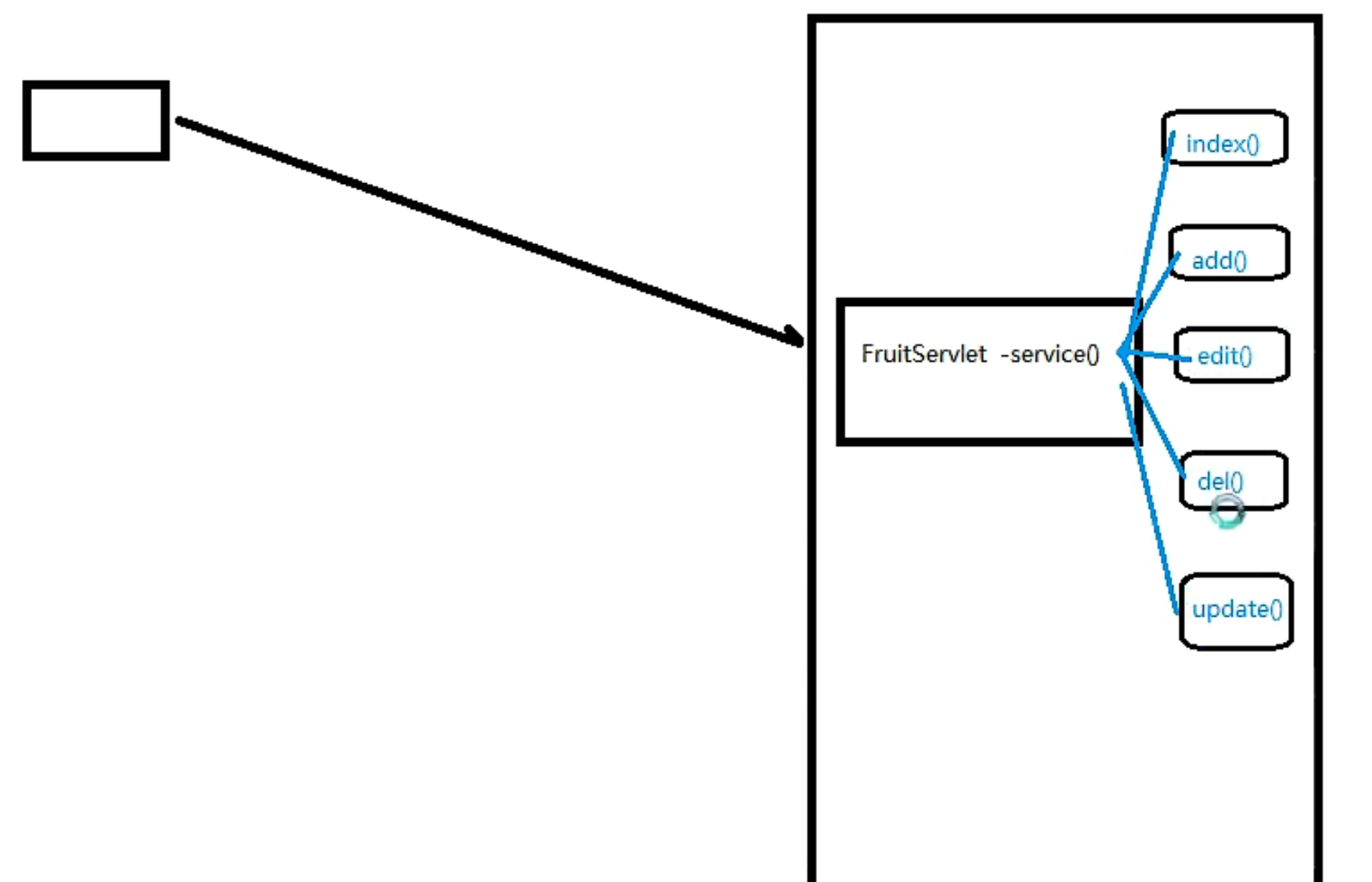
MVC模式

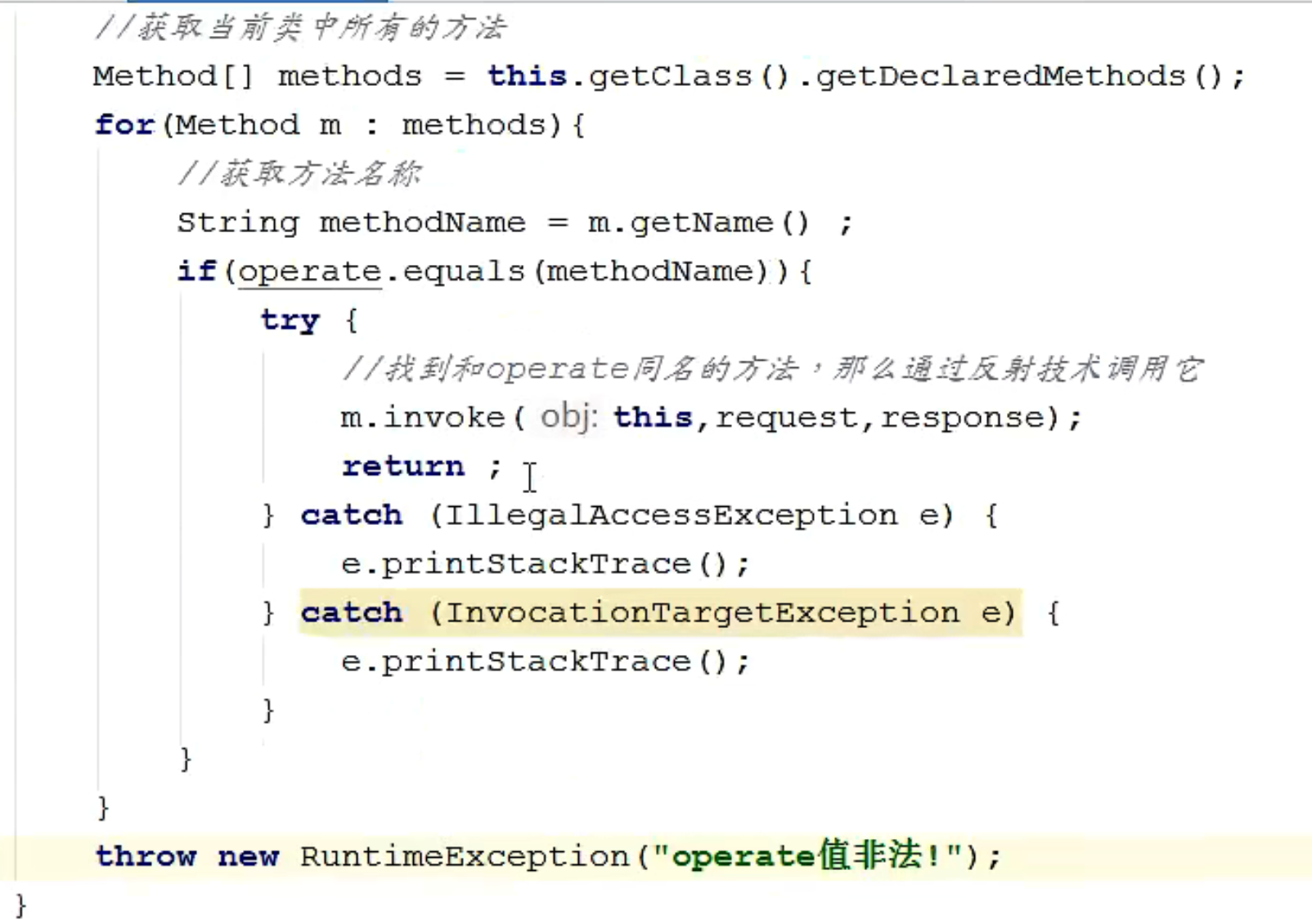
## 优化一



即，无论增删查改在所有跳转到Servlet的表单，input，onclick中新加入一个参数operate，然后都跳转到fruitServlet，然后获取operate参数值，switch，根据值调用Fruitservlet中对应的方法，这些方法原来都是独立的Servlet，现在合并到一个fruitservlet中，这就是mvc优化。

