面向对象技术

邓正宏 薛 静 郑玉山 编著

国防工业出版社
http://www.ndip.cn

面向对象技术

邓正宏　薛静　郑玉山　编著

国防工业出版社

内 容 简 介

本书主要介绍了面向对象技术的基本内容，在了解对象建模技术的基本概念的基础上，详细讲解了统一建模语言(UML)、UML 分析设计过程、抽象类、应用框架、设计模式、组件、面向对象设计原则、面向对象自动化测试框架等内容。内容覆盖了这一学科的一些最基本的内容，同时也涵盖最新的面向对象技术。

本教材可供计算机专业高年级学生及硕士研究生使用，也可供从事系统分析、设计的计算机工作者参考。

图书在版编目(CIP)数据

面向对象技术 / 邓正宏等编著 . —北京：国防工业出版社，2004.11

ISBN 7-118-03620-X

Ⅰ. 面... Ⅱ. 邓... Ⅲ. 面向对象语言－程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2004)第 090114 号

国防工业出版社出版发行

(北京市海淀区紫竹院南路 23 号)

(邮政编码 100044)

北京奥鑫印刷厂印刷

新华书店经售

*

开本 787×1092 1/16 印张 22¼ 513 千字

2004 年 11 月第 1 版 2004 年 11 月北京第 1 次印刷

印数：1—4000 册 定价：30.00 元

(本书如有印装错误，我社负责调换)

国防书店：**68428422** 发行邮购：**68414474**

发行传真：**68411535** 发行业务：**68472764**

前　言

面向对象方法是当今软件系统分析、设计与实现的最有影响的方法。本教材结合作者使用面向对象多种技术的实际经验，通过对面向对象方法、面向对象分析设计过程、面向对象程序设计、面向对象测试等知识的综合介绍，为读者提供了一整套实用的面向对象技术，并通过应用实例为读者学习和使用面向对象方法提供了生动而具体的参考资料。

在本教材中，我们系统地阐述了对象建模技术的基本概念，详细讲解了统一建模语言(UML)、UML 分析设计过程、抽象类、应用框架、设计模式、面向对象设计原则、组件、面向对象自动化测试框架件等内容。

由于我们编写这本教材的目的是向读者介绍这一学科的一些最基本的内容（同时也涵盖最新的面向对象技术），因而在讲述时尽量避免一些严格的形式化系统。根据以往的教学经验，我们相信读者在掌握了这些基本内容以后，可以较顺利地阅读其他有关的专著，以求对这一学科有更深入的了解。

另外，本书中没有采用统一的程序设计语言描述程序，主要使用目前流行的 C++、Java 语言。我们感到，这样做尽管从表面上看不够统一，但是可以使读者接触较多的面向对象设计方法，而且有利于读者阅读有关专著。因而，权衡利弊我们做出了这样的选择。

本教材可供计算机专业高年级学生及硕士研究生使用，讲授时数约 40 学时，也可供计算机工作者参考。

本书的第 1 章、第 3 章～第 5 章及第 7 章、第 8 章由西北工业大学计算机科学与工程系邓正宏编著，第 2 章和第 6 章由西北工业大学计算机科学与工程系郑玉山编著，第 9 章、第 10 章由西北工业大学计算机科学与工程系薛静编著。如果我们的合作能对读者了解和掌握这一新兴学科有所帮助的话，我们将感到十分高兴。另外，西北工业大学计算机科学与工程系的樊蓉、张世芳、齐震、梁春泉同志参加了部分编写工作，在此向他们表示感谢。

西北工业大学的蒋立源教授在百忙中认真审阅了全书的内容，并提出了宝贵的意见，在此也表示衷心的感谢。

由于时间仓促，加之编者水平有限，不妥之处在所难免，诚恳希望同行的专家及广大读者提出宝贵意见。

作　者

目　录

第 1 章

面向对象技术概述

1.1 面向对象技术的产生

面向对象技术提供了一种新的认知和表示世界的思想和方法，它对计算机工业的影响是深远的。计算机从业人士利用它提出了面向对象的分析设计方法、计算机程序设计语言、面向对象的软件设计方法、面向对象的数据库等。同时面向对象技术为软件工业实现工程化提供了强有力的支持，正是面向对象技术造就了架构、统一建模语言（UML）、框架、模式、组件、构件、中间件等概念。

