14 异常处理 & 日志

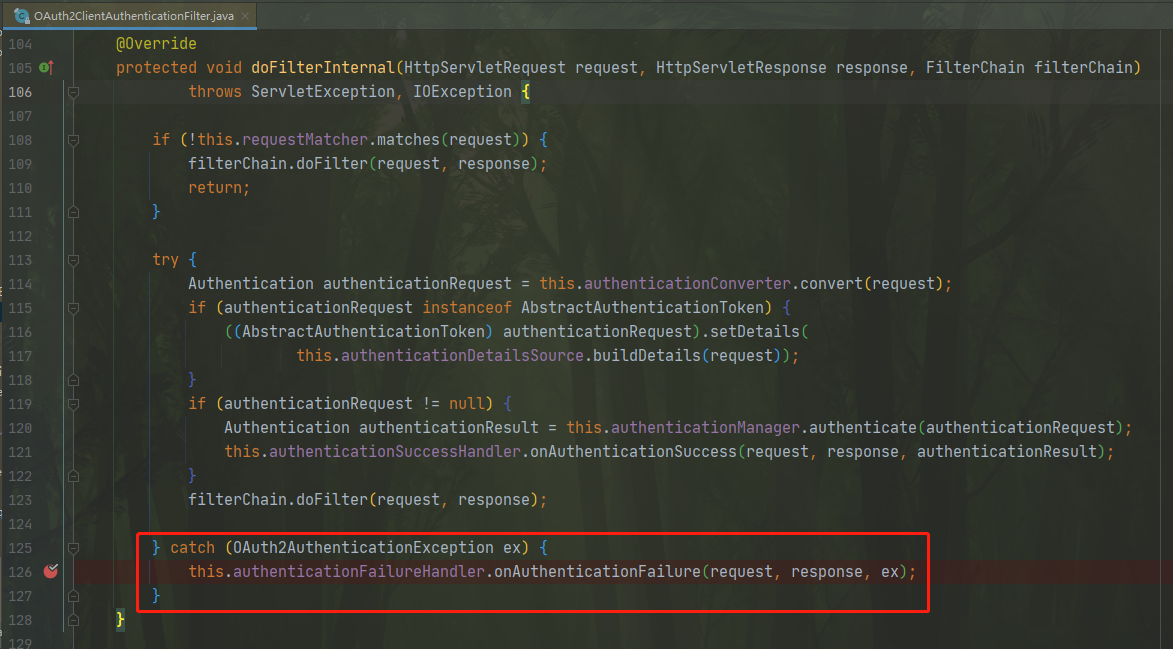
个人认为 Spring Authorization Server 的异常提示目前还不是很完善，很多信息都是简化的信息，很难直观看出根本原因，而且后台还没有打印异常日志（笔者已经提了个issue，寄希望于后续版本能通过日志级别控制异常日志的输出，便于我们开发时调试）。

