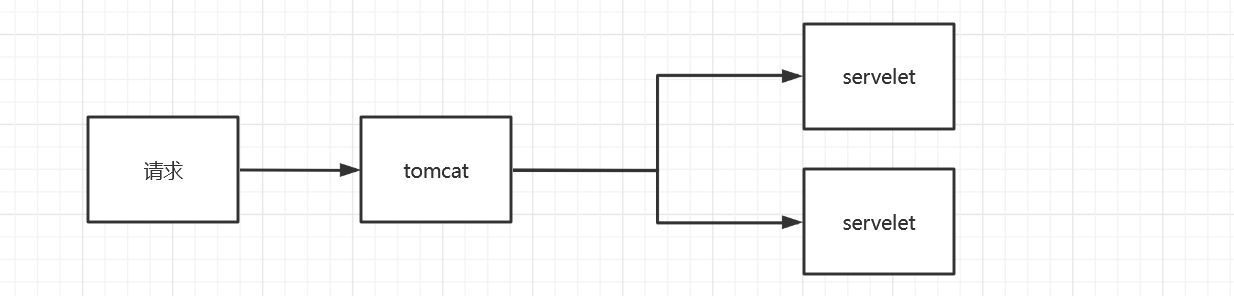
# 对servelet的简单前后端分离的开发封装

在一些特殊的场景中，比如在学校的一些期末项目中，学校会要求我们只能用最简单的jsp和servelet来敲代码，但是习惯了springboot前后端分离的我们就会觉得这样非常难受，所以，我们可以考虑在servelet的基础上对我们的代码进行一定的封装，达到我们可用的效果。

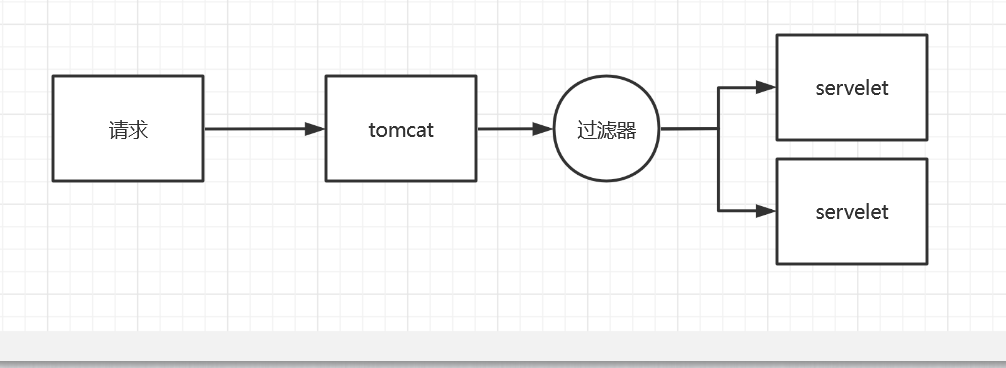
实现，我们需要去了解一下我们的请求到达服务器的一个过程，在没有过滤器时，

1. 服务器接收到请求
2. 把请求转发给tomcat
3. Tomcat把请求转换为可以获取的数据类型，然后传递给servelet



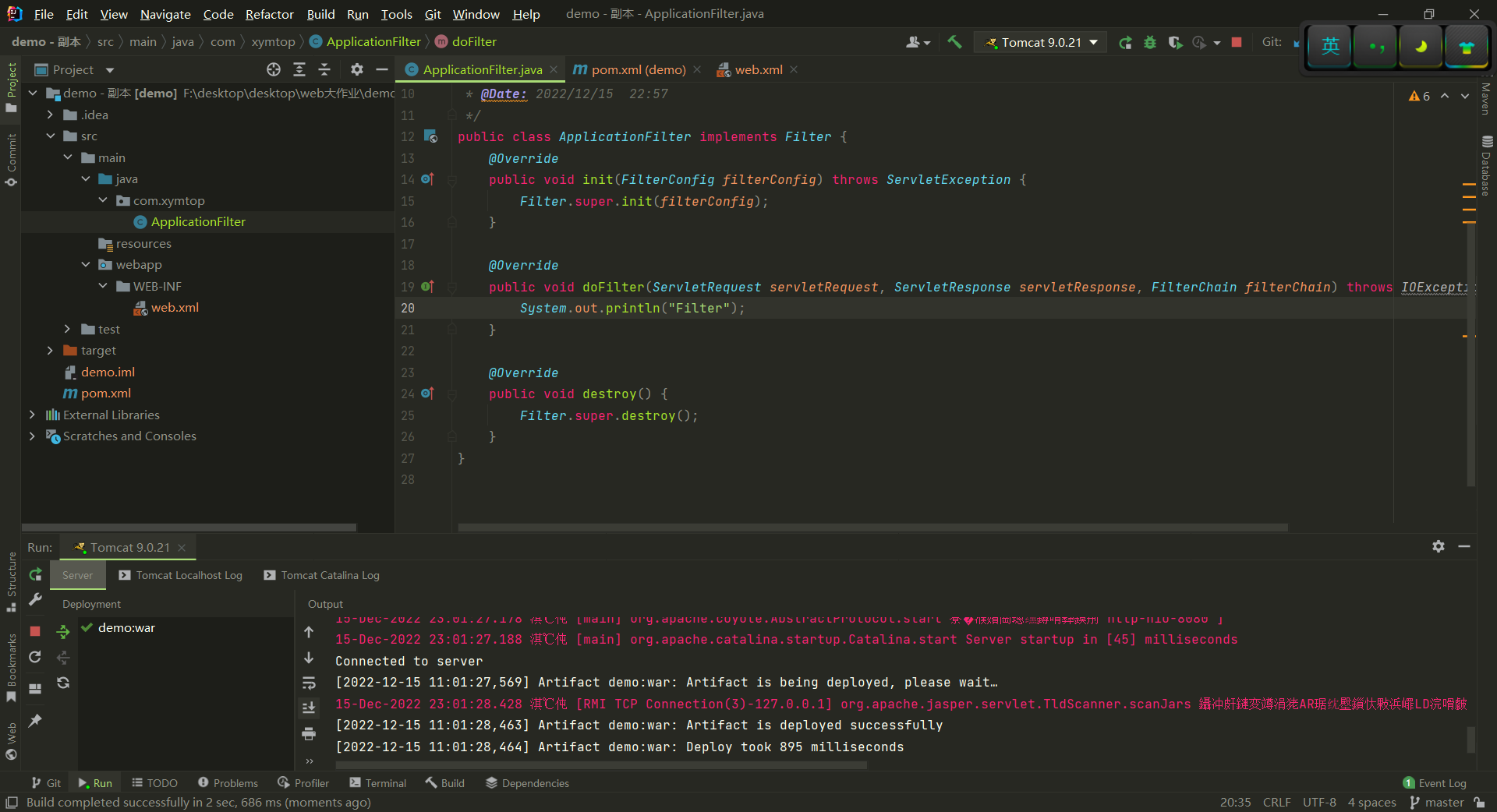
在有过滤器时，

1. 服务器接收到请求
2. 把请求转发给tomcat
3. Tomcat把请求转换为可以获取的数据类型，然后传递给过滤器
4. 过滤器进行处理再传递给servelet

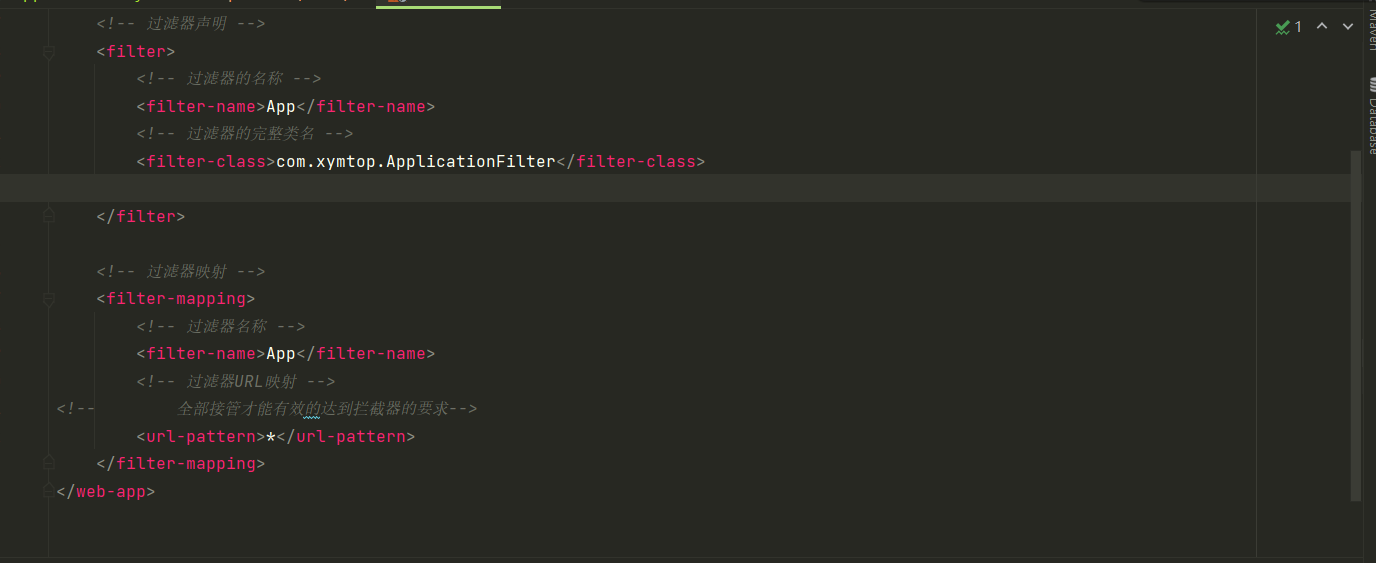


经过上面的分析，我们可以很容易的思考到，在servelet中，我们可以很容易的定义一个过滤器，所以，我们是否可以在过滤器上面下功夫，我们自己写一个分发请求到各个控制器的内容。