计算机的不断发展为计算机及网络应用提供了大量技术先进、功能强大的应用软件系统，同时也给软件开发者和用户带来了相应的问题：

(1) 软件系统规模庞大，研制周期长，维护费用高；

(2) 软件系统过于复杂，在一个系统中集成了各种功能，大多数功能不能灵活地装卸、单独升级或重复利用；

(3) 应用软件不易集成，即使各应用程序是用相同的编程语言编写的，并且运行在相同的计算机上，特定应用程序的数据和功能也不能提供给其他应用程序使用。

1969 年 NATO 会议之后，“软件危机”成为人们关注的焦点。为迎接软件危机的挑战，人们进行了不懈的努力，这些努力大致上是沿着两个方向同时进行的。

一是从管理的角度，希望实现软件开发过程的工程化。这方面最为著名的成果就是提出了大家都很熟悉的“瀑布式”生命周期模型。它是在 20 世纪 60 年代末“软件危机”后出现的第一个生命周期模型。如下所示：

分析 → 设计 → 编码 → 测试 → 维护

后来，又有人针对该模型的不足，提出了快速原型法、螺旋模型、喷泉模型等对“瀑布式”生命周期模型进行补充。现在，它们在软件开发的实践中被广泛采用。这方面的努力，还使人们认识到了文档的标准以及开发者之间、开发者与用户之间的交流方式的重要性。一些重要文档格式的标准被确定下来，包括变量、符号的命名规则以及原代码的规范格式。值得一提的是，在开发者之间、开发者与用户之间的交流方式这方面，由于 Internet 的出现提供了一种全新的交流手段，也产生了一种基于 Internet 的全新的开发方式，即 OSS(Open Source Software)，其代表作有 Linux（操作系统）、Apache（web server）、Sendmail（Mail server）等。OSS 是一种极有前途的开发方式，借 Internet 发展的大潮，它势必会对整个软件开发模型产生难以估量的影响。

二是侧重于对软件开发过程中分析、设计方法的研究。这方面的第一个重要成果就是在20世纪70年代风靡一时的结构化开发方法，即PO(面向过程的开发或结构化方法)。PO是人们在用计算机世界来表达现实世界时，追求过程、模块化、封装以及更高的抽象的结果。人们用计算机来映射现实世界时，最低层的实现无非是靠数字电路技术产生的高电平与低电平信号。用数学的语言来表示，就是像“010101000010111”这样的二进制串。这样的抽象层次是极低的，远离了自然语言，对一般人是不可理解的。人们把这些二进制串分块定义，提出了字节、ASCII码这样的更高抽象层次的概念，使之对应于自然语言的一个个字母。在此基础再借助某种形式语言，抽象出变量、表达式、运算、语句等概念。在这个层面上，一般经过训练的程序员已经可以比较轻松地进行软件开发了。更进一步的抽象就产生了PO。在PO中，人们关注的是如何用函数和过程来实现对现实世界的模拟，将其映射到计算机世界之中。面向对象（OO）是这种抽象层次不断提高的过程的自然发展结果，它采用类和对象的概念，把变量以及对变量进行操作的函数和过程封装在一起，用这种更高一级的抽象来表达客观世界。通常，一个对象包含一些属性和方法，它对应于自然语言中一个有意义的名词，描述了现实世界中的一个物体（物理实体）或概念（抽象实体）。这个抽象层次如下所示：

计算机世界中的抽象层次

* XO(X?-Oriented)：最高的抽象层次
* OO(对象、类)
* PO(过程、函数、变量)
* 变量、运算、表达式、语句
* 字节(4位、8位、16位、32位、64位)
* 二进制串0101011110001：最低的抽象层次

