什么才是架构师的真内核？

宝惜 阿里云开发者 2024年11月15日 08:30 浙江

阿里妹导读

本文旨在帮助大家深入理解技术、架构和团队领导力的本质，从而获得持续成长的方法。欢迎在文末留言，你觉得架构师需要具备的核心能力是什么？

技术架构师是在技术领域扮演着关键角色的专业人员。他们在业务需求分析、项目实施、技术架构治理等多个环节中发挥着重要的作用。

技术架构师不仅需要具备高超的专业技能，还需要具备良好的系统思维和认知心态。他们要能在宏观层面上进行技术架构的规划和治理，同时也要在微观层面上带领团队进行业务项目的交付实施。技术架构师是技术人从最初的研发编码，到成长为技术团队的核心骨干、技术主管、高阶技术主管，甚至是技术 CTO 的关键一步，如图 1–1 所示。

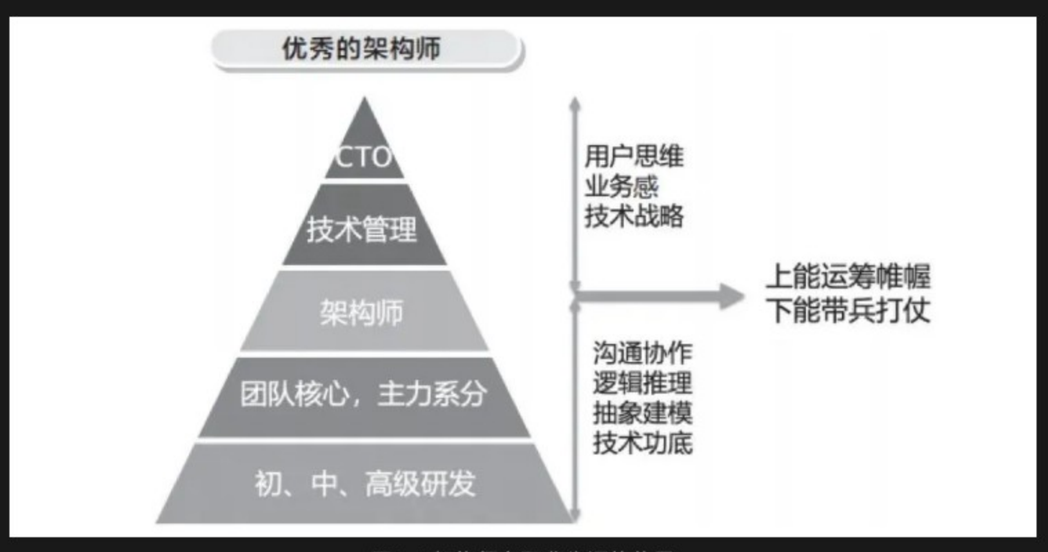


图1–1 架构师在职业生涯的位置

架构师的生态位置

架构师的角色既像是踏着五彩祥云的救世英雄，又常常成为各种问题的挡箭牌。他们经常被借用来赞成或反对某些事项的推进：

- 业务需求不仅数量庞大，而且频繁变化。技术人员即使加班加点，也常常被抱怨效率低下、问题繁多。
- 产品经理在进行产品分析和设计时，经常需要技术人员参与评估，甚至要一行行检查代码以了解系统现状，但他们还会抱怨系统能力不足，无法提供必要的数据支持。
- 技术人员在项目实施过程中，最常遇到的是系统边界问题，最常抱怨的是架构不够先进，导致对业务需求的支持缓慢，生产环境中的应急事件频发，缺乏时间和空间进行深入思考和个人成长，从而感到技术成就感较低。

这些现象反映出对技术架构师在软件研发过程中的角色定位和价值贡献认识不清晰。在探讨架构师的系统思维之前，我们有必要明确架构师在生态系统中的位置，以及他们的权责。这些因素决定了架构师应具备的能力，并对他们提出了独特的要求。

架构在产研中的位置

在大多数产品研发过程中，通常遵循一种瀑布式协作模式，如图 1–2 所示。在这个过程中，业务人员首先了解需求，然后由产品经理进行收集和分析，最后传达给技术人员实施。

