从语言进步到编程

殷 明 慧

ymhui@21cn.com

在培训的授课阶段即将结束之际，趁脑子还比较清楚，总结一下课堂上向大家灌输的编程思想。现在看不太懂也没关系，等大家积累了一定的开发经验和编程思考，再回过来头来看看这些内容。或许你会有种“蓦然回首，那人却在灯火阑珊处”的感觉。由于时间匆忙，文中讲述不恰当的地方还请各位读者拍砖。

1. 语言和编程本是两码事

分不清语言和编程的人，估计很能学得好编程。

1.1 盲从技术只能让你晕的更久

和其它事物一样，软件编程的发展也经历了从低级到高级的各个阶段。从机器代码到汇编语言，从汇编语言到高级语言，从高级语言到封装完善的编程框架，以后肯定还会出现更多高级的语言或是编程框架。

作为学习者，在初涉软件编程领域时，常常被众多的语言搞昏了头脑，被众多的编程框架折腾的晕头转向。在找不到软件编程规律之前，他们只能跟着所谓新技术走。正所谓 C 不流行了，要去学 C＋＋；C＋＋不流行了，赶着学 JAVA；MFC 框架封装的不好，赶快去用.NET 框架吧；还有太多的太多的名词，太多太多的中间件技术。某天在与一位程序员聊天时，他感言到：“现在的编程，就是数据库和网络，其它的都没什么用!!”。是啊，连有经验的程序员都这么说，更何况那些没有编程经验的初学者。他们唯一能做的就是，什么新学什么，什么好学什么。到底，什么是新，什么是好？谁也说不清楚，倒是网上总有类似的大讨论，比较 C＋＋和 JAVA 的先进性，比较 VC＋＋和 BCB 优劣性。在这里，我只想告诫同学们，盲从只能让你晕得更长。

1.2 语言和编程其实两回事情

那么什么是编程？什么是好的编程？如果同学们能理解了这两句，我想你会发觉盲从编程新技术是一件很可怕也很可笑的事情。编程说到底就是向计算机说明一件事情，让计算机

按照你的意图去做这件事情。事情的复杂性决定了说明的难度，即决定了编程的难度。编程的好坏其实就是说明事情的水平高低，如果你把事情说明的简明、扼要，而且还能为将来留有余地，那么你的编程就是好的。

语言又是什么呢？语言是人们与计算机交流的工具，人们借助编程语言来描述和说明要安排给计算机的事情。可见，语言是死的，编程是活的。前几日在网上发现一帖，把语言比作兵器，一个很生动形象的例子。对于不会编程的人，给他再好的兵器都是白搭；对于会编程的人，给他一把好兵器那是如虎添翼。《VC＋＋编程技术与难点剖析》一书把语言比作猎枪，试想给你一把上等猎枪，如果你不会打猎的话，一样是一无所获。

可见，语言是形，编程是意；编程是语言的内在精神，语言是编程的外在载体。你把语言规范学得再好，如果不会编程的话，你依然会觉得茫然不知所措。当你领悟到编程思想时，你会发觉，其实用C语言也能写出面向对象的程序[1]。

1.3 语言靠学、编程需悟

回首十几年的英语学习，我不禁汗颜，除了背得一些单词，记住一些语法外，我张口还是说不出一句完整的英语句子来。一个深刻的体会便是就是不知道该怎么说英语。我想大部分同学对于软件编程的感受也是类似的，给你一个编程要求，即便你掌握了所有的 C＋＋语言规则，你还是觉得不知道该怎么编，不知道该从那里下手。

正如前面所述，语言是死的、是形。它是由一系列关键字和语法规则组成。掌握这些关键字和语法规则并不需要太长时间，因为这些内容远比英语单词要少的多的多。这部分内容是容易讲授，也容易被学生掌握。

关键字和语法规则是死的，但是如何灵活运用这些规则、综合这些规则确是活的。而这些正是编程真实含义所在。因此，运用语法规则描述一件事情其实非常复杂，实现功能只是最基本的，之上还有运行效率问题、扩展性问题、可移植性问题等等。同学们在学习编程时，一是需要老师在课堂上引导，但老师的引导只能是启发大家的思维，激发大家的思考；更多的还是今后在实际工作中的经验积累和不断的思考。不管是自己写代码，还是看别人代码，你只要思考清楚如下几个问题就可以了：

