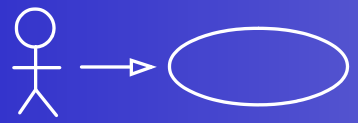
软件需求设计UML全程实作

分析

潘加宇



建模工作流

*业务建模
- 愿景
- 业务用例图
- 现状业务序列图
- 改进业务序列图

*需求
- 系统用例图
- 系统用例规约

需求

提升销售

---

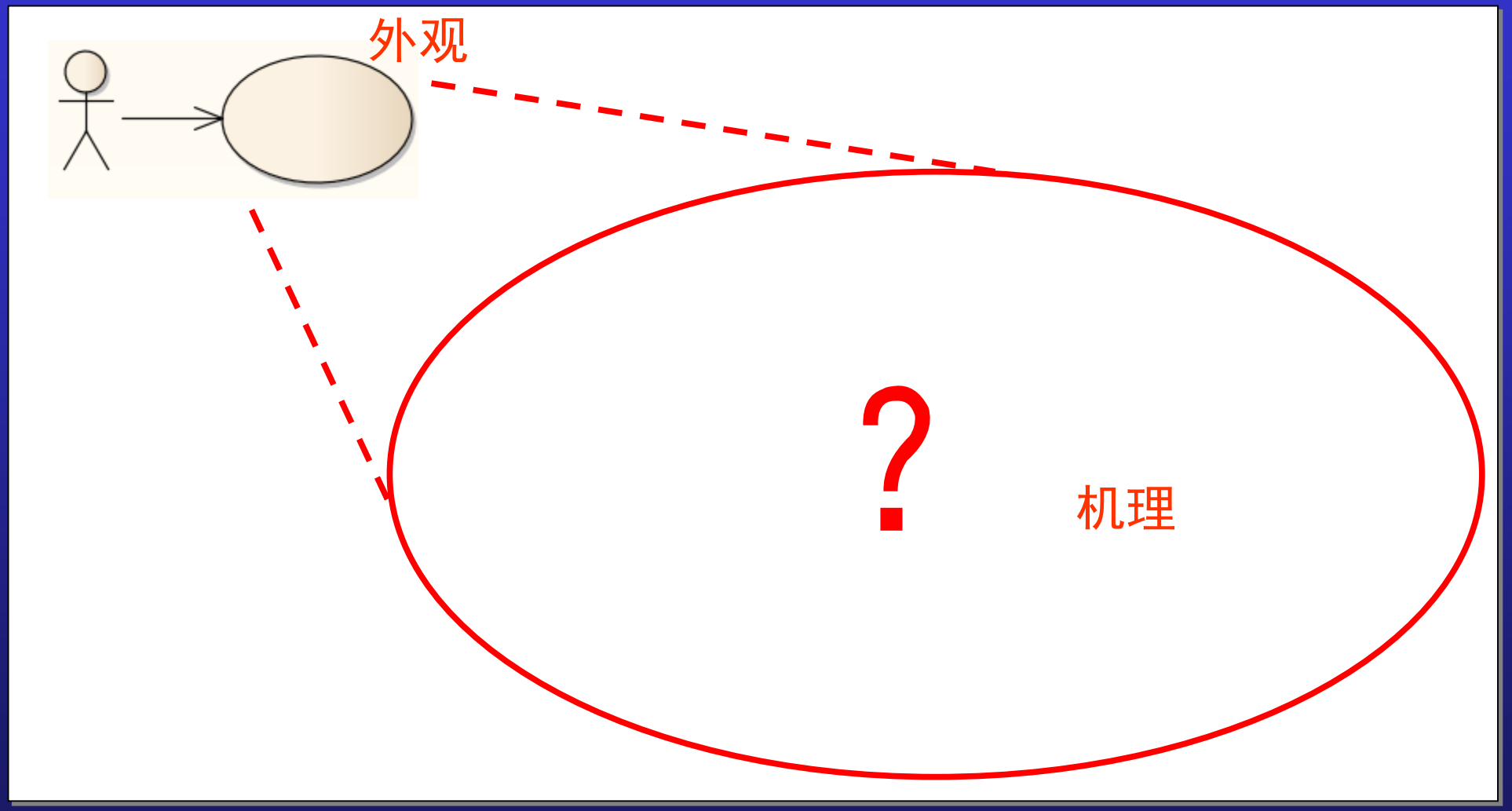
*分析
- 分析类图
- 分析序列图
- 分析状态机图

*设计
- 建立数据层
- 精化业务层
- 精化表示层

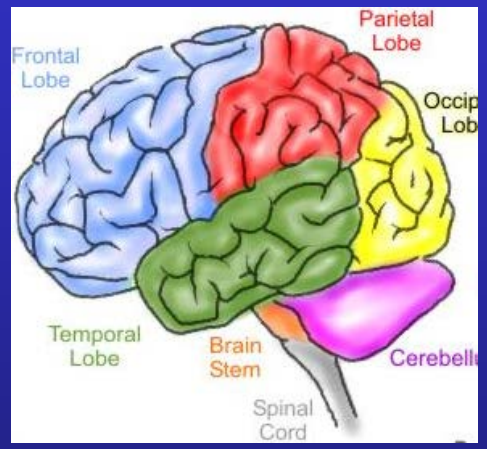
设计

降低成本

外观和机理



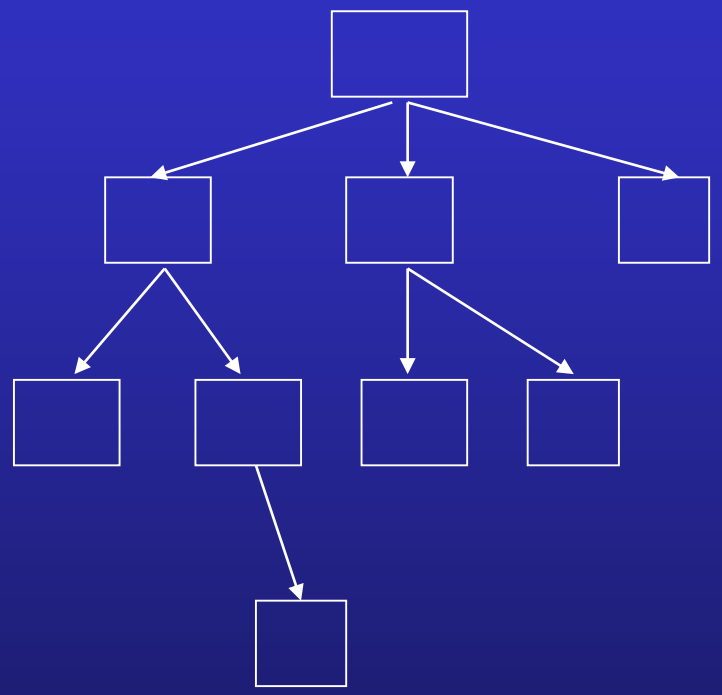
分解



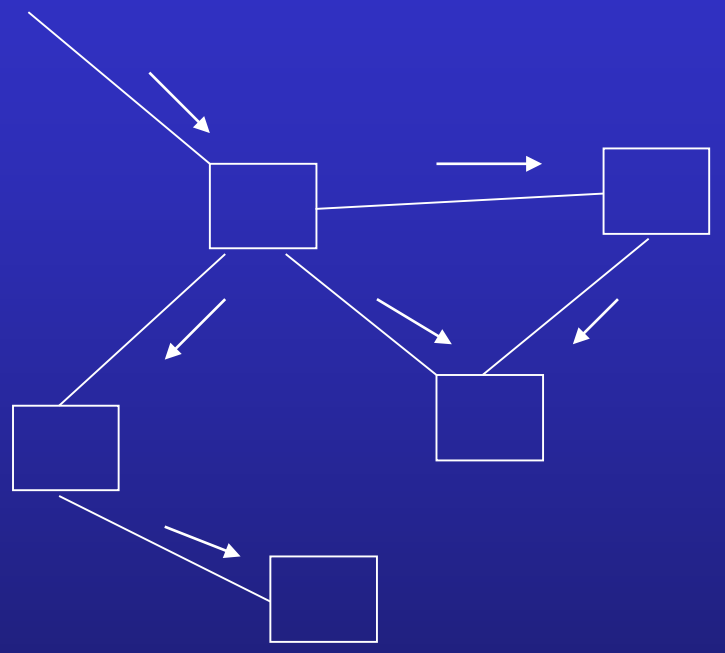