下文便来先看看核心流程的异常处理。

本文使用 0.3.1 版本

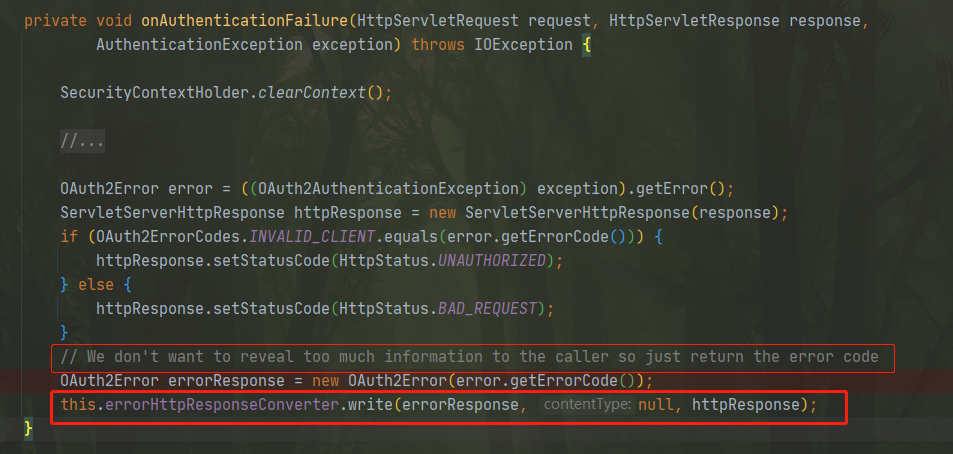
客户端认证流程的异常处理

我们知道，客户端认证的核心过滤器是 OAuth2ClientAuthenticationFilter，查看其处理逻辑。



异常处理类为 AuthenticationFailureHandler，实现类是方法引用 this::onAuthenticationFailure

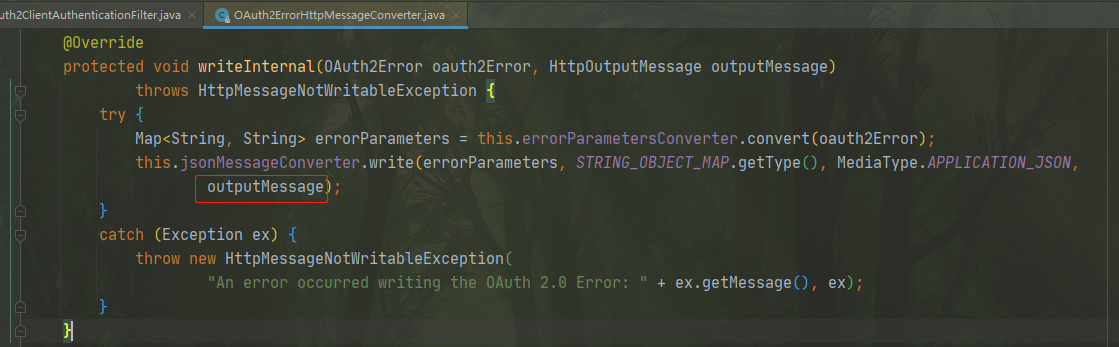
IMG_256



核心类是 OAuth2ErrorHttpMessageConverter

IMG_256

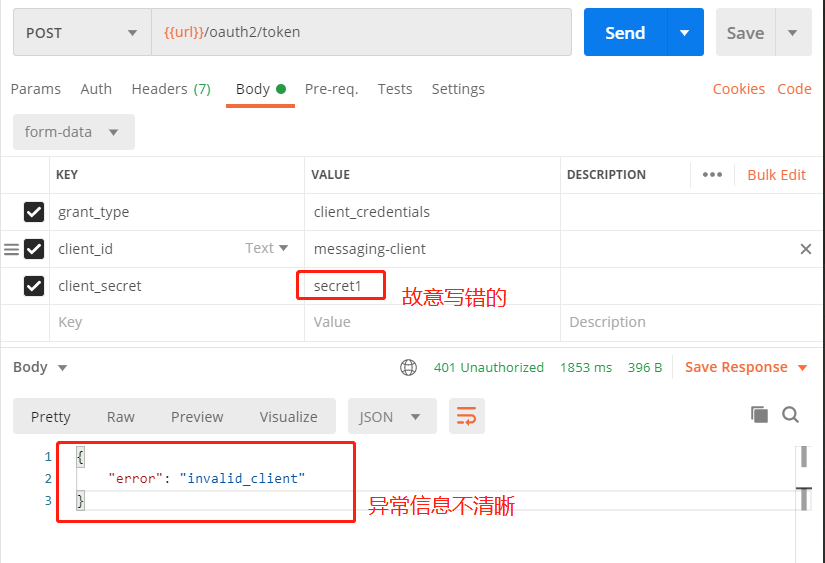
OAuth2ErrorHttpMessageConverter 的核心方法为 writeInternal(...)，可以看到，它直接将异常信息写入响应流outputMessage 中，返回给了前端。没有其他处理，后台也就看不了详细异常日志。

