第 分	理解	OOP: 编程语言的历史	29
5	3.1	OOP 的出现具有必然性	31
	3.2	最初使用机器语言编写程序	31
	3.3	编程语言的第一步是汇编语言	32
	3.4	高级语言的发明使程序更加接近人类	33
	3.5	重视易懂性的结构化编程	34
	3.6	提高子程序的独立性,强化可维护性	35
	3.7	实现无 GOTO 编程的结构化语言	38
	3.8	进化方向演变为重视可维护性和可重用性	39
	3.9	没有解决全局变量问题和可重用性差的问题	41
专栏	编辑	程往事	
,	СО	BOL编译器的鸡和蛋的问题	45
第	面向	对象编程技术:去除冗余、进行整理	47
4	4.1	OOP 具有结构化语言所没有的三种结构	49
	4.2	OOP 的结构会根据编程语言的不同而略有差异	51
	4.3	三大要素之一: 类具有的三种功能	51
	4.4	类的功能之一: 汇总	52
	4.5	类的功能之二: 隐藏	55
	4.6	类的功能之三: 创建很多个	58
	4.7	实例变量是限定访问范围的全局变量	61
	4.8	三大要素之二: 实现调用端公用化的多态	63
	4.9	三大要素之三: 去除类的重复定义的继承	67
	4.10	对三大要素的总结	70
	4.11	通过嵌入类型使程序员的工作变轻松	71
	4.12	将类作为类型使用	72
	4.13	编程语言"退化"了吗	74
	4.14	更先进的 OOP 结构	74
	4.15	进化的 OOP 结构之一:包	75
	4.16	进化的 OOP 结构之二: 异常	76
	4.17	进化的 OOP 结构之三: 垃圾回收	78
	4.18	对 OOP 进化的总结	80

81

4.19 决心决定 OOP 的生死

第二〇	理解	4内存结构:程序员的基本素养	83
4			
章	5.1	理解 OOP 程序的运行机制	85
	5.2	两种运行方式:编译器与解释器	85
	5.3	解释、运行中间代码的虚拟机	88
	5.4	CPU 同时运行多个线程	89
	5.5	使用静态区、堆区和栈区进行管理	91
	5.6	OOP 的特征在于内存的用法	94
	5.7	每个类只加载一个类信息	95
	5.8	每次创建实例都会使用堆区	96
	5.9	在变量中存储实例的指针	97
	5.10	复制存储实例的变量时要多加注意	99
	5.11	多态让不同的类看起来一样	103
	5.12	根据继承的信息类型的不同, 内存配置也不同	105
	5.13	孤立的实例由垃圾回收处理	107
专栏	编	程往事	
	00	P中dump看起来很费劲?	113
⁹ 6	重用	1: OOP 带来的软件重用和思想重用	115
章	6.1	OOP 的优秀结构能够促进重用	117
	6.2	类库是 OOP 的软件构件群	119
	6.3	标准类库是语言规范的一部分	120
	6.4	将 Object 类作为祖先类的继承结构	121
	6.5	框架存在各种含义	122
	6.6	框架是应用程序的半成品	123
	6.7	世界上可重用的软件构件群	124
	6.8	独立性较高的构件: 组件	125
	6.9	设计模式是优秀的设计思想集	125
	6.10	设计模式是类库探险的路标	128
	6.11	扩展到各个领域的思想的重用	129
	6.12	通过类库和模式发现的重用的好处	130

第二〇〇	化为	通用的归纳整理法的面向对象	133
010			
の一章	7.1	软件不会直接表示现实世界	135
	7.2	应用于集合论和职责分配	137
	7.3	在上流工程中化为通用的归纳整理法	139
	7.4	两种含义引起混乱	140
	7.5	分为 OOP 的扩展和归纳整理法进行思考	141
	7.6	为何化为了通用的归纳整理法	142
专栏	对领	象的另一面	
	语言	言在先, 还是概念在先	143
第丘〇	UM	L:查看无形软件的工具	145
S		H 2010 WILLHAM	
() (章	8.1	UML 是表示软件功能和结构的图形的绘制方法	147
	8.2	UML 有 13 种图形	148
	8.3	UML 的使用方法大致分为三种	150
	8.4	UML 的使用方法之一:表示程序结构和动作	151
	8.5	类图表示 OOP 程序的结构	151
	8.6	使用时序图和通信图表示动作	154
	8.7	UML 的使用方法之二:表示归纳整理法的成果	156
	8.8	使用类图表示根据集合论进行整理的结果	157
	8.9	表示职责分配的时序图和通信图	160
	8.10	UML 的使用方法之三:表示非面向对象	163
	8.11	使用用例图表示交给计算机的工作	163
	8.12	使用活动图表示工作流程	164
	8.13	使用状态机图表示状态的变化	165
	8.14	弥补自然语言和计算机语言缺点的"语言"	166
第二〇	建模	: 填补现实世界和软件之间的沟壑	171
9	9.1	现实世界和软件之间存在沟壑	173
	9.1	计算机擅长固定工作和记忆工作	173
	9.3	通过业务分析、需求定义和设计来填补沟壑	175
	9.4	建模是顺利推进这3个阶段的工作的技术	176
	9.5	应用程序不同,建模的内容也不一样	177
	0.0	WIND THE PROPERTY OF THE PARTY	1.7.3

