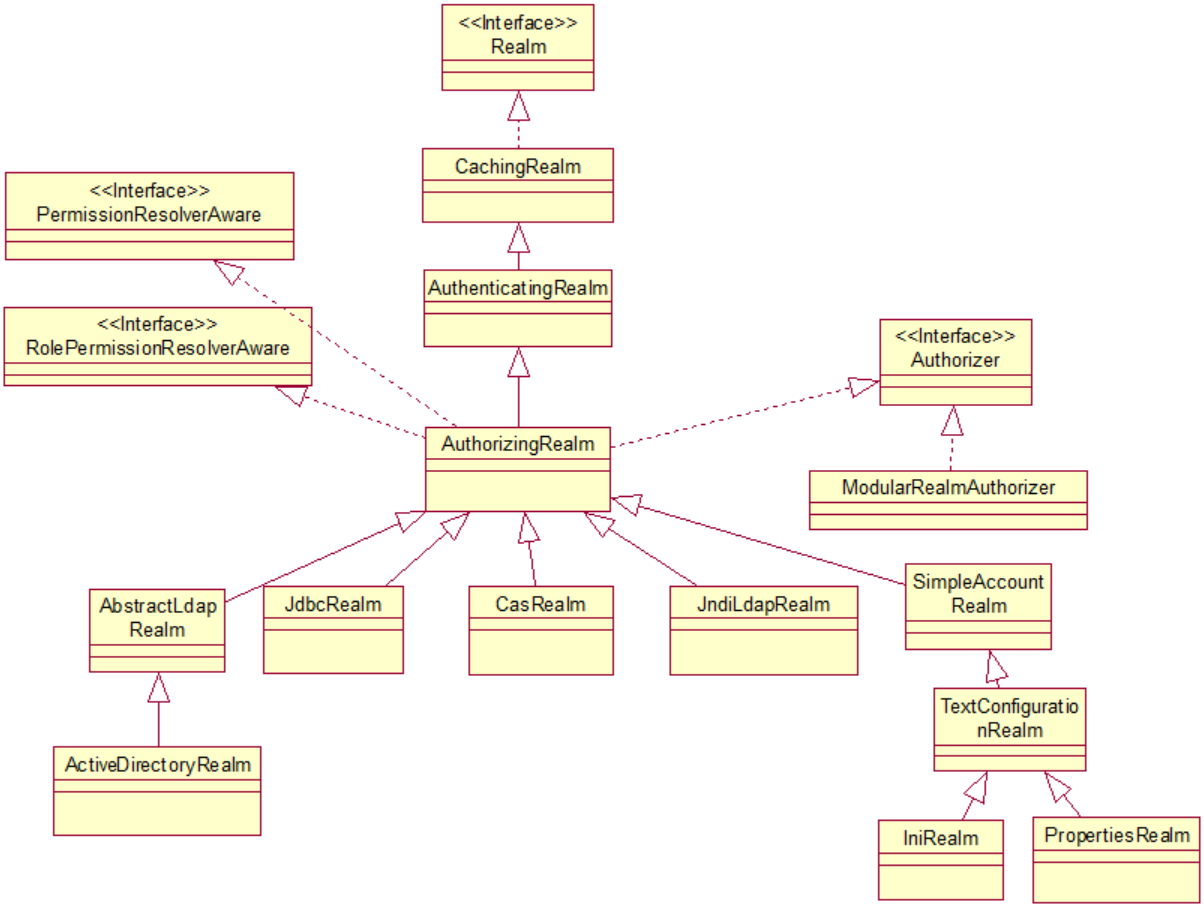
# shiro 源码分析（四）具体的 Realm

首先还是 Realm 的接口设计图：




这里只来说明 SimpleAccountRealm 和 JdbcRealm 的实现。

首先是 SimpleAccountRealm 不用关心数据的具体来源，只提供了与上层的交互，即实现了 AuthenticatingRealm 遗留的 AuthenticationInfo doGetAuthenticationInfo 和 AuthorizingRealm 遗留的 AuthorizationInfo doGetAuthorizationInfo。

