北京航空航天大學
BEIHANG UNIVERSITY

Systems Design

王宝会　北航软件学院

2020年12月

System Design

软件设计最终目标是要取得最佳方案。
最佳：在所有候选方案中，就节省开发费用，降低资源消耗，缩短开发时间的条件，选择能够赢得较高的生产率、较高的可靠性和可维护性的方案。
设计过程中，各个时期的设计结果需要经过一系列的设计质量的评审，以便及时发现和及时解决在软件设计中出现的问题，防止把问题遗留到开发的后期阶段，造成后患。
软件设计分两个阶段
(1)概要设计(总体设计)
确定软件的结构以及各组成成分
(子系统或模块)之间的相互关系。
(2)详细设计
确定模块内部的算法和数据结构，产生描述各模块程序过程的详细文档。
利用模块结构减少开发和维护软件的费用

软件概要设计

把已确定了的各项需求转换成一个相应的体系结构，以结构设计和数据设计开始，建立程序的模块结构，定义接口并建立数据结构。此外要使用一些设计准则来判断软件的质量。
总体架构设计一般采用分层设计，分几层，B/S或者C/S
有没有用到构件
组件是怎么定义的、接口是怎么连接的？组件构件开发技术，
系统功能模块设计（系统总体包括几个模块，画出主功能模块结构图）,其中以工作流为主要支撑模板，以酒店预定系统、机票预定系统为主要功能模板，其他动态帮助等为辅助功能模块。
模块之间连接关系（是隶属关系、还是调用关系等）
模块接口，接口如何定义（源代码）
设计如何对应需求（回溯性）

软件详细设计

考虑设计每一个模块部件的过程描述，对每个模块要完成的工作进行具体的描述。
编写设计说明书，提交评审。
包括各个业务模块接口设计、类的设计以及数据库的详细设计。
数据可以分为结构化数据和非结构化数据

软件设计说明书

概要设计说明书：说明对程序系统的设计考虑，包括程序系统的基本处理流程、程序系统的组织结构、模块划分、功能分配、接口设计、 运行设计、数据结构设计和出错处理设计等，为程序的详细设计提供基础；

详细设计说明书：说明一个软件系统各个层次中的每一个程序 （每个模块或子程序）的设计考虑，如果一个软件系统比较简单，层次很少，本文件可以不单独编写，有关内容合并入概要设计说明书；

数据库设计说明书：是对于设计中的数据库的所有标识、逻辑结构和物理结构做出具体的设计规定。

衡量软件设计的技术标准

衡量软件系统质量的标准

(1)软件系统开始变坏的表现：硬化、脆弱、绑死、胶着

(2)软件系统关键的质量特性：正确性、健壮性、可扩展性、可复用性、兼容、可移植性、高效性、timeliness、economy and functionality

衡量软件系统设计的技术标准

(1)设计必须实现分析模型中描述的所有显式需求，必须满足用户希望的所有隐式需求。

(2)设计必须是可读、可理解的，使得将来易于编程、易于测试、易于维护。

(3)设计应从实现角度出发，给出与数据、功能、行为相关的软件全貌。

McGlanghlin给出在将需求转换为设计时判断设计好坏的三条特征

① 设计出来的结构应是分层结构，从而建立软件成份之间的控制。
② 设计应当模块化，从逻辑上将软件划分为完成特定功能或子功能的构件。
③ 设计应当既包含数据抽象，也包含过程抽象。
④ 设计应当建立具有具有独立功能特征的模块。
⑤ 设计应当建立能够降低模块与外部环境之间复杂连接的接口。
⑥ 设计应能根据软件需求分析获取的信息，建立可驱动可重复的方法。

More……

软件架构设计方法

(1)什么是架构, 它和系统是如何关联的
(2)如何获得可维护性、可扩展性、可重用性、互操作性等
(3)在系统中如何组织组件
(4)如何组织组件的内部
(5)如何保持平台相关的细节和应用的分离
(6)如何应用封装(encapsulation)、抽象(abstraction)和委派(delegation)的原则
(7)如何应用设计模式来实现好的架构

软件架构设计

表现层框架设计
(1)使用MVC模式设计表现层
(2)使用XML设计表现层
(3)表现层UIP设计
(4)表现层动态生成设计
(5)表现层模块通讯设计

中间层架构设计
(1)业务逻辑层组件设计
(2)业务逻辑层工作流设计
(3)服务界面设计
(4)业务逻辑层实体设计
(6)业务逻辑层框架
(5)Web Service应用场景

数据访问层设计（持久层架构设计）
(1)5种数据访问模式（在线访问，DataAccessObject， DataTransferObjec，离线数据模式，对象/关系映射）
(2)数据访问层组件设计
(3)工厂模式在数据访问层应用
(4)ORM、Hibernate与CMP设计
(5) 运用Xml Schema
(6)事务处理设计
(7)连接对象管理设计

XML设计、数据架构规划与数据库设计
(1)数据库设计与类的设计融合
(2)数据库设计与XML设计融合
(3)数据库性能规划
(4)数据库封装设计

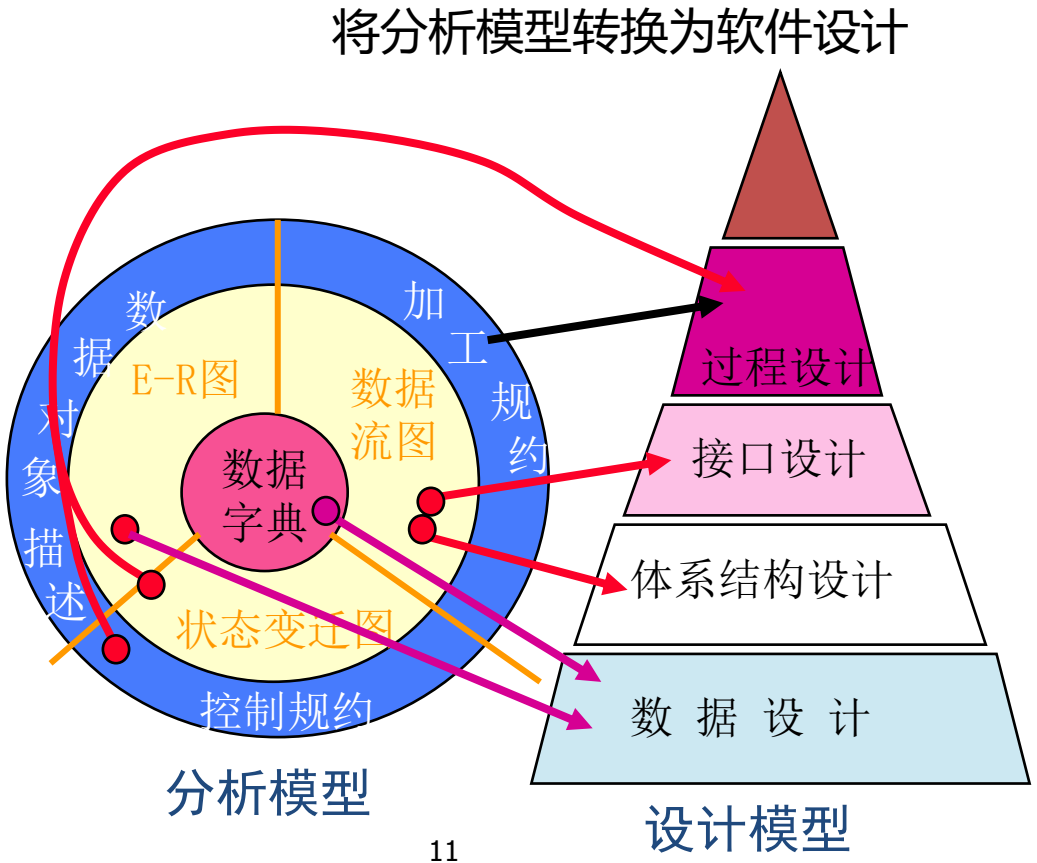
企业集成框架设计
(1)集成解决方案、数据集成、应用（接口）集成及应用服务
(2)EAI参考模型：业务模式、概念模式、逻辑模式、物理模式和实现模式
(3)企业应用系统集成设计

软件设计模式

- **分析创建型模式：**单例模（Singleton）、抽象工厂模式(Abstractfactory)
- **分析结构型模式：** 桥梁模式（Bridge）、装饰模式（Decorator）、适配器模式（Adapter）、代理模式（Proxy）、合成模式（Composite）
- **分析行为型模式：**命令模式（Command ）、观察者模式（Observer）、状态模式（State）、策略模式（Strategy）、模板方法模（Template Method）、访问者模（Template Method）

Structured Analysis and Design

软件设计


将分析模型转换为软件设计
数据对象描述
加工规约
E-R图
数据流图
数据字典
状态变迁图
控制规约
过程设计
接口设计
体系结构设计
数 据 设 计
分析模型
设计模型

Process Modeling过程建模

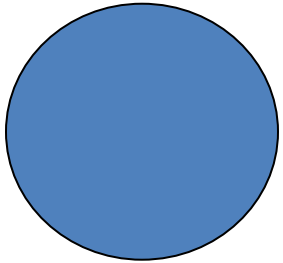
A technique for organizing and documenting the structure and flow of data through a system' s processes and/or the logic, policies, and procedures to be implemented by a system' s processes.

