

# 一、接口性能测试概念

---

(可以简单，也可以很难)

简单：只需要验证接口，符合业务需求的性能指标：系统10W用户、公司内部的项目：APP 4k人

难：精益求精

一个系统拥有多个服务程序/进程，拥有更多的接口，并不是所有接口都需要做性能测试

每个程序负责多个功能，每一个功能由单独的接口进行逻辑处理

**并不是所有接口都需要做性能测试**：因为这部分接口使用的频率较低，有时会比较少人去使用

## 性能测试的概念

---

通过使用一定的工具/手段在多并发（模拟多用户操作）的情况下，获取服务器系统各项性能指标，验证系统在高并发的处理能力（多），响应能力（快），稳定性（好）等，是否满足预期。

定位性能瓶颈，排查性能隐患，保障系统质量，提升用户体验（满足多快好省 处理请求多 响应速度快 稳定性 节省后台资源）

把服务程序当做 高铁站：

多：入口--多个入口：高并发(同时)的处理能力

快：响应能力--过高铁身份验证的速度

好：稳定性--长时间运行验证系统流畅运行

省：资源使用率少

## 按照测试种类分：

---

### 性能测试

**【满足我们的业务需求就好】**

模拟用户负载来测试系统在负载情况下，系统的响应时间、吞吐量等指标是否满足性能要求。

内部系统：公司内部4000人，做出来的系统只要满足6000人同时使用就好。

## 负载测试

### 【逐渐加压，测试出极限】

在一定/固定软硬件环境下，通过不断加大负载（设置不同虚拟用户数）来确定在满足性能指标情况下能够承受的最大用户数。简单说，可以帮我们对系统进行定容定量，找出系统性能的拐点，给予生产环境规划建议。这里的性能指标包括TPS(每秒事务数)、RT（事务平均响应时间）、CPU Using (CPU利用率)、Mem Using(内存使用情况)等软硬件指标。从操作层面上来说，负载测试也是一种性能测试手段，比如下面的配置测试就需要变换不同的负载来进行测试。

**生产环境=商用环境=线上环境**

**研发环境=开发环境**

## 配置测试

### 【服务器配置、寻找合适配置，合适的才是最好的】

服务程序：16核64G 2W 实际服务运行需求只要8核32G 5K

为了合理地调配资源，提高系统运行效率，通过测试手段来获取、验证、调整配置信息的过程。通过这个过程我们可以收集到不同配置反映出来的不同性能，从而为设备选择、设备配置提供参考。

## 压力测试

### 【极限测试，在极限状态300人同时操作，业务接口处理能力是否正常】

在一定软硬件环境下，通过高负载的手段来使服务器资源（强调服务器资源，硬件资源）处于极限状态，测试系统在极限状态下长时间运行是否稳定，确定是否稳定的指示包括TPS、RT、CPU Using、Mem Using 等。

## 稳定性测试

### 【长时间运行，业务处理能力是否正常】

在一定软硬件环境下，长时间运行一定负载，确定系统在满足性能指标的前提下是否运行稳定。与上面的压力/强度测试区别在于负载并不强调是在极限状态下(很多测试人员会持保守观念，在测试时会验证极限状态下的稳定性)，着重的是满足性能要求的情况下，系统的稳定性、比如响应时间是否稳定、TPS是否稳定。一般我们会在满足性能要求的负载情况下加大1.5到2倍的负载量进行测试。

## 什么样的系统/接口需要做性能测试

- **用户量大，pv比较高的系统**（pv page view网页浏览量、点击量 不管是否是同一个人去访问，按次数算。UV unique view 查看用户有多少次请求(去重，按用户算)） 淘宝 天猫 京东（百万/千万）
- **系统的核心模块和接口**：公司内部项目--打卡上班、下班 请假？ 汇报工作？

- **业务逻辑/算法比较复杂**：搜索商品--添加购物车--下单、火车票抢票
- **促销/活动推广计划**：双十一、年货节、618
- **公司新做 新系统、新项目**：小版本迭代--功能测试
- 线上性能问题验证和调优：线上环境出现了问题，那么就需要回过来在测试环境上进行性能测试验证
- 新技术选型：java-->python/C++
- 性能容量评估和规划：内部4000人同时使用--4500/4800
- 日常系统性能回归：

## 性能测试指标

### TPS/QPS 重点

TPS：每秒钟处理的事务数（每秒处理的业务请求量） post

QPS：每秒查询率（只查询，没有请求需要处理） get

每秒钟处理的事务数：transaction per second,query per second,性能测试领域，衡量一个系统性能的好坏，主要看单位时间内系统可以处理多少业务量，各个系统的业务各不相同，为了方便使用统一指标衡量业务的性能，用事务代表业务操作，一个事务代表一个业务，也可以代表多个业务操作，事务是用户人为定义的，测试什么业务的性能就把该业务加到事务中

举例：事务就是从头到尾的操作流程看作一个事务/事件（人为定义）

单个业务：下单---事务

多个业务：搜索--添加购物车--下单--付款 -- 事务

### 响应时间 重点

#### 平均响应时间

一个请求的响应包含哪些时间



网络设备：路由器

中间件：平台+通信，Apache、PHP

应用程序：比如fanwe

数据库：处理数据

响应时间=网络传输总时间+各组件业务处理时间

平均响应时间：在测试过程中，所有每次请求的平均耗时 25/30/35

#### top响应时间

算法：对于所有的请求都从大到小排列，计算指定比例请求小于某个时间，该指标统计的是大多数的耗时

Tp90（90%的响应时间）90%的请求耗时都低于某个时间

Tp95（95%的响应时间）95%的请求耗时都低于某个时间

Tp99（99%的响应时间）99%的请求耗时都低于某个时间

拓展(了解)：性能指标：

对于网络的请求：2,5,8秒，这种说法不恰当，能满足需求就好

大公司服务接口(内网)：100ms以内，小公司项目接口200~300ms 以上 5 S

基于协议的性能测试

## 并发用户数 重点

**系统用户数**：在系统里面创建了多少个成员（数据库中有记录的用户数）

**在线用户数**：在线的用户但是没有进行具体的操作（登陆之后的用户，创建了session会话，但不一定有实时操作。如挂QQ）

**并发用户数**（抢购）：在一个节点内同时进行操作的用户（同时间段（几秒内、十几秒内、几十秒内）操作）

压测工具中设置的并发线程/进程数量，指的是后端服务端支持的并发用户，而不是指单个客户端支持的并发用户数。（总结：接口性能测试，是针对后台服务的接口，不依赖客户端）

