学习文档：

[【压测环境的搭建】](https://mp.weixin.qq.com/s?__biz=MzU4NzU0MDIzOQ==&mid=2247486698&idx=3&sn=c3a2af9322c2bf6ad25d8c4823d1461f&chksm=fdeb3e8aca9cb79c3ff4f0d61eed4db4e7fd72c54e0ef670ad9d4c12caa05000dec80bbd179a&scene=21" \l "wechat_redirect)

[【性能压测工具选型对比】](https://mp.weixin.qq.com/s?__biz=MzU4NzU0MDIzOQ==&mid=2247486886&idx=2&sn=cd9f4b50afd3d6a03dfeb724905ee428&chksm=fdeb3fc6ca9cb6d0ee48b5d1cdf390656b10eb5ee4c295a0cf97f78afb50a6f4fd6512e4c1b4&scene=21" \l "wechat_redirect)

[【阿里巴巴在开源压测工具 JMeter 上的实践和优化】](https://mp.weixin.qq.com/s?__biz=MzU4NzU0MDIzOQ==&mid=2247487005&idx=1&sn=9c8837261ce97f69f019fe909e09e789&chksm=fdeb3c7dca9cb56b6c142d52f08eae5143cfc38bf214c2e716f65202f83ffd9325b845da48dc&scene=21" \l "wechat_redirect)

[【并发模式与 RPS 模式之争，性能压测领域的星球大战】](https://mp.weixin.qq.com/s?__biz=MzU4NzU0MDIzOQ==&mid=2247487138&idx=1&sn=be66769443f8157461c9ef12cba7722c&chksm=fdeb3cc2ca9cb5d423dceb71977e01a07be3be552a9b7cc63afec96936ee0b14ef07ff602345&token=1058504863&lang=zh_CN" \l "rd)

[【go实现的压测工具【单台机器100w连接压测实战】】](https://www.jianshu.com/p/8a2358af1593)

【[生产环境上 “全链路压测体系” 建设方案的思考与实践](https://blog.csdn.net/alitech2017/article/details/118303450)】

[【带你重走 TiDB TPS 提升 1000 倍的性能优化之旅】](https://www.163.com/dy/article/GQJTA5UQ0511CUU0.html)

|  |  |
| --- | --- |
| TPS（transaction per second） | TPS表示服务器每秒处理的事务数。他是衡量系统处理能力的一个非常重要的指标，在性能测试中，通过检测不同用户的TPS,可以估算出系统处理能力的拐点。 |
| RPS（Request Per Second） | 每秒请求数 |
| QPS(Queries Per Second) | 是每秒查询率，是一台服务器每秒能够相应的查询次数，即1秒内完成的请求数量，是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准。 |
| HPS（Hit Per Second） | 点击率，每秒点击数，一般在性能测试中，用来描述HTTP Request,它代表每秒发送的HTTP请求数量，和RPS概念完全一样 |
| RT(Response-time) | 响应时间是一个系统最重要的指标之一，它的数值大小直接反应了系统的快慢。响应时间是指执行一个请求从开始到最后收到响应数据所花费的总体时间,即从客户端发起请求到收到服务器响应结果的时间 |
| 并发数 | 系统同时处理的request/事务数。QPS（TPS）= 并发数/平均响应时间。 |

**TPS与QPS是有区别的**

事务表示客户端发起请求到收到服务端最终响应的整个过程，这是一个TPS

而在这个TPS中，为了处理第一次请求可能会引发后续多次对服务端的访问才能完成这次工作，每次访问都算一个QPS。

所以，一个TPS可能包含多个QPS

|  |  |
| --- | --- |
| us | 用户态使用的cpu时间百分比 |
| sy | 系统态使用的cpu时间百分比 |
| ni | 用作nice加权的进程分配的用户态cpu时间百分比，值比较高时，须要从服务器Linux系统配置、被测服务运行参数查找缘由。 |
| id | 空闲的cpu时间百分比 |
| wa | cpu等待IO完成时间百分比 |
| hi | 硬中断消耗时间百分比 |
| si | 软中断消耗时间百分比 |

零、如何计算压测指标

**压测我们需要有目的性的压测，这次压测我们需要达到什么目标。**

可以通过以下计算方法来进行计算:

压测原则: 每天80%的访问量集中在20%的时间里，这20%的时间就叫做峰值。

公式: ( 总PV数80% ) / ( 每天的秒数20% ) = 峰值时间每秒钟请求数(QPS)

机器: 峰值时间每秒钟请求数(QPS) / 单台机器的QPS = 需要的机器的数量\

1、假设:网站每天的用户数(100W)，每天的用户的访问量约为3000W PV(页面浏览量 Page View)，这台机器的需要多少QPS?

