## shiro集成Web项目

### 1.1 pom.xml文件的shiro包的引入（shiro-core、shiro-web、shiro-spring）

|  |
| --- |
| <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>    <!-- 添加Servlet支持 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version>  </dependency>    <dependency>  <groupId>javax.servlet.jsp</groupId>  <artifactId>javax.servlet.jsp-api</artifactId>  <version>2.3.1</version>  </dependency>    <!-- 添加jtl支持 -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>    <!-- 添加Spring支持 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>    <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>4.1.7.RELEASE</version>  </dependency>    <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.2.3</version>  </dependency>  <!-- 添加日志支持 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>    <!-- 添加mybatis支持 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.3.0</version>  </dependency>    <!-- jdbc驱动包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.37</version>  </dependency>    <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-core</artifactId>  <version>1.2.4</version>  </dependency>    <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>1.7.12</version>  </dependency>    <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-web</artifactId>  <version>1.2.4</version>  </dependency>  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>1.2.4</version>  </dependency>    </dependencies> |

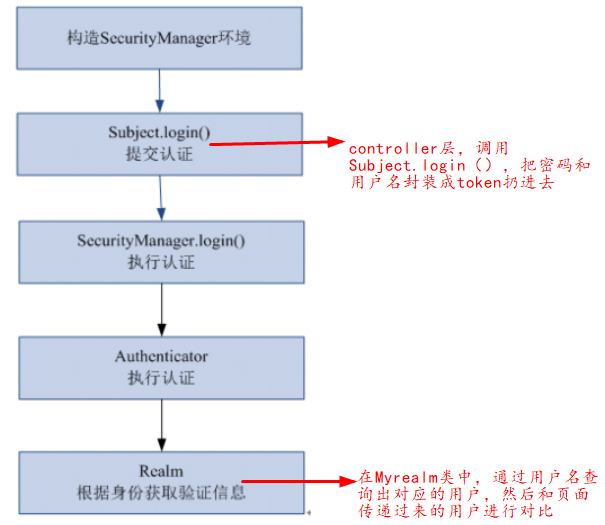
### 1.2 web.xml文件中shiro框架的“shiro过滤配置”

|  |
| --- |
| <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>    <!-- 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>  <!-- 编码过滤器 -->  <filter>  <filter-name>encodingFilter</filter-name> <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- Spring配置文件 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  <!-- Spring监听器 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 添加对springmvc的支持 -->  <servlet>  <servlet-name>springMVC</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring-mvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>    </servlet>  <servlet-mapping>  <servlet-name>springMVC</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping> |

