原创 ITKaven 于 2022-01-04 22:55:06 发布 阅读量1.5k 收藏 4 点赞数 3

分类专栏： Spring Security 文章标签： spring java redis

在上一篇博主中，博主介绍了 Spring Security 的 UserDetails 接口及其实现， Spring Security 使用 UserDetails 实例（实现类的实例）表示用户，行验证时（提供用户名和密码）， Spring Security 会通过用户服务（ UserDetailsService 接口及其实现）来获取对应的 UserDetails 实例（相同的用
如果该 UserDetails 实例存在并且与客户端输入的信息匹配，则验证成功，否则验证失败，想了解 UserDetails 接口及其实现可以看下面这篇博客：

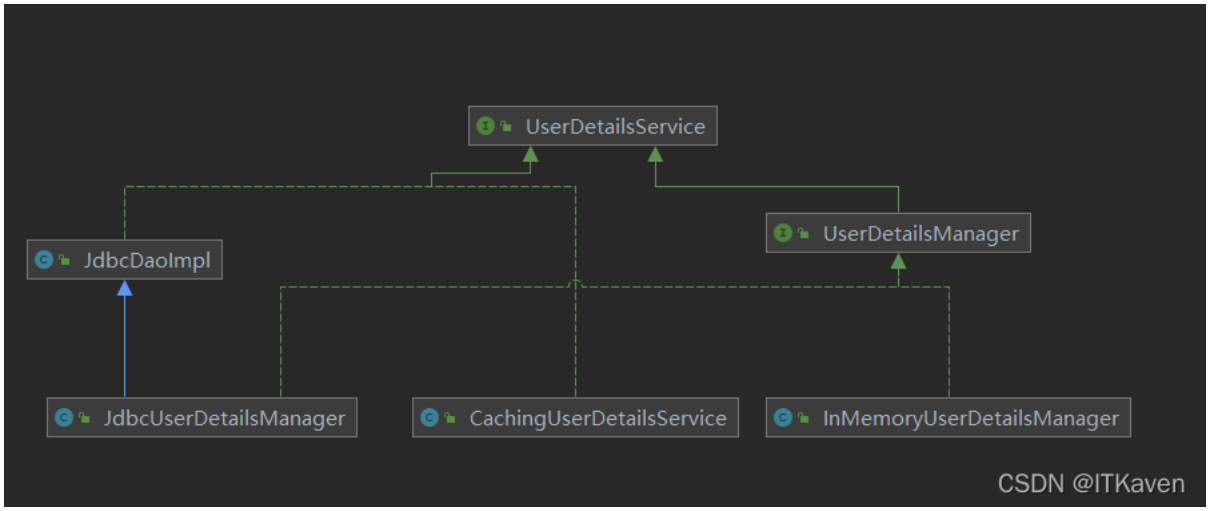
- Spring Security：用户UserDetails源码与Debug分析

UserDetailsService

UserDetailsService 接口源码：

```
package org.springframework.security.core.userdetails;

/**
 * 加载用户特定数据的核心接口
 * 它在整个框架中作为用户的DAO层（用户数据访问层）
 */
public interface UserDetailsService {
    /**
     * 根据用户名定位用户
     */
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}
```

UserDetailsService 接口只定义了一个方法，即通过用户名查找 UserDetails 实例的方法，因此子类可以有各种各样的实现，可以基于 JVM 的堆内存 ConcurrentHashMap 存储 UserDetails 实例），或者基于中间件（比如 Mysql 、 Redis ），或者两者的混合模式，可以根据需求来自定义实现。
UserDetailsService 接口的继承与实现关系如下图所示：



