权限管理框架

尚学堂·百战程序员

1. shiro 介绍

1.1 什么是 shiro

Apache Shiro 是 Java 的一个安全框架。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 可以帮助我们完成：认证、授权、加密、会话管理、与 Web 集成、缓存等。

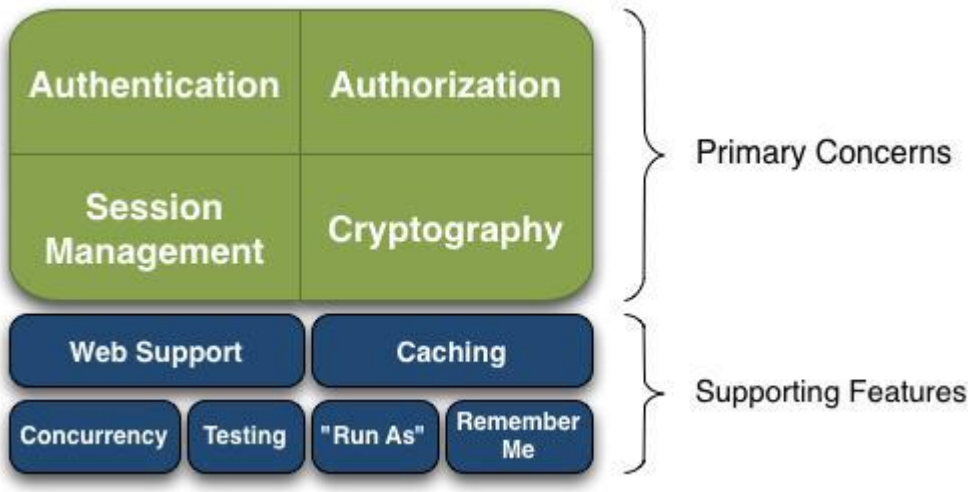
1.2 为什么要学 shiro

既然 shiro 将安全认证相关的功能抽取出来组成一个框架，使用 shiro 就可以非常快速的完成认证、授权等功能的开发，降低系统成本。

shiro 使用广泛，shiro 可以运行在 web 应用，非 web 应用，集群分布式应用中越来越多的用户开始使用 shiro。

java 领域中 spring security(原名 Acegi)也是一个开源的权限管理框架，但是 spring security 依赖 spring 运行，而 shiro 就相对独立，最主要是因为 shiro 使用简单、灵活，所以现在越来越多的用户选择 shiro。

1.3 基本功能



尚学堂·百战程序员

1.3.1 Authentication

身份认证/登录，验证用户是不是拥有相应的身份；

1.3.2 Authorization

授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；

1.3.3 Session Manager

会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通 JavaSE 环境的，也可以是如 Web 环境的；

