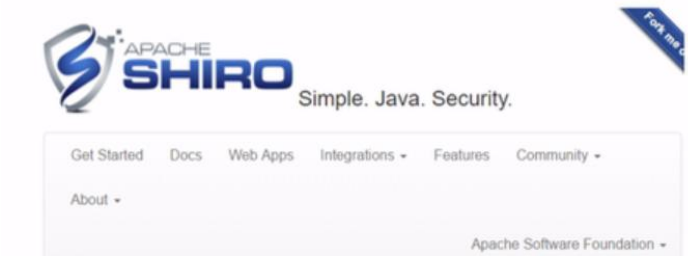
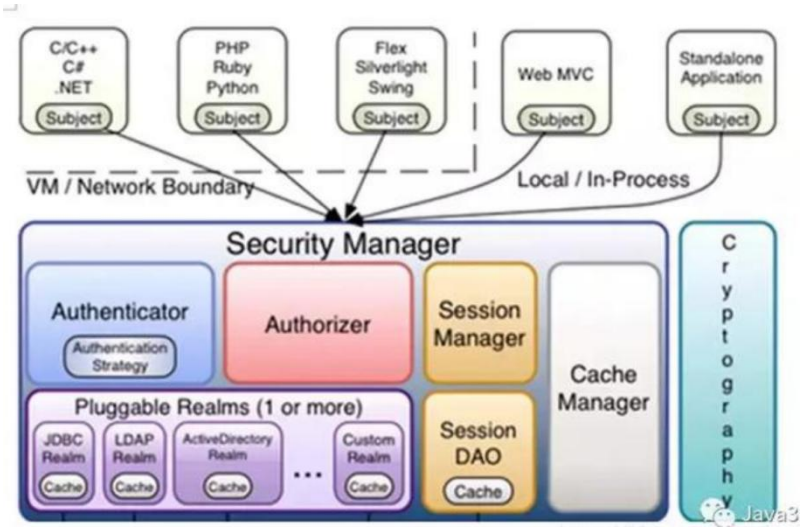
1. aShiro

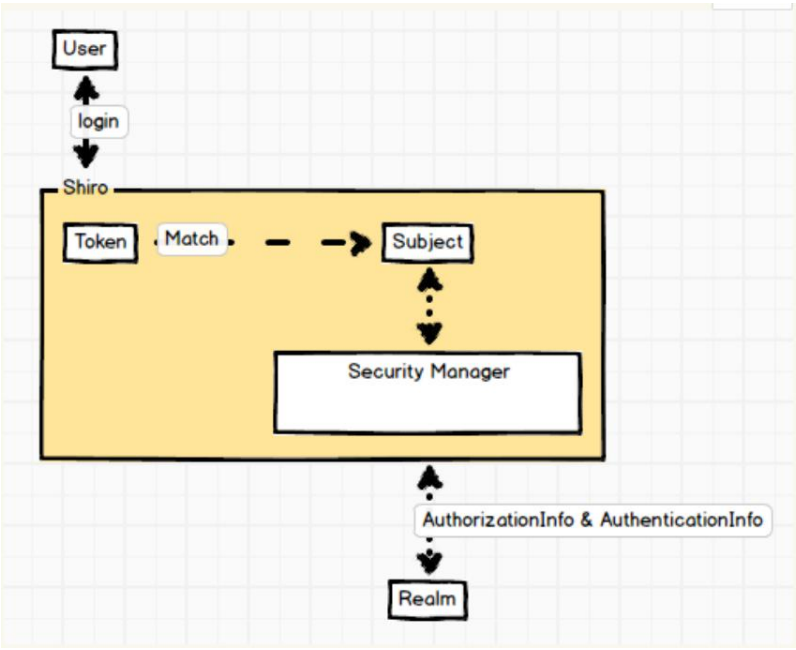
1.1. shiro 简介

- **Apache** Shiro 是 **Java 的一个安全（权限）框架。**
- Shiro 可以非常容易的开发出足够好的应用，其不仅可以用在 **JavaSE** 环境，也可以用在 **JavaEE** 环境。
- Shiro 可以完成：认证、授权、加密、会话管理、与Web 集成、缓存等。
- 下载：http://shiro.apache.org/



