第五章　总体设计

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

5.6面向对象的设计方法

设置

一个模块直接控制（调用）的下层模块的数目称为模块的（　　）。

A 扇入数

B 扇出数

C 宽度

D 作用域

提交

单选题 1分

设置

软件结构图的形态特征能反映程序重用率的是（　　）。

A 深度

B 宽度

C 扇入

D 扇出

提交

软件结构化设计中，好的软件结构应该力求做到（ ）

A 顶层扇出较少，中间扇出较高，底层模块低扇入

B 顶层扇出较高，中间扇出较少，底层模块高扇入

C 顶层扇入较少，中间扇出较高，底层模块高扇入

D 顶层扇入较少，中间扇入较高，底层模块低扇入

提交

设置

划分模块时，下列说法正确的是（ ）

A 作用范围与控制范围互不包含

B 作用范围与控制范围不受限制

C 作用范围应在其控制范围之内

D 控制范围应在其作用范围之内

提交

填空题 3分

设置

耦合性和内聚性是衡量 [填空1] 的两个定性标准 。

将软件系统划分模块时，尽量做到 [填空2] 内聚、

[填空3] 耦合。

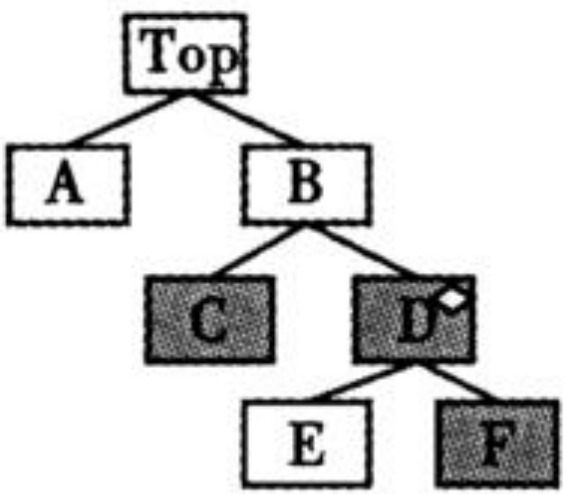
作答

正常使用填空题需3.0以上版本雨课堂

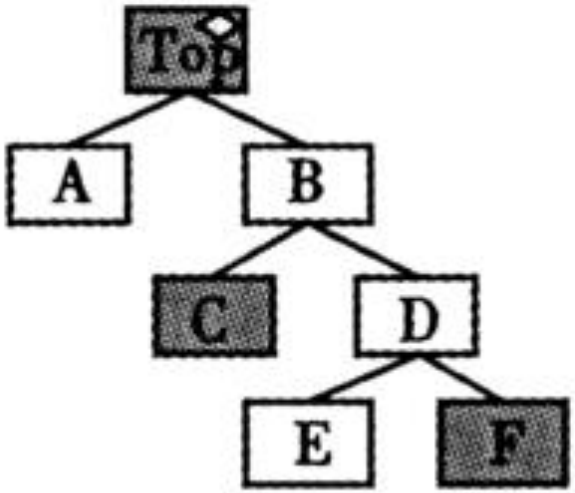
主观题 10分

设置

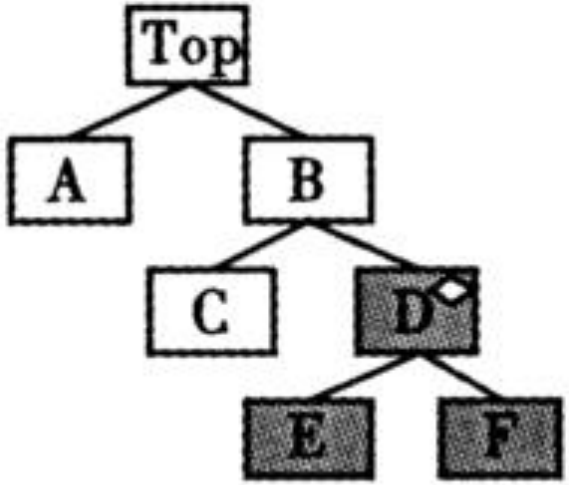
分析对比下图3个模块结构图的优劣。菱形块代表判定所在的位置，灰色块为受该判定影响的模块。



A



B



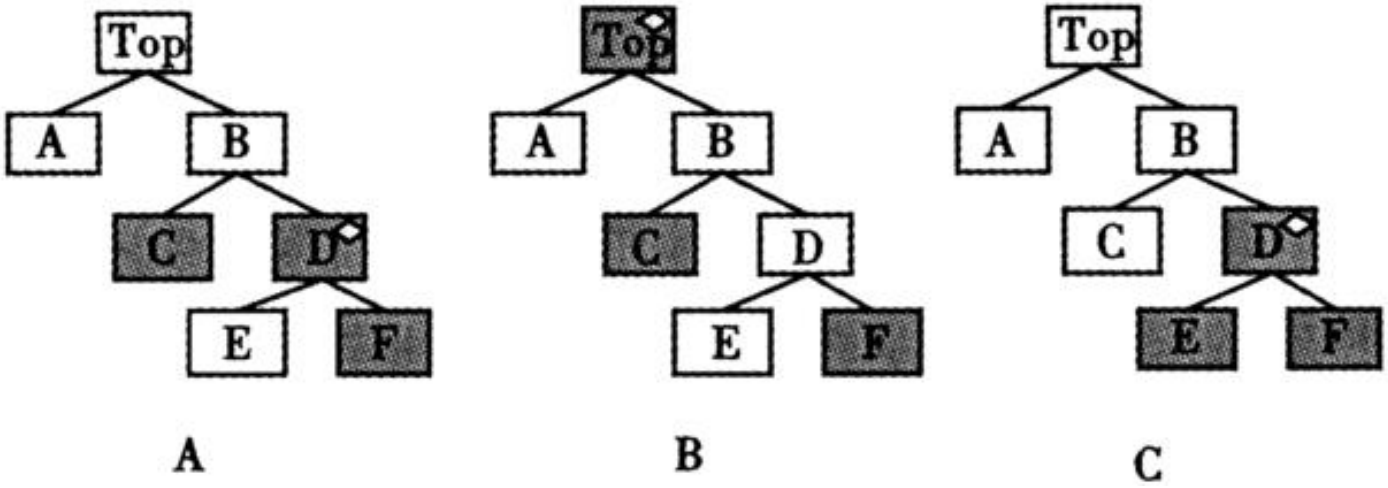
C

作答

正常使用主观题需2.0以上版本雨课堂

课堂练习

10、分析对比下图3个模块结构图的优劣。



菱形块代表判定所在的位置，灰色块为受该判定影响的模块。

A最差，作用域在控制域之外。

B较好，作用域在控制域中，但判定所在的位置过高。

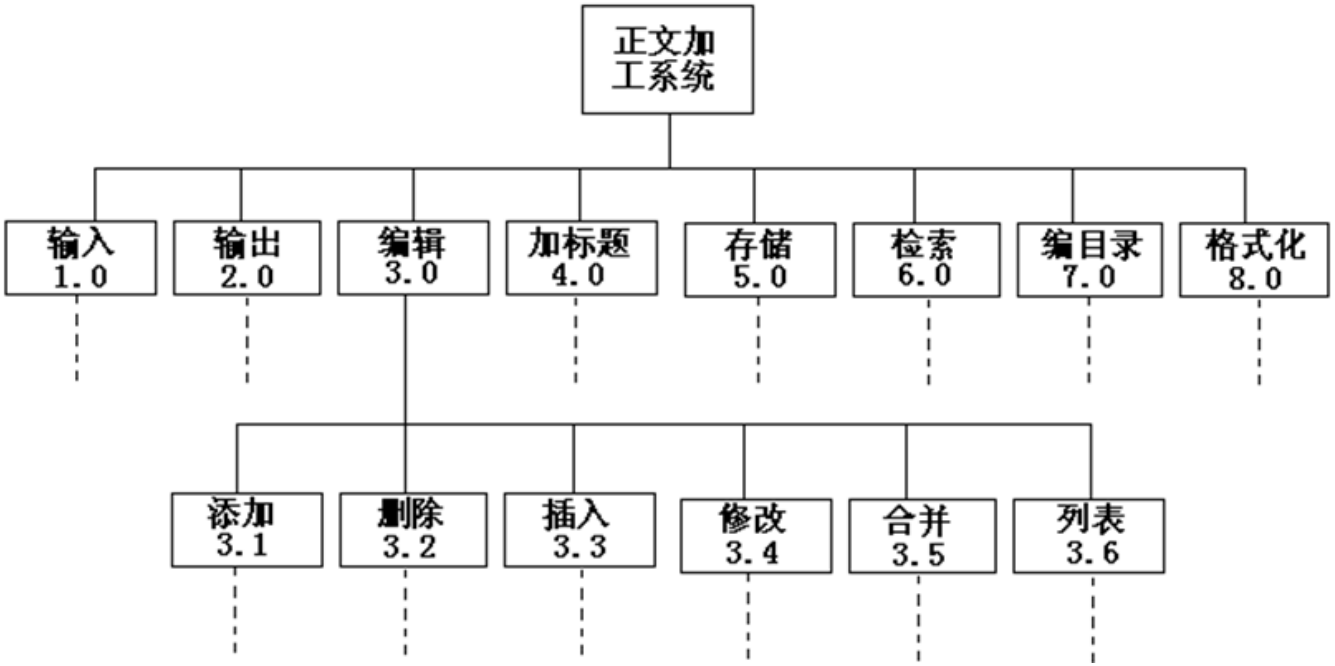
C最好，作用域在控制域中，且判定的位置恰当。

5.4 描绘软件结构的图形工具

- **层次图**：描述软件的层次结构（H图）。
  - 层次图中每个矩形框代表一个模块，矩形框之间的连线表示模块**调用关系**。
  - 层次图适合用来描绘软件的层次结构。

