

# 实验题目

---

【实验题目1， 必选】 ---> pythonProject

1. 使用原始socket()实现一个B/S架构的服务端；  
要求：从浏览器发送请求，服务器固定回复下面简单的HTML消息给浏览器显示；

```
html>
```

```
# this is test HTML
```

---

【实验题目2， 必选】 ----> pythonProject

2. 使用select方式实现一个C/S架构的应用；  
要求：服务器同时支持多个客户端连接，一个客户端发送消息，服务器回复一个XML消息（内容不限）；

【实验题目3， 必选， 3选1】 ----> lab3

3. 使用libevent在linux（或windows）下实现一个简单C/S应用系统，返回一个json格式数据，内容不限(C/C++);
4. 使用springboot实现一个支持RESTful接口的WEB应用服务器，回一个json格式数据，内容不限（java）；
5. 使用django实现一个支持RESTful接口的WEB应用服务器，回一个json格式数据，内容不限（python);  
要求：从浏览器发送消息，可以回显接收到的json数据；

【实验题目4， 高级加分项】 ----> LearnMQ

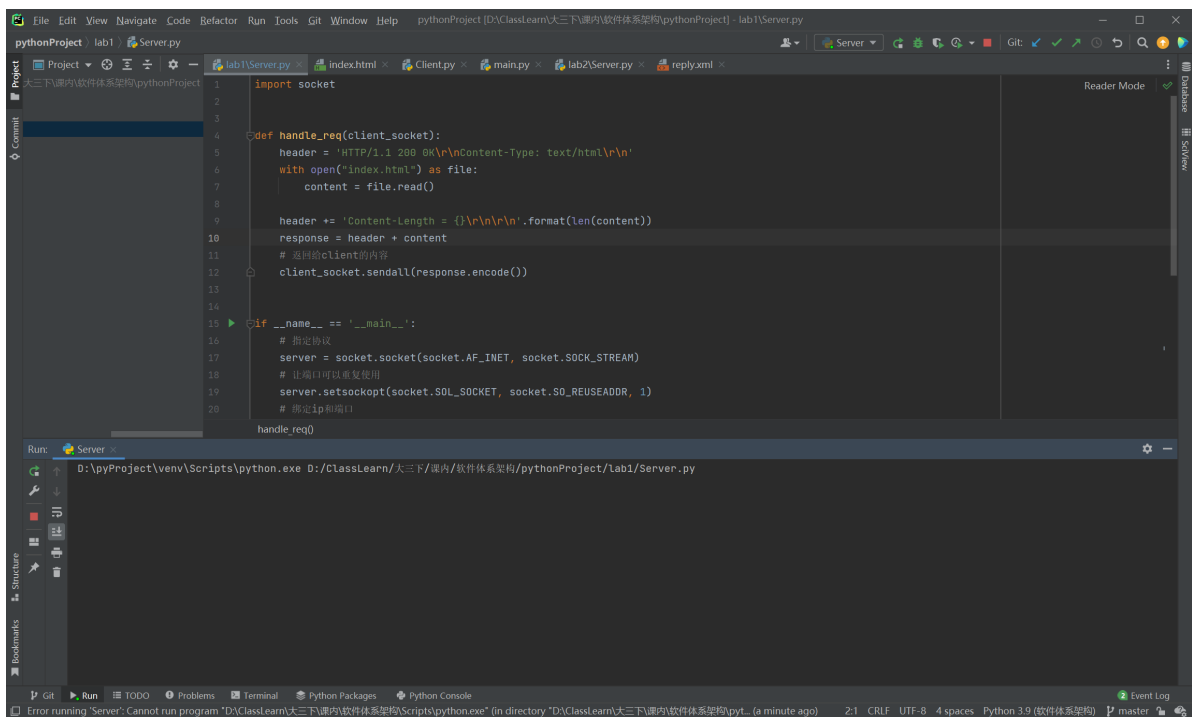
6. 使用Kafka、RocketMQ、RabbitMQ等实现一个消息队列应用系统，模拟多个消息生产者和多消费者；  
要求：实现一个客户端模拟车辆位置传感器，定时发送（车辆编号，经度、纬度、车速）数据到接收服务器，并保存到多个文件或数据库；

# 实验结果展示

## # 实验一

1. 使用的环境是python3.9
2. 先运行服务端代码，服务端会监听本地的8080端口，如果有客户端连接的话，就会调用handle\_req(clientsocket)方法，进行响应消息的处理，返回index.html这个页面
3. 运行完服务端代码之后，只需要在浏览器输入 <http://localhost:8080/> 即可访问