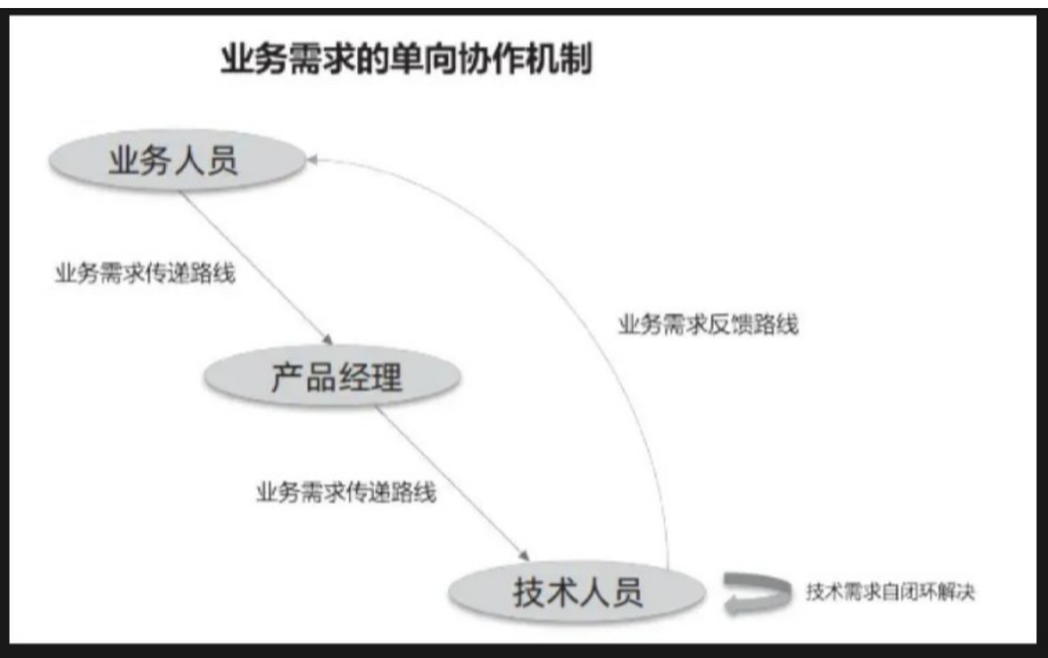


图1–2 瀑布式协作模式

这种模式的问题在于，它使得上游角色更加面向未来，而下游的技术人员则更多地依赖历史经验，处于信息劣势的地位。这种信息的不对等很容易导致上游角色在制定方向和目标时，较少考虑实施路径的可行性。因此，当研发的产品出现问题时，人们往往倾向于归咎于架构的不灵活或缺乏前瞻性。

当技术架构团队竭尽全力弥补信息劣势，提出一个相对可靠的架构方案，并能够识别出对未来需求复用有影响的改造点时，他们通常会与业务人员、产品经理一样，主动地自我合理化地认为，这样的架构优化一定会影响项目的上线时间，因此倾向于先实施临时方案，而不是进行架构优化。然而，当项目上线后出现问题，技术架构团队再次主动排查并提出清理和解决历史架构负债的方案时，业务人员和产品经理往往会指责说：这就是一个技术架构问题。

业产技是三角关系

为了改变瀑布式协作模式带来的信息不对称问题，可以采用业务人员、产品经理和技术人员之间的三角关系协作模式。这种模式强调三个角色之间的直接沟通和协作，以确保信息的及时传递和共享。如图 1–3 所示。

