02 | 大厦基石：无生有，有生万物

2019-04-19 许式伟

讲述：姚迪迈
时长 16:02 大小 14.69M

你好，我是七牛云许式伟。

在上一篇中，我们把“构建一个应用程序”类比成“构建一座大厦”，并从宏观全局的视角剖析了应用程序这座大厦的构成。今天，我们将更加近距离地去解剖这座大厦的地基：冯·诺依曼体系结构。

解剖架构的关键点是什么？

在解剖之前，我想和你先谈谈“解剖学”：**我们应该如何去分析架构设计中涉及的每一个零部件。换一句话说，当我们设计或分析一个零部件时，我们会关心哪些问题。**

第一个问题，是需求。这个零部件的作用是什么？它能被用来做哪些事情？（某种意义上来说更重要的是）它不会被用来做哪些事情？

你可能会说，呀，这个问题很简单，既然我设计了这个零部件，自然知道它是用来干嘛的。但实质上这里真正艰难的是“为什么”：为何这个零件被设计成用来干这些事情的，而不是多干一点事情，或者为什么不是少干某些事情？

第二个问题，是规格。这个零部件接口是什么样的？它如何与其他零件连接在一起的？

规格是零部件的连接需求的抽象。符合规格的零部件可以有非常多种可能的实现方案，但是，一旦规格中某个条件不能满足了，它就无法正常完成与其他零件的连接，以达到预期的需求目标。

规格的约束条件会非常多样化，可能是外观（比如形状和颜色），可能是交互方式（比如用键盘、鼠标，或者语音和触摸屏），也可能是质量（比如硬度、耐热性等等）。

那么，冯·诺依曼体系结构的需求和规格又是什么样的呢？

为“解决一切的问题”而生

冯·诺依曼体系结构不但是应用程序这座大厦的地基，同时也是整个信息科技的地基。

当我们去审视整个信息科技时，仅把它形容为一座大厦显得如此不贴切，甚至你也不能用“一个城市”去形容它，事实上，它更像是一个无中生有的全新世界：在其中，有个体、有族群、有生态，还有喜怒哀乐。

冯·诺依曼体系结构的迷人之处在于，从需求来说，它想解决一切问题。解决一切可以用“计算”来解决的问题。

“计算”的边界在哪里？今天我们还没有人能够真正说得清。计算能不能解决“智能”的问题？通过计算能力，计算机是否终有一天可以获得和人类一样的智能？

今天人工智能热潮的兴起，证明对于这个问题我们很乐观：计算终将解决智能的问题。尽管我们不能确定什么时候能够达到，但是让人欣慰的是，我们一直在进步 —— 如果人类智能无法完成进一步的进化，那么我们就一直一直在前进，最终无限逼近甚至超越人类智能。

甚至有科幻小说家设想（例如在 Google 的“AlphaGo”大热后，霍炬和西乔创作的漫画“BetaCat”），计算机演进出超过人类的智能是生物进化的一个自然演进路径，它将取代人类成为新的食物链顶端，并最终基于其悠久的生命力，去完成人类有限生命无法实现的星际航行之路。

冯·诺依曼体系的规格

为了实现“解决一切可以用‘计算’来解决的问题”这个目标，冯·诺依曼引入了三类基础零部件：

中央处理器；

存储；

输入输出设备。

首先我们来看看存储。它负责存放计算涉及的相关数据，作为计算的输入参数和输出结果。

我们日常见到的存储设备非常的多样化。比如：中央处理器自己内置的寄存器、内存、传统机械硬盘、USB 固态硬盘、光盘等等。

从中央处理器的角度，存储可简单分为两类：一类是内置支持的存储，通过常规的处理器指令可直接访问，比如寄存器、内存、计算机主板的 ROM。一类是外置存储，它们属于输入输出设备。中央处理器本身并不能直接读写其中的数据。

