高级 web Homework report

————shiro 安全框架

水泽农 13302010061

1. 作业简介：

Apache Shiro 是 Java 的一个安全框架，旨在简化身份验证和授权。本次项目用 myeclipse 的编译器，利用 springmvc 和 thiro 的技术实现一个带有身份验证加密的登录系统。

2. 程序截图

登陆失败：

登陆成功：

3. 程序介绍

1）.定义 shiro 的拦截器

对 url 进行拦截，如果没有验证成功的需要验证，然后额外给用户赋予角色和权限。自定义的拦截器需要继承 AuthorizingRealm 并实现登录验证和赋予角色权限的两个方法。具体代码实现在 com.shiro.realm 中。

```
import java.util.HashSet;
import java.util.Set;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import com.util.DecriptUtil;
public class MyShiroRealm extends AuthorizingRealm {
    //直接默认只有一个用户("shuizenong"，"123456")
    private static final String USER_NAME = " shuizenong ";
    private static final String PASSWORD = "123456";

    //授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
        Set<String> roleNames = new HashSet<String>();
        Set<String> permissions = new HashSet<String>();
        roleNames.add("administrator");//添加角色
        permissions.add("newPage.jhtml");   //添加权限
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo(roleNames);
        info.setStringPermissions(permissions);
        return info;
    }
    //  登录验证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken authcToken) throws AuthenticationException {
        UsernamePasswordToken token = (UsernamePasswordToken) authcToken;
```

```
            if(token.getUsername().equals(USER_NAME)){
                return new SimpleAuthenticationInfo(USER_NAME, DecriptUtil.MD5(PASSWORD), getName());
            }else{
                throw new AuthenticationException();
            }
        }
}
```

2）shiro 配置

spring-shiro.xml 文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="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-3.0.xsd"
    default-lazy-init="true">
    <description>Shiro Configuration</description>
    <!-- Shiro ‘s main business-tier object for web-enabled applications -->
    <bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <property name="realm" ref="myShiroRealm" />
        <property name="cacheManager" ref="cacheManager" />
    </bean>
    <!-- 项目自定义的 Realm -->
    <bean id="myShiroRealm" class="com.shiro.realm.MyShiroRealm">
        <property name="cacheManager" ref="cacheManager" />
    </bean>

    <!-- Shiro Filter -->
    <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <property name="securityManager" ref="securityManager" />
        <property name="loginUrl" value="/login.jhtml" />  //没有登录的用户请求需要登录的页面时自动跳转到登录页面
        <property name="successUrl" value="/loginsuccess.jhtml" />  //登录成功默认跳转页面。
        <property name="unauthorizedUrl" value="/error.jhtml" /> //没有权限默认跳转页面
        <property name="filterChainDefinitions">
            <value>
                /index.jhtml = authc
                /login.jhtml = anon
                /checkLogin.json = anon
                /loginsuccess.jhtml = anon
                /logout.json = anon
                /** = authc
```

```
                </value>
            </property>
        </bean>

        <!-- 用户授权信息 Cache -->
        <bean id="cacheManager" class="org.apache.shiro.cache.MemoryConstrainedCacheManager" />

        <!-- 保证实现了 Shiro 内部 lifecycle 函数的 bean 执行 -->
        <bean id="lifecycleBeanPostProcessor" class="org.apache.shiro.spring.LifecycleBeanPostProcessor" />

        <!-- AOP 式方法级权限检查 -->
        <bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"
            depends-on="lifecycleBeanPostProcessor">
            <property name="proxyTargetClass" value="true" />
        </bean>
        <bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
            <property name="securityManager" ref="securityManager" />
        </bean>
</beans>
```

其中跳转页面已经在注释中指出，这里应该介绍下过滤器配置 filterChainDefinitions：

①Shiro 验证 URL 时,URL 匹配成功便不再继续匹配查找(所以要注意配置文件中的 URL 顺序,尤其在使用通配符时)，故 filterChainDefinitions 的配置顺序为自上而下,以最上面的为准

②当运行一个 Web 应用程序时,Shiro 将会创建一些有用的默认 Filter 实例,并自动地在[main]项中将它们置为可用自动地可用的默认的 Filter 实例是被 DefaultFilter 枚举类定义的,枚举的名称字段就是可供配置的名称

③通常可将这些过滤器分为两组：

anon,authc,authcBasic,user 是第一组认证过滤器

perms,port,rest,roles,ssl 是第二组授权过滤器

注意 user 和 authc 不同：当应用开启了 rememberMe 时,用户下次访问时可以是一个 user,但绝不会是 authc,因为 authc 是需要重新认证的 。user 表示用户

不一定已通过认证,只要曾被 Shiro 记住过登录状态的用户就可以正常发起请求,比如 rememberMe。

实际上就是一个用户登录时开启了 rememberMe,然后他关闭浏览器,下次再访问时他就是一个 user,而不会 authc。

3）web.xml 配置引入对应的配置文件和过滤器

