架构师之道：介绍了那么多，技术中架构到底什么？

MobotStone 2024-04-24 3,548 阅读26分钟 专栏：架构师之道：程序员修炼法典

智能总结 复制 重新生成

文章探讨了软件架构的相关内容，包括其诞生背景、与相关概念的关系、定义及边界等。软件架构的出现是为应对系统规模扩大带来的挑战。通过对比系统与子系统、模块与组件、框架与架构等概念深入理解。多位大师给出不同定义，如 IEEE 的组织、组件等关系与原则，Martin Fowler 的重要且难改的决策等。还阐述了结构、架构属性、架构决策、设计原则等方面。最后总结了行业内大师对软件架构定义的特色，并提到作者对 Ralph Johnson 定义的偏爱。

关联问题：架构决策如何做权衡 架构属性怎样选重点 软件架构为何难定义

基于该文章内容继续向AI提问

通过前面的介绍，我们对架构的历史脉络有了一些基本的认识。我们来深入思考一下：这个经常出现在各种技术讨论中的“架构”，到底是什么意思呢？

是的，我们经常听到“架构”这个词，但是真正停下来思考它的定义和内涵，你会发现不是那么简单。在技术界，尤其是软件开发领域，“架构”几乎成了一个时髦的词汇，人人都在谈论它，但是真正理解它的精髓的人却不多。那么，什么是软件架构呢？它又为什么如此重要？

在我们继续深入探索之前，让我们首先澄清一下“架构”的基本含义。接下来的内容，我们将从软件架构的定义谈起，探索它的核心组成部分，然后再进一步讨论它在软件开发过程中的重要作用和价值。这样，我们不仅能够更好地理解架构这个概念，而且还能够明白为什么它对于构建高效、可维护的软件系统如此关键。

1、架构的诞生

在探讨软件开发的众多新方法和理念时，我们发现“软件架构”的概念显得与众不同。不同于其他为应对新兴软件危机而诞生的理念，软件架构的出现似乎并不直接源于行业共面的某个特定问题。这里面有何玄机呢？

随着软件系统规模的日益庞大，一项明显的变化是，传统的计算算法和数据结构已不再是设计挑战的主要焦点。在一个由众多部件构成的系统中，如何高效地组织这些部件——即我们所称的“软件架构”——成为了设计师们面临的一系列新问题。比如，你可能遇到如下挑战：

- 系统规模庞大至一定程度，内部耦合变得异常复杂，严重拖慢了开发效率；
- 由于部件之间的紧密耦合，对系统的任何微小修改都可能引发连锁反应，使得后续的维护和扩展工作变得异常困难；
- 复杂的系统逻辑使得问题频发，一旦出现问题，定位和修复的难度极大。

在这样的背景下，“软件架构”的概念应运而生，其历史地位和必然性不言而喻。回望过去，我们可以发现，第一次软件危机推动了“结构化编程”的发展，带来了“模块”的概念；随后，第二次软件危机促使“面向对象编程”的普及，引入了“对象”概念。而“软件架构”的提出，则标志着“组件”概念的诞生。

这一发展历程揭示了一个核心思想：无论是“模块”、“对象”还是“组件”，其本质都在于对达到一定规模的软件系统进行有效的拆分和高层次组织。随着软件的复杂度不断攀升，这种拆分的粒度和层次也随之提高，从而更好地应对软件开发过程中遇到的各种挑战，提高软件的开发效率和系统的可维护性。

2、架构指什么

在我们程序员的世界里，“架构”这个词几乎无处不在。它就像是我们的老朋友，经常挂在嘴边。但是，当你真正停下脚步，试图去探究一下这个问题：“架构”究竟指的是什么？你可能会惊讶地发现，这个问题并没有想象中那么简单。其实，如果你随便问1000个技术人员，“架构”的定义是什么，你可能会得到1001种不同的答案。这就像是一个技术界的谜题，每个人心中的答案都有微妙的差异。

那么，在这个众说纷纭的情况下，我们如何才能找到对“架构”的准确理解呢？一个有效的方法是先从理解与“架构”紧密相关且相似的几对概念开始。我们可以聚焦于三对基本但至关重要的概念：**系统与子系统、模块与组件、以及框架与架构**。

