第一部分　COBOL语言

第1章　COBOL简介

- COBOL的目的
- COBOL版本与可移植性
- COBOL的演变
- COBOL标准发展
- 当前COBOL应用

本章要介绍COBOL并浏览本书其他部分的内容。COBOL不仅是一种编程语言，而且是地球上几乎每个国家的公用或专用基础结构中的集成组件。当今世界中很少有人的生活没有受到COBOL程序的影响。这些程序服务于政府、银行、运输系统、制造业及批发系统的日常工作中。COBOL并不引人注目，但它却很强大，它是信息系统的砖和瓦。

COBOL及其重要性

COBOL是Common Business-Oriented Language（公用面向商业的语言）的缩写。COBOL自60年代初开始就广泛应用于计算机应用领域（商业和其他领域）。COBOL是采用英语语法的高级语言，以其可读性、可维护性和可移植性受到商业单位和政府部门的青睐。COBOL不断地演变并吸收计算机技术的进展。与许多当代编程语言不同的是，COBOL针对商业世界的使用，它是真正商用应用程序开发的首选语言。如今，COBOL是企业的解决方案。

如果想了解核心商业的大型信息系统或开发新的商用应用程序，则必须知道COBOL。好在COBOL很容易理解、很直观也很容易编写。此外，COBOL丰富多彩、富于挑战，可以从最复杂的系统中遇到，让人百看不厌。

标准、版本与可移植性

如果只有很少的计算环境中编程经验，则不会了解将代码从一种环境移植到另一种环境时具有的问题，不会知道语言标准、特定厂家版本和移植性之类的问题。但如果要掌握COBOL，就必须对这些问题有所了解。

随着时间的推移，公用应用领域中的技术发展需要标准化才能有效地使用。计算机软件和编程语言更是如此。

COBOL 语言的定义是科技人员几十年工作的结果，他们致力于制定在特定平台上实现COBOL时可用的标准。标准委员会由各有关单位的代表、各个COBOL 编译器厂家的代表和用户社区的成员组成。

当某个组织实现COBOL标准时，用该组织的COBOL编写而成的程序可以和任何其他采用同一标准的组织的COBOL 编写而成的程序一样工作。只要代码元素严格按照标准，代码就可以在不同环境间移植。

尽管COBOL 标准对语法作了严格定义，但早期版本通常并未考虑特定的平台特性和运行环境。后来的版本解决了这个问题，例如，它们考虑了不同的数据格式，如压缩十进制格式和二进制格式。这样，标准随计算机行业软硬件的改变而不断地演变。

在实际中，大多数COBOL 厂家在实现标准之后都进一步提供扩展特性，以吸引开发人员。有时，这些特性被证明为非常有用，并被吸收到下一版标准中。记住，非标准特性的版本是没有统一定义的，移植到另一环境时不一定可行。

本书第一部分几乎全部介绍当前COBOL 标准语言定义，即修订COBOL-85标准。我们希望使其中介绍的COBOL 基础和原理尽量一般化和平台独立。但实际上COBOL 有许多版本，针对不同机器和操作系统。具体演示采用大型机上的IBM COBOL和PC机上的Fujitsu与Micro Focus COBOL。

第三部分将介绍下一版标准准备加入的特性，包括面向对象编程元素、可移植的标准算法和其他开发新应用程序的工具。我们将包括这些特性的新标准称为COBOL 2000，因为标准委员会希望在2000年推出这个新标准。

COBOL简史

要了解COBOL的全部影响，就要知道， COBOL是专业人员为普通程序员设计的。这种语言不是针对某类机器的，各类机器上的用户都能随心所欲地工作。专家小组在过去的40年里建立、维护并扩展了这个充满活力的语言。而行业领头羊的投资、支持和指导也是COBOL取得成功的重要原因。

早期COBOL

50年代有一种潮流，就是建立专门解决商业问题的编程语言。这种潮流与许多编程人员和计算机权威的观点相反，该潮流的推崇者认为机器语言不是计算机唯一理解的语言，而是适合各种任务的最佳语言。这种潮流的成功改变了编程人员和计算机权威的观点。

成功的商业编程语言范例是1954～1958年之间Sperry-Rand的开发并修订的FLOW-MATIC。许多大公司和美国空军都采用了 FLOW-MATIC。FLOW-MATIC易于编写程序和调试，因为它采用了英语式语法（如 ADD和MOVE），数据名可以较长和带含义（如STATE-TAXES 和TOTAL-PAY）。

