第 17 章：软件设计概述　　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

软件设计基本原则

第17章 软件设计概述

从功能上的划分来看，软件设计应该是软件设计师的工作。作为一名软件设计师，必须懂得软件设计的基本原则和理论，掌握基本的软件设计方法，具有丰富的软件设计经验。

17.1 软件设计基本原则

在软件设计过程中，必须遵循一些原则，例如信息隐蔽和模块独立性[illegible]本的原则。

上一节　本书简介　下一节

第 17 章：软件设计概述　　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

信息隐蔽

在一节不和谐的课堂里，老师叹气道："要是坐在后排聊天的同学能像中间打牌的同学那么安静，就不会影响到前排睡觉的同学了。"

这个故事告诉我们，如果不想让坏事传播开来，就应该把坏事隐藏起来，"家丑不可外扬"就是这个道理。为了尽量避免某个模块的行为去干扰同一系统中的其他模块，在设计模块时就要注意信息隐藏。应该让模块仅仅公开必须让外界知道的内容，而隐藏其他一切内容。

在软件设计中同样有信息隐蔽原则。Parnas提出：在概要设计时列出将来可能发生变化的因素，并在模块划分时将这些因素放到个别模块的内部。也就是说，每个模块的实现细节对于其他模块来说是隐蔽的，模块中所包含的信息（包括数据和过程）不允许其他不需要这些信息的模块使用。这样，在将来由于这些因素变化而需修改软件时，只需修改这些个别的模块，其他模块不受影响。信息隐蔽技术不仅提高了软件的可维护性，而且也避免了错误的蔓延，改善了软件的可靠性。现在信息隐蔽原则已成为软件工程学中的一条重要原则。

上一节　本书简介　下一节

第 17 章：软件设计概述　　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

模块独立性

软件设计中的模块独立性是指软件系统中每个模块只涉及软件要求的具体子功能，而和软件系统中其他的模块接口是简单的。模块独立的概念是模块化、抽象、信息隐蔽和局部化概念的直接结果。

如何定义模块大小，Meyer定义了以下5条标准。

模块的可分解性：如果一种设计方法提供了将问题分解成子问题的系统化机制，它就能降低整个系统的复杂性，从而实现一种有效的模块化解决方案。

模块的可组装性：如果一种设计方法使现存的（可复用的）设计构件能被组装成新系统，它就能提供一种不需要一切从头开始的模块化解决方案。

模块的可理解性：如果一个模块可以作为一个独立的单位（不用参考其他模块）被理解，那么它就易于构造和修改。

模块的连续性：如果对系统需求的微小修改只导致对单个模块，而不是整个系统的修改，则修改引起的副作用就会被最小化。

模块的保护性：如果模块内部出现异常情况，并且它的影响限制在模块内部，则错误引起的副作用就会被最小化。

一般采用两个准则度量模块的独立性，即模块间耦合和模块内聚。

耦合是模块之间的相对独立性（互相联系的紧密程度）的度量。模块之间的联系越紧密，联系越多，耦合性就越高，而其模块独立性就越弱。

内聚是模块功能强度（一个模块内部各个元素彼此结合的紧密程度）的度量。一个模块内部各个元素之间的联系越紧密，则它的内聚性就越高；相对地，它与其他模块之间的耦合性就会减低，而模块独立性就越强。因此，模块独立性比较强的模块应是高内聚、低耦合的模块。

1.内聚

内聚是信息隐蔽功能的自然扩展。内聚的模块在软件过程中完成单一的任务，同程序其他部分执行的过程交互很少，简而言之，内聚模块（理想情况下）应该只完成一件事。在设计模块时应尽量争取高内聚。

一般模块的内聚性分为7种，如图17-1所示。

图17-1 模块的内聚性

一般认为，巧合（偶然）、逻辑和时间上的聚合是低聚合性的表现；信息的聚合则属于中等聚合性；顺序的和功能的聚合是高聚合性的表现。表17-1列出了各类聚合性与模块各种属性的关系。

表17-1 各类聚合性与模块各种属性的关系

	内部联系	清晰性	可重用性	可修改性	可理解性
巧合内聚	很差	差	很差	很差	很差
逻辑内聚	很差	很差	很差	很差	差
时间内聚	差	中	很差	中	中
过程内聚	中	好	差	中	中
通信内聚	好	中	中	中	中
信息内聚	好	好	中	好	好
功能内聚	好	好	好	好	好

1）功能内聚（Functional Cohesion）

一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作、紧密联系、不可分割的，则称该模块为功能内聚模块。它是内聚程度最高的，也是模块独立性最强的模块。

2）信息内聚（Informational Cohesion）

这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该执行哪一个功能。由于这个模块的所有功能都是基于同一个数据结构（符号表）的，因此，它是一个信息内聚的模块。

信息内聚模块可以看成是多个功能内聚模块的组合，并且达到信息的隐蔽。即把某个数据结构、资源或设备隐蔽在一个模块内，不为别的模块所知晓。

3）通信内聚（Communication Cohesion）

如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据，则称为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。

4）过程内聚（Procedural Cohesion）

使用流程图作为工具设计程序时，把流程图中的某一部分划出组成模块，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成3个模块，这3个模块都是过程内聚模块。

5）时间内聚（Classical Cohesion）

时间内聚又称为经典内聚。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行，如初始化模块和终止模块。

6）逻辑内聚（Logical Cohesion）

这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。

7）巧合内聚（Coincidental Cohesion）

巧合内聚又称为偶然内聚。模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为巧合内聚模块，它是内聚程度最低的模块。

2.耦合

耦合是程序结构中模块相互关联的度量。耦合取决于各个模块间接口的复杂程度、调用模块的方式，以及哪些信息通过接口。

耦合的强度依赖于以下几个因素：

一个模块对另一个模块的调用。

一个模块向另一个模块传递的数据量。

一个模块施加到另一个模块的控制的多少。

模块之间接口的复杂程度。

一般模块之间可能的连接方式有7种，它们构成耦合性的7种类型，如图17-2所示。

图17-2 模块之间的耦合性

耦合是影响软件复杂程度的一个重要因素。在软件设计过程中，应尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合。表17-2列出了各类耦合性与模块各种属性的关系。

