# 好的文章

https://blog.csdn.net/u014745069/article/details/100812556

# 核心概念

## Subject（主体）

Subject代表当前与应用程序交互的用户，可以是一个人、一个设备或者其他系统。Subject可以进行身份验证和授权操作。在Shiro中，Subject是通过Subject实例来表示的。

## SecurityManager（安全管理器）

SecurityManager是Shiro的核心，负责处理所有的安全操作。它是Subject的入口点，协调身份验证、授权和会话管理等功能。SecurityManager还管理着所有的Realm，用于验证Subject的身份信息和获取授权信息。

## Realm（领域）

Realm是Shiro与应用程序交互的桥梁，用于验证Subject的身份信息和获取授权信息。它从应用程序中获取相关的数据，例如用户信息、角色权限等。Shiro可以配置多个Realm，用于支持多种不同的身份验证和授权方式。

从某种意义上讲，Realm 实际上就是一个安全相关的 DAO：它封装了数据源的连接细节，并在需要时将相关数据提供给 Shiro。

## Authentication（身份认证）

身份认证是验证Subject的身份凭证是否有效的**过程**。通过Subject提供的用户名和密码等凭证，Shiro可以进行身份认证操作，以确定用户是否是合法的。

## Authorization（授权）

授权是确定Subject是否具有执行某个操作的权限的**过程**。通过Subject的角色和权限等信息，Shiro可以对用户进行授权操作，决定是否允许用户进行特定的操作。

## Session（会话）

Session在Shiro中是可选的，用于跟踪用户在单个会话期间的状态和数据。Shiro提供了会话管理的功能，可以将会话信息存储在内存、数据库或其他持久化存储中。

## Cryptography（密码学）

Shiro提供了密码加密和解密的支持。通过使用Shiro提供的加密算法和工具类，可以对用户的密码等敏感信息进行安全的存储和传输。

# 优点

## 1、简单易用

快速上手：Shiro 的 API 设计非常简洁，开发者可以快速学习和使用其核心功能。

低侵入性：Shiro 的集成非常灵活，不强制依赖特定的框架或技术栈，适用于任意 Java 项目。

## 2、功能全面

（1）身份认证（Authentication）：

提供登录验证功能，支持多种认证方式（用户名/密码、OAuth 等）。

（2）授权（Authorization）：

支持基于角色和权限的细粒度访问控制。

（3）会话管理（Session Management）：

提供与 Web 容器无关的会话管理，可以在任何 Java 应用中使用。

（4）加密功能（Cryptography）：

内置加密工具类，支持常见的加密算法（如 MD5、SHA）。

## 3、灵活性强

（1）支持多种数据源：

支持从数据库、LDAP、内存、文件等多种数据源加载用户数据。

（2）模块化设计：

各个模块相互独立，可以按需集成。例如，可以只使用 Shiro 的会话管理模块，而不使用认证和授权功能。

（3）自定义能力：

提供丰富的扩展接口，如自定义 Realm 进行身份验证，自定义权限验证逻辑。

## 4、独立于容器

Shiro 的会话管理独立于 Web 容器，可以在非 Web 应用中使用（如桌面应用或微服务）。

## 5、支持分布式环境

提供对分布式会话的支持，可以通过 Redis 或其他存储机制来实现会话共享。

## 6、与其他框架兼容性好

Shiro 可以与 Spring、Spring Boot 等主流框架无缝集成。

# 缺点

## 1、社区支持较弱

（1）与更流行的安全框架（如 Spring Security）相比，Shiro 的社区规模较小，更新和维护频率较低。

（2）文档和教程相对较少，遇到复杂问题时可能需要自行摸索解决。

## 2、功能相对有限

（1）授权功能简单：

Shiro 的授权机制仅支持 RBAC（基于角色的访问控制）模型，无法实现复杂的 ACL（访问控制列表）或 ABAC（基于属性的访问控制）模型。

（2）不支持 OAuth2/OpenID Connect：

虽然可以通过扩展支持，但默认情况下 Shiro 并未内置 OAuth2 或 OpenID Connect 协议的支持。

## 3、不适合复杂的大型项目

Shiro 更适合中小型项目，特别是需要快速实现基本安全功能的场景。对于复杂的权限管理需求或分布式系统的统一认证管理，Shiro 的功能可能不足。

## 4、默认配置安全性一般

默认的加密方式和密码存储机制（如 MD5、SHA1）可能已经过时，开发者需要自行配置更安全的算法。

## 5、缺乏开箱即用的功能

相比于 Spring Security，Shiro 缺乏一些现代化的安全功能（如内置的 JWT 支持、单点登录、细粒度的权限控制等），需要开发者手动集成。

# 适用场景

## 1、中小型项目

快速集成身份认证和授权功能。

## 2、独立模块的会话管理

适用于非 Web 应用，如桌面应用或微服务。

## 3、自定义需求强

需要灵活调整安全机制的项目。

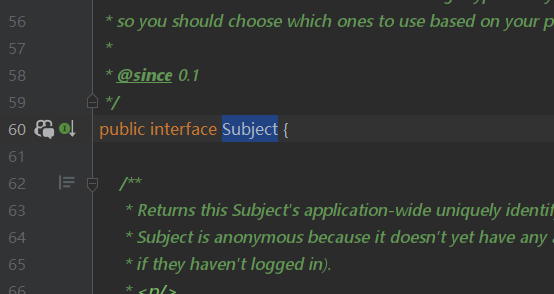
# 接口

## Subject

### 基本概念

Subject 接口是 Shiro 核心组件之一，代表了当前与应用交互的 **“用户”** 或 **“主体”**（不仅仅是人，也可以是设备、系统进程等）。Subject 接口提供了关于身份验证、授权、会话管理和运行上下文的操作。

### 源码



### 作用

#### 1、身份认证（Authentication）

（1）管理用户的登录和注销。

（2）判断用户是否已经通过身份认证。

（3）提供认证相关的信息，如获取当前用户的主体标识（通常是用户名）。

#### 2、授权（Authorization）

（1）检查用户是否具有某些权限或角色。

（2）通过角色和权限对用户进行访问控制。

#### 3、会话管理（Session Management）

（1）提供会话的相关操作，管理用户的会话数据。

（2）支持获取、设置、删除会话属性。

#### 4、访问控制上下文（Run-As）

（1）支持切换主体的身份（Run-As 功能）。

（2）允许一个主体以其他身份执行操作。

#### 5、应用与用户的交互

（1）在任何时候都可以获取当前与系统交互的主体（用户）。

（2）提供了用户与系统交互的上下文信息。

### 方法

#### void login(AuthenticationToken token)

登录

1、使用传入的 AuthenticationToken 进行登录验证（例如用户名和密码）。

2、如果验证失败，会抛出异常，如 UnknownAccountException 或 IncorrectCredentialsException。



#### void logout()

登出

1、注销当前主体，清除其身份信息。

2、常用于实现退出功能。



#### boolean isAuthenticated();

判断当前用户是否已通过身份认证。



#### boolean hasRole(String role);

判断当前主体是否具有指定角色。



#### boolean isPermitted(String permission);

判断当前主体是否具有指定权限。



#### Session getSession();

获取与当前主体关联的会话对象，用于存储和管理用户的会话信息。



#### void runAs(PrincipalCollection principals)

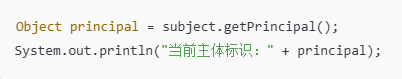
1、支持一个主体以其他身份执行操作，常用于代理或委托场景。

2、可以切换到其他身份后执行操作，然后恢复到原身份。



#### Object getPrincipal();

获取当前主体的主要标识信息（通常是用户名）。



### 使用场景

#### 1、用户登录和登出

验证用户的身份，完成登录操作。

在用户登出时清除身份信息。

#### 2、权限控制

判断用户是否具有访问某资源的权限。

通过角色或权限控制访问逻辑。

#### 3、会话管理

管理用户的会话数据，例如保存登录时间、偏好设置等。

#### 4、上下文切换

实现主体身份的动态切换，适用于委托操作或代理服务场景

### 优点

#### 1、简洁统一

将用户的认证、授权、会话管理、上下文切换等操作统一封装在 Subject 接口中，简化了开发过程。

#### 2、灵活性强

Subject 不仅可以代表人类用户，还可以代表系统进程、设备等，使其适用于多种场景。

#### 3、安全性高

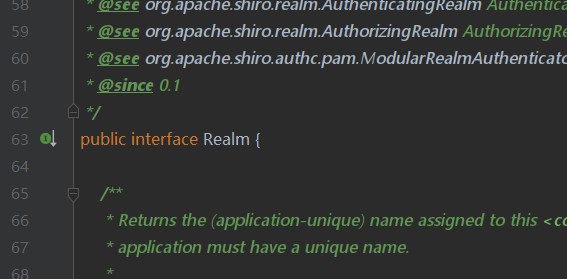
Shiro 的设计强调安全性，通过 Subject 实现了对用户操作的全面控制。

## Realm

### 基本概念

在 Apache Shiro 框架中，Realm 接口是核心组件之一，其作用是作为 Shiro 与应用安全数据之间的桥梁，负责从数据源中获取认证和授权信息。Realm 是一种安全数据的来源，用于验证用户身份和判断用户是否具有特定权限。

### 源码



### 作用

#### 1、身份认证（Authentication）

（1）校验用户的身份是否合法。

（2）获取用户的认证信息（如用户名、密码等）并与数据库或其他存储中的数据进行比对。

#### 2、权限授权（Authorization）

根据用户角色或权限，判断用户是否有权限访问特定资源。

获取用户的角色信息和权限信息。

#### 3、对接数据源

Realm 将 Shiro 与实际的数据源（如数据库、LDAP、文件等）连接起来。

开发者可以通过自定义 Realm 来支持各种数据源。

### 核心功能

#### 身份认证

身份认证是指验证用户的身份信息（例如用户名和密码）。在 Shiro 中，认证通过以下机制实现：

1、用户登录时，Shiro 将用户提交的认证信息（AuthenticationToken）传递给 Realm。

2、Realm 将从数据源中查询到的正确信息与用户提交的信息进行比对。

##### 主要方法

AuthenticationInfo getAuthenticationInfo(AuthenticationToken token) throws AuthenticationException;

