二

07 编程语言原理：面向对象编程是编程的终极形态吗？

软件架构师必须站在一个很高的高度去审视自己软件的架构，去理解自己的工作在更宏大的背景中的位置和作用，才能构建出一个经得起时间考验的软件系统。这个高度既包括技术的高度和深度，也包括对软件编程这件事认知的程度，比如对软件编程的历史和未来的理解，以及对自己工作的价值和使命感的理解。

计算机软件编程是个非常新兴的行业，程序员这一职业的出现不过半个多世纪，但是人类从事软件编程的探索却要久远得多，在计算机出现之前，甚至蒸汽机出现之前，人类就开始探索软件编程了。

最早开始编程探索的人是德国人莱布尼兹，早在1700年代，莱布尼兹就期望将各种事物都通过一种逻辑语言进行描述，然后用一种可执行演算规则的机器进行计算，就可以计算出事物的各种结果。这种思想其实和我们现代的软件编程与计算机已经差不多了，莱布尼兹为了实现这个想法，进行了大量的工作，获得了丰硕的成果，其中就包括了微积分和**二进制**。

但是人不能超越自己的时代，莱布尼兹制造可编程计算机的梦想并没有成功。又过了100年，法国人雅卡尔发明了一台可编程的织布机，这种织布机通过读取纸带上的打孔，进而控制织布机织出不同的图案。于是人们开始尝将**打孔纸带**用于计算机编程，19世纪中叶，当英国人**Ada**利用打孔纸带写出人类第一个软件程序的时候，距能够运行这个程序的计算机的发明还有100年的时间，而这个程序已经包含了**循环和子程序**。Ada因此被认为是人类第一个**程序员**，准确的说，是**程序媛**。科技发明受时代的限制，天才们的想象力和聪明才智却可以超越时代。

人类发明制造计算机器有非常悠久的历史，但是这些计算机器都是专门进行数值计算的，加减乘除、微分积分等等。而从莱布尼兹、Ada，到图灵、冯诺依曼，这些现代计算机的开创者们试图创造的是一种通用的计算机，这种计算机不是读取数值进行计算，而是读取**数据**进行计算，这些数据本身包含着计算的逻辑，这个数据就是**程序**。当冯诺依曼在ENIAC计算机上输入第一个程序的时候，标志着现代计算机的诞生，也意味着软件编程这一新兴的行业即将出现。信息时代、互联网时代接踵而至，人类开启了有史以来最大的一次科技革命。

现在我们编程已经习惯打开IDE，编写程序代码然后编译执行或者解释执行，认为编程就该如此。觉得那些不需要IDE，只需要写字板或者Vim就可以编程的人就是大牛了。事实上，

最早的计算机编程非常麻烦，程序员需要将电线编来编去，输入数据，以控制计算机的执行，这也是**编程**这个词的由来。不过很快人们就将打孔纸带应用到计算机上，编程的效率极大提升。

接近我们现在理解的软件编程要追溯到1949年，随着第一台可存储程序的计算机的发明而出现，程序员终于可以**写代码**了。这个阶段的程序要需要牢记**计算机指令**的二进制编码，软件开发就是直接使用这些二进制指令进行编程，每个计算机指令后面要带**操作数**，操作数也是二进制编码，所有这些二进制就是程序的代码，由程序员输入到计算机中。

现在的程序员们光是听听早期软件编程这一番神操作怕是就崩溃了，早期的程序员也意识到这一点，宝贵的时间不应该浪费在记忆计算机指令的二进制编码上，于是他们发明了**汇编语言**。和使用机器指令二进制编码唯一的不同就是，汇编语言提供了机器指令助记符，编程的时候，机器指令二进制可以用助记符代替。但是软件编程依然需要使用计算机指令，一个指令一个指令进行编程。因此，机器指令二进制编程和汇编语言编程本质上都是**面向机器的编程**。汇编语言程序如下，这已经是PC时代的汇编语言程序了，早期计算机的汇编程序要更加古老。

```
2000: BMI $2009       ;若结果为负数,那么转地址2009
2002: BEQ $200C       ;若 = 0,转 地址200C
2004: CLC             ;这里说明 > 0
2005: ADC #$01
2007: TAY
2008: RTS
2009: LDY #$01
200B: RTS
200C: LDY #$00
200E: RTS
```

