CHAPTER

第5章 详细设计

详细设计又称为过程设计，在概要设计阶段，已经确定了软件系统的总体结构，给出系统中各个组成模块的功能和模块间的联系。接下来的工作，就是要在上述结果的基础上考虑“怎样实现”这个软件系统，直到对系统中的每个模块给出足够详细的过程性描述。需要指出的是，这些描述应该用详细设计的表达工具来表示，它们还不是程序，一般不能够在计算机上运行。

详细设计是编码的先导，这个阶段所产生的设计文档的质量将直接影响下一阶段程序的质量。为了提高文档的质量和可读性，本章除了说明详细设计的目的、任务与表达工具外，还将扼要介绍结构程序设计的基本原理以及如何用这些原理来指导模块内部的逻辑设计，提高模块控制结构的清晰度。

学习目标：

- 了解详细设计的内容和原则。
- 掌握数据代码设计的工具。
- 掌握人-机界面设计和实现原则。
- 了解程序结构复杂性的定量度量方法。

5.1 详细设计的内容与原则

5.1.1 详细设计的内容

详细设计的目的是为软件结构图(SC 图或 HC 图)中的每一个模块确定使用的算法和块内数据结构，并用某种选定的表达工具给出清晰的描述。表达工具可以由开发单位或设计人员自由选择，但它必须具有描述过程细节的能力，进而可在编码阶段能够直接将它翻译为用程序设计语言书写的源程序。

这一阶段的主要任务如下。

(1) 为每个模块确定采用的算法，选择某种适当的工具表达算法的过程，写出模块的详细过程性描述。

(2) 确定每一模块使用的数据结构。

(3) 确定模块接口的细节,包括对系统外部的接口和用户界面,对系统内部其他模块的接口以及模块输入数据、输出数据及局部数据的全部细节。

在详细设计结束时,应该把上述结果写入详细设计说明书,并且通过复审形成正式文档,交付给下一阶段(编码阶段)作为工作依据。

(4) 要为每一个模块设计出一组测试用例,以便在编码阶段对模块代码(即程序)进行预定的测试,模块的测试用例是软件测试计划的重要组成部分,通常包括输入数据、期望输出等内容,其要求和设计方法将在第 8 章详细介绍,这里需要说明的一点是,负责详细设计的软件人员对模块的情况(包括功能、逻辑和接口)了解得最清楚,由他们在完成详细设计后提出对各个模块的测试要求。

5.1.2 详细设计的原则

详细设计的原则包括以下内容。

(1) 由于详细设计的蓝图是给人看的,所以模块的逻辑描述要清晰易读、正确可靠。

(2) 采用结构化设计方法改善控制结构,降低程序的复杂程度,从而提高程序的可读性、可测试性、可维护性。根据 E. W. Dijkstra 首先提出在高级语言中取消 GOTO 语句,Boehm 与 Jacopini 证明用顺序、选择、循环 3 种结构可构造任何程序结构,并能实现单入口、单出口的程序结构,IBM 公司的 Mills 进一步提出程序结构应该坚持单入口、单出口。Wirth 又对结构化程序设计的逐步求精抽象分解做了总结概括,从而形成结构程序设计的基本方法与原则,其基本内容归纳为以下几点。

① 在程序语言中应尽量少用 GOTO 语句,以确保程序结构的独立性。

② 使用单入口、单出口的控制结构,确保程序的静态结构和动态执行情况相一致,保证程序易理解。

③ 程序的控制结构一般采用顺序、选择、循环 3 种结构构成,确保结构简单。

④ 用自顶向下逐步求精方法完成程序设计。结构化程序设计的缺点是存储容量和运行时间增加 10%～20%,但易读、易维护性好。

⑤ 经典的控制结构有顺序、IF THEN ELSE 分支、DO-WHILE 循环。扩展的控制结构有多分支 CASE、DO-UNTIL 循环结构、固定次数循环 DO-WHILE。

