59 | 少谈点框架，多谈点业务

2019-11-22 许式伟

讲述：姚迪迈
时长 08:15　大小 7.56M

你好，我是七牛云许式伟。

架构是共识确认的过程

对于架构这件事情，有不少让人误解的地方。前面在 “57 | 心性：架构师的修炼之道”一讲中，我们提到过架构师需要掌握的三大技能：

理需求的能力；

读代码的能力；

抽象系统的能力。

这里面除了 “读代码” 这件事可能允许没有什么显性的产出外（其实也应该有，去读代码通常意味着缺架构设计文档，所以按理应该去补文档），其他两类事情要做好都不容易。

就理需求的能力而言，很多架构师一不知道要做需求分析，二不知道需求分析的产出到底应该是什么样的。需求分析可以说是架构师最没有概念的一个环节，尽管它至关重要。这一块领域特征比较明显，课堂上讲师授课的方式，很难有好的成效，更适合以实训的方式来强化。

就抽象系统的能力而言，很多架构师爱画架构图。画完了架构图，他就认为架构做完了，下一步该去编码。

这有什么问题？

首先，架构过程是团队共识形成与确认的过程。共识是需要精确的、无歧义的。而架构图显然并不精确。

团队没有精确的共识很可怕，它可能导致不同模块的工作牛头不对马嘴，完全无法连接起来，但是这个风险没有被暴露，直到最后一刻里程碑时间要到了，要出版本了，大家才匆匆忙忙联调，临时解决因为架构不到位产生的“锅”。

这时候人们的动作通常会走形。追求的不再是架构设计的好坏，而是打补丁，怎么把里程碑的目标实现了，别影响了团队绩效。

我们作个工程师的类比，这种不精确的架构，就好比建筑工程中，设计师画了一个效果图，没有任何尺寸和关键细节的确认，然后大家就分头开工了。最后放在一起拼接（联调），发现彼此完全没法对上，只能临时修修改改，拼接得上就谢天谢地了。是不是能够和当初效果图匹配？让老天爷决定吧。

更精确描述架构的方法是定义每个模块的接口。接口可以用代码表达，这种表达是精确的、无歧义的。架构图则是辅助模块接口，用于说明模块接口之间的关联。

为了证明接口的有效性，架构师还应该过一遍所有的用户故事，以伪代码或流程图的方式，把所有用户故事过一遍，确认模块之间的接口串起来是可以正常工作的。

实际上更有效的方法是在概要设计（也叫系统设计）阶段就把框架代码写出来，真真正正用代码，而不是伪代码，把用户故事串一遍。

代码即文档。代码是理解一致性更强的文档。

这样做的好处是，我们把联调工作做到了前头，工程的最大风险就得到了管理。剩下来的就是每个模块自身的好坏，这就和组织能力无关，只取决于我们招聘的工程师个体素质了。

所以模块的接口，是架构设计的核心。

别让框架绑架业务

接口代表什么？接口代表业务。架构图代表什么？架构图代表框架。

不要让框架绑架业务。

在架构的两侧，一边是用户需求，一边是技术。接口代表用户需求，代表业务。框架代表技术，是我们满足需求的方法。

框架它是重要的。但是不要让框架反客为主，溢出模块边界。在系统迭代的过程中，框架会经受变化，以适应需求的演进过程。

抓住稳定的东西，比追逐变化更重要。

框架，体现的是需求泛化的能力。从架构思维角度上来说，它是通过抽象出需求模板，把多个需求场景中变化的部分抽离出来，形成相对稳定的泛化需求。

框架的抽象能力不是一蹴而就的，它既依赖我们抽象系统的能力，也依赖我们对领域需求的理解程度。所以框架会随着时间而迭代，逐步向最理想的状态逼近。

如果框架不能满足需求，但我们不迭代框架，而是硬生生去添加这样的功能需求，会发生什么？

结果是，代码逃逸出框架，把系统搅得支离破碎。这时候你可能能够嗅到一丝危险的气息。但是你可能说没办法，里程碑的截止时间就在那里，没办法。

这实际上是大框架面临的最大挑战。它最好能够提前预测所有可能的需求，以此抑制潜在代码逃逸的风险。

但这很难。

所以我们应该换一个角度看这个问题。在如何持续保证系统洁净的这件事情上，我个人给的建议是：

> 连接性的代码越少越好。

什么是连接性的代码？就是把两个子的业务系统连接，构成一个大业务场景的代码。如果有大业务场景，应该抽象出新的更大范畴的业务系统。

这样我们的焦点就始终在业务上。

每个模块都是一个业务。这里我们说的模块是一种泛指，它包括：函数、类、接口、包、子系统、网络服务程序、桌面程序等等。

抽象出符合业务自然语义的接口，远比抽象出泛需求的框架要容易得多。因为，业务语义是稳定的。

关注业务接口的定义，我们自然就把焦点转向关注业务如何由相互正交的子业务组合而来。

我们举例来说明关注业务与关注框架这两种思维方式的差异性。

我们知道，在 IO 系统中，读取磁盘文件中的数据有两种常见的模型：SAX 和 DOM 模型。

SAX 模型是一种基于事件的读盘机制。在读完一个完整的数据单元时，就发送一个读到某数据单元的事件。比如在 XML 中，它的事件接口看起来是这样的：

复制代码

```
type ContentHandler interface {
  StartDocument()
  StartElement(element string, attrs Attributes)
  Characters(chars []byte)
```

```
  EndElement(element string)
  EndDocument()
}
```