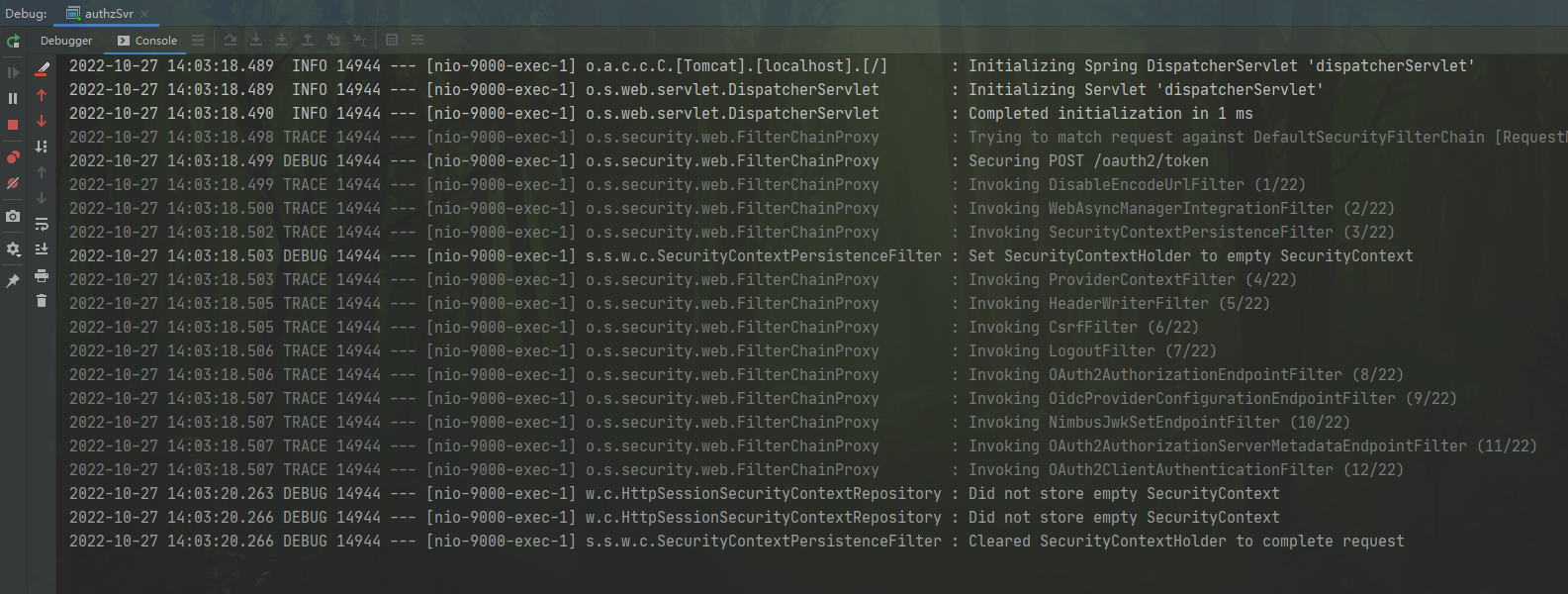


查看异常日志详情

默认情况

我们试下故意写错信息来发送请求，看看响应和日志，如下：



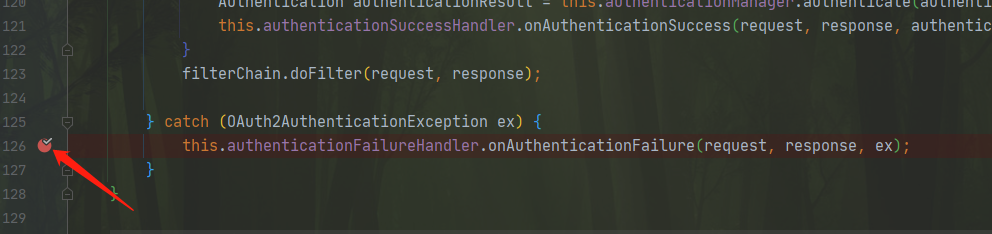


