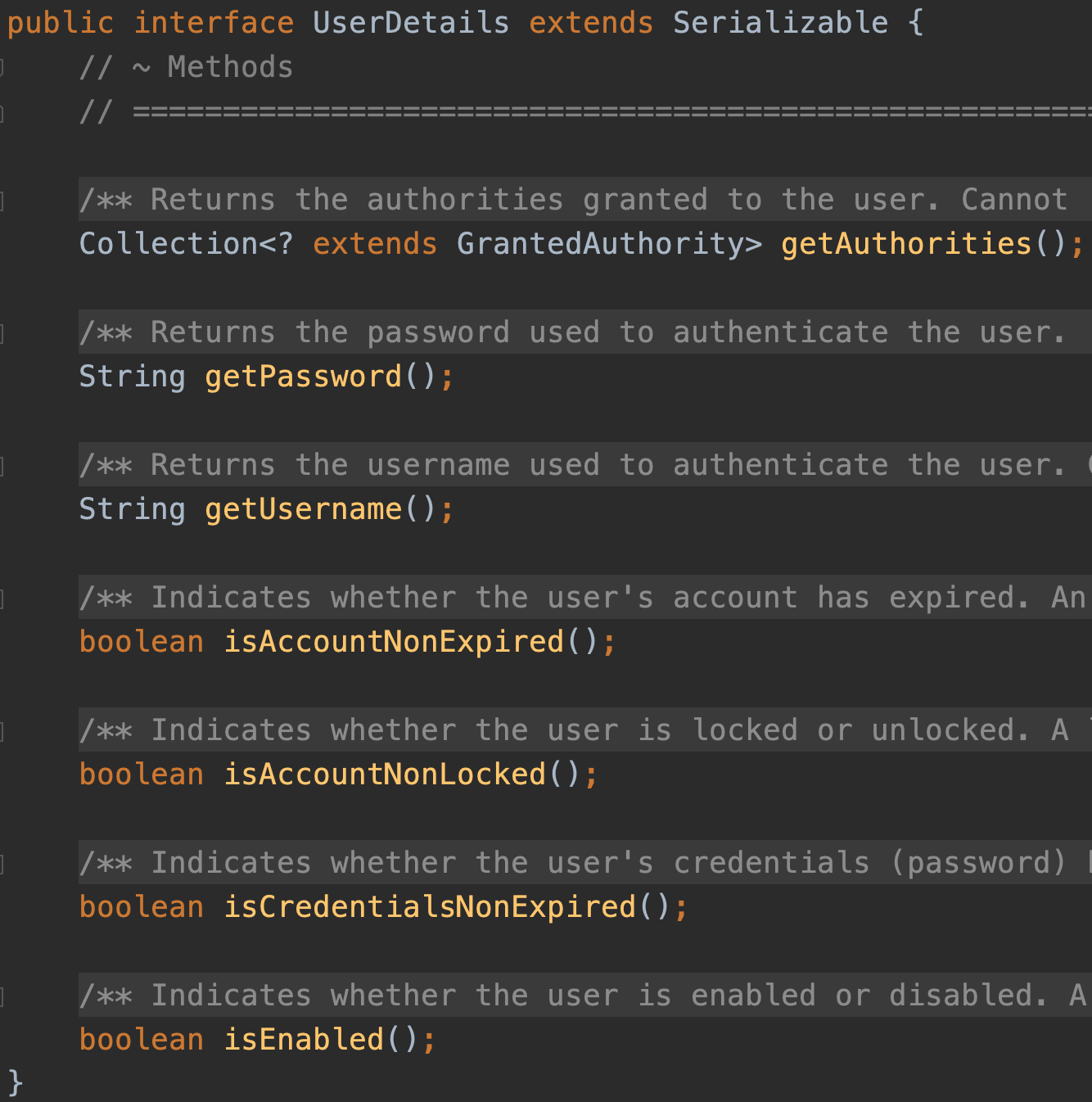


* UserDetails => Spring Security基础接口，包含某个用户的账号，密码，权限，状态（是否锁定）等信息。只有getter方法。
* Authentication => 认证对象，认证开始时创建，认证成功后存储于SecurityContext
* principal => 用户信息对象，是一个Object，通常可转为UserDetails。

#### UserDetails接口

用于表示一个principal，但是一般情况下是作为（你所使用的用户数据库）和（Spring Security的安全上下文需要保留的信息）之间的适配器。

实际上就是相当于定义一个规范，Security这个框架不管你的应用是怎么存储用户和权限信息的。只要你取出来的时候把它包装成一个UserDetails对象给我用就可以了。



#### UserDetails接口用来做什么？为什么还要带上权限集合？

如果不用认证框架，怎么手动实现登录认证？

基本上根据前端提交上来的用户名从数据库中查找这个账号的信息，然后比对密码。再进一步，可能还会添加一个字段来判断，当前用户是否被锁定。这个接口就是这么用的。即把这些信息取出来，然后包装成一个对象交由框架去认证。

