58 | 如何判断架构设计的优劣？

2019-11-20 许式伟

许式伟的架构课 进入课程 >

讲述：姚迪迈
时长 09:15 大小 8.49M

你好，我是七牛云许式伟。

想要让自己进步，我们首先得知道什么是好的。所以我们今天的话题是，如何判断架构设计的优劣？

架构设计的基本准则

架构设计会有它的一些基本准则。比如：

KISS：简单比复杂好；

Modularity：着眼于模块而不是框架；

Testable：保证可测试性；

Orthogonal Decomposition：正交分解。

KISS 全称是 Keep it Simple, Stupid，用最直白的话说，“简单就是美”。不增加无谓的复杂性。正确理解系统的需求之后才进行设计。要避免过度设计，除非有人为复杂性买单。

KISS 的“简单”，强调的是易实施性。让模块容易实现，实现的时候心智负担低，比复杂的优化更重要。

KISS 的“简单”，也是主张让你的代码，包括接口，符合惯例。接口语义要自然，最好让人一看方法名就知道怎么回事，避免惊异。

Modularity，强调的是模块化。从架构设计角度来说，模块的规格，也就是模块的接口，比模块的实现机制更重要。

我们应着眼于模块而不是框架。框架是易变的。框架是业务流，可复用性相对更低。框架都将经历不断发展演化的过程，逐步得到完善。

所以不让模块为框架买单。模块设计时应忽略框架的存在。认真审视模块的接口，发现其中“过度的（或多余的）” 约束条件，把它提高到足够通用的、普适的场景来看。

Testable，强调的是模块的可测试性。设计应该以可测试性为第一目标。

可测试往往意味着低耦合。一个模块可以很方便地进行测试，那么就可以说它是一个设计优良的模块。模块测试的第一步是环境模拟。模块依赖的模块列表、模块的输入输出，这些是模块测试的需要，也是模块耦合度的表征。

当然，可测试性不单单因为是耦合的需要。测试让我们能够发现模块构架调整的潜在问题。通常模块在架构调整期（代码重构）最容易引入 Bug。 只有在模块开发过程中我们就不断积累典型的测试数据，以案例的形式固化所有已知 Bug，才可能在架构调整等最容易引发问题的情形下获得最佳的效果。

Orthogonal Decomposition，中文的意思是 “正交分解”。架构就是不断地对系统进行正交分解的过程。

相信大家都听过一个设计原则：“优先考虑组合，而不是继承”。如果我们用正交分解的角度来诠释这句话，它本质上是鼓励我们做乘法而不是做加法。组合是乘法，它是让我们用相互正交、完全没有相关性的模块，组合出我们要的业务场景。而继承是加法，通过叠加能力把一个模块改造成另一个模块。

核心系统的伤害值

正交分解，第一件事情就是要分出哪些是核心系统，哪些是周边子系统。核心系统构成了业务的最小功能集，而后通过不断增加新的周边功能，而演变成功能强大的复杂系统。

对于核心系统的变更要额外小心。如果某新功能早期没有规划，后期却被界定为属于核心功能，我们就需要认真评估它对既有架构的破坏性。

至于周边功能，我们核心考虑的是，如何降低添加一个新的周边功能对核心系统的影响？

不论哪一种情况，如果我们不够小心，系统就会由于不断增加功能而变老化，散发出臭味。

为了减少一个功能带来的负面影响，这个功能相关代码首先要做到尽可能内聚。

代码不一定要写到独立的模块（如果代码量不算大的话），但一定要写到独立的文件里面。对于周边系统，这部分独立出来的代码算是它的功能实现代码，不隶属于核心系统。

我们的关注点是某个周边功能对核心系统的影响。为了添加这个功能，它必然要求核心系统添加相关的代码以获得执行的机会。

我们根据经验可以初步判断，核心系统为这个周边功能增加的代码量越少，那么这个功能与核心系统的耦合就越低。那么，是否有可能把一个功能的添加对核心系统的影响降低到零，也就是不改一行代码？

这当然是可能的，只不过这要求核心系统需要提供所谓 “插件机制”。后续我们会继续探讨这个话题，今天暂且按下不表。

我们先把话题收回到架构设计质量的评估。

上面我们谈了一些架构设计的基本准则，但还谈不上是质量评估的方法。质量判定的方法可以是定性的，也可以是定量的。

定性的判断方法有一定的数据支撑，虽然这种支撑有可能是模糊而感性的。比如我们通常会说，“从 XXX 的角度看，我感觉这个更好”。这里 XXX 是某种定性分析的依据。

从科学严谨性的角度，有定量的判断方法是更理想的状态。可惜的是，到目前为止，我个人并没有听到过任何定量的判断方法来确定架构设计的优劣。但今天我会给出一些个人发明的判定公式。它们都只是经验公式，并没有经过严谨的数学证明。

我们假设，某个架构设计方案将系统分成了 n 个模块，记为：[M_1, M_2, ..., M_n]。其中 M_1 是核心系统，其他模块是周边系统。为简化，我们不妨假设周边系统与周边系统间是正交的，相互没有耦合。

那么，我们第一个最关注的问题是：

> 核心系统受到各周边系统的总伤害是多少？

这里有一个周边功能对核心系统总伤害的经验公式：

$$\sum_{\text{对每一处修改}} log_2(\text{修改行数} + 1)$$

同一个周边功能相邻的代码行算作一处修改。不同周边功能的修改哪怕相邻也算作多处。

这个公式核心想表达的含义是：修改处数越多，伤害越大。对于每一处修改，鼓励尽可能减少到只修改一行，更多代码放到周边模块自己那里去。

这个伤害值公式，当然也同样适用于度量某个周边功能对核心系统的影响面。

核心系统越干净，增加新功能越容易。由于核心系统的地位，所以这个公式实际上是最重要的测量公式。

模块的耦合度测量

我们第二个关注的问题，是每个模块自身的质量。模块自身的质量具体来说，又包括模块接口的质量和模块实现的质量。

我们先看模块接口的质量，这是模块级别最重要的东西。它取决于以下两个方面：

其一，接口与业务的匹配性。简单说，就是接口越自然体现业务越好。然而，从机器判定的角度来说，这一点是不可计算的，完全依赖于个人的主观判断。我们在下一讲 “少谈点框架，多谈点业务” 中将会继续探讨这个话题。