DOM 模型则基于对象的组织模型来提供数据读取的能力。细分来说，它又有两种不同的选择。一种是基于抽象的 DOM 树，它看起来是这样的：

复制代码

```
type Nodes interface {
  Len() int
  Elem(i int) Node
}

type Node interface {
  Childs() Nodes
  Name() string
  Type() NodeType
  Text() []byte
  Attributes() Attributes
}
```

另一种是基于更具体的业务逻辑，与具体的领域相关，比如对于 Word/WPS 这样的字处理软件看起来是这样的：

复制代码

```
type Span interface {
  Text() []byte
  Attributes() SpanAttrs
}

type Spans interface {
  Len() int
  Elem(i int) Span
}

type Paragraph interface {
  Spans() Spans
  Attributes() ParagraphAttrs
}

type Paragraphs interface {
  Len() int
  Elem(i int) Paragraph
}
```

```
type Document interface {
  Paragraphs() Paragraphs
  Attributes() DocumentAttrs
}
```

基于 SAX 模型是非常典型的框架思维。它的确足够的通用，但它有两个问题。

一方面，基于事件模型是一个非常简陋的编程框架，与大部分 IO 子系统的需求方的诉求并不那么匹配。关于这一点我们在下一讲 “60 | 架构分解：边界，不断重新审视边界” 还会详细展开。

另一方面，它不体现业务，使用方不能在缺乏文档配合的情况下正确地使用这个接口。本来代码应该是精确的，但是这样的接口把精确性这个最佳的优点给放弃了。

基于 DOM 模型的两种模型中，看起来前者的接口很简洁，但实际上它和上面的 SAX 模型有类似的问题：不体现业务。而后者虽然看起来非常冗长，但是它可以脱离额外的接口说明文档而直接毫无心智负担地使用。毫无疑问，这才是我们该追寻的接口描述方式。

别用实现替代业务

在接口设计中，我们还看到另一种倾向，可以认为是用框架来替代业务的特例：用实现机制替代业务。

我个人经常给架构师们说的一句话是：

> 比框架（架构图）更重要的是数据结构，比数据结构更重要的是接口。

为什么数据结构比框架（架构图）更重要？业务数据结构是架构实现机制的灵魂。从共识确认的角度，数据结构相比框架而言，是更重要的共识。

一些架构师能够想清楚实现，但是想不清楚业务。他们用实现替代对业务系统的抽象。

用实现机制替代业务的典型案例是定义了数据结构，但是不抽象数据的业务逻辑，直接让使用方操作成员变量，或者定义一堆成员变量的 get/set 接口。

另一个例子是当我们用 ORM 框架操作数据库时，工程师非常容易犯的错误是，直接操作数据结构，而忽略定义业务接口的重要性。

结语

今天谈的内容，核心指向一点：

> 架构就是业务的正交分解。每个模块都有它自己的业务。

> 这里我们说的模块是一种泛指，它包括：函数、类、接口、包、子系统、网络服务程序、桌面程序等等。

它看似简单，但是它太重要了，重要到需要单独一讲来把它谈清楚。它是一切架构动作的基础。

架构行为的三步曲：“需求分析”、“概要设计”、模块的 “详细设计”，背后都直指业务的正交分解，只是逐步递进，一步步从模糊到越来越强的确定性，直至最终形成业务设计的完整的、精确无歧义的解决方案。

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。下一讲我们的话题是 “架构分解：边界，不断重新审视边界”。

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　58 | 如何判断架构设计的优劣?

精选留言 (7)

写留言

Fs
2019-11-24

定义了数据结构，但是不抽象数据的业务逻辑，直接让使用方操作成员变量，或者定义一堆成员变量的 get/set 接口。

这句话含义是DDD中的贫血模型了。比如我需要判断某个实体是否是合格的，需要通过a,b字段来判断。这个实现没有定义在数据结构里，却把a,b字段通过get暴露出去，让调用...
展开

作者回复: 是的

1

leslie
2019-11-22

老师今天课程中提到了一个很关键的现状：框架绑架开发。这其实是许多程序过程中经常碰到的问题：某种框架不错大家去用，可是随着业务的复杂度和需求的复杂度的增加我们会发现被框架绑架了-实现不了，然后就开始调整需求或者实现方式。

这些年更多基于内核或者核心去改/写框架的：稍微有点特色就火了；其实我们去看本质会发现，只是降低了对某些框架的绑定能力而已，当程序中一大堆框架时其实我们就...

展开

1

Aaron Cheung

2019-11-22

理清楚需求 列出验收标准 让一线开发理解做成啥样OK 有一些感触

1

沫沫（美丽人生）

2019-11-25

许老师好，现在有些SaaS 企业PaaS化，请问从SaaS 向PaaS 过度的关键技术是什么呢？在系统架构上两者有什么本质区别吗？望不吝赐教。

展开

睡觉

2019-11-22

定好边界，提前抽象出各个子模块对接时的接口与数据，然后是各个子模块的结构。架构师之路，道阻且长

tt

2019-11-22

“框架，提现的是需求泛化的能力”。

体现需求泛化的能力，就是架构可以适应需求的变化。需求的变化就是接口的变化，因为本文说了，接口代表需求。...

展开

作者回复: 赞

Jxin
2019-11-22

目前来看，架构设计套路有限，但业务领域无限。如何应用有限的套路做出无限的架构设计，难点应是在业务上。没有足够的业务背景积累和业务需求洞察，架构设计就会出现知易行难的窘境。但是想要架构师一步到位具有健全的业务领域知识也是不现实的。所以，横竖都不好，那么就落地灵活的设计，动态去演变，也就所谓的演进式架构。而代码变动是高成本的，所以要想办法降低成本。首先代码可读要高。接着，单模块内各层，以及...

展开