- 过程建模，对数据流进行系统性的管理和记录，或者是进行系统性实现逻辑、策略、过程的方式。

数据流图

Data Flow Diagram (DFD) - a process model used to depict the flow of data through a system and the work or processing performed by the system　一种被用于描述系统的数据流和系统的工作过程的建模方法

数据源点和终点 ——————— 文件

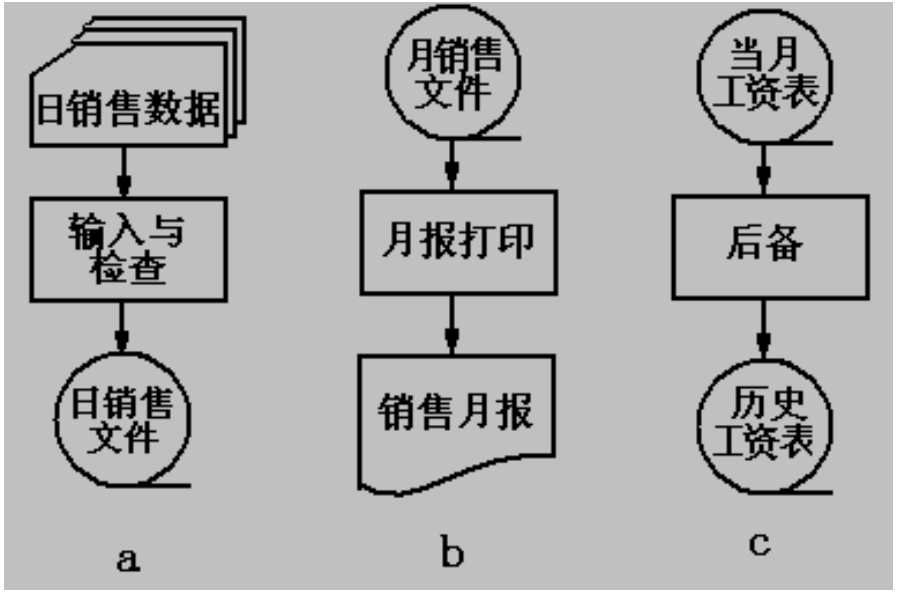


变换数据的加工 ——————→ 数据

逻辑关系符号：与、或、异或

系统流程图

系统流程图又叫事务流程图，是在计算机事务处理应用进行系统分析时常用的一种描述方法，它描述了计算机事务处理中从数据输入开始到获得输出为止，各个处理工序的逻辑过程。

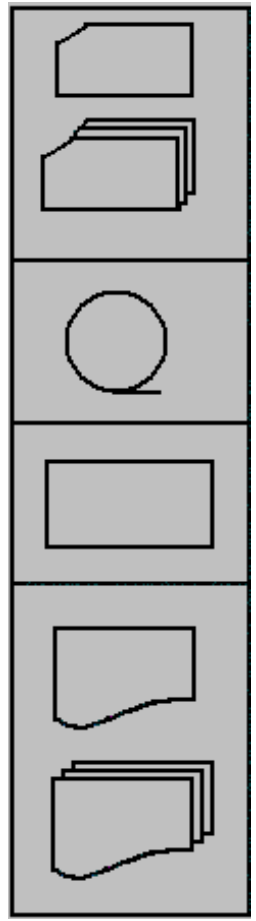


输入单据

磁盘文件

处理

输出单据



Process Modeling Toolbox 过程建模工具箱

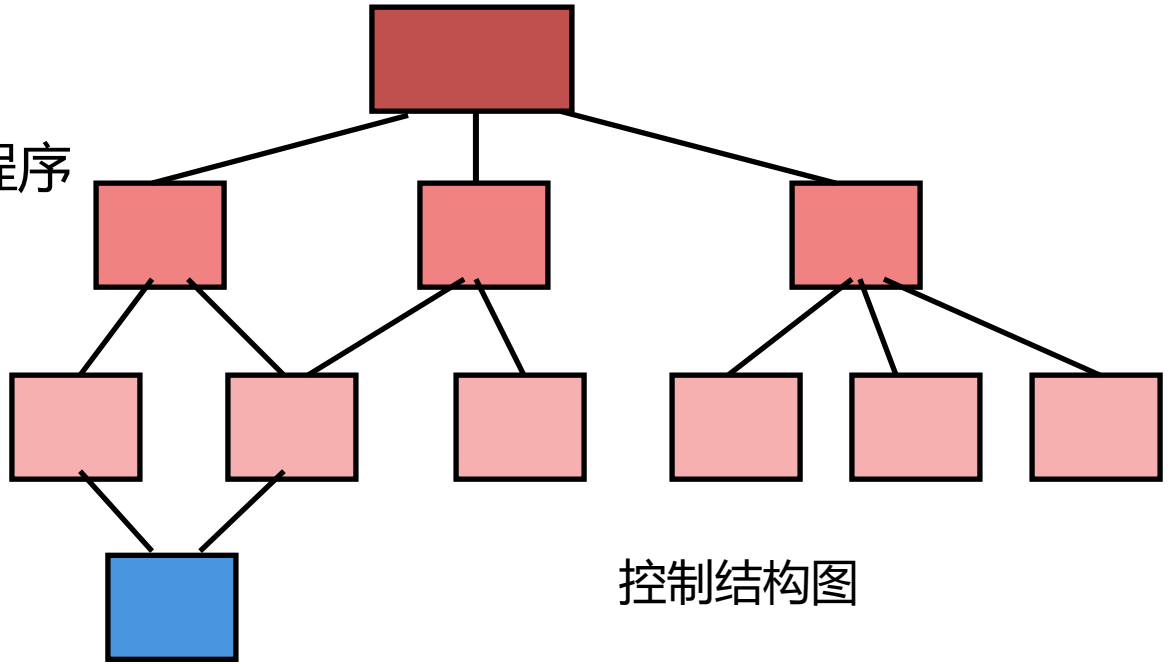
- [DFD] Context Data Flow Diagram 上下文数据流图
- Functional Decomposition Diagram 功能分解图
- Event-Response / Use Case List 事件反馈/用例表
- Event Decomposition Diagram 事件分解图
- [DFD] Event Diagram 事件图
- [DFD] System Diagram 系统图
- [DFD] Primitive Diagram 原始图

控制结构(程序结构)是软件模块间关系的表示

模块

- 模块是具有一定功能的可以用名词调用的程序语句集合，如：
  - 独立的汇编程序
  - COBOL的段和节
  - Pascal过程
  - FORTRAN的子程序
  - 汇编的宏

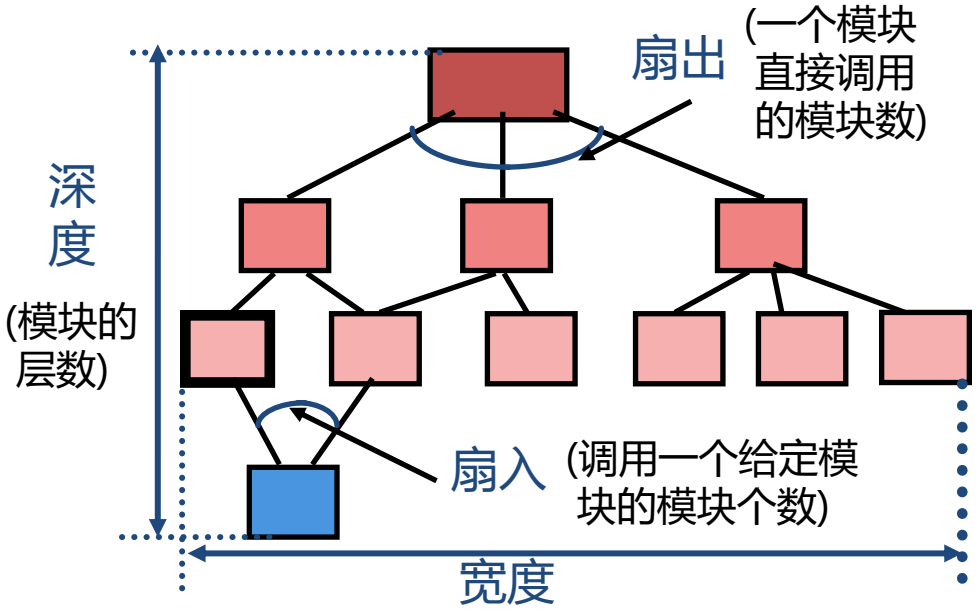
只有一个顶层(0层)模块
0层外任一模块都会在它的邻层存在一模块与它有关
同层模块间不发生联系



