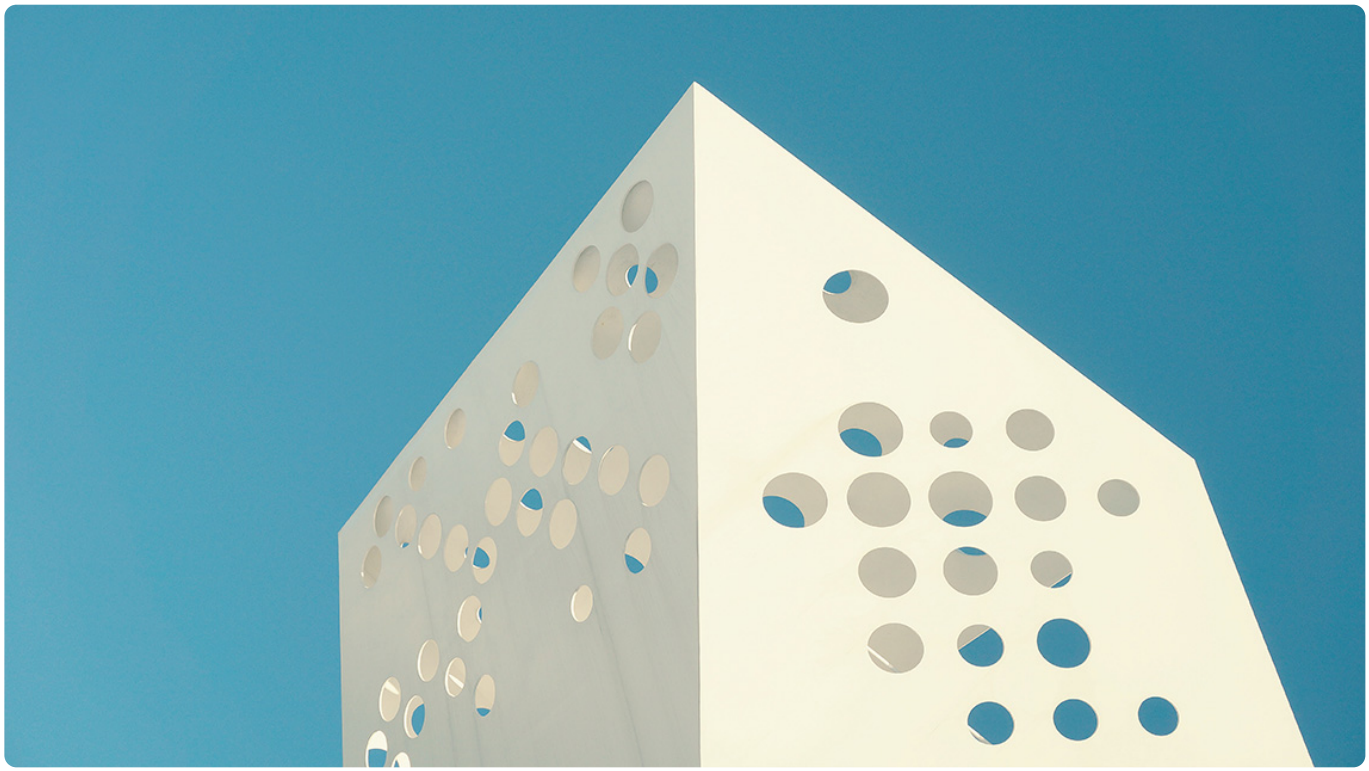
45 | 架构：怎么做详细设计？

2019-09-27 许式伟

许式伟的架构课 进入课程 >



讲述：姚迪迈
时长 11:24　大小 10.46M

你好，我是七牛云许式伟。

我们第三章 “服务端开发篇” 就快要结束了。我们原计划的第三章会分拆为两章：

第三章：服务端开发篇。主要介绍服务端的基础架构与业务架构。

第四章：服务治理篇。主要介绍服务端程序上线与线上服务如何管理的问题。

原先计划的 “第五章：通用架构范式篇” 会取消，核心内容会融合到其他的章节中。详细的调整结果，近期我们会与大家同步新的大纲。

今天我们把话题重新回到架构上。

关于架构，前面我们已经聊了第一步的需求分析和第二步系统的概要设计：

17 | 架构：需求分析（上）

18 | 架构：需求分析（下）- 实战案例

32 | 架构：系统的概要设计

需求分析并不是纯技术的东西，和编程这件事情无关。它关乎的是用户需求的梳理、产品的清晰定义、可能的演变方向。

需求分析的目标和最终结果，都是要最终形成清晰的产品定义。产品定义将明确产品的元素，明确产品的边界，与产业上下游、合作伙伴的分工。

在需求分析阶段，我们关注用户需求的精确表述。我们会引入角色，也就是系统的各类参与方，以及角色间的交互方式，也就是用户故事。

在概要设计阶段，我们一般以子系统为维度来阐述系统各个角色之间的关系。对于关键的子系统，我们还会进一步分解它，甚至详细到把该子系统的所有模块的职责和接口都确定下来。