冯·诺依曼体系中涉及的“存储”，指的是中央处理器内置支持的存储。

我们再来看看输入输出设备。它是计算机开放性的体现，大大拓展了计算机的能力。每个设备通过一个端口与中央处理器连接。通过这个端口地址，中央处理器可以和设备进行数据交换。数据交换涉及的数据格式由设备定义，中央处理器并不理解。

但这并不影响设备的接入。设备数据交换的发起方（设备使用方）通常理解并可以解释所接收的数据含义。为了方便使用，设备厂商或操作系统厂商通常会提供设备相关的驱动程序，把设备数据交换的细节隐藏起来，设备的使用方只需要调用相关的接口函数就可以操作设备。

最后我们来看看中央处理器。它负责程序（指令序列）的执行。指令序列在哪里？也存放在存储里面。计算机加电启动后，中央处理器从一个固定的存储地址开始执行。

中央处理器支持的指令大体如下（我们在第一篇文章中也曾提到过）：

计算类，也就是支持我们大家都熟知的各类数学运算，如加减乘除、sin/cos 等等；

I/O 类，从存储读写数据，从输入输出设备读数据、写数据；

指令跳转类，在满足特定条件下跳转到新的当前程序执行位置、调用自定义的函数。

和“解决一切可以用‘计算’来解决的问题”这个伟大的目标相比，冯·诺依曼体系的三类零部件的规格设计显得如此精简。

为什么这么简洁的规格设计，居然可以解决这么复杂的需求？

需求是怎么被满足的？

我们来设想一下：假如今天让我们从零开始设计一个叫电脑的东西，我们的目标是“解决一切可以用‘计算’来解决的问题”。

对于这么含糊的需求，如果你是“电脑”这个产品的主架构师，你会如何应对？

让我们来分析一下。

一方面，需求的变化点在于，要解决的问题是五花八门包罗万象的。如何以某种稳定但可扩展的架构来支持这样的变化？而另一方面，需求的稳定之处在于，电脑的核心能力是固定的，怎么表达电脑的核心能力？

电脑的核心能力是“计算”。什么是计算？计算就是对一个数据（输入）进行变换，变为另一个数据（输出）。在数学中我们把它叫“函数”。如下：

> y = F(x)

这里 x、y 是数据。它们可能只是一个简单的数值，也可能是文本、图片、视频，各种我们对现实问题进行参数化建模后的测量值，当然也可能是多个输入数据。但无论它的逻辑含义为何，物理上都可以以一段连续的字节内容来表达。用 Go 的语法表达就是：