( 30000000\*0.8 ) / (86400 \* 0.2) ≈ 1389 (QPS)

假设:单台机器的的QPS是69，需要需要多少台机器来支撑？

1389 / 69 ≈ 20

2、单台机器的性能为1000QPS? 网站能同时满足100W人同时在线操作同一接口。需要机器多少台？

1000000 / 1000 = 1000 台

一、如何对一个服务进行性能优化

准备阶段：了解应用的概况：总体架构、服务器信息等；通过性能测试，对性能瓶颈的进行粗略评估，明确优化目标；

分析阶段：通过各种工具或手段，初步定位性能瓶颈点；

调优阶段：根据定位到的瓶颈点，进行应用性能调优；

测试阶段：让调优过的应用进行性能测试，与准备阶段的各项指标进行对比，观测其是否符合预期，如果瓶颈点没有消除或者性能指标不符合预期，则重复步骤2和3。

最佳线程数

二、如何进行一个完整的压测：

1.梳理高峰业务场景：可以按照一些问题描述模板比如5W2H来梳理业务场景，例如：

场景1：2018年11月11日00:00~01:00，共计xx万个用户同时操作xx。

场景2：每天xx时间段，xx系统会轮询调用xx服务xxxx次。

2.明确压测目标，根据业务场景评估或者历史流量峰值评估，并且制定目标时需要留一些Buffer。

3.准备压测数据&链路改造&检查压测环境（写库问题）

1、直接DB建数据 2、通过业务接口建数据 3、线上流量COPY

4.接口单元测试：需要验证压测链路是否通，如果有影子/隔离环境，看流量是否打到正确的环境中去。

5.压测前通知到相关运营、业务、开发同学，进行小流量试压。

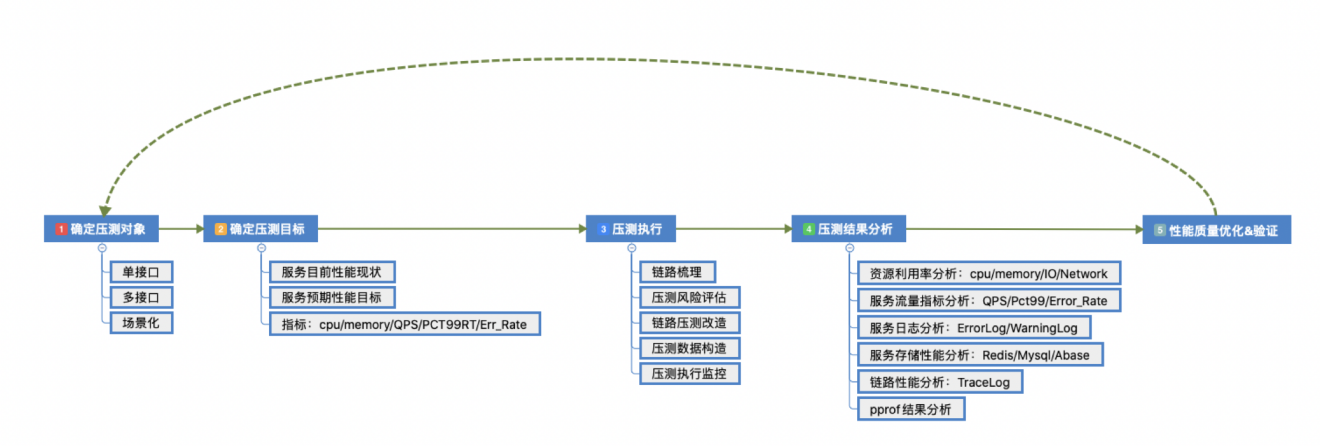
6.正式压测：

1.逐步加压到目标QPS，记录系统指标：CPU使用率、内存使用率、Load等，服务指标：QPS、RT、

2.在加压过程中，观测并记录发现的问题，如果出现阻塞性问题，停止压测。

7.压测后发出压测报告，压测报告对于目标和结论尽量一句话表述出来。

1. 如何进行完整的压测全链路：



一、压测节奏：大促活动前，以及每两个季度例行化进行公司级全链路压测。

二、设计压测方案：

1.明确此次压测背景、涉及对象、请求数据流向、涉及链路服务、目标等，流量识别方式和环境识别方式；

1、活动大促一般是压测商城，

2、例行压测结合线上监控与核心场景，压测app端核心业务接口；

2.设计并评审压测方案：明确开发、运维的负责人、链路改动方案、压测时间，压测过程中的checklist，邮件周知此次全链路压测方案和时间；

三、压测执行：

1、相关服务同学再次确认本次压测接口、影响范围，以及是否有变动；

2、完成压测数据准备、监控大盘准备、压测环境检查；

