12 | 法则五：如何提升一个架构设计的外部适应性？

2022-01-11 郭东白

《郭东白的架构课》 课程介绍 >

讲述：郭东白

时长 27:03 大小 24.78M

你好，我是郭东白。

上节课我们讲了外部适应性这个概念，也强调了架构师的职责是通过架构活动为企业不断注入外部适应性，从而帮助企业更好地实现它的战略意图。

那么该怎么注入呢？

上节课在讲影响技术体系外部适应性的因素这部分，我们提到了挑战主要来自三个方面：企业的内部压力、企业的外部环境和企业的组织结构。这三方面的挑战，其实已经为我们指引了应对的思路。那么今天这节课，我们就来谈谈应对挑战的解决办法。

如何抵抗内部压力而为企业注入外部适应性？

企业内部压力的挑战，也是有三个，分别是业务交付时间的压力、技术岗供给压力导致技术人员的稳定性差和设计短视的问题和考核带来的短期效应。

高速响应业务与技术的需求

交付压力其实就是**在一个高速迭代的业务中维持技术体系的外部适应性？**

虽然大多数高科技公司都会在技术上做比较充足的人员投入，但业务和产品职能对技术的首要要求往往是在**需求实现的时效**上。这个态度非常合理，就像上节课提到的，商业机会稍纵即逝，所以高速响应机会是每个职能的本能。

所以我们真正要解决的挑战应该是，**技术人员如何在高速响应业务和技术需求的同时，还能够保障技术体系的外部适应性？**

正如我们在法则一所讲的，必须依据目标做出正确的取舍。为什么要做取舍呢？根据我的经验，绝大多数的业务和产品需求只是一个小尝试，而并非战略性投入。对于这类业务尝试，响应的速度是第一优先级，只要不破坏外部适应性即可。

不过这里面还有一些挑战。 首先，但是我们大都处在一个博弈的状态下：**业务、产品和技术人员的智能内部和职能之间都都在互相博弈：**也就是没有任何一个业务、产品和技术人员愿意主动承认自己的需求是个小尝试。大多数人都试图找出充足的理由去说服其他人把自己的需求作为最重要的工作。

如果你有比较深度的业务理解，其实不难对业务尝试和战略投入做出区分，你甚至可以判断一个需求的商业价值和成功概率究竟有多大。你也可以与高层不断沟通。确认自己对业务尝试和战略投入的判断。在一个更高层级决策者的心中，他对两者应该是有清晰区分的。

对于业务尝试，我们需要以最低成本完成这么一任务：**从这个尝试中得出一个正确的结论。**这种情况下你要做的就是**在保障尝试结论有效性的前提下，最小化这个尝试对系统的冲击，**因为大多数尝试是不成功的。

什么是结论的有效性呢？意思是说，如果把一个小范围的尝试放到更大的范围内，它的价值依然存在，方法依然有效，那就说明结论是有效的。它与产品的完整性，技术可扩展

性、安全性、可维护性等等几乎毫无关系。也就是说大多数时候，你在实现这类需求的时候可以忽略多数横向问题（Cross-cutting concerns)， 把开发成本降低到极致。

事实上，结论有效性与实验环境有非常大的关系。比如说你圈选了哪些城市、什么目标人群、花了多少营销预算、在什么季节做投放等等。很多做算法的同学都知道，有的团队经常发天花乱坠的战报，还有各种经过严格 A/B 测试后得出转化增长的结论，但是最终这些本来独立的实验却很少有累积效应，甚至随着时间会逐渐衰减。这里当然有竞争、人群等复杂因素，但其中至少有一个原因：**很多人都在试图挑选有利于拿结果的实验环境来做出结论。**因此，哪怕是那些所谓的“成功”的尝试，其实结论有效性也要打个折扣的。

所以，从架构设计的角度来说，我们要把大多数的尝试尽量封装到一个小的领域内。这个时候，多次的业务尝试不会随着时间而导致混乱，削减技术体系的外部适应性。做到这点，下面的这些架构原则非常重要：