(3) 选择恰当的描述工具描述各模块算法。

5.2 数据代码设计的工具

在理想的情况下,算法过程描述应采用自然语言来表达,这样能够使不懂软件的人较易理解这些规格说明。但是,自然语言在语法和语义上有时具有多义性,常常要参考上下文才能够把问题描述清楚,因此必须采用更严密的描述工具来表达过程细节。对详细设计的工具简述如下。

(1) 图形工具:利用图形工具可以把过程的细节用图形描述出来。

(2) 表格工具:可以用一张表来描述过程的细节,在这张表中列出了各种可能的操

作和相应的条件。

(3) 语言工具：用某种高级语言(称为伪码)来描述过程的细节。

5.2.1 程序流程图

程序流程图又称为程序框图，它是软件开发者最熟悉的一种算法表达工具。它独立于任何一种程序设计语言，比较直观和清晰地描述过程的控制流程，易于学习掌握。因此，它至今仍是软件开发者最普遍采用的一种工具。

流程图也存在一些严重的不足，主要表现在利用流程图使用的符号不够规范，人们常常使用一些习惯性用法。特别是表示程序控制流程的箭头，使用的灵活性极大，程序员可以不受任何约束，随意转移控制。这些问题常常很大地影响了程序质量。为了消除这些不足，应严格地定义流程图所使用的符号，不允许随心所欲地画出各种不规范的流程图。

为使用流程图描述结构化程序，必须限制在流程图中只使用下述 5 种基本控制结构。

1. 顺序型

顺序型由几个连续的处理步骤依次排列构成，如图 5-1 所示。

2. 选择型

选择型是指由某个逻辑判断式的取值决定选择两个处理中的一个，如图 5-2 所示。

3. WHILE 型循环

WHILE 型循环是先判定型循环，在循环控制条件成立时重复执行特定的处理，如图 5-3 所示。

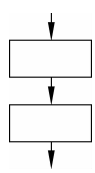

图 5-1 顺序型

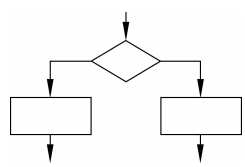

图 5-2 选择型

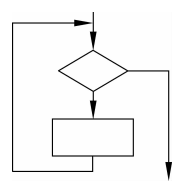

图 5-3 WHILE 型循环

4. UNTIL 型循环

UNTIL 型循环是后判定型循环，重复执行某些特定的处理，直到控制条件成立为止，如图 5-4 所示。

5. 多情况型选择

多情况型选择列举多种处理情况，根据控制变量的取值选择执行其一，如图 5-5 所示。

任何复杂的程序流程图都应由上述 5 种基本控制结构组合或嵌套而成。图 5-6 所示的是一个结构化程序的流程图。

为了能够准确地使用流程图，要对流程图所使用的符号做出确切的规定。除了按规定使用定义了的符号之外，流程图中不允许出现其他任何符号。图 5-7 给出国际标准化组织提出的已被我国国家技术监督局批准的一些程序流程图标准符号，其中大多数规定