1.2. shiro 架构

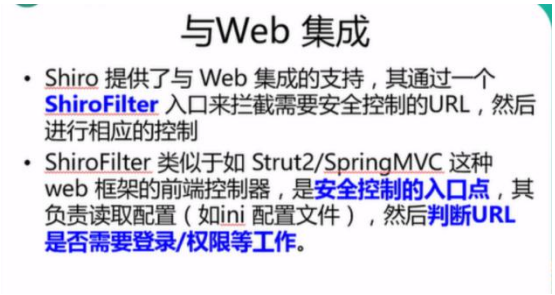




- **Subject**：**应用代码直接交互的对象是 Subject**，也就是说 Shiro 的对外 API 核心就是 Subject。**Subject 代表了当前“用户”**，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是 Subject，如网络爬虫，机器人等；**与 Subject 的所有交互都会委托给 SecurityManager**；**Subject 其实是一个门面，SecurityManager 才是实际的执行者**；
- **SecurityManager**：安全管理器；即**所有与安全有关的操作都会与 SecurityManager 交互**；且其管理着所有 Subject；可以看出它是 **Shiro 的核心**，它**负责与 Shiro 的其他组件进行交互**，它相当于 SpringMVC 中 DispatcherServlet 的角色
- **Realm**：Shiro **从 Realm 获取安全数据（如用户、角色、权限）**，就是说 SecurityManager 要验证用户身份，那么它需要从 Realm 获取相应的用户进行比较以确定用户身份是否合法；也需要从 Realm 得到用户相应的角色/权限进行验证用户是否能进行操作；可以把 Realm 看成 DataSource

- **Subject**：任何可以与应用交互的“用户”；
- **SecurityManager**：相当于SpringMVC 中的 DispatcherServlet；是 Shiro 的心脏；所有具体的交互都通过 SecurityManager 进行控制；它管理着所有 Subject、且负责进行认证、授权、会话及缓存的管理。
- **Authenticator**：**负责 Subject 认证**，是一个扩展点，可以自定义实现；可以使用认证策略（Authentication Strategy），即什么情况下算用户认证通过了；
- **Authorizer**：**授权器**、即访问控制器，用来决定主体是否有权限进行相应的操作；即**控制着用户能访问应用中的哪些功能**；
- **Realm**：可以有 1 个或多个 Realm，可以认为是安全实体数据源，即用于获取安全实体的；可以是JDBC 实现，也可以是内存实现等等；由用户提供；所以一般在应用中都需要实现自己的 Realm；
- **SessionManager**：**管理 Session 生命周期的组件**；而 Shiro 并不仅仅可以用在 Web 环境，也可以用在如普通的 JavaSE 环境
- **CacheManager**：**缓存控制器**，来管理如用户、角色、权限等的缓存的；因为这些数据基本上很少改变，放到缓存中后可以提高访问的性能
- **Cryptography**：**密码模块**，Shiro 提高了一些常见的加密组件用于如密码加密/解密。

2. 与 web 集成



2.1. Shiro 过滤器配置

```
<!-- 配置 Spring 框架提供的用于整合 Shiro 框架的过滤器    shiro 过滤器一定要保证 要先进入！！
        这个过滤器的名字必须与 Spring 配置文件中的 bean 配置 id 相同
    -->

    <filter>
        <filter-name>shiroFilter</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

    </filter>

    <filter-mapping>
        <filter-name>shiroFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

2.2. Shiro-spring.xml 配置

2.2.1. 依赖坐标

```
<!-- https://mvnrepository.com/artifact/org.apache.shiro/shiro-web -->
<dependency>
  <groupId>org.apache.shiro</groupId>
  <artifactId>shiro-web</artifactId>
  <version>1.4.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.shiro/shiro-spring -->
<dependency>
```

```
  <groupId>org.apache.shiro</groupId>
  <artifactId>shiro-spring</artifactId>
  <version>1.4.0</version>
</dependency>
```

2.2.2. 安全管理器

```
<!-- 安全管理器 -->
    <bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <property name="realm" ref="realmDemo"/>
    </bean>
```

2.2.3. Realm 是一个类

```
  <bean id="realmDemo" class="com.hy.shiro.realm.RealmDemo"></bean>
```

```
/**
 * @Auther: wangsq
 * @Date: 2019/7/16 09:33
 * @Description:
 */
public class RealmDemo extends AuthorizingRealm {

    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollec
        return null;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticatio
        return null;
    }
}
```

2.2.4. Shiro 的 web 过滤器 web 中的拦截器拦截以后会交由这个拦截器处理

```
<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <!--必须配置-->
        <property name="securityManager" ref="securityManager" />
        <!-- 如果没有认证访问其他需要认证的连接则跳转到的页面 -->
        <property name="loginUrl" value="/login.jsp" />