1959年，高级语言的潜力得到充分证明，计算机用户和学术机构与制造商联合定义了商业化商业语言的目标。当年四月在宾西法尼亚大学召开了一次会议，提出要有一种所有计

算机上均能使用的通用商业语言。这个小组向美国国防部寻求资助，得到了国防部的同意。

说明：有趣的是，人们经常以为美国国防部支持COBOL作为军队标准化计算的途径。虽然这不完全属实，但五角大楼的确很快发现了这个工作的意义，并投入了大量的精力。

1959年5年，五角大楼召集了一次会议，暂时把他们的项目称为公用商用语言（CBL）。小组确定这种语言的主要要求如下：简单英语动词，易用性优先于功能，数据与过程相分离。这样，COBOL的语法中就借用了段、句、词的概念。也许COBOL程序并不像创始人预料的那样可读（显然比现代的编程语言更冗长），但这种把“神秘和高深”赶出程序的做法值得称赞。

COBOL创始人致力于扩大潜在的编程队伍，使新程序的开发要求商业知识而不要求对特定计算机的过细了解。为此，小组认为这种语言应能在多种计算机系统间移植。

这样就形成了三个委员会：分别针对该语言的短期、中期和长期开发。实际上，短期委员会理由充足地提出了这种语言，根本毋庸置疑。委员会成员指派一个管理小组监视这种语言的开发，这个小组就是著名的CODASYL。

到1959年夏天，这个委员会主要根据FLOW-MATIC词法及其三个部分（过程、数据描述和环境）开发了一种语言。这些部分现已成为COBOL的四个部。这个委员会还借用了IBM的COMMERCIAL TRANSLATOR，特别是其PICTURE从句和组项目（组成01、02等表示的层）。这个委员会将该语言命名为COBOL，表示公用面向商业语言。

1959年12月，第一个COBOL规范的最后草案完成。尽管从此以后COBOL在标准化过程中不断修订，但当时的许多技术决策一直影响至今。

作为委员会建立和维护的语言，COBOL的演变主要通过争论达成，从而形成某种折衷。例如，构造数字公式时是否使用符号或词的争论就产生了用COMPUTE作为表示等式的更加符号化方式，并认为ADD、SUBTRACT、MULTIPLY和DIVIDE更易读。

将“折衷语言”不可移植的专用语言相比，付出的代价是值得的——COBOL得以迅速普及。1960年，美国国防部宣布，所有购买的计算机都要包括COBOL编译器，私营公司迅速跟随其后。

尽管COBOL最初只要求这个标准能维持几年，但COBOL一直是大型机上最普及的语言，其用途还在其他平台上不断扩展。COBOL如此普及的原因之一是，国防部要求用COBOL，避免多种语言的争夺。国防部的参与使他们可以要求COBOL包括商业数据处理最有用的元素。COBOL普及的另一个原因是它要求在所有计算机上产生一致性结果。这个特性通过了政府接收测试的验证，使公司和政府有信心在COBOL中大量投资。

COBOL的硬件无关的特性在该语言规范的演变中起着巨大的推动作用。这个规范强调标准输入／输出方法，因为这些操作是商业计算机的骨干。由于输入／输出设备的特性和计算机技术的日新月异，COBOL需要相应扩展和改变。COBOL自1960年首次出现以来，进行了许多次修改。大多数修改都是通过严格标准化过程的论坛实现的。

COBOL标准的发展

1960年，美国计算机与商业设备制造商协会（CBEMA）成立了一个委员会，叫做美国计算机与信息处理国家标准委员会，简称X3。X3委员会的分会X3.4编程语言分会建立了X3.4.4工作组“处理器规范与COBOL标准小组”。

X3.4.4（后更名为X3J4）小组负责建立COBOL标准。这个小组的第一次会议于1963年召开，由计算机制造商和用户的代表参加。他们确定了小组的目标是根据COBOL的CODASYL标准（1959年该委员会产生的最初COBOL标准）定义COBOL国家标准。

1968年，美国标准协会（USASI）批准了COBOL分会开发的标准，发表号为X3.23-1968。这个文档定义了COBOL包括内核（Nucleus）和下列八个功能模块：

- Table Handling（表格处理）
- Sequential Access （顺序访问）
- Random Access （随机访问）
- Random Processing （随机处理）
- Sort （排序）
- Report Writer （报表写入）
- Segmentation （分段）
- Library （库）

每个模块分成最多三层，高层提供更多功能，低层提供高层的子集。 COBOL的基本版本应包括内核、表格处理和顺序访问模块的低层功能。完全版本的COBOL应包括所有模块的高层功能。美国标准协会（USASI）于1966年由ASI更名为USASI，1969 年更名为美国国家标准协会（ANSI）。这个ANSI标准即著名的COBOL-68。

