第三章：Web 项目集成 Shiro

一、Web 项目集成原理

1. 新建项目
2. 完成 pom.xml 文件的配置，导入相关依赖
3. 配置 web.xml

```
<!--启动时，加载shiro的环境-->
<listener>
  <listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>
</listener>

<!--创建权限管理器的上下文-->
<context-param>
  <param-name>shiroEnvironmentClass</param-name>
  <param-value>org.apache.shiro.web.env.IniWebEnvironment</param-value>
</context-param>

<!--加载ini文件-->
<context-param>
  <param-name>shiroConfigLocations</param-name>
  <param-value>classpath:shiro.ini</param-value>
</context-param>

<!--配置shiro的统一过滤路径-->
<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>
</filter-mapping>

</web-app>
```

二、默认过滤器

1. 认证相关

过滤器	过滤器类	说明	默认
authc	FormAuthenticationFilter	基于表单的过滤器；如“/**=authc”，如果没有登录会跳到相应的登录页面登录	无
logout	LogoutFilter	退出过滤器，主要属性：redirectUrl：退出成功后重定向的地址，如“/logout=logout”	/
anon	AnonymousFilter	匿名过滤器，即不需要登录即可访问；一般用于静态资源过滤；示例“/static/**=anon”	无

2. 授权相关

过滤器	过滤器类	说明
roles	RolesAuthorizationFilter	角色授权过滤器，验证用户是否拥有所有角色；主要属性： logir（/login.jsp）；unauthorizedUrl：未授权后重定向的地址；示例“/admin/**=roles[admin]”
perms	PermissionsAuthorizationFilter	权限授权过滤器，验证用户是否拥有所有权限；属性和roles一样例“/user/**=perms[“user:create”]”
port	PortFilter	端口过滤器，主要属性：port（80）：可以通过的端口；示例“/t用户访问该页面是非80，将自动将请求端口改为80并重定向到该数等都一样
rest	HttpMethodPermissionFilter	rest风格过滤器，自动根据请求方法构建权限字符串（GET=read POST=create,PUT=update,DELETE=delete,HEAD=read,TRACE= MKCOL=create）构建权限字符串；示例“/users=rest[user]”，会出“user:read,user:create,user:update,user:delete”权限字符串进得匹配，isPermittedAll）
ssl	SslFilter	SSL过滤器，只有请求协议是https才能通过；否则自动跳转会htt他和port过滤器一样；

三、完整案例

1. 配置 pom.xml

2. 编写 shiro.ini

```
#声明自定义的realm，且为安全管理器指定realms
[main]
definitionRealm=com.itheima.shiro.realm.DefinitionRealm
securityManager.realms=$definitionRealm
#用户退出后跳转指定JSP页面
logout.redirectUrl=/login.jsp
#若没有登录，则被authc过滤器重定向到login.jsp页面
authc.loginUrl = /login.jsp
[urls]
/login=anon
#发送/home请求需要先登录
/home= authc
#发送/order/list请求需要先登录
/order-list = roles[admin]
#提交代码需要order:add权限
/order-add = perms["order:add"]
#更新代码需要order:del权限
/order-del = perms["order:del"]
#发送退出请求则用退出过滤器
/logout = logout
```

3. 编写 LoginService

```
/**
 * @Description: 登录服务实现
 */
public class LoginServiceImpl implements LoginService {

    @Override
    public boolean login(UsernamePasswordToken token) {
        Subject subject = SecurityUtils.getSubject();
        try {
            subject.login(token);
        }catch (Exception ex){
            return false;
        }
        return subject.isAuthenticated();
    }

    @Override
    public void logout() {
        Subject subject = SecurityUtils.getSubject();
        subject.logout();
    }
}
```

4. 编写 Web 层：编写各种页面的 Servlet

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IO
    //获得用户名和密码
    String username = req.getParameter( s: "loginName");
    String password = req.getParameter( s: "password");
    //构建登录使用的token
    UsernamePasswordToken token = new UsernamePasswordToken(username,password);
    //登录操作
    LoginService loginService = new LoginServiceImpl();
    boolean isLogin = loginService.login(token);
    //如果登录成功，跳转home.jsp
    if (isLogin){
        req.getRequestDispatcher( s: "/home").forward(req, resp);
    }
    //如果登录失败，跳转继续登录页面
    resp.sendRedirect( s: "login.jsp");
}
```

```
/**
 * @Description: home首页
 */
@WebServlet(urlPatterns = "/home")
public class HomeServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOE
        doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IO
        req.getRequestDispatcher( s: "home.jsp").forward( servletRequest: req,  servletResponse: resp);
    }
}
```

5. 编写 jsp 页面

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Title</title>
</head>
<body>
<form method="post" action="${pageContext.request.contextPath}/login">
    <table>
        <tr>
            <th>登陆名称</th>
            <td><input type="text"  name="loginName"></td>
        </tr>
        <tr>
            <th>密码</th>
            <td><input type="password" name="password"></td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" value="提交"/>
            </td>
        </tr>
    </table>

</form>
</body>
</html>
```