3、在”产品客服沟通群“、“产研内部沟通群”同步压测安排，包括压测服务名称、压测计划时间，可能造成影响；

4、小流量试压，在kibana上验证流量请求到线上；

5、开始正式压测，记录压测过程中数据，压测中遇到问题，压测结果，压测结论，改进计划；

一定要记录性能瓶颈，是否需要优化，如需优化，确定责任人、排期。

6、完成压测后，在”产品客服沟通群“、“产研内部沟通群” 同步压测结束；

四、总结：压测结束后，组织压测参与者开个总结小会，总结压测中发现的服务问题、操作流程问题，并确定问题跟进负责人，整理压测过程数据，将压测报告发出，进一步规范化操作流程，根据压测过程中遇到的问题，进一步调整压测流程与工具。

五、下次例行化压测时验证优化效果。

1. Xhprof

Xhprof 信息统计可以基于xhprof.io这个工具进行修改，基本能满足我们的需求

这是我以前搭建测试的一个 http://xhprof.z.it603.com/xhprof.io/ 账户密码 chujilu chujilu0725

参考文章：https://www.cnblogs.com/ningskyer/articles/5554733.html

可以做到信息保存到mysql，根据域名、接口、时间进行检索统计等，全部机器的数据都保存到同一个数据库中，性能上有问题的话可以修改他的保存逻辑，通过redis做中转

https://www.it603.com/article/36.html

xprof：下载插件，修改php.ini配置，重启配置，在文件入口出口处打上标签，请求服务，利用graphviz生成可视化图形。

火焰图：使用perf 指定进程id进行采样，用perf script工具对采样数据perf.data进行解析成perf.unfold，将perf.unfold中的符号进行折叠成perf.folded，使用flamegraph生成svg图，可在浏览器上进行查看。

五、压测过程中遇到的一些问题

一、 压测环境压测服务时出现链接数过多

1、时间：2020/6/11

2、请求链路：施压机-- 负载均衡--nginx--服务器。

机器：4c8g

3、遇到问题：

1、从施压机给服务器每秒发送50个请求，使用netstat -a ，施压机本地链接未超过200个，服务器链接快速增加，服务器链接跟负载均衡10.4.11.160的链接在6000+。

4、定位方法：

1、修改负载均衡的/etc/sysctl.conf 文件，将net.ipv4.tcp\_fin\_timeout 由60变成10，具体操作： 在/etc/sysctl.conf 中添加 tcp\_fin\_timeout = 10， 并执行sysctl -p 生效。再次压测，情况无大变化。