的使用方法与普通的习惯用法一致。

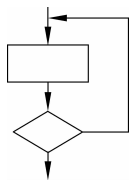

图 5-4　UNTIL 型循环

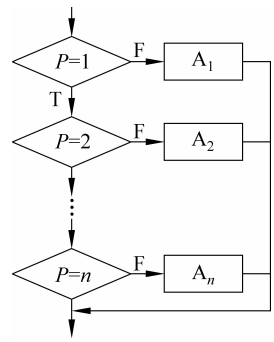

图 5-5　多情况型选择

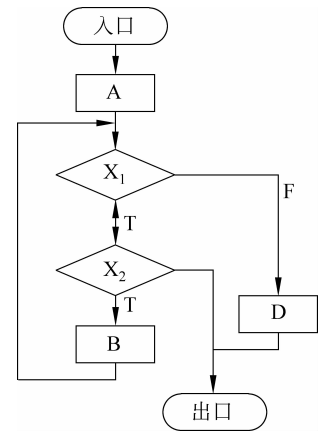

图 5-6　结构化程序流程图

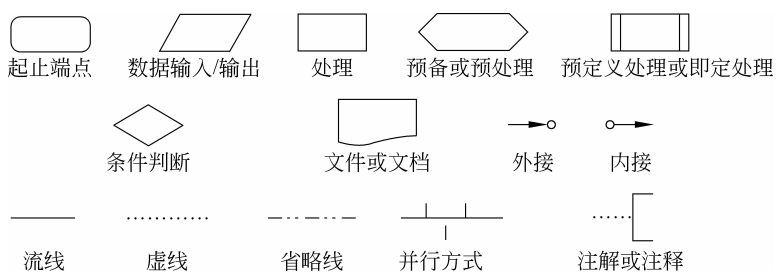

图 5-7　标准程序流程图的规定符号

5.2.2　N-S 图

Nassi 和 Shneiderman 提出了一种符合结构化程序设计原则的图形描述工具，称为盒图，又称为 N-S 图。在 N-S 图中，为了表示 5 种基本控制结构，规定了 5 种图形构件。

1. 顺序型

如图 5-8 所示，在顺序型结构中先执行 A，后执行 B。

2. 选择型

如图 5-9 所示，在选择型结构中，如果条件 P 成立，可执行 T 下面的内容，当条件 P 不成立时，则执行 F 下面的内容。

3. WHILE 重复型

如图 5-10 所示，在 WHILE 重复型循环结构中先判断 P 的值，再执行 S。其中，P 是循环条件，S 是循环体。

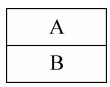

图 5-8　顺序型结构

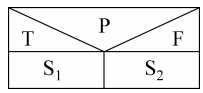

图 5-9　选择型结构

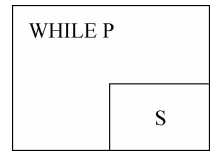

图 5-10　WHILE 重复型循环结构

4. UNTIL 重复型

如图 5-11 所示,在 UNTIL 重复型循环结构中先执行 S,后判断 P 的值。

5. 多分支选择型

图 5-12 给出了多出口的判断图形表示,P 为控制条件,根据 P 的取值相应地执行其值下面的各框内容。

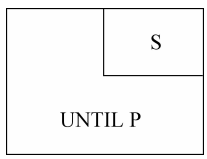

图 5-11　UNTIL 重复型循环结构

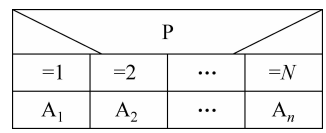

图 5-12　多分支选择型

【例 5-1】　将图 5-6 所示的程序流程图转化为 N-S 图的结果如图 5-13 所示。

N-S 图的特点如下。

(1) 图形清晰、准确。

(2) 控制转移不能任意规定,必须遵守结构化程序设计原则。

(3) 很容易确定局部数据和全局数据的作用域。

(4) 容易表现嵌套关系和模块的层次结构。

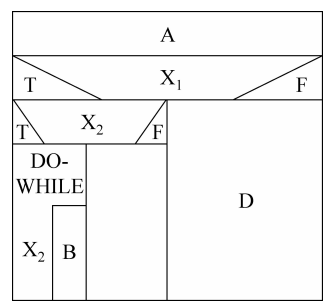

图 5-13　N-S 图举例

5.2.3　PAD 图

PAD 是 Problem Analysis Diagram 的英文缩写,它是日本的日立公司提出的,是用结构化程序设计思想表现程序逻辑结构的图形工具。

PAD 也设置了 5 种基本控制结构的图示,并允许递归使用。

1. 顺序型

如图 5-14 所示,按顺序先执行 A,再执行 B。

2. 选择型

如图 5-15 所示,给出了判断条件为 P 的选择型结构。当 P 为真值时执行上面的 A 框,当 P 取假值时执行下面的 B 框中的内容。如果这种选择型结构只有 A 框,没有 B 框,表示该选择结构中只有 THEN 后面有可执行语句 A,没有 ELSE 部分。