如下：

Java 代码

```java
protected final Map<String, SimpleAccount> users; //username-to-SimpleAccount
    protected final Map<String, SimpleRole> roles; //roleName-to-SimpleRole
    protected final ReadWriteLock USERS_LOCK;
    protected final ReadWriteLock ROLES_LOCK;

    public SimpleAccountRealm() {
        this.users = new LinkedHashMap<String, SimpleAccount>();
        this.roles = new LinkedHashMap<String, SimpleRole>();
        USERS_LOCK = new ReentrantReadWriteLock();
        ROLES_LOCK = new ReentrantReadWriteLock();
        //SimpleAccountRealms are memory-only realms - no need for an additional cache mechanism since we're
        //already as memory-efficient as we can be:
        setCachingEnabled(false);
    }
```

SimpleAccountRealm 内部有四个属性，Map<String, SimpleAccount> users：用于存放用户账号信息，Map<String, SimpleRole> roles 用于存放角色名的信息。这两个都是各种配置的最终归属存储地。

ReadWriteLock USERS_LOCK：由于这些配置信息，一般不会去修改，大部分时间用于查询，所以要使用读写锁。一般的 synchronized 同步，不管你是读还是写，都要进行等待。写与写需要进行同步，写与读也要进行同步，但读与读却并不需要进行同步，所以对于那些经常读的场景，要使用读写锁 ReadWriteLock 来提升性能。

ReadWriteLock ROLES_LOCK 同理。

有了以上数据源，实现父类的遗留的方法就比较简单了。如下：

Java 代码

```java
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
        UsernamePasswordToken upToken = (UsernamePasswordToken) token;
        SimpleAccount account = getUser(upToken.getUsername());

        if (account != null) {

            if (account.isLocked()) {
                throw new LockedAccountException("Account [" + account + "] is locked.");
            }
            if (account.isCredentialsExpired()) {
                String msg = "The credentials for account [" + account + "] are expired";
                throw new ExpiredCredentialsException(msg);
            }

        }

        return account;
    }

    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
        String username = getUsername(principals);
        USERS_LOCK.readLock().lock();
        try {
            return this.users.get(username);
        } finally {
            USERS_LOCK.readLock().unlock();
        }
    }
```

代码就很简单了，就是从 users 中取出相应的用户数据。接下来要分析清几个概念：

AuthorizationInfo、AuthenticationInfo、SimpleAccount、SimpleRole、PrincipalCollection。

PrincipalCollection：看下文档介绍

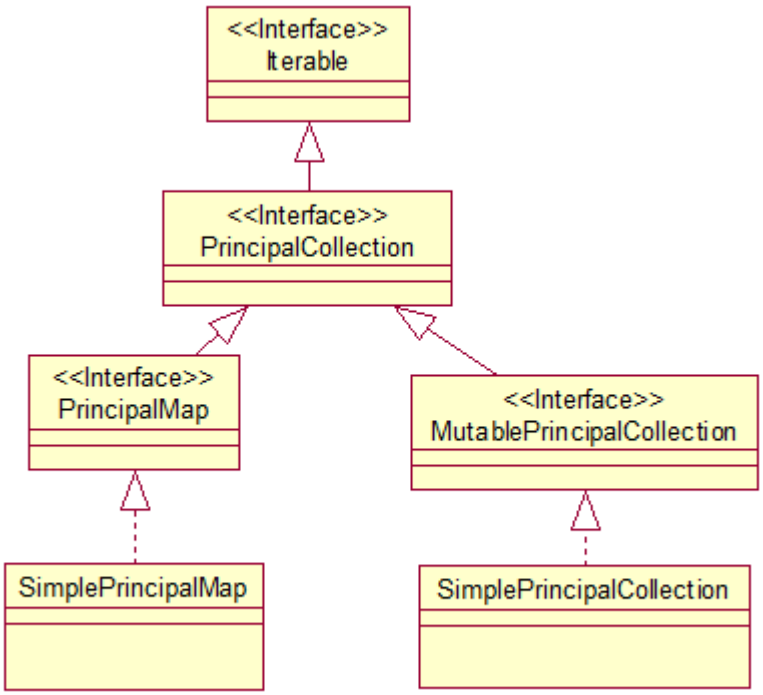
引用

```
/**
* A collection of all principals associated with a corresponding {@link Subject
Subject}.  A <em>principal</em> is
* just a security term for an identifying attribute, such as a username or user id or social
security number or
* anything else that can be considered an 'identifying' attribute for a {@code Subject}.
* <p/>
* A PrincipalCollection organizes its internal principals based on the {@code Realm}
where they came from when the
* Subject was first created.  To obtain the principal(s) for a specific Realm, see the
{@link #fromRealm} method.  You
* can also see which realms contributed to this collection via the {@link
#getRealmNames() getRealmNames()} method.
*/
```

