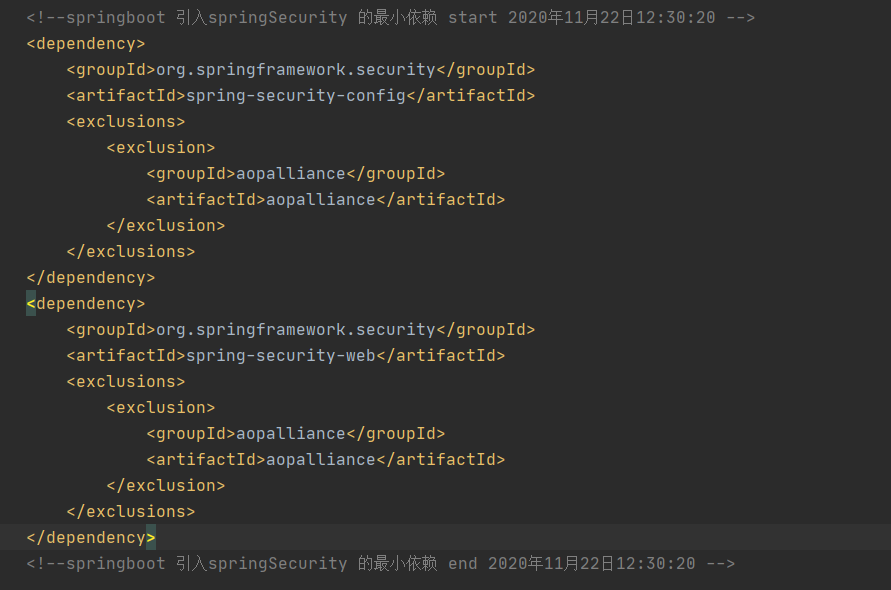
# Spring Boot 整合SpringSecurity

## Springboot 通过 依赖引入springSecurity的最小依赖



|  |
| --- |
| <!--springboot 引入springSecurity 的最小依赖 start 2020年11月22日12:30:20 -->  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-config</artifactId>  <exclusions>  <exclusion>  <groupId>aopalliance</groupId>  <artifactId>aopalliance</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-web</artifactId>  <exclusions>  <exclusion>  <groupId>aopalliance</groupId>  <artifactId>aopalliance</artifactId>  </exclusion>  </exclusions>  </dependency>  <!--springboot 引入springSecurity 的最小依赖 end 2020年11月22日12:30:20 --> |

