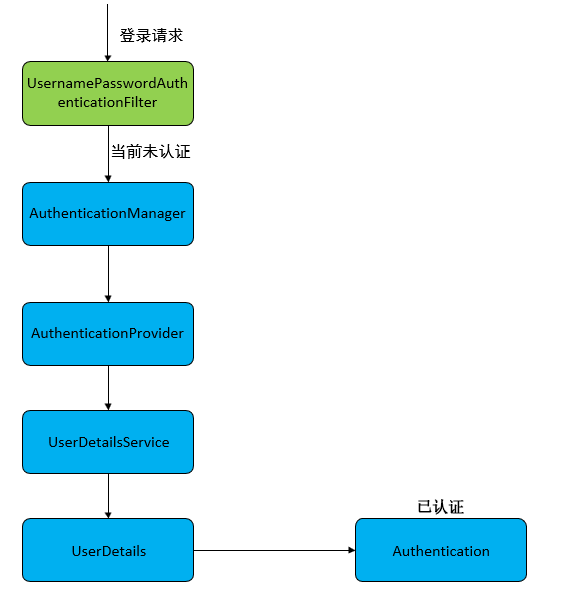
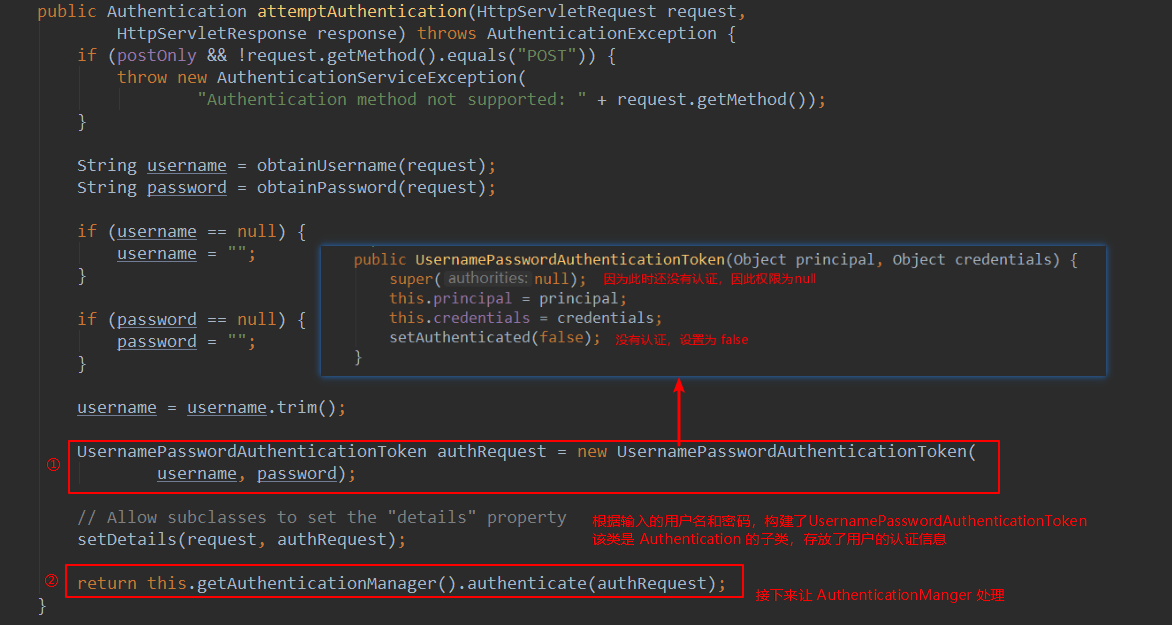
1. **认证流程**