### 1.3 applicationContext.xml的spring文件中，配置shiro的“身份认证”“权限认证”的url

|  |
| --- |
| <!-- 自动扫描 -->  <context:component-scan base-package=*"com.java1234.service"* />  <!-- 配置数据源 -->  <bean id=*"dataSource"*  class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*/>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/db\_shiro"*/>  <property name=*"username"* value=*"root"*/>  <property name=*"password"* value=*"123456"*/>  </bean>  <!-- 配置mybatis的sqlSessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"* />  <!-- 自动扫描mappers.xml文件 -->  <property name=*"mapperLocations"* value=*"classpath:com/java1234/mappers/\*.xml"*></property>  <!-- mybatis配置文件 -->  <property name=*"configLocation"* value=*"classpath:mybatis-config.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.java1234.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>    <!-- 1.自定义Realm（**提供方法——身份验证和权限验证**）-->  <bean id=*"****myRealm****"* class=*"com.java1234.realm.MyRealm"*/>    <!-- 2.安全管理器 -->  <bean id=*"securityManager"* class=*"org.apache.shiro.web.mgt.DefaultWebSecurityManager"*>  <property name=*"realm"* ref=*"****myRealm****"*/>  </bean>  <!-- 3.Shiro过滤器（url配置等） -->  <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（登录）认证失败，则跳转到登录页面的配置 -->**  <property name=*"loginUrl"* value=*"/login.jsp"*/>  **<!-- 权限角色认证失败，则跳转到指定页面 -->**  <property name=*"unauthorizedUrl"* value=*"/unauthor.jsp"*/>  <!-- Shiro连接约束配置,即过滤链的定义 -->  <property name=*"filterChainDefinitions"*>  <value>  /login=anon **//对于login的请求url，无需过滤**  /admin\*=authc**//对于admin的请求，需要身份认证（是否登录）**  /student=roles[teacher]**//对于student的请求，只有教师角色可访问**  /teacher=perms["user:create"]**//只有“user：create”权限可以访问/teacher**  </value>  </property>  </bean>    <!-- 4.保证实现了Shiro内部lifecycle函数的bean执行 -->  <bean id=*"lifecycleBeanPostProcessor"* class=*"org.apache.shiro.spring.LifecycleBeanPostProcessor"*/>    <!-- 5.开启Shiro注解 -->  <bean class=*"org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"* depends-on=*"lifecycleBeanPostProcessor"*/>  <bean class=*"org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor"*>  <property name=*"securityManager"* ref=*"securityManager"*/>  </bean>    <!-- 配置事务通知属性 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <!-- 定义事务传播属性 -->  <tx:attributes>  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"edit\*"* propagation=*"REQUIRED"* />  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"new\*"* propagation=*"REQUIRED"* />  <tx:method name=*"set\*"* propagation=*"REQUIRED"* />  <tx:method name=*"remove\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"change\*"* propagation=*"REQUIRED"* />  <tx:method name=*"check\*"* propagation=*"REQUIRED"* />  <tx:method name=*"get\*"* propagation=*"REQUIRED"* read-only=*"true"* />  <tx:method name=*"find\*"* propagation=*"REQUIRED"* read-only=*"true"* />  <tx:method name=*"load\*"* propagation=*"REQUIRED"* read-only=*"true"* />  <tx:method name=*"\*"* propagation=*"REQUIRED"* read-only=*"true"* />  </tx:attributes>  </tx:advice>    <!-- 配置事务切面 -->  <aop:config>  <aop:pointcut id=*"serviceOperation"*  expression=*"execution(\* com.java1234.service.\*.\*(..))"* />  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"serviceOperation"* />  </aop:config> |

