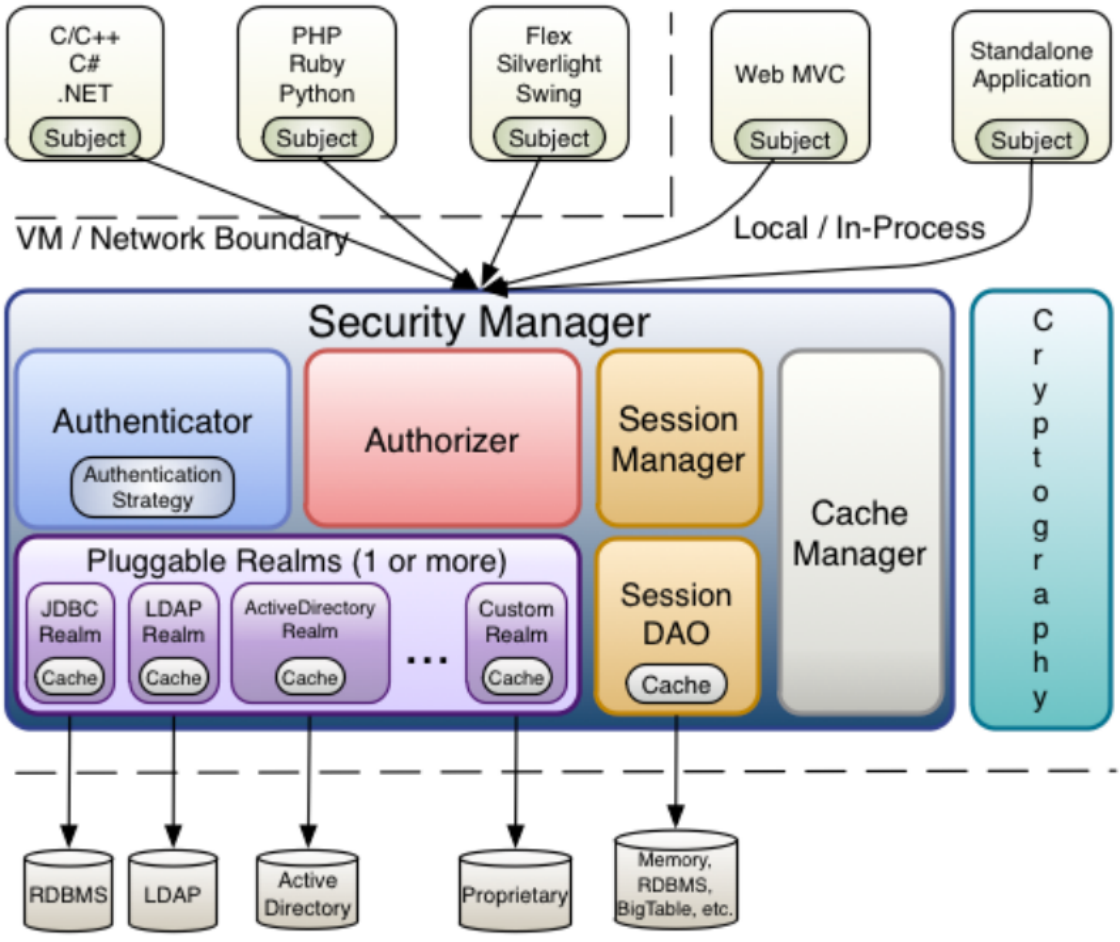
Shiro安全管理框架

Shiro能为我们做什么?

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可以帮助我们完成：认证、授权、加密、会话管理、与Web集成、缓存等。

Shiro的主要组件

Shiro主要组件包括：Subject，SecurityManager(安全管理器)，Authenticator(认证器)，Authorizer(用户访问控制权)， SessionManager(会话管理器)，CacheManager(缓存组件)，Cryptography，Realms(数据连接结果集)。



配置前的问题

首先思考下面几个问题：

1. Shiro认证是在访问系统资源的时候，对用户的身份等进行检查。那么身份检查操作应该在哪里实现呢?

2. 在SE环境中，我们使用的是DefaultSecurityManager来实现认证的控制，那么现在再EE环境中应该使用哪一个SecurityManager来实现认证控制呢?

3. 前面我们在ini配置文件中指定自定义的Realm，现在又是在哪里指定具体使用的Realm?
4. 前端页面需要根据后台认证的结果来处理页面的跳转，如何返回页面需要的数据呢?

