

# 2022 7.12

## 工作说明

实现了大作业 1. 功能要求 中的第1, 2点功能。

## 详细步骤

### 1. 创建并初始化Spring boot项目

1. 使用idea插件 `Spring boot initializer` 初始化一个名为 `cloud-native-proj` 的springboot项目。
2. 在 `src/main/resource/application.properties` 文件下修改web服务端口:

```
server.port=8080
```

3. 新建 `src/main/java/com/example/cloud_native_proj/controller` 文件夹, 在该文件夹下新建 `UserController.java` 文件, 并自定义 `UserController` 类, 返回基本的 json 信息。(返回 json 信息需要导入 `alibaba fastjson` 依赖)

```
<!--json工具-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.15</version>
</dependency>
```

导入json依赖

```
public class UserController {
    //ResponseBody
    @RequestMapping("/hello")
    //每秒并发量, 设置为10
    @CurrentLimiter(QPS = 10)
    public Object sayHello() {
        JSONObject jsonObject=new JSONObject();

        jsonObject.put("code",200);
        jsonObject.put("group","1");
        jsonObject.put("member","Qinyu Chen,Jifeng Duan,Huiyuan Yan");
        jsonObject.put("msg","Welcome!");
        return jsonObject.toString();
    }
}
```

*UserController*类

## 2.实现接口限流功能

采用 `current limiting` 工具实现了接口限流工作，但还没有实现多pod统一限流。

### 1. 引入依赖

```
<!--接口限流，采用Current Limiting 工具进行-->
<dependency>
  <groupId>cn.yueshutong</groupId>
  <artifactId>spring-boot-starter-current-limiting</artifactId>
  <version>0.0.8.RELEASE</version>
</dependency>
```

### 2. 在 `src` 根文件夹下新建 `application.yaml` 配置文件并写入流量控制的相关配置内容：

```
current:
  limiting:
    #开启全局限流
    enabled: false
    #开启注解限流,可使注解失效
    part-enabled: true
    #每秒并发量 这里的qps是全局限流开启的时候的值,如果使用注解在注解里设置QPS值
    qps: 100
    #开启快速失败,可切换为阻塞
    fail-fast: true
    #系统启动保护时间为0
    initial-delay: 0
```

### 3. 在 `UserController` 类下添加注解：

```
public class UserController {
    //@ResponseBody
    @RequestMapping("/hello")
    //每秒并发量，设置为10
    @CurrentLimiter(QPS = 10)
    public Object sayHello() {
```

为测试方便，每秒最大限流为10次（作业最终提交时可修改成100次）。

### 4. 新建 `src/main/java/com/example/cloud_native_proj/co/FC` 文件夹(Flow Control)，在该文件夹下新建 `MyCurrentLimitHandler.java` 文件，并自定义处理类处理超时函数。

```

@Component
public class MyCurrentLimitHandler implements CurrentAspectHandler {
    @Override
    public Object around(ProceedingJoinPoint pjp, CurrentLimiter rateLimiter) {

        //限流的返回数据可以自己根据需求场景设计

        JSONObject jsonObject=new JSONObject();

        jsonObject.put("code",429);
        jsonObject.put("msg","Too Too many requests");
        return jsonObject.toString();
        //return "Too many!!!";
    }
}

```

## 功能演示

### 1. 编译运行

在项目文件夹下运行命令：

```
mvn clean install package '-Dmaven.test.skip=true'
```

```
java -jar target/cloud_native_proj-0.0.1-SNAPSHOT.jar
```