2.1、系统与子系统

当我们聊到“系统”，可能首先想到的是各种复杂的技术或机械设备。维基百科给出了一个非常宽泛但精确的定义：

> 系统泛指由一群有关联的个体组成，根据某种规则运作，能完成个别元件不能单独完成的工作的群体。

我们来简化一下上面的高深论述，抓住几个核心点：

- **关联**：想象一下，仅仅把一个发动机和一台电脑摆在一起，并不能让它们变成什么特别的东西，对吧？但如果你把发动机、底盘、轮胎和车架这些有关联的部分放在一起，它们就能组合成一台汽车。这就是说，系统里的每个部分都得有点关系，才能一起工作形成一个有用的整体。
- **规则**：在这个组合里，每个部分不是随便干自己的事。它们需要按照一定的规则来操作。比如说，汽车里的发动机产生动力，通过变速器和传动轴这样的装置，最终把动力传到轮胎上，让汽车能够前进。这些规则决定了谁负责干什么，怎么协作。
- **能力**：当这些部件按规则合作时，整个系统就能做到单个部件做不到的事情，比如汽车的载人载物。这种系统的能力，不是简单地把各个部件的能力加起来那么简单，而是能创造出新的能力来。

再说说子系统，其实它的概念和系统差不多，只不过是从另一个角度来看。一个系统在更大的环境中，可能就是另一个系统的子系统：

> 子系统由一群有关联的个体所组成的系统，多半会是更大系统中的一部分。

MobotStone LV.5
高级系统架构师 @保密
作者榜No.6 优秀作者
230 文章 364k 阅读 539 粉丝
关注 私信

目录 收起

子系统和系统的概念其实是一回事，只是取决于你站在哪个视角看问题。一个系统在更大的环境中就可能成为另一个系统的子系统。这听起来可能有点绕，但如果我们用微信这个例子来说明，一切就变得清晰多了。

- **微信作为一个系统**：首先，把微信想象成一个庞大的生态圈，它自身就是一个系统。在这个生态圈里，有许多功能和服务，如聊天、登录、支付、朋友圈等。这些功能在微信这个大系统中，实际上都是独立运行的子系统，每个子系统负责不同的任务，但共同支撑起微信这个巨大的服务平台。
- **朋友圈的结构**：再深入到朋友圈，我们可以看到它不仅仅是微信的一个功能，实际上它自己也是一个由多个功能组成的系统。比如，动态发布、评论、点赞等功能，在朋友圈这个“小系统”里，它们各自也可以被看作是独立的子系统。
- **评论功能的深层次**：评论功能虽然是朋友圈的一部分，但如果我们仔细分析，会发现它自身也包含多个子系统，如防刷子系统、审核子系统、发布子系统、存储子系统等。这些子系统通过精密的协作，共同确保了评论功能的正常运行和数据的安全性。
- **技术层面的系统**：当我们聚焦于评论审核这一环节，它可能不再被划分为更多的业务子系统，而是转向技术实现的视角。在这个层面，各种技术模块和组件，如数据库系统MySQL、缓存系统Redis等，它们虽然是从技术角度构成的系统，但同样在整个评论功能中扮演着关键的角色。这些技术组件确保了数据的快速处理和安全存储，虽然它们不直接参与到业务流程中，却是支撑整个系统运行不可或缺的部分。

这种从整体到部分，再从部分回到整体的视角，不仅帮助我们理解了系统内部的复杂结构，也让我们明白了在设计和构建复杂系统时，如何通过层次分明的子系统来管理和简化这种复杂性。每个子系统的设计和实现，都需要考虑如何与其他子系统协作，以及如何在满足当前功能需求的同时，保持系统的灵活性和扩展性。

通过微信这个例子，我们得到的不仅仅是对系统和子系统理论的深入理解，还有对实际应用中这些理论如何被运用以解决实际问题的洞见。这种理论与实践的结合，是软件工程中不可或缺的一部分，它指导我们如何更好地设计和优化软件系统，以适应不断变化的需求和挑战。

2.2、模块与组件

要深入理解软件系统的拆分，我们可以从两个不同的视角来探讨：逻辑和物理。这就像是看待一个复杂机器的两种方式。逻辑上的拆分产生了我们所说的“模块”，而物理上的拆分则给我们“组件”。

- 模块：把这个概念想象成是把一本厚厚的教科书分成不同的章节，每一章节专注于讲解一个特定的主题。在软件里，模块就是这样一个逻辑上的单位，它封装了一系列功能，这些功能紧密相关，共同完成一项或几项特定的任务。划分模块的目的很明确：为了让我们的代码更加有条理，每部分都有明确的职责，这样不仅使得代码更易于理解和维护，还能在团队中更高效地协作。
- 组件：再想想组件，可以将其比喻为乐高积木中的每一块积木。这些积木是实实在在的，你可以用它们来构建各种各样的结构。在软件开发中，组件是物理上的实体，可以在不同的系统中重复使用。它们就像是标准化的零件，可以根据需要组装或更换，极大地提高了开发的灵活性和效率。