一个 principal 仅仅是 Subject 的一个标识而已，如可以是用户名，用户 id 等。PrincipalCollection 则是这些属性的集合。每个用户属性可以来自不同的 Realm。Collection fromRealm(String realmName)可以获取某个 Realm 的所有用户属性。Set<String> getRealmNames()可以获取到与 Subject 关联的用户的属性来自于哪些 Realm。

Object getPrimaryPrincipal()：主要是用于获取唯一标示，如 UUID、username 等。

接口如下：




MutablePrincipalCollection 如下：

Java 代码

```
1. public interface MutablePrincipalCollection extends PrincipalCollection {
2.     void add(Object principal, String realmName);
3.     void addAll(Collection principals, String realmName);
4.     void addAll(PrincipalCollection principals);
5.     void clear();
6. }
```

我们知道每一个标示都有所属的 realm，所以再添加的时候，要带上 realmName。

SimplePrincipalCollection：

Java 代码

```
1. private Map<String, Set> realmPrincipals;
```

一个重要的数据集合，key 是 realm 的 name，value 是 principal 集合。

这个接口分支一直在强调，每个 principal 都是有所属的 realm 的。

PrincipalMap：我这一块没有搞明白，先放下。

AuthenticationInfo 它是含有用户和密码信息的地方：

Java 代码

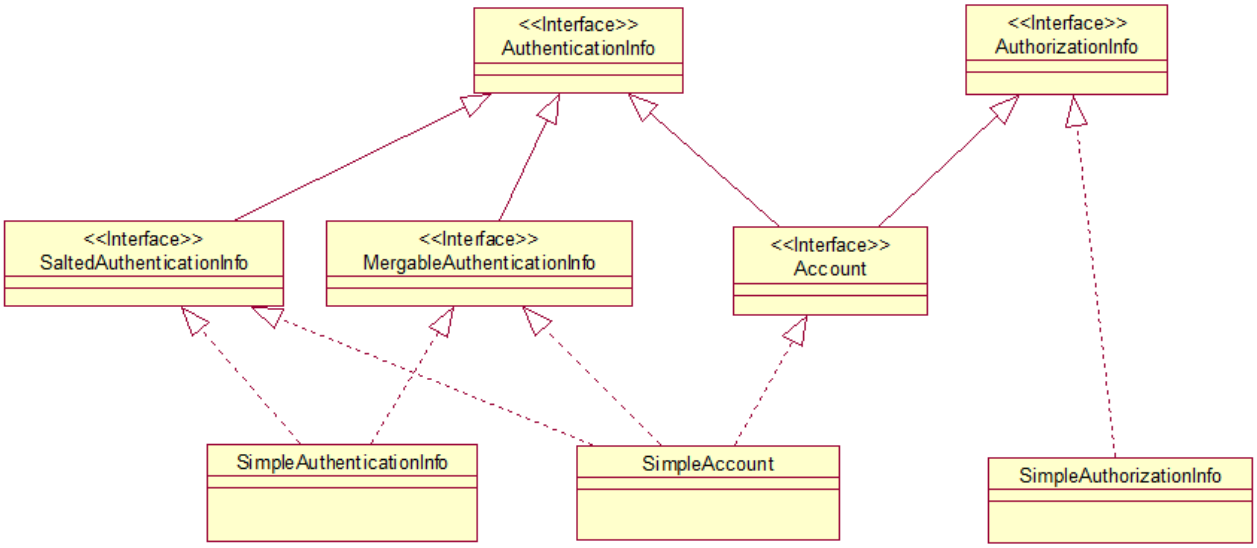
```java
public interface AuthenticationInfo extends Serializable {
    PrincipalCollection getPrincipals();
    Object getCredentials();
}
```