表17-2 各类耦合性与模块各种属性的关系

	对修改的敏感性	可重用性	可修改性	可理解性
内容耦合	很强	很差	很差	很差
公共耦合	强	很差	中	很差
外部耦合	一般	很差	很差	中

	对修改的敏感性	可重用性	可修改性	可理解性
控制耦合	一般	差	差	差
标记耦合	不一定	中	中	中
数据耦合	不一定	好	好	好
非直接耦合	好	好	好	好

1）非直接耦合（Nondirective Coupling）

如果两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的，这就是非直接耦合。这种耦合的模块独立性最强。

2）数据耦合（Data Coupling）

如果一个模块访问另一个模块时，彼此之间是通过简单数据参数（不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的，则称这种耦合为数据耦合。

3）标记耦合（Stamp Coupling）

如果一组模块通过参数表传递记录信息，就是标记耦合。这个记录是某一数据结构的子结构，而不是简单变量。

4）控制耦合（Control Coupling）

如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。

5）外部耦合（External Coupling）

一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，称为外部耦合。

6）公共耦合（Common Coupling）

若一组模块都访问同一个公共数据环境，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。

公共耦合的复杂程度随耦合模块的个数增加而显著增加。若只是两模块间有公共数据环境，则公共耦合有两种情况：松散公共耦合和紧密公共耦合。

7）内容耦合（Content Coupling）

如果发生下列情形，两个模块之间就发生了内容耦合：

一个模块直接访问另一个模块的内部数据。

一个模块不通过正常入口转到另一模块内部。

两个模块有一部分程序代码重叠（只可能出现在汇编语言中）。

一个模块有多个入口。

3.深度、宽度、扇出与扇入

深度表示软件结构中控制的层数。如果层数过多则应考虑是否有许多管理模块过于简单了，能否适当合并。

宽度是软件结构中同一个层次上的模块总数的最大值。一般说来，宽度越大系统越复杂。对宽度影响最大的因素是模块的扇出。

一个模块的扇出是指该模块直接调用的下级模块的个数。扇出大表示模块的复杂度高，需要控制和协调过多的下级模块；但扇出过小（例如总是1）也不好。扇出过大一般是因为缺乏中间层次，应该适当增加中间层次的控制模块。扇出太小时可以把下级模块进一步分解成若干个子功能模块，

或者合并到它的上级模块中去。

一个模块的扇入是指直接调用该模块的上级模块的个数。扇入大表示模块的复用程度高。

设计良好的软件结构通常顶层扇出比较大，中间扇出较小，底层模块则有大扇入。

但我们也应当注意，不应为了单纯追求深度、宽度、扇出与扇入的理想化而违背模块独立原则，分解或合并模块必须符合问题结构。

4.作用域和控制域

模块的作用域是指受该模块内一个判定影响的所有模块的集合。模块的控制域是指该模块本身及被该模块直接或间接调用的所有模块的集合。软件设计时，模块的作用域应在控制域之内，作用域最好是做出判定的模块本身及它的直属下级模块。

5.功能的可预测性

功能可预测是指对相同的输入数据能产生相同的输出。软件设计时应保证模块的功能是可以预测的。

上一节　本书简介　下一节

结构化设计方法

结构化设计方法是在模块化、自顶向下逐层细化、结构化程序设计等程序设计技术的基础上发展起来的。该方法实施的过程如下：

总结出系统应有的功能，对一个功能，从功能完成的过程考虑，将各个过程列出，标识出过程转向和传递的数据。这样，可以将所有的过程都画出来。

细化数据流。确定应该记录的数据。

分析各过程之间的耦合关系，合理地进行模块划分以提高它们之间的内聚性。

上一节　本书简介　下一节

系统结构图中的模块

模块就如同人的器官，具有特定的功能。人体中最出色的模块设计之一是手，手只有几种动作，却能做无限多的事情。人体中最糟糕的模块设计之一是嘴巴，嘴巴将最有价值但毫不相干的几种功能，如吃饭、说话、亲吻，混为一体，使之无法并行处理，真乃人类之不幸。

在系统结构图中，不能再分解的底层模块称为原子模块。如果一个软件系统的全部实际加工都由原子模块来完成，而其他所有非原子模块仅仅执行控制或协调功能，这样的系统就是完全因子分解的系统。如果系统结构图是完全因子分解的，就是最好的系统。但实际上，这只是力图达到的目

标，大多数系统做不到完全因子分解。

一般来说，结构图中可能出现如图17-3所示的4种类型的模块。

传入模块：如图17-3（a）所示，从下属模块取得数据，经过某些处理，再将其传送给上级模块。它传送的数据流叫做逻辑输入数据流。

传出模块：如图17-3（b）所示，从上级模块取得数据，进行某些处理，传送给下属模块。它传送的数据流叫做逻辑输出数据流。

变换模块：如图17-3（c）所示，从上级模块取得数据，进行特定处理后，送回原上级模块。它加工的数据流叫做变换数据流。

协调模块：如图17-3（d）所示，对其下属模块进行控制和管理的模块。在一个好的系统结构图中，协调模块应在较高层出现。

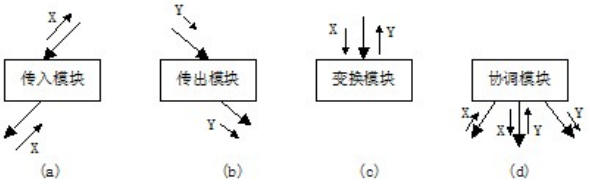

图17-3 4种模块类型

值得注意的是，结构图着重反映的是模块间的隶属关系，即模块间的调用关系和层次关系。它和程序流程图（常称为程序框图）有着本质的差别。程序流程图着重表达的是程序执行的顺序及执行顺序所依赖的条件。结构图则着眼于软件系统的总体结构，它并不涉及模块内部的细节，只考虑模块的作用，以及它和上、下级模块的关系。而程序流程图则用来表达执行程序的具体算法。