	9.6	业务应	范用程序记录现实中的事情	178
	9.7	对图书	馆的借阅业务进行建模	179
	9.8	使用用例图来表示图书馆业务		
	9.9	用概念	校型表示图书馆系统的信息	183
	9.10	业务	应用程序中只有数据是无缝的	184
	9.11	嵌入	式软件替换现实世界的工作	186
	9.12	嵌入	式软件中设备的研究开发很重要	187
	9.13	使用	状态机图来表示全自动工作的情形	189
	9.14	嵌入	式软件一直执行单调的工作	190
	9.15	建模	蕴含着软件开发的乐趣	191
# 10	C	面向	对象设计: 拟人化和职责分配	195
000	章	10.1	设计的目标范围很广	197
		10.2	相比运行效率, 现在更重视可维护性和可重用性	198
		10.3	设计目标之一: 去除重复	199
		10.4	设计目标之二: 提高构件的独立性	200
		10.5	提高构件独立性的诀窍	202
		10.6	设计目标之三: 避免依赖关系发生循环	203
		10.7	面向对象设计的"感觉"是拟人化和职责分配	205
		10.8	进行了职责分配的软件创建的奇妙世界	206
#	0	衍生	: 敏捷开发和TDD	211
0500	章	11.1	仅靠技术和技术窍门, 软件开发并不会成功	213
		11.2	系统地汇总了作业步骤和成果的开发流程	214
		11.3	限制修改的瀑布式开发流程	214
		11.4	瀑布式开发流程的极限	215
		11.5	灵活响应变化的迭代式开发流程	216
		11.6	RUP按时间分解和管理开发	217
		11.7	打破诸多限制的XP	219
		11.8	快速编写优秀软件的敏捷宣言	221
		11.9	支持敏捷开发的实践	222
		11.10	先编写测试代码,一边运行一边开发的测试驱动开发	222
			在程序完成后改善运行代码的重构	224

	11.12	经常进行系统整合的持续集成	225
	11.13	敏捷开发和TDD源于面向对象	226
	11.14	不存在最好的开发流程	227
专栏 经	程往事		
过	去不被	允许的XP	231
	孰练	掌握面向对象	233
120	Maranto	and the first of his	
-	12.1	面向对象这一强大概念是原动力	235
	12.2	时代追上了面向对象	236
	12.3	面向对象的热潮不会结束	237
	12.4	将面向对象作为工具熟练掌握	238
	12.5	享受需要动脑的软件开发	239
# A S	函数	式语言是怎样工作的	241
15			
の与うで意	13.1	面向对象的"下一代"开发技术	243
	13.2	函数式语言的7个特征	244
	13.3	特征1: 使用函数编写程序	244
	13.4	特征2: 所有表达式都返回值	246
	13.5	特征3: 将函数作为值进行处理	250
	13.6	特征4: 可以灵活组合函数和参数	252
	13.7	特征5: 没有副作用	256
	13.8	特征6: 使用分类和递归来编写循环处理	261
	13.9	特征7: 编译器自动进行类型推断	266
	13.10	对7个特征的总结	270
	13.11	函数式语言的分类	271
	13.12	函数式语言的优势	271
	13.13	函数式语言的课题	272
	13.14	函数式语言和面向对象的关系	273
	13.15	函数式语言会普及吗	275
	后记		279
	致谢		280

本章的关键词

第二章

术语洪流、比喻滥用、<mark>"一切都是对象"</mark> 综合征

面向对象: 让软件开发变轻松的技术

热身问答

在阅读正文之前,请挑战一下下面的问题来热热身吧。



下列哪一项是最早提出"面向对象"概念的艾伦·凯(Alan Kay)的表述?