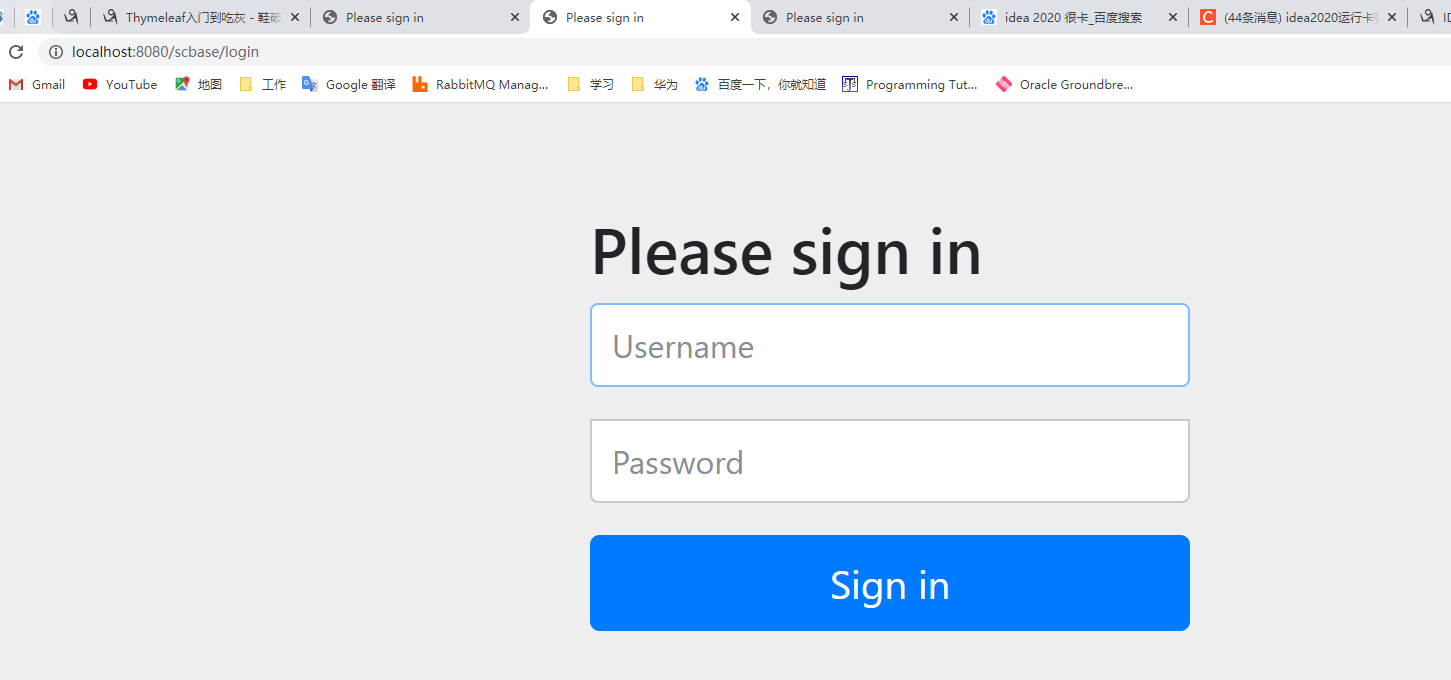
项目启动的相关日志如下：

|  |
| --- |
| 2020-11-22 12:50:46.447 INFO 245456 --- [ main] .s.s.UserDetailsServiceAutoConfiguration :  Using generated security password: c9d04d01-8464-4d8e-85b5-ebad19bc8a36  2020-11-22 12:50:46.578 INFO 245456 --- [ main] o.s.s.web.DefaultSecurityFilterChain : Creating filter chain: any request, [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@1235151c, org.springframework.security.web.context.SecurityContextPersistenceFilter@75b38c36, org.springframework.security.web.header.HeaderWriterFilter@4cc7d00d, org.springframework.security.web.csrf.CsrfFilter@115ca7de, org.springframework.security.web.authentication.logout.LogoutFilter@36790bec, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@7af443a3, org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@609319c3, org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@5a1f778, org.springframework.security.web.authentication.www.BasicAuthenticationFilter@66c9b52f, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@2349f14d, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@7ebe2e47, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@4cc65c2, org.springframework.security.web.session.SessionManagementFilter@91da29b, org.springframework.security.web.access.ExceptionTranslationFilter@7bc6935c, org.springframework.security.web.access.intercept.FilterSecurityInterceptor@16c1519e]  2020-11-22 12:50:46.656 INFO 245456 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path '/scbase'  2020-11-22 12:50:46.661 INFO 245456 --- [ main] com.example.demo.DemoApplication : Started DemoApplication in 3.832 seconds (JVM running for 7.07) |

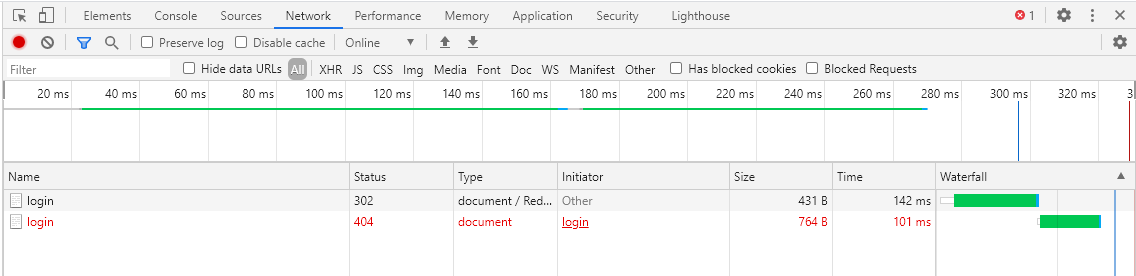
在项目启动过程中，生成了一个默认的密码， 以及默认的拦截器链。

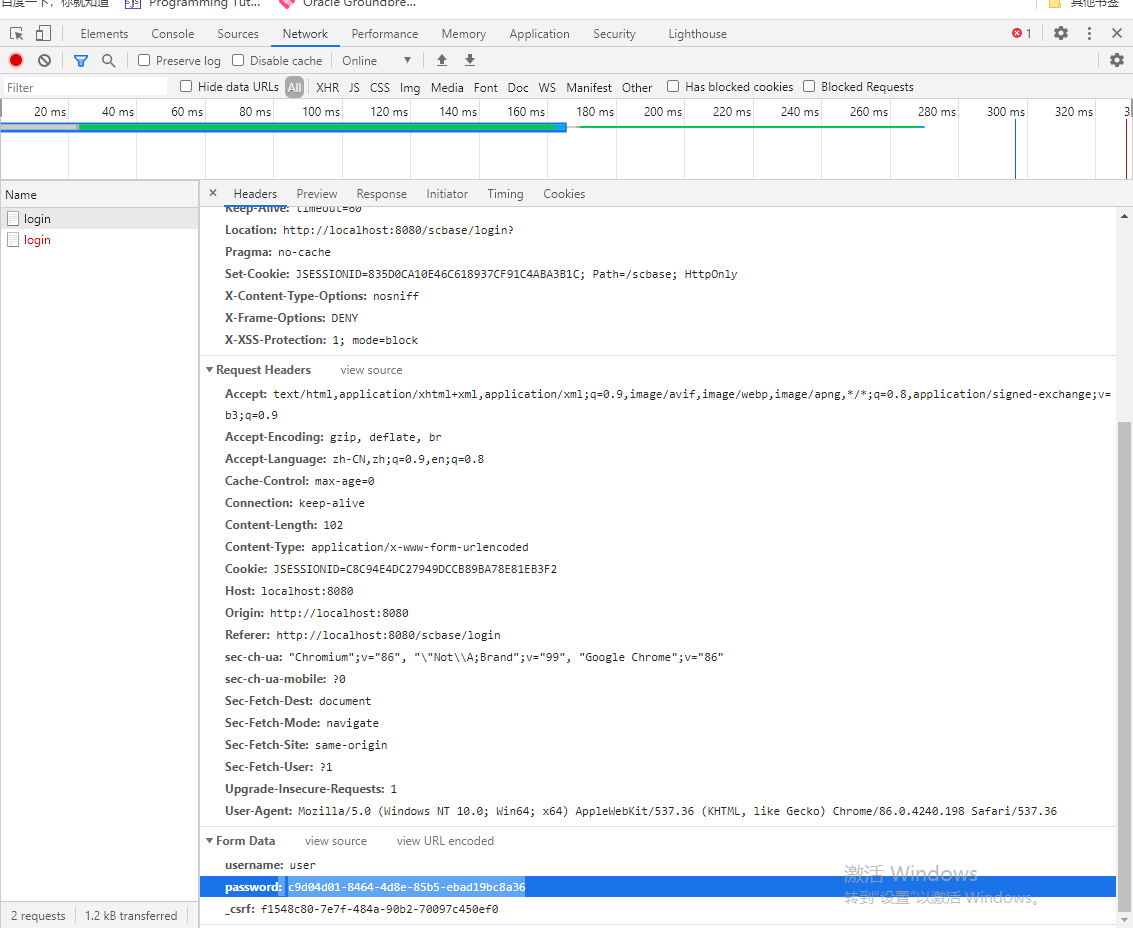
通过直接指定springsecurity 的最小依赖后，直接访问 <http://localhost:8080/scbase> 可以看到springSecurity的默认登录页面。