        <!--
          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>
            /user/login01=anon
            /login.jsp=anon
            /**=authc
        </value>
    </property>
</bean>

loginUrl：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。
unauthorizedUrl：没有权限默认跳转的页面，登录的用户访问了没有被授权的资源自动跳转到的页面。
其他的一些配置，如下：
successUrl：登录成功默认跳转页面，不配置则跳转至”/”，可以不配置，直接通过代码进行处理。
securityManager：这个属性是必须的，配置为 securityManager 就好了。
filterChainDefinitions：配置过滤规则，从上到下的顺序匹配。
```

这里需要配置一个 bean,此 bean 的名字必须与 web.xml 的 filter-name 相同,spring 代理时就是通过<filter-name>shiroFilter</filter-name>找到代理对象的,如果不配置就会抛出 error,create bean shiroFilter Class not found 的异常,所以我们需要将 shiro 的过滤器注册到 spring 容器中给 spring 代理

2.2.5. 过滤器

当 Shiro 被运用到 web 项目时，Shiro 会自动创建一些默认的过滤器对客户端请求进行过滤。以下是 Shiro 提供的过滤器：

过滤器简称	对应的 Java 类
anon	org.apache.shiro.web.filter.authc.AnonymousFilter
authc	org.apache.shiro.web.filter.authc.FormAuthenticationFilter
authcBasic	org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter
perms	org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter
port	org.apache.shiro.web.filter.authz.PortFilter
rest	org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter
roles	org.apache.shiro.web.filter.authz.RolesAuthorizationFilter
ssl	org.apache.shiro.web.filter.authz.SslFilter
user	org.apache.shiro.web.filter.authc.UserFilter
logout	org.apache.shiro.web.filter.authc.LogoutFilter
noSessionCreation	org.apache.shiro.web.filter.session.NoSessionCreationFilter

解释：

```
/admins/**=anon # 表示该 uri 可以匿名访问
/admins/**=auth # 表示该 uri 需要认证才能访问
/admins/**=authcBasic # 表示该 uri 需要 httpBasic 认证
/admins/**=perms[user:add:*] # 表示该 uri 需要认证用户拥有 user:add:* 权限才能访问
/admins/**=port[8081] # 表示该 uri 需要使用 8081 端口
/admins/**=rest[user] # 相当于 /admins/**=perms[user:method] ,其中，method 表示 get、post、delete 等
/admins/**=roles[admin] # 表示该 uri 需要认证用户拥有 admin 角色才能访问
/admins/**=ssl # 表示该 uri 需要使用 https 协议
/admins/**=user # 表示该 uri 需要认证或通过记住我认证才能访问
/logout=logout # 表示注销,可以当作固定配置
```

注意：

anon，authcBasic，auchc，user 是认证过滤器。

perms，roles，ssl，rest，port 是授权过滤器。

```
/shiro/*=anon
                /accounts/Add.do=anon
                /thymeleaf/AccountAdd.html=anon
                /courses/query_course_ajax.do=anon
<!--                /accounts/queryALl.do=perms[queryALl]-->
                /accounts/**=perms[aaa]
```

3. Shiro 认证思路

```
1. 获取当前的 Subject. 调用 SecurityUtils.getSubject();
2. 测试当前的用户是否已经被认证. 即是否已经登录. 调用 Subject 的 isAuthenticated()
3. 若没有被认证, 则把用户名和密码封装为 UsernamePasswordToken 对象
1). 创建一个表单页面
2). 把请求提交到 SpringMVC 的 Handler
3). 获取用户名和密码.
4. 执行登录: 调用 Subject 的 login(AuthenticationToken) 方法.
5. 自定义 Realm 的方法, 从数据库中获取对应的记录, 返回给 Shiro.
1). 实际上需要继承 org.apache.shiro.realm.AuthenticatingRealm 类
2). 实现 doGetAuthenticationInfo(AuthenticationToken) 方法.
6. 由 shiro 完成对密码的比对.
```

3.1. 登录实现

```
@Controller
public class LoginController {
    @Autowired
    private UserService userService;

    @RequestMapping("login")
    public ModelAndView login(@RequestParam("username") String username, @RequestParam("password") String
password) {
        UsernamePasswordToken token = new UsernamePasswordToken(username, password);
        Subject subject = SecurityUtils.getSubject();
        try {
            subject.login(token);
        } catch (IncorrectCredentialsException ice) {
            // 捕获密码错误异常
            ModelAndView mv = new ModelAndView("error");
            mv.addObject("message", "password error!");
            return mv;
        } catch (UnknownAccountException uae) {
            // 捕获未知用户名异常
            ModelAndView mv = new ModelAndView("error");
            mv.addObject("message", "username error!");
            return mv;
        } catch (ExcessiveAttemptsException eae) {
            // 捕获错误登录过多的异常
            ModelAndView mv = new ModelAndView("error");
            mv.addObject("message", "times error");
            return mv;
        }
        User user = userService.findByUsername(username);
        subject.getSession().setAttribute("user", user);
        return new ModelAndView("success");
    }
}
```

3.2. Realm 实现思路

