高并发&压力测试&服务端优化

课程主题：

1、什么是高并发（你是否真的什么是高并发？？ 100wQPS 亿级流量）

2、分析高并发基本指标（QPS,TPS,RT）

3、应用服务部署（服务部署在阿里云服务器：4 台阿里云 4 核心 8GB）

4、jmeter 进行压力测试（未优化之前压力测试）--- 观察单机极限（TPS,QPS）

5、服务优化测试（并发压力测试）

* 线程池

* keepalive

6、性能瓶颈分析方法（参数分析，linux 本身自带指令）

第二天：

Jvm 优化（jvm 原理讲解，jvm 实战调优）

第三天：

分布式部署（mysql，服务分离，集群）

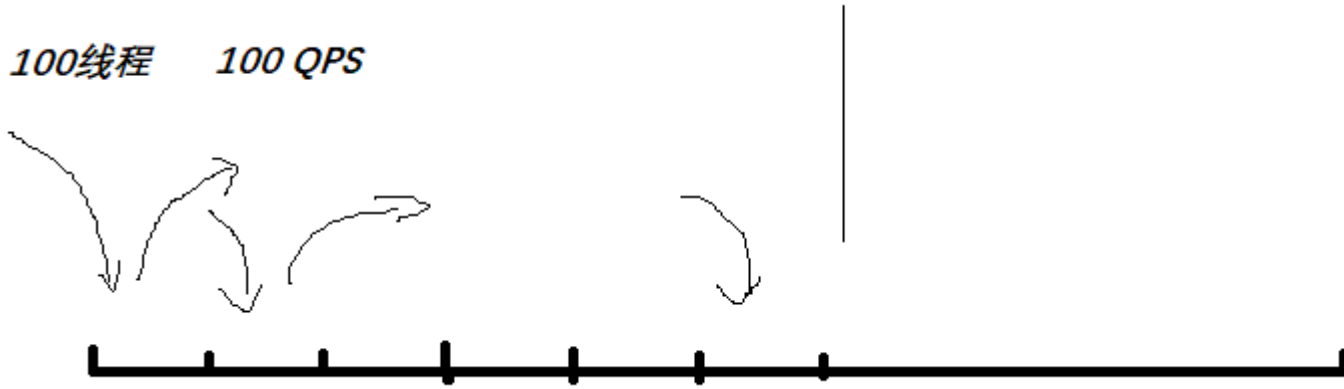
微服务部署

K8s 中模拟在高并发压力下进行可伸缩容的实战演练

一、 并发&高并发

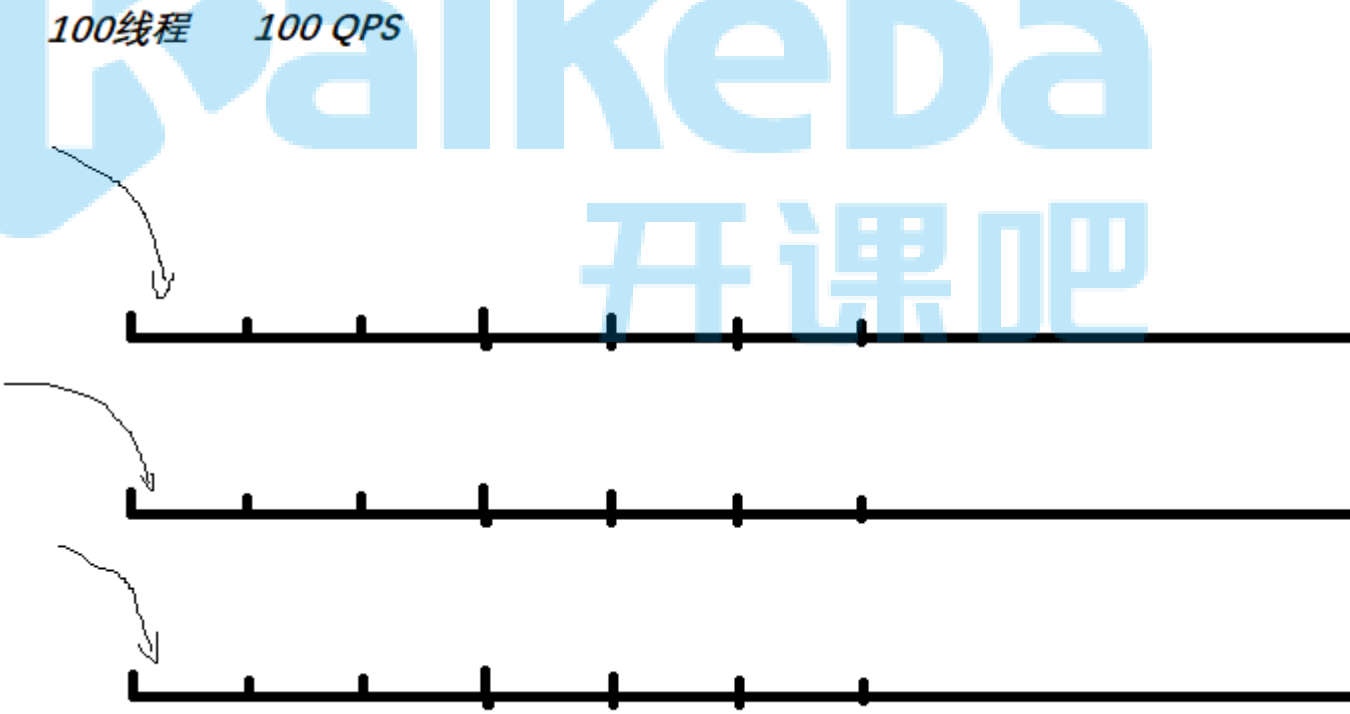
1、什么是并发

在一段时间之内，几个程序处于启动到运行结束之间这段时间，叫做并发。



2、什么是并行

并发操作是指有多个 cpu（多核心），在同一个时候，有多个 cpu 可以同时运行一个线程，那么这几个线程都是并行线程。



通常所说的 QPS,TPS 就是 并发操作。

3、什么是高并发

问题： 多线程是高并发吗？

多线程实现高并发手段，但是多线程不等于高并发。

实现高并发考虑哪些问题：

1）系统架构设计，如何在架构层面减少不必要处理（网络请求，数据库操作）

例如：使用 cache 减少 io 操作，使用异步方法提升服务吞吐量

2）网络拓扑结构优化，如何设计系统架构拓扑结构 （分布式架构，微服务架构，service mesh）

3）系统代码级别的优化（使用什么设计模式进行工作：单例模式，减少 new 对象的操作，提升系统性能）