这里url跳转到 <http://localhost:8080/scbase/login>



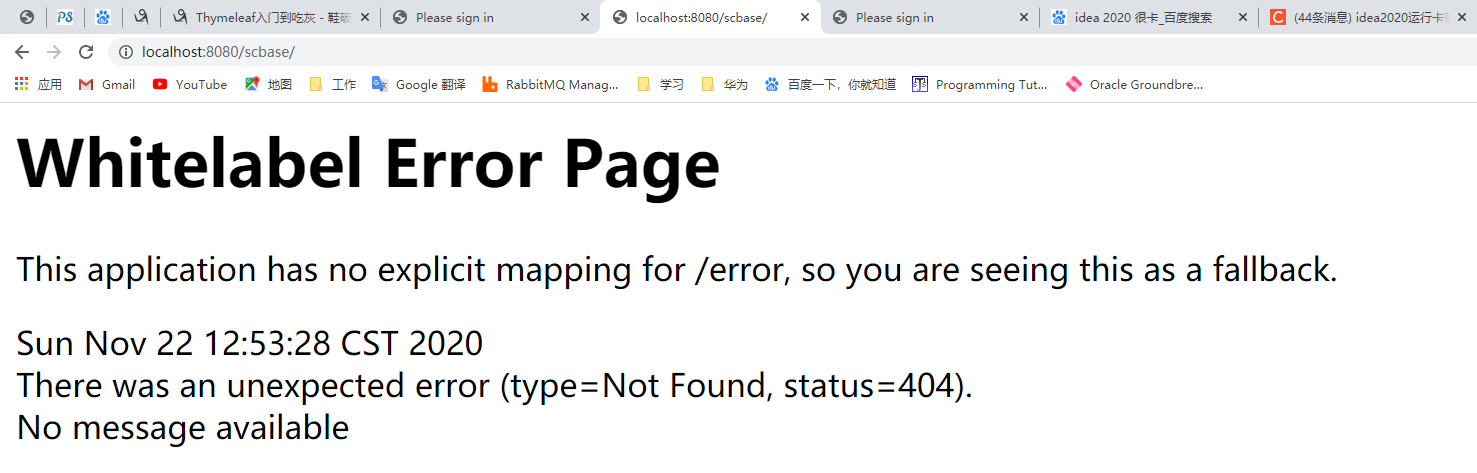
此时相关的http请求如下：





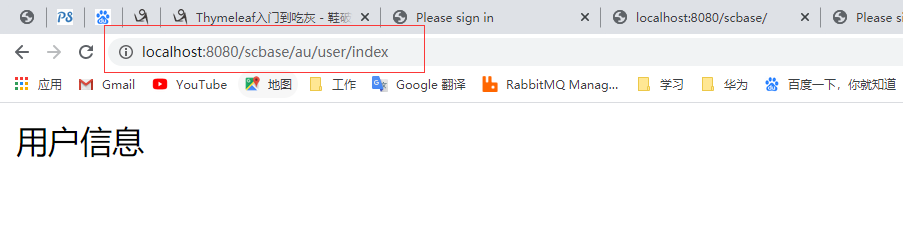
在这里用户名和密码是明文传输的。 这种认证方式是 【HTTP基本认证】

输入用户名和密码后



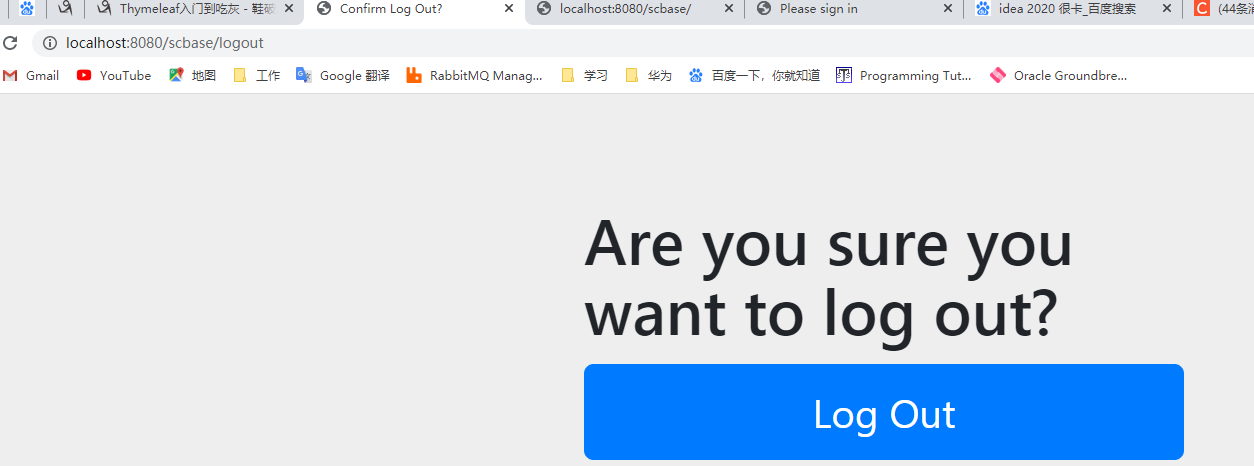
显示这个页面是因为系统没有配置，系统登录成功后的跳转页面。

目前这种认证方式是 【HTTP 基本认证】。

这种情况下，访问 <http://localhost:8080/scbase/au/user/index> 可以看到用户信息的页面。 

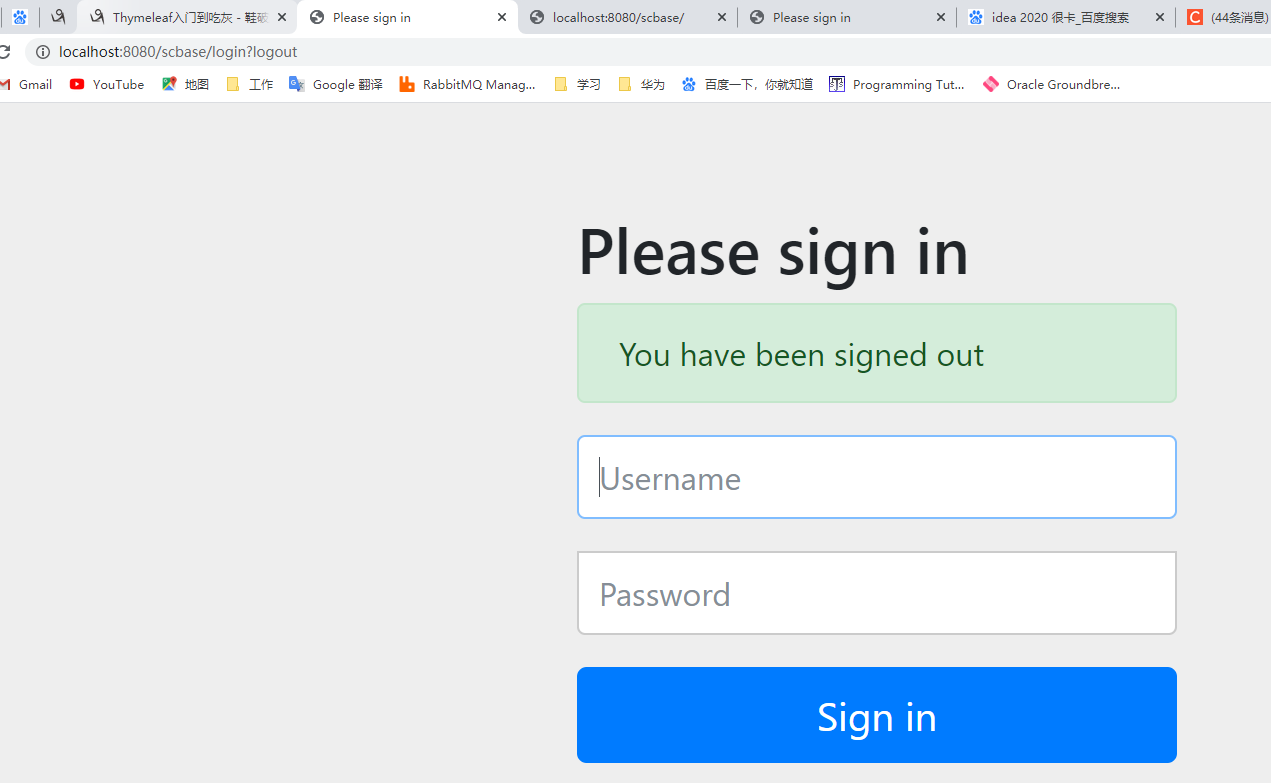
退出登录：

访问 <http://localhost:8080/scbase/logout> ，这是页面会提示你是否退出



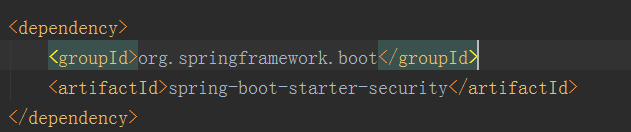
点击Logout 按钮，退出系统。

退出成功后，跳转到 <http://localhost:8080/scbase/login?logout> 地址。 让用户重新登录