服务端：



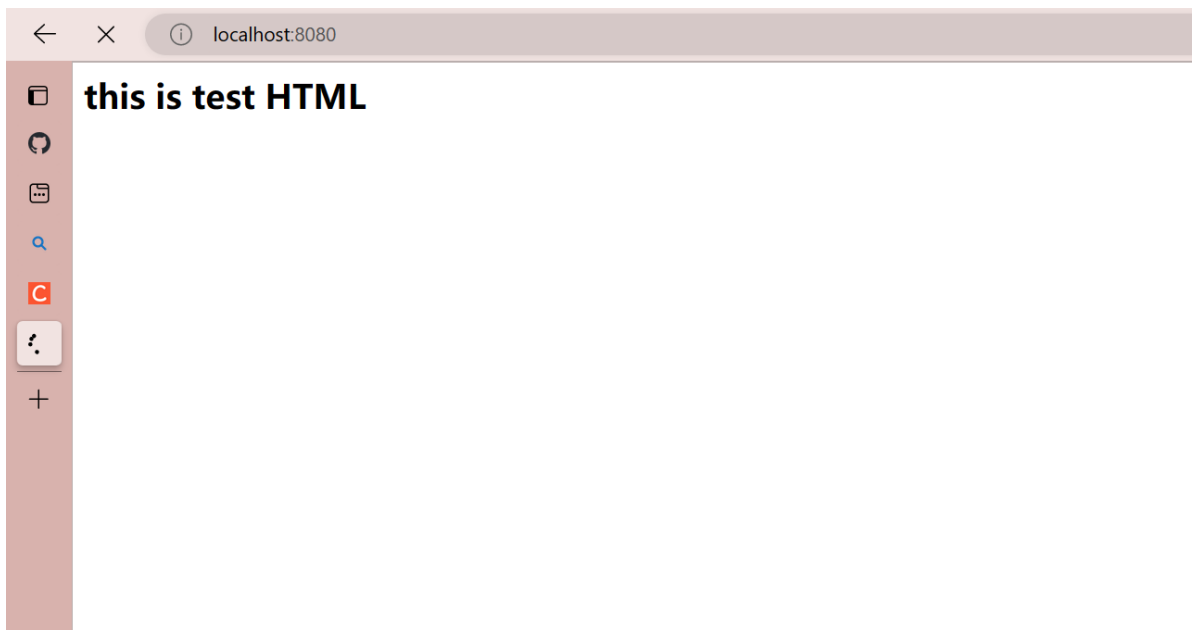
```
pythonProject [D:\ClassLearn\大三下\课内\软件体系架构\pythonProject] - lab1\Server.py
pythonProject lab1 Server.py
lab1\Server.py index.html Client.py main.py lab2\Server.py reply.xml
1 import socket
2
3
4 def handle_req(client_socket):
5     header = 'HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n'
6     with open("index.html") as file:
7         content = file.read()
8
9     header += 'Content-Length: {}'.format(len(content))
10    response = header + content
11    # 返回给client的内容
12    client_socket.sendall(response.encode())
13
14
15 if __name__ == '__main__':
16     # 指定协议
17     server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18     # 让端口可以重复使用
19     server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
20     # 绑定ip和端口
21
22     handle_req
```

Run: Server

D:\pyProject\venv\Scripts\python.exe D:/ClassLearn/大三下/课内/软件体系架构/pythonProject/lab1/Server.py

Error running 'Server'. Cannot run program "D:\ClassLearn\大三下\课内\软件体系架构\Scripts\python.exe" (in directory "D:\ClassLearn\大三下\课内\软件体系架构\pyt... (a minute ago)