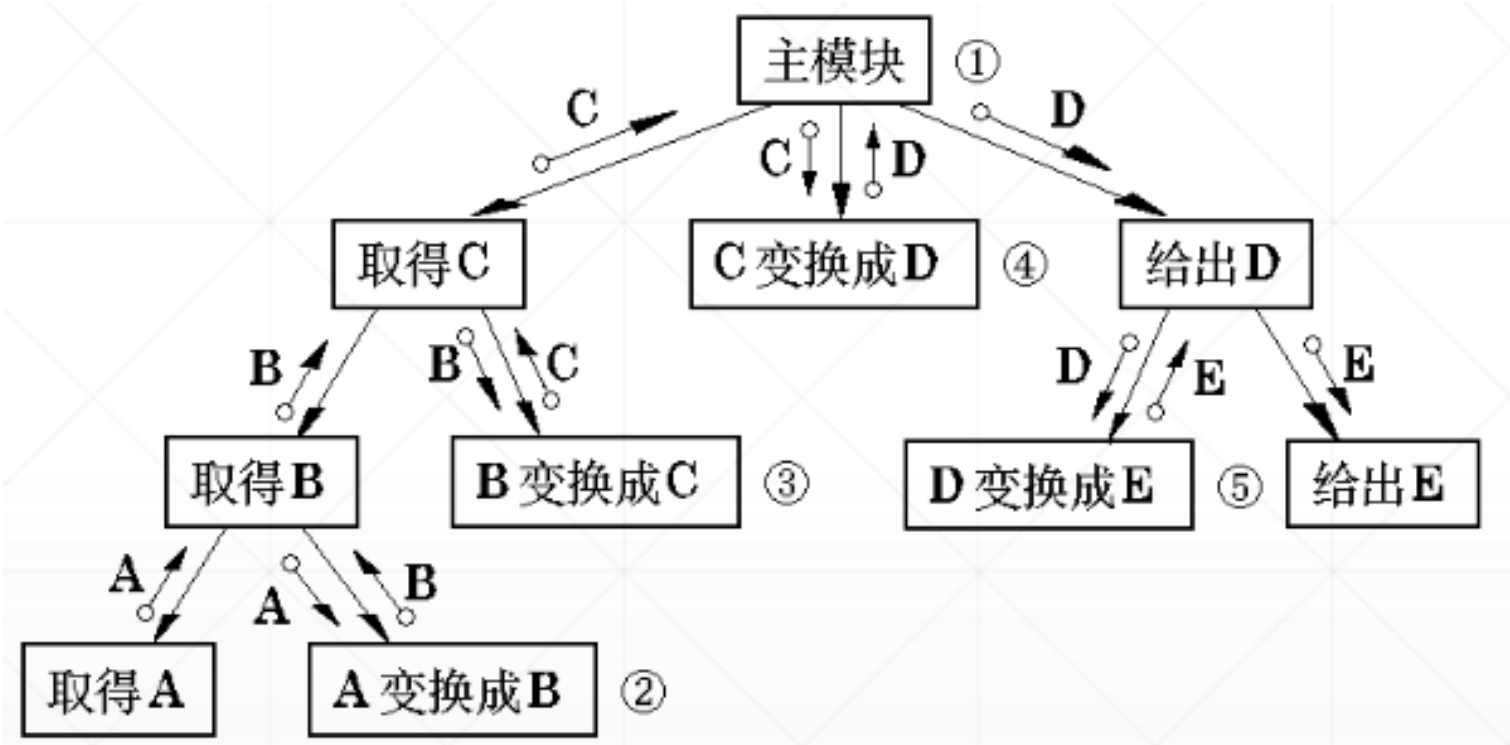
需求分析中介绍的层次方框图描述的是数据结构的组成关系。

正文加工系统的层次图



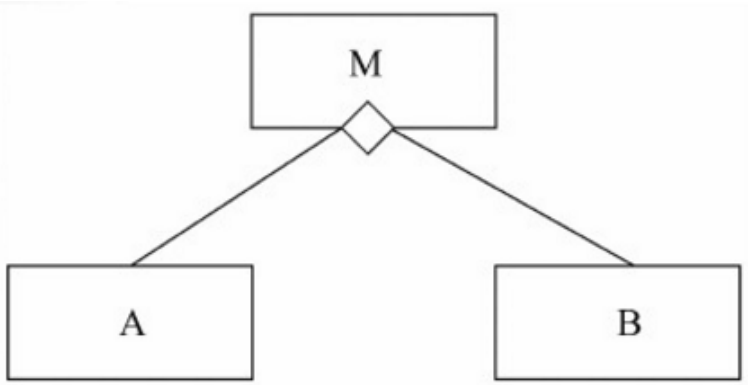
5.4 描绘软件结构的图形工具

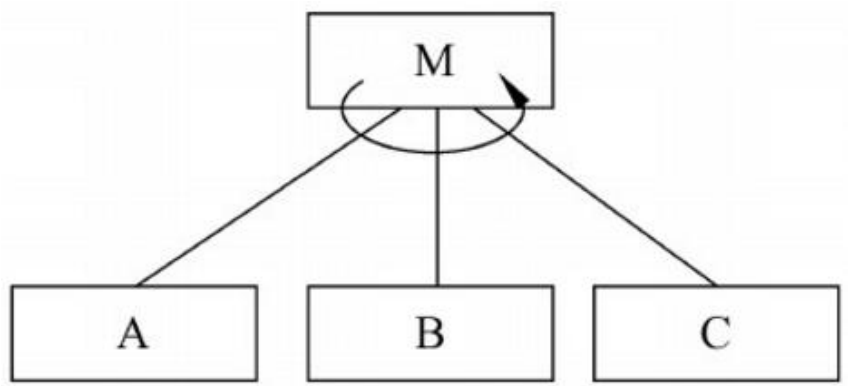
结构图是进行软件结构设计的另一个工具。方框代表模块，框内注明模块的名字或主要功能，方框之间的箭头(或直线)表示模块的调用关系。尾部是空心圆表示传递的是数据，实心圆表示传递的是控制信息。



5.4 描绘软件结构的图形工具

- 一些附加符号，可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。





层次图和结构图并不严格表示调用次序，只表明一个模块调用哪些模块，至于模块是否还有其它成分则完全没有表示。

结构化设计

- 结构化设计(Structured Design，简称SD)是将结构化分析得到的数据流图映射成软件体系结构的一种设计方法。
  - 强调模块化、自顶向下、逐步求精、信息隐蔽、高内聚低耦合等设计准则。分为概要设计和详细设计两大步骤：
    - 概要设计是对软件系统的总体设计，结构化设计方法的任务是将系统分解成模块，确定每个模块的功能、接口(模块间传递的数据)及其调用关系。
    - 详细设计是对模块实现细节的设计，采用结构化程序设计(Structured Programming，简称SP)方法

SA、SD和SP构成完整的结构化方法体系

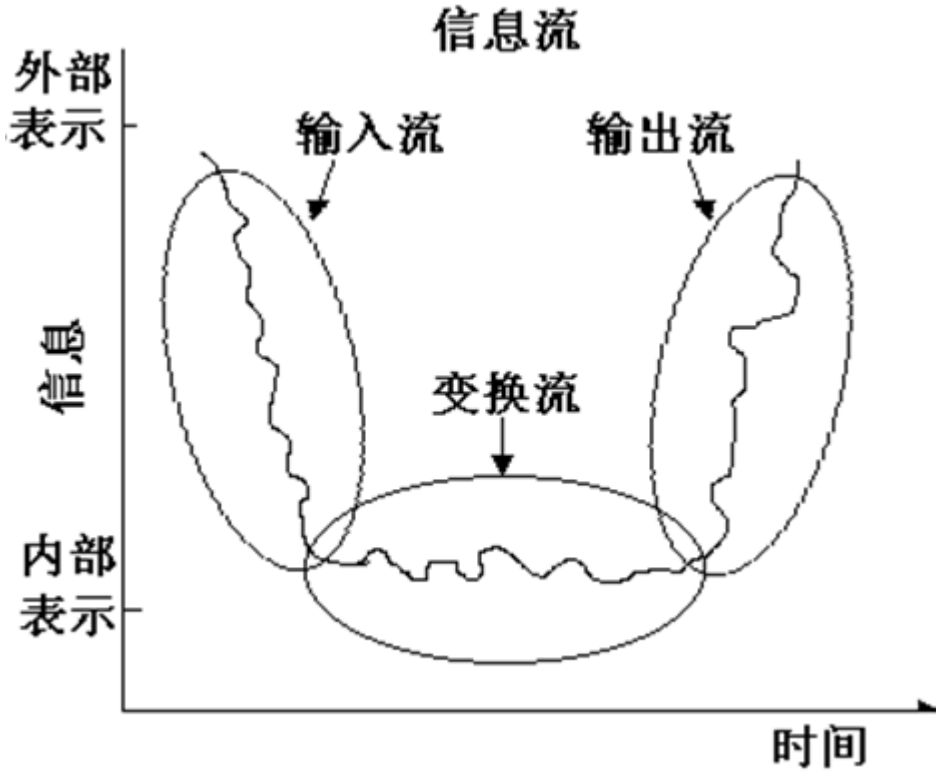
5.5 面向数据流的设计方法

一、概念

面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定映射方法。信息流有两种类型：

1、变换流

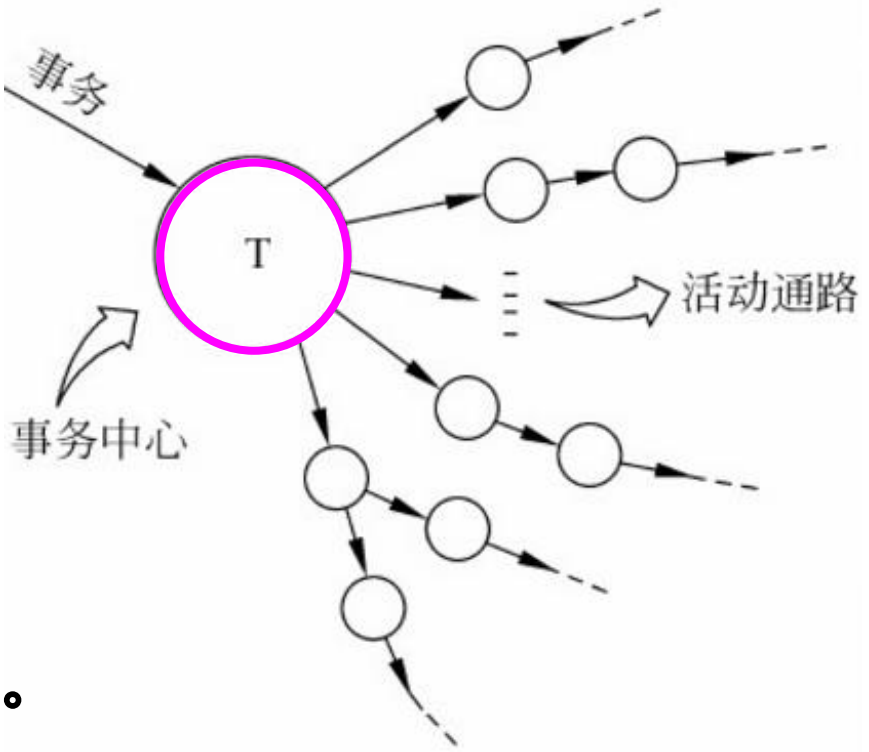
信息流沿输入通路进入系统，由外部形式变换成内部形式，进入系统的信息通过变换中心，经加工处理后再沿输出通路变换成外部形式离开软件系统。当数据流图具有这些特征时，这种信息流就叫作变换流。



