02 | 架构设计的历史背景

2018-05-01 李运华

02 | 架构设计的历史背景
朗读人：黄洲君　11′05″ | 5.08M

理解了架构的有关概念和定义之后，今天，我会给你讲讲架构设计的历史背景。我认为，如果想要深入理解一个事物的本质，最好的方式就是去追寻这个事物出现的历史背景和推动因素。我们先来简单梳理一下软件开发进化的历史，探索一下软件架构出现的历史背景。

机器语言（1940 年之前）

最早的软件开发使用的是“**机器语言**”，直接使用二进制码 0 和 1 来表示机器可以识别的指令和数据。例如，在 8086 机器上完成“s=768+12288-1280”的数学运算，机器码如下：

```
10110000000000000000011
00000101000000000110000
00101101000000000000101
```

不用多说，不管是当时的程序员，还是现在的程序员，第一眼看到这样一串东西时，肯定是一头雾水，因为这实在是太难看懂了，这还只是一行运算，如果要输出一个“hello world”，面对几十上百行这样的 0/1 串，眼睛都要花了！

看都没法看，更何况去写这样的程序，如果不小心哪个地方敲错了，将 1 敲成了 0，例如：

```
10110000000000000000011
00000101000000000110000
00101100000000000000101
```

如果要找出这个程序中的错误，程序员的心里阴影面积有多大？

归纳一下，机器语言的主要问题是三难：**太难写、太难读、太难改**！

汇编语言（20 世纪 40 年代）

为了解决机器语言编写、阅读、修改复杂的问题，**汇编语言**应运而生。汇编语言又叫“**符号语言**”，用助记符代替机器指令的操作码，用地址符号（Symbol）或标号（Label）代替指令或操作数的地址。

例如，为了完成“将寄存器 BX 的内容送到 AX 中”的简单操作，汇编语言和机器语言分别如下。

```
机器语言：1000100111011000
汇编语言：mov ax,bx
```

相比机器语言来说，汇编语言就清晰得多了。mov 是操作，ax 和 bx 是寄存器代号，mov ax,bx 语句基本上就是“将寄存器 BX 的内容送到 AX”的简化版的翻译，即使不懂汇编，单纯看到这样一串语言，至少也能明白大概意思。

汇编语言虽然解决了机器语言读写复杂的问题，但本质上还是**面向机器**的，因为写汇编语言需要我们精确了解计算机底层的知识。例如，CPU 指令、寄存器、段地址等底层的细节。这对于程序员来说同样很复杂，因为程序员需要将现实世界中的问题和需求按照机器的逻辑进行翻译。例如，对于程序员来说，在现实世界中面对的问题是 4 + 6 = ？。而要用汇编语言实现一个简单的加法运算，代码如下：

```
.section .data
  a: .int 10
  b: .int 20
  format: .asciz "%d\n"
.section .text
.global _start
```

```
_start:
  movl a, %edx
  addl b, %edx
  pushl %edx
  pushl $format
  call printf
  movl $0, (%esp)
  call exit
```

这还只是实现一个简单的加法运算所需要的汇编程序，可以想象一下，实现一个四则运算的程序会更加复杂，更不用说用汇编写一个操作系统了！

除了编写本身复杂，还有另外一个复杂的地方在于：不同 CPU 的汇编指令和结构是不同的。例如，Intel 的 CPU 和 Motorola 的 CPU 指令不同，同样一个程序，为 Intel 的 CPU 写一次，还要为 Motorola 的 CPU 再写一次，而且指令完全不同。

高级语言（20 世纪 50 年代）

为了解决汇编语言的问题，计算机前辈们从 20 世纪 50 年代开始又设计了多个高级语言，最初的高级语言有下面几个，并且这些语言至今还在特定的领域继续使用。

- Fortran：1955 年，名称取自”FORmula TRANslator”，即公式翻译器，由约翰·巴科斯（John Backus）等人发明。
- LISP：1958 年，名称取自”LISt Processor”，即枚举处理器，由约翰·麦卡锡（John McCarthy）等人发明。
- Cobol：1959 年，名称取自”Common Business Oriented Language”，即通用商业导向语言，由葛丽丝·霍普（Grace Hopper）发明。

