面向目标的编程设计
—C++举例

韩 兰　张玉乔
郭爱民　董立新　编译

北京希望电脑公司

面向目标的编程设计

—— C++ 举例

韩　兰　　张玉乔
郭爱民　　董立新　　编　译

北京希望电脑公司
一九九一年九月

★北京市新闻出版局
准印证号：3586-91586
★订购单位：北京8721信箱资料部
★邮　　码：100080
★电　　话：2562329
★传　　真：01—2561057
★乘　　车：320、302、332、路车
到海淀黄庄下车
★办公地点：希望公司大楼一楼往里走
101房间

针对C程序员的面向目标编程基础简介

为了编写出模块化，可移植，可复用的程序，众多的科研人员和程序员都选择面向目标编程（OOP）做为研制方法。然而，面向目标编程是一种刚刚兴起的技术方法，就软件如何设计和构造来说，它是一种全新的思维方法。如果想要充分利用象C++和Smalltalk这样的面向目标的编程语言，就必须深刻地理解面向目标的编程设计的主要原则。在设计阶段，你会感受到OOP的真正作用。

面向目标的程序设计，是对OOP设计的基本概念和技术方法的简要指导，对于那些想要学习OOP设计方法，并具有一定经验的C程序员来说，本书清楚地解释了面向目标编程的主要概念，如目标，类，实体，分级结构和继承等等。

面向目标的编程设计使用一个典型的数据库实例，以此来说明每个OOP标题，并为C程序员提供了一个熟悉的参考点和一种实用的OOP软件研制指导方法。本书前几章侧重数据库系统和模型制造的设计，接着，详细地分析了每个数据库设计的元素，并将其做为目标加以定义。读者通过众多的C++举例，可以了解构造应用程序的过程，以达到清楚地理解每个关键概念的目的。

面向目标的编程设计一书，文笔清晰、流畅、易懂，避免了不必要的技术术语，直接涉及到程序员和系统设计人员的实际需要。本书是掌握面向目标编程实质的必不可少的基础用书。

简　介

近年来，科研人员对面向目标编程（OOP)进行了详细的研究并加以使用。有些人认为它已走向没落，而有些人则认为它是在编程方面的一场实质性的革命。这两种观点争论得相当激烈。我们则倾向于第二种观点。

C++是一种语言，它运用了OOP编程的几种关键成份，允许你使用某些强功能的面向目标(OOP) 的设计方法。在C++中，其目标是编写一种程序员很快便可以使用的语言，并用这一语言进一步扩大其编程知识，而不是从头学习一种新的语言。尽管C++达到了这一目标，且它对程序研制的方法也具有深刻的影响，但是，如果要用与C语言编程相同的方法来用C++研制程序的话，这本身就意味着失败。当然，这并不是说如此编写的程序不能运转，而是这样做未能很好地利用C++。

如果想要学会和掌握面向目标编程，程序员必须首先了解语言是如何工作的。因为，尽管很多程序员知道如何使用这些面向目标的概念，但是他们并不理解使用这些概念的原因何在，这就是所谓知其然而不知其所以然。导致这种情形的原因在于，面向目标的程序在结构上和功能上与非面向目标的程序不同，尽管语言的公共特性是一样的。

编写本书的主要目的是想要探讨以下的课题：即帮助你将用以前所学过的语言解决问题的方法转变为用另一种语言来解决。例如，如果你掌握了C语言，而你又要使用Pascal语言，那么，你真正需要了解的就是如何将你用C语言所做的一切翻译成与其等价的Pascal 形式。如果你掌握了C++或任意其它面向目标语言或系统，那么这一过程就不再是一个简单的翻译过程，而成为一个发展过程了。面向目标系统基本上是基于如下假设而设计的：即系统和程序员分享着一个公共环境，而所有的程序则是对该环境的扩展。C语言就好比是一种原子语言，你可以一个原子，一个原子地描述它，从而构造一个家族。面向目标编程允许你定义象砖、板和钉子这样的材料，然后，再将其组合起来，即用术语定义整个一个家族。