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。

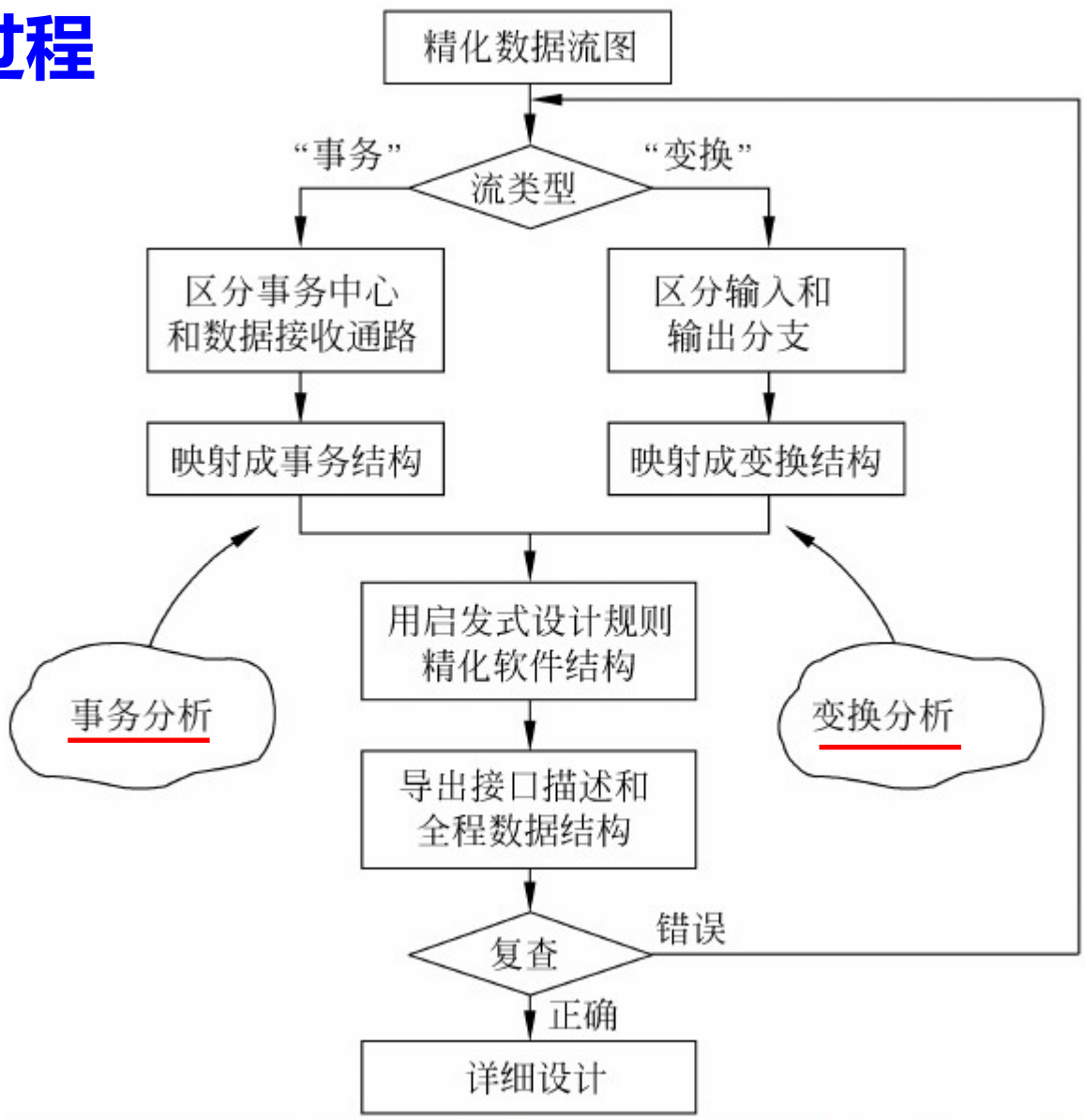
1)接收输入数据(又称为事务)。

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

- 以具有“智能”的汽车数字仪表板的设计为例介绍变换分析设计的具体步骤。
  - 假设仪表板将完成下述功能描述：

    （1）通过模数转换实现传感器与微机的接口

    （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）

    （4）指示加速或减速

    （5）超速警告，超过55英里/小时，发出超速警告铃声。

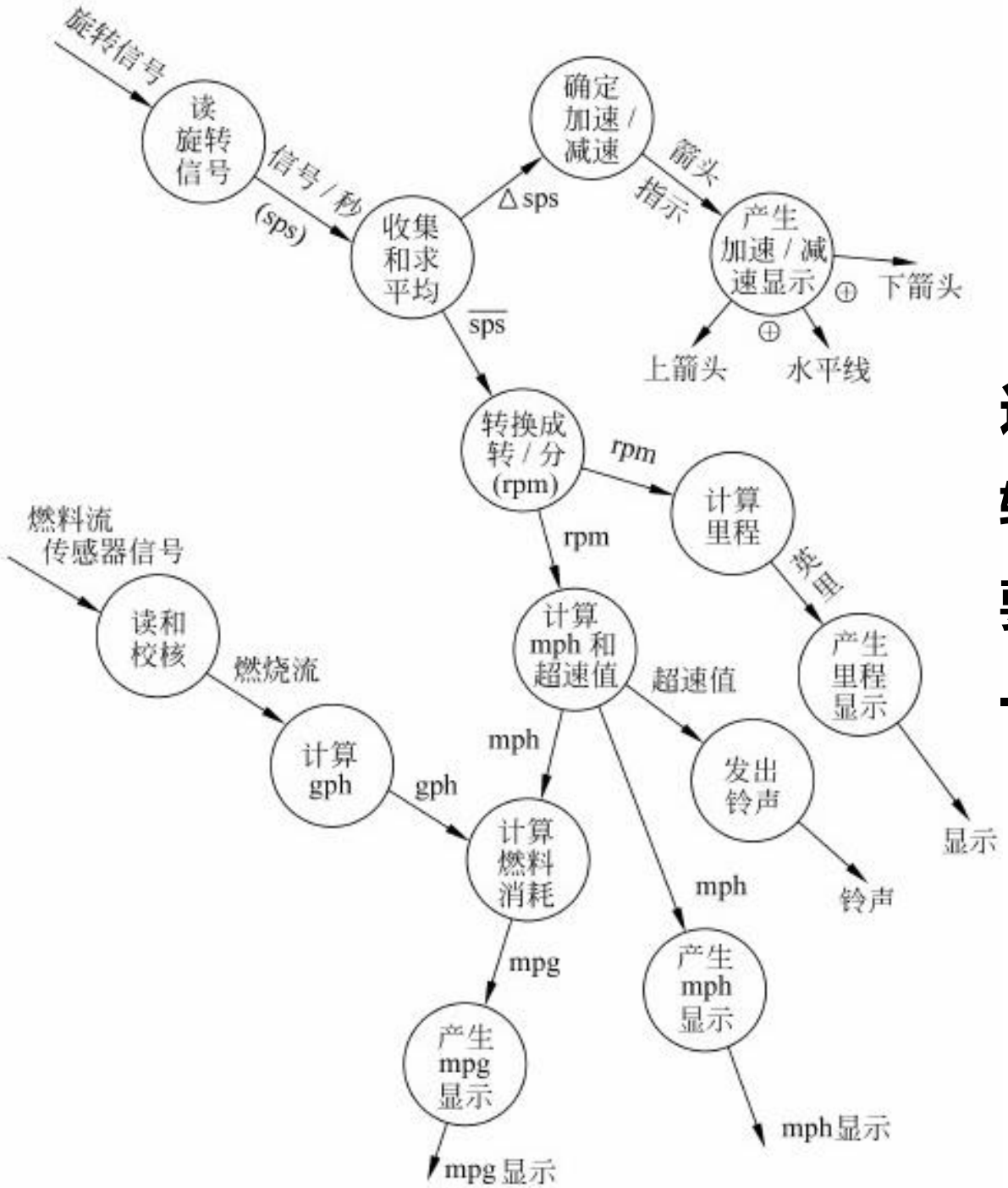
5.5 面向数据流的设计方法

1、复查基本系统模型。

- 确保系统的输入数据和输出数据符合实际。

2、复查并精化数据流图。

- 确保数据流图给出了目标系统的正确逻辑模型，并使数据流图中每个处理都代表一个规模适中相对独立的子功能。
- 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：



这个数据流图已经比较详细了，因此不需要精化就可以进行下一个设计步骤。

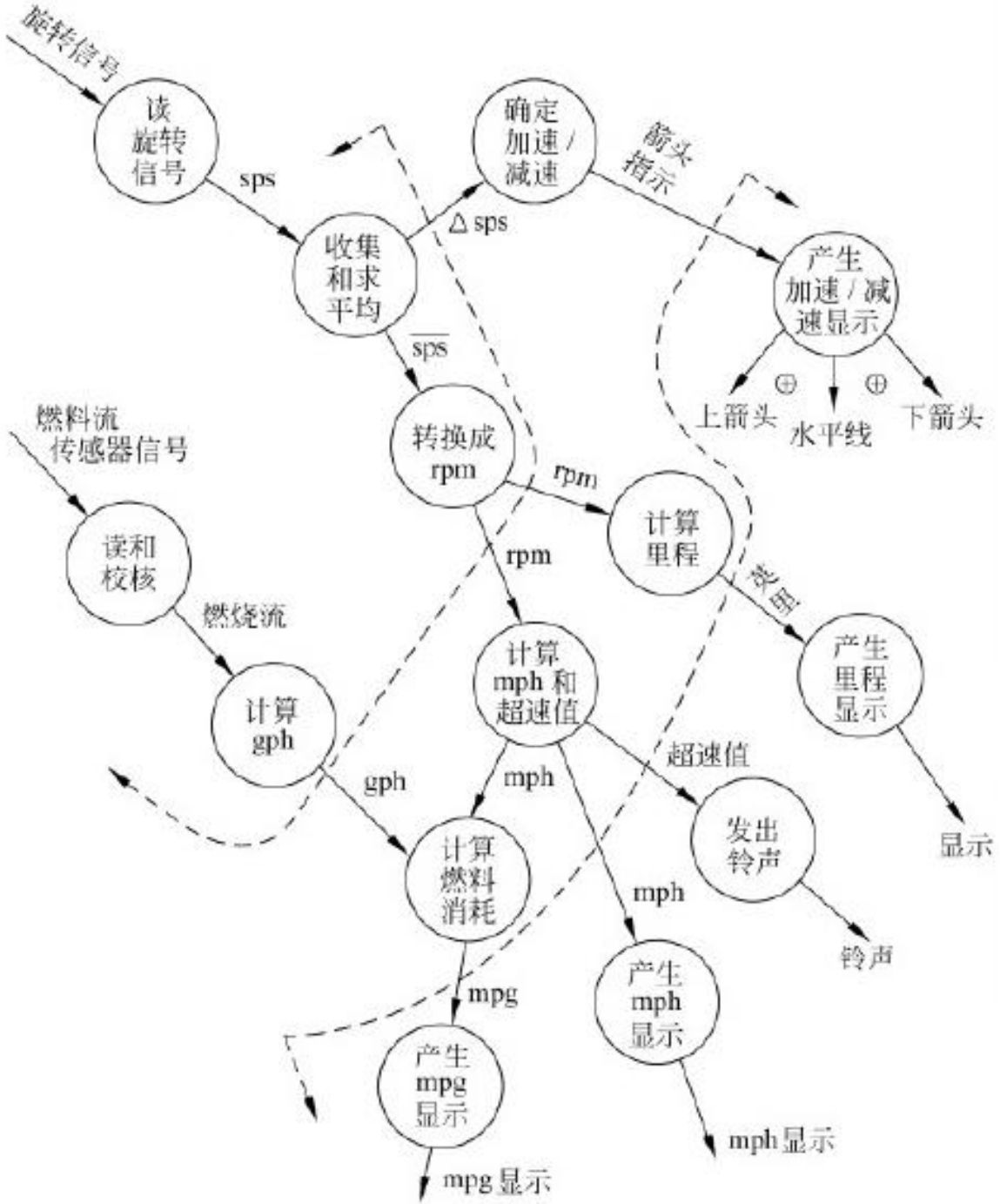
5.5 面向数据流的设计方法

3、确定数据流图具有变换特性还是事务特性。

- 一个数据流图中可能含有变换流的成分，又含有事务流的成分，应根据其中占优势的属性来确定**数据流的全局特性**。

4、确定输入流和输出流的边界，孤立出变换中心

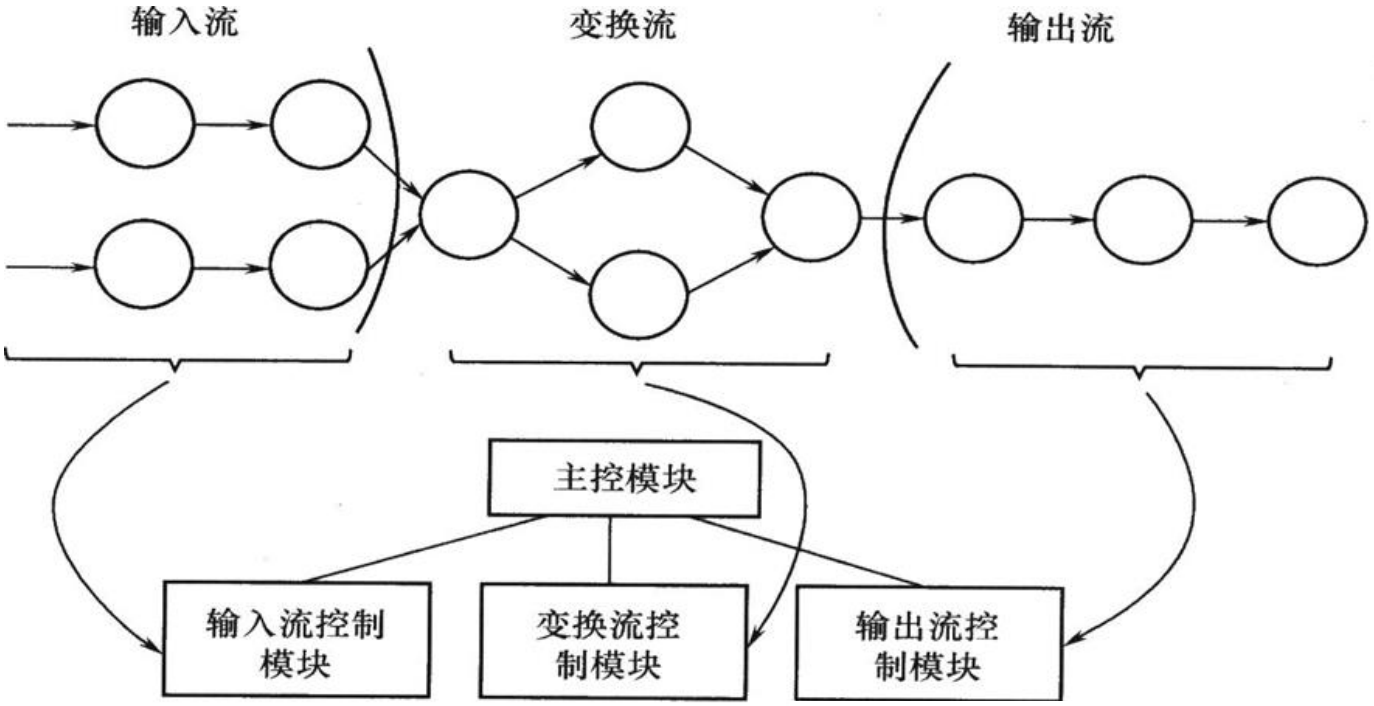
- **确定输出流边界**：一般具有显示、打印、产生控制等。
- **确定输入流边界**：一般作为有效输入的数据是在输入原始数据后进行一些简单处理，转换成实际需要的内部形式。
- 输入、输出流边界的划分可能因人而异，不同的设计人员可能把边界沿着数据通道向前推进或后退一个处理框，这对最后的软件结构影响不大。