- A. 程序模块、图标和数据库等万物都可以表示为对象
- B. 正如万物都在变化,编程技术也在变化
- C. IT 领域的创新技术基本上都出现于 1960 年之前
- D. 预测未来最好的方法就是创造它



D. 预测未来最好的方法就是创造它

面向对象的起源可以追溯到挪威的两名技术人员在 1967 年 开发的 Simula 67 编程语言。之后,任职于美国施乐公司的艾伦·凯率领的团队开发了 Smalltalk,沿用了 Simula 67 语言的结构,确立了面向对象的概念。除此之外,凯在 IT 领域还做出了很多贡献,比如开发出图形用户界面 (Graphical User Interface,GUI)、提出作为现代笔记本电脑原型的"DynaBook 设想"等。

"预测未来最好的方法就是创造它。" (The best way to predict the future is to invent it.) 据说这句名言是公司高层追问研究内容的未来走向时,凯给出的回答。想必也只有提出了诸多创新性的技术概念的凯,才能讲出这样的名言吧。

本章 重点

本章将介绍面向对象的基本思想,以及面向对象所面 向的技术领域的全貌。

面向对象最初是作为一种编程语言提出的,后来人们将其不断扩展,并应用到各个领域,如今将其称为"软件开发的综合技术"可能更为合适。

遗憾的是,尽管这是一门非常优秀的技术,但很多从事软件开发的人好像都认为它非常难。本章我们将分析造成这种局面的3个主要因素,而消除这些因素正是正确理解面向对象的关键。

🧰 1.1 面向对象是软件开发的综合技术

我们先从一个简单的问题开始介绍。

"为什么要基于面向对象来开发软件?"

不管谁问这样的问题, 笔者都会这样回答:

"为了轻松地开发软件。"

可能有的人听到"轻松"二字会感觉很意外。这是因为当提到面向对象时,不少人仍感觉"很难,难以对付"。

面向对象包含了各种技术,几乎涵盖了从 Java、Ruby 等编程语言到需求规格书和设计内容的图形表示、可重用的软件构件群、优秀设计的技术窍门、业务分析和需求定义的有效推进方法、顺利推进系统开发的开发方法等软件开发的所有领域。

不过,这些技术单独来看是完全不同的。如果要找出它们的共同点, 大概就是它们都是软件开发相关的技术,都是用来顺利推进软件开发的。

因此、如果用一句话来概括面向对象,那就是"能够轻松地进行较难的软件开发的综合技术"。

1.2 以对象为中心编写软件的开发方法

面向对象的英文是 Object Oriented, 直译为"以对象为中心"。

在面向对象普及之前,主流的开发方法是"面向功能"的,具体地说,就是把握目标系统整体的功能,将其按阶段进行细化,分解为更小的部分。如果采用面向功能的开发方法来编写软件,当规格发生改变或者增加功能时,修改范围就会变得很广,软件也很难重用。

面向对象技术的目的是使软件的维护和重用变得更容易,其基本思想 是重点关注各个构件,提高构件的独立性,将构件组合起来,实现系统整 体的功能。通过提高构件的独立性,当发生修改时,能够使影响范围最小, 在其他系统中也可以重用。

1.3 从编程语言演化为综合技术

面向对象最初是以编程语言的身份出现的,在之后的40多年里,经过不断发展,逐渐被应用到了开发的各个领域。这里我们来简单回顾一下面向对象的全貌和发展过程,如图1-1所示。

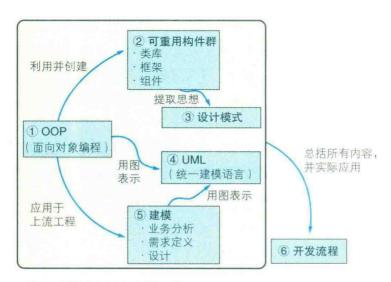


图 1-1 面向对象的全貌和发展过程

面向对象起源于 1967 年在挪威设计的 Simula 67 编程语言。该语言拥有类、多态和继承等以往的编程语言中没有的优良结构,被称为最早的面向对象编程语言 (Object Oriented Programming language, OOP^{-1})。后来,艾伦·凯率领的团队开发的 Smalltalk 沿用了该结构,确立了"面向对象"的概念。此后,具有相同结构的 C++、Objective-C、Java、C# 和 Ruby 等诸多编程语言相继被开发出来,并延续至今。