四、Web 项目授权

(一)java 代码方式

1. 登录相关

Subject 登录相关方法	描述
isAuthenticated()	返回true 表示已经登录，否则返回false。

2. 角色相关

Subject 角色相关方法	描述
hasRole(String roleName)	返回true 如果Subject 被分配了指定的角色，否则返回false。
hasRoles(List<String> roleNames)	返回true 如果Subject 被分配了所有指定的角色，否则返回false。
hasAllRoles(Collection<String>roleNames)	返回一个与方法参数中目录一致的hasRole 结果的集合。有性能的提高如果许多角色需要执行检查（例如，当自定义一个复杂的视图）。
checkRole(String roleName)	安静地返回，如果Subject 被分配了指定的角色，不然的话就抛出AuthorizationException。
checkRoles(Collection<String>roleNames)	安静地返回，如果Subject 被分配了所有的指定的角色，不然的话就抛出AuthorizationException。
checkRoles(String... roleNames)	与上面的checkRoles 方法的效果相同，但允许Java5 的var-args 类型的参数

3. 资源相关

Subject 资源相关方法	描述
isPermitted(Permission p)	返回true 如果该Subject 被允许执行某动作或访问被权限实例指定的资源集合，否则返回false
isPermitted(List<Permission> perms)	返回一个与方法参数中目录一致的isPermitted 结果的集合。
isPermittedAll(Collection<Permission>perms)	返回true 如果该Subject 被允许所有指定的权限，否则返回false有性能的提高如果需要执行许多检查（例如，当自定义一个复杂的视图）
isPermitted(String perm)	返回true 如果该Subject 被允许执行某动作或访问被字符串权限指定的资源，否则返回false。
isPermitted(String...perms)	返回一个与方法参数中目录一致的isPermitted 结果的数组。有性能的提高如果许多字符串权限检查需要被执行（例如，当自定义一个复杂的视图）。
isPermittedAll(String...perms)	返回true 如果该Subject 被允许所有指定的字符串权限，否则返回false。
checkPermission(Permission p)	安静地返回，如果Subject 被允许执行某动作或访问被特定的权限实例指定的资源，不然的话就抛出AuthorizationException 异常。
checkPermission(String perm)	安静地返回，如果Subject 被允许执行某动作或访问被特定的字符串权限指定的资源，不然的话就抛出

4. 案例

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    doPost(req, resp);
}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    //获得当前主体
    Subject subject = SecurityUtils.getSubject();
    //当前主体是否登录
    boolean isAuthenticated = subject.isAuthenticated();
    if (isAuthenticated){
        req.getRequestDispatcher( s: "home.jsp").forward(req, resp);
    }
    req.getRequestDispatcher( s: "/login").forward(req, resp);
}
```

(二)jsp 标签方式

1. 使用方式

Shiro提供了一套JSP标签库来实现页面级的授权控制，在使用Shiro标签库前，首先需要在JSP引入shiro标签：

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

2. 相关标签

标签	说明
< shiro:guest >	验证当前用户是否为“访客”，即未认证（包含未记住）的用户
< shiro:user >	认证通过或已记住的用户
< shiro:authenticated >	已认证通过的用户。不包含已记住的用户，这是与user标签的区别所在
< shiro:notAuthenticated >	未认证通过用户。与guest标签的区别是，该标签包含已记住用户
< shiro:principal />	输出当前用户信息，通常为登录帐号信息
< shiro:hasRole name="角色">	验证当前用户是否属于该角色
< shiro:lacksRole name="角色">	与hasRole标签逻辑相反，当用户不属于该角色时验证通过
< shiro:hasAnyRoles name="a,b">	验证当前用户是否属于以下任意一个角色
<shiro:hasPermission name="资源">	验证当前用户是否拥有制定权限
<shiro:lacksPermission name="资源">	与permission标签逻辑相反，当前用户没有制定权限时，验证通过

3. 案例

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title></title>
</head>
<body>
<h6>
    <a href="${pageContext.request.contextPath}/logout">退出</a>

    <shiro:hasRole name="admin">
        <a href="${pageContext.request.contextPath}/order-list">列表</a>
    </shiro:hasRole>

    <shiro:hasPermission name="order:add">
        <a href="${pageContext.request.contextPath}/order-add">添加</a>
    </shiro:hasPermission>
</h6>
</body>
</html>
```

4. 注意：jsp 方式只能控制标签是否显示，但不能控制是否可以通过链接访问