为什么说优秀架构师往往是一个悲观主义者?

DBAplus社群 4天前

以下文章来源于阿里技术 ， 作者游骥

阿里技术

阿里巴巴官方技术号，关于阿里的技术创新均呈现于此。

一个优秀的架构师通常都是一个悲观主义者，除了设计好能够支撑业务持续发展的优雅架构，另一个容易被忽略的重要能力在于充分考虑失败场景。如果对失败场景考虑不够充分，轻则出现业务不可用，影响用户体验和企业声誉；重则导致数据永久丢失、业务再无恢复可能。

2001年9月11日，美国世贸中心双子大厦遭受了谁也无法预料的恐怖打击，灾难发生前约有350家企业在世贸大厦中工作，事故发生一年后,重返世贸大厦的企业变成了 150家,有200家企业由于重要信息系统的破坏,关键数据的丢失而永远关闭、消失了，其中的一家公司声称自己要恢复到灾难前的状态需要50年的时间。

“Everything fails, all the time”，无论是在传统软件时代还是在互联网、云时代，系统终究会在某个时间点失败，面向失败的设计理念数十年来并没有多大的变化，不同的是在分布式、云架构的互联网时代：失败将由小概率偶发事件变成常态，同时应对和处理失败的具体实现方式也大相径庭。

一、无所不在的失败场景

单个技术点在绝大部分时间都能按照设想正常工作，但是当规模和复杂度到达一定程度，失败其实无所不在。

当你的业务场景从服务企业内部的几百号员工变成面向上亿的外部用户，你不确定你的用户群会有些什么样的角色，也不知道他们会在你的系统平台上创造出什么样的业务行为；当你的技术框架从单机、一体机演进到分布式的多层、多组件架构，原本5个以内的技术组件可能变成了今天的500个，并且为了用较低的成本保持服务能力的扩展能力，你可能放弃了稳定性更好但也昂贵的商业技术、转而用开源自建来替代。

互联网业务快速发展不仅直接带来了流量、安全等不确定性，同时促使了技术架构的快速演进，使架构变得越来越复杂，这些因素都将导致失败发生的概率大幅提升。

当人类的工作、生活越来越依赖互联网，一旦出现失败，造成的影响和损失将是空前巨大的。在远古时代，人类没有自来水也没有电，一切都很好；今天如果停电停水一段时间，相信很多人都会无法适应，而互联网正在逐步演变成跟水和电一样的基础设施。失败的原因多种多样，抽象来看可以分为以下几类：

1、硬件问题

首先，硬件是有生命周期的，它一定会老化，并且你不知道它会在什么时候坏；其次，硬件是一个实体，它存在于客观环境当中，它的状态会受外部环境干扰，比如火灾、地震等外力因素都可能导致硬件损坏；最后，所有硬件都会存在残次品，你很可能就是那个不幸者。通常情况下单个硬件出问题的概率不高，但是当有几十万的硬件设备，硬件的失败问题每天都会发生。

2、软件bug

即便是最优秀程序员写出来的程序，经过最优秀测试同学的严格测试后的代码，上线依然无法做到完全没有bug。互联网业务迭代往往讲究一个“快”字，以往几个月或者几年升级一次的软件程序，现在一周就需要升级一次或者多次，这大幅提升了软件出错的可能性。

3、配置变更错误

系统运行态的日常运维过程当中，难免会因为疏忽或者考虑不周全导致灾难。当上万名技术同学跟上百个变更系统做笛卡尔积，哪怕是6个9的可靠性，依旧无法做到万无一失。全局的流量入口、权限与安全验证体系、统一网关与接口平台等技术环节是可能促发全站不可用的重要风险点，对于影响面大的配置的变更需要尤为谨慎。

4、系统恶化

原本工作得很好的程序随着时间的推移可能有一天不再正常工作，举几个常见的例子：自增变量运行了很长一段时间后出现越界、缓存随着数据量的逐渐变大而出现空间不足、数据库连接池随着机器的扩容而不够用等等。千万不要认为运行良好的系统是不会出问题的，它的代码里面可能藏了定时炸弹，只是你不知道会在什么时间点爆炸。

5、超预期流量

某一天你的系统可能突然会承受远超过预期的每秒请求数，特别是在“中国特色”的互联网场景之下，你很难精确预估系统各个时间点的业务访问量。

6、外部攻击

你需要考虑各种攻击行为，包含流量攻击和安全攻击。你的系统可能随时会面临着DDOS和CC类攻击，你传输的数据可能会被盗取或者篡改。

7、依赖库问题

你的系统很可能会用大量的二方库或者三方库，它们对你来说是黑盒子，你不了解它们存在哪些风险，并且你无法掌控。这些库可能会存在漏洞、可能会有bug，可能会大量消耗你的系统资源，总之不要太信任它们。

8、依赖服务问题

你依赖的服务也一定不会100%可用，它们可能会超时，可能会失败。当依赖服务超时的时候，如果你没有很好地处理，可能会导致你自己的系统无法工作，在分布式场景下，这种失败状态会持续辐射，最终导致大面积的不可用。