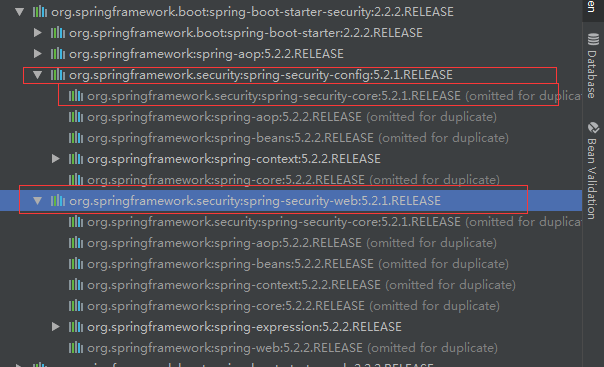


## Springboot 通过starter整合springSecurity

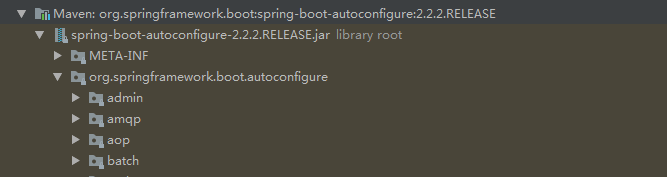
引入相关的starter

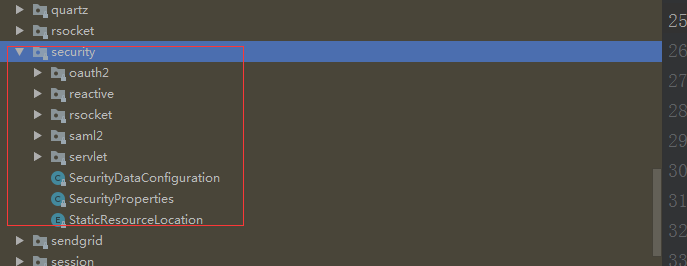


相关的依赖如下：

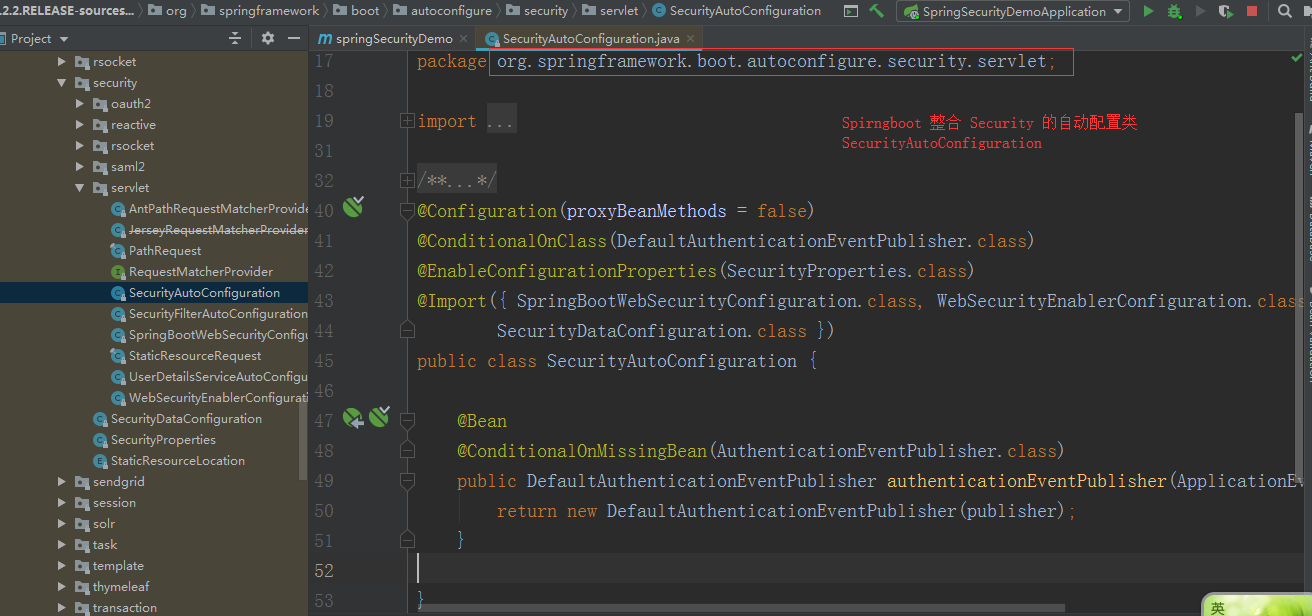


Spring Boot 整合 springSecurity 的相关的配置都是在springboot-autoconfig 中进行了配置





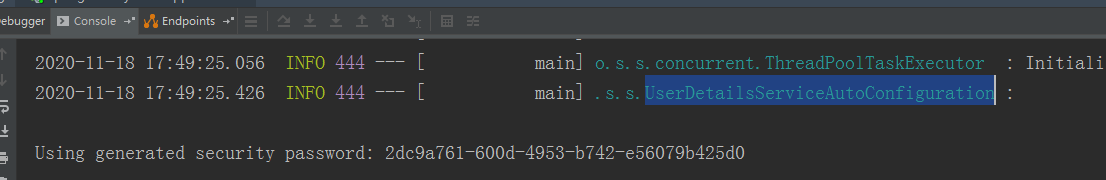
SpringBoot 整合Spring Security 的配置类是SecurityAutoConfiguration 类



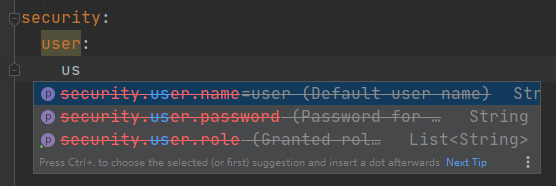
SecurityAutoConfiguration 顾名思义安全配置类。该类引入（@import）了 SpringBootWebSecurityConfiguration、WebSecurityEnablerConfiguration 和 SecurityDataConfiguration 三个配置类。 让这三个模块的类生效。是一个复合配置，是 Spring Security 自动配置最重要的一个类之一。 Spring Boot 自动配置经常使用这种方式以达到灵活配置的目的，这也是我们研究 Spring Security 自动配置的一个重要入口 同时 SecurityAutoConfiguration 还将 DefaultAuthenticationEventPublisher 作为默认的 AuthenticationEventPublisher 注入 **Spring IoC** 容器。如果你熟悉 Spring 中的事件机制你就会知道该类是一个 Spring 事件发布器。该类内置了一个HashMap<String, Constructor<? extends AbstractAuthenticationEvent>>维护了认证异常处理和对应异常事件处理逻辑的映射关系，比如账户过期异常 AccountExpiredException 对应认证过期事件AuthenticationFailureExpiredEvent ，也就是说发生不同认证的异常使用不同处理策略。