第一，单一职责，指的是要把每个业务尝试尽量封装到一个单一模块中。好处是一旦尝试失败，就可以迅速把业务逻辑下线，避免影响整体的复杂性。

我的经验是业务尝试在 90% 的情况下，都是不符合预期的。在这种情况下，符合单一职责的设计很容易下线旧逻辑。这么做很重要，你接受一个烂摊子的时候，肯定恨死那些不下线无效逻辑的前辈们了。打个比方：不下线失效逻辑就是一个不冲厕所的行为！Gitlab 里面清清楚楚的写着：“某年某月某日，张三在此拉下一坨烂代码！”

第二，最小依赖，指的是整体架构设计要保障大多数业务尝试可以在业务层完成。如果每个业务方的需求都会侵入到底层的逻辑，那么每次尝试都会变成跨团队合作，这种架构会大幅降低业务尝试的速度。

第三，最小数据共享。一个正在尝试中的业务应该尽量减少与其他业务模块的数据交换，尤其是输出，这样才能最小化它的爆炸半径。否则该业务尝试的数据模型会污染到其他业务，在尝试失败之后对其他业务的影响也会很难剥离。

第四，最小暴露。指的是在业务尝试期接口不对部门或企业外部暴露，包括 API、数据共享、事件、消息流等一切对外界造成影响的通讯机制。

你可能会说：“我们公司不一样啊，我们的项目都是基于 100% 的科学决策，多数项目都是战略级项目。你这些原则是小步快跑的公司里才会用的原则，对我们不适用。就像我手头这个项目就是 CEO 亲自指定的战略级项目！”

那我给你一个对战略级项目的简单判断原则吧：**战略级项目应该在全公司层面明示，而且要维持足够长的时间。**

所谓足够长，对于阿里腾讯这样的大厂来说，至少三年起。如果你的部门体量小，那至少也要一年起。所以如果你的业务方每月给你一个新战略，那你可以非常确定地告诉自己：这哥们分不太清楚业务尝试和企业战略，他的需求我还是要封装好。

华为创始人任正非有一句话，可以完美总结这个原则：“先开一枪，再打一炮。” 也就是说，我们对业务尝试的技术设计，都应该先开一枪，尽量最小化变更范围（footprint），直到结果让我们笃定到可以把这个项目上升成为企业的战略级项目了，才可以改变做法。顺便说一句，我认为华为的成功与这个原则有非常大的关系。

你如果能准确区分业务尝试和战略投入， 你也学会了上面的对业务尝试的封装方法， 那么对于战略投入， 是不是我也可以用同样的方法呢？

不是的， 你如果真的碰到了三年一遇的战略转型，你的方法就不一样了。

这个时候，你的架构设计就会遇到一个很大的不连续性。这样一来，**你的目标不是保障这次变更的最小爆炸半径，而是让战略转型做得尽量彻底，甩掉尽可能多的老包袱，使得未来的企业变得更灵活**（至于如何做好战略级项目的架构规划，我们会在模块二详细描述）。

分析到这里，你就明白为什么我们特别强调要区分尝试和战略转型项目，因为你的架构原则在这两个场景之下完全不同。对于后者，你的目标是要做彻底而不是做快。其实这也是一个区分尝试和战略项目的一个好办法。 你可以问问业务方：“你是想让我尽快上线呢？还是让我花大量精力准备好一次战略大进攻呢？” 我相信 90% 的情形下，你得到的答案是前者。

也就是说，大多数时候， **保护技术体系长期的外部适应性和实现当下的需求一样重要， 如果不是更重要的话。**

当然我们还提到技术岗供给压力导致技术人员的稳定性差和设计短视的问题。 想减少这种问题，你作为架构师可以：

第一，提升对封装能力重要性的认知。你要帮每个做技术的同学认识到： 我们都要有有效地封装业务尝试的能力， 这个能力最终会转化为你提升技术的外部适应性的能力。这是一个技术人员的基本功，从你写代码的第一天就需要。只不过随着你的职业发展，你从封装代码逻辑，到了封装业务逻辑，最后到了封装业务尝试。 一个程序员在日常工作中忽略这方面的能力训练，其实是个不明智的选择。