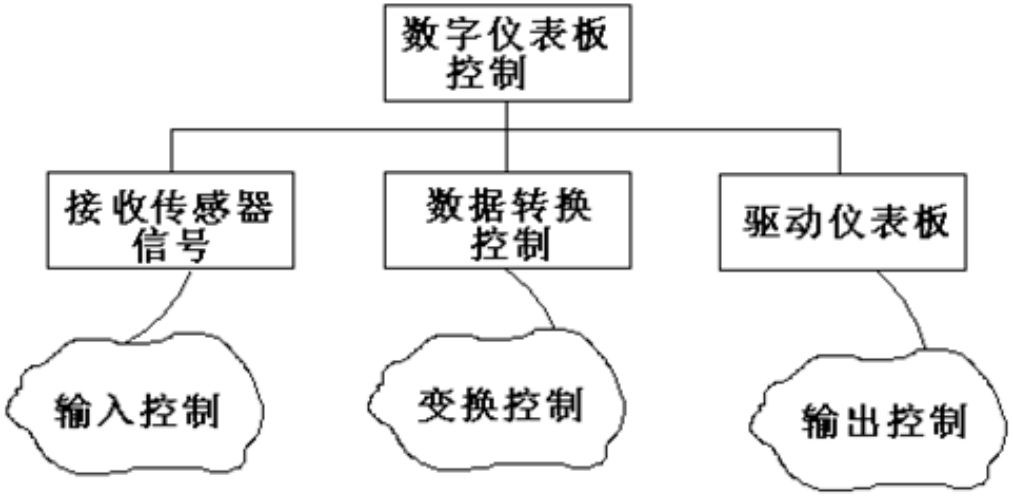

旋转信号
读旋转信号
sps
收集和求平均
△sps
确定加速/减速
箭头指示
产生加速/减速显示
上箭头
水平线
下箭头
sps
转换成rpm
燃料流传感器信号
读和校核
燃烧流
计算gph
rpm
计算里程
里程
产生里程显示
显示
计算mph和超速值
超速值
发出铃声
铃声
gph
mph
计算燃料消耗
mpg
产生mpg显示
mpg显示
产生mph显示
mph显示

5、完成第一级分解，设计系统的上层模块。

- 任务：确定软件结构的最顶层模块(只含一个用于控制的主模块)控制第二层哪些模块。
- 对于变换流，可以明显地分为输入、中心变换和输出三部分，其数据流图通常被映射成一种特殊的软件结构。
- 三叉控制结构
  - **数据输入控制模块**：协调对所有输入数据的接收。
  - **中心变换控制模块**：协调对数据的各种加工处理。
  - **数据输出控制模块**：协调对输出数据的产生和输出。

5.5 面向数据流的设计方法





5.5 面向数据流的设计方法

6、完成“第二级分解”。

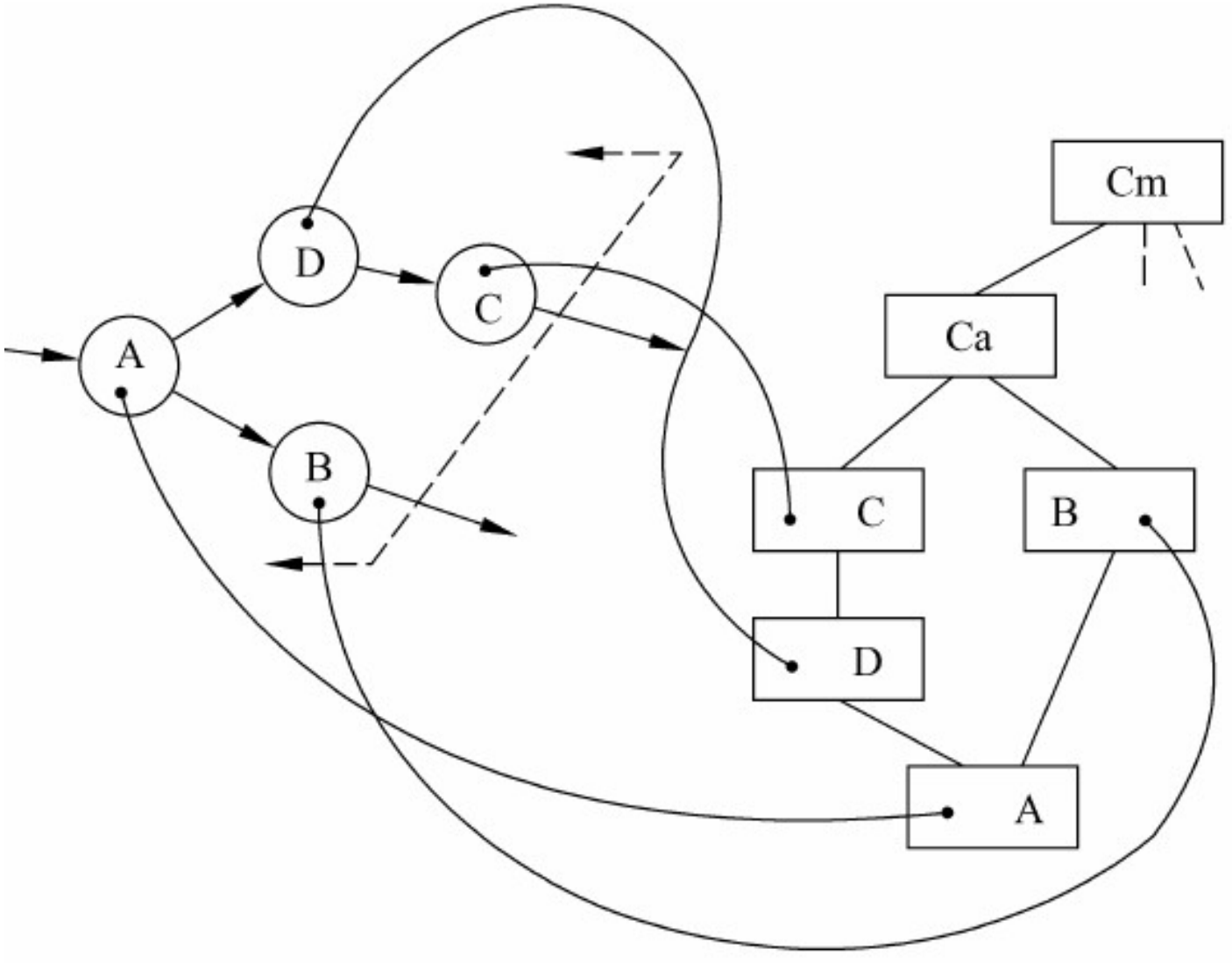
- 把数据流图中每个处理映射成软件结构中的一个适当模块。
- 方法：

1、从变换中心的边界开始逆着输入通路向外移动，把输入通路中的每个处理映射成输入数据控制模块控制下的低层模块

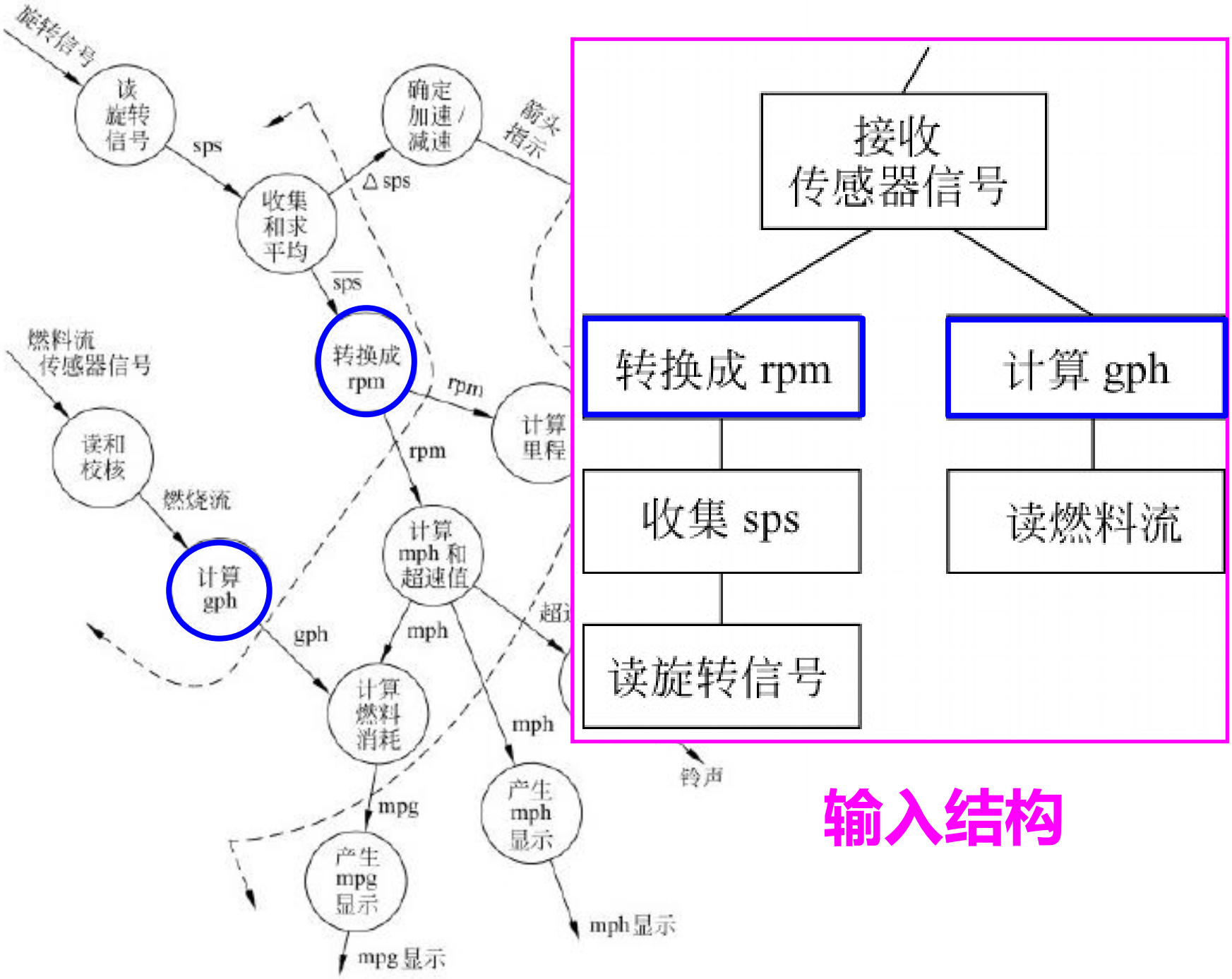
2、沿着输出通路向外移动，把输出通路中的每个处理映射成输出数据控制模块控制下的低层模块。

