# 30 What the hell！测试也要懂架构设计？！

更新时间：2019-10-23 17:30:05



> 富贵必从勤苦得。——杜甫

我曾经 N 次的跟很多测试朋友聊起这个事情：测试是需要懂架构设计的。很多测试的小伙伴都觉得，架构设计是架构师、开发，甚至是项目经理去背书的，关测试什么事儿？更有很多同学直接跟我说：风落啊，我没有想过要做那么高 Level 的测试，我觉得把自动化性能什么的搞搞好就可以了。甚至有跟我说：你说测试要懂代码我都忍了，居然还让我懂架构？！往往大家都觉得测试与架构是面包与牛排的选择，先抓下来面包吃进肚子里的好，至于牛排，等再看看自己有没有那种能力吧。

其实如果倒退到十年，或者是八年之前可能我也有这个想法，但是走过这几年的路，我发现架构设计在某种程度上对于测试来说是非常重要的，如果完全不懂或者只有一知半解的话，往往会发生意料之外的线上问题。我们一起来看一些实际工作中的场景：

缓存在我们的系统架构中是非常常见的中间件，一些频繁操作数据库去读写的数据我们可以放到缓存中加速，但是由于缓存命中很低，导致生产上大部分的用户还是直接读取到了数据库中，带来了非常大的压力，甚至是崩溃。这样在我们自己测试的过程中是很难设计测试用例去覆盖的。

**【场景二】**

还说缓存，如果项目代码中对某一个关键 key 值不断的重写，那么这个缓存实例就可能会过载掉，降低了整体的缓存性能。我们在功能测试过程中也是很难复现的。

**【场景三】**

除了缓存，我们现在在系统设计中还经常用到的组件还有队列。目前在各种系统中的异常设计经常用到队列来进行异步处理，一般正常的异步队列处理逻辑是：

发生异常 →（生产者）产生队列消息 →（消费者）获取队列消息 → 异步处理 → 成功消费队列消息 / 异常将消息推回队列

如果在最后一个步骤程序的错误导致异常没有重新推回队列会怎样呢？

**【场景四】**

我们经常会出现数据迁移的状况，有时候会在不同的数据结构中进行转换，那这时我们就需要程序来进行处理了，但是每次要迁移转换的数据量不宜过大，比如我们每次处理 5000 条。那在程序处理上我们简单的排序，然后 LIMIT 就可以取到了。于是我们获取数据的方式：

```
SELECT * FROM table LIMIT 0,5000;
SELECT * FROM table LIMIT 50001,10000;
...
```

在正常测试时候，会很正常，不会发现问题。可是如果数据量很大呢？如果超过 1 个亿、10 个亿呢？由于 LIMIT 执行的性能问题就会执行速度越来越慢，直到数据库崩溃。

**【场景五】**

微服务已经是时代的潮流了，但是在微服务开发过程中的循环依赖问题一直是一个需要小心避开的坑。一旦在进行某次大型迭代的时候，出现了 A 依赖 B、B 依赖 C、C 依赖 A 的情况，在测试时自然不会出现问题，但是生产打包、发布的过程中就很容易由于循环依赖导致无法正常进行。

简单的聊了几个场景，这些场景都是源于我在工作中的实际情况，那么到这里你还真的觉得不懂架构也无所谓么？如果完全不懂的话，你没办法想象一个在你的认知里根本不存在不理解的东西究竟是什么样子，这比盲人摸象更加的虚无缥缈。如果你的意识里系统架构都不存在的话，那么你怎么知道架构在什么情况下会出现异常呢？

所以大家可以把自己的技能点大大的往业务能力上招呼，但是一定不要忘记，学点架构知识，还是非常有好处的，可以避免一部分由于设计上带来的问题。也许你觉得这些问题跟测试没关系啊，但是当领导问你：为什么没测出来？可能很难解释清楚了吧。结合上面的种种，我觉得架构的知识反而是我们想把测试甚至是功能测试做到极致必不可少的一点。

从我自己来说，我养成的习惯是：我要接手的系统，我一定要梳理清楚系统应用到的所有组件、数据流转情况、业务的时序图以及上下游的依赖关系。如果有一个新的需求，涉及到这个系统，首先先自己打一个大体设计的腹稿，并且参与研发的设计评审会议，将研发的架构设计与自己的相比较，并思考与之前的架构设计是否有冲突的地方、是否有不合理之处。最后把架构层面上可能出现的异常，比如队列、缓存、数据库、上下游系统都考虑进来，单独设计异常的测试场景。能够在测试阶段覆盖的尽量去覆盖，如果无法覆盖的就需要进行代码 review 和白盒测试来保证代码实现的正确性。

我相信，听起来虽然复杂，但对于每个人来说，并不算是不可能完成的任务。当然，我没有说每个人都要达到架构师的水平，懂未必达，实际上很难达。在上一节中我们在最后说过，一个真正优秀的测试开发需要有架构设计能力，这里的架构设计不仅仅懂就可以了。我在这里多啰嗦两句，无论测试还是开发领域，想要完成一个优秀的架构设计、成为一个架构师不是简单的把现有东西用起来或者填一填坑就可以的，真正的架构设计不仅仅是靠设计，还要靠演进，这要求我们对技术具有前瞻性，是技术的导演而不是演员，更不能是观众。同时要依托自己的技术能力由浅入深的掌握设计模式、微服务、分布式等等，逐步了解、熟悉、融入架构设计之中，让自己的设计不仅仅能能够很好的支撑现在，而且能够合理的应对未来的趋势和变化。

似乎已经开始要跑题到成为一个架构师上面去了，实际上我今天聊的所谓的懂是可以理解。让你跳出现有的架构去从零做一个产品的架构设计当然很难，可是依托了现有成熟的架构设计、丰富的中间件，即便是测试人员，搞清楚所有组件的作用和工作方式，理解现有的架构设计，评估新需求的架构适配，我觉得还是可以 HOLD 住的。

如果再进一步呢？

这也是我一直努力去做的改变。我希望一方面不仅仅是评估即将到来的修改对当前架构的影响，也能够对当前架构的演进贡献出自己的力量，能够以测试这样一个特殊的角度推动架构设计的前

现在可用的轮子越来越多是不争的事实，所以每个架构师、研发人员都希望做出更有深度和创造力的架构和代码设计，但是往往这时候忘记了一个原则：软件越容易测试，测试的成本就会越少，遗留的缺陷也会越少，软件质量也就会越高。所以回到前边的技术，自动化测试、单元测试、TDD 的前置将会为软件系统的可测试性提供更大的保证。当然，这不是某种技术就可以实现的，而是需要团队所有人都能在研发设计的概念阶段就开始关注它的可测试性，考虑到应该如何测试，更多的是一种思维上的变化。

那么聊到这里，你对于测试眼中的架构设计是不是有了新的认识？

← 29 到底什么才算测试开发?

31 黑科技：有没有一天，AI（人工智能）会取代软件测试 →

**精选留言 1**

欢迎在这里发表留言，作者筛选后可公开显示

**慕标3246374**

实际工作中服务间依赖错综复杂++

0 回复 2020-02-16



千学不如一看，千看不如一练