为什么还要带上权限？

因为登录成功后也不是什么都能访问的，还要根据你所拥有的权限进行判断。有权限你才能访问特定的对象。Security框架是这样设定的，即认证成功后，就把用户信息和拥有的权限都存储在SecurityContext中，当访问受保护资源（某个对象/方法）的时候，就把权限拿出来比对，看看是否满足。

#### 框架提供的UserDetails默认实现方式

UserDetails有一个默认实现（框架提供的），User。用户可以从自己的数据库中取出此用户的账号，密码，以及相关权限，然后用构造方法填充创建一个User对象即可。

注：实现CredentialsContainer接口是为了在登录成功后，清除用户信息中的密码。（登录成功后会将用户信息存储在SecurityContext中）

public class User implements UserDetails, CredentialsContainer {

private static final long serialVersionUID = 500L;

private static final Log logger = LogFactory.getLog(User.class);

private String password;

private final String username;

private final Set<GrantedAuthority> authorities;

private final boolean accountNonExpired;

private final boolean accountNonLocked;

private final boolean credentialsNonExpired;

private final boolean enabled;

public User(String username, String password, Collection<? extends GrantedAuthority> authorities) {

this(username, password, true, true, true, true, authorities);

}

public User(String username, String password, boolean enabled, boolean accountNonExpired, boolean credentialsNonExpired, boolean accountNonLocked, Collection<? extends GrantedAuthority> authorities) {

if (username != null && !"".equals(username) && password != null) {

this.username = username;

this.password = password;

this.enabled = enabled;

this.accountNonExpired = accountNonExpired;

this.credentialsNonExpired = credentialsNonExpired;

this.accountNonLocked = accountNonLocked;

this.authorities = Collections.unmodifiableSet(sortAuthorities(authorities));

} else {

throw new IllegalArgumentException("Cannot pass null or empty values to constructor");

}

}

//省略部分代码

}

#### 什么时候提供UserDetails信息，怎么提供？

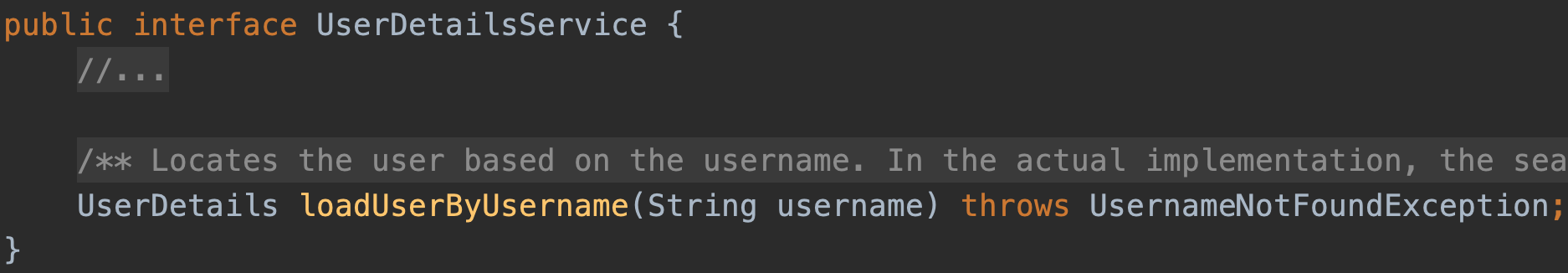
**1）UserDetailsService接口**

认证的时候。其实认证的操作，框架都已经帮你实现了，它所需要的只是，你给我提供获取信息的方式。所以它就定义一个接口，然后让你去实现，实现好了之后再注入给它。

框架提供一个UserDetailsService接口用来加载用户信息。

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

如果要自定义实现的话，用户可以实现一个CustomUserDetailsService的类，然后把你的应用中的UserService和AuthorityService注入到这个类中，用户获取用户信息和权限信息，然后在loadUserByUsername方法中，构造一个User对象（框架的类）返回即可。



**2）框架提供的UserDetailsService接口默认实现**

* InMemoryUserDetailsManager => 存储于内存
* JdbcDaoImpl => 存储于数据库（磁盘）

其中，如果你的数据库符合JdbcDaoImpl中的规范，你也就不用自己去实现UserDetailsService了。但是大多数情况是不符合的，因为你用户表不一定就叫users，可能还有其他前缀什么的，比如叫tb\_users。或者字段名也跟它不一样。如果你一定要使用这个JdbcDaoImpl，你可以通过它的setter方法修改它的数据库查询语句。（注：它是利用Spring框架的JdbcTemplate来查询数据库的）

#### 注入到哪里去呢？

注入到认证处理类中的，框架利用AuthenticationManager（接口）来进行认证。而Security为了支持多种方式认证,它提供ProviderManager类,这个实现了AuthenticationManager接口。它拥有多种认证方式，可以根据认证的类型委托给对应的认证处理类进行处理，这个处理类实现了AuthenticationProvider的接口。

所以，最终UserDetailsService是注入到AuthenticationProvider的实现类中。

注意：UserDetailsService只单纯地负责存取用户信息，除了给框架内的其他组件提供数据外没有其他功能。而认证过程是由AuthenticationManager来完成的。