因此，面向目标的语言对研制的影响主要侧重于结构框架，对其语法来说则相对少些。所有的程序都是在此基础上汇编的，所有的程序成份也是在此基础上建立联系的。因此，要想真正实现OOP的功能，最重要的是理解做为其基础的设计原理，而不仅是其语言的语法。

如果你真正理解了面向目标的设计原理和方法，那么你就会理解为什么没有必要用面向目标的语言来实现各种程序。面向目标语言的作用就是进一步地限制支持面向目标的编程。它主要的优点就是提供了一种良好的操作环境。在这一环境中，最重要的特性就是目标的概念，它包含数据，操作该数据的函数，以及与其它目标的联系等。

为了清楚地说明在面向目标系统中运行的设计过程，我们将全面研究整个数据库的设计，使用商品目录，供货商、客户、雇员档案，以及一般总帐。毫无疑问，我们不可能在一本书中包含系统设计的每一个简单概念，但却可以讲解面向目标的设计原理和过程。其中包括错误的步骤和修改过程。并以设计方法为工具说明了面向目标设计是如何进行的，以及程序段是如何根据该设计实现的。

我们试图说明在整个设计过程中，是如何做出某种决定的，为什么会做出这种决定，以及这些决定是如何直接使用面向目标的编程语言发生联系的。

全书分成两个部分：战略和战术。战略部分讲述了面向目标系统的设计，它适用于从C++到smalltalk的任意一种面向目标语言，并说明设计是如何从粗略分类发展到完成产品的详细过程。以及不可避免的修改和解释是如何溶汇到进程中去的。当你读完这一部分时，你将会理解面向目标系统能够在软件研制中所发挥的作用。在第二部分中，在战略部分设计的部分程序将通过C++予以实现。

目　录

第二部分 战 略

第一部分　战略

第一章　面向目标编程简介

如果你不清楚你所编程序的主要目的是什么，那么，我建议你最好不要急于着手编写这个程序。

现在，在一家公司（我们不妨之为Bancroft）的决策人要建立一个联合数据库，该公司是一个拥有许多家子公司的联合集团公司，其经营范围从邮政服务到股票市场应有尽有，无所不包，因此，它需要一种新的软件，该软件不仅能满足是现有多方面的需求，而且还能够适应不断增长的新需求，为了达到这一目的，我们准备用面向目标的编程技术来研制软件。在与该公司经理商讨之后，我们选定了目标，先予实施：

- 商品目录
- 销售额
- 个人数据库
- 含有相应数据的总账目
- 直接装运

1.1　了解问题

简而言之，该公司需要一个联合数据库。因为每个软件包都具有某种局限性，故该公司对市场上现有的任意商用软件包都不大满意。这样，研制一个能满足其要求的软件包的工作便提到了日程上。

读者也许能够看出，我们现在已经遇到了第一个难题，这就是该公司的要求不够明确。也就是说他们没有对软件研制的绝大多数请求清楚地加以定义。当然，随着软件研制过程的不断深入，其请求是会逐步清楚的。这就需要研制者能够充分地利用该公司所提供的信息，然后，再进一步向该公司询问他们是如何进行信息处理的。对于面向目标设计来说，这是不成问题的，因为其设计方法是基于一种连续的进程的，即技术要求，实施和逐步完善技术要求。这种循环存在于许多标准语言中，而在OOP中，它成为某种艺术形式，它有益于程序员和用户。在研制软件的过程中，用户可以与软件交互作用，从而标识出软件能够做什么和不能做什么。而程序员则可以将这些数据做为研制进程的自然成分而获取，并不需要编写专门的测试台，便可以从用户那里获得信息。

由此看来，我们所要做的第一步便是分解用户所陈述的要求。目前的情形是用户提出了一个总的轮廓，也可以说是基本的、总的、具有指导性的要求，而具体的，实用的要求尚不够明确。因此，我们所要搞清的第一个问题便是，每个陈述目标所包含的具体内容是什么？