3. WHILE 重复型和 UNTIL 重复型

如图 5-16 所示，P 是循环判断条件，S 是循环体。循环判断条件框的右端为双纵线，表示该矩形域是循环条件，以区别于一般的矩形功能域。

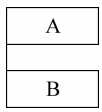

图 5-14　顺序型结构

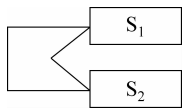

图 5-15　选择型结构

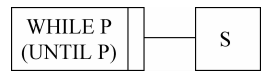

图 5-16　WHILE 重复型和 UNTIL 重复型结构

4. 多分支选择型

如图 5-17 所示，多分支选择型是 CASE 型结构。当判定条件 P 等于 1 时执行 A_1 框的内容，当 P 等于 2 时执行 A_2 框的内容，当 P 等于 n 时执行 A_n 框的内容。

5. PAD 图的应用举例

图 5-18 给出了图 5-6 所示的程序流程图的 PAD 图。

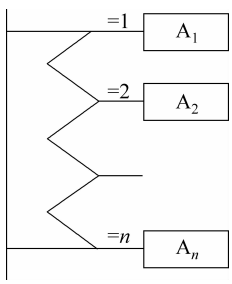

图 5-17　多分支选择型结构

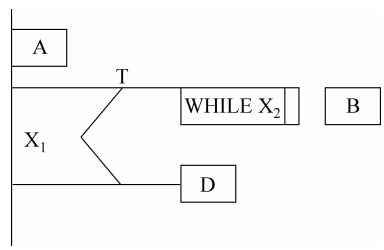

图 5-18　PAD 图举例

6. PAD 图的特点

(1) PAD 图的清晰度和结构化程度高。

(2) PAD 图中最左端是程序的主干线，即程序的第一层结构。其后，每增加一个层次，则向右扩展一条纵线。程序中的层数就是 PAD 图中的纵线数。因此，PAD 图的可读性强。

(3) 利用 PAD 图设计出的程序必定是结构化的程序。

(4) 利用软件工具可以将 PAD 图转换成高级语言程序，进而提高软件的可靠性和生产率。

(5) PAD 图支持自顶向下的逐步求精的方法。

7. PAD 图的扩充结构

为了反映增量型循环结构，在 PAD 图中增加了对应于

FOR i:= n1 to n2 step n3 do

的循环控制结构，如图 5-19(a)所示。其中，n1 是循环初值，n2 是循环终值，n3 是循环增量。

另外,PAD 所描述程序的层次关系表现在纵线上,每条纵线表示一个层次。把 PAD 图从左到右展开,随着程序层次的增加,PAD 逐渐向右展开,有可能会超过一页纸,这时,对 PAD 增加了一种如图 5-19(b)所示的扩充形式。该图中用实例说明,当一个模块 A 在一页纸上画不下时,可在图中该模块相应位置的矩形框中简记一个“NAME A”,再在另一页纸上详细地画出 A 的内容,用 def 及双下画线来定义 A 的 PAD。这种方式可使在一张纸上画不下的图在几张纸上画出,还可以用它定义子程序。

