“十二五”普通高等教育本科国家级规划教材

北京高等教育精品教材

21世纪软件工程专业规划教材

# 软件工程导论（第6版）

## 第5章　总体设计

张海藩，牟永敏编著　　清华大学出版社

# 第5章　总体设计

总体设计的基本目的就是回答“概括地说，系统应该如何实现”这个问题，因此，总体设计又称为概要设计或初步设计。

总体设计阶段的另一项重要任务是设计软件的结构，也就是要确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系。

# 主要内容

5.1 设计过程

5.2 设计原理

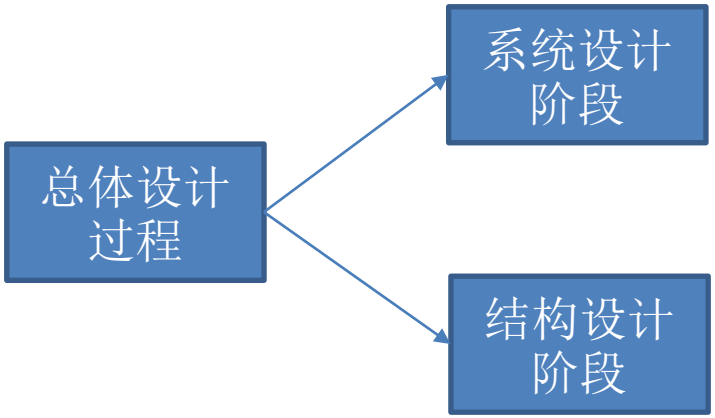
5.3 启发规则

5.4 描绘软件结构的图形工具

5.5 面向数据流的设计方法

# 主要内容

# 5.1 设计过程


总体设计过程
系统设计阶段
结构设计阶段


1.设想供选择的方案
2.选取合理的方案
3.推荐最佳方案
4.功能分解
5.设计软件结构
6.设计数据库
7.制定测试计划
8.书写文档
9.审查和复审
总体设计步骤

# 5.1 设计过程

## 典型的总体设计步骤

### 1. 设想供选择的方案

在总体设计阶段分析员应该考虑各种可能的实现方案，并且力求从中选出最佳方案。

需求分析阶段得出的数据流图是总体设计的极好的出发点。

设想供选择的方案的一种常用的方法是，设想把数据流图中的处理分组的各种可能的方法，抛弃在技术上行不通的分组方法(例如，组内不同处理的执行时间不相容)，余下的分组方法代表可能的实现策略，并且可以启示供选择的物理系统。

# 5.1 设计过程

## 2.选取合理的方案

应该从前一步得到的一系列供选择的方案中选取若干个合理的方案，通常至少选取低成本、中等成本和高成本的3种方案。在判断哪些方案合理时应该考虑在问题定义和可行性研究阶段确定的工程规模和目标，有时可能还需要进一步征求用户的意见。

对每个合理的方案，分析员都应该准备下列4份资料。

(1) 系统流程图。

(2) 组成系统的物理元素清单。

(3) 成本/效益分析。

(4) 实现这个系统的进度计划。

# 5.1 设计过程

**3.推荐最佳方案**

用户和有关的技术专家应该认真审查分析员所推荐的最佳系统，如果该系统确实符合用户的需要，并且是在现有条件下完全能够实现的，则应该提请使用部门负责人进一步审批。在使用部门的负责人也接受了分析员所推荐的方案之后，将进入总体设计过程的下一个重要阶段——结构设计。

# 5.1 设计过程

## 4.功能分解

为了最终实现目标系统，必须设计出组成这个系统的所有程序和文件(或数据库)。对程序(特别是复杂的大型程序)的设计，通常分为两个阶段完成：首先进行结构设计，然后进行过程设计。

为确定软件结构，首先需要从实现角度把复杂的功能进一步分解。分析员结合算法描述仔细分析数据流图中的每个处理，如果一个处理的功能过分复杂，必须把它的功能适当地分解成一系列比较简单的功能。

# 5.1 设计过程

## 5. 设计软件结构

通常程序中的一个模块完成一个适当的子功能。应该把模块组织成良好的层次系统，顶层模块调用它的下层模块以实现程序的完整功能，每个下层模块再调用更下层的模块，完成程序的一个子功能，最下层的模块完成最具体的功能。

## 6. 设计数据库

对于需要使用数据库的那些应用系统，软件工程师应该在需求分析阶段所确定的系统数据需求的基础上，进一步设计数据库。

# 5.1 设计过程

## 7.制定测试计划

在软件开发的早期阶段考虑测试问题，能促使软件设计人员在设计时注意提高软件的可测试性。第7章具体讨论

## 8. 书写文档

应该用正式的文档记录总体设计的结果，在这个阶段应该完成的文档通常有下述几种。

**(1) 系统说明**

**(2) 用户手册**

**(3) 测试计划**包括测试策略，测试方案，预期的测试结果，测试进度计划等

**(4) 详细的实现计划**

**(5) 数据库设计结果**

# 5.1 设计过程

**9.审查和复审**

最后应该对总体设计的结果进行严格的技术审查，在技术审查通过之后再由客户从管理角度进行复审。

# 主要内容

5.1 设计过程

5.2 设计原理

5.3 启发规则

5.4 描绘软件结构的图形工具

5.5 面向数据流的设计方法

# 5.2 设计原理

## 5.2.1 模块化

**模块**是由边界元素限定的相邻程序元素（例如，数据说明，可执行的语句）的序列，而且有一个总体标识符代表它。**模块**是构成程序的基本构件。

**模块化**就是把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求。

# 5.2 设计原理

模块化是为了使一个复杂的大型程序能被人的智力所管理，是软件应该具备的唯一属性。下面论证上面结论。

设函数C(x)定义问题x的复杂程度，函数E(x)确定解决问题x需要的工作量(时间)。对于两个问题P1和P2，如果

$$C(P1)>C(P2)$$

显然

$$E(P1)>E(P2)$$

根据人类解决一般问题的经验，另一个有趣的规律是

$$C(P1+P2)>C(P1)+C(P2)$$

# 5.2 设计原理

也就是说，如果一个问题由P1和P2两个问题组合而成，那么它的复杂程度大于分别考虑每个问题时的复杂程度之和。

综上所述，得到下面的不等式

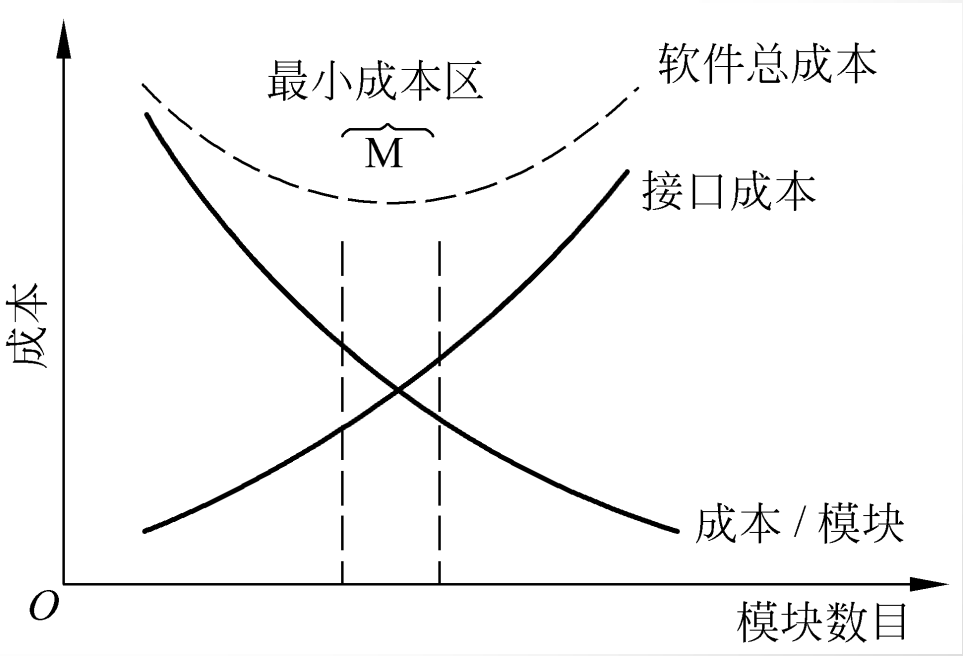
$$E(P1+P2)>E(P1)+E(P2)$$

这个不等式导致“各个击破”的结论——把复杂的问题分解成许多容易解决的小问题，原来的问题也就容易解决了。这就是模块化的根据。

# 5.2 设计原理

由上面的不等式似乎还能得出下述结论：如果无限地分割软件，最后为了开发软件而需要的工作量也就小得可以忽略了。事实上，还有另一个因素在起作用，从而使得上述结论不能成立。

