.2 ShiroFilter入口**

**1、Shiro 1.1及以前版本配置方式**

**Java代码  [收藏代码](javascript:void())**

1. <filter>
2. <filter-name>iniShiroFilter</filter-name>
3. <filter-**class**>org.apache.shiro.web.servlet.IniShiroFilter</filter-**class**>
4. <init-param>
5. <param-name>configPath</param-name>
6. <param-value>classpath:shiro.ini</param-value>
7. </init-param>
8. </filter>
9. <filter-mapping>
10. <filter-name>iniShiroFilter</filter-name>
11. <url-pattern>/\*</url-pattern>
12. </filter-mapping>

1、使用IniShiroFilter作为Shiro安全控制的入口点，通过url-pattern指定需要安全的URL；

2、通过configPath指定ini配置文件位置，默认是先从/WEB-INF/shiro.ini加载，如果没有就默认加载classpath:shiro.ini，即默认相对于web应用上下文根路径；

3、也可以通过如下方式直接内嵌ini配置文件内容到web.xml

**Java代码  [收藏代码](javascript:void())**

1. <init-param>
2. <param-name>config</param-name>
3. <param-value>
4. ini配置文件贴在这
5. </param-value>
6. </init-param>

**2、Shiro 1.2及以后版本的配置方式**

从Shiro 1.2开始引入了Environment/WebEnvironment的概念，即由它们的实现提供相应的SecurityManager及其相应的依赖。ShiroFilter会自动找到Environment然后获取相应的依赖。

**Java代码  [收藏代码](javascript:void())**

1. <listener>
2. <listener-**class**>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-**class**>
3. </listener>

通过EnvironmentLoaderListener来创建相应的WebEnvironment，并自动绑定到ServletContext，默认使用IniWebEnvironment实现。

可以通过如下配置修改默认实现及其加载的配置文件位置：

**Java代码  [收藏代码](javascript:void())**

1. <context-param>
2. <param-name>shiroEnvironmentClass</param-name>
3. <param-value>org.apache.shiro.web.env.IniWebEnvironment</param-value>
4. </context-param>
5. <context-param>
6. <param-name>shiroConfigLocations</param-name>
7. <param-value>classpath:shiro.ini</param-value>
8. </context-param>

shiroConfigLocations默认是“/WEB-INF/shiro.ini”，IniWebEnvironment默认是先从/WEB-INF/shiro.ini加载，如果没有就默认加载classpath:shiro.ini。

**3、与Spring集成**

**Java代码  [收藏代码](javascript:void())**

1. <filter>
2. <filter-name>shiroFilter</filter-name>
3. <filter-**class**>org.springframework.web.filter.DelegatingFilterProxy</filter-**class**>
4. <init-param>
5. <param-name>targetFilterLifecycle</param-name>
6. <param-value>**true**</param-value>
7. </init-param>
8. </filter>
9. <filter-mapping>
10. <filter-name>shiroFilter</filter-name>
11. <url-pattern>/\*</url-pattern>
12. </filter-mapping>

DelegatingFilterProxy作用是自动到spring容器查找名字为shiroFilter（filter-name）的bean并把所有Filter的操作委托给它。然后将ShiroFilter配置到spring容器即可：

**Java代码  [收藏代码](javascript:void())**

1. <bean id="shiroFilter" **class**="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
2. <property name="securityManager" ref="securityManager"/>
3. <!—忽略其他，详见与Spring集成部分 -->
4. </bean>

最后不要忘了使用org.springframework.web.context.ContextLoaderListener加载这个spring配置文件即可。

因为我们现在的shiro版本是1.2的，因此之后的测试都是使用1.2的配置。

**7.3 Web INI配置**

ini配置部分和之前的相比将多出对url部分的配置。

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. #默认是/login.jsp
3. authc.loginUrl=/login
4. roles.unauthorizedUrl=/unauthorized
5. perms.unauthorizedUrl=/unauthorized
6. [users]
7. zhang=123,admin
8. wang=123
9. [roles]
10. admin=user:\*,menu:\*
11. [urls]
12. /login=anon
13. /unauthorized=anon
14. /**static**/\*\*=anon
15. /authenticated=authc
16. /role=authc,roles[admin]
17. /permission=authc,perms["user:create"]

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：

**Java代码  [收藏代码](javascript:void())**

1. /bb/\*\*=filter1
2. /bb/aa=filter2
3. /\*\*=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。

拦截器将在下一节详细介绍。接着我们来看看身份验证、授权及退出在web中如何实现。

**1、身份验证（登录）**

**1.1、首先配置需要身份验证的url**

**Java代码  [收藏代码](javascript:void())**

1. /authenticated=authc
2. /role=authc,roles[admin]
3. /permission=authc,perms["user:create"]