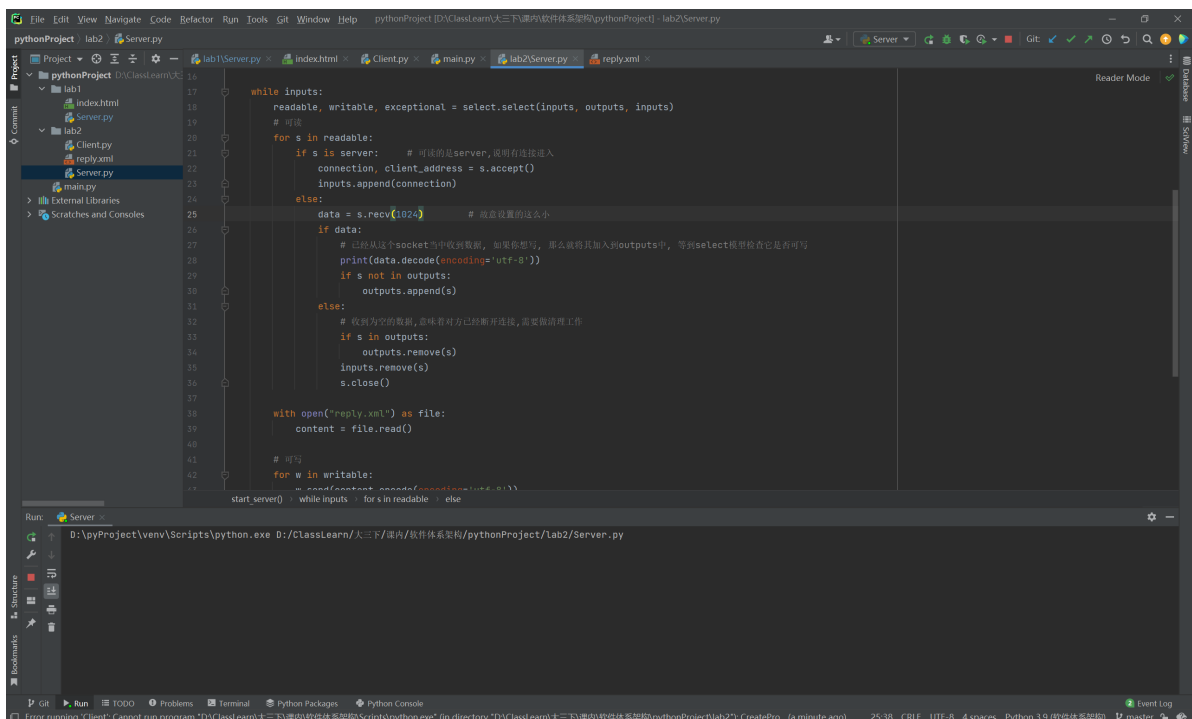
客户端：



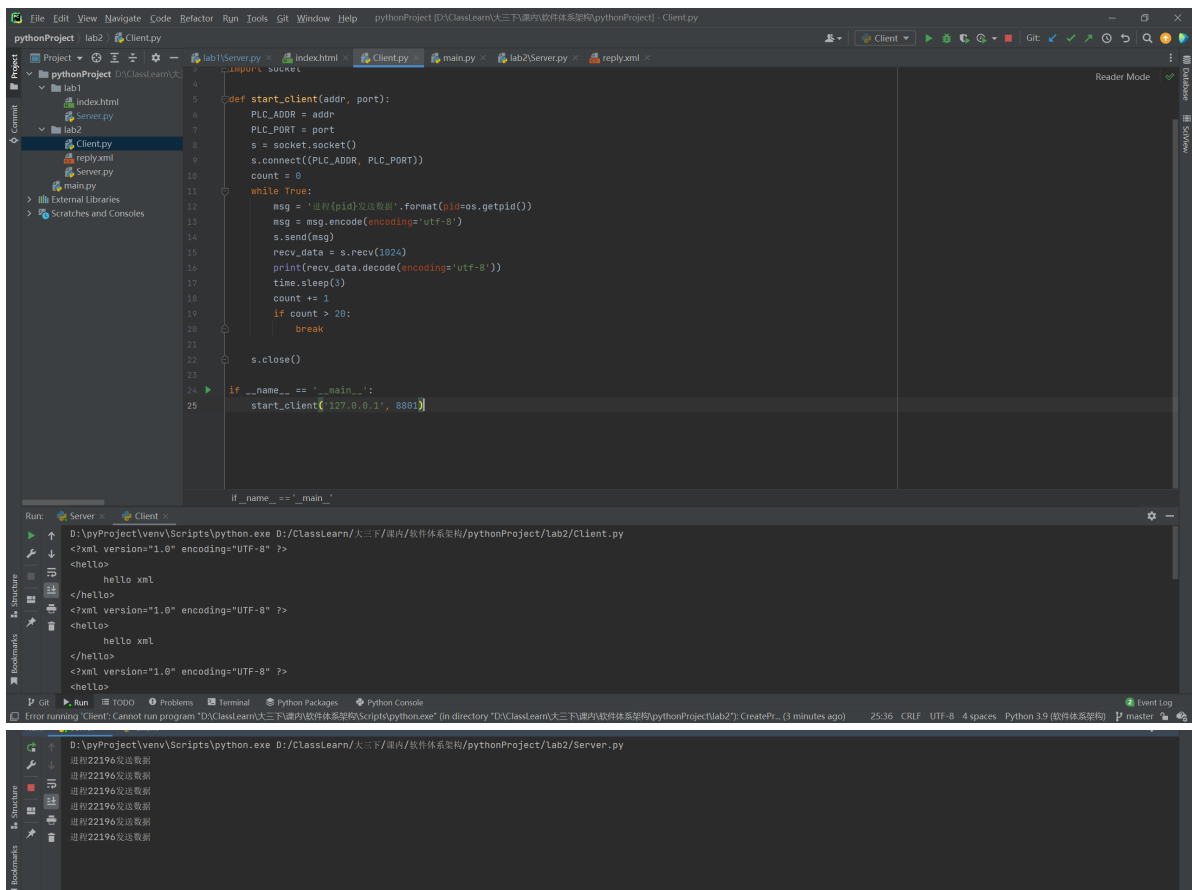
## # 实验二

运行环境：python3.9

1. 运行运行服务端，绑定的是本地的8081端口，不断循环读取inputs，如果有输入的话，就会返回给客户一段xml



2. 运行客户端，每个3s发送一段msg，发送20次之后就会结束，每次发送服务端都会响应一段xml