Cryptography：加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；

Web Support：Web 支持，可以非常容易的集成到 Web 环境；

Caching：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；

Concurrency：shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；

Testing：提供测试支持；

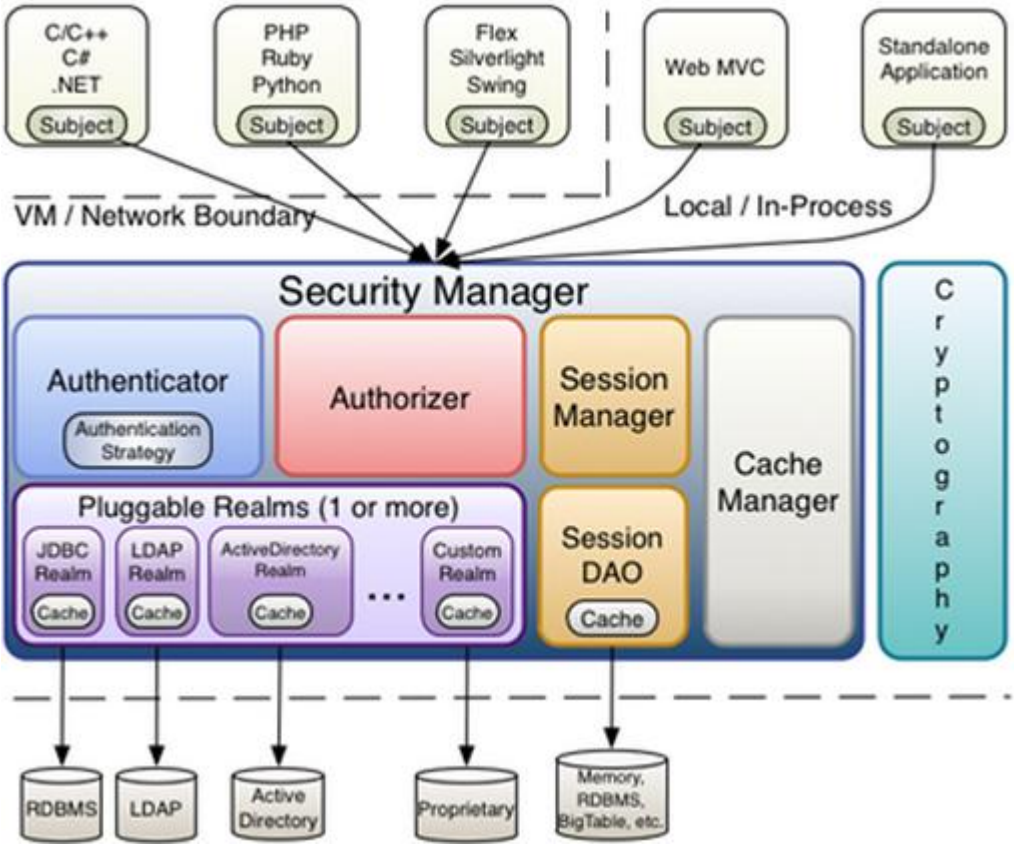
Run As：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；

Remember Me：记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。

Shiro 不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给 Shiro 即可。

1.4 Shiro 架构

尚学堂·百战程序员



1.4.1 Subject

Subject 即主体，外部应用与 subject 进行交互，subject 记录了当前操作用户，将用户的概念理解为当前操作的主体，可能是一个通过浏览器请求的用户，也可能是一个运行的程序。

Subject 在 shiro 中是一个接口，接口中定义了很多认证授相关的方法，外部程序通过 subject 进行认证授，而 subject 是通过 SecurityManager 安全管理器进行认证授权

1.4.2 SecurityManager

SecurityManager 即安全管理器，对全部的 subject 进行安全管理，它是 shiro 的核心，负责对所有的 subject 进行安全管理。通过 SecurityManager 可以完成 subject 的认证、授权等，实质上 SecurityManager 是通过 Authenticator 进行认证，通过 Authorizer 进行授权，通过 SessionManager 进行会话管理等。

SecurityManager 是一个接口，继承了 Authenticator, Authorizer, SessionManager 这三个接口。

1.4.3 Authenticator

Authenticator 即认证器，对用户身份进行认证，Authenticator 是一个接口，shiro 提供 ModularRealmAuthenticator 实现类，通过 ModularRealmAuthenticator 基本上可以满足大多数

尚学堂·百战程序员

需求，也可以自定义认证器。

1.4.4 Authorizer

Authorizer 即授权器，用户通过认证器认证通过，在访问功能时需要通过授权器判断用户是否有此功能的操作权限。

1.4.5 realm

Realm 即领域，相当于 datasource 数据源，securityManager 进行安全认证需要通过 Realm 获取用户权限数据，比如：如果用户身份数据在数据库那么 realm 就需要从数据库获取用户身份信息。

注意：不要把 realm 理解成只是从数据源取数据，在 realm 中还有认证授权校验的相关的代码。

1.4.6 sessionManager

sessionManager 即会话管理，shiro 框架定义了一套会话管理，它不依赖 web 容器的 session，所以 shiro 可以使用在非 web 应用上，也可以将分布式应用的会话集中在一点管理，此特性可使它实现单点登录。

1.4.7 SessionDAO

SessionDAO 即会话 dao，是对 session 会话操作的一套接口，比如要将 session 存储到数据库，可以通过 jdbc 将会话存储到数据库。

1.4.8 CacheManager

CacheManager 即缓存管理，将用户权限数据存储在缓存，这样可以提高性能。

1.4.9 Cryptography

Cryptography 即密码管理，shiro 提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能。