控制结构图

控制结构的层次规则

软件结构度量术语



模块化(Modularity)

模块化是好的软件设计的一个基本准则

高层模块 ——从整体上把握

问题，隐蔽细节

分解

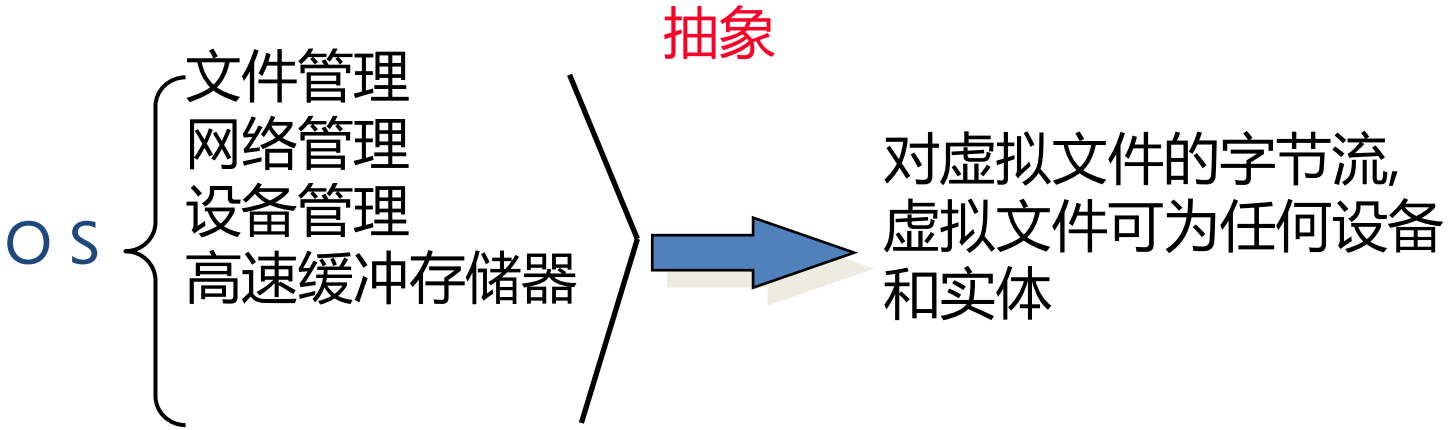
复杂问题  较小问题

分解

可减小解题所需的总的工作

抽象(Abstraction)

抽象原则应用举例
Windows NT一体化的I/O系统设计



" 各个击破 " 理论

例：将问题(P1+P2)分解为P1,P2

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**
对问题P1**和**P2**，如：**

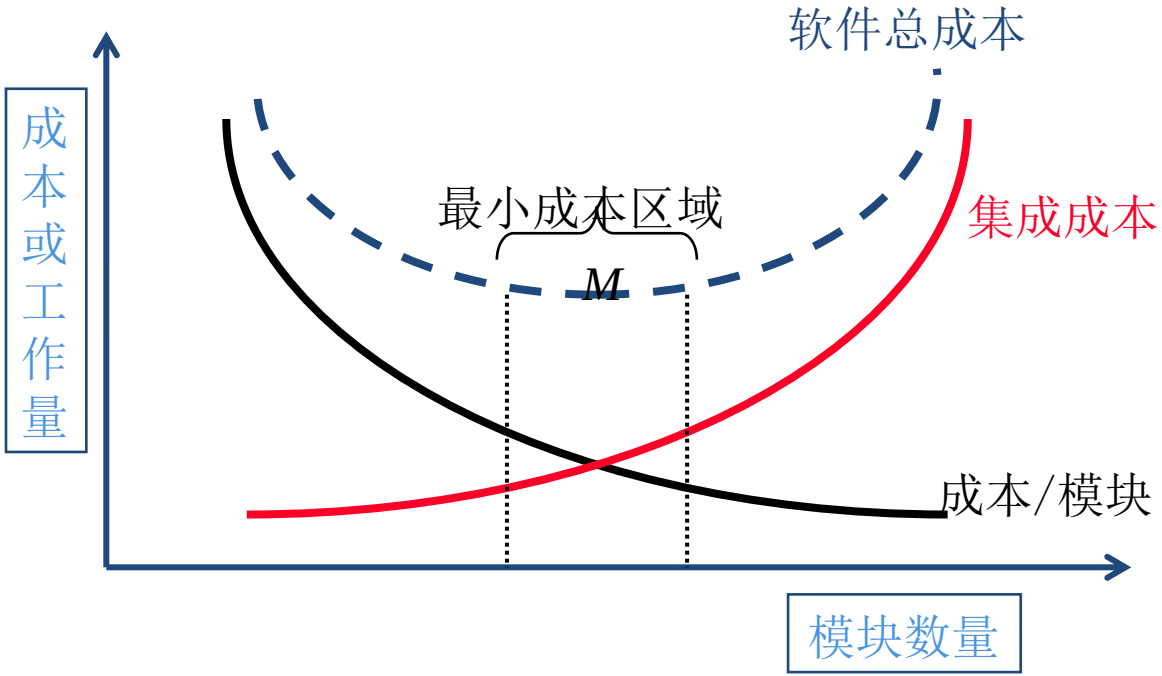
C(P1) > C(P2)

显然： E(P1) > E(P2)

有规律： C(P1+P2) > C(P1)+C(P2)

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

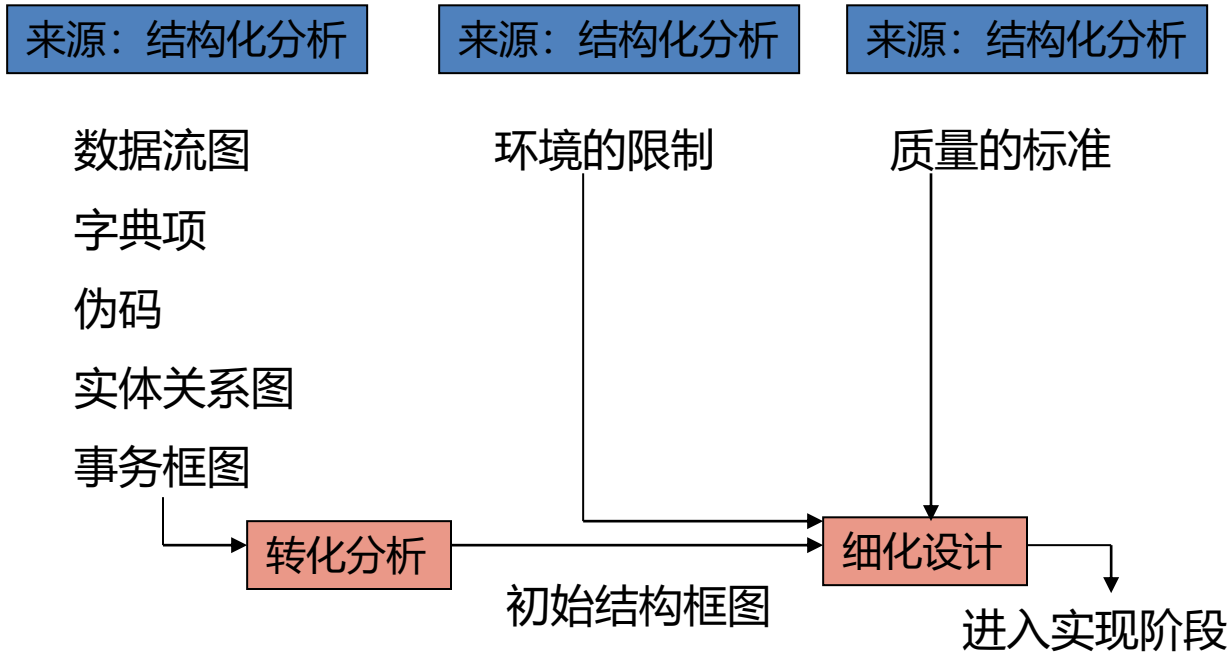
模块度



SA与SD的关系

结构化分析的结果	结构化设计的工具
数据流图	初始结构图
生存周期字典的数据部分	设计数据字典
伪码 实现方面	伪码
实体关系图	数据库设计
事务框图	分层、细化事务模型

SD来源于SA


来源：结构化分析
来源：结构化分析
来源：结构化分析
数据流图
字典项
伪码
实体关系图
事务框图
环境的限制
质量的标准
转化分析
细化设计
初始结构框图
进入实现阶段