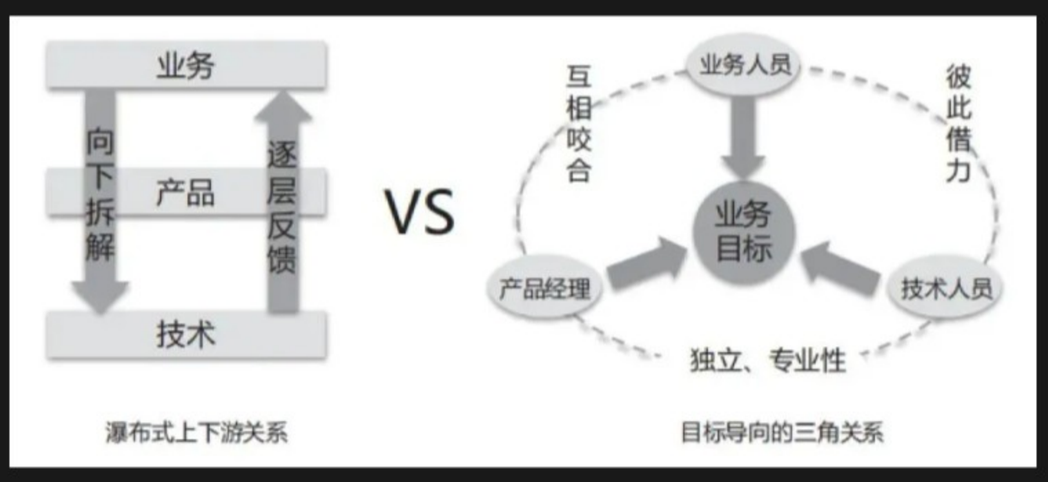


图1–3 三角关系协作模式

在面对不确定性的市场环境时，决策需要考虑多个维度，包括清晰的商业模式、客户导向的产品设计，以及长远前瞻的技术架构思考。这种多维度思考决策要求团队成员不仅要专业，还要能够独立思考并输出意见和判断。

在三角关系协作模式中，业务人员、产品经理和技术人员各自围绕业务目标发挥自己的专业价值。产品经理专注于设计用户体验，技术人员负责论证技术架构方案和可能的风险，业务人员则考虑市场策略。这种模式适合互联网产品的研发，因为它鼓励每个角色在自己的专业领域内发挥最大的作用，同时保持对业务目标的共同关注。

然而，团队成员在未做好自己本职工作之前，应避免盲目补位或越位。技术人员的首要任务是确保产品研发的质量，而不是过度介入产品设计。产品经理应专注于提升产品体验，而不是参与销售谈判。这种混乱的生产关系可能导致团队效率低下和目标不一致。

业务人员、产品经理、技术人员三方保持独立性与专业性，并不意味着各自为战。相反，他们应该围绕业务目标互相咬合，形成合力。产品经理根据业务目标设计产品体验，技术人员根据业务目标设计系统架构，业务人员根据业务目标考虑市场策略。这种互相咬合的关系需要取舍和共识。例如，如果业务人员认为需要迅速抢占市场，而产品经理和技术人员认为不应推出不完美的产品，那么就需要通过沟通和协商来达成共识。

三角关系协作模式能够发挥出“1+1+1>3”的效果，因为每个角色都能为其他角色提供支持和补充，彼此借力。技术人员可以通过数据分析提前发现客户体验问题，产品经理可以给技术人员提供前瞻性的信息，业务人员可以提供真实客户案例以完善产品设计。这种有效的补位是在每个角色做好本职工作的前提下进行的。

在这种三角关系设计下，技术架构师能够更好地找到自己的位置，感受到业务目标的压力，并更加积极主动地为业务成功贡献自己的力量。三角关系协作模式有助于确保技术架构师不再处于信息链路的末端，而是成为推动业务成功的关键角色。

