架构提效的矛盾和矛盾的主要方面

原创 技术架构 王新栋 京东零售技术 2025年01月03日 08:15 北京

本文导读

在软件开发领域，架构设计是确保系统高效、稳定运行的重要环节或者称之为重要动作。无论架构从简单到复杂，还是从复杂回归简洁的演变过程。在这个过程中，又无论是初创公司还是大型企业，架构提效始终是技术团队的核心追求。本文将从稳定、性能、代码三大维度出发，结合实战经验，探讨如何有效提升架构效能。

为什么要选择或者认为这三个维度是必要要素呢？

> “一切事物中包含的矛盾方面的相互依赖和相互斗争，决定一切事物的生命，推动一切事物的发展。没有什么事物是不包含矛盾的，没有矛盾就没有世界。”

当然架构也有自身的矛盾统一，在架构提效上，系统的运行正常和问题频出是一对矛盾，功能的快和慢是一对矛盾，工程的整洁有序和无序是一对矛盾。这三对矛盾正是架构提效的矛盾。

如果不稳定，系统三天两头出故障，研发人员成了救火队员，系统的效率将无从谈起，稳定是我们谈架构效率的基础。如果性能不高，在网络基础环境稳定的情况下，访问一个页面3S才响应，那我们也不好意思说架构有效率。如果代码乱成一锅粥，比如大段大段面条式的代码，再比如满眼望去N多个if结构语句，研发人员加一个功能都要查找好久，也是无颜谈效率。

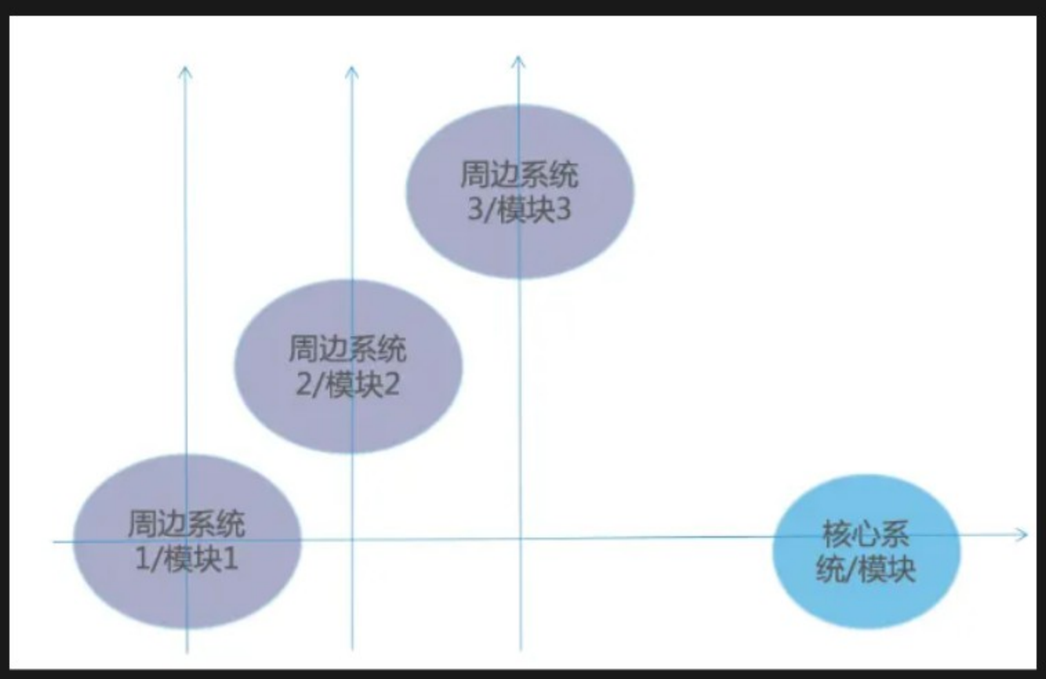
因此，我们认为，稳定、性能、代码是架构提效矛盾中的主要方面。接下来我们将从这三个主要方面去介绍。

软件工程发展了这么多年，高可用、高扩展、高并发已经有大量的文章篇幅，从宏观的角度去讲如何做微服务、如何分库分表，如何使用缓存等等。因此呢，本篇文章想聚焦到架构矛盾的微观层面，也就是偏工程结构、偏代码方面去阐述这三个要素。另外本篇文章的思想也参考了前辈们的研究成果，我也附在了文末。

稳定：架构的基石与守护神

“万事万物都是运动的，发展的”。业务功能变多，用户数量变多，团队规模变大。如果没有规则和规范的引导和约束，系统逐渐野蛮生长，逐渐碎片化。那么，我们的系统何谈稳定呢。