复制代码

```
func F(x []byte) (y []byte)
```

那么 x、y 物理上在哪里？思路推理到这里，“存储” 这个概念自然就产生了：存储，就是存放计算所要操作的数据的所在。

下面的问题是：一个具体的计算（也就是 F 函数）怎么表达？

这里的难点在于，F 对于电脑的架构师来说是未知的。那么，怎么设计一种系统架构让用户可以表达任意复杂的计算（函数）？

逻辑上来看，无论多复杂的自定义函数，都可以通过下面这些元素的组合来定义：

内置函数，比如整数或小数运算（加减乘除、sin/cos 等）；

循环和条件分支；

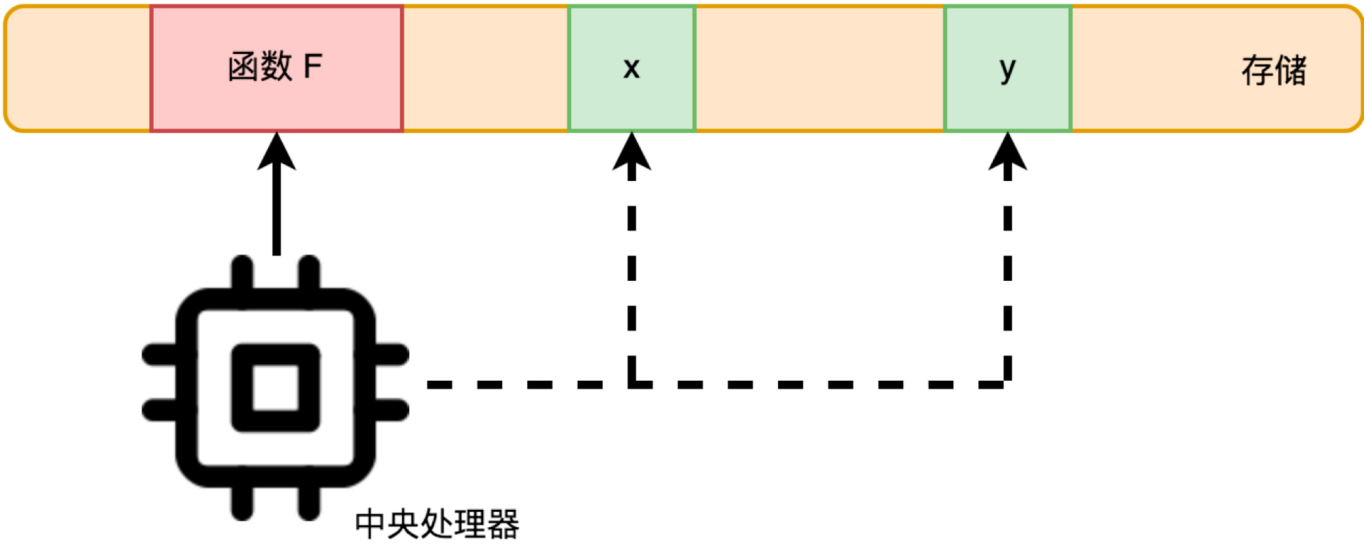
子函数（也是自定义函数）。

这样一来，对于任意的一个具体的计算（自定义函数）来说，都可以用一组指令序列来表达。

那么函数 F 物理上在哪里？以指令序列形式存放在存储里面。所以，存储不只存放计算所要操作的数据，也存放“计算”本身。

只是，存储里面存放的“计算”只是数据，需要有人理解并执行这些数据背后的计算行为，才变成真正意义的“计算”。这个执行者，就是中央处理器（CPU）。它支持很多计算指令，包括执行内置函数、循环和条件分支、执行子函数等。

所以，有了中央处理器 + 存储，就可以支持任意复杂的“计算”了。



只是如果电脑只有“中央处理器 + 存储”，那它就如同一个人只有头脑而没有四肢五官，尽管很可能很聪明，但是这种聪明无法展现出来，因为它没法和现实世界发生交互。

交互，抽象来看就是输入和输出。对人来说，输入靠的是五官：眼睛看、耳朵听、鼻子闻、舌头尝，以及肌肤接触产生的触觉。输出靠语言（说话）和各种动作，如微笑、眨眼、皱眉、手势等等。