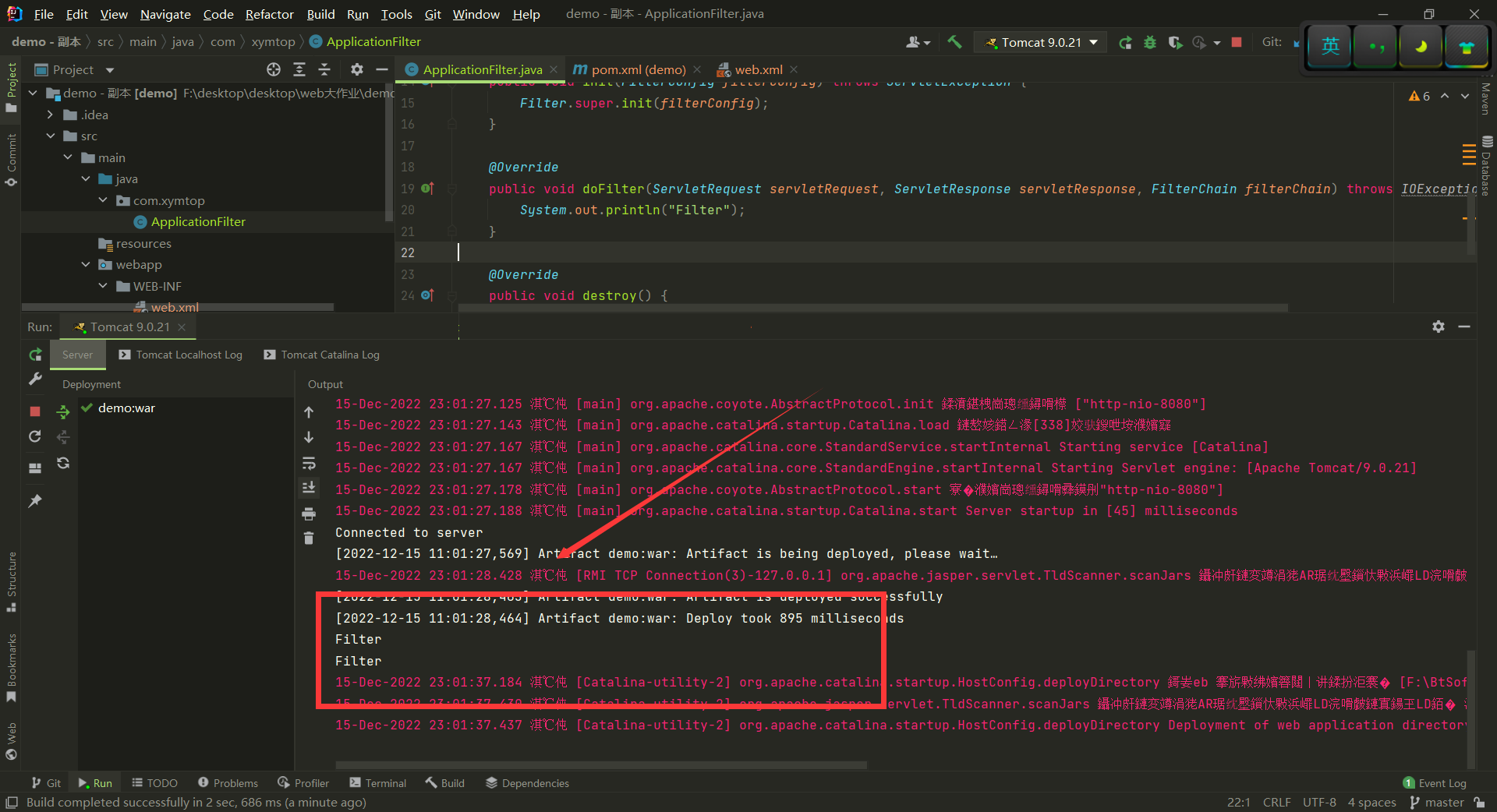
下面我们定义一个过滤器，并且随便写一些内容，测试一下。



过滤器在web.xml中的配置

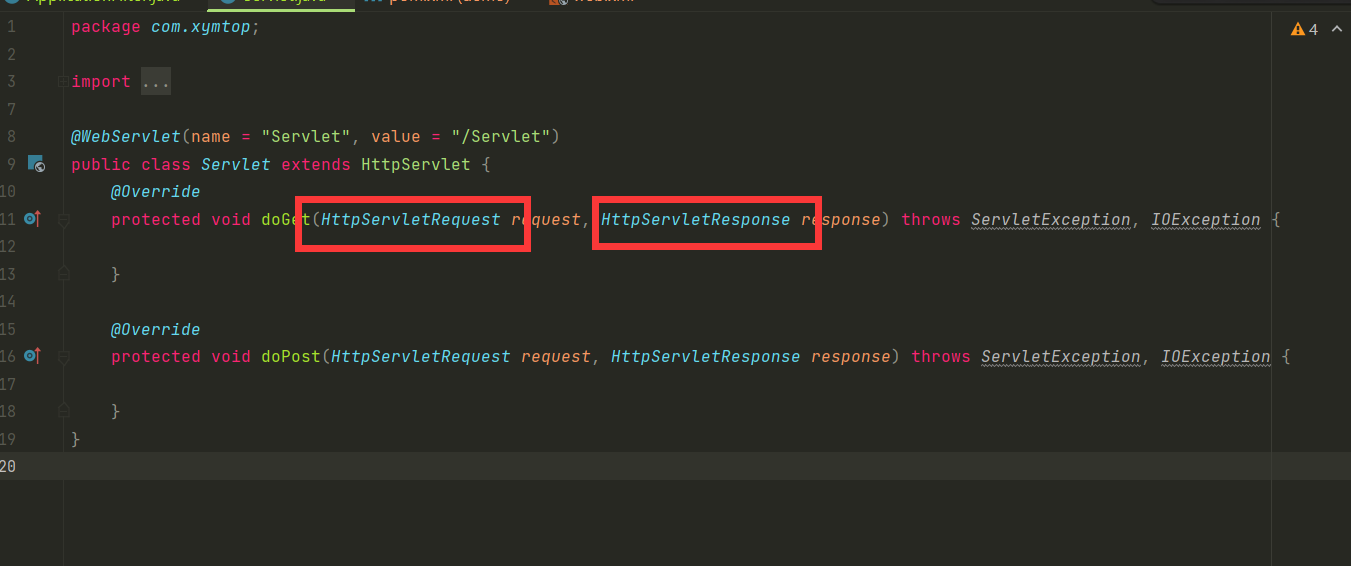


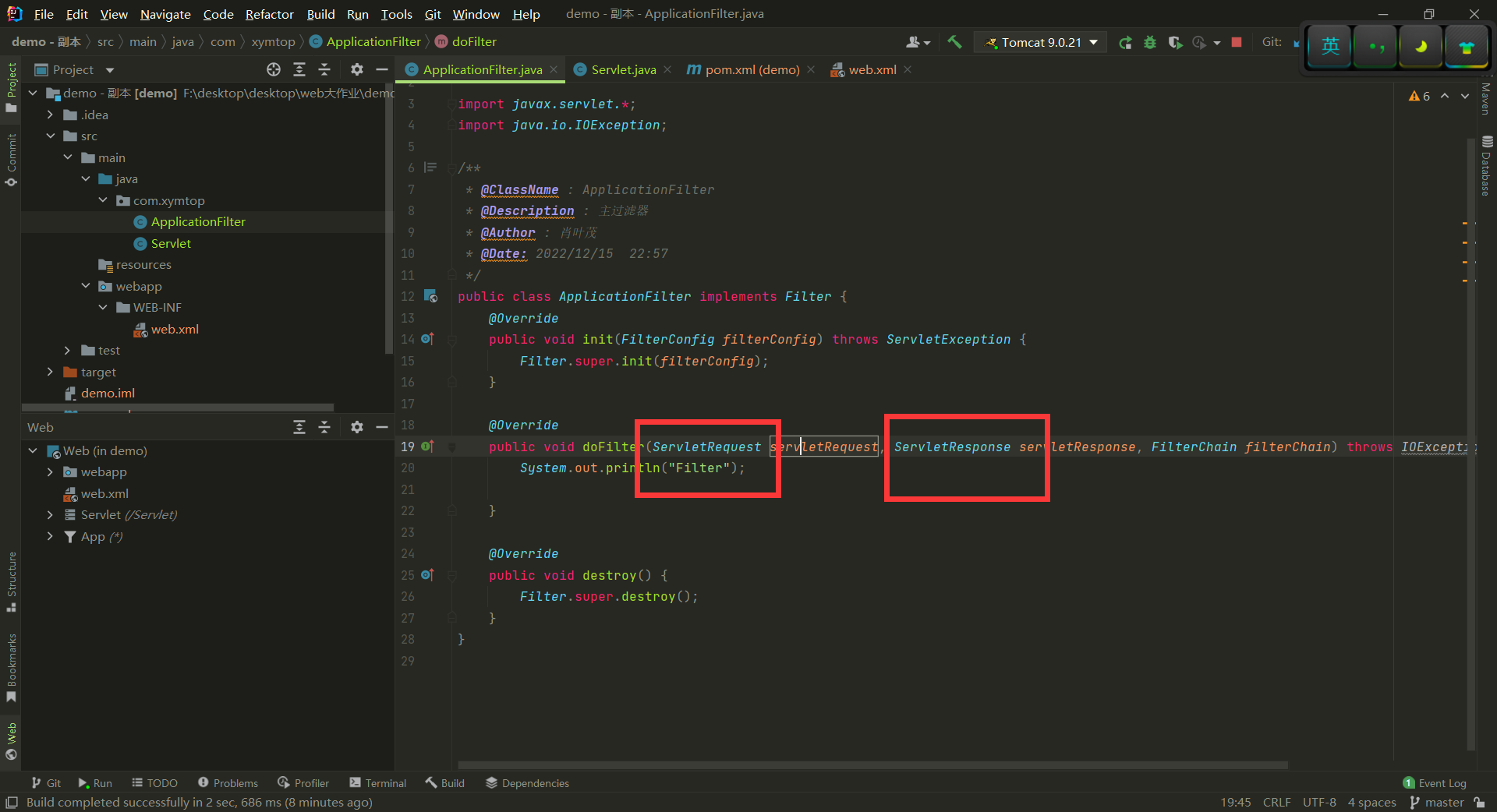
我们对该地址发送一个请求，可以看到过滤器已经在正常工作了。



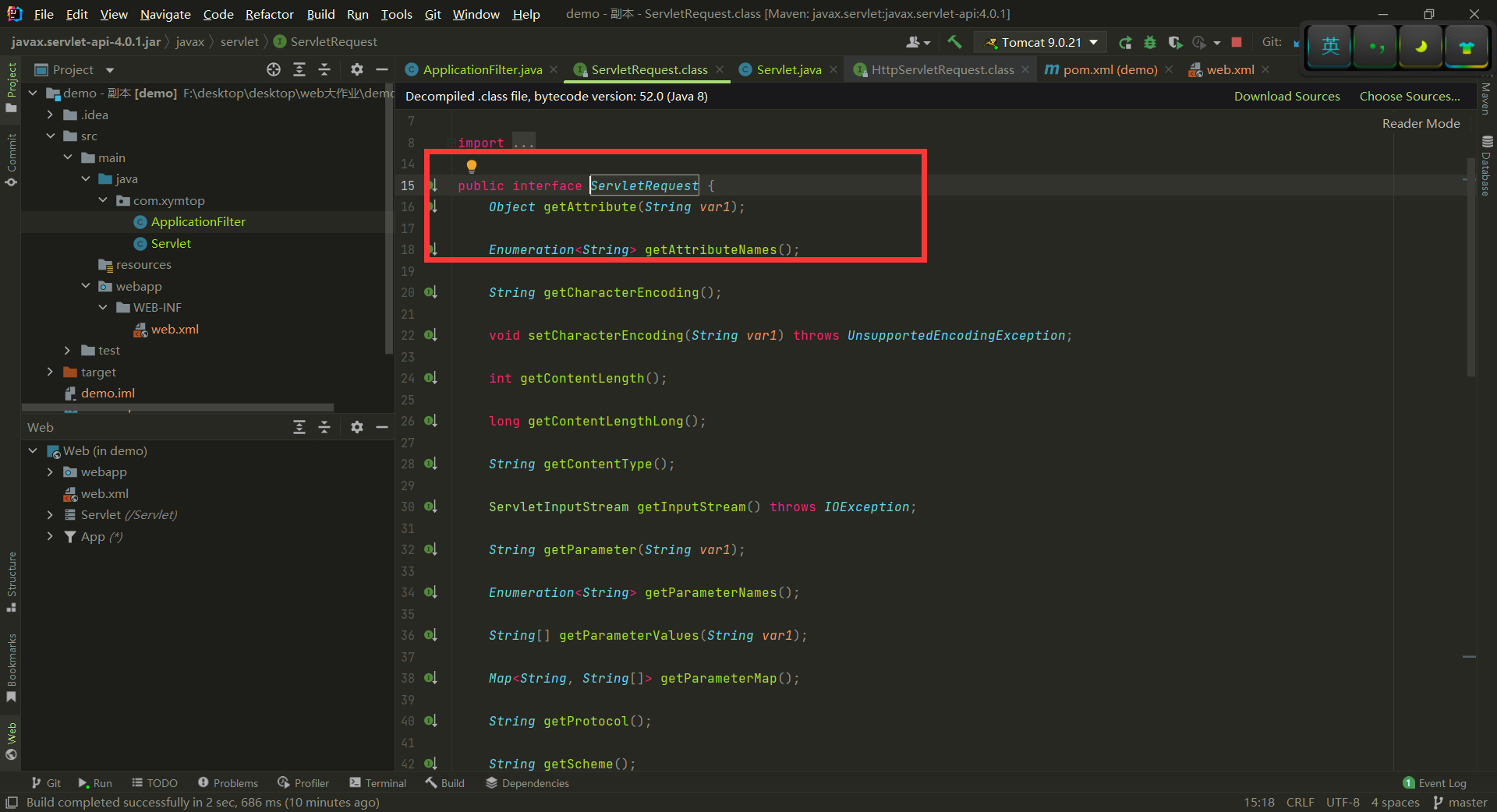
现在，我们可以在doFilter这个方法中写我们的方法即可。

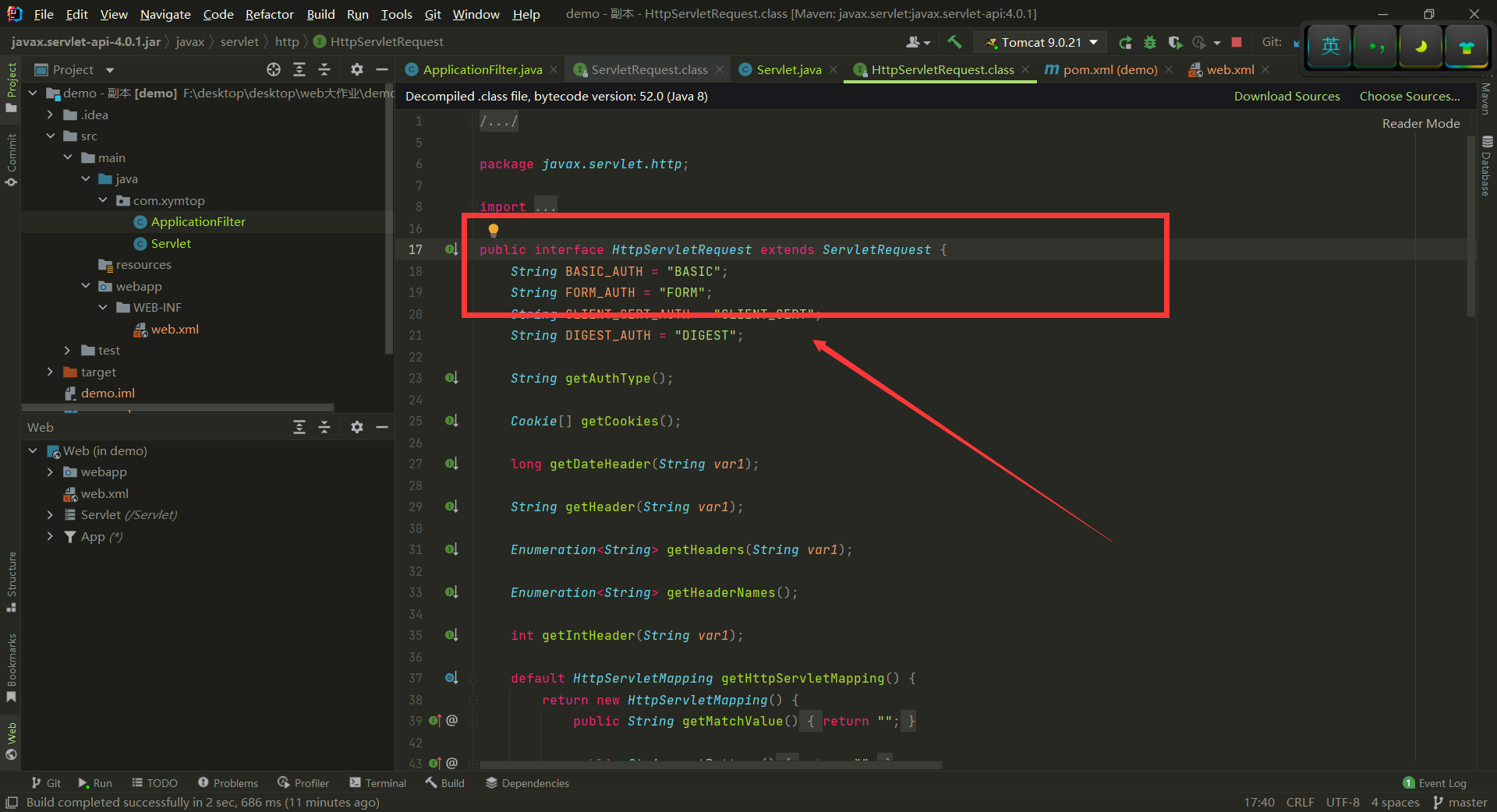
现在，我们已经可以获取到我们的响应和请求的对象了，但是他们目前是serveletrequest类型和serveletresponse类型，不是我们可以直接在jsp中进行操作的httpserveletrequest和httpserveletresponse。