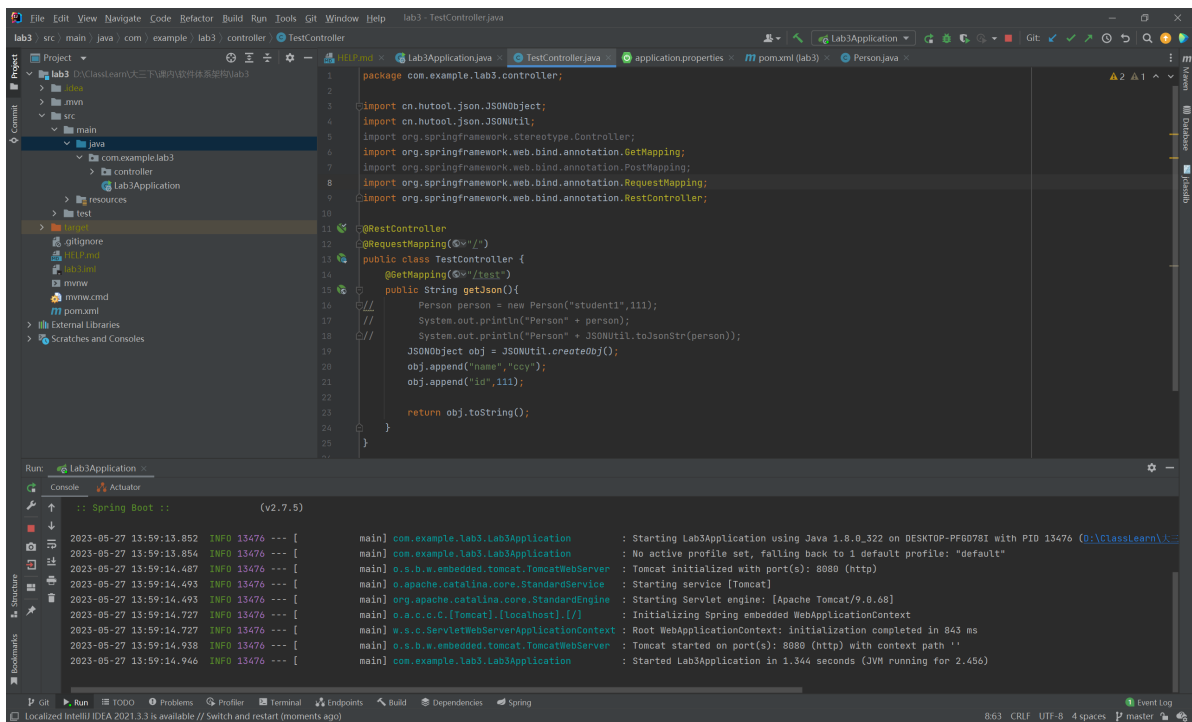


## # 实验三

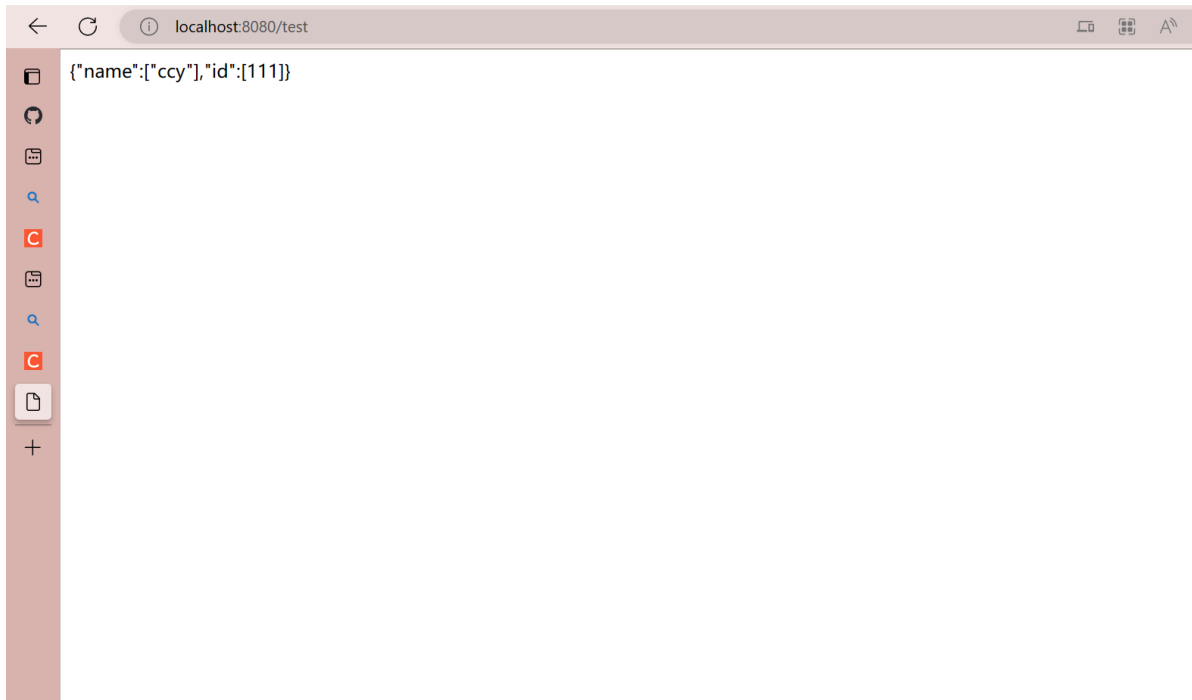
java - jdk1.8

springboot2.7.5

1. 运行springboot项目启动类，我们这里是绑定的本地的8080端口。并且写了一个restful的GET接口，路径为 localhost:8080/test