AuthorizationInfo ：存放用户权限的地方

Java 代码

```java
public interface AuthorizationInfo extends Serializable {
    Collection<String> getRoles();
    Collection<String> getStringPermissions();
    Collection<Permission> getObjectPermissions();
}
```

类图如下：




MergableAuthenticationInfo 意味着 AuthenticationInfo 可以进行合并：

Java 代码

```java
public interface MergableAuthenticationInfo extends AuthenticationInfo {
    void merge(AuthenticationInfo info);
}
```

SaltedAuthenticationInfo 主要用于密码匹配，后续文章专门说明：

Java 代码

```java
public interface SaltedAuthenticationInfo extends AuthenticationInfo {
    ByteSource getCredentialsSalt();
}
```

SimpleAuthenticationInfo：存储了三个重要的属性：

Java 代码

```java
protected PrincipalCollection principals;
protected Object credentials;
protected ByteSource credentialsSalt;
```

然后就是实现了 MergableAuthenticationInfo 接口，可以进行合并，

这里的合并在第一篇文章中 Realm 认证中提到过：

Java 代码

```java
public void merge(AuthenticationInfo info) {
        if (info == null || info.getPrincipals() == null || info.getPrincipals().isEmpty()) {
            return;
        }

        if (this.principals == null) {
            this.principals = info.getPrincipals();
        } else {
            if (!(this.principals instanceof MutablePrincipalCollection)) {
                this.principals = new SimplePrincipalCollection(this.principals);
            }
            ((MutablePrincipalCollection) this.principals).addAll(info.getPrincipals());
        }

```

```
        //only mess with a salt value if we don't have one yet.  It doesn't make sense
        //to merge salt values from different realms because a salt is used only within
        //the realm's credential matching process.  But if the current instance's salt
        //is null, then it can't hurt to pull in a non-null value if one exists.
        //
        //since 1.1:
        if (this.credentialsSalt == null && info instanceof SaltedAuthenticationInfo) {
            this.credentialsSalt = ((SaltedAuthenticationInfo) info).getCredentialsSalt();
        }

        Object thisCredentials = getCredentials();
        Object otherCredentials = info.getCredentials();

        if (otherCredentials == null) {
            return;
        }

        if (thisCredentials == null) {
            this.credentials = otherCredentials;
            return;
        }

        if (!(thisCredentials instanceof Collection)) {
            Set newSet = new HashSet();
            newSet.add(thisCredentials);
            setCredentials(newSet);
        }

        // At this point, the credentials should be a collection
        Collection credentialCollection = (Collection) getCredentials();
        if (otherCredentials instanceof Collection) {
            credentialCollection.addAll((Collection) otherCredentials);
        } else {
            credentialCollection.add(otherCredentials);
        }
    }
```

主要分 principals、credentialsSalt 和 credentials 三项的合并，代码也和简单。

SimpleAuthorizationInfo：存放了认证用户的角色和用户权限。

Java 代码

```java
protected Set<String> roles;
protected Set<String> stringPermissions;
protected Set<Permission> objectPermissions;
```

最重要的就是 SimpleAccount：

Java 代码

```java
public class SimpleAccount implements Account, MergableAuthenticationInfo, SaltedAuthenticationInfo, Serializable {
    private SimpleAuthenticationInfo authcInfo;
    private SimpleAuthorizationInfo authzInfo;
    private boolean locked;
    private boolean credentialsExpired;
}
```

