S. J. 扬 著
田淑清 谭浩强
周朝龙 张伟民 译

ADA导论

清华大学出版社

ADA 导 论

S.J. 扬 著

田淑清 谭浩强
周朝龙 张伟民 译

清华大学出版社

内 容 简 介

Ada语言是美国国防部的标准高级语言，它具有如 Pascal 等通用语言和某些专用 语 言 的 长处，既具有通用控制结构，又具有定义数据类型和分程序的能力。它可使用抽象的数据类型并易于控制并行任务和处理异常情况。本书对Ada语言作了系统全面的详细解释，并用例子进行说明。结合每章内容列举了使用Ada进行程序设计的实例。全书内容丰富，由浅入深，易学易懂，因此十分适用于作为高等院校学生和从事 程序设计工作的人员学习Ada语言的教课书。

ADA 导 论

S. I. 扬著

田淑清　谭浩强
周朝龙　张伟民　译

☆

清华大学出版社出版

北京　清华园

北京昌平县振南排版厂排版

中国科学院印刷厂印装

新华书店北京发行所发行

☆

开本：787×1092 1/16　印张：20.75　字数：530千字

1988年11月第1版　1988年11月第1次印刷

印数：0001—5000　　定价：4.05元

ISBN 7-302-00235-5/TP·91(课)

作 者 前 言

Ada主要是为内部计算机系统进行程序设计而设计的语言；在这些系统中，计算机是一种执行联机监测和（或）控制的主要部件。Ada是一种实时语言，因此，除了包括一组完整的通常的语言特征之外，还提供了多任务、同步实时处理以及为低级硬件设备直接进行程序设计的功能。但是，Ada并不只限于实时应用范围，实际上它的"非实时"功能也远远超过了目前通常使用的那些语言所提供的功能。因此，除了在工业控制、通讯和军事系统这些常用的实时领域之外，Ada还可能在系统程序设计、商业软件、数值分析、远程信息处理等各种领域得到广泛的应用。

Ada是建立在Pascal基础之上的，但这并不意味着Pascal是Ada的子集。事实上，几乎不能发现有任何Pascal方式未经改变而出现在Ada中。从Pascal那儿主要继承下来的是它的原则，即算法和数据结构两者都必须精确而又明确地说明，程序逻辑上的统一无论在何处都要尽可能由编译程序来确保。因此，Ada有较好的可读性、可维护性、尤其是安全性。

Ada是一种现代语言，它被设计成可在将来的许多年内能满足软件工程师的需要。实际上，现在所有常用的语言都是很久以前设计的，即使Pascal也已有十年的历史了。当初设计这些语言时，计算机系统仅需要较少软件，提供计算的硬件只限于有限的内存空间，多重处理配置还极少见。然而，今天，超大规模集成电路的技术提供了潜在的无限处理能力和内存容量，其结果是可以承担越来越多的复杂应用了。因此，现在对软件工具有一种不断增长的需求，使用这些软件工具来支持非常大的程序结构。Ada通过为程序的模块化提供大量的功能来满足这种需要。

一个Ada程序是由一组程序包（*package*）表示的。每个程序包是一个封闭体，内含一批数据对象和与之相关的操作。因此，程序包可以用来实现一个程序在其执行中所需的各种资源。这里，资源（*resource*）一词是广义的，它包括这样一些内容：像输入输出操作、缓冲区、堆栈、公共数据区，也包括用户自定义的数据类型，尤其是抽象数据类型。每个程序包有两部分，它们是接口规格和包体。接口规格指定那些在包内的、可被程序其它部分访问的实体。包体包含了对这些实体的完整定义以及一些为实现这个程序包所需的另外一些实体，因此程序包允许对一组逻辑上相关的实体进行可控制的访问，因而是使程序模块化的一种有力工具。接口规格有助于文件编制，并允许编译程序去检查包的所有用法是否合法。此外，Ada允许程序包规格和包体分开进行编译，因此直接支持了自顶向下的程序设计方法、逐步增加系统结构，并支持由个人或以集体的方式进行程序设计。