4）提高代码层面运行效率（选举合适的数据结构，让代码执行效率更高）

5）jvm 调优（如何设置 heap,eden,old）

6）服务端调优（tomcat 线程池，队列）

7）数据调优（线程池，SQL 调优，服务端调优）

8）缓存的使用（Redis）

9）数据通信问题 （服务内部：使用 tcp）

10）硬件配置

4、你真的了解高并发

100w,亿级流量高并发项目？？？

在一瞬间，系统遇到了超高的流量访问，系统 scoket 端口遇到了超高的流量访问。这样场景就叫做高并发。

例如：某一个服务 一天 （时间段）36 w 笔，平均耗时：RT = 100ms

Qps = 36w / 10 * 3600 * 10（扩大 10 倍） = 100 QPS

一天： 1h 36w 笔

36w / 3600 * 10 = 1000 QPS

64 核心 CPU, 128G 内存： QPS 40w

二、 服务部署

项目打包： 可以使用 idea 直接打包上传，也可以在 gitlab 服务器直接通过 maven 进行打包，或者使用 jenkins 来进行打包，现在我们先使用 idea 的 maven 进行打包。

```
# 注意打包必须的依赖
<build>
    <plugins>
  <--此依赖必须有，否则项目的依赖包无法被打进项目：mysql.jar,spring*.jar 都无法打包进入-->
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <!-- 编译项目，然后打包，只会打包自己的代码 -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
    </plugins>
</build>
```

