# Spring 集成 Shiro

作者：Single

原文链接：https://ld246.com/article/1498704081557

来源网站：链滴

web.xml

```
1. 配置  Shiro 的 shiroFilter.
2. DelegatingFilterProxy 实际上是 Filter 的一个代理对象. 默认情况下, Spring 会到 IOC 容器中查
和
<filter-name> 对应的 filter bean. 也可以通过 targetBeanName 的初始化参数来配置 filter bean 的
id.
<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>/*</url-pattern>
</filter-mapping>
```

applicationContext-shiro.xml

```
<!--
1. 配置 SecurityManager!
-->
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager"

    <property name="cacheManager" ref="cacheManager"/>
   <!--  多个realm认证策略 -->
   <!--  <property name="authenticator" ref="authenticator"/> -->
    <property name="realm" ref="jdbcRealm"/>
   <!--  多个realm配置 -->
  <!--   <property name="realms">
        <list>
            <ref bean="jdbcRealm"/>
            <ref bean="secondRealm"/>
        </list>
    </property> -->

    <property name="rememberMeManager.cookie.maxAge" value="10"></property>
</bean>

<!--
2. 配置 CacheManager.
2.1 需要加入 ehcache 的 jar 包及配置文件.
-->
<bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">
    <property name="cacheManagerConfigFile" value="classpath:ehcache.xml"/>
</bean>
 <!--  多个realm认证策略 -->
<!-- <bean id="authenticator"
    class="org.apache.shiro.authc.pam.ModularRealmAuthenticator">
```

```
    <property name="authenticationStrategy">
        <bean class="org.apache.shiro.authc.pam.AtLeastOneSuccessfulStrategy"></bean>
    </property>
</bean> -->

<!--
    3. 配置 Realm
    3.1 直接配置实现了 org.apache.shiro.realm.Realm 接口的 bean
-->
<bean id="jdbcRealm" class="com.atguigu.shiro.realms.ShiroRealm">
    <property name="credentialsMatcher">
        <bean class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">
            <property name="hashAlgorithmName" value="MD5"></property>
            <property name="hashIterations" value="1024"></property>
        </bean>
    </property>
</bean>

<bean id="secondRealm" class="com.atguigu.shiro.realms.SecondRealm">
    <property name="credentialsMatcher">
        <bean class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">
            <property name="hashAlgorithmName" value="SHA1"></property>
            <property name="hashIterations" value="1024"></property>
        </bean>
    </property>
</bean>

<!--
4. 配置 LifecycleBeanPostProcessor. 可以自定的来调用配置在 Spring IOC 容器中 shiro bean 的
命周期方法.
-->
<bean id="lifecycleBeanPostProcessor" class="org.apache.shiro.spring.LifecycleBeanPostProc
ssor"/>

<!-- Enable Shiro Annotations for Spring-configured beans.  Only run after
     the lifecycleBeanProcessor has run: -->
<!--
5. 启用 IOC 容器中使用 shiro 的注解. 但必须在配置了 LifecycleBeanPostProcessor 之后才可以使用.

-->
<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreato
"
      depends-on="lifecycleBeanPostProcessor"/>
<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvis
r">
    <property name="securityManager" ref="securityManager"/>
</bean>

<!--
6. 配置 ShiroFilter.
6.1 id 必须和 web.xml 文件中配置的 DelegatingFilterProxy 的 <filter-name> 一致.
                若不一致, 则会抛出: NoSuchBeanDefinitionException. 因为 Shiro 会来 IOC 容器中查
和 <filter-name> 名字对应的 filter bean.
-->
```

```
<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
    <property name="securityManager" ref="securityManager"/>
    <property name="loginUrl" value="/login.jsp"/>
    <property name="successUrl" value="/list.jsp"/>
    <property name="unauthorizedUrl" value="/unauthorized.jsp"/>
     <!-- 动态配置权限从数据库获取 -->
    <property name="filterChainDefinitionMap" ref="filterChainDefinitionMap"></property>
    <!-- 配置权限 -->
    <!--
       配置哪些页面需要受保护.
       以及访问这些页面需要的权限.
       1). anon 可以被匿名访问
       2). authc 必须认证(即登录)后才可能访问的页面.
       3). logout 登出.
       4). roles 角色过滤器
    -->
    <!--
    <property name="filterChainDefinitions">
        <value>
            /login.jsp = anon
            /shiro/login = anon
            /shiro/logout = logout

            /user.jsp = roles[user]
            /admin.jsp = roles[admin]

            # everything else requires authentication:
            /** = authc
        </value>
    </property>
    -->
</bean>
 <!-- 动态配置权限从数据库获取 -->
<!-- 配置一个 bean, 该 bean 实际上是一个 Map. 通过实例工厂方法的方式 -->
<bean id="filterChainDefinitionMap"
    factory-bean="filterChainDefinitionMapBuilder" factory-method="buildFilterChainDefinitio
Map"></bean>

<bean id="filterChainDefinitionMapBuilder"
    class="com.atguigu.shiro.factory.FilterChainDefinitionMapBuilder"></bean>
```