2、在施压机通过绑定hosts直达机器，情况无大变化。

3、修改服务所在机器的/etc/sysctl.conf， 修改此三项配置如果，服务器链接正常。但此配置与线上配置不一致。

net.ipv4.tcp\_tw\_recycle = 1

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_fin\_timeout = 10

4、三个参数的解释，一般不建议开启：https://www.cnblogs.com/lulu/p/4149312.html

net.ipv4.tcp\_tw\_reuse = 0 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭

net.ipv4.tcp\_tw\_recycle = 0 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭

net.ipv4.tcp\_fin\_timeout = 60 表示如果套接字由本端要求关闭，这个参数决定了它保持在FIN-WAIT-2状态的时间

二、不同机器之间性能差距较大，性能差机器可能会拖慢整体服务性能。

1、时间：2020/5/17

2、请求链路：施压机-- 外网服务。（绑到hosts到某台外网负载均衡）

3、遇到问题：

1、扩容后进行复测，服务性能没有上升。

2、对不同型号机器进行绑定hosts到指定机器，服务性能差距比较大。

4、解决方案：

1、服务均衡对机器分流进行调整；

2、下掉性能较差机器（采用此方法）

3、在服务部署在不同型号的机器上时，建议先进行框架压测。

4、参考数据：裸金属 200左右 ；4c8g的云主机 500 ；10c32g云主机 1800（php7）

三、压测过程中，出现压测图形周期性出现断崖式下跌现象。

1、时间：2019/5/18

2、施压机-- 外网服务。（绑到hosts到某台外网负载均衡）

3、遇到问题：

在压测过程中观察压测流量曲线 发现在一瞬间请求出现断崖式下跌 然后迅速爬升 出现一个明显的抖动点。

4、定位原因：

服务机器资源正常，观察下流调用，发现mysql数据库连接数被打满，导致无法新建连接。 分析导致连接未被释放原因：发现sql没走到索引上，所以部分查询过慢，过慢链接累加导致连接数被打满。

5、修改mysql查询语句。

四、gatling ssl握手超时问题

1、时间：2019/5/18

2、施压机-- 外网服务。（绑到hosts到某台外网负载均衡）

3、遇到问题：

压测https服务过程中，gatling出现大量ssl握手超时报错。

4、定位原因：

gatling的rps每秒都新建socket连接，因为https是双向认证的，nginx会去解析ssl证书和密文 ，当nginx内存不够大的时候 连接数增加导致，导致nginx划分的32m解密内存迅速被打满，后续解密握手在等待最后出现握手超时。

六、如何搭建一个压测平台

背景：小猪近一年来业务量大幅上涨，多次发生因流量过大导致的服务不可用问题，故服务性能优化迫在眉睫。而小猪还没有性能测试相关的平台。大量的压测需要手工进行，存在重复劳动，效率低下的问题。在此背景下，我们设计并开发了业务压测平台。

设计方案：因为是从0到1进行设计开发，我们参考学习了公司一些现有的平台框架，并结合自身的代码情况，最终选定了压测平台前端使用vuejs+element，后端使用微服务框架spring cloud、spring boot、网关zuul等框架进行压测任务的接收、下发和数据收集等。由于开源的gatling方便模块化，搭建环境和编码都相对简单，所以压测工具选择了gatling，数据库使用redis、mysql进行数据存储。

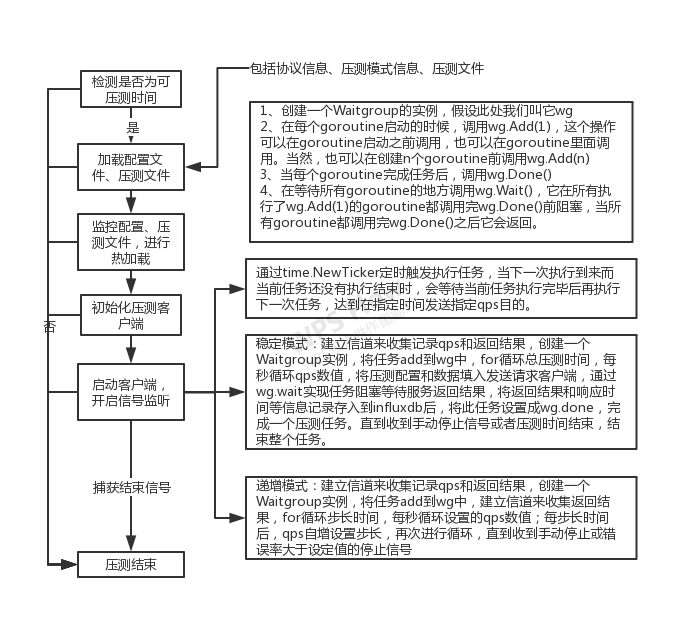
实现过程：在我们实施设计方案的过程中，也遇到了一些与预期不符合的情况，比如说：设计时计划在压测平台实时展示压测过程数据，在编码过程中发现会发生因数据量过大导致web页面卡死的问题。通过对gatling工具的学习了解，发现gatling封装了接口往时序数据库influxdb中存储数据，grafana读取influxdb展示数据也是目前比较成熟的方案。所以我们调整了设计方案，嵌套grafana进行压测过程数据展示的方案。在开发过程中，也遇到了排期的问题，所以我在完成前端的基础上，也开发了一些后端功能。