idea 的 maven 打包、

注意： 打包服务的时候必须注意服务的配套的 ip 地址,由于此时服务和 mysql，redis 都在同一个服务器上，因此连接访问地址设置为 localhost 即可。

```
server:
port: 9000
spring:
application:
  name: sugo-seckill-web
datasource:
  url:
jdbc:mysql://127.0.0.1:3306/shop?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true
#    url:
jdbc:mysql://47.113.81.149:3306/shop?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true
  username: root
  password: root
  driver-class-name: com.mysql.jdbc.Driver
  druid:
    #配置初始化大小、最小、最大
    initial-size: 1
    min-idle: 5
    max-active: 5
    max-wait: 20000
    time-between-eviction-runs-millis: 600000
    # 配置一个连接在池中最大空闲时间，单位是毫秒
    min-evictable-idle-time-millis: 300000
    # 设置从连接池获取连接时是否检查连接有效性，true 时，每次都检查;false 时，不检查
    test-on-borrow: true
    #设置往连接池归还连接时是否检查连接有效性，true 时，每次都检查;false 时，不检查
    test-on-return: true
    # 设置从连接池获取连接时是否检查连接有效性，true 时，如果连接空闲时间超过 minEvictableIdleTimeMillis 进行检查，否则不检查;false 时，不检查
    test-while-idle: true
    # 检验连接是否有效的查询语句。如果数据库 Driver 支持 ping()方法，则优先使用 ping()方法进行检查，否则使用 validationQuery 查询进行检查。(Oracle jdbc Driver 目前不支持 ping 方法)
```

```
    validation-query: select 1 from dual
    keep-alive: true
    remove-abandoned: true
    remove-abandoned-timeout: 80
    log-abandoned: true
    #打开 PSCache，并且指定每个连接上 PSCache 的大小，Oracle 等支持游标的数据库，打开此开关，会以数量级提升性能，具体查阅 PSCache 相关资料
    pool-prepared-statements: true
    max-pool-prepared-statement-per-connection-size: 20
    # 配置间隔多久启动一次 DestroyThread，对连接池内的连接才进行一次检测，单位是毫秒。
    #检测时:
    #1.如果连接空闲并且超过 minIdle 以外的连接，如果空闲时间超过 minEvictableIdleTimeMillis 设置的值则直接物理关闭。
    #2.在 minIdle 以内的不处理。
redis:
  host: 127.0.0.1
  port: 6379
mybatis:
type-aliases-package: com.supergo.pojo
mapper:
not-empty: false
identity: mysql
```

1、打包上传

启动命令： java -jar jshop-web-1.0-SNAPSHOT.jar

注意：服务器部署的时候，由于服务器环境的不同，往往都需要额外的修改服务的配置文件，重新编译打包，必须服务器 ip 地址，本地开发环境的 ip 和线上的 ip 是不一样的，部署的时候，每次都需要修改这些配置，非常麻烦。因此服务部署时候应该具有一个外挂配置文件的能力。

```
    #启动命令,注意：配置文件的名称必须是 application.yaml，或者 application.properties
java -jar xxx.jar --spring.config.addition-location=/usr/local/src/application.yaml
```

外挂配置文件使用本地连接地址：

2、启动脚本

创建 deploy.sh 这样一个 shell 脚本文件，执行 java 程序的后端启动工作

```
#使用 nohup 启动，使得 Java 进程在后台以进程模试运行
nohup java -Xms500m -Xmx500m -XX:NewSize=300m -XX:MaxNewSize=300m -jar
jshop-web-1.0-SNAPSHOT.jar --spring.config.addition-location=application.yaml >
log.log 2>&1 &

#授权
chmod 777 deploy.sh

#查询进程
jps
jps -l
```

浏览器接口测试访问，发现服务已经启动成功：

Keepavile

三、 性能参数分析

	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常
求	100000	583	625	964	1028	1348	3	2756	
	100000	583	625	964	1028	1348	3	2756	