第二，建设复杂度控制机制。在这里，设计评审很关键。业务尝试也要有设计评审，而评审的一个固定环节就是逻辑、数据和接口的最小爆炸半径的设计。

第三，推行最小必要架构原则。在公司范围内推行奥卡姆剃刀（Occam’s Razor）原则[1]。 任何增加功能、引入复杂性的设计，都要做一个正式的评审，而简化的行为则不需要。

上面就是我们如何解决交付和压力的挑战的办法。 下面我们就简单聊一下短期效应。

完善技术体系的考核

短期效应这种挑战，它的根因在于公司的激励机制不够合理。这个架构师其实没有真正有效的抓手。 这是制度上的问题， 要靠改变公司的制度来修复。 我见到比较好的办法是把绩效考核仅仅与业务结果绑定，而把技术线晋升与长期的技术体系建设绑定。

阿里在这一点上做得非常不错，尤其是 P8 和 P9 层级的晋升比较看重这一点。我觉得大多数公司都可以借鉴。如果你所在的企业没有做到，那么你可以向 CTO 建议在晋升中加入与技术体系建设相关的考核，也可以建议逐渐增加这个考核维度的占比，比如从 10% 逐渐提升到 30%，甚至是 50%。

业务理解

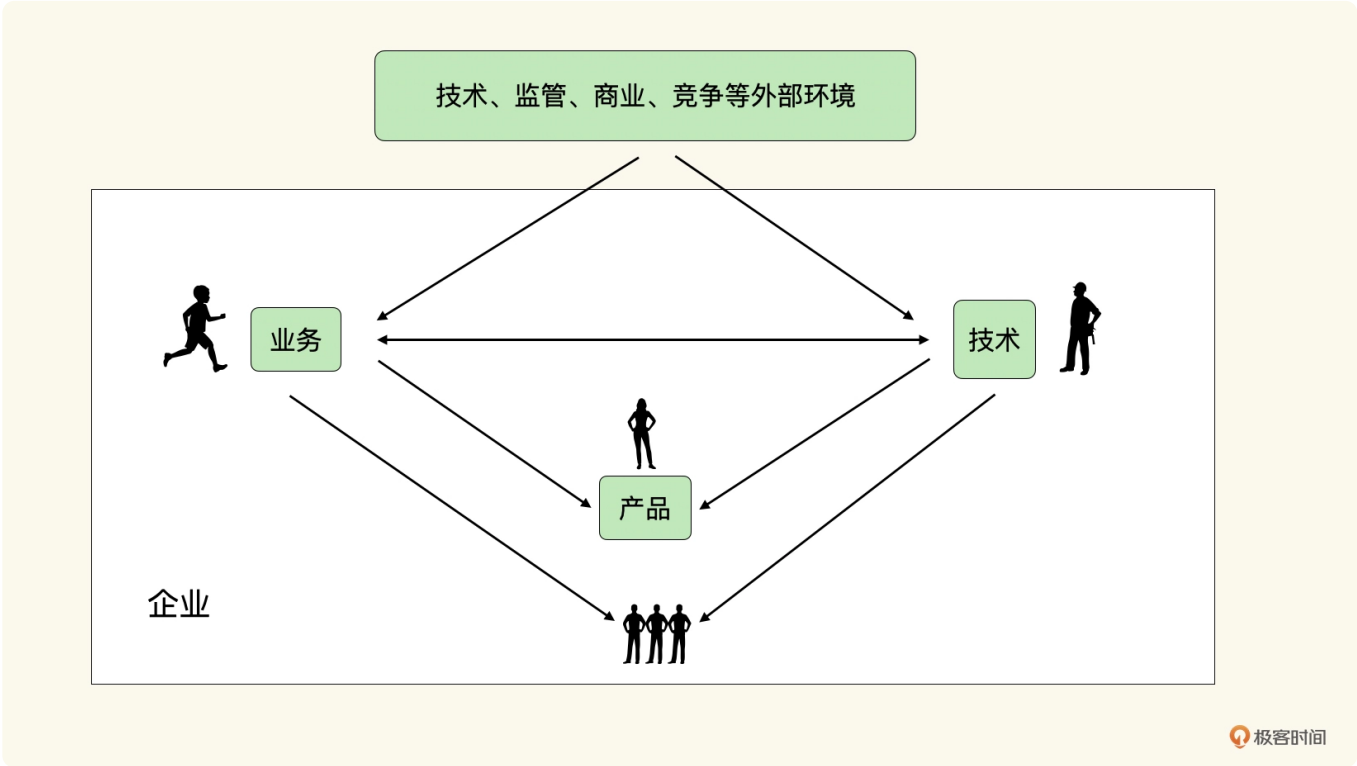
我们刚才提到，你需要比较深的业务理解才能辨别一个需求的战略价值。这和我们上节课提到的一个互联网现象有关：在时间的压力下，不论是业务、产品人员，还是技术人员，都不会在完全看清楚市场后才行动。因此处在最底层的技术体系，在外部适应性上就不太