为什么称这些语言为“高级语言”呢？原因在于这些语言让程序员不需要关注机器底层的低级结构和逻辑，而只要关注具体的问题和业务即可。

还是以 4 + 6=？这个加法为例，如果用 LISP 语言实现，只需要简单一行代码即可：

```
(+ 4 6)
```

除此以外，通过编译程序的处理，高级语言可以被编译为适合不同 CPU 指令的机器语言。程序员只要写一次程序，就可以在多个不同的机器上编译运行，无须根据不同的机器指令重写整个程序。

第一次软件危机与结构化程序设计（20 世纪 60 年代~20 世纪 70 年代）

高级语言的出现，解放了程序员，但好景不长，随着软件的规模和复杂度的大大增加，20 世纪 60 年代中期开始爆发了第一次软件危机，典型表现有软件质量低下、项目无法如期完成、项目严重超支等，因为软件而导致的重大事故时有发生。例如，1963 年美国（http://en.wikipedia.org/wiki/Mariner_1）的水手一号火箭发射失败事故，就是因为一行 FORTRAN 代码错误导致的。

软件危机最典型的例子莫过于 IBM 的 System/360 的操作系统开发。佛瑞德·布鲁克斯（Frederick P. Brooks, Jr.）作为项目主管，率领 2000 多个程序员夜以继日地工作，共计花费了 5000 人一年的工作量，写出将近 100 万行的源码，总共投入 5 亿美元，是美国的“曼哈顿”原子弹计划投入的 1/4。尽管投入如此巨大，但项目进度却一再延迟，软件质量也得不到保障。布鲁克斯后来基于这个项目经验而总结的《人月神话》一书，成了畅销的软件工程书籍。

为了解决问题，在 1968、1969 年连续召开两次著名的 NATO 会议，会议正式创造了“软件危机”一词，并提出了针对性的解决方法“软件工程”。虽然“软件工程”提出之后也曾被视为软件领域的银弹，但后来事实证明，软件工程同样无法根除软件危机，只能在一定程度上缓解软件危机。

差不多同一时间，“结构化程序设计”作为另外一种解决软件危机的方案被提了出来。艾兹赫尔·戴克斯特拉（Edsger Dijkstra）于 1968 年发表了著名的《GOTO 有害论》论文，引起了长达数年的论战，并由此产生了结构化程序设计方法。同时，第一个结构化的程序语言 Pascal 也在此时诞生，并迅速流行起来。

结构化程序设计的主要特点是抛弃 goto 语句，采取“自顶向下、逐步细化、模块化”的指导思想。结构化程序设计本质上还是一种面向过程的设计思想，但通过“自顶向下、逐步细化、模块化”的方法，将软件的复杂度控制在一定范围内，从而从整体上降低了软件开发的复杂度。结构化程序方法成为了 20 世纪 70 年代软件开发的潮流。

第二次软件危机与面向对象（20 世纪 80 年代）

结构化编程的风靡在一定程度上缓解了软件危机，然而随着硬件的快速发展，业务需求越来越复杂，以及编程应用领域越来越广泛，第二次软件危机很快就到来了。

第二次软件危机的根本原因还是在于软件生产力远远跟不上硬件和业务的发展。第一次软件危机的根源在于软件的“逻辑”变得非常复杂，而第二次软件危机主要体现在软件的“扩展”变得非常复杂。结构化程序设计虽然能够解决（也许用“缓解”更合适）软件逻辑的复杂性，但是对于业务变化带来的软件扩展却无能为力，软件领域迫切希望找到新的银弹来解决软件危机，在这种背景下，面向对象的思想开始流行起来。