Shiro的配置需要哪些

EE项目的配置

web项目的配置

通常我们需要新增一个Shiro.xml来配置Shiro相关的组件

1. 添加依赖

```
        <!--日志文件-->
         <dependency>
             <groupId>commons-logging</groupId>
             <artifactId>commons-logging</artifactId>
             <version>1.1.3</version>
         </dependency>
         <!--Shiro的核心组件-->
         <dependency>
             <groupId>org.apache.shiro</groupId>
             <artifactId>shiro-core</artifactId>
             <version>1.3.2</version>
         </dependency>
         <!--Shiro与web关联的组件-->
         <dependency>
             <groupId>org.apache.shiro</groupId>
             <artifactId>shiro-web</artifactId>
             <version>1.3.2</version>
         </dependency>
         <!--Shiro底层使用的ehcache缓存-->
         <dependency>
             <groupId>org.apache.shiro</groupId>
             <artifactId>shiro-ehcache</artifactId>
             <version>1.3.2</version>
         </dependency>
         <dependency>
             <groupId>commons-collections</groupId>
             <artifactId>commons-collections</artifactId>
             <version>3.2.1</version>
         </dependency>
         <dependency>
             <groupId>net.sf.ehcache</groupId>
```

```
            <artifactId>ehcache-core</artifactId>
            <version>2.6.8</version>
        </dependency>
        <!--Shiro与Freemarker关联的依赖-->
         <dependency>
            <groupId>net.mingsoft</groupId>
            <artifactId>shiro-freemarker-tags</artifactId>
            <version>1.0.0</version>
        </dependency>
```

如果要关联Spring的话需要配置Shiro与Spring关联的依赖

```
        <!--Shiro与Spring关联-->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.3.2</version>
        </dependency>
```

2. 配置Shiro代理过滤器

```
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProx</filter-class>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

作用是用来过滤请求,交由代理对象来完成, 配置这个后, 需要在配置文件中指定配置一个名称和代理过滤器名称一样的真实类型过滤器

3. 指定系统资源需要使用的具体过滤器