可能有充足的时间和精力去做到完美。这个时候，就需要技术同学有深度的业务理解，才能保障好技术体系的长期外部适应性。

业务理解这个概念到处都是。我经常看到技术同学晋升时展示自己的业务理解，就是把自己团队的业务讲一遍，或者重复一个业务 leader 的宣讲内容，甚至有些同学的 PPT 还是从业务团队要过来的。这与真正的业务理解有着天壤之别。

什么是真正的业务理解？

我们通过下面这张图来解释一下什么是真正的业务理解：



在一个互联网企业，特别是小公司，业务、产品和技术同学需要一起 ** 去认识行业、市场和竞争等外部环境。这意味着每个职能的认知是同步的，是平行迭代的，是没有偏差的。**

只不过你作为技术人员，要从技术视角去理解业务，并将自己对业务的认知转化成一个技术动作。而这个技术动作，最终会和业务动作、产品动作一起，将企业带到一个更好的生存位置上。这才是真正的业务理解，也是你独立于其他职能所创造的长期价值。

你会发现，这是你通过了解外部环境而获得的第一手的业务认知，而不是你从业务、产品同学那里获得的二手、三手的认知。从某种程度上来说，这是多个职能在做分布式的学

习。在这个过程中，你只是一个技术锤子，在用技术视角去理解业务这个钉子。你不仅要看懂业务人员做业务、产品人员做产品。还要看出门道，最好能提出建议，以及最终预测出的结果。这才是你的业务理解。

所以说，一个架构师要拥有的业务理解的能力，**是基于自己对业务深度认知之后的技术洞察，然后通过这个技术洞察来推动业务发展的能力**。

拥有了这个技术洞察后，你甚至能从业务视角上看到业务机会，或者从产品视角上看到产品机会。再进一步，你的技术洞察才能为业务和产品赋能。最终，三个职能合力推动企业的发展，而不是三个职能单打独斗。这也是图中红线的真正含义。

其实技术视角和动作也不止一种，对架构师而言是架构抽象和架构设计，对数据工程师而言是数据建模和数据资产建设，对算法工程师而言是建模和算法调优。这些技术动作最终都会作用到业务层面上，从而创造出增长和新的业务机会。

案例指引

我们用上节课里提到的物流的例子，来具化一下技术动作。假设你通过架构抽象定义了物流服务供应商这样的概念，那么你对供应商的能力就可以做进一步的抽象和量化。

比如你抽象出了四个维度：

1. 容量：指某供应商对每个类型的服务所能提供的最大服务容量。这是一组与业务场景有关的度量，比如说多个小件包裹 / 天，多少次登门安装 / 天等等。
2. 价格曲线：指该供应商的服务价格随着服务容量变化的关系。
3. 质量曲线：指该供应商的服务质量随着服务容量变化的关系。
4. 最小容量：指与该供应商维持商务关系所要保留最小服务的容量。