目前已实现压测数据自动产生、压测过程监控和报警、压测报告自动生成功能。正在开发压测环境自动化检测功能。

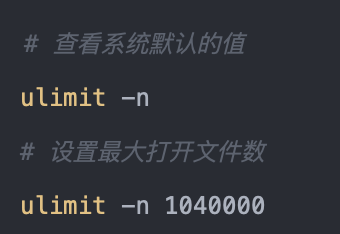
收益：压测系统上线后，针对业务系统做了大量压测，摸清了系统的性能状况。业务同学据此优化核心接口，性能平均提高了5倍，并经受了线上大促活动的检验，

长期规划：通过更多的内置场景例如：线上首页压测，小程序首页压测等，结合公司现有的工具平台，实现一键式自动化压测平台。

七、内部压测工具实现原理



1. 一些压测中常用的系统配置文件
2. /etc/security/limits.conf 设置文件句柄，修改以后重启服务器，也可以尝试ulimit -n \*\*\*\*修改。



1. /proc/sys/fs/file-max 表示系统级别的能够打开的文件句柄的数量，不能小于limits中设置的值。如果file-max的值小于limits设置的值会导致系统重启以后无法登录。

3、100W长连接，每台机器有 65535-1024 个端口， 100W / (65535-1024) ≈ 15.5，所以需要16台服务器。可通过 /etc/sysctl.conf 文件进行修改。

