第四部分 软件设计

- 4.1 软件工程开发方法与软件设计
- 4.2 体系结构设计
- 4.3 类/数据建模与设计
- 4.4 行为建模与设计
- 4.5 物理建模与设计

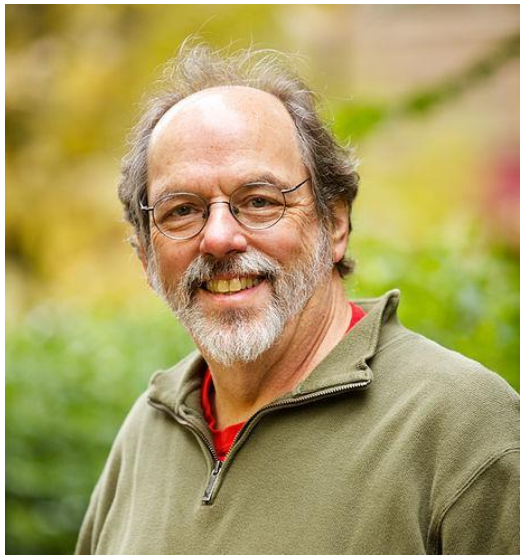
4.3 类/数据建模与设计

- CRC卡片分拣法-面向对象方法
- DFD-结构化方法

使用CRC卡片来辅助设计

- CRC Cards (class-responsibility-collaborator)
  - 1989年由Kent Beck和Ward Cunningham设计的CRC卡片是一种用来帮助设计人员在设计阶段前期进行设计的工具，此时设计人员应该做的事情就是：从问题域中找到合适的对象并理解它们之间的关系。

Kent Beck, 1961



Ward Cunningham,1949

识别类的方法

- 使用CRC寻找类。
  - CRC是类(Class)、责任(Responsibility)和协作(Collaboration)的简称，CRC分析法根据类所要扮演的职责来确定类。
  - 根据边界类、控制类和实体类的划分来帮助发现系统中的类
  - 对领域进行分析，或利用已有的领域分析结果得到类
  - 参考分析、设计模式来确定类

CRC卡片

Class：类定义	Classname 类名	
	Responsibili. es 类职责定义	Collabora.ons 交互协作关系定义

Blackjack游戏

- 21点游戏的目标是获得点数最接近或等于21的手牌
- 胜者需要赢过庄家的手牌，而且不能超过21点
- 无论桌上有几名玩家，总是只和庄家论输赢
- 21点游戏中，2-10的牌其点数就是牌面的数值
- A的点数根据玩家的需要可以取1或11
- J，Q，K的点数都是10

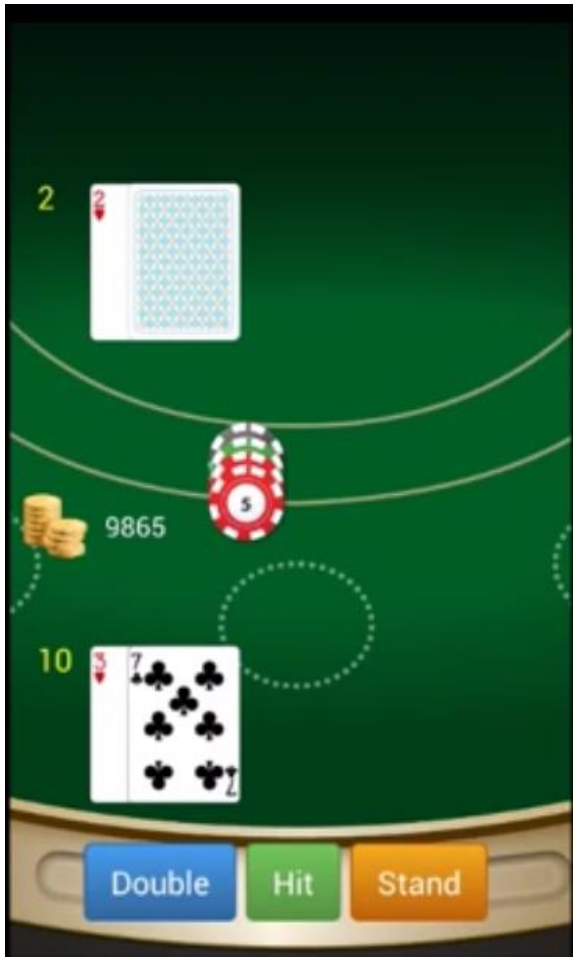
Blackjack游戏

- 游戏开始时，玩家首先下注，通常下注的额度是有上限的
- 如果你是唯一玩家，下注后，庄家给你发两张牌，牌面向上，双方均能看到
- 庄家给自己发两张牌，一张牌面向上（明牌），一张牌面向下（底牌）
- 如果有多个玩家，则庄家给每位玩家都是先发一张明牌，庄家自己发一张底牌，再给每个玩家和庄家各发一张明牌

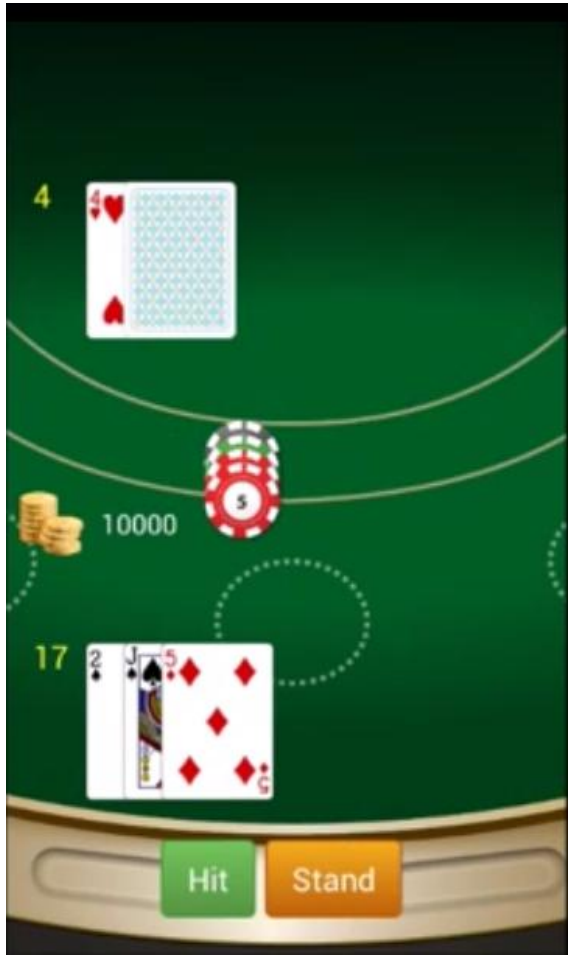
Blackjack游戏

- 发完两张牌之后，每位玩家自己决定是否要更多的牌，当对手上的牌数满意之后，就停手等待
- 停手之后，庄家开始加牌，不能超过21点
- 当你的手牌点数超过庄家时，你赢，当你的手牌点数不及庄家时，庄家赢
- 玩家赢：
  - 你的手牌点数超过庄家，且不超过21点
  - 庄家超过21点
- 玩家和庄家点数持平，则不输不赢，庄家将玩家赌注归还
- 玩家输：
  - 玩家超过21点

Blackjack游戏



Blackjack游戏



识别21点游戏中的对象类

- 类对象通常对应一个命名实体，因此，用名词表达，于是我们先从名词开始

- 切记：Don't try to get things right the first time…

对象几乎无处不在

- 外部实体
  - 与建模中的系统存在交互
  - 例如：人，设备和其他系统
- 事物
  - 建模的应用领域中存在的事物
  - 例如：报表，信号，文字输出
- 事件
  - 系统上下文中发生
  - 例如：资源传递，控制命令发出
- 角色
  - 由与系统交互的人扮演的角色
- 组织单元
  - 与应用领域相关的部分
  - 例如：分支机构、群组，团队
- 位置、地点
  - 建模中的问题的物理上下文
  - 例如：厂房、车间、货架
- 结构体
  - 定义类或者对象组合
  - 例如：传感器、车辆、计算机