1974年对标准进行了修订， 8个功能处理模块扩充为11个：

- Table Handling （表格处理）
- Sequential I/O（顺序I/O）
- Relative I/O （相对I/O）
- Indexed I/O（索引I/O）
- Sort-Merge （排序／合并）
- Report Writer （报表写入）
- Segmentation（分段）
- Library （库）
- Debug（调试）
- Inter-Program Communication （程序间通信）
- Communication （通信）

每个模块包括两层或三层。在9个模块中，最低层为空集。每个低层都是高层的子集。对于COBOL-68，基本版本应包括内核、表格处理和顺序I/O模块的最低层。

ANSI COBOL的最新版本于1985年发布，本书采用了这个版本（我们将注明COBOL-85对COBOL-74作出补充和改变的地方）。 1985模块与COBOL-74中相同，只是没有单独的表格处理模块，这个功能加进了内核模块。下一个标准COBOL 2000将消除模块和层的概念。

CODASYL COBOL 委员会（CCC）长时间与X3J4并列存在。CCC负责所有语言发展，从而产生各种版本的“发展月刊”（JOD）。X3J4根据JOD定义标准。这种运行方式一直持续到COBOL-85 标准完成。之后两个委员会最终合二为一， X3J4最近更名为J4，负责处理COBOL语言的发展标准化。

尽管COBOL是由美国的小组用英语作为语法基础进行定义和标准化的，但商业数据处理的国际社区对这种语言及其在其他环境中的使用兴趣浓厚。 ANSI COBOL 委员会与国际组织密切合作，建立的COBOL-68 标准符合COBOL的ISO （国际标准化组织）推荐标准。同样，后来的两个ANSI COBOL标准:COBOL-74和COBOL-85也被ISO批准和采用。国际工作小组从ISO角度协调并控制这个发展过程。

COBOL与结构化编程

50年代，编程人员需要很高的编程技巧。 GO TO语句是汇编跳转指令的改进，在当时使用非常普及。但编程人员要维护、修改和改进这种程序相当困难。

说明：GO TO语句通常跳到程序代码中另一部分。这种跳转很难跟踪，让人看不出代码中如何转移到某个分支或转移后要执行哪块代码。

尽管编程人员可以使用流程图和编写GO TO语句，但这个方法很快便显示出弱点。人们发现，有些程序的平均寿命通常在七年以上。事实上，有些系统运行更长时间，可以工作30多年。编程人员必须设法生成更可靠、更易维护的系统。这种可维护性变得比软件创作的具体技巧更重要。

由于对软件期望的改变， 60年代，人们提出了聚合软件结构理论，从而导致了模块化（或结构化）编程的运动。一开始，人们强调得更多的是模块内代码的形式。后来，开发人员认识到，系统的总体模块布局更加重要。

所有程序并非生而平等。分析问题的方法有好有坏。许多人开始寻找好的模块结构应有的特征。而功能分解（即把程序分解成最小功能）产生了整洁利索的模块。

结构化编程揭示了程序设计有好有坏，从而引出了评估模块设计的两大问题：

- 模块是否聚合，模块内部是否很好地相关联？
- 模块是否耦合，参数与模块功能是否相关？

COBOL ANSI标准的改变反映了结构化编程的潮流。 GO TO 语句最终在ANSI COBOL-85标准中从常用COBOL编程做法中消失。这个标准引入了范围限制符、 Case结构（动词EVALUATE ）和PERFORM 语句。1989 年补充指定了内部函数。

当代软件开发不仅要求可靠、代码可复用，而且使可复用代码成为新系统和重新设计传统系统的关键。面向对象编程的思想将在第三部分介绍，它是开发可复用代码和图形用户接口（GUI）的方法。新的COBOL标准预计在2000年推出，新版标准将提供面向对象编程结构和其他现代编程结构。

COBOL工作环境

COBOL语言经历了 40年的风雨沉浮，已经存在近千亿条COBOL 语句。所以一定要了解COBOL操作的环境。

从发展看，COBOL应考虑其创建的编程环境。出现个人计算机之前，各大硬件制造商（例如DEC、EDG、Unisys和IBM等公司）都有某种COBOL系统。

多年来，IBM一直是计算机行业先驱。IBM成功地在全世界100多个国家销售其大型计算

机。只要有IBM机器的地方，就有COBOL。IBM COBOL应用程序在政府和公司系统中随处可见。在IBM环境中，系统围绕COBOL成长、与COBOL混用、直到与COBOL密不可分。

如今，到IBM商店购买COBOL应用程序时，你会发现，它与JCL、CICS、IMS、SQL等混在一起。由于许多环境中必须了解这些组件才能使用COBOL，本书第二部分将介绍这些组件。