软件概要设计

- 将系统划分成模块
- 决定每个模块的功能
- 决定模块的调用关系
- 决定模块的界面，即模块间传递的数据

结构化设计（SD方法）

- 相对独立、单一功能的模块
- 块间联系和块内联系
- 描述方法
- 步骤

结构图(SC Structure Chart)

结构图主要成分

模块：用方框表示，方框中写有模块的名字，一个模块的名字应适当地反映这个模块的功能，这就在某种程度上反映了块内联系；

调用：从一个模块指向另一个模块的箭头表示前一模块中含有对后一模块的调用；

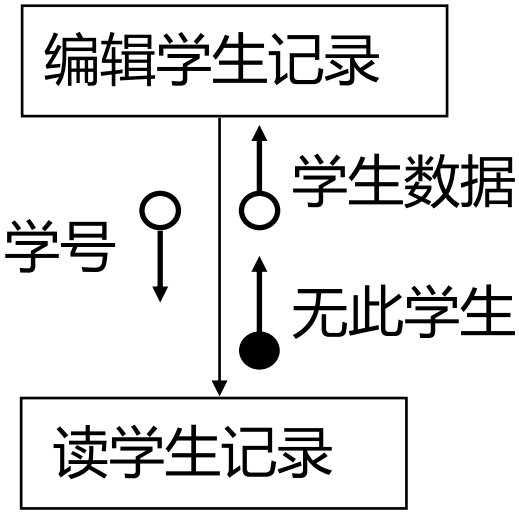
数据：调用箭头边上的小箭头表示调用时从一个模块传入送给另一个模块的数据，小箭头也指出了传送的方向。