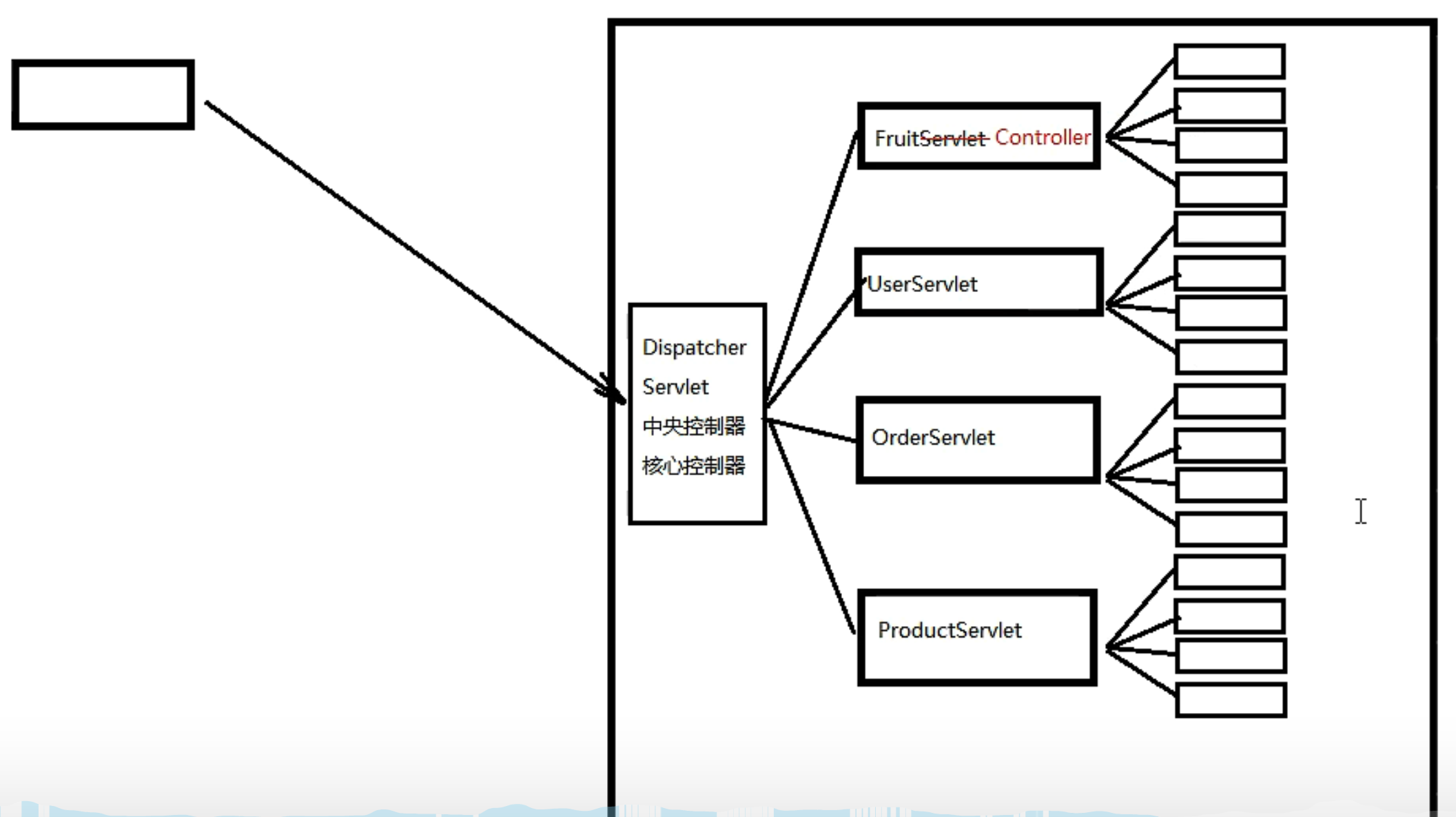
## 优化二

方法比较多时，switch臃肿，可以使用反射动态获取当前类的所有方法数组，然后for循环，将operate.equals（数组每个方法名字），如果匹配，则使用反射调用数组中此方法。



## 优化三

### 添加调度者Servlet

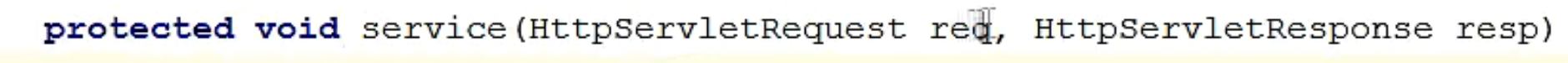


如果每个大的模块Servlet过多，那么我们还是需要在每个Servlet中都写反射匹配。不如就在最外面再加一个调度者Servlet也叫核心控制器。

### 调度者中截取url

DispatcherServlet之上，@WebServlet（“\*.do”）表示拦截所有以.do结尾的请求，注意此处不可加/

然后重写另一个doget dopost方法

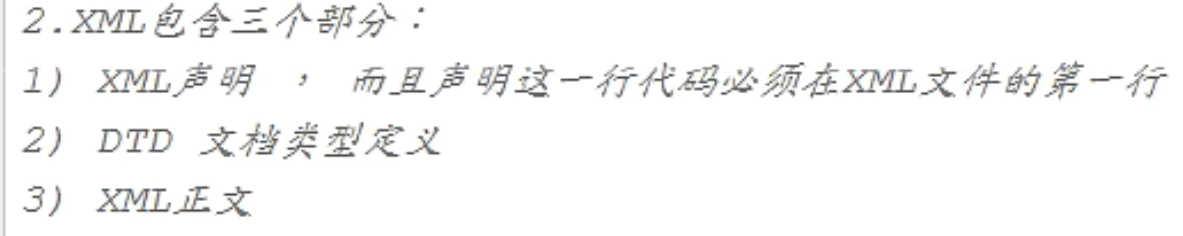


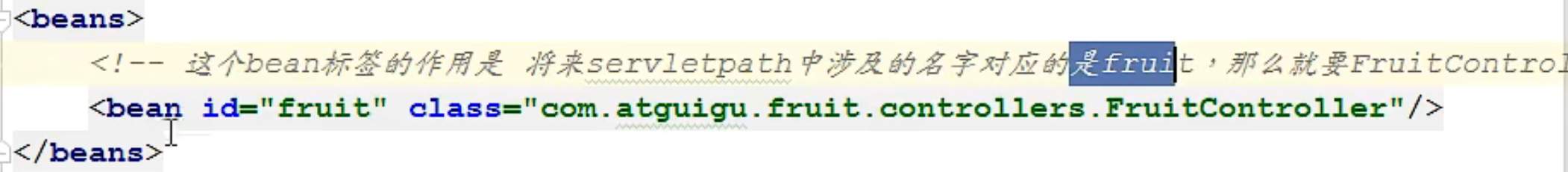
在此方法内，利用req的getServletPatch方法获取用户url是什么，然后字符串截取/之后，.do之前的部分，根据这部分再匹配对应的Controller。

此时的Controller就是原本的各个Servlet，但是去掉了@WebServlet注释，依然继承于ViewBaseServlet。使用其内部的渲染方法。

如何将从url截取的字符串与Controller对应呢？需要在配置文件中说明，此时在模块的src目录下新建applicationContext.xml（意思是应用上下文）文件