有了这些维度的度量和建模，你就可以设计一个调度引擎，使其在多个供应商之间做调拨，同时最大化服务质量、最小化成本。

而有了这个调拨能力，业务的运营也会发生变化。你可以不断量化和监控服务商的真实服务容量、服务质量曲线，同时对价格和质量曲线的预测做出调整。而业务方呢，也可以通过这组监控和对业务未来走势的预测，决定是否要对现有供应商组合做出调整。

一旦有了这些数字，业务方就可以有新的运营方法， 比如说通过最低容量保障去刺激某个供应商扩大自己的服务容量，或者是提升他们的服务质量承诺。业务方甚至可以决定把某些质量监控直接返回给供应商，使得供应商能够主动响应质量分的反馈。

更进一步地，业务方可以开始度量和管理一个供应商的可运营度。也就是说，当业务方用一组规则驱动一个运营商做出容量或质量改变的时候，这个供应商响应这些规则的能力和时间延迟也可以被度量出来， 这些度量就可以转化成一个数字化的可运营度指标， 未来可以用来筛选有潜力的运营商。 而这种数字化的运营方式，就是技术推动业务发展的一个案例。

这时候你可能要挑战我了，你是怎么一下子找到这四个维度呢？它们充分且必要吗？这些维度到了一个新行业还有效呢？

答案很简单，这就像你写代码的时候会做类和接口设计一样，是一个循环迭代的认知提升的过程。你做多了，就会看到一些常见的模式。这也是你作为一个架构师的经验所在。 我也不知道这四个是否充分必要， 跑起来之后我就知道了。

或许你还会问，我作为架构师，难道首要任务不是提升技术实力吗？如果花很多时间来理解业务，那我的取舍是不是错了？长期这么做，会不会我的技术比不上做技术的，业务也比不上做业务的？

是的，一个架构师当然还是要以技术实力取胜。但你要想在架构师这个职业上走得更长远，就必须做好比别人付出更多的努力，下更大的功夫来提升自己对产品的思考，对业务的理解。 这是一个内卷的时代， 你除了需要卷技术还需要卷业务理解， 哈哈。

深度理解竞争环境

第二类挑战具体来说就是：在一个充分竞争的行业，大多数企业并不具备对外部适应性做长期规划的条件。针对这种情况，架构师该如何应对呢？

解法与我们前面讲的跨职能业务理解的过程类似。只不过这里更侧重于**站在业务、运营、产品和技术的视角，不断监控和思考竞争对手，然后用技术手段做出应对方案。**

先举一个利用技术来竞争的正面例子。在电商竞争中，最常见的就是对新买家的争夺。而新买家的常见行为就是比价，以至于有的电商平台会建设专门的新人专区，通过打折的方

式来提升他们对平台价格竞争力的认可度。

那么这里就有很多技术视角的创新机会了。举个例子，你可以设计一个竞争对手的商品价格监控网络，确保自家专区商品的定价低于竞争对手。而这个监控能力又会衍生出大量新的技术机会，比如商家识别、商品识别、商品到 SKU 的建模、商品主数据、同款识别、图文相似性、合手价计算、竞争价格趋势分析等等。有些领域甚至复杂到需要多个博士学位的研发人员，甚至还可以发展衍生出一些对抗能力。

再举一个非正面竞争的例子。你开发风控能力对付费流量和新人做出识别，防止老买家或黑客团伙反复注册来薅羊毛。

这里的风控能力，又可以拓展到多个技术方向，比如通过深度学习来发现新的风控规则，就是现在比较热门的一个创新方向。乍一想风控似乎与竞争无关，但风控就像一个财主家的高墙。如果你的墙就是个木篱笆，一推就倒，那么贼就蜂拥而至。而和你形成竞争的、躲在砖墙后面的老财主，则平安无事。也就是说，打磨风控技术能够提升一个企业的相对竞争力，最终可以帮助提升企业的外部适应性。

这两个例子的应用方向看起来很窄，似乎每个领域都要花不少时间和人力来搭建。从我个人的经验看，这属于比较正常的现象。一个竞争对手的出招，如果是靠一两个成熟的技术范式就能应对，那么这个竞争对手其实也不够高明，估计很快就会被淘汰掉。真正的高手出招之前，必然是经过了深思熟虑，肯定会防止被你一步绝杀。所以说商业竞争是一个持续对抗的过程。