UserDetailsManager

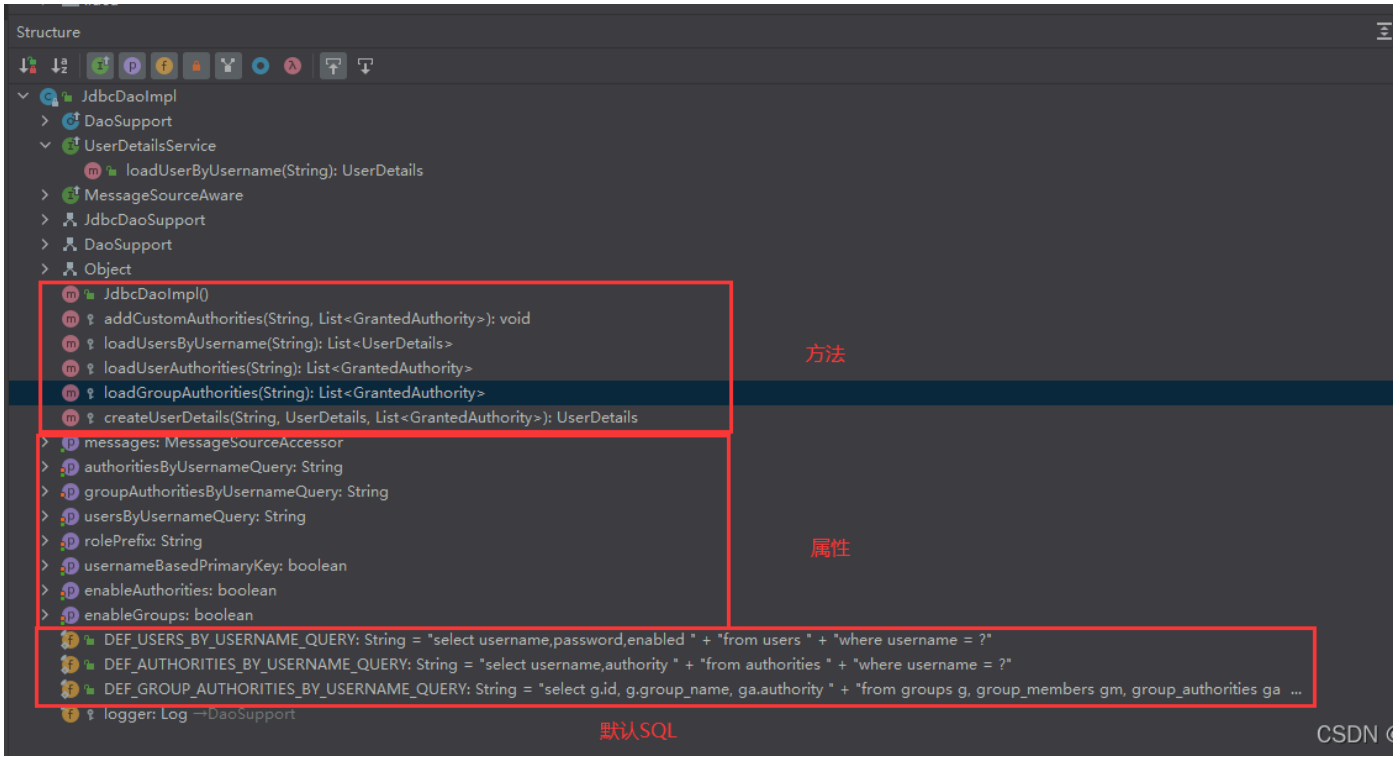
UserDetailsManager 接口源码（继承 UserDetailsService 接口）：

```
package org.springframework.security.provisioning;

import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;

/**
 * UserDetailsService的扩展，提供创建新用户和更新现有用户的能力
 */
public interface UserDetailsManager extends UserDetailsService {

```

很显然 `UserDetailsManager` 接口是 `UserDetailsService` 接口的扩展，提供了创建新用户和更新现有用户的能力。

JdbcDaoImpl

`JdbcDaoImpl` 类的结构如下图所示：



使用 `JDBC` 从数据库中检索用户详细信息（用户名、密码、启用标志和权限）。假设有一个默认的数据库模式（有 `users` 和 `authorities` 两个表）。

```
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(500) not null,
    enabled boolean not null
);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fk_authorities_users foreign key(username) references users(username)
);
create unique index ix_auth_username on authorities (username,authority);
```

如果数据库模式和默认不一样，可以设置 `usersByUsernameQuery` 和 `authorityByUsernameQuery` 属性以匹配数据库设置，不然它们的值都是默认 `SQL` 。

```
    public JdbcDaoImpl() {
        this.usersByUsernameQuery = DEF_USERS_BY_USERNAME_QUERY;
        this.authoritiesByUsernameQuery = DEF_AUTHORITIES_BY_USERNAME_QUERY;
        this.groupAuthoritiesByUsernameQuery = DEF_GROUP_AUTHORITIES_BY_USERNAME_QUERY;
    }
```