因此，程序包是Ada中的中心概念，是Ada功能的关键所在。程序包支配着Ada程序的设计方法、书面阅读的方式，并支配着它的组成、测试以及以后进行维护的方式。

本书针对学生和熟练的程序员用Ada语言进行程序设计做了完整的介绍，详细解释了Ada语言的全部特征，并且在凡是可能的地方都有例子说明。本书的主要特点是在每

章的最后包括有一个较全面的实例，这是为了进一步解释该章所包括的要点，但更重要的是通过它们来举例说明如何用Ada语言来设计一个程序。特别是，把重点放在支持数据抽象的程序包使用方面。另外，每章都提供了习题，并在本书末尾给出了部分习题的答案。

Ada是一个庞大而又复杂的语言，要学会如何有效的使用它所遇到的困难是不可低估的。尽管如此，由此能获得的益处可能还是很多的，读者应当一开始就确信，这种努力不只是学会了Ada，而且还将学会一种新的、有效的程序设计方法。

最后，我要感谢Brian Meek，感谢他在这部书的手稿的准备过程中提出的有益的建议，并且感谢我的妻子对我的不断鼓励和支持。

S.J.YOUNG 1982年2月

目　录

第一章 ADA程序的结构

1.1 绪 言

本章打算初步介绍 Ada 程序设计的基本概念，这将通过两个非常简单但又完整的Ada程序来说明，目的是想在详细介绍Ada语言以前先给出一个综合的形象。当然，所提到的很多特点要到学完后面各章之后才能完全弄懂。尽管我们将在以后各章中给出语言的各种结构和例题，但希望本章的这些简介将有助于读者能了解到如何把哪些内容装配成一个完整的程序结构。

在此之前，有必要对软件开发所涉及的内容进行简单的讨论，并介绍一些基本术语。计算机程序实质上是一系列指令，这些指令描述了计算机所实现的功能，并以二进制代码的形式存储在计算机的存储器中。一个程序，不论大小如何，要由程序员直接来产生这些代码基本上是不可能的。因此，必须使用高级程序设计语言，用符号的形式来表示所要求的功能，这种符号面向所要解决的问题而不是面向所使用的机器。

把高级语言翻译成相应的机器代码是由一种叫做编译程序的程序来自动完成的，一旦翻译完成，最后的程序就可由计算机来执行。程序的开发有两个主要阶段，这两个阶段通常称为编译（*Compilation*）阶段和执行（*execution*）阶段。编译期间所出现的事件称为“编译时（*Compile-time*）发生的事件”；执行期间所发生的事件称为“执行时（*execution-time*）发生的事件”，通常或者称为“运行时（*run-time*）发生的事件”。就出错而言，区别编译时和运行时的错误是十分重要的。一个程序中可能会出现各种各样的错误，如拼写错误、语法错误和程序的逻辑上不合理等。Ada被设计成能在编译期间尽可能多地检查出这些错误。这是非常理想的，因为由编译程序检查出来的错误易于查找和修正；而运行时的错误很难查找而且费时。此外，在运行期间，错误的检查还要求计算机去完成特殊的工作。因此，Ada能在编译期间查出许多错误的这个功能使程序员有更强的程序设计能力，使程序有较高的可靠性和运行效率。

现在，再回到实际的Ada程序设计上来。解任何计算问题需要给出不同的两方面信息：第一，必须精确定义所处理的数据；第二，必须指定对这些数据所进行的操作。因此，在Ada中，一个简单的程序也应当由两部分组成，其简要形式如下所示：

```
procedure NAME is
    --specification of the data to
    --be used by the program (程序所用的数据规格)
begin
    --sequence of statements defining
    --the actions to be performed (规定所执行动作的语句序列)
end NAME;
```

此程序的第一行由关键字**procedure**(过程)开头，其后跟着由程序员选定的程序名。

关键字是Ada语言的成分，用来代表各种特定的结构。在上例中，关键字**procedure**代表一段可执行程序的开始。注意：关键字写成小写字母，而程序员定义的名词用大写字母。在过程名的后面是关键字**is**，然后是数据的规格，当然这些数据是指在本 程序用到的。在关键字**begin**和**end**之间，按语句序列的形式写出程序对这些数据所要进行的操作。最后，为提高可读性，再次把程序名写在**end**之后。