架构师的系统思维

我们根据常见的结构化思考方法总结了一套架构设计思考法，旨在帮助技术架构师从多个视角和维度建立全面的系统思维，如图 1–4 所示。

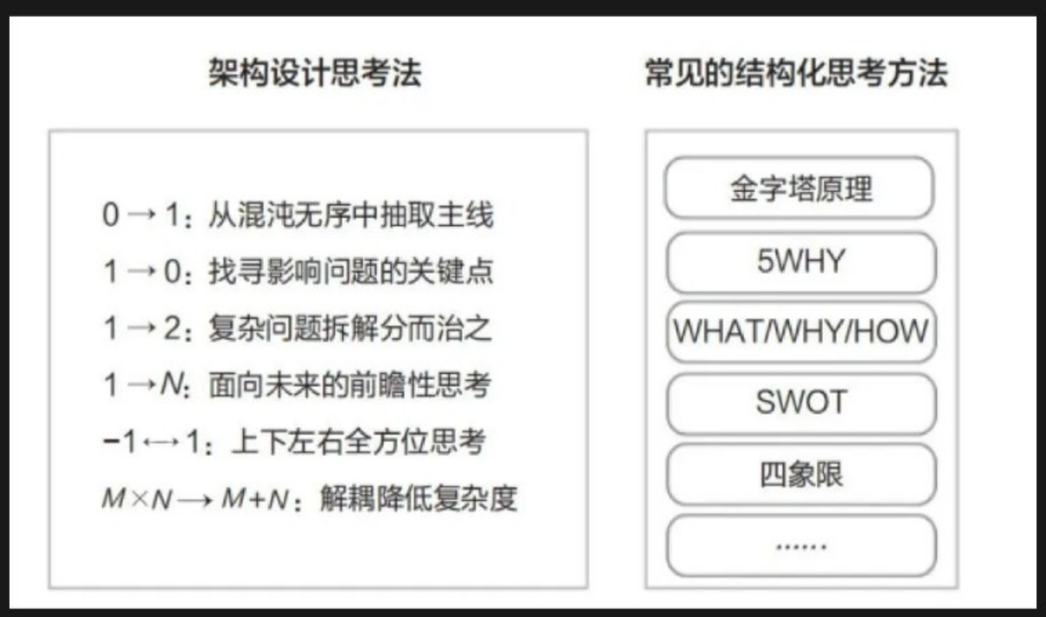


图1–4 架构设计思考法

0→1：从混沌无序中抽取主线

这个架构设计思考法的核心意义在于：在面对一系列困惑和混乱时，首先应紧密关注问题本身，还原客观事实，并迅速构建一个初步版本，以此作为启动认知和迭代优化的讨论基础。随后，围绕这个初步版本逐步叠加和丰富其他维度的内容，直至形成一个完整的架构设计方案。

例如，在讨论某系统能力升级方案时，有些人倾向于使用 PPT 绘制模块概念图，并辅以抽象词汇进行讲解，这往往让人感到困惑。单纯通过概念推导概念的方式难以准确传达意图。为此，可以采取以下策略。

（1）用户视角的客观世界还原：首先，应该从用户视角出发，通过讲述用户故事，使用交互流程和真实数据来描述问题。这种方法可以帮助团队快速达成共识，并清晰地理解客观事实。这个“1”将成为后续讨论的核心对象，可以通过交互时序图或 Excel 表格来直观展示，而不是使用复杂的概念模块图。

（2）客观信息的结构化整合与提炼：仅仅确定“1”是不够的，还需要对它进行结构化的整合和提炼。通过结构化处理，信息才能转化为有意义的知识，并与以往的经验相结合，从而进行初步的架构设计。例如，在处理数据流时，可以识别出哪些是可合并的同类项，哪些需要进行平衡校验逻辑。

（3）加入多元视角的检验与抽象：经过第（2）步的处理，使“1”变得更加完整。接下来，需要加入多元视角的抽象和校验，例如考虑重要异常、投资回报率、合理性、扩展性等因素，以形成一个完整的可实施主线。通过这种方法，在讨论方案时，使用交互序列图或表格来聚焦讨论，可以帮助团队从混乱中快速提炼出关键点，并形成一个清晰、可行的架构设计主线。

1→0：找寻影响问题的关键点

架构设计思考法的第二条建议是，在面对众多因素而难以抓住关键点时，可以采用删除法（即考虑去掉某个因素是否可行），以识别出真正的关键点。

例如，技术团队每年进行技术规划时常常感到痛苦，这是因为脑海中有无数需求、有无数的待优化点，还有无数的想法在萦绕，每个点都似乎值得在新的一年里进行突破。然而，如果没有有效的方法来抓住关键点，最终的结果可能仅仅是一个简单的表格。

为了应对这一问题，可以采取以下几种方法。

因果判断法：当需要透过现象探究真实原因，以识别影响事情发展的关键点时，推荐使用因果判断法进行检验：这个点到底是“因”，还是“果”。例如，某系统近期研发效能低下，就是“果”。要找到造成“果”的“因”：比如，系统架构太复杂，使得实现任何业务需求，都要修改大量代码。值得注意的是，因果关系往往是嵌套的，系统架构太复杂本身也可以视为“果”，而它的“因”可能是系统模型抽象层次不清晰。

树干树枝法：有时候，各个因素之间并非单纯的因果关系，而是依附关系，如同树枝依附于树干。为找到关键点，可以尝试使用树干树枝法进行检验：如果去除这个点，是否会影响到整体，是否会导致结论不成立。通过多轮分析，绘制出树干与树枝的关系，树干就是要找的关键点。例如，在上述研发效能低下的问题中，“因”还有很多，如系统架构的复杂性、研发流程的冗长、人员技能熟悉度低、业务需求分析不到位等。每个因素都可能成为树干级关键点。在项目组组建的初期阶段，可以将人员技能熟悉度低视为树干级因素。人员技能熟悉度低可能导致业务需求分析不充分，对研发流程不熟悉，以及对系统架构缺乏了解。