如图，当模块数目增加时每个模块的规模将减小，开发单个模块需要的成本(工作量)确实减少了；但是，随着模块数目增加，设计模块间接口所需要的工作量也将增加。根据这两个因素，得出了图中的总成本曲线。每个程序都相应地有一个最适当的模块数目M，使得系统的开发成本最小。



# 5.2 设计原理

虽然目前还不能精确地决定M的数值，但是在考虑模块化的时候总成本曲线确实是有用的指南。在第六章和5.3节讲解。

采用模块化原理可以使软件结构清晰，不仅容易设计也容易阅读和理解。模块化也有助于软件开发工程的组织管理，一个复杂的大型程序可以由许多程序员分工编写不同的模块，并且可以进一步分配技术熟练的程序员编写困难的模块。

# 5.2 设计原理

## 5.2.2 抽象

人类在认识复杂现象的过程中使用的最强有力的思维工具是抽象。人们在实践中认识到，在现实世界中一定事物、状态或过程之间总存在着某些相似的方面(共性)。把这些相似的方面集中和概括起来，暂时忽略它们之间的差异，这就是抽象。或者说抽象就是抽出事物的本质特性而暂时不考虑它们的细节。

软件工程过程的每一步都是对软件解法的抽象层次的一次精化。在可行性研究阶段，软件作为系统的一个完整部件；在需求分析期间，软件解法是使用在问题环境内熟悉的方式描述的；当由总体设计向详细设计过渡时，抽象的程度也就随之减少了；最后，当源程序写出来以后，也就达到了抽象的最低层。

# 5.2 设计原理

## 5.2.3 逐步求精

逐步求精定义为为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。

逐步求精最初是由Niklaus Wirth提出的一种自顶向下的设计策略。按照这种设计策略，程序的体系结构是通过逐步精化处理过程的层次而设计出来的。通过逐步分解对功能的宏观陈述而开发出层次结构，直至最终得出用程序设计语言表达的程序。

- 求精实际上是细化过程。
- 抽象与求精是一对互补的概念。

# 5.2 设计原理

## 5.2.4 信息隐藏和局部化

信息隐藏原理：应该这样设计和确定模块，使得一个模块内包含的信息(过程和数据)对于不需要这些信息的模块来说，是不能访问的。

局部化是指把一些关系密切的软件元素物理地放得彼此靠近。

如果在测试期间和以后的软件维护期间需要修改软件，使用信息隐藏原理作为模块化系统设计的标准就会带来极大好处。

# 5.2 设计原理

## 5.2.5 模块独立

模块的独立性很重要，因为

① 有效的模块化(即具有独立的模块)的软件比较容易开发出来。

② 独立的模块比较容易测试和维护。

模块的独立程度可以由两个定性标准度量，这两个标准分别称为内聚和耦合。

# 5.2 设计原理

**1 耦合**

耦合是对一个软件结构内不同模块之间互连程度的度量。耦合强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。

模块耦合分为数据耦合、控制耦合、特征耦合、公共环境耦合和内容耦合

# 5.2 设计原理

**① 数据耦合**

两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，那么这种耦合称为数据耦合。数据耦合是低耦合。系统中至少必须存在这种耦合。

**② 控制耦合**

传递的信息中有控制信息(尽管有时这种控制信息以数据的形式出现)，则这种耦合称为控制耦合。控制耦合是中等程度的耦合。

**③ 特征耦合**

当把整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素时，就出现了特征耦合。

# 5.2 设计原理

**④ 公共环境耦合**

当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合。

公共环境可以是全程变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等。

公共环境耦合的复杂程度随耦合的模块个数而变化，当耦合的模块个数增加时复杂程度显著增加。

# 5.2 设计原理

只有两个模块有公共环境，耦合有下面两种可能。

(1) 一个模块往公共环境送数据，另一个模块从公共环境取数据。这是数据耦合的一种形式，是比较松散的耦合。

(2) 两个模块都既往公共环境送数据又从里面取数据，这种耦合比较紧密，介于数据耦合和控制耦合之间。

# 5.2 设计原理

⑤ 内容耦合

最高程度的耦合是内容耦合。如果出现下列情况之一，两个模块间就发生了内容耦合。

- 一个模块访问另一个模块的内部数据。
- 一个模块不通过正常入口而转到另一个模块的内部。
- 两个模块有一部分程序代码重叠(只可能出现在汇编程序中)。
- 一个模块有多个入口(这意味着一个模块有几种功能)。

**应该坚决避免使用内容耦合。**

# 5.2 设计原理

总之，耦合是影响软件复杂程度的一个重要因素。

应该采取下述设计原则：

尽量使用数据耦合，少用控制耦合和特征耦合，限制公共环境耦合的范围，完全不用内容耦合。

# 5.2 设计原理

**2） 内聚**

内聚衡量一个模块内部各个元素彼此结合的紧密程度。

内聚标志着一个模块内各个元素彼此结合的紧密程度，它是信息隐藏和局部化概念的自然扩展。简单地说，理想内聚的模块只做一件事情。

内聚和耦合是密切相关的，模块内的高内聚往往意味着模块间的松耦合。

内聚分为三大类低内聚、中内聚和高内聚

# 5.2 设计原理

① 低内聚

一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。

一个模块完成的任务在逻辑上属于相同或相似的一类，则称为逻辑内聚。

一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。

# 5.2 设计原理

**② 中内聚**

一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。

模块中所有元素都使用同一个输入数据和(或)产生同一个输出数据，则称为通信内聚。

**③ 高内聚**

一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行(通常一个处理元素的输出数据作为下一个处理元素的输入数据)，则称为顺序内聚。

模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

# 5.2 设计原理

耦合和内聚的概念是Constantine,Yourdon,Myers和Stevens等人提出来的。上述7种内聚的优劣评分，将得到如下结果：


事实上，没有必要精确确定内聚的级别。重要的是设计时高内聚，并且能够辨认出低内聚的模块，有能力通过修改设计提高模块的内聚程度并且降低模块间的耦合程度，从而获得较高的模块独立性。

# 主要内容

5.1 设计过程

5.2 设计原理

5.3 启发规则

5.4 描绘软件结构的图形工具

5.5 面向数据流的设计方法

# 5.3 启发规则

**1.改进软件结构提高模块独立性**

设计出软件的初步结构以后，应该审查分析这个结构，通过模块分解或合并，力求降低耦合提高内聚。

**2. 模块规模应该适中**

一个模块的规模不应过大，最好能写在一页纸内(通常不超过60行语句)