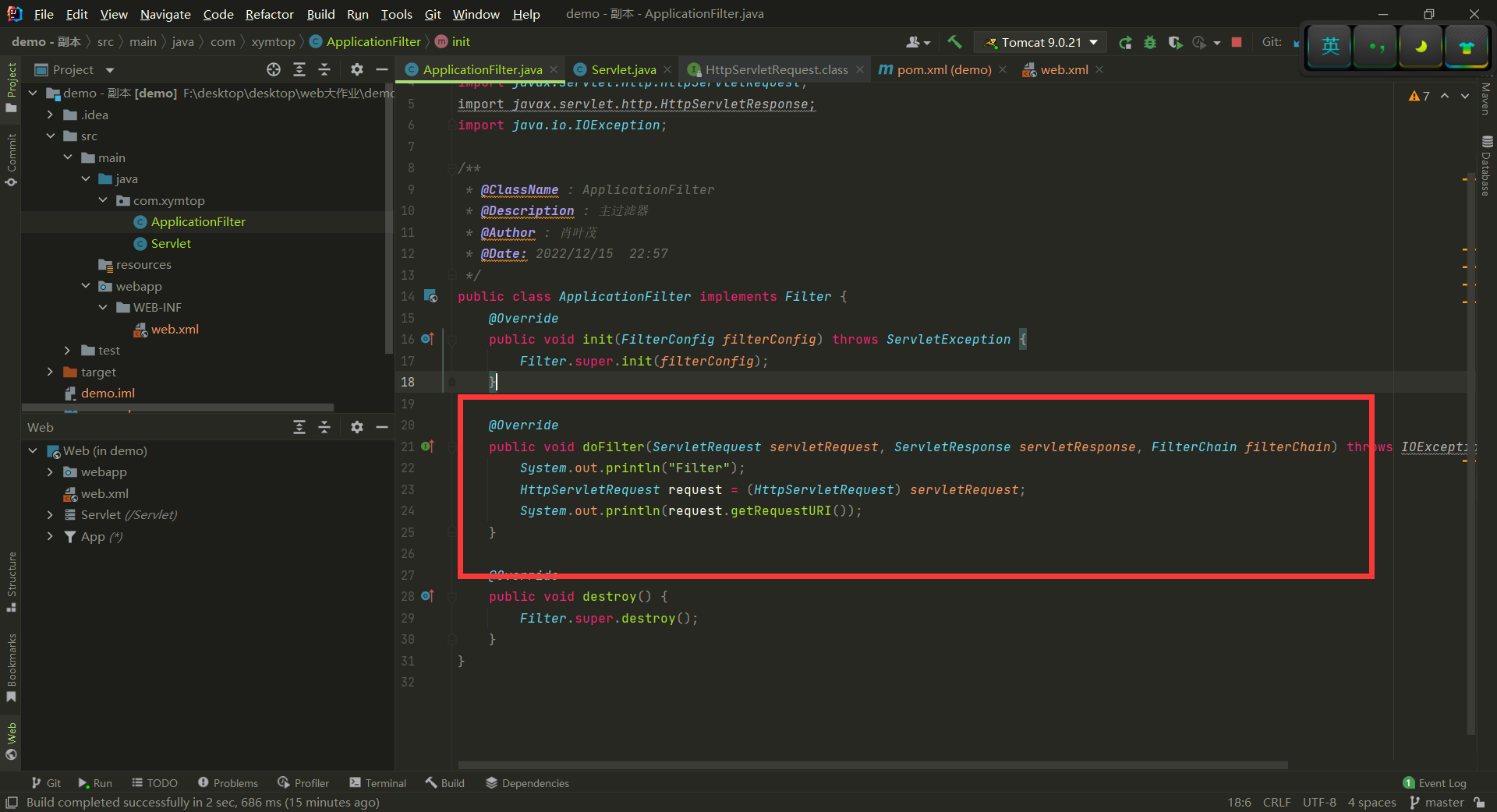


我们可以先探究他们两的关系，其实serveletrequest和httpserveletrequest是父类接口和子类接口的关系，也就是说，httpserveletrequest中有的属性和方法在serveletrequest中都有，所以，我们可以尝试把serveletrequest转换为httpserveletrequest。

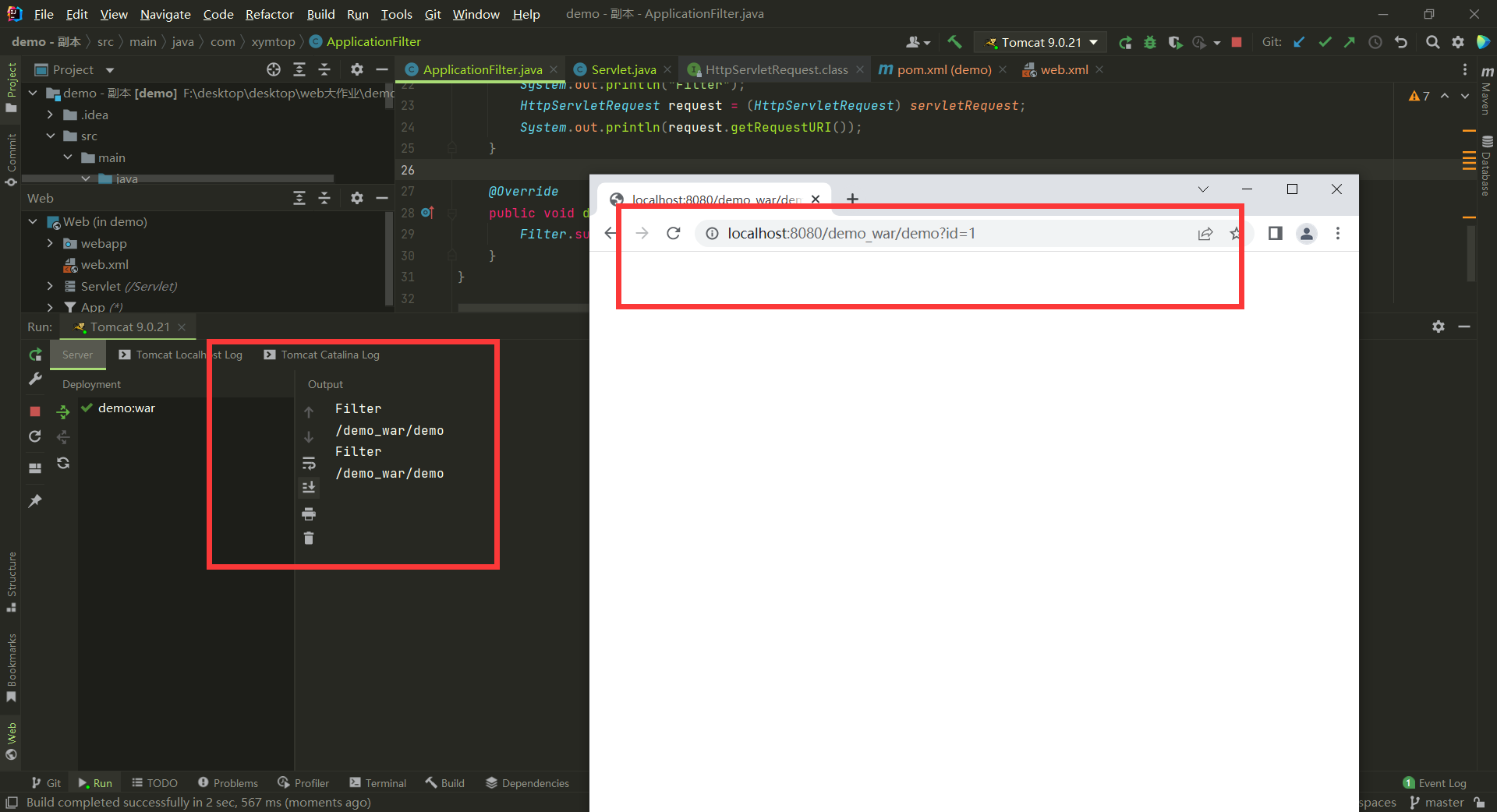




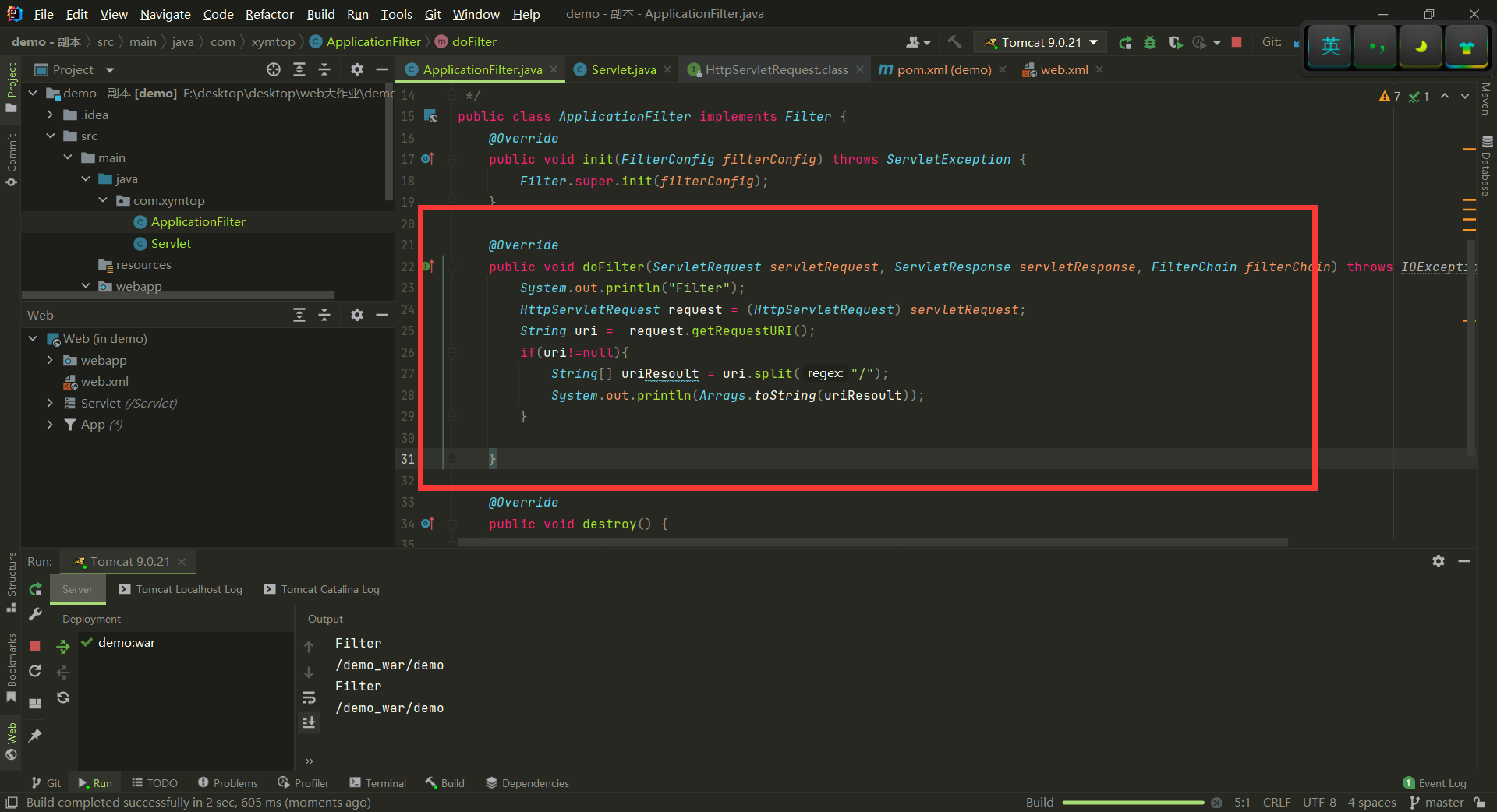
我们将serveletrequest强转为httpservelet，然后我们可以尝试获取一下我们当前请求的URI，如果可以获取到的话，说明我们通过这个办法是可以获取到当前请求的路径的，我们就可以开始构思下一步的开发。