面向对象的思想并不是在第二次软件危机后才出现的，早在 1967 年的 Simula 语言中就开始提出来了，但第二次软件危机促进了面向对象的发展。面向对象真正开始流行是在 20 世纪 80 年代，主要得益于 C++ 的功劳，后来的 Java、C# 把面向对象推向了新的高峰。到现在为止，面向对象已经成为了主流的开发思想。

虽然面向对象开始也被当作解决软件危机的银弹，但事实证明，和软件工程一样，面向对象也不是银弹，而只是一种新的软件方法而已。

软件架构的历史背景

虽然早在 20 世纪 60 年代，戴克斯特拉这位上古大神就已经涉及软件架构这个概念了，但软件架构真正流行却是从 20 世纪 90 年代开始的，由于在 Rational 和 Microsoft 内部的相关活动，软件架构的概念开始越来越流行了。

与之前的各种新方法或者新理念不同的是，“软件架构”出现的背景并不是整个行业都面临类似相同的问题，“软件架构”也不是为了解决新的软件危机而产生的，这是怎么回事呢？

卡内基·梅隆大学的玛丽·肖（Mary Shaw）和戴维·加兰（David Garlan）对软件架构做了很多研究，他们在 1994 年的一篇文章《软件架构介绍》（An Introduction to Software Architecture）中写到：

> “When systems are constructed from many components, the organization of the overall system-the software architecture-presents a new set of design problems.”

简单翻译一下：随着软件系统规模的增加，计算相关的算法和数据结构不再构成主要的设计问题；当系统由许多部分组成时，整个系统的组织，也就是所说的“软件架构”，导致了一系列新的设计问题。

这段话很好地解释了“软件架构”为何先在 Rational 或者 Microsoft 这样的大公司开始逐步流行起来。因为只有大公司开发的软件系统才具备较大规模，而只有规模较大的软件系统才会面临软件架构相关的问题，例如：

- 系统规模庞大，内部耦合严重，开发效率低；
- 系统耦合严重，牵一发动全身，后续修改和扩展困难；
- 系统逻辑复杂，容易出问题，出问题后很难排查和修复。

软件架构的出现有其历史必然性。20 世纪 60 年代第一次软件危机引出了“结构化编程”，创造了“模块”概念；20 世纪 80 年代第二次软件危机引出了“面向对象编程”，创造了“对象”概念；到了 20 世纪 90 年代“软件架构”开始流行，创造了“组件”概念。我们可以看

到，“模块”“对象”“组件”本质上都是对达到一定规模的软件进行拆分，差别只是在于随着软件的复杂度不断增加，拆分的粒度越来越粗，拆分的层次越来越高。

《人月神话》中提到的 IBM 360 大型系统，开发时间是 1964 年，那个时候结构化编程都还没有提出来，更不用说软件架构了。如果 IBM 360 系统放在 20 世纪 90 年代开发，不管是质量还是效率、成本，都会比 1964 年开始做要好得多，当然，这样的话我们可能就看不到《人月神话》了。

小结

今天我为你回顾了软件开发进化的历史，以及软件架构出现的历史背景，从历史发展的角度，希望对你深入了解架构设计的本质有所帮助。

这就是今天的全部内容，留一道思考题给你吧。为何结构化编程、面向对象编程、软件工程、架构设计最后都没有成为软件领域的银弹?

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

精选留言

公号-Java大后端
2018年5月1日心得

239

在古代的狼人传说中，只有用银质子弹（银弹）才能制服这些异常凶残的怪兽。在软件开发活动中，“银弹”特指人们渴望找到用于制服软件项目这头难缠的“怪兽”的“万能钥

匙”。

软件开发过程包括了分析、设计、实现、测试、验证、部署、运维等多个环节。从IT技术的发展历程来看，先辈们在上述不同的环节中提出过很多在当时看来很先进的方法与理念。但是，这些方法、理念在摩尔定律、业务创新、技术发展面前都被一一验证了以下观点：我们可以通过诸多方式去接近“银弹”，但很遗憾，软件活动中没有“银弹”。