支点撬动法：有时，各个因素之间的关联关系较弱，难以通过上述方法确定关键点。此时，支点撬动法就可以发挥作用，即寻找能够激发这一系列因素的关键要素。举一个例子，在 2024 年的《政府工作报告》 中提出，“充分发挥创新主导作用，以科技创新推动产业创新，加快推进新型工业化，提高全要素生产率，不断塑造发展新动能新优势，促进社会生产力实现新的跃升。”在这个场景中，科技创新便是撬动现代化产业建设的关键要素。

以上是目前实践中总结的一些抓取关键点的方法。但在此过程中，我们一定要注意粒度问题，避免走极端。例如，一提到关键点，就立刻深入本质；一触及本质，就急于找根因；一找根因，就深挖到人性层面，最终将所有问题归咎于人性原罪。这种做法既缺乏实际价值，也无助于问题的有效解决。

1→2：复杂问题拆解分而治之

架构设计思考法的第三条建议是，面对抽象问题时，应该采用拆解的方法（一分为二），通过分而治之的策略来确定每个小问题的边界。通过解决这些小问题，可以降低全局思考的难度，从而更快地形成解决方案。在处理复杂问题时，可以采取以下两种典型的技术架构动作。

纵深拆解：拆解是一种有效的分而治之的方法，但它不仅仅是简单的物理分割，而应该是有逻辑的、有机的拆解。例如，将生产环境故障指标直接分配给每个团队成员是无效的。相反，应该拆解为建设如故障发现、故障应急恢复等能力，然后对这些能力进行技术架构设计，以实现最终目标。

横向解剖：在讨论业务需求时，经常会遇到僵局，比如产品经理认为这是技术架构问题，技术架构师认为这是业务需求问题，而业务人员则认为这是产品设计问题。要打破这种僵局，就需要对问题进行一层层的解剖，清晰地区分业务需求问题、产品设计问题和技术架构方案。每一层都应该向上游屏蔽下游的细节，这样才能清楚地定义问题。通常，从参与角色的角度进行解剖更容易获得全面和深入的理解。

1 →N：面向未来的前瞻性思考

在进行技术架构设计时，前瞻性至关重要，避免被业务需求牵着走。在考虑技术方案时，不仅要关注当前条件，还要预见到系统从 1 到 N 过程中可能遇到的挑战。

前瞻性的核心在于对时间的考量：从长远角度出发，预测关键生产资料可能的变化，并据此进行架构设计。这些关键生产资料如下所述。

业务场景：作为系统演进的驱动力，包括市场份额、客户价值、竞争对手动态等。

团队组织：人是系统的创造者和推动者，若不能发挥人的积极性，系统就无法支持业务的快速发展。

技术架构：本身是一种极其重要的生产资料，这一点却常常被许多人忽视。例如，在设计不合理的代码中，我们经常看到同一个语义的常量在多个不同的地方被重复定义，这无疑增加了维护的难度和成本。相反，通过简单的技术架构优化，如定义一个静态常量，就能轻松解决这一问题。

针对这些生产资料，**以下是从 1 到N 扩展的几个技巧。**

（1）架构考虑所有可能性，但做有限且明确的实施。

在处理充满不确定性的业务场景时，技术架构师面临的一个挑战是如何在缺乏详细信息的情况下提供专业的意见和评估。在这种情况下，技术架构师应该基于自己的经验和对业务变化的理解，罗列所有可能的情况，并为每种情况提供相应的架构方案和评估。

例如，技术架构师可以提出：“基于××业务的假设，系统架构需要××量级的工作量，进行××样的能力迭代升级，可以实现××业务效果和价值，但需要进一步××业务输入。”这样的做法可以帮助业务人员更清楚地理解不同选择的影响，并做出更明智的决策。

然而，在实施技术架构时，并不需要实现所有可能性。技术架构设计可以很宏大，但实施应该从小处着手。对于不确定的部分，应该做好扩展设计，以便在未来有需要时可以轻松添加新功能。对于实在无法预测的部分，应该明确拒绝（宁愿不做也不错做），以避免留下潜在的风险和隐患。