这个阶段我们的核心意图并不是确定系统完整的模块列表，我们的焦点是整个系统如何被有效地串联起来。如果某个子系统不做进一步的分解也不会在项目上有什么风险，那么我们并不需要在这个阶段对其细化。

为了降低风险，概要设计阶段也应该有代码产出。

这样做的好处是，一上来我们就关注了全局系统性风险的消除，并且给了每个子系统或模块的负责人一个更具象且确定性的认知。

代码即文档。代码是理解一致性更强的文档。

经过系统的概要设计，整个系统的概貌就了然于胸了。详细设计阶段，是需要各个子系统或模块的负责人，对他负责的部分进行进一步的细化。

详细设计关注的是子系统或模块的全貌。

请记住，详细设计并不是只谈实现就完事，更不是一个架构图。它包括以下这些内容。

现状与需求

现在在哪里，遇到了什么问题，要做何改进。

需求满足方式

要做成啥样？交付物的规格，或者说使用界面（接口）。

怎么做到？交付物的实现原理。

概要设计和详细设计的工作内容会有一定的重叠。

概要设计的核心目标是串联整个系统，消除系统的重大风险。在这个过程中，对一些关键模块的实现细节有所考虑是非常正常的。但从另一个角度来说，分解粒度也不能过粗，不应该把特别庞大的子系统直接分出去，这样项目执行的风险就太高了。

但两者的分工不同，考虑的问题重心不同。

比如，从使用界面（接口）来说，概要设计不一定会把子系统或模块的完整接口都列出来，实际上它只关注最核心的部分。但是从详细设计角度来说，接口描述的完备性是必需的。

现状与需求

我们先看看现状与需求。

从逻辑自洽的角度，我们任何一篇文档，首先关注的都应该是要解决的问题与目标。

现状与需求的陈述，要简明扼要。

现状大家都知道，所以不要长篇累牍。更多的是陈述与我们要做的改变相关的重要事实，侧重点在于强调这些事实的存在性和重要性。

比如，假设我们要对某个模块重构。那么，现状就是要谈清楚现在的业务架构是怎样的？它到底有什么样的问题。

需求陈述是对痛点和改进方向的一次共识确认。痛点只要够痛，大家都知道，所以同样不需要长篇累牍。

每个子系统或模块，都有自己的角色分工与用户故事。我们不用重新做一遍需求分析，但对需求分析的核心结论，在详细设计开始之前需要明确。

这很重要。它是我们详细设计所要满足的业务目标。

使用界面（接口）

聊完了现状与需求，接着我们就要谈需求的满足方式。它分两个方面：一方面是交付物的规格，或者说使用界面（接口）。另一方面是背后的实现原理，我们怎么做到的。

规格，或者说使用界面，体现的是别人要怎么使用我。

我们前面一直在强调，使用界面（接口）应该自然体现业务需求，就是强调程序是为用户需求服务的。而我们的架构设计，在需求分析与后续的概要设计、详细设计等过程之间也要有自然的延续性。

使用界面这一部分要详细写，它是团队共识确认的关键。

我们的交付物有哪些可执行文件，有哪些包（package）？如果可执行文件，那么它是一个界面程序，还是服务？如果是服务，网络协议是什么样的？如果是包，它又包含哪些公开的类或函数。

在 “32 | 架构：系统的概要设计” 这一讲中，我们花了非常长的篇幅介绍使用界面（接口）是怎么回事，今天我们就不对这一点进行展开。

需要强调的是，使用界面需要有明确的书写规范。它也是团队共识管理的重要组成，是团队效率、团队默契形成的象征。

更需要强调的是，使用界面的稳定是至关重要的。

接口的变更需谨慎！

对使用界面的不兼容调整，可能出现严重的后果。技术上，可能会导致客户异常，出现编译失败需要重写代码，或者更严重的是，可能导致他们的系统崩溃。商业上，则可能导致大量的客户流失。

实现：数据结构 + 算法

聊完使用界面，接下来就要谈实现原理了，它要体现的是我如何做到。

在 “42 | 实战（二）：“画图”程序后端实战” 一讲中，我们提到过以下这个大家耳熟能详的公式：

> 程序 = 数据结构 + 算法

它是一个很好的指导思想。当我们谈程序的实现时，我们总是从数据结构和算法两个维度去描述它。

我们先看数据结构。

数据结构从大的层面分，可分为基于内存的数据结构，和基于外存（比如 SSD 盘）的数据结构。

对于桌面程序，大部分情况下我们打交道的都是基于内存的数据结构。外存数据结构也会有所涉及，但往往局限于 IO 子系统。

但对于服务端程序，数据结构不完全是我们自己能够做主的。数据结构大部分情况下都是基于外存的，而且有极高的质量要求。

在 “36 | 业务状态与存储中间件” 这一讲中我们也说过，存储即数据结构。所以，服务端程序在数据结构这一点上，最为重要的一件事是选择合适的存储中间件。然后我们再在该存储中间件之上组织我们的数据。