1）为什么要这么设计代码，这样设计有何优点；

2）导致这些优点的原因是什么；

---

[1] 网上有“C语言的面向对象编程”的资料，大家可去看看。目的不是真的让大家用C语言写OOP程序，那样着实很累；而是让大家体会一下思想和语言确实是不同的东西。

3）把优点总结出来，尝试在自己的程序中去尽量运用这些优点[2]；

4）这些优点能否再改进，并如何改进。

如果能坚持能做到前三点，那已经相当不得了了。最后一点是为大师级人物准备的，说不定在读某位未来可能就是××公司的首席架构设计师哦。

2. 在使用语言中进化编程思想

从语言到编程，从编程到语言 —— 语言学习和使用的过程是编程思想的积累过程，当你掌握思想之后，任何的语言形式都能体现你的思想精华。所以，首先把语言学习好吧。你真的掌握一门语言了吗？看看下面的文字再下结论不迟。

2.1 用低级语言去理解高级语言

低级语言是高级语言的基础，当我们对于一个高级语言的知识点难以把握时，最好的办法、也是最有效的办法就是去低级语言中寻求规律、寻求答案、寻找原则。这里我想举个例子，即指针的理解和使用。

指针可以说是一个没有完全完成从汇编语言到高级语言进化的变量，它是一个介于汇编语言和高级语言的东东。而 C 语言也是因为它，才变得更加的灵活强大，同时也让许多初学者望指针生畏。指针变量保存的是内存地址，地址访问体现了低级语言的特征；指针变量具有类型，变量类型体现了高级语言的特征。不要以为你知道指针是个内存地址，你就能从汇编语言上理解指针、使用好指针。对于指针的理解和使用还必须注意如下几个方面：

1）可执行程序中的什么元素会占用内存空间？与语言代码相关的，一般是数据和函数执行代码，这导致变量指针和函数指针概念的出现；

2）可执行程序是如何开辟和释放内存空间的？在C语言中包括局部变量、全局变量、静态变量、malloc/free函数对、结构体、联合体等等，到了C＋＋语言又增加了对象（包括成员变量、静态成员变量、普通成员函数、虚拟函数和虚拟函数表）、new/delete操作符对，在Object Pascal语言中甚至还有类[3]。可见，只要是语言中使用到的系统资源（包括变量和函数）都需要关心它在内存中的开辟和释放机制。

3）只有在掌握 2）的基础之上，你才能准确的知道代码中的指针是否指向存在的或是

---

[2] 这其实就是编程模式

[3] 在C＋＋语言中，是不会对类分配内存空间的。但是在Object Pascal中，即Delphi，为了更加方便的实现RTTI（Run-Time Type Identification）机制，引入了元类的概念，为类定义分配内存空间。这一做法实在是高明。

未被释放的内存空间，而这是正确使用指针的重要原则之一。

4）如果是变量指针，指针类型对于四个字节的内存地址而言毫无意义。它的引入，不过是在告诉 C/C++语言编译器，当代码对指针进行加减操作时，它该如何以多大的步长增减地址；当代码对指针进行取值（*,->）操作时，它该以多大的位移取多大内存空间的值。仅此而已。

5)如果是函数指针，指针类型对于内存地址也是毫无意义的。它的引入，是在告诉 C/C++语言编译器，当代码使用指针调用函数时，它该如何为函数调用生成汇编代码。函数的调用会涉及一些堆栈或是寄存器操作，而不同的输入、输出参数对应的这些操作是不相同的。

当我们分析清楚可执行程序的内存开辟和释放机制时，当我们从编译器的角度去理解指针类型的含义时，可以自然的总结如下的指针使用原则：

1）保证对指针取值时，指针指向的内存空间是有效的

2）存储指针时，把它当作占四个字节的任何变量，指针指向的类型可以是任何变量类型。

3）对指针取值操作时，确保恰当的指针类型的转型。

讲到这里，请大家再去理解以前课本中反反复复提到的传值和传地址的区别。其实把指针参数理解为传地址不过是便于理解吧。如果你从汇编角度理解，传值和传地址的操作是完全一样。当你从指针本身的内容来看，传地址就是在传递指针的地址值，就是传值，和传递个 int 变量毫无区别；当你从指针所指向的内容来看，就是一般书中所讲述的传地址。