我们就希望能找到这样的一种规则、规范 —— 正交分解或者叫做正交设计。



架构设计的过程就是一个业务正交分解的过程。

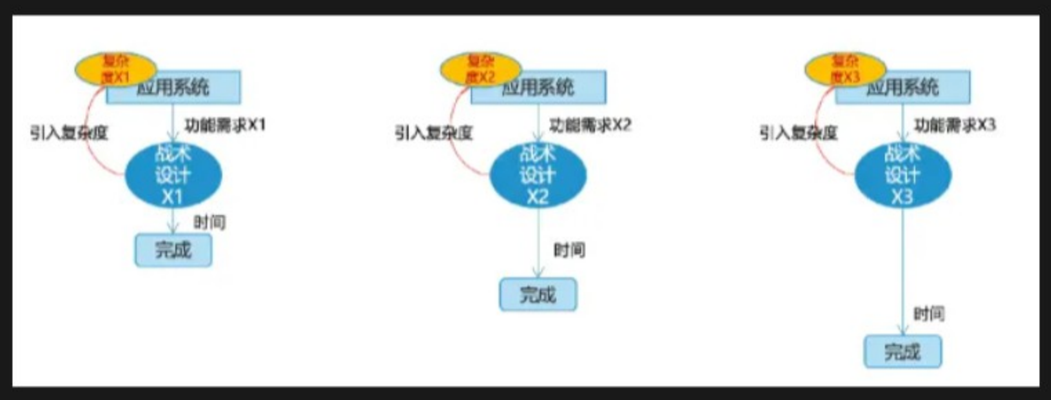
架构设计并不仅仅是技术层面的规划，更重要的是对业务逻辑的深入理解和把握。通过正交分解，我们可以将复杂的业务系统拆解成若干个相互独立但又彼此关联的模块或组件。这些模块或组件在保持功能完整性的同时，还能实现高度的内聚和松散的耦合，从而提高系统的可扩展性、可维护性和可重用性。

正交分解的关键在于消除重复、分离关注点和管理依赖。通过这一方法，我们可以将业务系统中的公共部分和可变部分进行明确的划分，从而实现对业务逻辑的精准掌控。在架构设计过程中，我们需要不断地对业务进行抽象和分解，直至得到一系列规模可控、结构清晰的小模块。这些小模块通过组合和协作，能够形成更加复杂且功能完善的软件系统。

因此，正交分解的思想是我们架构设计保障稳定的重要方法基础。

性能：速度与效率的双重考验

想快，就使用“战术设计”。曾经这是很多程序员的法宝，因为他们认为这样开发“确实快”。



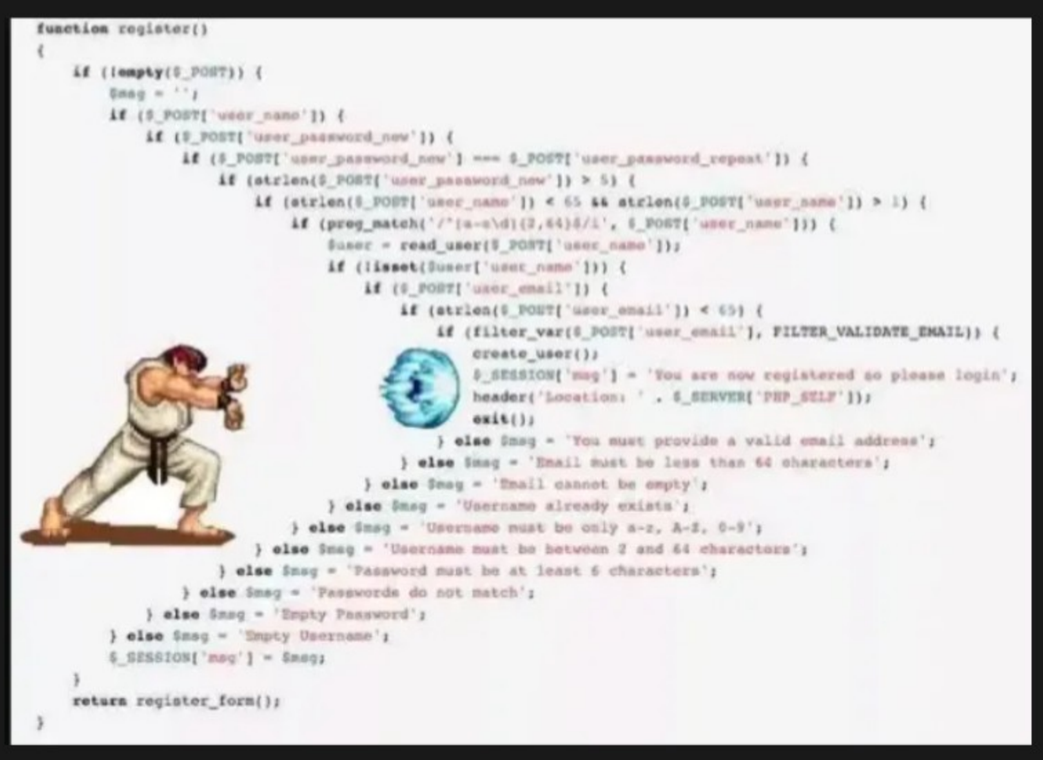
大多数程序员以称为战术编程的心态来进行软件开发。在战术方法中，主要重点是使某些功能正常工作，例如新功能或错误修复。乍一看，这似乎是完全合理的：还有什么比编写有效的代码更重要的呢？但是，战术编程几乎不可能产生出良好的系统设计。