拓展（了解）：进程和线程的区别：

进程：操作系统资源的分配

线程：任务的调度和执行

进程：公司 线程：员工

【一个进程内可以拥有多个线程、但至少有一个线程】

## 成功率 重点

请求成功率，跟异常率是互补的

## PV（page view）

页面/接口的访问量，强调的是页面的访问次数、访问量、浏览量：10个人操作了500次，PV =500

## UV (unique visitor)

页面/接口的唯一接口，强调的都是不同用户的请求：同一个用户的操作，都属于同一UV。10个人不管操作多少次（成千上万次），UV都是10

## 吞吐量 重点

网络上行和下行的流量总和（发送的请求+接收的响应 总流量），吞吐量代表网络的流量，TPS越高，吞吐量越大

交互应用：吞吐量越大的是服务器承受的压力越大，说明系统负载的能力越强

性能测试工具直接会统计出吞吐量，不需要人工去计算

拓展(了解)：计算公式： $F=VU \times R/T$

F:吞吐量

V U：虚拟用户的个数 virtual user

R：代表每个虚拟用户发出的请求数

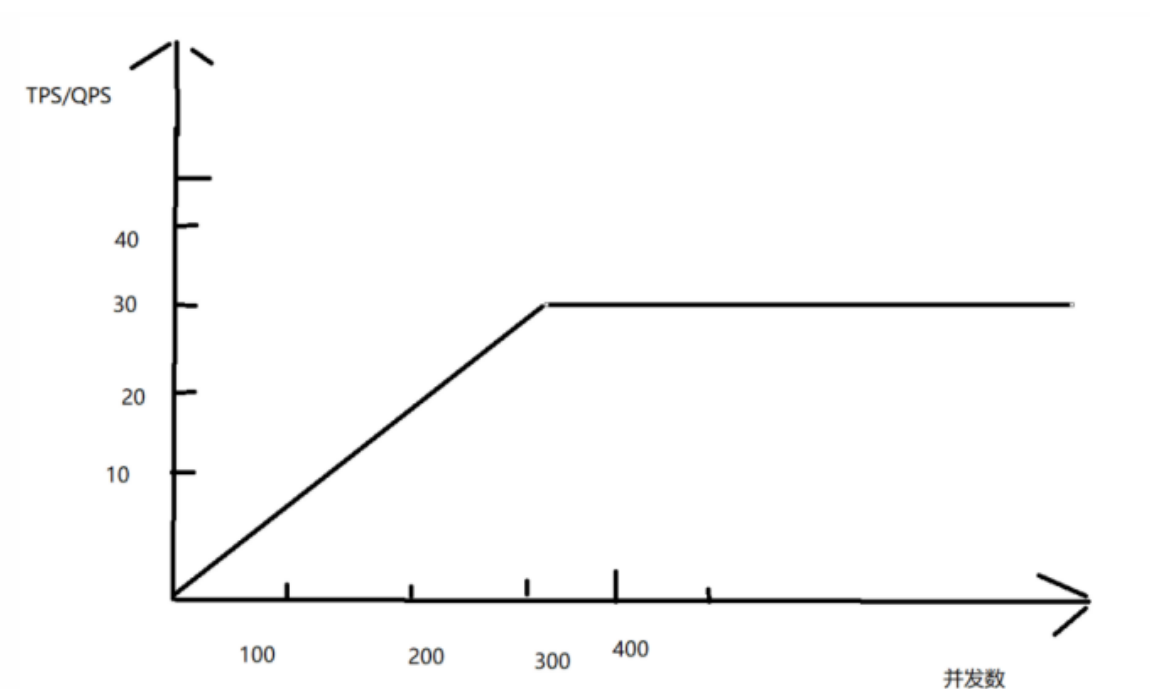
T：代表性能测试使用的时间

## TPS 响应时间 并发数的关系（主要应用在负载测试）

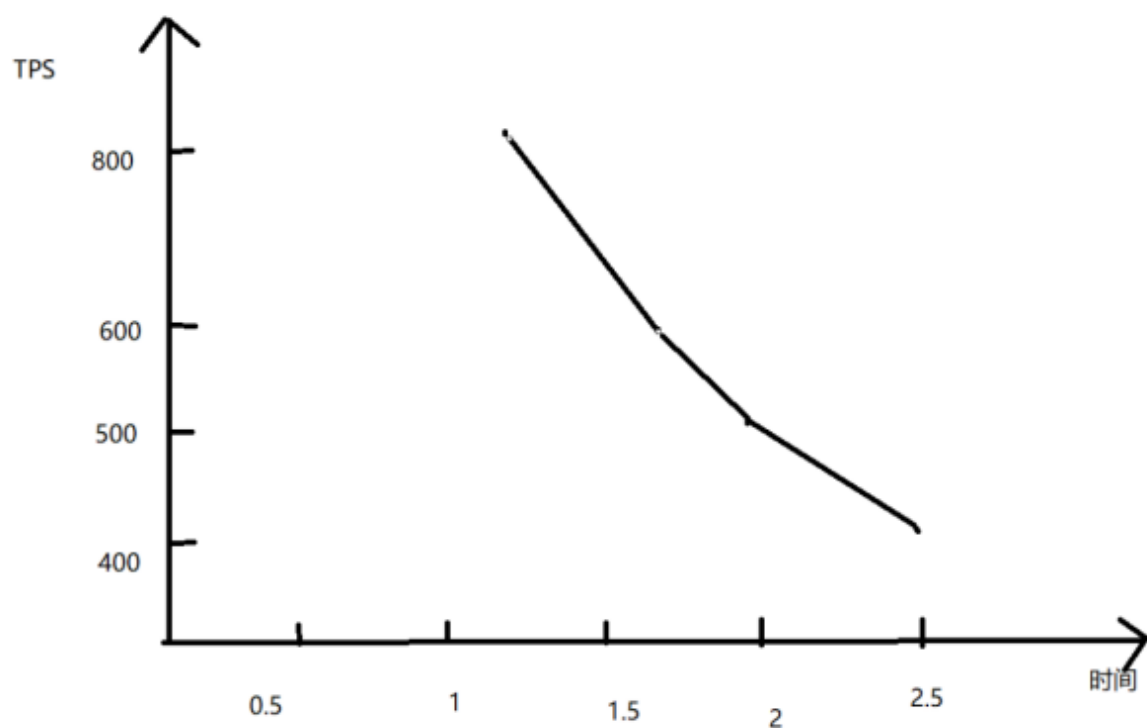
【系统达到极限瓶颈之前，tps和并发数成正比关系】

G	H	I
	并发用户数	TPS/QPS
以饭堂为例： 总的窗口40	10	10
	20	20
	30	30
	40	40
	50	40
	60	40
	100	40

系统达到瓶颈之前，tps和并发数成正比关系（下图）



并发数不变的情况下，tps和响应时间成反比关系 TPS越大 时间越短 TPS越小 时间越长



响应时间为秒的情况下：TPS = 1/响应时间\*并发数 或者 并发数/响应时间

## 磁盘的I/O

## 【不同服务器配置有不同的磁盘读写速度】

访问应用离不开系统的磁盘数据的读写，IO读写的性能直接会影响系统程序的性能。读写的性能直接会影响系统程序的性能，磁盘IO系统是系统中最慢的部分。主要是CPU处理频率较磁盘的物理操作快几个数量级，如果拿读取磁盘和内存的时间作比较就是分钟级到毫秒的区别。

# 性能测试流程

---

## 需求调研

- 项目调研
- 测试范围：**并不是所有模块/接口都需要做性能测试**
- 业务逻辑&数据流向：数据从哪里产生，流向哪里（有哪些模块用得上）
- 系统架构（中间件信息 需要找开发了解）
- 配置信息（服务器硬件配置）--项目组（架构师跟开发确定）  
服务器配置：几核几G
- 测试数据量（量级要一致：要在测试环境制造跟线上环境一样的数据）
- 外部依赖（支付宝、微信接口mock：支付要模拟真实用户支付。拓展：mock沙箱，将测试环境与真实环境隔离开来，所有的外部接口，都直接返回处理成功）
- 系统使用场景，业务比例（核心用户场景的设计，以及数据流向）
- 日常业务量
- 预期指标
- 上线时间：性能测试通过，才能上线

## 性能测试计划

（工作中是有模板，“抄作业”）

可能包含的内容：

- 项目描述
- 业务模型(项目组--产品)以及性能指标（所有的需求来源：项目组--产品进入，测试跟开发说了都不算）
- 测试环境说明
- 测试资源
- 测试方法以及场景设计原则

**基准测试：**找到性能基准数据（能满足业务需求性能指标）

**单交易负载测试：**单接口的测试，单交易接口先做 重要

搜索

下单

支付

查看订单详情

**混合场景测试：**需要在需求调研中分清楚比例 重要

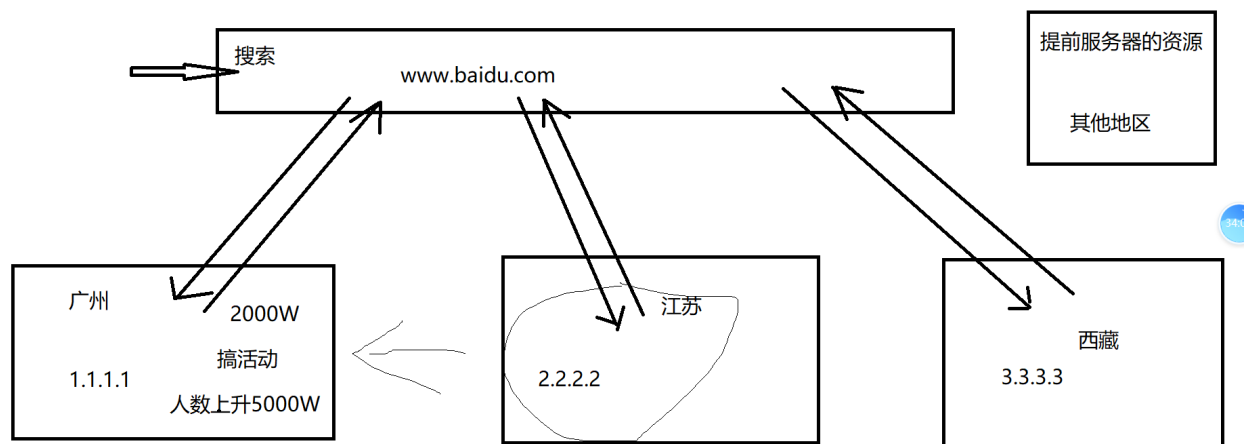
搜索--下单--支付--查看订单详情

(使用动态关联)