```
<bean id="shiroFilter"
 class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
    <!--引用指定的安全管理器-->
    <property name="securityManager" ref="securityManager"/>
    <!--设置登录访问的地址-->
    <property name="loginUrl" value="/login.html"/>
    <property name="filterChainDefinitions"> <value>
        /js/**=anon
        /images/**=anon
        /css/**=anon
        /**=authc
    </value>
    </property>
</bean>
```

4. 解决安全管理器的指定问题，在JavaEE环境中，我们需要使用的安全管理器是：

   DefaultWebSecurityManager，并且在该安全管理器中指定我们自定义的Realm

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

5. 自定义Realm来实现认证器认证处理

   方式如下

6. 自定义认证过滤器来实现认证后的请求处理

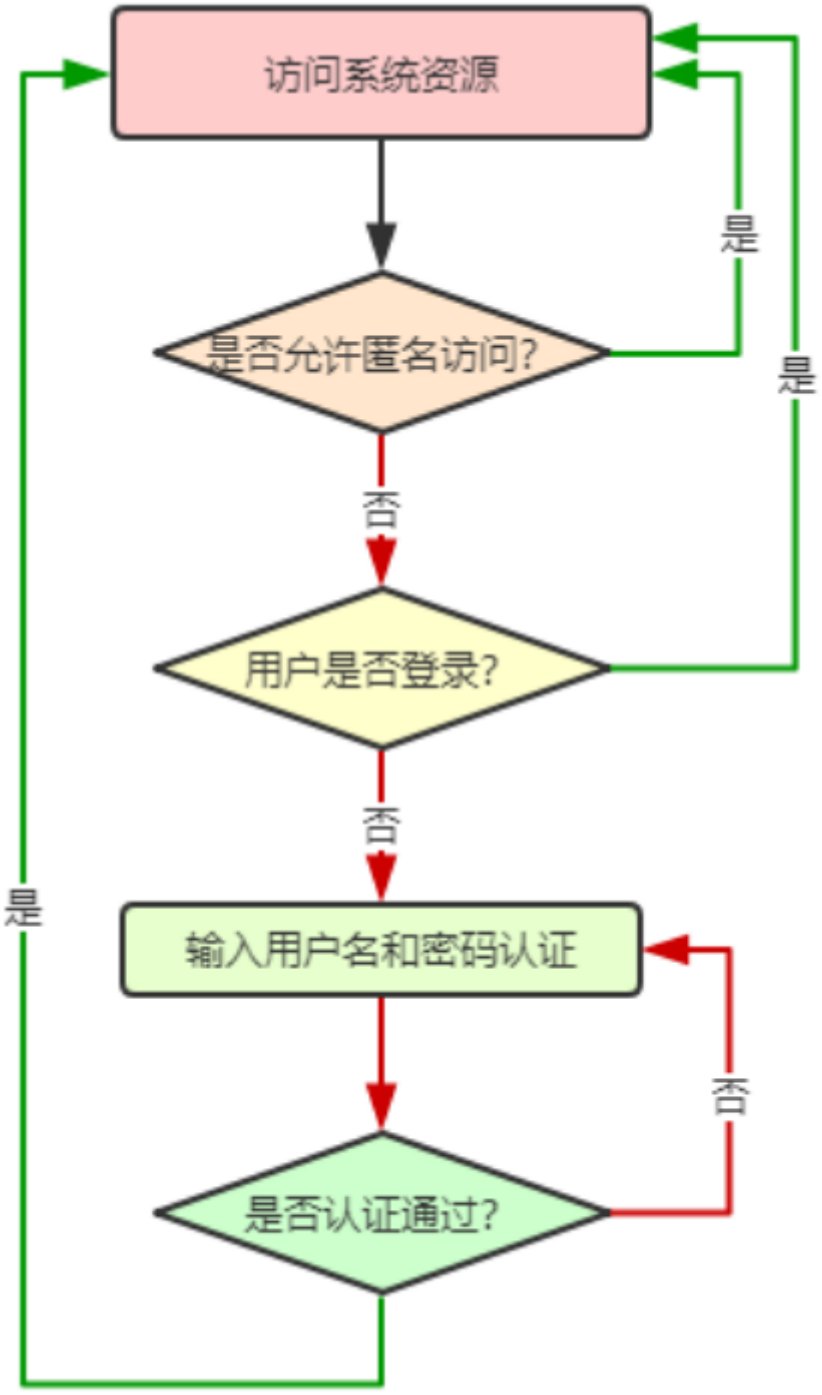
   方式如下

7. 引用当前的认证过滤器

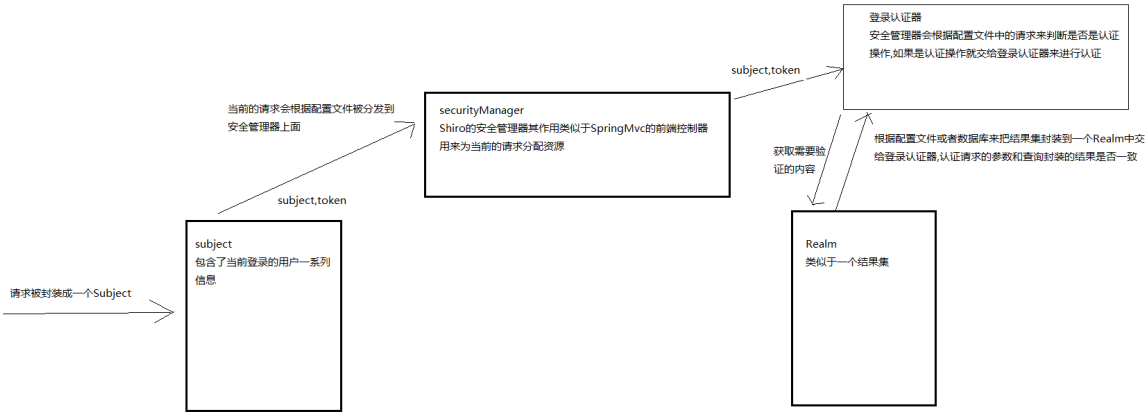
```
<property name="filters">
   <map>
      <!--设置当前的认证过滤器-->
      <entry key="authc" value-ref="crmFormAuthenticationFilter">
</entry>
   </map>
</property>
```

Shrio身份认证的概述

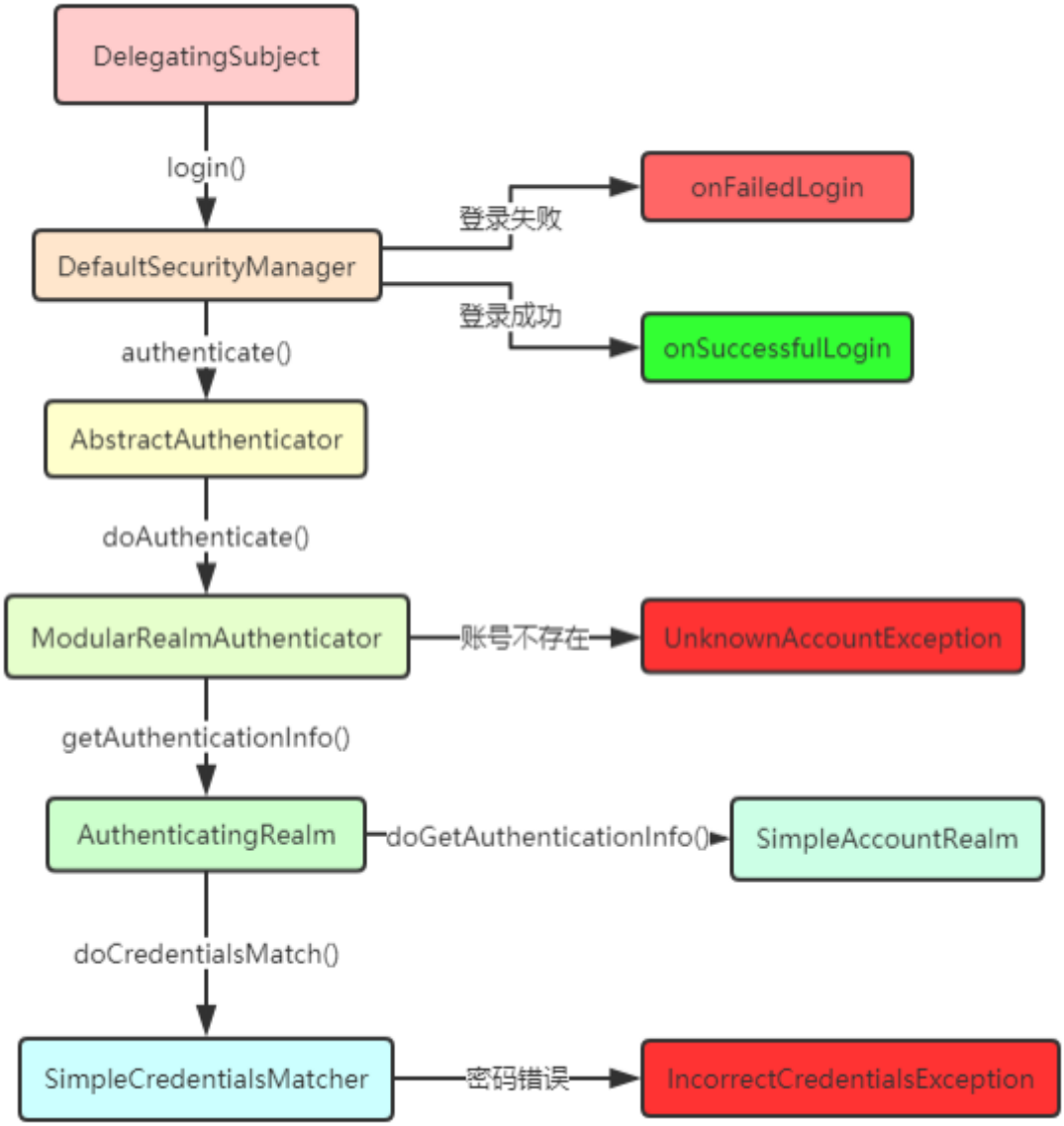
认证的过程即为用户的身份确认过程，所实现的功能就是我们所熟悉的登录验证，用户输入账号和密码提交到后台，后台通过访问数据库执行账号密码的正确性校验。



Shiro如何实现登录验证



认证的执行流程



Shiro需要的配置*

> 根据登录验证的Shiro流程,由此我们可知 , Shiro需要有两个参数,一个配置

1. subject:用户传递的参数
2. Realm: Shiro从我们的数据库或者配置文件读取到的要与用户传递的参数匹配的数据
3. 配置securityManager安全管理器 !(很重要)

自定义Realm

> 为什么要自定义Realm

Shiro自带的匹配Realm不能满足需求，我们需要把数据库里面的数据封装成Realm用作匹配，所以需要使用自定义的Realm

> 我们如何来自定义Realm来实现认证?