https://blog.csdn.net/yuanlaijike/article/details/84703690



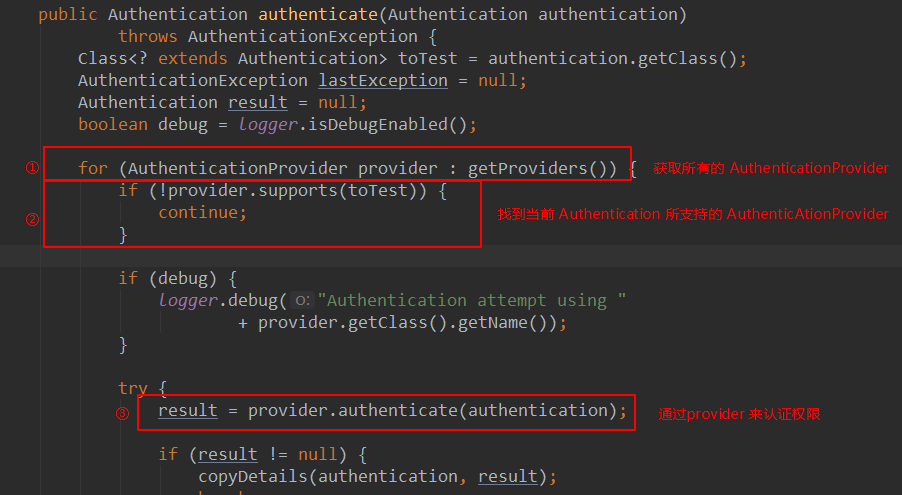
1. 用户发起表单登录请求后，首先进入 UsernamePasswordAuthenticationFilter：



在 UsernamePasswordAuthenticationFilter 中根据用户输入的用户名、密码构建了 UsernamePasswordAuthenticationToken，并将其交给 AuthenticationManager 来进行认证处理。

AuthenticationManager 本身不包含认证逻辑，其核心是用来管理所有的 AuthenticationProvider，通过交由合适的 AuthenticationProvider 来实现认证。

1. 下面跳转到了 ProviderManager ，该类是 AuthenticationManager 的实现类：



我们知道不同的登录逻辑它的认证方式是不一样的，比如我们表单登录需要认证用户名和密码，但是当我们使用三方登录时就不需要验证密码。

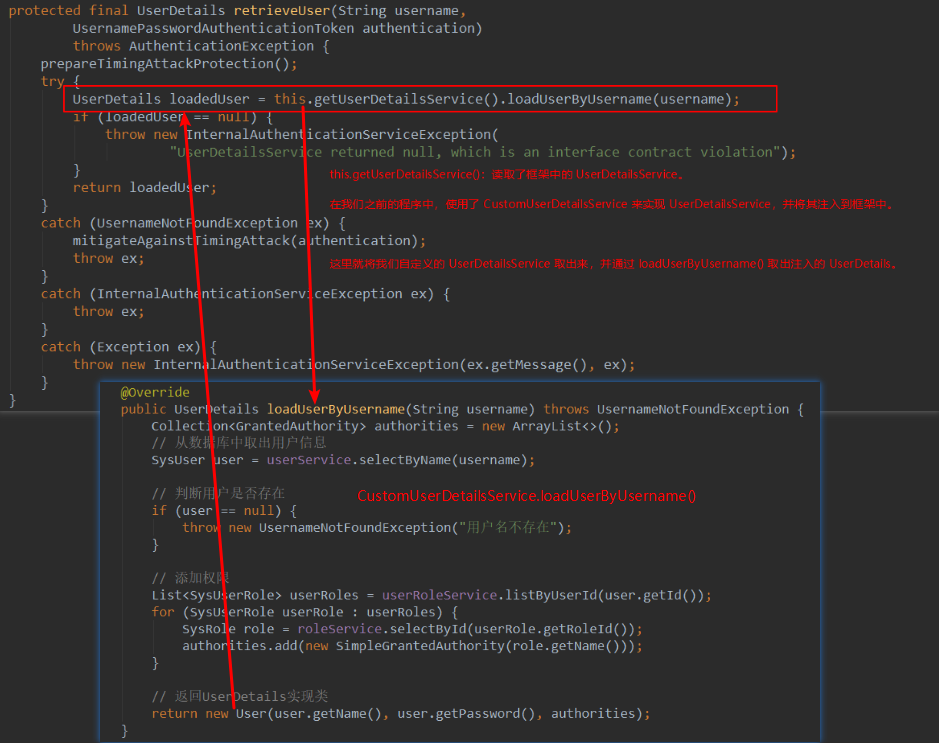
Spring Security 支持多种认证逻辑，每一种认证逻辑的认证方式其实就是一种 AuthenticationProvider。通过 getProviders() 方法就能获取所有的 AuthenticationProvider，通过 provider.supports() 来判断 provider 是否支持当前的认证逻辑。