(不同核心业务场景，对应的接口性能指标不同，比如抢购时的添加购物车、下单支付要求在100ms内，而次要商品说明/评论，要求可以放宽到1000ms)

**高可用性测试：**集群测试

(集群作用：可以把多台服务器连接到一起，把用户的请求平均分布到每一台服务器上面，以此分担单台服务器的压力)



异常场景测试：比如考虑一些弱网场景：linux延迟时间注入（数据库读取数据速度差异）

稳定性测试：长时间运行混合场景的性能测试 重要

其他特殊场景：

- 测试进度安排以及测试准则

## 环境搭建

注意要点：

1. 测试服务器硬件配置尽量和线上环境(商用环境)一致：测试环境、线上环境。

如果不一致，那么测试要求按配置比例缩减

举例，线上：16核32G 测试环境：8核16G，4核8G，

线上4000，测试环境 2000，1000

线上10W，测试环境5W，2.5W

2. 操作系统(代码服务程序)版本和线上一致 线上：V 7.2.1 测试环境：不能是V7.1 V7.5 运维操作
3. 测试环境部署线上最小模块：不关联的模块没必要部署，不相关，非此次项目的系统不要部署

(很多公司都会出现一种情况：多个项目公用服务器，



尽可能项目有独立的服务器，如果没有，那么可以把该服务器上的其他服务先停掉)

4. 应用程序、中间件、数据库配置与线上一致

5. 其他特殊配置

初始化环境关注点：

网络：

比如数据库的查询，两台在一起的服务器和两台不在一起的时候，服务器查询出来的数据耗时是不一样的，比如：A在北京的服务器，B在广州，两者的请求的时间是完全不一样的，解决方案：运维使用linux系统注入网络延时方法解决

磁盘空间：200M 10G（磁盘空间不足，需要定期清理日志文件）

多或者少完全不一样的

## 在线用户数

初始化环境：warm-up

热机(准备运动)：从磁盘跑入内存先将数据调用出来 仿真环境

个人电脑刚开机是比较忙，就行因为开机时进程刚起来，一段时间后恢复正常

## 数据准备

（在测试环境量级，跟线上一样）

通常使用一下三种方法进行构造(准备数据)：

**业务接口（功能/接口）：**

（通过相应的接口生成数据）

--适合数据表关系复杂

--优点：数据完整性比较好

--缺点：费时费力

100ms注册一个用户 1秒10 造10w数据 需要1万秒 2.7小时

**存储过程：**

--适合关联表 数量少，简单

--优点：速度最快

**脚本导入：**

（让运维人员把线上生产环境数据导出来给我们，测试人员直接导入测试环境）

要求提供的脚本类型：SQL。敏感信息需要脱敏--把真数据改成假数据)

--适合数据逻辑复杂

--自由度比较高

## 性能测试脚本编写

### 工具选择

loadrunner收费软件、**Jmeter (免费开源)**、locust(python压测)

### 选择协议

http TCP RPC

参数化

关联

检查点

事务判断(断言)

## 压测执行

**项目并发用户数：5000满足大多数项目 单台电脑模拟的并发用户建议不超1000，甚至600以内**

分布式执行

如果是单机电脑 那么一台机器就可以

如果是集群测试，那么需要对应的1:1来进行测试， 几台服务器--几台电脑

监控--jmeter聚合报告

收集测试结果--简单的性能测试：满足业务需求即可

精益求精：数据分析、瓶颈定位

## 调优回归

性能调优（优化，提升）需要整个项目团队来完成：项目经理牵头，组织开发、测试、运维

反复尝试

回归验证

监控工具

全链路排查

日志分析

模块隔离：逐个模块隔离开来去判断对接口响应时间是有有影响

## 测试报告

(可能包含的内容, “抄作业”)

概述

测试环境

结果与分析

调优说明

项目时间表

结论

建议

## 现状与趋势

性能自动化, 平台化 --- 更好用, 方便测试, 才好收费

测试工具多样性: 开源(jmeter) 二次开发

在高并发下验证功能正确性

线上 (商用--真实用户) 线下 (测试数据) 相结合 线上发现问题 线下调优

## 二、性能测试操作:

---

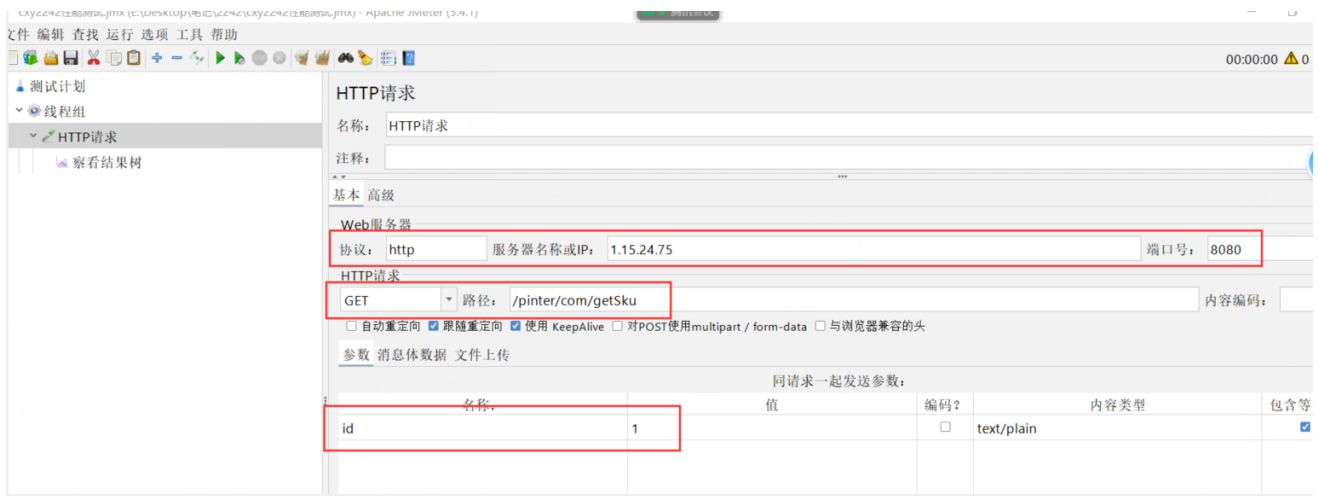
接口信息:

请求行: <http://1.15.24.75:8080>

请求方式: GET

请求期路径: /pinter/com/getSku

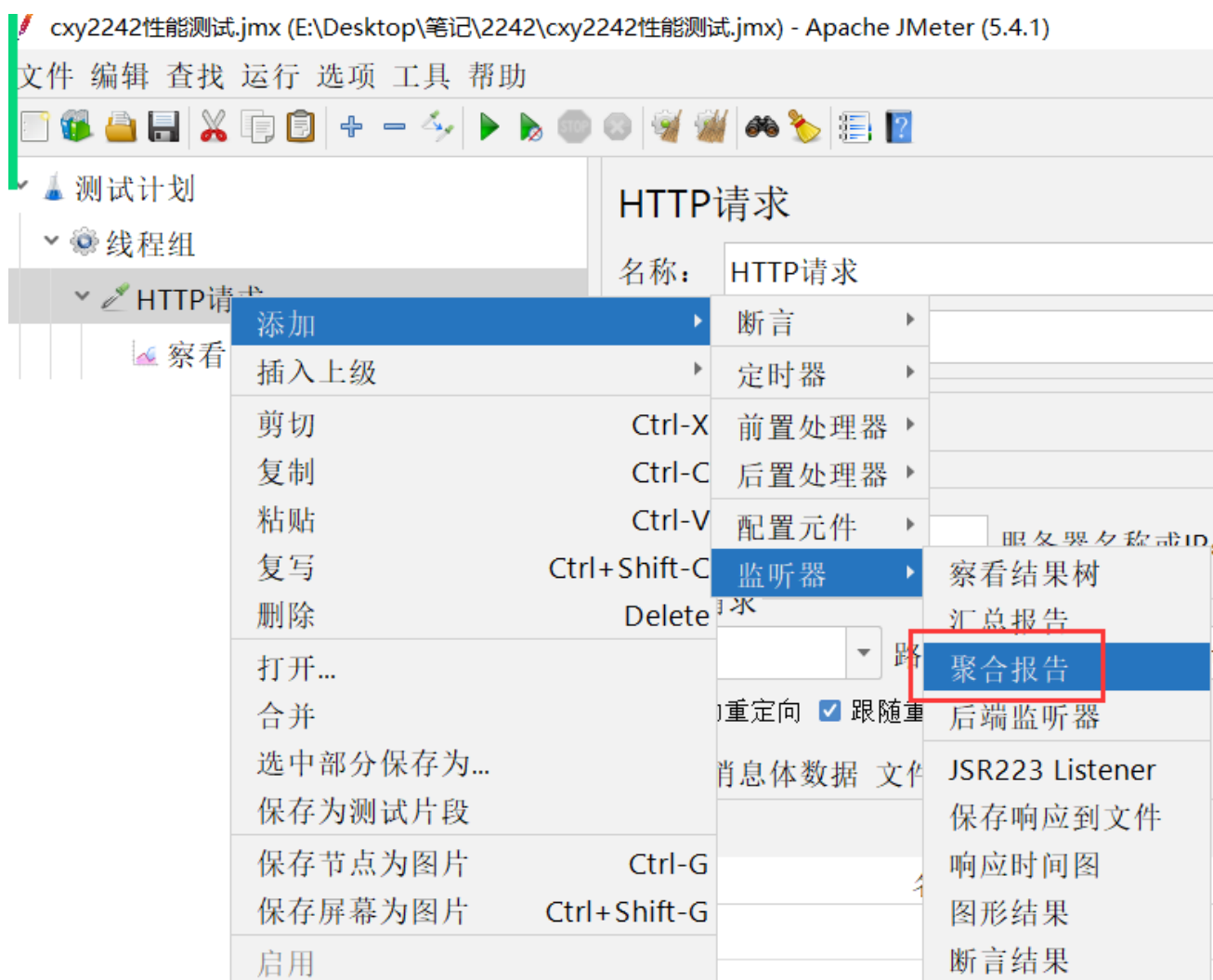
请求参数: id=1



## 线程组参数解释:



添加聚合报告: 查看接口性能指标



## 聚合报告的数据查看说明

聚合报告

名称：聚合报告

注释：

所有数据写入一个文件

文件名

浏览...

显示日志内容：

☐ 仅错误日志

☐ 仅成功日志

配置

Label	# 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 %	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	31845	54	35	75	104	366	28	7723	0.00%	2335.4/sec	579.07	323.85
总体	31845	54	35	75	104	366	28	7723	0.00%	2335.4/sec	579.07	323.85

发送的请求

中间请求的响应时间

top95响应时间

最小的响应时间

失败率/错误率

吞吐量(上行+下行)

平均响应时间

top90响应时间

top99响应时间

最大的响应时间

TPS

TPS==QPS

## 怎么判断接口性能指标是否符合业务需求?

性能指标: TPS, 达到多少可以需求?

- 1、产品告知需求文档:
- 2、通过历史数据来判断 (适用于后续的版本迭代)

今年618活动，拿去年618（双十一、促销抢购活动）活动接口性能指标参数来做参照。

去年用户量 10W，TPS --2300

今年用户量15W，TPS ---3500

3、项目新成立：没有历史数据，只能预估

二八原则：每天80%用户点击量访问量 集中在 每天20% 时间段内；

计算公式：（总PV x 80%） / (24x60x60x20%) = 峰值每秒处理事务数/查询率 峰值TPS/QPS

购物电商：2000W系统用户，一天可以贡献 PV：5000W

$(50000000 \times 80\%) / (24 \times 60 \times 60 \times 20\%) = 2320$

(有备用服务器，以防万一，可以随时满足线上需求)

## 定位接口性能瓶颈：

测试时间15~30分钟

并发用户数 峰值TPS/QPS

第一次：200 2300?

第二次：300 3300

第三次：400 3200

第四次：700 3300

(操作要点：

1、刚开始性能测试的时候，可以三五十/成百的增加，

2、待TPS平稳之后，查看有无错误率，

3、没有的话可以即刻停止，增加并发用户，再次发起测试，直到出现错误率，或者出现性能瓶颈

如果有错误率，那么说明并发用户数过高，需要适当减少，直到没有错误率

错误率只要不超0.05%可以忽略)

通过尝试增加并发用户数，发现平稳下来之后的TPS都在3300左右，

结论：该接口的TPS瓶颈为3300

方维登陆接口：

并发用户数 TPS/QPS

第一次：10 19  
第二次：15 11  
第三次：20 11  
第四次：30 31  
第五次：50 16  
第五次：100 29  
接口需要进行优化改善

### 三、定时器与逻辑控制器的操作

#### 1、固定定时器：

作用：可以设置每个请求固定等待的时间，等待设置时间后才开始执行。

测试计划

线程组

固定定时器

方维登陆

HTTP请求

名称：方维登陆

注释：

添加

插入上级

剪切

复制

粘贴

复写

删除

打开...

合并

选中部分保存为...

