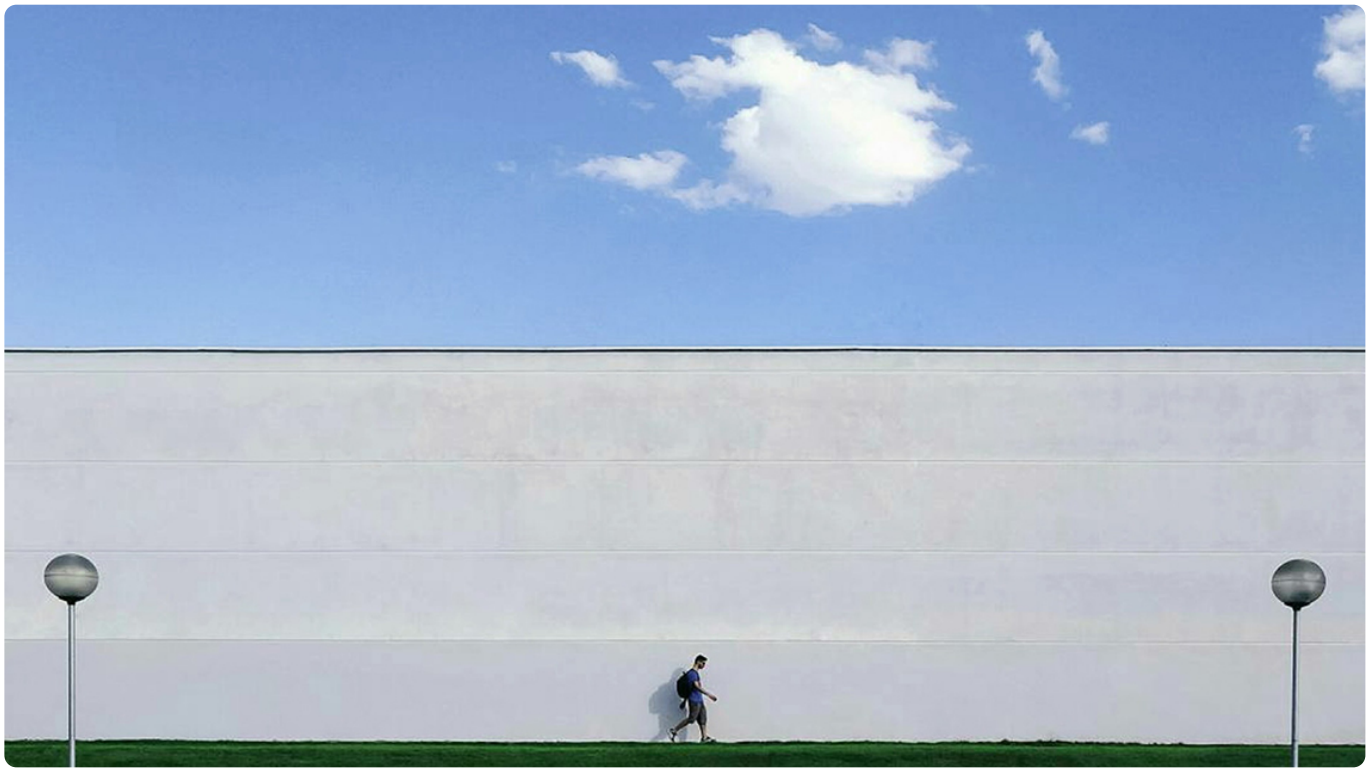
加餐 | 想当架构师，我需要成为“全才”吗？

2019-08-09 极客时间编辑部



你好，这里是极客时间编辑部。

不知不觉，“许式伟的架构课”专栏已经更新了 3 个多月，我们的后台收到了同学们数以千计的留言。许式伟老师每天都在实时关注着留言并回答同学们的问题，同时根据同学们的留言不断优化调整课程的设置。