参数：

AuthenticationToken：用户提交的认证信息（如用户名、密码等）。

返回值：

AuthenticationInfo：包含从数据源中查询到的认证信息。

抛出异常：

如果认证失败，会抛出 AuthenticationException 或其子类。

##### 示例



#### 授权

授权是指验证用户是否有权限访问特定资源。在 Shiro 中，授权通过以下机制实现：

1、在用户尝试访问受保护资源时，Shiro 会调用 Realm 获取用户的角色和权限信息。

2、根据这些信息判断用户是否有权限访问资源。

##### 主要方法

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals);

参数：

PrincipalCollection：包含已认证的用户信息（如用户名）。

返回值：

AuthorizationInfo：包含用户的角色和权限信息。

##### 示例



### 具体实现

#### 1、JdbcRealm

（1）从数据库中获取用户的认证和授权信息。

（2）适合以关系型数据库作为数据源的项目。

#### 2、IniRealm

从 .ini 配置文件中获取用户的认证和授权信息。

适合简单的配置或开发环境测试。

#### 3、LdapRealm

从 LDAP 服务器中获取用户的认证和授权信息。

适合基于 LDAP 协议的系统。

#### 4、自定义 Realm

开发者可以根据业务需求实现自己的 Realm，从任意数据源获取认证和授权信息（如 Redis、MongoDB 或外部 API）。

### 自定义 Realm 的步骤

#### 1、继承 Shiro 的 AuthorizingRealm 类

需要实现 doGetAuthenticationInfo 和 doGetAuthorizationInfo 两个方法。

#### 2、实现认证逻辑

在 doGetAuthenticationInfo 方法中，实现对用户身份的验证逻辑。

#### 3、实现授权逻辑

在 doGetAuthorizationInfo 方法中，实现对用户权限的分配逻辑。

#### 4、注册 Realm 到 Shiro 环境

在 Shiro 配置文件或代码中注册自定义的 Realm。

#### 示例代码



注册自定义 Realm



### 优缺点

#### 优点

##### 1、高度灵活

支持多种数据源（如数据库、LDAP、文件等）。

可通过自定义 Realm 满足复杂的业务需求。

##### 2、易于扩展

开发者可以通过自定义实现 Realm，以对接任何类型的数据源。

##### 3、统一接口

认证和授权逻辑都通过 Realm 实现，便于管理和维护。

#### 缺点

##### 1、需要开发者手动实现逻辑

对于复杂业务场景，自定义 Realm 需要更多的开发工作量。

##### 2、性能依赖于数据源

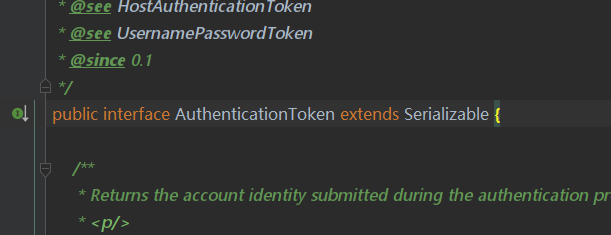
如果数据源的查询性能较差，可能会影响认证和授权的效率。

## AuthenticationToken

### 基本概念

在 Apache Shiro 框架中，AuthenticationToken 是一个接口，用于封装用户的登录信息（即认证信息）。它是 Shiro 身份认证流程的核心部分，代表用户在尝试登录时提交的凭证数据，例如用户名和密码、令牌等。

### 源码



### 作用

#### 1、封装认证信息

AuthenticationToken 用于保存用户登录时提交的认证信息，包括身份标识（例如用户名）和凭证（例如密码）。

#### 2、传递给 Realm

当用户提交登录请求时，Shiro 会将 AuthenticationToken 传递给 Realm。

Realm 使用 AuthenticationToken 提供的信息进行身份验证。

#### 3、抽象化认证数据

AuthenticationToken 作为接口，抽象了不同认证方式（用户名密码认证、OAuth、JWT 等）的数据表示。

开发者可以通过实现 AuthenticationToken 接口，定义自定义的认证数据结构。

### 方法

#### Object getPrincipal();

返回用户的身份标识信息，例如用户名、邮箱、手机号等。这是用来标识用户的唯一标识。

#### Object getCredentials();

返回用户的凭证信息，例如密码、令牌、指纹等。这是用来验证用户身份的敏感信息。

### 实现类

#### UsernamePasswordToken

1、用于处理用户名和密码登录的场景。

2、包含了用户名、密码以及是否记住我的功能。

#### BearerToken

1、用于基于 JWT（JSON Web Token）或其他令牌机制的认证。

2、通常在无状态认证场景（如 REST API）中使用。

#### OAuthToken（需要自定义或使用第三方扩展库）

用于 OAuth 2.0 的认证场景，封装了授权码或访问令牌。

### 认证流程

#### 1、用户提交登录请求

用户通过前端（例如登录页面）提交身份和凭证（例如用户名和密码）。

#### 2、生成 AuthenticationToken：

Shiro 的 Subject 会将用户的登录信息封装成一个 AuthenticationToken 对象。

示例：

UsernamePasswordToken token = new UsernamePasswordToken("user123", "password123");

#### 3、传递给 SecurityManager

Subject.login(token) 方法会将 AuthenticationToken 传递给 Shiro 的 SecurityManager。

#### 4、交给 Realm 进行认证

SecurityManager 将 AuthenticationToken 传递给配置的 Realm，由 Realm 查询用户信息并验证身份。

#### 5、认证成功或失败

如果 Realm 验证成功，用户会被认为已登录，Shiro 会生成会话。

如果验证失败，会抛出异常（例如 UnknownAccountException、IncorrectCredentialsException 等）。

### 示例

#### 使用 AuthenticationToken 登录



#### 自定义AuthenticationToken

在一些复杂认证场景下（例如使用邮箱+验证码、JWT 认证等），需要自定义实现 AuthenticationToken 接口。



### 优缺点

#### 优点

##### 1、灵活性

提供了抽象的认证信息接口，可以封装任何类型的认证数据（用户名密码、JWT、OAuth 等）。

##### 2、可扩展性

支持自定义实现，满足复杂业务场景。

##### 3、与 Shiro 流程无缝集成

认证信息通过统一的接口传递到 Realm，方便管理。

#### 缺点

##### 1、复杂性

对于简单项目，需要开发者理解 Shiro 的认证流程并手动实现。

##### 2、敏感信息保护

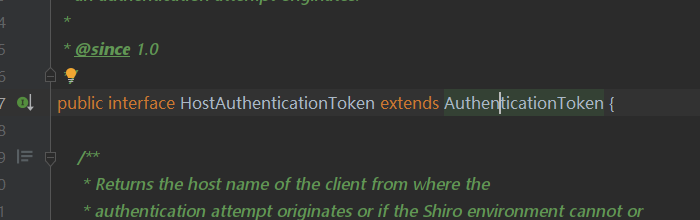
如果处理不当（例如日志中打印密码），可能会导致安全隐患。

## HostAuthenticationToken

### 基本概念

HostAuthenticationToken 是一个接口，用于扩展认证信息的功能，主要在需要获取客户端主机信息的场景下使用。它继承自 AuthenticationToken 接口，并添加了对主机信息（IP 地址或主机名）的支持。

### 源码



### 作用

#### 1、支持主机信息的传递

它在基础的 AuthenticationToken 上增加了一个 getHost() 方法，用于获取提交认证请求的客户端主机信息（例如 IP 地址或主机名）。

这种主机信息在某些场景下（例如安全审计、地理位置验证、异常登录检测等）非常重要。

#### 2、适用于支持主机信息的认证场景

例如，在需要限制登录来源（仅允许特定 IP 地址或主机名登录）的场景下，HostAuthenticationToken 可以提供额外的支持。

#### 3、扩展性

开发者可以自定义实现 HostAuthenticationToken 接口，并在认证逻辑中使用主机信息作为额外的认证依据。

### 实现类

#### UsernamePasswordToken

它实现了 HostAuthenticationToken 接口，能够存储用户名、密码以及客户端主机信息。



### 使用场景

#### ****1、限制登录来源****

通过主机信息（如 IP 地址）限制用户登录。

例如，只允许来自公司内部网络的用户登录。

#### ****2. 安全审计****

记录用户登录时的主机信息，便于安全审计和日志记录。

#### ****3. 地理位置验证****

结合主机信息（IP 地址）与地理位置服务，判断登录请求是否来自可信位置。

### 示例

#### 1、使用HostAuthenticationToken

以下是一个使用主机信息进行身份认证的示例



#### 2、如何在Realm 中使用主机信息

在自定义 Realm 的认证逻辑中，可以通过 token.getHost() 获取主机信息并进行验证：





### 优缺点

#### 优点

##### 1、增强认证功能

提供主机信息作为额外的认证依据，提升了安全性。

##### 2、扩展性强

可以结合主机信息实现更多安全功能（如 IP 黑白名单、地理位置限制等）

#### 缺点

##### 1、需要额外配置

依赖客户端正确传递主机信息，某些场景下（如代理服务器）可能获取不到准确的 IP。

##### 2、复杂性增加

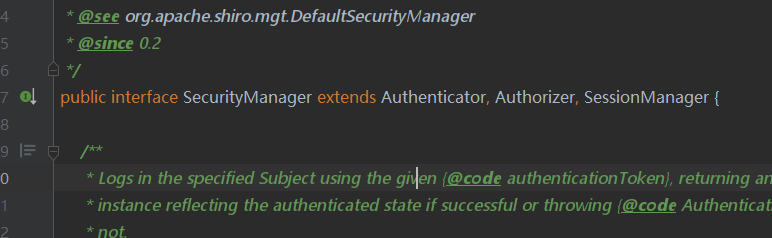
使用主机信息可能引入额外的逻辑，例如处理多种网络环境的差异。

## SecurityManager

### 基本概念

SecurityManager 是 Shiro 安全框架的核心组件之一，负责整个应用程序的安全操作和管理。它协调认证、授权、会话管理等功能，是 Shiro 安全控制的 "大脑"。所有的安全操作（如登录、角色和权限检查、会话管理等）都通过 SecurityManager 来管理。