不能定义为对象的事物：
- 过程（打印、转换）
- 属性（红颜色，50Mb）

Blackjack游戏

- 21点游戏的目标是获得点数最接近或等于21的手牌
- 胜者需要赢过庄家的手牌，而且不能超过21点
- 无论桌上有几名玩家，你总是只和庄家论输赢
- 21点游戏中，2-10的牌其点数就是牌面的数值
- A的点数根据玩家的需要可以取1或11
- J，Q，K的点数都是10

Blackjack游戏

- 游戏开始时，玩家首先下注，通常下注的额度是有上限的
- 如果你是唯一玩家，下注后，庄家给你发两张牌，牌面向上，双方均能看到
- 庄家给自己发两张牌，一张牌面向上（明牌），一张牌面向下（底牌）
- 如果有多个玩家，则庄家给每位玩家都是先发一张明牌，庄家自己发一张底牌，再给每个玩家和庄家各发一张明牌

通过名词过滤识别出的对象类

- 游戏
- 21点
- 庄家
- 玩家
- 玩家们
- 牌
- 牌堆
- 手牌
- 点数
- 花色
- 赢家
- A
- 数字牌
- K
- Q
- J
- 赌注

类筛选

- 在候选类中排除以下类：
  - 超出问题关注的范围的类；
  - 指代整个系统的类；
  - 功能重复的类；
  - 过于含糊或过于具体的类；
  - 可观察到的现象是，实例对象过多过少。

- Coad & Yourdon's 的筛选原则：
  - 保存对象信息：系统需要保存对象信息吗？
  - 提供所需服务：类对象是否对外提供修改属性值的操作？
  - 具有多个属性：只有一个属性的类，应该建模为属性；
  - 具有公共属性：类属性是否为所有实例对象共享?
  - 具有公共操作：类操作是否为所有实例对象共享?
  - 外部实体：如果生产或使用对象的信息，也应考虑建模为系统类。

确定初始类（Class）

牌　　牌堆　　手牌

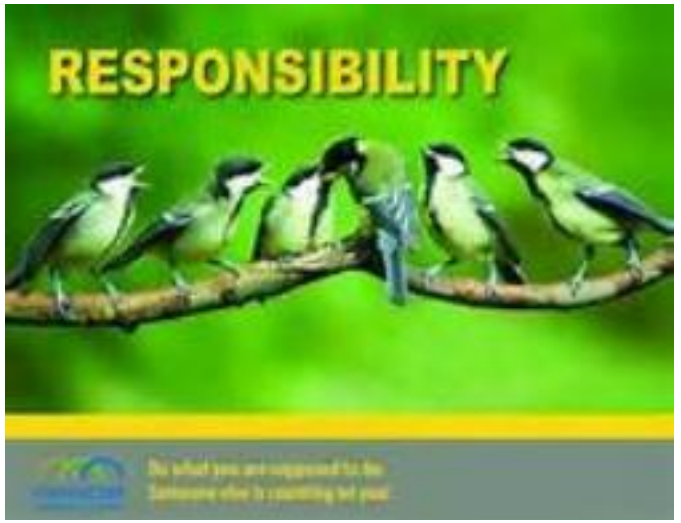
庄家　　玩家　　赌注

识别类的功能职责（Responsibility）

功能职责关乎行为动作，因此是问题描述中的动词...

- 注意：

1. 并非所有动词均将成为类职责
2. 有时多个动作合并为一个职责
3. 随着分析过程深入会发现新的职责
4. 不断修正类定义和职责定义
5. 当两个类分享职责时，为二者同时添加该职责



Blackjack游戏

- 游戏开始时，玩家首先下注，通常下注的额度是有上限的
- 如果你是唯一玩家，下注后，庄家给你发两张牌，牌面向上，双方均能看到
- 庄家给自己发两张牌，一张牌面向上（明牌），一张牌面向下（底牌）
- 如果有多个玩家，则庄家给每位玩家都是先发一张明牌，庄家自己发一张底牌，再给每个玩家和庄家各发一张明牌

Blackjack游戏

- 发完两张牌之后，每位玩家自己决定是否要更多的牌，当对手上的牌点数满意之后，就停手等待
- 停手之后，庄家开始加牌，不能超过21点
- 当你的手牌点数超过庄家时，你赢，当你的手牌点数不及庄家时，庄家赢
- 玩家赢：
  - 你的手牌点数超过庄家，且不超过21点
  - 庄家超过21点
- 玩家和庄家点数持平，则不输不赢，庄家将玩家赌注归还
- 玩家输：
  - 玩家超过21点

候选类对应的功能职责

- 牌 （Card）
  - 读取牌面数字
  - 读取花色
  - 读取点数
  - 判断是否数字牌
  - 判断是否为A
  - 判断是否为 J、Q、K
- 赌注（Bet）
  - 赌注类型
  - 赌注的大小
  - 剩余赌资
  - 判断赌资是否够下注
- 牌叠 （Deck）
  - 洗牌
  - 发牌
  - 查询剩余牌数
  - 是否能够新开一叠牌
- 手牌（Hand）
  - 读取手中牌数
  - 读取手牌总点数
  - 亮牌

候选类对应的功能职责

- 庄家(Dealer)
  - 发牌
  - 洗牌
  - 发一张牌给玩家
  - 庄家亮牌
  - 计算庄家手牌点数
  - 查询庄家手牌点数
  - 加一张牌
  - 确定赢家
  - 开始下一轮游戏
- 玩家(Player)
  - 请求发牌
  - 玩家亮牌
  - 计算玩家手牌点数
  - 查询玩家手牌点数
  - 查看手牌点数是否超过21点
  - 查看手牌点数是否等于21点
  - 查看手牌点数是否低于21点