其中，第一个所关心的问题便是商品目录，它包含着商品目录项的存在。而第二个所关心的问题则是销售额，其中包含着客户的存在。此外，第三个所关心的问题就是个人数据库，

它包含着雇员的存在。最后一个所关心的问题则是总账目，这一问题是基于以上三个问题之间的相互关系，因而现在对此可以忽略，因为，只有当我们了解了前面三个问题之后，我们才能够对最后一个问题做出回答。这里必须切记的是，当我们解决了前三个问题之后，千万不要忘记这最后一个问题。

OOP系统涉及目标，面向目标系统中的目标和在用非面向目标语言编写的系统中任意元素之间的主要区别在于，目标是含有数据和编码的。而在标准C中，是由程序员负责将函数与它们操作的数据连接起来，而象C++这样的面向目标语言当然可以这样做。在标准C中，由函数计算日期和时间，程序员负责确保这些函数能求出正确的数据。而在C++中，有时间目标，它们含有表示时间的数据，以及知道如何处理该值的函数。每当我们遇到一个时间目标，我们就可以假设它含有表示特定时间的数据和知道如何处理该数据的函数。在传统语言中，我们必须告诉程序做什么和如何做。而在面向目标系统中，我们则可以假设系统本身就已经知道如何去做，而我们仅需要告诉它做什么就可以了。当然，这不等于说你绝对不必解释如何做某事。

OOP设计较之计算机模型更注重设计方法，因为它要产生一种与现实具有自然关系的软件系统，而现实就是一种模型。实际上，在OOP编程中的关键字就是模型。程序员的责任就是构造现实的软件模型。如果这项工作做得好，那么，在这些目标间的高级交互作用中所遇到的许多问题便可迎刃而解了。这种情况与传统语言中常见的情况有很大的区别。要理解这一点，请考虑这样一个事实，即所有的软件，不管是面向目标软件还是非面向目标软件，在两种不同级上操作，首先，存在着针对特殊数据的操作，然后才是对作用域的全程操作。

简而言之，程序的许多全程特性不是新建立的，而是基于系统的功能。在面向目标设计中，如果目标专用操作的第一级（如：计算雇员的收入税）设计得好，那么第二级（如：Bancroft拥有的资金）实现起来就相对简单。这是因为全程应用操作的建立是基于应用的专用子元素。

当我们考虑由C编译程序提供的标准库时，整个情况就更为明显了。这些库提供由整个应用程序所用的低级函数，以完成原始操作。为了工作，它们必须清楚地定义。当一个应用程序开始研制时，就需要定义。这是编程的特性。针对该公司的软件，你可以获取3个项目中的每一个，并开始分解它们。为此，应简要地查看下这些项的要求，以及它们之间的关系。

首先，商品目录含有商品项目。当对其进行讨论时，我们要么谈一谈商品目录中的全部项目，要么谈一谈商品目录中的某一特殊项目。因此，商品目录是一个含有许多商品项目的目标，并提供某些可存取特殊项的方法。每个商品项目也是一个目标，可以假设如下：

- 对每个项目，在堆栈中有各个项目
- 每个项目有一个商品编码或成分编号
- 每个商品项目是从一个或多个供应商进货的
- 对每个项目都有一个买卖史

其次，销售额模型含有各种销售额。对每一种销售额，都必须有以下因素：

- 客户订货单
- 有关最后交货的信息
- 有关支付的信息

最后，就是雇员数据库。此时，它含有一个各个雇员的集合。它至少应含有以下内容：

- 有关雇员住址的信息
- 税款信息
- 工作性质和就业履历

此时，面向目标设计扮演了一个主要角色。在构造程序的整个进程中，必须认识到系统中各个目标间的相似之处，并立即加以利用，在该例中，每种情况的某些元素分享一个公共概念，从品商目录系统可了解有关供应商的情况，从销售额系统可了解有关客户的情况，而从个人系统可以了解有关雇员的情况。这些目标（客户、供应商和雇员）中的每个目标都是人，他们都共享公共特性，即：他们都有姓名和地址。因此，在系统中必须有一个更为一般的目标类型，一种定义每种成分一般特性的目标，而它们所拥有的特性是共同的。我们可以说，这些成分是另一个更为一般的目标的所有特性，我们称之为实体。