从以上的讨论可知，软件工程的发展历史就是人们不断追求更高的抽象、封装和模块化的历史，面向对象技术当然不会是历史的终结。尽管不能精确得到OO之后是什么，我们至少可以推知，OO之后的XO，必然将是比OO更高一级的抽象。它所依赖的核心概念必然高于并包容对象这一概念，正如对象高于并包容了函数和变量一样。

针对日趋复杂的软件需求的挑战，软件业界发展出了OO的软件开发模式。目前作为针对“软件危机”的最佳对策，OO技术已经引起人们的普遍关注。最初被多数人看作只是一种不切实际的方法和满足一时好奇心的研究，现在得到了人们近乎狂热的欢迎。

为什么面向对象运动发展到了现在这样火暴的程度？部分是源于人们长久以来的一个希望：人们希望它，像以前其他的软件开发技术一样，能够满足软件开发对于生产效率、可靠性、易维护性、易管理等方面的更高、更快、更强的迫切需求。除此之外，还有许多原因都促使了它的流行。

面向对象的开发强调从问题域的概念到软件程序和界面的直接映射；心理学的研究也表明，把客观世界看成是许多对象更接近人类的自然思维方式。对象比函数更为稳定；软件需求的变动往往是功能相关的变动，而其功能的执行者——对象，通常不会有大的变动。另外，面向对象的开发也支持、鼓励软件工程实践中的信息隐藏、数据抽象和封装，在一个对象内部的修改被局部隔离。面向对象开发的软件易于修改、扩充和维护。

面向对象也被扩充应用于软件生命周期的各个阶段（从分析到编码）。而且，面向对

象的方法自然而然地支持快速原型法和 RAD(Rapid Application Development)。面向对象开发的使用鼓励重用，不仅软件的重用，还包括分析、设计的模型的重用。更进一步，OO 技术还方便了软件的互换性，即网络中一个节点应该能够利用另一个节点上的资源。面向对象的开发还支持并发、层次和复杂等一些在目前的软件系统中常见的现象。今天我们常常会需要建造一些软件系统（不止是黑盒应用），这些复杂系统通常包含由多个子系统组成的层次结构。面向对象的开发支持开放系统的建设；利用不同的应用来进行软件集成有了更大的柔性。最后，由于系统集成遍布软件生命周期的各个阶段，面向对象开发的使用可以减小开发复杂系统所面临的危险。

1.2　面向对象思想

我们在这一节中主要讨论面向对象方法学和面向对象程序设计的问题。首先从方法学的角度，对面向对象分析、设计思想做了一个综述，对不同的方法做了比较，为我们后续几章中讨论的对象模型、动态模型、UML 分析设计做好铺垫。另外，对目前流行面向对象编程 OOP 中的许多核心概念和机制进行了有益的讨论，希望通过这些讨论能够让读者对于 OOP 有更深入的理解，同时明白 OOP 作为已经发展将近 30 年的程序设计思想，为我们后面讲述框架、设计模式、构件、设计原则做好准备。

1.2.1　面向对象方法学

对于方法学，我们的理解如下所述。

(1) 方法学的目的是使后人分享前人的成功，避开前人的失败，把注意力集中在尚未开拓领域的创造性劳动上。所以方法学与开发人员的创造性是绝不冲突的，它既不能像法律那样靠权威来界定是非边界，也不能像定律那样通过证明和推理给出普遍结论。如果一定要做比喻的话，它好比人的世界观。

(2) 没有放之四海而皆准的方法学，任何方法学都有其局限性，所以软件开发人员大可不必拘泥于某种特定的方法学。

例如，面向对象方法的对象模型图，这种形式化语言远不如结构化方法的结构图和数据流图简单明了，倘若把信息系统全部用对象模型图表述出来，至少也要几十页。

20 世纪 80 年代末以来，随着面向对象技术成为研究的热点出现了几十种支持软件开发的面向对象方法。其中，Booch、Coad/Yourdon、OMT 和 Jacobson 的方法在面向对象软件开发界得到了广泛的认可。特别值得一提的是统一建模语言 UML（Unified Modeling Language），该方法结合了 Booch、OMT 和 Jacobson 方法的优点，统一了符号体系，并从其他的方法和工程实践中吸收了许多经过实际检验的概念和技术。UML 方法已提交给对象管理组织（OMG），申请成为面向对象方法的标准。