识别类交互协作关系

识别对象及其消息交互

- 注意:目的并非写出所有场景，而是对类和职责定义进行精化

情景举例

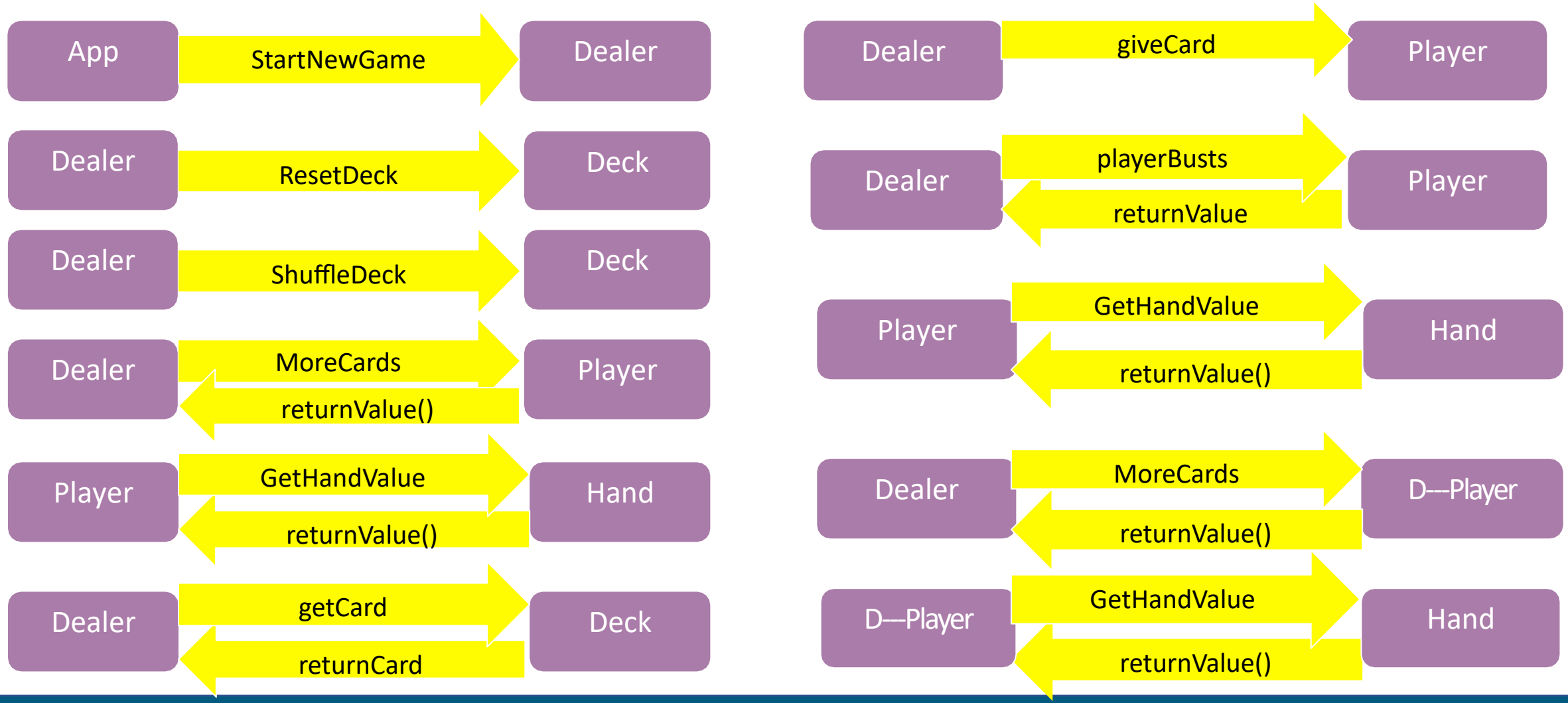
庄家

- 庄家洗牌
- 庄家发第一张牌
- 庄家给自己发牌
- 手牌返回庄家手牌点数
- 庄家发第二张牌
- 庄家询问玩家是否继续要牌
- 庄家询问玩家手牌点数
- 庄家亮牌
- 庄家分配赌金

玩家

- ➜ 玩家下注
- ➜ 玩家拿牌到手里
- ➜ 手牌返回玩家手牌点数
- ➜ 玩家继续要牌
- ➜ 玩家拿牌到手里
- ➜ 手牌返回玩家手牌点数
- ➜ 玩家增加或减少赌注
- ➜ 玩家亮牌

识别交互关系


App
StartNewGame
Dealer
Dealer
ResetDeck
Deck
Dealer
ShuffleDeck
Deck
Dealer
MoreCards
returnValue()
Player
Player
GetHandValue
returnValue()
Hand
Dealer
getCard
returnCard
Deck
Dealer
giveCard
Player
Dealer
playerBusts
returnValue
Player
Player
GetHandValue
returnValue()
Hand
Dealer
MoreCards
returnValue()
D---Player
D---Player
GetHandValue
returnValue()
Hand

类定义：纸牌类（Card）

类名（Class）	纸牌 Card
功能职责(Responsibilities) GetName GetValue	交互类(Collaborations) Deck

类定义：牌叠类（Deck）

类名（Class）	牌堆Deck

功能职责(Responsibili. es)	交互类 （Collabora.ons）
Reset deck	Dealer
Get deck size	Card
Get next card	
Shuffle Deck	
Show deck	

类定义：庄家（Dealer）

类名（Class）	庄家 Dealer
功能职责(Responsibili. es)	交互类 （Collabora.ons）
Start a new game	Hand
Get a card	Player
	Deck

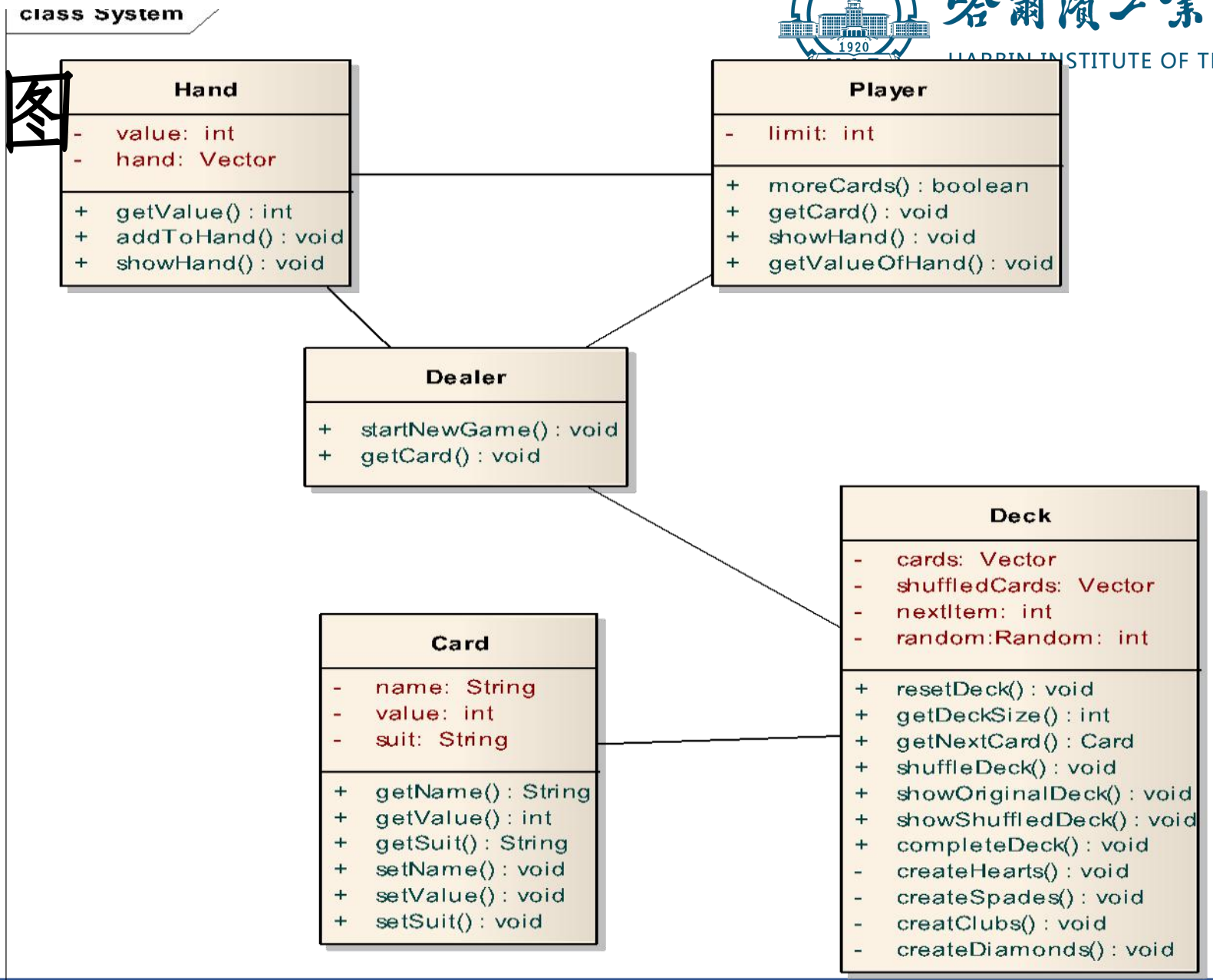
类定义：玩家（Player）

类名（Class）	玩家 Player
功能职责(Responsibili. es)	交互类 （Collabora.ons）
Want more cards	Hand
Get a card	Dealer
Show hand	
Get value of hand	

类定义：手牌类（Hand）

类名（Class）	手牌 Hand
功能职责(Responsibili. es)	交互类 （Collabora.ons）
Return Value Add a card Show hand	Player Dealer

UML之类图


class System
Hand
- value: int
- hand: Vector
+ getValue() : int
+ addToHand() : void
+ showHand() : void
Player
- limit: int
+ moreCards() : boolean
+ getCard() : void
+ showHand() : void
+ getValueOfHand() : void
Dealer
+ startNewGame() : void
+ getCard() : void
Deck
- cards: Vector
- shuffledCards: Vector
- nextItem: int
- random:Random: int
+ resetDeck() : void
+ getDeckSize() : int
+ getNextCard() : Card
+ shuffleDeck() : void
+ showOriginalDeck() : void
+ showShuffledDeck() : void
+ completeDeck() : void
- createHearts() : void
- createSpades() : void
- creatClubs() : void
- createDiamonds() : void
Card
- name: String
- value: int
- suit: String
+ getName() : String
+ getValue() : int
+ getSuit() : String
+ setName() : void
+ setValue() : void
+ setSuit() : void