当选择好一个合适的 AuthenticationProvider 后，通过 provider.authenticate(authentication) 来让 AuthenticationProvider 进行认证。

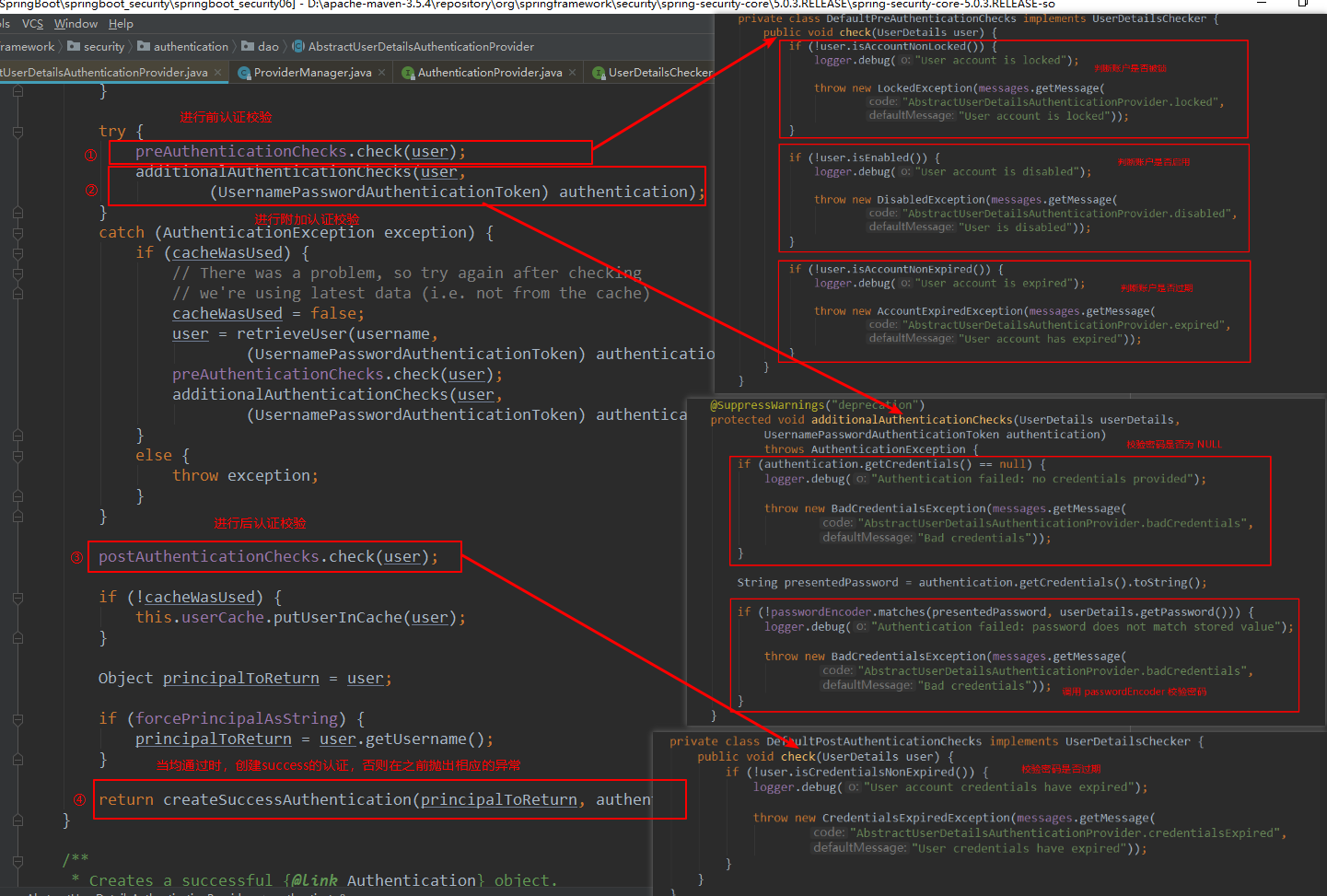
1. 传统表单登录AuthenticationProvider 主要是由 AbstractUserDetailsAuthenticationProvider 来进行处理的，我们来看下它的 authenticate()方法。