George Miller

人脑的把握度有限

分解

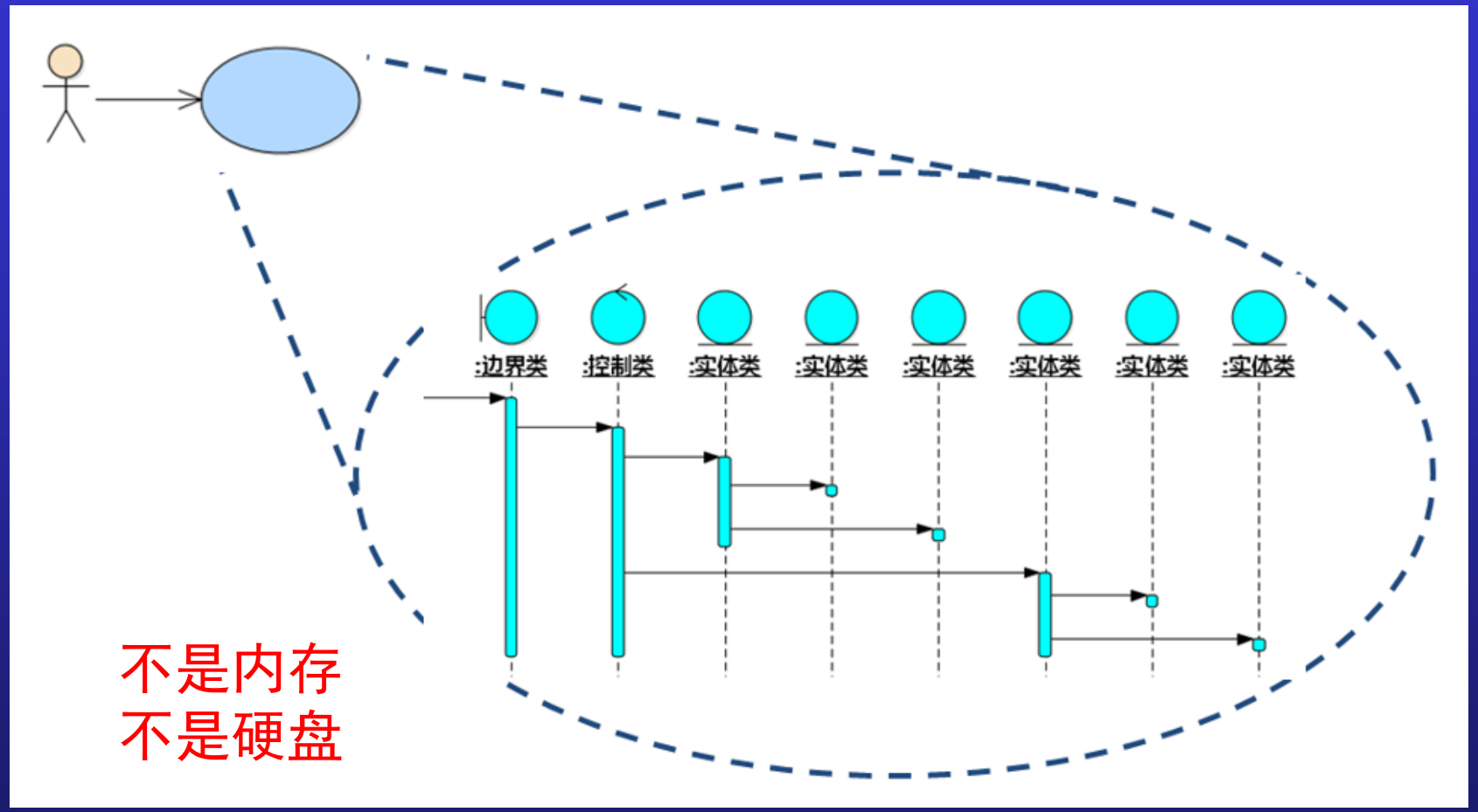


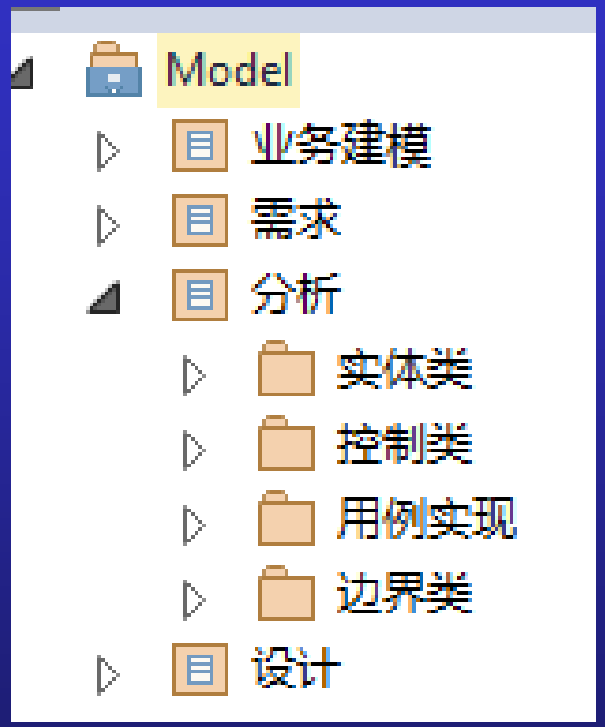
面向过程－功能分解



面向对象－对象协作

分析类





头脑中的虚“对象空间”

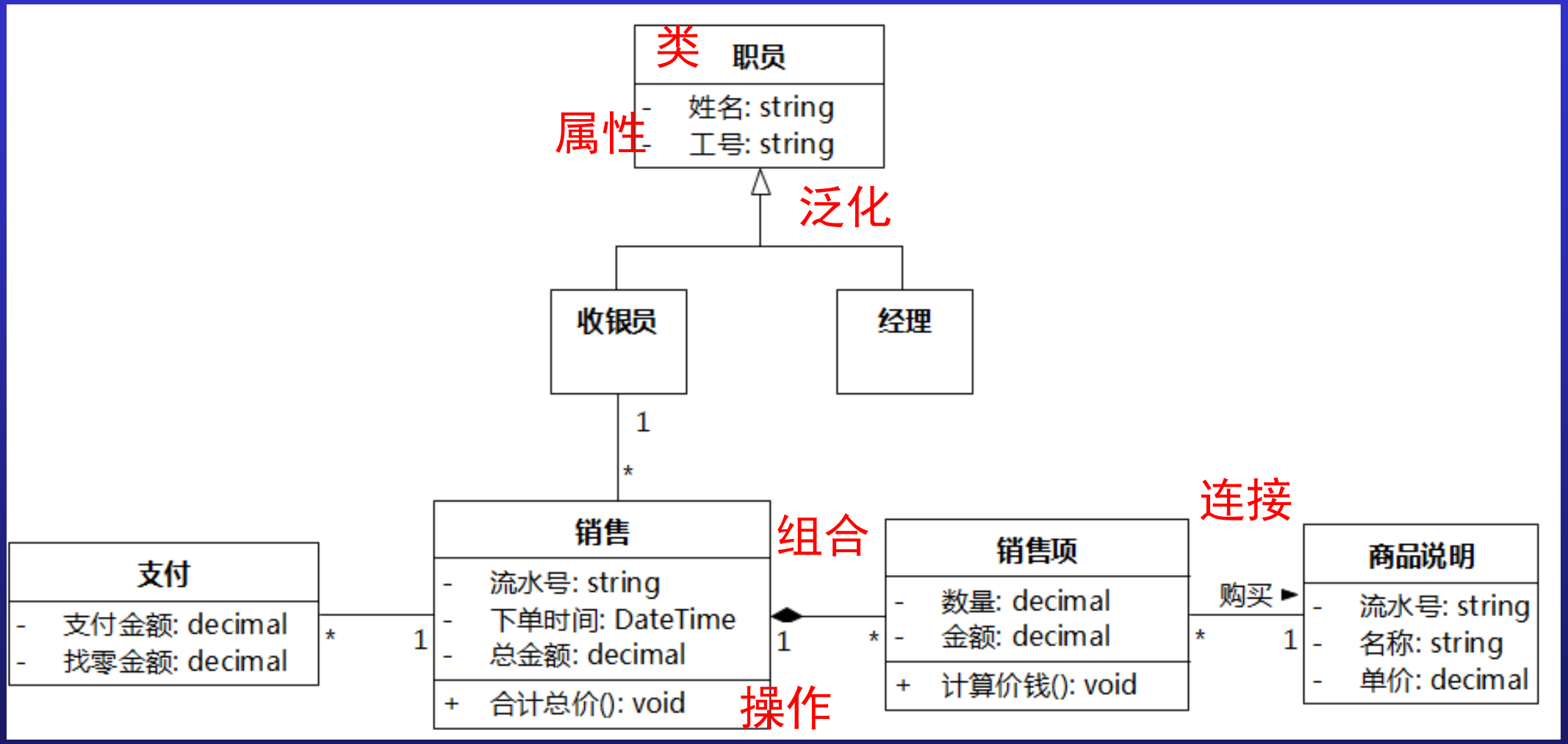
分析类

构造型	责任	和用例的关系	命名
边界类	输入、输出以及简单的过滤	每个有接口的外系统映射一个边界类。	外系统名称+接口
控制类	控制用例流，为实体类分配责任。	每个用例映射一个控制类。	用例名称+控制
实体类	系统的核心，封装领域逻辑和数据。	用例和实体类的关系是多对多的，一个用例可以由一到多个实体类协作实现，一个实体类可以参与一到多个用例的实现。	领域概念名称