**3.深度、宽度、扇出和扇入都应适当**

深度：软件结构中控制的层数

宽度：软件结构内同一个层次上的模块总数的最大值

扇出：一个模块直接控制(调用)的模块数目

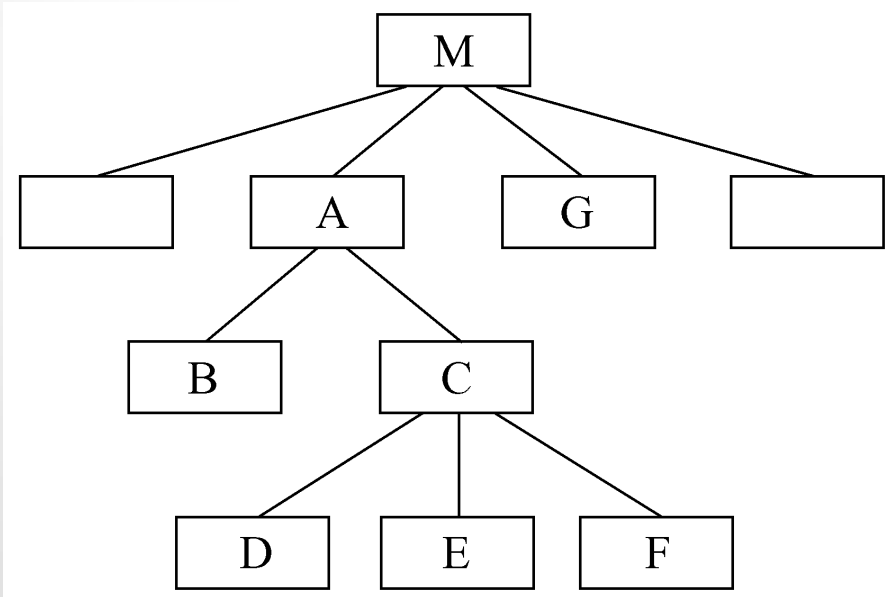
扇入：一个模块被多少个上级模块直接调用的数目

# 5.3 启发规则

**4.模块的作用域应该在控制域之内**

作用域：受该模块内一个判定影响的所有模块的集合。

控制域：模块本身以及所有直接或间接从属于它的模块的集合。




在图中模块A的控制域是A、B、C、D、E、F等模块的集合。

# 5.3 启发规则

在一个设计得很好的系统中，所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块及它的直属下级模块

怎样修改软件结构才能使作用域是控制域的子集呢?

方法1是把做判定的点往上移（例如，把判定从模块A中移到模块M中）

方法2是把那些在作用域内但不在控制域内的模块移到控制域内（例如，把模块G移到模块A的下面，成为它的直属下级模块。）

# 5.3 启发规则

**5.力争降低模块接口的复杂程度**

模块接口复杂是软件发生错误的一个主要原因。应该仔细设计模块接口，使得信息传递简单并且和模块的功能一致。

如：QUAD_ROOT(TBL,X)

求一元二次方程的根的模块，其中用数组TBL传送方程的系数，用数组X回送求得的根。

使用接口可能是比较简单，如：

QUAD_ROOT(A,B,C,ROOT1,ROOT2)