保存为测试片段

保存节点为图片

Ctrl-X

Ctrl-C

Ctrl-V

Ctrl+Shift-C

Delete

Ctrl-G

断言

定时器

前置处理器

后置处理器

配置元件

监听器

路径：

☐ 跟随重定向

本数据 文件上传

固定定时器

统一随机定时器

Precise Throughput Timer

Constant Throughput Timer

JSR223 Timer

Synchronizing Timer

泊松随机定时器

高斯随机定时器

BeanShell Timer

名称：

test

## 固定定时器

名称： 固定定时器

注释：

线程延迟（毫秒）： 1000

设置延迟发送请求的时间

## 线程组

名称： 固定定时器

注释：

在取样器错误后要执行的动作

☒ 继续 ☐ 启动下一进程循环 ☐ 停止线程 ☐ 停止测试 ☐ 立即停止测试

线程属性

线程数： 1

Ramp-Up时间（秒）： 1

默认

循环次数 ☒ 永远

☒ Same user on each iteration

☐ 延迟创建线程直到需要

☒ 调度器

持续时间（秒） 10

设置运行10s

启动延迟（秒）

## 2、同步定时器：

作用：可以使请求达到设置用户数之后在同一时刻发起请求。

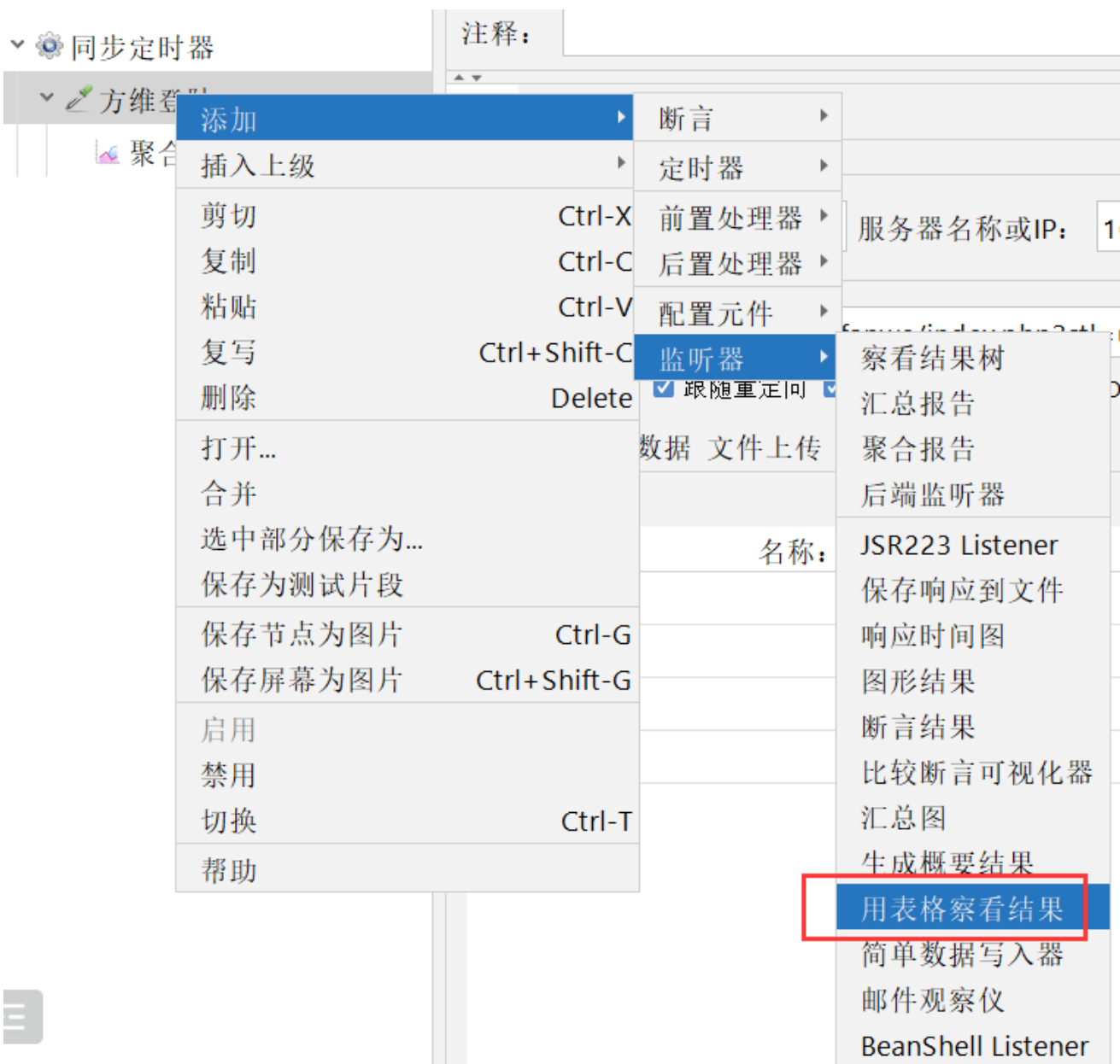
场景：抢购、抢红包

数量5份，抢购人员超过30人，设置在同一时间毫秒内有超过5个用户的请求，看是不是只有5个抢购成功。

（断言的方式去看是否就5个断言成功，而其他是断言失败）

使用表格查看结果





用表格察看结果

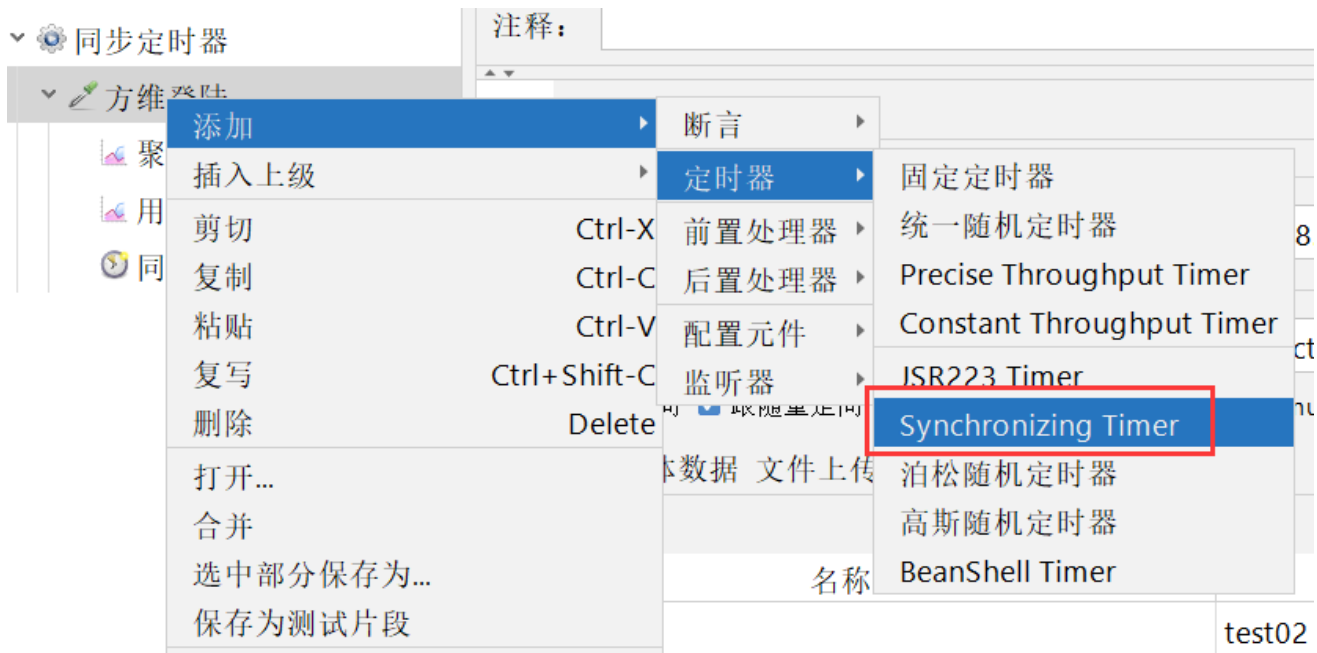
名称： 用表格察看结果

注释：

所有数据写入一个文件——没有按照序号的先后顺序进行发送  
文件名 发送的时间是有间隔

Sample #	Start Time ↑	Thread Name	Label	Sample Time(m..
1	17:23:48.575	同步定时器 1-1	方维登陆	187
2	17:23:48.606	同步定时器 1-2	方维登陆	257
3	17:23:48.639	同步定时器 1-3	方维登陆	277
5	17:23:48.673	同步定时器 1-4	方维登陆	295
4	17:23:48.706	同步定时器 1-5	方维登陆	262
6	17:23:48.739	同步定时器 1-6	方维登陆	296
9	17:23:48.756	同步定时器 1-1	方维登陆	329
7	17:23:48.772	同步定时器 1-7	方维登陆	263
8	17:23:48.806	同步定时器 1-8	方维登陆	279
10	17:23:48.837	同步定时器 1-9	方维登陆	248
11	17:23:48.858	同步定时器 1-2	方维登陆	228
39	17:23:48.871	同步定时器 1-10	方维登陆	1236
36	17:23:48.904	同步定时器 1-11	方维登陆	1156
12	17:23:48.910	同步定时器 1-3	方维登陆	317
52	17:23:48.939	同步定时器 1-12	方维登陆	1680
13	17:23:48.968	同步定时器 1-4	方维登陆	370

添加同步定时器



## 同步定时器

名称: 同步定时器

注释:

分组

模拟用户组的数量: 10

同时发送请求数量 (每10个为一组进行并发请求)

超时时间以毫秒为单位: 500

最长等待时间, 超出这个时间还没用够设置的用户数, 那么就不再进行等待

可以按照设置的用户数为一组进行发送请求

Sample #	Start Time ↑	Thread Name	Label	Sample Time(m...	Status
135	17:32:29.259	同步定时器 1-16	方维登陆	707	✓
138	17:32:29.259	同步定时器 1-14	方维登陆	710	✓
125	17:32:29.286	同步定时器 1-30	方维登陆	675	✓
127	17:32:29.286	同步定时器 1-7	方维登陆	677	✓
136	17:32:29.286	同步定时器 1-26	方维登陆	680	✓
120	17:32:29.287	同步定时器 1-29	方维登陆	673	✓
121	17:32:29.287	同步定时器 1-24	方维登陆	673	✓
129	17:32:29.287	同步定时器 1-17	方维登陆	676	✓
130	17:32:29.287	同步定时器 1-6	方维登陆	677	✓
137	17:32:29.287	同步定时器 1-22	方维登陆	682	✓
139	17:32:29.287	同步定时器 1-13	方维登陆	682	✓
140	17:32:29.287	同步定时器 1-27	方维登陆	738	✓
142	17:32:29.961	同步定时器 1-12	方维登陆	828	✓

### 3、常数吞吐量定时器

作业：可以让接口以固定TPS来运行

在实际工作中，有些时候是不需要让TPS有太大的波动，或者不需要太高的TPS来进行稳定性测试(长时间运行)，可以使用常数吞吐量来固定TPS。

常数吞吐量定时器

Web服务器

方维登陆

添加

插入上级

剪切

复制

粘贴

复写

删除

打开...