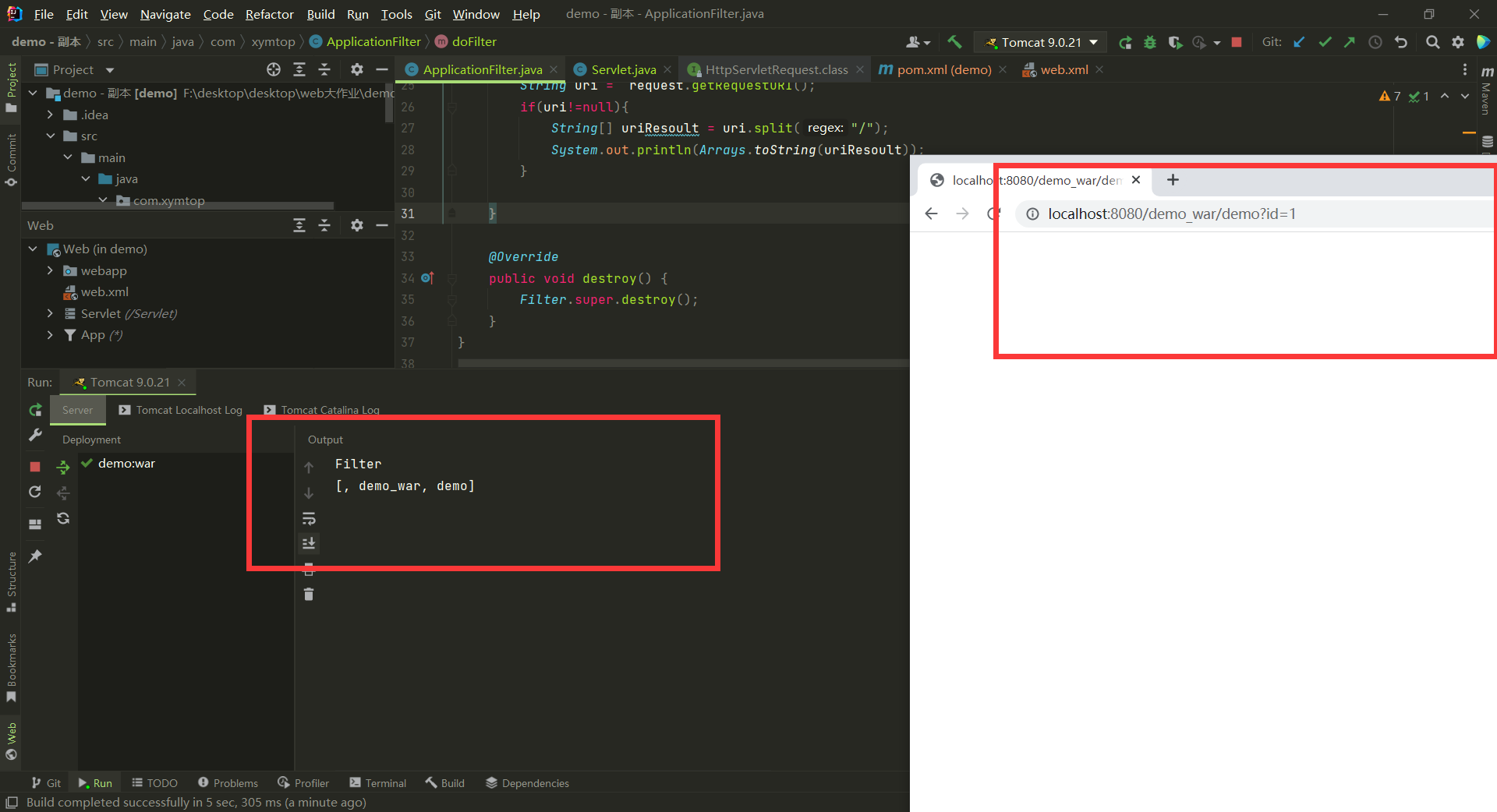


可以看到，目前已经获取到参数，可以进行下一步--解析参数。



下面我们对获取到的字符串类型的uri进行解析，我们使用最简单的字符串处理办法用 / 来分开参数。





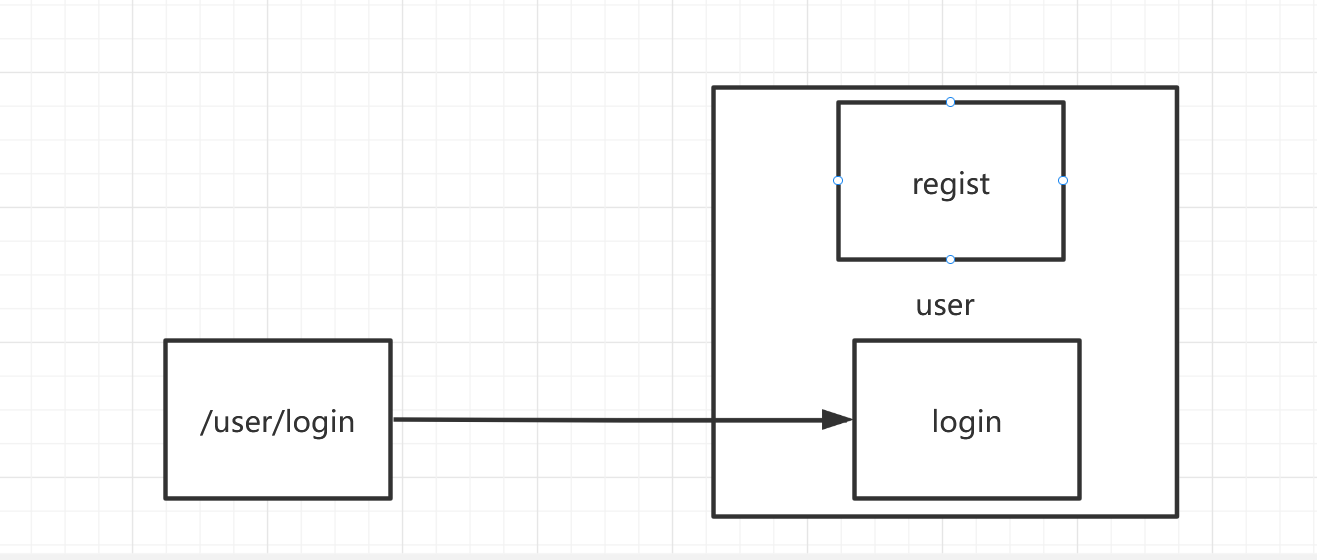
下面，我们可以考虑根据这些参数来构建一套体系，也就是如何通过这些参数来确定目前请求处理的方法，返回值是什么，如何返回的。

对于上面这些问题，我们分析出一些我们需要去执行的步骤：

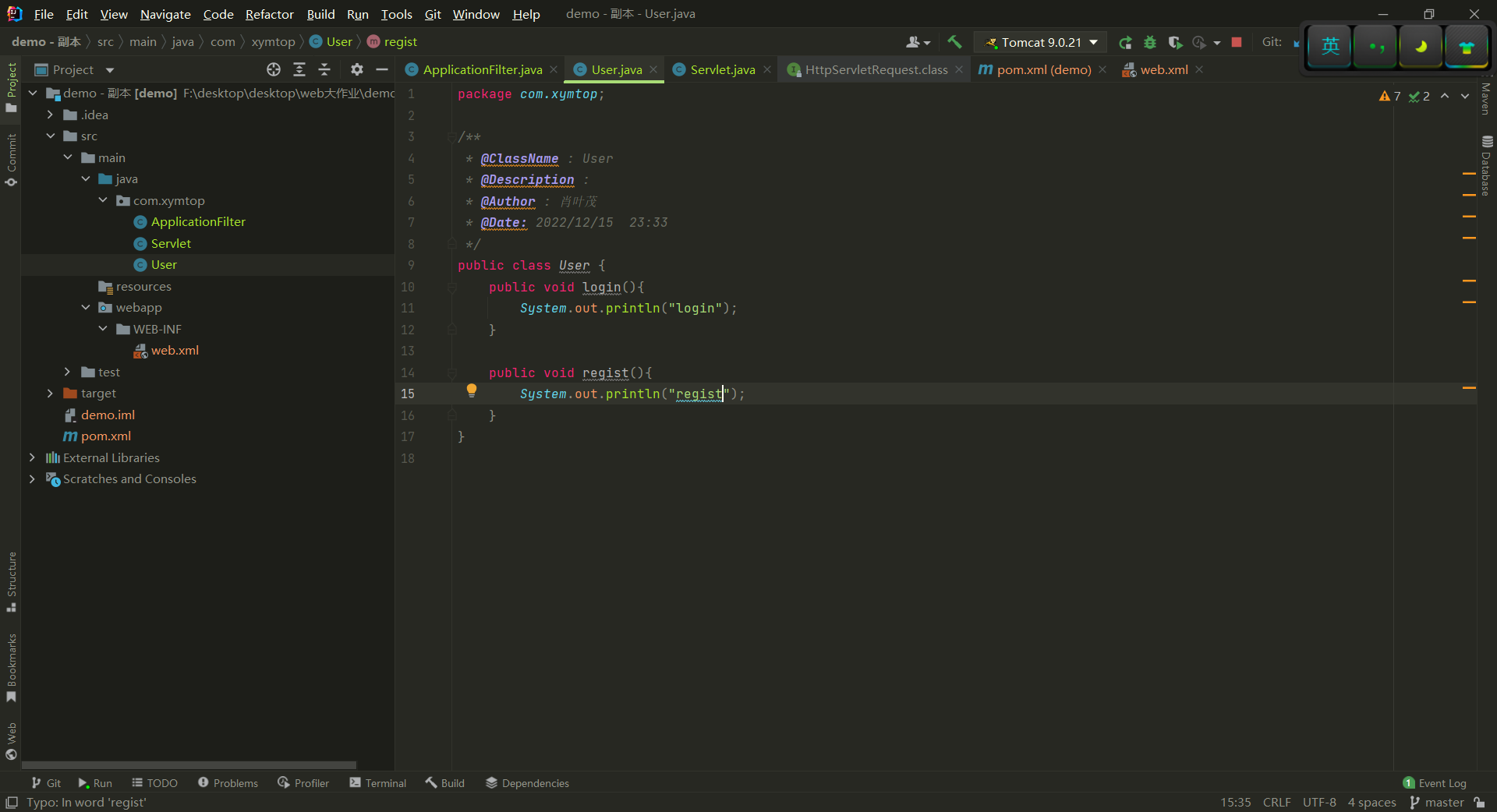
1. 解析出请求真正的参数
2. 判断需要执行的函数所在的类
3. 判断需要执行的函数
4. 执行该函数
5. 构造返回值对象并且接收返回值
6. 确定返回方式，是json返回还是直接返回字符串，图片等。

下面我们开始一步步的解决这些问题，实现解决第一个问题，如何解析出真实的参数?

在这个简单的demo中，我们主要是为了满足高效方便的要求，而我们对其他的一些内容要求并不高，所以，在这里，我们不妨定义后两个参数可以直接唯一确定我们需要执行的方法，也就是我们可以直接使用URI解析后的最后两个字符串来确定我们需要做的操作。就比如最后两个字符串为user和login，我们就应该有理由相信，user可能是用户在控制器中定义的一个对于某个控制器的同意的调用的参数，而login可能和这个类下面定位到需要执行的方法相关，所以，后面两个字符串就是我们需要解析的参数。



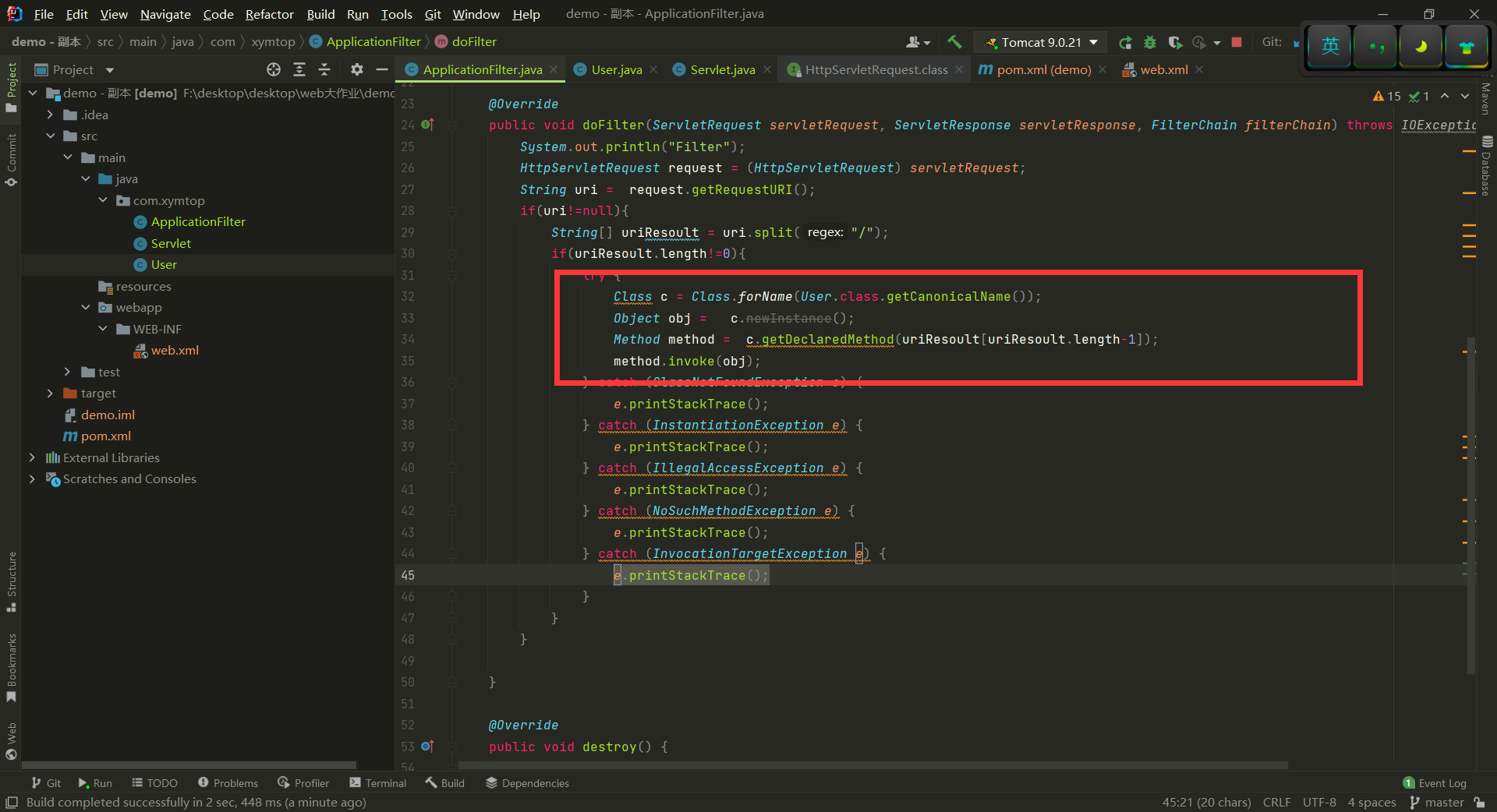
我们新建一个类，叫做user类，下面有login方法和regist方法。



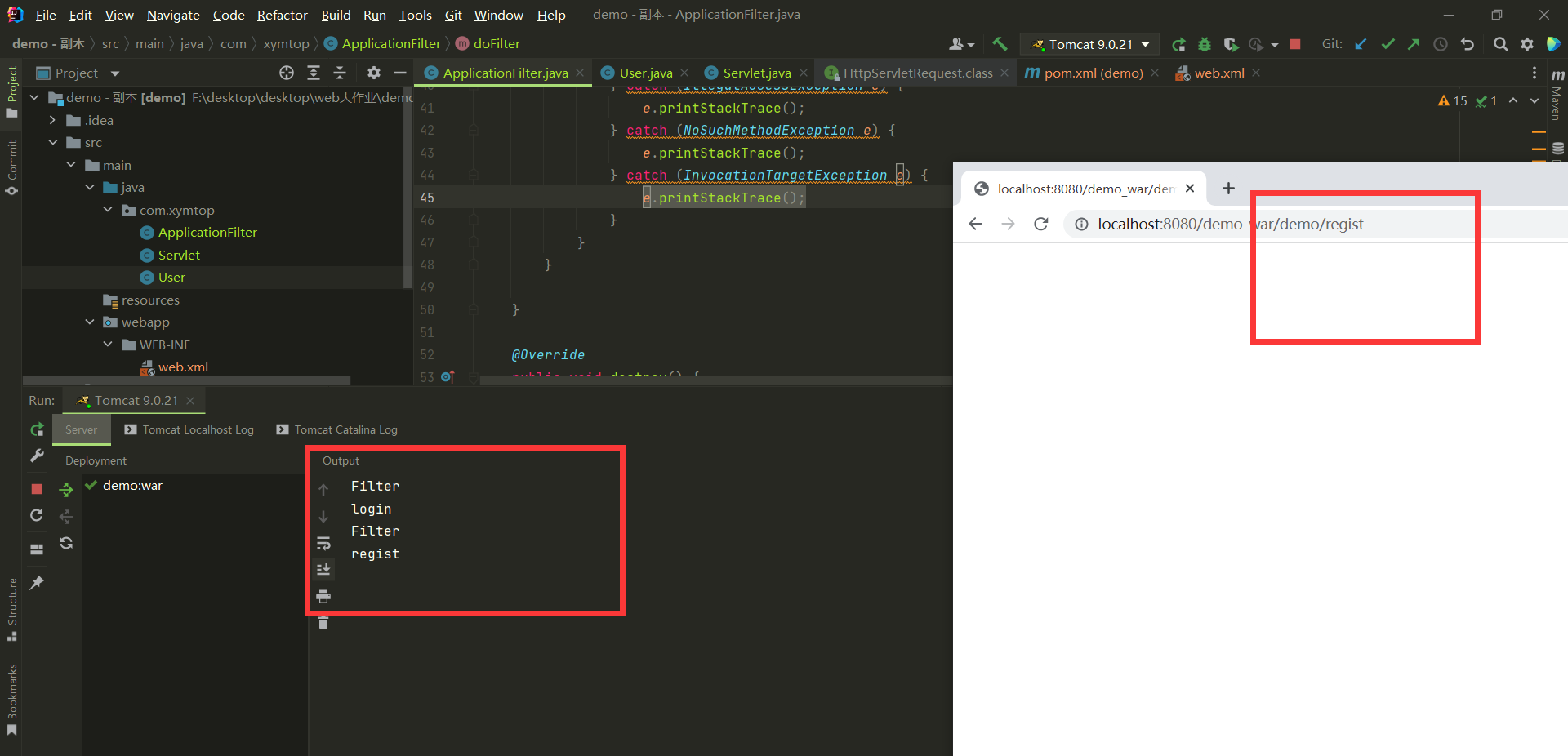
当我们请求的参数为user/login时，我们让他执行login函数，如果是user/regist时，执行regist函数。