3、把变换中心内的每个处理映射成中心变换控制模块下的低层模块。

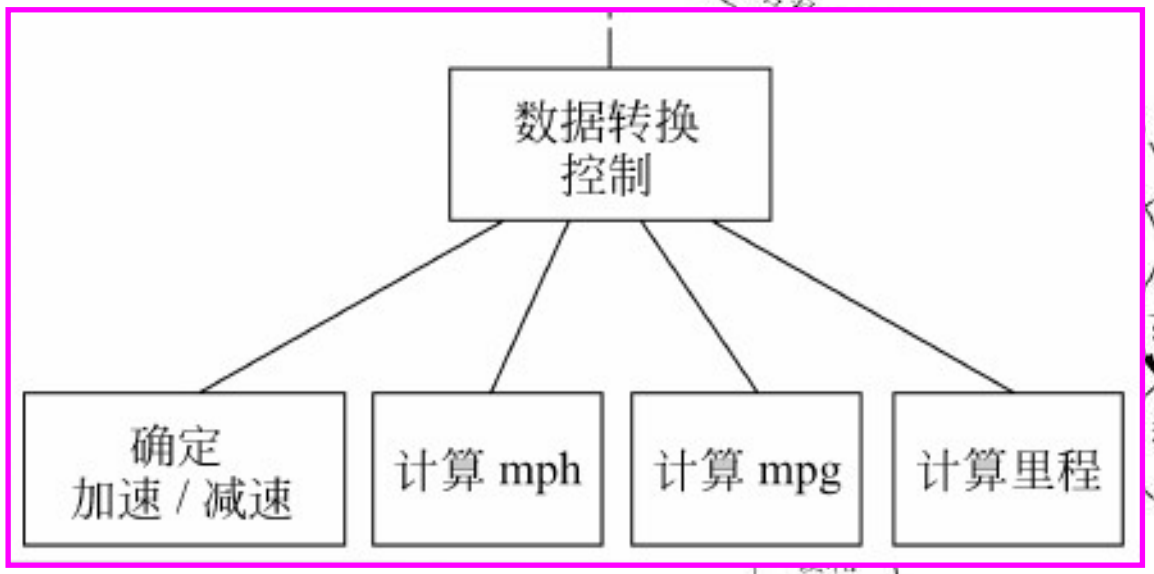
5.5 面向数据流的设计方法



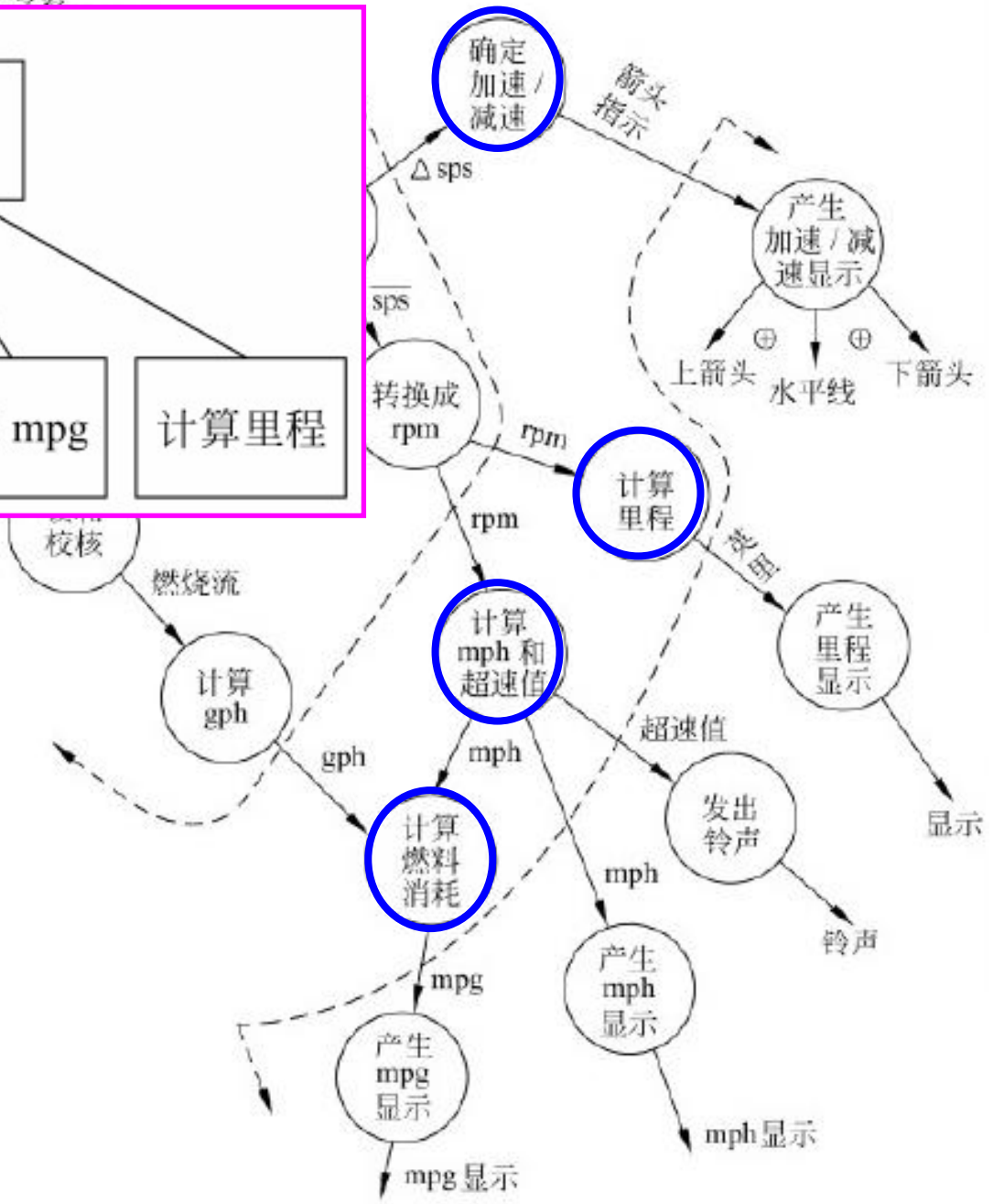
第二级分解的普遍途径

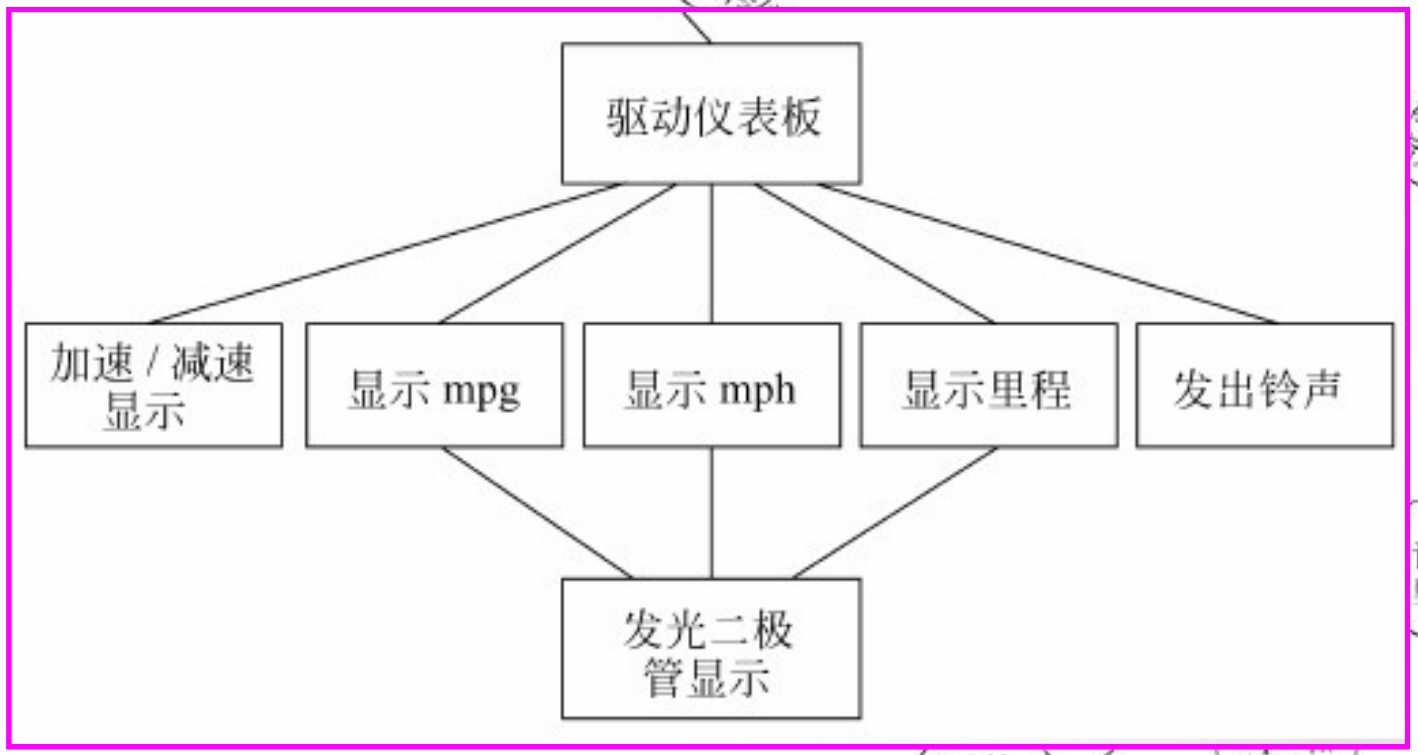

旋转信号
读旋转信号
sps
收集和求平均
确定加速/减速
Δ sps
箭头指示
sps
转换成rpm
rpm
计算里程
rpm
燃料流传感器信号
读和校核
燃烧流
计算gph
计算mph和超速值
gph
mph
超速
计算燃料消耗
mph
mpg
产生mpg显示
产生mph显示
mpg显示
mph显示
铃声
接收传感器信号
转换成 rpm
计算 gph
收集 sps
读燃料流
读旋转信号