什么是xml文件？xml叫做可扩展的标记语言，html叫做超文本标记语言，html是xml的子集。可扩展意思就是标签随便起名，<jakbfwef>都可以是一个标签。但是要在<beans></beans>标签中声明。





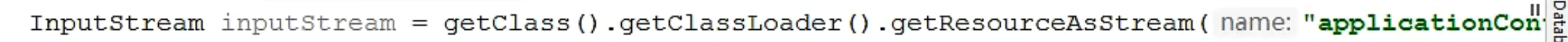
即在beans标签中含有一个个的bean标签，每个标签都有一个id，这个id就是我们截取的url字符串，这个class就是url对应的Controller

### 调度者servlet中init方法中解析xml

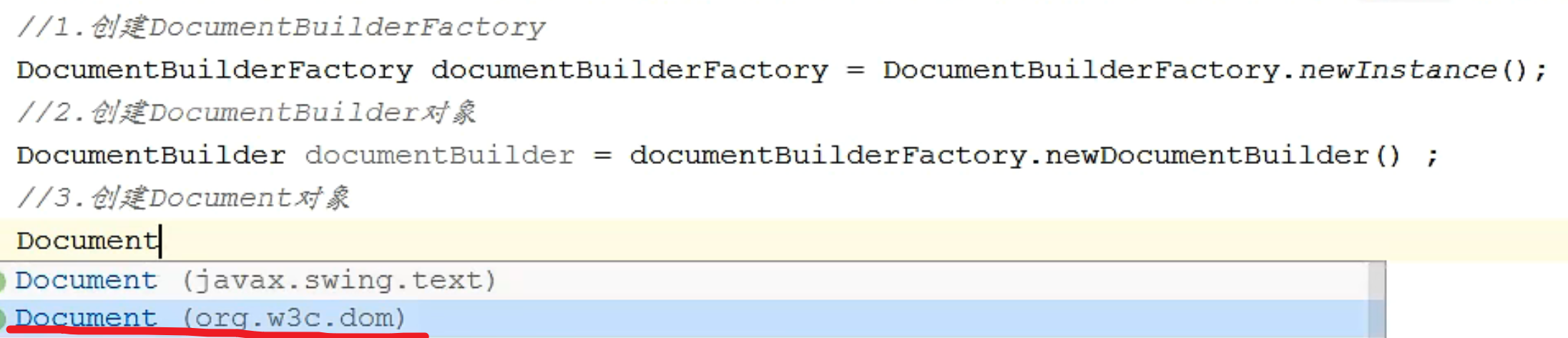
那么如何使用xml呢？因为Servlet有实例化初始化服务销毁四个阶段，所以我们在调度者Servlet的初始化方法init（）中解析xml。