Ada中有各种各样的语句，但最常用的是赋值语句。可以利用它把一个表达式代表的值赋给由一个指定名字所代表的变量。例如语句：

```
X:=(Y+1)*2;
```

先计算出变量Y加1乘以2的值，然后把结果赋给X。实际的赋值 由符号:=来代表，:=号右边的一些符号组成一个表达式，即一个计算数值的公式。当然，如果此语句出现在上述的程序中，那么，变量X和Y必须先在数据规格部分中定义。

程序的执行分两个阶段进行：第一阶段，确立（*elaborated*）数据规格 部分中的所有说明，这是一种处理过程，通过此过程，每个专用的说明都完成了它的作用。在一些简单的情况下，此过程可简单地把一个名字和某个数据对象联系起来并可能给它置一个初值。第二阶段中，执行（*executed*）一些语句。这两个阶段中的任一阶段 都可能需要求出（*evaluate*）表达式的值。例如，数据对象的初值可能用一个表达式 来定义；或者如上所示，在一个赋值语句中由表达式来代表所赋的值。

因此，有三个必须记住的术语，这就是：确立（*elaboration*），执行（*execution*），求值（*evaluation*）。然而，关键之处是：为了执行Ada中各种不同程 序单位中的语句，必须首先确立数据的说明，而且这种说明可能要求进行一些辅助的计算。确立并非是一种被动的动作，而是如同执行语句部分那样，它要求计算机去执行某些操作。

给出了这些基本术语以后，现在就可以来介绍一个实际的Ada程序。

1.2 一个简单的ADA程序

先来举一个非常简单的例子，以下所示的叫做PRINT_BRACKETS（打印括号）的程序，它将从与计算机连接的标准输入设备上读入字符，并把从输入信息流中挑选出来的所有左括号和右括号，即"("和")"，都打印出来。假定用点号作为输入结束的标志。

```
with TEXT_IO;
procedure PRINT_BRACKETS is
    CH:CHARACTER;
begin
    TEXT_IO.GET(CH);
    while CH/='.'loop
      if CH='('or CH=')'then
          TEXT_IO.PUT(CH);
      end if;
      TEXT_IO.GET(CH);
    end loop;
```

end PRINT_BRACKETS;

此程序的基本结构遵循前面所给出的简要形式。数据规格部分由一条说明"CH: CHARACTER"组成。这种写法意味着此程序对一个名为CH的变量进行操作，而且它的类型为CHARACTER（字符）。数据对象的类型规定了此对象可以具有哪种类型的值，可以对它采用什么操作。在本例中，变量CH只能保存字符值。

在begin和end之间是实际的语句，这些语句规定了本程序所执行的操作。因此，当开始运行此程序时，首先确立了CH的说明，然后去执行语句。在本例中，确立阶段就是简单地把名字CH与内存中的一个存储单元联系起来，这个存储单元将用来存放CH的当前值。执行阶段就是不断从输入设备读入一个字符，如果这个字符是括号，就把它打印出来；这种操作一直执行到读入一个点号为止。进行读和打印的实际动作是由过程调用语句（procedure call statements）来完成的，例如，每个字符都由TEXT_IO.GET (CH)来读入。此处名字GET就象PRINT_BRACKETS一样，代表了一个子程序，两者的唯一区别就在于PRINT_BRACKETS是在程序开始运行时自动执行的，而GET只是在需要的时候才通过过程调用语句调用执行。GET的作用是在输入设备上读入下一个字符，并把它的值赋给变量CH。定义GET的实际代码被装在叫做TEXT_IO的程序包内。因为是在一个程序包内定义了GET，为了能引用它，程序包名必须作为一个前缀放在子程序名的前面，即写成：TEXT_IO.GET。TEXT_IO程序包定义了一整套输入和输出功能，这在十五章将作详细介绍。然而，这里要强调的一点是：Ada程序可由很多程序包组成，而TEXT_IO只是其中的一个。有些程序包，将由系统预先定义好了，如TEXT_IO就属于这一类；而有些程序包将由程序员定义，以适合于他的特殊需要。