尚学堂·百战程序员

2 认证

2.1 基本概念

2.1.1 身份验证

即在应用中谁能证明他就是他本人。一般提供如他们的身份 ID 一些标识信息来表明他就是他本人，如提供身份证，用户名/密码来证明。
在 shiro 中，用户需要提供 principals （身份）和 credentials（证明）给 shiro，从而应用能验证用户身份：

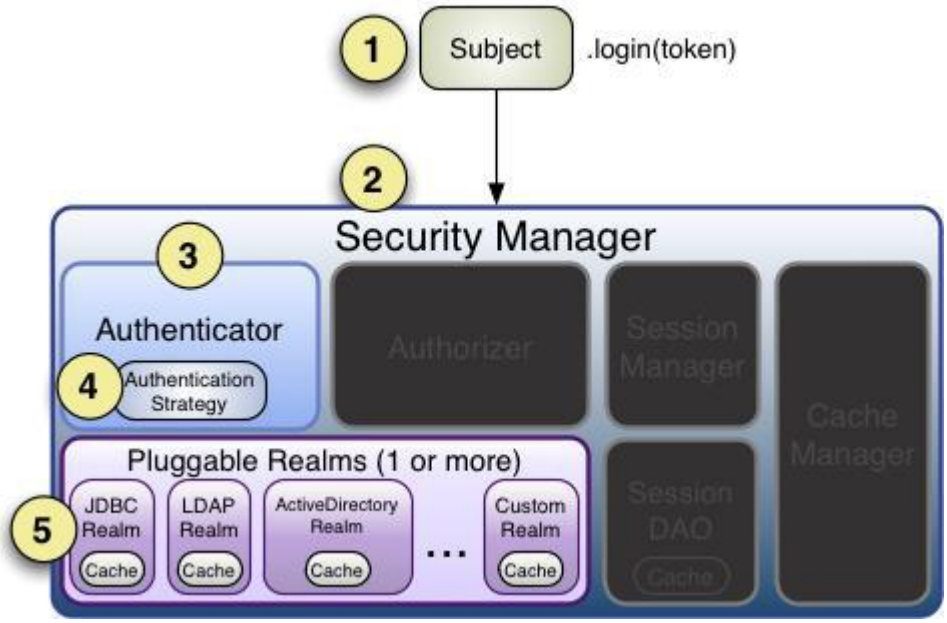
2.1.2 principals

身份，即主体的标识属性，可以是任何东西，如用户名、邮箱等，唯一即可。
一个主体可以有多个 principals，但只有一个 Primary principals，一般是用户名/密码/手机号。

2.1.3 credentials

证明/凭证，即只有主体知道的安全值，如密码/数字证书等。
最常见的 principals 和 credentials 组合就是用户名/密码了。接下来先进行一个基本的身份认证。

2.2 认证流程



尚学堂·百战程序员

2.3 入门程序（用户登陆和退出）

2.3.1 创建 java 工程

2.3.2 加入相关 jar 包

commons-beanutils-1.9.2.jar
commons-logging-1.2.jar
junit-4.10.jar
shiro-all-1.2.3.jar
slf4j-api-1.7.7.jar
log4j-1.2.17.jar
slf4j-log4j12-1.7.5.jar

2.3.3 log4j.properties 日志配置文件

```
log4j.rootLogger=debug, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m %n
```

2.3.4 配置 shiro 环境文件 shiro.ini

通过 Shiro.ini 配置文件初始化 SecurityManager 环境。

```
[users]
zhangsan=1111
lisi=1111
```

2.3.5 代码实现

```
//用户登录和退出
    @Test
    public void testAuthenticator(){
        // 构建 SecurityManager 工厂，IniSecurityManagerFactory 可以
从 ini 文件中初始化 SecurityManager 环境
        Factory<SecurityManager> factory = new
IniSecurityManagerFactory("classpath:shiro.ini");
        //通过工厂获得 SecurityManager 实例
```