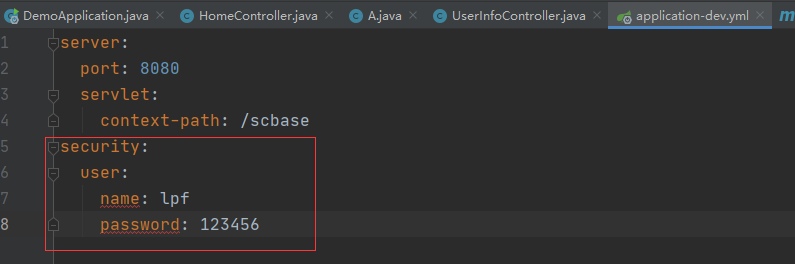
引入starter-security 之后，不做任何配置，然后启动springboot 项目

默认会调用 UserDetailsServiceAutoConfiguration 类来生成一个默认的密码。



## 在配置文件中指定用户名

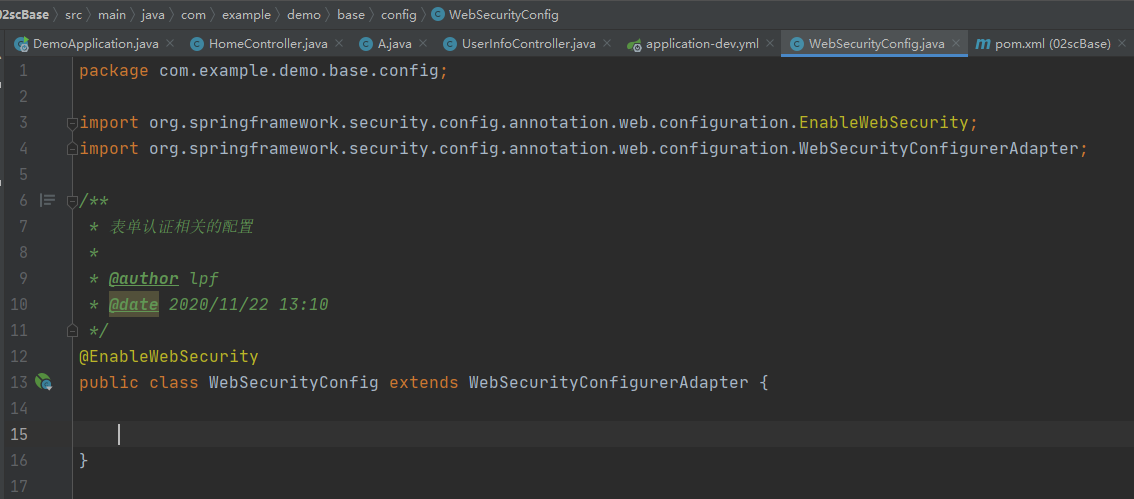




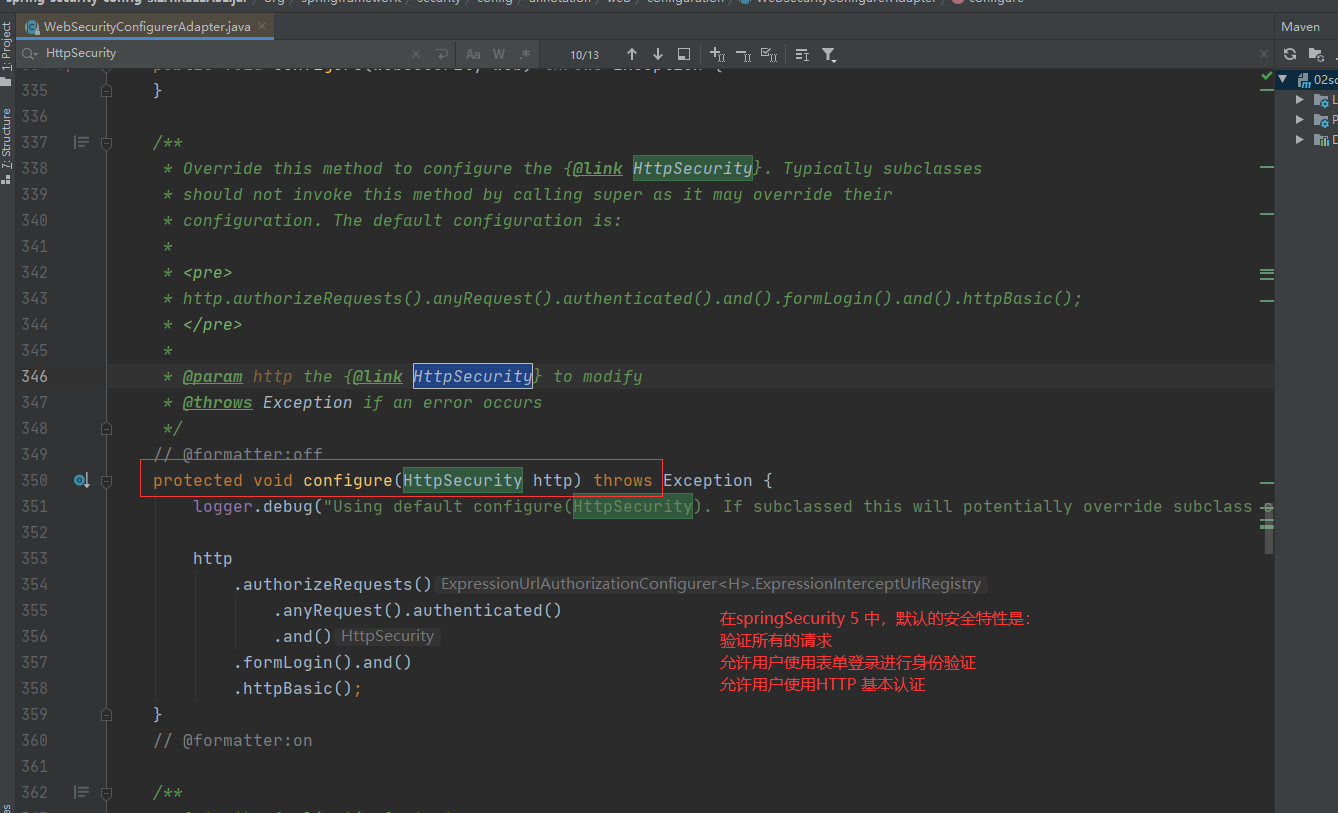
这里配置之后，执行，mvn clean 命令，重启项目，但是springSecurity 依然会生成默认的密码，且配置的用户名和密码无效。