在计算机出现的早期，即使对程序员而言，计算机也是一个神奇的存在，同一台计算机，可以进行科学计算，也可以进行弹道轨迹计算，还可以进行财务核算计算。计算机强大、神奇且昂贵，程序员匍匐在计算机的脚下，使用计算机的指令进行编程，面向机器编程。但是随着计算机技术的不断发展和计算机的普及，程序员们逐渐意识到，计算机本身呆板而机械，真正强大、无所不能的是软件程序。程序员为了更高效地进行编程，应该采用一种对程序员更加友好的编程方式，一种更接近人类语言的编程语言，于是各种各样的高级编程语言出现了。

最早的高级编程语言是Fortran，这是一种专门用于科学计算的高级语言，诞生于1957年。但是真正主流的、被广泛使用的各种高级语言则诞生于1970年前后，其中就包括**C语言**，传说丹尼斯·里奇发明了C语言，然后为了验证C语言的特性，开发了一个Demo，就是**Unix操作系统**。

那个年代美国正陷于越战的泥潭，大量的美国青年魂断东南亚的丛林，更多的美国青年则在国内聚集起来，集会、示威、游行，他们要独立、自由、和平，他们有的人背着吉他，从一

个城市流浪到另一个城市，而另一些人则坐在计算机终端前面，摆脱了对计算机指令的束缚，使用高级编程语言进行软件编程，用另一种方式表达独立和自由。这些高级语言使用人类语言作为编程指令，if…else…，while…break…，for…goto…，这些语句更符合**人类的习惯和逻辑思维方式**，由于这些语言关注逻辑处理过程，所以也被称作**面向过程的编程语言**。事实上，这些语言的本质是面向人的，因此这一时期爆发的各种编程语言本质上说是**面向人的编程语言，准确的说，是面向程序员的编程语言**。Basic编程语言示例：

```
INPUT "What is your name: ", UserName$
PRINT "Hello "; UserName$
DO
  INPUT "How many stars do you want: ", NumStars
  Stars$ = STRING$(NumStars, "*")
  PRINT Stars$
  DO
    INPUT "Do you want more stars? ", Answer$
  LOOP UNTIL Answer$ <> ""
  Answer$ = LEFT$(Answer$, 1)
LOOP WHILE UCASE$(Answer$) = "Y"
PRINT "Goodbye "; UserName$
```

高级编程语言的普及极大地释放了程序员的自主性，软件开发迎来黄金时期，程序员的第一个极客时代到来，比尔·盖茨、乔布斯都是在那个时代成长起来的。但是人的欲望是没有止境的，人能做到的越多，想得到的也就越多，越来越庞大的软件开发计划被不断地提了出来。

但是面向过程的复杂性随着软件规模的膨胀以更快的速度膨胀。面向过程的软件关注逻辑流程，更容易被设计成面条式程序，长长的过程调用执行，像一根面条。而大型项目最后由这样一根一根面条组成，就成了一个毛线团，最后谁也理不清了。于是很多大型软件的开发过程开始失控，最终以失败告终，人们遇到了**软件危机**。

软件危机使人们开始重新审视软件编程这件事情的本质，除了一部分科学计算或者其他特定目的的软件，大部分的软件是为了解决现实世界的问题，企业的库存管理、银行的账务处理等等。所以，**软件编程的本质是程序员用代码的方式使现实世界的事务运行在计算机上，计算机软件是为了解决现实世界的问题而开发出来的，那么软件编程这件事情应该关注的重点是客观世界的事物本身，而不是程序员的思维方式或者计算机的指令。**