可以看到，异常响应不清晰，日志也没啥有用的内容。如果这不是我们故意写错，根本无法判断哪里出错了。

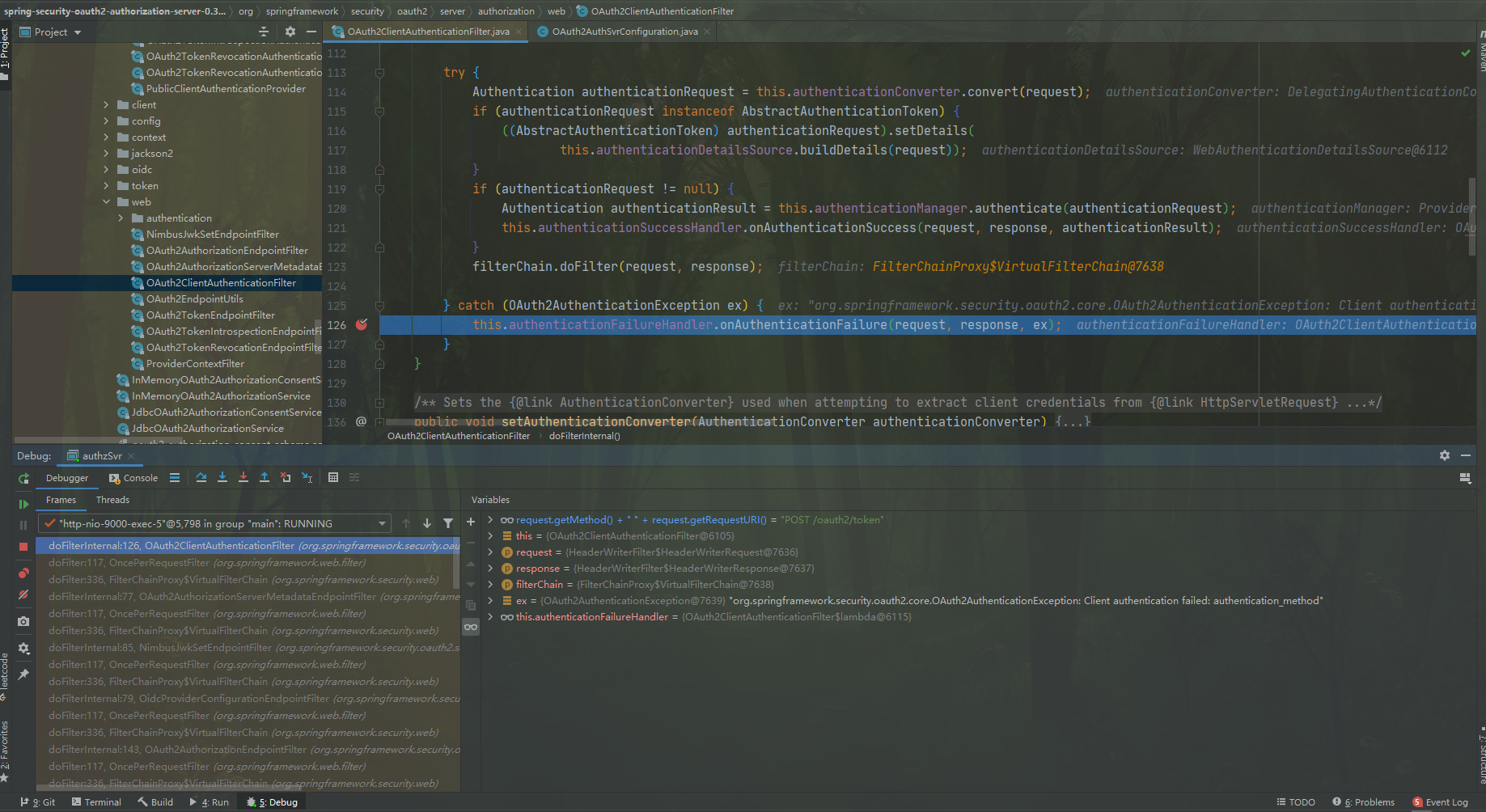
解决：通过 Debug 查看日志