在面向目标编程中，每种目标是由类定义的，而类可以看成是一个建立以相同方式工作的专用目标的属性单元。类综合在一起构成一个分级结构。从适用于许多专用类的非常一般的特性到适用于单个类的非常特殊的特性，当我们使用类时，我们基于其相似之处将目标综合在一起。在面向目标语言中，类是基于这一相同的概念，如：可将动物分成相关的组。这样，当我们谈及食肉动物时，我们不能仅指狗或狮子，相反，我们应指出狗和狮子所共有的东西，即食肉性。当我们涉及到狗或狮子的详细情况时，它们的确是有着很大差别的。但是，当我们就食肉这一更为一般的含义谈及它们时，它们在食肉性这一点上则是相同的。实体类就允许我们对客户、供应商和雇员这样做。尽管这三个目标有着很大的差异，但它们都具有实体的所有特性，确切地说就犹如狗和狮子都具有特性一样，他们也都有共同的特性，如：姓名、住址等等。这样，我们就不必描述每一个新目标的各个方面了，相反，得们仅需要描述该新目标与我们已定义的类的区别何在就可以了。我们不必专门陈述客户有地址，就如同我们不必专门指出狗食肉一样。客户有地址，因为它们是实体的特性，面实体已含有地址，就如同狗已具有食肉特性一样。

此时，我们对如何构造Bancroft公司所要求的联合数据库的模型的基本思路已经变得愈来愈清晰了。由此看来，该结构至少应含有三种不同的目标集合，它们分别是商品目录、销售额和个人履历。这些集合中的每一个单个的集合，都以相同的方式管理各自类型的元素列表，并提供存取它们的统一方法。然而，这些子类型都具有适用于其各自目的的唯一组织，都使用实体类保存有关它们必须记录在案的各种人员的信息。

这允许就系统的实际操作做出两个至关重要的假设。首先，由于所有信息组是数据集合，所以它们应以相同的方式操作，不管商品目录项，客户记录或雇员记录是否从一集合中检索，面检索机制是相同的。在应用程序的最高级，应用程序希望知道它们是否是集合并理如解如何存取它们的内容。希望最高级视商品目录项的集合与雇员集合完全不同，这会使程序不必要地复杂化。其次，有一个目标类，实体，将客户、供应商和雇员记录在案。该类定义所有人员的共同内容，因此，它应含有：

- 人的名字（包括公司的名字）
- 人的地址（包括公司的地址）

应记住一点，即实体目标不能判定它是否在检查一个人还是一个公司。实体目标本身不仅不知道，而且也不会知道，目标永远以一种统一的方式工作，因此，实体应以相同的方式

工作，而不管它是否含有人或公司。保证这一点的最简单方法是不要把信息放在两者之间存有区别的实体中。面向目标编程允许软件研制以自上而下的方式进行。显然，系统需要一个称之为集合的普通目标。该目标必须以一种统一的方式工作。对这种专用数据库的一切存取都是通过这些集合目标进行的，并保证所有存取以相同的方式执行。我们现在可以忽略最高一级不管，对每一个数据库开始进行更为详细的分解。

在解决Bancroft公司的问题之前，必须说明程序员的某些战术问题。这些问题不是什么实际问题，但是，却是程序员必须在此操作的限制范围，并确切地规定了程序员此时该做什么和不该做什么。

第一个问题是：程序员应该访问哪些信息和程序员不应该访问哪些信息。第二个问题是此时，需要程序员假设什么功能性，且什么可以安全地移到其它地方。

1.2 一般性和特殊性

在我们开始设计专用数据库成分之前，必须清楚地阐明程序员的职责，如前所述，这一设计阶段可描述为查找相似性。例如，在商品目录项中，它们都有一个成分编号和一个说明，它们都没有颜色或大小，面向目标的编程系统使设计者能清楚地区分哪些对所有情况是共同的，哪些对个别情况是专用的。更为重要的是，面向目标的编程系统使设计者将特殊性区别开来，直至一般结构已经构造。因此，设计一个面向目标程序的基本原则之一便是：忽略特殊情况，直至它们不再被忽略。

