HOPE

（共五册）

Turbo Pascal 6.0 面向对象程序设计参考手册与程序设计技巧

其它四种书是

- 用户指南
- 程序员指南
- Turbo Vision 指南
- 库函数参考指南

中国科学院希望高级电脑技术公司

石 放 编译
吕肖庆

5

Turbo Pascal 6.0 面向对象
程序设计参考手册
与
程序设计技巧

序　言

面向对象程序设计是当今世界软件工程程序设计的潮流。在 IBM PC 程序设计语言中，支持面向对象性能的语言很少，Turob Pascal、C++、Turbo C++、ADA、Quick Pascal 等。其中大多数要么性能不强，要么开发环境不好。

Turbo Pascal 6.0 在 Turbo Pascal 5.5 尝试的基础上，扩展了 ANSI Pascal 标准，具有很强的面向对象的性能，使 OOP 建立在良好的集成环境之上，能对支持 OOP 特有的数据结构——方法和类进行源级调试，完成单步执行、断点执行、变量和表达式检查和跟踪，调试时给变量和数据结构赋新值。在 Turbo Debugger 2.0 中，增加了支持 Turbo Pascal 5.5 6.0 面向对象性能的调试，对 OOP 的继承性、多态性、封装性，静态方法和虚拟方法进行调试和剖析。

本书的第一部分是 Turbo Pascal 6.0 软件面向对象部分的联机手册，第二部分是用 Turbo Pascal 6.0 进行程序设计的技巧。前面是进行 OOP 程序设计不可或缺的，后面是进行 OOP 程序设计必需掌握的技巧和　实例教材。

在成书过程中为了使读者满意，本书经过认真编译而成。但由于时间仓促，错误难免，望读者赐教。

感谢希望公司经理秦人华、杨淑欣在本书出版过程中所给与的帮助。

编译者

一九九一年四月

目　录

简　介

Turbo Pascal 以高速度为根本提供了一种功能强，效率高的面向对象的编程方法。除了原已深得用户信赖的特性之外，新版的 Turbo Pascal 又推出了下述前途无量的编程技术：

. 最高效率的静态对象和最大实时灵活性的动态对象。
. 静态方法和虚方法。
. 创建和回收对象的构造过程和析构过程(它能节省编程时间，并提高代码的可读性。)
. 对象常量——静态对象数据被自动初始化。
. 高速度——Turbo Pascal 6.0 的编译速度更快。
. 改进的覆盖管理程序(减少了磁盘 I／O 操作，提高了覆盖速度)。
. 增强的帮助屏幕，允许用户将例子剪贴到自己的代码中。
. 联机施教，引导用户步入 Turbo Pascal 的集成开发环境。

Turbo Pascal 6.0 中面向对象的拓展得益于 Larry Tesler 的“对象 Pascal 报告”(“Object Pascal Report” APPLE，1985)和 BJAME Stroustrup 的“C＋＋程序设计语言”(“THE C++ Programming Language”，1986，ADDISON－WESLEY)。

关于本手册

本手册的内容是 Turbo Pascal 6.0 中面向对象的新增特性。要了解 Turbo Pascal 的其它信息，可参阅 Turbo Pascal 用户手册(“Turbo Pascal User'S Guide”)或 Turbo Pascal 参考指南。(Turbo Pascal Reference Guide)。

以下是本书各章及附录的提要：

. 第一章：“OOP 概要”介绍面向对象程序设计的主要概念，即：对象与记录的区别，封装数据和代码的优点，继承性，多态性，静态对象与动态对象实例。采用实际例子展示面向对象的程序设计原理。
. 第二章：“面向对象的调试”概括了 Turbo DEBUGGER 中为支持 Turbo Pascal 6.0 所做改进，包括了对象的考察和对象层次结构(谱系)窗口。
. 第三章：“Turbo Pascal 6.0 的语言定义”包括了 Turbo Pascal 中所有面向对象扩展部分的规范定义。
. 第四章：“覆盖”论述对 Turbo Pascal 覆盖管理程序的改进。
. 第五章：“Turbo Pascal 内核”解析 Turbo Pascal 6.0 中面向对象的特性是如何实现的。
. 附录 A：“新的和有改动的错误信息”列出了面向对象的 Turbo Pascal 专有的新增编译错误信息和警告信息。