```
@RequestMapping("/login01")
public String login01(String userName,String password){
    UsernamePasswordToken token=new UsernamePasswordToken(userName, password);
    Subject sub=SecurityUtils.getSubject();
    try {
        sub.login(token);
    } catch(UnknownAccountException e){
        System.out.println(e.getMessage());
    }catch (AuthenticationException e) {
        System.out.println(e.getMessage());
    }
    return "redirect:/list.jsp";
}
```

```java
public class ShiroRealm extends AuthenticatingRealm {

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken token) throws AuthenticationException {
        //1. 把 AuthenticationToken 转换为 UsernamePasswordToken

        //2. 从 UsernamePasswordToken 中来获取 username

        //3. 调用数据库的方法, 从数据库中查询 username 对应的用户记录

        //4. 若用户不存在, 则可以抛出 UnknownAccountException 异常

        //5. 根据用户信息的情况, 决定是否需要抛出其他的 AuthenticationException 异常.

        //6. 根据用户的情况, 来构建 AuthenticationInfo 对象并返回.

        return null;
    }
```

```java
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws Authenticat
    //1、AuthenticationToken转回usernamepasswordtoken 取用户名
    UsernamePasswordToken usernamePasswordToken=(UsernamePasswordToken)authenticationToken;
    String username=usernamePasswordToken.getUsername();
    //2、根据用户名查询数据库
    QueryWrapper queryWrapper=new QueryWrapper();
    queryWrapper.eq( column: "uname",username);
    User user=(User)userService.getOne(queryWrapper);
    //3、用户是否存在
    if(user==null){
        throw new UnknownAccountException("此用户不存在");
    }
    //4、构造AuthenticationInfo对象, 并返回
    SimpleAuthenticationInfo authenticationInfo=new SimpleAuthenticationInfo(user.getUsername(),user.getPasswor
    return authenticationInfo;
}
```

抛出的异常:

org.apache.shiro.authc.IncorrectCredentialsException: Submitted credentials for token [org.apache.shiro.authc.UsernamePasswordToken - hello, rememberMe=false] did not match the expected credentials.

did not match the expected credentials. 与预期的凭证不匹配 其实就是用户名与密码不对

3.3. 密码比对

3.3.1. 密码加密匹配器

```xml
<!--配置-->
<bean id="testRealm" class="com.hy.shiro.realm.TestRealm">
    <property name="credentialsMatcher">
        <bean class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">
            <!--加密方式-->
            <property name="hashAlgorithmName" value="MD5"></property>
            <!--加密次数-->
            <property name="hashIterations" value="1024"></property>
        </bean>
    </property>
</bean>
```

3.3.2. 密码 MD5 加密测试

```
public static void main(String[] args) {
    String hashAlgorithmName = "MD5";//加密方式
    Object credentials = "123456";//要加密的密码
    Object salt = ByteSource.Util.bytes( string: "user");//加的盐
    int hashIterations = 1024;//加密次数

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

3.3.3. 盐值加密

我们如果知道 md5，我们就会知道 md5 是不可逆的，但是如果设置了一些安全性比较低的密码：111111...即时是不可逆的，但还是可以通过暴力算法来得到 md5 对应的明文...

建议对 md5 进行散列时加 salt（盐），进行加密相当 于对原始密码+盐进行散列。

正常使用时散列方法：

在程序中对原始密码+盐进行散列，将散列值存储到数据库中，并且还要将盐也要存储在数据库中。

```
//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: SimpleAuthenticationInfo
//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 = "fc1709d0a95a6be30bc5926fdb7f22f4";
//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);
```

3.4. 多 realm 验证

Realm1:

```
<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>
```

Realm2:

```
<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>
```

SecurityManager:

```
 <bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <property name="cacheManager" ref="cacheManager"/>
        <property name="authenticator" ref="authenticator"></property>

        <property name="realms">
            <list>
                <ref bean="jdbcRealm"/>
                <ref bean="secondRealm"/>
            </list>
        </property>
 </bean>
```

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>
```

- AuthenticationStrategy 接口的默认实现：
- **FirstSuccessfulStrategy**：只要有一个 Realm 验证成功即可，只返回第一个 Realm 身份验证成功的认证信息，其他的忽略；
- **AtLeastOneSuccessfulStrategy**：只要有一个Realm验证成功即可，和 FirstSuccessfulStrategy 不同，将返回所有Realm身份验证成功的认证信息；
- **AllSuccessfulStrategy**：所有Realm验证成功才算成功，且返回所有Realm身份验证成功的认证信息，如果有一个失败就失败了。
- ModularRealmAuthenticator **默认是 AtLeastOneSuccessfulStrategy 策略**

3.5. Session

就是 web 容器帮我们创建的 session.
比如在 service 层取 session 比较方便：