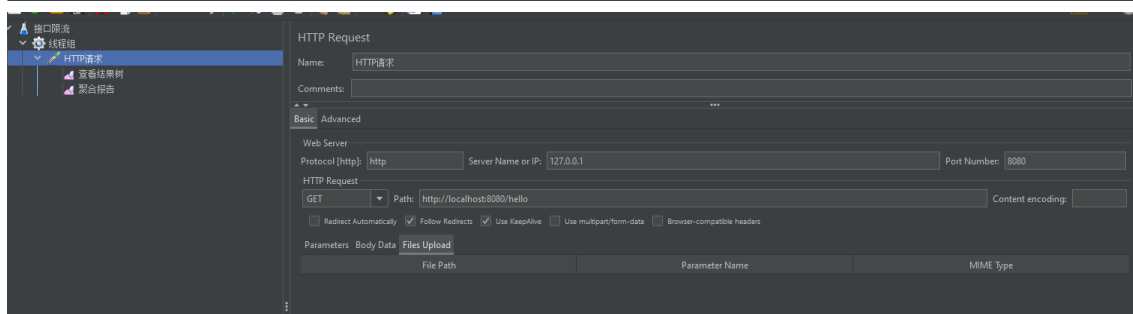
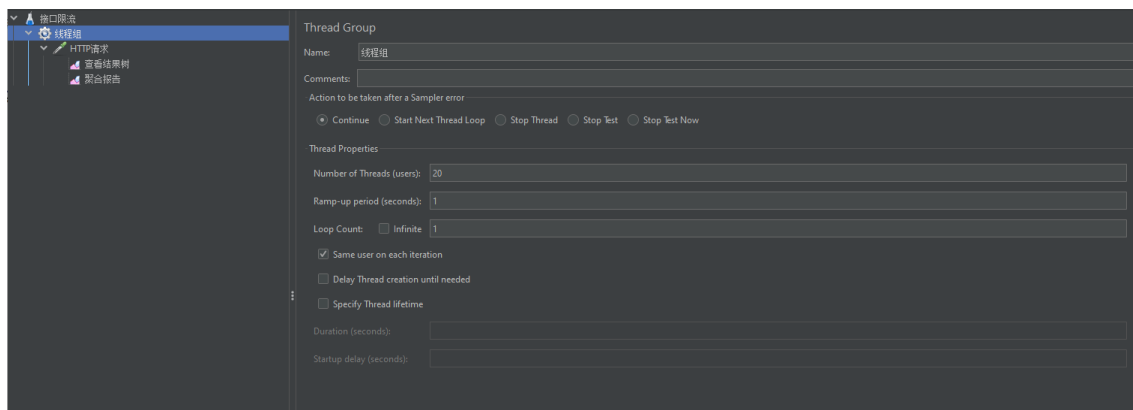
构建并运行项目。

### 2. 打开浏览器输入网址：`http://localhost:8080/hello`（注意mapping有/hello）查看内容：

```
{"msg":"Welcome!","code":200,"member":"Qinyu Chen,Jifeng Duan,Huiyuan Yan","group":"1"}
```