没有学过软件开发技术的人，一般习惯于使用流程图编写程序，往往在模块还未做划分、程序结构的层次尚未确定以前，便急于用流程图表达他们对程序的构想。这就像造一栋大楼，在尚未决定建筑面积和楼层有多少时，就已经开始砌砖了。这显然是不合适的。

Adele Goldberg在《Succeeding with Objects》中叙述了一位犹太教教士在新年伊始的宗教集会上讲述的故事：

一位教士登上一列火车，由于他经常乘坐这辆车，因此列车长认识他。教士伸手到口袋中掏车票，但没有找到，他开始翻他的行李。列车长阻止了他："教士，我知道您肯定有车票。现在别急着找。等找到后再向我出示。"但教士仍在找那张车票。当列车长再次见到他时，教士说："你不明白。我知道你相信我有车票，但……我要去哪里呢？"

有太多项目失败就是因为它们没有明确的目标就开始了。

在结构化分析和设计技术中，通常存在着两种典型的问题类型：变换型问题和事务型问题。它们的数据流图和结构图都有明显的特征。下面分别讨论它们的数据流图形态及其映射成结构图的过程。

结构图（Structured Charts,简称SC）是准确表达程序结构的图形表示方法，它能清楚地反映出程序中各模块间的层次关系和联系。与数据流图反映数据流的情况不同，结构图反映的是程序中控制流的情况。如图17-4所示为某大学教务管理系统的结构图。

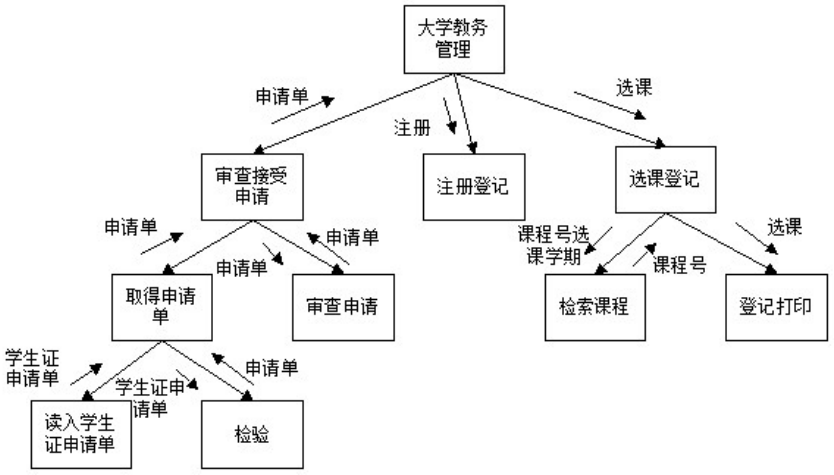

图17-4 大学教务管理系统结构图

上一节 本书简介 下一节

第 17 章：软件设计概述 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

系统结构图中的主要成分

结构图中有以下主要成分。

1.模块

以矩形框表示，框中标有模块的名字。对于已定义（或者已开发）的模块，则可以用双纵边矩形框表示，如图17-5所示。

图17-5 模块的表示

2.模块间的调用关系

两个模块，一上一下，以箭头相连，上面的模块是调用模块，箭头指向的模块是被调用模块，如图17-6所示，模块A调用模块B.在一般情况下，箭头表示的连线可以用直线代替。

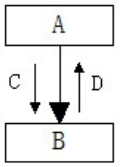

图17-6 模块的调用关系及信息传递关系的表示

3.模块间的通信

模块间的通信以表示调用关系的长箭头旁边的短箭头表示，短箭头的方向和名字分别表示调用模块和被调用模块之间信息的传递方向和内容。如图17-6所示，首先模块A将信息C 传给模块B,经模块B加工处理后的信息D再传回给A.

4.辅助控制符号

当模块A有条件地调用模块B时，在箭头的起点标以菱形。模块A反复地调用模块D时，另加一环状箭头。如图17-7所示。

在结构图中条件调用所依赖的条件和循环调用的循环控制条件通常都无须注明。

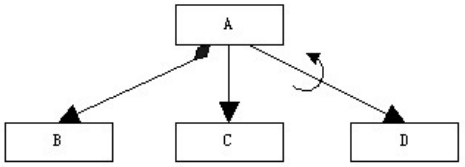

图17-7 条件调用和循环调用的表示

上一节 本书简介 下一节

 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

常用的系统结构图

常用的系统结构图有以下几种。

1.变换型系统结构图

在数据处理问题中，我们通常会遇到这样一类问题，即从（程序）"外部"（如从键盘、磁盘文件等）取得数据，对取得的数据进行某种变换，然后再将变换得到的数据传回给"外部".其中，取得数据这一过程称为传入信息（数据）流程，变换数据的过程称为变换信息（数据）流程，传回数据的过程称为传出信息（数据）流程，如图17-8所示。

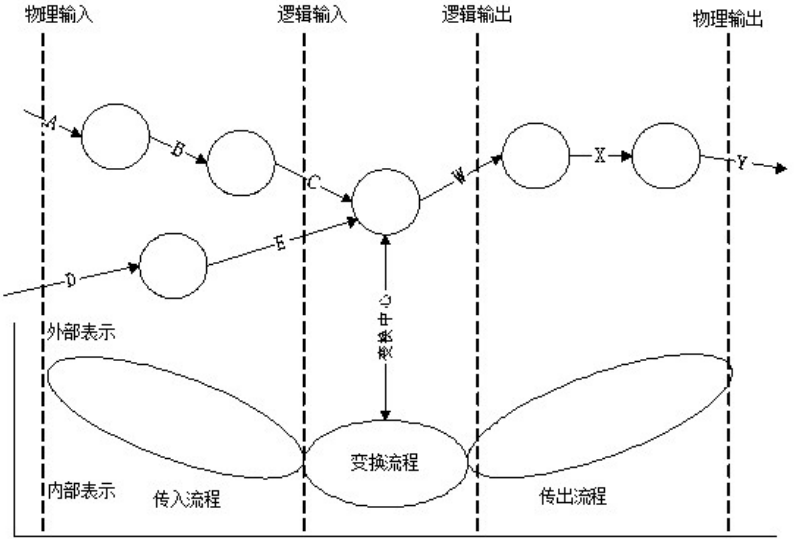

图17-8 变换型问题

当数据流图或其中某一段数据流表现出上述特征时，该数据流图或该段数据流图表示的就是一个变换型问题。完成数据变换的处理单元叫变换中心。变换型问题数据流图基本形态及其对应的基本结构图分别如图17-9（a）和图17-9（b）所示。