首先通过 retrieveUser() 方法读取到数据库中的用户信息：

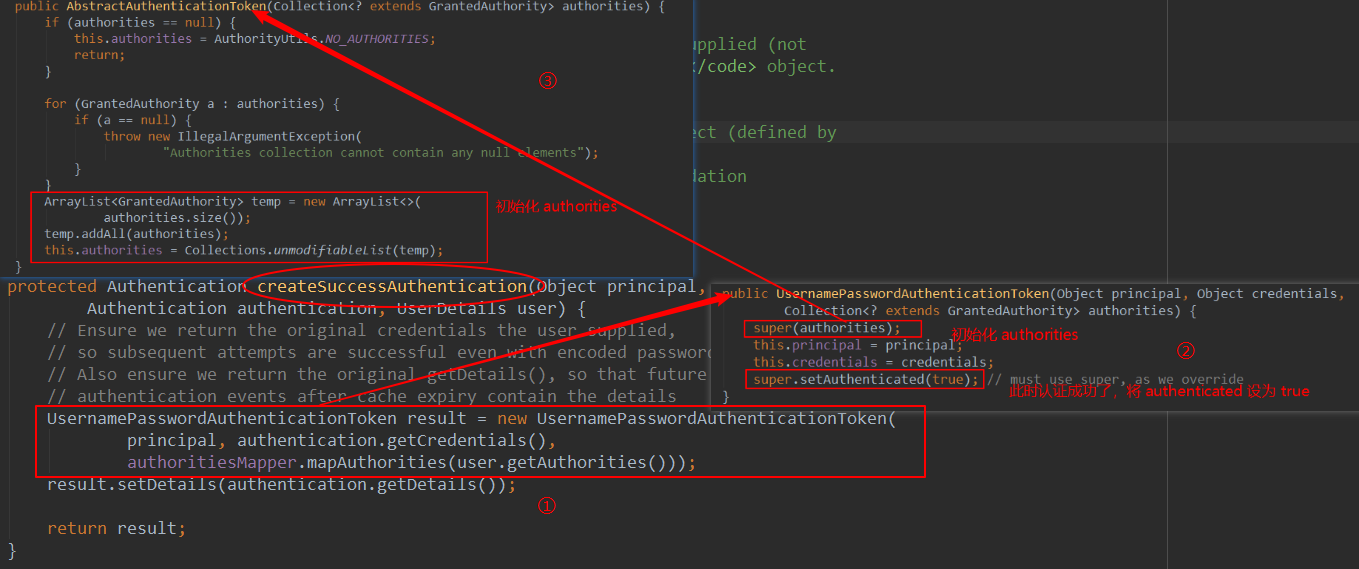
retrieveUser() 的具体实现在 DaoAuthenticationProvider 中，代码如下：



