shiro

相比有做过企业级开发的童鞋应该都有做过权限安全之类的功能吧，最先开始我采用的是建用户表,角色表,权限表，之后在拦截器中对每一个请求进行拦截，再到数据库中进行查询看当前用户是否有该权限，这样的设计能满足大多数中小型系统的需求。不过这篇所介绍的Shiro能满足之前的所有需求，并且使用简单，安全性高，而且现在越来越的多企业都在使用Shiro，这应该是一个收入的你的技能库。

创建自定义MyRealm类

有关Shiro的基础知识我这里就不过多介绍了，直接来干货，到最后会整合Spring来进行权限验证。

首先在使用Shiro的时候我们要考虑在什么样的环境下使用：

- 登录的验证
- 对指定角色的验证
- 对URL的验证

基本上我们也就这三个需求，所以同时我们也需要三个方法：

1. findUserByUserName(String username)根据username查询用户，之后Shiro会根据查询出来的User的密码来和提交上来的密码进行比对。
2. findRoles(String username)根据username查询该用户的所有角色，用于角色验证。
3. findPermissions(String username)根据username查询他所拥有的权限信息，用于权限判断。

下面我贴一下我的mapper代码(PS:该项目依然是基于之前的SSM，不太清楚整合的请看SSM二)。

很简单只有三个方法，分别对应上面所说的三个方法。对sql稍微熟悉点的童鞋应该都能看懂，不太清楚就拷到数据库中执行一下就行了，数据库的Sql也在我的github上。实体类就比较简单了，就只有四个字段以及get,set方法。我就这里就不贴了，具体可以去github上fork我的源码。

现在就需要创建自定义的MyRealm类，这个还是比较重要的。继承至Shiro的AuthorizingRealm类，用于处理自己的验证逻辑，下面贴一下我的代码：

```
1  package com.crossoverJie.shiro;
2  import com.crossoverJie.pojo.T_user;
3  import com.crossoverJie.service.T_userService;
4  import org.apache.shiro.authc.AuthenticationException;
5  import org.apache.shiro.authc.AuthenticationInfo;
6  import org.apache.shiro.authc.AuthenticationToken;
7  import org.apache.shiro.authc.SimpleAuthenticationInfo;
8  import org.apache.shiro.authz.AuthorizationInfo;
9  import org.apache.shiro.authz.SimpleAuthorizationInfo;
10 import org.apache.shiro.realm.AuthorizingRealm;
11 import org.apache.shiro.subject.PrincipalCollection;
```

```
import javax.annotation.Resource;
import java.util.Set;
/**
 * Created with IDEA
 * Created by ${jie.chen} on 2016/7/14.
 * Shiro自定义域
 */
public class MyRealm extends AuthorizingRealm {
    @Resource
    private T_userService t_userService;
    /**
     * 用于的权限的认证。
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        String username = principalCollection.getPrimaryPrincipal().toString() ;
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo() ;
        Set<String> roleName = t_userService.findRoles(username) ;
        Set<String> permissions = t_userService.findPermissions(username) ;
        info.setRoles(roleName);
        info.setStringPermissions(permissions);
        return info;
    }
    /**
     * 首先执行这个登录验证
     * @param token
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
            throws AuthenticationException {
        //获取用户账号
        String username = token.getPrincipal().toString() ;
        T_user user = t_userService.findUserByUsername(username) ;
        if (user != null){
            //将查询到的用户账号和密码存放到 authenticationInfo用于后面的权限判断。第三个参数随
便放一个就行了。
            AuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(user.getUserName(),user.getPassword(),
                    "a") ;
            return authenticationInfo ;
        }else{
            return  null ;
        }
    }
}
```

继承 `AuthorizingRealm` 类之后就需要覆写它的两个方法，`doGetAuthorizationInfo`,`doGetAuthenticationInfo`，这两个方法的作用我都有写注释，逻辑

也比较简单。

doGetAuthenticationInfo是用于登录验证的，在登录的时候需要将数据封装到Shiro的一个token中，执行shiro的login()方法，之后只要我们将MyRealm这个类配置到Spring中，登录的时候Shiro就会自动的调用doGetAuthenticationInfo()方法进行验证。