其中A、B、C是方程的系数，ROOT1和ROOT2是算出的两个根。

# 5.3 启发规则

**6.设计单入口单出口的模块**

这条启发式规则警告软件工程师不要使模块间出现内容耦合。当从顶部进入模块并且从底部退出来时，软件是比较容易理解的，因此也是比较容易维护的。

**7.模块功能应该可以预测**

模块的功能应该能够预测，但也要防止模块功能过分局限。

# 主要内容

5.1 设计过程

5.2 设计原理

5.3 启发规则

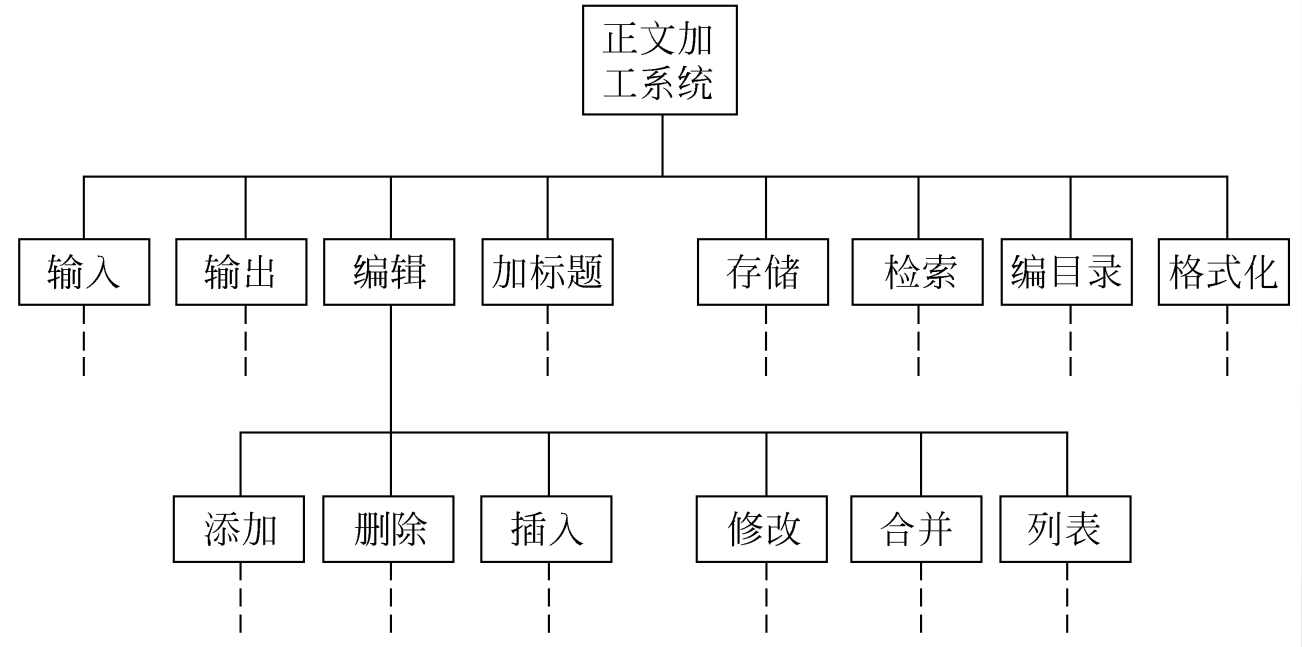
5.4 描绘软件结构的图形工具

5.5 面向数据流的设计方法

# 5.4 描绘软件结构的图形工具

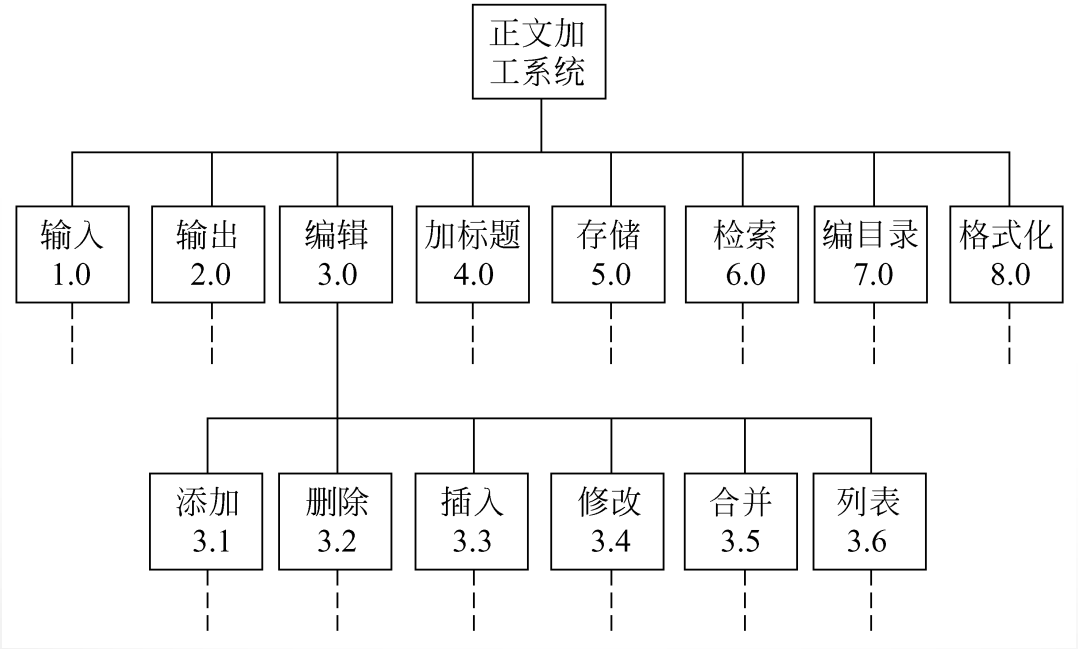
## 5.4.1 层次图和HIPO图

层次图用来描绘软件的层次结构。数据结构的层次方框图相同，但是表现的内容却完全不同。层次图很适于在自顶向下设计软件的过程中使用。



# 5.4 描绘软件结构的图形工具

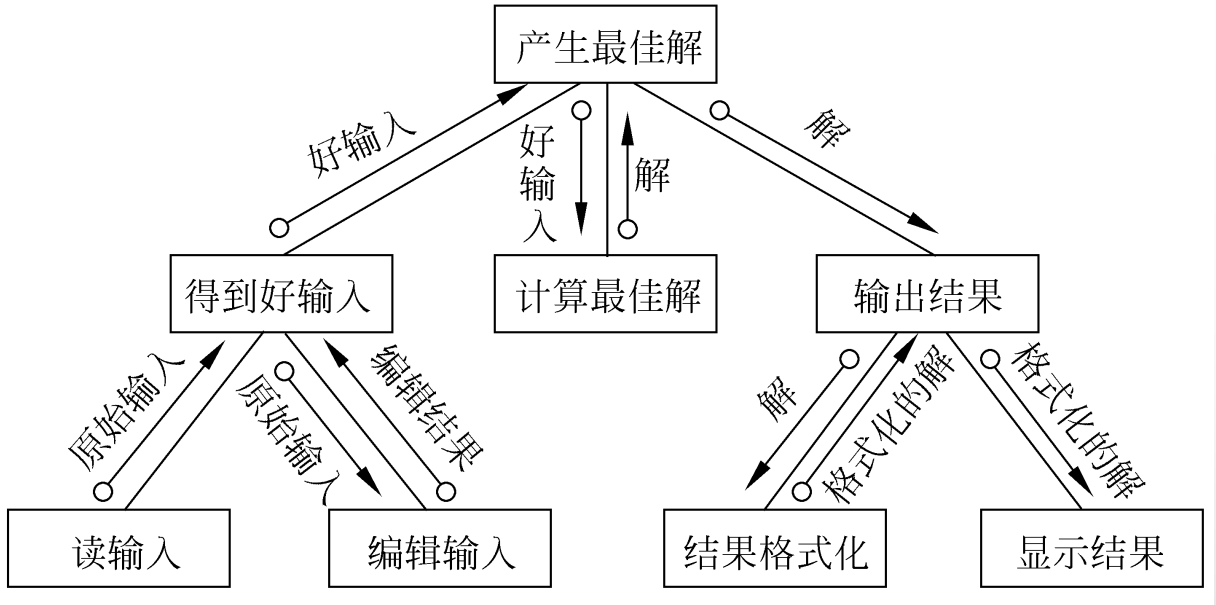
HIPO图是美国IBM公司发明的“层次图加输入/处理/输出图”的英文缩写。为了能使HIPO图具有可追踪性，在H图(层次图)里除了最顶层的方框之外，每个方框都加了编号。



# 5.4 描绘软件结构的图形工具

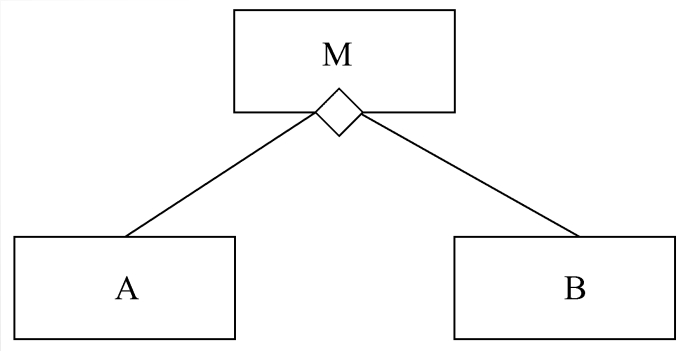
## 5.4.2 结构图

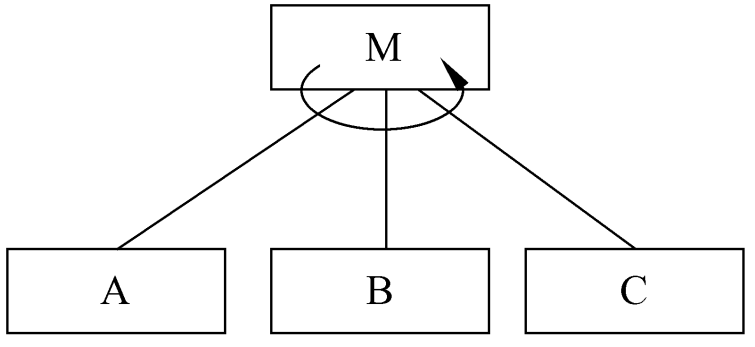
Yourdon提出的结构图是进行软件结构设计的工具。图中一个方框代表一个模块，框内注明模块的名字或主要功能；方框之间的箭头(或直线)表示模块的调用关系。尾部是空心圆表示传递的是数据，实心圆表示传递的是控制信息。



# 5.4 描绘软件结构的图形工具

一些附加的符号，可以表示模块的选择调用或循环调用。左图表示当模块M中某个判定为真时调用模块A，为假时调用模块B。右图表示模块M循环调用模块A、B和C。



# 主要内容

5.1 设计过程

5.2 设计原理

5.3 启发规则

5.4 描绘软件结构的图形工具

5.5 面向数据流的设计方法

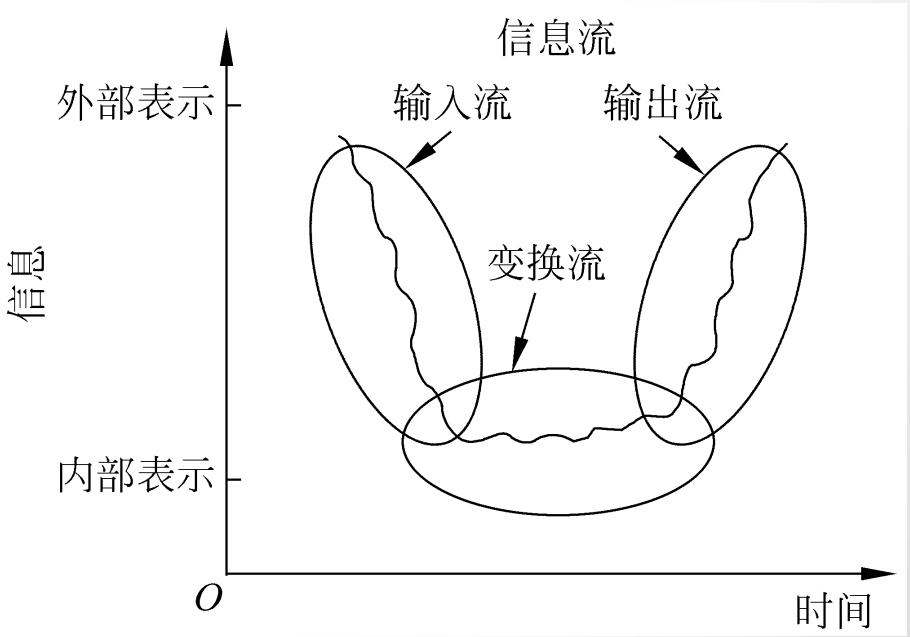
# 5.5 面向数据流的设计方法

## 5.5.1 概念

面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的方法。信息流有下述两种类型。

**1）变换流**

信息沿输入通路进入系统，由外部形式变换成内部形式，进入系统的信息通过变换中心，经加工处理以后再沿输出通路变换成外部形式离开软件系统。当数据流图具有这些特征时，这种信息流就叫作变换流。



