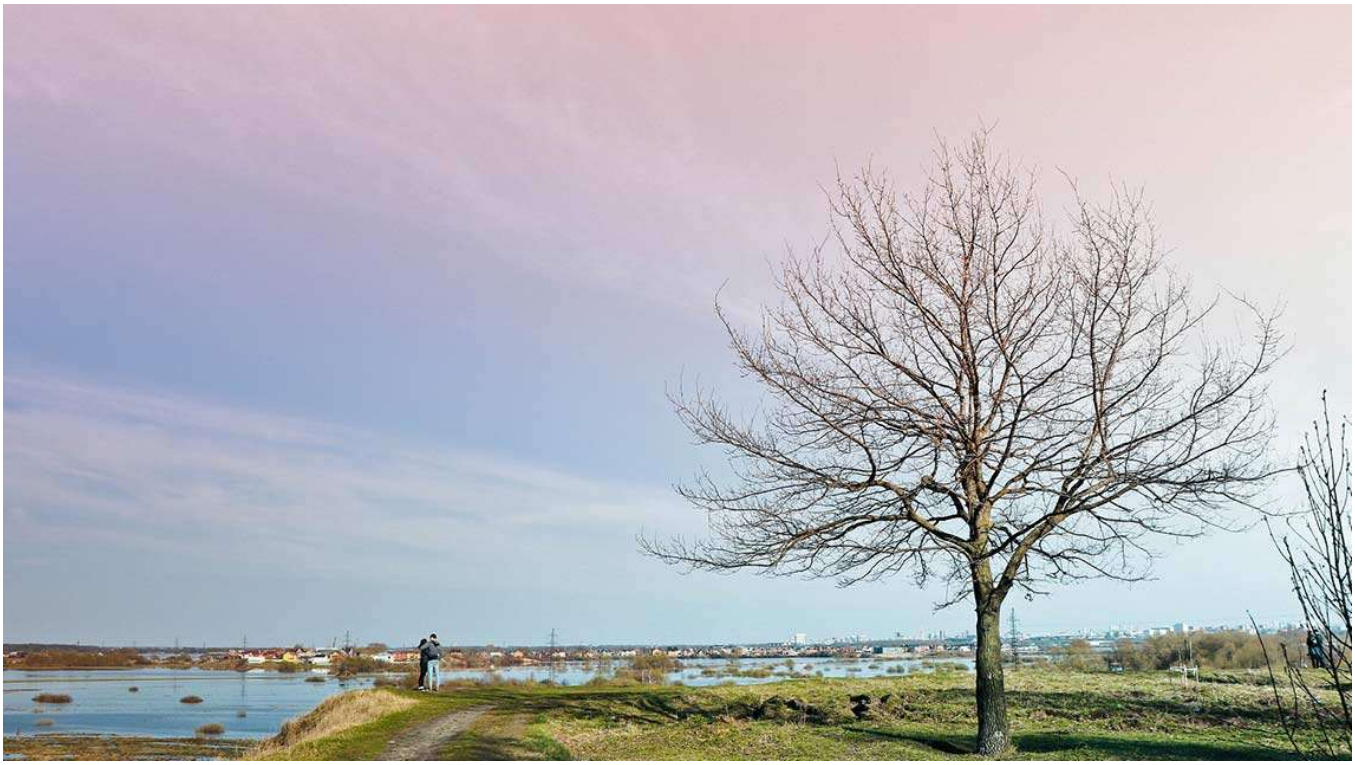
26 | 越来越重要的破坏性测试

2018-09-01 王潇俊



26 | 越来越重要的破坏性测试
朗读人：王潇俊　10′01″ | 4.60M

你好，我是王潇俊。今天我和你分享的主题是：越来越重要的破坏性测试。

其实，持续交付中涉及到的与测试相关的内容，包括了单元测试、自动化测试、冒烟测试等测试方法和理念，我为什么我把破坏性测试拿出来，和你详细讨论呢？

原因无非包括两个方面：

- 其一，单元测试等传统测试方法，已经非常成熟了，而且你肯定也非常熟悉了；
- 其二，破坏性测试，变得越来越重要了。

那么，破坏性测试到底是因为什么原因变得原来越重要呢？

随着 SOA、微服务等架构的演进，分布式系统对测试的要求越来越高，不再像传统的单体应用测试一样，可以很容易地无缝嵌入到持续交付体系中。因为分布式系统的测试不仅需要大量的前提准备，还存在着非常严重的服务依赖问题。