输入结构

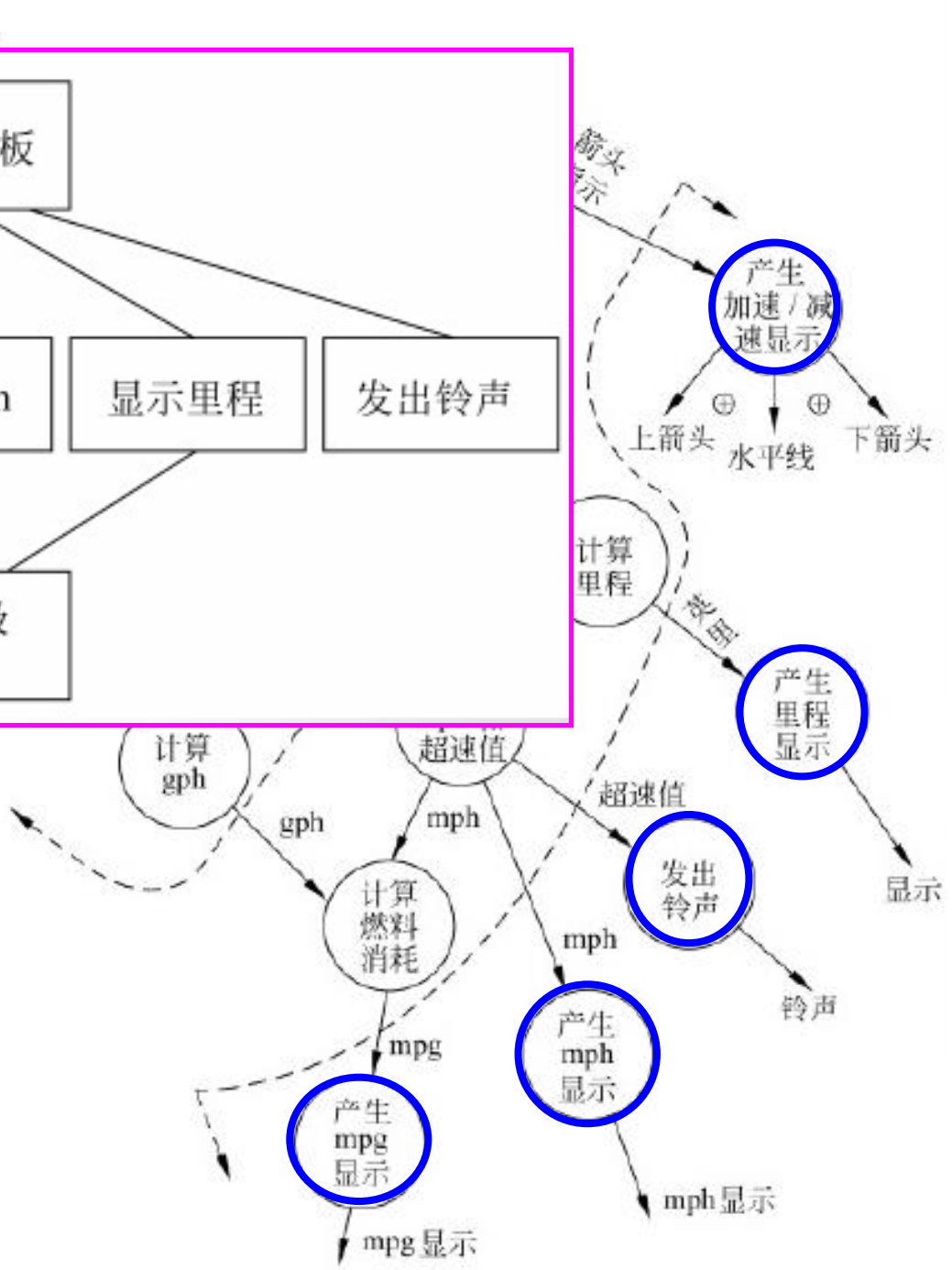

数据转换
控制
确定
加速 / 减速
计算 mph
计算 mpg
计算里程

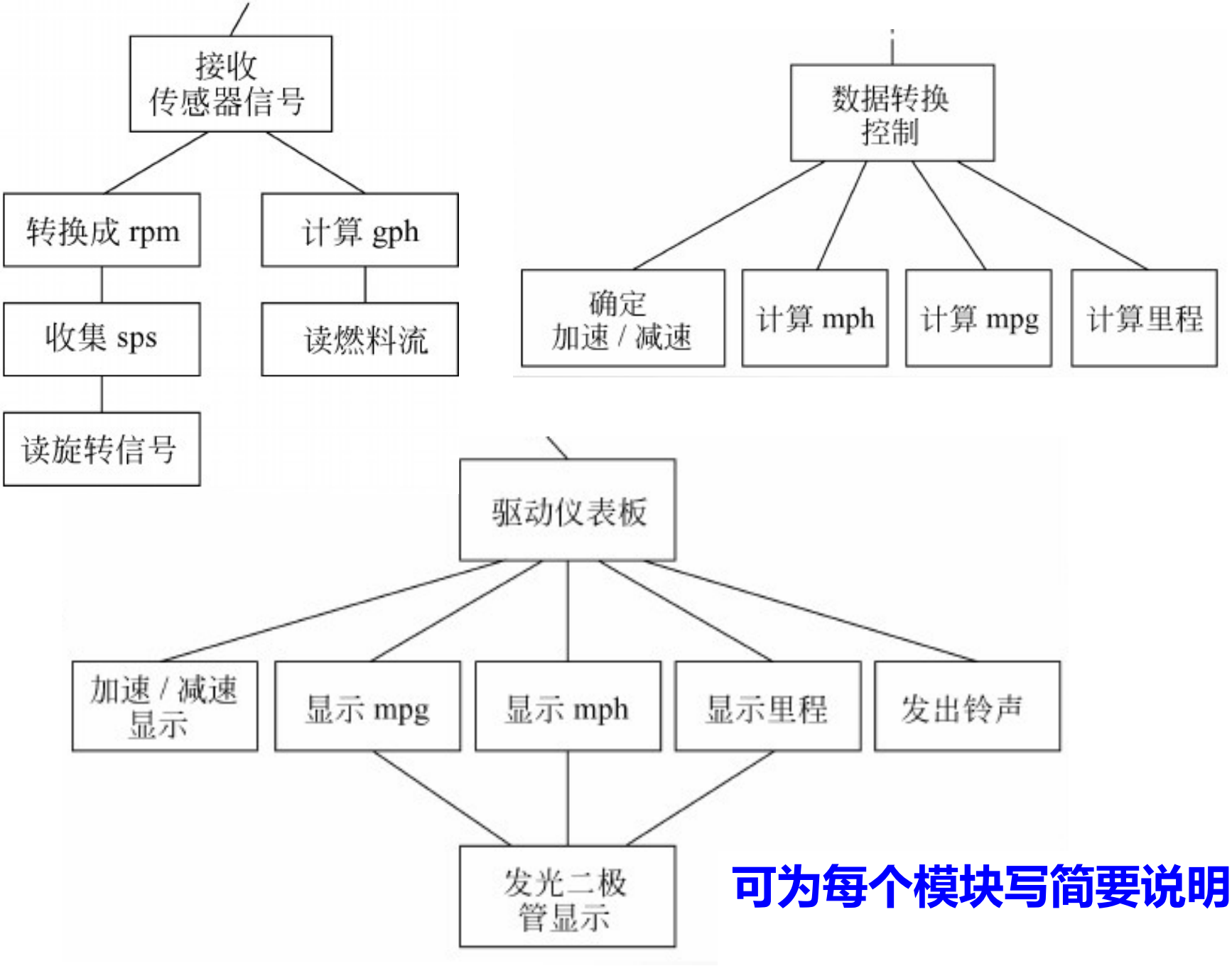
变换结构


确定
加速 /
减速
箭头
指示
Δ sps
产生
加速 / 减
速显示
上箭头
水平线
下箭头
sps
转换成
rpm
rpm
计算
里程
rpm
校核
燃烧流
计算
mph 和
超速值
产生
里程
显示
计算
gph
gph
mph
超速值
显示
发出
铃声
计算
燃料
消耗
mph
铃声
产生
mph
显示
mpg
产生
mpg
显示
mph 显示
mpg 显示


驱动仪表板
加速 / 减速显示
显示 mpg
显示 mph
显示里程
发出铃声
发光二极管显示

输出结构


箭头
产生加速 / 减速显示
上箭头
水平线
下箭头
计算里程
里程
产生里程显示
显示
计算gph
超速值
gph
mph
超速值
发出铃声
铃声
计算燃料消耗
mph
产生mph显示
mph显示
mpg
产生mpg显示
mpg显示



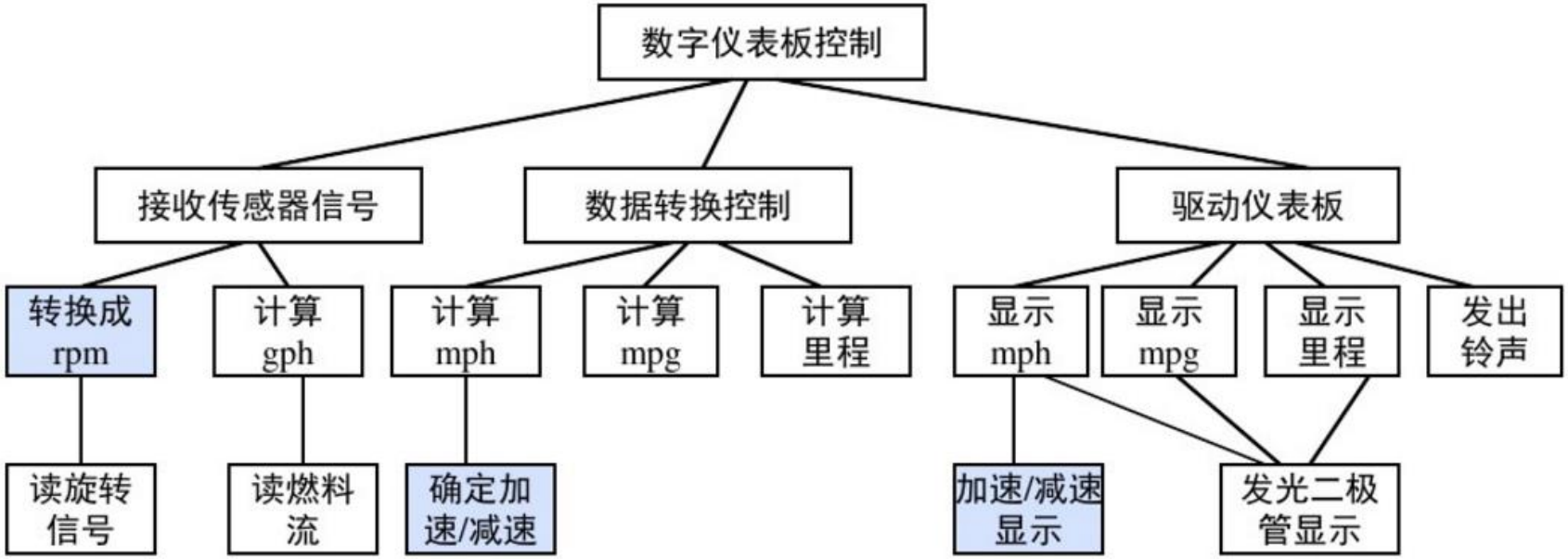
可为每个模块写简要说明

5.5 面向数据流的设计方法

7、使用设计度量和启发式规则精化第一次分割得到的软件结构。

- 通过合并分解，得到易于实现、测试和维护的软件结构。如：

1. 输入结构中的“转换成rpm”与“收集sps”可以合并。
2. “确定加速/减速”可放在“计算mph” 下面。
3. “加速/减速显示”可放在“显示mph”下面。



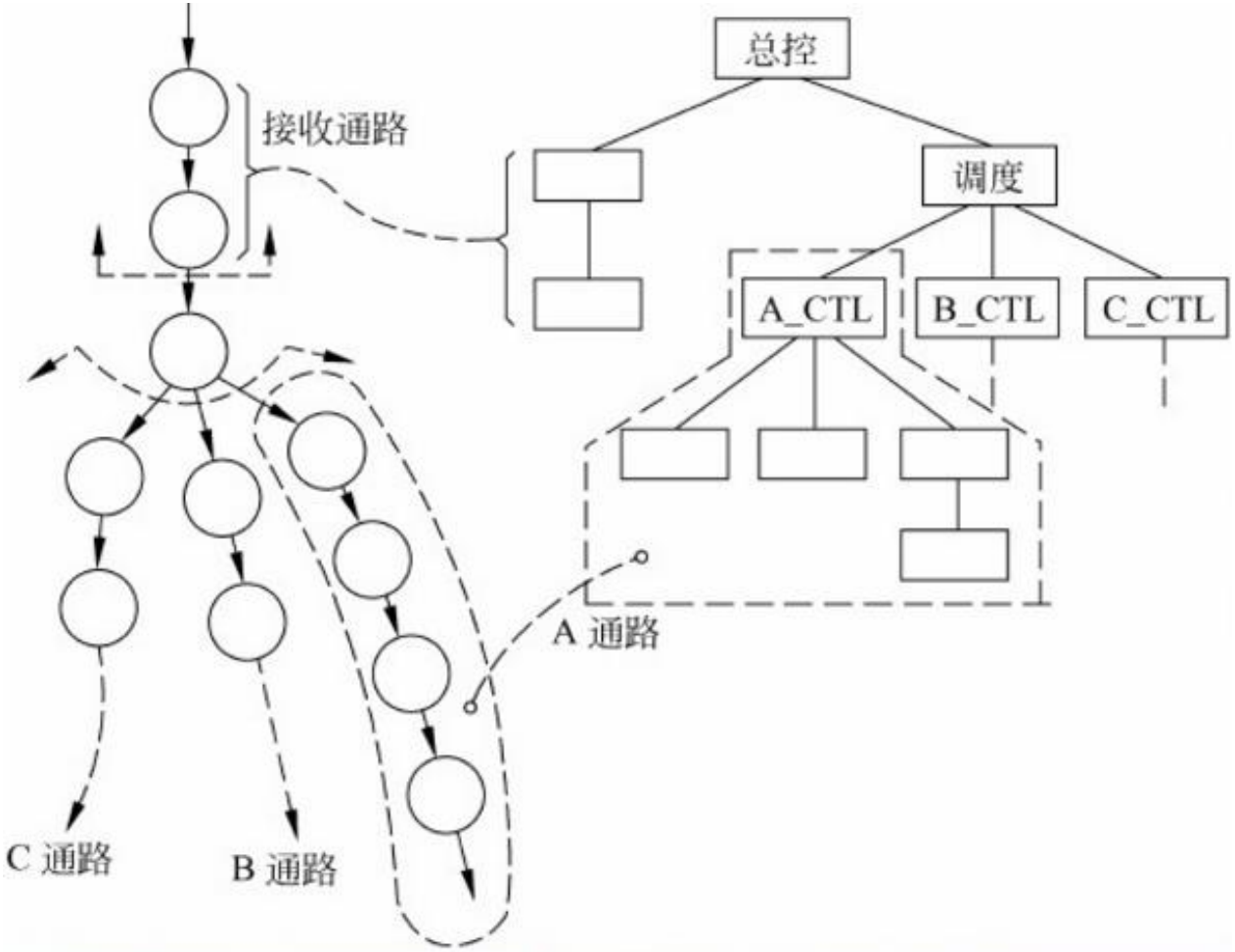
精化后的数字仪表板系统的软件结构

二、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。

设计步骤和变换分析的设计步骤大部分相同或类似，主要差别在于映射方法不同。

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。



三、混合流分析

在大型系统中往往既有变换流，也有事务流，一般采用变换流为主，事务流为辅的原则 。

(1) 先划出输入输出；

(2) 设计软件结构的上层；

(3) 根据DFD各部分的特征，进行变换流和事务流设计 。

5.6 面向对象的设计方法

一、面向对象设计的准则

1. 模块化

- 对象就是模块
- 把数据结构和操作这些数据的方法紧密结合在一起

2. 抽象

- 过程抽象
- 数据抽象：类

3. 信息隐藏

- 通过对象的封装性实现
- 类分离了接口与实现，支持信息隐藏

5.6 面向对象的设计方法

4. 弱耦合

- 对象之间的耦合：交互耦合&继承耦合
- **交互耦合**：对象之间的耦合通过消息连接来实现。
  - 交互耦合应尽可能松散。尽量降低消息连接的复杂度，减少对象发送或接收的消息数。
- **继承耦合**：一般化类与特殊类之间耦合的一种形式。
  - 与交互耦合相反，应该**提高继承耦合程度**。
  - 通过继承关系结合起来的基类和派生类，构成了系统中粒度更大的模块。彼此之间应该越紧密越好。

