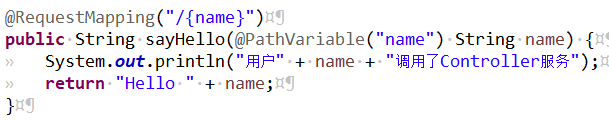
# Spring Boot单元测试（Mock）

单元测试的重要性就不多说了，我这边的工程一般都是Spring Boot+Mybatis（详情可参看《Spring boot+MyBatis+PageHelper+JSON》），现在写一下Spring Boot下怎么测试Controller、Service。

## Controller测试

本文就简单一点，写一个Hello接口。



### HTTP请求测试

既然是测试Controller接口，肯定就是发送HTTP请求了，之前的文章我也有提到，可以使用Postman、Swagger进行测试，本文我们采用编码的方式测试，新建HttpRequestTest类：



注意三个红框，这是告诉Spring Boot启动的时候采用一个随机的端口，有助于在测试环境中避免冲突（官网解释）。

绿框的restTemplate，看过之前我的有关Spring Cloud负载均衡Ribbon的文章就知道，与RestTemplate差不多，这就是个HTTP客户端。

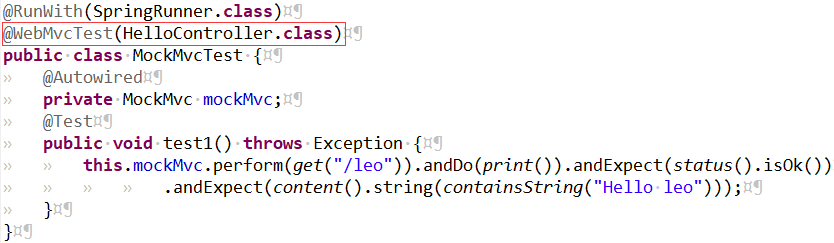
运行测试通过。注意看日志，会找到一行日志：

Tomcat started on port(s):XXXXX

这说明整个Tomcat已经启动成功了。

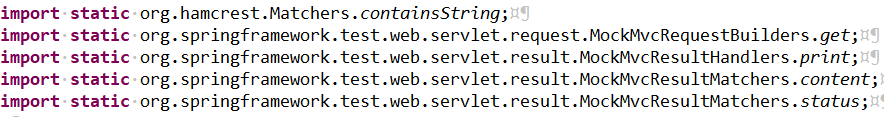
### MockMVC测试

注意上面的测试，通过日志我们知道，其实就是启动了Tomcat，然后通过TestRestTemplate构建了HTTP请求。但是在我们实际开发中，这有一个现实的问题，我们一个工程会有很多的Controller、Service、Mapper，但是我们本次测试可能就测一个接口。为了一个接口，要启动整个Tomcat，太浪费了。所以我们可以采用MockMVC，在不起的服务的情况下，测试接口：



实际开发中，肯定会有很多Controller，红框内我们指定本次测试只实例化HellController这一个。

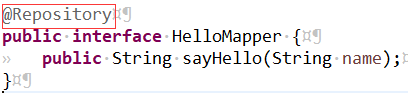
另外在开发的时候，Eclipse没法通过快捷键进行import static，所以我们要手工导入以下：



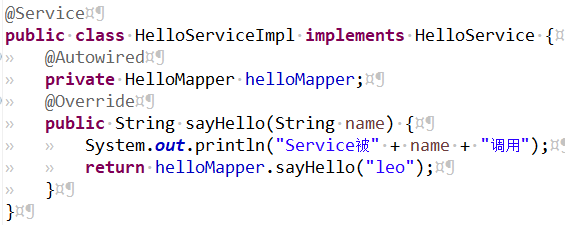
运行本测试，注意看日志，就不会出现Tomcat started on port(s):XXXXX的记录了。

## Service测试

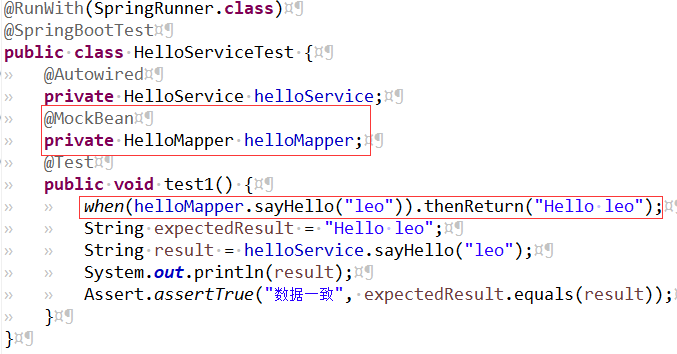
在实际开发中，Service的逻辑处理比较多，应该是最先被测试的，本示例增加一个Service和Mapper，如下：



在我之前那个完整的工程中，这个注解应该是@Mapper，本例我没引入数据库，只是模拟一下，所以用了@Repository。



在@Service下应该还有事务注解@Transactional，本例只是模拟，所以也没有加。



注意红框，在实际开发中，我们自己写的Controller、Service很可能去调用别的同事或别的项目组写的Service、Mapper，对方可能只写了一个接口，没有实现（比如本例中我写的Mapper），这样式没法进行测试的。

Mock的作用就是创建一个虚拟的对象替代那些不易构造或不易获取的对象。

本例中我都没有引入数据库的相关依赖，本来应该是连接数据库的Mapper，现在就是一个单纯的接口，如果不对Mapper进行处理，这个测试就没法进行下去。

于是在第一个红框内Mock了一个HelloMapper，第二个红框其实就是对这个Mock对象的方法进行了处理，等于Mapper接收参数“leo”，返回结果“Hello leo”，给Service使用。

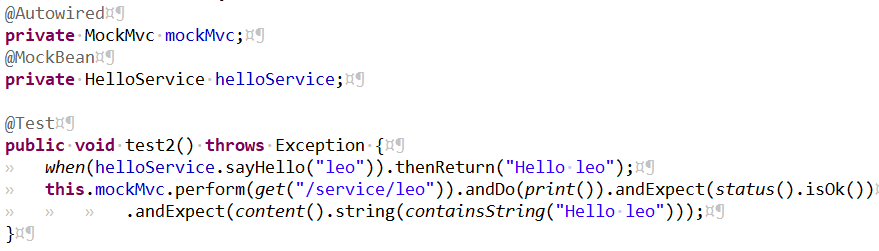
运行测试通过。

## Controller测试（含Service）

在“MockMVC测试”一节中，我们只测了Controller里的一个方法，没有引入Service，这在实际工作中是不可能的，我们要么引入自己写的Service，要么引入其他同事写的Service（接口形式），如果这时候引入的Service同样没有实现，这测试也没法做下去。结合“Service测试”的内容，我们可以这么做：



Controller新增一个方法，调用了Service。



测试类增加如上代码，Mock一个假的Service，用when来处理Service，用MockMVC模拟HTTP请求，测试通过。