想快的“战术设计”会造成常见的下面这样的情况。

“团队新人不熟悉系统，为了急于上线一个特性，又不想影响到系统的其他部分，就会很自然地在某个地方加一个 flag，然后在所有需要改动的地方加 if 判断，而不是去调整系统设计以适应新的问题空间。”

在一个充满活力的软件开发团队中，新成员小张刚刚加入不久。他对于团队正在使用的复杂系统还不是很熟悉，但面对紧迫的项目进度和上级施加的压力，他急于证明自己，并希望能尽快为团队做出贡献。团队正计划上线一个新的特性，这个特性需要在不干扰系统其他部分的前提下实现。

小张在浏览了系统的代码库后，发现要全面理解并调整整个系统设计以适应新的特性，需要花费大量的时间和精力。他深知自己作为新人，在这方面还有所欠缺，因此，他决定采取一个他认为更为“高效”的方法：在某个关键位置添加一个临时的标志位（flag），然后在所有需要改动的地方都加上if判断，以确保新特性能够按时上线，同时尽量减少对现有系统的影响。



（图自网络）

虽然这种方法在短时间内确实让新特性得以顺利上线，但团队中的资深成员很快便发现了潜在的问题。这种做法虽然看似快速解决了问题，但实际上却在系统中埋下了隐患。它不仅增加了代码的复杂性，降低了代码的可读性和可维护性，还可能在未来引发更多的bug和性能问题。更重要的是，这种做法违背了软件开发中的最佳实践，**即应通过优化系统设计来适应新的问题空间，而不是通过添加临时性的补丁来解决问题。**

“几乎每个软件开发组织都有至少一个将战术编程发挥到极致的开发人员：战术龙卷风”，而且常常被视为团队”英雄“，因为能“快速完成任务且高产”。

“战术龙卷风”通常以战术编程为主要手段，即采用最快速、最直接的方法来解决当前的问题，而不考虑长远的影响和代码的可持续性。这种方法在初次使用时往往能够取得显著的效果，任务完成得既快又好，赢得了团队成员的赞誉和领导的认可。

然而，随着时间的推移，“战术龙卷风”所留下的隐患逐渐暴露出来。由于缺乏对系统设计的深入理解和长远规划，他的代码往往难以维护和扩展。当需要添加新功能或修复bug时，团队成员往往需要花费更多的时间和精力来理解和修改他的代码。因此，第二次和第三次修改时，效率会大幅下降，甚至可能引发新的问题。

实际造成结果：第一次快、第二次慢、第三次更慢。

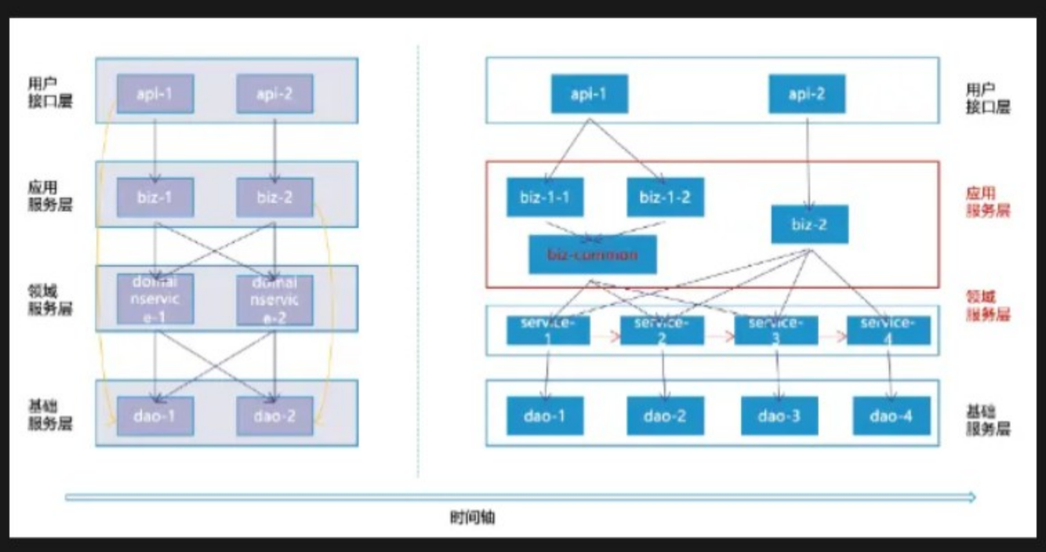
代码：简洁与优雅的双重追求

我们做业务开发，代码的优雅简洁，不能局限在一段方法，还是要从整个工程结构然后再到类、到方法，这样从宏观到中观再到微观的整体去要求。我们的应用工程结构，常见大致分为四层。分别是api层、biz层、domain层和dao层。