面向对象方法都支持 3 种基本的活动：识别对象和类，描述对象和类之间的关系，以及通过描述每个类的功能定义对象的行为。为了发现对象和类，开发人员要在系统需求和系统分析的文档中查找名词和名词短语，包括可感知的事物（汽车、压力、传感器）、角色（母亲、教师、政治家）、事件（着陆、中断、请求）、互相作用（借贷、开会、交叉）、人员、场所、组织、设备和地点。通过浏览使用系统的脚本发现重要的对象及其责

任，是面向对象分析和设计过程的初期重要的技术。

当重要的对象被发现后，通过一组互相关联的模型详细表示类之间的关系和对象的行为，这些模型从 4 个不同的侧面表示了软件的体系结构：静态逻辑、动态逻辑、静态物理和动态物理。静态逻辑模型描述实例化（类成员关系）、关联、聚集（整体/部分）和一般化（继承）等关系，这被称为对象模型。一般化关系表示属性和方法的继承关系。定义对象模型的图形符号体系通常是从用于数据建模的实体关系图导出的。对设计十分重要的约束，如基数（一对一、一对多、多对多），也在对象模型中表示。

1. Booch 方法

动态逻辑模型描述对象之间的互相作用。互相作用通过一组协同的对象、对象之间消息的有序序列、参与对象的可见性定义，来定义系统运行时的行为。Booch 方法中的对象交互作用图被用来描述重要的互相作用，显示参与的对象和对象之间按时间排序的消息。可见性图用来描述互相作用中对象的可见性。对象的可见性定义了一个对象如何处于向它发送消息的方法的作用域之中。例如，它可以是方法的参数、局部变量、新的对象，或当前执行方法的对象的部分。静态物理模型通过模块描述代码的布局，动态物理模型描述软件的进程和线程体系结构。

Booch 方法的过程包括以下步骤：

(1) 在给定的抽象层次上识别类和对象；

(2) 识别这些对象和类的语义；

(3) 识别这些类和对象之间的关系；

(4) 实现类和对象。

这 4 种活动不仅仅是一个简单的步骤序列，而是对系统的逻辑和物理视图不断细化的迭代和渐增的开发过程。

类和对象的识别包括找出问题空间中关键的抽象和产生动态行为的重要机制。开发人员可以通过研究问题域的术语发现关键的抽象。语义的识别主要是建立前一阶段识别出的类和对象的含义。开发人员确定类的行为（即方法）和类及对象之间的互相作用（即行为的规范描述）。该阶段利用状态转移图描述对象状态的模型，利用时态图（系统中的时态约束）和对象图（对象之间的互相作用）描述行为模型。

在关系识别阶段描述静态和动态关系模型。这些关系包括使用、实例化、继承、关联和聚集等。类和对象之间的可见性也在此时确定。

在类和对象的实现阶段要考虑如何用选定的编程语言实现，如何将类和对象组织成模块。

在面向对象的设计方法中，Booch 强调基于类和对象的系统逻辑视图与基于模块和进程的系统物理视图之间的区别。它还区别了系统的静态和动态模型。然而，Booch 的方法偏向于系统的静态描述，对动态描述支持较少。

Booch 方法的力量在于其丰富的符号体系，包括：

(1) 类图（类结构－静态视图）；

(2) 对象图（对象结构－静态视图）；

(3) 状态转移图（类结构－动态视图）；

(4) 时态图（对象结构－动态视图）。

用于类和对象建模的符号体系使用注释和不同的图符（如不同的箭头）表达详细的信息。Booch 建议在设计的初期可以用符号体系的一个子集，随后不断添加细节。对每一个符号体系还有一个文本的形式，由每一个主要结构的描述模板组成。符号体系由大量的图符定义，但是，其语法和语义并没有严格的定义。