自定义realm

```
public class ShiroRealm extends AuthorizingRealm {

@Override
protected AuthenticationInfo doGetAuthenticationInfo(
        AuthenticationToken token) throws AuthenticationException {
    System.out.println("[FirstRealm] doGetAuthenticationInfo");

    //1. 把 AuthenticationToken 转换为 UsernamePasswordToken
    UsernamePasswordToken upToken = (UsernamePasswordToken) token;
```

```
    //2. 从 UsernamePasswordToken 中来获取 username
    String username = upToken.getUsername();

    //3. 调用数据库的方法, 从数据库中查询 username 对应的用户记录
    System.out.println("从数据库中获取 username: " + username + " 所对应的用户信息.");

    //4. 若用户不存在, 则可以抛出 UnknownAccountException 异常
    if("unknown".equals(username)){
        throw new UnknownAccountException("用户不存在!");
    }

    //5. 根据用户信息的情况, 决定是否需要抛出其他的 AuthenticationException 异常.
    if("monster".equals(username)){
        throw new LockedAccountException("用户被锁定");
    }

    //6. 根据用户的情况, 来构建 AuthenticationInfo 对象并返回. 通常使用的实现类为: SimpleAuthe
tionInfo
    //以下信息是从数据库中获取的.
    //1). principal: 认证的实体信息. 可以是 username, 也可以是数据表对应的用户的实体类对象.
    Object principal = username;
    //2). credentials: 密码.
    Object credentials = null; //"fc1709d0a95a6be30bc5926fdb7f22f4";
    if("admin".equals(username)){
        credentials = "038bdaf98f2037b31f1e75b5b4c9b26e";
    }else if("user".equals(username)){
        credentials = "098d2c478e9c11555ce2823231e02ec1";
    }

    //3). realmName: 当前 realm 对象的 name. 调用父类的 getName() 方法即可
    String realmName = getName();
    //4). 盐值.
    ByteSource credentialsSalt = ByteSource.Util.bytes(username);

    SimpleAuthenticationInfo info = null; //new SimpleAuthenticationInfo(principal, credentials,
realmName);
    info = new SimpleAuthenticationInfo(principal, credentials, credentialsSalt, realmName);
    return info;
}

public static void main(String[] args) {
    String hashAlgorithmName = "MD5";
    Object credentials = "123456";
    Object salt = ByteSource.Util.bytes("user");;
    int hashIterations = 1024;

    Object result = new SimpleHash(hashAlgorithmName, credentials, salt, hashIterations);
    System.out.println(result);
}

//授权会被 shiro 回调的方法
@Override
protected AuthorizationInfo doGetAuthorizationInfo(
        PrincipalCollection principals) {
```

```
    //1. 从 PrincipalCollection 中来获取登录用户的信息
    Object principal = principals.getPrimaryPrincipal();

    //2. 利用登录的用户的信息来用户当前用户的角色或权限(可能需要查询数据库)
    Set<String> roles = new HashSet<>();
    roles.add("user");
    if("admin".equals(principal)){
        roles.add("admin");
    }

    //3. 创建 SimpleAuthorizationInfo, 并设置其 reles 属性.
    SimpleAuthorizationInfo info = new SimpleAuthorizationInfo(roles);

    //4. 返回 SimpleAuthorizationInfo 对象.
    return info;
}
}
```

FilterChainDefinitionMapBuilder.java

```
public class FilterChainDefinitionMapBuilder {

public LinkedHashMap<String, String> buildFilterChainDefinitionMap(){
    LinkedHashMap<String, String> map = new LinkedHashMap<>();
    //这里可以从数据库读取
    map.put("/login.jsp", "anon");
    map.put("/shiro/login", "anon");
    map.put("/shiro/logout", "logout");
    map.put("/user.jsp", "authc,roles[user]");
    map.put("/admin.jsp", "authc,roles[admin]");
    map.put("/list.jsp", "user");

    map.put("/**", "authc");

    return map;
}
}
```