老师和同学在留言区的互动也十分热闹精彩，今天我们就精选出一批留言，一起来看一看。

精选问答

1. 老师好，人精力有限，如果什么都懂，那不是不精了？通才还能做架构师吗？还是“一专多能”，先“专”，精通一样；再“多能”，了解其它技术？

答：挺好的问题。架构师绝对不是要把自己打造为全才。架构师掌控全局的核心思想是打通经络，让自己的内力在全身自然流通，浑然一体。在不影响理解的情况下，你需要放弃很多

2. 许老师，自己现在已经工作快三年了，想往架构师这个方向走，但现在自己有些迷茫，接触到的技术也算挺多了，但不知道该如何入手架构师，之前您也提到过先广度然后深度，但我想问达到什么算广度够了，怎么进行深度学习？

答：架构师核心是把知识串起来，构建一个完整的认知，不留疑惑。大部分知识是不需要深入细节的，只在你需要的时候深入，但深入的时候要很深。

3. 如何来确定需求中哪些是稳定的？对架构角度，关注需求到什么层次？

答：挺好的问题。需求分析的重要性怎么形容都不过分。准确的需求分析是做出良好架构设计的基础。很多优秀的架构师之所以换到一个新领域一上来并不一定能够设计出好的架构，往往需要经过几次迭代才趋于稳定，原因在于新领域的需求理解需要一个过程。除了心里对需求的反复推敲的严谨态度外，对客户反馈的尊重之心也至关重要。

4.**老师好，我有三个问题。**

> **现在运维开发基本上都用 Go，Python 慢慢变少了，Java 也少用了……现在运维开发是要学 Go 吗？**
>
> **架构师要学数据结构和算法吗？很多都说算法是“内功”，中小公司好像学了都基本用不到。**
>
> **现在不是流行 Docker+k8s、微服务、DevOps、AI 等，那些主流技术都要了解吗？OpenStack 云计算这两年基本不讲了，是否不用学习？**

答：关于你的三个问题，我的意见是下面这样的。

学 Go 挺好，建议学，生产效率很高的开发工具。

“算法用不到”其实更准确的说法是“想不到”，或者是已经有人实现了你只需要调用，不需要自己实现。但是只有你知道了背后的道理，你才能明白算法对应的限制在哪里，什么情况下应该用什么算法。

高阶的技术可以按需学，按精力学，更根本的还是要打好基础，这也更有助于你判断是否应该深入学习某些技术。

题变得无所不包。”

交互能力也看做一种计算能力吗？这句话应该怎么理解呢？这种交互不就是输入和输出？哪有计算？

答：广义的计算包含有副作用的函数（有 IO 的函数）。因为数据交换本身也是计算的需求，否则计算没有办法与现实世界相互作用。任何工具都需要解决现实问题才有用。计算器还有交互按钮呢，并不是只有纯正的计算。

6. 我要做一个最小机器人系统，需要考虑需求的变化点和稳定点。该怎么考虑呢？

答：挺典型的问题。这个问法是一种典型的需求陈述误区。

描述需求需要有几个典型的要素：

用户，面向什么人群；

他们有什么要解决的问题；

我解决这个问题的核心系统。

只有满足这几个要素的需求才能进一步讨论变化点和稳定点。最小机器人可能符合上面的第三点，但是用户人群和要解决的问题没有描述，也就无法进一步去思考到底哪些因素是稳定的，哪些是易变的。

7. 编程框架和编程范式具体有什么区别呢？感觉它们都具备约束、规范的作用。

答：最主要的差别是：编程框架通常是领域性的，比如面向消息编程是多核背景下的网络服务器编程框架；编程范式则是普适性的，不管解决什么领域的问题都可以适用。

8. 老师提到了如果需要重修数据结构这门课程，大学里面学的数据结构是不顶用了。那应该学习什么呢，您可以给个建议吗？

答：这方面的资料不太多。可以给你一个我当年翻过的资料： Purely Functional Data Structures

你可以参考看看。

9. 老师您好，我不太理解您说的继承是个过度设计的原因，我目前在架构过程中大量使用了继承，而且我也觉得继承功能将我的代码功能高度抽象化，给我带来了很大的方便。我想咨询下您如果不是用继承的话，用什么方法替代继承的功能呢？

答：建议继承只使用接口继承；正常情况下，优先用组合；当然因为大部分语言的组合功能不够强大，有时候从便捷性的角度继承可以适度使用，但是应当意识到如果过度使用继承对工程来说是有害的。