### 源码



### 作用

#### 1、身份认证管理（Authentication）

SecurityManager 协调并管理所有与身份认证相关的操作，例如登录过程、认证流程等。

#### 2、授权管理（Authorization）

负责用户授权，决定用户是否拥有访问某些资源的权限。

#### 3、会话管理（Session Management）

管理与用户相关的会话数据，允许 Shiro 跟踪用户的会话，并对会话数据进行处理。

#### 4、安全配置中心

作为安全管理的中心，它负责协调和整合所有与安全相关的功能模块（如 Realm、SecurityManager、Subject 等）。

### 方法

#### Subject login(Subject subject, AuthenticationToken authenticationToken)

处理登录操作，验证 AuthenticationToken 并授权 Subject（当前用户）。

参数：

subject：代表当前的用户。

token：携带身份信息的认证 token（如 UsernamePasswordToken）。

返回：执行成功后，用户会被认证，subject 会变为已认证状态。

示例：



#### void logout(Subject subject);

注销当前用户，清理与用户会话相关的所有安全信息。

参数：

subject：需要注销的 Subject 对象。

返回：无返回值，注销后用户状态将被清除。

#### Subject createSubject(SubjectContext context)

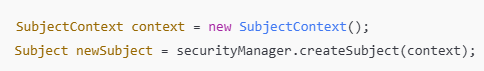
根据上下文创建一个新的 Subject 对象。SubjectContext 可以包含用户的认证信息、角色、权限等信息。

参数：

context：包含有关用户和请求的上下文信息。

返回：返回一个 Subject 实例。

示例：



### 实现类

#### DefaultSecurityManager

Shiro 提供了 SecurityManager 的默认实现类，最常用的是 DefaultSecurityManager。它结合了认证、授权、会话管理和其他安全服务，并将各个功能模块集成在一起。

##### 作用

1、集成认证和授权：

它管理所有 Realm，负责根据不同的 Realm 完成认证和授权操作。

2、会话管理：

管理与用户会话相关的所有事务，支持 Session 机制。

3、协调各组件：

协调 Realm、Session、SecurityManager 和 Subject，确保它们之间的正常工作。

##### 创建和配置 DefaultSecurityManager

在 Shiro 配置中，通常会通过 Spring 配置或 Java 配置来初始化和配置 SecurityManager。以下是一个简单的配置示例：



### SecurityManager 的工作流程

#### 1、身份认证

Subject.login()：将用户的认证信息传递给 SecurityManager。

SecurityManager 根据配置的 Realm 进行用户认证，调用 Realm 的 doGetAuthenticationInfo() 方法，完成身份验证。

#### 2、权限检查

在访问受保护资源时，SecurityManager 会调用 doGetAuthorizationInfo() 方法检查用户是否具有相应的角色或权限。

#### 3、会话管理

SecurityManager 管理与当前用户的会话数据，可以通过 getSession() 方法访问当前用户的会话信息。

### 优缺点

#### 优点

##### 1、集中式安全管理

SecurityManager 是 Shiro 的核心，负责所有安全相关操作的管理，简化了安全策略的集中控制。

##### 2、高可扩展性

可以集成多个 Realm，支持多种认证和授权方式。通过配置不同的 Realm 可以支持不同的数据源（如数据库、LDAP）。

##### 3、会话管理

提供完善的会话管理，支持 Web 会话和非 Web 会话，适用于各种类型的应用。

#### 缺点

##### 1、学习曲线

对于刚接触 Shiro 的开发者，可能需要一些时间来理解 SecurityManager 的配置和工作原理。

##### 2、配置复杂性

SecurityManager 的配置相对复杂，特别是在多个 Realm 和会话管理的情况下，需要详细的配置和调试。

## WebSecurityManager

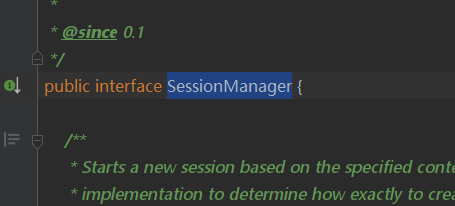
## Session

## SessionManager

### 基本概念

SessionManager 是一个接口，主要用于管理和控制用户的会话。它负责创建、维护和销毁会话，并提供会话相关的信息和功能。SessionManager 是 Shiro 安全框架中的关键组件之一，特别是在 Web 应用程序中，它提供了与会话管理相关的所有功能。

### 源码



### 作用

#### 1、会话的创建

在用户登录时，SessionManager 会为每个用户创建一个新的会话。会话可以包含用户的信息、认证状态、权限等数据。

#### 2、会话的维护

会话管理器会持续跟踪会话的生命周期。它维护与会话相关的数据，例如会话的创建时间、过期时间等。

#### 3、会话的销毁

在用户登出或会话超时后，SessionManager 会销毁会话，确保不会占用过多资源。

#### 4、会话存储

SessionManager 还负责存储和检索会话数据，可以将会话保存在内存、数据库或其他持久化存储中。

#### 5、多种会话方案支持

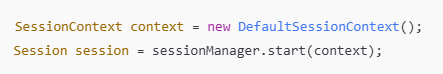
支持通过 Cookie、HTTP 会话或其他方式进行会话存储，适应不同的 Web 应用场景。

### 方法

#### Session start(SessionContext context)

启动并创建一个新的会话。

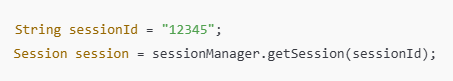
示例：



#### Session getSession(SessionKey key)

根据会话 ID 获取指定的会话。

示例：



### 会话管理流程

#### 1、创建会话

（1）当用户首次登录时，SessionManager 会创建一个新的会话，并返回该会话对象。

（2）会话对象包含用户的认证信息、权限、角色等数据。

#### 2、会话维持

（1）会话会保持一定的有效期，可以根据配置设置会话的过期时间。

（2）会话可以存储用户在会话期间产生的状态信息，如用户偏好设置。

#### 3、会话销毁

（1）当用户退出时，SessionManager 会销毁会话并清除会话数据。

（2）如果会话过期，SessionManager 也会自动销毁会话。

#### 4、会话存储

会话数据可以存储在内存、数据库或其他存储介质中。通过实现 SessionDAO，可以配置不同的存储方案。

### SessionManager 与其他组件的关系

#### 1、与 Subject 的关系

（1）SessionManager 和 Subject 紧密集成。在 Shiro 中，Subject 代表当前用户，SessionManager 负责管理与该用户相关的会话信息。

（2）每个 Subject 都会关联一个会话，SessionManager 负责会话的创建、维护和销毁。

#### 2、与 SecurityManager 的关系

SecurityManager 是 Shiro 的核心组件，负责协调安全相关的操作。SessionManager 是 SecurityManager 的一部分，负责处理与会话管理相关的事务。

#### 3、与 SessionDAO 的关系

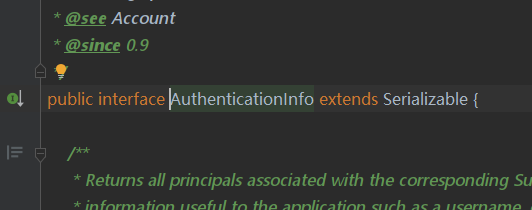
SessionDAO 是会话的持久化层，负责会话的存储和检索。SessionManager 使用 SessionDAO 来实现会话的持久化。

## AuthenticationInfo

### 基本概念

AuthenticationInfo 接口是认证过程的核心组件之一。它的主要作用是封装用户的认证信息，用于与用户提交的身份信息（如用户名和密码）进行比较，以完成身份认证过程。

### 源码



### 作用

#### 1、封装认证信息

AuthenticationInfo 是 Shiro 认证体系中的重要接口，包含与用户身份认证相关的信息。

它将从数据源（如数据库、LDAP）中获取的用户信息封装为一个对象，用于后续的认证过程。

#### 2、验证凭证

AuthenticationInfo 中存储的用户信息会与用户提交的认证凭据（如密码）进行比对。

比对过程由 Shiro 的认证模块（如 CredentialsMatcher）负责完成。

#### 3、支持多种数据源

由于 AuthenticationInfo 是一个接口，它的实现可以适配多种数据源（如数据库、文件、LDAP 等）。

#### 4、提供扩展性

开发者可以自定义 AuthenticationInfo 的实现类，以满足特殊的认证需求。

### 常见实现类

#### SimpleAuthenticationInfo

用途：用于大多数简单的认证场景。

特点：将用户的身份信息和凭证简单地封装到一个对象中。

示例：



### 工作原理

#### 1、认证过程启动

用户提交身份认证信息（如用户名和密码）时，Shiro 会调用 Realm 的 doGetAuthenticationInfo() 方法。

#### 2、AuthenticationInfo 的返回

doGetAuthenticationInfo() 方法从数据源中查询用户信息，将其封装为一个 AuthenticationInfo 对象返回。

#### 3、凭证验证

Shiro 的 CredentialsMatcher 会对用户提交的凭证（如密码）和 AuthenticationInfo 中的凭证进行比对。

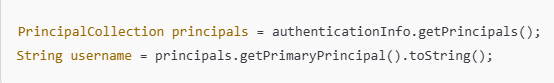
如果匹配成功，则认证通过；否则认证失败。

### 方法

#### PrincipalCollection getPrincipals()

返回用户的身份信息（如用户名或用户标识），身份信息可以是单一值（如用户名），也可以是多个值的集合（如用户名、用户 ID）。

示例：



#### Object getCredentials();

返回用户的凭证信息（如密码）。Credentials 是认证的核心，用于与用户提交的凭证进行比对。

示例：



### 认证流程

#### 1、用户提交凭证

用户通过 Subject.login() 方法提交认证凭据。

#### 2、传递给 Realm

Shiro 将用户提交的凭据交给配置的 Realm，调用其 doGetAuthenticationInfo() 方法。

#### 3、AuthenticationInfo 返回

Realm 从数据源中查找用户信息，并将其封装为 AuthenticationInfo 对象返回。

#### 4、凭据验证

Shiro 使用配置的 CredentialsMatcher 对用户提交的凭据和 AuthenticationInfo 进行比对。

#### 5、认证结果

如果凭据匹配，则认证通过；否则抛出认证异常（如 IncorrectCredentialsException）。

### 优缺点

#### 优点

##### 1、封装用户数据

将用户的认证信息集中封装，便于认证逻辑的统一处理。

##### 2、支持扩展性

开发者可以通过实现 AuthenticationInfo 接口自定义认证逻辑。

##### 3、安全性强

结合 CredentialsMatcher 和盐值，可以实现更安全的认证过程。

#### 缺点

##### 1、需要配合其他组件

AuthenticationInfo 需要与 Realm 和 CredentialsMatcher 协同工作，单独使用意义不大。

##### 2、对安全性要求高

如果 Realm 中的 AuthenticationInfo 实现不当，可能导致认证漏洞。

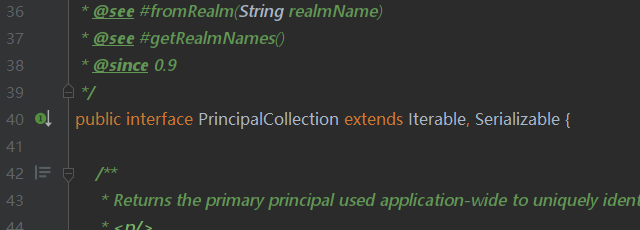
## AuthorizationInfo

## PrincipalCollection

### 基本概念

PrincipalCollection 是一个用于封装用户身份信息的接口。它是认证和授权过程中非常重要的一个类，主要用于表示一个或多个身份标识（principal）。Shiro 使用 PrincipalCollection 来管理与用户相关的所有身份信息（如用户名、用户 ID 等），在认证、授权过程中传递用户的身份信息。