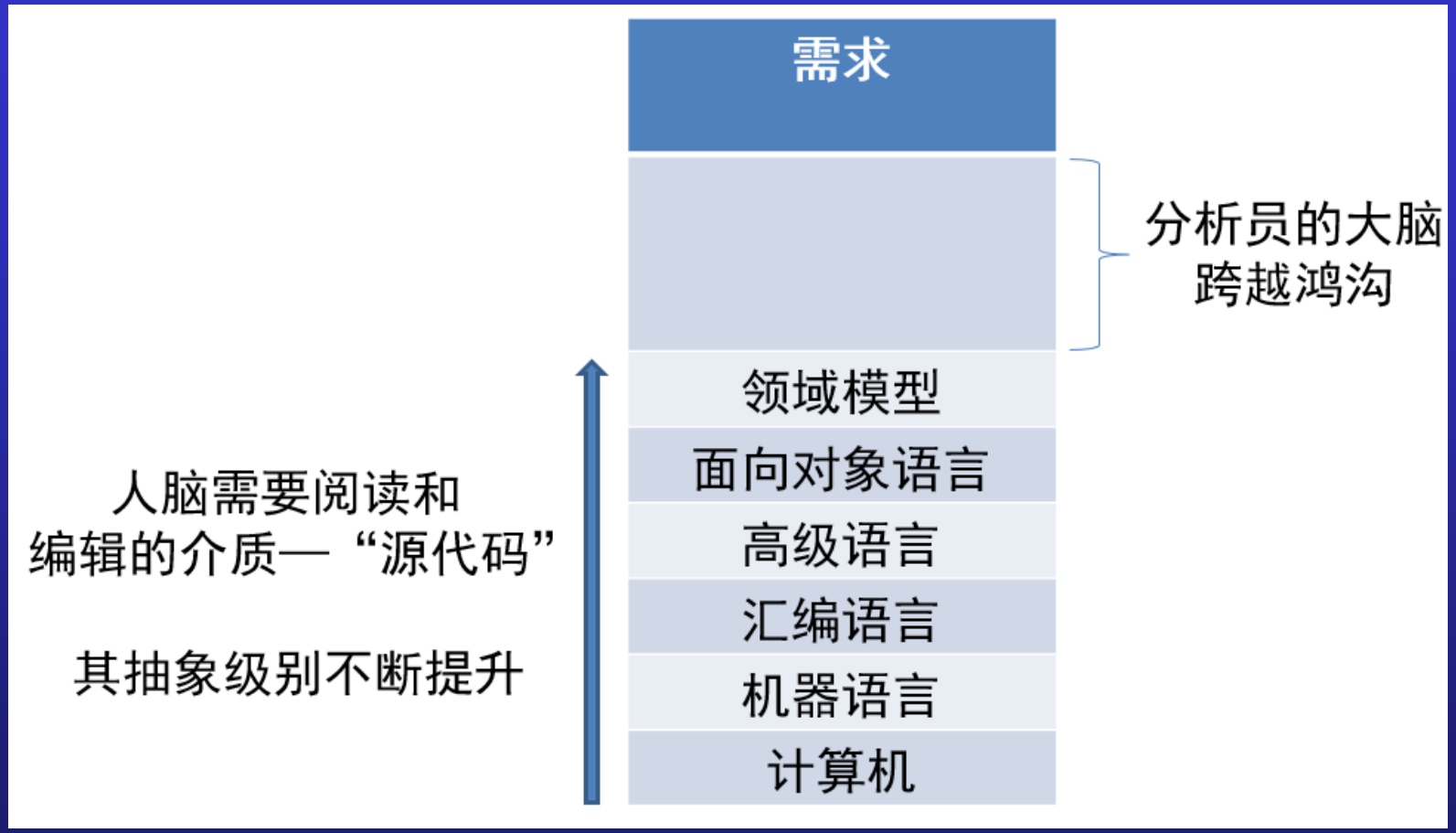
分析类

实体类图

主要元素



实体类和属性



分析员：领域知识+建模知识

实体类和属性

用例编号：用例名
执行者

前置条件

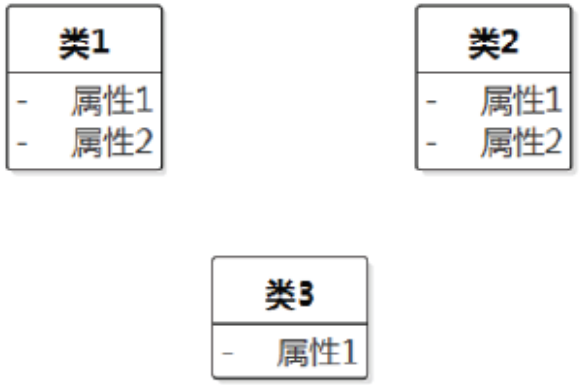
后置条件

涉众利益

基本路径
*****
扩展
*****
字段列表
*****
业务规则
*****
非功能需求

设计约束

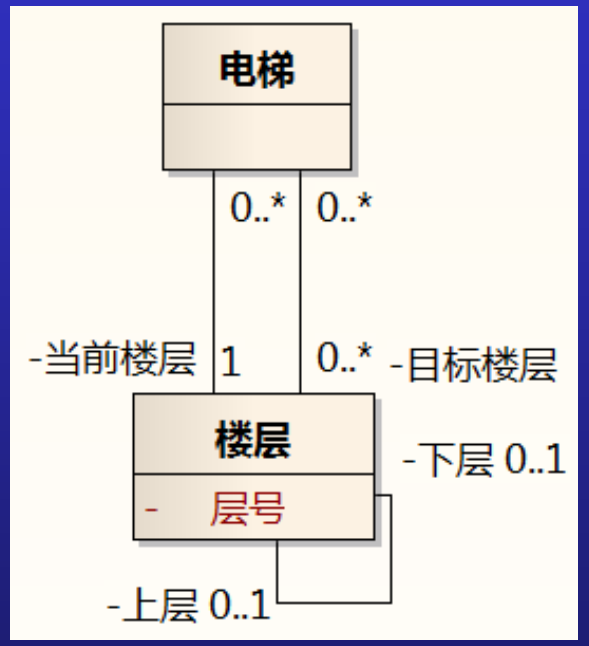
系统要维护的核心概念



抽取用例规约中的名词和事件