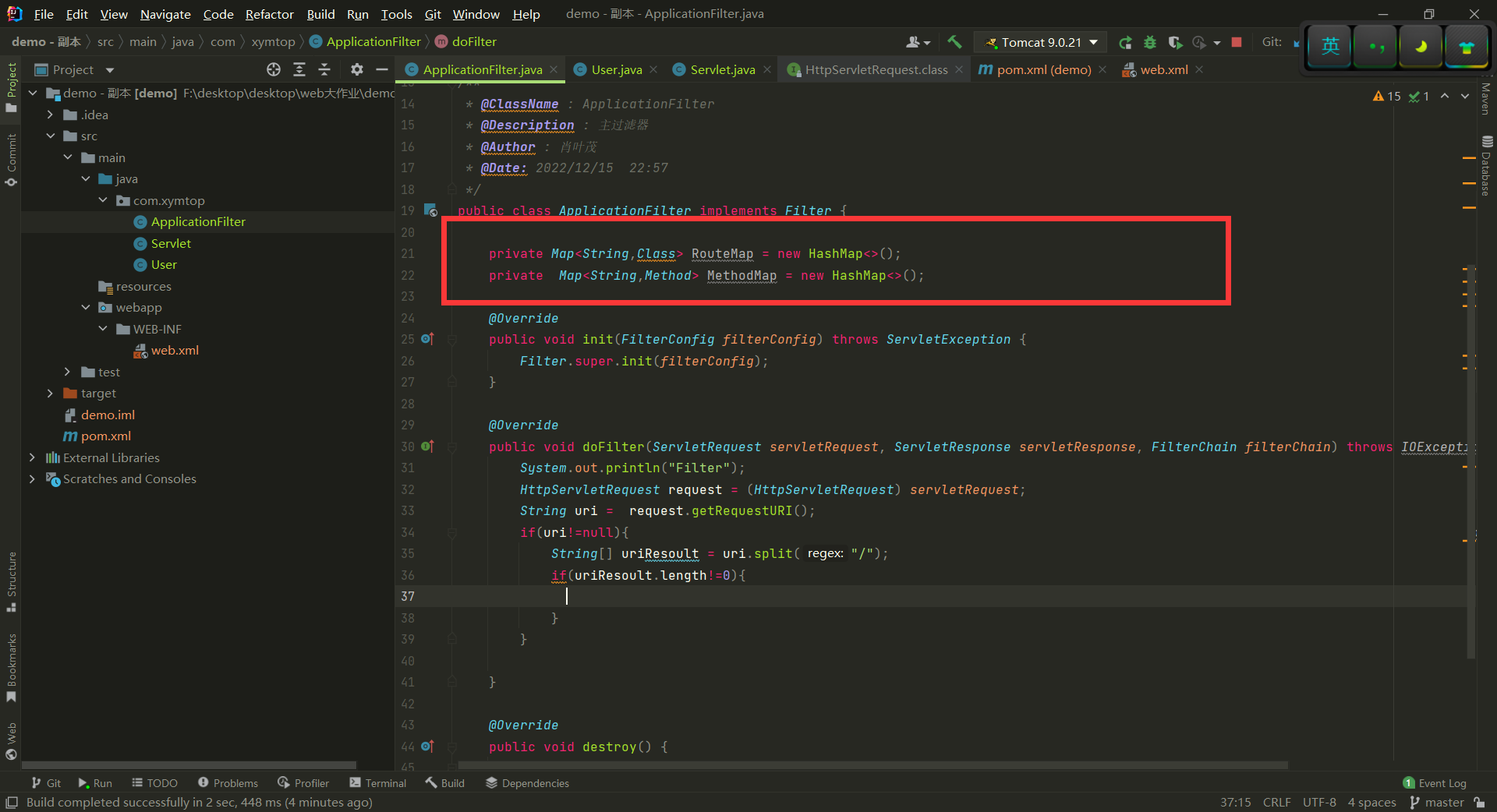
上面我们实现了通过参数来静态调用函数，但是，在我们日常的开发中，我们根本不知道到底有几个函数，到底有几个类，所以，我们这种静态的办法是行不通的，我们需要寻求一种动态的函数调用方法。这时，我们会想到我们学习到的Java反射，我们是不是可以用反射来动态调用函数呢。



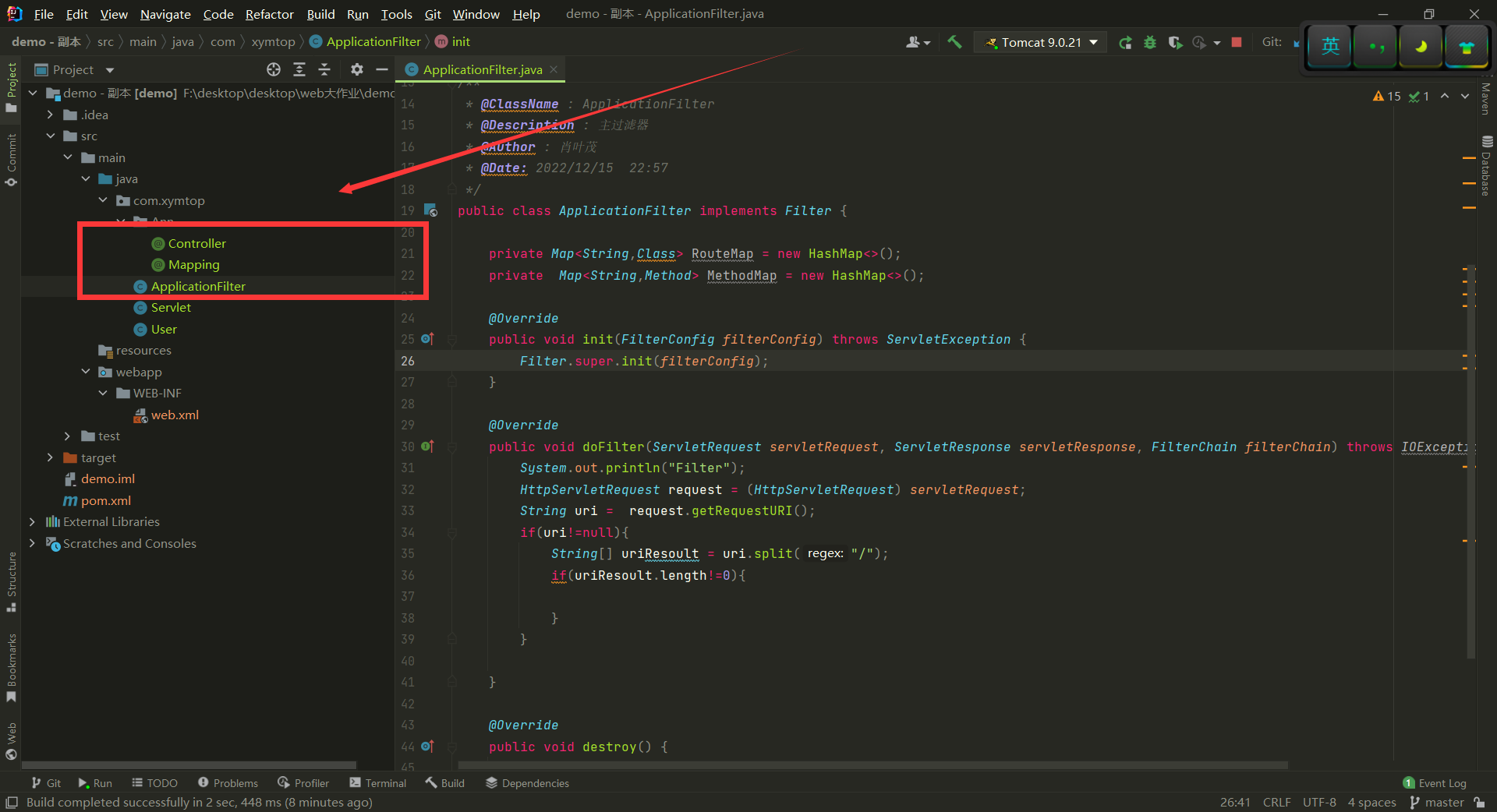
可以看到，我们目前已经可以实现对于浏览器的请求动态的调用函数，但是，目前我们只有一个类，我们知道请求可能需要去调用那个类的方法，但是在真实的环境中，我们无法提前预判到需要调用哪个函数的。



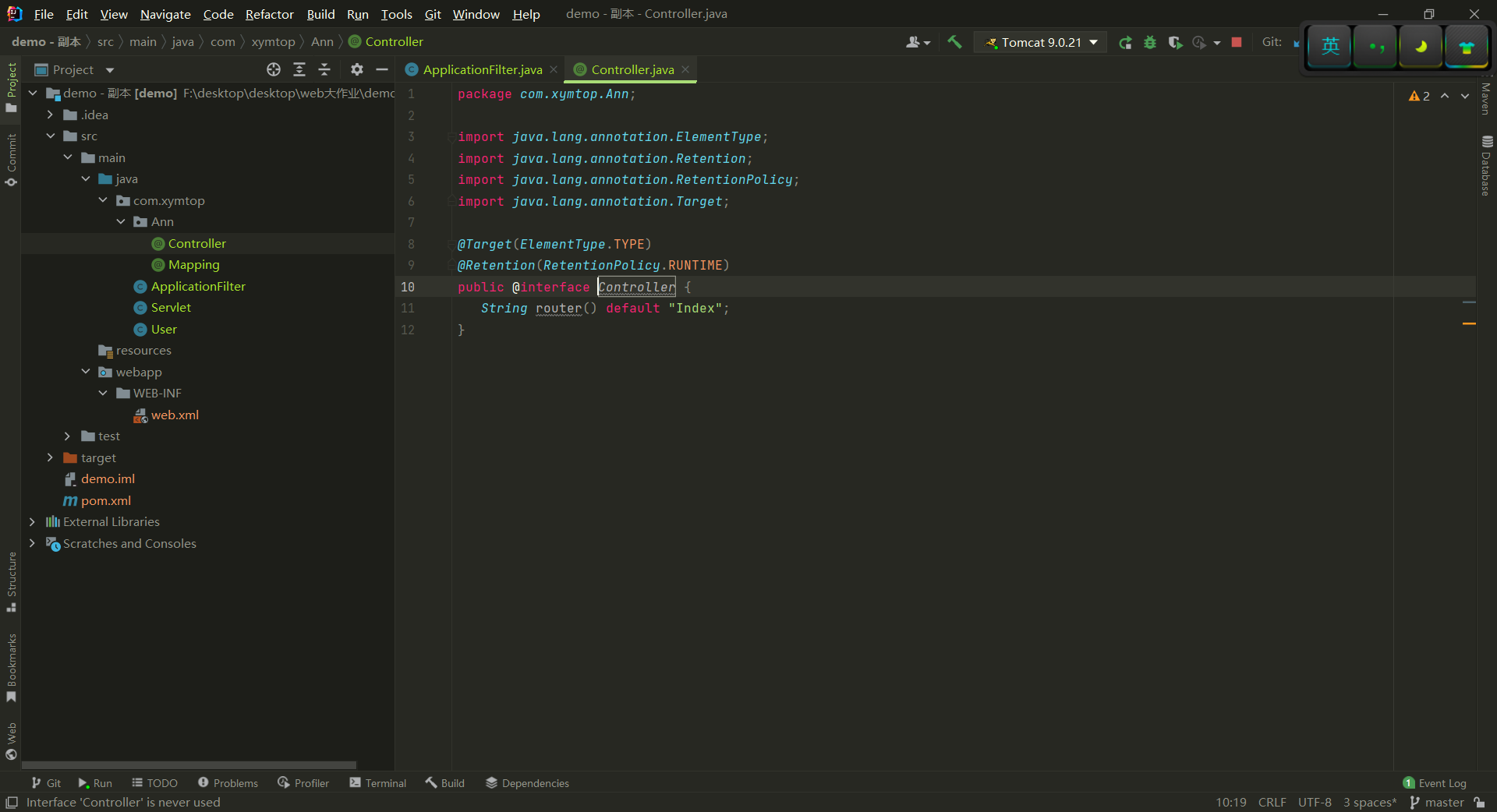
所以，这里我们可以维护两个map，一个是请求及其对于需要调用的类是什么，一个是请求需要调用的类下面的哪个函数。