```
<!-- 读取 spring 和 shiro 配置文件 -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:application.xml,classpath:shiro/spring-shiro.xml</param-value>
</context-param>

<!-- shiro 过滤器 -->
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
        <param-name>targetFilterLifecycle</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>*.jhtml</url-pattern>
    <url-pattern>*.json</url-pattern>
</filter-mapping>
```

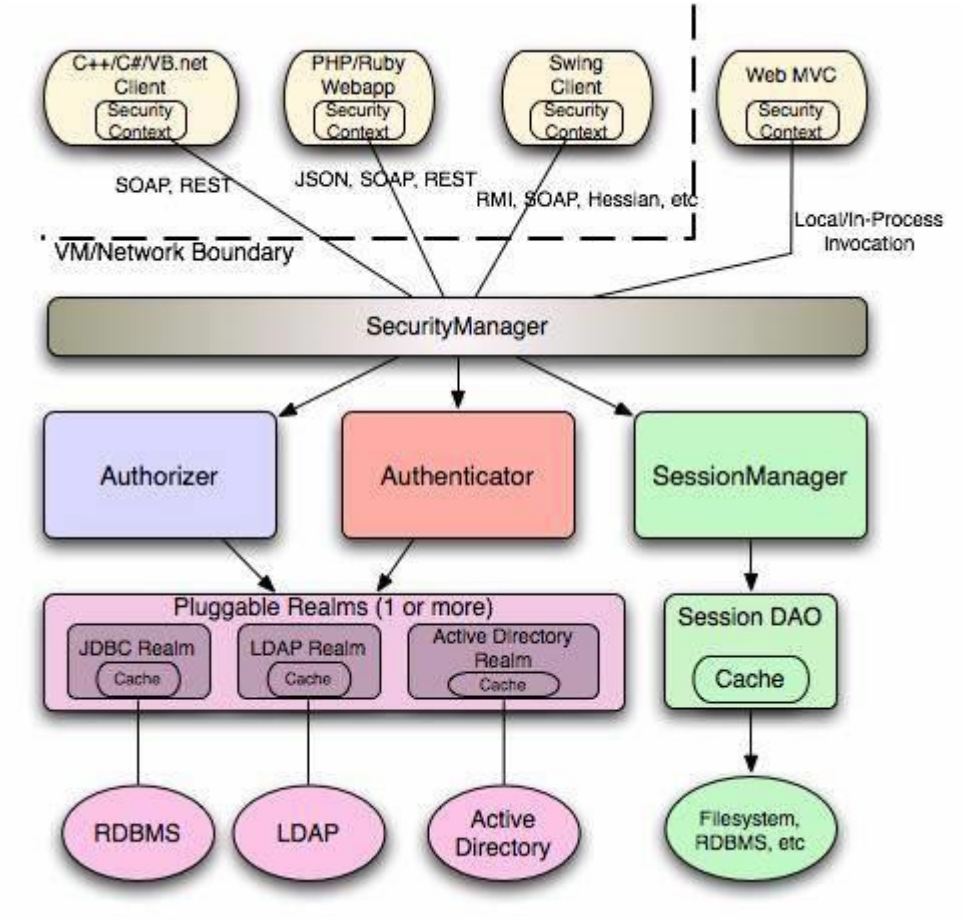
4）controller 代码(package.com.controller)

控制跳转到登录界面、成功登录界面 ;并且使用 shiro 来验证用户名和密码,退出登录。其中借用了网上的工具类 DecriptUtil 对密码进行 md5 加密解密。

5）login 等 jsp 文件，就是登录和成功登录的界面，实现页面跳转，如果失败则弹出调用失败、

3． 对 shiro 的理解：

Shiro 的整体构架：



Shiro 主要有四个组件：

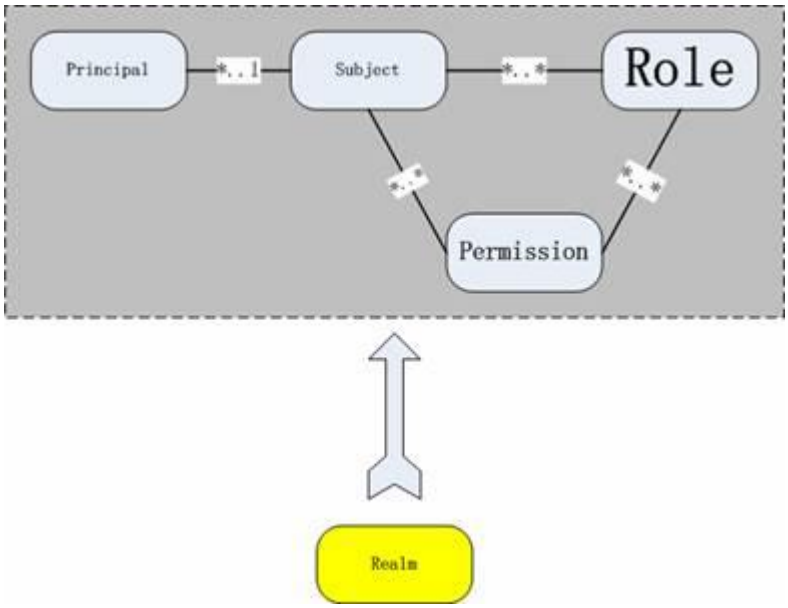
1. SecurityManager：典型的 Facade，Shiro 通过它对外提供安全管理的各种服务。
2. Authenticator：对身份进行核实，通常涉及用户名和密码。这个组件负责收集 principals 和 credentials，并将它们提交给应用系统。如果提交的 credentials 跟应用系统中提供的 credentials 吻合，就能够继续访问，否则需要重新提交 principals 和 credentials，或者直接终止访问。
3. Authorizer：身份份验证通过后，由这个组件对登录人员进行访问控制的筛查，比如“who can do what”， 或者“who can do which actions”。

Shiro 采用“基于 Realm”的方法，即用户、用户组、角色和 permission 的聚合体。

4. Session Manager

这个组件保证了异构客户端的访问，配置简单。它是基于 POJO/J2SE 的，不跟任何的客户端或者协议绑定。

Shiro 的安全模式



1）Subject 是安全领域术语，除了代表人，它还可以是应用。在单应用中，可将其视为 User 的同义词。