4.3 类/数据建模与设计容

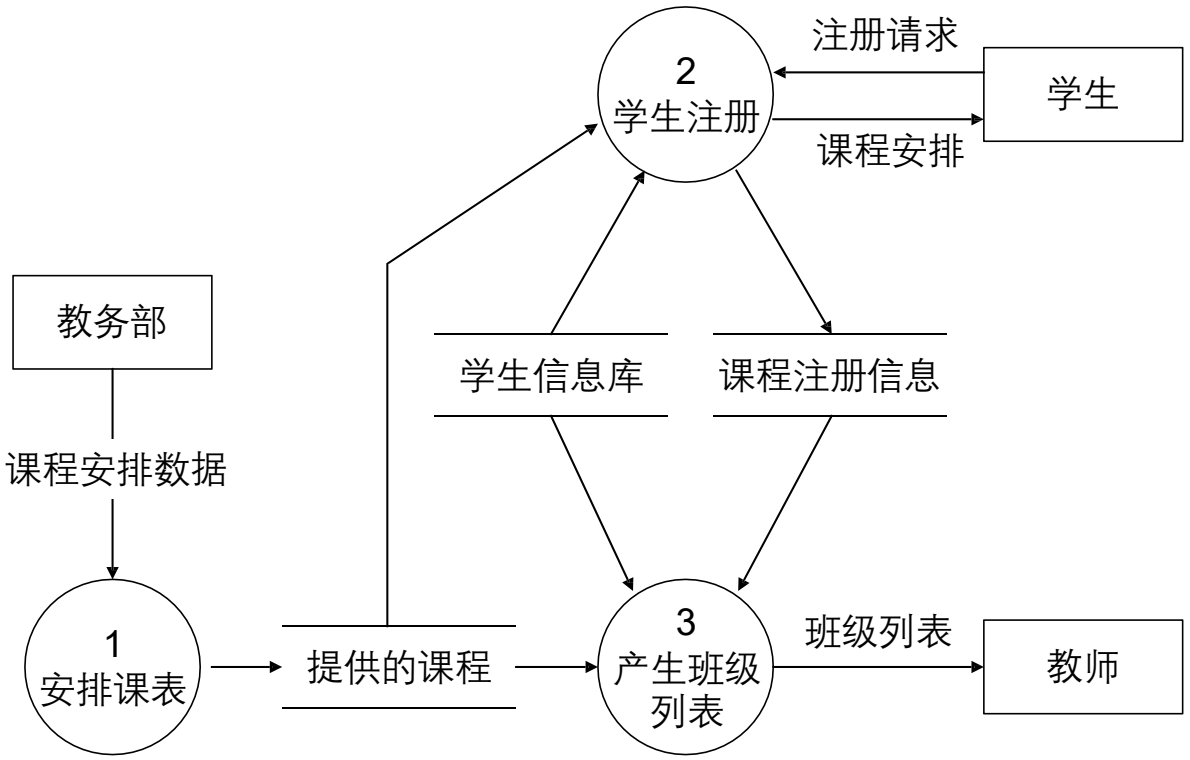
- CRC卡片分拣法-面向对象方法
- DFD-结构化方法

结构化方法的模型

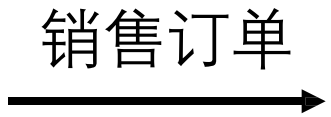
- 结构化需求分析方法通常需建立以下模型：
  - 数据流图(Data Flow Diagram, DFD)
    - 描述系统由哪些部分组成、各部分之间有什么联系等
  - 数据字典(Data Dictionary, DD)
    - 定义了数据流图中每一个数据元素
  - 结构化语言(Structured Language)
  - 判定表或判定树(Decision Table/Tree)
    - 详细描述数据流图中不能被再分解的每一个加工的内部处理逻辑
  - 实体联系图(Entity-Relationship Diagram, E-R)
  - 状态转换图(State Transition Diagram, STD)

数据流图(DFD)

- **数据流图(Data Flow Diagram, DFD)：结构化系统分析的基本工具**
  - 描绘数据在系统中各逻辑功能模块之间的流动和处理过程，是一种功能模型
  - 主要刻画“功能的输入和输出数据”、“数据的源头和目的地”



DFD的主要元素



数据流

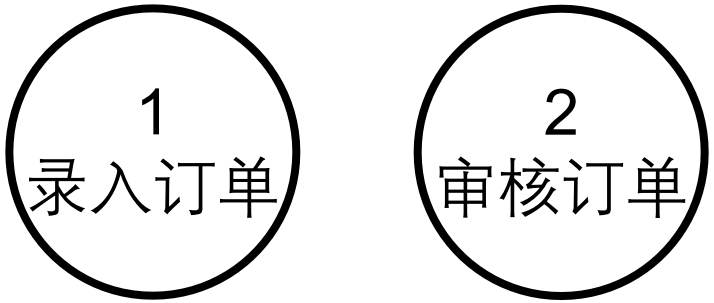
加工

数据存储

外部实体

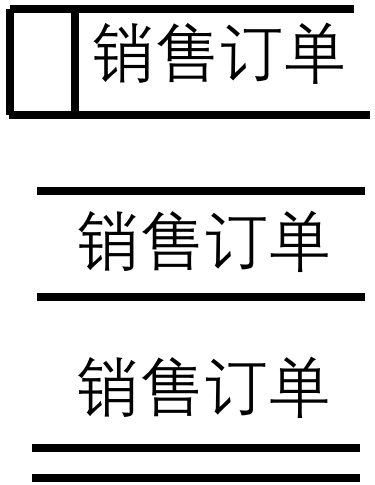
DFD的主要元素(1)：加工

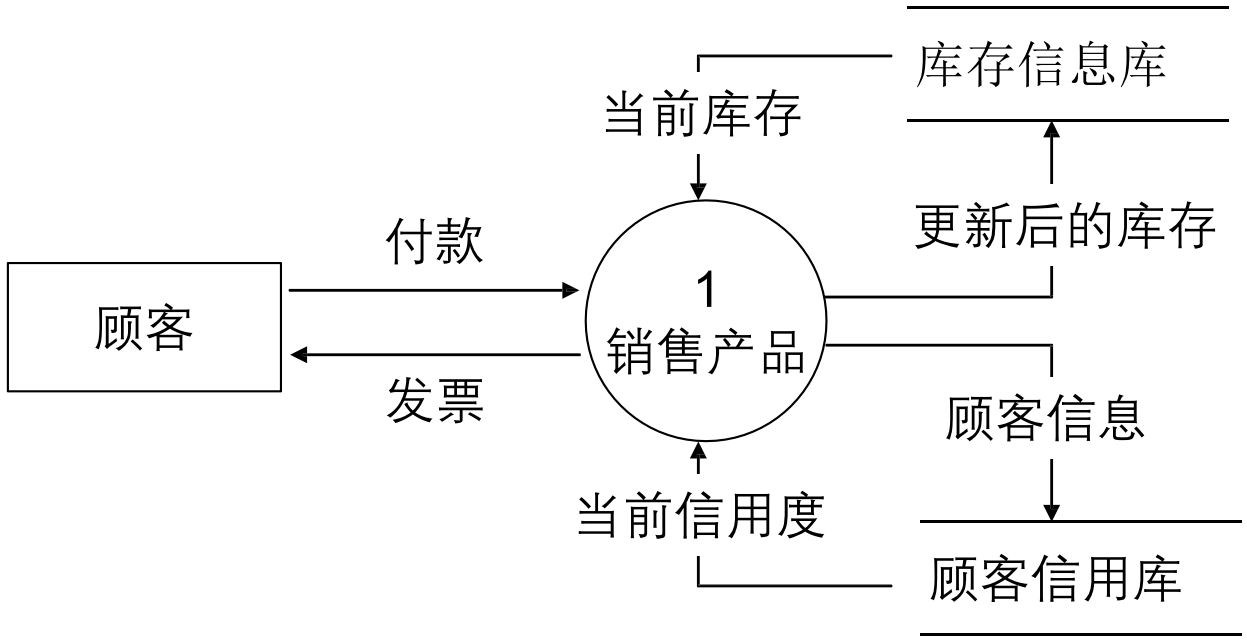
- **加工(又称数据处理，data processing)：对数据流进行某些操作或变换。**
  - 收集、排序、选择、聚集、分析等
  - 加工要有名字，通常是动词短语，简明地描述完成什么事情
  - 在分层的数据流图中，加工还应编号
  - 三种类型：计算机自动加工、手工加工、人机协作的加工