布鲁克斯发表《人月神话》三十年后，又写了《设计原本》。他认为一个成功的软件项目的最重要因素就是设计，架构师、设计师需要在业务需求和IT技术中寻找到一个平衡点。个人觉得，对这个平衡点的把握，就是架构设计中的取舍问题。而这种决策大部分是靠技术，但是一定程度上也依赖于架构师的“艺术”，技术可以依靠新工具、方法论、管理模式去提升，但是“艺术”无法量化，是一种权衡。

软件设计过程中，模块、对象、组件本质上是对一定规模软件在不同粒度和层次上的“拆分”方法论，软件架构是一种对软件的“组织”方法论。一分一合，其目的是为了软件研发过程中的成本、进度、质量得到有效控制。但是，一个成功的软件设计是要适应并满足业务需求，同时不断“演化”的。设计需要根据业务的变化、技术的发展不断进行“演进”，这就决定了这是一个动态活动，出现新问题，解决新问题，没有所谓的“一招鲜”。

以上只是针对设计领域的银弹讨论，放眼到软件全生命周期，银弹问题会更加突出。

小到一个软件开发团队，大到一个行业，没有银弹，但是“行业最佳实践”可以作为指路明灯，这个可以有。

2018-05-01

作者回复

赞，666，你已经提前帮我做了后面相关内容的预热了👍👍

2018-05-02

cruise 60

从哲学角度来说，是不存在银弹的。任何技术或方法都不是独立来看的，要综合其它各种相关因素来考虑的。因此对别人来说可能是银弹的，对你来说可能是个炸弹了。架构设计也是一样的，不能脱离业务、公司实际情况、人员配置、经费预算、时间投入等等与技术本身无关的因素，但却又是影响，甚至决定架构设计方向的因素。因此说没有最好，只有更合适。

2018-05-01

narry 55

软件开发最本质的挑战有两个:复杂和变更，而软件的价值是保证业务的响应力，而与之相对的是开发资源的有限，而各种的软件开发方法论，也都是在研究有限的资源下，如何应对着两个挑战，寻找平衡点，实现业务目标，因为是在寻找平衡点，就说明是有取舍的，所以就没有所谓的银弹的存在

2018-05-02

作者回复

回答的很好，作者也受到了启发，谢谢👍👍

2018-05-02

felix 17

变化才是唯一的不变，所以银弹不会存在

2018-05-02

作者回复

言简意赅，抓住了核心本质，“银弹”产生于一定的历史背景和大环境，而历史和环境总是会变化的

2018-05-02

李志博 15

软件开发的结果在于人，而不在于方法论，面向对象，设计模式，架构，这些概念的推出距离现在，好几十年了吧，可真正理解透彻的能有多少呢，就算有像作者这样理解透彻的，还在一线开发的能有多少……阿里的p9难道还在一线写代码嘛……最终写代码的人还是理解不到位的我们，技术强的，写的项目能多撑两年，但是复杂到一定程度，没有良好关系架构指导，都是坑

2018-05-01

作者回复

其实不一定要P9才要理解到位呢，我2014年就写了《面向对象葵花宝典》，那时我还在写代码的哦，其实我现在也写代码，不写代码很多技术没法确切理解，我现在写demo代码比较多，例如用golang写个简单的区块链，用java写个reactor等

2018-05-02

合民 14

作者这个问题是否在考验，读者认真看了这篇文章没有？我认为文章的软件发展历史正是答案，软件工程归根结底是为各行各业的需求服务的，而随着需求的复杂度越来越高，用户的要求越来越高，软件也越复杂，形态也在不断变化，所以没有一种方法论能称得上是银弹，只能说某一种方法论适合某一种需求。这也正是架构师存在的意义，去选择合适的技术，如果有银弹，还要架构师干嘛！以上只是个人见解！