## 2. 使用浏览器进行访问，可以得到对应的JSON字符串



## # 实验四

环境：jdk1.8 、springboot2.3.9、rabbitMQ、AMQP依赖、docker

### 1. 我们先要启动rabbitMQ服务，我这里使用的是windows本地的docker来进行服务的部署

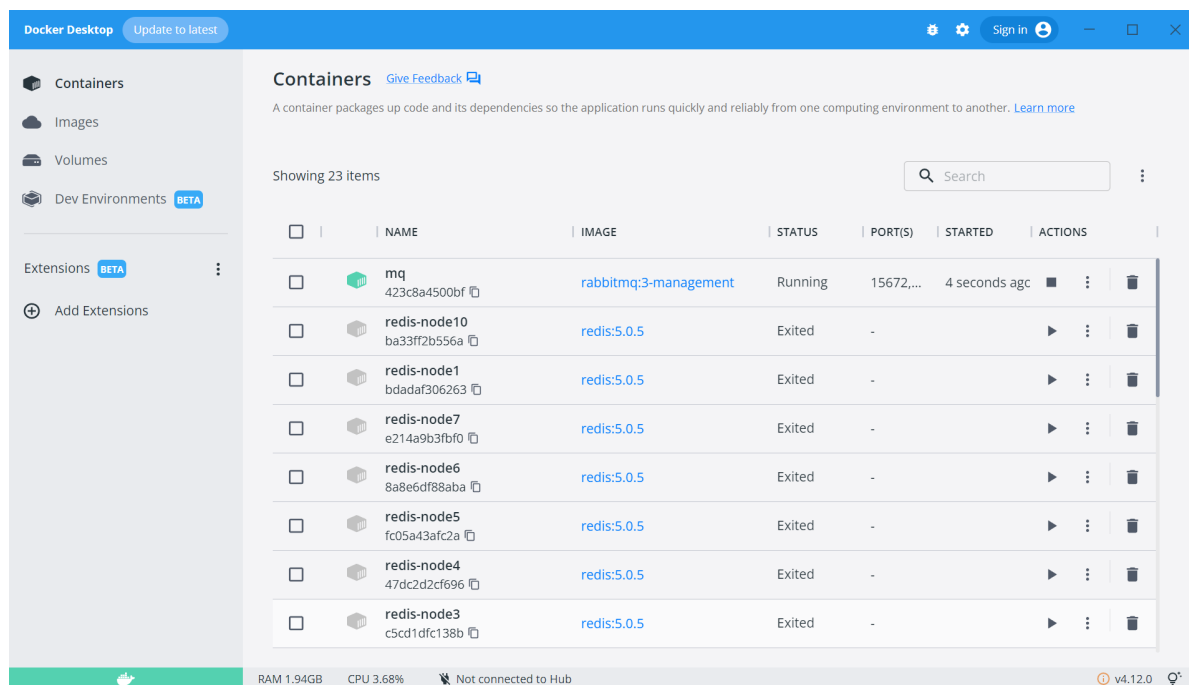
- 先拉取镜像

```
docker pull rabbitmq:3-management
```