（2）没有靠谱的人，只有靠谱的机器。

在生产环境故障复盘会议中，技术人员的积极性和责任感是值得肯定的，但过度依赖个人审核并不利于系统的长期稳定和发展。当技术人员提出“以后××类变更都加上我来审核一个环节，我确认没问题再往后走”的建议时，虽然体现了个人担当，但这种做法并不推荐。

这种做法可能导致系统变得脆弱，因为它过度依赖个别人员的判断和决策。在技术方案设计时，我们应该追求自动化和系统化的解决方案。能够由系统自动执行的任务，就不应该依赖人工流程来保障。能够通过领域模型进行校验的问题，就不应该依赖旁路系统的侧面印证。

（3）提前思考“幸福”的烦恼。

许多技术人员渴望参与高并发大流量系统的开发，但在实际编码过程中，往往采取简单直接的方式，忽视了未来可能面临的大流量和高并发问题，以及对资损风险的考量。他们可能会辩称，当前的业务量尚未达到需要考虑这些问题的水平，或者认为这类事件发生的概率极低，因此在一期工程中不必过度关注。

然而，要构建能够应对未来挑战的技术架构，必须从第一行代码开始就进行深思熟虑。这意味着在编码过程中，技术人员应该提前考虑高并发、大流量，以及系统的严谨性。

通常，技术人员更倾向于享受从 0 到 1 的创新过程，而在互联网快速迭代的环境中，从 1 到N 的扩展过程往往被压缩。因此，在从 0 到 1 的阶段就加入对从1 到 N 架构的预判至关重要。很多时候，系统的结构性问题在最初的设计阶段就已经确定，后期很难更改。如果要修改，只有一个机

会，那就是在设计之初。

🟧 -1 ←→ 1：上下左右全方位思考

在考虑技术方案时，不要一条道走到黑，应采取前后、上下、左右、正反全方位的思考方式，以确保方案具有全面性和多维视角。以下是一些实践中常用的思考方法。

正反思考法：在日常的架构设计、系统分析和测试文档编写中，对于正常业务需求功能的描述通常没有太大问题，只需遵循既定步骤进行撰写即可。然而，普遍存在的一个现象是对问题的反面论述不足。例如，文档中可能会详细描述支付的正常流程，但对于退款或拒付等反向流程的描述却往往不够细致。业务功能的正常流转可能被充分论述，而异常场景的讨论却常常被草草带过。然而，只有将正面和反面结合起来，才能构成一个完整的体系，对反面的思考实际上是对正面的有效补充。通常情况下，虽然正面情况出现的概率较高，但消除反面情况中可能出现错误的影响所需投入的精力，往往远超过正面情况带来的收益。

极限思考法：在评估技术架构风险时，应深入思考最坏情况下可能对业务造成的影响。通过极限设问，可以激发团队考虑最极端的情况，并准备相应的风险应对措施。这样，即使在面对最坏情况时，也能有足够的准备和应对策略，从而更加乐观地进行方案设计。

对称思考法：在审查代码或逻辑结构时，应从整体上检查其逻辑结构的完整性、对称性和美观性。例如，使用 if 语句时，应确保有相应的 else 语句，以覆盖所有可能的场景。对称思考法不仅有助于提高代码的可读性和美观性，还能强制性地提醒开发者考虑逻辑的对立面，从而减少因考虑不周而导致的错误。对称的美是一种生产力，因为美的事物往往简洁并能够直接触及本质。同样，技术人在编写程序时追求的也是逻辑清晰、简洁，并直达业务本质。通常，逻辑结构清晰的程序基本上没有大问题。而那些逻辑不清晰的程序（例如，变量命名随意，方法缺乏语义）往往都有 Bug。

🟧 M×N → M+N：解耦降低复杂度

在技术架构设计过程中，从系统耦合的角度出发，寻找解耦的突破口是一种有效的方法。例如，高速公路网的连接并不是在所有目的地之间都修建一条高速公路，而是通过建立主干道和分支道路来实现，这种方式降低了系统的耦合度，从 M×N 的复杂度降低到了 M+N。

技术架构解耦可从以下两个方向进行。

解耦上下游关联性：在业务和技术架构发展的早期阶段，为了快速解决问题，通常将多个模块混合在一起，例如电商网站最初可能将会员模块、交易模块、库存模块等杂糅在一个系统中实现。然而，随着业务的发展和架构的演进，对这些模块进行解耦是必然的。解耦的目的在于重新定义各个模块的边界，平衡新业务发展要求下各方发展速度的差异，并通过解耦实现各自的快速发展。