DFD的主要元素(2)：数据存储

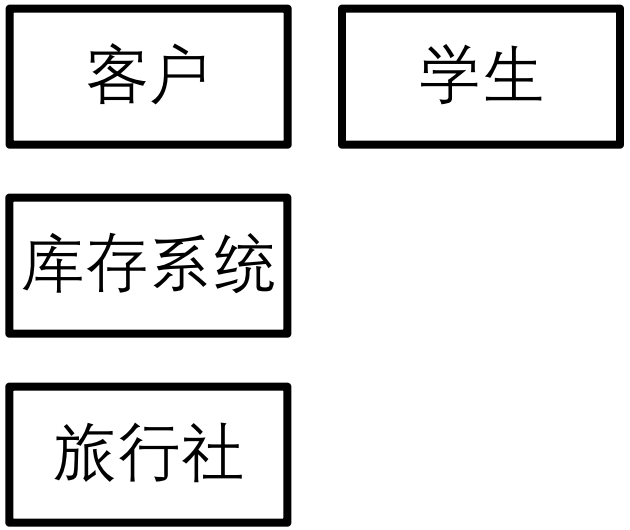
- **数据存储(data storage，也称文件)：需要在外存储器上保存的数据，它可以是数据库文件或任何形式的数据组织。**
  - 以名词命名





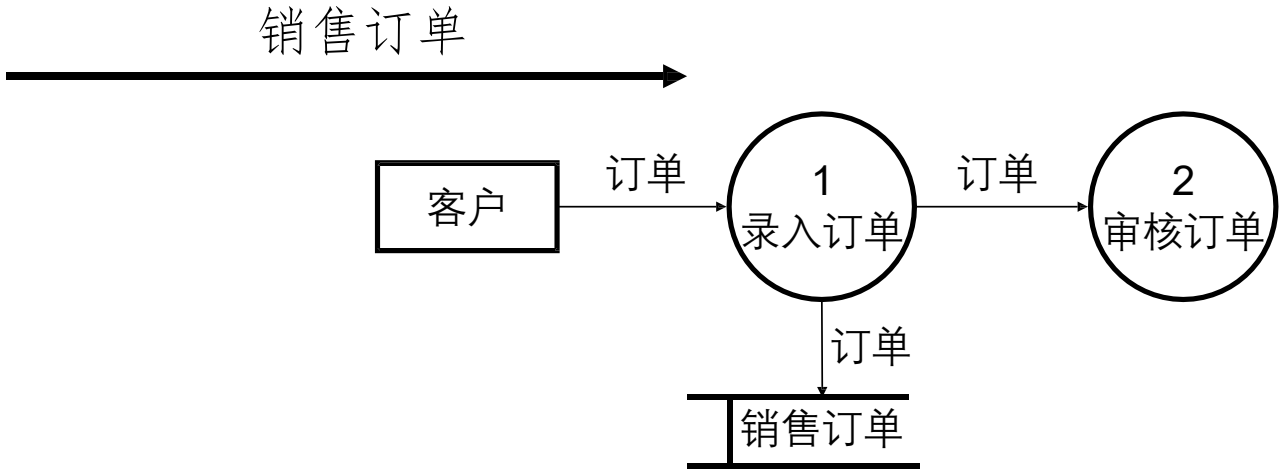
DFD的主要元素(3)：外部实体

- **外部实体(external entity)：本系统外部环境中的实体(包括人员、组织或其他软件系统)**
  - 也称为“数据源点/数据终点”，表示产生数据的源头或消费数据的终点
  - 以名词短语命名
  - 不能直接访问数据存储



DFD的主要元素(4)：数据流

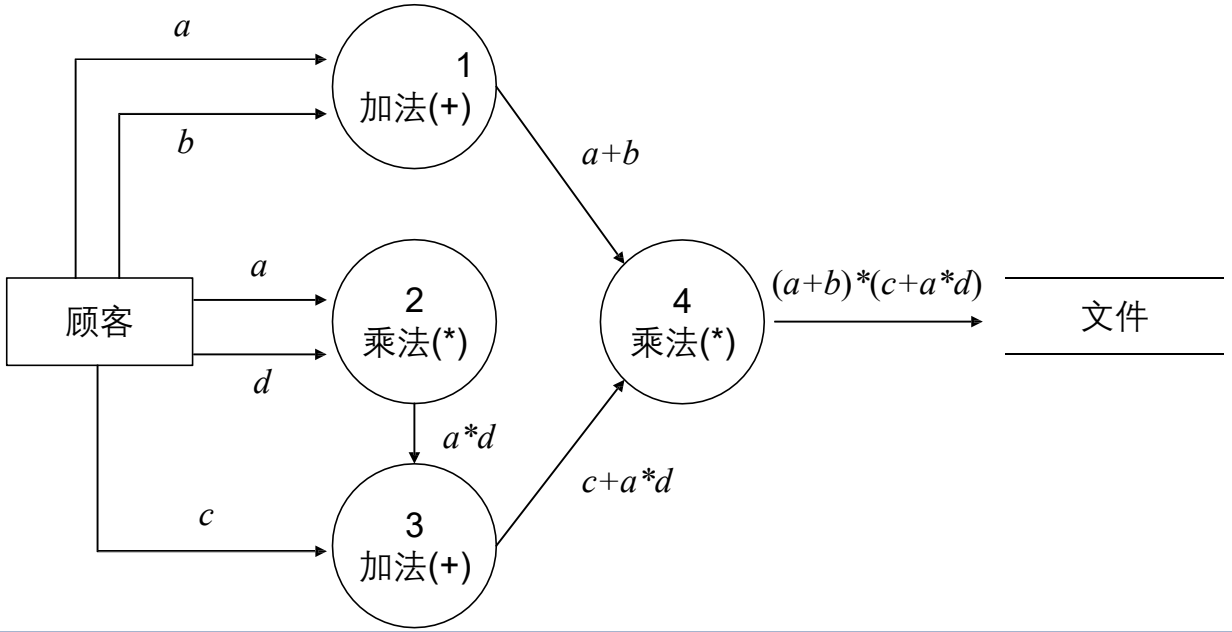
- 数据流(data flow)：数据在系统内传播的路径
  - 由一组成分固定的数据组成。
  - 由于数据流是流动中的数据，所以必须有流向
  - 应用名词或名词短语命名
  - 可能是纸张上的数据、电子数据、通过网络传输的数据等