- 通过下面命令运行

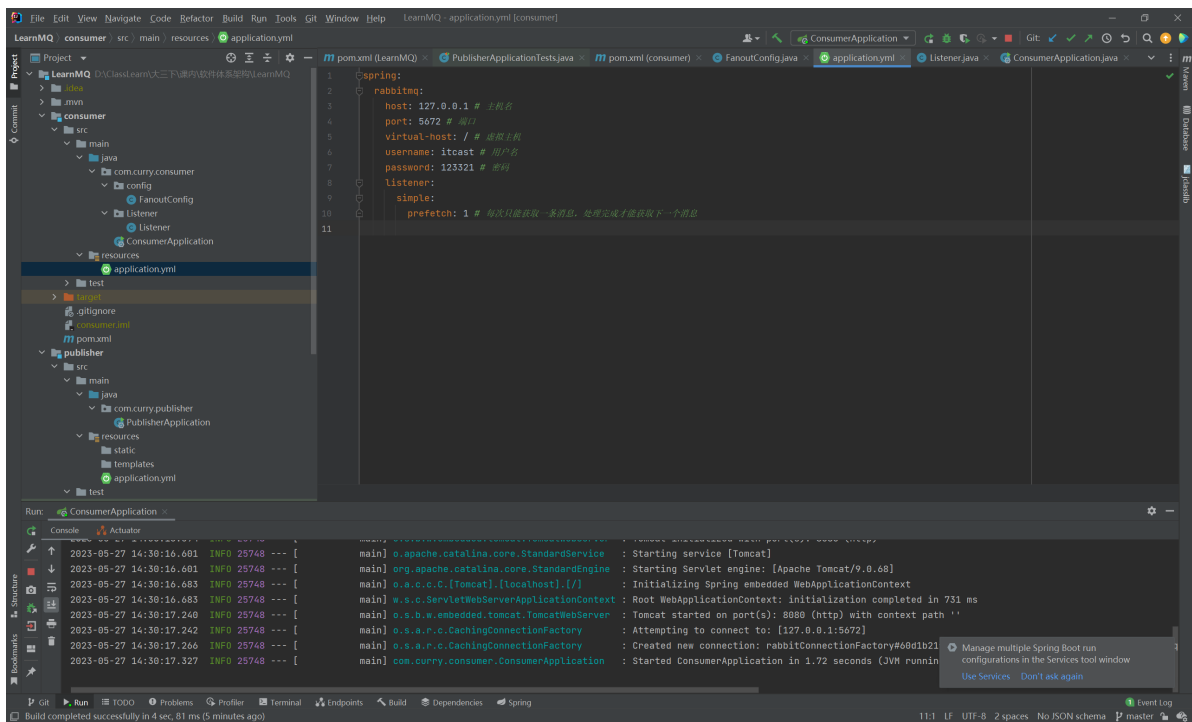
```
docker run \
-e RABBITMQ_DEFAULT_USER=itcast \
-e RABBITMQ_DEFAULT_PASS=123321 \
--name mq \
--hostname mq1 \
-p 15672:15672 \
-p 5672:5672 \
-d \
rabbitmq:3-management
```