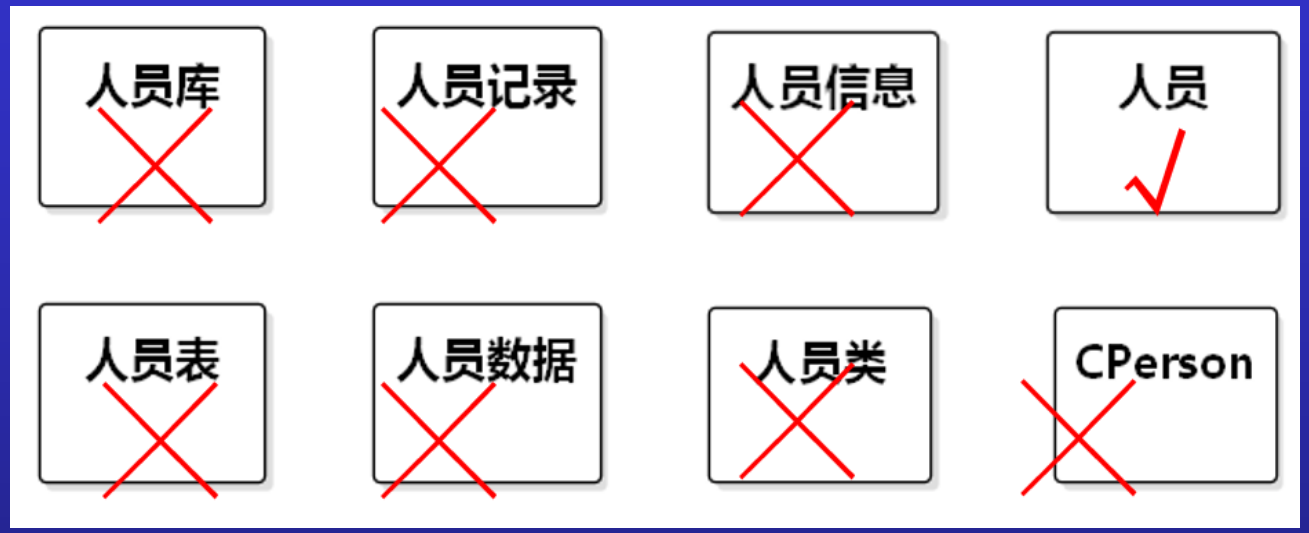
实体类和属性

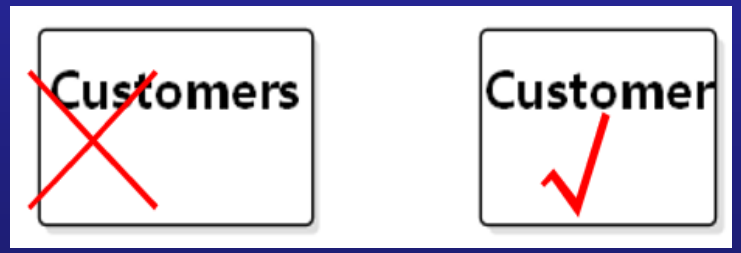
int 目标楼层=5; ╳



电梯调度系统的恰当抽象？

实体类和属性





不要在类名的最后加"类"字；

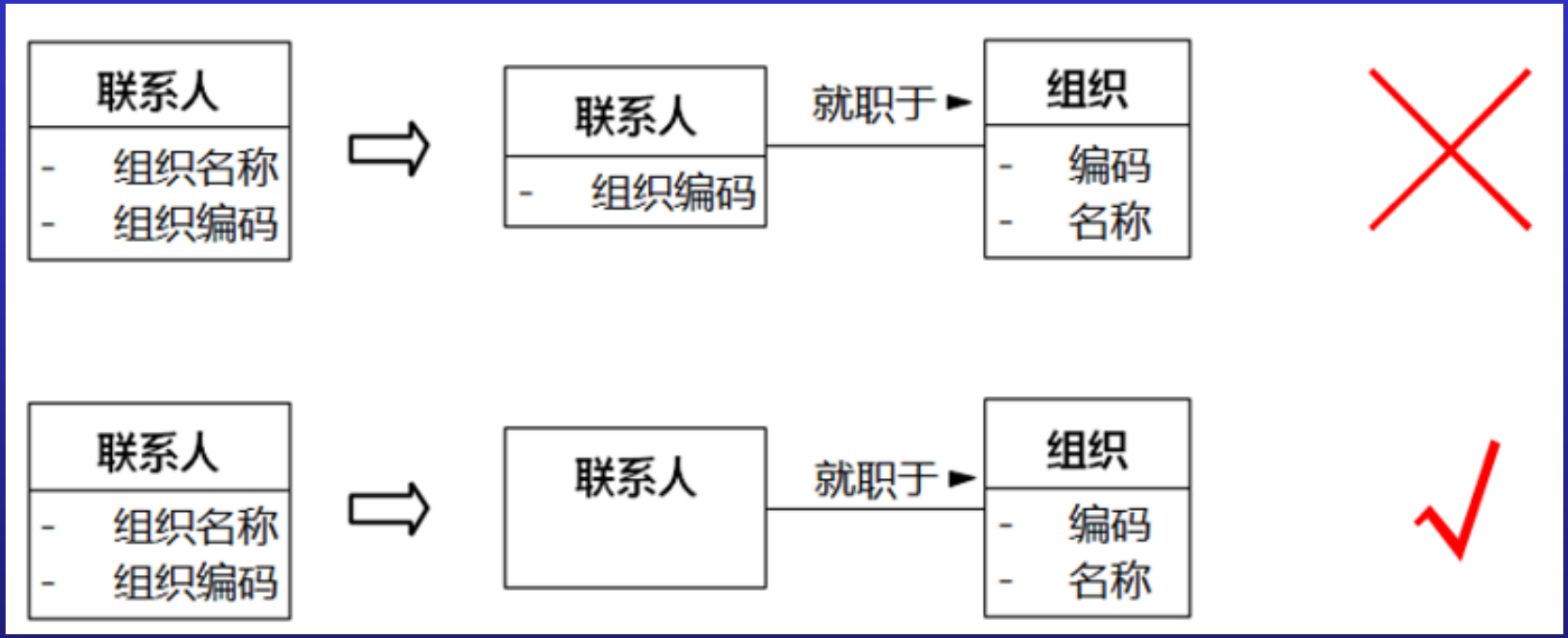
不要在类名的前后加"Class"或"C"；

不要在类名的最后加"情况"、"信息"、"记录"、"数据"、"表"、"库"、"单"等词。

英文：不用缩写、单数

类命名