  - 可能存在于：
    - 外部实体与加工之间
    - 加工与加工之间
    - 加工与数据存储之间

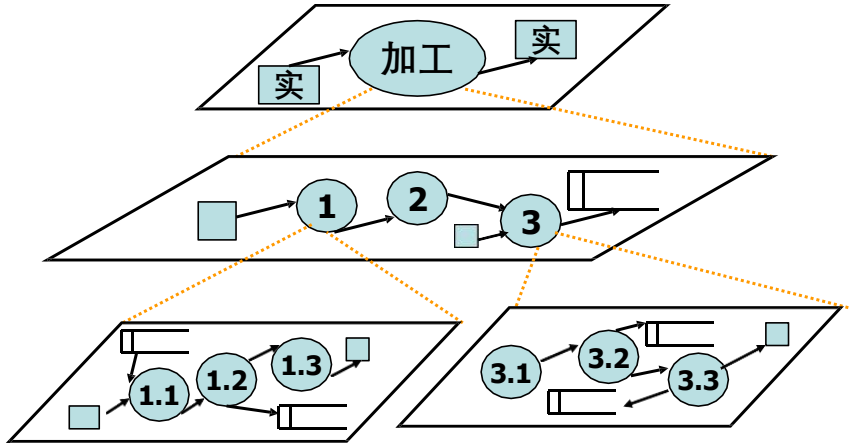
DFD的简单练习

- 背景：用户输入a、b、c、d四个值，系统计算(a+b)*(c+a*d)，并将结果输出到一个文件中存储。
- 问题：绘制该系统的DFD



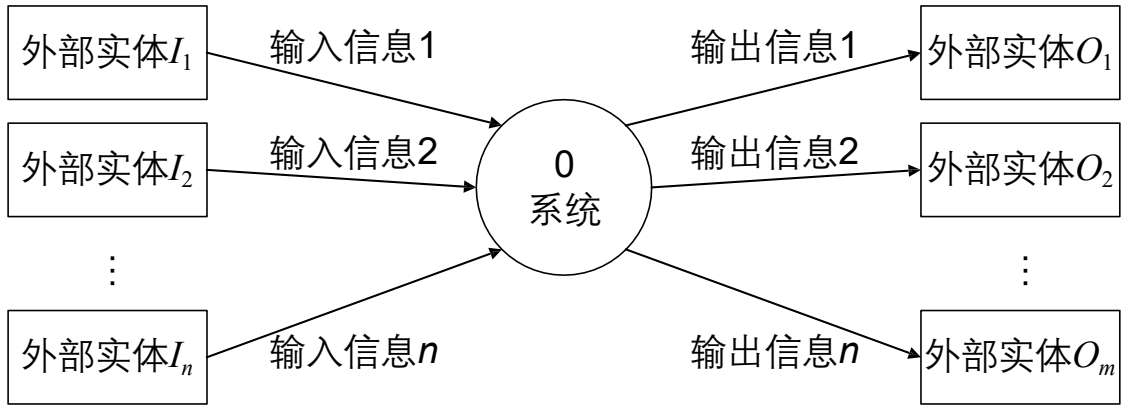
DFD的层次性

- DFD的层次性：自顶向下的分解(top-down)
- DFD的两种类型：
  - 环境关联DFD图(Context-level DFD，或Context Diagram)：也称顶层DFD图，描述了系统与外部环境之间的数据输入/输出关系；
  - 系统内部DFD图(Inner-level DFD)：描述系统内部各功能模块之间的数据流动关系
    - 0-层DFD图
    - 1-层DFD图
    - …
    - N层DFD图

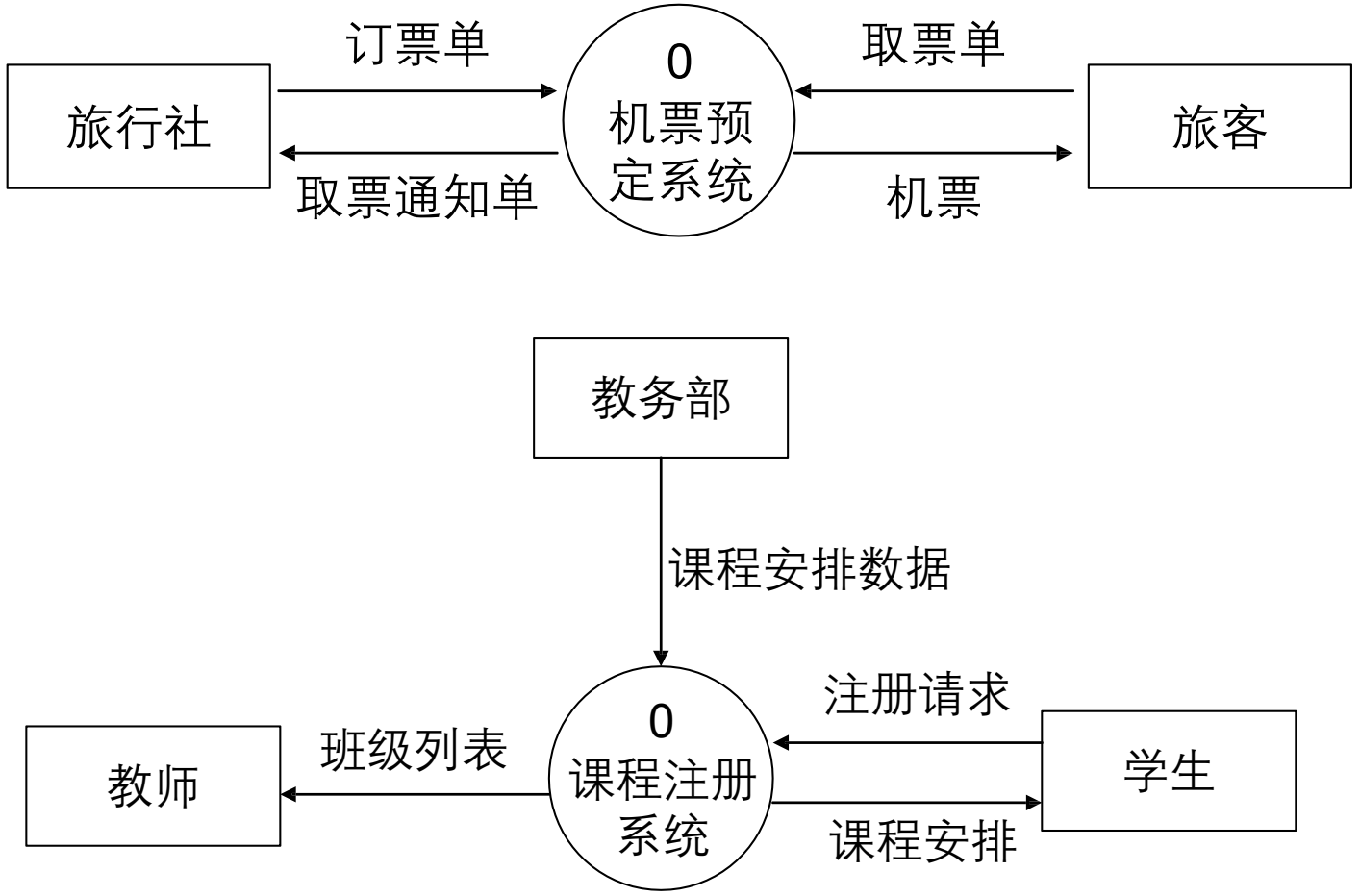


顶层DFD

- **顶层DFD图(关联图)**
  - 通过系统和外部世界之间的联系来描述系统的范围
  - 确定了通过某一接口与系统相连的外部实体，同时也确定了外部实体和系统之间的数据流
  - 只包含一个加工，用以表示被开发的系统，然后考虑该系统有哪些输入数据、输出数据流
  - 加工编号：0