2018-05-01

作者回复

你已经看穿一切👍👍

确实是想通过介绍历史来启发大家思考

2018-05-02

Alspadger 12

因为设计者都是站在当时的业务瓶颈下考虑问题的，因为你不可预测当业务发展的一定程度后，又会遇到怎么样的技术瓶颈。也就是所谓的技术支撑业务发展，业务推动技术发展。

2018-05-01

xuan 10

“No Silver Bullet”的原文是：“没有任何技术或管理上的进展，能够独立地许诺十年内使生产率、可靠性或简洁性获得数量级上的进步。”之所以这样说，是因为软件的根本困难（E

ssence，包括复杂度、一致性、可变性、不可见性)
复杂度:规模上, 软件实体可能比任何由人类创造的其他实体更复杂, 因为没有任何两个软 件部分是相同的
一致性:软件的变化必须遵循一系列接口标准规范,有些情况下它的变化就是要兼容;
可变性:一般有如下几种情况:
1.当客户喜欢用某个功能或者某个功能能解决他的某些问题时,他会针对这方面提出很多优化该功能的需求点
2.硬件或者其他配件的升级变化 必须升级现有软件平台
不可见性:软件不存在一种空间形态 可以通过一张图
或者其他载体来可视化展示 ,不能通过地图 电路设计图等来全面展示.
由于这几个点的变化，导致系统越来越臃肿,从而导致管理成本上升,沟通困难,可靠性逐年下降等等；而结构化 面向对象等主要是来提高生产率 可靠性和简洁性

2018-05-02

作者回复

没有看过《人月神话》的程序员不能成为好的架构师😃😃👍👍

2018-05-02

Mark Yao 8

软件本身的复杂度难以度量，随时间和规模发展，原有的解决方案很快难适应，人们就不断总结经验模式和设计解决新困难的办法，但是不管什么样的架构设计都是在尽量满足适应我们可能遇到的问题的解决方案，不是解决问题方案。生活中我们的应用从单体到主备再到集群、分布式、微服务最后到最新的Service Mesh，这些其实都是解决和改善、完善、优化我们在软件开发遇到的问题。There is no silver bullet.

2018-05-02

作者回复

回答正确👍

2018-05-02

crazyone 5

感觉像是看大佬们在华山论剑般，评论相当精彩

2018-05-06

淡云天 5

解空间是建立在问题空间之上的，问题空间的扩展速度远超解空间时，就会架空解空间。而这时就需要新的、适应问题空间扩展速度的解空间来担当这个阶段的银弹。这一点类似于宏观物理学和量子物理学，只不过物理学几百年的进化之路，计算机只用了二十年就走完了。。。

2018-05-02

带刺的温柔 5

软件架构是为了解决大规模开发时遇到的效率、复杂及扩展性问题。听老师所说让我对架构认知又更加清晰落地。但是对拆分粒度越来越粗，层次越来越高理解的还是不够，其实与我

的一些开发习惯是相悖的，一般我会尽可能拆分细来保证后期的扩展性。不知道老师我是哪里理解偏差了

2018-05-01

闭嘴 4

感觉作者对整个软件行业有比较深入的了解。就是内容太少。还没看就没了。希望后面的文章多来一点干货。让我这种小白能够学习到一点实质的东西。能够解决项目问题的一些东西。希望大神能够把自己的功力展现60%就行。

2018-05-02

作者回复

这是提炼出来的，为了写这一篇，我写了2~3周，如果觉得意犹未尽，可以在这个基础上继续去探索

2018-05-02

候鸟归来的季节 4

技术在不断发展，新的业务需求催生新的技术，没有银弹

2018-05-01

阿罗 3

超赞👍，

饱含认识论和系统论的理论知识与软件实践知识。太难得了，非大集成者无以为之。我买过最值的课程！

2018-05-01

Welton 3

技术在不断发展和更新，没有一劳永逸的事物能够代替不断变化的大千世界需求！

2018-05-01

laolinshi 3

每一种开发方法都是应对当时出现的危机而发展出来的，有一定的局限性。随着软件技术的发展，新的问题会层出不穷的出现，必然会催生新的开发方法来解决对应的问题。

2018-05-01

猴哥 2

为什么软件架构还不是银弹，因为还有更加高级的等着我们去开发挖掘。究其原因，我个人认为主要是人在进步，当下看可能是最好的方法，过十年人类经验的积累，可能会发现更好的理论，这可能是个死循环，所以没有银弹。