# 5.5 面向数据流的设计方法

**2）事务流**

数据沿输入通路到达一个处理T，这个处理根据输入数据的类型在若干个动作序列中选出一个来执行。这类数据流应该划为一类特殊的数据流，称为事务流。

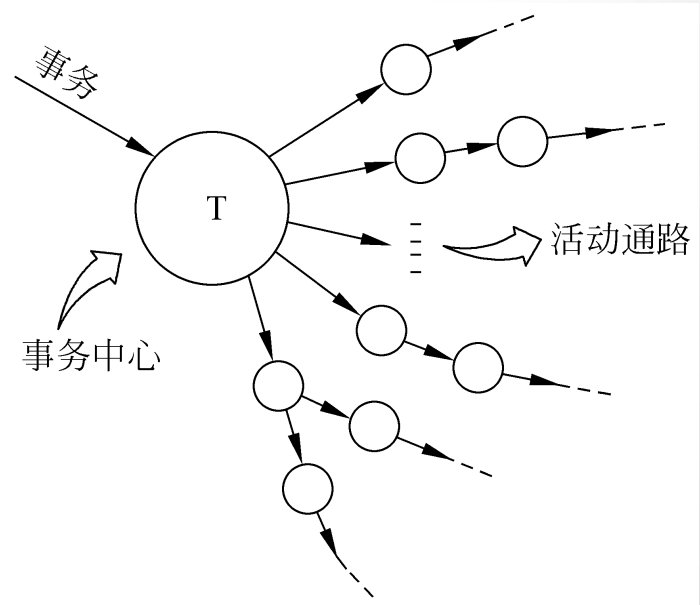
图中的处理T称为事务中心，它完成下述任务。

(1) 接收输入数据(输入数据又称

(2) 为事务)。

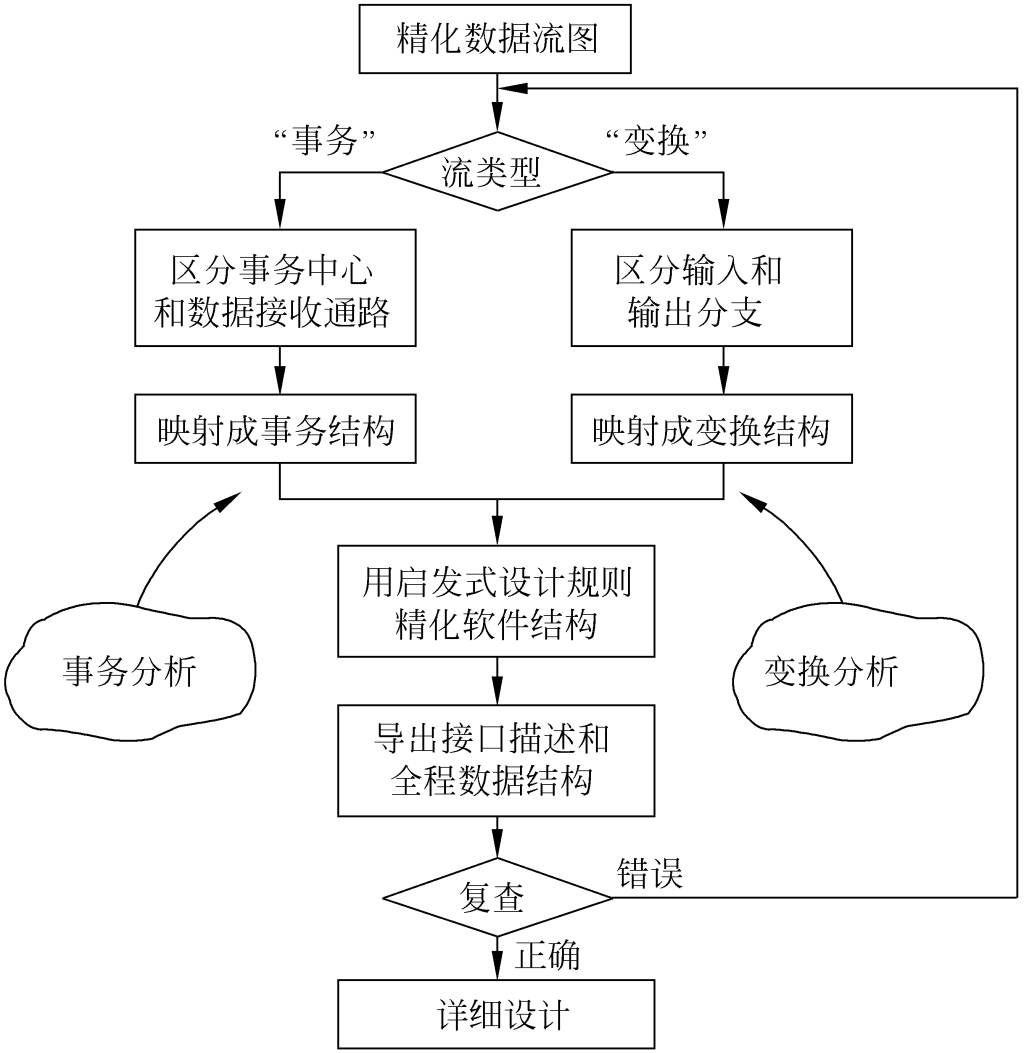
(2) 分析每个事务以确定它的类型。

(3) 根据事务类型选取一条活动通路。



# 5.5 面向数据流的设计方法

**3）设计过程**



# 5.5 面向数据流的设计方法

## 5.5.2 变换分析

变换分析是一系列设计步骤的总称，经过这些步骤把具有变换流特点的数据流图按预先确定的模式映射成软件结构。

**① 例子：汽车数字仪表板的设计**


假设的仪表板将完成下述功能。

(1) 通过模数转换实现传感器和微处理机接口。

(2) 在发光二极管面板上显示数据。

(3) 指示每小时英里数(mph)，行驶的里程每加仑油行驶的英里数(mpg)等。

(4) 指示加速或减速。

(5) 超速警告：如果车速超过55英里/小时，则发出超速警告铃声。