如图：



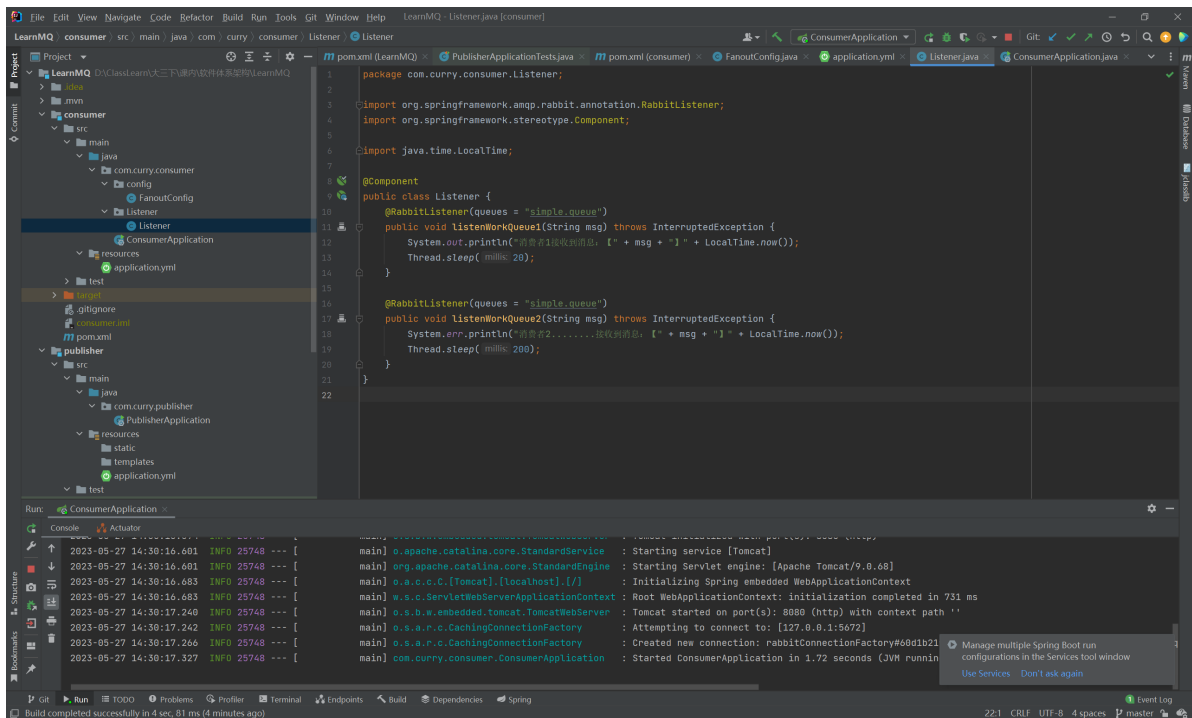
## 2. 先启动消费者服务

我们先在对应模块中，把application.yml配置好

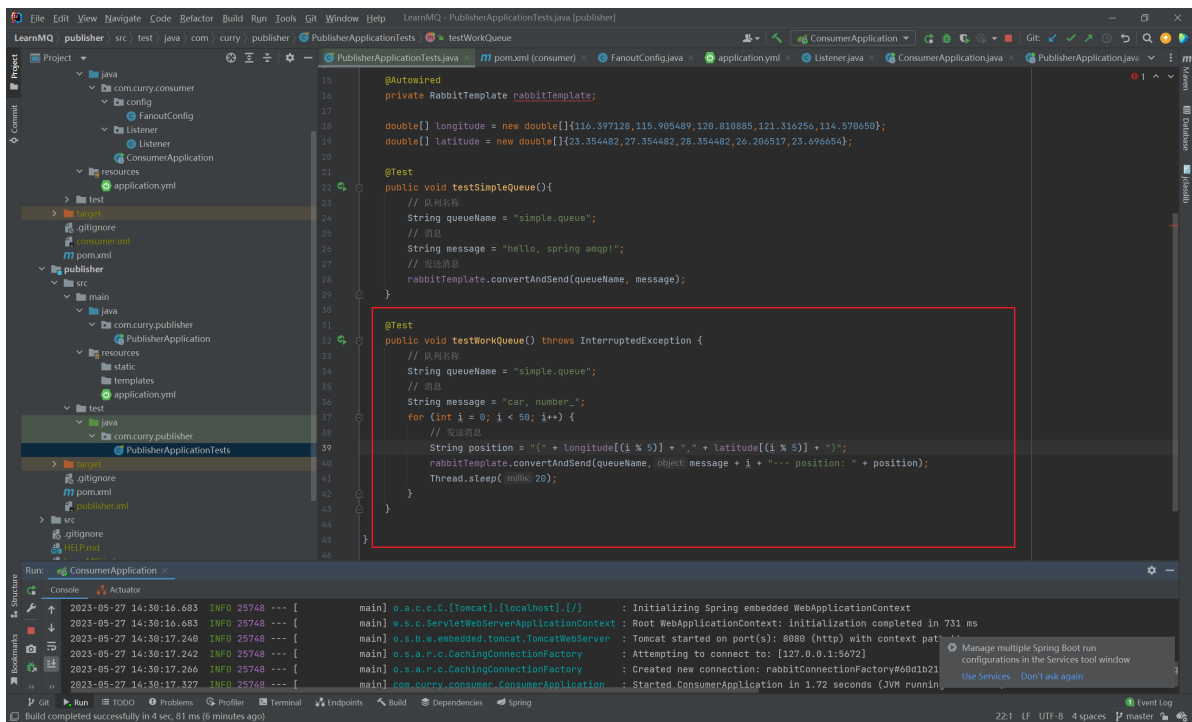


我这里使用的是AMQP框架，只需要通过注解`@RabbitListener(queues = "simple.queue")`就可以把一个消费者与一个队列进行绑定。