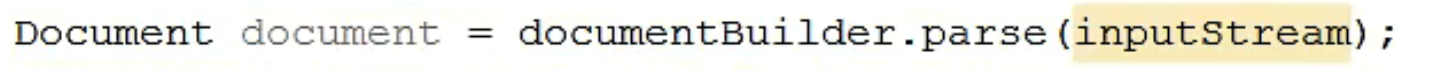
init方法中利用反射（反射获取德鲁伊差不多），加载xml文件

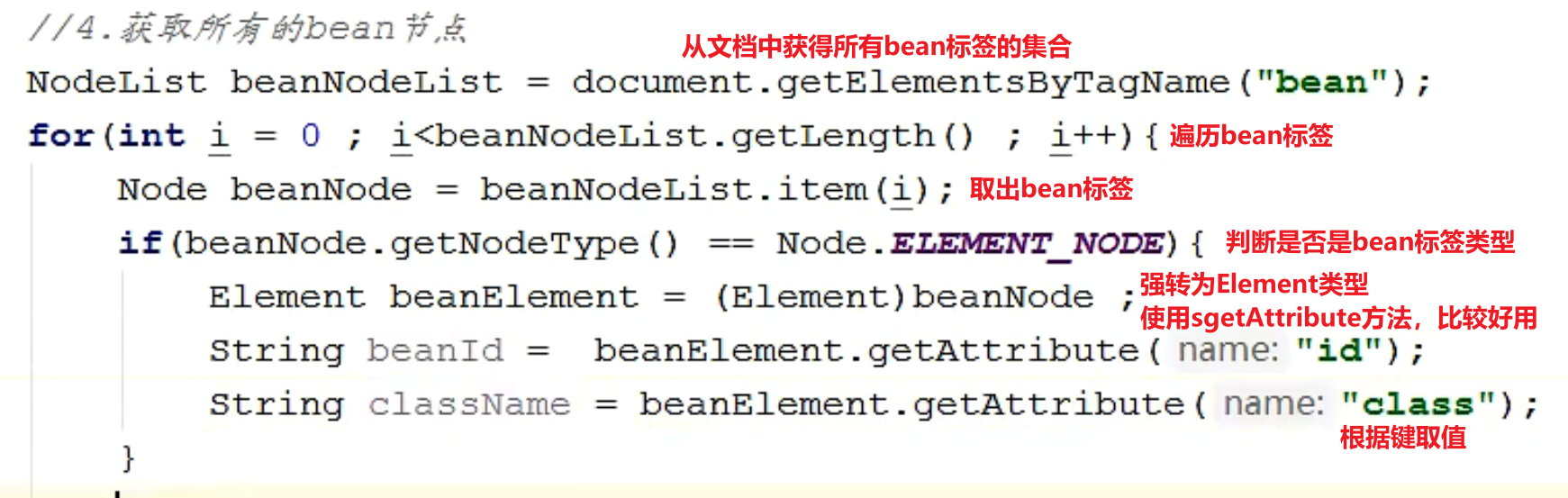
先类加载器反射生成一个输入流对象



然后原本的properties对象在这里是document（文档）对象，注意导包是w3c的包





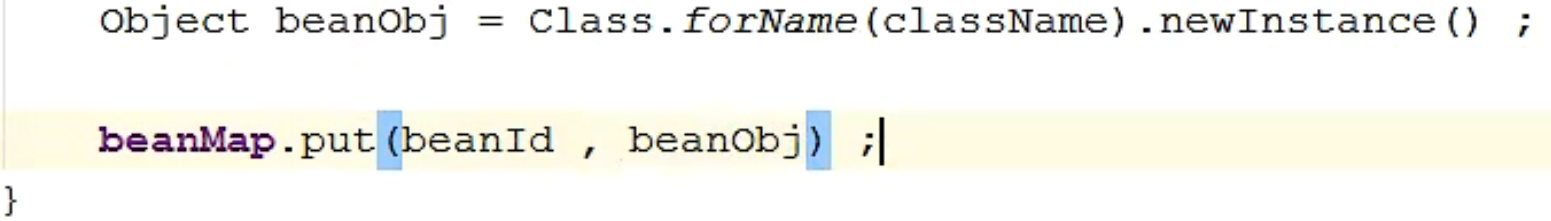


到这里我们想将每个id和其对应的class存储起来，所以我们需要一个Map集合。

但是class在这里是一个Controller的路径，我们想将他变成对应的Controller对象，既然路径有了我们可以使用反射，Class.foName(className).newInstance这样返回的就是Controller的对象。

然后在调度者Servlet中定义一个Map，将id和Controller对象以键值对的形式put进去

再添加两行：



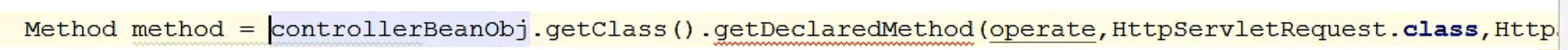
### 调度者中调用controller中方法

经过上述过程，在service中截取url，然后在Map中就能根据截取的url取出对应的Controller对象。

至此，先将document封装起来，然后根据用户的url，实例化对应的Controller对象，那么我们还需要根据用户req的operate值，来调用Controller对象的对应方法。

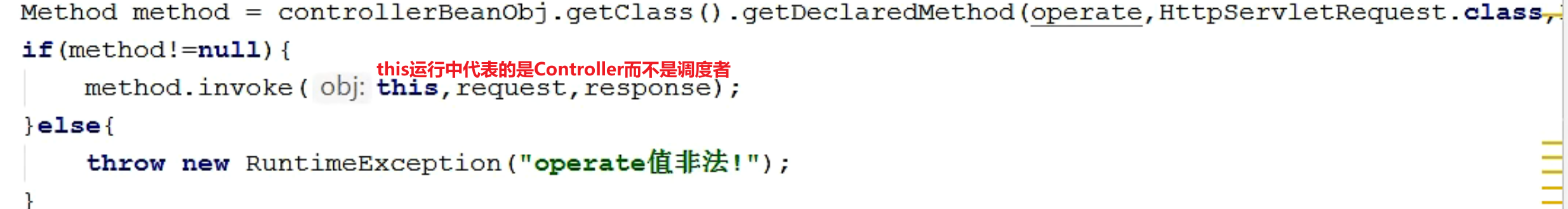
那么将原来的Servlet中反射获取operate调用Servlet方法的这个方法提取出来，放到调度者Servlet中。并做修改：

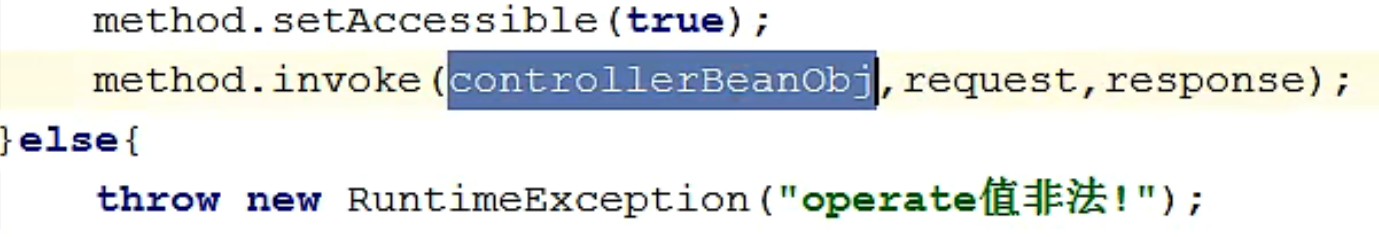
如果operate为null说明第一次访问页面，则给operate赋值index。如果不是null，则我们需要调用operate与Map中对应的Controller对象中反射获取的对应方法。此处就不使用for循环了。