## HTTP的表单认证

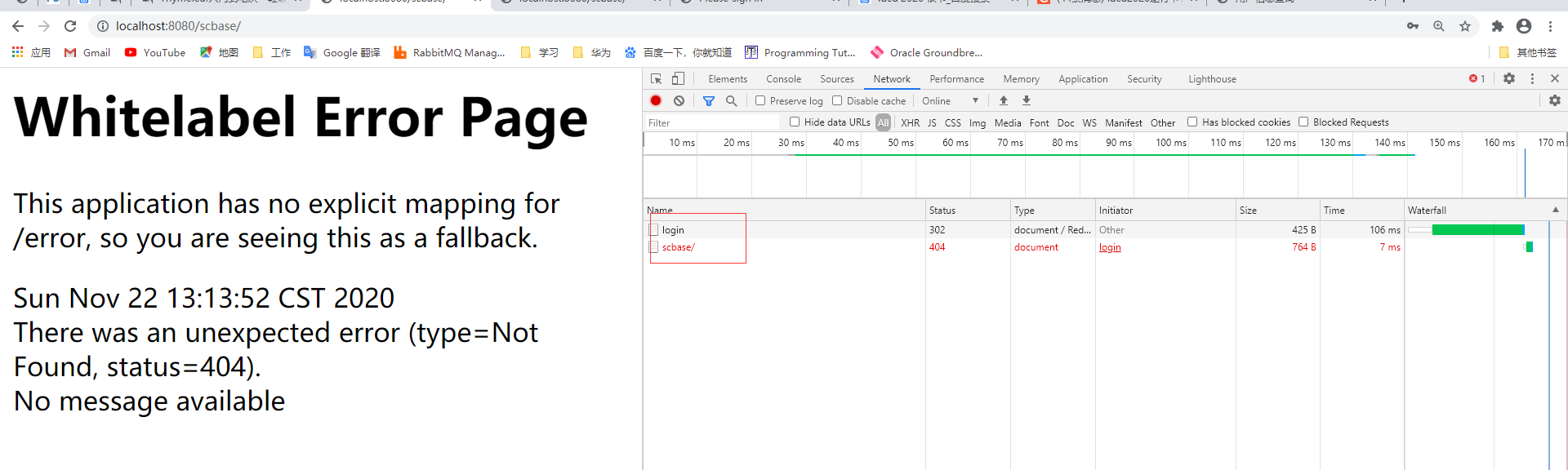
在项目中新建WebSecurityConfig 类并集成WebSecurityConfigurationAdapter 类来实现表单认证。