即访问这些地址时会首先判断用户有没有登录，如果没有登录默会跳转到登录页面，默认是/login.jsp，可以通过在[main]部分通过如下配置修改：

**Java代码  [收藏代码](javascript:void())**

1. authc.loginUrl=/login

**1.2、登录Servlet（com.github.zhangkaitao.shiro.chapter7.web.servlet.LoginServlet）**

**Java代码  [收藏代码](javascript:void())**

1. @WebServlet(name = "loginServlet", urlPatterns = "/login")
2. **public** **class** LoginServlet **extends** HttpServlet {
3. @Override
4. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)
5. **throws** ServletException, IOException {
6. req.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(req, resp);
7. }
8. @Override
9. **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp)
10. **throws** ServletException, IOException {
11. String error = **null**;
12. String username = req.getParameter("username");
13. String password = req.getParameter("password");
14. Subject subject = SecurityUtils.getSubject();
15. UsernamePasswordToken token = **new** UsernamePasswordToken(username, password);
16. **try** {
17. subject.login(token);
18. } **catch** (UnknownAccountException e) {
19. error = "用户名/密码错误";
20. } **catch** (IncorrectCredentialsException e) {
21. error = "用户名/密码错误";
22. } **catch** (AuthenticationException e) {
23. //其他错误，比如锁定，如果想单独处理请单独catch处理
24. error = "其他错误：" + e.getMessage();
25. }
26. **if**(error != **null**) {//出错了，返回登录页面
27. req.setAttribute("error", error);
28. req.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(req, resp);
29. } **else** {//登录成功
30. req.getRequestDispatcher("/WEB-INF/jsp/loginSuccess.jsp").forward(req, resp);
31. }
32. }
33. }

1、doGet请求时展示登录页面；

2、doPost时进行登录，登录时收集username/password参数，然后提交给Subject进行登录。如果有错误再返回到登录页面；否则跳转到登录成功页面（此处应该返回到访问登录页面之前的那个页面，或者没有上一个页面时访问主页）。

3、JSP页面请参考源码。

**1.3、测试**

首先输入http://localhost:8080/chapter7/login进行登录，登录成功后接着可以访问http://localhost:8080/chapter7/authenticated来显示当前登录的用户：

**Java代码  [收藏代码](javascript:void())**

1. ${subject.principal}身份验证已通过。

当前实现的一个缺点就是，永远返回到同一个成功页面（比如首页），在实际项目中比如支付时如果没有登录将跳转到登录页面，登录成功后再跳回到支付页面；对于这种功能大家可以在登录时把当前请求保存下来，然后登录成功后再重定向到该请求即可。

Shiro内置了登录（身份验证）的实现：基于表单的和基于Basic的验证，其通过拦截器实现。

**2、基于Basic的拦截器身份验证**

**2.1、shiro-basicfilterlogin.ini配置**

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. authcBasic.applicationName=please login
3. ………省略users
4. [urls]
5. /role=authcBasic,roles[admin]

1、authcBasic是org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter类型的实例，其用于实现基于Basic的身份验证；applicationName用于弹出的登录框显示信息使用，如图：



2、[urls]部分配置了/role地址需要走authcBasic拦截器，即如果访问/role时还没有通过身份验证那么将弹出如上图的对话框进行登录，登录成功即可访问。

**2.2、web.xml**

把shiroConfigLocations改为shiro-basicfilterlogin.ini即可。

**2.3、测试**

输入http://localhost:8080/chapter7/role，会弹出之前的Basic验证对话框输入“zhang/123”即可登录成功进行访问。

**3、基于表单的拦截器身份验证**

基于表单的拦截器身份验证和【1】类似，但是更简单，因为其已经实现了大部分登录逻辑；我们只需要指定：登录地址/登录失败后错误信息存哪/成功的地址即可。

**3.1、shiro-formfilterlogin.ini**

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. authc.loginUrl=/formfilterlogin
3. authc.usernameParam=username
4. authc.passwordParam=password
5. authc.successUrl=/
6. authc.failureKeyAttribute=shiroLoginFailure
8. [urls]
9. /role=authc,roles[admin]

1、authc是org.apache.shiro.web.filter.authc.FormAuthenticationFilter类型的实例，其用于实现基于表单的身份验证；通过loginUrl指定当身份验证时的登录表单；usernameParam指定登录表单提交的用户名参数名；passwordParam指定登录表单提交的密码参数名；successUrl指定登录成功后重定向的默认地址（默认是“/”）（如果有上一个地址会自动重定向带该地址）；failureKeyAttribute指定登录失败时的request属性key（默认shiroLoginFailure）；这样可以在登录表单得到该错误key显示相应的错误消息；