这个时候我们就要很清晰地熟悉每一层的职责，然后将我们的代码放入进去。首先，api层的作用，正如它的名字一样，是提供api服务的。向谁提供api呢，比如客户端，比如APP端、pc端等等，公司外面的客户，比如isv等。其次，biz层的作用，这一层也叫业务服务层。它主要负责编排。把一个业务场景下的主流程逻辑处理完成。这个主流程会涉及到多个原子接口，就在这层负责组装。再次，domain层的作用，也叫做领域服务层。按照OO思想，领域编程的思维，我们的”厚对象“的代码都在这层。比如订单域、运费域等。这里对“这一层的位置”多说几句，在没有形成领域之前，这层一般叫service层，不过我们都是建议领域思维编写代码。最后是dao层，也就是我们的存储层了，负责持久化。

在清晰了每一层的作用之后，如果我们的代码职责也是按照这样逐层放入的，那么大体是符合我们的整洁要求的。但是呢，随着时间的推移，需求的增多和变化。原来整洁的工程结构和代码已经不那么优雅了。

这个时候，一般会出现两类现象，一类是业务层（biz）变的臃肿，能力层（domain）变的单薄。另一类是出现了网状调用。而且这两类现象也很有可能是混合在一起出现。



这两类现象会直接带来下面4种结果。

1、biz层越来越”胖“。胖了之后，还长成了两小层。上小层是面向单一业务场景的“业务biz层”，下小层成了通用场景可复用的“通用biz层”。

2、service层越来越”瘦“。当service层变薄了以后，也就只能沦为service了，而这样的service层跟dao层实际没什么区别，更不能再称之为domain层或者没有机会演变成domain层。

3、但是也不是所有的service层都变瘦、变薄了。可能有的萎缩，有的膨胀。人员与设计的差异，导致颗粒度不一。

4、出现了网状调用。原本biz-1 -> service-1 的实现链路下，随着新增业务逻辑，又新起了一个service-2，链路演变成了biz-1 -> service-1-> service-2。“这样的趋势持续发展下去，会发现biz-1下的service调用链路越发的复杂，呈现为一棵深度调用树，而biz层失去了业务编排的作用退化为一个业务场景入口的标志符”。有可能后面继续3-4-5-6，越挖越深，不见尽头。

很显然，到这里，这样的结构现状，代码现状，已经远离了我们简洁和优雅的初衷。也谈不上提效了。

到此，我们介绍了架构提效中的稳定、性能和代码这三个主要的方面，限于篇幅和架构本身的实践性，还需要我们在架构提效上进行持续的优化。需要我们在稳定、性能、代码三大维度上不断探索和实践。通过高可用架构设计、性能优化策略、模块化与解耦、代码质量与规范等措施，我们可以构建一个既稳定又高效，且易于维护和扩展的系统。

> “在复杂的事物发展过程中，有许多的矛盾存在，其中必有一种是主要的矛盾，由于它的存在和发展规定和影响着其他矛盾的存在和发展。”

架构的发展本身也是对抗熵增这一矛盾的过程，我们上面描述的稳定、性能和代码中的矛盾方面有是围绕和关联这一主要矛盾的。在这个过程中不仅有系统的有序变无序，也有组织的简单变复杂。我们既要关注技术层面的提升，更要注重团队协作、知识共享和持续改进的文化建设。只有这样，我们才能在快速变化的市场环境中，保持竞争力，不断前行。

最后，如果您对文章中的表述和观点有任何问题，亦或者您当前也正在致力于架构提效方面的工作，我们可以一起学习交流。

REF：

- https://time.geekbang.org/column/article/167844 如何判断架构设计的优劣

- https://cactus-proj.github.io/A-Philosophy-of-Software-Design-zh/ 软件设计的哲学
- https://mp.weixin.qq.com/s/jJzzJlGozOpt7KaXwBS3Ww 业务系统架构实践总结
- 《矛盾论》,《毛泽东选集》第一卷

你可能还想看👇

Java代码之美，从遵循样式规范开始

一位架构师的自述：在尚未踏入的世界成为你自己

- **END** -

👇点关注，不迷路👇