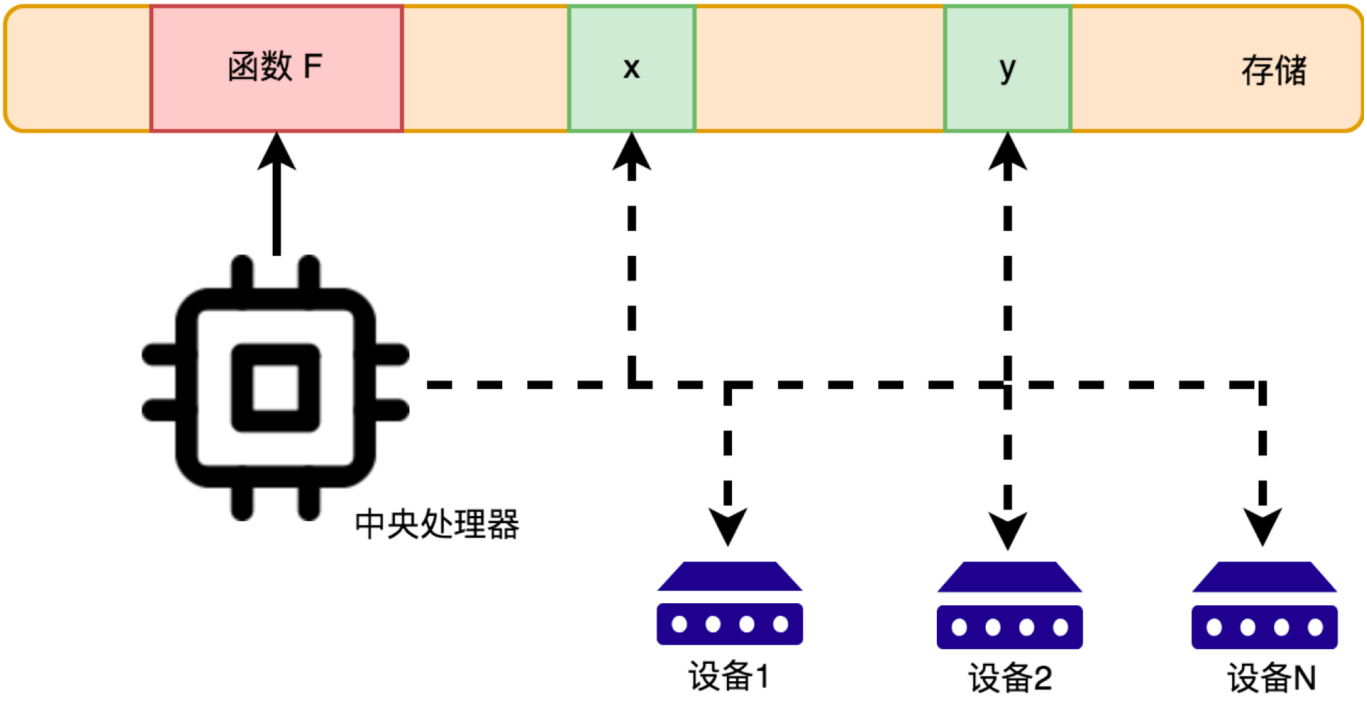
对于电脑来说，输入输出的需求就更多了，不只是四肢五官，而可能是千肢万官。

从输入需求来说，可能采集静态图像、声音、视频；也可能采集结构化数据，如 GPS 位置、脉搏、心电图、温度、湿度等；还可能是用户控制指令如键盘按键、鼠标、触摸屏动作等。

从输出需求来说，可能是向屏幕输出信息；也可能是播放声音；还可能是执行某项动作，如交通灯开关、汽车马达转动、打印机打印等。

但不管是什么样交互用途的器官（设备），我们要做的只是定义好统一的数据交换协议。这个数据交换机制，和网络上两台电脑通过互联网，需要通过某种数据交换协议进行通讯，需求上没有实质性的差别。

也就是说，除了纯正的“计算”能力外，中央处理器还要有“数据交换”能力（或者叫 IO 能力）。最终，**电脑可以被看做由 “中央处理器 + 存储 + 一系列的输入输出设备”** 构成。如下图：



尽管输入输出设备引入的最初灵感可能是来自于“交互”，但是当我们去审视输入输出设备到底是什么的时候，我们很自然发现，它能够做的不单单是交互。

比如常见的外置存储如机械硬盘、光盘等，它们也是输入输出设备，但并不是用于交互，而是显著提升了电脑处理的数据体量。

输入输出设备从根本上解决的问题是什么？

是电脑无限可能的扩展能力。

最重要的一点，输入输出设备和电脑是完全异构的。输入输出设备对电脑来说就只是实现了某项能力的黑盒子。

这个黑盒子内部如何？没有规定。它可以只是一个原始的数字化的元器件，也可以是另一台冯·诺依曼架构的电脑，还可以是完全不同架构的电脑，比如 GPU 电脑、量子计算机。

你可以发现，引入了输入输出设备的电脑，不再只能做狭义上的“计算”（也就是数学意义上的计算），如果我们把交互能力也看做一种计算能力的话，电脑理论上能够解决的“计算”问题变得无所不包。