10. 老师授课知识的角度很有深度，更贴切地说是一种思维方式，这种深度思考，从事情的本质重新推演与复盘的思考方式是很值得学习的。因为我们大部分人应该都没有想过自己去重新设计一个计算机的实现。不知道我这么理解的对不对。

答：从无到有到万物，我们这个课的脉络之一就是重新从零构建整个信息世界，这一点在开篇词中提到过，这一点非常非常关键。另一个脉络是架构思维的递进，这一章重点是需求分析。这两个脉络相辅相成，交织在一起。

11. 许老师，您好，虽然自己是科班出身，但是对于下面这个问题困扰了我很久。

> **一般来说程序的运行需要 OS 的支撑，那么在 BIOS 之前，选择运行哪个操作系统那一段程序是怎么运行的？再问一句，编译器可以独立于操作系统运行吗？**
>
> **可以用 C 语言去实现很多其他的语言，比如说 Python、Go 等，那在 C 语言之前，这么一直追寻下去，会衍生成鸡生蛋的问题。但是编程语言又具有自举的功能，那自举是怎么实现的？比如目前版本的 Go 核心实现中，很多是用 Go 本身实现的，它是怎么做到自己编译自己的？**

答：先回答你的第一个问题，程序运行不需要操作系统支持，有 BIOS 支持就可以（把控制权交给它）。编译器可以独立于操作系统存在，而且它应该先于操作系统产生。

接下来是第二个问题，语言诞生的过程是这样的：机器码 => 汇编 =>C =>C 写的汇编、C 写的 C（自举）。当然这个过程不需要每次新架构的 CPU 或操作系统都重新来一遍，因为

12. 许老师：选择某种语言无关的接口表示；能举个例吗？

答：先看看是网络协议层的接口，还是跨语言的二进制接口。

前者比如 protobuf 之类就挺好，后者可以了解一下 IDL 之类的东西，不过我觉得都有点重。如果要跨语言，我的建议在网络协议层跨，或者用操作系统的动态库机制（有点原始但很轻）；如果语言内的接口，就别太复杂了，用语言自己的机制挺好的。

13. 请问每个应用的虚拟内存地址是怎么分配的？起始地址都是 0 吗？函数 F 可以跨多个虚拟内存页吗？

答：操作系统会保留一个地址空间，0 通常也在保留区间内，因为 0 开始往往是中断向量表的地址，其他的地址区间怎么分配其实应用自己说了算。函数和数据都可以跨内存页。

14. 老师您好，有两个问题希望解答。

> **淘汰的内存页数据保存在哪里；是保存在外置存储设备中吗；**
>
> **CPU 加载对应程序的代码段到内存中，那么 CPU 是如何知道这个对应程序的代码段在什么位置的呢？**

答：第一个问题：是的，保存在外置存储中。对于 unix 系的系统往往是 swap 分区；windows 则是一个隐藏属性的.swp 文件。
第二个问题：代码段在哪里，是操作系统约定的，因为负责加载的人是操作系统，它设计程序文件的数据格式。

15. 使用 Java 四年了，看到封装，继承，多态的描述，特别精准，又有了更深刻的理解。不了解 Go 语言，比如有一个表单的基类，里面有基本的处理，子类继承这个基类，有自己特殊的实现。这种情况，如何用组合实现呢？

答：这是受继承思维的影响了。其实继承实现了代码复用和多态两个东西，揉在一起。在 Go 里面，组合实现代码复用，接口实现多态，彼此完全独立，非常清晰。

答：挺好的问题。硬件中断和软中断不一样。硬件中断你可以理解为总是会定期检查。软中断本身是一条指令，所以不存在检查这样的概念。

17. 交叉编译是什么意思，不是很理解，老师能讲讲吗？

答：其实理解清楚一个实质：编译器就是把高级语言翻译成为机器码，更抽象说，它其实就是格式转换器。

目标格式是不是编译器正在运行的环境并不重要，只不过如果目标格式刚好是当前机器的 CPU+ 操作系统，那么目标格式就可以直接执行，否则就编译出一个当前环境下无法执行的目标格式，这种情况就叫交叉编译。