它有 SimpleAuthenticationInfo 、SimpleAuthorizationInfo ，所以是用户认证信息和权限信息的汇总。

还有两个属性 locked 和 credentialsExpired，用来表示用户的锁定和密码过期的状态。

至此整个 SimpleAccount 便介绍完了。

回到 SimpleAccountRealm，SimpleAccountRealm 已经拥有 Map<String, SimpleAccount> users 和 Map<String, SimpleRole> roles 数据了，但是这些数据是怎么产生的呢？这就需要交给它的子类 TextConfigurationRealm 来完成：

Java 代码

```java
private volatile String userDefinitions;
private volatile String roleDefinitions;
```

仅仅两个字符串包含了所有的用户和角色的配置总来源。所以 TextConfigurationRealm 主要就是对这两个字符串的解析：

Java 代码

```java
@Override
protected void onInit() {
        super.onInit();
        processDefinitions();
    }
protected void processDefinitions() {
        try {
          //解析角色配置
            processRoleDefinitions();
         //解析用户配置
            processUserDefinitions();
        } catch (ParseException e) {
            String msg = "Unable to parse user and/or role definitions.";
            throw new ConfigurationException(msg, e);
        }
    }
protected void processRoleDefinitions() throws ParseException {
        String roleDefinitions = getRoleDefinitions();
        if (roleDefinitions == null) {
            return;
        }
        //先将角色字符串按行分割，然后每行再按照 key value 分割
        Map<String, String> roleDefs = toMap(toLines(roleDefinitions));
        processRoleDefinitions(roleDefs);
    }
protected void processRoleDefinitions(Map<String, String> roleDefs) {
        if (roleDefs == null || roleDefs.isEmpty()) {
            return;
        }
        for (String rolename : roleDefs.keySet()) {
            String value = roleDefs.get(rolename);

            SimpleRole role = getRole(rolename);
            if (role == null) {
```

```
                role = new SimpleRole(rolename);
                add(role);
            }

            Set<Permission> permissions = PermissionUtils.resolveDelimitedPermissions(value, getPermissionResolver());
            role.setPermissions(permissions);
        }
    }
```

再通过 PermissionResolver 将字符串形式的权限转化成 Permission 对象，知道大致情况了，就可以了，不需要每一步都弄清楚。TextConfigurationRealm 主要用于解析两个配置字符串，这两个配置字符串的产生则继续交给子类来完成。IniRealm 则是通过 ini 配置文件来产生这两个字符串，PropertiesRealm 则是通过 properties 文件来产生这两个字符串。

至此，SimpleAccountRealm 这一路就大致走通了，接下来就是另一条路 JdbcRealm 了。

Java 代码

```
public class JdbcRealm extends AuthorizingRealm {
    protected static final String DEFAULT_AUTHENTICATION_QUERY = "select password from users where username = ?";
    protected static final String DEFAULT_SALTED_AUTHENTICATION_QUERY = "select password, password_salt from users where username = ?";
    protected static final String DEFAULT_USER_ROLES_QUERY = "select role_name from user_roles where username = ?";
    protected static final String DEFAULT_PERMISSIONS_QUERY = "select permission from roles_permissions where role_name = ?";
    protected DataSource dataSource;
}
```

首先是含有这几个默认的 sql 和 DataSource dataSource，用于从数据库中获取相应的用户、角色、权限等数据。

根据上一篇文章我们知道 JdbcRealm 要实现 AuthenticatingRealm 遗留的 AuthenticationInfo doGetAuthenticationInfo 和 AuthorizingRealm 遗留的 AuthorizationInfo doGetAuthorizationInfo。下面就看看是怎么来实现的：