### 源码



### 作用

#### 1、封装用户身份信息

（1）PrincipalCollection 用于封装一个或多个用户的身份标识（例如用户名、用户 ID、邮箱等）。这些信息通常由用户提交并用于身份验证和授权。

（2）一个用户可能有多个身份标识（例如用户名和邮箱），PrincipalCollection 可以将这些标识集中管理。

#### 2、在 Shiro 的认证和授权过程中传递身份信息

PrincipalCollection 会被传递到 Shiro 的 Realm、SecurityManager、Subject 等组件，用于执行身份验证、授权和访问控制。

#### 3、支持多重身份

一个用户可能有多个身份标识（例如用户名、邮箱等）。PrincipalCollection 可以封装这些多重身份，供认证和授权使用。

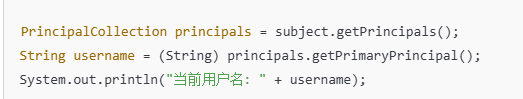
#### 4、用于权限验证和会话管理

在授权检查时，Shiro 会使用 PrincipalCollection 中的信息来判断用户是否具备访问某个资源的权限。

### 方法

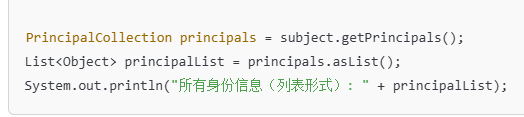
#### Object getPrimaryPrincipal();

返回 PrincipalCollection 中的主身份标识。通常这是用户的第一个身份（如用户名。



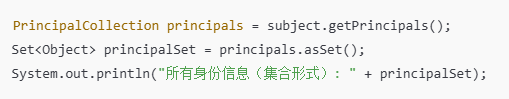
#### List asList();

返回 PrincipalCollection 中所有的身份标识，按照列表顺序排列。



#### Set asSet()

返回 PrincipalCollection 中的身份标识，去重后返回一个集合（Set）



### 应用场景

1、适用于多身份管理的场景，支持一个用户的多个身份标识（如用户名、邮箱等）。

2、在 Shiro 的认证和授权过程中，PrincipalCollection 用于存储和传递用户的身份信息。

## CredentialsMatcher

## SaltedAuthenticationInfo

## RememberMeManager

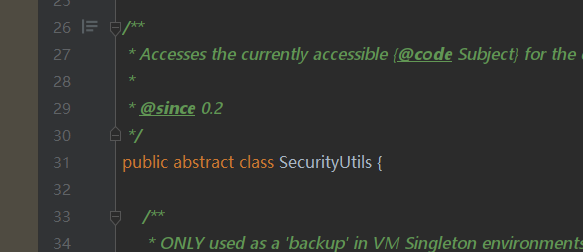
# 类

## SecurityUtils

### 基本概念

SecurityUtils 是一个工具类，提供了一组静态方法，用于简化对 Shiro 核心功能的访问和操作。它是开发者使用 Shiro 框架的入口之一，主要用于获取当前用户的 Subject（用户上下文）或配置 SecurityManager。

### 源码



### 作用

#### 1、获取当前用户的 Subject

SecurityUtils 提供了获取当前用户会话信息的便捷方法。Subject 是 Shiro 中用户认证和授权的核心接口，代表了与当前用户会话的交互。

通过 SecurityUtils.getSubject()，开发者可以获取当前线程关联的 Subject，从而实现身份认证、权限校验等功能。

#### 2、管理全局 SecurityManager

SecurityManager 是 Shiro 的核心组件，负责管理所有的认证、授权、会话和缓存操作。

SecurityUtils 提供静态方法用于设置或获取全局的 SecurityManager。

#### 3、简化 Shiro 的操作

通过 SecurityUtils，开发者无需手动管理 Subject 或 SecurityManager 的生命周期和依赖注入，直接调用静态方法即可完成相关操作。

### 方法

#### public static Subject getSubject()

获取当前线程绑定的 Subject 对象。

用途：

1、获取当前用户的认证信息。

2、执行身份认证（登录）操作。

3、检查权限（授权）。

示例：



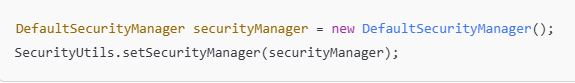
#### public static void setSecurityManager(SecurityManager securityManager)

设置全局的 SecurityManager 实例。

在应用程序启动时配置 Shiro 的核心组件 SecurityManager。

通常配置在启动代码中，如 Spring 或独立 Java 应用中。

示例：



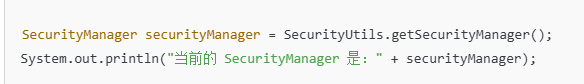
#### public static SecurityManager getSecurityManager()

获取当前全局的 SecurityManager 实例。

用途：

在某些场景中需要直接操作 SecurityManager 时调用。

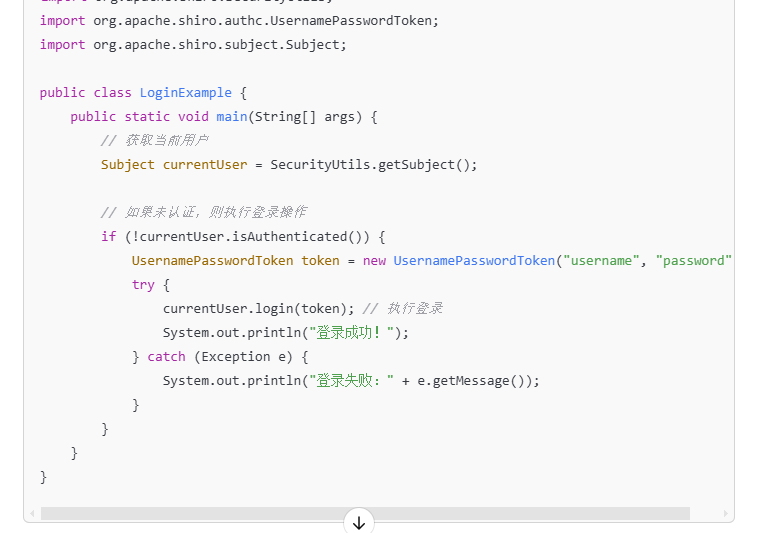
示例：



### 典型用法

#### 1、用户登录

SecurityUtils 与 Subject 配合，可以实现用户的登录认证逻辑。



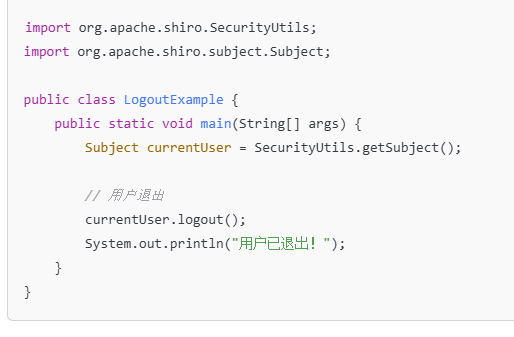
#### 2、用户权限校验

使用 SecurityUtils 获取当前 Subject，然后调用其授权相关方法（如 isPermitted）进行权限校验。



#### 3、用户退出

通过 SecurityUtils 获取当前 Subject，然后调用 logout() 方法实现用户退出



### 核心原理

#### 1、SecurityManager 全局管理

SecurityUtils 是 Shiro 框架中访问 SecurityManager 和 Subject 的统一入口。SecurityManager 管理所有与安全相关的操作，SecurityUtils 则提供对其的简化访问。

#### 2、线程绑定 Subject

每个线程都会绑定一个 Subject 对象，这个绑定是通过 ThreadContext 实现的。

SecurityUtils.getSubject() 方法会从 ThreadContext 中获取与当前线程关联的 Subject。

#### 3、静态方法封装

SecurityUtils 的方法本质上是对 Shiro 核心功能的静态封装，使开发者无需直接操作低层组件。

### 优缺点

#### 优点

##### 1、简化操作

提供静态方法，开发者可以方便地获取核心组件（如 Subject 和 SecurityManager），简化开发流程。

##### 2、统一入口

通过一个工具类管理 Shiro 的主要功能，增强了代码的可读性和一致性。

##### 3、线程安全

Subject 的线程绑定机制确保了线程安全性，每个线程都有独立的用户会话上下文。

#### 缺点

##### 1、与 Shiro 强耦合

使用 SecurityUtils 会使代码对 Shiro 框架产生依赖，不利于框架替换或扩展。

##### 2、适用范围有限

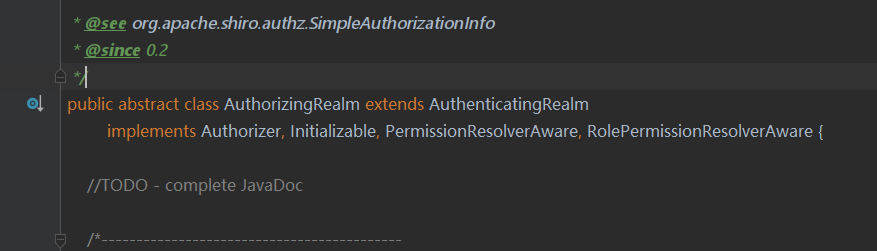
SecurityUtils 假设应用程序的安全管理器已经配置完成，否则调用会导致运行时异常。