尚学堂·百战程序员

```
        SecurityManager securityManager = factory.getInstance();
        //将 securityManager 设置到运行环境中
        SecurityUtils.setSecurityManager(securityManager);
        //获取 subject 实例
        Subject subject = SecurityUtils.getSubject();
        //创建用户名,密码身份验证 Token
        UsernamePasswordToken token = new
UsernamePasswordToken("zhangsan", "1111");
        try {
            //登录，即身份验证
            subject.login(token);
        } catch (AuthenticationException e) {
            e.printStackTrace();
            //身份认证失败
        }
        //断言用户已经登录
        Assert.assertEquals(true, subject.isAuthenticated());
        //退出
        subject.logout();
    }
```

2.4 自定义 Realm

Shiro 默认使用自带的 IniRealm，IniRealm 从 ini 配置文件中读取用户的信息，大部分情况下需要从系统的数据库中读取用户信息，所以需要自定义 realm。

2.4.1 Realm 接口

尚学堂·百战程序员

最基础的是 Realm 接口，CachingRealm 负责缓存处理，AuthenticationRealm 负责认证，AuthorizingRealm 负责授权，通常自定义的 realm 继承 AuthorizingRealm。

2.4.2 自定义 Realm 实现

```
/**
 * 自定义 Realm 实现
 * @author 邹波
 * @version 1.0
 * @date 2016-1-21
 */
public class UserRealm extends AuthorizingRealm {
    @Override
    public String getName() {
        return "UserRealm";
    }
    //用于认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken token) throws
AuthenticationException {
        //从 token 中获取身份信息
        String username = (String)token.getPrincipal();
        //根据用户名到数据库中取出用户信息  如果查询不到 返回 null
        String password = "1111";//假如从数据库中获取密码为 1111
        //返回认证信息
        SimpleAuthenticationInfo simpleAuthenticationInfo = new
SimpleAuthenticationInfo(username, password, this.getName());
        return simpleAuthenticationInfo;
    }
    //用于授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(
            PrincipalCollection principals) {
        return null;
    }
}
```

2.4.3 配置 Realm

需要在 shiro.ini 配置 realm 注入到 securityManager 中。

```
[main]
#自定义 realm
```

尚学堂·百战程序员

```
userRealm=cn.siggy.realm.UserRealm
#将 realm 设置到 securityManager
securityManager.realms=$userRealm
```

2.4.4 测试

同上一样

2.5 散列算法

散列算法一般用于生成数据的摘要信息，是一种不可逆的算法，一般适合存储密码之类的数据，常见的散列算法如 MD5、SHA 等。一般进行散列时最好提供一个 salt（盐），比如加密密码“admin”，产生的散列值是“21232f297a57a5a743894a0e4a801fc3”，可以到一些 md5 解密网站很容易的通过散列值得到密码“admin”，即如果直接对密码进行散列相对来说破解更容易，此时我们可以加一些只有系统知道的干扰数据，如用户名和 ID（即盐）；这样散列的对象是“密码+用户名+ID”，这样生成的散列值相对来说更难破解。

2.5.1 MD5 算法

```
/**
 *
 * @author 邹波
 * @version 1.0
 * @date 2016-1-21
 */
public class ShiroTest {
    //shiro 提供了现成的加密类  Md5Hash
    @Test
    public void testMd5(){
        //MD5 加密
        String password = new Md5Hash("1111").toString();
        System.out.println("加密后：“+password);
        //加盐  salt  默认一次散列
        String password_salt=new Md5Hash("1111",
"siggy").toString();
        System.out.println("加盐后：“+password_salt);
        //散列 2 次
        String password_salt_2 = new Md5Hash("1111", "siggy",
2).toString();
        System.out.println("散列 2 次：“+password_salt_2);
        //使用 SimpleHash
```

尚学堂·百战程序员

```
        SimpleHash hash = new SimpleHash("MD5", "1111", "siggy", 2);
        System.out.println("simpleHash:"+hash.toString());
    }
}
```