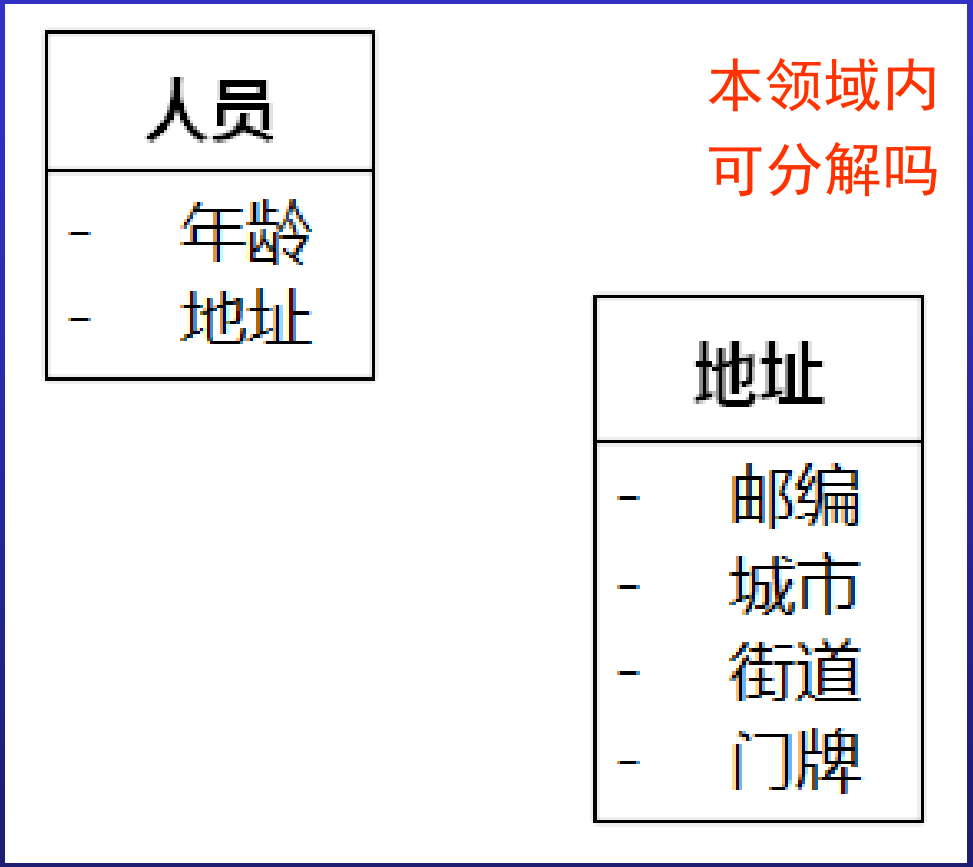
实体类和属性



什么的什么

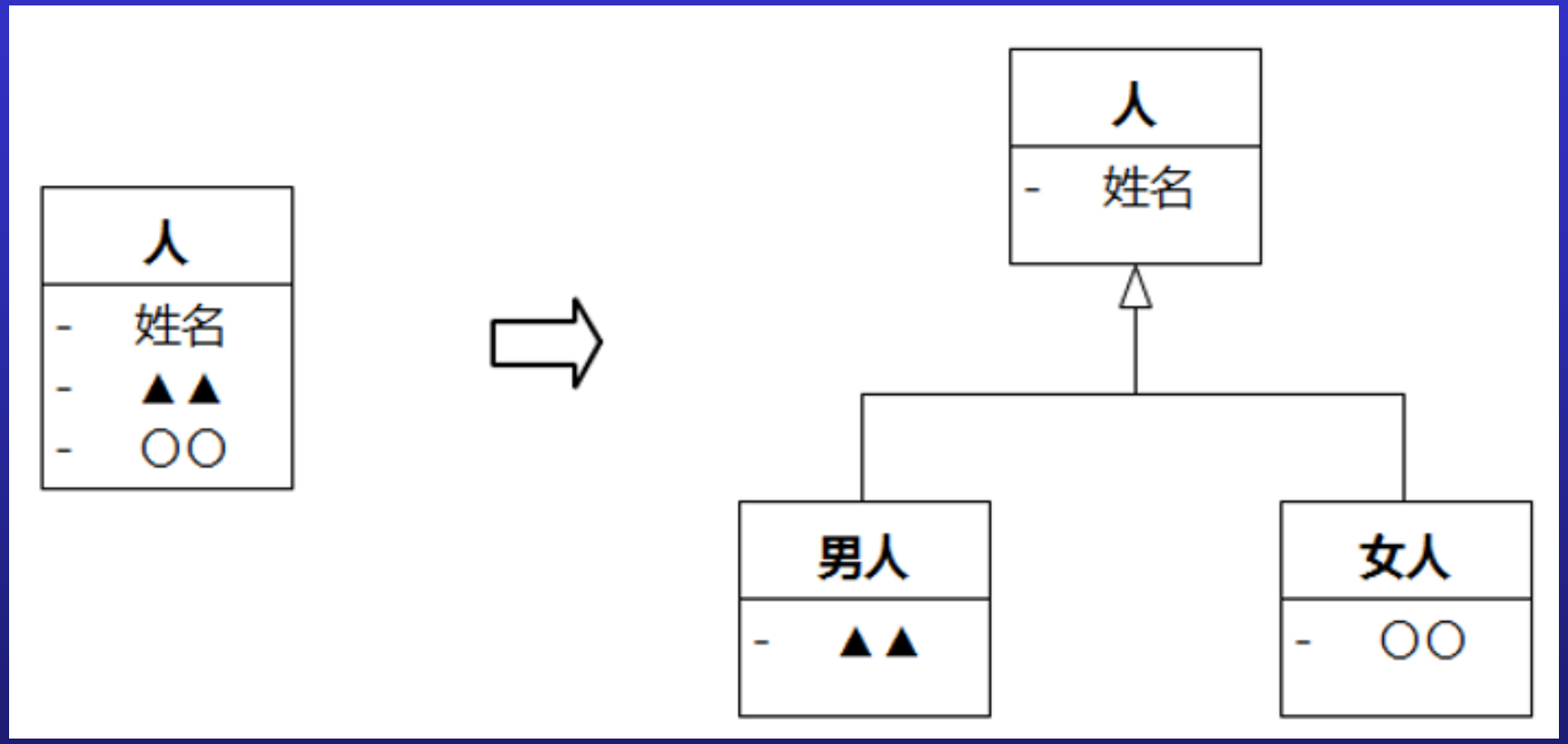
审查——属性是否直接描述类的特征

实体类和属性



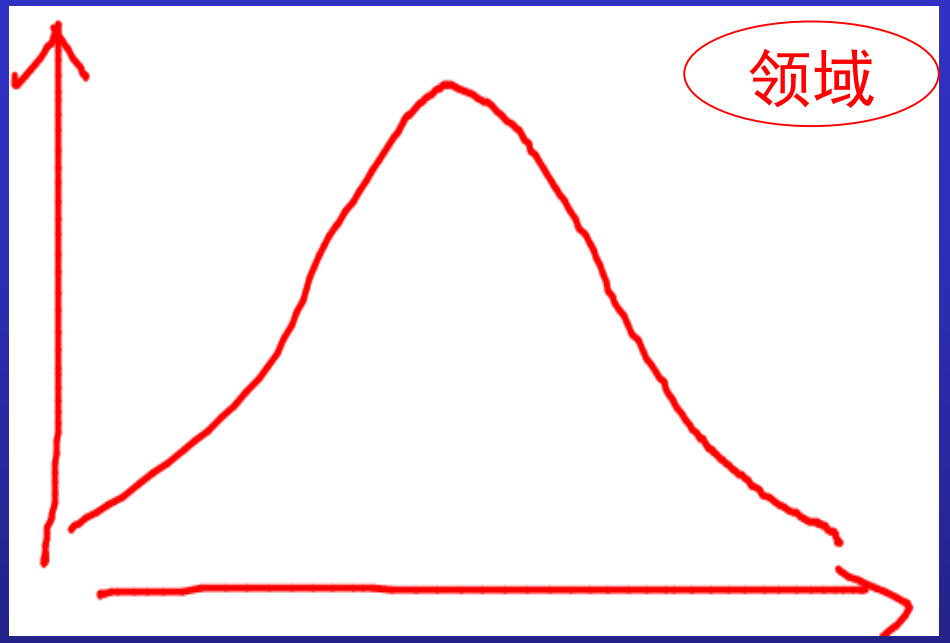
审查——是否有复杂结构或1对多的属性

实体类和属性



审查——属性是否对类的所有对象都有意义

实体类和属性



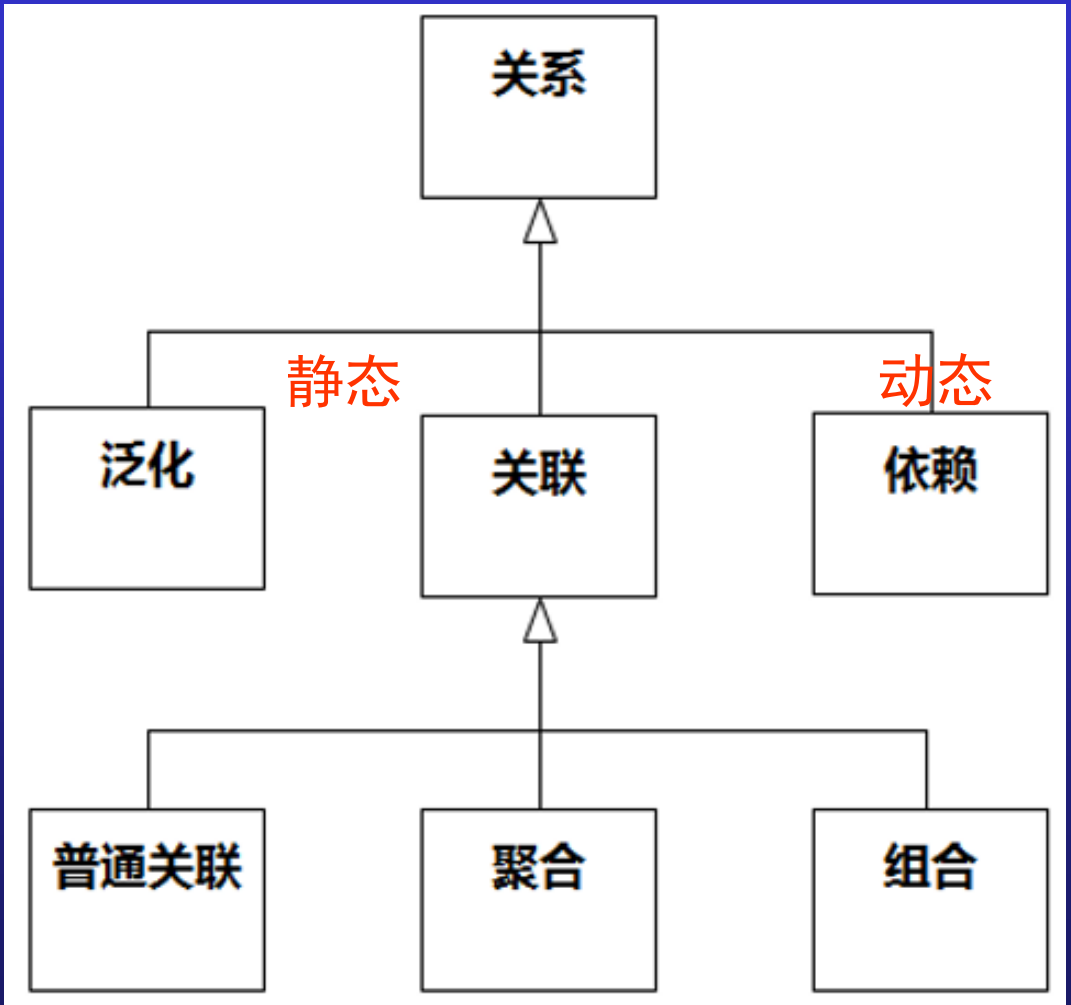
文章本天成
妙手偶得之

不是“设计”
而是“描述”