在计算机科学与软件工程世界中，COBOL备受指责。有些人要用户单纯地使用Unix和Dos命令。有些没有用过COBOL的人也对COBOL持很坏的印象。但COBOL是个无所不在的商业开发语言，其现代版本能解决当前商业问题。事实上，一流COBOL开发人员正在编写使用COBOL的Web应用程序。本书第三部分将介绍如何开发新的COBOL应用程序。

第2章　COBOL程序概述

- 一个简单的COBOL 程序
- COBOL的语法说明
- COBOL 单字与直接数
- 源程序格式
- COPY与REPLACE语句

本章首先概述COBOL程序的结构和元素，还介绍COBOL 语言语法和源程序格式，最后再介绍如何用COPY 和REPLACE 语句在COBOL 程序中插入文本。

要了解这些概念，我们首先介绍　一个简单的COBOL程序，体会一下这种语言的总体结构

一个简单的COBOL程序

COBOL程序与其他语言写成的程序不同。一旦熟悉了COBOL程序的结构，你就会发现，COBOL 程序比其他类型的程序更易读。当然，具体的程序可读性取决于编程人员对程序中使用的各个元素名称的选择。但COBOL的基本结构和性质有利于人们编写简单易读的程序。

要介绍COBOL程序的结构，可以从一个简单例子开始。清单2.1是计算贷款每月偿还额的COBOL程序。图2.1是运行清单2.1计算汽车贷款的结果。

清单2.1　MORTGAGE程序

```
000010 IDENTIFICATION DIVISION.
000020  PROGRAM-ID.  MORTGAGE.
000030  AUTHOR.  Mastering-COBOL.
000040*
000050 ENVIRONMENT DIVISION.
000060*
000070 DATA DIVISION.
000080 WORKING-STORAGE SECTION.
000090 77 AMOUNT         PICTURE 9(8)V99.
000100 77 INTEREST       PICTURE 999V999.
000110 77 MONTHS         PICTURE 9(5).
000120 77 M-INTEREST     PICTURE 999V9999.
000130 77 PAYMENT        PICTURE ZZZ,ZZ9.99.
000140*
000150 PROCEDURE DIVISION.
000160*
```

```
000170 GET-PARAMETERS.
000180     DISPLAY "Amount borrowed: "
000190     ACCEPT AMOUNT.
000200     DISPLAY "Annual interest rate: "
000210     ACCEPT INTEREST.
000220     DISPLAY "Number of months: "
000230     ACCEPT MONTHS.
000240*
000250  COMPUTE-PAYMENT.
000260     COMPUTE M-INTEREST = INTEREST / 12.
000270     COMPUTE PAYMENT ROUNDED = AMOUNT * (M-INTEREST /
000280           (1.0 - (1.0 / (1.0 + M-INTEREST)) ** MONTHS)).
000290*
000300  DISPLAY-RESULTS.
000310     DISPLAY "Monthly payment: ", PAYMENT
000320     STOP RUN.
```

图2.1 计算汽车贷款的结果

源程序每行的前6列（6位整数）是序号，这不是程序本身的内容， COBOL编译器会忽略它们。序号不是必须的，但其所占区域会保留。序号和代码格式中的其他元素将在本章稍后详细介绍。

这个程序全部用大写字母，这是过去的COBOL 标准所要求的，但COBOL-85允许采用小写字母。许多旧程序都是全部用大写字母，但利用小写和大小写混合可以提高程序的可读性。

COBOL程序的部

COBOL程序部分成四个部（DIVISION 如清单2.1的第10行、第50行、第70行和第150行所示：

```
000010 IDENTIFICATION DIVISION.
000050 ENVIRONMENT DIVISION.
000070 DATA DIVISION.
000150 PROCEDURE DIVISION.
```

这些部应按上述顺序出现（但不一定在指定的这些行上）

部标题表示部的开始，在部标题之间，可以插入说明行， 即第7列以*号开头的行，例如：

```
000040*
```

说明行可以放上任何说明。这里我们用说明行使部容易定位，也可以用空行。

部又进一步划分为节和段。这些段内有各种COBOL 元素。标识部、环境部和数据部内的元素包括字、分隔符和字符串，组成 COBOL所谓的节、段和项目。过程部包括程序逻辑。在过程部中，段中的语句包含COBOL动词、字、分隔符和字符串。

下面几节简要介绍清单2.1中COBOL程序的部，第3章将详细介绍部。

标识部

标识部的功能是提供程序的一般性文档说明，如程序名和程序员名。清单2.1中的标识部包含两段：

```
000020  PROGRAM-ID.  MORTGAGE.
000030  AUTHOR.  Mastering-COBOL.
```