5.6 面向对象的设计方法

5. 强内聚

- 面向对象设计中存在3种内聚。
  - **服务内聚**：一个服务应该完成一个且仅完成一个功能。
  - **类内聚**：一个类应该只有一个用途，它的属性和服务应该是高内聚的。
  - **一般-特殊内聚**：符合领域知识的表示形式。

6. 可重用

- 尽量使用已有的类
- 如果确实需要创建新类，则在设计这些新类的协议时，应该考虑将来的可重复使用性

二、启发规则

1.设计结果应该清晰易懂

- 用词一致：尽量使用习惯的名字，不同类中相似服务的名字应该相同。
- 使用已有的协议
- 减少消息模式的数目
- 避免模糊的定义

2. 一般-特殊结构的深度应适当

- 一个中等规模(大约包含100个类)的系统中，类等级层次数应保持为7±2。

二、启发规则

3.设计简单的类

- 避免包含过多的属性
- 有明确的定义：分配给类的任务应简单
- 尽量简化对象之间的合作关系
- 不要提供太多服务：一般一个类提供的公共服务不超过7个

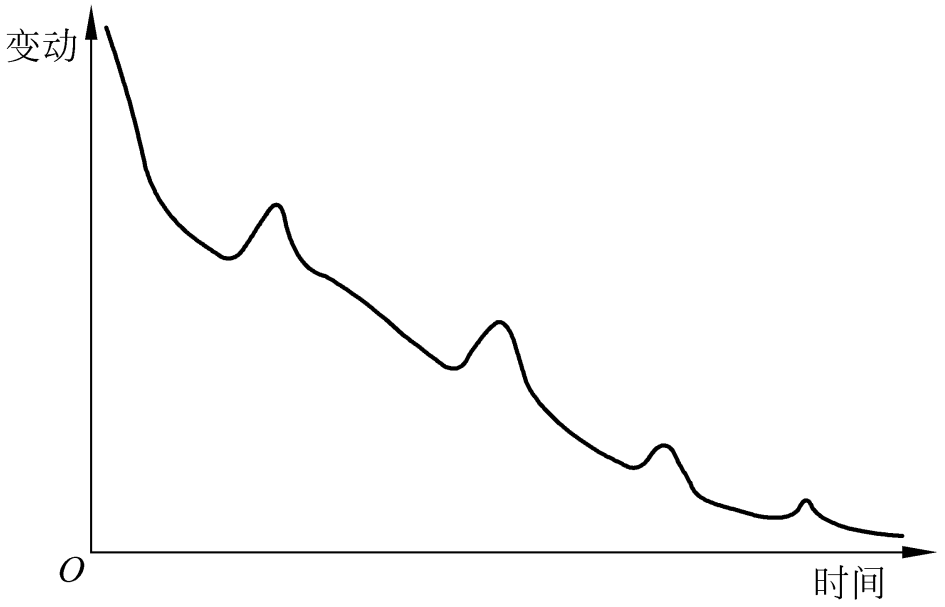
4.使用简单的协议

- 消息中的参数不要超过3个。通过复杂消息相互关联的对象是紧耦合的，对一个对象的修改往往导致其他对象的修改。

二、启发规则

5.使用简单的服务

- 类中的服务通常都很小
- 尽量避免使用复杂的服务



6.把设计变动减至最小

- 理想的设计变动曲线如图所示
- 在设计的早期阶段，变动较大，随着时间推移，设计方案日趋成熟，改动也越来越小了

5.6 面向对象的设计方法

三、系统分解

- 先把系统分解成若干个比较小的部分，再分别设计每个部分。
- 系统的主要组成部分称为子系统。通常根据功能划分。

1．子系统之间的两种交互方式

(1) 平等伙伴关系

- 每个子系统都可调用其他子系统，须了解其他子系统的接口
- 子系统之间的交互复杂，可能存在通信环路。

5.6 面向对象的设计方法

(2) 客户-供应商关系

- 作为“客户”的 子系统调用作为 “供应商”的子系统，后者完成某些服务工作并返回结果。
- 前者必须了解后者的接口，后者却无须了解前者的接口，因为任何交互行为都是由前者驱动的。

单向交互比双向交互容易理解、设计和修改，应尽量使用客户-供应商关系。

5.6 面向对象的设计方法

2. 组织系统的两种方案

（1）层次组织

- 把软件系统组织成一个层次系统，每层是一个子系统。
- 上层在下层的基础上建立，下层为实现上层功能而提供必要的服务。
- 每一层内所包含的对象，彼此间相互独立，而处于不同层次上的对象，彼此间往往有关联。
- 在上、下层之间存在客户-供应商关系。下层相当于供应商，上层相当于客户。
- 层次结构又可进一步划分成两种模式：封闭式和开放式。

2. 组织系统的两种方案

(2) 块状组织

- 把软件系统垂直地分解成若干个相对独立的、弱耦合的子系统。
- 一个子系统相当于一块，每块提供一种类型的服务。

利用层次和块的各种可能的组合，可把多个子系统组成一个完整的软件系统。

<table>
<tr><td colspan="3">应 用 软 件 包</td></tr>
<tr><td rowspan="3">人机对话控制</td><td>窗口图形</td><td rowspan="3">仿真软件包</td></tr>
<tr><td>屏幕图形</td></tr>
<tr><td>像素图形</td></tr>
<tr><td colspan="3">操 作 系 统</td></tr>
<tr><td colspan="3">计 算 机 硬 件</td></tr>
</table>

混合使用层次与块状的的应用系统的组织结构

5.6 面向对象的设计方法

四、面向对象设计的特点

- 面向对象分析和设计的界限是模糊的，**从面向对象分析到面向对象设计是一个逐渐扩充模型的过程**。分析的结果通过细化直接生成设计结果，在设计过程中逐步加深对需求的理解，从而进一步完善需求分析的结果。
  - 将分析阶段产生的类图分配到相应的用例中。
  - 检查每个用例功能，依靠当前的类能否实现。
  - 细化每个用例的类图，并描述类之间的相互关系。
- 面向对象方法学在概念和表示方法上的一致性，保证了各个开发阶段之间的平滑性。

5.6 面向对象的设计方法

传统方法分析与设计之间的鸿沟

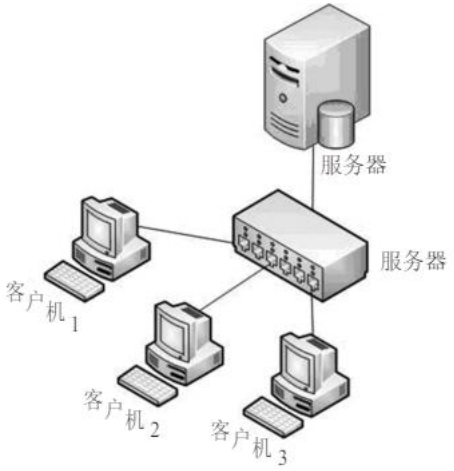
面向对象的分析与设计之间不存在鸿沟

5.6 面向对象的设计方法

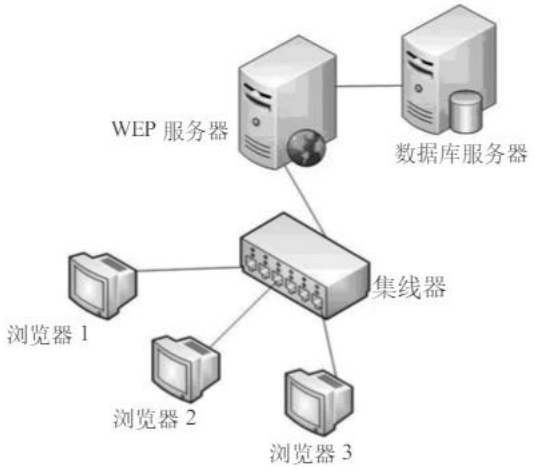
五、软件体系结构

□ Client/Server结构—C/S结构

- 将应用一分为二，服务器(后台)负责数据管理，客户机(前台)完成与用户的交互任务。充分利用两端硬件环境的优势，将任务合理分配到Client端和Server端来实现，降低了系统的开销。