通过源码可以知道SpringSecurity的默认安全配置如下：

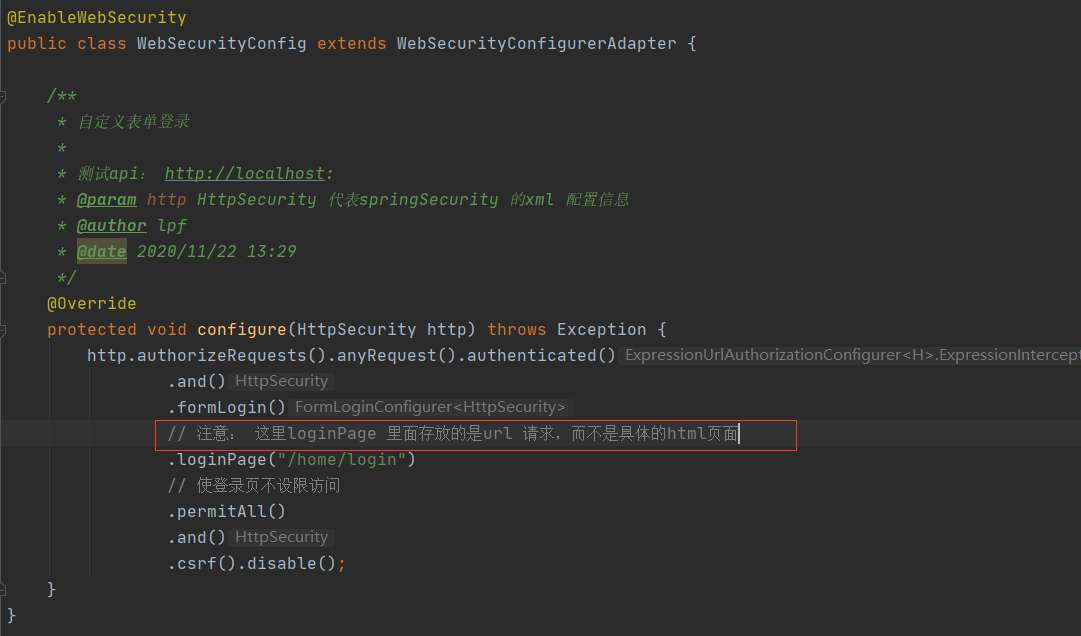


重启项目



## 自定义表单登录页

重写WebSecurityConfigurationAdapter 类的子类中 configure(HttpSecurity http) 方法



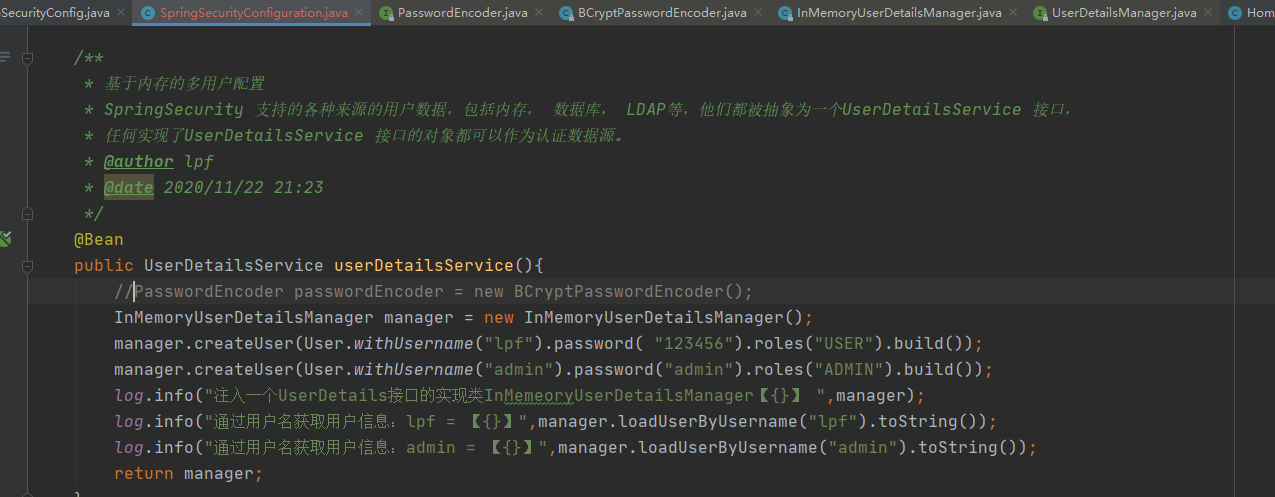
注意：在自定义登录页面时，loginPage() 里面的是跳转到登录页面的url，而不是具体的登录页面。



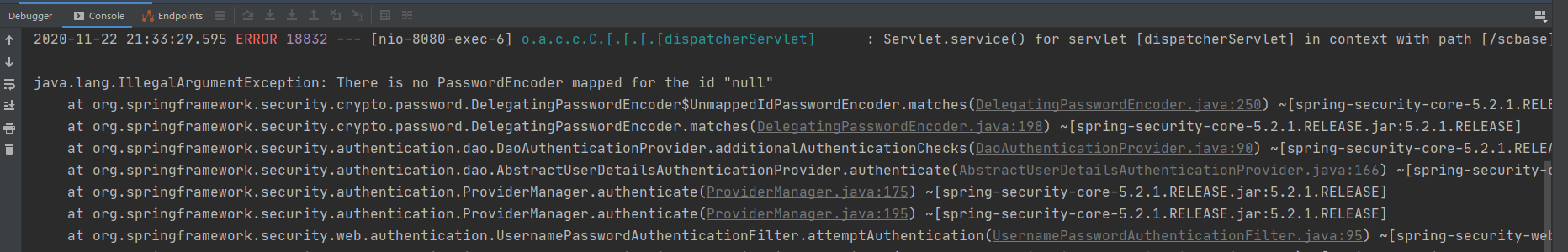
// 注意： 这里loginProcessingUrl 是指定springSecurity的默认认证接口的url地址。 这个地址需要和 登录页面的 form表单的action 保持一致。

## 认证与授权

### 基于内存的多用户支持



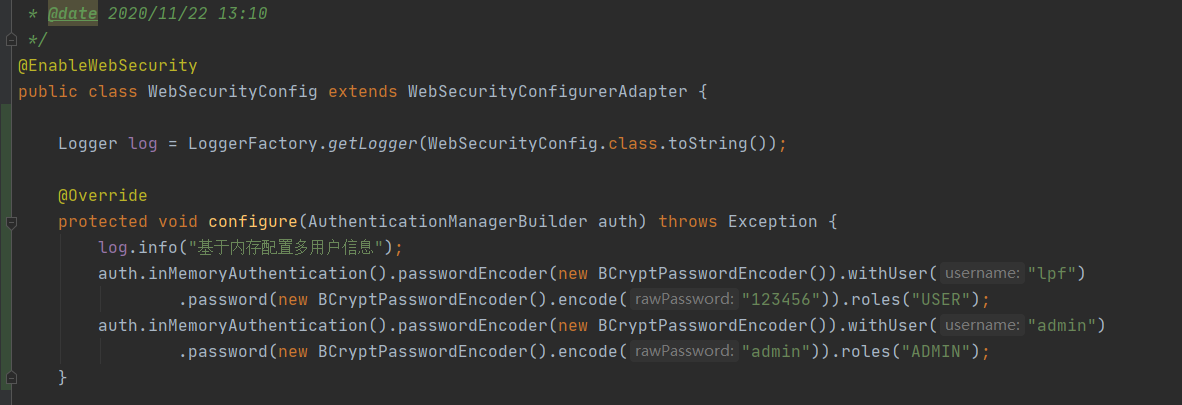
登录的时候，会报错：



java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"

是因为springSecurity 5 要求密码必须经过加密器处理才可以。

解决方案：



在WebSecurityConfigurationAdapter 子类中进行配置，并且对密码进行加密。

### 基于默认数据库模型的认证与授权

### 基于自定义数据库模型的认证与授权

## 疑惑：

1. 创建类 WebSecurityConfigure类继承 WebSecurityConfigurationAdapter 配置后，为什么还是采用http基本认证，而没有采用表单认证，因为 用户名和密码是可见的。 如何可以让用户名和密码不可见。