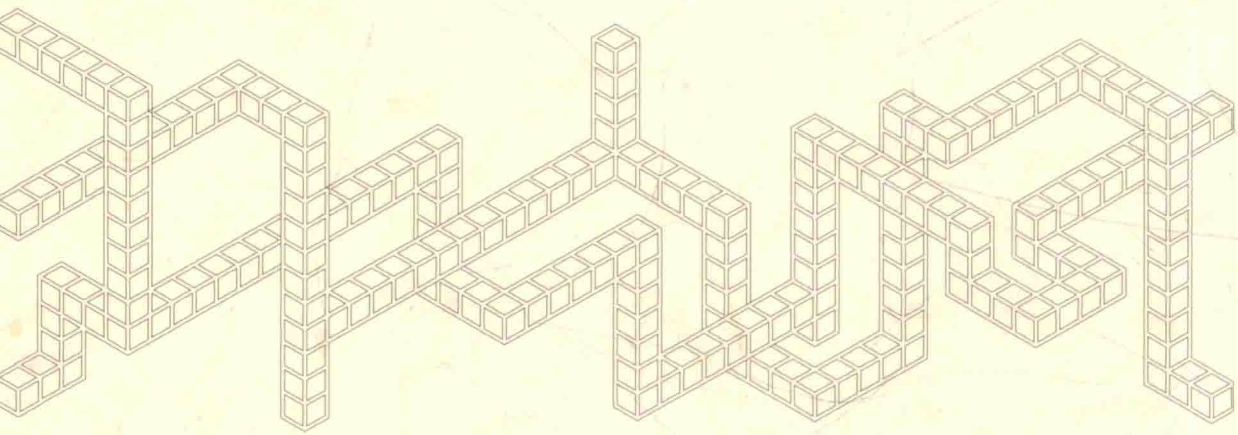
Broadview®
www.broadview.com.cn

软件小设计

董向阳 编著

深挖设计经典，
完整再现设计的自然面貌

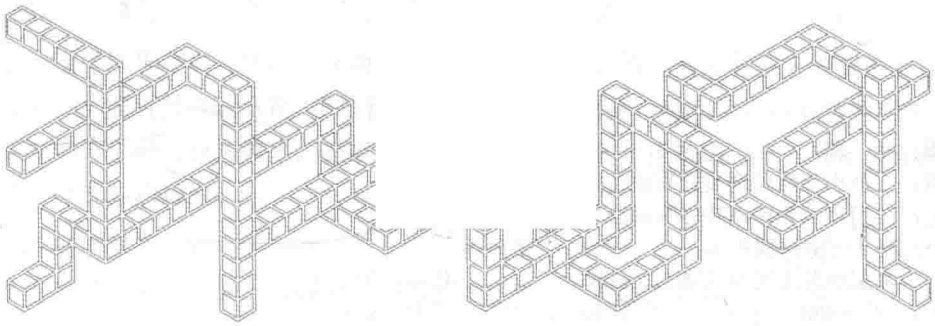
Software Design



中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

软件小设计

董向阳 编著



電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书从最基本编程理论开始，探讨了软件设计中的基本概念，比如过程、对象、封装、继承、多态等；然后，在理清这些概念的基础上，书中集中探讨了构建好对象的若干原则；随后，在这些思想和原则的基础上，书中使用了大量的例子和篇幅分析了软件设计过程中可能遇到的典型问题及可能的解决方案。最后，本书会尝试脱离面向对象设计经验的束缚，直面设计的自然面貌：设计也许不轻松，但是也许并不那么的复杂。

图书在版编目（CIP）数据

软件小设计 / 董向阳编著. —北京：电子工业出版社，2016.5
ISBN 978-7-121-28538-7

Ⅰ. ①软… Ⅱ. ①董… Ⅲ. ①软件设计 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字（2016）第 071776 号

责任编辑：安　娜
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：18.75　　字数：450 千字
版　　次：2016 年 5 月第 1 版
印　　次：2016 年 5 月第 1 次印刷
定　　价：55.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：anna@phei.com.cn。

前言

计算机编程诞生至今，理论和实践的发展越来越丰富，各种编程语言也如雨后春笋般不断出现，不断壮大，不断消亡；随之而来的就是各种介绍计算机程序设计的书籍不断涌现。

在这些著作中，不仅有介绍编程语言的书，如初级的语言规范介绍、中级的语言特性应用、高级的设计模式和软件架构的实现，还有介绍编程技巧的书，如对象的设计、算法的分析、重构的实施、框架的应用，等等。这些书大部分都十分经典，内容集中在某一个具体的方面，就好比一颗颗珍珠零散地放在我们的案头上。