2018-05-01

作者回复

唯一不变的是变化本身

2018-05-02

夏大伟 2

同意布鲁克斯的观点，没有银弹的一个原因是由于软件本身具有复杂性，一致性，可变性，不可见性。复杂性主要是指软件的复杂程度可以看成代码数量的函数，但是这个函数究竟是什么类型的，不确定，和具体面对的业务领域也有关系。一致性指软件需要和硬件保持一致，而硬件种类繁多，也一直在发展。

2018-05-01

KingPoker 2

推荐一本书「伟大的计算原理」，把计算机的本质问题描述的很透彻，也给我有一些全新的认知。

2018-05-01

老甘 2

所有的解决方案都是为了解决某一类问题，既然是某一类问题势必是术业有专攻，从辩证的角度也很好理解，事物都有两面性，不存在某种方案可以解决所有问题而没有任何弊端，有弊端自然会出现针对某个场景更好的解决方案。而且，世界在变化，针对现实世界解决问题的软件业务逻辑也势必不断变化，所以只有针对多数场景暂时的银弹，而没有永久的。未来还有更好的

2018-05-01

时光之刃 2

知识量好大，循序渐进，读来挺顺畅

2018-05-01

南友力max先森 2

软件系统是为业务服务的，随着业务的变化，软件系统面临的问题也会不断变化，要想得出所有业务问题（包括系统自身的维护问题）的通用解解决方案必须先知道所有的问题域，但现在谁又能说所有的业务和系统问题都已经浮现呢

2018-05-01

kx163 2

各种软件开发方法都有各自的利弊点，能提高开发效率和降低开发维护成本，但很难降低软件系统本身的复杂性和易变性的特点

2018-05-01

弓土 2

软件领域没有银弹我理解根因是需求的不断的变化包括底层软件每隔一段时间都会面对上层支撑业务的变化，而当初的制定的策论及方法都没有预料到未来的变化

2018-05-01

VC 1

大道至简，治大国，如烹小鲜，软件开发也是如此，所有复杂事情都一样，最难把控的是度。微服务盛行的当下，有多少架构师能把握好度呢？

2018-07-27

作者回复

微服务被用烂了

2018-07-30

水月洞天 1

软件架构是在特定的环境下，对于特定业务复杂度的系统做出的设计和应对。硬件效率在变，网络资源在变，业务扩张水平在变，互联网的应用体量在变，导致复杂性在变，架构的设计是为了保证软件的稳定和顺利交付，而每次的人力投入水平和资源不同，所以没有万能的银弹。但不同的复杂度情况下，通用的设计能保证稳定性，但不能保证顺利交付

2018-06-08

王旭东 1

银弹只能是某个历史时间点上存在，但时代在变化，业务在变化（比如用户量剧增），所有以前的银弹并不可能满足现在。故而更多问题的出现也在催生那个时间点的最佳解决方案。大数据、分布式、人工智能等等

2018-05-10

QuincySx 1

感觉架构实际上是在思想上进行引领与指导，而且与业务逻辑有强耦合，所以每个公司具体业务不同，架构也不太一样，只能说相同的功能去借鉴一下架构设计，并不是那种一针见血的大招，通用性很强

2018-05-07

ciciywg 1

Jeff Atwood对《人月神话》的推荐：毫无疑问，这是我们领域里惟一的一本经典图书。如果你还没读过，你应该觉得丢脸！——我终于在2014年7月系统的读完了一遍……