```
SecurityUtils.getSubject().getSession()
```

4. Shiro 授权

4.1. 授权概念

- **授权**，也叫**访问控制，即在应用中控制谁访问哪些资源**（如访问页面/编辑数据/页面操作等）。在授权中需了解的几个关键对象：主体（Subject）、资源（Resource）、权限（Permission）、角色（Role）。
- **主体(Subject)**：访问应用的用户，在 Shiro 中使用 **Subject** 代表该用户。用户只有授权后才允许访问相应的资源。
- **资源(Resource)**：**在应用中用户可以访问的 URL**，比如访问 JSP 页面、查看/编辑某些数据、访问某个业务方法、打印文本等等都是资源。用户只要授权后才能访问。
- **权限(Permission)**：安全策略中的原子授权单位，通过权限我们可以表示在应用中用户有没有操作某个资源的权力。即**权限表示在应用中用户能不能访问某个资源**，如：访问用户列表页面查看/新增/修改/删除用户数据（即很多时候都是CRUD（增查改删）式权限控制）等。权限代表了用户有没有操作某个资源的权利，即反映在某个资源上的操作允不允许。
- Shiro 支持粗粒度权限（如用户模块的所有权限）和细粒度权限（操作某个用户的权限，即实例级别的）
- **角色(Role)**：**权限的集合**，一般情况下会赋予用户角色而不是权限，即这样用户可以拥有一组权限，赋予权限时比较方便。典型的如：项目经理、技术总监、CTO、开发工程师等都是角色，不同的角色拥有一组不同的权限。

4.2. Realm 实现授权

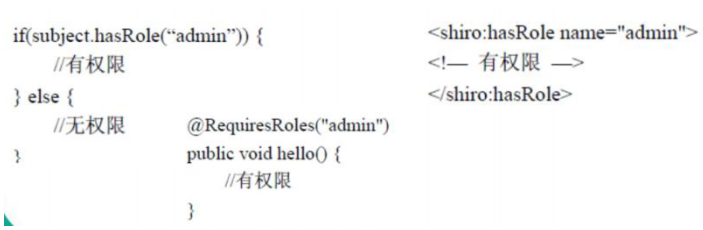
```
/**
 * 授权
 * @param principalCollection
 * @return
 */
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
    //授权
    //1、先拿到用户名
    Object object=principalCollection.getPrimaryPrincipal();
```

```
    System.out.println(object);
    //2、根据用户名查询数据库得到角色和权限
    //角色
    Set<String> roles=new HashSet<String>();
    roles.add("admin");

    //权限
    Set<String> permission=new HashSet<String>();
    permission.add("update");
    permission.add("delete");

    //3、返回授权的信息类
    SimpleAuthorizationInfo authorizationInfo=new SimpleAuthorizationInfo();
    authorizationInfo.setRoles(roles);
    authorizationInfo.addStringPermissions(permission);
    return authorizationInfo;
}
```

4.3. 授权方式



4.3.1. 编程式

通过写 if/else 授权代码块完成，用的不多

4.3.2. 注解

通过在执行的 Java 方法上放置相应的注解完成，没有权限将抛出相应的异常

4.3.3. Jsp 标签

在 JSP 页面通过相应的标签完成。

4.4. 默认拦截器

- Shiro 内置了很多默认的拦截器，比如身份验证、授权等相关的。默认拦截器可以参考 org.apache.shiro.web.filter.mgt.DefaultFilter中的枚举拦截器：