有些程序员在开始设计时，就立即查找可找到的最为详细的信息，到头来却被众多的，毫无联系的信息压得喘不过气来，而这些毫无联系的信息主要是一些特例，而不是没有特例的相关信息的集合。这是面向目标编程极力要避免的。特殊性是研制的最后一个步骤，当系统中所有相似性已全部用完之后才考虑特殊性。当进行完一般设计，且已知特例已应用之后，继续考虑更为罕见的特例，直到顾客满意为止。如果你概括得好，你可以将特殊性添加到系统中，而不会影响你以前所做的任何工作。

1.3 功能性

在以往的编程过程中，当编写完传统程序之后，便建立了数据结构，而后，处理它们的函数便可直接建立了。在面向目标编程中，并不是这种情况。当建立一个新的数据结构或目标时，就标志着程序员当前职责的结束。例如，许多记账系统使用时间标记来标志每一个需处理的事项。在传统设计中，程序员用一内部时间标记字段定义记账记录，然后再定义处理时间标记的操作。而在面向目标编程中，程序员的职责是伴同事项目标中的时间标记目标而结束。由于时间标记目标是一个特殊的目标，所以，事项目标对在其中执行操作不负责任。这些操作在时间标记目标中得以确认，目标从不关心其它目标的内部详情，仅关心其自身，因为，目标与其它目标通信是根据它们要完成什么而进行的，并假设它们正在与之通信的目标知道如何完成之。如果目标之间用英语通信，最常见的话语是："Don't ask me how to do it, just do it you're the expert.I told you what I want" 目标假设它对时间标记目标的请求会给出它们所希望的时间标记。

当定义了事项目标时，它所包含的时间标记的内部结构便可能被忽略。这并不是说，不需要编写针对时间标记操作的程序段，而只是因为，当设计事项目标时，该程序段不令人感兴

趣罢了。当完成事项目标的设计后，便可以定义时间标记目标，或放置起来以便日后考虑。

1.4 商品目录

对商品目录中的每一项目，不论是衣物、计算机还是书籍，如何进行描述呢？当我们试图标识商品目录中所有项目间的相似性时，上述问题是在设计阶段所必须予以回答的。这里所说的相似性，是指商品目录中每一项目都拥有的特性。我们应再次寻找商品目录项的相似性，标识出所有可能出现的商品目录项中的相似性，从而构造出一个可能特性的列表。其中可包括以下内容：

- 成分编号
- 数量
- 每项的价格
- 项目的供应商
- 销售历史
- 项目的描述
- 项目的颜色
- 项目的DOS版本

其中每种特性是由如下准则测试的：其中某一项目没有该特性吗？如果问题的答案是yes，那么取消特性，因为它对一个商品目录项目标太特殊了。我们在该列表中不能包含某一项目的特性，非除它适用于每个可能的商品目录项。例如，颜色仅适用于有颜色的产品项目，象衬衣，可能是红色的，白色的，或兰色的；而电压则仅仅适用于电子产品；DOS版本又仅适用于计算机系统。然而，成分编号和文本编述则适用于任意商品目录项。这乃是绝大多数传统程序员难以处理的复杂问题。有些程序员可能会建立以下这种结构：

```
typedef struct {
    int part_no;
    char *description;
    .
    .
    .
    int type;
    int color;
    int_size
} Inventory_Item;
```

然后，他们再通过软件分解特殊测试，以检查结构的类型字段。如果是1，则假设有一个有效颜色；如果是2，那么则有一个有效尺寸；如果是3，那么既有颜色又有尺寸。他们也许会用0表示既无颜色也无尺寸。有时，为了节省空间，也许会用单个字段替换尺寸和颜色字段，其内容由类型字段定义。

目标本身仅仅表示一种方式，它们不根据其内容以不同的方式工作。目标永远是一致的——前面的机制不管是什么，但都是一致的。我们使用目标的意图就在于：我们可以根据它们以一种正规统一的方式工作。