二、如何面向失败设计

作为一个悲观主义者，你需要在一开始的系统设计阶段就考虑到以上各种失败场景，把面向失败当成系统设计的一部分，并且准备好从失败中恢复的策略，这有助于更好地提升整个系统的可用性。

只有你意识到事情会随着时间的推移而失败，并将这种思想融入到体系结构中，那么在失败发生的时候你才能完全不受影响或者将失败损失降到最低。面向失败的设计理念数十年来并没有多大的变化，一些好的经典原则在今天依旧被广泛运用。

1、冗余设计避免单点故障

硬件和软件都不可靠，环境和人都存在极大的不确定性，虽然无法避免失败场景的发生，但是可以通过冗余设计来规避局部失败对系统的影响。

冗余设计避免单点故障这一策略在互联网技术架构中处处可见，比如重要的服务通常都会部署多个、数据库的主备结构、服务调用的重试机制、存储的多副本等概念都属于这一范畴。

2、面向失败的宏观多活架构

除了局部失败场景，你的系统可能还面临着大范围的失败场景。大范围的原因有两个：

- 天灾，比如火灾、地震、台风、雷电等大的自然灾害可能导致大面积的基础设备被毁坏；
- 人祸：人的失误或者刻意破坏行为有时候也会酿成大祸，如操作错误、破坏、植入有害代码和恐怖袭击。

“面向失败的宏观多活架构”从宏观架构的高可用层面来解决系统的整体可用性问题，随着技术的演进，冷备、热备、两地三中心、异地多活等应对大范围失败场景的技术体系这些年频频被提起。

3、服务能力与依赖调用自我保护

如何来衡量一个软件系统的设计是否优良？一条很重要的衡量标准——在任何情况之下你的软件系统都应该工作在当前环境的最优状态。

每个人都知道机翼是飞机的重要部件，一旦机翼出现问题，飞机很可能就会坠落。然而在二战当中，许多战斗机即便机翼千疮百孔了，依然保持着最佳战斗能力；甚至还有更夸张的情况：1983年的一次战

斗机演习当中，一架飞机由于事故损失了一个机翼，这架缺少一个机翼的飞机依然保持了飞行能力、最终完成安全着陆。

软件系统由两部分构成：系统自身的代码和依赖的库以及服务。“服务能力与依赖调用自我保护”需要从这两块分别切入构建系统在任意情况都始终工作在最佳状态的能力。服务限流、系统负载保护、给依赖的服务设置超时或者资源限制等都是相应的应对策略。

4、为一切不可预料的情况备好预案

能够抵抗失败和从失败中快速恢复是面向失败设计的核心思想，然而即便已经做了万全的设计，也并非所有的失败场景都是系统能够自动抵御的。

你需要考虑到所有的失败场景，并准备好相应的应对预案。为一切不可预料的情况备好预案才能在失败场景真正发生时做到有条不紊。做好预案需要对失败场景有全面的考虑：会发生哪些失败？失败会带来什么问题？应对策略是什么？预期的恢复时间多久？恢复后的影响面有多大？需要通知到哪些角色？等这一系列的因子构成了一个完整的预案体系。

5、自动化运维管控

大量的系统故障是因为人的失误造成的，即便让一个优秀的运维工程师进行一万次同样的运维操作也难免不出错。

唯一的解决办法便是在运维过程当中尽可能降低人为操作的比重。系统化、白屏化是第一个阶段——将人为的操作步骤固化成系统程序，避免操作失误；自动化以及智能化是第二个阶段——将正确的决策过程也固化成智能程序，避免决策失误。

同时所有的运维动作都需要遵循灰度原则，做到可灰度、可监测、可回滚，即便出现了失误也能控制好爆炸半径，并且做到快速恢复。

6、精细化的监控体系

面向失败设计不仅要求你的系统足够健壮，同时要求你能够在第一时间感知到失败的发生。无论是自动化的系统恢复，还是人为介入，如果你压根都不知道是哪里出问题了，一切都将束手无策。

精细化的监控体系一方面能够在出现问题的时候以最快的速度将最准确的信息传递到人或者运维系统，同时它还能够展现趋势、进行提前预警。

AI技术的结合使得监控领域在近几年得到了新的发展驱动力：智能监控报警、根因定位、智能预测、智能决策等能力都是学术界和工程界非常热衷的课题。

7、故障与攻防演练锤炼容灾应急能力

最后，即便以上工作都做好了，你也不能高枕无忧去等待失败到来。你的设计、系统、流程、技术人员等需要通过不断演练，来保障能力和进化升级。对于代价非常巨大的事件，做好前期的充分演练是非常有必要的，比如军事演练、消防演练等都属于这一范畴。

而系统不可用的代价对于企业来讲很可能是无法承受的，因此需要在平时做好充分的演练：通过故障与攻防演练锤炼容灾应急能力，对面向失败的设计做好充分验证。只有当所有的失败场景都被提前演练过，当失败真正来临时才能做到胸有成竹。

作者丨游骥
来源丨阿里技术（ID：ali_tech）
dbaplus社群欢迎广大技术人员投稿，投稿邮箱：editor@dbaplus.cn