体现领域的真正味道
扭曲的映射难以应变

恰当抽象--抓住领域内涵应对需求变更

类的关系



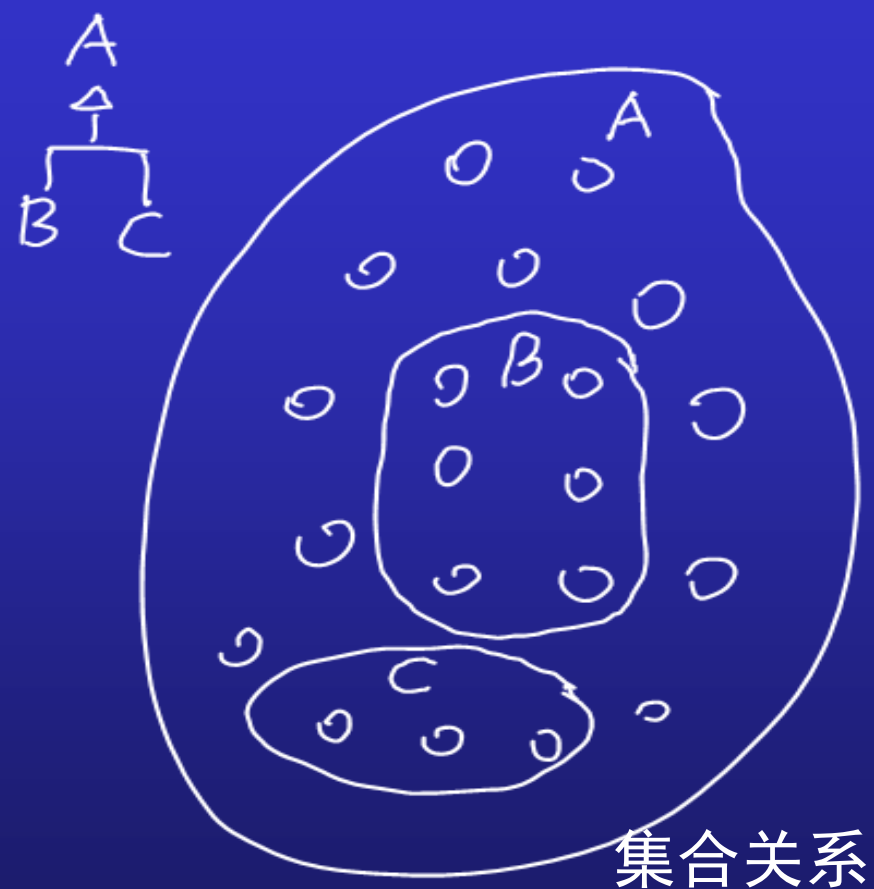
类的关系

- 张三和李四是夫妻
- 张三和李四昨天下午四点在某某地方发生了××交互

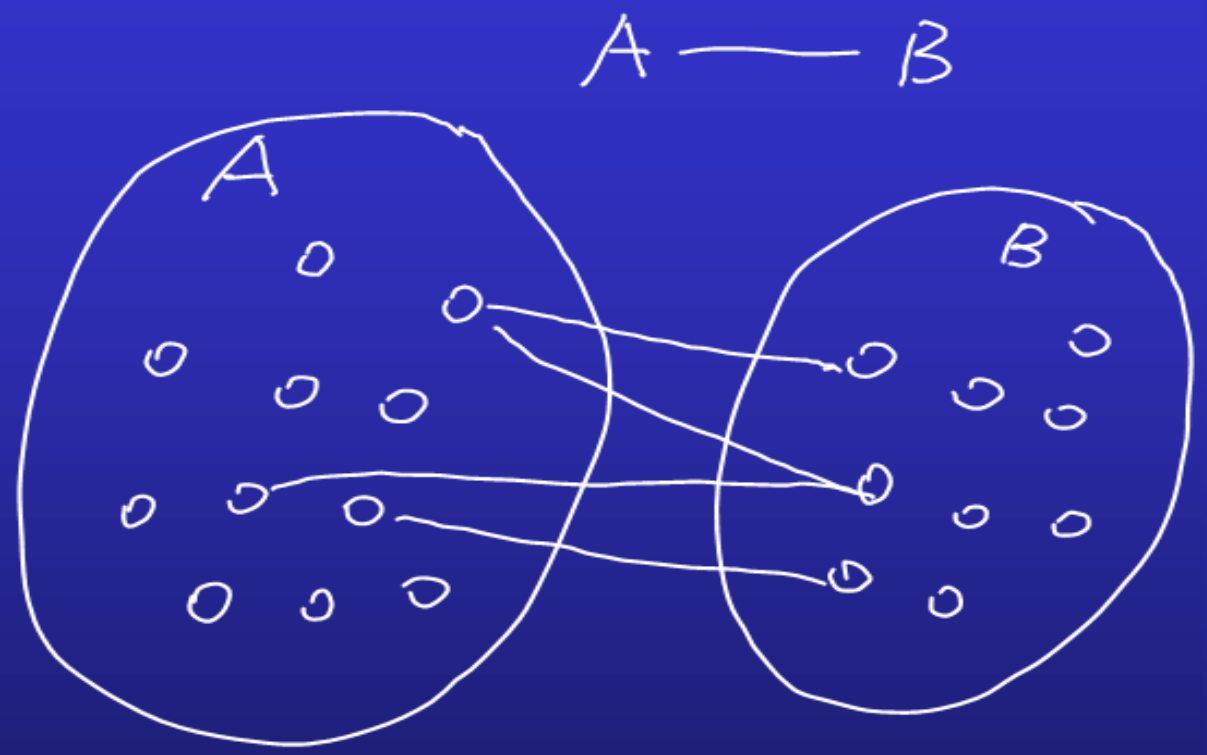
- 先对泛化关联建模，剩下的看作依赖

静态是原因，动态是结果

类的关系