当我们成功的读取 UserDetails 后，下面开始对其进行认证：

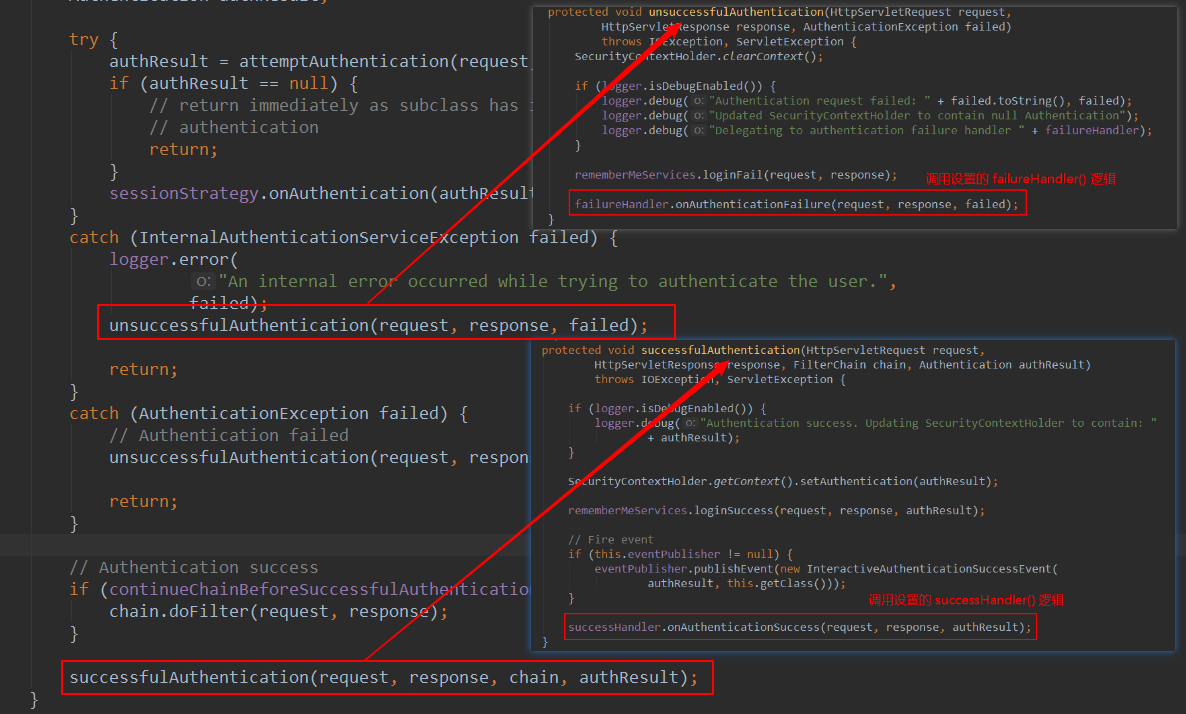


在上图中，我们可以看到认证校验分为 **前校验**、**附加校验**和**后校验**，如果任何一个校验出错，就会抛出相应的异常。所有校验都通过后，调用 createSuccessAuthentication() 返回认证信息。



在 createSuccessAuthentication 方法中，我们发现它重新 new 了一个 UsernamePasswordAuthenticationToken，因为到这里认证已经通过了，所以将 authorities 注入进去，并设置 authenticated 为 true，即需要认证。

（4）至此认证信息就被传递回 UsernamePasswordAuthenticationFilter 中，在 UsernamePasswordAuthenticationFilter 的父类 AbstractAuthenticationProcessingFilter 的 doFilter() 中，会根据认证的成功或者失败调用相应的 handler：