SD方法在概要设计中的主要表达工具约定：

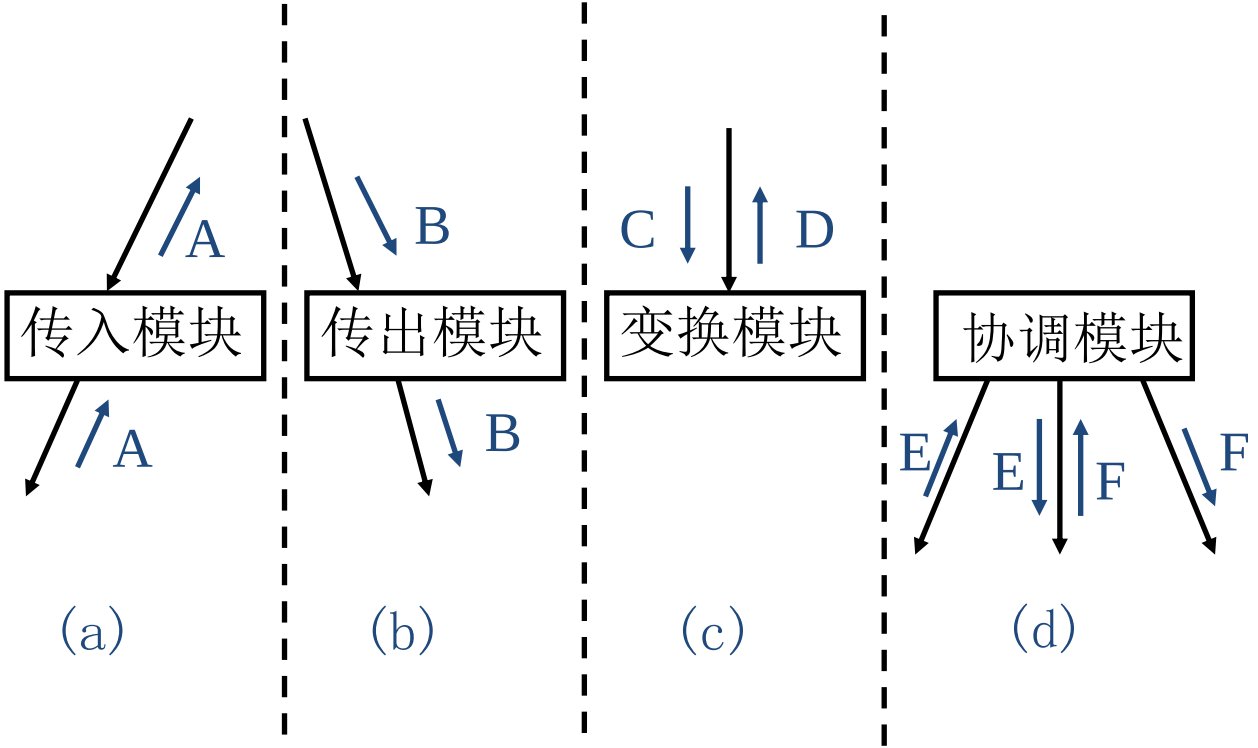
不加区分的数据

数据信息

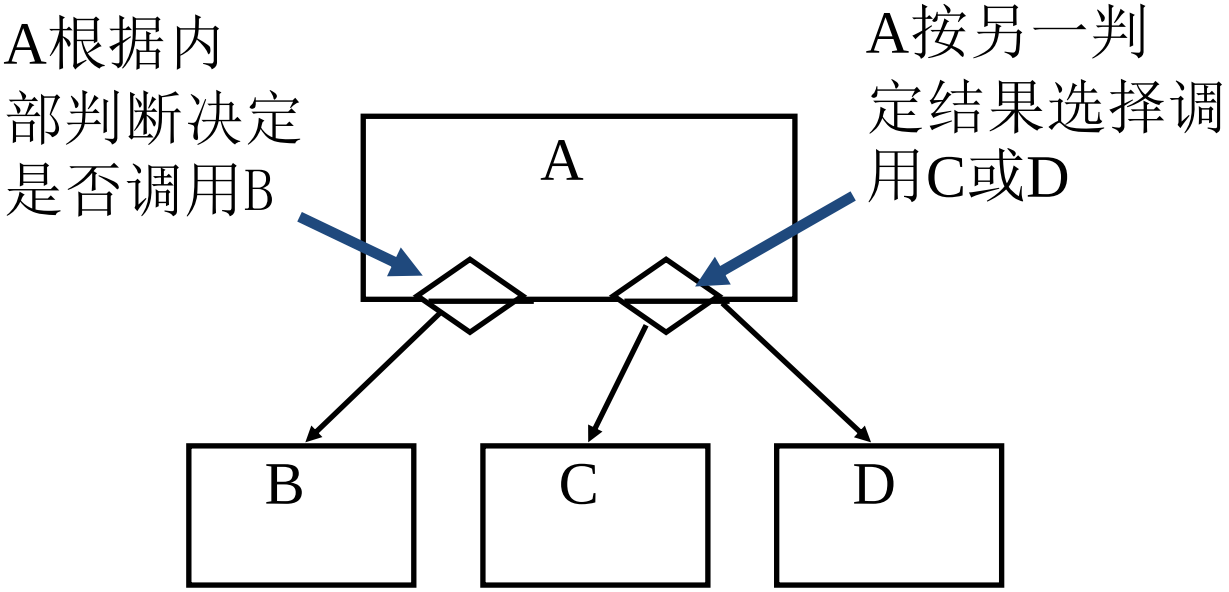
控制信息



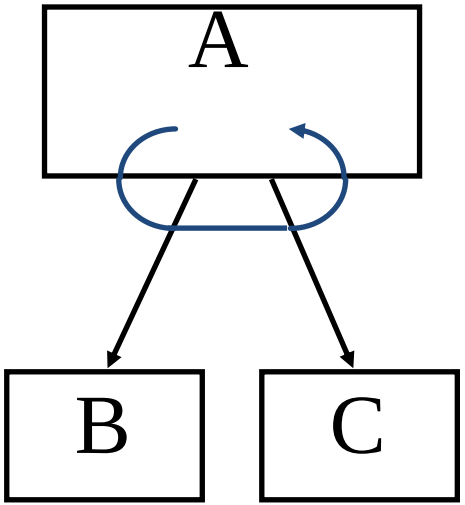
SC中的四种模块



SC中的选择调用

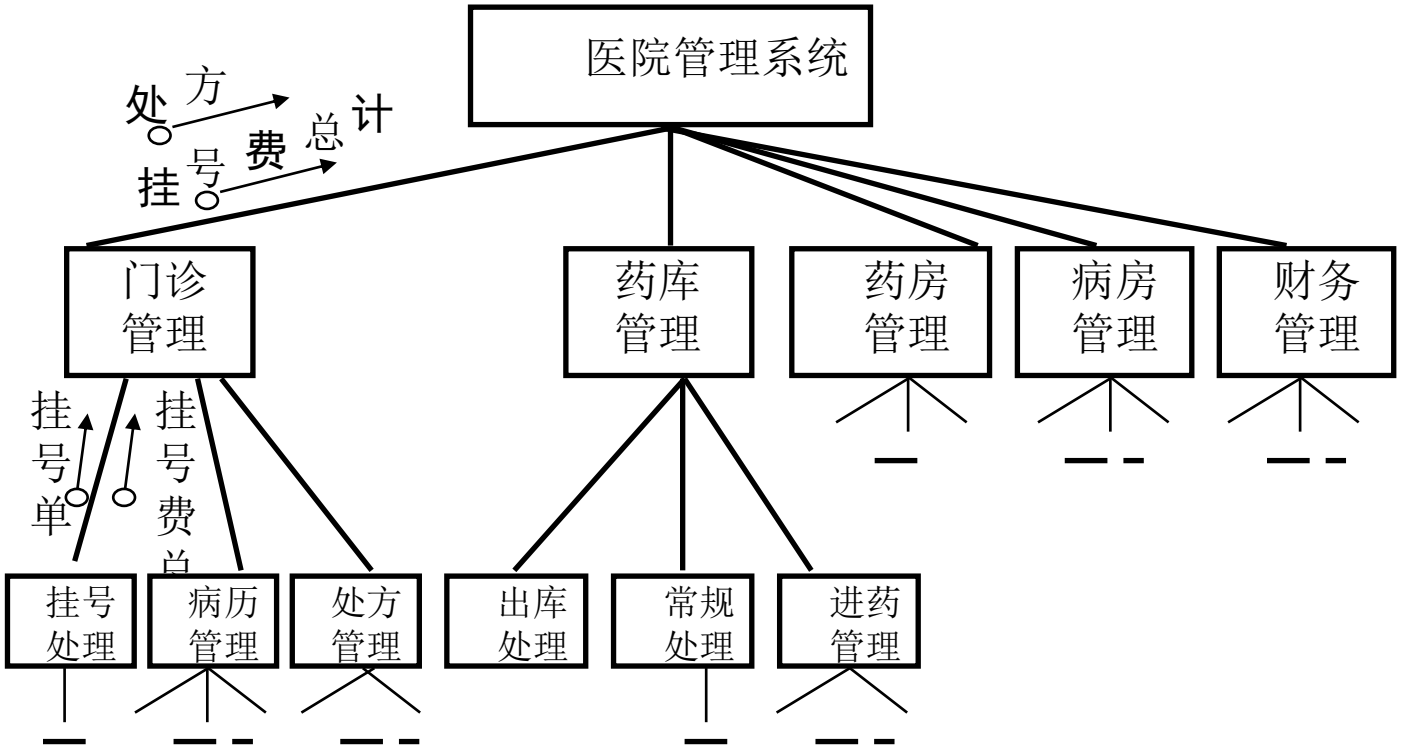


SC中的循环调用

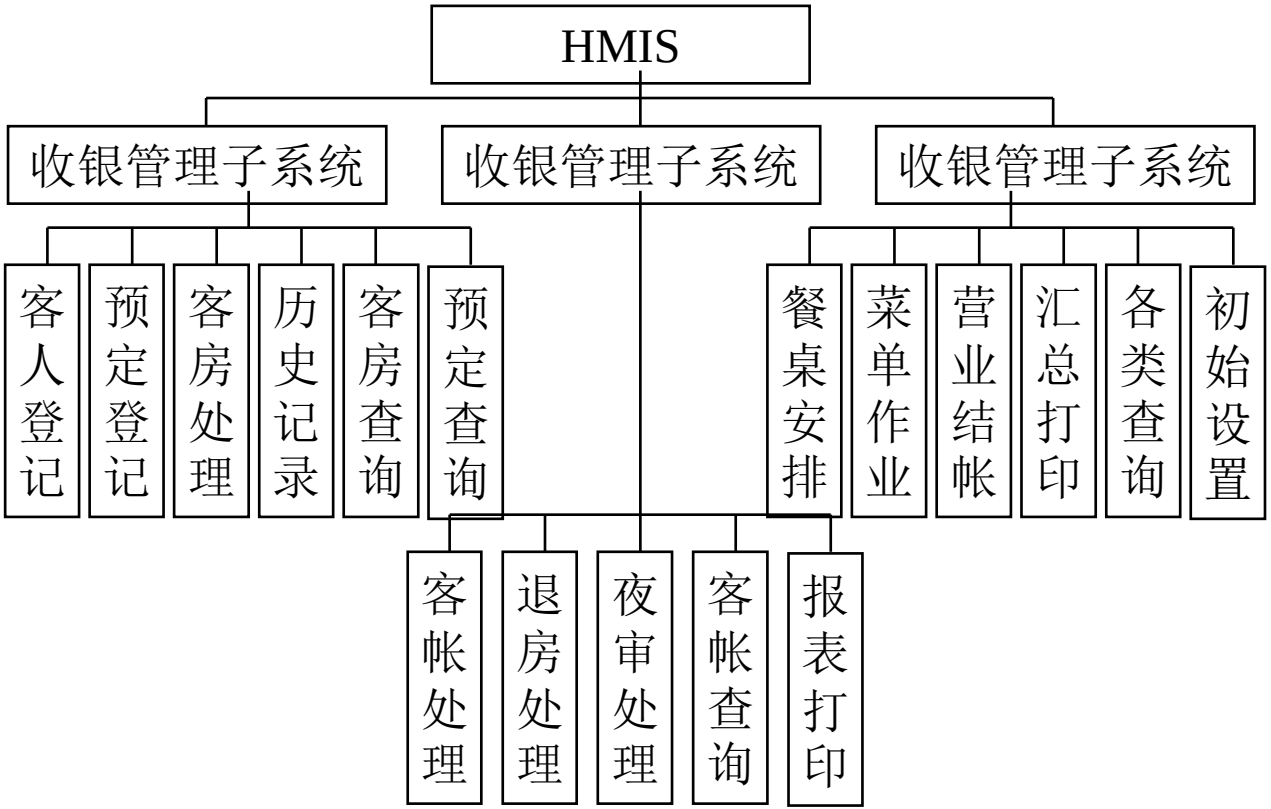


A根据内在的循环重复调用B、C等模块

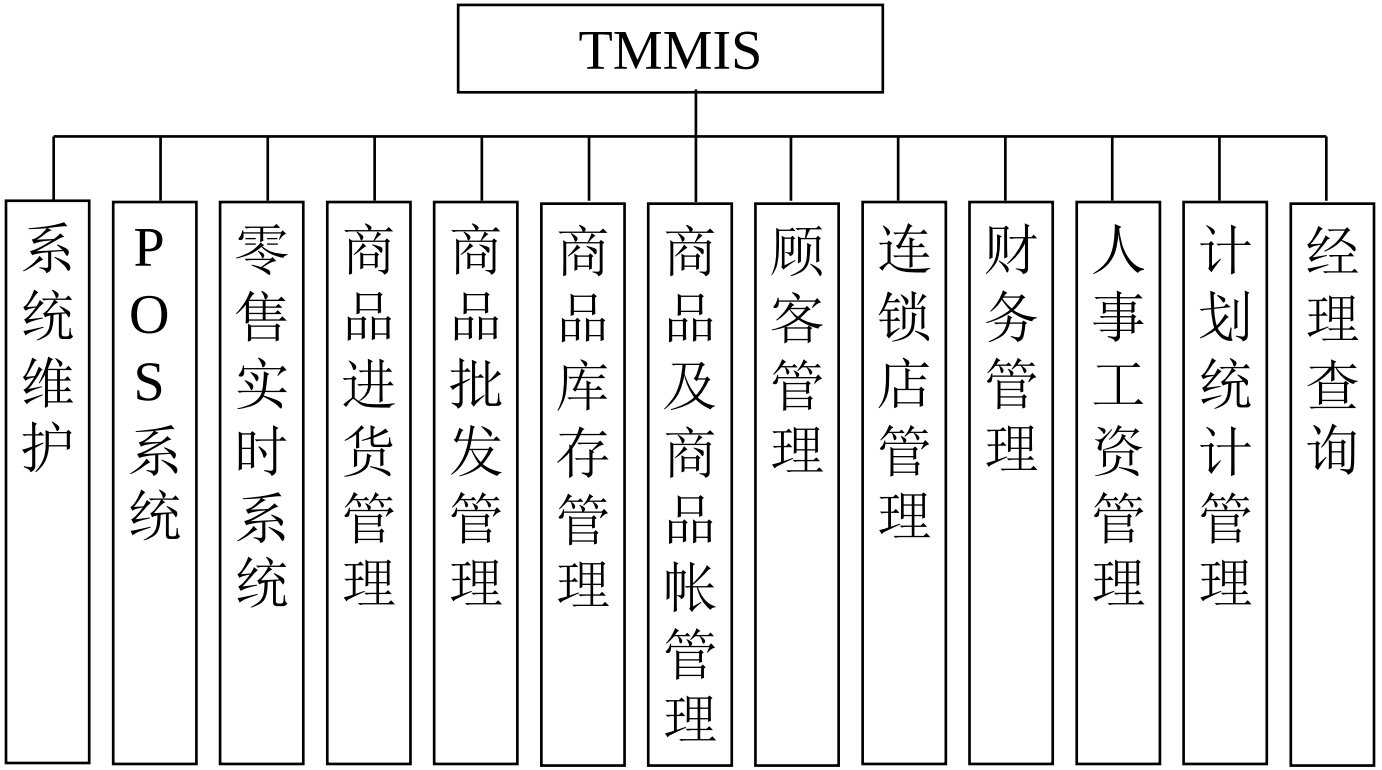
结构图(SC)举例


医院管理系统
处方
挂号费
总计
门诊管理
药库管理
药房管理
病房管理
财务管理
挂号单
挂号费
挂号处理
病历管理
处方管理
出库处理
常规处理
进药管理

酒店管理信息系统功能结构图



大型零售商场管理信息系统功能结构图