### 1.4 shiro身份认证（controller类提交身份认证、MyRealm类执行身份认证）



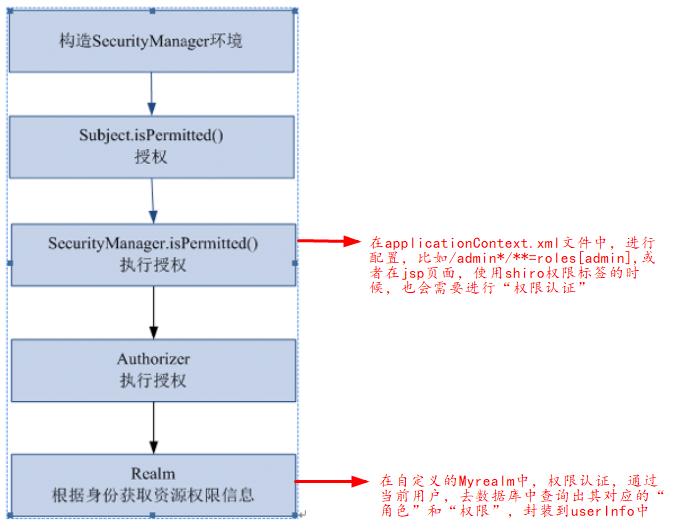
**Controller类中：**

|  |
| --- |
| /\*\*  \* 用户登录  \* **@param** user  \* **@param** request  \* **@return**  \*/  @RequestMapping("/login")  **public** String login(User user,HttpServletRequest request){  Subject subject=SecurityUtils.*getSubject*();**//1.获取当前Subject**  UsernamePasswordToken token=**new// 2.把用户名和密码封装成token** UsernamePasswordToken(user.getUserName(), user.getPassword());  **try**{  subject.login(token);**//3.提交身份认证，失败抛出异常**  Session session=subject.getSession();  System.***out***.println("sessionId:"+session.getId());  System.***out***.println("sessionHost:"+session.getHost());  System.***out***.println("sessionTimeout:"+session.getTimeout());  session.setAttribute("info", "session的数据");  **return** "redirect:/success.jsp";  }**catch**(Exception e){  e.printStackTrace();  request.setAttribute("user", user);  request.setAttribute("errorMsg", "用户名或密码错误！");  **return** "index";  }  } |

**自定义的Myrealm类，“底层会真正的执行身份认证”**

|  |
| --- |
| /\*\*  \* 根据传过来的token，进行验证当前登录的用户  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String userName=(String)token.getPrincipal();//通过token，获取用户名  User user=userService.getByUserName(userName);//通过用户名获取用户  **if**(user!=**null**){  AuthenticationInfo authcInfo=**new** // user和token里面的用户进行对比SimpleAuthenticationInfo(user.getUserName(),user.getPassword(),"xx");  **return** authcInfo;  }**else**{  **return** **null**;  }  } |

### 1.5 shiro权限认证（封装用户的“角色”和“权限”）



**比如success.jsp页面中：**

|  |
| --- |
| <body>  ${info }  欢迎你!  <shiro:hasRole name=*"admin"*> //有admin角色的用户才会显示下面的内容  欢迎有admin角色的用户！<shiro:principal/>  </shiro:hasRole>  <shiro:hasPermission name=*"student:create"*>//有该权限的用户，才可以下载  欢迎有student:create权限的用户！<shiro:principal/>  </shiro:hasPermission>  </body>  </html> |

**在自定义的Myrealm中：**

|  |
| --- |
| **public** **class** MyRealm **extends** AuthorizingRealm{  @Resource  **private** UserService userService;  /\*\*  \* 为当限前登录的用户授予角色和权  \*/  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  **//1.获取当前“身份认证”过的用户**  String userName=(String)principals.getPrimaryPrincipal();  SimpleAuthorizationInfo authorizationInfo=**new** SimpleAuthorizationInfo();  **//2.通过用户名获取该用户所有“角色”，封装起来**  authorizationInfo.setRoles(userService.getRoles(userName));  **//3.通过用户名获取当前用户的所有“权限”，封装起来**  authorizationInfo.setStringPermissions(userService.getPermissions(userName));  **return** authorizationInfo;  }  } |

### 1.6 dao层接口、mapper层映射

Dao层接口：（一个用户有多个角色，一个角色有多个权限）

|  |
| --- |
| **public** **interface** UserDao {  /\*\*  \* 通过用户名查询用户  \* **@param** userName  \* **@return**  \*/  **public** User getByUserName(String userName);    /\*\*  \* 通过用户名查询角色信息  \* **@param** userName  \* **@return**  \*/  **public** Set<String> getRoles(String userName);    /\*\*  \* 通过用户名查询权限信息  \* **@param** userName  \* **@return**  \*/  **public** Set<String> getPermissions(String userName);  } |

Mapper层映射：（内连接查询）

|  |
| --- |
| <select id=*"getByUserName"* parameterType=*"String"* resultMap=*"UserResult"*>  select \* from t\_user where userName=#{userName}  </select>    <select id=*"getRoles"* parameterType=*"String"* resultType=*"String"*>  select r.roleName from t\_user u,t\_role r where u.roleId=r.id and u.userName=#{userName}  </select>    <select id=*"getPermissions"* parameterType=*"String"* resultType=*"String"*>  select p.permissionName from t\_user u,t\_role r,t\_permission p where u.roleId=r.id and p.roleId=r.id and u.userName=#{userName}  </select> |