PROGRAM-ID段包含程序名，是每个程序必需的。 AUTHOR段包含程序人员所要的说明，但它通常包含程序员名。 AUTHOR 段是可选的，事实上，在COBOL-85 中这个段已经过时。过时元素 （obsolete element）就是COBOL标准委员会准备在下一版标准中删除的元素，但传统代码中仍然有可能遇到这些元素。

环境部

环境部提供与程序外部有关的项目，具体地说，它列出程序所用的文件。我们的程序不需要外部文件，因此环境部是空的，可以省略部标题。

数据部

数据部提供程序变量的存放位置，对不同数据类型分成不同节：文件数据（File Section）、静态数据（Working-Storage Section）、参数（Linkage Section ）和其他。我们的程序只需要静态变量，因此只有Working-Storage Section。程序所需要变量的定义在90行～130行：

```
000080 WORKING-STORAGE SECTION.
000090 77 AMOUNT         PICTURE 9(8)V99.
000100 77 INTEREST       PICTURE 999V999.
000110 77 MONTHS         PICTURE 9(5).
000120 77 M-INTEREST     PICTURE 999V9999.
000130 77 PAYMENT        PICTURE ZZZ,ZZ9.99.
```

每个变量定义以层号开头，层只是个整数，将数据放在与其他数据的层次关系中。这个程序中每个变量都是独立的，因此用特殊层号77显示。

数据项目和定义包括名称和数据类型。在COBOL中，数据类型称为类别（category）。我们用PICTURE 从句描述类别。格式字符串是PICTURE字后面的字符串，描述数据项的长度和特征。这里除PAYMENT项目以外的数据项都是numeric 类型，其格式字符串由字符9（表示十进制数字）和字符V （表示小数点位置）组成。

项目 PAYMENT的类型为numeric-edited （可编辑数值），即这个数据项目将数字格式

化成用户易于理解的形式。 Z字符抑制前头的O。这个项目包括显式小数点，并用逗号作为千位分隔符。

数据部的数据项定义将在第4章详细介绍。

过程部

最后一个部是过程部，包含构成程序的过程性语句。我们将其分成第170行、第250行和第300行的段（paragraph）：

```
000170 GET-PARAMETERS.
000250  COMPUTE-PAYMENT.
000300  DISPLAY-RESULTS.
```

这些段与其他语言中的标号相同。这个程序实际上并不需要任何段，但它们可以作为说明提示信息，提示读者程序在干什么。每个段包含几条语句。

在COBOL 中，所有语句都以一个动词（verb）开头，表示语句的类型。下面几章介绍COBOL时会经常提到语句、它们的组成短语（多种形式）和对数据或程序流的影响。

清单2.1中，可以看出DISPLAY语句将字符串写到终端屏幕。例如，下列语句显示文本Amount borrowed:

```
000180   DISPLAY "Amount borrowed:
```

DISPLAY 语句可以写入字母数字直接数（引号中的内容），也可以写入程序数据，例如：

```
000310   DISPLAY "Monthly payment   :", PAYMENT
```

下列ACCEPT语句从终端向程序输入数据：

```
000190   ACCEPT AMOUNT
```

COMPUTE 语句向变量赋值的一个数字表达式。在清单2.1第二条COMPUTE 语句（第270行和第280行）中，我们用ROUNDED 短语取得便士值的计算结果：

```
000270   COMPUTE PAYMENT ROUNDED = AMOUNT * (M-INTEREST
000280          (1.0 - (1.0 / (1.    + M-INTEREST)) ** MONTHS))
```

语句STOP RUN终止程序，但可以省略，程序到达过程部末尾时即会停止。

清单2.1中的语句和过程部可以使用的许多其他COBOL语句将在第4章介绍。

COBOL程序要求

清单2.1中的程序还有几点值得一提。一个是英文句号（.）的使用。这个字符在COBOL中很重要。必须用这个字符结束部标题、节标题、段名、项目和语句。如果省略英文句号，则程序无法编译。用句号作为小数点时（如COMPUTE 语句中的直接数数字值1.0），两边都要有数字，否则编译器会把它看成句子结束。

COBOL程序格式的另一个重要表现形式是缩排。 COBOL对行的布局和哪个列包含元素有严格规定。例如，过程部的语句必须从12列以上开始。第3章和第4章将介绍各个元素的具体规则。

说明：许多厂家开始在COBOL 程序格式上提供更大的灵活性。但采用标准格式的扩展时，必须事先考虑程序在不同编译器之间的移植。

COBOL语法

描述COBOL语言的记号很简单，只用几个规则。要了解COBOL语句和项目如何工作，需要了解这些规则：

大写字：参考格式中的大小字是COBOL单字，在COBOL 中有特定含义，只能在指定场合使用。保留字加下划线时，是格式中必需的，不加下划线时，则是可选的，对该元素的意义没有影响，可有可无。加上可选保留字通常能使语句更易读。