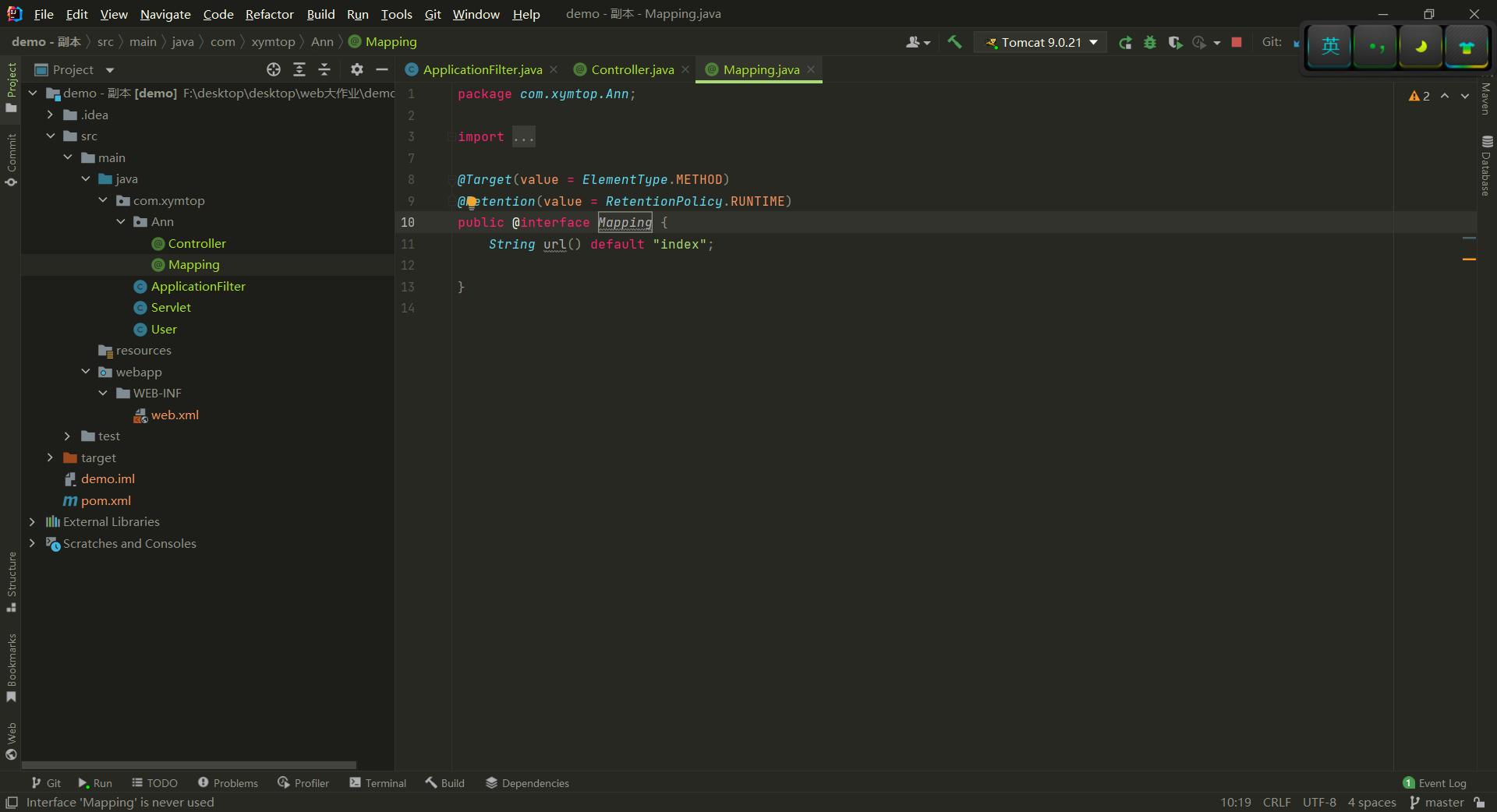


但是现在，我们只是定义了这两个map，里面还没有东西。那什么时候我们去往里面添加东西呢？肯定是我们知道了用户定义了一个请求的地址，然后得到了这个请求地址对应的类和函数。那我们怎么去获取到用户定义的地址和对应的类和函数呢。我们这里可以借用注解来实现。

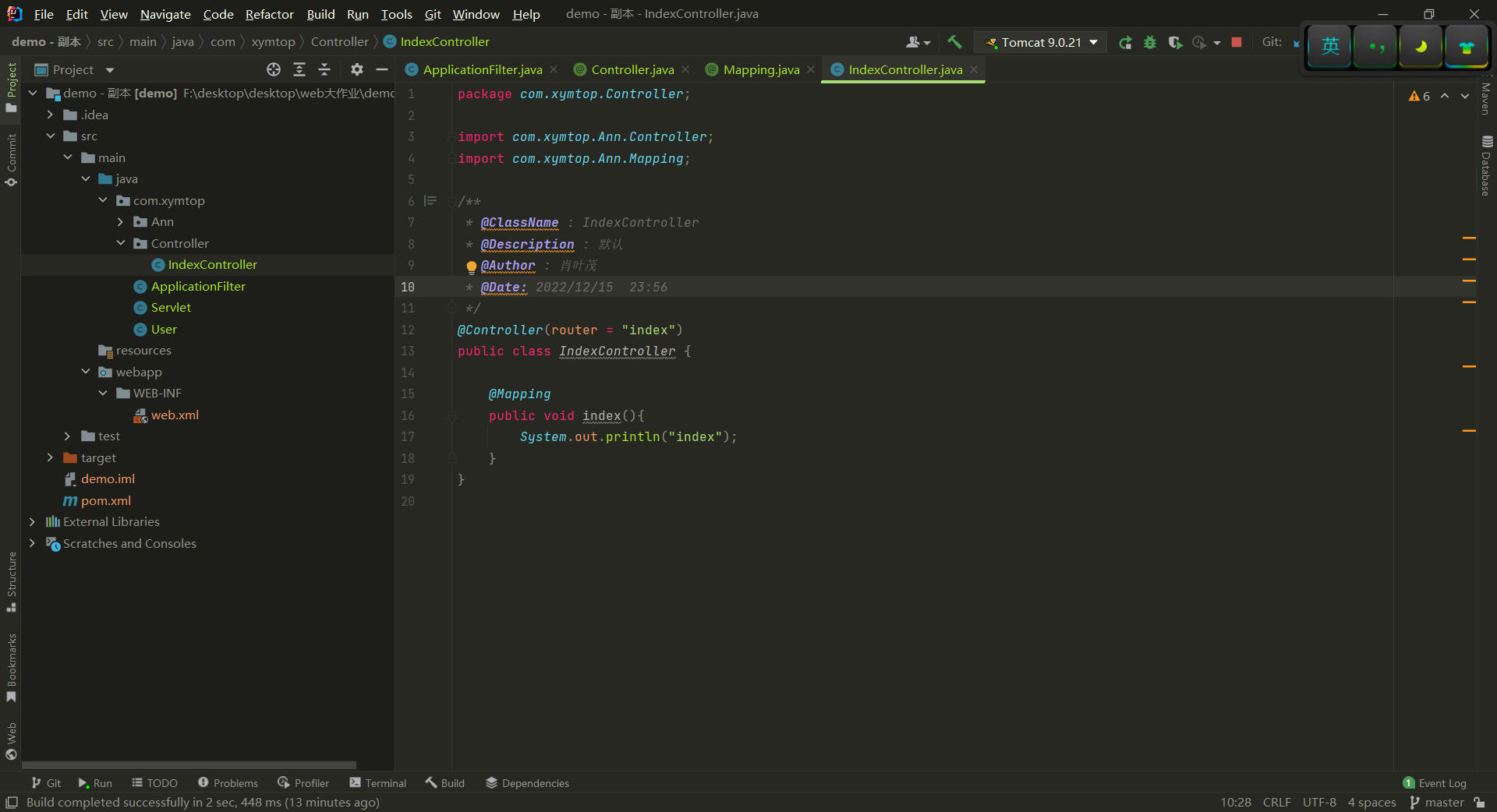


我们定义了两个注解，我们可以分析得到，为了实现最基础的功能，我们需要得到用户定义的参数，比如请求的地址是什么，其他的内容，比如需要调用哪个方法，可以通过扫描包来实现。为了我们开发更加方便，我们需要去开发几个注解，分别是设置映射类的请求地址和映射方法的请求地址，我们给他们命名为@Controller和@Mapping。





下面我们可以尝试新建控制器并且添加注解。但是现在运行并没有什么用，因为我们还没有添加注解的执行函数。



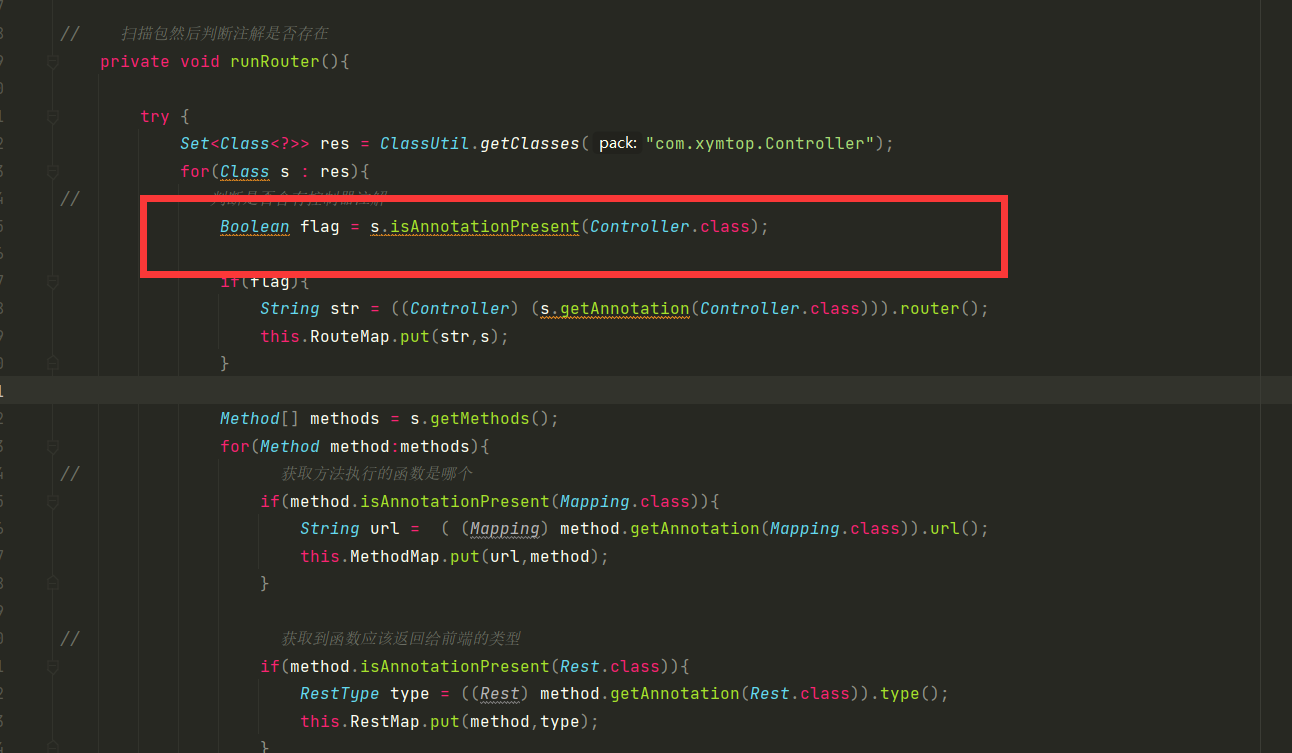
在写执行函数时，我们思考，我们怎么使用这些注解来生成我们刚才定义的两个map。

1. 扫描包，获取到包下面所有的类
2. 判断类是否具有@Controller注解，有的话类控制的map加一
3. 循环获取类下面的函数
4. 判断函数下面是否有@Mapping注解，有的话函数控制的map加一

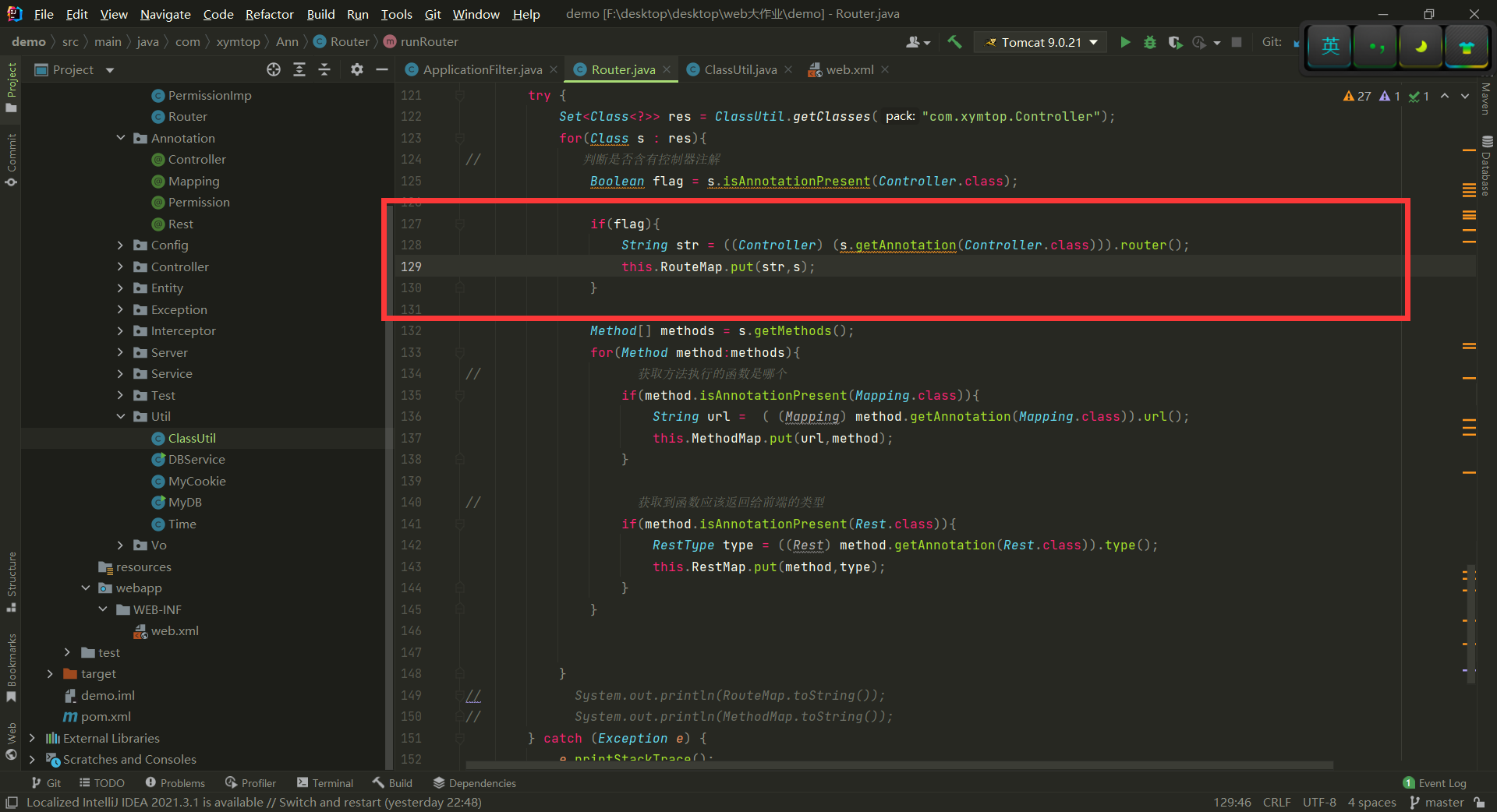
首先，第一步，扫描包，目前扫描包的方式很多，可以自己去选择一种，我这边给出我自己用的一种扫描的方法。