2.5.2 在自定义 Realm 中使用散列

Realm 实现代码

```
public class UserRealm extends AuthorizingRealm {
    @Override
    public String getName() {
        return "UserRealm";
    }
    //用于认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken token) throws
AuthenticationException {
        //从 token 中获取身份信息
        String username = (String)token.getPrincipal();
        //根据用户名到数据库中取出用户信息  如果查询不到 返回 null
        //按照固定规则加密码结果 ，此密码 要在数据库存储，原始密码 是
1111，盐是 siggy 2 次散列
        String password = "1620d20433da92e2523928e351e90f97";//假
如从数据库中获取密码为 1111
        //返回认证信息
        SimpleAuthenticationInfo simpleAuthenticationInfo = new
SimpleAuthenticationInfo(username,
                password,
ByteSource.Util.bytes("siggy"),this.getName());
        return simpleAuthenticationInfo;
    }
    //用于授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(
            PrincipalCollection principals) {
        return null;
    }
}
```

尚学堂·百战程序员

2.5.3 Realm 配置

Shiro.ini 在配置文件中，需指定凭证匹配器

```
[main]
#定义凭证匹配器
credentialsMatcher=org.apache.shiro.authc.credential.HashedCredentialsMatcher
#散列算法
credentialsMatcher.hashAlgorithmName=md5
#散列次数
credentialsMatcher.hashIterations=2

#将凭证匹配器设置到 realm
userRealm=cn.siggy.realm.UserRealm
userRealm.credentialsMatcher=$credentialsMatcher
securityManager.realms=$userRealm
```

2.5.4 测试

同上

3 授权

授权，也叫访问控制，即在应用中控制谁能访问哪些资源（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）。

3.1 关键对象介绍

主体

主体，即访问应用的用户，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

在应用中用户可以访问的任何东西，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

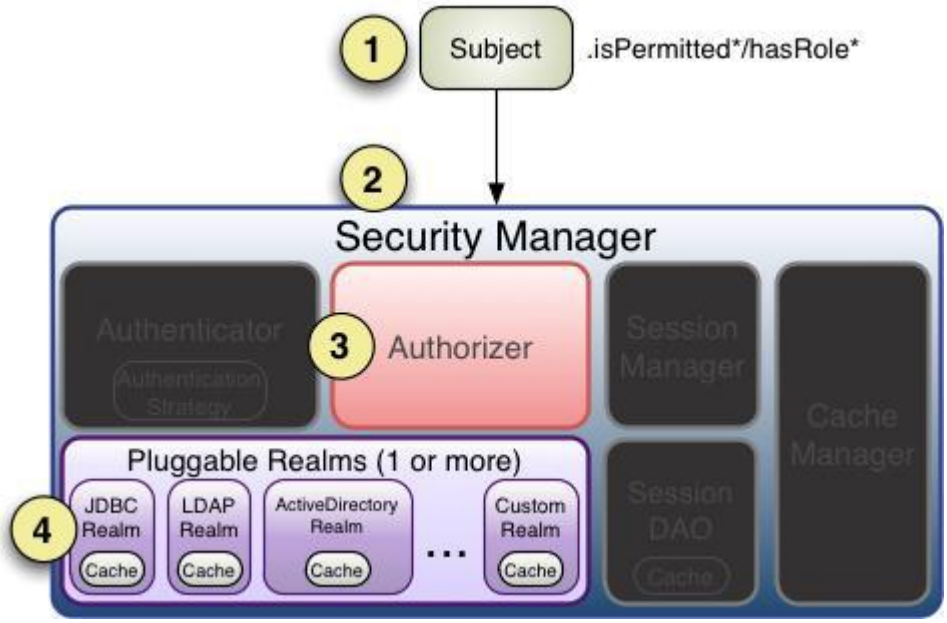
尚学堂·百战程序员

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）打印文档等等。。。

角色

角色代表了操作集合，可以理解为权限的集合，一般情况下我们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。

3.2 授权流程



流程如下：

1、首先调用 Subject.isPermitted*/hasRole*接口，其会委托给 SecurityManager，而 SecurityManager 接着会委托给 Authorizer；

2、Authorizer 是真正的授权者，如果我们调用如 isPermitted(“user:view”)，其首先会通过 PermissionResolver 把字符串转换成相应的 Permission 实例；