哦对了，忘了贴下登录的Controller了：

```
package com.crossoverJie.controller;
import com.crossoverJie.pojo.T_user;
import com.crossoverJie.service.T_userService;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.subject.Subject;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import javax.annotation.Resource;
/**
 * Created with IDEA
 * Created by ${jie.chen} on 2016/7/14.
 * 后台Controller
 */
@Controller
@RequestMapping("/")
public class T_userController {
    @Resource
    private T_userService t_userService ;
    @RequestMapping("/loginAdmin")
    public String login(T_user user, Model model){
        Subject subject = SecurityUtils.getSubject() ;
        UsernamePasswordToken token = new
UsernamePasswordToken(user.getUserName(),user.getPassword()) ;
        try {
            subject.login(token);
            return "admin" ;
        }catch (Exception e){
            //这里将异常打印关闭是因为如果登录失败的话会自动抛异常
//            e.printStackTrace();
            model.addAttribute("error","用户名或密码错误") ;
            return "../../login" ;
        }
    }
    @RequestMapping("/admin")
    public String admin(){
        return "admin";
    }
    @RequestMapping("/student")
    public String student(){
        return "admin" ;
    }
    @RequestMapping("/teacher")
    public String teacher(){
        return "admin" ;
    }
}

```

```
50
51
52
53
54
55
```

主要就是`login()`方法。逻辑比较简单，只是登录验证的时候不是像之前那样直接查询数据库然后返回是否有用户了，而是调用`subject`的`login()`方法，就是我上面提到的，调用`login()`方法时`Shiro`会自动调用我们自定义的`MyRealm`类中的`doGetAuthenticationInfo()`方法进行验证的，验证逻辑是先根据用户名查询用户，如果查询到的话再将查询到的用户名和密码放到`SimpleAuthenticationInfo`对象中，Shiro会自动根据用户输入的密码和查询到的密码进行匹配，如果匹配不上就会抛出异常，匹配上之后就会执行`doGetAuthorizationInfo()`进行相应的权限验证。

`doGetAuthorizationInfo()`方法的处理逻辑也比较简单，根据用户名获取到他所拥有的角色以及权限，然后赋值到`SimpleAuthorizationInfo`对象中即可，Shiro就会按照我们配置的XX角色对应XX权限来进行判断，这个配置在下面的整合中会讲到。

整合Spring

接下来应该是大家比较关系的一步：整合`Spring`。

我是在之前的`Spring SpringMVC Mybatis`的基础上进行整合的。

web.xml配置

首先我们需要在`web.xml`进行配置Shiro的过滤器。

我只贴Shiro部分的，其余的和之前配置是一样的。

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
<!-- shiro过滤器定义 -->
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
        <!-- 该值缺省为false,表示生命周期由SpringApplicationContext管理,设置为true则表示由ServletContainer管理 -->
        <param-name>targetFilterLifecycle</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

配置还是比较简单的，这样会过滤所有的请求。

之后我们还需要在Spring中配置一个`shiroFilter`的bean。

spring-mybatis.xml配置

由于这里配置较多，我就全部贴一下：

```
1
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-
3.1.xsd">
  <!-- 自动扫描 -->
  <context:component-scan base-package="com.crossoverJie" />
  <!-- 引入配置文件 -->
  <bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location" value="classpath:jdbc.properties" />
  </bean>
  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
     init-method="init" destroy-method="close">
    <!-- 指定连接数据库的驱动 -->
    <property name="driverClassName" value="${jdbc.driverClass}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.user}" />
    <property name="password" value="${jdbc.password}" />
    <!-- 配置初始化大小、最小、最大 -->
    <property name="initialSize" value="3" />
    <property name="minIdle" value="3" />
    <property name="maxActive" value="20" />
    <!-- 配置获取连接等待超时的时间 -->
    <property name="maxWait" value="60000" />
    <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
    <property name="timeBetweenEvictionRunsMillis" value="60000" />
    <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
    <property name="minEvictableIdleTimeMillis" value="300000" />
    <property name="validationQuery" value="SELECT 'x'" />
    <property name="testWhileIdle" value="true" />
    <property name="testOnBorrow" value="false" />
    <property name="testOnReturn" value="false" />
    <!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->
    <property name="poolPreparedStatements" value="true" />
    <property name="maxPoolPreparedStatementPerConnectionSize"
          value="20" />
    <!-- 配置监控统计拦截的filters，去掉后监控界面sql无法统计 -->
    <property name="filters" value="stat" />
  </bean>
  <!-- spring和MyBatis完美整合，不需要mybatis的配置映射文件 -->
  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <!-- 自动扫描mapping.xml文件 -->
    <property name="mapperLocations" value="classpath:mapping/*.xml">
</property>
  </bean>
  <!-- DAO接口所在包名，Spring会自动查找其下的类 -->
  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.crossoverJie.dao" />
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory">
</property>
  </bean>
  <!-- (事务管理)transaction manager, use JtaTransactionManager for global tx -->
```

```
    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
      <property name="dataSource" ref="dataSource" />
    </bean>
    <!-- 配置自定义Realm -->
    <bean id="myRealm" class="com.crossoverJie.shiro.MyRealm"/>
    <!-- 安全管理器 -->
    <bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
      <property name="realm" ref="myRealm"/>
    </bean>
    <!-- Shiro过滤器 核心-->
    <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
      <!-- Shiro的核心安全接口,这个属性是必须的 -->
      <property name="securityManager" ref="securityManager"/>
      <!-- 身份认证失败，则跳转到登录页面的配置 -->
      <property name="loginUrl" value="/login.jsp"/>
      <!-- 权限认证失败，则跳转到指定页面 -->
      <property name="unauthorizedUrl" value="/nopower.jsp"/>
      <!-- Shiro连接约束配置,即过滤链的定义 -->
      <property name="filterChainDefinitions">
        <value>
          <!--anon 表示匿名访问，不需要认证以及授权-->
          /loginAdmin=anon
          <!--authc表示需要认证 没有进行身份认证是不能进行访问的-->
          /admin*=authc
          /student=roles[teacher]
          /teacher=perms["user:create"]
        </value>
      </property>
    </bean>
    <!-- 保证实现了Shiro内部lifecycle函数的bean执行 -->
    <bean id="lifecycleBeanPostProcessor"
class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>
    <!-- 开启Shiro注解 -->
    <bean
class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator
"
        depends-on="lifecycleBeanPostProcessor"/>
    <bean
class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdviso
r">
      <property name="securityManager" ref="securityManager"/>
    </bean>
</beans>
```