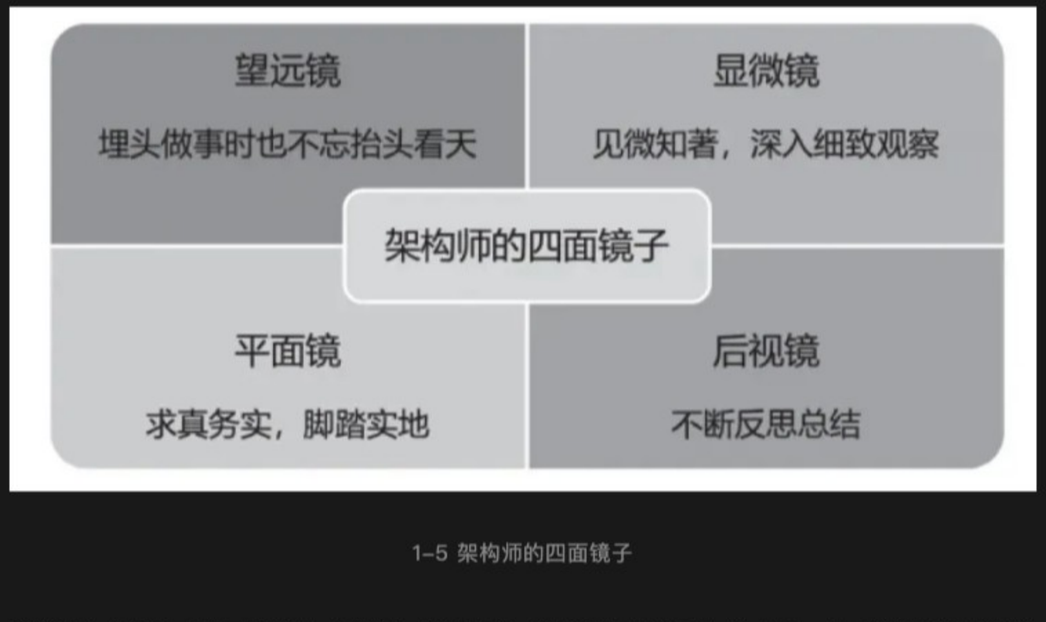
在服务化的分布式架构中，解耦是非常常见的做法。几乎所有的跨域系统架构升级都涉及解耦。例如，电商网站后续发展大多会进行领域服务的拆分，将会员模块独立为会员系统，交易模块独立为交易系统等。

解耦各个角色的依赖：在领域模型抽象之前，需要注意到解耦各个角色之间的依赖关系。技术架构师在信息不足的情况下，应基于经验做出假设，并将技术架构设计与商业选择、产品设计等解耦开来。通过这种解耦，不同角色可以基于服务水平协议（SLA）进行交互，并基于自身的专业知识为对方提供更多的选项和可能性。这种解耦有助于提高系统的前瞻性和竞争力。

在几乎所有大型系统架构的升级中，都可以看到解耦思考法的应用。因此，当缺乏设计思路时，建议从解耦的角度审视和思考，可能会带来新的发现和收获。

架构思维胜在无招

架构设计思考法为技术架构师提供了一种理解和应用系统思维的途径，但它并不是架构思维的唯一或最终解决方案。架构思维是需要不断打磨和提升的，对于架构师而言，建立系统化的思维模式并不是依靠一成不变的方法，而是为了在不同的时间、空间背景下，通过广泛的经验积累和实践训练，灵活地提出适合当前情境的架构方案。



1–5 架构师的四面镜子

希望以上的分享内容可以帮助大家深入理解技术、架构和团队领导力的本质，从而获得持续成长的方法。

技术硬实力是技术人的立身之本，技术架构力让技术人能够脱颖而出，而技术领导力则使技术人能够协同作战，取得更大的成功。

欢迎一起聊聊，你觉得架构师需要具备的核心能力是什么？

#11月技术文章大赏 22

11月技术文章大赏 · 目录 ≡

‹ 上一篇
再也不用心惊胆战地使用FastJSON了——序列化篇

下一篇 ›
11月技术文章大赏：选出你的最爱

阅读原文