```
public enum DefaultFilter {

    anon(AnonymousFilter.class),
    authc(FormAuthenticationFilter.class),
    authcBasic(BasicHttpAuthenticationFilter.class),
    logout(LogoutFilter.class),
    noSessionCreation(NoSessionCreationFilter.class),
    perms(PermissionsAuthorizationFilter.class),
    port(PortFilter.class),
    rest(HttpMethodPermissionFilter.class),
    roles(RolesAuthorizationFilter.class),
    ssl(SslFilter.class),
    user(UserFilter.class);
```

4.4.1. 认证的拦截器

默认拦截器名	拦截器类	说明（括号里的表示默认值）
身份验证相关的		
authc	org.apache.shiro.web.filter.authc.FormAuthenticationFilter	基于表单的拦截器；如“/**=authc”，如果没有登录会跳到相应的登录页面登录； 主要属性： usernameParam：表单提交的用户名参数名（username）； passwordParam：表单提交的密码参数名（password）； rememberMeParam：表单提交的密码参数名（rememberMe）； loginUrl：登录页面地址（/login.jsp）； successUrl：登录成功后的默认重定向地址； failureKeyAttribute：登录失败后错误信息存储key（shiroLoginFailure）；
authcBasic	org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter	Basic HTTP身份验证拦截器，主要属性： applicationName：弹出登录框显示的信息（application）；
logout	org.apache.shiro.web.filter.authc.LogoutFilter	退出拦截器，主要属性：redirectUrl：退出成功后重定向的地址（/）;示例： “/logout=logout”
user	org.apache.shiro.web.filter.authc.UserFilter	用户拦截器，用户已经身份验证/记住我登录的都可；示例“/**=user”
anon	org.apache.shiro.web.filter.authc.AnonymousFilter	匿名拦截器，即不需要登录即可访问；一般用于静态资源过滤；示例：“/static/**=anon”

4.4.2. 授权拦截器

授权相关的		
roles	org.apache.shiro.web.filter.authz.RolesAuthorizationFilter	角色授权拦截器，验证用户是否拥有所有角色； 主要属性： loginUrl：登录页面地址（/login.jsp）； unauthorizedUrl：未授权后重定向的地址； 示例：“/admin/**=roles[admin]”
perms	org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter	权限授权拦截器，验证用户是否拥有所有权限；属性和roles一样；示例 “/user/**=perms[“user:create”]”
port	org.apache.shiro.web.filter.authz.PortFilter	端口拦截器，主要属性：port（80）：可以通过的端口；示例“/test= port[80]”，如果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80端口，其他路径/参数等都一样
rest	org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter	rest风格拦截器，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=read,OPTIONS=read, MKCOL=create）构建权限字符串；示例：“/users=rest[user]”，会自动拼出“user:read,user:create,user:update,user:delete”权限字符串进行权限匹配（所有都得匹配，isPermittedAll）；
ssl	org.apache.shiro.web.filter.authz.SslFilter	SSL拦截器，只有请求协议是https才能通过；否则自动跳转会https端口（443）；其他和port拦截器一样；

4.5. Shiro 标签

4.5.1. 导入标签库

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

4.5.2. 标签

```
<shiro:guest>
    游客访问 <a href = "login.jsp"></a>
</shiro:guest>
```

user 标签：用户已经通过认证\记住我 登录后显示响应的内容

```
<shiro:user>
    欢迎[<shiro:principal/>]登录 <a href = "logout">退出</a>
</shiro:user>
```

authenticated 标签：用户身份验证通过，即 Subjec.login 登录成功 不是记住我登录的

```
<shiro:authenticted>
    用户[<shiro:principal/>] 已身份验证通过
</shiro:authenticted>
```

notAuthenticated 标签：用户未进行身份验证，即没有调用 Subject.login 进行登录,包括"记住我"也属于未进行身份验证

```
<shiro:notAuthenticated>
    未身份验证(包括"记住我")
</shiro:notAuthenticated>
```

principal 标签：显示用户身份信息，默认调用

Subjec.getPrincipal()获取，即 Primary Principal

```
<shiro:principal property = "username"/>
```

hasRole 标签：如果当前 Subject 有角色将显示 body 体内的内容

```
<shiro:hashRole name = "admin">
    用户[<shiro:principal/>]拥有角色 admin
</shiro:hashRole>
```

hasAnyRoles 标签：如果 Subject 有任意一个角色(或的关系)将显示 body 体里的内容

```
<shiro:hasAnyRoles name = "admin,user">
    用户[<shiro:pricipal/>]拥有角色 admin 或者 user
</shiro:hasAnyRoles>
```

lacksRole:如果当前 Subjec 没有角色将显示 body 体内的内容

```
<shiro:lacksRole name = "admin">
    用户[<shiro:pricipal/>]没有角色 admin
</shiro:lacksRole>
```

hashPermission:如果当前 Subject 有权限将显示 body 体内容

```
<shiro:hashPermission name = "user:create">
    用户[<shiro:pricipal/>] 拥有权限 user:create
</shiro:hashPermission>
```

lacksPermission:如果当前 Subject 没有权限将显示 body 体内容