## AuthorizingRealm

### 基本概念

在 Apache Shiro 框架中，AuthorizingRealm 是一个非常重要的类，它主要负责 **认证**（Authentication）和 **授权**（Authorization）操作的逻辑实现。通过继承和实现该类，开发者可以自定义用户的身份验证和权限控制。

### 源码



### 作用

AuthorizingRealm 是 Shiro 提供的一个抽象类，继承自 Realm，并扩展了认证和授权功能。

#### 1、认证

（1）验证用户提供的凭证（如用户名和密码）是否正确。

（2）认证成功后，将用户信息存储到 AuthenticationInfo 对象中，供后续的授权使用。

#### 2、授权

（1）确定用户是否具备某些权限或角色。

（2）从数据库或其他数据源中加载用户的角色和权限信息，并将其封装到 AuthorizationInfo 对象中。

#### 3、缓存支持

AuthorizingRealm 提供了缓存支持，可以缓存认证和授权信息，提升性能。

### 核心方法

#### AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)

1、负责用户的身份认证。

2、核心逻辑是根据传入的 AuthenticationToken，从数据库或其他数据源中查询用户信息，并返回一个 AuthenticationInfo 对象。

3、如果认证失败，抛出异常（如 UnknownAccountException 或 IncorrectCredentialsException）。

示例：

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

// 获取用户输入的用户名

String username = (String) token.getPrincipal();

// 查询数据库，获取用户信息

User user = userService.findByUsername(username);

if (user == null) {

throw new UnknownAccountException("用户不存在");

}

// 返回认证信息 (参数: 用户名, 密码, 当前 Realm 的名称)

return new SimpleAuthenticationInfo(user.getUsername(), user.getPassword(), getName());

}

#### AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)

1、负责用户的授权。

2、核心逻辑是根据认证后的用户信息（PrincipalCollection），从数据库或其他数据源中查询用户的角色和权限信息，并返回一个 AuthorizationInfo 对象。

示例：

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// 获取当前登录用户的用户名

String username = (String) principals.getPrimaryPrincipal();

// 查询数据库，获取用户的角色和权限信息

Set<String> roles = userService.findRolesByUsername(username);

Set<String> permissions = userService.findPermissionsByUsername(username);

// 返回授权信息

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

info.setRoles(roles); // 设置用户角色

info.setStringPermissions(permissions); // 设置用户权限

return info;

}

### 工作流程

1、登录（认证）：

用户登录时，Shiro 调用 doGetAuthenticationInfo 方法。

如果用户输入的用户名和密码与数据库中存储的一致，则登录成功，否则抛出异常。

2、访问受保护资源（授权）：

当用户访问需要权限的资源时，Shiro 调用 doGetAuthorizationInfo 方法。

该方法返回用户的角色和权限信息，Shiro 根据这些信息判断是否允许访问。

3、缓存机制：

默认情况下，AuthorizingRealm 会将授权信息缓存到内存中，减少数据库查询次数。

### 优缺点

#### 优点

##### 1、灵活性

可以自定义认证和授权逻辑，满足复杂的业务需求。

##### 2、缓存支持

内置缓存机制，大幅提升性能。

##### 3、高扩展性

通过继承 AuthorizingRealm，可以轻松扩展 Shiro 的功能。

#### 缺点

##### 1、手动实现逻辑

需要开发者手动实现认证和授权逻辑，对于复杂系统，代码量较多。

##### 2、权限管理复杂性

如果角色和权限设计不合理，可能导致权限管理的复杂度增加。

### 总结

1、AuthorizingRealm 的作用： 提供认证和授权的核心逻辑。

2、开发流程：

重写 doGetAuthenticationInfo 实现认证逻辑。

重写 doGetAuthorizationInfo 实现授权逻辑。

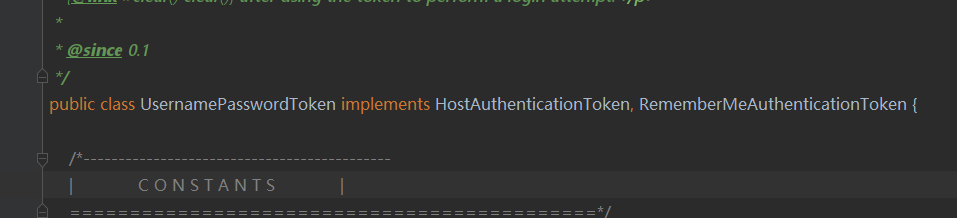
3、使用场景： 适用于需要用户认证和权限管理的系统，是 Shiro 权限控制的核心部分。

## UsernamePasswordToken

### 基本概念

UsernamePasswordToken 是一个非常重要的类，主要用于处理用户认证时的登录凭证（如用户名和密码）。它封装了用户的 **用户名** 和 **密码**，并将这些信息传递到 Shiro 的认证机制中。

### 源码



### 作用

#### 1、封装用户登录凭证

（1）主要用于存储用户提交的用户名和密码信息（即用户的登录凭证）。

（2）避免直接操作用户名和密码，增强代码的可读性和安全性。

#### 2、传递登录信息

通过将 UsernamePasswordToken 对象传递给 Shiro 的认证方法（如 Subject.login()），认证机制会使用该对象中的信息进行验证。

#### 3、支持额外信息

除了用户名和密码，还支持存储其他信息，例如 rememberMe（记住我功能）等。

### 属性

#### username

用户名。

#### password

用户密码（以 char[] 数组形式存储，增强安全性）。

#### rememberMe

是否启用“记住我”功能。

#### host

登录的主机地址（例如，IP 地址）。

### 构造方法

#### public UsernamePasswordToken(String username, char[] password)

基本构造方法：只封装用户名和密码

#### public UsernamePasswordToken(String username, char[] password, boolean rememberMe);

可支持 RememberMe 功能的构造方法

#### public UsernamePasswordToken(String username, char[] password, boolean rememberMe, String host);

可支持指定 Host 的构造方法

### 工作流程

#### 1、用户登录时创建 UsernamePasswordToken 对象：

用户在登录表单中输入用户名和密码。

后端通过表单数据创建 UsernamePasswordToken 对象。

#### 2、通过 Shiro 的认证方法进行登录：

将 UsernamePasswordToken 对象传递给 Shiro 的 Subject.login() 方法。

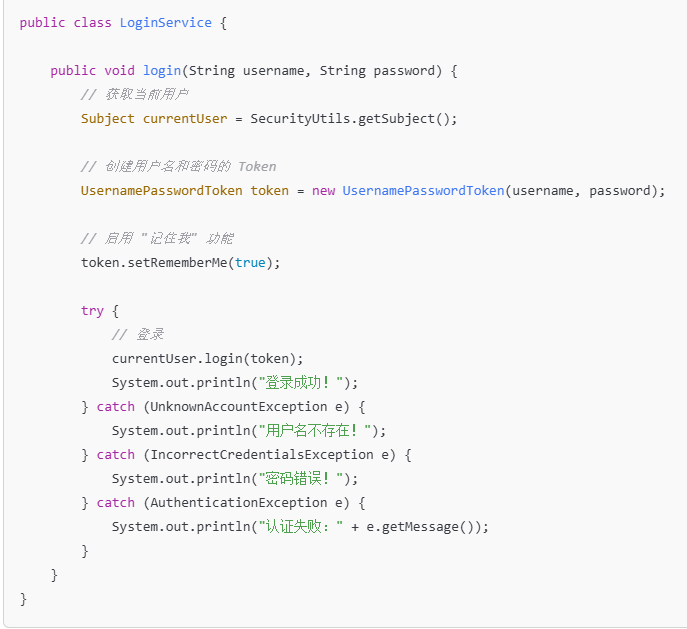
认证逻辑会验证用户名和密码是否正确。

#### 3、认证通过或失败

如果认证通过，则登录成功。

如果认证失败（如用户名不存在或密码错误），会抛出相应异常。

### 使用示例



### 优缺点

#### 优点

##### 1、封装性

将用户名和密码封装成一个对象，简化了代码逻辑。

##### 2、灵活性

支持扩展（如 rememberMe 和 host）。

##### 3、安全性

采用 char[] 存储密码，降低敏感信息泄露的风险。

#### 缺点

##### 1、依赖性

只能用于 Shiro 框架。

##### 2、扩展性限制

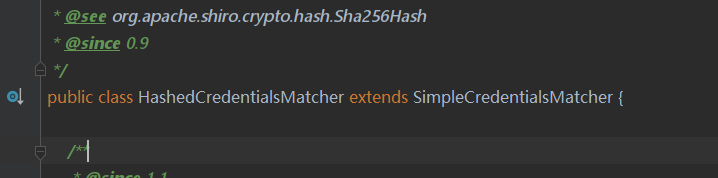
在处理复杂的登录需求时，可能需要自定义 Token。

## HashedCredentialsMatcher

### 基本概念

HashedCredentialsMatcher 是一个用于对用户的凭证（如密码）进行哈希处理的类。它主要用于在认证时验证用户输入的密码是否与存储在数据库中的密码匹配。HashedCredentialsMatcher 提供了密码的哈希比较功能，通过加密算法对密码进行哈希并进行比对，确保安全地存储和验证用户的凭证。

### 源码



### 作用

#### 1、密码存储的安全性

在 Shiro 中，用户的密码通常不会以明文形式存储，而是通过哈希算法对密码进行加密后存储。HashedCredentialsMatcher 在认证时，将用户输入的密码进行相同的哈希运算，然后与存储在数据库中的哈希值进行比对。

#### 2、哈希验证

HashedCredentialsMatcher 不会直接将明文密码与数据库中的密码进行比较，而是将输入密码与存储的哈希值进行比较。通过这种方式，可以有效防止密码泄露，因为即使数据库中的密码被泄露，攻击者也无法直接得到明文密码。