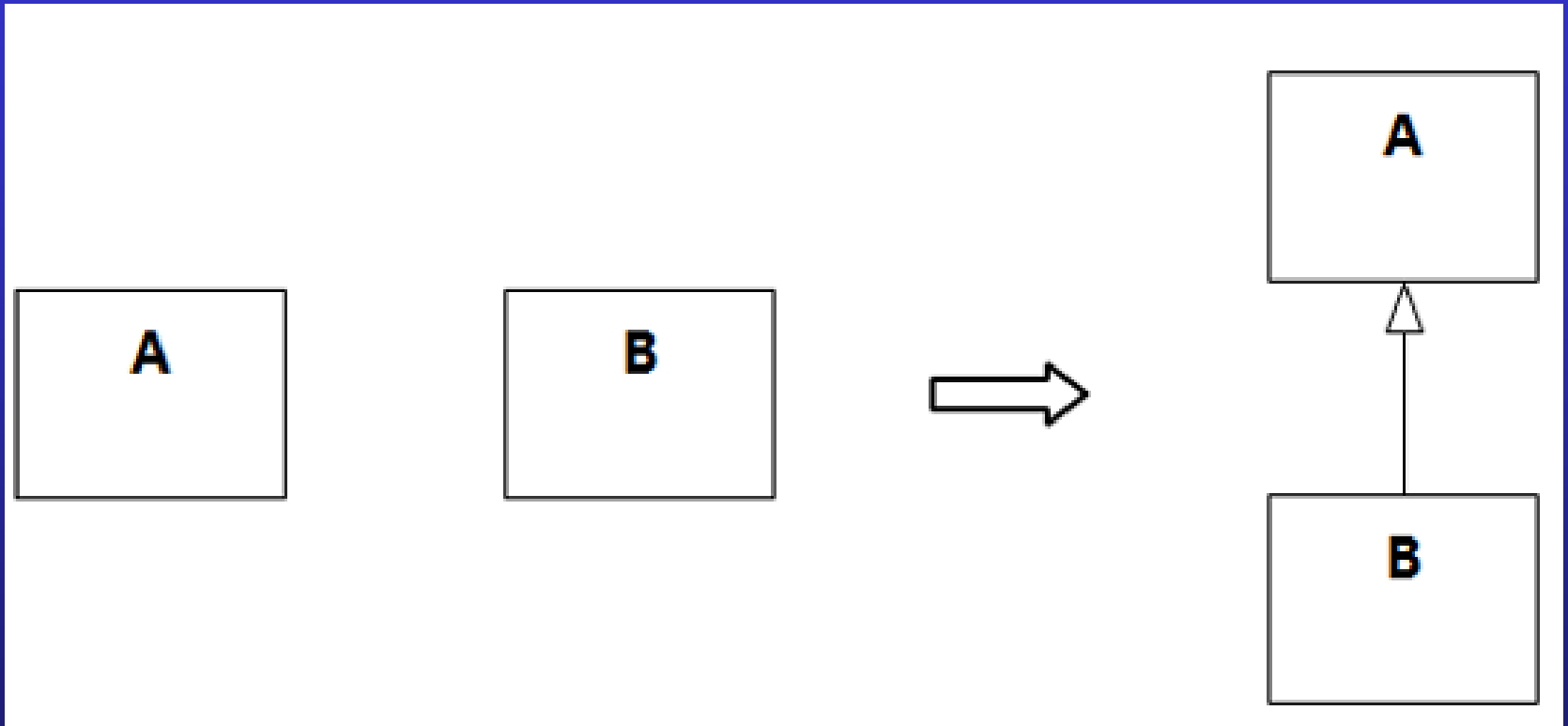
集合关系



个体关系

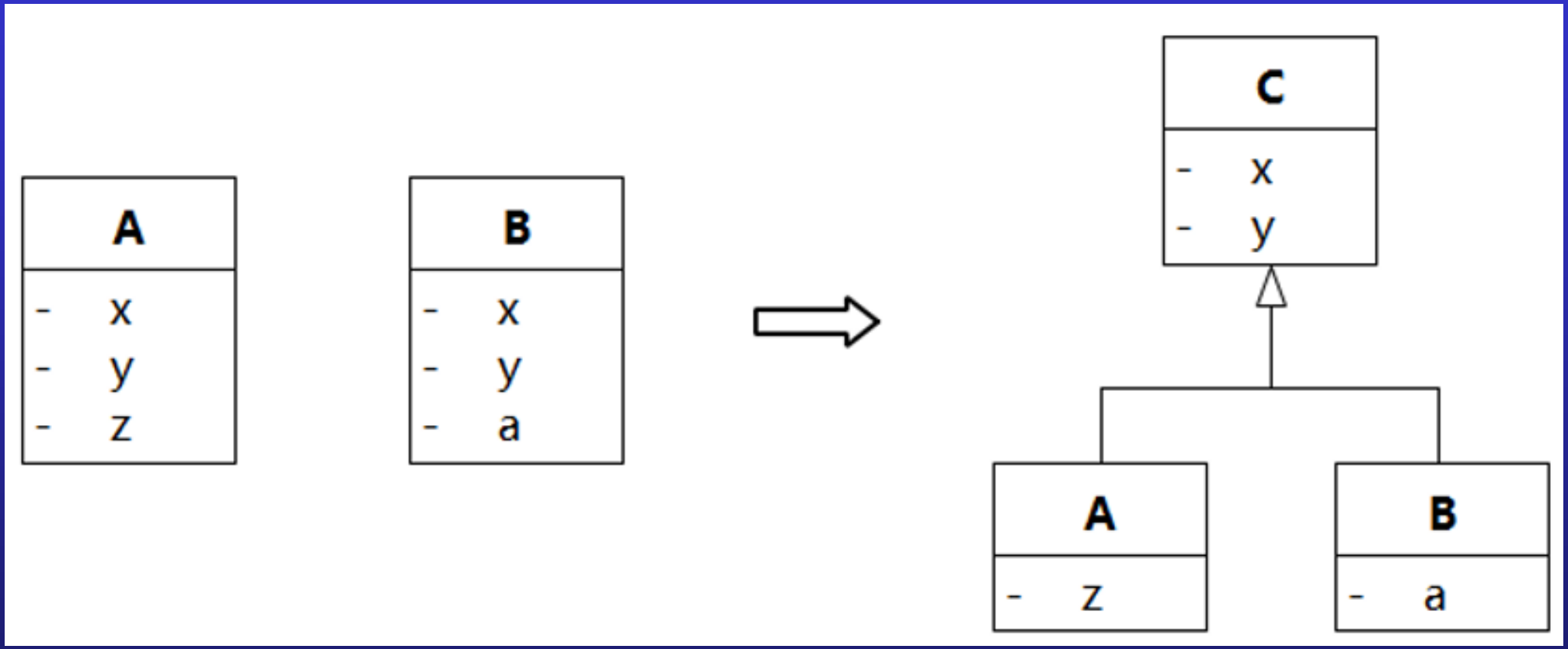
泛化和关联

泛化



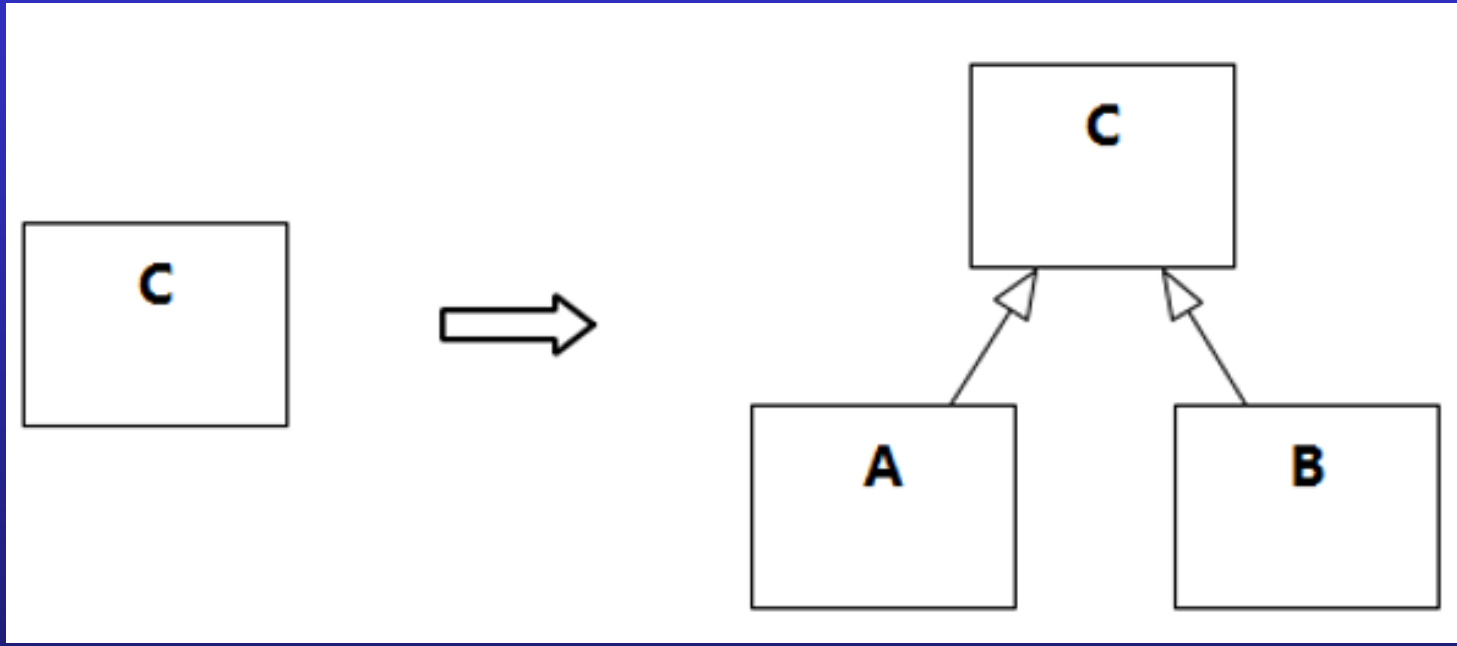
直接形成

泛化



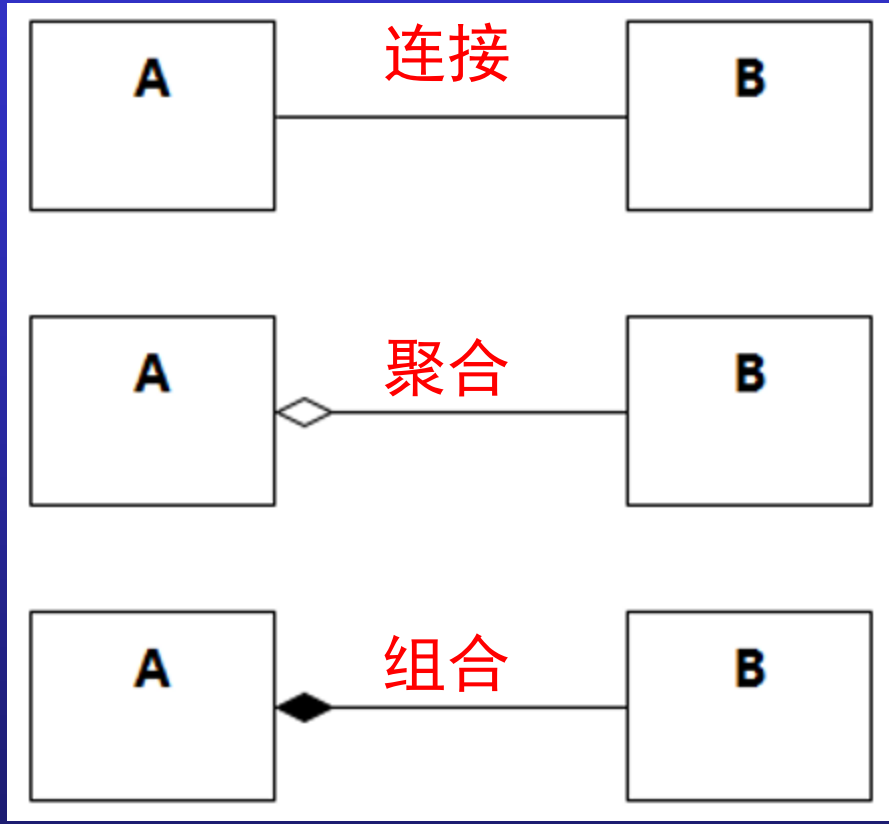
自下而上——从特殊到一般

泛化



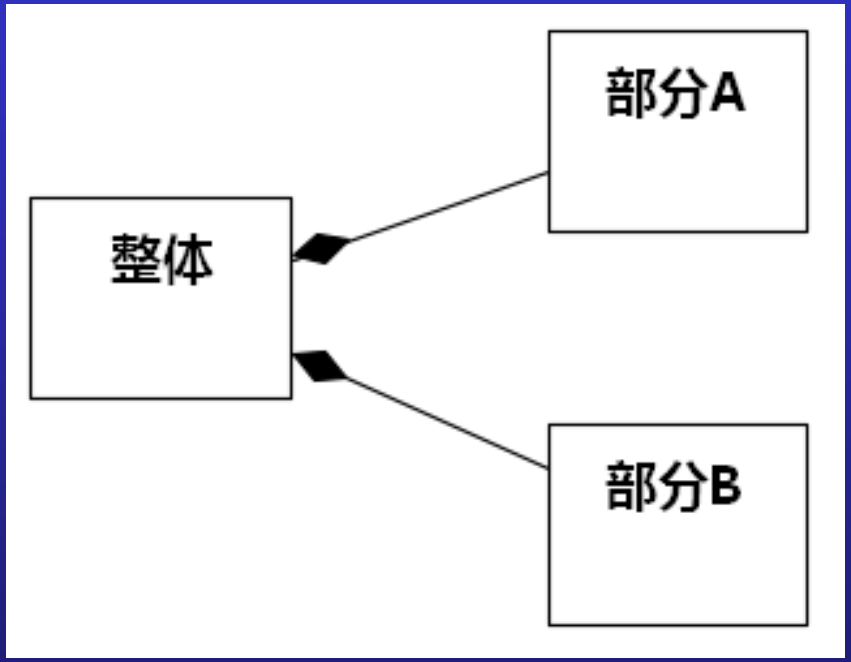