Java 代码

```java
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

        UsernamePasswordToken upToken = (UsernamePasswordToken) token;
        String username = upToken.getUsername();

        // Null username is invalid
        if (username == null) {
            throw new AccountException("Null usernames are not allowed by this realm.");
        }

        Connection conn = null;
        SimpleAuthenticationInfo info = null;
        try {
            conn = dataSource.getConnection();

            String password = null;
            String salt = null;
            switch (saltStyle) {
            case NO_SALT:
                //根据用户名去查找密码
                password = getPasswordForUser(conn, username)[0];
                break;
            case CRYPT:
                // TODO: separate password and hash from getPasswordForUser[0]
                throw new ConfigurationException("Not implemented yet");
                //break;
            case COLUMN:
                String[] queryResults = getPasswordForUser(conn, username);
                password = queryResults[0];
                salt = queryResults[1];
                break;
```

```
            case EXTERNAL:
                password = getPasswordForUser(conn, username)[0];
                 //此时 salt 不存在数据库中，默认的值为 username
                salt = getSaltForUser(username);
            }

            if (password == null) {
                throw new UnknownAccountException("No account found for user ["  + username + "]");
            }
            //根据用户名、密码、盐值构建一个 SimpleAuthenticationInfo
            info = new SimpleAuthenticationInfo(username, password.toCharArray(), getName());

            if (salt != null) {
                info.setCredentialsSalt(ByteSource.Util.bytes(salt));
            }

        } catch (SQLException e) {
            final String message = "There was a SQL error while authenticating user [" + username + "]";
            if (log.isErrorEnabled()) {
                log.error(message, e);
            }

            // Rethrow any SQL errors as an authentication exception
            throw new AuthenticationException(message, e);
        } finally {
            JdbcUtils.closeConnection(conn);
        }

        return info;
    }
private String[] getPasswordForUser(Connection conn, String username) throws SQLException {

        String[] result;
        boolean returningSeparatedSalt = false;
        switch (saltStyle) {
        case NO_SALT:
        case CRYPT:
        case EXTERNAL:
            result = new String[1];
            break;
```

```
72.         default:
73.             result = new String[2];
74.             returningSeparatedSalt = true;
75.         }
76.
77.         PreparedStatement ps = null;
78.         ResultSet rs = null;
79.         try {
80.             ps = conn.prepareStatement(authenticationQuery);
81.             ps.setString(1, username);
82.
83.             // Execute query
84.             rs = ps.executeQuery();
85.
86.             // Loop over results - although we are only expecting one resul
    t, since usernames should be unique
87.             boolean foundResult = false;
88.             while (rs.next()) {
89.
90.                 // Check to ensure only one row is processed
91.                 if (foundResult) {
92.                     throw new AuthenticationException("More than one user ro
    w found for user [" + username + "]. Usernames must be unique.");
93.                 }
94.
95.                 result[0] = rs.getString(1);
96.                 if (returningSeparatedSalt) {
97.                     result[1] = rs.getString(2);
98.                 }
99.
100.                 foundResult = true;
101.             }
102.         } finally {
103.             JdbcUtils.closeResultSet(rs);
104.             JdbcUtils.closeStatement(ps);
105.         }
106.
107.         return result;
108.     }
109. protected String getSaltForUser(String username) {
110.         return username;
111.     }
```

代码很简单就不再一一细说。再看下 doGetAuthorizationInfo 是怎么实现的：

Java 代码

```java
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

        //null usernames are invalid
        if (principals == null) {
            throw new AuthorizationException("PrincipalCollection method argument cannot be null.");
        }

        String username = (String) getAvailablePrincipal(principals);

        Connection conn = null;
        Set<String> roleNames = null;
        Set<String> permissions = null;
        try {
            conn = dataSource.getConnection();

            // Retrieve roles and permissions from database
            roleNames = getRoleNamesForUser(conn, username);
            if (permissionsLookupEnabled) {
                permissions = getPermissions(conn, username, roleNames);
            }

        } catch (SQLException e) {
            final String message = "There was a SQL error while authorizing user [" + username + "]";
            if (log.isErrorEnabled()) {
                log.error(message, e);
            }

            // Rethrow any SQL errors as an authorization exception
            throw new AuthorizationException(message, e);
        } finally {
            JdbcUtils.closeConnection(conn);
        }

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