可以通过将 `enableGroups` 属性设置为 `true` 来启用对组权限的支持（还可以将 `enableAuthorities` 设置为 `false` 以直接禁用权限加载）。 通过这种方法
分配给组，并且用户的权限是根据他们所属的组来确定的。 最终结果是相同的（加载了包含一组 `GrantedAuthority` 的 `UserDetails` 实例）。使用组时
`groups` 、 `group_members` 和 `group_authorities` 。

```
create table groups (
    id bigint generated by default as identity(start with 0) primary key,
    group_name varchar_ignorecase(50) not null
);

```

```
    authority varchar(50) not null,
    constraint fk_group_authorities_group foreign key(group_id) references groups(id)
);
```

关于加载组权限的默认查询，可以参考 DEF_GROUP_AUTHORITIES_BY_USERNAME_QUERY 。 同样，可以通过设置 groupAuthoritiesByUsernameQuery 属性来

JdbcDaoImpl 类源码（实现了 UserDetailsService 接口，提供了使用 JDBC 获取用户数据的基本实现，下面代码删除了上面已经提到的内容）：

```
public class JdbcDaoImpl extends JdbcDaoSupport
        implements UserDetailsService, MessageSourceAware {

   /**
     * 允许子类将他们自己授予的权限添加到UserDetail实例的权限列表中
     */
    protected void addCustomAuthorities(String username,
            List<GrantedAuthority> authorities) {
    }

    // 通过用户名加载用户的核心逻辑，通过使用其他方法来完成
```

JdbcDaoImpl 类提供了使用 JDBC 获取用户数据的基本实现。

JdbcUserDetailsManager

JdbcUserDetailsManager 类继承了 JdbcDaoImpl 类（提供了使用 JDBC 获取用户数据的基本实现），并且实现了 UserDetailsManager 和 GroupManager 接口。 UserDetailsManager 接口提供了创建新用户和更新现有用户的能力。 GroupManager 接口允许管理组权限及其成员，通常用于在以下情况下补充 UserDetailsManager 的功能：

- 将应用程序授予的权限组织到组中，而不是直接将用户与角色进行映射。
- 在这种情况下，用户被分配到组并获得分配组的权限列表，从而提供更灵活的管理选项。

新增的默认 SQL （ UserDetailsManager SQL 和 GroupManager SQL ）：

```
    // UserDetailsManager SQL
    public static final String DEF_CREATE_USER_SQL = "insert into users (username, password, enabled) values (?,?,?)";
    public static final String DEF_DELETE_USER_SQL = "delete from users where username = ?";
    public static final String DEF_UPDATE_USER_SQL = "update users set password = ?, enabled = ? where username = ?";
    public static final String DEF_INSERT_AUTHORITY_SQL = "insert into authorities (username, authority) values (?,?)";
    public static final String DEF_DELETE_USER_AUTHORITIES_SQL = "delete from authorities where username = ?";
    public static final String DEF_USER_EXISTS_SQL = "select username from users where username = ?";
    public static final String DEF_CHANGE_PASSWORD_SQL = "update users set password = ? where username = ?";

    // GroupManager SQL
    public static final String DEF_FIND_GROUPS_SQL = "select group_name from groups";
```

源码就不贴了，太多了，实现方式和 JdbcDaoImpl 类差不多（通过使用默认 SQL ，也可以使用满足要求的自定义 SQL ，查询用户相关数据），需要使用

读源码（还是要多看源码）。

CachingUserDetailsService

CachingUserDetailsService 类源码（实现了 UserDetailsService 接口）：

```
public class CachingUserDetailsService implements UserDetailsService {
    // 用户缓存，NullUserCache不执行任何缓存
```

```
    private final UserDetailsService delegate;

    public CachingUserDetailsService(UserDetailsService delegate) {
        this.delegate = delegate;
    }

    public UserCache getUserCache() {
```