深度理解市场竞争环境还有另外一层含义， 就是帮助你思考技术驱动业务的天花板， 避免过分乐观。为什么呢？ 因为很多商业上的对抗属于消耗战，不可能无限期的打到底。这意味着**很多技术动作带来的业务价值，并不能做大尺度的外延**。也就是说你利用技术招数来挖掘市场竞争中机会是有限的。

我们还是用物流来举个例子帮助你理解。假设有几个物流供应商之间在进行充分的竞争。他们都有自己的冗余运力，都要用更便宜的竞价来获取额外的订单，来扩大自己的规模和市场份额。

在这个假设下，物流调拨系统可以在多个供应商之间肆意挑起并放大价格战，从而最大化你作为甲方的利益。

但事实上，不是任何一个市场都是充分竞争的。供应商仅仅会在一段时间内通过烧钱来争夺市场份额。如果长时间亏钱，供应商也撑不住。

此外，供应商也不是随意受你摆布的。如果你在供应商之间做调拨游戏，而你的竞争对手却采取了完全不同的策略，比如保底单量、资金注入、股权交易，或者与供应商形成商业伙伴关系，以此来换取长期的、有保障的深度合作，那么你的策略在这个供应商身上肯定不奏效。

也就是说，供应商路由这样的技术手段，仅仅在某段时间、某个地区发生充分竞争的时候才有效。 如果天时不对，你发明路由算法时竞争格局已定，那么头部玩家根本不会和你做动态竞价。哪怕做动态竞价，吃亏的也是你。

所以你一定要多和业务同学讨论，尤其是单凭技术视角做大尺度外延的时候，一定要论证有效性。

认知和组织冲突

最后一类挑战就是研发人员的人性与企业的组织结构。前者我们在法则二里已经深入探讨过了，而企业组织如何影响外部适应性这个话题，其实与企业的文化环境有关。这个话题，我们会在接下来的法则六中深入探讨。

小结

这两节课我们讲了外部适应性的注入。作为一个架构师，处在一个主动的位置上做一件很了不起的事情：为企业注入技术基因来帮助它长期生存。

那么你是怎么做到呢？

首先，你要有意识主动去思考如何注入外部适应性这件事情。

这个思考会引导你做有清晰业务价值的技术创新，从而为一个企业创造出经得起时间考验的架构设计，提供更持久的外部适应性。 而能做好这一点首先要理解业务，其次才能找到价值点。 当然你需要有深厚的技术实力，你才能最大化价值挖掘。

不过，因为竞争和市场的作用，哪怕是逻辑上完美的尝试，最终往往仅在短时间内有效。也就是说你对一个行业的理解是必须持续提升，而你的创新也不能停止。

注入外部适应性最终是个循环往复的认知迭代的过程，也是一个试错的过程。靠尝试、靠创新，而不是靠祖传的招数。因为你的每一步尝试都在消耗企业的现金、时间成本、人力成本和心力，这就回到了法则三提到的最大化 ROI 的决策过程。所以你看，这些法则不是孤立的，而是相互勾连的。

未来我会在第二个模块，分享一些具体的招数。

思考题

1. 你所在的企业或者是行业里面，由技术带来的外部适应的场景是什么？请你稍微解释一下你所在的行业，具体的场景，技术手段和带来的价值，以及维持这个价值创造的方法。
2. 你有没有见到过一个失败案例，就像我们文章中解释的物流供应商逻辑一样，虽然从技术创新逻辑上非常清晰，但是某些假设却商业环境的动态变化中失效了。 为什么会是这样呢？
3. 在小范围尝试上，也就是“先放一枪、再开一炮”的方法，你有成功案例可以分享呢？你在技术上具体做什么事情来提升整个事情的成功概率？

上一篇 11 | 法则五：架构师为什么要关注技术体系的外部适应性？

下一篇 13 | 法则六：如何鉴别文化环境是否有利于架构师的生存？

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。