对于我这样从程序员过来的人来说，从入行的那天起，就不断地有高人给我推荐这些由浅入深的经典到不能再经典的书籍，于是我不断地采购这些书籍。每当手里拎起这些沉重的书时，我就会深深地体会到知识的厚重。当然，从这些作者流畅而严谨的述说中，我还是逐渐地学习到了程序设计的各种知识。

在这样的情况下，有一天，在思考之余，我的脑中忽然有了一些大胆的想法：能否有一本书把这些经典的知识串起来，形成一个完整的体系呢？并且能否在这些理论和实践的基础上，再深挖一下设计本来的面目呢？良好的设计是否真的像那些高深的书籍描述的那样是那么高不可攀，必须要经过多年的修炼才能谈及呢？良好的设计是否真的需要那几十本厚重的书才能描述清楚吗？

为了回答这些问题，我决定写些什么，这是本书的写作初衷。有时想想，觉得我还真的有点不自量力，但是反过来想想，我又觉得，不妨就当抛砖引玉吧。我真心希望这本拙劣的小书能引起真正想学习编程技术的读者们的思考，能自己找到这些问题的答案。

本书将从最基本的面向对象编程开始，探讨面向对象设计中的基本概念，比如对象、元素、

继承、多态等；然后，会探讨面向对象设计的基本原则；随后，会在这些思想和原则的基础上分析设计的基本流程及可能遇到的各种问题。在最后一部分，我们会脱离面向对象的束缚，直面设计的自然面貌。

书中的例子大部分都使用C#来实现，一部分例子使用Java和JavaScript来实现，不过好在代码都很简单，喜爱其他语言的读者看起来应该也是毫无压力的。

不过请大家注意，这本书里没有严密的论证，也没有严谨的定义，更没有高深的理论，因而还请大家抱着怀疑的态度阅读本书吧。如果你期望读到的是一本严谨、严密、高深的设计类书籍，那么这本书可能不适合你！

此外，现在大家经常会谈到“互联网思维”。在互联网时代，知识被快速地创建出来，然后飞速地累积和传播。本书的部分资料和图片，是通过搜索引擎直接找到并使用的，根本无法了解其最初的来源在哪里，因此书中如果引用了哪位仁兄的大作但是没有注明，还请见谅！你可以直接联系我，我会把你的作品及有关情况加到引用列表中。对于本书来说，我决定按照“互联网思维”的玩法来对待，就是你可以在引用本书观点的时候，说明引用自本书，当然不引用也没关系，这取决于你自己的选择，我只在意思想的传播。

最后，感谢我的老婆周花梅和我两个可爱的女儿董雨婷和董雨佳，正因为有了她们的爱和鼓舞，我才能够坚持完成本书！

目录

第 1 章

设计概论

记者："金庸先生，你认为在你的小说中，哪一位主角的武功最高强？"

金庸："很大可能是张无忌。张无忌集各家之长，应该比较全面。"

1.1 面向对象程序设计

今天与人闲谈中无意间提到了"倚天屠龙记"中的张无忌，这是一个普通的甚至性格有点软弱的主角，性格复杂，很多人读完小说以后觉得他一点都不像一个英雄人物，与乔峰乔大侠、郭靖郭大侠等相比，简直是差得太远了。回来之后细细回味思索一番，除了张教主是不是英雄这个问题外，我意外地发现他的学艺之路竟然颇有启发性，想来金庸大师也是哲学方面的高才啊。下面让我们一起来回忆一下张教主的学艺之路，并结合金庸小说中的各种成名绝技，看是否有值得借鉴之处。

1.1.1 面向对象思想——任督二脉

打通任督二脉，是修习高等武术的基础和必经步骤，大家看过武侠小说的都知道，也基本都见识过了。之所以是基础，道理很简单，如果任督二脉不通，真气流转不流畅，当然就不能发挥自己体内最大的能量了。就好比一条非常细小的水渠，

怎么能承受得了陡然暴发的洪水。这一关通常看似简单，但是非常有难度，即使是主角兄弟们，也只是会“意外”通过，一般都是借助了外力，比如武功高强的师傅、能力超凡的道具、不可思议的奇遇，等等。对于张教主来说，这一关在说不得大师的乾坤布袋中顺带通过了，凶险万分但非常轻松。对于大部分普通人来说，这一关可能终生都无法逾越，所以武功怎么练也无法达到上层境界。

基本上，任何用过面向对象语言（如 Java、C++、C#等）的同学们张口就能说出面向对象的三大特征：继承、封装、多态。但是又有几个人曾试图深层次地了解过这 6 个字背后的含义呢?