小写字：小写字母泛指其他地方定义的COBOL元素，所选名称用于表示其含义。本书用斜体字表示这类单词。在说明COBOL项目时，小写字母后面通常加上一个整数（例如identifier-1） 。整数表示该元素发生的次序。

中括号：中括号（[ ]）中的部分是元素中的可选部分，如果中括号内有竖条，则可以选择其中一项内容。

大括号：大括号（{}）中的部分是必需的元素。如果大括号内有竖条，则可以选择其中一项内容。

选择标识符：选择标识符（||）包括一组元素，其中要有一个或几个元素，但任何元素都不能重复。

省略号：省略号 （……）表示格式中前面的元素可以重复。前面的元素可以是一个元素（小写字母单词引用的内容）或放在大括号、中括号中的整组元素。

特殊字符：格式中出现的特殊字符（+ - > <= > = < =）是必需的，即使不加下划线，也具有COBOL单字的状态。

标点符号：标点符号（逗号和分号）放在空格后面，可以用在格式中使用空格处，对格式没有影响。标点符号（句号）放在空格后面具有必需字的状态。

例如，下列格式是SUBTRACT语句说明的一部分：

$$\underline{\text{SUBTRACT}}\ \left\{\begin{matrix}identifier\text{-}1\\literal\text{-}1\end{matrix}\right\}\ \ldots\ \underline{\text{FROM}}\ \left\{\begin{matrix}identifier\text{-}2\\literal\text{-}2\end{matrix}\right\}$$

$$\underline{\text{GIVING}}\ \{identifier\text{-}3\ [\underline{\text{ROUNDED}}]\}\ \ldots$$

SUBTRACT、FROM、GIVING和ROUNDED是保留字，ROUNDED 是可选的，因为放在中括号中，但由于还加了下划线，因此加上这个字会改变语句的意义。

元素identifier-1、identifier-2和identifier-3都是元素identifier的实例，应事先定义。元素literal-1和literal-2是literal的实例，也应事先定义。 SUBTRACT字后面要选择标识符（identifier-1）或直接数（literal-1），因为括号内有个垂直竖线。保留字GIVING在这个SUBTRACT语句的格式中是必需的，其后面至少要有一个标识符（identifier-3），还可选ROUNDED字。这个identifier [ROUNDED] 序列可以重复。

下面是个合法SUBTRACT 语句（假设identifier和literal 已经定义）：

```
SUBTRACT 1.00 FROM AMOUNT GIVING NEW-AMOUNT ROUNDED.
```

```
SUBTRACT TAX, MEDICAL, DENTAL FROM GROSS-PAY
    GIVING NET-PAY, TEMP-PAY
```

注意，第二个例子中用几个逗号增加可读性，这并不改变程序的作用，但语句更明晰。事实上，利用语句中的混合大小写还可以进一步增加可读性。例如，变量用小写：

```
SUBTRACT tax, medical, dental FROM gross-pay
    GIVING net-pay, temp-pay
```

说明：尽管COBOL中大小写字母均合法，但通常的习惯是全部用大小写字母。大多数传统代码都是全部用大写字母，因为COBOL-85 以前的标准要求这样，COBOL 编程人员已经习以为常。新程序建议改用混合大小写，这样可以增加可读性。本书第三部分的例子程序采用混合大小写。

COBOL元素

COBOL 元素用 COBOL字符集生成，见表2.1。下面几节介绍字符集在COBOL 单字和直接数中的用法。

表2.1 COBOL字符集

字符	说明
0...9	Digits（数字）
a...z	Lowercase letters（小写字母）
A...Z	Uppercase letters（大写字母）
	Space（空格）
+	Plus sign（加号）
-	Minus sign or hyphen（减号或连字符）
*	Asterisk（星号）
/	Slant or solidus（斜杠）
$	Currency sign（美元号）
,	Comma or decimal point（逗号或小数点）
;	Semicolon（分号）
.	Period or decimal point（句号或小数点）
"	Quotation mark（引号）
(	Left parenthesis（左括号）
)	Right parenthesis（右括号）
>	Greater than（大于号）
<	Less than（小于号）
:	Colon（冒号）

说明：在非数字直接数和说明中，字符集扩展成包括计算机字符集中的任何字符。

COBOL单字

COBOL单字分为三种：用户定义字和系统名称和保留字。 COBOL单字的规则如下：

- COBOL的构成字符为数字、字母和连字符。
- 第一个和最后一个字符不能用连字符。

- 一个字长不能超过30个字符。
- 小写字母与大写字母等价。
- 保留字不能用作用户定义字和系统名称，但用户定义字和系统名称可以用同一个单词。

警告：对C语言编程人员：COBOL构词时采用连字符，而不是下划线字符，但下一版COBOL可能扩展成包括下划线字符。采用连接字符在数学表达式中可能产生问题，因为它们同时表示负号。记住在负号周围加上空格。