介于这种情况，我们在开发时调试，可以通过 debug 来查看异常日志了。

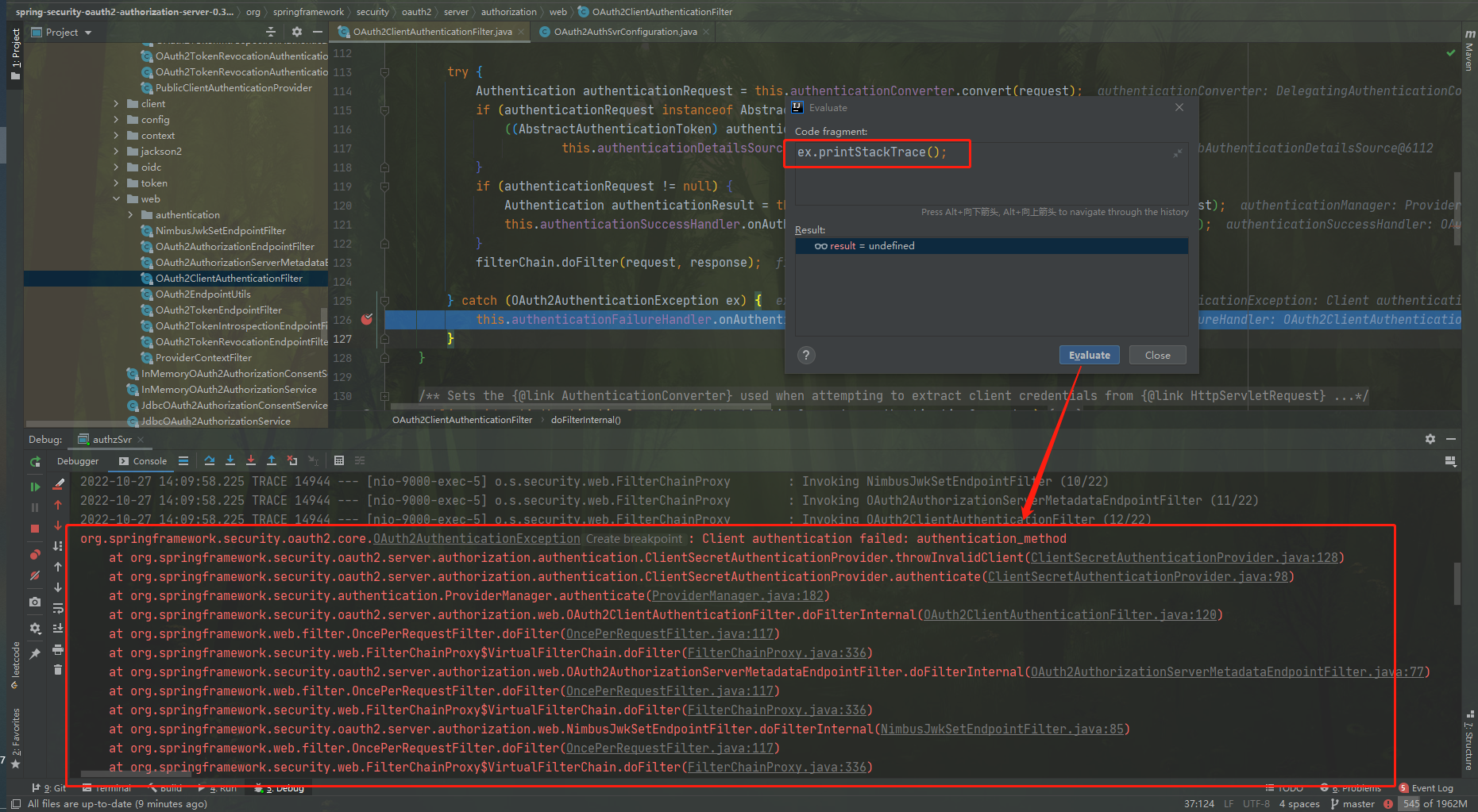
在 OAuth2ClientAuthenticationFilter 的 catch 逻辑块打上断点（以IDEA为例）