注意除了方法名参数，其他参数要传Class类型。因为反射获取方法，反射生成方法的过程中需要生成形参的类型才对。注意此时还没调用方法呢，只是反射获取。

然后就是invoke调用方法，将哪个对象（Controller），方法形参req，resp传进去。





# 优化步骤123小结

1. 调度者Servletinit方法中加载xml配置文件，将url截取字符串与Controller对象封装到Map
2. 调度者的服务方法中，截取url，并从Map取出对应的Controller对象。
3. 从Controller反射获取的方法中，根据req.getParamate(“operate”)的值，调用对应的方法。

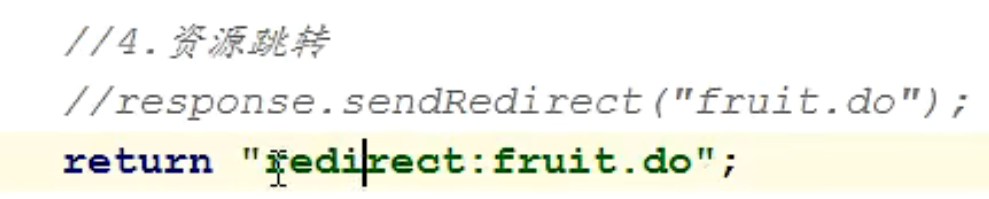
Controller到此彻底沦为一个只含有方法的类。

对于用户请求的响应，至此只有一个调度者Servlet。

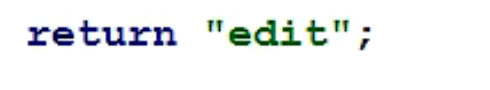


# 优化4

每个Controller中，最后都要进行重定向，注释掉以后，每个Controller中的每个方法都返回一个字符串 如 返回重定向的字符串

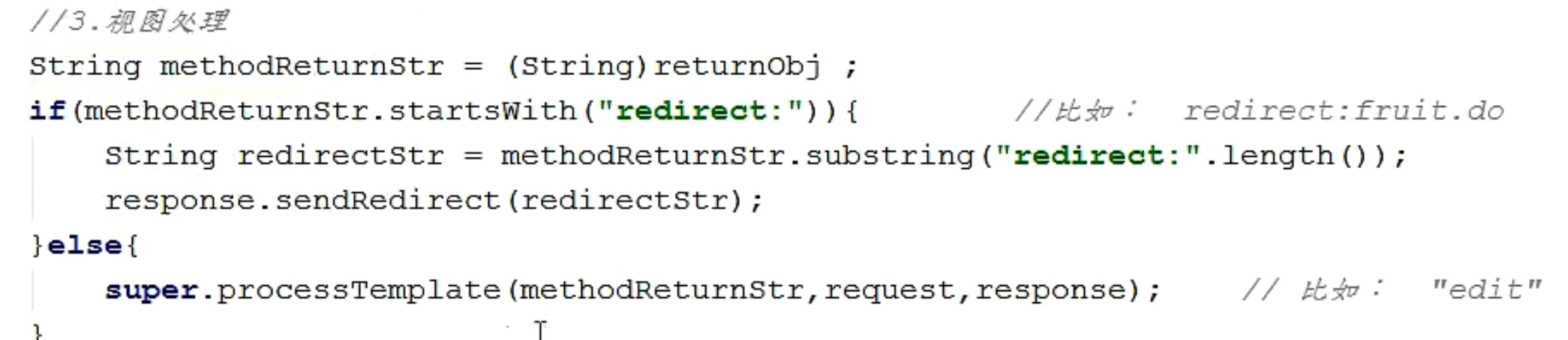


当然Controller中过的方法还可能要渲染模板，原来的processTemplete（edit,req,resp）可以改成：



所以实际上Controller中的各个方法形参除了req是必须，resp是可以去掉的。因为req还是要用来getParameter值的，而resp就是用来做重定向和渲染模板的，结果都不用了，都在调度者中完成就可。

然后在调度者中，反射调用Controller中方法时，接收此返回值，如



注意调度者init方法中还要调用一下父类的init方法



# 优化5

每个Controller方法中都要使用req.getParameter获取参数，当然这也是为啥在上一个优化中方法形参中req保留，resp可以去掉的原因。

在此，一些方法的req则可以去掉，方法形参改为每个方法中实际所需要的形参，如

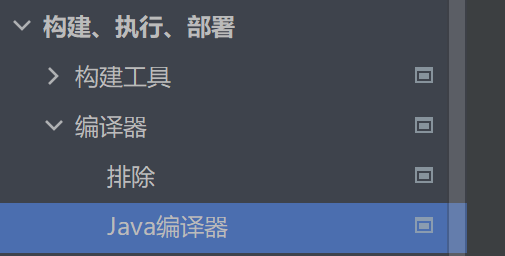


然后Controller方法中就不需要再使用req获取一大堆参数，而是直接使用方法形参。

那么当然是调度者反射实例化Controller中方法的时候统一形成这些参数，然后反射调用方法的时候传进去，那么如果知道这些方法形参想要什么呢？还是要使用反射获取形这些方法的参列表。