样本： 测试请求数量（在 5s 之内启动了 2000 个线程，持续循环 50 次）

平均值：所有的请求平均耗时

中位数：50%请求平均耗时

90%百分位：90%请求平均耗时

99%百分位：99%的请求都此时间之内完成请求

吞吐量：2798　　每秒请求数量

吞吐量 QPS　TPS 并发数量

大多数情况下：吞吐量 = QPS（每秒查询数） = TPS（每秒事务数） = 并发数 （1/RT）

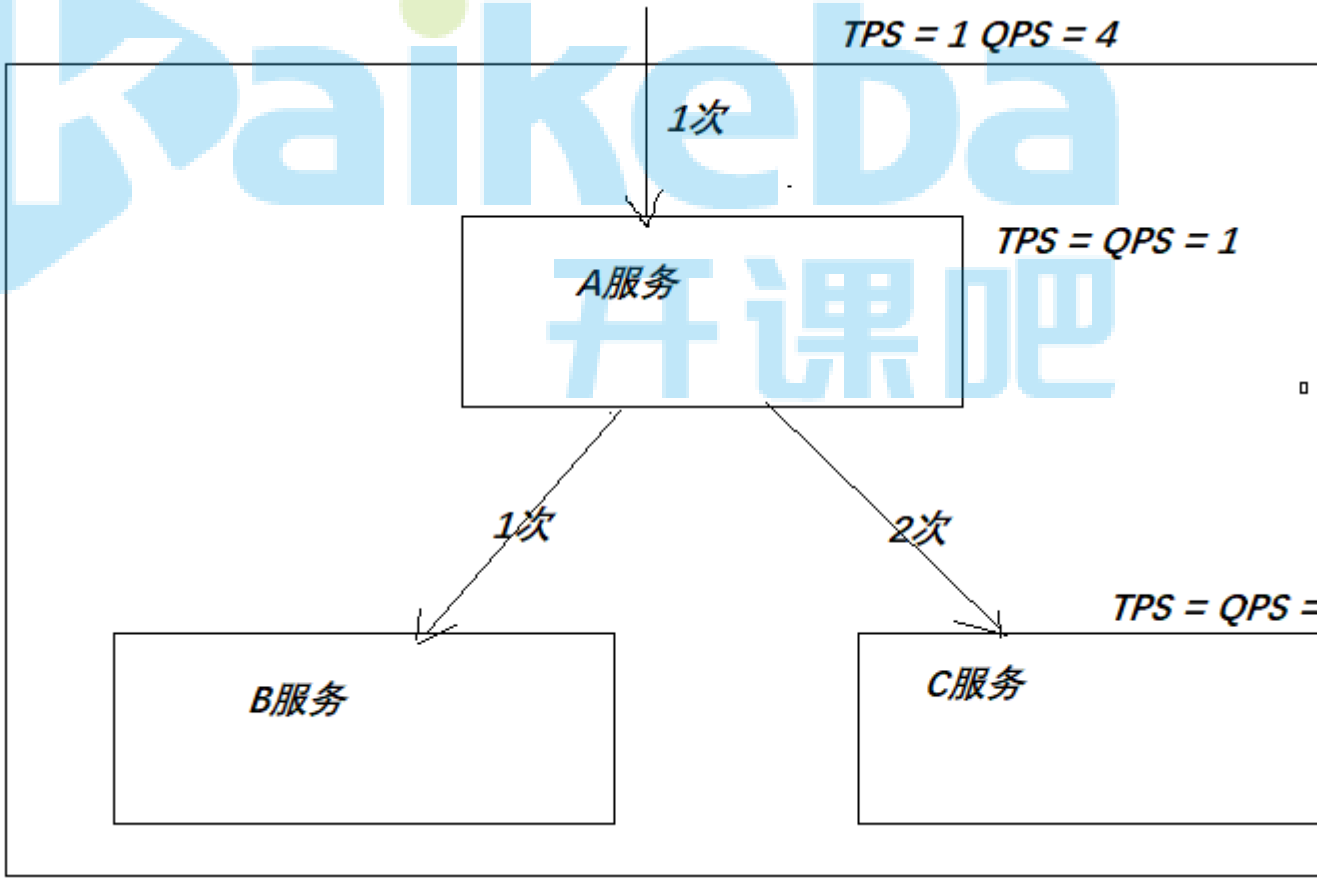
举个例子：

有一个页面（1、加载 js,2、加载 css 、 3、接口 goods/test） --- 请求/index.html

TPS : 1

QPS:3

举个例子：



TPS：一个请求从发送到响应的过程，就是一次 TPS

QPS: 每秒查询数量（每秒的请求数）