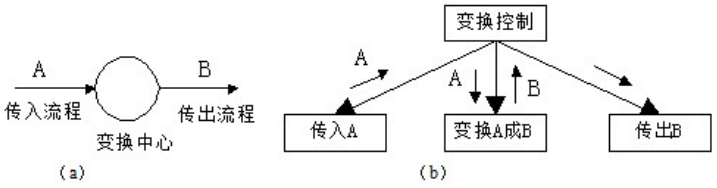

图17-9 基本变换型问题数据流图及其结构图

根据图17-9表示的基本映射关系所得到的如图17-8所示变换型问题的结构图如图17-10所

示。"变换控制"模块首先获得控制，然后控制沿着结构到达底层的"传入A"模块，物理输入数据A由"传入A"模块读入后，从底层逐步向上传送。在传送过程中，数据经"变换A成B"、"变换B成C"等模块的预处理，逐渐变换成纯粹的逻辑输入C、E.接着在"变换控制"模块的控制下，将逻辑输入经变换中心模块"变换C和E成W"处理后，变换成逻辑输出W,再从顶层逐步向下传送。在这一传送过程中，数据经"变换W成X"、"变换X成Y"等模块的后处理，逐渐变换成适当的输出形式，最后由"传出Y"模块完成物理输出。

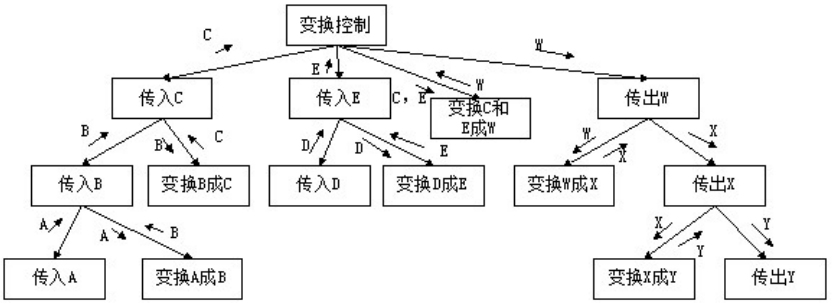

图17-10 变换型问题结构图

2.事务型系统结构图

在实际中，我们还常常会遇到另一类问题，即通常在接受某一项事务后，根据事务的特点和性质，选择分派给它一个适当的处理单元，然后给出结果，如图17-11所示。

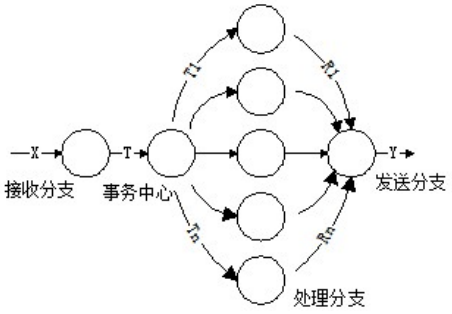

图17-11 事务型问题

这类问题就是事务型问题。它的特点是，数据沿着接收分支把外部信息（数据）转换成一个事务项，然后计算该事务项的值，并根据它的值从多条数据流中选择其中的某一条数据流。发出多条数据流的处理单元叫事务中心。这类问题的典型结构图如图17-12所示。事务控制模块按所取得事务的类型，选择调用某一个处理事务模块。

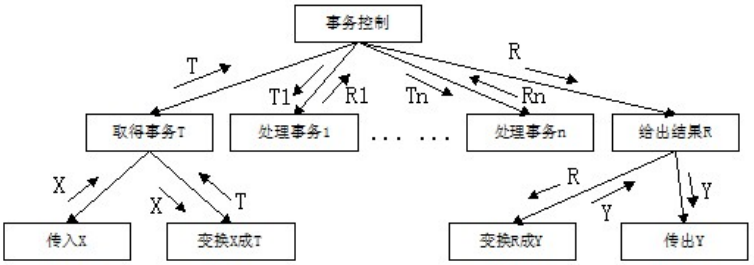

图17-12 事务型问题结构图

3.混合型系统结构图

在实际中，一些大型问题往往既不是单纯的变换型问题，也不是单纯的事务型问题，而是两种混合在一起的混合型问题。图17-13表示的就是一个混合型问题数据流图，图17-14是图17-13相应的结构图。

