

## 测试专栏特别放送 | 答疑解惑第五期

2018-11-07 茹炳晟

软件测试52讲

[进入课程 >](#)



你好，我是茹炳晟。

今天的“答疑解惑”系列文章，我们一起来解决性能测试系列中 7 篇文章的问题。你可以通过下面对每篇文章的简单总结回顾一下文章内容，也可以点击链接回到对应的文章复习。

现在，我们就开始今天的问题吧。

### 问题一：你在性能测试项目中，选择具体的系统吞吐量指标时，会考虑哪些因素呢？

在专栏的第 28 篇文章 [《带你一起解读不同视角的软件性能与性能指标》](#) 中，我首先从终端用户、系统运维人员、软件设计开发人员和性能测试人员，这四个维度介绍了软件系统的性能到底指的是什么；然后，和你分享了软件性能的三个最常用的指标：并发用户数、响应时间、系统吞吐量。

项目中，选择具体的系统吞吐量指标时，会考虑哪些因素呢？

**其实选择哪种类型的吞吐量指标，和你的测试目标以及被测系统的特点是息息相关的。**

如果你的被测系统是后台处理系统，而且你的测试目标是要优化它的处理能力，那么这个时候你的关注点必然就是每秒能够处理的请求数量，即 Requests/Second。当然如果你发现你的后台处理能力有可能是受限于网络传输的带宽，那么这个时候你就可能需要去考虑 “Bytes/Second” 这种类型的吞吐量指标了。

总结来讲，选取哪个吞吐量指标，取决于你最关注的内容。

下面这位昵称为 “假装乐” 的读者的留言很典型，是很多刚刚结束性能测试的同学都会有的疑惑。



## 假装乐

写于 2018/08/31

说到性能测试一直有个疑问，性能测试何时开展，实际环境开展感觉有点晚了，生产环境又不知道衡量指标是否具有借鉴价值，不知道老师的团队实际工作中，各阶段如何开展

引自：软件测试52讲

28 | 带你一起解读不同视角的软件性能与性能指标

识别二维码打开原文  
「极客时间」 App



其实，性能测试应该贯穿于软件研发生命周期的各个阶段：

单元测试阶段就要衡量代码级别的时间复杂度和空间复杂度，以及多线程并发情况下的功能准确性等等；

每个 API 也需要进行单独的性能测试和评估；

集成测试阶段需要考虑跨组件或者模块的数据大小，以及缓存的使用情况等等；

说到底，一个最基本的原则就是，性能问题一定是越早发现越容易定位，也越容易被修复。而到了软件研发生命周期的后期，性能问题的定位成本和复杂度会呈指数级增长。

所以，如果你有机会去了解大型软件公司的测试的话，就会发现它们没有所谓的性能测试团队，而是有一个性能工程团队。这个团队会从软件研发生命周期的各个阶段去测试、评估和优化软件的性能。

## 问题二：你在实际工程项目中，接触过性能测试的哪些方法，其中遇到了哪些问题，又是如何解决的？

在专栏第 29 篇文章 [《聊聊性能测试的基本方法与应用领域》](#) 中，我通过一个医院体检的例子，和你分享了并发用户数、响应时间和系统吞吐量这三个指标之间的关系和约束；然后，又和你分享了性能测试七种常用方法，以及四大应用领域。

在这篇文章最后，我希望你能够分享一下你在实际开展性能测试时，都遇到过哪些问题，又是如何解决的。虽然这篇文章的留言比较少，但也能从中看出大家在开展性能测试的时候，确实也如我当初一样，遇到了各种各样的问题。

那么，现在我就来和你分享一下性能测试中可能遇到的一些典型问题吧。

其实，性能测试中可能会遇到的问题实在是太多了，架构中的各个层面、每个软件模块、模块配置、数据库中的数据量、多线程的锁机制、进程设计、JVM 配置参数、数据库配置参数，以及网络参数等等，都会成为性能测试中的问题。

可以说，性能测试的问题，只有你想不到的，没有你遇不到的。所以，如果我通过一个实际案例和你分享的话，肯定会是长篇大论，有违答疑系列文章的设计初衷。为什么？因为性能测试的问题，一般都和架构、设计、配置、数据量有着密不可分的关系。所以，**我今天会通过一个简化的案例，和你展开分享，意在抛砖引玉。**

首先，我想问你一个问题：当你做压力测试的时候，你觉得硬件资源占用率是低好，还是高好？很多人可能会说，当面对大量并发请求的时候系统资源占用率当然低好。因为资源用得少，说明系统后续的容量可以继续大幅度扩充。

听起来很有道理，但真的是这样吗？

发用户的上升而有持续上升的趋势。所以，一定是有某些机制限制了 CPU 的使用。

其实，在这种情况下我们希望看到的是，随着并发用户数的不断增长，这些 CPU 敏感性的并发操作会尽可能多地去使用 CPU 的计算能力，而不是现在这种 CPU 使用率上不去的情况。