2. Rumbaugh 的 OMT 方法

Rumbaugh 的 OMT 方法从 3 个视角描述系统，相应地提供了 3 种模型：对象模型、动态模型和功能模型。

对象模型描述对象的静态结构和它们之间的关系，主要的概念包括：类、属性、操作、继承、关联（即关系）、聚集。

动态模型描述系统那些随时间变化的方面，其主要概念有：状态、子状态和超状态、事件、行为、活动。

功能模型描述系统内部数据值的转换，其主要概念有：加工、数据存储、数据流、控制流、角色。

该方法将开发过程分为以下 4 个阶段。

(1) 分析。基于问题和用户需求的描述，建立现实世界的模型。分析阶段的产物有：问题描述、对象模型（对象图＋数据词典）、动态模型（状态图＋全局事件流图）、功能模型（数据流图＋约束）。

(2) 系统设计。结合问题域的知识和目标系统的体系结构（求解域），将目标系统分解为子系统。

(3) 对象设计。基于分析模型和求解域中的体系结构等添加的实现细节，完成系统设计。对象设计的主要产物包括：细化的对象模型、细化的动态模型、细化的功能模型。

(4) 实现。将设计转换为特定的编程语言或硬件，同时保持可追踪性、灵活性和可扩展性。

3. Coad/Yourdon 方法

Coad/Yourdon 方法严格区分了面向对象分析 OOA 和面向对象设计 OOD。该方法利用 5 个层次和活动定义及记录系统行为，输入和输出。这 5 个层次的活动包括以下内容。

(1) 发现类及对象。描述如何发现类及对象。从应用领域开始识别类及对象，形成整个应用的基础，然后，据此分析系统的责任。

(2) 识别结构。该阶段分为两个步骤：①识别一般——特殊结构，该结构捕获了识别出的类的层次结构；②识别整体——部分结构，该结构用来表示一个对象如何成为另一个对象的一部分，以及多个对象如何组装成更大的对象。

(3) 定义主题。主题由一组类及对象组成，用于将类及对象模型划分为更大的单位，便于理解。

(4) 定义属性。其中包括定义类的实例（对象）之间的实例连接。

(5) 定义服务。其中包括定义对象之间的消息连接。

在面向对象分析阶段，经过 5 个层次的活动后的结果是一个分成 5 个层次的问题域模型，包括主题、类及对象、结构、属性和服务 5 个层次，由类及对象图表示。

面向对象设计模型需要进一步区分以下 4 个部分。

(1) 问题域部分（PDC）。面向对象分析的结果直接放入该部分。

(2) 人机交互部分（HIC）。这部分的活动包括对用户分类，描述人机交互的脚本，设计命令层次结构，设计详细的交互，生成用户界面的原型，定义 HIC 类。

(3) 任务管理部分（TMC）。这部分的活动包括识别任务（进程）、任务所提供的服务、任务的优先级、进程是事件驱动还是时钟驱动、以及任务与其他进程和外界如何通信。

(4) 数据管理部分（DMC）。这一部分依赖于存储技术，是文件系统，还是关系数据库管理系统，或是面向对象数据库管理系统。

4. Jacobson 方法

Jacobson 方法与上述 3 种方法有所不同，它涉及到整个软件生命周期，包括需求分析、设计、实现和测试等 4 个阶段。需求分析和设计密切相关。需求分析阶段的活动包括定义潜在的角色（角色指使用系统的人和与系统互相作用的软、硬件环境），识别问题域中的对象和关系，基于需求规范说明和角色的需要发现 use case，详细描述 use case。

设计阶段包括两个主要活动，从需求分析模型中发现设计对象，以及针对实现环境调整设计模型。第一个活动包括从 use case 的描述发现设计对象，并描述对象的属性、行为和关联，以及把 use case 的行为分派给对象。

在需求分析阶段的识别领域对象和关系的活动中，开发人员识别类、属性和关系。关系包括继承、熟悉（关联）、组成（聚集）和通信关联。定义 use case 的活动和识别设计对象的活动，两个活动共同完成行为的描述。Jacobson 方法还将对象区分为语义对象（领域对象）、界面对象（如用户界面对象）和控制对象（处理界面对象和领域对象之间的控制）。