合并

选中部分保存为...

断言

定时器

前置处理器

后置处理器

配置元件

监听器

名称:

名称或IP: 106.52.182.140

固定定时器

统一随机定时器

Precise Throughput Timer

Constant Throughput Timer

JSR223 Timer

Synchronizing Timer

泊松随机定时器

高斯随机定时器

BeanShell Timer

## 常数吞吐量定时器

名称：常数吞吐量定时器

注释：

在每个受影响的采样器之前延迟

目标吞吐量（每分钟的样本量）：600

每分钟600，那么TPS是每秒600/60=10

基于计算吞吐量：只有此线程

## 聚合报告

名称：聚合报告

注释：

所有数据写入一个文件

文件名

浏览...

显示日志内容：

☐ 仅错误日志

☐ 仅成功日志

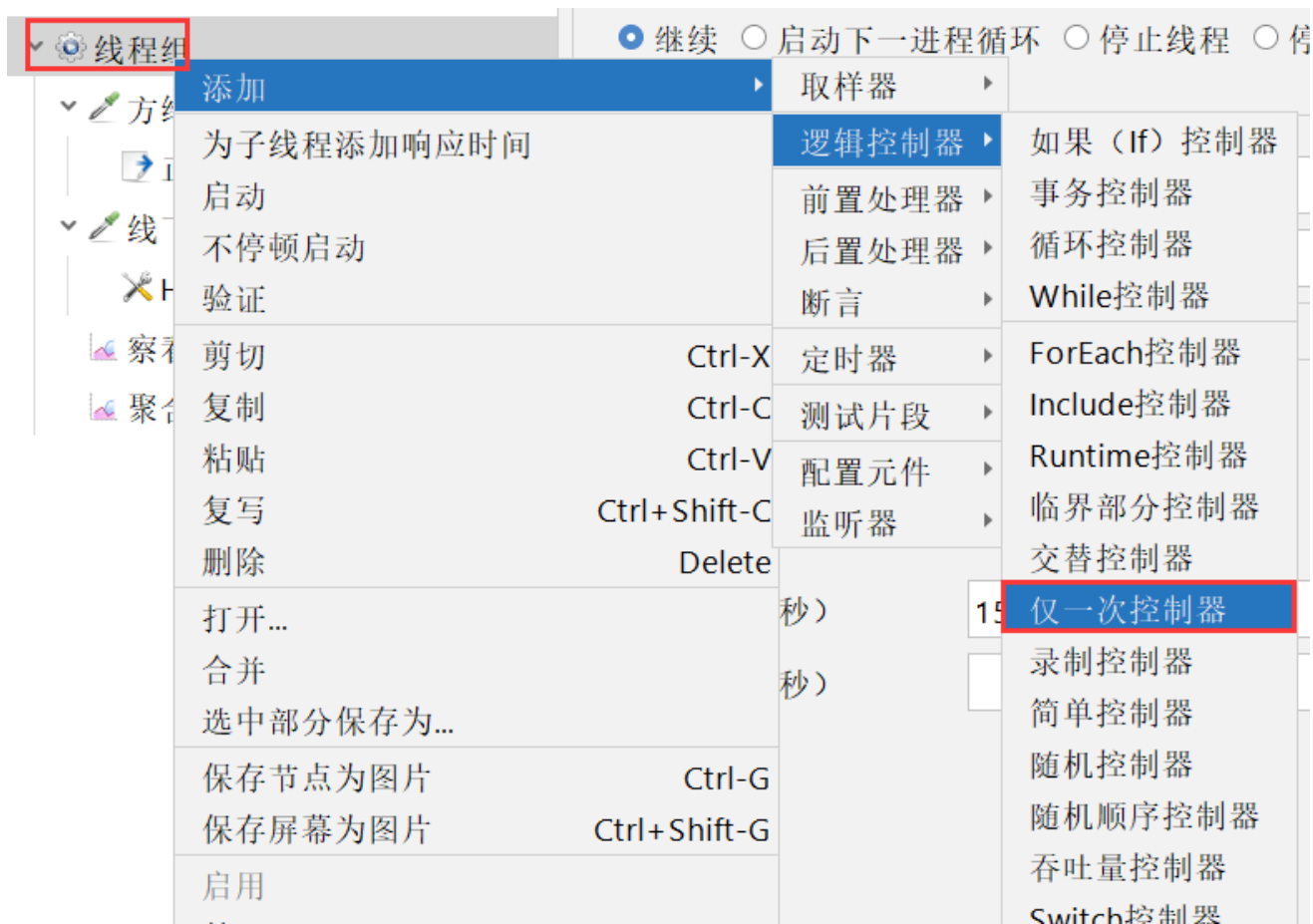
配置

Label	# 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 %	吞吐量	接收 KB/sec	发送 KB/sec
方维登陆	44	95	95	97	97	120	93	120	0.00%	10.0/sec	8.30	4.50
总体	44	95	95	97	97	120	93	120	0.00%	10.0/sec	8.30	4.50