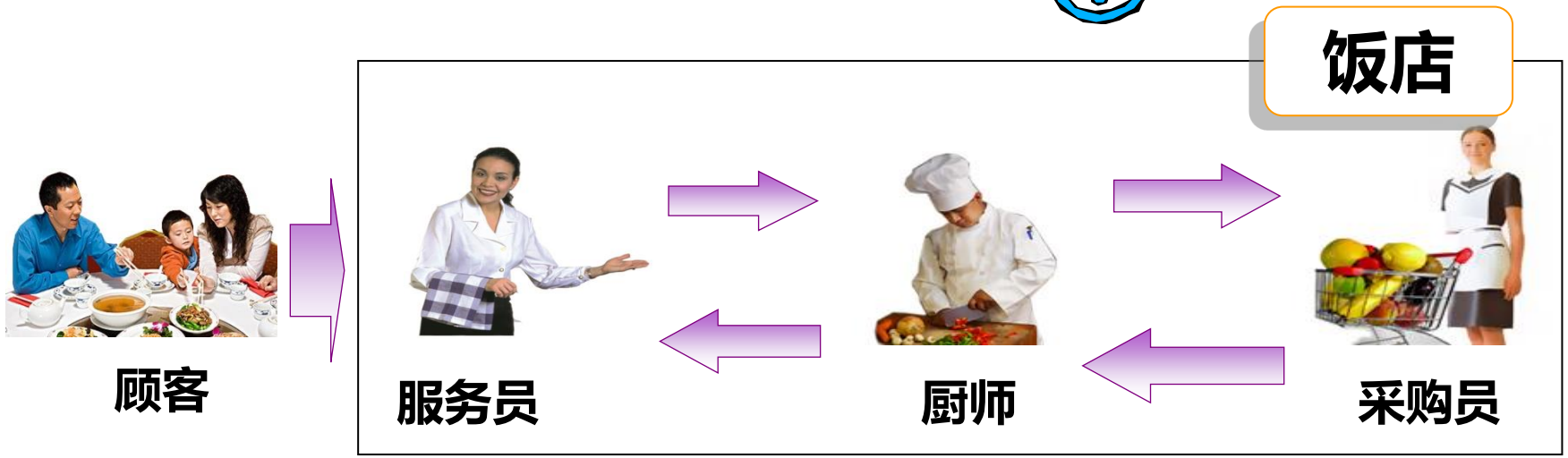
□ Browser/Server结构—B/S结构



5.6 面向对象的设计方法

□ 分层式的架构设计：三层架构

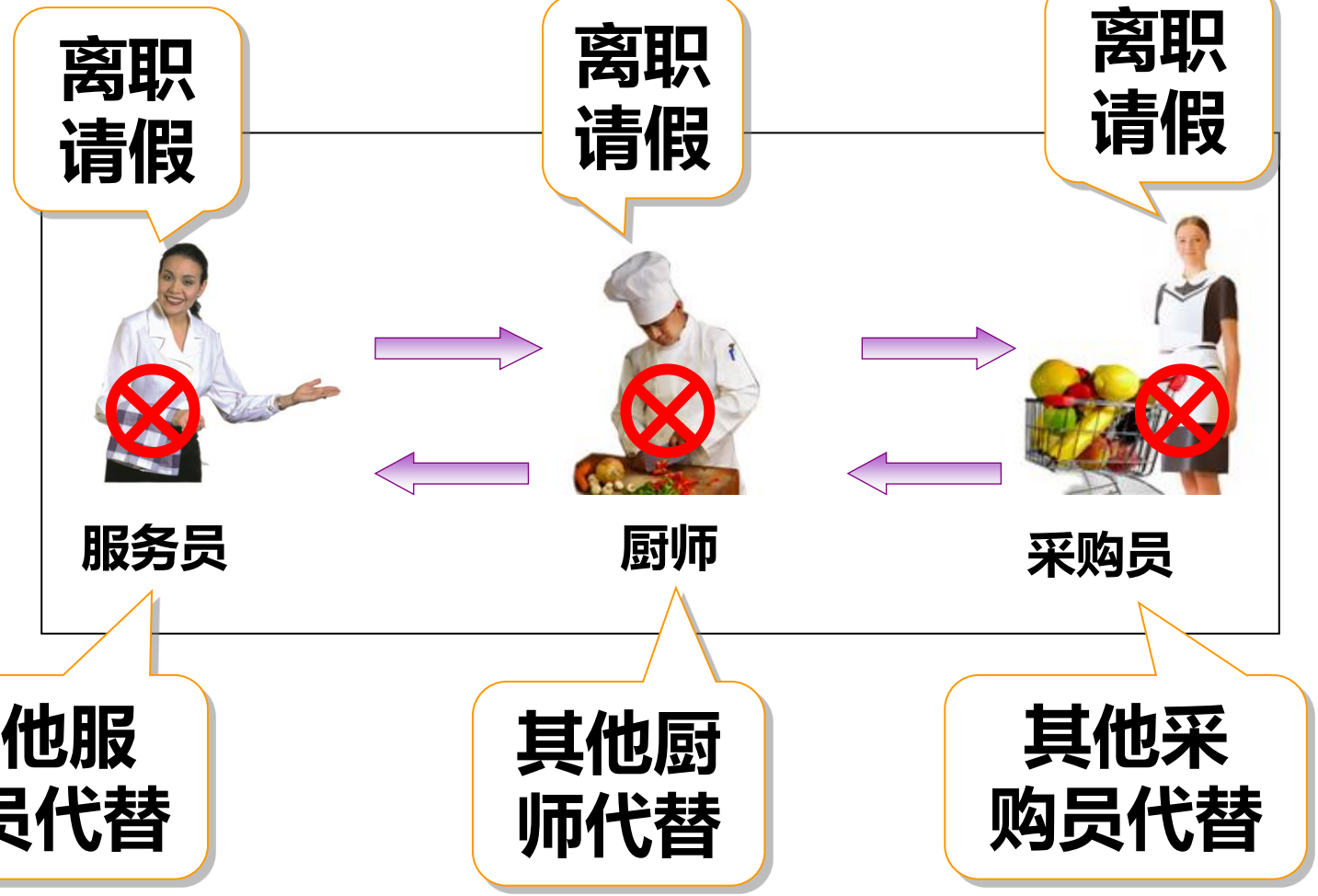
为什么需要三层架构？



服务员只管接待客人
厨师只管烹炒客人要的美食
采购员只管按客人需求采购肉，海鲜，蔬菜
他们各司其责**共同协作**为客人提供美食

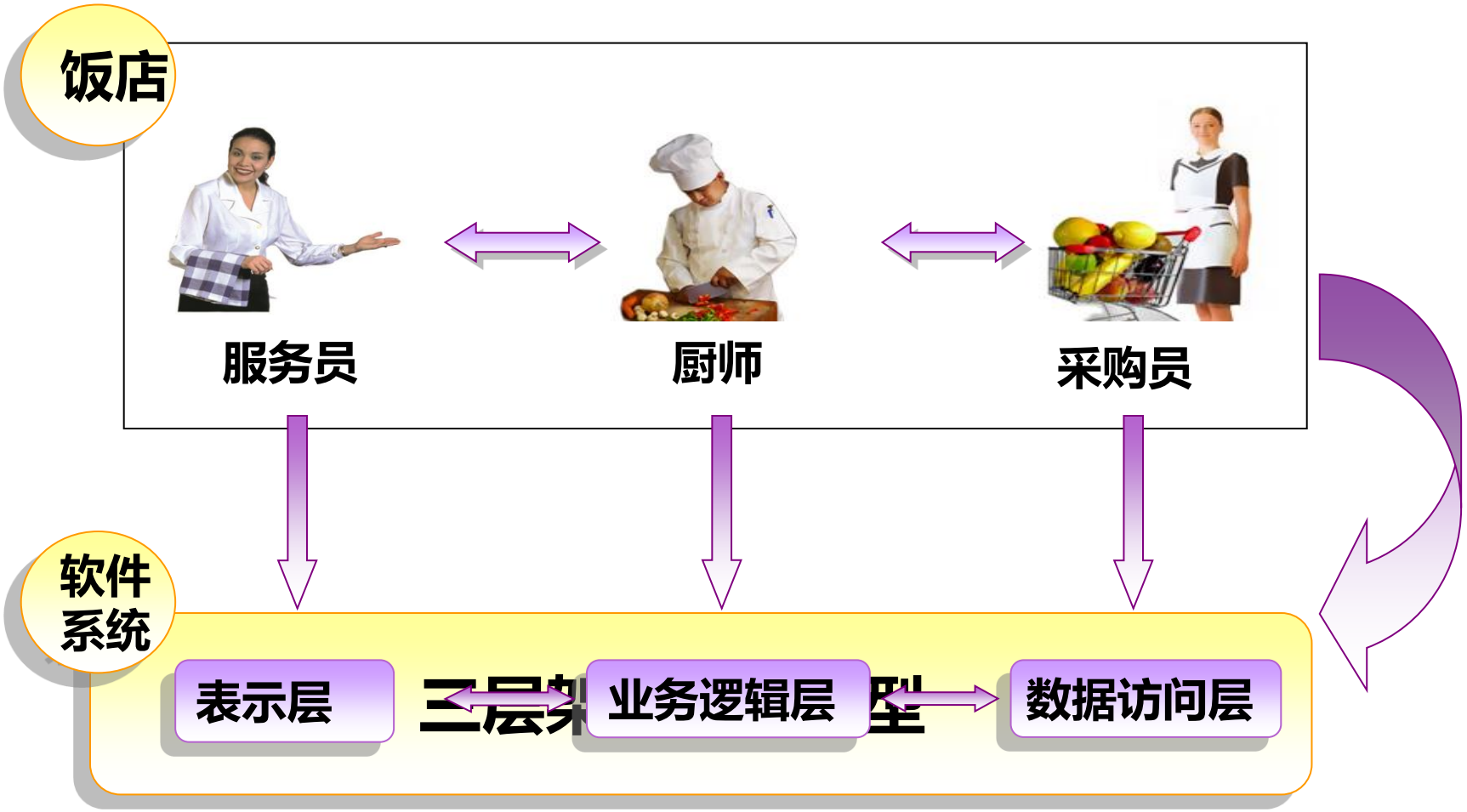
5.6 面向对象的设计方法

为什么需要三层架构？



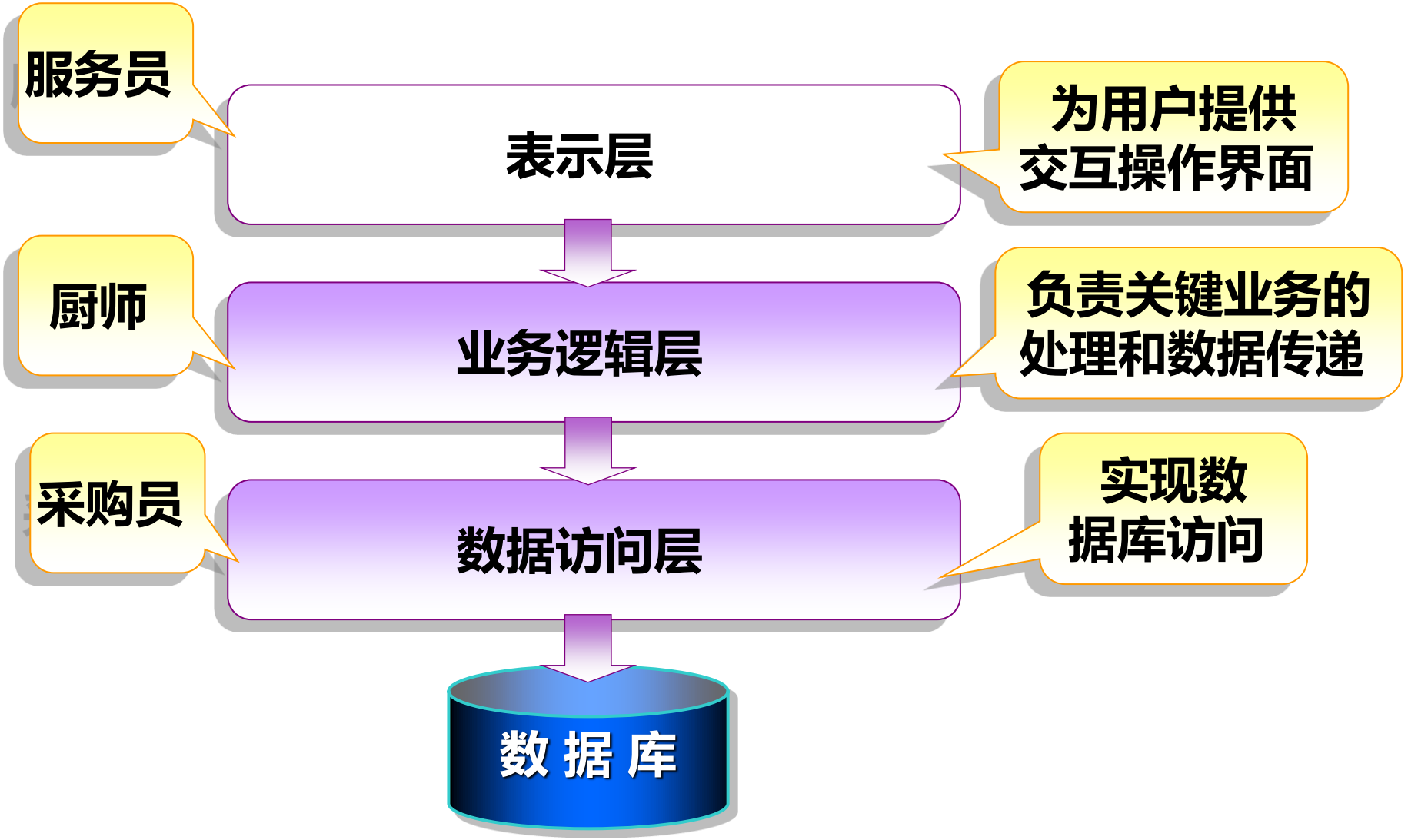
5.6 面向对象的设计方法

三层架构



5.6 面向对象的设计方法

什么是三层结构?



5.6 面向对象的设计方法

什么是三层结构?

1、表示层(UI)：通俗讲就是展现给用户的界面，用于显示数据和接收用户输入的数据，为用户提供交互式操作的界面。

2、业务逻辑层(BLL)：处理用户输入的信息或将信息发送给数据访问层保存，或从数据访问层读取数据。即对数据层的操作，对数据业务逻辑处理。

3、数据访问层(DAL)：直接操作数据，针对数据的增添、删除、修改、更新、查找等。

5.6 面向对象的设计方法

□ 表示层

表示层负责接收用户在界面输入的数据，然后传给业务逻辑层

为用户提供一种交互式操作界面

5.6 面向对象的设计方法

□ 业务逻辑层

业务逻辑层对数据进行处理，需要访问数据库的时候就向数据访问层发出访问数据的请求（调用数据访问层的函数）

```
//.....................
switch (type)
 {
      case "管理员":
         loginPwd = GetAdminLoginPwd(loginID);
         break;
      case "学员":
         loginPwd = GetStudentLoginPwd(loginID);
         break;
  }
//.....................
```

是表示层与数据访问层之间的桥梁，负责数据处理、传递。

5.6 面向对象的设计方法

□ 数据访问层

```
//.............
conn.Open();
SqlDataReader objReader = objCommand.ExecuteReader();
if (objReader.Read())
 {
    studentlist.Add(objReader["LoginPwd"]);
    studentlist.Add(objReader["UserStateId"]);
 }
objReader.Dispose();
conn.Close();
//...............
```

数据访问层访问数据库并获得需要的数据，比如正确的登录密码，然后返回给业务逻辑层。

实现对数据的保存和读取操作