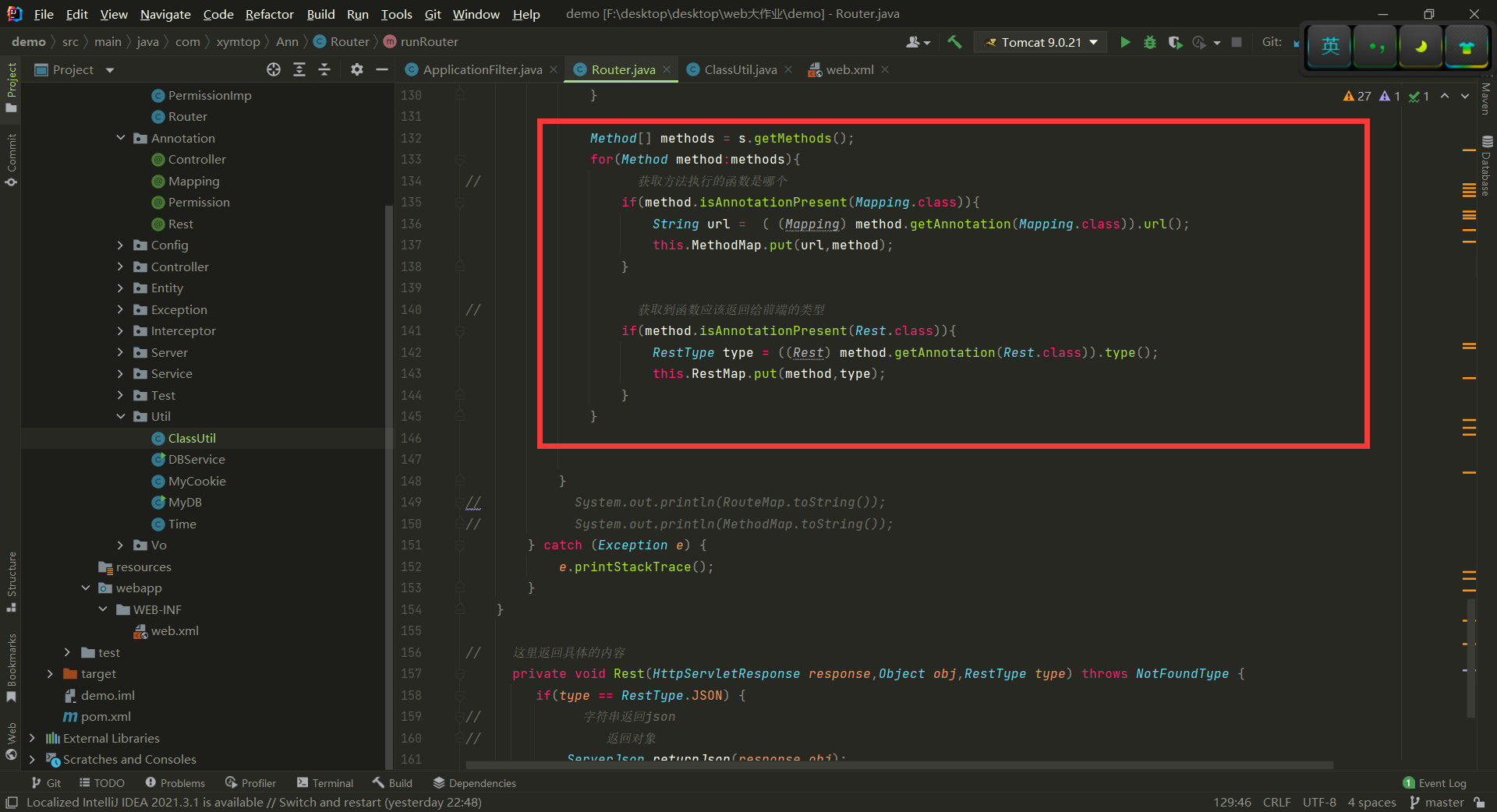
第二步，判断是否有某个注解，只需要用isAnnotationPresent即可，比较简单。



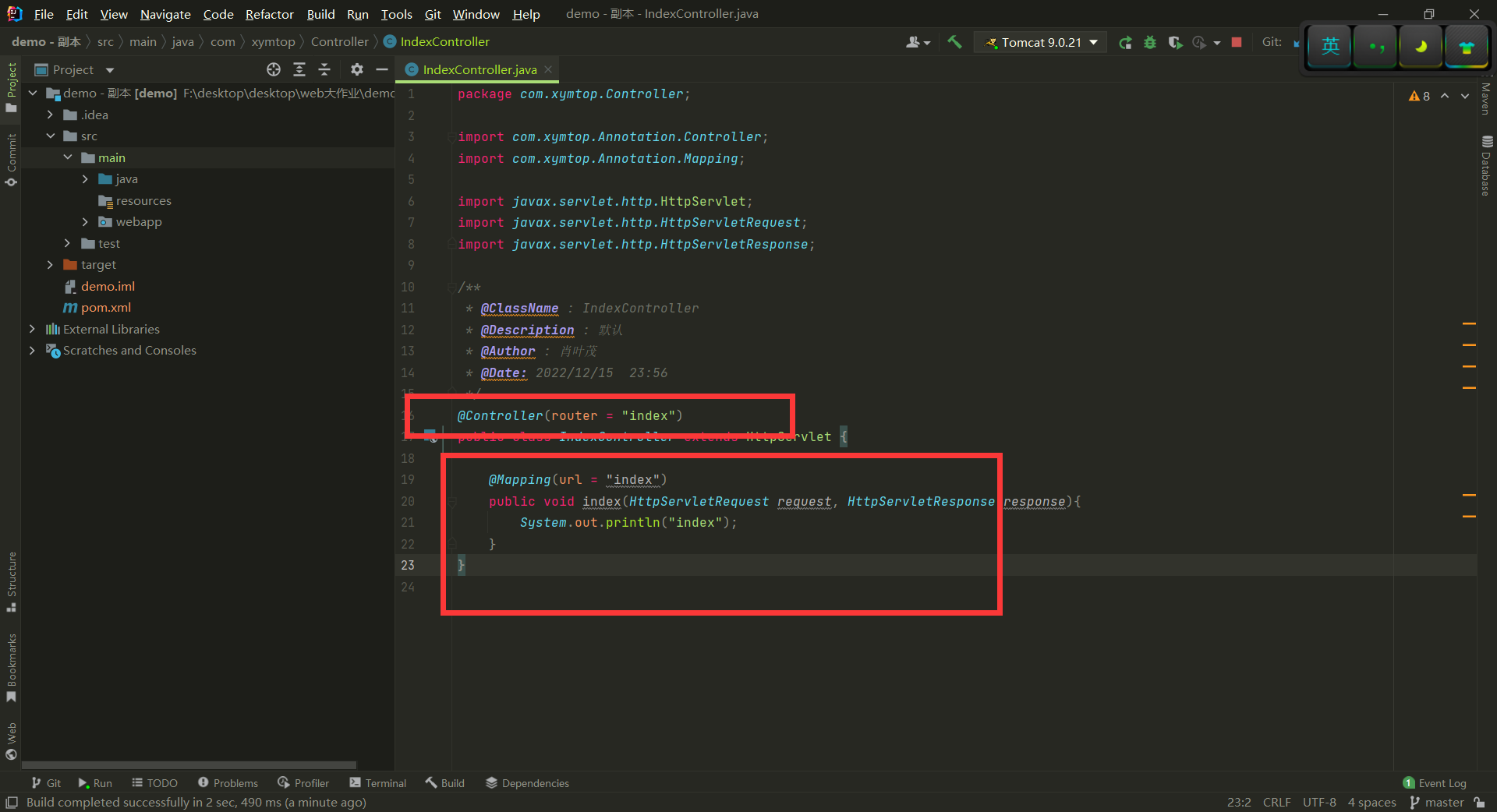
然后，如果有这个注解，我们需要将该注解的值，也就是用户定义的请求地址和该类进行绑定。绑定的话直接在map里面put一个值即可。



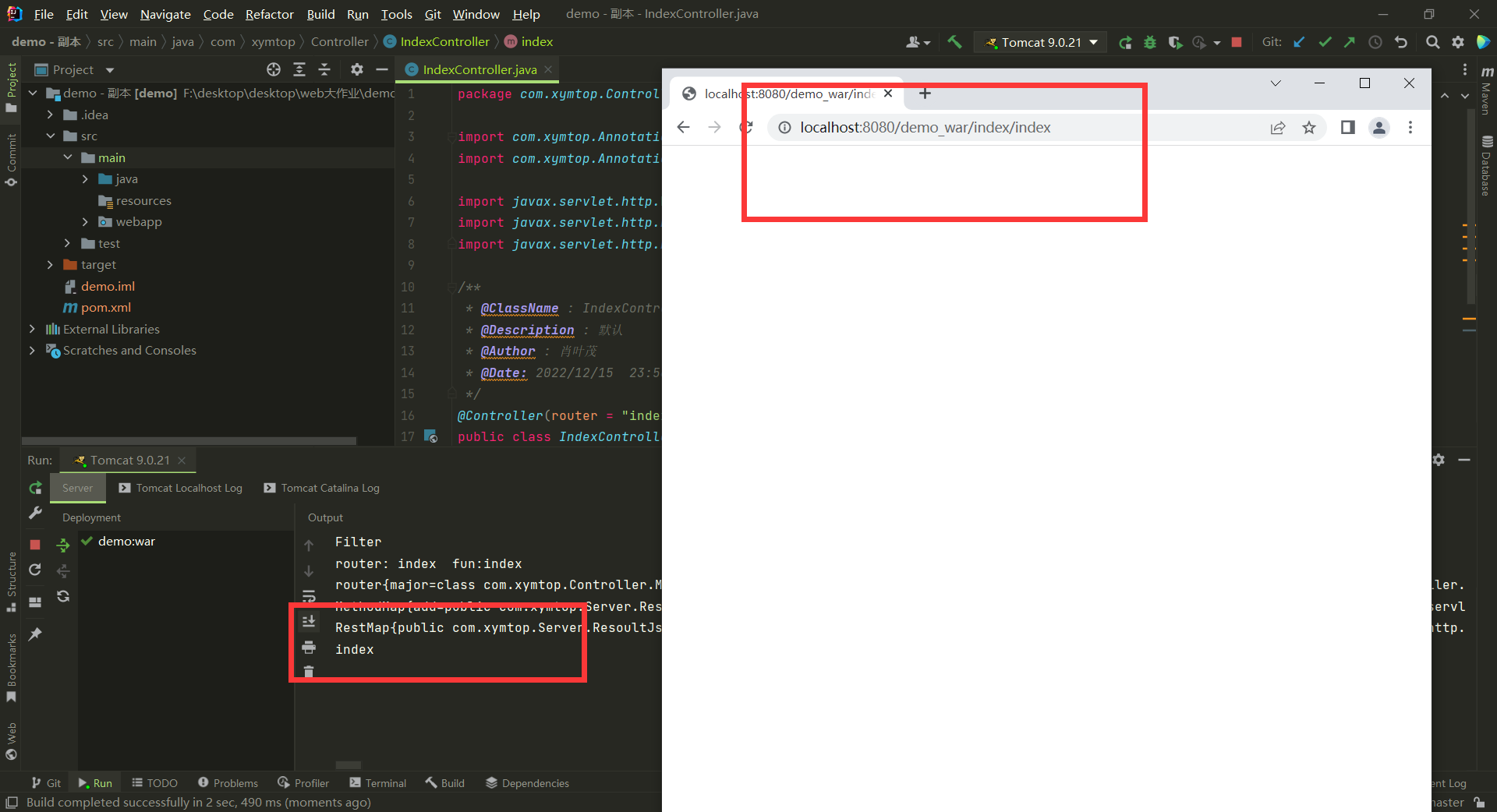
判断方法否具有@Mapping的内容和判断类是否有注解的思想差不多，但是需要注意的一点是：方法需要先被获取到，有可能一个类有很多个方法，所以我们需要遍历判断并且添加到map。