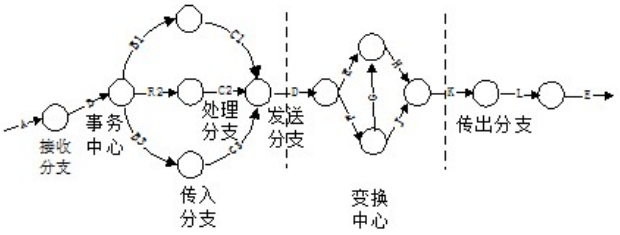

图17-13 混合型问题数据流图

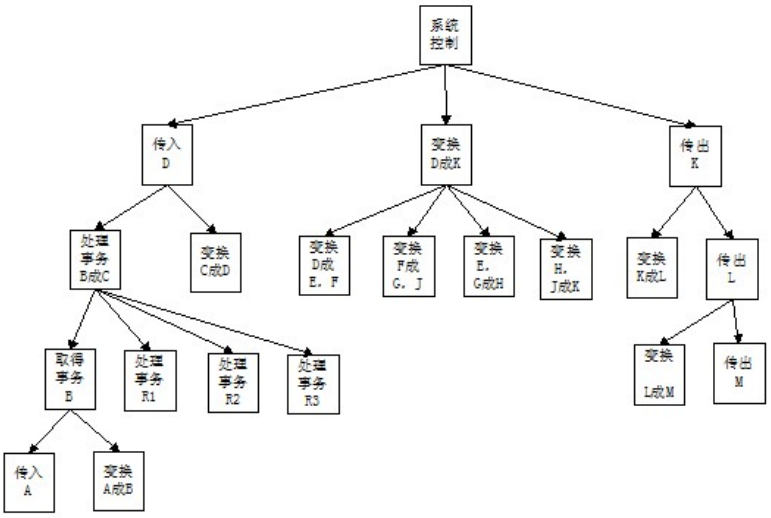

图17-14 混合型问题结构图

对于这种混合型问题，一般以变换型问题为主，首先找出变换中心，设计出结构图的上层；然后根据数据流图的各部分具体类型分别映射得到它们的结构图。例如，对于图17-14所表示的混合型问题，从整体上可以将其看成是一个从A到M的变换型问题：从D到K之间的变换是变换中心；从A到D是传入分支，具有事务型问题的特点；从K到M是传出分支。因此，该混合型问题结构图的上层可以由"传入D"模块、"变换D成K"模块和"传出K"模块组成，"传入D"模块的下层结构图由从传入分支映射得到的事务型问题结构图组成，"变换D成K"模块和"传出K"模块的下层结构图可以按通常的变换型问题映射方法获得。

上一节 本书简介 下一节

面向对象设计

为了讨论面向对象的技术和方法，必须首先明确什么是"面向对象".为什么要讨论面向对象的方法？什么是对象？

上一节 本书简介 下一节

对于上述这些问题，有许多不同的看法。其中Booch、Coad/Yourdon 和Jacobson的方法在面向对象软件开发界得到了广泛的认可。特别值得一提的是统一建模语言（Unified Modeling Language,UML），该方法结合了Booch、OMT和Jacobson方法的优点，统一了符号体系，并从其他的方法和工程实践中吸收了许多经过实际检验的概念和技术。

1.对象

对象是建立面向对象程序所依赖的基本单元。用更专业的话来说，所谓对象就是一种代码的实例，这种代码执行特定的功能，具有自包含或者封装的性质。这种封装代码通常叫做类、对象类、模块或者在不同编程语言中所应用的其他名称。以上这些术语在含义上稍微有些不同，但它们都是代码的集合。

正如上面提到的那样，对象本身是类或者其他数据结构的实例。这就是说，现有的代码起到了创建对象的模板作用。执行特定功能的代码只需要编写一次却可以被引用多次。每一种对象具有自己的标识，也就是令对象相互区别的对象名称。

对象并不是类的实际拷贝。每一对象都有自己的名称空间，在这种名称空间中保存自己的标识符和变量，但是对象要引用执行函数的原有代码。

"封装"的对象具有自己的函数，这种函数被称做"方法",而对象的变量则被称为属性。当对象内部定义了属性的时候，它们通常不能扩展到实例以外。假设现有一个类叫autocar（汽车），同时又创建了两个对象实例sedan（小轿车）和bus（客车），那么给sedan设置的值就不会影响到bus内部的值。autocar自身内部的变量却永远不会得到定义，因为autocar类只是一种模板。

在特定的场合下，有些函数确实会影响类而不是由类所创建的对象。类属性指的是专门设计来保留对象之间所用的值。类方法则用来定义和跟踪类属性。

某些编程语言可以让用户调用类的函数而不是创建整个实例。如果函数被分配以标识符（或者句柄），在某些情况下它们可以被视做具有自身权限的对象。不过，在大多数的情况下，函数只是用来实现某种结果的方法。

2.类

类是对象的抽象定义，是一组具有相同数据结构和相同操作的对象的集合。类的定义包括一组数据属性和在数据上的一组合法操作。类定义可以视为一个具有类似特性与共同行为的对象的模板，可用来产生对象。在一个类中，每个对象都是类的实例（Instance），它们都可使用类中提供的函数。一个对象的状态则包含在它的实例变量中。

3.继承

继承是使用已存在的定义作为基础建立新定义的技术。新类的定义可以是现存类所声明的数据、定义与新类所增加的声明的组合。新类复用现存类的定义，而不要求修改现存类。因为这种类的一部分已经实现和测试，故开发费用较少。现存类可当做父类（泛化类、基类或超类）来引用，则新类相应地可当做子类（特化类、子女类或派生类）来引用。

面向对象分析方法

面向对象分析（Object-Oriented Analyst,OOA）方法是对问题空间的理解和分析，分析阶段通过类或对象的认定，确定类之间（或对象间）的关系，然后对它们的属性、所提供的方法和所需要的方法进行描述，并按照它们之间的关系进行组织，得到类（或对象）结构。OOA建立在信息模拟（ER图和语义数据模型）和面向对象程序设计语言的概念基础上。如图17-15所示。从信息模拟中吸取了属性、关系、结构及对象作为问题域中某些事物的实例的表示方法等概念，从面向对象程序设计语言中吸取了属性和方法的封装，属性和方法作为一个不可分割的整体，以及各类结构和继承等概念。