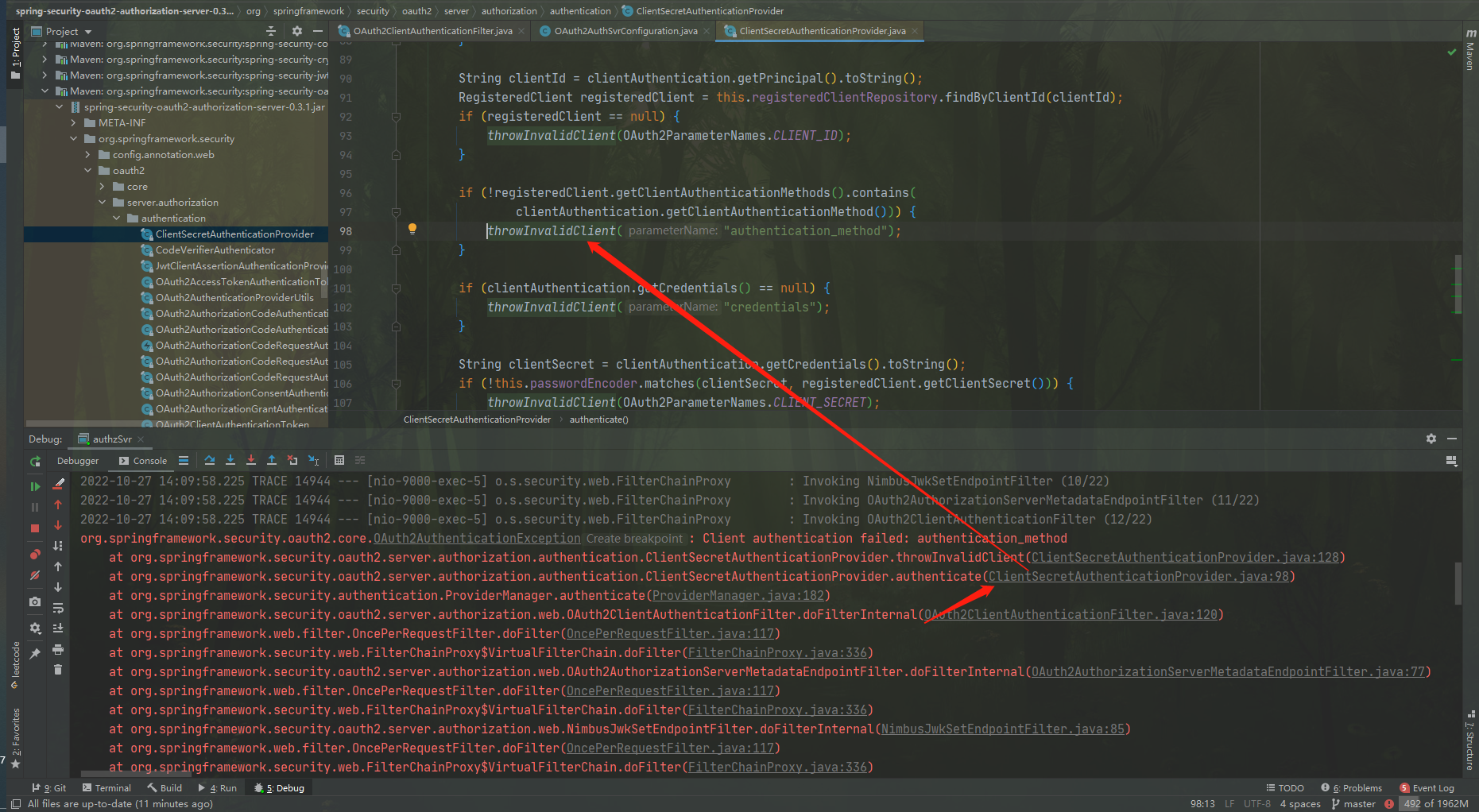
重新发送上述请求，可以看到，会在断点处暂停。



打开动态执行代码框（IDEA：顶部菜单 Run -> Debugging Actions -> Evaluate Expression…），输入代码 ex.printStackTrace();，点击 Evaluate 按钮执行代码。可以在控制台看到日志。



点击异常栈，可以直接跳转到对应的代码行，就能知道哪里出错了。



解决：重写 AuthenticationFailureHandler 查看日志

在分析流程时我们可以看到，发生异常时是由 AuthenticationFailureHandler 进行处理的。自然我们可以通过重写该实现来打印异常日志。

默认实现为：OAuth2ClientAuthenticationFilter#authenticationFailureHandler

参照原有异常处理逻辑，仅增加一行异常栈打印语句，新建一个实现类，代码如下：

public class ClientAuthenticationFailureHandler implements AuthenticationFailureHandler {

private final HttpMessageConverter<OAuth2Error> errorHttpResponseConverter = new OAuth2ErrorHttpMessageConverter();

@Override

public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, AuthenticationException exception) throws IOException, ServletException {

// 打印异常栈

exception.printStackTrace();

SecurityContextHolder.clearContext();

OAuth2Error error = ((OAuth2AuthenticationException) exception).getError();

ServletServerHttpResponse httpResponse = new ServletServerHttpResponse(response);

if (OAuth2ErrorCodes.INVALID\_CLIENT.equals(error.getErrorCode())) {

httpResponse.setStatusCode(HttpStatus.UNAUTHORIZED);

} else {

httpResponse.setStatusCode(HttpStatus.BAD\_REQUEST);

}

// We don't want to reveal too much information to the caller so just return the error code

OAuth2Error errorResponse = new OAuth2Error(error.getErrorCode());

this.errorHttpResponseConverter.write(errorResponse, null, httpResponse);

}

}

重写的类自然需要替换原有实现，在原有的配置代码 OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http); 之后，增加如下配置：

http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)

.clientAuthentication(new Customizer<OAuth2ClientAuthenticationConfigurer>() {

@Override

public void customize(OAuth2ClientAuthenticationConfigurer oAuth2ClientAuthenticationConfigurer) {

oAuth2ClientAuthenticationConfigurer.errorResponseHandler(new ClientAuthenticationFailureHandler());

}

});

搞定。

授权端点的异常处理

授权端点的核心过滤器是 OAuth2AuthorizationEndpointFilter。和上文同理，可利用相同的办法打印日志。

令牌颁发流程的异常处理

令牌颁发的核心过滤器是 OAuth2TokenEndpointFilter。和上文同理，可利用相同的办法打印日志。

另外

异常响应的 error 枚举值：org.springframework.security.oauth2.core.OAuth2ErrorCodes