2018-05-07

Will 1

怎么可能出现一套方法能解决一万年后出现的问题呢?
可能时间是“银弹”吧。

2018-05-05

波波安 1

软件是用来满足业务的。随着时代的变化，人们思想的变化，软件需要满足和支撑的业务也在变化，所以软件结构也需要不断的演进来满足这种变化的需求。就像在数据爆炸的现在，要利用好这些数据，我们就要有分布式的架构方案来满足大数据存储，大数据的分析。

2018-05-04

路易斯陈凯瑞 1

银弹的出现是否意味着这个行业已经到头了

2018-05-04

强 1

关于银弹，我想从另外一个角度聊聊。上学时候，老师(c++标准编写者)跟我们分享的一思考题: 软件究竟属于工程行业还是偏艺术(或工艺)行业。前几十年，软件从业者基本是努力将其往工程化发展，像硬件制造一样可控，高度复用，流水生产。经过这些年的发展，工程化基本未实现(否则码农就和生产线工人一样，工资不会越来越高)，现在越来越多人思考，也行软件更多是艺术行业。既然是艺术类，自然就无银弹的说法

2018-05-02

守拙 1

“没有银弹”直接说是没有意义。在特定的上下文，银弹是存在的。这是模块，对象，组件等概念。来解决特定的通常会碰到的问题。但会存在更多的地方还没有银弹。所以单独的“没有银弹”这句话是没有意义的，而比起这句话，更重要的是不上下文弄懂。

2018-05-01

Jaime 1

应该说都有自己的缺点和优点。看具体场合使用具体的方法，在软件领域就没有银弹的说法。因为业务场景不同，需要用不同的方法去解决业务上的问题。技术是为业务服务的。有些业务可能需要代码简单清晰，此时结构化编程就够了，业务规模加大时，发现耦合过于严重，面向对象就要拿出来解决耦合问题，提高软件可维护性。当规模继续加大，就可能要拆分系统，使用到了微服务的技术，那么此时架构设计就需要解决各个服务之间调用的问题。而在各个微服务里面又可能是用了结构化编程或者面向对象的方法(要看各个服务自己的业务需求)。软件工程我是不太了解😅😅😅

2018-05-01

杜晓东 1

一个猜测：软件是人造物，理论上是存在银弹的。
从情感上讲：从业者也需要ta的存在😁
这样的话，结构化编程、面向对象编程、软件工程、架构设计都是银弹的组成部分。

2018-05-01

作者回复

这是说我们一直在完善，从来没达成😃

2018-05-02

成功 1

硬件发展速率和软件设计发展不均衡，形成的矛盾

2018-05-01

作者回复

那硬件会不会是银弹呢？例如量子计算机出现，假如性能比现在的计算机提升1亿倍，会不会导致软件领域出现革命性变化？如果有可能，那可能会是什么？会是人工智能么？

2018-05-02

阿斌 1

我理解的架构设计，是职责划分的过程，但是对于架构师来说，怎么划分职责一直是一个难题。

2018-05-01

作者回复

架构设计的本质和目的，后续章节会详细展开

2018-05-02

夏天的味道 1

现在组件级别的粒度都已经不够了，所以提出以服务划分的微服务😄

2018-05-01

华林 1

解决问题的办法绝对不是依赖于越来越高的抽象层次，人类的弱点决定了这样的结果，无论抽象层次有多高，人都会把它变得更复杂，于是导致了更高级别的抽象，然后递归/死循环

2018-05-01

liyue326 0

要是有银蛋。那也是万能的程序员吧

2018-07-22

作者回复

没有这样的程序员，Jeff Dean也不行😄

2018-07-24

绿豆先生 0

软件的出现就是为了满足需求，需求的变化是永无止境的，不断的适应新变化才是软件的生命力所在，因此上银弹不存在。

2018-07-16

黄大仙 0

随着芯片物理极限的不断提升，限制软件运行的主要矛盾，从计算能力和计算成本，加速过渡到想象力上来，而想象力不可能有通用方案。虽然方案不是万能的，但是方案的基本要素是相通的，谁在什么条件下，通过哪些方法，解决哪些人的什么问题。