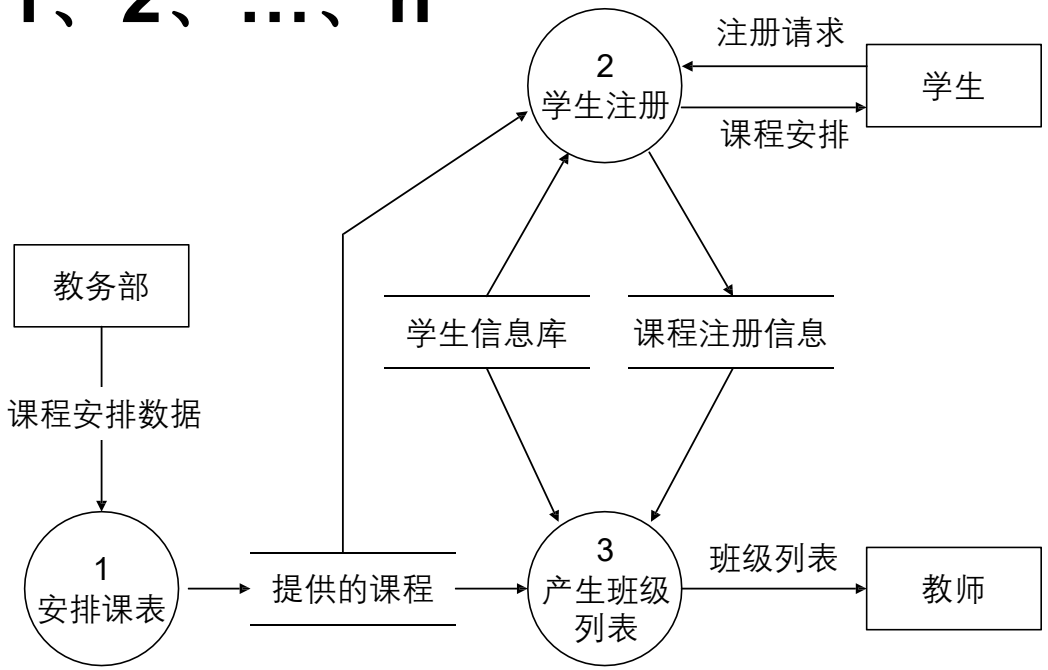


示例：顶层DFD


旅行社
订票单
取票通知单
0
机票预
定系统
取票单
机票
旅客
教务部
课程安排数据
教师
班级列表
0
课程注册
系统
注册请求
课程安排
学生

0层DFD

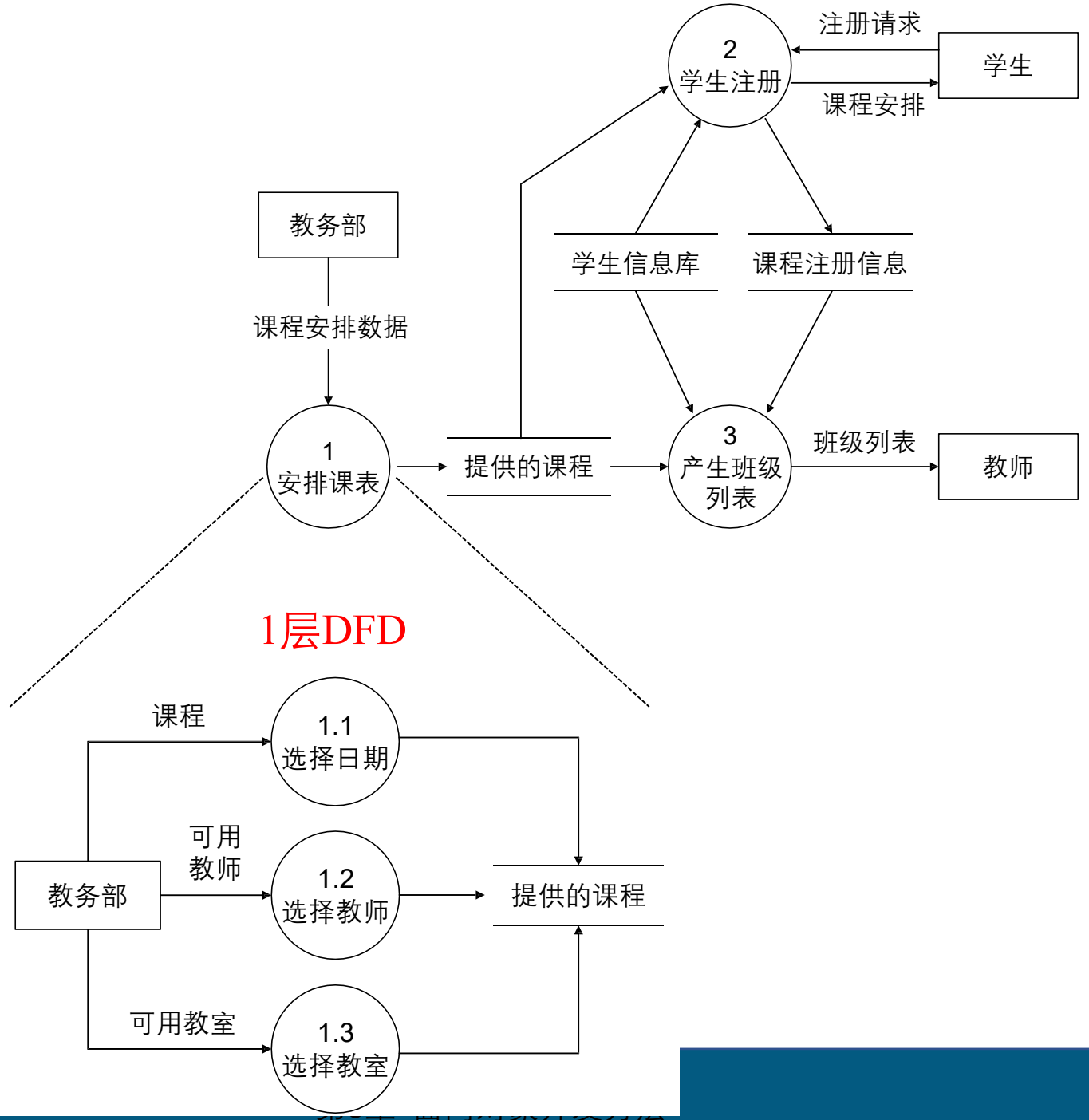
- 将顶层**DFD**图中的系统分解为若干个子系统，决定每个子系统间的数据接口和活动关系，得到**0层DFD图；**
- 加工编号：**1、2、…、n**



底层DFD

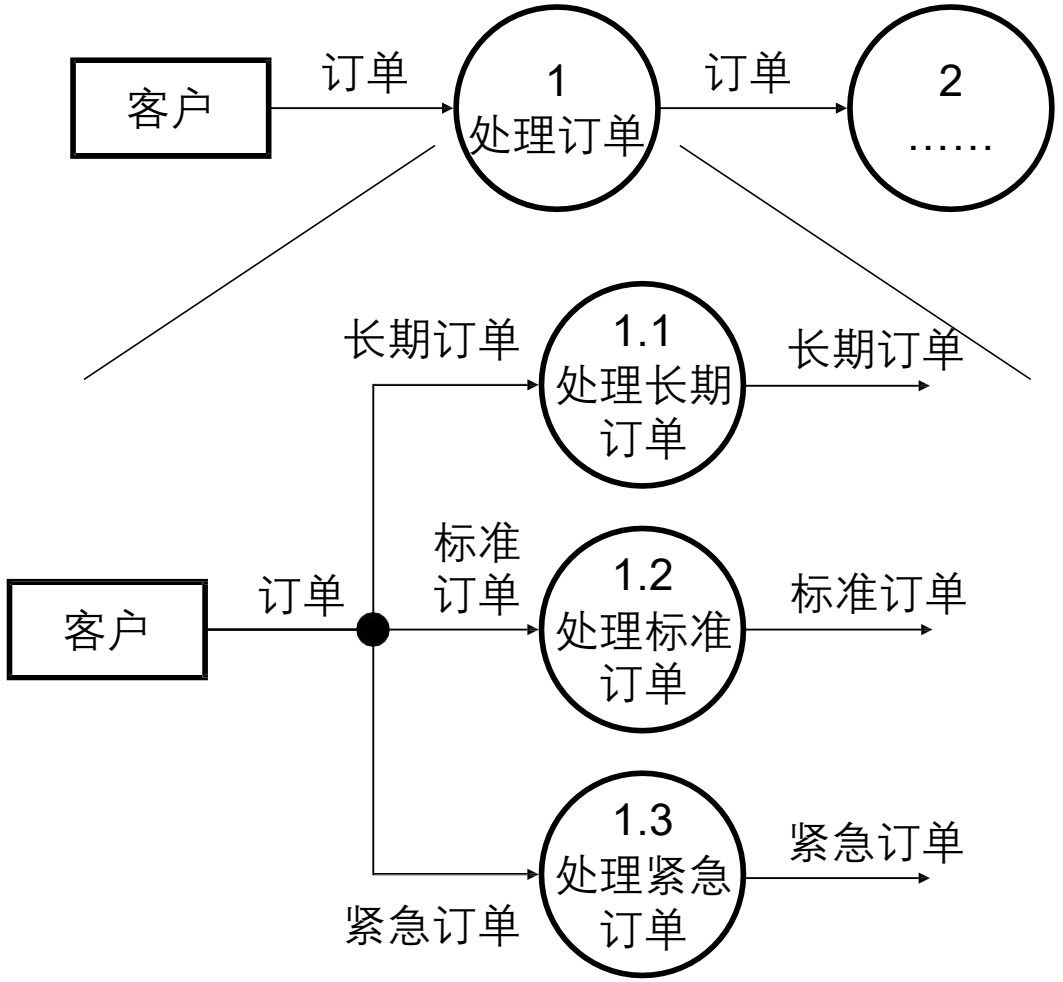
- 针对**0层DFD**中的每一个子系统，对其继续分解得到细化的加工，进而逐渐向下构造得到**1层DFD、2层DFD、…、n层DFD**，一直到不能或不需再分解为止。
- 最底层**DFD**中的加工称为“基本加工”。