安装

用户要做的第一件事就是在自己的系统上安装 Turbo Pascal。一套 Turbo Pascal 包括

了运行集成环境式和命令行式编译器所必需的所有文件程序。Install 程序可将 Turbo Pascal 安装在用户系统上，它可以是硬盘系统也可以是软盘系统。

Install 将帮助用户完成安装过程，用户只须一步步地按照屏幕上出现的指令去做就行，不过，请认真地读懂这些指令。如果是在软盘上安装，而不是硬盘，那么手头上就要准备好至少 4 张空白的，已格式化的 360K 软盘。

安装过程：

1.将标有 InstallATION DISK 的产品磁盘插入驱动器 A。

2.键入 A：并按 ENTER 键。

3.键入 Install 并按 ENTER 键。

从现在开始，只须按屏幕上 Install 显示的指令去做…

当 Install 运行完毕后，就可以启用 Turbo Pascal 了。

在试用完 Turbo Pascal 的集成环境后，也许用户希望重设其中一些选项。使用 TINST 可以完成这一任务，该程序在用户手册的附录 D 中有讲解。

特别注意事项

. 若使用了 Install 的 UPGRADE 选项，6.0 版本的文件将重写所有与其同名的 5.0 版本中的文件。

. 如果将图形文件安装到一个单独的子目录中(如 C:\TP\BGI)，要在调用 INITGRAPH 时，为驱动程序和字体文件指明全路径，如：

```
InitGraph(Drive,Mode,'C:\TP\BGI');
```

. 如果 GRAPH.TPU 未在当前目录下，为编译 BGI 程序，必须用 OPTIONS／DIRECTORIES／UNINDIRECTORIES 命令将它的地址加入单元目录中。(也可以用命令行编译器的／U 选项。)

. 如果无法读出 Install 或 TINST 程序显示的文本，可以用它们接受的可选命令行参数强制换为黑白方式。

```
.A: Install  /B        强制Install进入BW80方式。
.A: TINST    /B        强制TINST进入BW80方式。
```

如果用的是 LCD 屏幕或一个有彩色图形适配器和单显的系统或复合监视器，也许不得不用／B 参数。要想知道如何永久地强制集成环境在 LCD 屏幕上(或 CGA 和单显／复合监视器的组合)使用黑白方式，请参阅用户手册上的注意事项。

联机帮助

用户可以获取有关集成环境和语言特有术语的联机帮助。在一菜单条目或窗口中，按 F1 即可获得帮助，再按 F1 可获得帮助系统的主索引。

语言特有的帮助只能在编辑窗口中按 CTRL－F1 得到。帮助的内容有：Pascal 保留字的语法，指定过程或函数的参数和用法，将例子剪贴到用户文件中以及查看编译器的转换开关等等。

要获取语言特有帮助，在编辑窗口中将光标定在欲了解的术语上，然后按 CTRL--F1。

从帮助(文件)中剪贴，仅须以下几步：

1.一旦调出了帮助屏幕，要拷贝其中的内容，按 C，激活光标，可将其定位在帮助屏

幕的任意位置上。

2.将光标放在欲拷文本的起始处，按 B 开始拷贝，然后用向上，向下，向左和向右的方向键移动块尾(同时待拷贝文件呈高亮度)。再次按 B 将光标位置重设为块首。

3.按 ENTER 键，结束剪贴，文本已被拷入了用户的编辑文件。

4.拷入编辑器的文本被标记成块，因而很容易移动这个剪贴块。

第一章 OOP概要

面向对象的程序设计(Object-oriented Program 简称 OOP)，这种方法恰如其份地模拟了我们的各种工作方式。它是前一时期崭新编程语言的自然发展，与以前在结构化程序设计中所做的尝试相比，它使结构化程度更高；与以前的数据抽象和细节隐藏相比，它的模块化和抽象度也略胜一筹。

三个主要性质就能刻画出面向对象的程序设计语言。

. 封装性：将域(RECORD)与过程和函数结合在一起，构成一个崭新的数据类型——对象。

. 继承性：定义一个对象并用它建立一个后辈对象的层次结构(谱系)，其中每个后辈可继承并使用其所有先辈的代码和数据。

. 多态性：给某个动作定一个名称，而该名称可由对象层次结构上下共享，即层次结构中的每个对象以适合于自己的方式执行这一动作。

Turbo Pascal 6.0 的扩展充分发挥了面向对象程序设计的能力，该语言熔合了更多的结构化和模块化特性，更强的抽象性和可重用性，这些特点合在一起就大大强化了编码的结构化程度，而且更易于扩展和维护。

多年来，一些考虑编程问题的习惯和方法已形成了标准，但是由于面向对象程序设计的挑战，不时地要求用户放弃这些东西，一旦这样做了以后，就会发现 OOP 不仅简洁明了，而且擅长于解决困扰着传统软件编制的难题。