为此，我分析了这部分的代码逻辑，发现其中使用了一个固定大小的数组来存放并发任务进程的句柄，当这个数组满了的时候，新进程的创建处于阻塞状态，只有之前的进程处理结束后，数组中出现了空位，新的进程才会被创建。当时这个数据的大小是 128，也就是最多只能有 128 个并发进程同时运行，我当时就怀疑这是限制 CPU 使用率的主要原因。

为了验证这个想法，我直接将这个固定数组的大小调整成了 256，然后继续并发测试。果然，CPU 的使用率徘徊在 30% 左右，就验证了我的猜测。

那么，接下来就需要修复这个问题了。显然，这是一个设计上的问题，压根儿这里就不应该采用固定大小的数组，而是应该采用可变长度的数据结构。

### 问题三：你接触过哪些后端性能测试工具？你认为这款工具中，有哪些好的设计吗？

在专栏第 30 篇文章 [《工欲善其事必先利其器：后端性能测试工具原理与行业常用工具简介》](#) 中，我以问答的形式，和你分享了后端性能测试的理论，以及工具使用的问题。这也是这个专栏中，唯一一篇采用问答形式的文章，有没有感觉读起来比较省力呢？

因为我后面增加了一篇 JMeter 的加餐文章，所以这里我也就不再过多地介绍后端性能测试工具了。这次，我来回答一下 “Robert 小七” 提到的问题。

正如我在今天的第一个问题中提到的，高效的性能测试一定是从源头抓起的，也就是研发的每个阶段都需要进行性能测试，而不是等到系统开发完了，再一次性地进行黑盒级别的性能测试。

所以，对每个 API 的性能测试是非常必要的。而且，很多公司，比如 eBay 等，都会对每个 API 进行独立的性能测试。其实，在对 API 开展独立的性能测试之前，还需要在一些关





Robert 小七

写于 2018/09/05

请问老师，实战中性能测试是如何一步步开展的？难道最开始就涉及到性能场景模型分析嘛？不是先测试单独的接口性能嘛？

引自：软件测试52讲

30 | 工欲善其事必先利其器：后端性能测试工具原理与行业常用工具简介

识别二维码打开原文  
「极客时间」App



**问题四：你在工作中接触过哪些前端性能测试工具，它们各自有什么特点呢？**

在专栏的第 31 篇文章 [《工欲善其事必先利其器：前端性能测试工具原理与行业常用工具简介》](#) 中，我以一个具体网站为例，和你分享了使用 WebPagetest 进行前端性能测试的方法，以及前端性能相关的主要概念与指标。

在这里，我来分享下我的经验吧。

前端性能测试工具除了我在文章中介绍的 WebPagetest，比较常用的还有 YSlow，但是这些工具的基本原理是类似的，所以如果你已经掌握了我在这篇文章中介绍的 WebPagetest 的原理的话，对于 YSlow 等前端性能测试工具的原理，基本就可以做到触类旁通了。

此外，有些公司还会特别关注一些特定的前端性能指标。这些性能指标，一般不能从性能测试工具中直接得到，需要自行定制开发。

这个昵称为“木然”的用户，提出的问题很典型。很多刚开始使用 WebPagetest 的同学都会有这个疑问，但是很不幸，WebPagetest 是无法来做这种需要登录才能访问到的页面的前端性能调优的。



木然

写于 2018/10/15

老师您好，我想请问如果某些页面需要登录才能访问到，用您介绍的工具怎么测试呢？目前登陆后的 url 拷进去后测试的都是登录页面

引自：软件测试52讲

31 | 工欲善其事必先利其器：前端性能测试工具原理与行业常用工具简介

识别二维码打开原文  
「极客时间」App



WebPagetest 这类工具的初衷，就是纯粹站在前端页面优化的角度来设计的，本身并不会涉及业务操作，所以对这块的支持很弱。虽然 WebPagetest 支持 Http Auth 以及自定义脚本的扩展，但是 Http Auth 还是会受到服务器端本身配置的影响，而自定义脚本的扩展还受限于特定的浏览器，所以实际的应用价值有限，也很少有人去用。

而至于有什么更好、更灵活的方法来处理这种需要登录，以及特定业务操作的前端页面性能优化，很可惜，目前我并没有什么好的方案。如果你对此有一些好的想法或者实践的话，就给我留言一起讨论吧。



在专栏的第 32 篇文章 [《无实例无真相：基于 LoadRunner 实现企业级服务器端性能测试的实践（上）》](#) 和第 33 篇文章 [《无实例无真相：基于 LoadRunner 实现企业级服务器端性能测试的实践（下）》](#) 中，我从最基础的性能测试需求获取、LoadRunner 的原理开始，和你分享了基于 LoadRunner 实际开展企业级服务器端性能测试的整个过程。

通过这两篇文章，我希望能够帮你快速建立服务器端性能测试的全局观，并了解各主要步骤的关键技术细节。其实，正如我在文中所说的，在开展整个性能测试的过程中，测试需求的获取是最关键、最难的一个环节。这里我再针对测试需求的获取，和你分享一个实例吧。希望可以真正帮到你。