此时，你也许会考虑颜色和尺寸是如何表示的，颜色和尺寸是某些商品目录项的特性，但不是所有商品目录项的特性。然后，我们可以假设，具有颜色的商品目录项是特殊的商品项，我们只有知道所有商品目录项的特性是什么，才能定义带颜色的商品目录项的特性是什么。

设计中的某些元素，如商品目录项，仅具有销售额历史，是有些不够清楚。当销售额历史无明确的定义时，一个项目如何才能有销售额历史呢？这说明面向目标设计的进程试图牵着程序员的鼻子走。当你不能清楚地定义一目标特性，但你知道它是一目标的一个特性时，你就可以放心地假设你正在涉及一种或另一种新的目标。在这种情形下，你不必完全定义新的目标，因为事情很简单，你没有足够的信息，你知道你需要一个新目标。因此，在进行设计时，你应应时刻记住这一点，等待着。这样，你便可以清楚地定义新的新目标了。

1.5 销售额

销售额永远是针对客户而言的。因此，销售额数据库实际是客户数据库。当回答诸如："去年卖了多少册书？"这样的问题时，有许多种可以选择的答案或回答问题的方式。最常见的方法是查看一下商品目录，看看一共卖掉了多少册书。销售额数据库可能没有什么，而只是商品目录数据库的子集，就公司管关心的销售额数据库而言，公司真正想知道和了解的是哪些人买了这本书。因此，销售额数据库实际是客户的数据库，它可以用客户目标表示。

客户目标用于记录有关各个客户的信息，如：他们住在哪里？过去他们在本公司订购过哪些货物？本公司向他们发送过哪种邮件？等等。再一次查找相似性，由此便产生了如下列表：

- 客户的姓名和地址
- 客户的订购历史
- 对客户的邮寄历史

首先，应注意到，客户的姓名和地址被视为单一项目。以前，在系统中有一个实体目标，它定义有关客户，雇员和供应商的一般信息。因此，很显然，每种客户目标便是一种实体目标。从定义实体目标开始，保证获取姓名和地址。从而，我们知道，在这一级，我们可以把姓名和地址合并在一起，因为，我们实际上一不必处理它们，它们已在实体级得到处理。

这是我们构造类分级结构的进程。一个客户就是一种实体，特别是当它为实体的导出类，而实体又是客户类的基本类时。当一个导出类使用其基本类的数据和基本功能，以完成某种操作的时候，实体提供姓名和地址数据元素，以及存取和处理的方法。当它使用其基本类的性能进行工作时，通过定义客户类为实体的导出类，它可立即与这两种数据项一起工作，也就是说，客户类继承了实体类的这些性能。

继承基于下述事实，即目标由数据和操作数据的函数所组成。回忆一下前面有关假设日期目标的讨论，它包含表示特定时间的数据和操作该数据的函数。例如，返回任意给定日期为星期几的函数将是日期目标。当我们把确切的机制留给后面一章时，重要的是记住这一事实。被称之为成员函数的某些函数不象标准C函数那样理所当然地存在，相反，这些函数严格地在一特殊目标范围内定义，如函数day of week ( ) 是在日期目标范围内定义的。因此，我们可定义日期目标的新导出类，并且我们可以假设这些新类继承了日期类的原始性能。我们再谈谈我们所举的动物的例子，理由是一样的：我们不必严格地阐述狗或狮子食肉，由于

狗和狮子都是食肉动物，而食肉动物吃肉，所以狗和狮子继承了食肉这一特性。

剩下的三种数据项仅由客户类使用，而对实体类是不能获取的。严格地说，继承是一种从一般级移至较为特殊级的方式。由于所有了种项目表示复杂的数据，所以，设计进程再次需要新的目标。我们先把对这三项的实际意义的探讨留给后面的章节，现在我们仅需认识到它们需要新的目标来表示即可。

1.6 雇员

最后是雇员数据库。雇员就象客户目标一样是实体目标的一个特例。但雇员不是客户，它们具有不相同的数据。然而，我们用与定义前面所有类的相同方式定义雇员类。我们标识类的这些特性，而这些特性适用于我们可以考虑到的每一特例中。因此，继续寻找相似性，通过分析每一雇员的特性，便可得出如下列表：