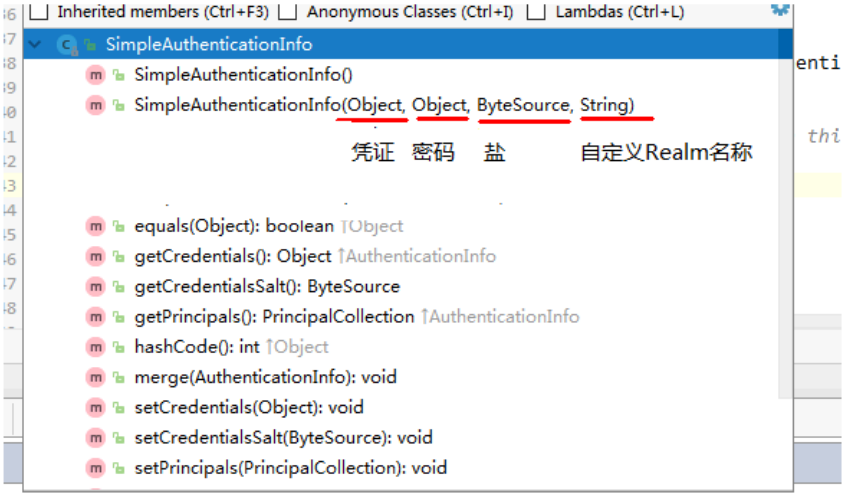
创建自定义Realm类,来继承AuthorizingRealm授权认证Reaml这个类重写内部的两个方法
doGetAuthorizationInfo(授权)和doGetAuthenticationInfo(认证)并且重写GetName方法

在Shiro认证的过程中,Shiro会先根据用户名来查找当前的用户是否存在,如果不存在就抛出用户未知的异常,当用户存在的时候,在进行判断密码

认证的方法

> doGetAuthenticationInfo(AuthenticationToken token)的参数
>
> > 此时的token包含了我们需要的用户的信息 通过`token.getPrincipal()`方法来获取到当前用户登录的用户名
> >
> > ```
> > 1 String username = token.getPrincipal().toString();
> > ```
>
> 返回类型
>
> > 如果查询到的结果确定存在，那么就将查询到的对象发送到一个实体类里面返回
> >
> > ```
> > 1 new SimpleAuthenticationInfo(employee, employee.getPassword(),
> > getName());
> > ```

认证方法的返回值



在认证完毕后Shiro会将此时认证的结果发送到过滤认证器上面

认证过滤器

认证过滤器通过认证器做出的判断认证来设置认证成功或者失败后要做什么事情,因为框架不会知道我们的想法,所以我们需要在实现一个认证过滤器来告诉框架我们在认证成功或者失败后要做什么事情

Shiro中定义了多个过滤器来完成不同的预处理操作:

过滤器的名称	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
roles	org.apache.shiro.web.filter.authz.RolesAuthorizationFilter
perms	org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter
user	org.apache.shiro.web.filter.authc.UserFilter
logout	org.apache.shiro.web.filter.authc.LogoutFilter
port	org.apache.shiro.web.filter.authz.PortFilter
rest	org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter
ssl	org.apache.shiro.web.filter.authz.SslFilter

- 1) 创建一个实现类来继承FormAuthenticationFilter类并重写内部的onLoginFailure(失败)方法 和 onLoginSuccess(成功)方法
- 需要在spring-shiro的配置文件中添加自定义的认证过滤器