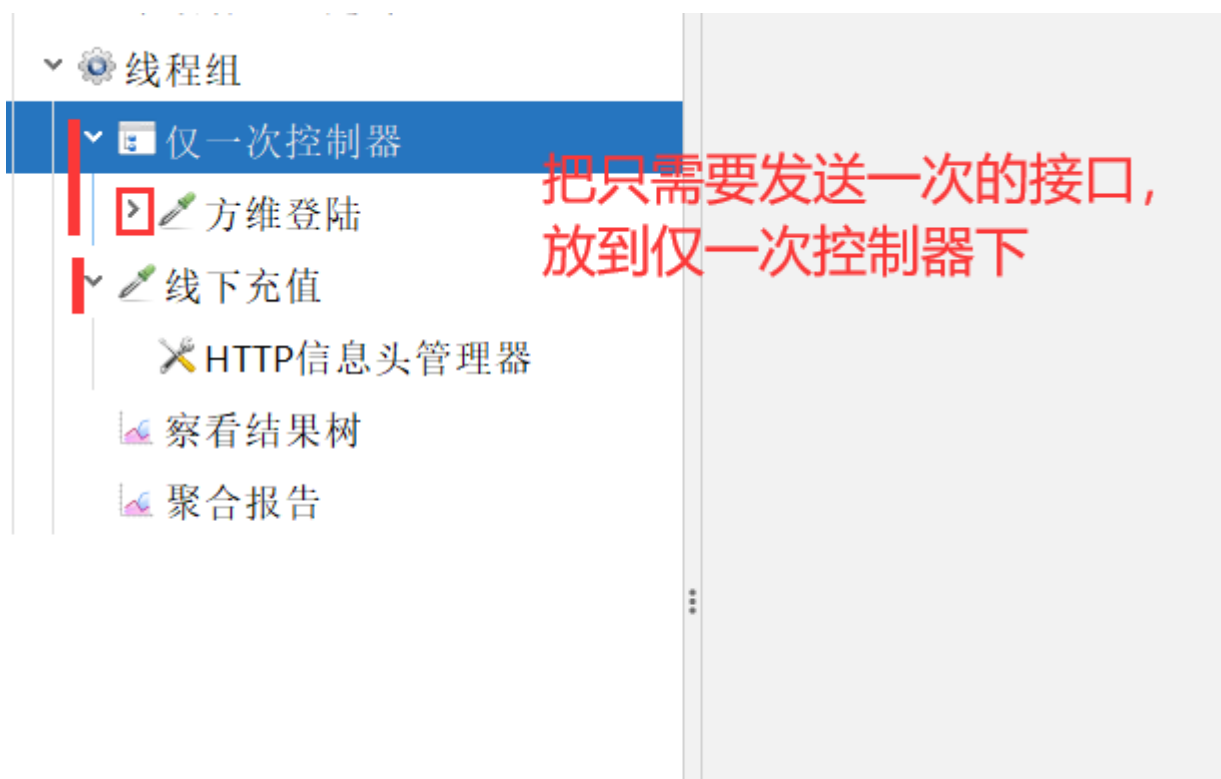
## 4、仅一次逻辑控制器：

在实际工作中如果用上动态关联，在并发多线程请求的时候，会将上下两个接口都进行多并发

仅一次控制器：可以使其中一个接口不会进行多并发请求



注意下从属关系：



查看操作结果：

只登陆了一次，而充值了多次

聚合报告

名称：聚合报告

注释：

所有数据写入一个文件

文件名

浏览...

显示日志内容：

☐ 仅错误日志

☐ 仅成功日志

配置

Label	# 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 %	吞吐量	接收 KB/sec	发送 KB/sec
方维登陆	1	3302	3302	3302	3302	3302	3302	3302	0.00%	18.2/min	0.25	0.14
线下充值	9	3007	3062	3403	3463	3463	2311	3463	0.00%	19.9/min	5.41	0.19
总体	10	3037	3062	3403	3403	3463	2311	3463	0.00%	19.8/min	4.85	0.19

# 四、性能测试的监控

## 为什么做性能监控？

出现性能瓶颈，需要对综合详细的数据对监控的数据进行分析，整个系统架构中的每一个环节都需要监控（压力机，网络，中间件，服务器硬件资源等）性能监控做好了就可以快速定位问题，找到问题的瓶颈。

压力机：对服务接口产生压力的机器

性能监控：对服务接口进行性能测试，监控就是监控该服务接口所在的的服务器

举例：对方维的接口进行测试，监控方维接口所在服务器

**压测哪台服务器，监控该服务器**

监控步骤：

- 1、在性能测试之前，先连接上服务器，使用命令查看服务的CPU、内存 状态
- 2、在性能测试过程中，再次进行查看
- 3、做对比，进行判断

总结：程序代码、服务器，只有两者的质量都高的情况下，性能指标才会高，但其中一个质量不行，那么性能指标会受影响（类似木桶原理）

如果代码质量OK，但服务器配置低，那么运行起来，CPU会接近100%使用情况，那是因为服务器配置跟不上；需要换服务器

如果CPU使用不高，没有跑满接近100%，但接口响应出现了异常报错，那么就可能是服务代码的问题。如果没有出现异常报错，那么可以增加并发用户数，去查看TPS有无增加。

# 操作系统级别的监控（掌握）

## CPU使用率

反映系统服务器的cpu繁忙程度

## 内存使用率

（使用内存，决定了可以运行多少服务）

反映系统内存的使用空间

## 磁盘I/O

反映磁盘的读写状态

## 操作系统监控--top命令--CPU检测

主要对cpu，内存，进程监控（在top命令下，按数字1，可以查看每个CPU使用情况）

load average 所有的等待和正在执行数据之和

total: 任务

us:用户空间占比

sy:内核空间占比

id 代表空闲cpu占比 idle

wa 代表io wait 如果比较高需要排查网络和磁盘

hi:硬中断 硬件中断 比如键盘或者鼠标、硬盘/U盘直接硬插拔导致的中断

si: 软中断 软件中断，其他服务程序影响导致的中断

性能测试之前

```
top - 14:18:59 up 3 min, 1 user, load average: 0.04, 0.03, 0.01
Tasks: 142 total, 2 running, 140 sleeping, 0 stopped, 0 zombie
%Cpu0 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu1 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 3861292 total, 3295984 free, 377868 used, 187440 buff/cache
KiB Swap: 2097148 total, 2097148 free, 0 used. 3263608 avail Mem
```

性能测试过程中:



```
top - 14:24:04 up 8 min, 1 user, load average: 22.00, 7.25, 2.58
Tasks: 165 total, 32 running, 133 sleeping, 0 stopped, 0 zombie
%Cpu0  : 85.2 us, 14.5 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st
%Cpu1  : 84.3 us, 15.7 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2  : 85.3 us, 13.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 1.7 si, 0.0 st
%Cpu3  : 84.7 us, 15.3 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 3861292 total, 3010580 free, 588352 used, 262360 buff/cache
KiB Swap: 2097148 total, 2097148 free, 0 used. 3038348 avail Mem
```

## 操作系统监控 -- free -m 内存使用

(比如：在性能测试前，剩余运行内存500M左右，但在压测时，内存还有剩余，只剩下300M左右)

内存使用情况

total: 总的内存

used: 已使用的内存

free : 剩余内存

-m 是以MB为单位显示

-g 是以GB为单位显示

buffer 即将写入磁盘中的数据

cache 从磁盘上读出来的数据

swap: 交换分区 临时的使用，一般情况下不使用，频繁发生swap说明整个系统资源不够使用会带来性能的问题

测试之前：

```
[root@CentOS ~]# free -m
              total        used        free      shared  buff/cache   available
Mem:           3770          491         2662           11           616          3015
Swap:          2047           0         2047
```

测试时：

```
[root@CentOS ~]# free -m
              total        used        free      shared  buff/cache   available
Mem:           3770          686         2355           11           729          2810
Swap:          2047           0         2047
```

通过性能测试前、过程中的数据对比，发现在性能测试过程中，运行内存需要额外消耗接近：200

如果这时候，服务器的剩余内存不足200，或者说只有50，那么可以进行性能测试吗？

在性能测试的过程中，剩余内容不能接近于0，接近的话，代表无法确定是否无法满足性能测试需求，还是刚好满足。

如果内存接近0，那么需要更换更高配置的服务器。或者从该服务器中的其他服务进程中释放出内存出来，以满足我们的性能测试需求

系统：A 已使用1500M 服务器空闲内存只有500，无法满足 B 服务性能测试需要的 700M

## 操作系统监控命令 -- 磁盘空间 df -h

(单台服务器可以部署多个服务)

工作常见的是因为服务运行日志会占用大量的磁盘空间(几十G)

监控磁盘

性能测试之前检查磁盘空间是否够用，需要清理之前的日志 避免日志出现问题导致报错（删除命令：rm）

（生成的日志路径，需要跟开发确认，不同的公司，或者不同的开发，设计的服务日志存放位置都不一样。）

```
[root@CentOS ~]# df -h
文件系统              容量  已用  可用 已用% 挂载点
devtmpfs              1.9G   0    1.9G   0% /dev
tmpfs                 1.9G   0    1.9G   0% /dev/shm
tmpfs                 1.9G  12M    1.9G   1% /run
tmpfs                 1.9G   0    1.9G   0% /sys/fs/cgroup
/dev/mapper/centos_centos-root 17G  3.1G   14G  19% /
/dev/sda1             1014M  151M   864M  15% /boot
tmpfs                 378M   0    378M   0% /run/user/0
[root@CentOS ~]#
```

## 操作系统监控 -- vmstat 监控内存

监控内存：vmstat 秒数， 间隔多少秒采集一次

主要检查服务器的内存是否频繁调用

si：每秒读取虚拟缓存磁盘文件大小

so：每秒写入虚拟缓存磁盘文件大小

如果si、so不为0，说明可能存在内存溢出或内存泄漏

```
[root@CentOS ~]# vmstat
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
r b   swpd   free   buff  cache   si   so    bi    bo    in   cs us sy id wa st
1 0     0  2405344   2108 904596    0    0    12    48   284  465 19  3 78  0  0

[root@CentOS ~]# vmstat 2
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
r b   swpd   free   buff  cache   si   so    bi    bo    in   cs us sy id wa st
1 0     0  2405088   2108 904628    0    0    12    48   282  462 18  3 78  0  0
0 0     0  2405096   2108 904628    0    0     0     0    78   134  0  0 100  0  0
0 0     0  2405096   2108 904628    0    0     0     0    69   122  0  0 100  0  0
0 0     0  2405096   2108 904628    0    0     0     0    69   128  0  0 100  0  0
```