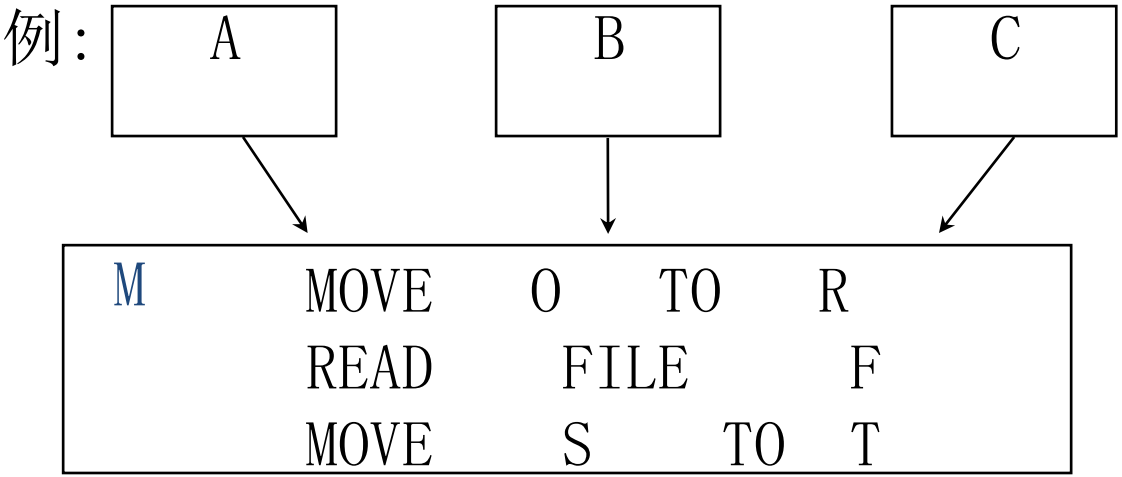
模块间联系·模块内联系

- 块间联系大小：
  - 方式、作用、数量
- 联系方式
  - 用过程语句调用、 直接引用
- 共用信息的作用
- 公用信息的数量

偶然型
逻辑型
瞬时型
通讯型
顺序型
功能型

信息隐蔽 (Information Hiding)：**模块所包含的信息，不允许其它不需要这些信息的模块访问，独立的模块间仅仅交换为完成系统功能而必须交换的信息。**

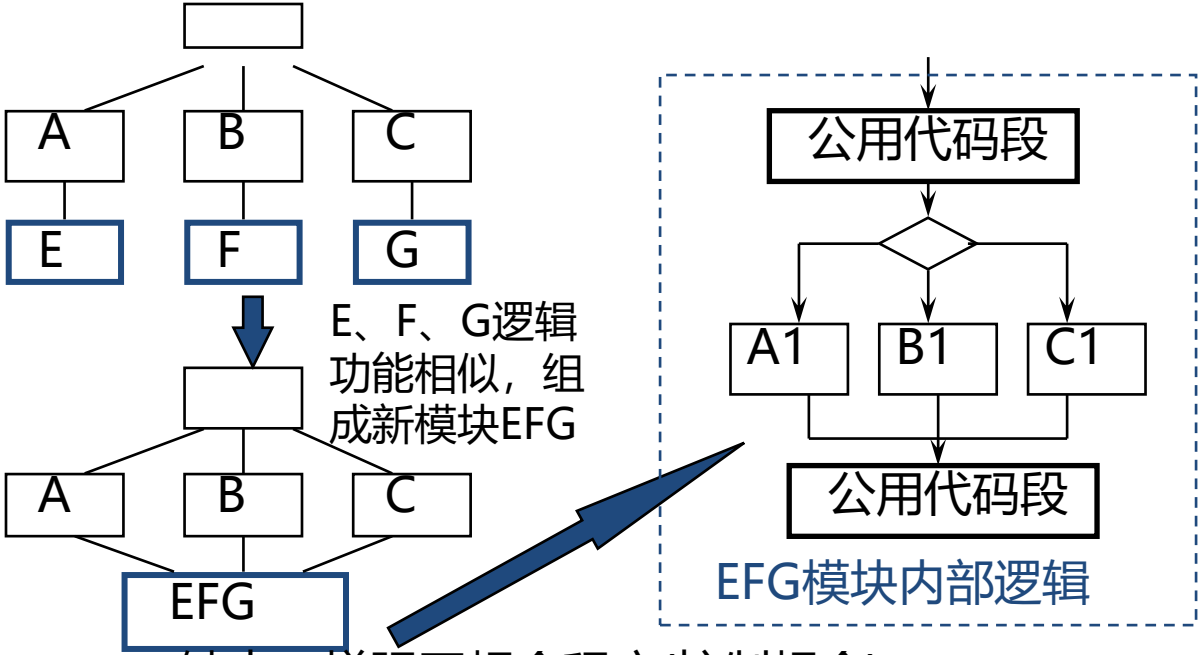
块内联系·偶然内聚(巧合内聚)



模块M中的三个语句没有任何联系
缺点：可理解性差，可修改性差

块内联系·逻辑内聚

- 把几种相关功能（逻辑上相似的功能）组合在一模块内，每次调用由传给模块的参数确定执行哪种功能。



缺点：增强了耦合程度(控制耦合)
不易修改，效率低

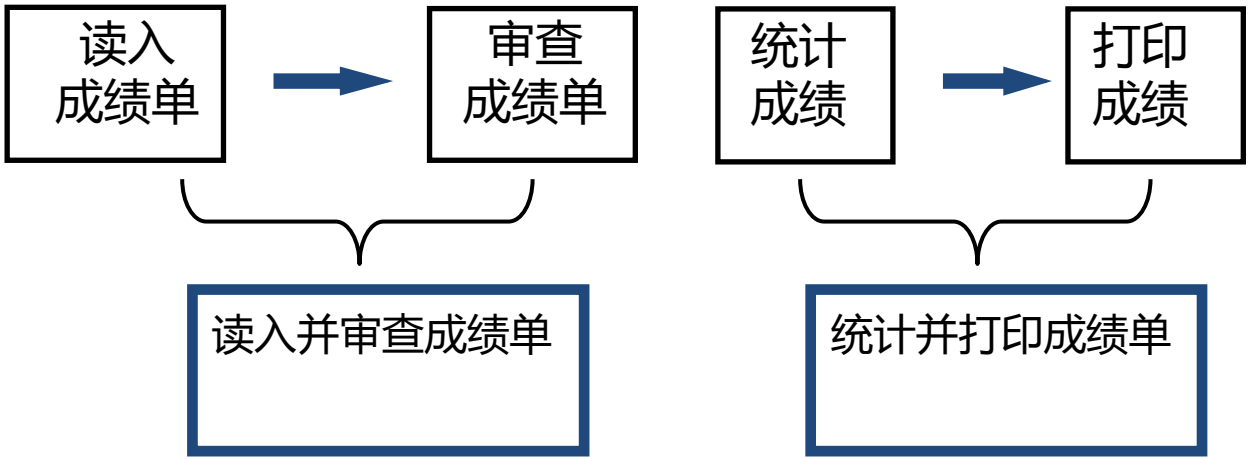
块内联系·时间内聚(经典内聚)