曾用其它语言进行过面向对象程序设计的人员：请放弃以前对 OOP 的印象，以 Turbo Pascal 6.0 自己的术语学习其面向对象的性能。OOP 并不是单一的方法，它是多种想法的延续。从其对象角度看，Turbo Pascal 6.0 更象 C++，而不象 Smalltalk。Smalltalk 是一个解释器，而从一开始 Turbo Pascal 就是一个纯正的本机代码编译器。本机代码编译器与解释器的工作方法是不一样的(它要快得多)。Turbo Pascal 从设计上讲，是一个开发工具，而不是研究工具。

对 OOP 一点概念都没有的人员须注意：

这一点并不坏。最近几年，太多的虚假和混乱，再加上不少人大肆谈论自己并不明白的东西，已经严重地将水搅混了。用户应尽力忘掉别人告诉你的有关 OOP 的东西，而要学到 OOP 的精华，最好的方法(事实上，也是唯一的方法)就是：坐下来，自己用一用。

§1.1 对象

环视一下周围，对象比比皆是。如一个苹果，如果用软件术语如何来描述一个苹果呢？首先，将其分成几个方面：令 S 表示表皮的面积；令 J 表示所含汁液的体积；令 F 表示果实的重量；令 D 表示种子的数量……若不以上述方式考虑，而从画家的角度出发，将看到的苹果画下来(在计算机上)，苹果的图象当然不是苹果，它只是平面上的一个符号。但它并没有被抽象地看成 7 个数(指颜色值，这些数值各自发挥作用并独立地存在数据段中某个地方。)这个苹果的各组成部份仍保持着联系，一种它们相互间不中缺少的联系。在我们生活的世界中对象模拟了其元素的特性和行为。它们是基本的数据抽象(目前是这

样的)。

一个苹果由几部分构成，可一旦分开，它就不再是苹果了。当各种东西都放入一个包装物中，同时，部分与整体，部分与部分之间的关系又是很明确的，即称做封装。这一点很重要，稍后还要讨论封装性。

注：对象将其所有特性和行为保存在一起。

与封装性同样重要的是：对象能够从所谓的前辈类中继承特性和行为。这显然是一个飞跃，继承性也许是面向对象的 Pascal 与现在编程用的 Pascal 之间唯一的最大不同点。

§1.2 继承性

科学之目的在于描述世界的变化与发展，大量的科学工作，为了达到这一目的，真正要做的就是描绘出一个家族树(谱系图)。当昆虫学家们用瓶子装着一只从未见过的昆虫，从亚马逊归来时，他们最起码的要求就是要在由所有昆虫学名汇集成的巨大图表中的合适位置。行星，鱼类，哺乳动物，爬行动物，化学元素，亚原子微粒和外界星系都有类似的图表。它们看起来很象家族树，最顶上是单一的总类，总类以下是数目递增的子类，它们不断地做着扇形扩展以达到多样性的极限。

例如在昆虫的类别中，有可见翅膀的昆虫和有隐藏翅膀或根本没有翅膀的昆虫。在有翅昆虫下面又分很多类别：蛾子，蝴蝶，苍蝇等等。每个类别都有大量的子类别，在子类别下面又是更多的子类别(参考图 1.1)

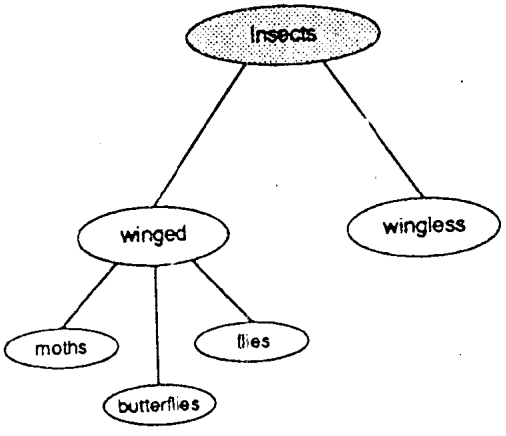

图 1.1 昆虫局部分类图

这个划分类别的过程称为分类学。用它来比喻面向对象程序设计的继承机制，就是一个认识 OOP 的良好开端。

在试图对一些新动物或对象分类时，科学家们会问：它与其一般类的其它成员多少相同之处？又有多少不同？各个不同的类有其一组行为和定义该类的特性。一个科学家可以从标准家族树的顶点开始向下寻访各分支，不断地提出上述问题。最高层是最一般的，因而问题也是最简单的："有没有翅膀？"往下，每一层都比其上一层更特别，同时也就更缺少一般性。最终，科学家为了区分两种昆虫，也许会达到这种程度——数一数昆虫后腿·