在该方法中的一个关键概念就是 use case。use case 是指行为相关的事务（transaction）序列，该序列将由用户在与系统对话中执行。因此，每一个 use case 就是一个使用系统的方式，当用户给定一个输入，就执行一个 use case 的实例并引发执行属于该 use case 的一个事务。基于这种系统视图，Jacobson 将 use case 模型与其他 5 种系统模型关联。

(1) 领域对象模型：use case 模型根据领域来表示。

(2) 分析模型：use case 模型通过分析来构造。

(3) 设计模型：use case 模型通过设计来具体化。

(4) 实现模型：该模型依据具体化的设计来实现 use case 模型。

(5) 测试模型：用来测试具体化的 use case 模型。

5. UML

UML 建模思想举起了统一的大旗。它融合了多种优秀的面向对象建模方法，以及多种得到认可的软件工程方法，消除了因方法林立且相互独立带来的种种不便。它通过统一的表示法，使不同知识背景的领域专家、系统分析和开发人员及用户可以方便地交流。它的出现为面向对象建模语言的历史翻开了新的一页，并受到工业界、学术界以及用户的广泛支持，成为面向对象技术领域占主导地位的建模语言。OMG 采纳它为标准建模语言，进一步将它推向事实上的工业标准的地位，目前它正向国际标准化组织（ISO）提出标准化申请。

尽管目前我国计算机界对 UML 的推崇程度很高，但我们应该客观地认识到 UML 依然存在许多缺憾甚至是错误，需要进一步完善。一个规范的标准化进程总是很漫长，在对它的修订过程中总会不断发现新问题，发现问题、解决问题是个循环反复的过程，在

这个过程中，人们不断改进和完善 UML。

UML 是在多种面向对象建模方法的基础上发展起来的建模语言，主要用于软件密集型系统的建模。在多种面向对象建模方法流派并存和相互竞争的局面中，UML 使不同厂商开发的系统模型能够基于共同的概念，使用相同的表示法，呈现彼此一致的模型风格。而且它从多种方法中吸收了大量有用（或者对一部分用户可能有用）的建模概念，使它的概念和表示法在规模上超过了以往任何一种方法，并且提供了允许用户对语言做进一步扩展的机制。

UML 在语法和语义的定义方面也做了大量的工作。以往各种关于面向对象方法的著作通常是以比较简单的方式定义其建模概念，而以主要篇幅给出过程指导，论述如何运用这些概念来进行开发。UML 则以一种建模语言的姿态出现，使用语言学中的一些技术来定义。尽管真正从语言学的角度看它还有许多缺陷，但它在这方面所做的努力却是以往的各种建模方法无法比拟的。

不过 UML 在取得巨大成功的同时，也不断地受到批评。来自工业界的批评主要是，它过于庞大和复杂，用户很难全面、熟练地掌握它，大多数用户实际上只使用它一少部分的概念。它的许多概念含义不清，使用户感到困惑。来自学术界的批评则主要针对它在理论上的缺陷和错误，包括语言体系结构、语法、语义等方面的问题。

目前国内也有不少软件企业在学习并尝试使用 UML。从总体上看，我国计算机界对 UML 的了解还相当初步，但是对它的崇拜程度却远远超过了西方发达国家。人们在学习和使用 UML 遇到和国外用户相同的疑难和困惑时，却不太敢怀疑 UML 有什么问题。所以国内几乎没有批评的声音，偶尔有一点，也会立即被捍卫的声音淹没，即使对 UML 一些最明显的缺点和错误也是如此。

相比之下，国际上对 UML 的讨论和评价则要客观得多。无论是 Internet 上的意见交流，或是每年一次的 UML 研讨会，还是学术期刊上发表的文章，都是既肯定其成绩，又指出其缺点和错误，并且以积极的态度提出建设性意见。在酝酿 UML 下一次的重大发布和筹划。UML 2.0 作为 ISO 标准提案的最近几年内，围绕 UML 的讨论更为活跃和热烈。

1.2.2　面向对象的编程技术(OOP)