面向对象思想是软件工程发展史上的伟大里程碑，它提供了软件设计与开发的新思路：那就是一切皆是对象。这种思想体现了人们对世界的重新认识。这种认识体现到实际的行动中，就是使用对象去表述世界发展的一切活动。为了生动描述事物之间特性的传承性，“继承”就诞生了；为了描述事物之间相同特性基础上表现出来的差异性，于是“多态”就被创造出来；为了描述事物的完整性和相对封闭性，“封装”就提上了日程，细节从此不需要再去关注。这是面向对象世界观的三大基本特征，也是面向对象设计的核心根基，只有深刻理解了它们，时时刻刻记住它们，把它们深深地融入到你的日常活动中去，你的神功才会一日千里。

1.1.2 面向对象设计原则——九阳神功

当你能做到任何时候、任何场合，眼中只有对象的时候，那么恭喜你，你的任督二脉已经打通！你可以学习任何一门高深的内功了。为什么先学习内功呢？这个大家也都非常熟悉，没有内功的配合，任何高深的招式都是浮云。所以要想天下无敌，必须要练就一身深厚的内功，比如九阳神功。

对象的三大特征全都有了，表明在你眼中一切都应该是对象了，你已经是面向对象了，但是并不是说这就是好的面向对象了。

如何评判一个系统是好的面向对象呢？一般是参照“SOLID”标准。

单一职责原则（SRP）：做一个专一的人

做好并且只做好一件事，这条原则其实不仅仅适用于对象，同样适用于函数、变量等一切编程元素。当然，在商业模式中，将一件事做到极致就是成功，笔者觉得也还是成立的。

开放封闭原则（OCP）：改造世界大部分不是破坏原来的秩序

在对象世界中，添加新的功能一般意味着新的对象，一个好的设计也意味着这个新的修改不要大幅度波及现有的对象。这一条理解起来简单，实施起来却最是困难。无数的模式和解耦方法都是为了达到这个目的而诞生的。

里氏替换原则（LSP）：长大后，我就成了你

父类使用的地方，子类也可以使用。这一条希望子类不要破坏父类的接口成员，一旦破坏了，就如同人与人之间破坏合同一样，有时候会很糟糕。

接口分离原则（ISP）：不要一口吃成胖子

接口不要过于庞大，繁杂的东西是难以理解、难以扩展、难以修改的。这一条的目的与单一职责原则类似，不过是更加强调了接口的逻辑一致性和简易性。

依赖倒置原则（DIP）：抽象的艺术才有生命力

高层与底层组件之间都应该依赖于抽象的组件。这一条深刻揭示了抽象的生命力，抽象的对象才是最有表达能力的对象，因为它通常是“无形”的，可以随时填充相关的细节。

除了这几个基本的设计原则外，还有一些衍生原则，掌握它们，你将能更好地面向对象。

迪米特法则：尽量不与无关的类发生关系。

对象之间联系越是简单，则越是容易管理。

好莱坞法则：不要调用我，让我调用你。

电影中常说，单线联系最安全，就是这样。

多使用组合，少使用继承

复用的手段除了继承这种强约束手段外，组合这种弱耦合的关系更加灵活。

实现依赖于抽象

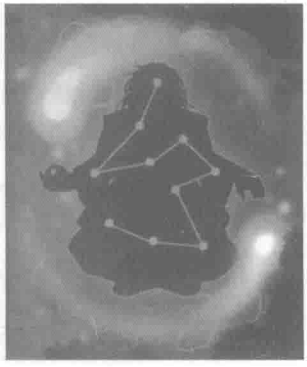
这一点一般也称为面向接口编程，保证了对象之间关系稳定。

修炼完这些原则，你已经神功初成了，那是不是活动活动手脚，就可以大干一场了呢？

1.1.3 模式——乾坤大挪移

当张无忌学会了九阳神功下山以后，他已经身负绝世内功了，但是不知大家发现了没有，他居然连最普通，只会些三脚猫功夫的毛贼都打不过，他只能依靠阿蛛的招式，来借力反击，只是这样谈何天下无敌呢。不过当他学会了乾坤大挪移以后，那真的是如同凤凰涅槃，瞬间完成了超级赛亚人变身。这个道理似乎也很简单，光有高深的内功，但是没有运转自如、充分发挥内功的招式，同样也是不成的。招式，那可是前辈们经验的总结，是精华啊。学会了这个，