OOP 使得大规模软件的可重用构件群的创建成为可能,这些被称为类 库或者框架。另外,创建可重用构件群时使用的固定的设计思想被提炼为 设计模式。

另外,使用图形来表示利用 OOP 结构创建的软件结构的方法称为统一建模语言(Unified Modeling Language, UML)。在此基础上,还出现了将 OOP 思想应用于上流工程的建模,以及用于顺利推进系统开发的开发流程。

如今,面向对象已经成为一门覆盖软件开发整体的综合技术。虽然这些技术所涉及的领域和内容并不相同,但目的都是顺利推进软件开发。因此,通过以各种形式灵活运用前人的研究和技术窍门,有助于我们提高软件的质量和开发效率。

1.4 在混乱的状态下去理解,就会觉得很难

尽管面向对象是众多优秀技术的集大成,却经常被认为很难理解,难以对付。也有人认为不擅长抽象思考的人在学习面向对象时会感觉很难,要经过很多年才能掌握,等等。不管是多么方便的工具,如果很难理解其内涵,无法熟练使用,那就没有办法了。

不过, 笔者认为, 相较于技术本身的复杂性, 面向对象让人感觉很难

① 严格来说,正确的表达方式是,将面向对象编程语言(Object Oriented Programming Language)称为OOPL,使用OOPL进行编程的操作称为面向对象编程(OOP)。不过,由于本书的很多讲解中并未严格区分面向对象编程语言和面向对象编程,所以都表述为OOP。

的更主要的原因在于混乱的现状。造成混乱的主要原因大致有三点:术语洪流、比喻滥用和"一切都是对象"综合征。下面我们就来分别介绍一下。

1.5 混乱之一: 术语洪流

第一点是术语洪流。

想必很多人在最开始接触面向对象时都是像图 1-2 那样,被灌输大量 陌生的术语吧。下面列举了一些具有代表性的术语,但其实人们被灌输的 术语远不止这些。

继承、泛化、特化、超类、子类、接口、多重继承、属性、关联、集合、委托、重写、重载、访问控制、构造函数、包、异常、垃圾回收机制、框架、类库、组件、设计模式、用例、建模、UML、重构、敏捷开发流程、RUP、XP······

系统是由汇总了数据和过程的"对象"组成的。在面向对象中,软件被定义为"类",然后创建"实例"并运行。系统是通过"实例"之间互相交换"消息"而运行的,但由于进行了"封装",所以无法查看内部的详细内容,这被称为"信息隐藏"。当发送"消息"时,并不会在意对方"实例"是哪一个"类",这被称为"多态"



图 1-2 对大量术语感到混乱的开发者

简直就是术语大集合! 在深入了解技术内容之前就被这么多术语吓退的人应该不在少数吧。

存在如此大量的术语的原因如下所示。

首先是面向对象所涉及的范围很广。如今面向对象几乎涵盖了软件开发的所有领域,在这些领域中,之前没有的新结构和新思想层出不穷。这些术语中也有不少英文词汇,可以说这是以欧美为中心迅速发展起来的技术的宿命。此外,还有一些词语被作为业界的宣传用语而被过度解释。像这样,术语多少本身源自该技术的广度,从一定意义上来说也是没有办法的事情。

而更本质的问题是基本术语的定义混乱,我们将在随后的第三点中对 此进行讨论。

🦲 1.6 混乱之二:比喻滥用

第二点是比喻滥用。

这与其说是技术本身的问题,倒不如说是说明方法的问题。一般来说, 比喻并不一定就是不好的。使用恰当的比喻能够直截了当、形象地说明内容;反之,如果比喻不恰当或者使用过度,就有可能造成混乱。尤其是在 仅通过比喻来说明编程语言等的具体结构的情况下,由于每个人的理解各 不相同,所以特别容易造成误解。

我们来举例说明一下。面向对象的基本结构有时会像下面这样说明 (图 1-3)。

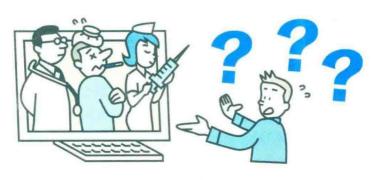


图 1-3 对使用比喻进行的说明感到混乱的开发者

"动物是超类,哺乳类和鱼类是子类。既产卵又用乳汁哺乳幼仔的鸭嘴兽也就相当于爬虫类和哺乳类的多重继承。"

"人具有'出生年月日'的属性。如果给小王这样具体的一个人发出'请告诉我你的年龄'的消息,就会得到'28岁'的回答。"

"正如医院里的医生、护士和药剂师互相联系、协同工作一样, 对象也是通过在计算机中互相发送消息来进行工作的。"