这就使得分布式系统的测试工作，除了要关注运行的应用本身外，还要考虑测试环境的因素。

很快，我们就发现，破坏性测试可以解决分布式系统测试的这些难题，而且还可以帮助我们解决更多的问题。它可以弥补传统持续交付体系只关注代码或应用本身，而忽略其他外部因素影响运行中代码的问题。而且，破坏性测试还能很好地证明整个分布式系统的健壮性。

所以，与其老生长谈一些传统的测试方法，不如我们一起看看更新鲜、更好用的破坏性测试。

什么是破坏性测试？

顾名思义，破坏性测试就是通过有效的测试手段，使软件应用程序出现奔溃或失败的情况，然后测试在这样的情况下，软件运行会产生什么结果，而这些结果又是否符合预期。

这里需要注意的是，我们需要使用的测试手段必须是有效的。为什么这样说呢，有两点原因。

第一，破坏性测试的手段和过程，并不是无的放矢，它们是被严格设计和执行的。不要把破坏性测试和探索性测试混为一谈。也就是说，破坏性测试不应该出现，“试试这样会不会出问题”的假设，而且检验破坏性测试的结果也都应该是有预期的。

第二，破坏性测试，会产生切实的破坏作用，你需要权衡破坏的量和度。因为破坏不仅仅会破坏软件，还可能会破坏硬件。通常情况下，软件被破坏后的修复成本不会太大，而硬件部分被破坏后，修复成本就不好说了。所以，你必须要事先考虑好破坏的量和度。

破坏性测试的流程与用例设计

说到底，破坏性测试还是一种人为、事先设计的测试方法，所以它的流程与普通的软件测试流程基本一致：都包括设计测试用例、开发测试脚本、执行测试脚本、捕获缺陷、报告缺陷的过程。