#### 3、支持盐值

为了增加哈希算法的安全性，HashedCredentialsMatcher 支持使用盐（salt）。盐是一个额外的随机字符串，它与密码一起被哈希，从而增加了哈希值的复杂性，避免了暴力破解攻击。

### 工作流程

#### 1、用户输入密码

用户在登录时输入密码。

#### 2、密码哈希化

HashedCredentialsMatcher 会使用配置的哈希算法（如 SHA-256）对用户输入的密码进行哈希运算。

#### 3、比对哈希值

HashedCredentialsMatcher 将用户输入的密码哈希值与数据库中存储的哈希值进行比较。如果两者相同，表示密码匹配，认证通过；如果不同，则认证失败。

### 主要功能

#### 1、支持哈希算法

HashedCredentialsMatcher 支持多种哈希算法（如 MD5、SHA-1、SHA-256 等）。可以根据需要配置不同的哈希算法。

#### 2、盐值支持

可以使用盐（salt）增强密码的安全性，避免使用简单的哈希算法（如 MD5）暴力破解攻击。

#### 3、迭代次数

HashedCredentialsMatcher 支持指定哈希算法的迭代次数，增加破解难度。

### 配置

在自定义的 Realm 中，通过调用 setCredentialsMatcher() 方法来配置 HashedCredentialsMatcher。





#### 带盐值的配置

如果密码存储和验证中需要使用盐值，可以扩展 SimpleAuthenticationInfo 的构造方法来包括盐。



在这个例子中，ByteSource.Util.bytes(salt) 用于将盐值封装成 ByteSource，这样 HashedCredentialsMatcher 会自动在哈希计算时使用这个盐值。

#### 整合到springboot中

在 Spring Boot 中，通常通过配置类来注册 Realm 和 HashedCredentialsMatcher。

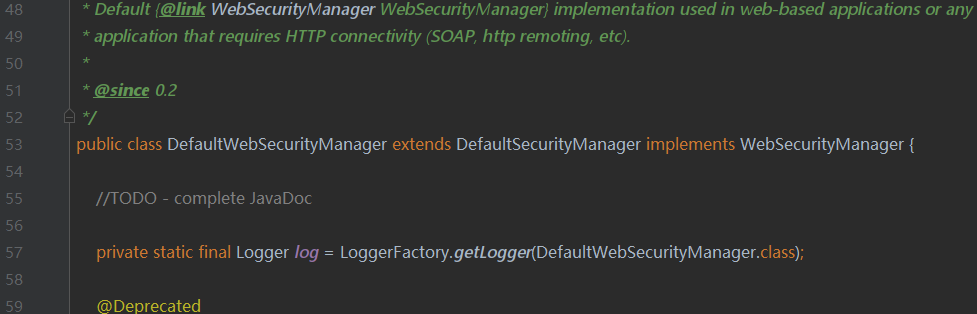


## DefaultWebSecurityManager

### 基本概念

DefaultWebSecurityManager 是 SecurityManager 的一个具体实现类，专门为 Web 应用程序设计。它继承了 Shiro 的核心 DefaultSecurityManager 类，并增强了 Web 场景下的安全管理功能。

### 源码



### 作用

#### 1、安全管理器核心组件

DefaultWebSecurityManager 是 Web 应用程序的核心安全管理组件，负责认证、授权、会话管理和缓存等操作。

它结合了 Shiro 的安全特性并优化了 Web 环境下的使用。

#### 2、管理 Web 特定的会话

支持 Web 环境下的会话管理，使用 Shiro 提供的 WebSessionManager 实现会话管理（如 Cookie 和 HTTP 会话支持）。

通过整合 Web 会话功能，可以统一管理用户会话生命周期。

#### 3、支持 HTTP 请求和响应的处理

通过与 WebSecurityManager 接口结合，提供对 HTTP 请求、响应等 Web 特定上下文的支持。

#### 4、整合 Realm

DefaultWebSecurityManager 会与开发者自定义的 Realm 交互，完成认证和授权逻辑。

可以配置多个 Realm 来处理不同的数据源或认证方式。

#### 5、提供缓存支持

集成了缓存机制，通过缓存提高了认证和授权的性能（如用户的权限信息和登录状态的缓存）。

#### 6、支持记住我（RememberMe）功能

内置了 "记住我" 的功能，通过 Cookie 实现用户的长时间登录状态维护。

### 主要功能

#### 1、认证管理

负责处理用户的登录认证逻辑，将用户凭据提交给配置的 Realm 验证，返回认证结果。

#### 2、授权管理

负责检查用户是否具有访问某个资源的权限，或是否属于某个角色。

#### 3、会话管理

使用 SessionManager 管理用户的会话信息。

提供与 Web 环境兼容的会话机制，例如支持基于 Cookie 的会话管理。

#### 4、缓存支持

集成 Shiro 的缓存机制，减少对数据源的频繁访问，提高系统性能。

#### 5、支持多 Realm 配置

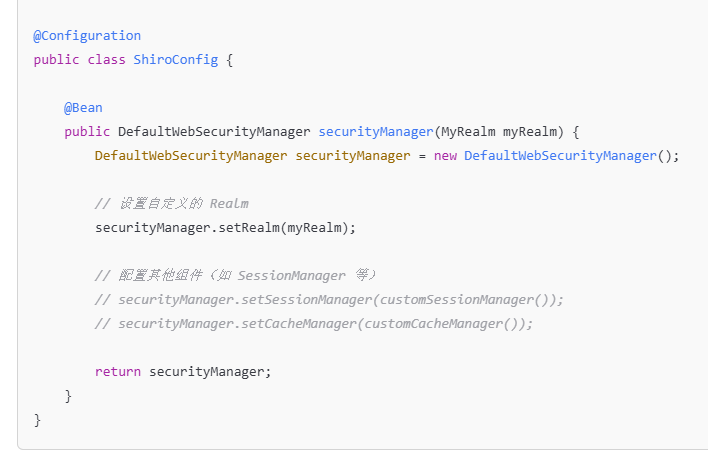
允许开发者同时配置多个 Realm，按顺序或策略进行认证和授权。

### 典型配置

在 Spring Boot 整合 Shiro 的场景中，DefaultWebSecurityManager 通常需要配置到上下文中，并与其他组件（如 Realm、SessionManager）关联。

#### 示例代码

##### 1、定义 SecurityManager Bean



##### 2、定义自定义 Realm

Realm 是 DefaultWebSecurityManager 的核心组件之一，用于处理具体的认证和授权逻辑。



### 核心组件

#### 1、Realm

Realm 是 Shiro 进行认证和授权的桥梁。

在 DefaultWebSecurityManager 中，可以配置一个或多个 Realm。

#### 2、SessionManager

默认使用 DefaultWebSessionManager 来管理用户会话。

在 Web 场景下，支持基于 Cookie 的会话管理。

#### 3、RememberMeManager

负责实现 "记住我" 功能，通常通过 Cookie 存储用户信息。

#### 4、CacheManager

提供缓存支持，用于存储用户的权限信息和会话信息，提高系统性能。

#### 5、Authenticator 和 Authorizer

Authenticator 负责用户的认证操作。

Authorizer 负责用户的授权操作。

### 优缺点

#### 优点

##### 1、简化 Web 场景开发

为 Web 应用提供了默认的会话和 Cookie 管理功能，无需手动实现。

##### 2、模块化设计

可插拔的设计允许开发者自定义各个组件，如自定义 Realm 或 SessionManager。

##### 3、与 Shiro 无缝集成

提供了 Shiro 的全部功能，如认证、授权、缓存和会话管理。

##### 4、扩展性强

可以通过继承和配置实现高度的扩展和定制。

#### 缺点

##### 1、需要手动配置

对于复杂的 Web 应用，需要编写大量的配置代码。

##### 2、依赖性强

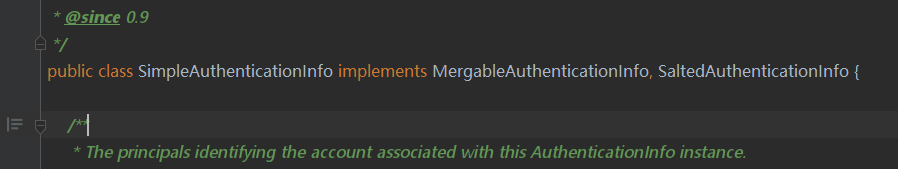
依赖于 Shiro 的核心架构，对框架的理解要求较高。

## SimpleAuthenticationInfo

### 基本概念

SimpleAuthenticationInfo 是一个非常重要的类，它用于封装与用户认证相关的信息。在认证流程中，SimpleAuthenticationInfo 主要用于保存从数据源（如数据库、LDAP、内存等）中获取的用户身份信息和凭证信息，供 Shiro 的认证模块进行比对。

### 源码



### 作用

SimpleAuthenticationInfo 类的作用是封装用户的认证信息（身份和凭证），包括：

**身份**（Principal）：唯一标识用户的信息，例如用户名、邮箱地址或用户 ID。

**凭证**（Credentials）：用户的认证凭证，例如密码、令牌等。

**盐值**（Salt）：增强密码哈希的安全性，通过加入盐值避免简单的哈希被暴力破解。

**Realm 名称**：标识该认证信息来自哪个 Realm。

### 使用场景

SimpleAuthenticationInfo 类通常在自定义 Realm 的 doGetAuthenticationInfo 方法中使用。当 Shiro 的认证机制调用 Realm 的认证方法时，doGetAuthenticationInfo 会返回一个 SimpleAuthenticationInfo 对象，Shiro 会利用该对象的内容（如凭证和盐值）来完成认证操作。