第三节上有几根茸毛，这样的分类就真是到家了。(幸好我们还不想成为昆虫学家)。

需要记住的一点是：一旦定义了某种特性，在该定义下面的所有类别都拥有这一特性。所以，如果将某个昆虫签定为双翅类的一员(苍蝇)，就不必再声明苍蝇有一对翅膀。因为我们称为苍蝇的这种昆虫从它的类中继承了这一特性(双翅)。

面向对象的程序设计就是为数据结构建立家族树的过程。面向对象的程序设计为传统语言，象 Pascal，所增加的特点之一是一个数据类型可从更简单、更一般的类型中继承特性。这种机制叫做继承性。

§1.3 对象：能继承的记录

用 Pascal 的术语来说，对象很象记录，它将几个有关的数据元素封装在一个名字下面。在图形环境中，可以将图形屏幕位置的 X、Y 坐标组合在一起，并给它以记录型的名字 LOCATION：

```
LOCATION = RECORD
    X,Y : INTEGER;
END;
```

其中 LOCATION 是记录类型，这个名字单元被编译器用创建成记录变量。一个 LOCATION 型的变量就是一个 LOCATION 类型的实例。术语“实例”在 Pascal 的领域内不时地用到，但 OOP 人员要一直使 用它，借助于术语“类型”和类型的“实例”很有利于开始认识 OOP。

对于类型 LOCATION，可有两种用法：当需要独立考虑 X 和 Y 时，可将它们分别作为记录的 X 域和 Y 域来看待。另一方面，要将 X、Y 坐标合在一起以指定屏幕一个位置时，可以把它们总起来看成 LOCATION。

如果想在由 LOCATION 记录所描述的屏幕位置上显示一个亮点。在 Pascal 中可再用一个 BOOLEAN 型域指出给定位置是否有一个加亮的象素，将它作为新的记录类型：

```
point  = record
    x,y : Integer;
  Visible: Boolean;
end;
```

如果多考虑一点，可以在 POINT 类型中创建 LOCATION 类型的域，将记录类型 LOCATION 封装在里面：

```
Point  = record
     position : location;
     visible  : boolean;
end;
```

Pascal 程序员总爱这样做。但有一件事这种方法办不到，它不能、迫使程序员考虑软件所处理的事物本质。这需要提出如下的问题：“屏幕上的点与屏幕上的位置有何不同？”回答是：一个点是一个加亮的位置。仔细看一下这个判断，就能体会到“一个点是一个位置”，即点的定义就蕴含着该点的位置。(归根到底，象素只存在于屏幕上)，在面向对象的程序设计中，我们要认识到这种特殊关系。因为所有的点都隐含一个位置，可以说类型 POINT 是类型 Location 的一个后裔，POINT 要继承 LOCATION 的所有东西，并加入属于

POINT 的新东西，就构成了自己的新面目。

一种类型从另一种类型继承特性的这种过程被称作继承性。继承者叫后辈类型；被后辈类型继承的类型叫前辈类型。

人们熟悉的 Pascal 记录类型不能继承，然而 Turbo Pascal 6.0 扩展了 Pascal 语言，支持继承性，其中一项扩展就是引入了一个新的数据结构类型，用一个新的保留字，Object，来定义。一个对象类型可以用 Pascal 记录的模式定义为一个完整而且独立的类型，也可以定义为已有对象一个后裔，这只须将前辈类型的名字放入保留字 Object 后面的参数中即可：

```
type
     location = object
     x,y : Integer;
end;
point = object(location)
     visible : boolean;
end;
```

注意：这里以小括号表示继承关系。

其中，LOCATION 是先辈类型，POINT 是后辈类型。一会儿便会看到，这个过程可以无限地持续下去，即，可以定义类型 POINT 的后辈，甚至 POINT 后辈类型的后辈，以此类推。设计一个面向对象的应用程序，很大一部分工作就是建立这种能表达出应用对象家族树的对象层次结构(或称对象谱系)。

所有从 LOCATION 那里继承的最终类型都称为 LOCATION 的后辈型，而 POINT 则是 LOCATION 的直接后辈之一，反过来说，LOCATION 是 POINT 的直接先辈。一个对象类型(很象一个 DOS 子目录)可以多个直接后辈，但只能有一个直接先辈。