在软件需求分析阶段应该对上述每条要求以及系统的其他特点进行全面的分析评价，建立必要的文档资料，特别是数据流图。

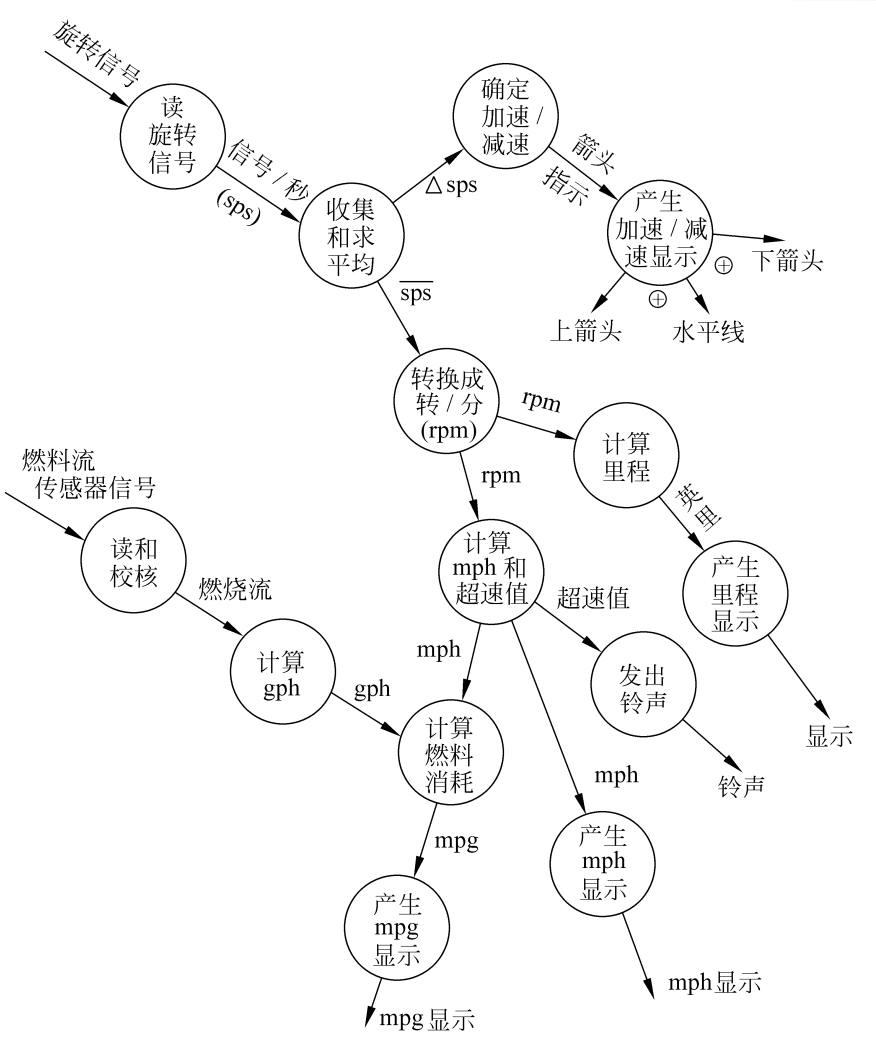
# 5.5 面向数据流的设计方法

② **设计步骤**

**第1步**复查基本系统模型。

**第2步**复查并精化数据流图。

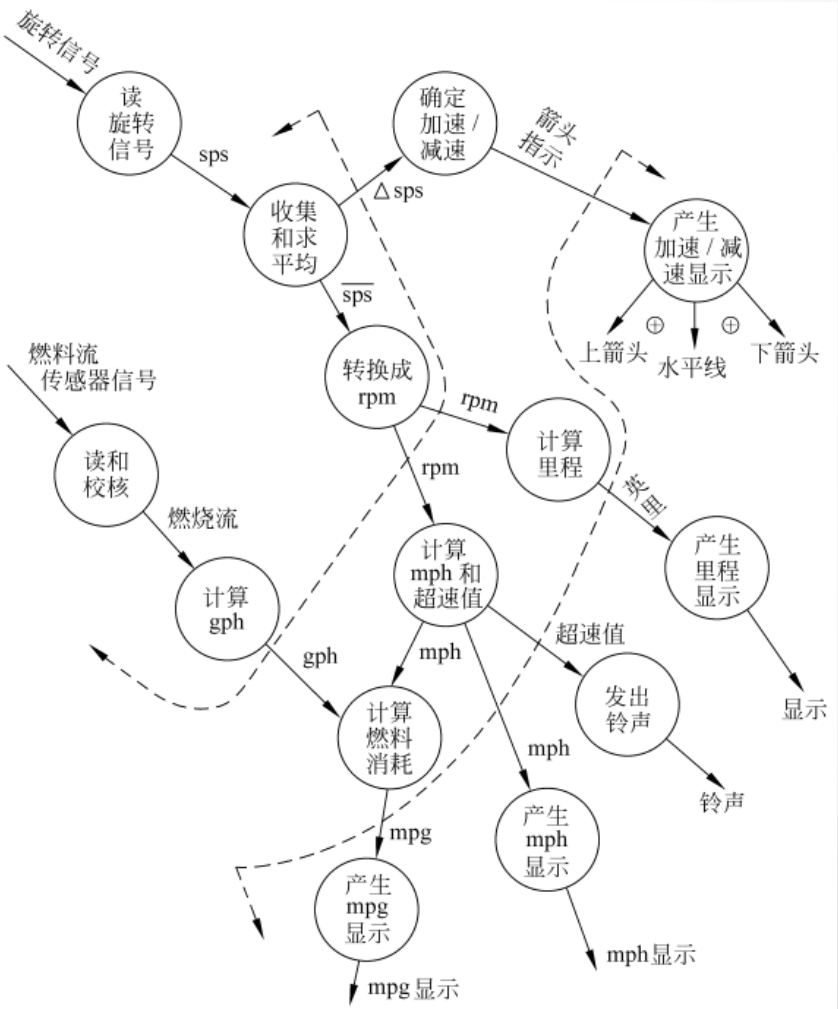
假设在需求分析阶段产生的数字仪表板系统的数据流图如图所示。



# 5.5 面向数据流的设计方法

**第3步**确定数据流图具有变换特性还是事务特性。

**第4步**确定输入流和输出流的边界，从而孤立出变换中心。



# 5.5 面向数据流的设计方法

**第5步完成“第一级分解”**

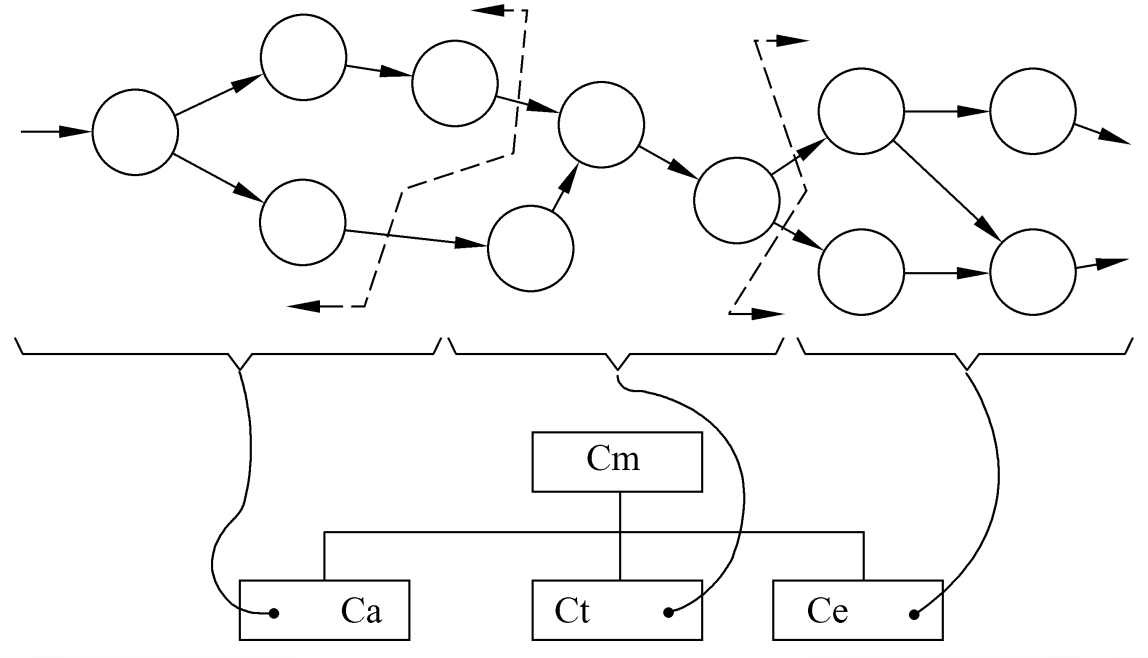
软件结构代表对控制的自顶向下的分配，所谓分解就是分配控制的过程。对于变换流的情况，数据流图被映射成一个特殊的软件结构，这个结构控制输入、变换和输出等信息处理过程。

位于软件结构最顶层的控制模块Cm协调下述从属的控制功能。

- 输入信息处理控制模块Ca,协调对所有输入数据的接收。
- 变换中心控制模块Ct,管理对内部形式的数据的所有操作。
- 输出信息处理控制模块Ce，协调输出信息的产生过程。