线程数：2000

Ramp-up : 5s　# 根据业务时间，判断进行设计，5s 之内启动 2000 个线程，建立 2000 个链接

循环次数：50

即使填写了 2000 个线程，发送给服务器并不一定是并发线程？？

线程数：　5 个线程

循环次数：a = 1000

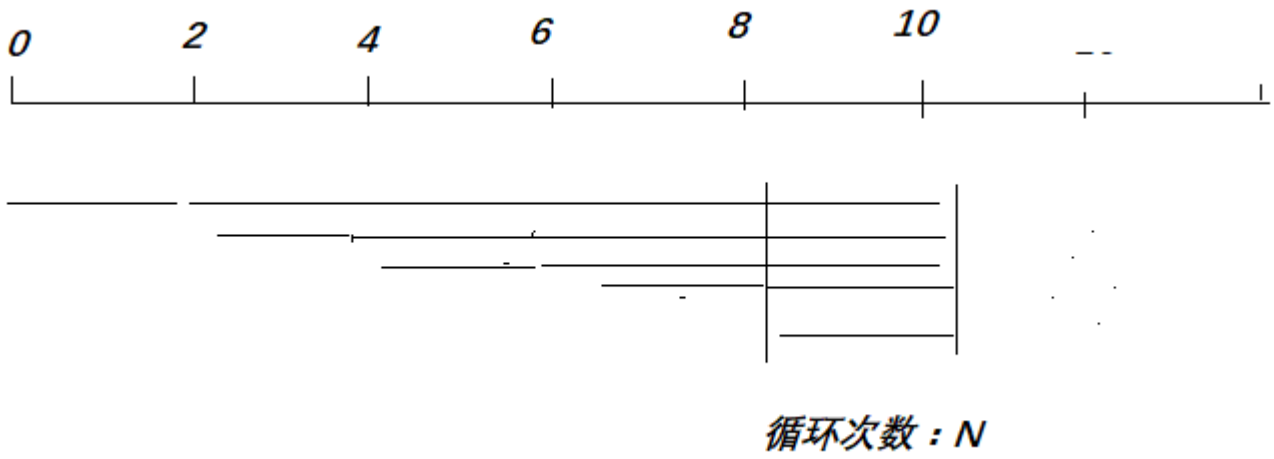
平均响应时间：t　= 0.2s

Ramp-up : T = 10s

S = (T － T/n) = 8 s

循环次数 * 平均响应时间

a * t > S ➔　a > S/t = 8 / 0.2 = 40　根据计算结果：循环次数至少要大于 40 次，才能产生并发效果。



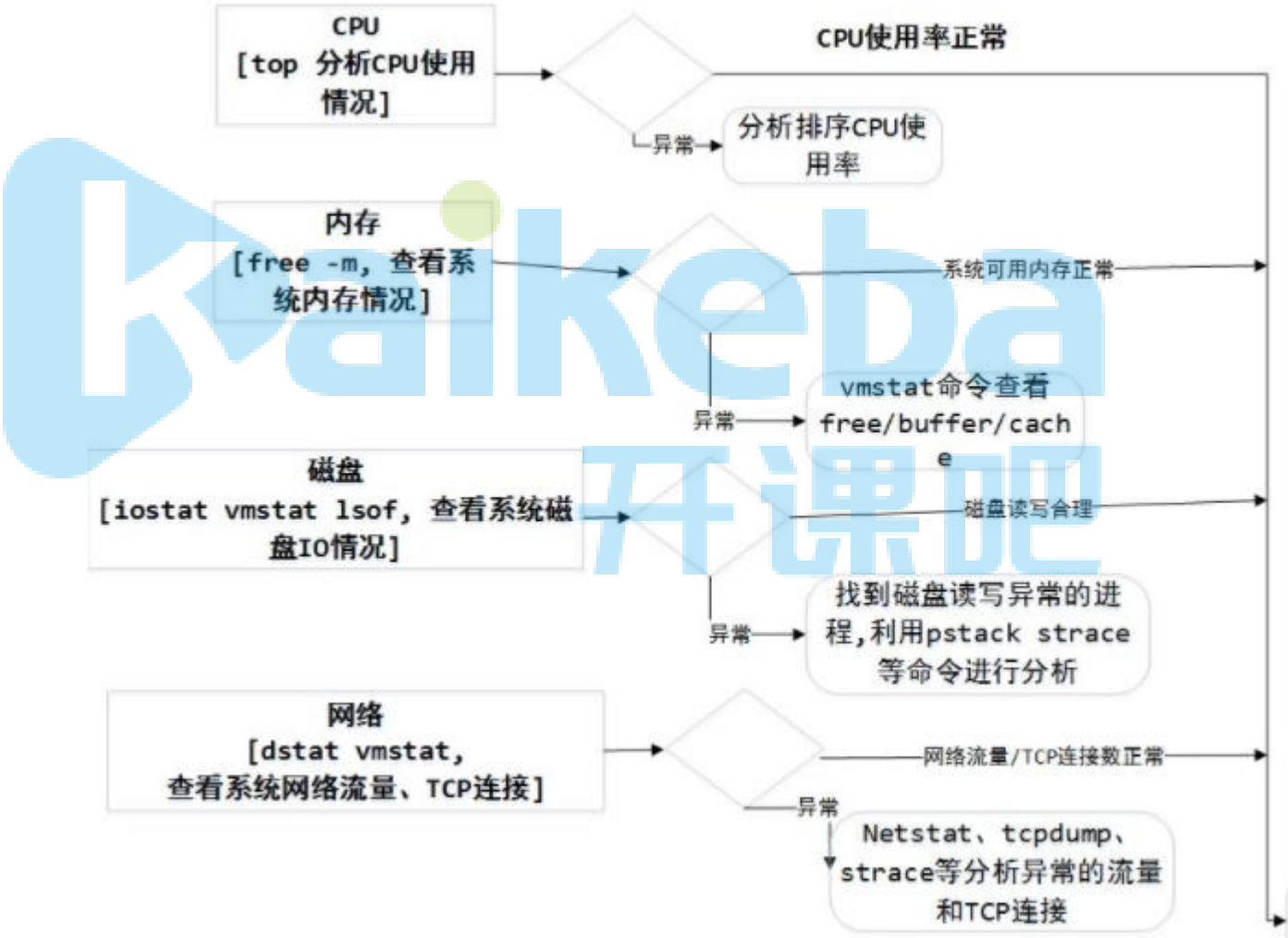
下节课:

jvm 优化（压力测试）

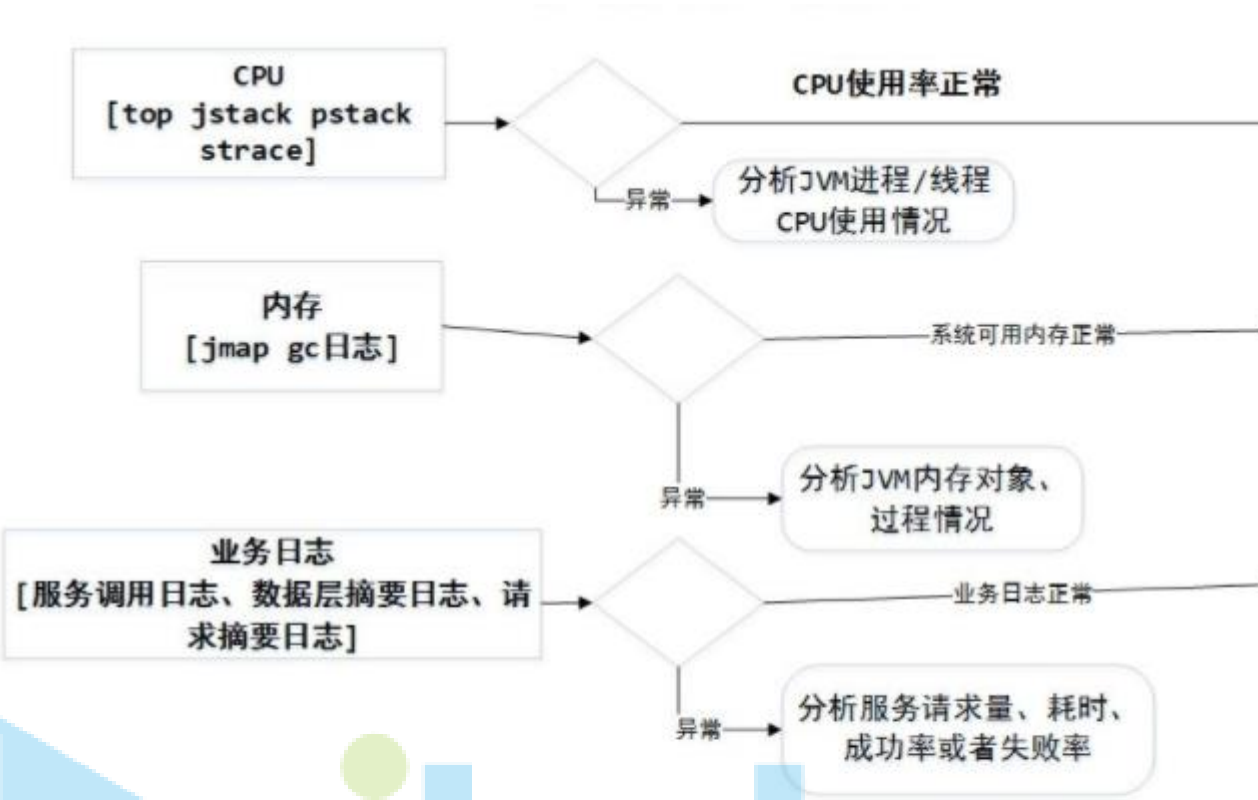
code 发给大家，笔记发给大家

kaikeba
开课吧

系统异常排查方法：



业务异常排查方法：


CPU
[top jstack pstack strace]
CPU使用率正常
异常
分析JVM进程/线程CPU使用情况
内存
[jmap gc日志]
系统可用内存正常
异常
分析JVM内存对象、过程情况
业务日志
[服务调用日志、数据层摘要日志、请求摘要日志]
业务日志正常
异常
分析服务请求量、耗时、成功率或者失败率