模块完成的功能必须在同一时间内执行，这些功能只因时间因素关联在一起。

例如：初始化系统模块、系统结束模块、紧急故障处理模块等均是时间性聚合模块

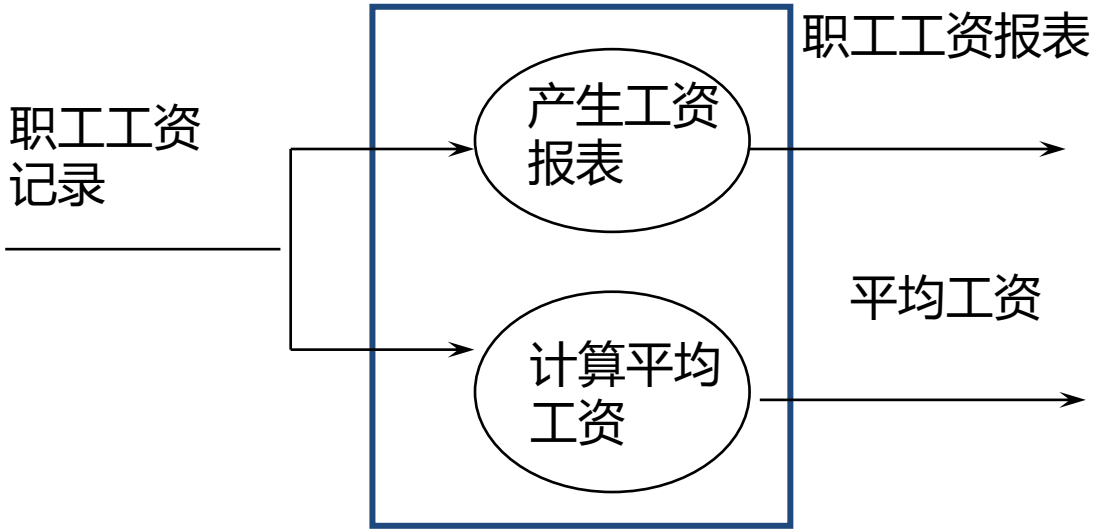
块内联系·过程内聚（顺序性组合）

模块内各处理成分相关，且必须以特定次序执行



块内联系·通信内聚

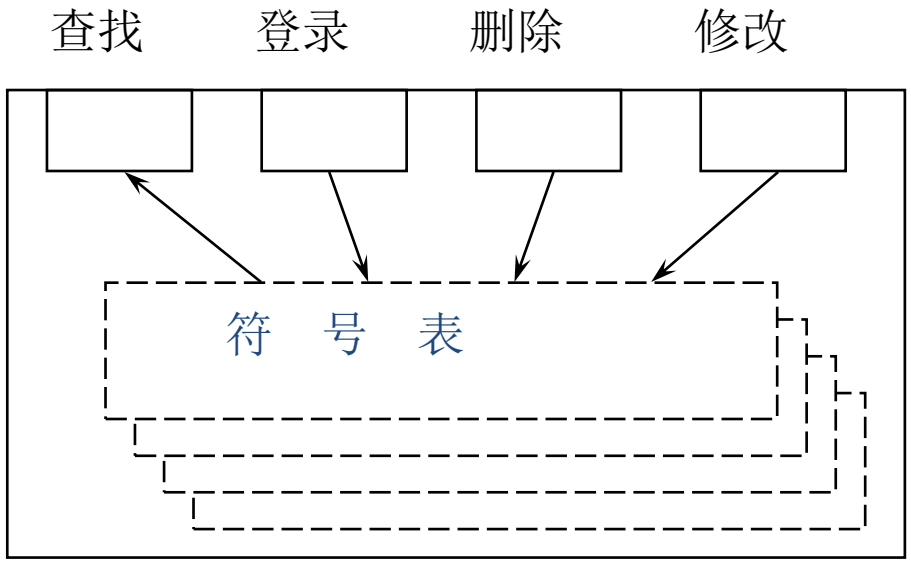
- 模块内各部分使用相同的输入数据，或产生相同的输出结果



产生职工工资报表并计算平均工资模块

块内联系·信息内聚

- 模块完成多个功能，各功能都在同一数据结构上操作，每一功能有唯一入口。



几个加工同时引用一个共同的数据

块内联系·功能内聚

- 模块仅包括为完成某个功能所必须的所有成分。
- 模块所有成分共同完成一个功能，缺一不可

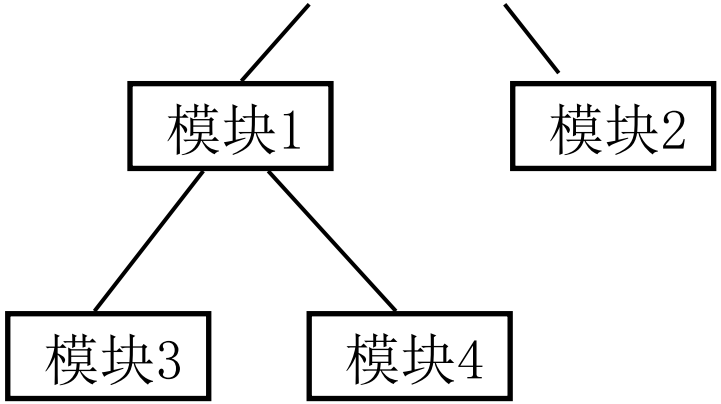
内聚性最强

块间联系·

- 无直接关系型
- 数据耦合
- 标记耦合
- 控制耦合
- 外部耦合
- 公共耦合
- 内容耦合

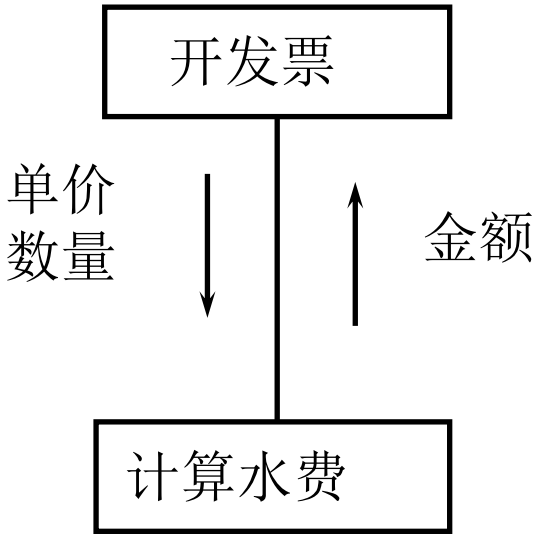
块间联系·无直接耦合

两个模块没有直接关系(模块1和模块2)，模块独立性最强。



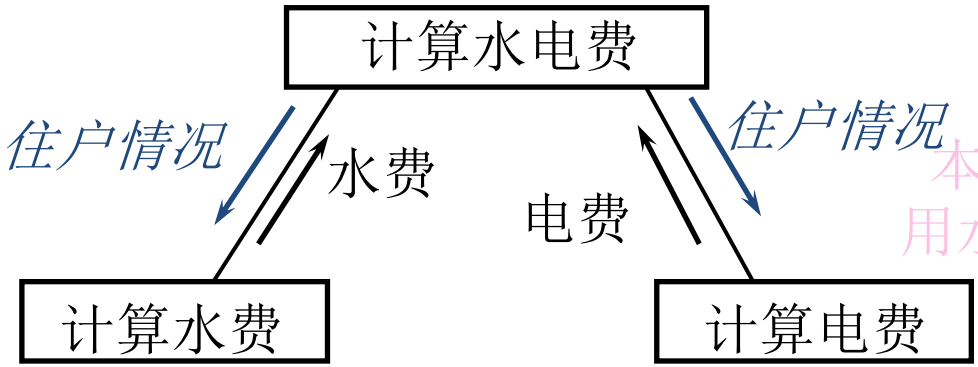
块间联系·数据耦合

一模块调用另一模块时，被调用模块的输入、输出都是简单的数据(若干参数)。属松散耦合。

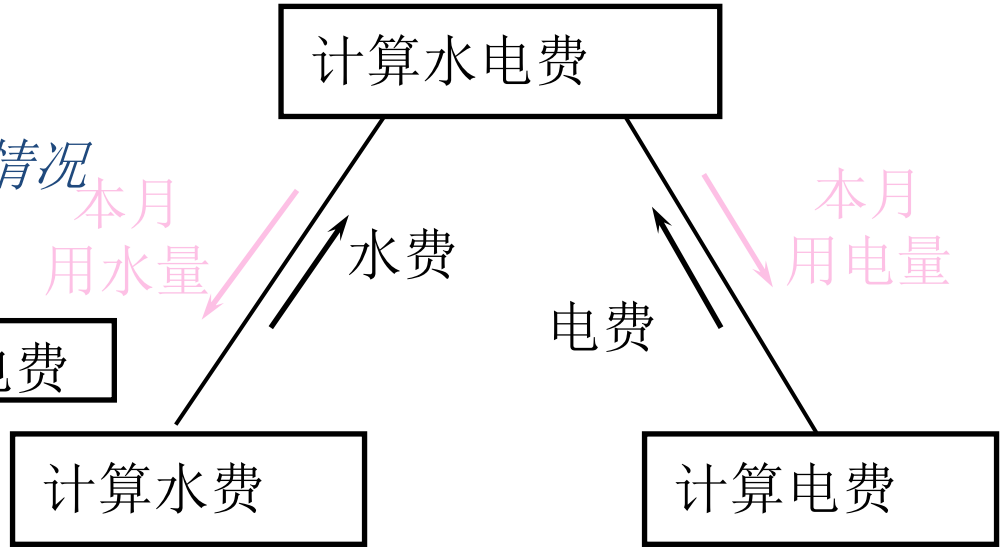


块间联系·标记耦合(复合型耦合)