破坏性测试与普通测试流程，唯一不同的是，绝大部分普通测试可以在测试失败后，继续进行其他的测试；而破坏性测试，则有可能无法恢复到待测状态，只能停止后续的测试。

所以，在持续交付的哪个步骤和阶段执行破坏性测试，就非常讲究了，你需要经过严密地设计和预判。

所以，在设计破坏性测试的测试用例时，我们通常会考虑两个维度：

第一个维度是，一个破坏点的具体测试，即设计一个或一组操作，能够导致应用或系统奔溃或异常。此时，你需要注意两个问题：

1. 出现问题后的系统或软件是否有能力按预期捕获和处理异常；
2. 确认被破坏的系统是否有能力按照预期设计进行必要的修复，以确保能够继续处理后续内容。

第二个维度是，整个系统的破坏性测试，我们通常会采用压力测试、暴力测试、阻断链路去除外部依赖等方法，试图找到需要进行破坏性测试的具体的点。

这两个维度的测试方法、流程基本一致，区别只是第二维度的测试通常不知道具体要测试的点，所以破坏范围会更大，甚至可能破坏整个系统。

破坏性测试的执行策略

由于具有切实的破坏力这个特点，我们在执行破坏性测试时需要考虑好执行策略，以避免发生不可挽回的局面。

一般情况下，在发布前执行破坏性测试相对比较安全。但这也不是绝对的，比如你一不小心把 UAT 等大型联调环境搞坏了，其代价还是很可观的。

因此，绝大部分破坏性测试都会在单元测试、功能测试阶段执行。而执行测试的环境也往往是局部的测试子环境。

那么问题又来了，真实环境要比测试子环境更复杂多变，在测试子环境进行的破坏性测试真的有效吗？这真是一个极好的问题。

所以，最近几年，技术圈衍生出一个很流行的理论：混沌工程。

混沌工程

随着分布式系统架构的不断进步，传统的破坏性测试也越发捉襟见肘，最主要的问题有两个：

第一，它被设计得太严格，以至于失真了。而真正有破坏力的故障，都是随机的、并行的、胡乱的。

第二，它覆盖不了生产环境，只能做到类似抽样检验的能力，且很难重复和持续。

所以，混沌工程的理论就应运而生了。

混沌工程是在分布式系统上建立的实验，其目的是建立对系统承受混乱冲击能力的信心。鉴于分布式系统固有的混乱属性，也就是说即使所有的部件都可以正常工作，但把它们结合后，你还是很难预知会发生什么。

所以，我们需要找出分布式系统的这些弱点。我把这些弱点归为了以下几类：

- 当服务不可用时，不可用或不完整的回退能力；
- 不合理的设置超时时间引起的重试风暴；
- 依赖服务接收过多的流量，从而导致中断；
- 由单个故障点引起的级联故障；
- ……

我们要避免这些弱点在生产过程中影响客户，所以需要一种方法来探知和管理这些系统固有的混乱，经实践证明，通过一些受控实验，我们能够观察这些弱点在系统中的行为。这种实验方法，就被叫作混沌工程。

说到具体的实验方法，需要遵循以下 4 个步骤，即科学实验都必须遵循的 4 个步骤：

1. 将正常系统的一些正常行为的可测量数据定义为“稳定态”；
2. 建立一个对照组，并假设对照组和实验组都保持“稳定态”；
3. 引入真实世界的变量，如服务器崩溃、断网、磁盘损坏等等；
4. 尝试寻找对照组和实验组之间的差异，找出系统弱点。

“稳定态”越难被破坏，则说明系统越稳固；而发现的每一个弱点，则都是一个改进目标。

混沌工程也有几个高级原则：

1. 使用改变现实世界的事件，就是要在真实的场景中进行实验，而不要想象和构造一些假想和假设的场景；
2. 在生产环境运行，为了发现真实场景的弱点，所以更建议在生产环境运行这些实验；
3. 自动化连续实现，人工的手工操作是劳动密集型的、不可持续的，因此要把混沌工程自动化构建到系统中；
4. 最小爆破半径，与第二条配合，要尽量减少对用户的负面影响，即使不可避免，也要尽力减少范围和程度。

这样，就更符合持续交付的需求和胃口了。

Netflix 公司的先驱实践

Netflix 为了保证其系统在 AWS 上的健壮性，创造了 Chaos Monkey，可以说是混沌工程真正的先驱者。

Chaos Monkey 会在工作日期间，随机地杀死一些服务以制造混乱，从而检验系统的稳定性。而工程师们不得不停下手头工作去解决这些问题，并且保证它们不会再现。久而久之，系统的健壮性就可以不断地被提高。

Netflix 公司有一句名言，叫作“避免失败的最好办法就是经常失败”。所以，Chaos Monkey 会在日常反复持续执行，真正地持续融合在系统中。这，也为其持续交付中的测试提供了很好的借鉴。

总结

破坏性测试能够很好地测试分布式系统的健壮性，但也因为其破坏特点，使得它在持续交付中无法显示真正的威力；而混沌工程的提出，很好地解决了这个问题，使破坏性测试的威力能够在持续交付过程中被真正发挥出来。

混沌工程的一个典型实践是，Netflix 公司的 Chaos Monkey 系统。这个系统已经证明了混沌工程的价值和重要性，值得我们借鉴。

思考题

你是否考虑过要在自己的公司引入 Chaos Monkey？如果要引入的话，你又需要做些什么准备呢？

感谢你的收听，欢迎给我留言。

通过留言可与作者互动