实际上，PRINT_BRACKETS（打印括号）程序的完整结构如图1.1所示。可以看

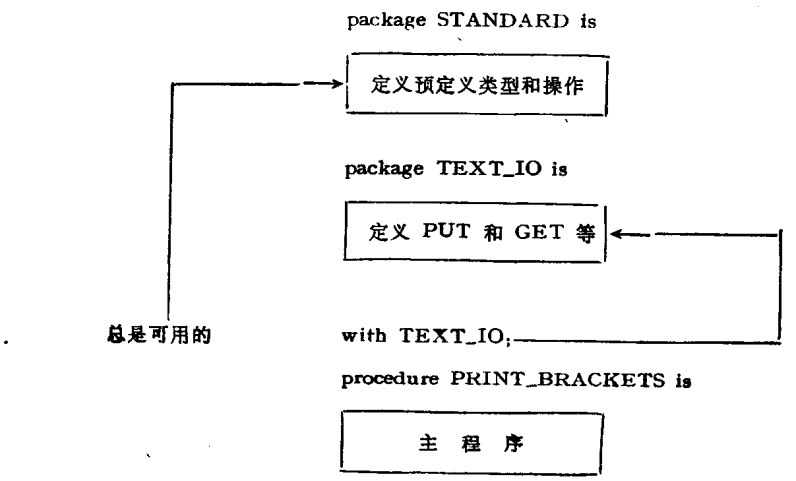

图 1.1 PRINT_BRACKETS的结构

出，除了PRINT_BRACKETS以外还有两个程序包。程序包STANDARD提供了预定义的Ada环境，它包括了对所有Ada预定义类型的定义和操作，并且总是隐含可用的。例如类型CHARACTER就是在这个程序包里定义的。程序包TEXT_IO是一个预定义库单位，它包含了一系列用于输入和输出的操作。STANDARD程序包内的各实体总是可自动存取的，它是Ada中唯一具有这种功能的程序包。而程序需要的所有其它程序包都必

须列在通常称为**with**的子句中，以便使它们成为可用的。因此，为了访问TEXT_IO程序包，必须把子句：

with TEXT_IO

放在过程PRINT_BRACKETS的前头。

通常，一个Ada程序将应用三类程序包：预定义程序包STANDARD，预定义库程序包（如TEXT_IO）和用户定义的程序包。下一节将简单地举例说明如何写一个用户定义的程序包。

1.3 程 序 包

现举例来说明编写和使用程序包的方法。假定对PRINT_BRACKETS例子作一些修改，使其不是打印出输入流中的左、右括号，而是计算左、右括号数之差，以便检查左右括号是否对称。

为此，需要某种形式的计数器。当然，可以直接使用一个简单变量，但在此将提出另一种解决方法，即：采用"用数据抽象来进行设计"的方法。这种方法在设计大型的具有良好结构的程序时是非常重要的，所介绍的Ada语言的许多特点特别适合于用这种方法。这种方法的实质就是：自顶向下，逐层地设计一个程序。在顶层，可通过一个或多个抽象数据类型和一组控制它们的相应操作来确定程序。

用抽象类型（*abstract type*）意味着用这种类型所说明的变量，其实际的内部结构并不知道，而且也与此设计阶段无关。它的任务就是引进一些新的、具有为解决当前问题所需特性的数据类型。在PRINT_BRACKETS中需要的是能存储字符值的能力。预定义类型CHARACTER已是一种可存放字符值的类型，因此，它是不存在问题的。但是，如果CHARACTER还不是一种可适用于存放字符的类型，那么，仍然可以按完全相同的方式来使用它，只是那时它是一个抽象类型而已。在这种情况下PRINT_BRACKETS的设计可仍旧不变，但是程序员在设计了PRINT_BRACKETS之后还有许多工作要做，因为他必须接着为CHARACTER类型设计一个实现（*implementation*）。