提到“组件”的英文“component”，如果我们将其译为“零件”，可能会更加形象和易懂。这个词帮助我们更好地把握组件的本质：它们是独立的、可以互换的物理单位，正如机械中的零件一样，具备了可插拔和可复用的特性。这种物理上的拆分不仅让我们的系统更加模块化，还为系统的升级和维护提供了极大的便利。

通过逻辑和物理拆分，我们能够更精细地管理和构建复杂的软件系统，使之既有条理又高效。模块化和组件化的设计思想，是现代软件工程中不可或缺的一部分，它们使得软件开发像搭建乐高一样既有趣又富有创造性。

2.3、框架与架构

单纯从定义的角度来看，框架关注的是“规范”，架构关注的是“结构”。框架（Framework）为你的项目提供了一套预设的工具和库，确保你可以遵循特定的规范和模式来构建应用。想象一下，如果你正在搭建一座房子，框架就像是提供给你的施工套件和详细指南，它指导你每一步如何建造，确保每一部分都能正确地承担起它的作用。架构（Architecture），则更像是整个系统的设计方案，它详细规划了系统的组织方式，展示了组件如何互相配合，数据如何流转，以及系统如何对外提供服务。这相当于在规划整个房子的布局，决定哪里放置客厅，哪里是厨房，以及如何确保房子既满足居住功能又具有美观性。

我们经常会说，“工程采用的是MVC架构”、“工程使用的是SSH框架”等。这里，**第一句话是站在结构的层面来说明**，它像是在描述房子的设计理念和内部结构，告诉你这个项目是如何划分不同的功能区域，以及这些区域是如何相互关联的。**第二句话是站在规范的层面来说明**，它更多关注于建造过程中应该遵循的标准和规则，比如使用什么样的技术栈，以及如何利用这些技术来实现预定的功能。

同时，不同的视角会影响我们对架构的理解和描述，例如：

- **从业务逻辑的角度分解，“后台管理系统”的架构**(图1-3所示)，我们会关注系统是如何处理用户的注册、信息管理、业务选择和数据统计等业务流程的。这个角度让我们深入了解系统是如何支持核心业务需求的。

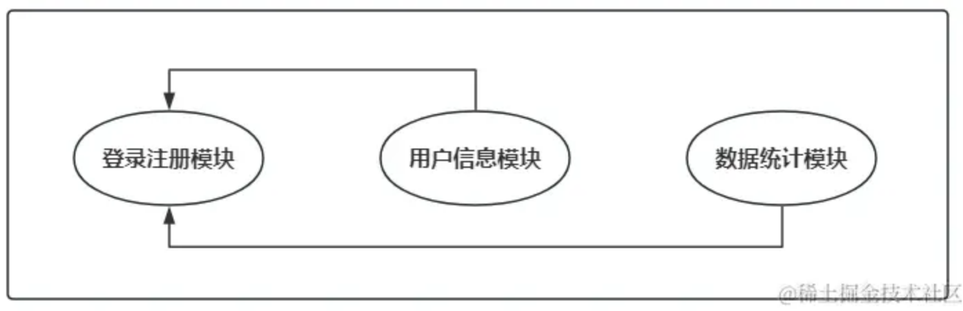
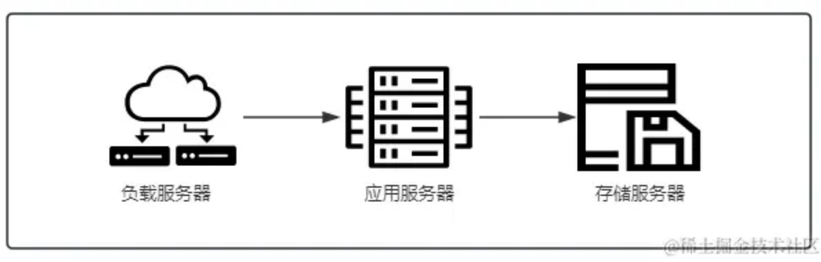
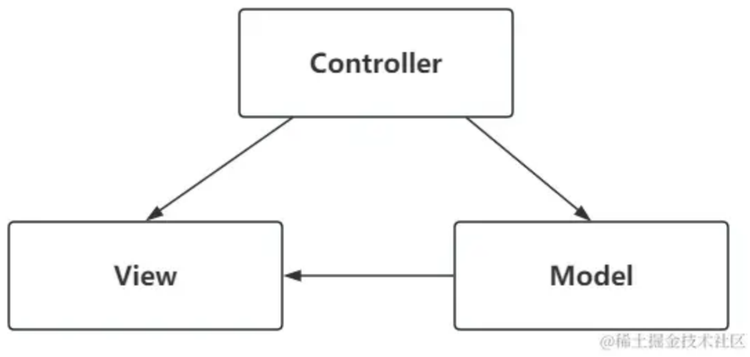