很多时候，**我们从产品经理那里拿到的需求是很笼统的，必须经过必要的分析和细化才能转换为可以用于性能测试场景设计的需求**，就像文章中提到的“每天支持完成 8000 个体检”的例子。这样的例子还有很多，比如我们经常可以看到类似“系统最大支持 500 万用户同时在线”的需求，这同样是一个看似具体，实则非常笼统的性能需求。

500 万用户在线，这些在线的用户在具体执行什么类型业务操作，对后端服务器造成的压力差别是巨大的。比如，这 500 万个用户都在执行查询操作和这 500 万个用户什么都不做，对后端服务器的压力是天壤之别的。

那么，这里需求获取的难点就是，要能够准确估算这 500 万在线用户执行的各种类型的业务操作所占的百分比，只有这样才能真实、客观地反应后端服务器承受的实际压力。

但是除此之外，**还有很多性能需求并不是直接从产品经理那里获取的，而是需要资深的性能测试人员根据以往的经验，以及同类系统和竞品的业务流量来自己估算。**

比如，产品经理不会告诉你一个实现具体业务的 API 操作应该要在多长时间内完成；产品经理也不会明确告诉你在 API 层面的业务吞吐量是多少。这些测试需求都是需要性能测试人员来预估，甚至是基于一些实验来细化的。

所以说，性能需求的获取是一个关键且困难的环节。

这个昵称为“Sunshine”的用户，在留言中的问题虽然简单，但也是个典型问题。我来和你一起分析一下。

时间段内每多少秒增加或者减少多少个并发用户，并且 LoadRunner 还会自动提供并发用户数随着时间变化的曲线图。你甚至可以直接修改这个曲线图，来修改用户数量变化的规律，使其符合你的需求。

另外，场景设计中的很多配置都可以在 LoadRunner 的场景设计界面中实现。具体内容，你可以参考[LoadRunner 的使用文档](#)。



Sunshine

写于 2018/09/13

老师可以介绍一下，如何去实现每隔 10s 增加 100 个用户的方法吗？上一篇提的一些场景不太明白如何去做。感谢🙏

引自：软件测试52讲

33 | 无实例无真相：基于LoadRunner 实现企业级服务器端性能测试的实践（下）

识别二维码打开原文  
「极客时间」 App



在专栏的第 28 篇文章 [《站在巨人的肩膀：企业级实际性能测试案例与经验》](#) 中，我挑选了最重要的四类性能测试方法（性能基准测试、稳定性测试、并发测试，以及容量规划测试），和你分享如何在实际项目中完成这些测试，确保软件的性能。

通过这篇文章，我希望可以帮助你从整体上理解性能测试，形成一个体系知识。而在这篇文章最后，我希望你能够分享一个你所在企业采用的性能测试方法，大家互相取长补短。

对于 eBay 这样的大型全球化电商企业，性能测试除了文章中提到四类性能测试方法以外，还会开展一些其他类型的性能测试。

对于关键业务代码以及中间件代码的核心算法部分，一般都会开展基于时间复杂度和空间复杂度的代码级别的性能评估；

对于各个独立的中间件或者公共服务组件本身，也会开展性能基准测试和容量规划测试；

对于基于微服务的各个 API 接口，会开展性能基准测试和压力测试；

对于前端 Web 页面，会开展基于前端性能的调优；

对于整体系统，会不定期开展全链路压力测试，其中还会使用历史流量回放等技术来模拟海量实际的并发场景。

以上这些是从性能测试的类型来讲的。从性能测试工具的支持上来看，eBay 还建立了一些内部使用的公共性能测试平台，任何人都可以通过这些性能测试平台方便地发起压力负载，而不用去关心诸如 Load Generator 之类的细节，对于后端性能测试以及 API 性能测试，你只要上传压测脚本和性能测试场景设计，就能很方便地发起测试。这个很像淘宝对外提供的 PTS 服务。

其实，上述的这些方法适用于很多的互联网产品。而至于到底实施哪几条，则取决于你想通过性能测试希望达到什么样的目的。

最后，感谢你能认真阅读第 28~34 这 7 篇文章的内容，并写下了你的想法和问题。期待你能继续关注我的专栏，继续留下你对文章内容的思考，我也在一直关注着你的留言、你的学习情况。

感谢你的支持，我们下一期答疑文章再见！

# 软件测试52讲

从小工到专家的实战心法

茹炳晟

eBay中国研发中心  
测试基础架构技术主管



新版升级：点击「👤请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 测试专栏特别放送 | 答疑解惑第四期

下一篇 测试专栏特别放送 | 答疑解惑第六期

## 精选留言 (1)

写留言



小青青

2019-03-17



后面哪一篇讲jmeter？最近压测，用jmeter并发请求数1500，吞吐量1000每秒，跑了6个小时，都不会超时报错，为什么呢