如两个模块通过传递数据结构(不是简单数据，而是记录、数组等)加以联系，或都与一个数据结构有关系，则称这两个模块间存在标记偶合。



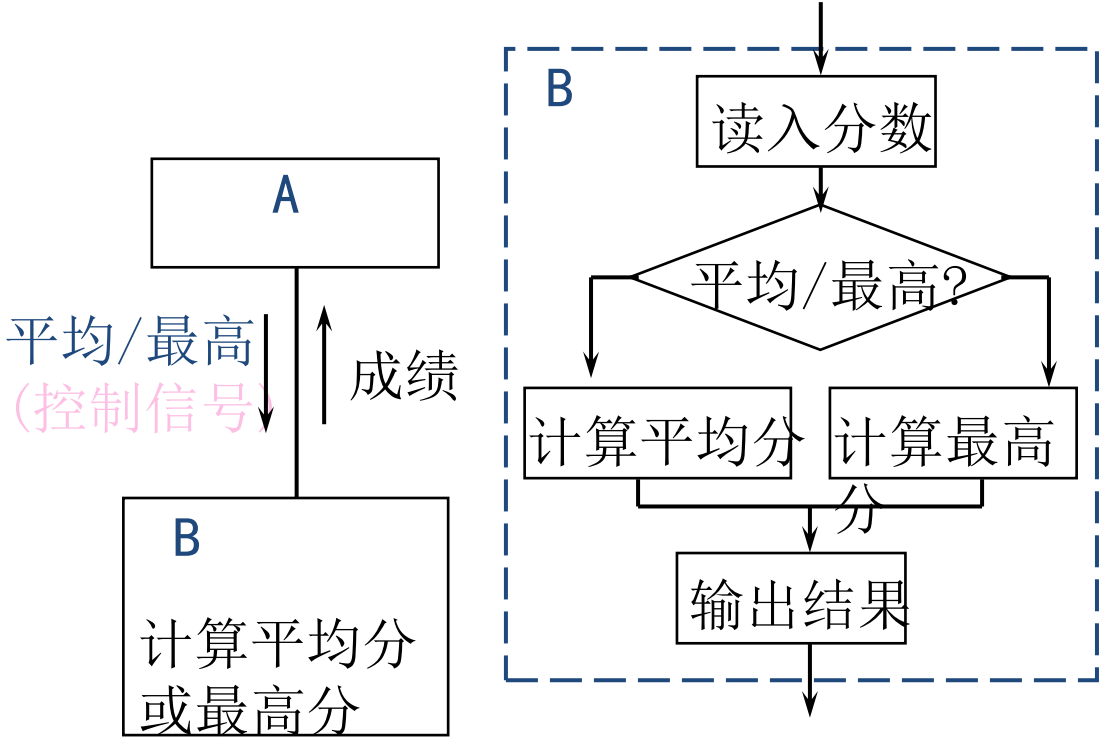
"*住户情况*"是一个数据结构。图中模块都与此数据结构有关。"计算水费"和"计算电费"本无关，由于引用了此数据结构产生依赖关系，它们之间也是标记偶合



将标记耦合修改为数据耦合

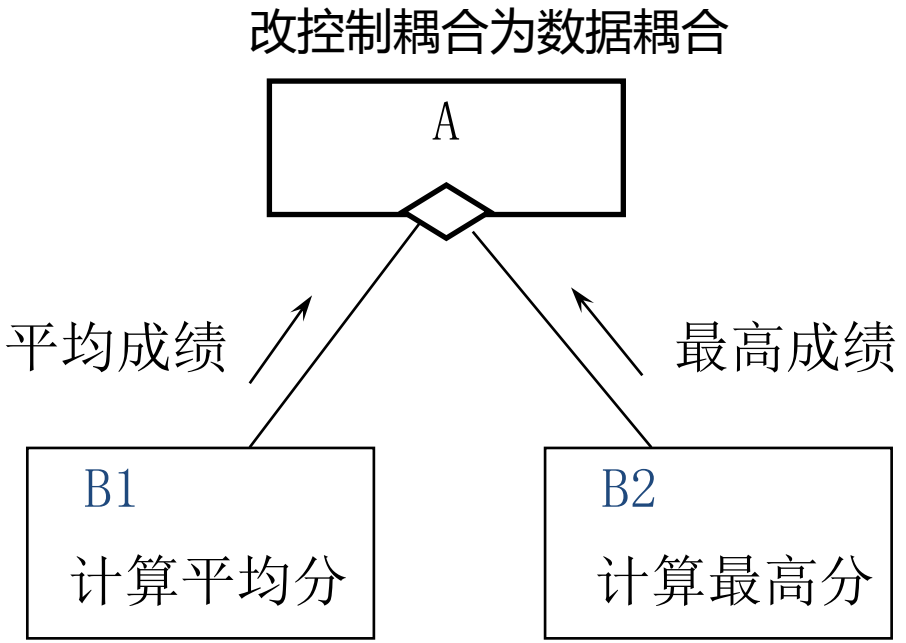
块间联系·控制耦合

一模块向下属模块传递的信息（开关量、标志等控制被调用模块决策的变量）控制了被调用模块的内部逻辑。



块间联系·控制耦合

去除模块间控制耦合的方法：
控制耦合增加了理解和编程的复杂性，调用模块必须知道被调模块的内部逻辑，增加了相互依赖
(1)将被调用模块内的判定上移到调用模块中进行
(2)被调用模块分解成若干单一功能模块



块间联系·外部耦合

- 一组模块均与同一外部环境关联(例如,I/O模块与特定的设备、格式和通信协议相关联)，它们之间便存在外部耦合。
- 外部偶合必不可少，但这种模块数目应尽量少。

块间联系·公共耦合(公共数据区耦合)

- 一组模块引用同一个公用数据区(也称全局数据区、公共数据环境)。
- 公共数据区指:
- 全局数据结构
- 共享通讯区
- 内存公共覆盖区等

公共耦合存在的问题

(1)软件可理解性降低

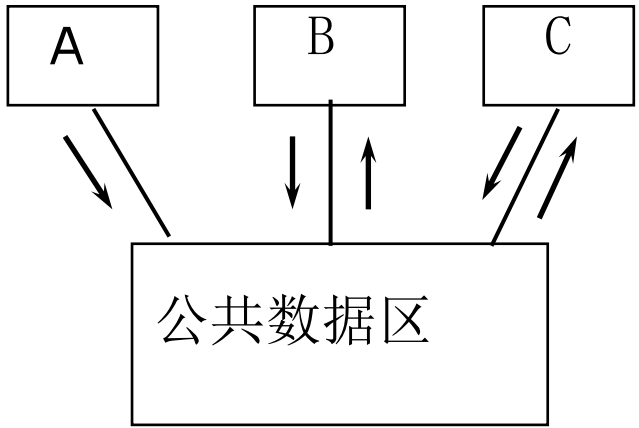
(2)诊断错误困难

(3)软件可维护性差,

(4)软件可靠性差

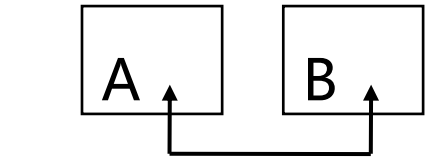
(公共数据区及全程变量无保护措施)

慎用公共数据区和全程变量!!!

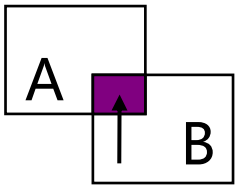


模块A、B、C间存在错综复杂的联系

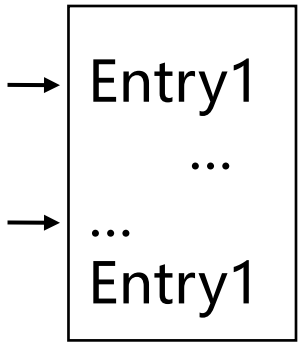
块间联系·内容耦合



一模块直接访问另一模块的内部信息 (程序代码或数据)



模块代码重叠



多入口模块

最不好的耦合形式 ！！！

谢谢！