### 工作流程

#### 1、用户输入凭证

用户通过登录表单或 API 提供身份和凭证（如用户名和密码）。

#### 2、Realm 查询数据

Realm 从数据源（如数据库）中查询用户信息，包括身份（如用户名）和经过加密的密码（以及盐值）。

#### 3、构造认证信息对象

Realm 使用从数据源中查询到的用户信息创建一个 SimpleAuthenticationInfo 对象。

#### 4、返回认证信息

将 SimpleAuthenticationInfo 对象返回给 Shiro 的认证模块。

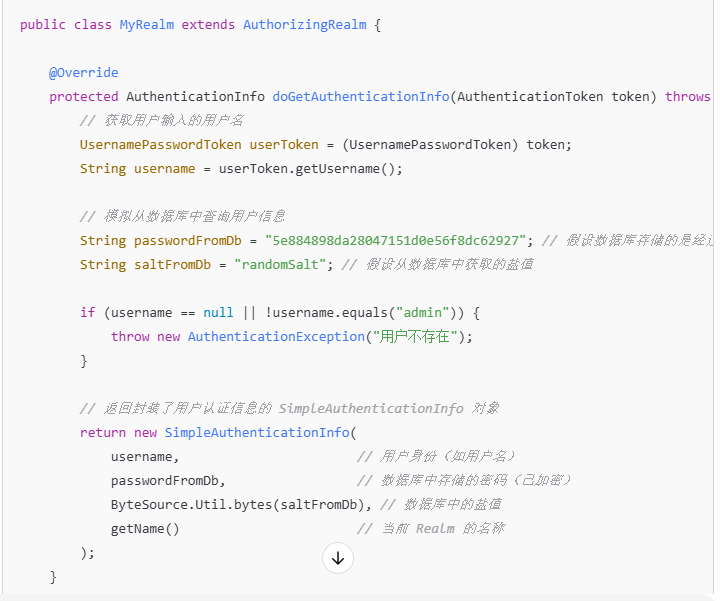
#### 5、Shiro 认证模块验证凭证

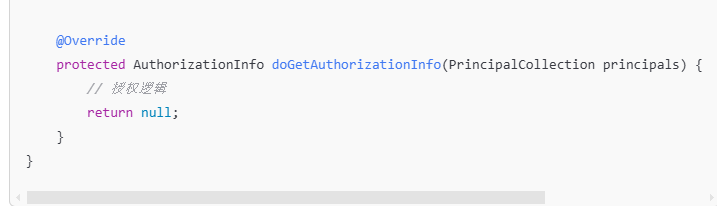
Shiro 使用 CredentialsMatcher（如 HashedCredentialsMatcher）对用户输入的凭证进行比对，验证是否匹配。

#### 6、认证结果

如果凭证验证通过，用户登录成功；否则登录失败。

### 示例代码



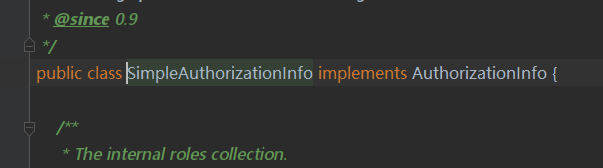


## SimpleAuthorizationInfo

### 基本概念

在 Apache Shiro 框架中，SimpleAuthorizationInfo 是一个实现了 AuthorizationInfo 接口的类，用于封装用户的授权信息（即角色和权限信息）。它是 Shiro 用于授权过程中的关键组件之一，主要作用是存储并传递用户的角色、权限等信息，在用户访问某个资源时，Shiro 会根据这些信息来判断是否允许访问。

### 源码



### 作用

#### 1、封装授权信息

（1）SimpleAuthorizationInfo 用于封装用户的角色和权限信息，这些信息通常是从 Realm 中获取的，并用于授权过程。

（2）它通过 setRoles() 和 setStringPermissions() 方法将角色和权限信息存储到对象中，供 Shiro 进行后续的授权判断。

#### 2、传递角色和权限信息

在用户进行授权检查时，SimpleAuthorizationInfo 作为授权信息的载体，传递给 SecurityManager 和 Subject，Shiro 通过这些信息判断用户是否有权限访问某个资源。

#### 3、支持多种授权策略

SimpleAuthorizationInfo 允许开发者根据需要设置用户的角色（roles）和权限（permissions）。它支持将角色和权限以集合的形式进行管理。

### 方法

#### public void setRoles(Set<String> roles)

设置用户的角色信息。

在授权过程中，Shiro 会通过角色信息判断用户是否具备访问特定资源的权限。

示例：



#### public void setStringPermissions(Set<String> stringPermissions)

设置用户的权限信息。

在授权过程中，Shiro 会通过权限信息判断用户是否具备执行特定操作的权限。

示例：



#### public void setObjectPermissions(Set<Permission> objectPermissions)

设置用户的权限信息（以对象的形式）。这个方法与 setStringPermissions() 类似，不过它用于存储权限信息作为对象。

如果需要将权限以更复杂的对象形式传递，可以使用这个方法。

示例：



#### public void addRole(String role)

向角色集合中添加一个角色

如果你只想动态地向角色列表中添加一个角色而不是重新设置整个角色集合，可以使用该方法。

示例：



#### public void addRoles(Collection<String> roles)

#### public void addStringPermission(String permission)

向权限集合中添加一个权限。

如果你只想动态地向权限列表中添加一个权限而不是重新设置整个权限集合，可以使用该方法。

示例：



#### public void addStringPermissions(Collection<String> permissions)

### SimpleAuthorizationInfo 与授权过程的关系

在 Shiro 的授权过程中，SimpleAuthorizationInfo 作为授权信息被 Realm 的 doGetAuthorizationInfo() 方法返回，并用于判断用户是否具有访问某个资源的权限。

#### 授权流程

1、用户请求访问某个受保护资源时，Shiro 会调用 Subject.isPermitted() 或 Subject.checkPermission() 方法来判断用户是否有足够的权限。

2、Shiro 会从 SecurityManager 获取到当前 Subject，并通过 Realm 来获取该用户的授权信息（AuthorizationInfo）。

3、Realm 会返回一个 SimpleAuthorizationInfo 对象，该对象包含用户的角色和权限信息。

4、Shiro 会根据 SimpleAuthorizationInfo 中的角色和权限信息，判断用户是否允许访问该资源。

#### 示例



### 优缺点

#### 优点

##### 1、简单易用

SimpleAuthorizationInfo 提供了简单直观的接口来管理角色和权限，易于理解和使用。

##### 2、灵活性强

通过 setRoles() 和 setStringPermissions()，开发者可以灵活地设置用户的角色和权限，也可以使用 addRole() 和 addStringPermission() 动态添加角色和权限。

##### 3、适应多种需求

支持基本的角色和权限管理，也支持通过 setObjectPermissions() 来处理复杂的权限需求。

##### 4、与 Shiro 集成紧密

SimpleAuthorizationInfo 是 Shiro 的标准授权信息实现，与 SecurityManager 和 Realm 紧密集成，能够方便地进行授权检查。

#### 缺点

##### 1、有限的权限类型支持

SimpleAuthorizationInfo 默认只支持基于字符串的权限表示。如果需要更复杂的权限类型（如基于资源的权限控制），可能需要自定义权限类或使用其他 Shiro 组件。

##### 2、可能过于简单

对于复杂的权限控制，SimpleAuthorizationInfo 可能不够灵活，可能需要定制更复杂的权限模型。

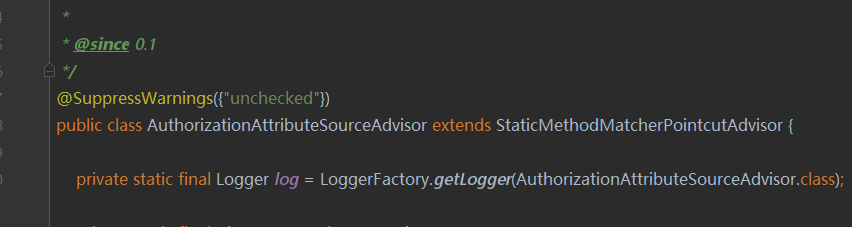
## WildcardPermission

## AuthorizationAttributeSourceAdvisor

### 基本概念

AuthorizationAttributeSourceAdvisor 类是 Shiro 的权限控制系统的一部分，负责在 AOP（面向切面编程）层面拦截带有授权注解（如 @RequiresPermissions、@RequiresRoles 等）的方法，并根据这些注解中的权限要求来执行访问控制。

### 源码



### 核心功能

#### **1、**授权注解的支持

AuthorizationAttributeSourceAdvisor 通过结合 Spring AOP，支持基于注解的方法级授权。它能够解析如 @RequiresRoles、@RequiresPermissions 等注解，并在方法执行前验证当前用户是否满足相应的权限要求。

#### 2、与 AOP 配合使用

Shiro 使用 AOP 作为授权检查的一部分，AuthorizationAttributeSourceAdvisor 就是一个 AOP 顾问（Advisor）。它通过拦截被授权注解标记的方法，在方法执行之前检查当前用户是否具有访问该方法的权限。

#### 3、权限检查与决策

在方法执行之前，AuthorizationAttributeSourceAdvisor 会根据注解中定义的角色或权限要求，通过 Shiro 的 SecurityManager 和 Realm 来检查当前用户是否具备相应的权限。如果权限检查通过，则允许方法执行；如果权限不足，则抛出 AuthorizationException 或类似的异常。

### 工作流程

#### 1、方法注解与权限配置

当方法上标注了 @RequiresRoles、@RequiresPermissions 等注解时，Shiro 会通过 AOP 拦截这些方法。AuthorizationAttributeSourceAdvisor 会读取这些注解的信息。

#### 2、拦截并检查权限

当方法被调用时，AuthorizationAttributeSourceAdvisor 会检查当前用户是否符合注解中的要求。它会通过 SecurityManager 来调用相关的 Realm，并使用 Subject 来判断用户是否具有相应的权限或角色。

#### 3、决定是否允许访问

如果用户拥有注解所要求的权限，则方法执行继续；如果没有权限，则抛出异常（通常是 AuthorizationException）。

### 示例代码



执行 performAdminAction 方法时，Shiro 会通过 AuthorizationAttributeSourceAdvisor 拦截该方法并检查 @RequiresRoles 注解中定义的角色要求。如果当前用户具有 admin 角色，方法会被正常执行；如果没有，则会抛出 AuthorizationException

### 配置AuthorizationAttributeSourceAdvisor

AuthorizationAttributeSourceAdvisor 需要在 Shiro 配置中启用。通常，需要一个 Shiro AOP 的配置类来注册该 Advisor。