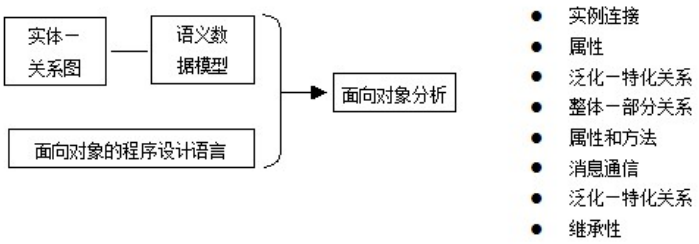

图17-15　OOA的形成

OOA基于以下几点：

在分析和规格说明的总体框架中贯穿结构化方法，如对象和属性、整体和部分、类和成员等。

用消息进行用户和系统之间，以及系统中实例间的相互通信。

在识别每个部分所提供方法的总体框架中对性能进行分类。

上一节　本书简介　下一节

面向对象设计

分析阶段主要是明确用户的功能需求，即满足用户所需的系统部件及其结构。设计阶段则主要是确定实现用户需求的方法，即怎样做才能满足用户需求，并构造出系统的实现蓝图。面向对象设计（Object-Oriented Design,OOD）也是如此，只不过是引入了一些面向对象的概念和原则，用以指导设计工作。OOD首先从OOA的结果开始，并将其从问题域映射到实现域；为满足实现的需要，还要增加一些类、结构及属性和服务，并对原有类及属性进行调整。此外，还要完成应用控制、人机交互界面的设计等。

面向对象的设计方法主要有以下几种。

1.Booch方法

动态逻辑模型描述对象之间的互相作用。互相作用通过一组协同的对象、对象之间消息的有序的序列、参与对象的可见性定义，来定义系统运行时的行为。Booch方法中的对象交互作用图被用来描述重要的互相作用，显示参与的对象和对象之间按时间排序的消息。可见性图用来描述互相作用中对象的可见性。对象的可见性定义了一个对象如何处于向它发送消息的方法的作用域之中。例

如，它可以是方法的参数、局部变量、新的对象，或当前执行方法的对象的部分。静态物理模型通过模块描述代码的布局。动态物理模型描述软件的进程和线程体系结构。

Booch方法的过程包括以下步骤：

在给定的抽象层次上识别类和对象。

识别这些对象和类的语义。

识别这些类和对象之间的关系。

实现类和对象。

这4种活动不仅仅是一个简单的步骤序列，而是对系统的逻辑和物理视图不断细化的迭代和渐增的开发过程。

类和对象的识别包括找出问题空间中关键的抽象和产生动态行为的重要机制。开发人员可以通过研究问题域的术语发现关键的抽象。语义的识别主要是建立前一阶段识别出的类和对象的含义。开发人员确定类的行为（即方法）和类及对象之间的互相作用（即行为的规范描述）。该阶段利用状态转移图描述对象的状态的模型，利用时态图（系统中的时态约束）和对象图（对象之间的互相作用）描述行为模型。

在关系识别阶段描述静态和动态关系模型。这些关系包括使用、实例化、继承、关联和聚集等。类和对象之间的可见性也在此时确定。

在类和对象的实现阶段要考虑如何用选定的编程语言实现，如何将类和对象组织成模块。

在面向对象的设计方法中，Booch强调基于类和对象的系统逻辑视图与基于模块和进程的系统物理视图之间的区别。他还区别了系统的静态和动态模型。然而，他的方法偏向于系统的静态描述，对动态描述支持较少。

Booch方法的力量在于其丰富的符号体系，包括：

类图（类结构-静态视图）

对象图（对象结构-静态视图）

状态转移图（类结构-动态视图）

时态图（对象结构-动态视图）

模块图（模块体系结构）

进程图（进程体系结构）

用于类和对象建模的符号体系使用注释和不同的图符（如不同的箭头）表达详细的信息。Booch建议在设计的初期可以用符号体系的一个子集，随后不断添加细节。对每一个符号体系还有一个文本的形式，由每一个主要结构的描述模板组成。符号体系由大量的图符定义，但是，其语法和语义并没有严格地被定义。

2.Rumbaugh的OMT方法

Rumbaugh的OMT方法从3个视角描述系统，相应地提供了3种模型：对象模型，动态模型和功能模型。

对象模型描述对象的静态结构和它们之间的关系。主要的概念包括：

类

属性

操作

继承

关联（即关系）

聚集

动态模型描述系统随时间变化的方面，其主要概念有：

状态

子状态和超状态

事件

行为

活动

功能模型描述系统内部数据值的转换，其主要概念有：

加工

数据存储

数据流

控制流

角色（源/潭）

OMT方法将开发过程分为4个阶段：

分析：基于问题和用户需求的描述，建立现实世界的模型。

分析阶段的产物有：

问题描述

对象模型 = 对象图 + 数据词典

动态模型 = 状态图 + 全局事件流图

功能模型 = 数据流图 + 约束

系统设计：结合问题域的知识和目标系统的体系结构（求解域），将目标系统分解为子系统。

对象设计：基于分析模型和求解域中的体系结构等添加的实现细节，完成系统设计。

主要产物包括：

细化的对象模型

细化的动态模型

细化的功能模型

实现：将设计转换为特定的编程语言或硬件，同时保持可追踪性、灵活性和可扩展性。

3.Coad/Yourdon方法

Coad/Yourdon方法严格区分了面向对象分析（OOA）和面向对象设计（OOD）。该方法利用5个层次和活动定义并记录系统行为、输入和输出。这5个层次的活动包括：

发现类及对象。描述如何发现类及对象。从应用领域开始识别类及对象，形成整个应用的基础，然后，据此分析系统的责任。

识别结构。该阶段分为两个步骤。第一，识别一般-特殊结构，该结构捕获了识别出的类的层次结构；第二，识别整体-部分结构，该结构用来表示一个对象如何成为另一个对象的一部分，以及多个对象如何组装成更大的对象。

定义主题。主题由一组类及对象组成，用于将类及对象模型划分为更大的单位，便于理解。

定义属性。其中包括定义类的实例（对象）之间的实例连接。

定义服务。其中包括定义对象之间的消息连接。

在面向对象分析阶段，经过5个层次的活动后的结果是一个分成5个层次的问题域模型，包括主题、类及对象、结构、属性和服务5个层次，由类及对象图表示。5个层次活动的顺序并不重要。