你才有决胜天下的资本。



世间原本没有模式，使用的人多了，才有了模式。模式是经验的总结，是“巨侠”们的心血。大型模式如架构模式（分层、MVC、PAC、黑板、中间人、反射、管道、微核、REST 架构等），它们描述了系统大骨架的构建过程；中等规模的模式如设计模式（GOF 23 种模式、POSA 中的各种设计模式），它们描述了子系统中组件的构建过程；小的模式如各种语言中的编程实践（C#中的 IDispose 模式、C++中的 Counted Point）模式等，它们描述了解决语言中特定问题的实施方案。这些模式合起来就是面向对象程序设计过程中针对特定问题的乾坤大挪移神功。

1.1.4 重构——太极拳

如果一定要评出金庸小说中的三大巅峰神功，那么太极拳必能入选。练习太极拳是需要超凡的悟性的，太极拳重在以意运转，不能有丝毫外在强加的约束，这一点看过原著的同学们都很清楚。自从太极拳问世后，人们才真正体会到了什么叫做“以柔克刚”！

重构是精心打磨、持续雕琢代码的过程，是任何资深码农的必备技能。重构不是无目的的，重构是一种在不改变代码行为的前提下，改善代码可读性、可扩展性的过程。

之所以需要重构，就是因为代码也是符合事物发展规律的，也有一个从出生到成长，从强壮到衰败，从衰败到腐烂的过程，而且是循序渐进的，不知不觉地就从好变烂了。毫无疑问，并不是所有的烂代码都是一次写成的，也许最初的代码设计是很好的，但是一旦被多个人修改过以后，就变坏了，很多人对此都深有体会。代码总是在所有人的共同“努力”下写烂的。

如果说上面提到的那些技术都有迹可循、规则性比较强的话，那么相对来说，重构的技术要相对柔和一点，没有那么强烈的约束条款，而且虽然也有像介绍“代码坏味道”的伟大著作（《重构》一书）诞生，但是，通常人们脑海中对于重构的冲动，从没有像使用模式那么强烈，根本不会想到重构会是那么的重要，就如同书中的那些高人们根本不会想到如此缓慢而且无固定招式的太极拳会是那么强悍一样。重构与太极拳一样，进行到多大程度和画多大多小的圈，都无法明确衡量，它们都是真正的“意识流”。

1.1.5 抽象与组合——独孤九剑

要说金庸小说中最厉害的武功，笔者认为既不是乾坤大挪移，也不是太极拳，更不是葵花宝典，而是独孤九剑。那是什么境界啊？九剑破遍天下任何拳脚、气功、暗器、十八般兵器，这才叫“无敌”。当然，这个武功不是随便是谁都能练得成的，也就是像令狐冲那样智慧超群，洒脱异常，具备一定内功（后期更辅助以吸星大法），武功又杂的小哥才能练；而且这个练习过程一般也比较长，据风清扬介绍，令狐老兄刻苦练习十几年后方能与东方不败一战。

我们再看看该神功原创者独孤大侠一辈子兵器的变迁：从利剑到重剑，到木剑，再到无剑。大家发现了没有，到了最后抛开一切招式、内功，融会贯通，水乳交融以后，才是真正的超凡入圣，这个时候也才发现原来一切又回到了起点：原来武功就是这么简单，就是锻炼与发挥人体的最大能量，随便地一挥一洒就有无与伦比的威力。返璞归真才是武者的终极、青铜门后面的永恒（出自盗墓笔记一书）。

不管是面向过程，还是面向对象，不管是面向组件，还是面向服务，归根结底只不过是人们对世界的抽象方式和抽象的粒度不同而已，把抽象出来的东西组合起来，形成一定的规范和流程，依靠这些去解决实际的需求，这就是编程。抽象与组合，才是终极的艺术。