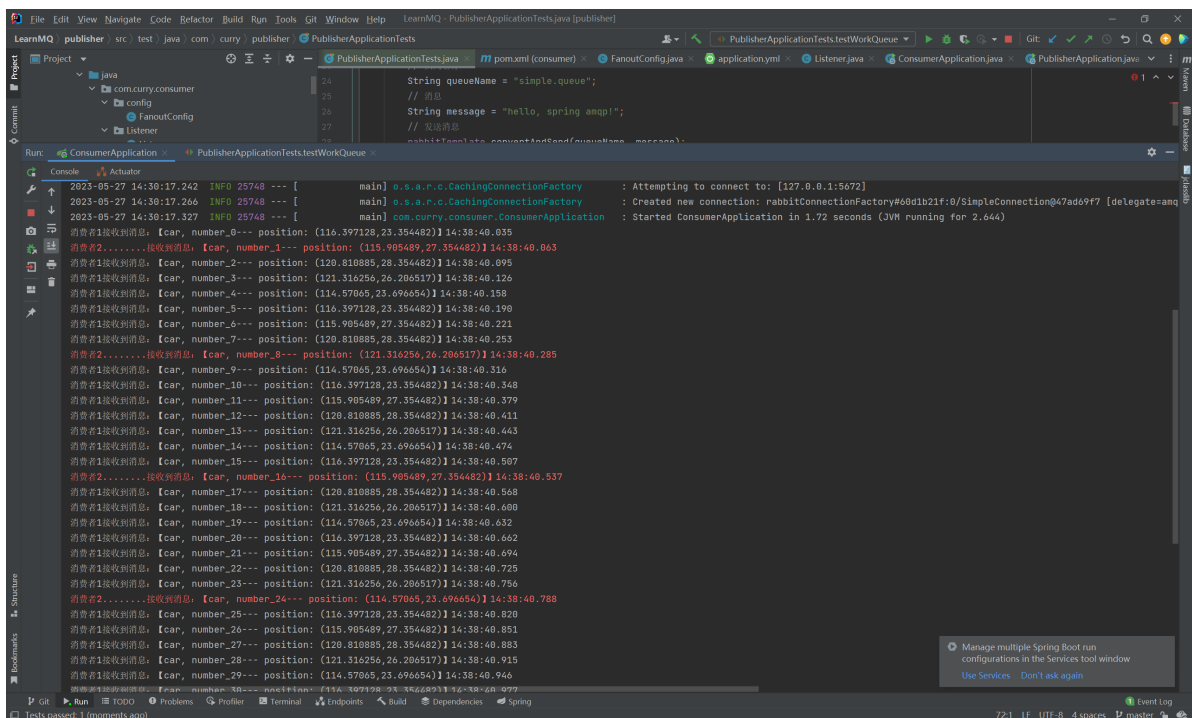
我这里绑定了两个消费者，并且两个消费者的消费速度是不一样的，通过`sleep`来模拟控制两个消费者的消费速度。消费者1 会比 消费者2 消费速度更快



3. 启动生产者服务，通过Test方法进行消息的模拟发送，我这里是会发送50条位置信息，每个20毫秒发送一条。



4. 发送之后，可以在消费者端看到接收到消息，消费者1的处理速度明显比消费者2的快。



5. 如果想看到RabbitMQ的管理界面，可以访问 <http://127.0.0.1:15672/> )，可以看到对应的队列信息等



## Queues

All queues (8)

Pagination

Page 1 of 1 Filter:  ☐ Regex

Displaying 8 items , page size up to: 100

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
direct.queue1	classic		Idle	0	0	0				
direct.queue2	classic		Idle	0	0	0				
fanout.queue1	classic		Idle	0	0	0				
fanout.queue2	classic		Idle	0	0	0				
object.queue	classic		Idle	0	0	0				
simple.queue	classic		Idle	0	0	0	0.00/s	0.00/s	0.00/s	
topic.queue1	classic		Idle	0	0	0				
topic.queue2	classic		Idle	0	0	0				

Add a new queue