架构思维上我们学习到什么？

架构的第一步是需求分析。从需求分析角度来说，关键要抓住需求的稳定点和变化点。需求的稳定点，往往是系统的核心价值点；而需求的变化点，则往往需要相应去做开放性设计。

对于“电脑”这个产品而言，需求的稳定点是电脑的“计算”能力。需求的变化点，一是用户“计算”需求的多样性，二是用户交互方式的多样性。

电脑的“计算”能力，最终体现为中央处理器的指令集，这是需求相对稳定的部分。

用户“计算”需求的多样性，最终是通过在存储中的指令序列实现。计算机加电启动后，中央处理器并不是按自己固有的“计算”过程进行，而是从一个固定的存储地址加载指令序列执行。

通常，这个固定的存储地址指向计算机主板的 ROM 上的一段启动程序（BIOS）。这段启动程序通常包含以下这些内容。

- 存储设备的驱动程序，用以识别常规的外置存储设备，比如硬盘、光驱、U 盘。
- 基础外部设备的驱动程序，比如键盘、鼠标、显示器（显卡）。
- 设备和启动配置的基础管理能力。
- 在外置存储上执行程序的能力（中央处理器只支持在内存上执行程序，当然它也为在外置存储执行程序提供了一些支持，比如内存页缺失的中断处理）。
- 将执行权转移到外置存储（第一次安装操作系统的时候可能是光驱甚至是网络存储，平常通常是硬盘）上的操作系统启动程序。这样，操作系统就开始干活了。

这样一来，“计算”需求的多样性只需要通过调整计算机主板上的 BIOS 程序，乃至外置存储中的操作系统启动程序就可以实现，而不必去修改中央处理器本身。

用户交互方式的多样性，则通过定义外部设备与中央处理器的数据交换协议实现。

当我们把所有的变化点从电脑的最核心部件中央处理器剥离后，中央处理器的需求变得极其稳定，可独立作为产品进行其核心价值的演进。

结语

总结一下，今天，我们近距离地去解剖了整个信息世界地基：冯·诺依曼体系结构。

冯·诺依曼体系结构的不凡之处在于，它想“解决一切可以用‘计算’来解决的问题”。

为了实现这个目标，冯·诺依曼引入了三类基础零部件：中央处理器、存储、输入输出设备。所有计算机都可以看做由 “中央处理器 + 存储 + 一系列的输入输出设备” 构成。

为了方便理解，我在尝试用 Go 语言模拟来实现冯·诺依曼架构体系的电脑：

https://github.com/qiniu/arch/tree/master/von

如果你对此感兴趣，欢迎 fork 并对其进行修改迭代。

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。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　01 | 架构设计的宏观视角

下一篇　03 | 汇编：编程语言的诞生

精选留言 (88)

写留言

明翼
2019-04-19

37

笔记：老师通过对冯诺依曼计算机架构体系的剖析，让我们了解在为支持复杂的计算时候，大师是如何将计算的核心设计为跳转，通用计算等简单核心功能的CPU；
通过对外设接口的抽象隔离了复杂的而又有无限可能的外设；
方便存取指令和数据的内存。
简单来说就是找到变化部分，通过抽象接口层来隔离变化，让核心部件功能单一化

展开

葫芦娃
2019-04-19

30

计算机专业第一课应该讲这个，然后才能知道学习算法，数据结构，编译原理，操作系统，网络通信的意义，都是核心能力的外围

MarksGui
2019-04-19

22

这不仅仅传授的是招式，这是一种意境！这样的举例分析确实让人眼前一亮。

展开

青石
2019-04-20

19

《倚天屠龙记》里有个很有意思的桥段，张三丰问了张无忌三次，第一次问：我刚刚演练的太极剑法你还记得多少呢？无忌答还记得一大半，过了一会儿张三丰又问：现在还记得多少呢？无忌答还记得一小半。再过了一会，张三丰又问：这次还记得多少呢？张无忌答全忘记了。张三丰神秘的一笑，说道：现在你可以和他们去打了。

...