用户定义字

用户定义字是文件、数据、标号、程序等所取的名称。一般来说，用户定义字应在程序中唯一，但也有例外。例如，可以存在多个同名数据项，只要它们的完全限定名唯一。第3章和第4章将详细介绍用户定义字。

系统名称

系统名称是COBOL保留字，用于与操作环境通信，由COBOL厂家定义，可分为三类：

- Computer name （计算机名）
- Implementers name （实现者名）
- Language name（语言名）

保留字

保留字是在COBOL程序中具有特定意义的COBOL单字或特殊字符（用作数学或关系操作符），只能在指定环境中使用。本章前面的SUBTRACT例子中已经介绍一些保留字。COBOL有300多个保留字，附录A列出了完整的清单。要记住哪个单词是保留字很不容易，但现代COBOL编辑器会高亮显示保留字。

COBOL是模块化的，包含一些（厂家可以不实现的）语言特性，如调试、通信和报告编写模块。但所有COBOL版本要认识全部保留字，不能用其作为用户定义字。

直接数

COBOL和其他编程语言中一样，直接数是表示自己的字符串。COBOL有两种直接数：非数字直接数和数字直接数，以及符号形式的象征常量。

非数字直接数

非数字直接数是放进引号（"）内的字符串，字符串中的字符可以是计算机字符集中的任何字符。字符串内的引号用两个连续引号表示。连接数的值是字符串，类别为字母数字字符（alphanumeric）。

下面是非数字直接数的例子：

```
"Expired"
"Bob's House"
"Quoth the Raven, ""Nevermore"""
```

数字直接数

数字直接数是由数字字符组成的字符串，可以包含代数符号（+和-）和小数点。省略符号等于包括正号。小数点不能作为直接数的最后一个字符（以免与逗号分隔和语句结束符相混）。小数点通常用句号（.），但如果程序环境部的SPECIAL-NAMES段中包括DECIMAL-POINT IS COMMA语句，则可以用逗号（见第3章）。数字直接数的值为它的代数值，类别为numeric。

下面是数字直接数的例子：

```
0
27
+51
-10.77
```

象征常量

象征常量用保留字表示某些常用直接数。除ZERO（用于数字数据）外，这些直接数都是字母数字字符。表2.2列出了象征常量及其表示的直接数。

表2.2 象征常量

象征常量	表示的直接数
[ALL] ZERO	根据具体情形，可以表示数字直接数0或内字符0构成的非数字直接数
[ALL] ZEROS	
[ALL] ZEROES	
[ALL] SPACE	由空格字符组成的非数字直接数
[ALL] SPACES	
[ALL] HIGH-VALUE	程序校正顺序中最高顺序位的字符组成的非数字直接数
[ALL] HIGH-VALUES	
[ALL] LOW-VALUE	程序校正顺序中最低顺序位的字符组成的非数字直接数
[ALL] LOW-VALUES	
[ALL] QUOTE	由引号组成的非数字直接数
[ALL] QUOTES	
ALL literal	Literal连续接合而成的字符串（Literal应为非数字直接而不能是用象征常量）
ALL symbolic-character	SPECIAL-NAMES段中指定的符号字符值连续接合而成的字符串

表示字符串的象征常量的长度及其数值取决于具体上下文。如果象征常量与数据项相关联，则其长度等于该项目的长度。如果不与另一数据项相关联，则其长度为一个字符，但采用ALL literal格式时其长度为直接数的长度（与省略关键字ALL时相同）。例如，如果数据项NAME的长度为20个字符，则对于下列语句：

```
MOVE SPACES TO NAME
```

象征常量SPACES等于20个空格组成的非数字直接数。对于下列语句：

```
MOVE ALL "XYZ" TO NAME
```

象征常量ALL "XYZ"等于非数字直接数"XYZXYZXYZXYZXYZXYZXY"

源程序格式

COBOL源程序格式比较原始，与穿孔卡输入时代有关，具有下列要求：

- 每行开头有六个字符的序号区。
- 序号区后面是一个字符的指示符区。
- 指示符区后面是四个字符的A区。
- A区后面是语句体，称为B区。B区的宽度由每个COBOL厂家确定，通常到第72个或第80个字符。
- 原先的COBOL格式不允许变长文本行。
- 空行的每个区都是空格（除序号区），忽略不计。