面向对象设计模型需要进一步区分以下4个部分。

1）问题域的设计

问题域部分的设计是任何OOD方法都必须完成的工作，它主要是对OOA结果进行改进和精化，并将其由问题域转化到解域。具体来说，有以下几个方面。

属性：有些属性在分析阶段有助于问题的理解，而到了设计阶段则可以由其他属性导出或根本没必要保留。因此，应去掉它们。相反地，为了实现服务算法还需要增加相应的一些属性。

服务：OOA只给出了服务的接口，其具体实现算法要在OOD阶段完成。

类及对象：在OOA阶段有助于问题理解的一些类在OOD阶段成为冗余，需要删除，而为了优化调整继承关系还要增加一些类。所有的类都确定以后还要明确哪些类的对象会引发哪些类创建新对象。

结构：对类间结构进行优化调整。

对象行为：明确对象间消息传递的实现算法，依据动态模型确定对象间消息发送的先后顺序，并设计相应算法，协调对象的行为。

2）人机交互界面的设计

有些设计方法并没有提到交互界面的设计，一方面是因为这些系统中交互界面不十分重要；另一方面是因为这部分的设计很有规律，设计方法也比较成熟，但为完整起见，仍将其列出。主要工作包括：

交互界面子系统的设计：与界面有关的类及类间结构的设计，以及有关算法的设计。

交互界面子系统和应用之间接口的设计。

3）应用控制的设计

这部分对象主要完成应用的驱动工作。这部分对象不同于从现实世界中抽象出来的对象，在现实世界和问题域中没有原型，它们同界面子系统中的对象及问题对象发生作用，控制系统的运行。

4）与问题领域有关的设计

一些系统具有与应用领域有关的特点，如多任务、分布式计算等，该项工作主要是针对这些特点而相应设计的。

4.Jacobson方法

Jacobson方法与上述3种方法有所不同，它涉及到整个软件生命周期，包括需求分析、设计、实现和测试等4个阶段。需求分析和设计密切相关。需求分析阶段的活动包括定义潜在的角色（角色指使用系统的人和与系统互相作用的软、硬件环境），识别问题域中的对象和关系，基于需求规范说明和角色的需要发现use case,详细描述use case.设计阶段包括两个主要活动，从需求分析模型中发现设计对象，以及针对实现环境调整设计模型。第一个活动包括从use case的描述发现设计对象，并描述对象的属性、行为和关联。在这里还要把use case的行为分派给对象。

在需求分析阶段的识别领域对象和关系的活动中，开发人员识别类、属性和关系。关系包括继承、熟悉（关联）、组成（聚集）和通信关联。定义use case的活动和识别设计对象的活动共同完成行为的描述。Jacobson方法还将对象区分为语义对象（领域对象）、界面对象（如用户界面对象）和控制对象（处理界面对象和领域对象之间的控制）。

在该方法中的一个关键概念就是use case.use case是指行为相关的事务（transaction）序列，

该序列将由用户在与系统的对话中执行。因此，每一个use case就是一个使用系统的方式。用户给定一个输入，就执行一个use case的实例并引发执行属于该use case的一个事务。基于这种系统视图，Jacobson将use case模型与其他5种系统模型关联：

领域对象模型：use case模型根据领域来表示。

分析模型：use case模型通过分析来构造。

设计模型：use case模型通过设计来具体化。

实现模型：该模型依据具体化的设计来实现use case模型。

测试模型：用来测试具体化的use case模型。

OOD详细设计阶段应该注意属性、消息和服务的如下特性和问题：

是类属性（所有同类对象的共同特征），还是实例属性？

该属性对实例是常数性的还是参数性的？在物资类的一个实例中，编码等是常数，而仓位、储量等是参数。常数在实例构造时确定，而参数在实例构造时最多可以有默认值，在系统运行时可以被改变。

服务是主动还是被动的？若业务规则决定对所有预定代码的物资必须进行质检，则自动请求质检服务就也可能是主动服务。而系统如果要求质检请求经过库房管理用户发出，则物资类的"请求质检"服务就是被动服务了。

是对内还是对外服务？对内服务是对实例本身的属性的操作，对外服务是对其他实例或系统发出消息或操作其他类实例的属性。

消息的分析对OOA的企业经营模型有特别意义，因为消息是对象间动态关系的要素，也是信息隐蔽的重要手段。在顺序系统中，消息是向其他对象或系统发出的服务请求，在并行系统中消息表述为对象之间在一次交互中所传递的信息。在C/S结构的系统中，客户端向服务器请求数据属于并行的线程之间的消息，而在单机数据库系统中的用户请求数据，则一般是顺序执行的线程内的消息。接受消息的对象是什么？请求的是什么服务？消息发送者是否要与接收者同步？消息是定向的还是广播的？接收消息者如何处理消息？发送者如何对待消息处理的结果？发出消息的条件是什么？消息是在线程之间还是线程内部传递？

属性、服务的性质是公共的、保护的还是私有的？这些特性的设置对封装性能影响很大，应该考虑全面和一致。应用面向对象的分析、设计方法时还要求考虑类、对象之间的关系和联系。类和对象间有：整体-部分关系，一般-特殊关系，以及实例连接、消息连接等联系。

多元连接使系统分析和设计复杂。可以采用增加类的方法解决。例如"出版计划-编辑-图书"这样的三元关联的语意为某编辑在某出版计划中负责某图书，增加图书项目组成员类。

扩充的OOA模型包括use case 和交互图，它们是在类图基本完成的基础上建立的。对企业信息系统应用而言，use case就是一个用户或一个活动者与系统进行交互的一个过程的描述，是一个用类似自然语言的语言或伪代码表述的语序，近似于功能模块描述，包含与系统责任有关的企业经营规则描述的信息化版本。use case所指的一个交互过程应该是原子过程。而活动者就是指系统外与系统交互作用的事物，活动者可能不是类图中的类和对象。在出版社系统中，"编辑"作为人员不是待实现的对象，而是要和系统进行交互的一种事物。总之，活动者是要与系统交互的事物，而其本身不是系统成分，即使可能有与活动者同名的类在类图中。