2018-07-16

作者回复

我表示我还没有达到这个境地😄

2018-07-16

yoummg 0

作者的用心令人敬佩。

为什么现在我们在谈“架构”，他不是平白无故产生的，他是在一定的背景下产生的。更好地理解他产生的原因，会在具体解决问题的时候做到有的放矢。

直到现在才看明白，what，why，how。这真是一个认清事物最本质的三步。👍👍👍

2018-07-08

作者回复

你已经洞悉天机👍👍😄整个专栏思路就是这样的

2018-07-09

杨玉龙 0

录音工作是主播擅长的，理论知识的讲解是知识专业人擅长的，这样搞有点儿照本宣科啦！课程内容我感兴趣，但是如此做实在理解不了

2018-07-07

作者回复

很多人说录音效果不错呢，如果我去读，这嗓子估计你更不想听😂😂

2018-07-09

varotene 0

由于接触的大部分都是面向对象的项目，能举几个现代软件业，不使用面向对象的例子吗？

2018-07-05

作者回复

redis, nginx都不是面向对象的，虽然里面用了函数指针来实现了类似面向对象的功能

2018-07-05

哼 0

现在所做的所有方案都是针对目前发生的问题、所显现或认知到的问题，而并非针对未来在实践过程中产生的新问题，并不存在一个万能的方法一劳永逸解决所有问题。所以并不存在银弹。

2018-06-26

林步蜓 0

银弹，作为直线子弹，是用来对付明确的一个小目标靶的。
而软件工程或者说一个大产品项目，不是一个目标靶，也不是战术目标，而是未知的战役。

2018-06-26

钢 0

设计模式是处于架构设计与面向对象之间？

2018-06-24

作者回复

设计模式就是面向对象的类和接口设计方法

2018-06-25

念一 0

时间不会停，问题一直会存在！

2018-06-24

wj 0

历史的发展已经说明了答案，软件开发演进从机器语言，汇编语言，高级语言，第一次软件危机产生的结构化理论，第二次软件危机的面向对象理论，演进过程中，技术的发展，需求的复杂变化，硬件的改进都会带来与当前的开发模式不协调的地方，例如多核带来的并发并行模型，区块链的发展，cpu的numa和smp架构，甚至未来量子理论的发展，都可能改变现在的研发思维..

2018-06-16

Cola 0

1.技术是一个渐进式的发展过程，任何设计思想都是针对当下环境的解决方案 2.架构设计是图纸，最后真正添砖加瓦的是执行人，架构设计的目的是让让系统接近业务处理的极限，这个上限始终是由业务层决定，如果业务代码混乱，到最后再好的架构设计都会无用武之地

2018-06-14

作者回复

好的架构能够降低业务代码混乱的影响范围

2018-06-15

火山飘雪 0

我认为软件是为业务需求服务的，只要业务服务不断的变化，复杂度不断得增加，软件开发就会跟着变更

2018-05-29

巫 0

个人理解，从某些角度来说，银弹还是存在的。不同问题、不同时期会有不同的银弹。银弹可以杀死狼人，但可能杀不死进化的狼人。
正像大家说的，软件有多方面的复杂性，银弹也就不止一种。

2018-05-29

Adun Ton 0

没有银弹，需求的发展使问题总是超前于有效的方法。

2018-05-29

mkmonkey 0

银弹:终极大招！没有一个软件是靠大招来上线的，是靠着前后端的配合，测试的严谨来完成项目的，没有哪个环节代替所有

2018-05-28

sensitivemix 0

需求总是变化～没有银弹，老师说的很好

2018-05-27

雨林霖 0

架构终究是为了实现业务，没有大一统的架构支撑千变万化的业务

2018-05-27

清晨之风

0

架构不只是技术的堆砌，更是一种艺术的权衡，和美学的体现。

2018-05-27

作者回复

是的，就像是梵高的画，很多时候看不懂😃😃

2018-05-27