**3.2、web.xml**

把shiroConfigLocations改为shiro- formfilterlogin.ini即可。

**3.3、登录Servlet**

**Java代码  [收藏代码](javascript:void())**

1. @WebServlet(name = "formFilterLoginServlet", urlPatterns = "/formfilterlogin")
2. **public** **class** FormFilterLoginServlet **extends** HttpServlet {
3. @Override
4. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)
5. **throws** ServletException, IOException {
6. doPost(req, resp);
7. }
8. @Override
9. **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp)
10. **throws** ServletException, IOException {
11. String errorClassName = (String)req.getAttribute("shiroLoginFailure");
12. **if**(UnknownAccountException.**class**.getName().equals(errorClassName)) {
13. req.setAttribute("error", "用户名/密码错误");
14. } **else** **if**(IncorrectCredentialsException.**class**.getName().equals(errorClassName)) {
15. req.setAttribute("error", "用户名/密码错误");
16. } **else** **if**(errorClassName != **null**) {
17. req.setAttribute("error", "未知错误：" + errorClassName);
18. }
19. req.getRequestDispatcher("/WEB-INF/jsp/formfilterlogin.jsp").forward(req, resp);
20. }
21. }

在登录Servlet中通过shiroLoginFailure得到authc登录失败时的异常类型名，然后根据此异常名来决定显示什么错误消息。

**4、测试**

输入http://localhost:8080/chapter7/role，会跳转到“/formfilterlogin”登录表单，提交表单如果authc拦截器登录成功后，会直接重定向会之前的地址“/role”；假设我们直接访问“/formfilterlogin”的话登录成功将直接到默认的successUrl。

**4、授权（角色/权限验证）**

**4.1、shiro.ini**

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. roles.unauthorizedUrl=/unauthorized
3. perms.unauthorizedUrl=/unauthorized
4. [urls]
5. /role=authc,roles[admin]
6. /permission=authc,perms["user:create"]

通过unauthorizedUrl属性指定如果授权失败时重定向到的地址。roles是org.apache.shiro.web.filter.authz.RolesAuthorizationFilter类型的实例，通过参数指定访问时需要的角色，如“[admin]”，如果有多个使用“，”分割，且验证时是hasAllRole验证，即且的关系。Perms是org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter类型的实例，和roles类似，只是验证权限字符串。

**4.2、web.xml**

把shiroConfigLocations改为shiro.ini即可。

**4.3、RoleServlet/PermissionServlet**

**Java代码  [收藏代码](javascript:void())**

1. @WebServlet(name = "permissionServlet", urlPatterns = "/permission")
2. **public** **class** PermissionServlet **extends** HttpServlet {
3. @Override
4. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)
5. **throws** ServletException, IOException {
6. Subject subject = SecurityUtils.getSubject();
7. subject.checkPermission("user:create");
8. req.getRequestDispatcher("/WEB-INF/jsp/hasPermission.jsp").forward(req, resp);
9. }
10. }

**Java代码  [收藏代码](javascript:void())**

1. @WebServlet(name = "roleServlet", urlPatterns = "/role")
2. **public** **class** RoleServlet **extends** HttpServlet {
3. @Override
4. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)
5. **throws** ServletException, IOException {
6. Subject subject = SecurityUtils.getSubject();
7. subject.checkRole("admin");
8. req.getRequestDispatcher("/WEB-INF/jsp/hasRole.jsp").forward(req, resp);
9. }
10. }

**4.4、测试**

首先访问http://localhost:8080/chapter7/login，使用帐号“zhang/123”进行登录，再访问/role或/permission时会跳转到成功页面（因为其授权成功了）；如果使用帐号“wang/123”登录成功后访问这两个地址会跳转到“/unauthorized”即没有授权页面。

**5、退出**

**5.1、shiro.ini**

**Java代码  [收藏代码](javascript:void())**

1. [urls]
2. /logout=anon

指定/logout使用anon拦截器即可，即不需要登录即可访问。

**5.2、LogoutServlet**

**Java代码  [收藏代码](javascript:void())**

1. @WebServlet(name = "logoutServlet", urlPatterns = "/logout")
2. **public** **class** LogoutServlet **extends** HttpServlet {
3. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)
4. **throws** ServletException, IOException {
5. SecurityUtils.getSubject().logout();
6. req.getRequestDispatcher("/WEB-INF/jsp/logoutSuccess.jsp").forward(req, resp);
7. }
8. }

直接调用Subject.logout即可，退出成功后转发/重定向到相应页面即可。

**5.3、测试**

首先访问http://localhost:8080/chapter7/login，使用帐号“zhang/123”进行登录，登录成功后访问/logout即可退出。

Shiro也提供了logout拦截器用于退出，其是org.apache.shiro.web.filter.authc.LogoutFilter类型的实例，我们可以在shiro.ini配置文件中通过如下配置完成退出：

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. logout.redirectUrl=/login
4. [urls]
5. /logout2=logout

通过logout.redirectUrl指定退出后重定向的地址；通过/logout2=logout指定退出url是/logout2。这样当我们登录成功后然后访问/logout2即可退出。

示例源代码：<https://github.com/zhangkaitao/shiro-example>；可