1. 压测过程中查看系统状态常用指令：

top #查看系统的整体状态

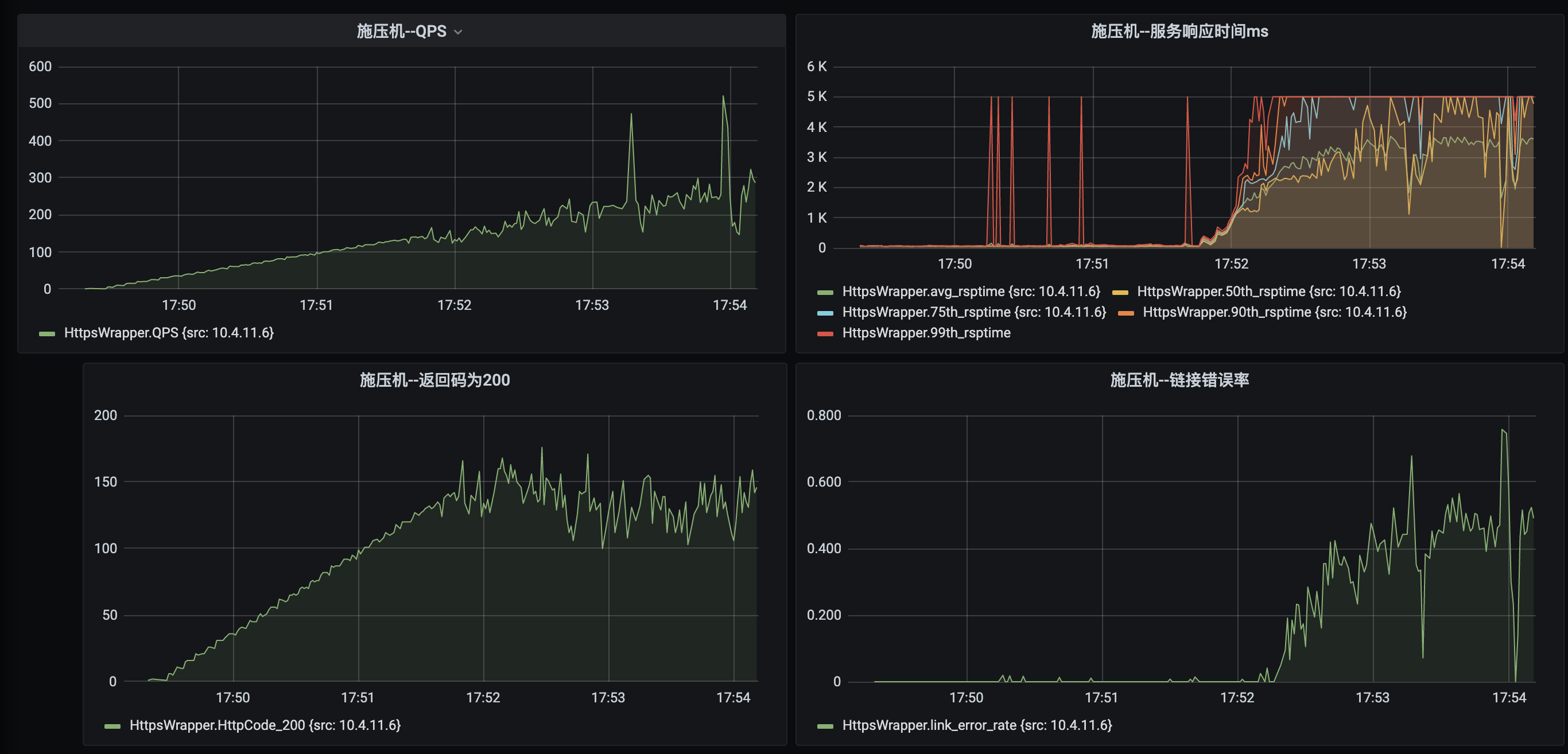
ps # 查看进程内存、cup使用情况

iostat # 查看系统IO情况

nload # 查看网络流量情况

/proc/pid/status # 查看进程状态

1. 常见的压测图形



1. 压测工具对比

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **ab/abs** | **Jmeter** | **Gatling** | **Locust** | **K6** | **自研** | **Wrk** |
| **亮点** | 非常简单易用，几乎不需要编程能力 | 相对简单易用，有一定的学习曲线，支持协议较多，支持录制，服务端性能指标监控 | 并发率高，动态web page report，支持录制，商业cloud，代码化 | 并发率高，代码化 | 并发率高，商业cloud，代码化 | 贴合业务自身 |  |
| **不足** | 不能编程，并发率低，支持协议较少 | 并发率低，用例不易维护和复用，扩展脚本执行速度慢 | 学习曲线较高，执行协议较少（可扩展），开启日志较占硬盘 | 学习曲线较高，不能动态改变vu(Virutal User)，支持协议较少（可扩展） | 学习曲线较高，支持协议较少（可扩展） | 学习曲线较高，执行协议较少(可扩展) |  |
| **分布式** | 不支持 | 支持 | 原生不支持(第三方扩展) | 支持 | 暂时不支持(在开发) | 暂时不支持 |  |
| **适用场景** | 简单 | 一般 | 复杂 | 复杂 | 复杂 | 复杂 |  |
| **语言** | shell | gui/scripts(java) | scala/dsl | Python | js/golang | golang |  |
| **报告** | Text | gui+web+3 | web+text | web+text | text+k6 cloud web + 3rd | text+grafana |  |
| **jenkins** | 无插件 | 有插件 | 有插件 | 无插件 | 无插件 | 无插件 |  |
| **社区支持** | 少 | 多 | 一般 | 一般 | 少 | 无插件 |  |
| **商业支持** | 无 | 支持 | 支持 | 无 | 支持 | 无 |  |
| **并发模型** | 单线程结合异步io | 多线程 | actor/akka | 消息循环 | csp | 协程消息循环 |  |