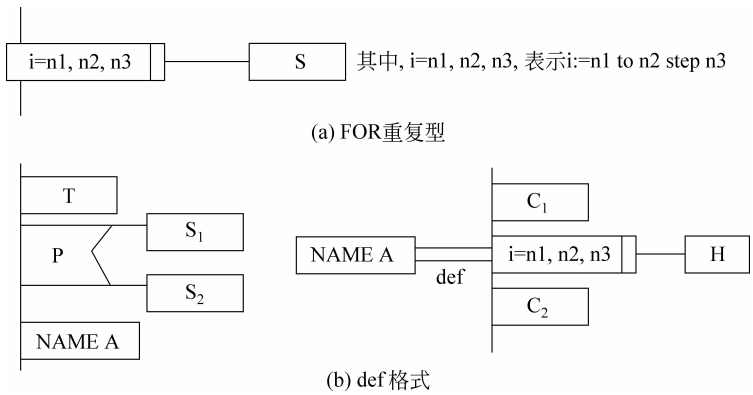

图 5-19　FOR 重复型和 def 格式

5.2.4　PDL 语言

PDL(Procedure Design Language)为过程设计语言的英文缩写,于 1975 年由 Caine 与 Gordon 在“PDL——一种软件设计工具”一文中首先提出。PDL 是所有非正文形式的过程设计工具的统称,到目前为止已出现多种 PDL 语言。PDL 具有“非纯粹”的编程语言的特点。

1. PDL 语言的特点

(1) 关键字采用固定语法并支持结构化构件、数据说明机制和模块化。

(2) 处理部分采用自然语言描述。

(3) 可以说明简单和复杂的数据结构。

(4) 子程序的定义与调用规则不受具体接口方式的影响。

2. PDL 描述选择结构

利用 PDL 描述的 IF 结构如下:

```
IF <条件>
一条或数条语句
ELSEIF <条件>
一条或数条语句
⋮
ELSEIF <条件>
一条或数条语句
```

```
ELSE
一条或数条语句
ENDIF
```

3. PDL 描述循环结构

对于 3 种循环结构，利用 PDL 描述如下：

1）WHILE 循环结构

```
DO WHILE <条件描述>
  一条或数条语句
ENDWHILE
```

2）UNTIL 循环结构

```
REPEAT UNTIL <条件描述>
  一条或数条语句
ENDREP
```

3）FOR 循环结构

```
DOFOR <循环变量>=<循环变量取值范围，表达式或序列>
ENDFOR
```

4. 子程序

```
PROCEDURE <子程序名> <属性表>
INTERFACE <参数表>
    一条或数条语句
END
```

属性表指明了子程序的引用特性和利用的程序语言的特性。

5. 输入/输出

```
READ/WRITE TO <设备>  <I/O 表>
```

综上可见，PDL 具有很强的描述功能，是一种十分灵活和有用的详细设计表达方法。

5.2.5 判定表和判定树

判定表将比较复杂的决策问题简洁、明确地描述出来，它是描述条件比较多的决策问题的有效工具。如果数据流图的加工需要依赖于多个逻辑条件的取值，使用判定表来描述比较合适。

判定表的格式说明如图 5-20 所示。

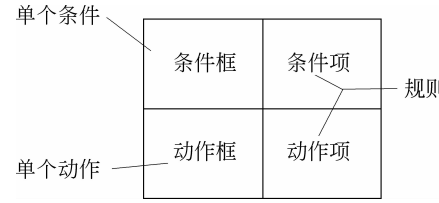

图 5-20　判定表的格式

例如某仓库的发货方案如下。

（1）客户欠款时间不超过 30 天，如果需要量不超过库存量立即发货，否则先按库存量发

货,进货后再补发。

(2) 客户欠款时间不超过 100 天,如果需要量不超过库存量先付款再发货,否则不发货。

(3) 客户欠款时间超过 100 天,要求先付欠款。

判定表表示如表 5-1 所示。

表 5-1　判定表表示

决策规则号		1	2	3	4	5	6
条件	欠款时间≤30	Y	Y	N	N	N	N
	欠款时间>100	N	N	Y	Y	N	N
	需求量≤库存量	Y	N	Y	N	Y	N
应采取的行动	立即发货	×					
	先按库存量发货,进货后再补发		×				
	先付款,再发货					×	
	不发货						×
	要求先付款			×	×		