所以调度者得到Controller实例以后，直接得到所有方法实例，然后for循环每个方法，反射得到每个方法中的形参数组。并且再嵌套for循环每个形参数组，得到形参名，然后使用调度者的req.getParameter(形参名)。这样就从调度者req中得到了要调用的反射获取的Controller中某个方法的所有形参值。然后使用一个**Object**数组将方法的形参存储起来（形参可能是字符串，数值和req）。

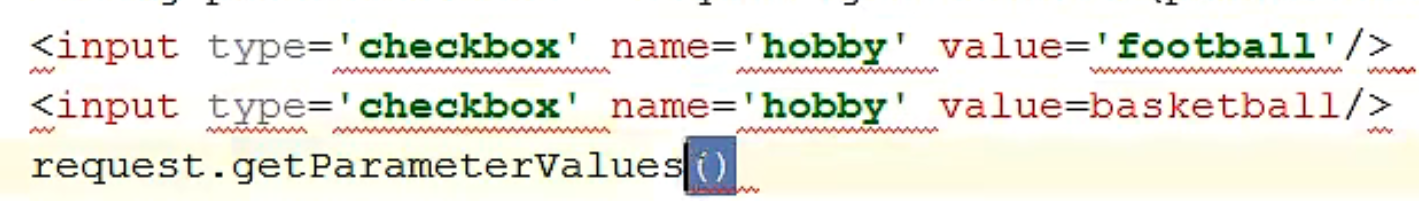
默认情况下反射获取的形参列表是没有具体名字的，只能获取args0，args1和形参类型