```
<shiro:lacksPermission name = "org:create">
    用户[<shiro:pricipal/>] 没有权限 org:create
</shiro:lacksPermission>
```

4.6. 权限注解

可以用在 controller 上或者用在 service 上。

- **@RequiresAuthentication**：表示当前Subject已经通过login进行了身份验证；即 **Subject. isAuthenticated() 返回 true**
- **@RequiresUser**：表示当前 Subject 已经**身份验证或者通过记住我登录的。**
- **@RequiresGuest**：表示当前Subject没有身份验证或通过记住我登录过，即是**游客身份。**
- **@RequiresRoles**(value={ “admin” , “user” }, logical= Logical.AND)：表示当前 Subject **需要角色** admin 和user
- **@RequiresPermissions** (value={ “user:a” , “user:b” }, logical= Logical.OR)：表示当前 Subject **需要权限** user:a 或 user:b。

@RequiresAuthenthentication:表示当前 Subject 已经通过 login 进行身份验证;即 Subjec.isAuthenticated()返回 true

@RequiresUser:表示当前 Subject 已经身份验证或者通过记住我登录的,

@RequiresGuest:表示当前 Subject 没有身份验证或者通过记住我登录过，即是游客身份

@RequiresRoles(value = {"admin","user"},logical = Logical.AND):表示当前 Subject 需要角色 admin 和 user

@RequiresPermissions(value = {"user:delete","user:b"},logical = Logical.OR):表示当前 Subject 需要权限 user:delete 或者 user:b

注意一下配置要配置 springmvc.xml 里面:配置以后才可以读取注解

```
<!-- 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 的注解，实现对 Controller 的方法级权限检查(如
@RequiresRoles,@RequiresPermissions),需借助 SpringAOP 扫描使用 Shiro 注解的类,并在必要时进行安全逻辑验
证-->
        <!--配置以下两个 bean 即可实现此功能 -->
        <!--Enable Shiro Annotations for Spring-configured beans. Only run after
thelifecycleBeanProcessor has run -->
        <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>
```

4.7. Springmvc 针对 shiro 认证失败异常处理

全局配置

```
<!--         <error-page>-->
<!--           <error-code>401</error-code>-->
<!--           <location>/thymeleaf/errers.html</location>-->
<!--         </error-page>-->
```

Spring-shiro 中配置在拦截器中配置

```
<!--    配置拦截器-->
    <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<!--        引入安全管理器-->
        <property name="securityManager" ref="shiroManager"></property>

<!--        当未登录需要认证时跳转到的页面-->
        <property name="loginUrl" value="/thymeleaf/login.html"></property>
<!--        当没有权限时跳转的页面-->
        <property name="unauthorizedUrl" value="/thymeleaf/errers.html"></property>
<!--        配置哪些页面需要受保护. 以及访问这些页面需要的权限.-->
        <property name="filterChainDefinitions">
            <value>
                /shiro/*=anon
                /accounts/Add.do=anon
```

```
                /thymeleaf/AccountAdd.html=anon
                /courses/query_course_ajax.do=anon
<!--                /accounts/queryAll.do=perms[queryAll]-->
                /accounts/**=perms[aaa]
                /**=roles
                /**=authc
            </value>
        </property>
    </bean>
```

如果没有权限访问某一个 controller 的方法，则让 springmvc 全局异常处理：以下配置配在 springmvc.xml 里面。

```
<!-- 配置 SpringMVC 的异常解析器 -->
        <bean
class="org.springframework.web.servlet.handler.SimpleMappingException
Resolver">
            <property name="exceptionMappings">
                <props>
                    <!-- 发生授权异常时，跳到指定页 -->
                    <prop
key="org.apache.shiro.authz.UnauthorizedException">/error</prop>

                    <!--SpringMVC 在超出上传文件限制时，会抛出
org.springframework.web.multipart.MaxUploadSizeExceededException-->
                    <!--遇到 MaxUploadSizeExceededException 异常时，自动跳转
到/WEB-INF/error_fileupload.jsp 页面-->
                                <!--
<propkey="org.springframework.web.multipart.MaxUploadSizeExceededExce
ption">WEB-INF/error_fileupload</prop>-->
                </props>
            </property>
        </bean>
```

5. 记住我功能

5.1. 登录页面

```
<form action="${pageContext.request.contextPath }/login" method="post" id="loginForm" >
        用户名：<input type="text" name="username"><br>
        密　　码：<input type="password" name="pazzword" id="pazzword"><br>
        <input type="checkbox" name="rememberme" value="1"> 记住我7天<br>
        <button type="button" onclick="checkForm()">登录</button>
</form>
```

5.2. 登录方法

```
@PostMapping("/login")
public String login(User user, HttpSession session, Integer rememberme) {
        //使用 shiro 登录验证
        //1 认证的核心组件：获取 Subject 对象
        Subject subject = SecurityUtils.getSubject();