通过使用抽象数据类型，从顶层设计中排除了某些枝节问题的细节。在第一层求精中，要对在顶层用到的所有抽象类型的内部结构和相应操作进行定义。当这些类型十分复杂时，可能还需要用另外的（稍微简单一些的）抽象类型对它们进行进一步的定义。因此，由于每一个相继的层次都以相似的方法逐步求精，这一处理过程可能会重复进行，直到所有类型都由语言本身提供的内部类型直接定义为止。

现在，再回到例题上来，在Ada中，可以利用一个程序包来定义一种抽象数据类型。一个程序包由两个独立的部分组成。首先是程序包的规格，它定义了程序包内提供的、可为用户采用的所有功能。其次是程序包体，它定义了这些功能实际上是如何实现的。

以下的程序包规格定义了一个叫做COUNT的抽象类型；同时对COUNT规定了INIT、INCREMENT、DECREMENT操作，这些操作要通过过程调用语句来产生，这和前面例子中使用GET的方法一样；此外还提供了IS_ZERO函数，用来检查计数器的状态。

```
package COUNTER is
    type COUNT is private;
```

```
        procedure INIT (C:in out COUNT) ;
        procedure INCREMENT (C:in out COUNT) ;
        procedure DECREMENT (C:in out COUNT) ;
        function IS_ZERO (C:COUNT) return BOOLEAN;
    end COUNTER;
```

符号private的意思是：类型COUNT的内部结构是本程序包所专用的（即，是程序包的用户所不知道的）。INIT、INCREMENT等都有一个定义成COUNT类型的参数。这允许程序包的用户能自己指定一个他希望对之进行处理的并具有COUNT类型的特定数据变量。例如，调用COUNTER.INIT(X)将给变量X置初值。

在给出了这个程序包规格后，就可以利用抽象类型COUNT来设计一个如下的程序。

```
with TEXT_IO, COUNTER;
procedure COUNT_BRACKETS is,
    use TEXT_IO;        --makes PUT and GET directly accessible
                          (使PUT和GET成为直接可访问的)
    use COUNTER;        --makes COUNT, INIT, etc. accessible
                          (使COUNT, INIT等可访问)
    CH:CHARACTER;
    NBRACKETS:COUNT; --an object of the abstract type COUNT
                          (COUNT抽象类型的对象)
begin
    INIT(NBRACKETS); GET(CH);
    while CH/='.' loop
        case CH is
            when'('=>INCREMENT(NBRACKETS) ;
            when')'=>DECREMENT(NBRACKETS) ;
            when others=>null;
        end case;
        GET(CH);
    end loop;
    if IS_ZERO(NBRACKETS)then
        PUT ("BRACKET MATCH");
    else
        PUT("ERROR:BRACKETS DO NOT MATCH") ;
    end if;
end COUNT_BRACKETS;
```

此程序的操作解释如下：具有COUNT抽象类型的变量NBRACKETS用来统计输入流中遇到的左、右括号之差。在预置了NBRACKETS和字符CH后，重复执行循环，直到CH中包含一个点号为止。在循环中，case语句允许根据CH的值对INCREMENT、DECREMENT或null操作进行选择，语句null表示什么也不做。

注意：放在COUNT_BRACKETS前面的**with**子句有所扩充，变成包括了COUNTER和TEXT_IO。在说明部分还加了一些**use**子句，这些子句使得像PUT和GET这样的程序包实体可以直接加以引用而不必给出程序包名作为前缀。例如只要写成 PUT 而不必写成TEXT_IO.PUT。

利用抽象类型COUNT已形成了顶层设计，下一层求精就是定义COUNT本身。这包括了确定COUNT的类型和它的操作实现这两部分。适合于表示COUNT类型的是整数类型，在Ada中它可以由：

```
type COUNT is new INTEGER;
```

来指定。此类型说明意味着COUNT具有预定义类型INTEGER所具有的全部特性，但它仍是一种与INTEGER性质不同的类型。逻辑上，这种类型说明应该附在COUNTER程序包体内，因为它所包含的信息不必让程序包的用户察觉。但Ada要求把它放在（由于实现的理由）程序包的规格内[1]。所以，以上给出的COUNTER的规格实际上是不完全的，完整的应该是：

```
package COUNTER is
    type COUNT is private;
    procedure INIT (C:in out COUNT) ;
    procedure INCREMENT (C:in out COUNT) ;
    procedure DECREMENT (C:in out COUNT) ;
    function IS_ZERO (C:COUNT) return BOOLEAN;
private
    type COUNT is new INTEGER;
end COUNTER;
```