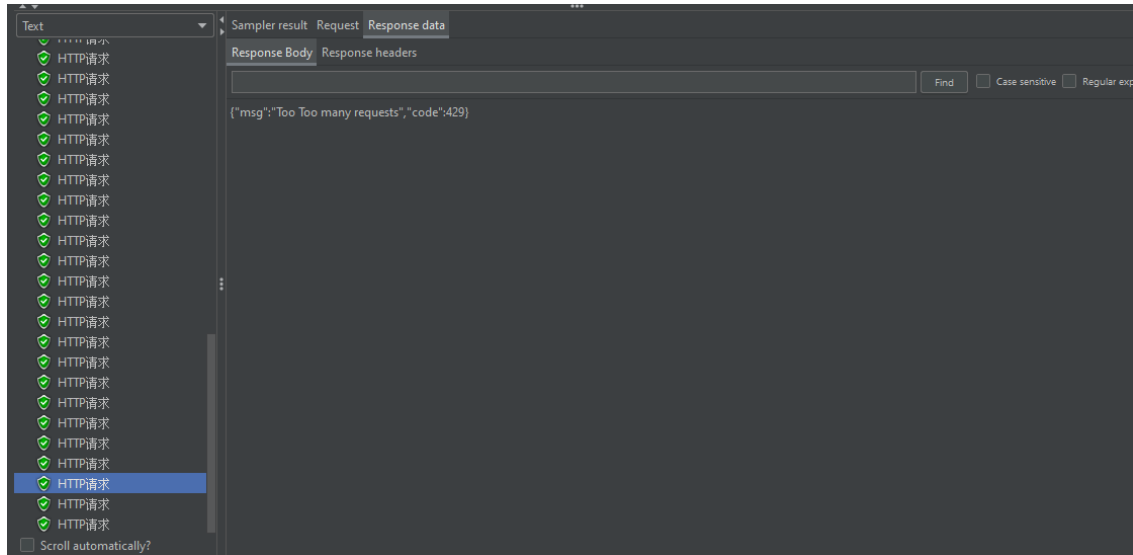
### 3. 用 Apache Jmeter 工具对接口进行压力测试。

配置如下：



会生成jmx文件。

然后点击右上角绿色箭头运行，在结果树中查看情况。



平均一秒内每十次左右访问就会出现访问次数过多的返回值。

## 2022.7.13

### 工作说明

实现了DevOps要求中的第一点和第二点，即完成了DockerFile和K8s编排文件的编写。

### 1.编写DockerFile

在target目录下创建DockerFile并写入如下内容：

```
FROM openjdk:8-jre-alpine
#作者
MAINTAINER Group1 1084683108@qq.com
#复制文件并命名，ADD支持远程获取URL资源
ADD cloud_native_proj-0.0.1-SNAPSHOT.jar /cloud_native_proj.jar
#VOLUME ，VOLUME 指向了一个/tmp的目录，由于 Spring Boot 使用内置的Tomcat容器，Tomcat 默认使用/tmp作为工作目录。这个命令的效果是：
#在宿主机的/var/lib/docker目录下创建一个临时文件并把它链接到容器中的/tmp目录
#VOLUME /tmp
#声明端口
EXPOSE 8080
ENTRYPOINT ["java","-jar","/cloud_native_proj.jar"]
#命令：docker build -t cloud_native_proj .
#docker run --name cloud_native_proj -p 8080:8080 -d cloud_native_proj
```

运行上述最后两行注释命令，即可在本地ip的8080端口使用Web服务。

注意DockerFile第一行**必须是** FROM xxx。

## 2.编写K8s编排文件

在根目录下创建 `proj_deploy.yaml` 和 `proj_svc.yaml` 文件，分别写入如下内容：

```
#proj_deploy.yaml
apiVersion: apps/v1    # 1.9.0 之前的版本使用 apps/v1beta2，可通过命令 kubectl api-
versions 查看
kind: Deployment      #指定创建资源的角色/类型
metadata:             #资源的元数据/属性
  name: cloud-native-proj #资源的名字，在同一个namespace中必须唯一
  namespace: cloud-native-namespace
spec:
  replicas: 2          #副本数量2
  selector:            #定义标签选择器
    matchLabels:
      app: cloud-native-proj
  template:            #这里Pod的定义
    metadata:
      labels:          #Pod的label
        app: cloud-native-proj
    spec:              # 指定该资源的内容
      containers:
        - name: cloud-native-proj    #容器的名字
          image: cloud_native_proj   #容器的镜像地址
          imagePullPolicy: Never
```

```
#proj_svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: cloud-native-proj-svc
  namespace: cloud-native-namespace
spec:
  type: NodePort
  selector:
    app: cloud-native-proj
  ports:
    - nodePort: 30001 # host's port,集群外部访问端口
      port: 8888      # service's port,集群内部访问端口
      targetPort: 8080 # target pod's port,服务最终端口，所有流量流至该端口，和
                        DockerFile中EXPOSE端口一致
                        #按照k8s_cmd.sh的命令执行完后，即可通过本机ip:30001端口访问web服务
```

用于创建环境和服务。

然后依次执行下面的命令（我已把命令放在文件 `k8s_cmd.sh` 中）

```
kubectl create namespace cloud-native-namespace
# kubectl get namespace
kubectl apply -f proj_deploy.yaml
# kubectl get deployment -n cloud-native-namespace
kubectl create -f proj_svc.yaml
# kubectl get svc -n cloud-native-namespace
```

分别是创建名为 `cloud-native-namespace` 的命名空间和在该空间下（yaml脚本中限定）创建 `deployment` 和 `service`。

带注释的命令是用来检查每一步创建是否成功的。

上述步骤都执行成功后，即可通过本机ip的 `30001` 端口访问Web服务。