18. 关于外存管理，有个问题从之前就困扰我。

磁盘的 IO 是由 CPU 完成的吗？但之前见到的说法是“CPU 只能操作内存”。既然今天又提到了这个问题，文中提到“大量的磁盘 IO 操作，非常占用 CPU 时间”，那这两种说法是否矛盾？

还想知道磁盘中的数据是怎么被加载到内存上来的呢？另外，更多的文章是说，“CPU 的速度远远大于磁盘 IO，CPU 经常需要‘等待’磁盘 IO”，这明显也是一种将 CPU 和外存割舍开的一种说法，而且按这种说法，CPU 不光无需分配很多时间片给 IO，而且还有很多“等待”时间。这也和本文中“非常占用 CPU 时间”相矛盾吧？

答：所有外设 CPU 都统一基于数据交换（IO）的方式操作。CPU 并不知道数据的含义，但是设备的使用方和设备知道。

这种情况下你可以简单理解 CPU 只是一根网线，但是很重要的一点是它让设备使用方和设备可以交互。CPU 并不负责磁盘 IO，但是它要等它结束以接收数据。这方面当然也有一些新技术出现改善这一点，可以想一想可能的优化路径，这里不表。

19. 有一个疑问：协程属于用户态的线程，它跟线程之间怎么对应呢？协程之间也需要切换，那线程切换的那些成本它一样有啊，没想明白它的优势在哪。

这种差异类似于系统调用和普通函数调用的差异。因为高性能服务器上 io 次数实在太多了，所以单位成本上能够少一点，积累起来也是很惊人的。

20. 这种对需求的前瞻性探索挺重要，但同时感觉也是最难的，应该如何培养呢

答：很多时候是思维方式的转变。首先要尝试去做前瞻，预测错了并不可怕，但可以事后复盘到底是缺失了什么重要的信息让你判断出现了什么偏差。

21. 隐隐感觉到架构的主要难点在于对需求的前瞻性判断，这要求的不仅仅是技术能力。目前几乎所有的架构课程，都是基于确定的需求来讲技术架构，例如秒杀系统怎么做高可用高并发。不知道我这么理解对不对。

答：架构在于创造，如果你从事的事情总是重复别人，那这个公司又有何价值？即使有所参考，也应该有自己的精气神，这个精气神是需要架构师把它干出来的。

精选学习留言

恭喜 @有铭和 @Enthusiasm 两位同学，你们的留言被选为精选留言，极客时间将送出价值 99 元的专栏阅码一份。1 个工作日之内，工作人员会与你取得联系。

@有铭 同学留言

对象范式的原始概念其实根本不包括类和继承，只有 1. 程序由对象组成，2. 对象之间互相发送消息，协作完成任务。

最初世界上第一个面向对象语言是 Simula-67，第二个面向对象语言是 Smalltalk-71。

Smalltalk 受到了 Simula-67 的启发，基本出发点相同, 但是最大的不同是 Smalltalk 是通过发消息来实现对象方法调用，而 Simula 是直接调用目标对象的方法。

Bjarne Stroustrup 在博士期间深入研究过 Simula，非常欣赏其思想，C++ 的面向对象思路直接受其影响，因为调用目标对象的方法来“传递消息”需要事先知道这个对象有哪些方

随着 C++ 的大行其道，继承和封装变成了面向对象世界的核心概念，OOP 至此被扭曲为 COP （ Class Oriented Programming，面向类程序设计）。

但是 COP 这套概念本身是有缺陷的：每个程序员似乎都要先成为领域专家，然后成为领域分类学专家，然后构造一个完整的继承树，然后才能 new 出对象，让程序跑起来。

到了 1990 年代中期，问题已经十分明显。UML 中有一个对象活动图，其描述的就是运行时对象之间相互传递消息的模型。1994 年 Robert C. Martin 在《 Object-Oriented C++ Design Using Booch Method 》中，曾建议面向对象设计从对象活动图入手，而不是从类图入手。