1.2 面向过程与面向对象

前面我们提到了面向过程和面向对象的概念，作为计算机编程理论发展过程中最为重要的两种思想，在正式接触设计细节之前，我们绝对有必要先来弄清楚一下它们的来龙去脉。

计算机之所以出现，就是为了帮助人们解决科学计算中人类无法承担的大量的繁重计算，这也是为什么我们称之为“计算机”的原因，所以初期的程序也基本都是集中在科学计算的相关方面。对于科学计算来说，问题解决起来就简单了，基本就是把数据放到一个统一的地方，然后把计算过程分为几步，每步使用一个函数去操作那些数据，最后用主函数把这些小函数串起来调用一遍就得到结果了。这个过程其实就像小品中说的：“请问，把大象装进冰箱需要几步？”答案是：“分为三步，第一步打开冰箱门，第二步把大象塞进去，第三步把冰箱门关上。”

随着计算机技术的不断提高，计算机逐渐被广泛用于解决各个领域的问题，于是计算机处理的问题越来越复杂，程序的业务流程也越来越庞大，因而实现业务的函数也越来越大，越来

越多。于是程序设计开始不得不先使用最主要的外层函数来实现空的流程外壳，然后逐步细化每个外层函数来生成更多的内层的函数，直到最内层的函数完成功能，设计结束，如图 1-1 所示。

这些最外层的意义丰富的函数就叫做模块，这就是模块化的思想。

> **模块化和函数**构成了面向过程（Procedure Oriented）编程的主旋律；从上往下，步步细化，以函数为编程的基本单元来组织整个软件的运行流程，这么做简单粗暴，但是相当有效。

这个思路一直高效运作，直到计算机处理问题的复杂程度超出了函数的能力范畴，人们才不得不重新考虑编程的基本单元。

就好比一个国家开始的时候，1 毛钱能买很多东西，大部分时候 1 毛钱就够用了，1 块钱的东西算是比较贵的，如果没有 1 块钱，也只需要掏出 10 个 1 毛钱就可以了，所以主要印刷 1 毛钱就行了。但是随着物价越来越高，很多东西都是以 10 块为起卖价了，甚至还有一些东西已经卖到 100 块，这个时候如果还是以 1 毛钱为支付的主要钱币，那么买 100 块的东西，乖乖，你要带着 1000 张 1 毛钱去买东西，这该多么可怕啊，于是大家不得不改为以印刷 10 块钱为主。这种新 10 块钱的主流纸币在编程理论中就是“对象”。

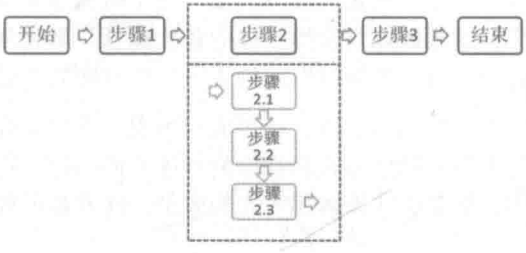


图 1-1　程序设计流程

我们回过头来看面向过程的过程，通过前面的分析，我们知道函数的本质就是计算的步骤，代表的是一种行为。这些函数使用的数据集中存放在一个地方，需要的时候去拿就可以了。这个过程是以函数作为最基本的逻辑单元构建出来的整个业务系统，这里处处都透出全局的味道，函数是全局的，数据是全局的。但是随着问题越来越复杂，有些数据很明显需要控制起来，不再是所有函数都能访问的，而且很多函数之间越来越表现出很强的关联性，于是一个更加强大的逻辑单元，第一次将程序的两个基本要素——数据和算法，结合起来的超级元素“对象”诞生了，设计思路迈出了关键的一步，如图 1-2 所示。

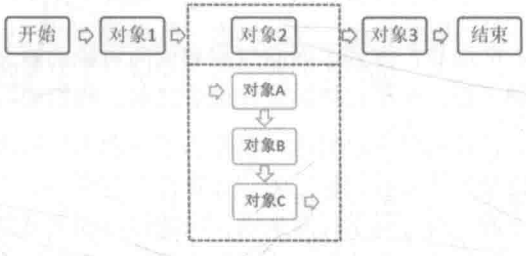


图 1-2　面向对象

> 面向对象（Object Oriented）不是抛弃面向过程，而是在过程的基础上，把数据加入抽象的范畴，理解这一点至关重要。

面向对象仍然是基于模块化基础上的思考方式，只不过描述业务逻辑的功能单元不再是代表单一行为的函数，而是行为和数据的混合体——“对象”。相比面向过程，面向对象最主要的进化就是把问题领域中的事物给对象化了，对象包括属性与行为。在这个抽象的过程中，为了描述私有性，封装来了；为了描述传承性，继承来了；为了描述差异性，多态来了。