这样就可以了获取形参名字了。注意重新编译一下删掉out文件夹中，重新部署一下。

这样args0就变成真实的形参名称了

如果Controller中方法的形参是 复选框的值，即形参实际上是hobby数组，数组名叫hobby

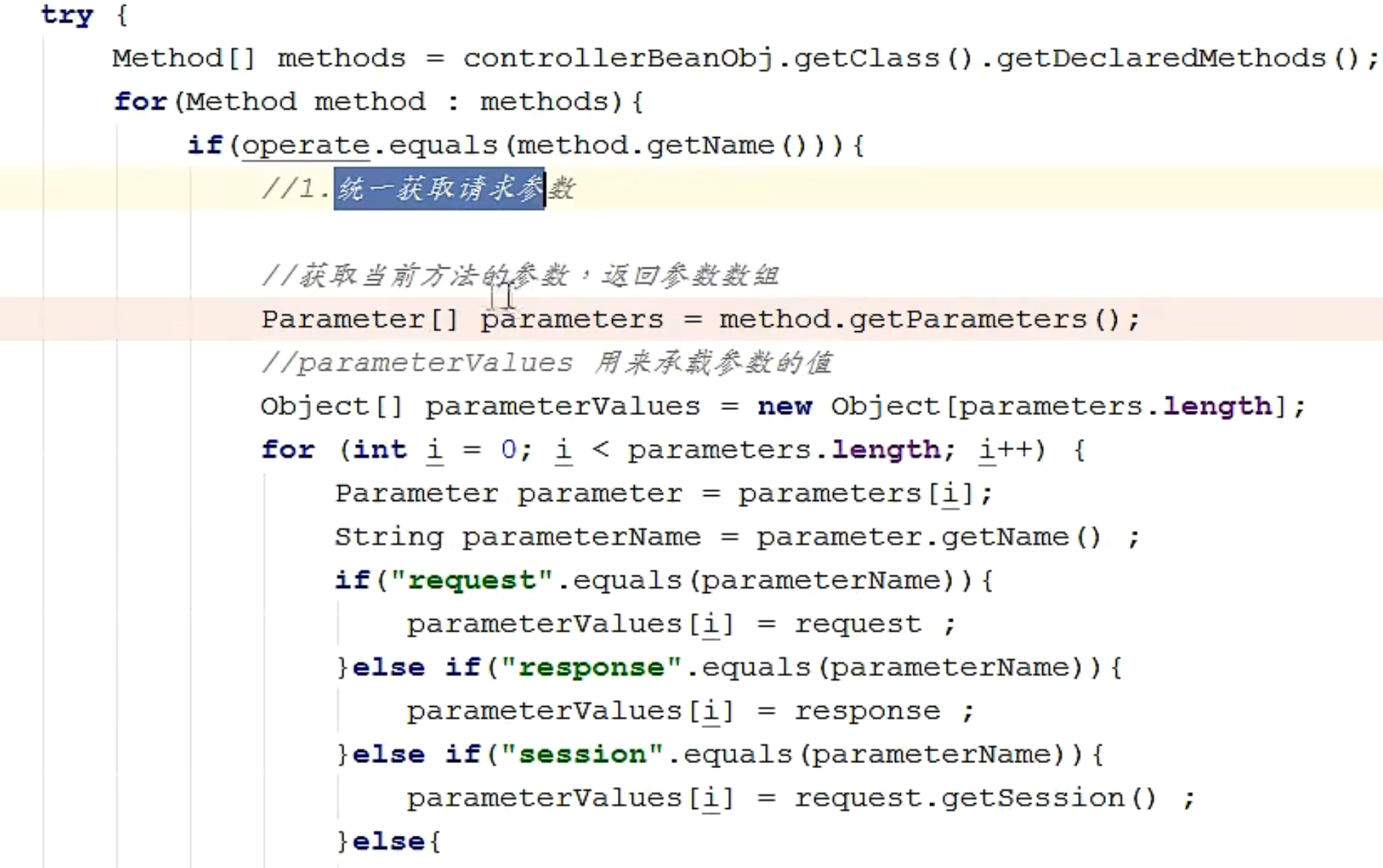


则在调度者中反射拿到数组名hobby后，使用req.getParametererValues（“hobby“）从req中得到hobby数组。

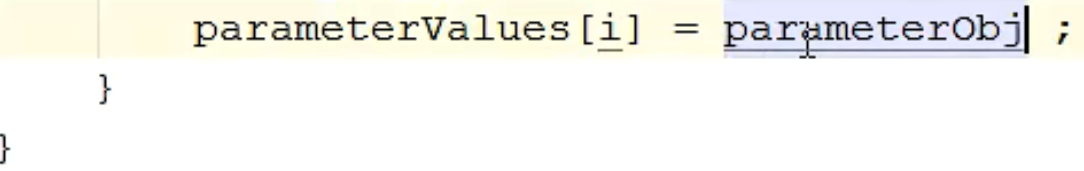
那么如果获取形参时，形参是req呢？那就不需要再getParameter（req）了，这样也得不到。

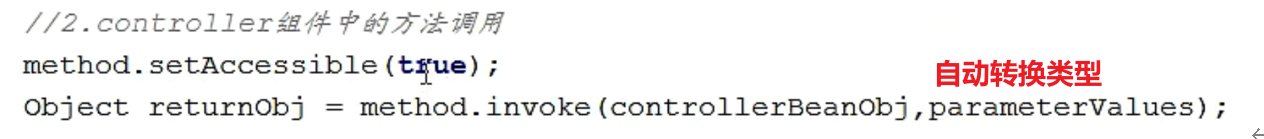
那么就需要我们做一个特殊判断即可，如果遍历形参列表时，形参名是req，则将调度者中的req直接赋值给Object数组即可。如果是resp，则将调度者的resp赋值给形参数组。如果是session则将调度者中req.getSession赋值给形参数组。