而 1995 年出版的经典作品《 Design Patterns 》中，建议优先考虑组合而不是继承，这也是尽人皆知的事情。

这些迹象表明，在那个时候，面向对象社区里的思想领袖们，已经意识到“面向类的设计”并不好用。只可惜他们的革命精神还不够，Delphi 之父在创建.Net Framework 的时候，曾经不想要继承，在微软内部引起了很大的争议，最后是向市场低头，加上了继承。

2000 年后，工程界明确提出：“组合比继承重要，而且更灵活”，Go 和 Rust 也许是第一批明确的对这种思路进行回应的语言，它们的对象根本不需要类本身来参与，也能完成对象范式的多态组合。

历史让 C++ 走上了舞台，历史也终将让 COP 重新回到 OOP 的本来面目

@Enthusiasm 同学学习笔记

总结：设计系统架构的前提是用户需求分析，用户需求包括分析出稳定需求点和变化需求点。从功能上看，稳定需求点一般是实现偏核心需求的需求点，变化需求点往往是实现偏扩展性需求的需求点。

从层次结构上看，稳定需求点往往在系统层次的底层，而变化需求点往往在更加抽象层（上层）。从从属关系上看，稳定点需要提供功能给变化点使用，变化点调用稳定点提供的功

系统架构类似于一个栈的结构，人机交互（变化点）放在栈顶，底层工作（稳定点）置于栈底。

这节课程让我联想到网络中的 OSI 7 层模型。大概其也体现了这种软件架构思想。好处就是架构清晰，职责明确，功能规范等等。

以往我认为的架构设计类似上面的描述，描述起来类似按自顶向下顺序，采用分治思想完成。但许老师的方法又有些巧：架构好比搭积木，许老师是先有了很多积木（需求点），然后把再确定这些积木放在哪一层次的格子里。这简化了架构设计的难度，好比用市场经济代替计划经济，很有趣。

架构设计博大精深，灵活多变，初学课程的我们，对架构设计的学习，也只能算是盲人摸象。

如果你在课程中有看不懂的地方，有想解答的架构问题，或者想分享的实战经验，都可以在文章下留言，如果你的留言被选中作为精选留言，我们将会为你送出价值 99 元的阅码一份。欢迎留言，与许式伟老师一起交流讨论，教学相长，共同精进。

上一篇 课外阅读 | 从《孙子兵法》看底层的自然法则

精选留言 (6)

写留言

未设置
2019-08-09

作为一个没毕业的小白，想知道学校里面学的软件设计就是架构吗，有什么不同呢。

作者回复: 是架构，不同是讲法不同，背后是对架构认知不同

1

fy
2019-08-11

忙这个词，就是借口。主要自己还是时间安排不好。

展开

fy
2019-08-11

谢谢老师提醒,这两天开始学Go语言，书用的是七牛云团队翻译关Go程序设计语言，发现徐文浩老师推荐的一本书《程序员的自我修养》-链接，装载，库对本专栏前面知识理解有很大帮助

作者回复:

Enthusiasm
2019-08-10

留言被选为精选，实属惭愧。专栏也并没有一字不落看完。当初无非是用心写了几句只言片语，想着给老师捧捧场而已。回过头来再看，自己已然记不清当初是如何写出那些简单稚嫩的话。信息过载的时代，获取信息在于精而不在于多，关键在于能把学到的知识转化为自己的内在和产出。学过许老师那么多节操作系统，自以为可以大概听懂鸿蒙系统介绍了吧，然而听到介绍鸿蒙系统的“微内核”时，我竟是懵逼的，自认为自己是业内人，...

fy
2019-08-09

一直在看，主要太忙了，没实践，现在计划花一点时间学习下go，然后跟着老师学习，并做把老师写的程序理解。同时真心觉得老师用go去写那些代码，提交到github上学习，特别好，赞！

作者回复: 做中学。首先还是要勤动手，计算机是一门实践科学。然后配合本专栏去思考和梳理背后的道理，方能快速进步。

1

江中芦苇
2019-08-09

什么时候进入第三章及以后章节？期待

展开

作者回复: 第二章还有2讲，总结系统架构的方法论，以及总结篇回顾第二章。