2）Principal 是 Subject 的标识，一般情况下是唯一标识，比如用户名。

3）Role 和 Permission 分别代表了不同粒度的权限，从上图中可以看出 Role 的粒度更大些，Permission 代表了系统的原子权限，比如数据的修改、删除权限。对于简单的权限应用，可以不需要 Permission。

4）Realm 是一个执行者，负责真正的认证和鉴权。

实现应用的安全模块的关键在于：定义合适的 role 和 permission，这就需要遵循如下原则：

1）role 没有实质内容，只是代表一组 permission，目的是为了管理的方便，一般都是动态定义；

2）permission 一般都是预先定义好的，不允许动态改变，除非源代码改动，它才会变化，它是整个安全模块的基础；

3）要使 permission 也能动态定义，并非不可能，但是这将使鉴权非常复杂，甚至可能导致鉴权语句遍布整个程序，得不偿失；

4）当然有一个例外：如果知道 permission 动态定义的规则和鉴权规则，如 Grails 的 fileter 中“${controllerName}:${actionName}:${params.id}”也可实现 permission 的动态定义

Shiro 的具体功能：

（1）身份认证/登录，验证用户是不是拥有相应的身份；

（2）授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；

（3）会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通 JavaSE 环境的，也可以是如 Web 环境的；

（4）加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；

（5）Web 支持，可以非常容易的集成到 Web 环境； Caching：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；

（6）shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；

（7）提供测试支持；

（8）允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；

（9）一次登录后，下次再来的话不用登录了。

一般 thiro 程序执行流程（本次项目的登录系统也类似于这个结构）：

任何人要访问应用中受保护的 URL，首先要通过 Filter 检查用户是否经过认证；对于没有认证的用户会将访问定向到登录页面；对于已经认证的用户，会对用户进行鉴权，这个用户是否具有访问其所提交的 URL 的权限；而管理员可以给角色授权。