```
<property name="filters">
<map>
  <!--设置当前的认证过滤器-->
  <entry key="authc" value-ref="crmFormAuthenticationFilter">
     </entry>
</map>
</property>
```

我们的认证都是在页面通过请求的方式完成的,如何知道我们的认证是否通过,并且通过后访问的路径呢?

- 一般我们的请求都是通过ajax来进行异步访问, 因为这样用户输入错误也不用刷新整个页面, 请求的速度因为传输内容格式为json也比xml格式传输的要快, 所以我们需要在页面提交ajax, 不过在提交的时候, 一般的请求都是直接访问控制器 , 但是如果配置了Shiro后 , 访问的路径需要设置成本身因为我们的Shiro觉得登录的逻辑基本都一致, 所以Shiro将登录的逻辑抽取了出来 , 在用户做ajax提交的时候回根据用户提交请求的方式来判断当前用户是登录还是验证
  - 如果提交的方式为get那么Shiro会认为当前只是访问请求地址并不会做登录认证
  - 如果提交的方式为post那么Shiro会调用安全管理器来交给认证器进行认证的访问

页面使用的是ajax异步提交 , 所以在提交的时候我们希望认证完服务器响应给我们一个json格式的数据前端用来判断是否登录成功

- 我们可以通过阿里的json封装来保证数据传送的格式为json

  导入阿里的 fastjson 依赖然后通过过滤认证器中的response来指定发送数据为什么格式