这是数据库这样的存储中间件流行起来的原因。无论是关系型数据库，还是文档型数据库，他们都被设计为一种泛业务场景的数据结构，有很好的业务适应性。

所以在服务端我们谈数据结构，谈的不是内存数据结构，往往谈的是数据库的表结构设计。当然表（Table）是在关系型数据库中的说法，在 mongodb 中我们叫集合（Collection）。但不管我们用的是哪种数据库，出于惯例我们往往还是以 “定义表结构” 一词来表达我们想干什么。

描述表结构，核心需要包含以下内容：

字段名；

类型；

字段含义，以及是否指向另一个表的某个字段；

索引。

你会发现，其实定义表结构和定义内存数据结构本质是完全一致的。定义内存中的一个类（或结构体），我们也关心字段名（成员变量名）和类型，也关心字段的含义，以及它是否指向另一个类（或结构体）的某个字段（成员变量）。

但表结构比内存数据结构多了一个概念：索引。

索引为何存在？我认为有这样几方面的原因。一方面是因为数据库是泛业务场景的通用数据结构，它是动态的，需要依赖索引来提升数据访问的效率。另一方面是因为多租户。多租户导致数据量的爆发式增长，导致大部分情况下遍历查找变得不现实。

索引怎么设计？它完全取决于算法。算法里面使用了哪些数据访问的特征，这些数据访问的频次预期是多少，这些决定了我们添加哪些索引是最划算的。

在涉及的类比较多，或数据库的表结构比较复杂的时候，有时我们会用 UML 类图来对数据结构进行直观的呈现。

谈清楚了数据结构，我们接着聊算法。

在 “程序 = 数据结构 + 算法” 这个说法中，“算法” 指的是什么？在 “42 | 实战（二）：“画图”程序后端实战” 一讲中，我们这么说：

> 在架构过程中，需求分析阶段，我们关注用户需求的精确表述，我们会引入角色，也就是系统的各类参与方，以及角色间的交互方式，也就是用户故事。

> 到了详细设计阶段，角色和用户故事就变成了子系统、模块、类或者函数的使用界面（接口）。我们前面一直在强调，使用界面（接口）应该自然体现业务需求，就是强调程序是为用户需求服务的。而我们的架构设计，在需求分析与后续的概要设计、详细设计等过程之间也有自然的延续性。

> 所以算法，最直白的含义，指的是用户故事背后的实现机制。

> 数据结构 + 算法，是为了满足最初的角色与用户故事定义，这是架构的详细设计阶段核心关注点。

那么，怎么描述一个用户故事对应的算法?

一种方式是基于 UML 时序图（Sequence Diagram）。以下是我个人用过的很好的在线版 UML 时序图制作工具：

https://www.websequencediagrams.com/

另一种方式是基于伪代码（Pseudo Code）。在逻辑较为复杂时，伪代码往往有更好的呈现效果。比如，服务端程序对数据库的 SQL 操作往往比较复杂，但是从 UML 时序图来说流程却并不长，这个时候去画 UML 时序图的意义就不大。

结语

今天我们聊的是怎么做详细设计。

详细设计并不是只谈实现就完事，更不是一个架构图。它包括以下这些内容。

现状与需求
现在在哪里，遇到了什么问题，要作何改进。

需求满足方式

要做成啥样？交付物的规格，或者说使用界面（接口）。

怎么做到？交付物的实现原理。

“程序 = 数据结构 + 算法” 是我们很熟悉的一个公式。它其实是怎么描述实现原理的很好的指导方针。当我们谈程序的实现时，我们总是从数据结构和算法两个维度去描述它。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们对第三章 “服务端开发篇” 进行回顾与总结。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

上一篇 44 | 实战（四）：“画图”程序后端实战

精选留言 (4)

写留言

sprzhing

2019-09-29

比较喜欢这种有启发的文字

展开

Eason

2019-09-28

一个前后端分离的系统中，在详细设计阶段，我们首先第一步是做好数据库设计。然后分工，让人根据数据模型去设计前端 UI的展示，让人做服务端的api 接口定义和接口算法实现。

请问许老师，这么一个详细设计过程分工安排合理吗？我觉得如果这么分工安排，那么...

展开

作者回复: 第一步应该是定义api，它比数据库设计重要

keshawn

2019-09-27

试了下老师推荐WebSequenceDiagrams，香疯了！

展开

Aaron Cheung

2019-09-27

好的架构师一定是好的产品经理 打卡45

展开