COBOL对序号区的内容没有要求。序号区用于放置序号，以便在卡片搞乱时能方便地（通过卡片排序器）放回正确顺序。

指示符区

指示符区有几个作用。在正常的COBOL源代码行中，指示符区为空白。指示符区的星号（*）或斜杠（/）表示其为说明行，行中的其他内容均被忽略。斜杠与星号的差别在于，程序编译时它强制程序清单另起一页。

指示符区的连字符（-）表示该行为上一行的续行， B区的第一个非空字符是上一行最后一个非空字符后面的字符。例外是占两行的非数字直接数续行中，第一个非空字符是应为引号（“），直接数从下一个字符开始。续行的A区应为空白。

说明：任何COBOL项目和语句都可以跨行，在任何允许插入空格分隔符的地方插入分行符，并在下一行的B区续行。对于很长的非数字直接数，要从单词中间分行时，只要在分行时使用-指示符。

指示符区的字母D表示该行为调试行，这个部分只在打开测试时才属于该程序。 COBOL调试特性在将来的标准中不支持，因为COBOL厂家开发了更复杂的程序调试方法，称为源代码级调试。

指示符区只允许下述五个字符：空格、 *、／、-和D。

A区和B区

COBOL程序的下列元素应放在A区：

- 部、节、段标题
- 段名
- 层指示符和层号01与77
- 关键字DECLARATIVES和END DECLARATIVES
- 程序结束标题

A区元素的目的在于通过最少的格式提高程序的可读性。过程部的语句必须从B区开始。这些元素将在后面几章介绍COBOL程序部和语句时详细介绍。

程序文本的复制与替换操作

利用 COPY语句可以将文本从源库复制到COBOL源程序中。利用 REPLACE语句可以替换程序中的现有文本。下面几节介绍这些语句的格式。

COPY语句

COPY 语句用COBOL 源文本text-name 替换从COPY 字开始到终止句号之前的内容。

```
COPY text-name [ { OF } library-name ]
               { IN }

[ REPLACING { ==pseudo-text-1== } BY { ==pseudo-text-2== } ... ].
            { identifier-1       }    { identifier-2       }
            { literal-1          }    { literal-2          }
            { word-1             }    { word-2             }
```

COPY语句无法嵌套（在标准COBOL 中）。但有些厂家允许不带REPLACE 从句的COPY语句嵌套，而COBOL 2000 之中两者均允许。COPY语句可以放在源程序中除了非数字直接数和说明行或项目以外的任何地方。

如果使用 REPLACING选项，则复制文本中匹配的pseudo-text-1、identirier-1、literal-1或word-1换成BY 后面的元素。

Pseudo-text是一组用空格、逗号或分号分开的字。特殊分隔符= =用于限定伪文本（Pseudo-text）。

其他REPLACING选项（identifier、literal和word）是只有一个字的伪文本。由于COBOL将所有分隔符一样处理，因此文本匹配与替换不是基于字符，而是基于单词。

提示：许多程序访问同一文件时， COPY语句特别适用于复制程序数据部中的文本描述和记录布局。

REPLACE语句

REPLACE语句将COBOL源程序某个段中的文本进行替换。 REPLACE语句不能嵌套，必须一个REPLACE 语句终止之后才能开始另一个REPLACE 语句。替换的文本不能生成新的REPLACE语句：

```
REPLACE {==pseudo-text-1== BY ==pseudo-text-2==} ...

REPLACE OFF
```

替换从REPLACE语句的第一个格式开始，到REPLACE OFF 语句结束。

COBOL 程序结构小结

本章曾介绍过，COBOL程序是COBOL单字的正确顺序。COBOL的元素组成四个部分，必须按顺序出现：

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
  [other-identification-division-entries]
[ENVIRONMENT DIVISION.
  [environment-division-entries]]
[DATA DIVISION.
  [data-division-entries]]
[PROCEDURE DIVISION.
  [procedure-division-entries]]
[end-program-header]
```

每个部从部标题开始。首先，标识部标识程序名并提供标准的程序建档位置。然后，环境部中包含与程序运行的特定计算机或特定操作系统有关的元素，放置与机器相关的信息。第三，数据部包含程序需要的所有数据定义。最后，过程部是程序的过程性语句。

COBOL程序的最后一个元素是可选的程序结束标题，表示源程序的结尾。

说明：COBOL程序中只有标识部标题是必须的，结束程序标题是可选的，只在COBOL程序中包括另一COBOL程序时才需要（第6章将介绍子程序）。

部进一步划分为节（Section），节又进一步划分为段（Paragraphs）。节从节标题或节名开始，段由段标题或段名开始。标题指COBOL单字，名称指用户定义字。前三个部只能用标题，即COBOL标准要求的节和段名。

部标题、节标题或节名、段标题或段名和结束程序标题要从A区开始，过程部的语句要从B区开始。