其二，接口的外部依赖，也就是模块接口对外部环境的耦合度。

下面我们要介绍的是模块的 “耦合度测量公式”。它同时适用于模块实现和模块接口的耦合度测量。

假设，我们的模块实现（或模块接口）依赖了模块 A，那么我们的模块实现（或模块接口）与所依赖的模块 A 的耦合度为：

$$\sum_{\text{对每一个依赖的符号}} log_2(\text{符号的出现次数} + 1)$$

依赖的符号（symbol）是指：

被引用的类型，包括 typedef（type alias）、class 或 struct；

被引用的全局变量、全局函数或成员函数。

单个模块清楚了，我们看模块实现（或模块接口）的所有外部依赖，也就是该模块的总耦合度公式为：

$$\sum_{\text{对每一个依赖模块}A} (\text{耦合度}_A * \text{不成熟度系数}_A)$$

其中，耦合度 A 是该模块与依赖模块 A 的耦合程度，公式见上。不成熟度系数 A 表征依赖模块 A 的不成熟度程度。如果依赖模块 A 完全成熟，不会再发生改变则为 0；如果模块在

发生非常剧烈变动，连规格都完全没法确定则为 1。

通过该耦合度测量公式可以看出，我们鼓励依赖外部成熟模块。理论上完全成熟的模块很可能就只有语言内置的数据类型如 int、string 等，其他模块多多少少还是会经受一些变化，所以还是尽量减少外部依赖。

另外值得一提的是，将模块接口引用的类型 A 改为 object 或 interface{} 类型并不能降低耦合度。也就是说如果某参数为 interface，那么这个 interface 的耦合度要看功能实际使用时，它存在各种可能的类型，都会计算在依赖中。

我们应该怎么看待耦合度测量公式?

需要强调的是，它是一个经验公式，仅仅是代表了某种价值主张。在实际应用中，计算得到的具体耦合度值并无物理含义，只能用来对比两个相同功能的系统（或模块）架构设计方案。对于两个功能完全不同的 A、B 系统（或模块），其计算结果并不能用于评判彼此的好坏。

结语

今天我们探讨的话题是如何评判架构设计的优劣。

首先我们谈的是架构设计的基本准则，它们虽然不足以明确说谁好或是不好，但是指明了方向。

然后我们开始对架构好坏做定性甚至定量的分析。考虑到核心系统的重要性，我们单独引入了一个伤害值来评估它的纯洁度。

最后，我们对模块自身的接口或实现，给出了耦合度测量公式。通过这个公式，我们明确了我们架构设计的价值主张。

但我们需要意识到的一点是，这些并不是全部。判断模块间的耦合度是复杂的。上面我们的公式某种程度上来说只考虑了静态依赖关系，而没有考虑动态依赖。

比如说，我们考虑两个网络模块 A 和 B，一个显而易见的耦合度判断是：

A 调用 B 的网络接口数量越多，依赖越大（静态依赖，上面我们已经考虑）；

A 调用 B 的网络接口的次数越多，依赖越大（动态依赖，上面我们未考虑）。

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 “少谈点框架，多谈点业务”。

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　用户故事 | 站在更高的视角看架构

下一篇　59 | 少谈点框架，多谈点业务

精选留言 (6)

写留言

吴

2019-11-21

精彩，用心的总结

展开

2

Geek_03056e
2019-11-20

两个核心模块之间由接口调用，并且是A调用B这种的单方面调用，可是两个模块有公用的结构体数据，这些结构体的定义，应该放到那个位置呢？

展开

作者回复: 如果认为B更加基础、可以放在B。也有人会放在单独的C。anyway，如果A、B、C都隶属于核心系统，全局来说放在哪里只是个细节，对外来说没区别，只不过最好有一个总的入口级模块D把所有这些模块组装起来形成一个整体的DOM。

2

Bachue Zhou
2019-11-22

其实还有一种耦合度经常被人忽视，就是技术耦合度。我们相信，无论是今天多么流行的技术，未来都有失去竞争力的一天，但要开一家长达数十年甚至数百年的技术公司，不可能长期使用已经失去竞争力的技术，必然涉及到技术的多次演进。如果公司的关键性产品与某一种技术耦合度太高，则当该技术失去竞争力或彻底淘汰后，技术迁移成本过高，极易造成迁移彻底失败。所以我从不看好那种一个 Git 库里数十万甚至数百万行 xx 语言或...

展开

作者回复: 👍

1

丁丁历险记
2019-11-21

笔记
1 kiss 有时候，效率就是少做无意义的事。
2 模块化
缺经验，伤害值，耦合度为啥那样，理解不来。
模块设计时应忽略框架的存在。认真审视模块的接口，发现其中“过度的（或多余的）...

展开

1

欧星星
2019-11-23

还是不太明白组合为什么比继承更好，老师说组合是做乘法继承是做加法为什么会是组合好呢，本来是想着继承是强依赖如果父类发生了变化会直接影响子类的行为，但如果用组合的话不一样也会影响实现的行为吗？希望老师解惑

展开


Aaron Cheung

2019-11-20

KISS：简单比复杂好；

封装很多次的代码 是否所谓体现抽象能力呢