人类世界的发展在继续，计算机的发展也在继续，渐渐地，人们发现在对象的使用过程中，总是有些东西是贯穿多个对象的，比如记日志，基本上每个核心调用都可能用到，其他的还有像安全性检查、登录状态检查、性能检查等，相对来说，它们虽然不是核心的业务逻辑，但却是系统不可缺少的步骤，这些离散的特性使得我们并不能用继承的方式去复用这些功能，于是立足于解决这种问题的编程理念——面向方面编程出现了。

面向方面编程将这些重复的但是无法通过简单继承来复用的功能称为方面，将方面抽取出来后创建专门的对象来表示它们，然后在编程中通过各种静态注入（代理），或者动态注入（反射）的方式来融合到原来的对象体系中实现复用。所以，方面是对象的一种补充，而没有从基本概念上做出新的突破。

> 面向方面（Aspect Oriented）不是替代面向对象的，而是作为对象编程的有益补充而存在。

接下来，随着分布式系统的大范围应用，软件分布的范围也在逐渐扩大，于是模块可能不在一起了，有些模块被单独放置在某个地方，供其他模块，特别是某些远程模块调用；也有可能公司业务重组，有些模块被保留供别的业务使用，总之，这些模块可统称为服务，如图 1-3 所示。

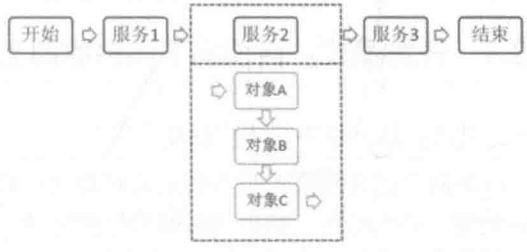


图 1-3 面向服务

> 面向服务（Service Oriented）也不是替代面向对象的，它是在对象的基础上，提供了更大粒度的抽象。

面向服务研究的重点在于模块的分布式部署和使用，目前基于 XML 这种中间格式的 SOAP 协议或者 REST 开发方式最为热门，它们解决了不同实现的系统（异构系统）之间的分布式开发。所以服务仍然是对象的补充，并没有从基本概念上做出关键突破。

回顾上面这个不断发展、不断创新的过程，我们发现：

> （1）科技的进步来源于实际的需求，请时刻记住“实际的需求”，因为这一点是很多理论的基石，比如敏捷开发、简单设计、重构等。

（2）量变带来质变，最重要的就是过程到对象的突破，大数据和云计算是新的量变，这次量变什么时候引发质变呢？大家拭目以待，当然也有可能由你来突破这关键的一点。

（3）设计的思路从来都没变，仍然还是模块化，层层细化或者反过来。

（4）面向对象仍是设计的核心，它包含了其他面向 XX 的思想。

1.3 设计的宏观面貌

软件设计听起来很高大上，很抽象，一般人闻之变色，但其实，软件设计的过程就是软件开发的过程，也就是从客户需求分析到构建软件系统的过程。设计的产物就是开发的产物，就是代码本身；俗话说得好，代码即设计。

面向对象设计的宏观面貌描述的就是设计的过程。

设计的过程有很多种，比如瀑布模型、原型模型、迭代模型、统一过程模型、极限编程、敏捷开发，等等，这么多的开发理论，读下来可以搞得你晕头转向，这是本书第一次提到它们，也差不多是最后一次，因为不管它们的定义和流程有多少差异，基本都可以归结为以下几个方面。

1.3.1 开发模式：自顶向下和自底向上

方式 1：从上往下、步步细化

自顶向下的开发模式是不断地将相对复杂的大问题分解为相对简单的小问题，找出每个问题的关键、重点所在，然后用精确的思维定性、定量地去描述问题，最终完成软件的过程。该模式的核心本质是“不断分解”，直到每个问题都变成比较简单的编程问题。

这是大部分模型采用的正统开发模式，采用这种模式比较稳健，开发的过程很容易控制。

以开发一个网络爬虫获取各大网页的新闻为例，自顶向下的设计过程大致如表 1-1 所示。

表 1-1 自顶向下的设计过程

分解过程	过程描述	分解结果
第一次分解	启动爬虫程序，输入需要获取的网址，运行后获得新闻	得到主程序类，有一个入口方法（如 main 方法），方法内描述了最为粗糙的程序流程
第二次分解	将主要的获取新闻的流程继续分解：将每个网站的爬虫存起来放到集合中，然后一次调用各自的爬取方法，得到新闻后存到数据库中	得到爬虫类，有一个公共的方法，调用以后可以获取到新闻并存入数据库中