性能曲线:

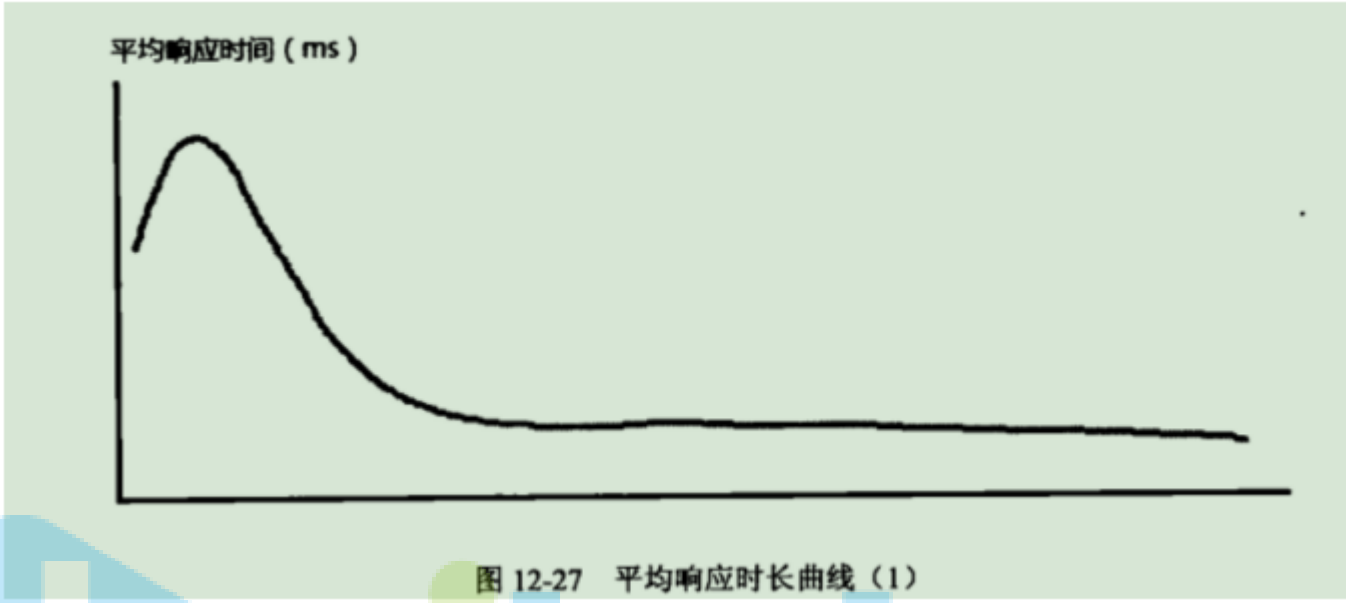


图 12-27　平均响应时长曲线（1）

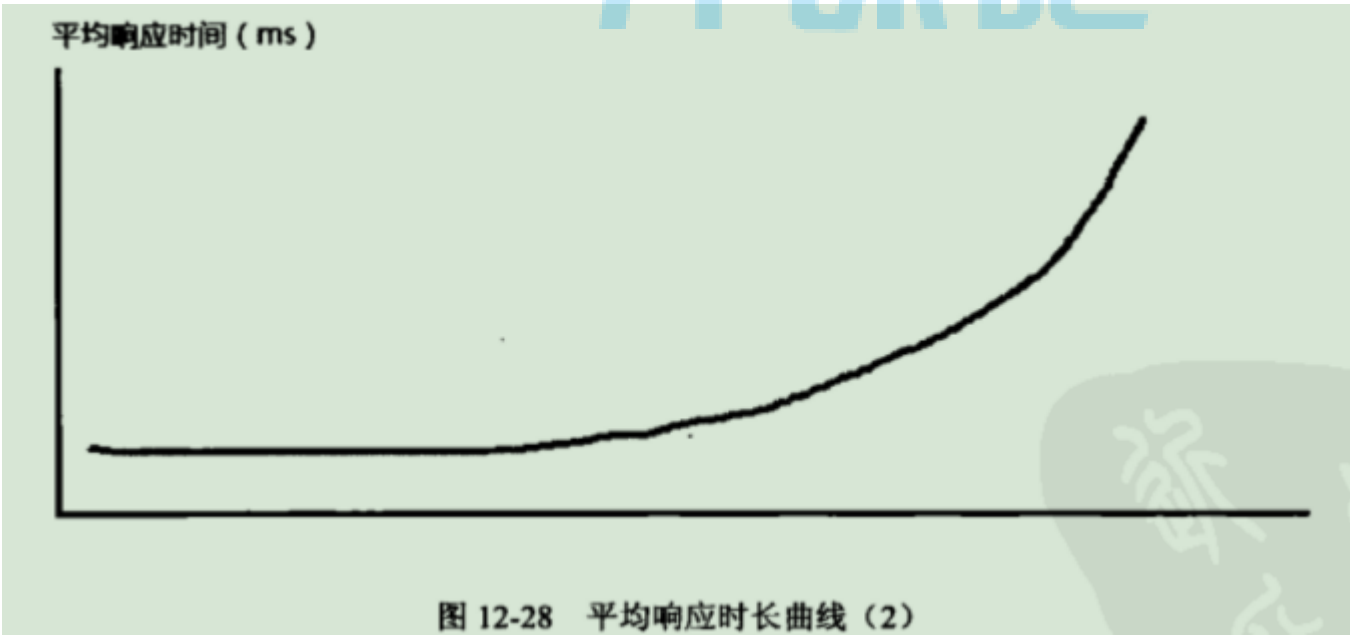