面向对象概念对软件解决方案具有莫大的好处，在设计优秀合理的情况下尤其如此。我们可以只编写一次代码而在今后反复重用，而在非 OOP 的情况下则多半要在应用程序内部各个部分反复多次编写同样的功能代码。所以说，由于面向对象编程减少了编写代码的总量，从而加快了开发的进度，同时降低了软件中的错误量。

用来创建对象的代码还可能用于多个应用程序。比如，我们的团队可以编写一组标准类来计算可用资源，然后用这些代码在所有需要同类对象的解决方案中创建对象，如客户定单接口、股票价值报表和发给销售队伍的通知等。

OOP 的另一优点是对代码结构的影响。像继承之类的面向对象概念通过简化变量和函数的方式而便利了软件的开发过程。OOP 可以更容易地在团队之间划分编码任务。同时，由于采用 OOP，辨别子类代码的依附关系也变得更简单了（如继承对象的代码）。此外，软件的测试和调试也得以大大简化。

但是 OOP 也存在一些固有的缺点。假如某个类被修改了，那么所有依赖该类的代码

都必须重新测试，而且还可能需要重新修改以支持类的变更。还有，如果文档没有得到仔细的维护，那么我们很难确定哪些代码采用了父类（被继承的代码）。假如在开发后期发现了软件中的错误，那么它可能影响应用程序中的相当大部分的代码。

面向对象编程在编程思想上同传统开发不同，需要开发人员转变传统开发中所具备的惯性思维方式。对一个有经验的 OOP 开发队伍来说，采用 OOP 的好处是显而易见的。如果您正在考虑转向 OOP，那么您必须保证已经拥有了富有经验的主要开发人员能负责地检查软件中的缺陷和体系结构。

1. OOP 的概念和术语

对象定义 对象是建立面向对象程序所依赖的基本单元。用更专业的话来说，所谓对象就是一种代码的实例，这种代码执行特定的功能，具有自包含或者封装的性质。这种封装代码通常叫做类、对象类，或者模块，或者在不同编程语言中所应用的其他名称。以上这些术语在含义上稍微有些不同，但它们都是代码的集合。

正如我们上面提到的那样，对象本身是类或者其他数据结构的实例。这就是说，现有的物理代码起到了创建对象的模板作用。执行特定功能的代码只需要编写一次却被引用多次。每一种对象具有自己的标识，也就是令对象相互区别的对象名称。

对象并不是类的实际拷贝。每一对象都有自己的名称空间，在这种名称空间中保存自己的标识符和变量，但是对象要引用执行函数的原有代码。“封装”的对象具有自己的函数，这种函数被称作“方法”，而对象的变量则被称为属性。当对象内部定义了属性的时候，它们通常不能扩展到实例以外。

在特定的场合下，有些函数确实会影响类而不是由类所创建的对象。类属性指的是专门设计来保留对象之间所用的值。类方法则用来定义和跟踪类属性。某些编程语言可以让用户调用类的函数而不是创建整个实例。如果函数被分配以标识符（或者句柄），在某些情况下它们可以被视做具有自身权限的对象。不过，在大多数的情况下函数只是用来实现某种结果的方法。

使用对象 在主程序里，定义对象的类通过实例化的方式构造对象。对象所具有的所有方法都可以用来创建所希望的结果，而属性则可以被引用和操作。当对象不再需要的情况下，主程序可以破坏对象。

对象类有一种功能强大的特性，这就是它们可以继承其他类。这就意味着，如果我们编写了某个 potato（土豆）类，那么它就可以继承 vegetable 类而防止我们重新编写已经存在的功能。vegetable 类可用的所有函数都可以被 potato 类使用。进而，vegetable 又可以继承 food（食品）类，以此类推。

某些 OOP 编程语言还具有动态绑定（dynamic binding）的概念。这项技术也被称作多重继承。比如，potato 类可以继承 vegetable 和 starch（淀粉）类。不过这样可能会产生一些问题，如两种类都具有同样名称的一些属性。在具体处理多重继承概念的时候各种语言的方式是不同的，某些语言完全禁用这一概念。