图1-3

- **从物理部署的角度分解，“后台管理系统”的架构**(如图1-4所示)，则关注系统的硬件布局，如何通过服务器集群、负载均衡等技术手段来保证系统的高可用性和高性能。



- **从开发结构的角度分解，“后台管理系统”的架构**(如图1-5所示)，我们则更加关注代码的组织方式、模块的划分以及技术框架的选择，以便于团队协作、功能迭代和系统维护。



通过不同的视角，我们不仅能够更全面地理解框架和架构的概念，还能够根据项目的实际需求，从多个维度评估和选择最合适的架构设计和技术框架。这种多角度的思考方式对于软件开发是非常重要的，它帮助我们构建出既稳健又灵活的系统，能够适应不断变化的业务需求和技术挑战。

3、架构的定义及边界

谈到“架构是什么”，你可能会发现这个问题没有那么简单。其实，不同的公司、不同的团队，乃至于我们每个人，对于“软件架构”的理解都各不相同。这就像我们不能百分之百精确描述一个复杂的模型一样，我们只能从多个角度来看，尝试给出一个大概的描述。所以，想要给出一个完全无懈可击的“架构定义”，那基本是不可能的。

> “道可道，非常道。名可名，非常名”。

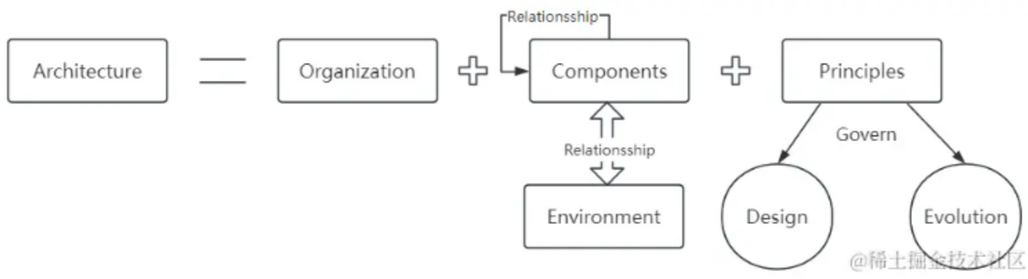
这个行业内部，充满了各种各样的声音和观点。有的组织可能会从技术的实用性出发，定义架构是一种确保系统稳定、高效运行的布局；有的个人可能更注重架构在解决复杂问题中的策略角色，视之为一种艺术。每个人都在尝试从自己独特的视角，解读和定义“架构是什么”，这就像是在无数的色彩中寻找那一抹最为合适的蓝，既充满了个人色彩，也反映了这个领域的多元和复杂。

3.1、架构的定义

> the fundamental **organization** of a system, embodied in its **components**, their **relationships** to each other and the **environment**, and the **principles** governing its design and evolution -- ANSI/IEEE

IEEE 关于架构的这个概念：其实，把它想象成一个三明治，上面是**系统的大架构**，中间夹着的是**各个组件以及它们之间的纽带**（这里的纽带，既包括组件彼此之间的联系，也涵盖了它们与外界环境的互动），底下则是**一套行动准则**。咱们若是用图形来表达这个定义，那么制作出来的图示应该既清爽又直观，能够将架构的主要组成部分和核心思想，一目了然地展示出来(如图1-6所示)：

- **系统的大架构**：这部分其实就是在讲，让人一眼就能看懂的系统整体布局。就像是给你一张地图，上面标注了所有重要的地标和路径，让你对整个地形有个基本的认识。
- **组件及其关系**：这里的意思是，把整个系统拆分成一个个模块，这样不仅便于理解和管理，同时也强调了这些模块之间，以及模块与外部环境之间的相互作用和联系。就像是一张复杂的网络图，每个节点都有它的位置和作用，而线则表示了节点之间的连接。
- **行动准则**：这部分提供的是设计和演进系统时需要遵守的规则和指导原则。就好比是一本指南，告诉你在追求目标的过程中，哪些是应该坚持的原则，哪些是需要避免的错误。