## 操作系统监控--ps--监控进程

监控进程

ps == process status

常用的命令：**ps -ef** 查看所有进程信息

拓展：

一般与 grep 连用 查看指定进程：通过查询指定进程来确定我们的服务是否启用。

（进程名怎么来，是由后端开发编写代码的时候设定好的）

ps -ef | grep 进程名关键字

```
[root@CentOS ~]# ps -ef
UID          PID    PPID  C  STIME TTY          TIME CMD
root           1        0  0  14:15 ?        00:00:01 /usr/lib/systemd/systemd --switched-root --system --deserialize 22
root           2        0  0  14:15 ?        00:00:00 [kthreadd]
root           4        2  0  14:15 ?        00:00:00 [kworker/0:0H]
root           6        2  0  14:15 ?        00:00:00 [ksoftirqd/0]
root           7        2  0  14:15 ?        00:00:00 [migration/0]
root           8        2  0  14:15 ?        00:00:00 [rcu_bh]
root           9        2  0  14:15 ?        00:00:03 [rcu_sched]
root          10        2  0  14:15 ?        00:00:00 [lru-add-drain]
root          11        2  0  14:15 ?        00:00:00 [watchdog/0]
```

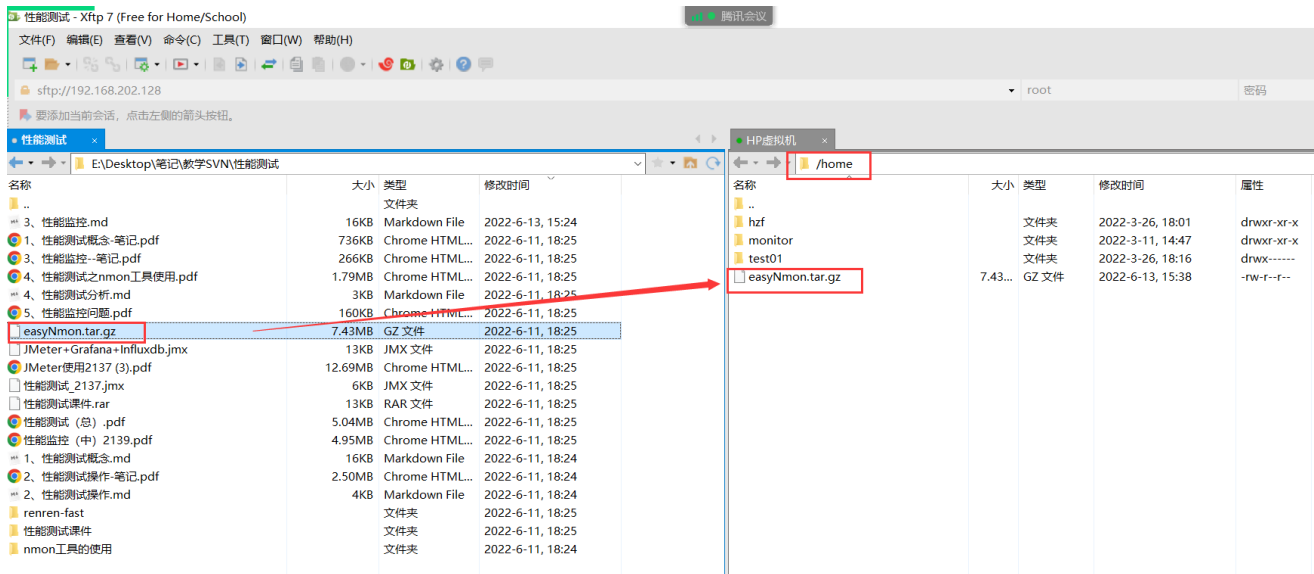
## 监控工具--easynmon工具

nmon IBM公司开发的linux性能测试工具，可以实时展示性能的情况，也可以将监控数据写入文件中，使用nmon分析器进行数据分析

easynmon是nmon的升级版，可以直接在线采集数据并生成图形化曲线图，以肉眼直观的形式看到服务器的性能指标的变化情况。

**easyNmon的部署与启动：**

- 1、将 **easyNmon.tar.gz** 包上传到服务器（虚拟机）



## 2、进行解压，命令：

`tar -xzf easyNmon.tar.gz`

解压后进行查看：

```
[root@CentOS home]# ls
easyNmon easyNmon.tar.gz hzf monitor test01
[root@CentOS home]#
```

## 3、给予755权限：`chmod -R 755 easyNmon`

```
[root@CentOS home]# chmod -R 755 easyNmon
[root@CentOS home]#
```

## 4、启动服务：

需要去到easyNmon目录下进行启动

`cd easyNmon`

```
[root@CentOS home]# cd easyNmon
[root@CentOS easyNmon]# ls
easyNmon nmon web
[root@CentOS easyNmon]# pwd
/home/easyNmon
[root@CentOS easyNmon]#
```

启动服务命令：`nohup ./easyNmon &`

(以默认状态进行启动，默认的端口号9999)

出现提示，直接回车即可

```
[root@CentOS easyNmon]# nohup ./easyNmon &  
[1] 5395  
[root@CentOS easyNmon]# nohup: 忽略输入并把输出追加到"nohup.out"  
[root@CentOS easyNmon]#
```

## 5、使用，通过IP+端口号进行访问：

(IP是使用自己虚拟机的IP，端口号默认统一9999)



关闭linux防火墙：systemctl stop firewalld.service

开启指定端口号：firewall-cmd --zone=public --add-port=9999/tcp --permanent

## easyNmon的使用

Name 任务名称	cxv_2242_fanwe	自定义，不建议中文
	不建议设置中文任务名，默认Linux编码会在图表展示页面出现乱码	
Time 监控时长	15	性能测试15~30，监控时间大于测试时间
	单位为分钟	
Frequency 监控频率	30	采集频率，默认即可
	单位为秒，推荐设置30秒以上	
<div>提交服务器监控任务</div>		
<div>结束服务器监控任务</div>		
<div>结束easyNmon服务</div> kill 命令		
<div>查看服务器监控报告</div>		
<div>查看服务器系统信息</div>		

easyNmon会采集服务器的指标：

CPU：处理器使用情况

内存：运行内容

磁盘读写：

网络宽带：

通过服务器性能指标以及jmeter的聚合报告参数结合起来进行判断。



## 性能测试汇总（重点）：

### 1、性能测试包含的种类：

性能测试：满足性能指标即可，主要是看TPS

压力测试：在极限情况(高并发)，服务的处理能力是否正常；

负载测试：逐步增加压力(并发用户数)，找出性能瓶颈(TPS)

配置测试：找到最适合服务器，没有造成资源浪费

稳定性测试：长时间运行的测试

### 2、什么样的系统/接口需要做性能测试？

并不是所有的系统，所有的接口都需要做性能测试

- 用户量大，pv比较高的系统
- 系统的核心模块和接口
- 业务逻辑/算法比较复杂
- 促销/活动推广计划：
- 公司新做 新系统、新项目

### 3、性能测试操作：

---

jmeter：先考虑单交易、混合场景

写好接口脚本 --- 线程组设置并发用户数 --- 聚合报告

运行时间：15~30 分钟

### 4、接口性能指标参数：聚合报告

---

平均响应时间、tp响应时间(90/95/99)、最小最大响应时间、异常错误率、TPS、吞吐量

### 5、如何确定接口性能（TPS/QPS）是否满足业务需求？

---

1、业务产品需求 --- 优先级最高

2、历史数据：参照历史活动 --- 优先级一般

3、新系统根据PV进行推算：二八原则（一天中的80%PV 是集中在 20%时间内） --- 迫不得已才选择