在继承了类后，我们可以通过重载方法来获得希望的结果。比如，vegetable 类可能有一个函数名叫 prepare，该方法主要指导如何备菜。可是，在实例化 potato 类的时候我们希望其中包含与土豆有关的特殊定义，于是创建了一个函数，它的名字和蔬菜类中的备菜函数名一样但却修改了原有的函数行为。如果没有重载 prepare 方法，则用到的是

vegetable 类中的函数。这就叫多态性（polymorphism）。

多态性的另一方面涉及到对象方法的类型一致性问题。这样有助于保证所引用的函数具有以下关系：如果能够实例化 vegetable 对象，那么就应该能够实例化 potato 对象。这是因为 potato 是 vegetable 的子类。可是，因为 vegetable 并不是 potato 的子类，所以反过来的实例化是不允许的。如果实例化了 potato 对象，那么就不需要实例化 vegetable 对象了。

如何定义多态性有各种观点，而其最终用途却是同样的。无论如何，这是一种重要的 OOP 概念。再结合继承技术，显然 OOP 就会具有强大的开发功能了。在后面几章中，我们会陆续提到框架、组件、OO 设计原则等与 OOP 相关的知识。

2. OOP 特性

OOP 的许多原始思想都来之于 Simula 语言，并在 Smalltalk 语言的完善和标准化过程中得到更多的扩展和对以前的思想的重新注解。可以说 OO 思想和 OOPL 几乎是同步发展相互促进的。与函数式程序设计（functional-programming）和逻辑式程序设计（logic-programming）所代表的接近于机器的实际计算模型所不同的是，OOP 几乎没有引入精确的数学描述，而是倾向于建立一个对象模型，它能够近似地反映应用领域内的实体之间的关系，其本质是更接近于一种人类认知事物所采用的哲学观的计算模型。由此，导致了一个自然的话题，那就是 OOP 到底是什么？在 OOP 中，对象作为计算主体，拥有自己的名称、状态及接受外界消息的接口。在对象模型中，产生新对象，销毁旧对象，发送消息，响应消息就构成 OOP 计算模型的根本。

对象的产生有两种基本方式。一种是以原型（prototype）对象为基础产生新对象；另一种是以类（class）为基础产生新对象。原型的概念已经在认知心理学中被用来解释概念学习的递增特性，原型模型本身就是企图通过提供一个有代表性的对象为基础来产生各种新的对象，并由此继续产生更符合实际应用的对象。而原型委托也是 OOP 中的对象抽象，代码共享机制中的一种。一个类提供了一个或者多个对象的通用性描述。从形式化的观点看，类与类型有关，因此一个类相当于是从该类中产生的实例的集合。而这样的观点也会带来一些矛盾，比较典型的就是在继承体系下，子集（子类）对象和父集（父类）对象之间的行为相融性可能很难达到，这也就是 OOP 中常被引用的——子类型（subtype）不等于子类（subclass）。而在一种“所有皆对象”的世界观背景下，在类模型基础上还诞生出了一种拥有元类（metaclass）的新对象模型。即类本身也是一种其他类的对象。以上 3 种根本不同的观点各自定义了 3 种基于类（class-based），基于原型（prototype-based）和基于元类（metaclass-based）的对象模型。而这 3 种对象模型也就导致了许多不同的程序设计语言（如果我们暂时把静态与动态的差别放在一边）。我们经常接触的 C++、Java 都是使用基于类的对象模型，但除此之外还有很多没有接触的 OOPL 采用了的完全不一样的对象模型，它们是在用另外一种观点诠释 OOP 的内涵。

类型　类型及类型系统的起源、研究与发展是独立于 OOP 的。早在 20 世纪 50 年代的 FORTRAN 语言编译器实现中，就已经采用类型系统作为类型检查的一种手段。广义的类型一般被定义为一种约束，也就是一种逻辑公式。而在对类型的研究过程中产生多种方法，如[C&W 1985]等。而代数方法（algebraic approach）是一种非常好的建立类型的形式化规范的方法。代数中的一个类型对应于一系列元素，在它们之上定义代数操作。