大师 **Martin Fowler**对于架构的定义有着更加简洁的抽象，Martin Fowler 认为软件架构是：**重要并且难以改变的决策**。架构设计是关于权衡的艺术，架构设计过程中充满了各种各样的决策，这些决策也终将反应系统架构。

在讨论软件架构这个课题时，大师 **Martin Fowler** 给出了自己的看法，既精炼又深刻：软件架构，那就是那些**特别重要而且改起来头疼的决策**。这就好比是在说，架构设计其实就是一个充满智慧的权衡过程，你在这个过程中做出的每一个决策，都像是在未来的系统蓝图上，悄悄地加上了你的个人签名。

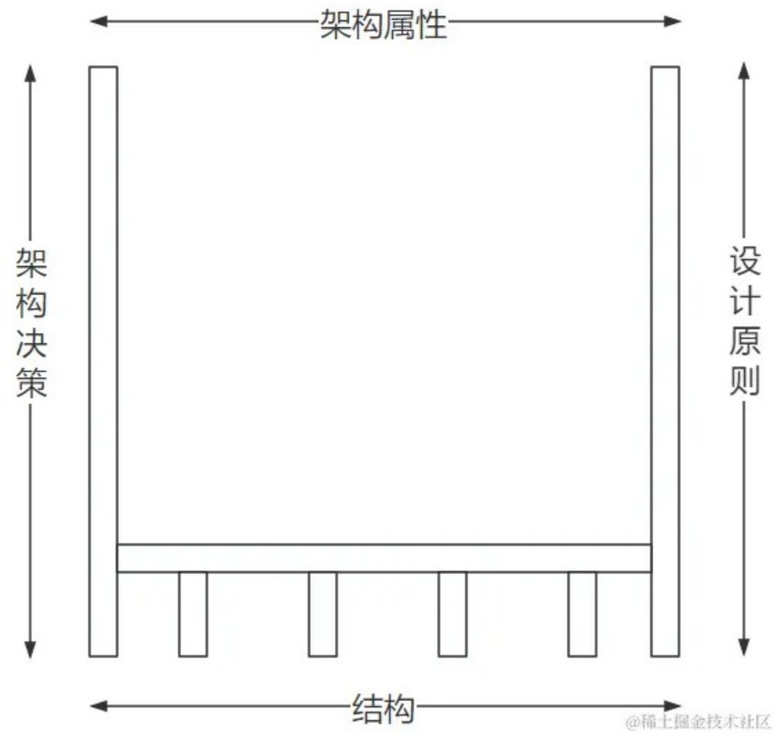
> Software Architecture = **Important** and **hard** to change **decisions** --**Martin Fowler**

而 **Ralph Johnson** 这位大佬，给出了一个更加哲学的定义，他说，软件架构嘛，简单点说，就是那些重要的东西，具体是啥呢？他似乎在告诉我们：这个问题，答案就像是宇宙的终极问题，重要的是你怎么去理解它！

> The software architecutre is the **important stuff** ! **Whatever it is** ! --**Ralph Johnson**

与之相比，Neil Ford 则像是那位脚踏实地的实践者，他不在云端飘，而是带着工具箱走到了前线。他从实际操作的角度出发，详细列出了构成软件系统架构的基石(如图1-7所示)：

- **结构**：这不仅仅是选一个风格的问题，而是定义了整个应用的骨架，是微服务、是单体，还是服务导向架构（SOA），每一种选择都像是在向系统注入不同的灵魂。
- **架构属性**：这些看不见摸不着的特性，比如性能如何、可用性高低、维护起来是不是友好，它们决定了系统能否在残酷的环境中生存下来。
- **架构决策**：在设计系统的过程中，那些至关重要的选择点，就像是在茫茫大海中航行的舵手，一次又一次地决定着软件的未来方向。
- **设计原则**：这些原则就像是老船长的智慧，指引着设计者避开暗礁，顺利抵达目的地。