- 姓名和地址
- 工作分类
- 当前工资
- 税款信息
- 福利信息
- 履历表

在定义雇员类时，我们已确保它是一种实体，或从技术角度说，它是实体的导出类。注意我们的定义包括雇员的姓名和地址。我们知道我们从实体基本类中继承。当你定义一个新类，而该类又是由现存类导出的时候，新类必须含有基本类的所有元素。在该例中，如果我们由于某种原因使每个雇员没有地址，那么，这就意味着雇员类不能从实体类导出，或实体类不会含有地址特性。因此，我们可以看出，查找相似性对面向目标的程序员来说并非一件容易而简单的事情。

我们可以在有关雇员的税款信息和到目前为止我们已定义的任意类之间建立一种清晰的关系。因此，我们可以假设，它运用在一种新类，即Employee TaxInfo中。出于最简单的原因，我们做出如下假设，即：在这一新数据和我们以前实现的目标之间不存在相似之处。请记住，我们构造了存在于现实生活中的某些事物的模型（目标），因此，在设计过程中，我们在现实生活中的经历和阅历非常有助于我们做出设计决定。简而言之，我们可以假设税款信息必须至少由一个专用目标表示。因为，税款信息是极为复杂的。我们知道，每个雇员都有一个税款目标，而它含有该雇员的免税数额。这说明了面向目标设计的另一特性，即我们可以依赖于这样一个事实．目标类似于现实生活中的目标或它们所表示的概念。如果它在现实生活中较为复杂，那么它就好比一个复杂的目标或目标集。在任意情况下，我们知道，如果它较为复杂，我们就不要把它表示成一个ASCII字符串，除非我们别无选择。

工作分类是另一目标类EmployeeJobclass，出于简单的原因，每个雇员仅有一个工作分类，但在现实生活中，却有许多种分类，因为，有的雇员可能有多项工作做。如果它是一个简单目标或一个目标列表，那么，每当存取该特性时，就必须进行测试，该例就是用来说明面向目标编程的另一重要特性。由于仅有一个雇员目标可以存取该数据，所以，仅有一个雇员目标必须判定是否有一个或多个工作分类。将它与绝大多数程序比较一下，在这些程序中，对该元素的引用必须进行测试。这称为封闭，在此，程序的特性在各个目标中确认，并

且不通过系统分散开。系统中的所有其它目标只是问一下雇员目标的工作分类。

福利信息是另一种特殊目标，该目标含有有关保险等福利信息。履历表记录则是一个更为复杂的目标。它含有雇号的考查情况，以前的工作分类，及以前的福利信息。由于它们是新的目标，所以，对它们的定义可以延缓。

1.7 小结

1）请记住这是模型构造。在你所建立的目标和它们在现实生活中所表示的目标之间存有一种直接的关系。它们同样地发挥作用。计算机的观点是不重要的，而现实生活的观点才是重要的。

2）目标以一种统一的方式工作，而不考虑它们所包含的数据。构造就是构造，不论是构造教堂还是办公室。

3）认可和利用相似性。

4）尽可能忽略特殊情况。当建立标准后，才考虑特殊情况。

5）不要让一个目标告诉另一个目标如何进行工作

第二章 目标特征

在这一章中，我们将揭示更多的数据库内部结构。由于没有哪个设计是不可挑剔的，除非其所有成分是完好无缺的，所以才可能是完美的。但是，这并不是说，在研制过程中延缓定义就是不好的，它也是好的。请记住，有些使用传统设计方法的程序员往往有着共同的弱点，即：当人们问及这些程序员时间时，他们会详细地解释钟表的概念。这主要是由于他们使用计算机的经验所至。这里，他们必须搞清和牢记的概念是，告诉别人时间和让他人理解时间概念是两个不同的知识领域。比方说，我有一块手表，并常常告诉别人现在是几点了，这并不等于说我知道这块表的内部工作原理。手表是一个目标，它能够让我们知道目前的时间，而无需我们来告诉它应如何运转。

此时，我们已定义了该公司所需的联合数据库的基本结构。这一结构是非常基本的，因为它是由少量具体数据所组成的。正如以前所述，在设计面向目标系统中，一次处理一个目标。当其它不同的目标由当前定义的一个目标所用时，不必定义它们。