```
response.setContentType("text/json;charset=UTF-8");
```

  通过response来把这一段json数据写出去

```
response.getWriter().print(JSON.toJSONString(new JsonResult()))
```

- 在浏览器上面 ajax 就会通过当前访问本身后交给shiro的请求回调获取到此刻json的数据

  我们通过判断响应回来的json数据内容是否为真,来进行下一部的逻辑操作

> 代码案例:

```
        JsonResult json = new JsonResult();
        String msg = "";
        if (e instanceof UnknownAccountException) {
            msg = "账号错误";
        } else if (e instanceof IncorrectCredentialsException) {
            msg = "密码错误";
        } else {
            msg = "未知错误";
        }
        json.mark(msg);
        try {
            response.getWriter().write(JSON.toJSONString(json));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
```

```
16          return false;
```

实现注销功能

在Shiro.xml的配置文件中　配置Shior内置的退出控制

/logout.do = logout

在退出的超链接上面 指定 退出的地址 /logout.do

Shrio授权认证

Shiro授权的三种方式

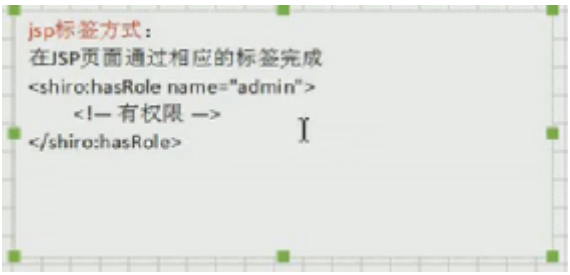
编程式权限访问

在方法的内容代码中判断当前的请求是否包含访问权限

> 这种方式有个缺陷就是按照正常逻辑来判断,如果当前登录的用户没有权限, 不应该让他访问控制器中的方法 , 如果频繁的访问会造成资源的浪费

标签式权限访问

在页面 使用 shiro的标签来隐藏掉那些没有权限的操作



> 这种方式的坏处就是, 如果当前登录的用户是一个开发人员, 可以通过浏览器地址上面的请求访问那些被隐藏的方法

注解式权限访问

在控制器的方法上面贴上Shiro权限认证的注解用来指定当前的方法是否需要权限注入

这种方式目前比较适用，操作简单

注解开发需要的三方参与

如何使用注解来进行权限访问?

使用注解授权需要的配置

1. 开启AOP动态代理

   因为使用shiro注解是需要动态代理,切面的为每一个请求方法进行功能增强

```
<aop:config>
   <aop:pointcut id="txPointcut" expression="execution(*
cn.wolfcode.shiro.service.impl.*.*(..))"/>
   <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>
</aop:config>
```

   第二个

```
<aop:config proxy-target-class="true"></aop:config>
```

   2.在Shiro的配置文件中，赋予注解权限

   开始Shiro的注解支持, 需要第三方程序赋予注解权限控制的功能

> 开启Shiro注解扫描器，当扫描到Controller中有使用@RequiresPermissions注解时，会使用
>
> 动态代理为当前Controller生成代理对象，增强对应方法的权限校验功能。

```
<bean
class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
<!--加载安全管理器-->
  <property name="securityManager" ref="securityManager"/>
</bean>
```

> 不同之处：
>
> 1. 从Spring容器中获取到的Controller对象是代理对象，该对象的类继承自原始的Controller，而注解是贴在原始的Controller类方法上的，所以需要获取到Controller对象（代理对象）的父类（原始Controller）的字节码对象，再获取方法。
> 2. 权限表达式是直接在注解中指定的，不需要执行拼接操作，所以获取到注解value属性值（数组），在获取对应的表达式和名称即可。

3. 在控制器贴上注解指定位置

   权限注解扫描器的作用

扫描类上的权限相关的注解`@RequiresPermissions`，如果发现类中有相关的注解，代理当前的Controller，重写对应的方法，增强其功能(拥有检查权限或者角色权限的功能)

Shiro代理的方式

如果贴了Shiro的权限注解，Shiro将会通过CGLB动态代理来动态的为这个控制器增强方法

权限相关的注解`@RequiresPermissions`不能被子类集成,所以在加载权限的时候需要获取的是父类中的方法中的注解

加载权限与分配权限

判断当前的注解的权限在获取到的当前用户中是否存在

> 在认证器中，如果已经通过了认证，那么认证器会把当前用户的对象传递到授权管理里面
>
> 授权管理在接受到这个对象后，判断他是否是超级管理员，如果是超级管理员就给他所有的权限

> 获取到认证器中授权管理的对象