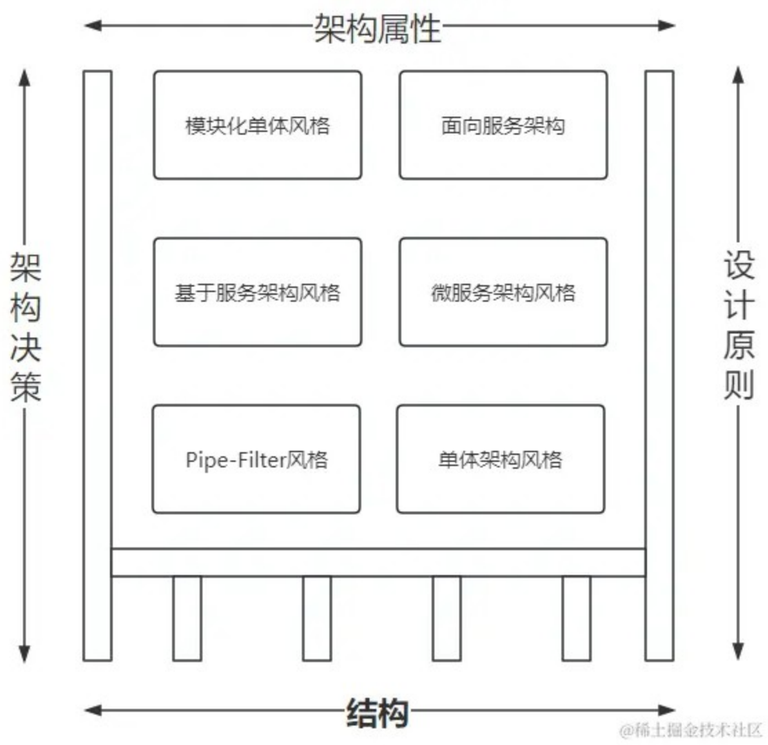
3.2、结构

结构是系统架构的重要组成部分，其从宏观上表述了系统的结构组成。架构设计的核心任务之一是为系统选择合适的架构风格。架构师需要基于上下文的权衡，可以选择模块化单体架构风格，也可以选择微服务架构风格。

说到架构中的“结构”，这块可不是随便一说就能过去的。它在整个系统架构里面占的位子，就好比心脏在人体里的角色一样重要。它从一个宏大的视角，告诉我们这个系统是由哪些部分组成的，每部分又是怎么协同工作的。搞架构设计的时候，挑选一个恰到好处的架构风格，基本上就是架构师的核心日常了。

举个例子，架构师们在面对具体的项目时，得先在大脑中过一遍这个系统的全貌，然后根据这个系统要面对的各种情况和挑战，来做一个权衡。这时候，他们可能会选择走模块化的单体架构风格，这种方式好处是简单、直接，容易管理；但如果项目特别复杂，服务之间需要高度的解耦，那微服务架构风格可能就更合适了，虽然这样会带来更多的管理和协调工作，但它能提供更好的灵活性和扩展性。

“选择合适的架构风格”的过程(如图 1-8所示)，其实就像是在做一场精心策划的活动，架构师需要凭借自己的经验和对项目需求的深刻理解，做一个详细的规划，希望能赢得未来的便利和效率。这个过程充满了挑战，但也正是这些挑战，让架构设计成为了一门艺术。



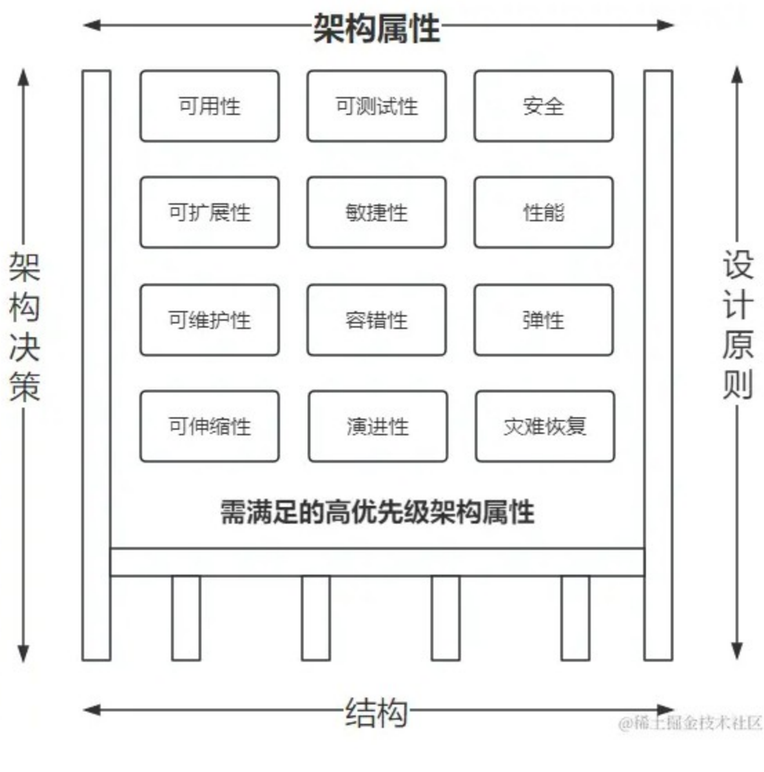
3.3、架构属性

在聊软件架构的时候，绝不能忽视的一个重点就是所谓的架构属性，也就是大家常说的质量属性或非功能属性。这些属性，其实就是在描述系统应该具备的一些“超能力”，比如能跑得快（高性能）、能灵活扩大或缩小规模（可扩展性和伸缩性）、碰到问题不崩溃（弹性和容错性）、好测好修（可测试性和可维护性）等等。设计架构的时候，我们的一个核心任务就是要确保系统能够满足这些架构属性的要求，因为它们直接关系到系统能否在野蛮生长的环境中存活下来。