图 12-28　平均响应时长曲线（2）

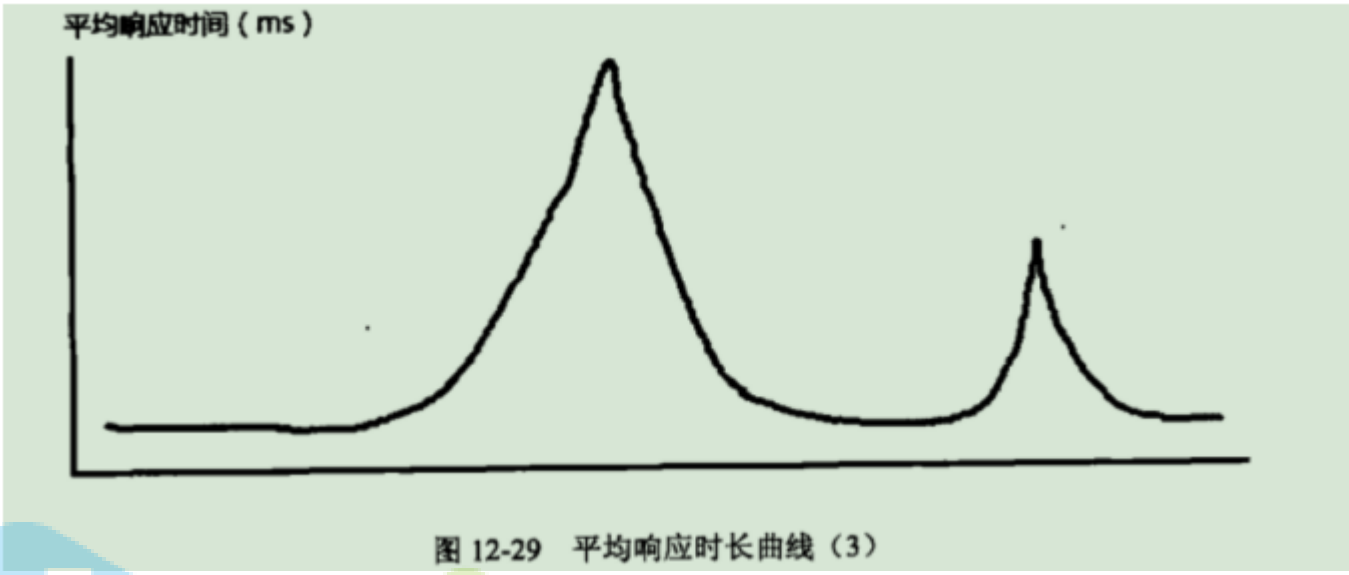


图 12-29　平均响应时长曲线（3）