3、在进行授权之前，其会调用相应的 Realm 获取 Subject 相应的角色/权限用于匹配传入的角色/权限；

4、Authorizer 会判断 Realm 的角色/权限是否和传入的匹配，如果有多个 Realm，会委托给 ModularRealmAuthorizer 进行循环判断，如果匹配如 isPermitted*/hasRole*会返回 true，否则返回 false 表示授权失败。

3.3 授权方式

Shiro 支持三种方式的授权：

编程式：通过写 if/else 授权代码块完成：

```
Subject subject = SecurityUtils.getSubject();
```

尚学堂·百战程序员

```
if(subject.hasRole(“admin”)) {
//有权限
} else {
//无权限
}
```

注解式：通过在执行的 Java 方法上放置相应的注解完成：

```
@RequiresRoles("admin")
public void hello() {
//有权限
}
```

没有权限将抛出相应的异常；

JSP/GSP 标签：在 JSP/GSP 页面通过相应的标签完成：

```
<shiro:hasRole name="admin">
<!— 有权限—>
</shiro:hasRole>
```

3.4 授权实现

3.4.1 在 ini 配置文件配置用户拥有的角色及角色-权限关系（shiro-permission.ini）

```
[users]
zhangsan=1111,role1,role2
lisi=1111,role1
[roles]
role1=user:create,user:update
role2=user:create,user:delete
```

规则：“用户名=密码，角色 1，角色 2” “角色=权限 1，权限 2”，即首先根据用户名找到角色，然后根据角色再找到权限；即角色是权限集合；Shiro 同样不进行权限的维护，需要我们通过 Realm 返回相应的权限信息。只需要维护“用户——角色”之间的关系即可。

权限字符串的规则是：“资源标识符：操作：资源实例标识符”，意思是对哪个资源的哪个实例具有什么操作，“:”是资源/操作/实例的分割符，权限字符串也可以使用*通配符。

例子：

用户创建权限：user:create，或 user:create:*

用户修改实例 001 的权限：user:update:001

用户实例 001 的所有权限：user：*：001

尚学堂·百战程序员

3.4.2 实现代码

```
/**
 *
 * @author 邹波
 * @version 1.0
 * @date 2016-1-21
 */
public class ShiroTest {
    //用户登录和退出
    @Test
    public void testPermission(){
        // 构建 SecurityManager 工厂，IniSecurityManagerFactory 可以
从 ini 文件中初始化 SecurityManager 环境
        Factory<SecurityManager> factory = new
IniSecurityManagerFactory("classpath:shiro-permission.ini");
        //通过工厂获得 SecurityManager 实例
        SecurityManager securityManager = factory.getInstance();
        //将 securityManager 设置到运行环境中
        SecurityUtils.setSecurityManager(securityManager);
        //获取 subject 实例
        Subject subject = SecurityUtils.getSubject();
        //创建用户名,密码身份验证 Token
        UsernamePasswordToken token = new
UsernamePasswordToken("zhangsan", "1111");
        try {
            //登录，即身份验证
            subject.login(token);
        } catch (AuthenticationException e) {
            e.printStackTrace();
            //身份认证失败
        }
        // 用户认证状态
        boolean isAuthenticated = subject.isAuthenticated();
        System.out.println("用户认证状态： " + isAuthenticated);

        //判断拥有角色：role1
        Assert.assertTrue(subject.hasRole("role1"));
        //判断拥有角色：role1 and role2

    Assert.assertTrue(subject.hasAllRoles(Arrays.asList("role1"
, "role2")));
        //判断拥有角色：role1 and role2 and !role3
```

尚学堂·百战程序员

```
        boolean[] result = subject.hasRoles(Arrays.asList("role1",
"role2", "role3"));
        Assert.assertEquals(true, result[0]);
        Assert.assertEquals(true, result[1]);
        Assert.assertEquals(false, result[2]);


        //判断拥有权限：user:create
        Assert.assertTrue(subject.isPermitted("user:create"));
        //判断拥有权限：user:update and user:delete
        Assert.assertTrue(subject.isPermittedAll("user:update",
"user:delete"));
        //判断没有权限：user:view
        Assert.assertFalse(subject.isPermitted("user:view"));
    }
}
```