但这里头有个问题，那就是架构属性多得跟星星一样数不清，我们必须明白，在不同的项目和场景下，重点关注哪一部分属性。这就要求架构师得根据具体的问题域和上下文环境，来做出精准的分析和选择。比如，在一个对性能要求极高的金融交易系统中，高性能和容错性可能就是你得重点关注的属性；而在一个云服务产品中，可扩展性和伸缩性可能更为关键。

同时，架构设计不是一帆风顺的，这些架构属性(如图 1-9所示)之间很可能存在着某种程度的冲突。例如，追求极致的性能可能会牺牲一定的可维护性和可测试性。这种时候，架构师就得像是在玩一个平衡游戏，既要保证系统的核心能力，又要尽量避免负面影响，这就需要他们运用自己的经验和智慧，做出恰当的权衡和决策。

架构设计其实就是一个不断权衡和选择的过程，旨在打造一个既强大又平衡的系统。这个过程中，架构师的角色就像是一位艺术家，既要有科学的严谨性，又要有艺术的创造力。



3.4、架构决策

当我们深入探讨软件架构设计的精髓时，架构决策这个概念浮现为其核心。它不仅仅是关于解决方案的选择，更是一系列必须遵守的规则的体现。这些规则或决策，正如导航灯指引船只航向，为整个系统设计指明方向。在这个过程中，要认识到**并非所有决策都能被称为架构决策**。只有那些对系统有着重大且深远影响的决策，才配得上这个称号。比如说，决定采用何种架构风格，这不仅会深刻影响到系统的整体设计，一旦设定，更改的代价极为昂贵，因此，它无疑属于架构决策的范畴。

架构决策的范围广泛，内容丰富，它们包括但不限于以下几个关键领域：

- 直接对**架构属性中的优先级高的问题**产生影响：这意味着任何影响性能、可扩展性、安全性等关键架构属性的决策都是至关重要的。
- **修改对外接口**：这类决策需要进行周全的影响分析，因为它们可能会对系统与外部世界的交互方式产生根本性的改变。
- **引入或移除依赖**：依赖关系的变化标志着系统功能的增减，对架构的影响深远，需要慎重考虑。
- **改变系统的通用结构**：这关乎到系统的基础架构布局，任何调整都是对系统架构的重大干预。
- **促使开发团队改变开发模式**：这类决策往往涉及到开发流程和文化，对团队的工作方式和产品的质量都会产生深远的影响。
- **承担战略性技术债务**：有时为了项目的长期发展，需要做出一些短期内看似不利但长远来看有益的技术选择，这些决策对系统的未来发展至关重要。

这些架构决策，远不止是技术层面的选择，它们背后蕴含的是对项目未来方向的深思熟虑。一个正确的架构决策可以为项目带来顺利的发展，而错误的选择可能导致项目陷入重构甚至失败的困境。因此，在架构设计过程中，架构师需要具备前瞻性和深度的思考能力，确保每一个决策都能促进项目向着正确的方向发展。

3.5、设计原则

设计原则和架构决策，这两者在软件设计的世界里，就像是指南和指导的关系。两者都指引着你前行的方向，但具体的用法和意义，却大有不同。核心区别在这儿：**设计原则，它更像是一位经验丰富的前辈，给你提供的那些指导和建议，并不会强迫你一定要这么做。而架构决策，那就相当于设计的规章制度，一旦确定下来，你就必须得遵守。**

举个例子来说得更明白一点，设计原则可能会这样建议你：在软件系统间的通讯上，尽可能利用异步消息机制。这么做的好处是啥呢？首先，它能够提升系统的性能，让你的应用跑得更快；其次，还能降低系统间的耦合度，让系统之间的关系更加松散，互不干扰。这就好比是在教你如何在繁忙的城市中开车，不仅能高效地从A点到B点，还能在路上享受风景，避免不必要的麻烦。