下面我们来编写一个控制器并且给它注解。

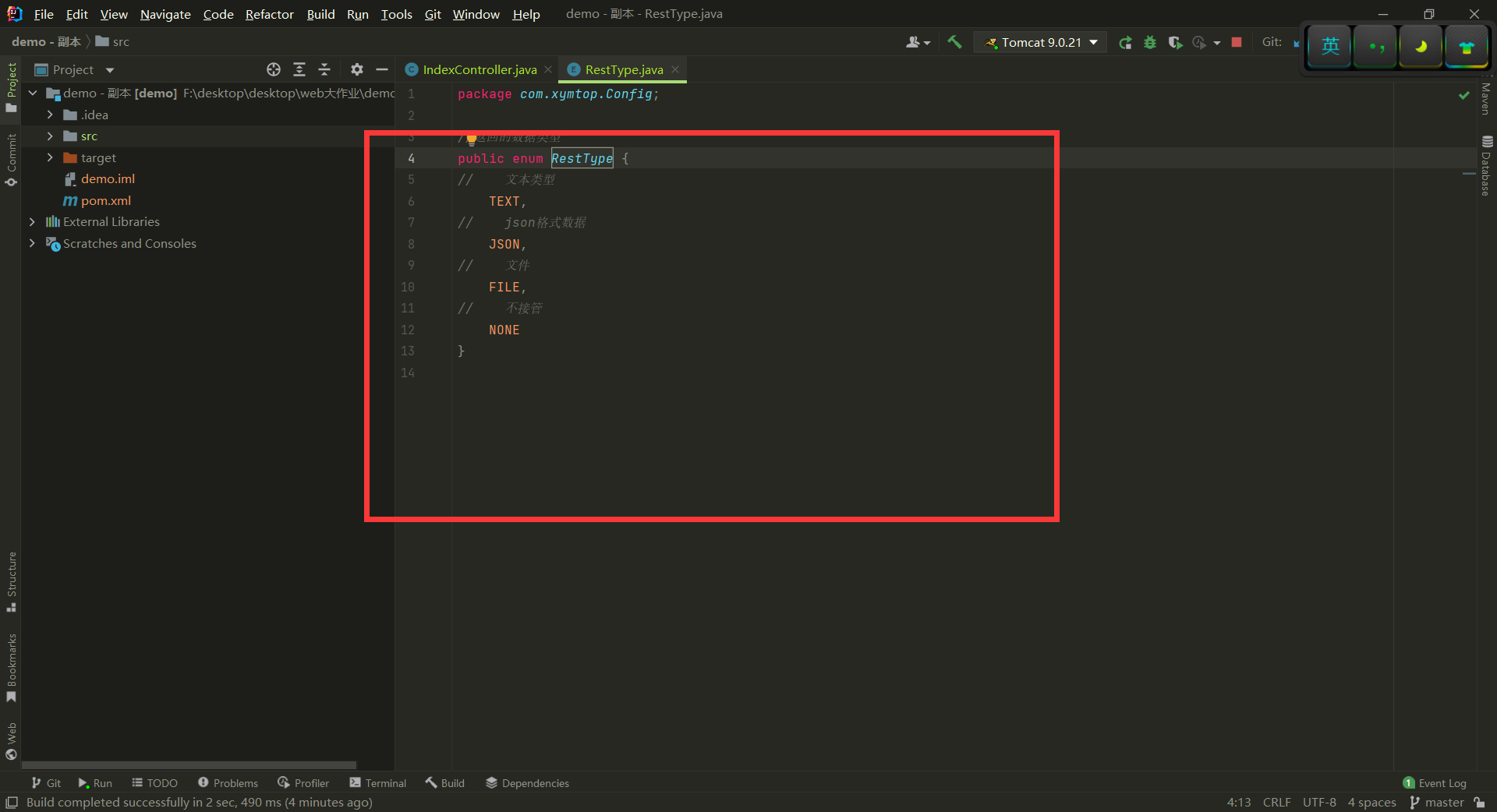


可以看到，它很好的运行了。



好的，现在我们已经可以进行正常的操作了，但是，我们在获得请求后，需要进行一定的处理，然后返回数据给前端，这个数据是什么呢，是字符串类型，JSON格式的数据，或者文件类型的数据,能不能在函数上加个注解就直接确定返回的类型那?

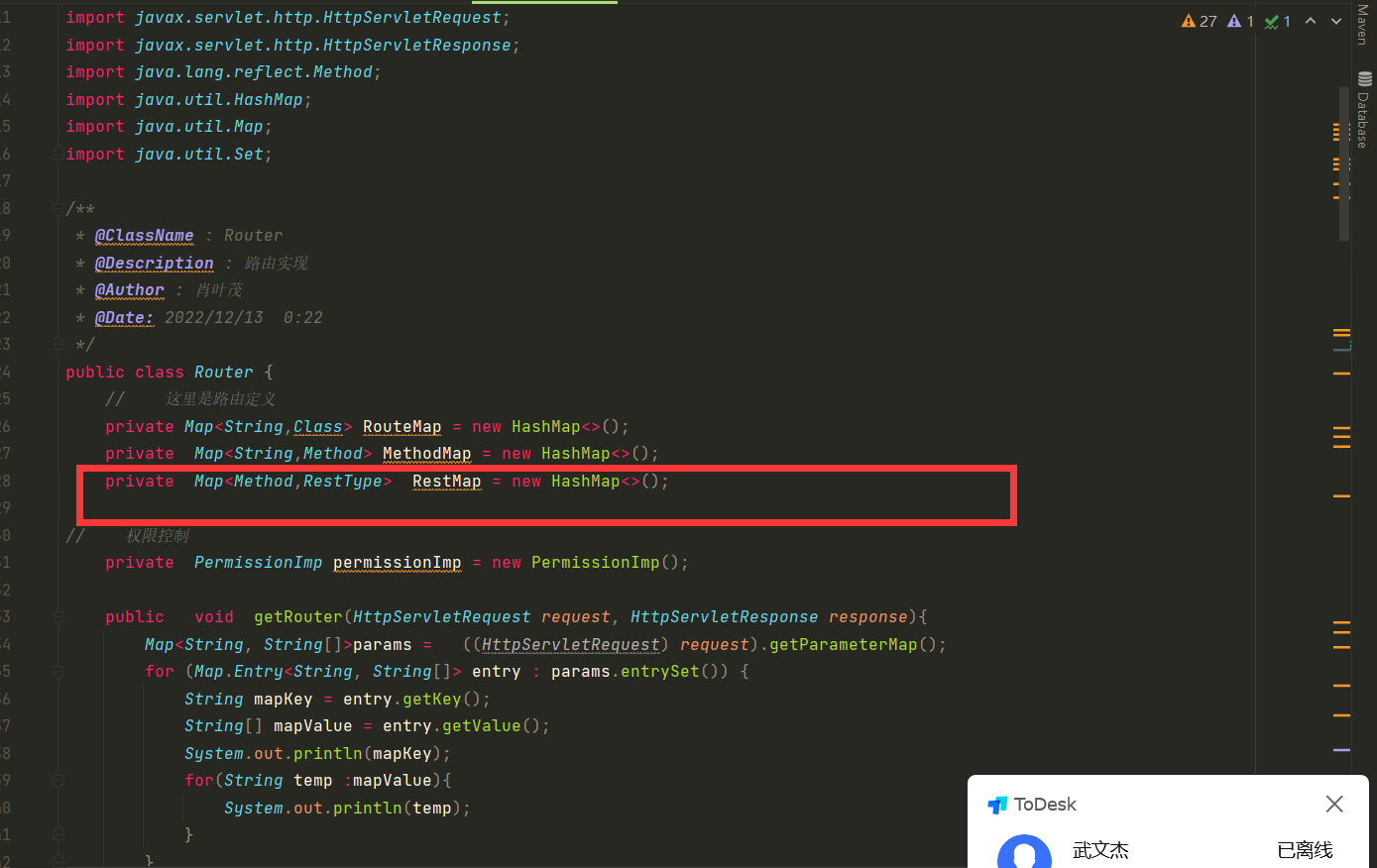
下面，我们枚举我们可能返回的类型。



下面编写返回类型的注解。



下面我们开始正常的扫包的步骤，维护一个map集合，对应这每个方法和这个方法的返回值。



如果没有参数，我们就直接提醒，如果有返回值，我们就把这些信息封装到另外一个方法中。



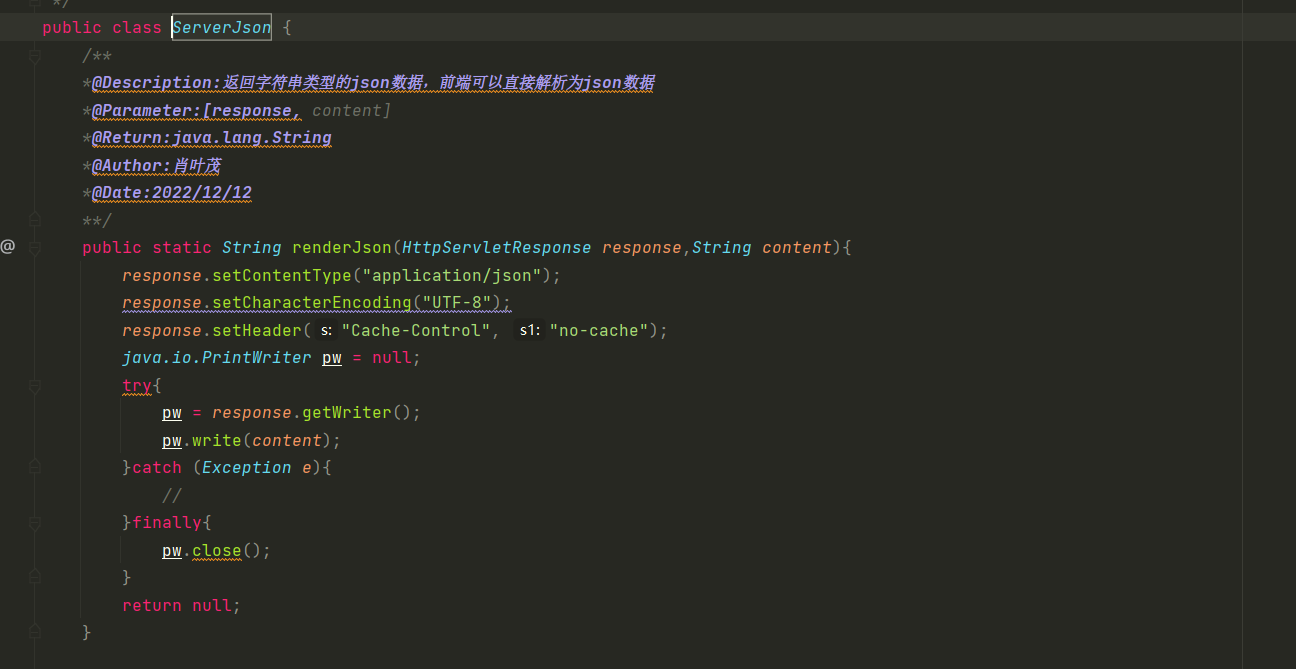
这个方法可以按类别来处理各种数据。



基于上述，我们还需要可以把对象转为json格式的一个方法。



封装为json格式后，我们仍然需要告诉浏览器，我们返回的是json格式。



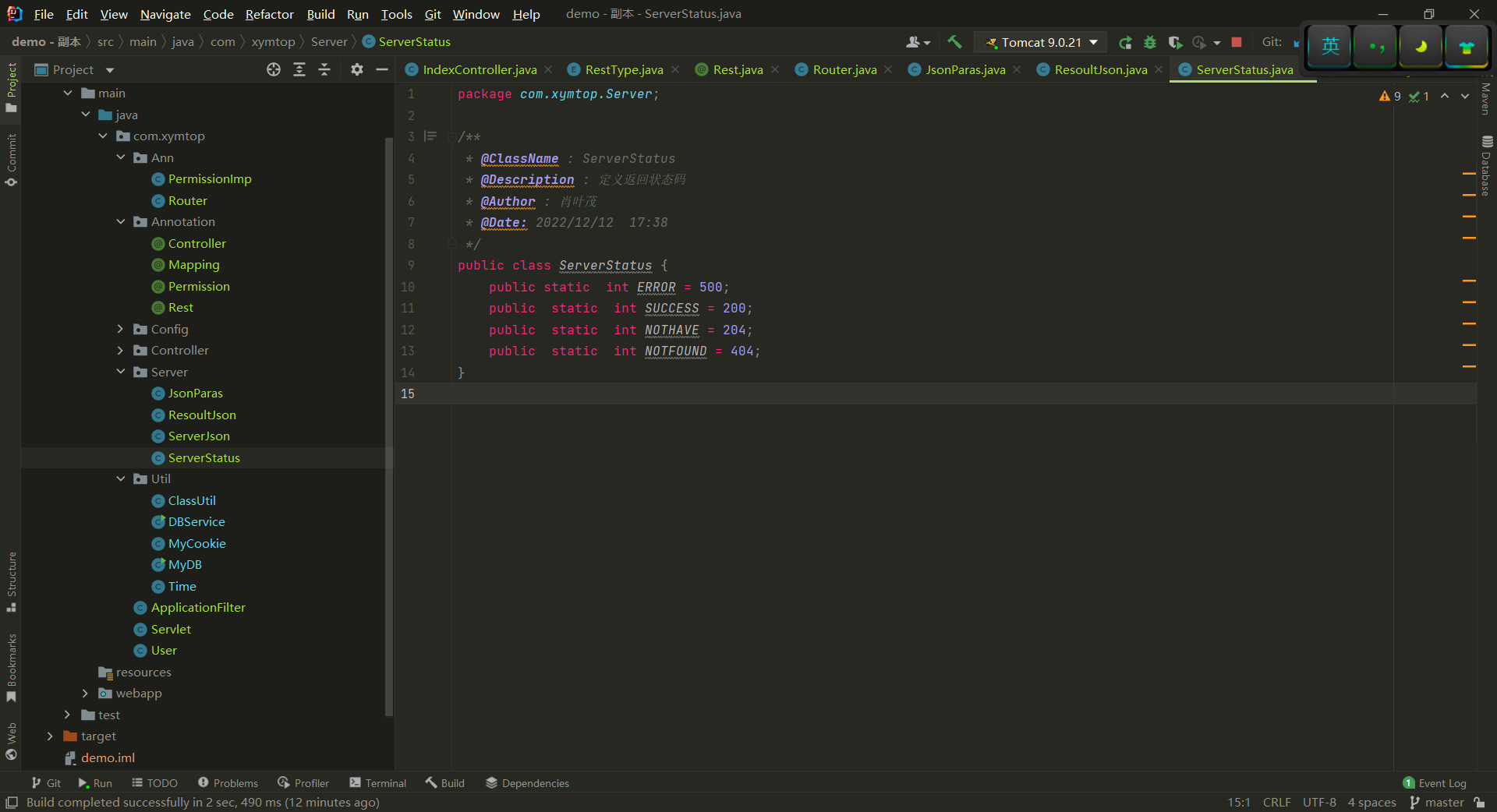
类似的，我们可以返回文本类型。



此外，我们可以封装一个类来规范化我们的数据返回格式。



服务器状态码也可以自己定义



我们可以写一个小接口来看看效果。