```
        info.setStringPermissions(permissions);
        return info;

    }
```

第一步先根据 PrincipalCollection 来获取用户名，第二步根据用户名来获取角色，第三部根据角色和用户名来获取权限。后两步都是执行简单的 sql，不再说，看下如何由 PrincipalCollection 获取用户名，该方法定义在 CachingRealm 中：

Java 代码

```
protected Object getAvailablePrincipal(PrincipalCollection principals) {
        Object primary = null;
        if (!CollectionUtils.isEmpty(principals)) {
            Collection thisPrincipals = principals.fromRealm(getName());
            if (!CollectionUtils.isEmpty(thisPrincipals)) {
                primary = thisPrincipals.iterator().next();
            } else {
                //no principals attributed to this particular realm.  Fall back to the 'master' primary:
                primary = principals.getPrimaryPrincipal();
            }
        }

        return primary;
    }
```

两种情况，首先是获取当前 Realm 的 Principals，如果有取其第一个。如果没有，则调用 getPrimaryPrincipal()方法。然后看下 JdbcRealm 的一个简单使用：

如果默认按照 JdbcRealm 的 sql 来作为数据库的查询来说，建表如下：

users 表：

Sql 代码

```sql
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `password_salt` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username_UNIQUE` (`username`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
```

## user_roles 表：

Sql 代码

```sql
CREATE TABLE `user_roles` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(45) DEFAULT NULL,
  `role_name` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;
```

## roles_permissions 表：

Sql 代码

```sql
CREATE TABLE `roles_permissions` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `role_name` varchar(45) DEFAULT NULL,
  `permission` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

## 文章最后会给出数据库 sql 文件。

## 然后就是配置 ini 文件：

Java 代码

```
[main]
#realm
dataSource=com.mchange.v2.c3p0.ComboPooledDataSource
dataSource.driverClass=com.mysql.jdbc.Driver
dataSource.jdbcUrl=jdbc:mysql://localhost:3306/shiro
dataSource.user=root
```

```
dataSource.password=XXXXXX
jdbcRealm=org.apache.shiro.realm.jdbc.JdbcRealm
jdbcRealm.dataSource=$dataSource
jdbcRealm.permissionsLookupEnabled=true
securityManager.realms=$jdbcRealm
```

使用的 dataSource 是 c3p0 的 dataSource，mysql 驱动也是必然不能少的，所以 maven 中要加入依赖：

**Xml 代码**

```xml
      <!-- mysql 驱动 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.29</version>
</dependency>

<!-- 连接池 -->
<dependency>
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
</dependency>
```

为了输出方便代码更改为：

**Java 代码**

```java
public class ShiroTest {

    @Test
    public void testHelloworld() {
        init();

        Subject subject=login("lg","123");
        System.out.println(subject.hasRole("role1"));
        System.out.println(subject.hasRole("role2"));
        System.out.println(subject.hasRole("role3"));
    }

    private Subject login(String userName,String password){
         //3、得到 Subject 及创建用户名/密码身份验证 Token（即用户身份/凭证）
        Subject subject = SecurityUtils.getSubject();
```

```
        UsernamePasswordToken token = new UsernamePasswordToken(userName,pas
sword);
        subject.login(token);
        return subject;
    }

    private void init(){
         //1、获取 SecurityManager 工厂，此处使用 Ini 配置文件初始化 SecurityManag
er
        Factory<org.apache.shiro.mgt.SecurityManager> factory =
                new IniSecurityManagerFactory("classpath:shiro.ini");
        //2、得到 SecurityManager 实例 并绑定给 SecurityUtils
        org.apache.shiro.mgt.SecurityManager securityManager = factory.getIn
stance();
        SecurityUtils.setSecurityManager(securityManager);
    }
}
```

对于 lg 用户，在数据库中它是有两个角色的,role1 和 role2。所以结果为 true、true、false。

Java 代码

```
true
true
false
```

OK，通过。最后附上 JdbcRealm 的使用例子。

作者：乒乓狂魔