但这都是建议，它不会对你说：“你必须得这么干！”它留给你足够的空间去探索和实验，找到最适合你自己项目的方法。而一旦我们谈到架构决策，比如你决定用微服务架构而非单体架构，这就像是签了一个契约，需要你严格遵守，因为它关系到整个项目的基础和未来的可维护性。

在设计和构建软件系统的时候，理解设计原则与架构决策之间的这种微妙差异，就显得尤为重要。它们一个给你灵感和自由，一个给你方向和规则，正确地运用它们，就能让你的软件设计既灵活又稳健。

4、总结

在深入探讨软件架构的定义时，我们会发现行业内各位大师提供了多样化且各具特色的解读，它们就像是不同风格的画派，每一种都有其独到之处：

- **IEEE**的定义，就好比是经典主义画派，追求结构的严谨和形式的规范，给我们提供了一个架构定义的标准框架，像是在告诉我们，建筑的每一砖每一瓦都要符合建筑学的原则。
- **Martin Fowler**的角度，则像是强调了画作中的构图决策，就如同在绘画中选择主题和色彩的重要性一样，他告诉我们在架构设计中，关键的决策是构筑整个系统的基石。
- **Ralph Johnson**的定义更像是抽象艺术，他没有给出具体的形式，而是强调了“重要性”这个核心元素，就像是抽象画中强调情感和概念的表达，让我们理解到架构的核心在于其重要的价值和作用。
- **Neil Ford**的看法则更接近于现实主义，他通过具体化的描述，就像是通过细腻的画面描绘出生活的样态，让我们能够清晰地理解和操作架构的具体元素。

在这些定义中，我个人特别偏爱**Ralph Johnson**的抽象化定义，它简洁而深刻，像是用一笔带过却又点出架构本质。在我的工作实践中，这种将复杂问题抽象化的能力，成为了我判断和理解架构边界时的重要准则。就像是在画画时，虽然不被具体形式所束缚，但能够深刻把握作品的灵魂，这样的定义不仅指导了我的架构设计思路，也帮助我在面对复杂系统时，能够更加聚焦于那些真正重要的事情。

标签： 架构 设计 话题： 金石计划征文活动

本文收录于以下专栏

架构师之道：程序员修炼法典 专栏目录

专栏"架构师之道：程序员修炼法典"旨在为那些渴望从程序员角色转变为架构师的职...

62 订阅 · 20 篇文章

订阅

上一篇 架构师之道：架构演变史：从建筑学...

下一篇 架构师之道：具备什么样的技能才能...

评论 1

登录 / 注册 即可发布评论！

最热 | 最新

Matthew_zou

666

9月前 点赞 评论

为你推荐

架构之道：物理架构到底是什么啊

MobotStone | 9月前 | 8.2k | 9 | 2 架构 设计

知识沉淀一：架构师是做什么？解决了什么问题

MobotStone | 2年前 | 714 | 6 | 评论 架构

软件架构（一）：什么嗯啊的叫软件架构?

寒草 | 2年前 | 1.1k | 14 | 评论 前端 JavaSc... 架构

想成为出色的架构师？那这本架构师修炼之道文档看了吗?

南白 | 2年前 | 119 | 点赞 | 评论 Java

在首席架构师眼里，架构的本质到底是什么?

本人秃顶程序员 | 5年前 | 698 | 2 | 评论 后端

需求分析之道——需求分析要做什么（C系架构设计法，sishuok）

云飞龙行 | 1年前 | 73 | 1 | 评论 架构

架构到底是什么?

架构精进之路 | 3年前 | 720 | 4 | 评论 后端 架构

微服务到底改变了什么，你知道吗?

IT老兵哥 | 5年前 | 407 | 1 | 评论 微服务

架构入门和书单推荐

架构师自习室 | 3年前 | 2.8k | 4 | 1 后端

怕入错行？这群技术人写了本"择业指南"

阿里云云栖号 | 4年前 | 555 | 点赞 | 评论 大数据

【学习总结】04 | 项目大了人员多了，架构怎么设计更合理?

iHTCboy | 5年前 | 959 | 3 | 评论 架构

到底什么是架构？架构设计有这篇文章就够了

samdyli的博客 | 4年前 | 2.0k | 3 | 1 架构

前端架构岗面试技巧

文艺码农天文 | 6月前 | 16k | 204 | 19 前端 面试

架构师常说的"技术架构"是指什么?

Blacktea | 4年前 | 11k | 32 | 7 架构

《架构整洁之道》 架构笔记

社长大人 | 4年前 | 3.7k | 1 | 2 架构