示例代码：





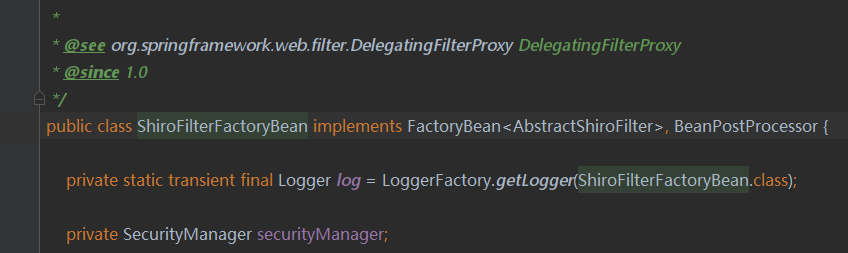
在这个配置中，我们通过 authorizationAttributeSourceAdvisor 方法将 AuthorizationAttributeSourceAdvisor 注册为一个 Bean。这个 Bean 会自动启用 AOP 权限控制，使得带有授权注解的方法能够被 Shiro 拦截并进行权限检查。

## ShiroFilterFactoryBean

### 基本概念

在 Apache Shiro 框架中，ShiroFilterFactoryBean 是一个非常重要的类，主要用于将 Shiro 的安全控制集成到基于 Servlet 的 Web 应用程序中，特别是在 Spring 框架中。它用于配置 Shiro 的过滤器链，并将其与 Spring 的 Filter 管理器进行集成，从而使得 Shiro 的认证、授权、会话管理等功能能够在 Web 应用中生效。

### 源码



### 作用

#### 1、配置 Shiro 的 Web 安全过滤器

ShiroFilterFactoryBean 用于定义一个过滤器链（FilterChain），该链决定了每个请求的安全策略（如认证、授权、会话管理等）。

通过配置过滤器链，Shiro 可以拦截请求并进行权限验证、登录检查等操作。

#### 2、集成 Shiro 和 Spring

ShiroFilterFactoryBean 是 Shiro 与 Spring Web 集成的桥梁，它将 Shiro 的过滤器与 Spring 的 Filter 容器结合，使得 Spring 应用能够使用 Shiro 的安全功能。

它通过在 web.xml 或 Spring 配置类中配置过滤器，确保 Shiro 过滤器能够处理 Web 请求。

#### 3、简化 Web 安全配置

ShiroFilterFactoryBean 提供了一种简洁的方式来配置 Shiro 的 Web 安全设置，例如配置认证、授权、会话管理和访问控制等策略。

#### 4、定义访问控制规则

通过 ShiroFilterFactoryBean，开发者可以轻松定义哪些路径需要进行权限认证，哪些路径需要进行角色授权，甚至设置哪些路径需要匿名访问。

### 方法

#### public void setSecurityManager(SecurityManager securityManager)

设置 Shiro 的 SecurityManager，这是 Shiro 的核心组件，负责处理用户认证、授权、会话管理等。

示例：



#### public void setLoginUrl(String loginUrl)

设置用户未认证时的登录页面 URL。通常在用户访问受保护资源时，如果未登录，Shiro 会将用户重定向到该 URL。

示例：



#### public void setSuccessUrl(String successUrl)

设置认证成功后的默认跳转 URL。当用户成功登录后，Shiro 会将其重定向到该 URL。

示例：



#### public void setUnauthorizedUrl(String unauthorizedUrl)

设置用户未授权时的跳转 URL。当用户没有足够的权限访问某个资源时，Shiro 会将其重定向到该 URL。

示例：



#### public void setFilterChainDefinitions(String definitions)

设置 Shiro 的过滤器链定义，它是一个字符串格式的 URL 路径规则映射，定义了哪些路径需要哪些安全控制。



#### public void setFilters(Map<String, Filter> filters)

为 Shiro 配置自定义的过滤器。



### 配置示例



### 工作流程

#### 1、接收请求

当 Web 请求到达时，ShiroFilterFactoryBean 会通过配置的过滤器链来检查该请求是否需要认证、授权或访问控制。

#### 2、认证检查

如果请求需要认证（例如路径匹配到 authc 过滤器），Shiro 会检查用户是否已登录。如果未登录，Shiro 会将请求重定向到登录页面。

#### 3、授权检查

如果请求需要授权（例如路径匹配到 roles[admin]），Shiro 会检查用户是否具有所需的角色。如果没有角色，Shiro 会将用户重定向到未授权页面。

#### 4、自定义过滤器

通过 setFilters()，开发者可以定义自定义的过滤器，以便更细粒度地控制请求的处理方式。

## DefaultAdvisorAutoProxyCreator

# 注解

## @RequiresPermissions

### 基本概念

@RequiresPermissions 注解是用于授权的注解，它用于标记一个方法或类，以便在方法执行之前检查当前用户是否具有特定的权限。如果用户没有所需的权限，Shiro 将抛出 AuthorizationException，从而阻止方法的执行。这是 Shiro 提供的声明式权限控制的一种方式。

### 作用

#### 1、权限控制

（1）@RequiresPermissions 注解用于控制方法或类的访问权限，确保只有具有特定权限的用户才能访问被注解的方法或类。

（2）它通过声明式的方式，简化了权限校验的代码，开发者不需要手动编写权限检查逻辑。

#### 2、提高可维护性

使用注解方式进行权限控制，使得权限信息更加清晰、易于管理。只需在方法或类上添加注解，无需修改业务代码。

#### 3、简化权限校验

该注解自动与 Shiro 的权限校验机制结合，Shiro 会在执行目标方法前自动进行权限验证，减少手动权限校验的繁琐。

### 属性

#### value

指定方法或类所要求的权限。通过 value 属性，开发者可以指定一个或多个权限标识，用逗号分隔的字符串数组表示。

示例



#### logical

设置多个权限标识时，指定权限的组合方式，默认是 Logical.AND（即要求用户同时拥有所有指定的权限），也可以设置为 Logical.OR（即只要求用户拥有其中一个权限即可）

示例：



### 使用场景

#### 1、方法级权限控制

最常见的使用场景是在方法上直接使用 @RequiresPermissions 注解，控制该方法的访问权限。只有当前用户拥有指定权限时，才能调用该方法。

示例：



#### 2、类级权限控制

@RequiresPermissions 也可以用于类级别，表示该类中的所有方法都需要满足某些权限要求。Shiro 会在访问该类的任何方法之前先检查权限。

示例：



### 工作流程

#### 1、注解解析

当 Shiro 执行目标方法时，它会检查该方法或类是否有 @RequiresPermissions 注解，并解析该注解的 value 属性，获取所需的权限列表。

#### 2、权限验证

Shiro 会调用 Subject.isPermitted() 方法检查当前用户是否具有指定权限。

如果当前用户具有所有权限（在 logical 为 AND 时）或任意一个权限（在 logical 为 OR 时），方法执行继续；否则，抛出 AuthorizationException。

#### 3、权限检查失败

如果用户没有所需的权限，Shiro 会抛出 AuthorizationException，阻止方法的执行。

## @RequiresRoles

### 基本概念

@RequiresRoles 注解用于对方法或类进行角色权限控制。它是 Shiro 提供的一个注解，目的是在方法调用之前检查当前用户是否具备特定的角色，如果用户不具备所需的角色，Shiro 会阻止该方法的执行，并抛出 AuthorizationException 异常。

### 作用

指定访问某个方法或类所需要的角色。它用于方法级别或类级别的角色检查。当一个方法被 @RequiresRoles 注解标注时，Shiro 会在方法调用前检查当前用户是否具有注解中指定的角色。如果当前用户缺少任一角色，Shiro 会阻止方法的执行，并抛出 AuthorizationException 异常。

### 使用场景

#### 1、方法级别授权

@RequiresRoles 常用于方法上，用来限制只有具备某个角色的用户才能访问该方法。例如，只有管理员角色的用户才能执行一些管理操作。

#### 2、类级别授权

可以用在类上，表示该类中的所有方法都需要特定的角色才能访问。

### 语法



value：指定需要检查的角色，可以是一个或多个角色。

logical：指定多个角色时的逻辑关系。默认是 Logical.AND，表示必须同时具备所有角色。如果设置为 Logical.OR，则表示只需要具备其中一个角色即可。

### 工作原理

#### 1、用户认证

Shiro 会通过 SecurityManager 和 Realm 完成用户的认证（即确认用户的身份）。

#### 2、权限授权

当方法上加上 @RequiresRoles 注解时，Shiro 会在方法执行之前检查当前用户是否具备相应的角色。这是通过 Subject 对象的 hasRole 方法实现的。

#### 3、角色检查

（1）如果角色检查通过，即用户拥有所需的角色，Shiro 允许方法继续执行。

（2）如果角色检查未通过，即用户没有所需角色，Shiro 会抛出 AuthorizationException，并阻止方法执行。

### 示例代码

#### 1、方法级角色检查



在这个例子中，只有具备 admin 角色的用户才可以调用 performAdminAction 方法。如果用户没有 admin 角色，Shiro 会抛出 AuthorizationException，禁止访问。

#### 2、多个角色的 AND 关系



只有同时具备 admin 和 manager 角色的用户才能访问该方法。两个角色之间的关系是 **AND**（即必须同时拥有）。

#### 3、多个角色的 OR 关系



用户只需要具备 admin 或 manager 其中一个角色，即可以访问该方法。角色之间的关系是 **OR**（即拥有任意一个角色即可）。

### 注意事项

#### 1、角色的配置

角色的判断是通过 Realm 中定义的权限信息来完成的。Shiro 会从 SecurityManager 配置的 Realm 中获取角色信息，并调用 Subject 的 hasRole 方法进行角色判断。

#### 2、抛出的异常

如果用户没有所需的角色，Shiro 会抛出 AuthorizationException，具体表现为 UnauthorizedException。

#### 3、AOP 需要启用

@RequiresRoles 注解依赖于 Shiro 的 AOP 功能，所以需要确保 Shiro AOP 被正确启用。一般来说，需要在 Spring 或 Spring Boot 配置中加入 Shiro AOP 相关配置，使得授权注解能够被正确处理。

@Configuration

public class ShiroConfig {

@Bean

public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager) {

AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();

advisor.setSecurityManager(securityManager);

return advisor;

}

}

#### 4、角色的定义

在 Shiro 中，角色通常是通过 Realm 来进行定义的，Realm 会提供一个方法来获取用户的角色列表。在进行权限检查时，Shiro 会根据 Realm 中的角色信息判断用户是否具备该角色。