如果软件编程的重点是客观世界的事物本身，那么编程语言如何才能更好地满足这一需求？于是，**面向对象的编程语言**应运而生。面向对象编程以对象作为软件编程的基本单位，提出**一切皆对象**，客观世界的用户、账号、商品是对象；创建、组合、关联这些对象的工厂、适配器、观察者也是对象；将所有这些对象分析、设计、开发出来，一个软件系统就完成了，这个软件系统灵活、强大，最重要的是可以根据需求变化快速更新维护。Java对象代码示例：

```
public class User {
    private String name;
    private Integer id;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
}
```

我们回顾一下现代编程技术的发展，发现大体经过面向机器编程，面向程序员编程，面向对象编程三个阶段，这正好对应马克思经济学关于劳动力的三个要素：劳动工具-计算机、劳动者-程序员、劳动对象-客观对象。编程从面向劳动工具进化到面向劳动者，再进化到面向对劳动对象。

面向对象编程似乎已经进化到编程这件事情哲学意义上的终点，是编程语言的终极形态。现实看起来也确实如此，最近三十年诞生的编程语言几乎全部都是面向对象的编程语言，面向对象一统天下。

但事实真的如此吗？回望历史我们站在上帝视角，一切都是如此清晰充满条理，凝望未来，我们还能如此笃定吗？

情况也许并非如此。事实上，现实中的面向对象编程几乎从未实现人们期望中的面向对象编程。上面举的Java的User对象示例就是典型，这是一个我们经常见到，却又非常不面向对象的对象。这个对象只有属性，没有行为，现实中的User对象显然不是这样。也许有部分企业和部分程序员做到了真正的面向对象编程，但是绝大多数程序员并没有做到，面向对象编程普及几十年了，如果大多数程序员依然做不到真正意义的面向对象编程，是程序员的问题还是编程语言的问题？

另一方面，一些新出现的面向对象编程语言对对象的态度似乎也有点暧昧，对象的边界和封装性开始模糊。go语言代码示例如下，这里NokiaPhone和iPhone都实现了Phone接口，但是并不是显式的。

```
type Phone interface {
    call()
}
type NokiaPhone struct {
}
func (nokiaPhone NokiaPhone) call() {
```

```
    fmt.Println("I am Nokia, I can call you!")
}
type IPhone struct {
}
func (iPhone IPhone) call() {
    fmt.Println("I am iPhone, I can call you!")
}
```

而随着科技的不断发展，特别是大数据，人工智能以及移动互联网的发展，**面向数据**的编程需求越来越多，能够更好迎合这一需求的编程模型开始得到青睐，比如**函数式编程**。而极客型的程序员对强类型的面向对象编程越来越不感冒，他们希望在编程的时候能够得到更多的自由，编程语言的重心似乎重新出现面向程序员的趋势。

随着计算机性能的不断增强，以及互联网应用对计算资源需求的不断增加，如何更好地利用CPU的多核以及分布式集群的多服务器特性，必须是软件编程以及架构设计时需要考虑的重要问题，软件编程越来越多需要考虑机器本身，相对应的，**反应式编程**得到越来越多的关注。

辩证唯物主义告诉我们，事物发展轨迹是波浪式前进，螺旋式上升，有的时候似乎重新回到过去，但是却有了本质的区别和进步。软件编程的进化史还在继续，你是否对未来充满期待和信心？

小结

今天我们回顾了编程技术的发展，通过这样的脉络梳理，你能更清楚目前面对对象编程的来源，更好地利用这一技术。如何利用面向对象编程的特性，进行真正的面向对象编程，而不是仅仅利用面向对象编程语言进行编程，我将在第16篇讲解。

思考题

不同的编程语言在不同的应用场景中，各有自己的优势和劣势，你觉得哪些编程语言更适合用在哪些地方，适合处理哪些问题？

欢迎在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事，一起交流进步一下。

上一页 下一页