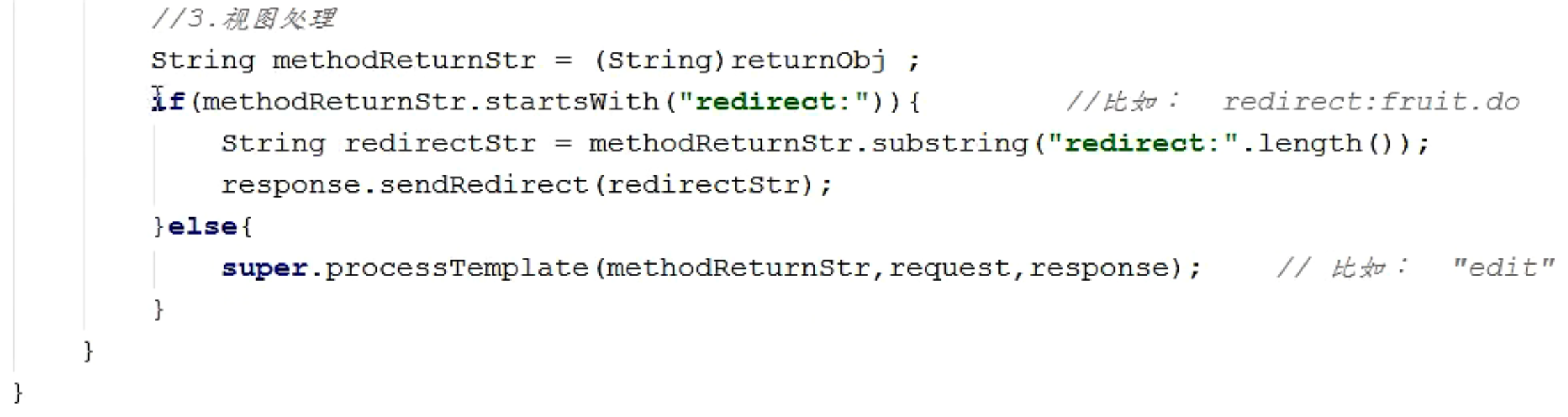
# 调度者中service方法小结











至此为止，调度者中init方法负责将Controller对象和url字符串存储到Map数组中。

而service方法则是根据用户再次发来的url中字符串取出Map数组的Controller对象，然后根据新的url中？后operate的值，反射得到对应的方法，且进一步获得形参名，形参类型，根据形参名和类型从调度者req中getParameter（形参名）获取这些形参值。

然后再invoke operate对应的方法，形参自然是哪个对象的，和形参列表。其中形参列表可以传数组进去，并且可以自动将Object类型的数组中元素转为每个方法形参的类型。

然后再根据invoke方法的返回值，做内部转发还是重定向还是渲染。