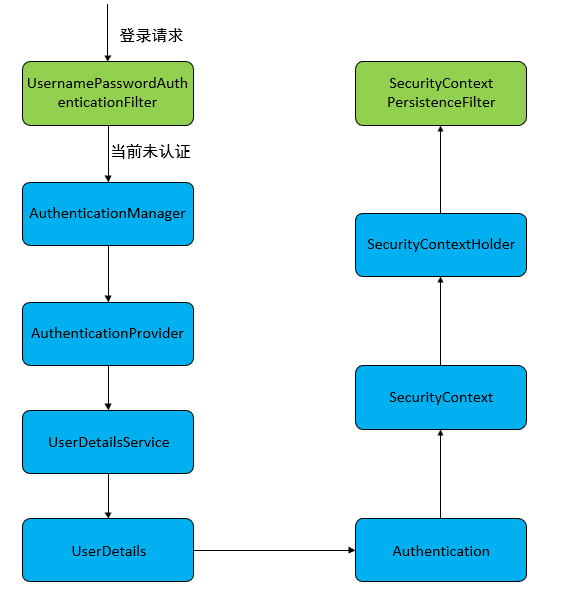


这里调用的 handler 实际就是在《SpringBoot集成Spring Security（6）——登录管理》中我们在配置文件中配置的 successHandler() 和 failureHandler()。

1. **多个请求共享认证信息**

Spring Security 通过 Session 来保存用户的认证信息，那么 Spring Security 到底是在什么时候将认证信息放入 Session，又在什么时候将认证信息从 Session 中取出来的呢？

下面将 Spring Security 的认证流程补充完整，如下图：



在上一节认证成功的 successfulAuthentication()方法中，有一行语句：

SecurityContextHolder.getContext().setAuthentication(authResult);

1

其实就是在这里将认证信息放入 Session 中。

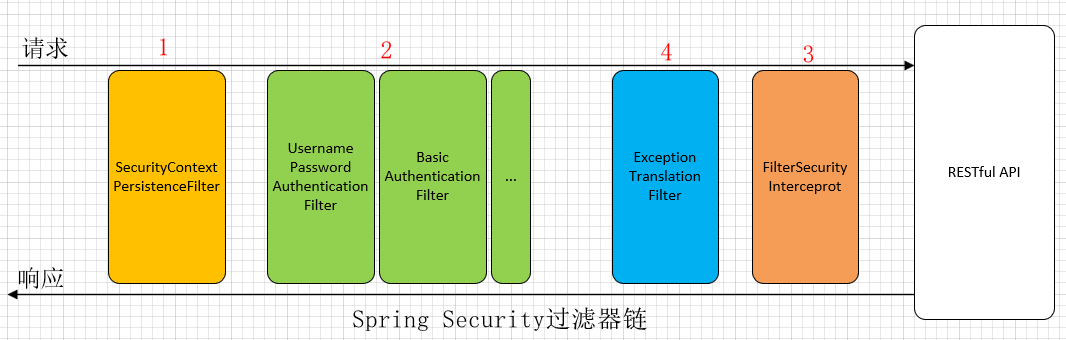
查看 SecurityContext 源码，发现内部就是对 Authentication 的封装，提供了 equals、hashcode、toString等方法，而SecurityContextHolder 可以理解为线程中的 ThreadLocal。

我们知道一个 HTTP 请求和响应都是在一个线程中执行，因此在整个处理的任何一个方法中都可以通过 SecurityContextHolder.getContext()来取得存放进去的认证信息。

从 Session 中对认证信息的处理由 SecurityContextPersistenceFilter 来处理，它位于 Spring Security 过滤器链的最前面，它的主要作用是：

当请求时，检查 Session 中是否存在 SecurityContext，如果有将其放入到线程中。

当响应时，检查线程中是否存在 SecurityContext，如果有将其放入到 Session 中。



三、获取用户认证信息

通过调用 SecurityContextHolder.getContext().getAuthentication() 就能够取得认证信息：

@GetMapping("/me")

@ResponseBody

public Object me() {

return SecurityContextHolder.getContext().getAuthentication();

}

上面的写法有点啰嗦，我们可以简写成下面这种， Spring MVC 会自动帮我们从 Spring Security 中注入：

@GetMapping("/me")

@ResponseBody

public Object me(Authentication authentication) {

return authentication;

}

如果你仅想获取 UserDetails 对象，也是可以的，写法如下：

@GetMapping("/me")

@ResponseBody

public Object me(@AuthenticationPrincipal UserDetails userDetails) {

return userDetails;

}