3.5 自定义 Realm 实现授权

与上边认证自定义 realm 一样，大部分情况是要从数据库获取权限数据，这里直接实现基于资源的授权。

3.5.1 UserRealm 实现代码

```
/**
 * 自定义 Realm 实现
 * @author 邹波
 * @version 1.0
 * @date 2016-1-21
 */
public class UserRealm extends AuthorizingRealm {
    @Override
    public String getName() {
        return "UserRealm";
    }
    //用于认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken token) throws
AuthenticationException {
        //从 token 中获取身份信息
```

尚学堂·百战程序员

```
        String username = (String)token.getPrincipal();
        //根据用户名到数据库中取出用户信息  如果查询不到 返回 null
        String password = "1111";//假如从数据库中获取密码为 1111
        //返回认证信息
        SimpleAuthenticationInfo simpleAuthenticationInfo = new
SimpleAuthenticationInfo(username, password, this.getName());
        return simpleAuthenticationInfo;
    }
    //用于授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(
            PrincipalCollection principals) {
        //获取身份信息
        String username =
(String)principals.getPrimaryPrincipal();
        //根据身份信息获取权限数据
        //模拟
        List<String> permissions = new ArrayList<String>();
        permissions.add("user:save");
        permissions.add("user:delete");
        //将权限信息保存到 AuthorizationInfo 中
        SimpleAuthorizationInfo simpleAuthorizationInfo = new
SimpleAuthorizationInfo();
        for(String permission:permissions){

    simpleAuthorizationInfo.addStringPermission(permission);
        }
        return simpleAuthorizationInfo;
    }
}
```

增加了红色部分代码

3.5.2 配置文件

```
[main]
#自定义 realm
userRealm=cn.siggy.realm.UserRealm
#将 realm 设置到 securityManager
securityManager.realms=$userRealm
```

3.5.3 测试代码

```
/**
```

尚学堂·百战程序员

```
 * 自定义 Realm 实现
 * @author 邹波
 * @version 1.0
 * @date 2016-1-21
 */
public class UserRealm extends AuthorizingRealm {
    @Override
    public String getName() {
        return "UserRealm";
    }
    //用于认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken token) throws
AuthenticationException {
        //从 token 中获取身份信息
        String username = (String)token.getPrincipal();
        //根据用户名到数据库中取出用户信息  如果查询不到 返回 null
        String password = "1111";//假如从数据库中获取密码为 1111
        //返回认证信息
        SimpleAuthenticationInfo simpleAuthenticationInfo = new
SimpleAuthenticationInfo(username, password, this.getName());
        return simpleAuthenticationInfo;
    }
    //用于授权
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(
            PrincipalCollection principals) {
        //获取身份信息
        String username =
(String)principals.getPrimaryPrincipal();
        //根据身份信息获取权限数据
        //模拟
        List<String> permissions = new ArrayList<String>();
        permissions.add("user:save");
        permissions.add("user:update");
        permissions.add("user:delete");
        //将权限信息保存到 AuthorizationInfo 中
        SimpleAuthorizationInfo simpleAuthorizationInfo = new
SimpleAuthorizationInfo();
        for(String permission:permissions){

    simpleAuthorizationInfo.addStringPermission(permission);
        }
```

尚学堂·百战程序员

```
        return simpleAuthorizationInfo;
    }
}
```

4 shiro 与项目集成开发

4.1 完成 springmvc+spring+mybatis 整合

4.2 整合 shiro

4.2.1 web.xml 中配置 shiro 的 filter

```
!-- shiro过虑器，DelegatingFilterProxy通过代理模式将spring容器中的bean和filter关联起来 -->
    <filter>
        <filter-name>shiroFilter</filter-name>

    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        <!-- 设置true由servlet容器控制filter的生命周期 -->
        <init-param>
            <param-name>targetFilterLifecycle</param-name>
            <param-value>true</param-value>
        </init-param>
        <!-- 设置spring容器filter的bean id，如果不设置则找与filter-name一致的bean-->
        <init-param>
            <param-name>targetBeanName</param-name>
            <param-value>shiroFilter</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>shiroFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

尚学堂·百战程序员

4.2.2 在 spring 中配置 shiro