2.1 集合

在第一章中，陈述的设计依赖于我们称之为集合目标的概念，该目标用于综合其它目标，因此，它们可做为单一的实体引用。在数据库中，这些目标都含有由某些元素索引的目标。商品目录通过成分编号检索，客户可通过客户ID检索，而雇员由其名字和工作描述进行检索。这并不排除做些通过姓名查找客户之类的事情，它只是指出各个客户是如何在集合中编码的。其目的是使用“用户”目标所使用的索引。如果它们不能提供该索引的值，那么，这些目标将不得不通过客户记录手工查找，以此来得到它们所需要的信息。

由于这一机制，便使得实际所需要的集合类型是某种索引集合类型。以前使用过面向目

标语言的人也许遇到过集合集。那些拥有这一系统的人也许能使用一个现存的索引集合类。而对某些人来说，则不得不从头研制一个集合类。请注意，我们是如何假设两种分级结构相关类的存在的，特别是应注意基本类集合和从该基本类导出的索引集合类。然后，再次查找相似性。在该例中，事实上，我们需要建立其它目标的同类集合。我们还知道，并非所有集合都需要索引。通过使用类分级结构，可将一般的概念与支持索引的特殊集合区分开来，以此，我们便可清晰地描述我们可能遇到的各种集合。下面我们便使用类分级结构。

· 所有集合都含有元素列表

· 有些集合还含有关键字列表

类之间的分级结构关系允许我们这样做，因为，基本类——集合——含有一个独立的目标组。由于对基本类为真，因此，由基本类导出的任意类亦为真。还有，有些集合含有关键字。这些关键字通过导出类索引集合，实现该性能，以便存取集合的专用元素。因此，我们仅需说明索引集合是基本类集合的一个导出类，它含有一个关字列表。索引集合继承了集合的性能，并将其扩展以支持关键字。

这又带来了另一个面向目标编程的程序员所必须加以考虑的问题。即当构造一个类时，构造类是为应用程序解决某一特殊问题，还是类要克服系统的某些弱点？例如，客户目标与Bancroft数据库相关。当其它程序也许有一个客户定义时，它们会完全不同地与之相互制约。集合类是不同的，它是一种通用类，用来将一组目标集合起来，而不管其各自的类型。不管各个应用程序的目的，从商品目录控制到字处理，但集合目标的概念是一样的。

正如目前绝大多数有关OOP的书籍所说，目标的主要优点之一就是它们可以复用。C函数也是如此。所不同的是表示数据结构和可对这些结构进行操作的函数的目标表示功能软件包。软件包永远是统一的，它们与其它目标一样，而不管给定目标的目的是什么。数据定义和成员函数是紧密相连的，使用数据会自动地保证能够使用对数据进行操作的函数。为了支持Bancroft数据库中的索引集合，我们必须运用索引算法。数据结构和大量的其它细节一旦完成这一操作，一切都装入一个称为“集合”的黑盒子中。它工作时，你绝不会再被迫查看其中的内容。这类似于使用象printf（）这样的C库函数。当所有的C程序员熟悉了这一函数时，没有多少人理解其内部操作情况。他们也不必知道其内部情况便可使用。这是目标为你提供的。你获取了相同的效果，而你却不必象printf（）的研制者那样花费很大的精力。

为Bancroft数据库研制的集合有两个优点。基本类集合基本上未按顺序编排目标，目标可以添加到集合中，也可以从集合中删除。它的元素可按它们插入的顺序检索。寻找专用元素需要从头开始，并通过每个目标向前查找，直到发现所希望的元素，或直到寻找到数据结尾处。

集合的导出类索引集合将一关键性能添加到基本类中，索引类通过使用关键字允许添加，删除和检索集合中的元素，它允许检索集合中的专用项而不需要请求目标进行实际查寻。

对集合目标来说，需要以下数据项：

· 当前在集合中的项目个数

· 此时集合可存放的项目个数

· 对集合数据所存地址的引用

· 当前工作位置