高级语言便于理解、易于掌握，也大大提高了程序员的编程效率。但是，如果我们仅停留在高级语言层面上，不去究根问底，不去追根溯源，那我们只能算是浅尝辄止。

其实，“用低级语言去理解高级语言”也意味着用底层的知识去理解上层的东西，用基础的知识去把握基础之上的东西。好比用汇编语言去理解高级语言、用 Win32、OOP 和 C++去理解 MFC 框架、用 Win32、OOP 和 Object Pascal 去理解 VCL 框架、用 Windows Socket、OOP 和 C++去理解 VC++中的网络编程技术、用 Windows Socket、OOP 和 Object Pascal 去理解 Delphi 中的网络控件。诸如此类的例子真是举不胜数。

请记住，当你站在优秀平台上舞刀弄棒、学个一招半式时，千万别沉浸于表面上的成功。试想，当你用 BCB 的几个数据库控件成功访问数据库的时候，有多少东西是你自己做的，你又学到了多少东西。可以说，你所学到的不过是类似于打字操作、文档编辑之类的简单操作而已。

任何高级的编程技术或是平台都不是空穴来风的。努力把它们的基础和结构分析清楚，

你至少可以得到如下两点好处：

1）对于新技术或新平台的全面掌握和灵活运用，正所谓庖丁解牛，恢恢呼而游刃有余。

2）从深入分析和探索中获取编程的思想，正所谓深入成就深度。

2.2 体会语言设计者的初衷

正如上面所述，任何新的编程技术或平台都不是空穴来风，更不是某位天才一拍脑袋就想出来的。创新肯定是有源动力的，新技术肯定是为解决既有技术的不足才出现的。说白了创新就是为了更高、更快、更强。试想，如果你能把握住新技术的来龙去脉，理解设计者的良苦用心，那你就可以把这项新技术在恰当的时间、恰当的地点、以最恰当的方式使用起来。在此我也想举个例子，即虚函数和动态函数[4]。

虚函数是C++语言中的重要概念之一。简单的说，虚函数因多态而生，多态因抽象统一接口（接口可以理解为方法）而起。虚函数出现的根源就是抽象统一的接口。

抽象是人类探索、描述客观世界的利器。如何把纷繁复杂、变化多样的各种事物描述清楚，唯一的方法就是抽象。语言也不例外。水果、衣服、车、食物、粮食等等，这些都是抽象出来的名词，正如在课堂中所说，“吃水果可以吃尽天下所有的水果，不管是已发现的，还是未发现的”。抽象抓住了事物的本质与共性。保证了相对的稳定性，实现了以不变应万变的强大功能[5]。可见，只有抽象才能**统一接口**。

通过抽象，可以把各种各样事物的接口都抽象成一个接口。例如，可以把吃苹果、吃梨子、吃菠萝、吃西瓜等等都抽象成吃水果。但是，在真正使用抽象接口时，必须把抽象接口还原到具体事物的真实接口中去。例如，水果是抽象的，没法吃，吃水果这个抽象方法一定要还原到吃具体水果中去才有意义。这样，一个抽象统一的接口，却有许多种具体的表现形式，这便是是**多态**。

当我们编写一个类库时，如果能尽可能多的使用抽象的思想去统一类库（或是各个子类）的接口，那么至少有如下几点好处：

1）对于使用者，由于接口简单，大大简化了学习类库和使用类库的工作；

2）对于使用者，由于接口统一，相对稳定，使用者编写出来的代码具有很强的扩展性，即便在今后类库又派生出新的子类，使用者编写的代码也无需作任何修改，因为接口是统一的，相对固定的。

---

[4] 动态函数是Delphi中提出的，BCB中也支持，为此多了一个关键字Dynamic。

[5] 引自《Delphi模式编程》 刘艺 44页

3）对于设计者，由于应用了抽象，类库的层次清晰。当派生新的子类时，父类已经划定了抽象的框框，按照既有框框实现具体接口即可。

试想，如果能够理解虚函数的设计初衷是为了抽象统一接口，那么在父类设计中该如何设计虚函数也就把握住了最根本的原则，即能够从多种对象中抽象统一出来，且每个对象的具体实现又都不同的方法，都设计为虚函数。

至此，文章已经分析了 C＋＋引入虚函数的根本原因。但此后，在 BCB 中，Borland 公司的天才们又设计出能够完全实现多态机制的另一种方法，动态函数。有心人不禁要问，既然虚函数可以实现多态机制，干吗还要引入动态函数呢？这两个函数有什么区别呢？孰优孰劣呢？引入动态函数的初衷又是什么呢？