附于类图的一个重要文档是详细说明，它包含了属性、服务、消息、联系等成分的准确语意。

OOA/D建立企业经营模型只是OOSE的初始环节。当上述文档完成后，还要进行物理设计、原

型设计、获取反馈、修改设计等。轻视分析、设计和建立文档，偏重代码和实现，是我国企业信息技术人员的共同问题。而要让企业信息化可持续地发展，必须重视分析、设计及建立文档。

　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

用户界面设计

用户界面就如同人的外表，最容易让人一见钟情或一见恶心。像人类追求心灵美和外表美那样，软件系统也追求（内在的）功能强大和（外表的）界面友好。但随着生活节奏的加快，人们已少有兴趣去品味深藏不露的内在美。如果把UNIX系统比做是健壮的汉子和妇人，那么Windows系统就像妩媚的小白脸和狐狸精。想不到Windows系统竟然能兴风作浪，占去大半市场。有鉴于此，我们应该鼓励女士多买化妆品（男士付钱）以获得更好的界面。

美的界面能消除用户由感觉引起的乏味、紧张和疲劳（情绪低落），大大提高用户的工作效率，从而进一步为发挥用户技能和为用户完成任务做出贡献。从人机界面发展历史与趋势上可以看出人们对界面美的需求，以及美在界面设计中的导向作用。

界面设计已经经历了两个界限分明的时代。第一代是以文本为基础的简单交互，如常见的命令行、字符菜单等。由于第一代界面考虑人的因素太少，用户兴趣不高。随着技术的发展，出现了第二代直接操纵的界面。它大量使用图形、语音和其他交互媒介，充分地考虑了人对美的需求。直接操纵的界面使用视听、触摸等技术，让人可以凭借生活常识、经历和推理来操纵软件，愉快地完成任务。更高层次的界面甚至模拟了人的生活空间，例如虚拟现实环境。

界面的美充分体现了人机交互作用中人的特性与意图，越来越多的用户将通过具有吸引力而令人愉快的人机界面与计算机打交道。

一个好的用户界面应具有以下特点。

1.可使用性

使用的简单性。

用户界面中所用术语的标准化和一致性。

拥有HELP（帮助）功能。

快速的系统响应和低的系统成本。

用户界面应具有容错能力。

2.灵活性

考虑用户的特点、能力和知识水平，应当使用户界面能够满足不同用户的要求。

用户可以根据需要制定和修改界面方式。

系统能够满足用户的希望和需要。

与其他软件系统交互应有标准的界面。

3.复杂性和可靠性

用户界面的规模和组织的复杂程度就是界面的复杂性。

用户界面的可靠性是指无故障使用的间隔时间。

　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

设计评审

在开发时期的每个阶段，特别是设计阶段结束时都要进行严格的技术评审，尽量不让错误传播到下一个阶段。设计评审一般采用评审会议的形式来进行。

软件设计评审流程如图17-16所示。

设计负责人的职责：一般工程设计均由软件公司选派设计负责人，设计负责人承担该项工程的全部设计管理任务，对设计质量、进度及各单项设计间的组织协调等全面负责，对各单项工程之间的衔接、协调和总体方案质量负主要责任，并负责编写总说明，汇编总概算。

高级管理人员的职责：确定主审员、审批评审记录。

主审员的职责：在评审会前提出项目的书面评审意见、确定评审组、确定评审结果并填写评审记录。

评审组的职责：专业评审组评委表决通过项目初评结论并报综合评审会议；通过设计报告。

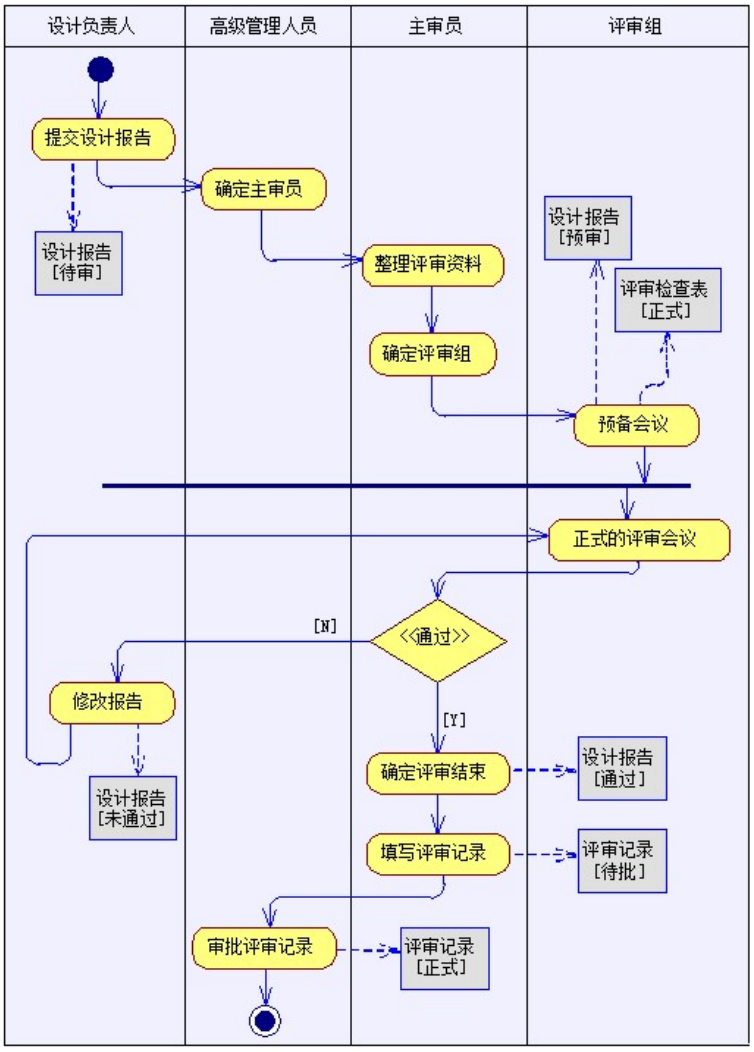

图17-16　软件设计评审流程图