判定树又称为决策树,是一种描述加工的图形工具,适合描述问题处理中具有多个判断的情况,而且每个决策与若干条件有关。在使用判定树进行描述时,应该从问题的文字描述中分清哪些是判定条件,哪些是判定的决策,根据描述材料中的连接词找出判定条件的从属关系、并列关系、选择关系,根据它们构造判定树。判定树用一种树形方式来表示多个条件、多个取值应采取的动作。

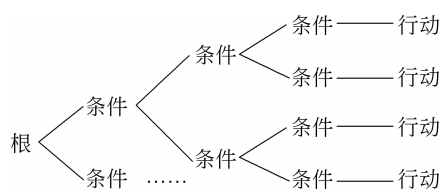

图 5-21　判定树的表示形式

判定树的表示形式如图 5-21 所示。

例如,以商店业务处理系统中的“检查发货单”为例:

```
IF 发货单金额超过$ 500 THEN
    IF 欠款超过了 60 天 THEN
        在偿还欠款前不予批准
    ELSE(欠款未超期)
        发批准书,发货单
ELSE(发货单金额未超过$ 500)
    IF 欠款超过 60 天 THEN
        发批准书、发货单及赊欠报告
    ELSE(欠款未超期)
        发批准书、发货单
```

判定树表示如图 5-22 所示。

判定表和判定树都是以图形形式描述数据流的加工逻辑，它们结构简单，易懂易读。尤其是遇到组合条件的判定，利用判定表或判定树可以使问题的描述清晰，而且便于直接映射到程序代码。在表达一个加工逻辑时，判定树、判定表都是好的描述工具，可以根据需要交叉使用。

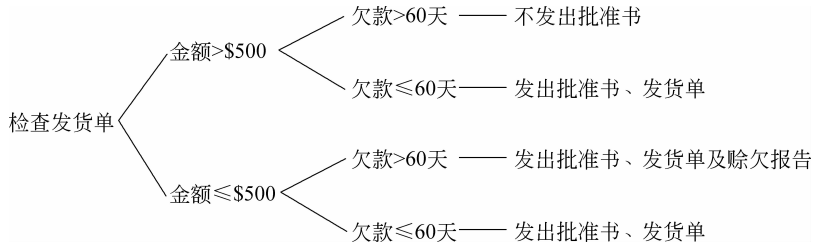

图 5-22　检查发货单判定树

5.2.6　详细设计工具的选择

在详细设计中，对一个工程设计选择的原则是使过程描述易于理解、复审和维护，过程描述能够自然地转换成代码，并保证详细设计与代码完全一致。为了达到这一原则，要求设计工具具有下述属性。

（1）模块化。支持模块化软件的开发，并提供描述接口的机制。例如，能够直接表示子程序和块结构。

（2）简洁。设计描述易学、易用和易读。

（3）便于编辑。支持后续设计和维护以及在维护阶段对设计进行的修改。

（4）机器可读性。设计描述能够直接输入，并且很容易被计算机辅助设计工具识别。

（5）可维护性。详细设计应能够支持各种软件配置项的维护。

（6）自动生成报告。设计者通过分析详细设计的结果来改进设计，通过自动处理器产生有关的分析报告，进而增强设计者在这方面的能力。

（7）强制结构化。详细设计工具能够强制设计者采用结构化构件，有助于采用优秀的设计。

（8）数据表示。详细设计具有表示局部数据和全局数据的能力。

（9）逻辑验证。软件测试的最高目标是能够自动检验设计逻辑的正确性，所以设计描述应易于进行逻辑验证，进而增强可测试性。

（10）可编码能力。可编码能力是一种设计描述，研究代码自动转换技术可以提高软件效率和减少出错率。

5.3　人-机界面设计

用户界面是用户与程序沟通的唯一途径，能为用户提供方便、有效的服务。Theo Mandel 在关于界面设计的著作中提出了三条“黄金原则”：置界面于用户的控制之下、减