- 编号：
  - 1层DFD：1.1、1.2、…、1.n
  - 2层DFD：1.1.1、1.1.2、…、1.1.n
  - …


2
学生注册
注册请求
学生
课程安排
教务部
学生信息库
课程注册信息
课程安排数据
1
安排课表
提供的课程
3
产生班级
列表
班级列表
教师
1层DFD
课程
1.1
选择日期
可用
教师
教务部
1.2
选择教师
提供的课程
可用教室
1.3
选择教室

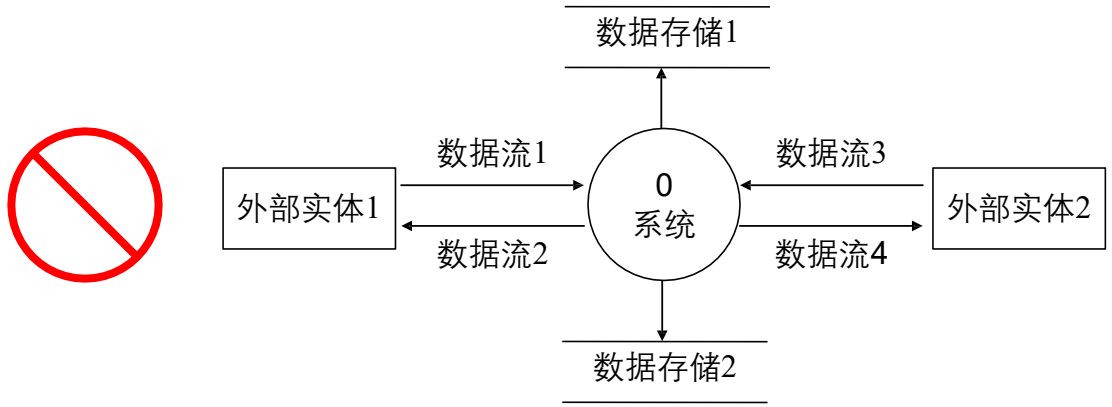
底层DFD

数据流的分解

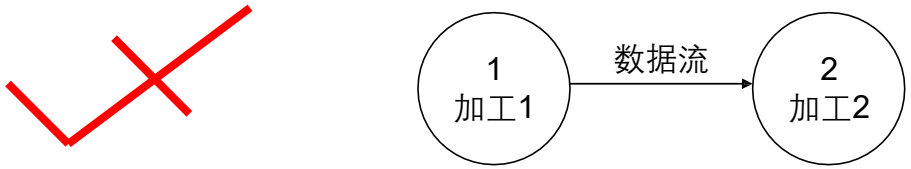

客户
订单
1
处理订单
订单
2
……
长期订单
1.1
处理长期
订单
长期订单
客户
订单
标准
订单
1.2
处理标准
订单
标准订单
1.3
处理紧急
订单
紧急订单
紧急订单

绘制DFD的一些基本原则

- 把数据存储放在**0**层数据流图或更低层子图上，不要放在顶层的关联图上

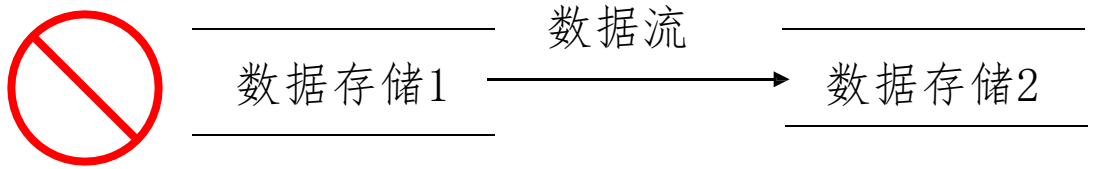


- 使用数据流图时，不要试图让数据流图反映处理的顺序，忽略系统的运行时的时间特性
- 加工通过数据存储进行通讯，而尽量避免从一个过程直接流到另一过程

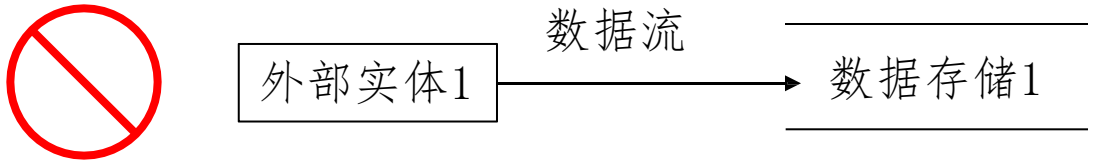


绘制DFD的一些基本原则

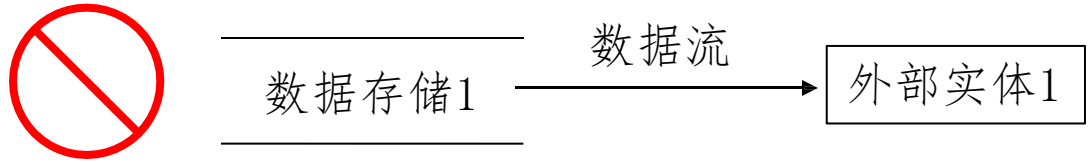
- 数据不能直接由一个数据存储直接流到另一个数据存储



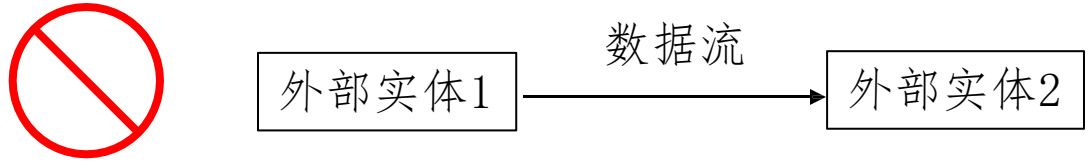
- 数据不能直接从一个外部实体直接流到一个数据存储



- 数据不能直接从一个数据存储直接流到一个外部实体

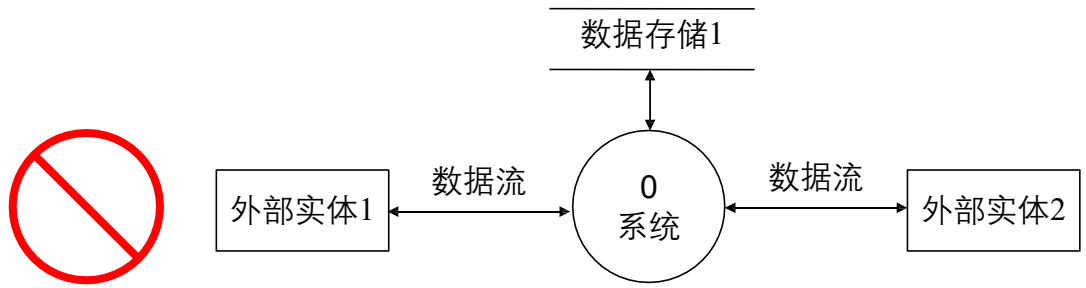


- 数据不能直接在外部实体之间流动

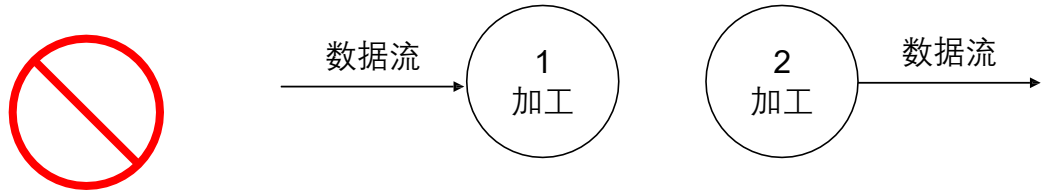


绘制DFD的一些基本原则

- **数据流是单向的**



- **任何加工必须有输入和输出数据流**



- **对现有加工进行持续的分解和组合，直到所有加工之间达到较高的聚合度**
- **尽量将每一张DFD上的所有元素数目控制在7-12个**

错误的DFD

- 找出下面DFD中存在的错误，并说明如何修改。