```
<!-- web.xml 中 shiro 的 filter 对应的 bean -->
    <!-- Shiro 的 Web 过滤器 -->
    <bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <property name="securityManager" ref="securityManager" />
        <!-- loginUrl 认证提交地址，如果没有认证将会请求此地址进行认证，
请求此地址将由 formAuthenticationFilter 进行表单认证 -->
        <property name="loginUrl" value="/login.do" />
        <!-- 认证成功统一跳转到 index.do，建议不配置，shiro 认证成功自
动到上一个请求路径 -->
        <property name="successUrl" value="/index.do"/>
        <!-- 通过 unauthorizedUrl 指定没有权限操作时跳转页面-->
        <property name="unauthorizedUrl" value="/refuse.jsp" />
        <!-- 过虑器链定义，从上向下顺序执行，一般将/**放在最下边 -->
        <property name="filterChainDefinitions">
            <value>
                <!-- /** = authc 所有 url 都必须认证通过才可以访问-->
                /login.jsp=anon
                /** = authc
                <!-- /** = anon 所有 url 都可以匿名访问 -->

            </value>
        </property>
    </bean>

    <!-- securityManager 安全管理器 -->
    <bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
            <property name="realm" ref="userRealm" />
    </bean>
    <!-- realm -->
    <bean id="userRealm" class="cn.siggy.realm.UserRealm">
        <!-- 将凭证匹配器设置到 realm 中，realm 按照凭证匹配器的要求进
行散列 -->
        <property name="credentialsMatcher"
ref="credentialsMatcher"/>
    </bean>
    <!-- 凭证匹配器 -->
    <bean id="credentialsMatcher"

    class="org.apache.shiro.authc.credential.HashedCredentialsM
```

尚学堂·百战程序员

```
atcher">
        <property name="hashAlgorithmName" value="md5" />
        <property name="hashIterations" value="1" />
    </bean>
```

4.3 登录

4.3.1 原理

Shiro 内置了很多默认的过滤器，比如身份验证、授权等相关的。默认过滤器可以参考 org.apache.shiro.web.filter.mgt.DefaultFilter 中的过滤器：

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

anon:例子/admins/**=anon 没有参数，表示可以匿名使用。
authc:例如/admins/user/**=authc 表示需要认证(登录)才能使用，FormAuthenticationFilter 是表单认证，没有参数
使用 FormAuthenticationFilter 过虑器实现 ，原理如下：

将用户没有认证时，请求 loginurl 进行认证，用户身份和用户密码提交数据到 loginurl
FormAuthenticationFilter 拦截住取出 request 中的 username 和 password（两个参数名称是可以配置的）
FormAuthenticationFilter 调用 realm 传入一个 token（username 和 password）
realm 认证时根据 username 查询用户信息（在 Activeuser 中存储，包括 userid、usercode、username、menus）。
如果查询不到，realm 返回 null，FormAuthenticationFilter 向 request 域中填充一个参数（记录了异常信息）

4.3.2 登陆页面

由于 FormAuthenticationFilter 的用户身份和密码的 input 的默认值（username 和

尚学堂·百战程序员

password），修改页面的账号和密码 的 input 的名称为 username 和 password

4.3.3 代码实现

```
@Controller
public class LoginController {

    @RequestMapping("/login.do")
    public String login(HttpServletRequest req,Model model){
        String exceptionClassName =
(String)req.getAttribute("shiroLoginFailure");
        String error = null;

    if(UnknownAccountException.class.getName().equals(exception
ClassName)) {
        error = "用户名/密码错误";
        } else
if(IncorrectCredentialsException.class.getName().equals(except
ionClassName))
        {
        error = "用户名/密码错误";
        } else if(exceptionClassName != null) {
        error = "其他错误: " + exceptionClassName;
        }
        model.addAttribute("error", error);
        return "redirect:login.jsp";
    }
}
```

尚学堂·百战程序员