        //2 将登陆表单封装成 token 对象
        UsernamePasswordToken token = new UsernamePasswordToken(user.getUsername(), use
        //开启记住我功能
        if(rememberme != null && rememberme == 1) {
                token.setRememberMe(true);
        }
        try {
                //3 让 shiro 框架进行登录验证：
                subject.login(token);
        } catch (Exception e) {
                e.printStackTrace();
                return "loginError";
        }
        return "redirect:/admin/index";
}
```

5.3. 在 spring-shiro.xml 配置 cookie 失效时间

6. 缓存

6.1. 为什么使用缓存

在没有使用缓存的情况下，我们每次发送请求都会调用一次 doGetAuthorizationInfo 方法来进行用户的授权操作，但是我们知道，一个用户具有的权限一般不会频繁的修改，也就是每次授权的内容都是一样的，所以我们希望在用户登录成功的第一次授权成功后将用户的权限保存在缓存中，下一次请求授权的话就直接从缓存中获取，这样效率会更高一些。

6.2. 使用 ehcache 实现缓存

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-ehcache</artifactId>
    <version>1.2.3</version>
</dependency>
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache-core</artifactId>
    <version>2.5.0</version>
</dependency>
```

6.3. 添加 Ehcache 的配置文件

```
<?xml version="1.0" encoding="UTF-8"?><ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

            xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">

            <!--diskStore：缓存数据持久化的目录 地址  -->

            <diskStore path="C:\tools\ehcache" />

            <!--

            eternal：缓存中对象是否为永久的，如果是，超时设置将被忽略，对象从不过期。

            maxElementsInMemory：缓存中允许创建的最大对象数

            overflowToDisk：内存不足时，是否启用磁盘缓存。

            timeToIdleSeconds：缓存数据的钝化时间，也就是在一个元素消亡之前，    两次访问时间的最大时间间隔值，这只能在元素不是永久驻留时有效，如果该值是 0 就意味着元素可以停顿无穷长的时间。
```

```
            timeToLiveSeconds：缓存数据的生存时间，也就是一个元素从构建到消亡的最大时间间隔值，这只能在元素不是永久驻留时有效，如果该值是 0 就意味着元素可以停顿无穷长的时间。

            memoryStoreEvictionPolicy：缓存满了之后的淘汰算法。

            diskPersistent：设定在虚拟机重启时是否进行磁盘存储，默认为 false

            diskExpiryThreadIntervalSeconds： 属性可以设置该线程执行的间隔时间(默认是 120 秒，不能太小

            1 FIFO，先进先出

            2 LFU，最少被使用，缓存的元素有一个 hit 属性，hit 值最小的将会被清出缓存。

            3 LRU，最近最少使用的，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。

            -->

            <defaultCache

                        maxElementsInMemory="1000"

                        maxElementsOnDisk="10000000"

                        eternal="false"

                        overflowToDisk="false"

                        diskPersistent="false"

                        timeToIdleSeconds="120"

                        timeToLiveSeconds="120"

                        diskExpiryThreadIntervalSeconds="120"

                        memoryStoreEvictionPolicy="LRU">

            </defaultCache></ehcache>
```

6.4. 在 shiro 的配置文件中配置缓存

```
<!-- 配置缓存管理器 -->
<bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">
    <!-- 关联配置文件 -->
    <property name="cacheManagerConfigFile" value="classpath:shiro-ehcache.xml"/>
</bean>
```

```xml
<!-- 安全管理器 -->
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
    <property name="realm" ref="realmDemo"/>
    <property name="cacheManager" ref="cacheManager"></property>
</bean>
```

6.5. 清空缓存

在自定义realm中添加清空方法

```java
/**
 * 清空缓存
 */
public void clearCache(){
  PrincipalCollection principals = SecurityUtils.getSubject().getPrincipals();
  super.clearCache(principals);
}
```

```java
    @Resource
    private MyRealm realm;

    /**
     * 清空缓存
     */
    @RequestMapping("/clear")
    @ResponseBody
    public void clearCache(){
        realm.clearCache();
    }
```

清空缓存

7. Shiro 会话管理

Shiro 提供了完整的企业级会话管理功能，**不依赖于底层容器**（如web容器tomcat），**不管 JavaSE 还是 JavaEE 环境都可以使用**，提供了会话管理、会话事件监听、会话存储/持久化、容器无关的集群、失效/过期支持、对Web 的透明支持、SSO 单点登录的支持等特性。

- **Subject.getSession()**：即可获取会话；其等价于Subject.getSession(true)，即如果当前没有创建 Session 对象会创建一个；Subject.getSession(false)，如果当前没有创建 Session 则返回null
- **session.getId()**：获取当前会话的唯一标识
- **session.getHost()**：获取当前Subject的主机地址
- **session.getTimeout() & session.setTimeout(毫秒)**：获取/设置当前Session的过期时间
- **session.getStartTimestamp() & session.getLastAccessTime()**：获取会话的启动时间及最后访问时间；如果是 JavaSE 应用需要自己定期调用 session.touch() 去更新最后访问时间；如果是 Web 应用，每次进入 ShiroFilter 都会自动调用 session.touch() 来更新最后访问时间。