由定义可以看出，对象与记录的关系很密切。新的保留字 Object 大概是它们最明显的不同点了，但除此之外，还有许多差异，而且有些差异还十分费解。

例如：LOCATION 的 X 域和 Y 域并没有明显地写入 POINT 类型，但借助于继承性可以讨论 POINT 的 X 值，就象可以讨论 LOCTION 的 X 值一样。

§ 1.3.1 对象类型的实例

对象类型的实例可以象 Pascal 中的任何变量声明那样被声明，即可以作为静态变量又可以作为在堆上分配的指针参量。

```
type
     pointer = ^point;
var
     startpoint : point;          {readly to go!}
     Dynapoint  : pointptr;       {must allocate with new before use}
```

§ 1.3.2 对象的域

可以象存取一般记录域那样存取一个对象的域，也可以通过 WITH 语句或引用圆点。例如：

```
mypoint.visble := FALSE;
with mypoint do
begin
    x := 341;
    y := 42;
end;
```

开始时只须记住：被继承的域只能作为给定对象类型中声明的域来存取。例如：尽管 X 和 Y 不是 POINT 声明的一部分(它们是从 LOCATION 类中继承的)，但能通过声明它们隶属 POINT 来指定它们：

```
MYPOINT.X=7;
```

请记住：一个对象的继承域因为是继承来的，所以不能简单地直接引用。

§1.3.3 好的习惯作法

尽管可以直接存取一个对象的域，但这不是一个特别好的想法。面向对象程序设计的原理要求尽可能地不涉及对象的域，这个限制初看起来好象是专断而且刻板的，但它正是本章力图建立的 OOP 全景的一部分。也许取得一致的认识还有一段过程，但迟早程序员都将看到这一新规定对养成良好编程习惯意义深远。现在，只要记住：避免直接存取对象的数据域。

既然这样，又如何存取对象的域呢？用什么来设置或读取它们呢？

回答是：尽可能地使用对象的“方法”(METHODS)来存取对象的数据域。方法是在对象中声明的一个过程或一个函数，它与该对象是紧密结合的。

注：一个对象的数据域是它所知道的东西，它的方法是它所作的事情。

§1.4 方法

“方法”是面向对象程序设计中最引人注目的特点之一，并已被一些人所习惯。现在，回到那个结构化编程中遇到的老大难问题上——初始化数据结构，考虑下述记录的初始化任务：

```
location = record
  x,y : Integer
end;
```

大多数程序员要用一条 WITH 语句给 X 和 Y 字段赋初始值：

```
Var
    mylocation : location;
  with mylocation do
    begin
        x := 17;
        y := 42;
    end;
```

这段程序能够完成任务，但它却与一个明确的记录实例 MYLOCATION 紧密相连。如果有多个 LOCATION 记录要初始化就需要多个 WITH 语句重复地做这一同样的工作。自

然下一步的改进是建立一个初始化过程，将 WITH 语句一般化，把 LOCATION 类型的任何实例作为传给它的参量包括进来。

```
procedure Initlocation(var target : location;
                                newx,newy  : Integer);
begin
     with target do
     begin
          x := newx;
          y := newy;
     end;
end;
```

这段程序当然能很好地完成任务，但它使人感到这种程序总还是多少有点浪费，这一点也为 OOP 早期的倡议者们感觉到了。

这种感觉实际上是：既然，已经明确设计了用于 LOCATION 类型的 INITLOCATION 过程，为什么这里还要不断地指明 INITLOCASTION 所用于的记录类型和实例呢？应该存在某种将记录类型和为其服务的代码“焊接”成一个“无缝”整体的途径。

现在已找到了这种途径。它被称为“方法”(METHOD)。一个方法就是一个与给定类型紧密结合的过程或函数，其紧密程度相当于该方法是被一条不可见的 WITH 语句所包围着，它使该类型的实例可通过自己的方法就在对象内部存取数据域。类型定义中包括了方法的头部。方法的完整定义要由类型名加以限定。对象类型和对象方法是新数据结构(所谓对象)的两个方面：

```
type
     location = object
          x,y : Integer;
     procedure Init(newx,newy : Integer);
end;

procedure location.Init(newx,newy : Integer);
begin
     x := newx;       {the x field of a location object}
     y := newy;       {the y field of a location object}
end;
```

至此，要初始化一个 LOCATION 类的实例，只须简单地调用它的方法，仿佛该方法也是记录的域一样。(从某种实用意义上看是这样)：

```
var
     mylocation : location;
     mylocation.Init(17,42);       (easy,no?)
```

§ 1.4.1 代码与数据的结合

OOP 最重要的宗旨之一是：程序员在进行程序设计时，把代码和数据结合在一起考