展开

作者回复: 很通透

公号-代码...
2019-04-19

17

输入–计算–存储–输出
输入–存储–计算–输出
以上两种抽象的计算模型在我们的每个函数、组件、服务、分层设计中是一脉相承的。大道至简，我理解这既是架构设计的套路，也是架构设计的第一性原理

展开

fy
2019-04-19

16

赞赞，用这么浅显的语言阐释了冯若依曼体系结构的用来“解决了一切可以‘计算’的问题”，同时理清楚了冯若依曼架构的由来，计算，存储，输入输出。这三者之间的联系。通过多样变化的需求，抽象出来说明冯若伊曼哪部分是稳定的，哪部分是变化的！

苟范儿
2019-04-19

12

看到了数学公式 y = F(x)，突然发现这个中学就接触的简单公式蕴含的内容十分丰富。再看老师这么结合冯诺伊曼的体系结构，y = F(x) 实际上代表的是一种世界观，所有看见的（输入 x），都可以理解（函数 F）出新的东西（输出 y）。
另外，最初的编程课学的循环、条件、顺序几种结构，看似简单，却是程序的根基，不管有多少编程语言，这几种结构从来都没变过。

展开

vwarship
2019-04-19

12

在学校学习计算机原理的时候，感觉那么枯燥难懂，被老师以架构视角进行解读感觉超级简单，不过一下子还很难搞懂，接下来研究一下老师精心准备的代码，希望可以站在架构的角度搞懂一二。

展开

作者回复: 🤝

kyushu
2019-04-19

9

需求分析：站的角度很重要。架构设计也一样，总是提高一个level去看待，才能不局限于当前。

作者回复: 是的，架构设计切记见招拆招

crazier....
2019-04-19

9

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吗？

展开

作者回复: 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。

fy
2019-04-19

8

赞👍，用最简单浅显的语言来说明冯若依曼体系结构，说明解决计算

展开

上善若水
2019-04-19

7

力透纸背

展开

虫大侠
2019-04-20

6

道理不外乎如是, 从业一段时间的,基本可以明了其中的运转逻辑;
难点在于,对于业务的把控,有点像是中医的望闻问切一样,很难get到计算机科学的严谨性规律,可以一以贯之;
个人理解还是要踩坑, 经历过了, 思考过了之后, 有了切肤之痛之后形成的架构观;
觉得架构的技能, 有点过于依赖于经验的积累, 鲜少有速成的道路;...

展开

作者回复: 架构无速成之路，一靠匠心，二靠悟心。所谓悟心，反复梳理脉络后之顿悟哉。

Barry
2019-04-19

6

写的真好，真正理解的人才能用最浅显易懂文字来描述

展开

梦里
2019-04-19

6

艺术家应该窃取灵感，老师是让我们懂的如何窃取冯.诺依曼的灵感。厉害

展开

coderfocus
2019-04-19

6

写的也太棒了，谢谢老师

展开

燕羽阳
2019-04-20

5

推荐一本书《系统架构，复杂系统的产品设计与开发》
书中提到了 形式、功能和涌现等概念。
形式是物理组成，每个形式有自己的功能。这里对应文中的规格。
涌现是指系统整体展现的功能、性能等。应当是1+1>2。有点类似于解决文中的需求。

展开

作者回复: 多谢推荐

朱成亮
2019-04-19

5

架构第一关键点是理解和分析需求，弄明白需求中核心和可扩展；然后针对性的设计系统来解决这两个问题。

翅膀
2019-04-22

4

这是许老师想法的有损压缩，背后的事情n多。但能还原出来的估计没多少人。我是不行

作者回复: 有趣的说法，我后面第一章总结也会提到这一点

梦翼
2019-04-19

4

看完文章，想起这么一句话“事情不会在出现问题的层面得到解决，只有上升到更高的层面才会得到解决”。老师站在高处，为我们全方位，多角度地讲解了计算机体系。

展开