此处，在符号**private**和**end**之间的说明是本程序包专用的，使用COUNTER程序包的程序，对专用部分给出的信息不进行访问。

这时，COUNTER程序包的实现就是去定义对COUNT类型所提供的各种操作。每种操作都具有和主程序本身一样的形式，只是在它们的数据规格部分没有任何说明。

```
package body COUNTER is
    procedure INIT (C:in out COUNT) is
    begin
        C:=0;
    end INIT;
    procedure INCREMENT (C:in out COUNT) is
    begin
        C:=C+1;
    end INCREMENT;
    procedure DECREMENT (C:in out COUNT) is
```

1) 见*Real Time Languages*, S.J.Young, Ellis Horwood, 1982, Chapter 9, for an explanation of this problem.

```
        begin
            C:=C-1;
        end DECREMENT;
        function IS_ZERO(C:COUNT)return BOOLEAN is
        begin
            return C=0;
        end IS_ZERO;
    end COUNTER;
```

过程COUNT_BRACKETS和程序包COUNTER构成了一个完整的Ada程序，它的完整结构如图1.2所示。

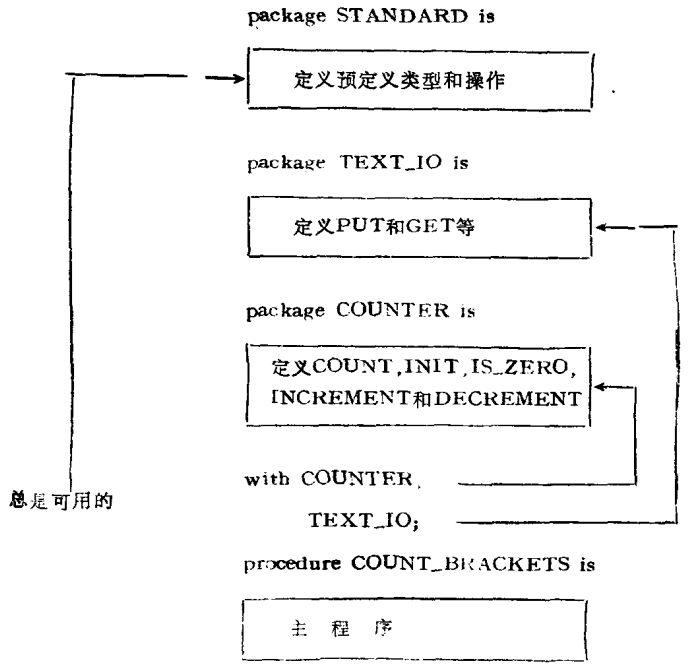

图 1.2 COUNT_BRACKETS的结构

这时读者可能很想弄明白，为什么象进行计数这样的普通操作却要写28行代码。当然，上面的例子确实是非常简单的，而且省略掉COUNTER程序包后也很容易实现。可以把NBRACKETS直接说明成INTEGER并在COUNT_BRACKETS过程内直接进行增减计数。但本例的目的是说明如何建立起一个大的程序。很少有哪种抽象类型能简单得象COUNT那样，而且把这些抽象类型的实现和它们的使用区别开来有着非常现实的优点。首先，它为通过逐步求精进行的程序设计提供了一种有效的策略。第二，程序包可以使抽象类型的使用受到保护；这就使程序包的操作可以在和它的工作环境相脱离的情况下进行检验。第三，可以改变程序包的实现部分而不影响使用它的程序，从而简化了程序的维护工作。例如，为了确保计数绝对不出现负值，可以在程序包COUNTER中加一个检查，而不必对过程COUNT_BRACKETS作任何修改。最后，一个程序可以分成几个独立的程序包，这就很容易把任务分配给一组程序员来分担。所以，希望读者能运