呵呵，疑问总是探索未知领域的源动力。而疑问意识并不是每个人都具备的，越是大家，越是对习以为常的事物产生疑问。好比牛顿被苹果砸了脑袋，这引发了他研究万有引力。我想，绝大部分人被苹果砸到吐血都不会产生类似疑问。思维定式和传统教育的灌输扼杀了绝大部分人的疑问意识，也就同时扼杀了他们的探索、创新意识。话题在回到动态函数。

C++语言通过虚拟函数表 VMT 实现了虚拟函数的多态机制。对于 VMT 的内存分配，许多 C++著作中都作了详细说明，在此不再赘述。了解 VMT 的内存分配机制之后，你会发现，C++设计者对于 VMT 的设计原则是以存储空间换取调用时间，即重复存储虚拟函数地址，保证虚拟函数的调用不会导致很多的指针访问。不管是父类还是子类，不管子类是否覆盖了父类的虚拟函数，虚拟函数的调用都只会触发相同数量的指针访问，既由对象指针（this）获取 VMT 指针，由 VMT 指针获取虚拟函数指针。

以存储空间换取调用时间的设计原则，提高了调用虚拟函数的运行效率，但耗费了较多的存储空间。在写一般 C++程序时，可能并不会感觉到虚拟函数的这一设计有什么缺点。但是，如果父类需要定义很多的虚拟函数，只有很小一部分的虚拟函数会被子类覆盖或调用，而且类的派生层次又非常深（如有 7 层以上）。在这种情况下，每个子类 VMT 都会因父类定义太多的虚拟函数而占据大量的内存空间，但程序运行时，通过子类 VMT 调用的虚拟函数又很少。此时，以存储空间换取调用时间的设计原则就变得效率低下，不太适用了。

有同学会问，会存在以上的特殊情况吗？的确存在，在 MFC 和 VCL 封装 Win32 消息机制时，就面临上述类似的问题。消息封装的最初思路就是在父类中为每个消息定义一个虚拟函数，作为消息的处理函数。如果子类控件需要处理某个消息，它就覆盖与该消息对应的虚拟函数。试想，Windows 有上百个消息，为每个消息定义一个虚拟函数，也就是至少 100 个虚拟函数。可是子类控件对于大部分消息都是默认处理，它只会处理一小部分消息，即覆

盖一小部分虚拟函数。可见，使用上述的完全虚拟函数的设计方法来实现消息的封装显然存在较大问题，一是 VMT 会消耗太多的内存空间；二是对 VMT 内存空间的访问率、使用率低下。

基于上述缺点，MFC和VCL对于消息的封装不得不另换思路。为此，Microsoft为MFC引入了消息映射网的设计思路[6]，而Borland仰仗在编译器设计上的深厚功力，引入了动态函数/消息函数。动态函数的设计原则与虚拟函数恰恰相反，即以调用时间换取存储空间。正如虚拟函数指针存放在VMT中一样，动态函数指针也存放在类似的DMT中。但DMT只存储本类覆盖或是定义的虚拟函数，不存储父类的虚拟函数。而且，子类DMT会存储一个指向父类DMT的指针，如此设计是为了子类对象能够调用到父类定义的虚拟函数。有个极端的情况就是，子类对象调用根类定义的虚拟函数，此时的动态函数调用因为多层父类DMT的访问而变得效率低下。

可见，如果我们能够很好的把握设计者的初衷，深入到设计者当时所处的场景，理解设计者当时的设计思路，自然就能够统观全局的把握新技术、心领神会的运用新技术。

2.3 其它的建议

还有些其它的建议，不成体系，单另列出：

1）多编写一些封装的小例子来培养自己面向对象的编程思想，如字符串类 CString、文件类 CFile、内存管理类 CBuffer。

2）要有选择的阅读参考书。如今的计算机书籍可谓良莠不齐，也让初学者挑花了眼睛。我觉得书籍的选择应遵循如下原则：不买只有例子的、只会抄袭的书；只买一本大全、宝典之类的工具查询书；多买讲解编程思想、源码分析的书。

冰冻三尺非一日之寒，从语言进步到编程需要长时间的积累和大量的思考。当质疑、探索、总结、再运用的方法帮助你掌握编程后，你会发现你所获得的东西远比编程本身要多得太多、高得太多、广得太多。

殷明慧

ymhui@21cn.com

2006-5-18

[6] MFC对于消息的封装可参考侯杰的名著《深入浅出MFC》