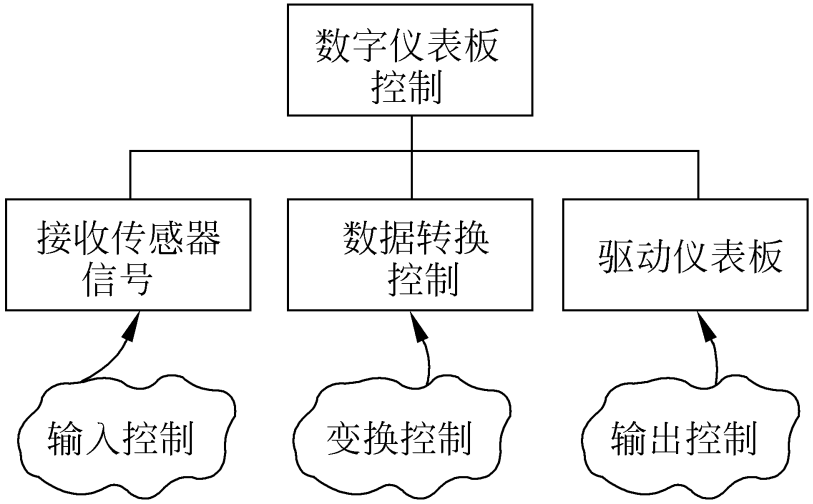
# 5.5 面向数据流的设计方法

图5.13说明了第一级分解的方法。



# 5.5 面向数据流的设计方法

对于数字仪表板的例子，第一级分解得出的结构如图所示。每个控制模块的名字表明了为它所控制的那些模块的功能。



# 5.5 面向数据流的设计方法

② 设计步骤

**第6步完成“第二级分解”**

第二级分解就是把数据流图中的每个处理映射成软件结构中一个适当的模块。

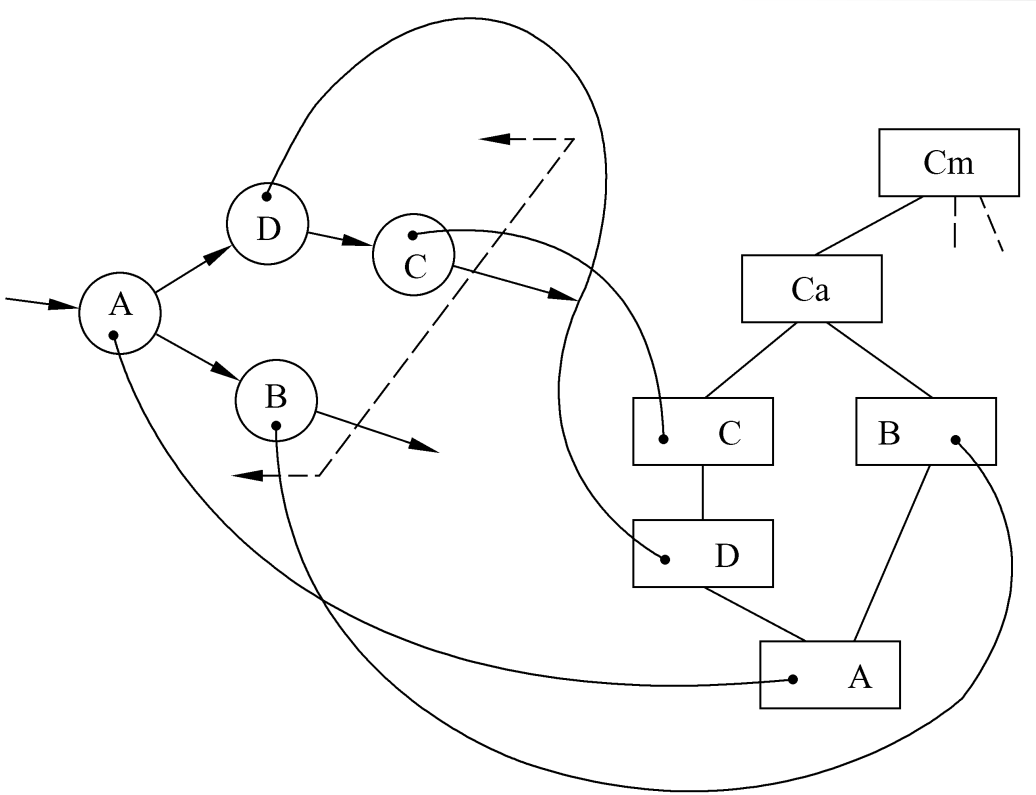
完成第二级分解的方法是，从变换中心的边界开始逆着输入通路向外移动，把输入通路中每个处理映射成软件结构中Ca控制下的一个低层模块；然后沿输出通路向外移动，把输出通路中每个处理映射成直接或间接受模块Ce控制的一个低层模块；最后把变换中心内的每个处理映射成受Ct控制的一个模块。

