# 第四章 Sentinel--服务容错

## 高并发带来的问题

在微服务架构中，我们将业务拆分成一个个的服务，服务与服务之间可以相互调用，但是由于网络 原因或者自身的原因，服务并不能保证服务的100%可用，如果单个服务出现问题，调用这个服务就会 出现网络延迟，此时若有大量的网络涌入，会形成任务堆积，最终导致服务瘫痪。

接下来，我们来模拟一个高并发的场景

### 编写java代码

@RestController @Slf4j

public class OrderController2 { @Autowired

private OrderService orderService;

@Autowired

private ProductService productService;

@RequestMapping("/order/prod/{pid}")

public Order order(@PathVariable("pid") Integer pid) {

log.info("接收到{}号商品的下单请求,接下来调用商品微服务查询此商品信息", pid);



//调用商品微服务,查询商品信息

Product product = productService.findByPid(pid);

log.info("查询到{}号商品的信息,内容是:{}", pid, JSON.toJSONString(product));

//模拟一次网络延时

try {

Thread.sleep(100);

} catch (InterruptedException e) { e.printStackTrace();

}

//下单(创建订单)

Order order = new Order(); order.setUid(1); order.setUsername("测试用户");

order.setPid(pid); order.setPname(product.getPname()); order.setPprice(product.getPprice());

order.setNumber(1);

//为了不产生太多垃圾数据,暂时不做订单保存

//orderService.createOrder(order);

log.info("创建订单成功,订单信息为{}", JSON.toJSONString(order));

return order;

}

@RequestMapping("/order/message") public String message() {

return "高并发下的问题测试";

}

}

### 修改配置文件中tomcat的并发数

server: port: 8091 tomcat:

max-threads: 10 #tomcat的最大并发值修改为10,默认是200

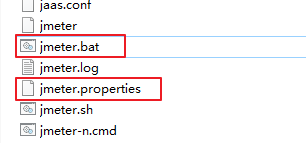
### 请求进行压力测试

接下来使用压测工具,对请求进行压力测试

下载地址https://jmeter.apache.org/

第一步：修改配置，并启动软件

进入bin目录,修改jmeter.properties文件中的语言支持为language=zh\_CN，然后点击jmeter.bat 启动软件。



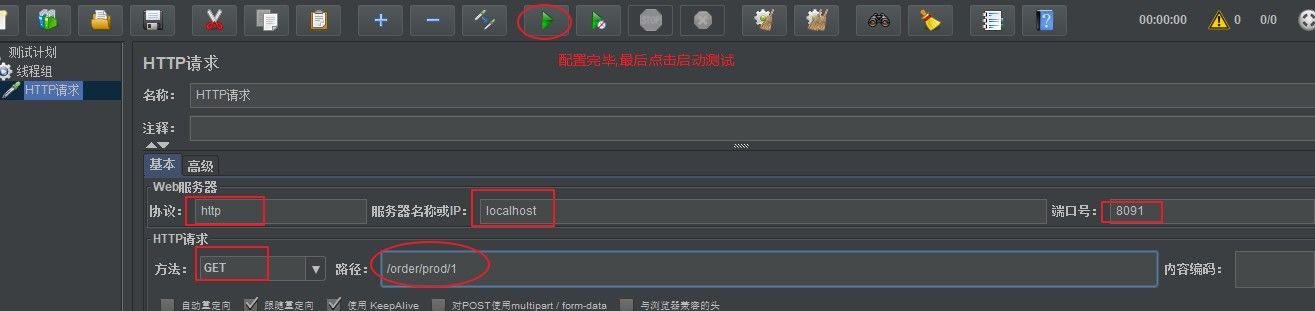
第二步：添加线程组

第三步：配置线程并发数

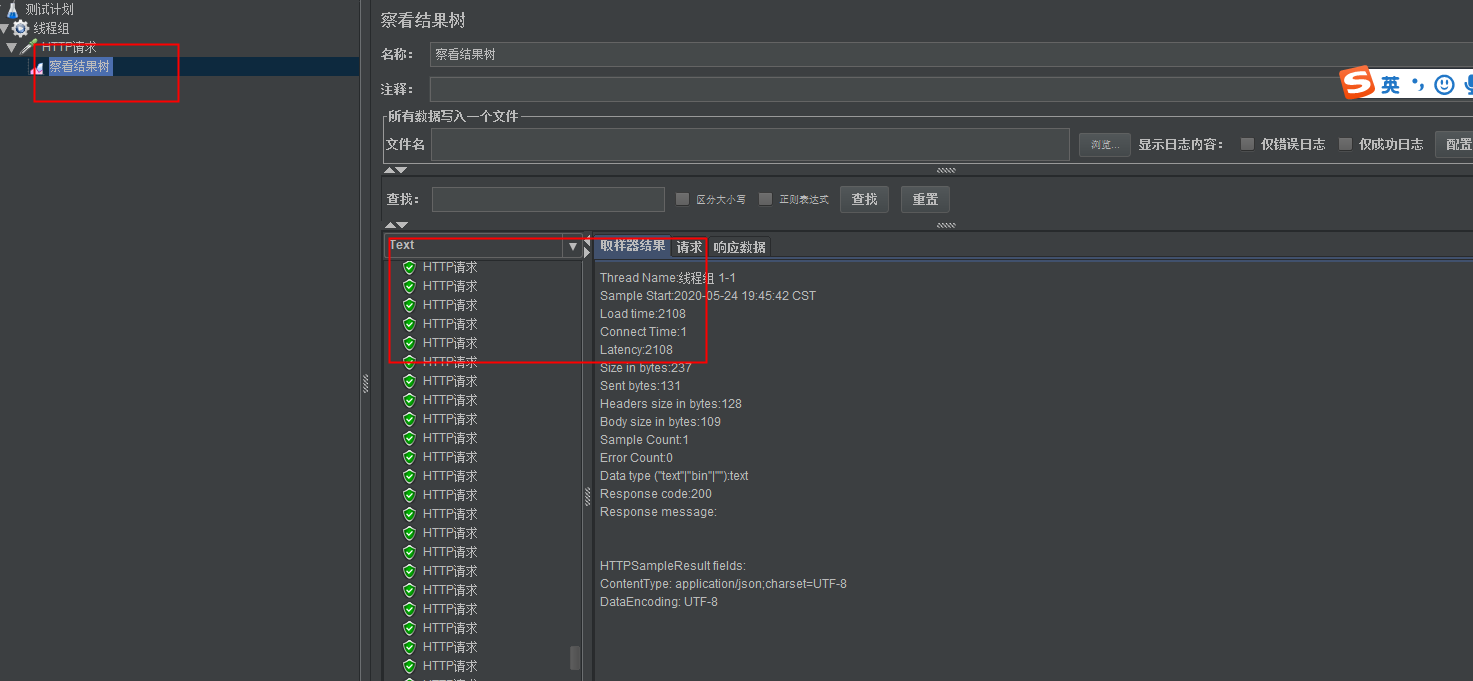


第四步：添加Http取样

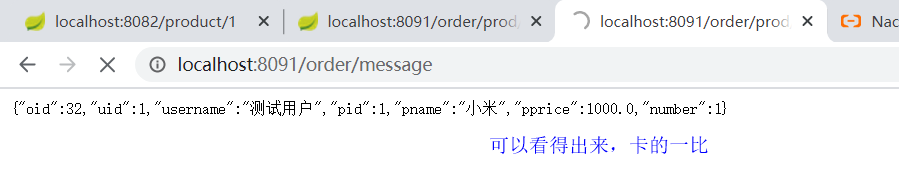
第五步：配置取样，并启动测试



查看结果树



### 访问ｍessage方法观察效果



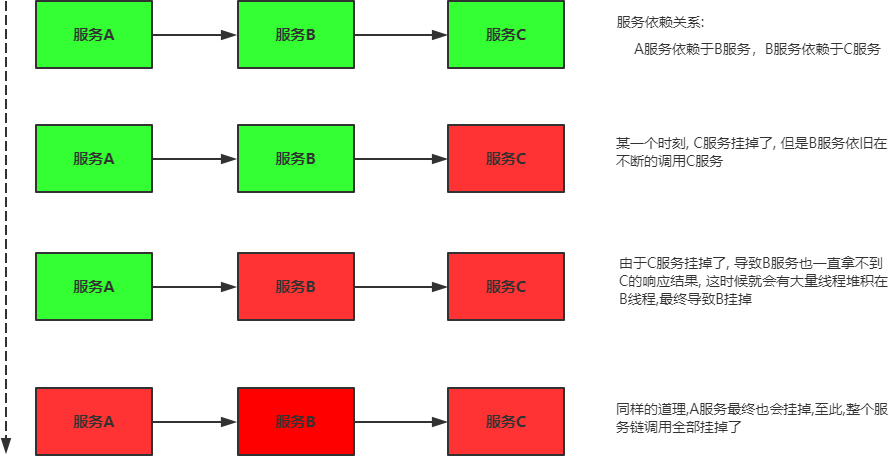
### 结论:

此时会发现, 由于order方法囤积了大量请求, 导致ｍessage方法的访问出现了问题，这就是**服务雪崩**的雏形。

# 服务雪崩效应

在分布式系统中,由于网络原因或自身的原因,服务一般无法保证 100% 可用。如果一个服务出现了问题，调用这个服务就会出现线程阻塞的情况，此时若有大量的请求涌入，就会出现多条线程阻塞等 待，进而导致服务瘫痪。

由于服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是 服务故障的 **“雪崩效应”** 。



雪崩发生的原因多种多样，有不合理的容量设计，或者是高并发下某一个方法响应变慢，亦或是某 台机器的资源耗尽。我们无法完全杜绝雪崩源头的发生，只有做好足够的容错，保证在一个服务发生问 题，不会影响到其它服务的正常运行。也就是＂雪落而不雪崩＂。