从直观理解 OOP 的结构来说,这样的讲解也不一定就是不好的做法。不过,如果只强调比喻而不详细介绍编程语言的结构和目的,那么就只有比喻能给人留下深刻的印象,而且听众很有可能会根据自己的理解,对实际的结构产生错误的认识。特别是在不考虑系统化范围,即计算机的作业范围的情况下乱用比喻,再加上随后介绍的"一切都是对象"综合征的影响,读者就有可能会产生"当使用面向对象时,可以直接将现实世界表示为程序"的重大误解(关于该误解,我们将在第2章深入讨论)。

1.7 混乱之三: "一切都是对象"综合征

第三点是"一切都是对象"综合征。

面向对象的含义是"以对象为中心"。如果按照字面意思进行解释,那么现实世界的人、组织、事件、计算机系统的功能、系统管理的信息和程序的构成元素等一切事物都可以说是对象。这里,我们将这种极端的抽象称为"一切都是对象"综合征(图 1-4)。

这种极端看法大概源自"万物都是变化的"这一点。立足于这种观点是很难说明和理解编程语言的结构,以及现实世界中业务流程的整理方法等具体技术的。尽管如此,但就像"正如现实世界是由对象(物)组成的,在面向对象中,程序也是以对象为中心创建的"这句话所说的那样,我们经常会见到实际上不同的对象都被解释为"对象"这一个词语的情况。这样的解释的确会对直观理解面向对象的结构有所帮助,但反过来也有可能引起"只要使用面向对象,就可以直接将现实世界表示为程序"的误解。



图 1-4 "一切都是对象"综合征

不过,这种混乱并不只是说明方法的问题,也是面向对象本身的问题。 这是因为面向对象的技术范围很广,比如对象和类等词语有时指编程语言 的结构,但在其他情况下也指管理的信息、现实世界的人和物。

笔者认为,"一切都是对象"综合征才是造成面向对象混乱的最大原因 (关于这一点,我们将在第7章中深入讨论)。

1.8 三种混乱增大了理解的难度

在很多情况下,以上为大家介绍的三点引发混乱的原因,即术语洪流、 比喻滥用和"一切都是对象"综合征会同时出现。也就是说,在讲解面向 对象时,讲解的人往往会一下子介绍很多陌生的术语,并以比喻为中心, 只强调现实世界和程序结构的共同之处。然而,这样的讲解非但不能让人 正确地理解该技术,反而会容易引起混乱和误解。

由于面向对象以范围很广、难度较大的软件开发为对象,所以在技术 层面上原本就非常复杂,而以上几点混乱又进一步增大了理解该技术的难 度,使其本质难以被看透。

■ 1.9 因为不理解,所以才感觉神秘

不理解的事情有时会为我们带来意外的惊喜。

如果某项技术只是比较难,让人无法理解,那么放弃学习也没什么, 而面向对象所涉及的编程语言、UML以及大规模的可重用构件等技术都很 基础,而且这些技术正在不断地渗透到软件开发的实际应用场景,所以现 阶段很难完全避开面向对象。

"虽然将现实世界直接表示为程序这种解释我怎么都想不通,但是由于 Java 和 UML 使用起来比较方便,所以我正在使用。""虽然其中的个别技术非 常有用,但是整体上却很难理解。"……想必抱有这些想法的人不在少数吧。

面向对象有时会被比作魔法,虽然无法用道理解释清楚,但效果很好,或者被解释为一般人难以触及的理想的开发方法。这是将"不理解但有用的事物"神秘化,并推崇为"超越人类智慧的存在"的心理在起作用。

不过,面向对象的目的是驱动逻辑电路组成的计算机,是具有实践性的一门极为有用的技术,因此完全不存在无法解释的内容

🧖 1.10 消除这三种混乱,就能看到面向对象的真面目

本书的目标是用清晰的逻辑向读者介绍面向对象所涉及的各项技术。

不过,本书并不会为大家详细讲解 Java 等编程语言的语法、UML 表示法等各项技术的详细内容,而是重点讲解这些技术是什么(What),以及为什么需要这些技术(Why)。

另外,为了使读者能够正确理解面向对象,笔者认为应该首先排除前 文中介绍的引发混乱的三点原因,因此本书将采取如下方针。

- ◈将对术语和概念的介绍控制在最小限度
- * 将最小限度地使用比喻进行讲解。如果使用比喻,则会明确其主旨
- 将编程的结构以及"以对象为中心"来把握事物的思想作为不同的 内容分开介绍