# 5.5 面向数据流的设计方法

② 设计步骤

**第6步完成“第二级分解”**

图表示进行第二级分解的普遍途径。



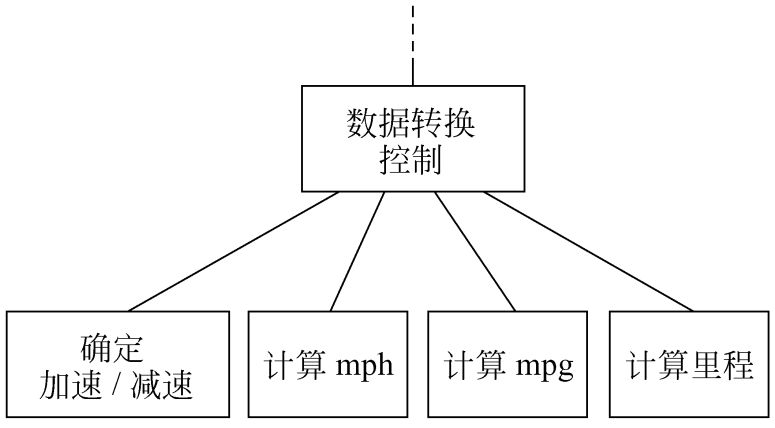
# 5.5 面向数据流的设计方法

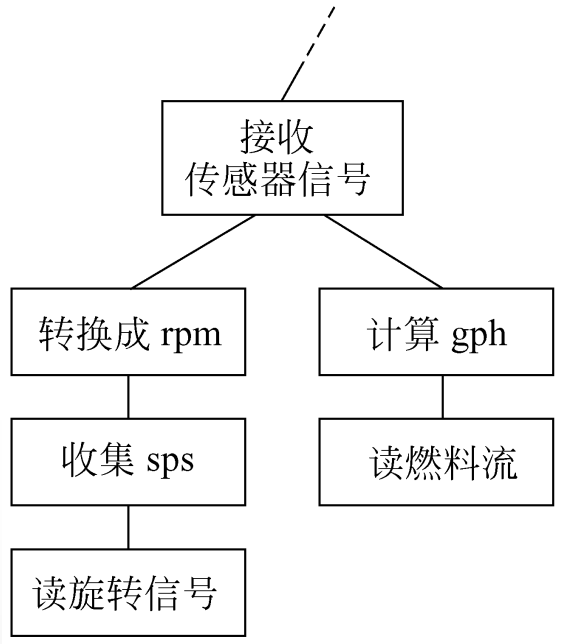
② 设计步骤

**第6步完成“第二级分解”**

应该根据实际情况以及“好”设计的标准，进行实际的第二级分解。

对于数字仪表板系统的例子，第二级分解的结果分别用下面两个图和下一张的图描绘



# 5.5 面向数据流的设计方法

② 设计步骤

**第6步完成“第二级分解”**

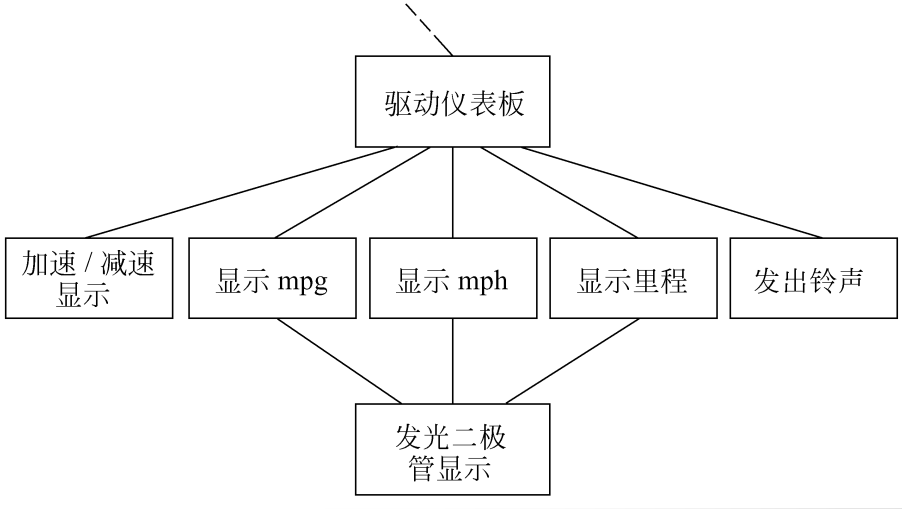
为每个模块写一个简要说明，

描述以下内容

- 进出该模块的信息(接口描述)。
- 模块内部的信息。
- 过程陈述，包括主要判定点

及任务等。

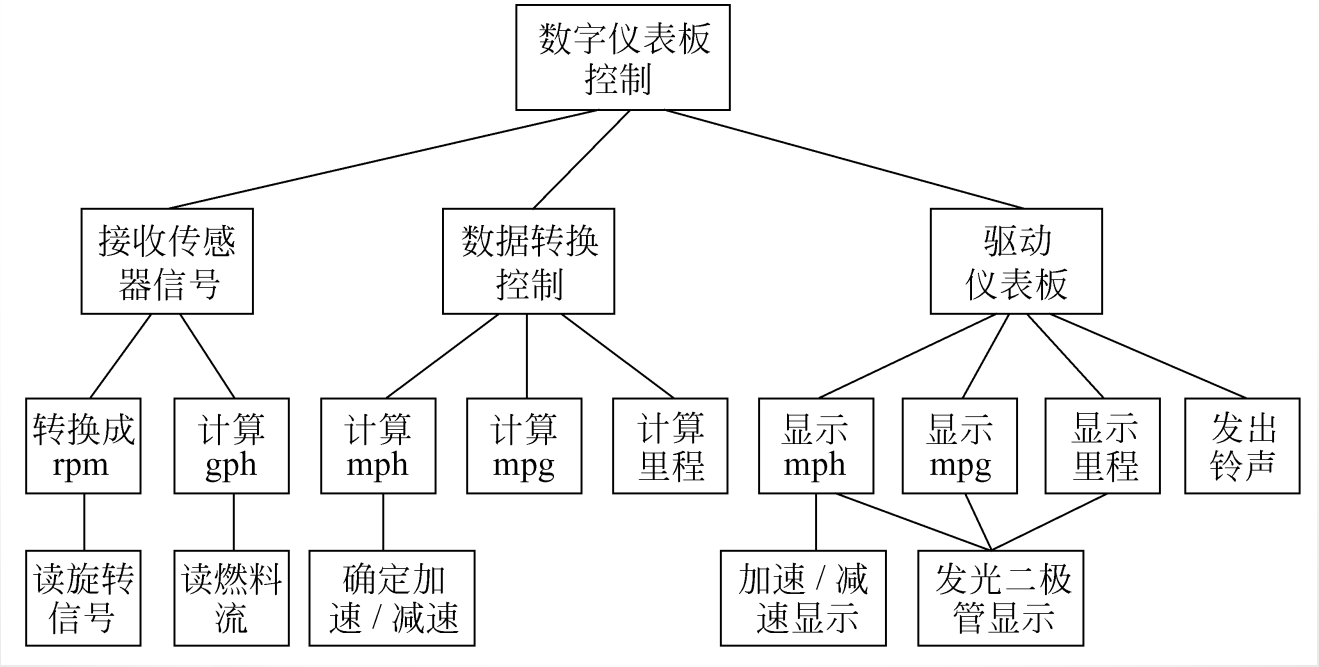
- 对约束和特殊特点的简短讨论。



# 5.5 面向数据流的设计方法

② 设计步骤

第7步使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化。

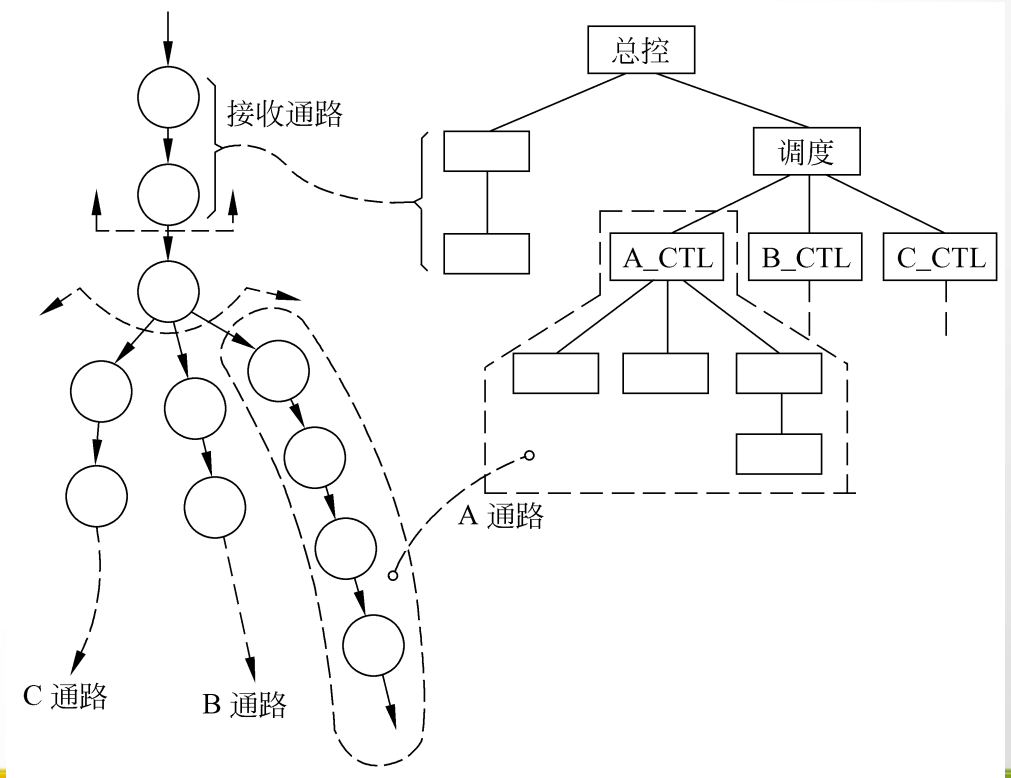


# 5.5 面向数据流的设计方法

## 5.5.3 事务分析

数据流具有明显的事务特点时采用事务分析方法。

事务分析的设计步骤和变换分析的设计步骤大部分相同或类似，主要差别仅在于由数据流图到软件结构的映射方法不同。由事务流映射成的软件结构包括一个接收分支和一个发送分支。



# 5.5 面向数据流的设计方法

## 5.5.4 设计优化

设计人员应该致力于开发能够满足所有功能和性能要求，而且按照设计原理和启发式设计规则衡量是值得接收的软件。

设计的早期阶段尽量对软件结构进行精化。

对时间起决定性作用的软件进行优化是合理的。

(1) 在不考虑时间因素的前提下开发并精化软件结构。

(2) 在详细设计阶段选出最耗费时间的那些模块，仔细地设计它们的处理过程(算法)，以求提高效率。

(3) 使用高级程序设计语言编写程序。

(4) 在软件中孤立出那些大量占用处理机资源的模块。

(5) 必要时重新设计或用依赖于机器的语言重写上述大量占用资源的模块的代码，以求提高效率。

# 本章小结

1.总体设计阶段主要由系统设计和结构设计两阶段组成。

2.进行软件结构设计时应该遵循的最主要的原理是模块独立原理。

3.在软件开发过程中既要充分重视和利用这些启发式规则，又要从实际情况出发避免生搬硬套。

4.层次图和结构图是描绘软件结构的常用工具。

5.用形式化的方法由数据流图映射出软件结构。

# 本章结束