`NullUserCache` 类，不执行任何缓存。

```
public class NullUserCache implements UserCache {

    public UserDetails getUserFromCache(String username) {
        return null;
    }

    public void putUserInCache(UserDetails user) {
    }

    public void removeUserFromCache(String username) {
    }
}
```

`CachingUserDetailsService` 实例可以通过设置不同的用户缓存（以后介绍）实例来达到不同的缓存效果。

```
    // 设置用户缓存实例
    public void setUserCache(UserCache userCache) {
        this.userCache = userCache;
    }
```

InMemoryUserDetailsManager

`InMemoryUserDetailsManager` 通过 `HashMap` 存储用户数据，是 `UserDetailsManager` 的一种非持久化实现。主要用于测试和演示目的，不需要完整的持

`InMemoryUserDetailsManager` 类源码（实现了 `UserDetailsManager` 和 `UserDetailsPasswordService` 接口， `UserDetailsPasswordService` 接口定义了
`UserDetails` 密码的方法）

```
public class InMemoryUserDetailsManager implements UserDetailsManager,
        UserDetailsPasswordService {
    protected final Log logger = LogFactory.getLog(getClass());

    // 存储用户数据的容器
    private final Map<String, MutableUserDetails> users = new HashMap<>();
    // 用于处理验证请求，以后会详细介绍
    private AuthenticationManager authenticationManager;

    // 无参构造器
    public InMemoryUserDetailsManager() {
```

自定义用户服务

自定义用户服务：

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // 配置自定义的用户服务

```

```
10  }
11
```

Spring Security 的用户服务 UserDetailsService 源码分析就到这里，如果博主有说错的地方或者大家有不同的见解，欢迎大家评论补充。

3 条评论

何壹时 热评 好文三连了！欢迎回访！回应互评五星！ 网页链接

spring-security-oauth2源码
Spring Security OAuth2允许开发者自定义各种组件，如TokenEnhancer（增强令牌）、UserDetailsService（用户详情服务）等，以满足特定需求。 6. **spring-security-o

Spring_Security3_源码分析
总结，Spring Security 3的源码分析是一个深度学习的过程，涵盖了安全领域的多个方面。通过理解其内部工作机制，开发者可以更好地利用这一强大的框架，为应用程序

详解Spring Security进阶身份认证之UserDetailsService(附源码)-附件资源
详解Spring Security进阶身份认证之UserDetailsService(附源码)-附件资源

【SpringSecurity】五、UserDetails接口和UserDetailsService接口（原理、流程） llg___的
接下来自己定义一个用户类SecurityUser类，也去实现UserDetails接口，重写UserDetails接口方法时，直接写死一个用户信息，一会儿new这个自定义的SecurityUser类，

Spring Security - UserDetails 崔
UserDetails 中几个字段的解释：//返回验证用户密码,无法返回则NULLString getPassword();String getUsername();//账户是否过期,过期无法验证boolean isAccountNonEx

Springsecurity之UserDetails weixin_33831196的
2019独角兽企业重金招聘Python工程师标准>>> ...

SpringSecurity (2) UserDetailsService 热门推荐 O
主要介绍如何利用 UserDetailsService 接口从数据库中获取用户信息, 并通过实现 AuthenticationProvider 接口编写自己的校验逻辑, 从而完成 SpringSecurity 身份校验.

SpringSecurity框架原理浅谈之UserDetailsService 黄先生的
UserDetailsService的作用就是从特定的地方（通常是数据库）加载用户信息.

Spring Security：用户UserDetails源码与Debug分析 ka
Spring Security身份验证与授权的对象是用户，这里说的用户可以是配置文件中定义的用户，也可以是数据库表中存储的用户，还可以是Spring Security自动创建的用户（

Spring Security多个UserDetailsService
Spring Security支持多个UserDetailsService。你可以通过在配置文件中定义多个UserDetailsService的实现类来实现这一点。在配置文件中，你可以使用<bean>标签为每

ITKaven
码龄7年 暂无认证

542	2万+	105万+	227万+	
原创	周排名	总排名	访问	等级

2万+	1368	1992	937	3844
积分	粉丝	获赞	评论	收藏

私信 关注

搜博主文章

分类专栏

最新评论

最新文章

目录