```
SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
```

> 利用授权管理对象将查询到的权限添加到Shiro内置的权限认证管理里面
>
> 代码示范:

```
Employee employee = (Employee) principals.getPrimaryPrincipal();
     //如果这个用户是admin
     if (employee.getAdmin()) {
         //为其分配所有的权限
         info.addRole("admin");
         info.addStringPermission("*:*");
     } else {
         //获取到当前登录用户的全部角色
         List<String> roles = roleMapper.queryRoles(employee.getId());
         List<String> expression =
roleMapper.queryPermissions(employee.getId());
         info.addRoles(roles);
```

```
            info.addStringPermissions(expression);
        }
    return info
```

分配权限后需要通知Shiro没有权限做什么事情

> 在配置文件中指定Shiro没有权限的后续操作，将没有权限的页面展示给用户

使用统一异常处理来设置跳转的页面

Shiro MD5加密

1.为什么我们要使用MD5加密

为了保证数据的安全,需要将数据库里面的密码都换成MD5加密的格式

2.如何实现MD5标签

Shiro内置了一个类用来将明文转换为MD5格式的文本

> new MD5Hash("密码" , "标识(盐)" , "加密的次数")

配置凭证匹配器

```
<!--指定凭证匹配器-->
<bean class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">
    <!--指定当前的加密算法-->
    <property name="hashAlgorithmName" value="MD5"></property>
</bean>
```

在自定义Realm中加载自定义的凭证匹配

```

    @Autowired
    @Override
    public void setCredentialsMatcher(CredentialsMatcher credentialsMatcher)
{
        super.setCredentialsMatcher(credentialsMatcher);
    }
```

3.应用场景

> 1.在保存员工的时候我们需要将保存进来的员工密码设置为密文格式

找到当前保存的员工 转换密码格式即可

代码示例:

```
//转换为密文
String password = new Md5Hash(employee.getPassword(),
employee.getName()).toString();
//保存数据
employee.setPassword(password);
employeeMapper.insert(employee);
```

2.在登录的时候需要在认证器处将标识也加入进去

在返回值里多加入一个参数即可

代码示例:

```
return new SimpleAuthenticationInfo(employee, employee.getPassword(),
ByteSource.Util.bytes(employee.getName()), getName());
```

配置缓存

1.为什么要使用Shiro的缓存

如果不使用缓存的话，shiro将会对每一个查询结果都查找一次是否具有权限，这样子及其浪费数据库性能

2.如何使用Shiro的缓存

1.使用第三方的缓存框架

添加依赖

```
<dependency>
            <groupId>net.sf.ehcache</groupId>
            <artifactId>ehcache-core</artifactId>
            <version>2.6.8</version>
        </dependency>
<dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-ehcache</artifactId>
            <version>1.3.2</version>
        </dependency>

```

2.配置缓存的管理器

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

3.配置缓存设置的xml文件

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

    <defaultCache
            maxElementsInMemory="1000"
            eternal="false"
            timeToIdleSeconds="600"
            timeToLiveSeconds="1200"
            memoryStoreEvictionPolicy="LRU"/>
</ehcache>
```

4.以往用过的缓存*

Mybatis的一级缓存　　Mybatis的二级缓存

清空缓存

如果用户正常退出，缓存自动清空。

如果用户非正常退出，缓存自动清空。

如果修改了用户的权限，而用户不退出系统，修改的权限无法立即生效。

当用户权限修改后，用户再次登陆shiro会自动调用realm从数据库获取权限数据，如果在修改权 限后想立即清除缓存则可以调用realm的clearCache方法清除缓存。

在realm中定义该方法:

```

public void clearCached() {
    PrincipalCollection principals =
SecurityUtils.getSubject().getPrincipals();
    super.clearCache(principals);
}
```