在这里我们配置了上文中所提到的自定义myRealm,这样Shiro就可以按照我们自定义的逻辑来进行权限验证了。其余的都比较简单，看注释应该都能明白。

着重讲解一下：

```
<property name="filterChainDefinitions">
  <value>
    <!--anon 表示匿名访问，不需要认证以及授权-->
    /loginAdmin=anon
    <!--authc表示需要认证 没有进行身份认证是不能进行访问的-->
    /admin*=authc
    /student=roles[teacher]
    /teacher=perms["user:create"]
  </value>
</property>
```

- /loginAdmin=anon的意思的意思是，发起/loginAdmin这个请求是不需要进行身份认证的，这个请求在这次项目中是一个登录请求，一般对于这样的请求都是不需要身份认证的。
- /admin*=authc表示 /admin,/admin1,/admin2这样的请求都是需要进行身份认证的，不然是不能访问的。
- /student=roles[teacher]表示访问/student请求的用户必须是teacher角色，不然是不能进行访问的。
- /teacher=perms[“user:create”]表示访问/teacher请求是需要当前用户具有user:create权限才能进行访问的。

更多相关权限过滤的资料可以访问shiro的官方介绍：传送门

使用Shiro标签库

Shiro还有着强大标签库，可以在前端帮我获取信息和做判断。

我贴一下我这里登录完成之后显示的界面：

要想使用Shiro标签，只需要引入一下标签即可：

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

其实英语稍微好点的童鞋应该都能看懂。下面我大概介绍下一些标签的用法：

- 具有admin角色才会显示标签内的信息。
- 获取用户信息。默认调用Subject.getPrincipal()获取，即 Primary Principal。
- 用户拥有user:create这个权限才回显示标签内的信息。

更多的标签可以查看官网：传送门

标签名称	标签条件（均是显示标签内容）
<shiro:authenticated>	登录之后
<shiro:notAuthenticated>	不在登录状态时
<shiro:guest>	用户在没有RememberMe时
<shiro:user>	用户在RememberMe时
<shiro:hasAnyRoles name="abc,123" >	在有abc或者123角色时
<shiro:hasRole name="abc">	拥有角色abc
<shiro:lacksRole name="abc">	没有角色abc
<shiro:hasPermission name="abc">	拥有权限资源abc
<shiro:lacksPermission name="abc">	没有abc权限资源
<shiro:principal>	默认显示用户名称

MD5加密

Shiro还封装了一个我认为非常不错的功能，那就是MD5加密，代码如下：

```
1  package com.crossoverJie.shiro;
2  import org.apache.shiro.crypto.hash.Md5Hash;
3  /**
4   * Created with IDEA
5   * 基于Shiro的MD5加密
6   * Created by ${jie.chen} on 2016/7/13.
7   */
8  public class MD5Util {
9      public static String md5(String str,String salt){
10         return new Md5Hash(str,salt).toString() ;
11     }
12     public static void main(String[] args) {
13         String md5 = md5("abc123","crossoverjie") ;
14         System.out.println(md5);
15     }
16 }
17
18
19
20
```

代码非常简单，只需要调用`Md5Hash(str,salt)`方法即可，这里多了一个参数，第一个参数不用多解释，是需要加密的字符串。第二个参数`salt`中文翻译叫盐，加密的时候我们传一个字符串进去，只要这个salt不被泄露出去，那原则上加密之后是无法被解密的，在存用户密码的时候可以使用，感觉还是非常屌的。

按钮权限有shiro标签库进行控制

url权限有xml配置文件进行控制