自上而下——从一般到特殊

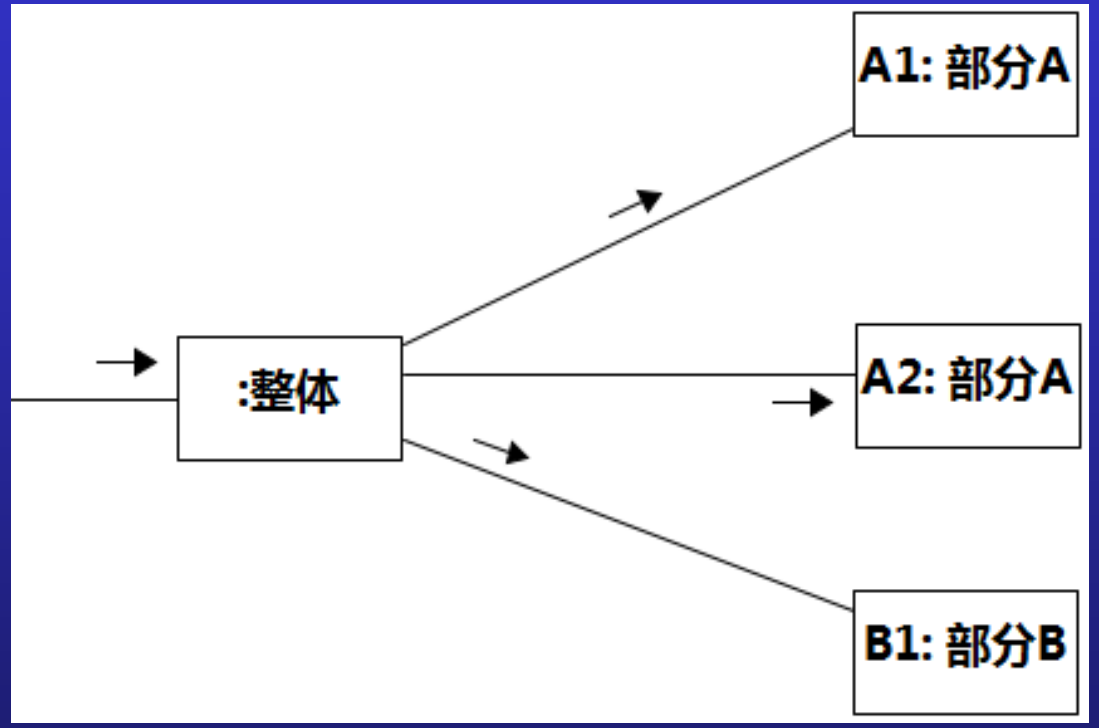
关联



关联的表现形式

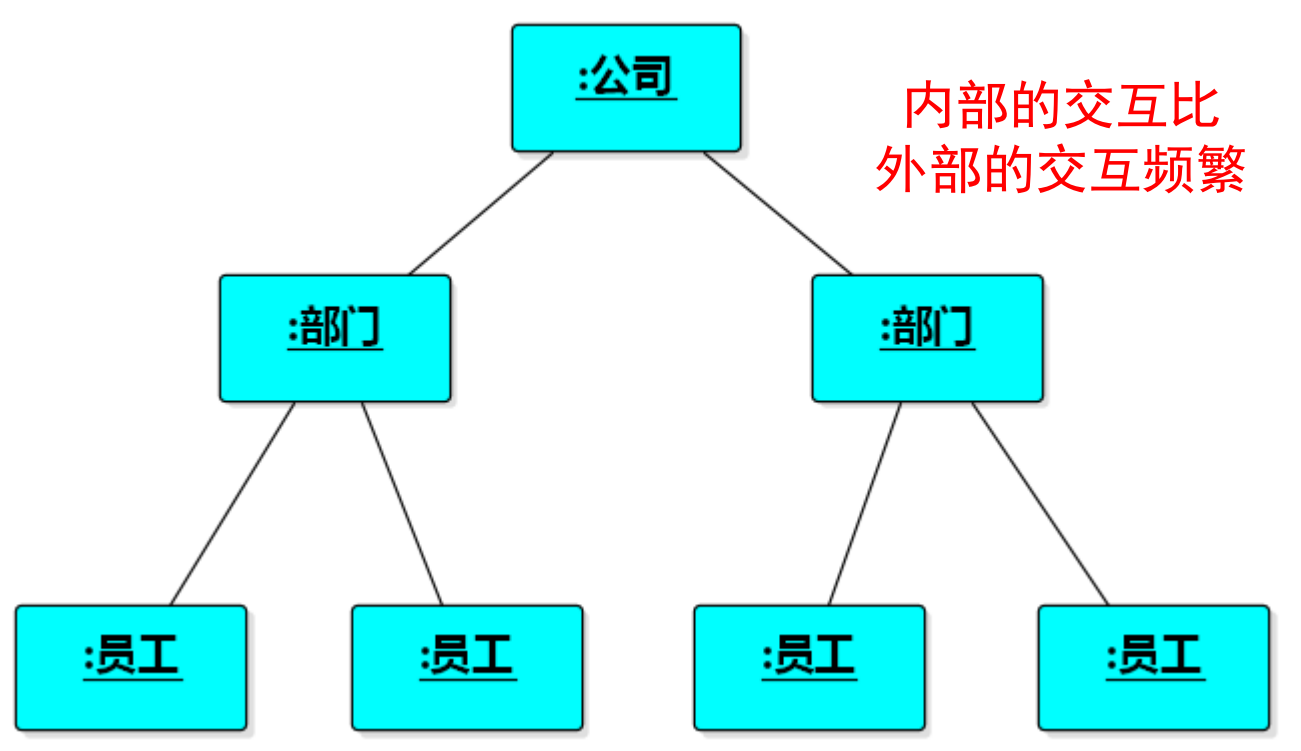
关联





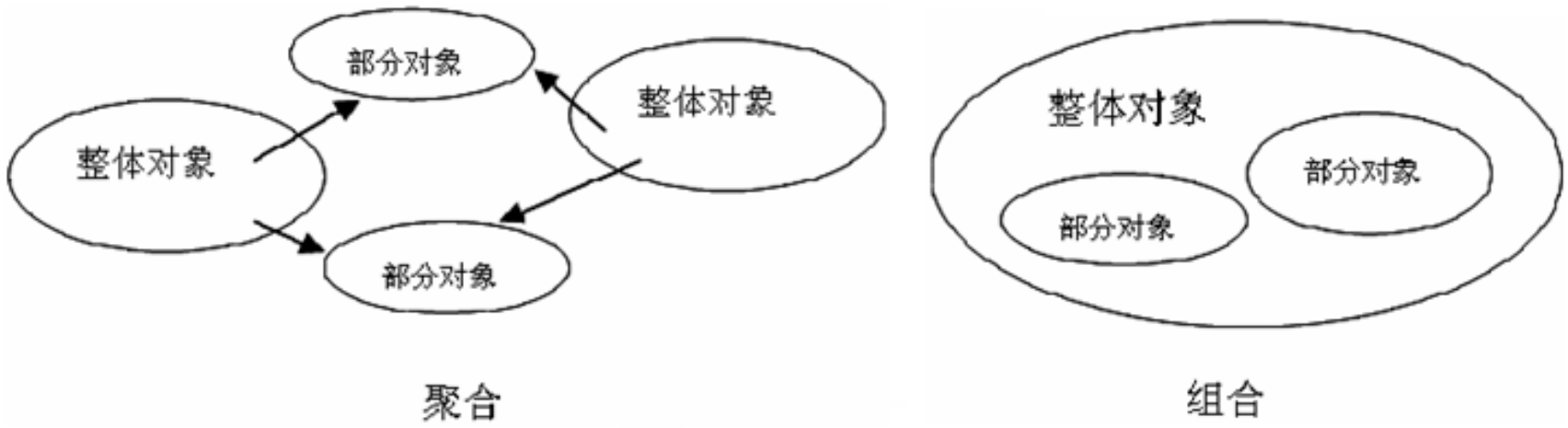
普通关联 vs. 聚合/组合：责任分配

关联



分离和聚集系统中的复杂性

关联



公司和雇员

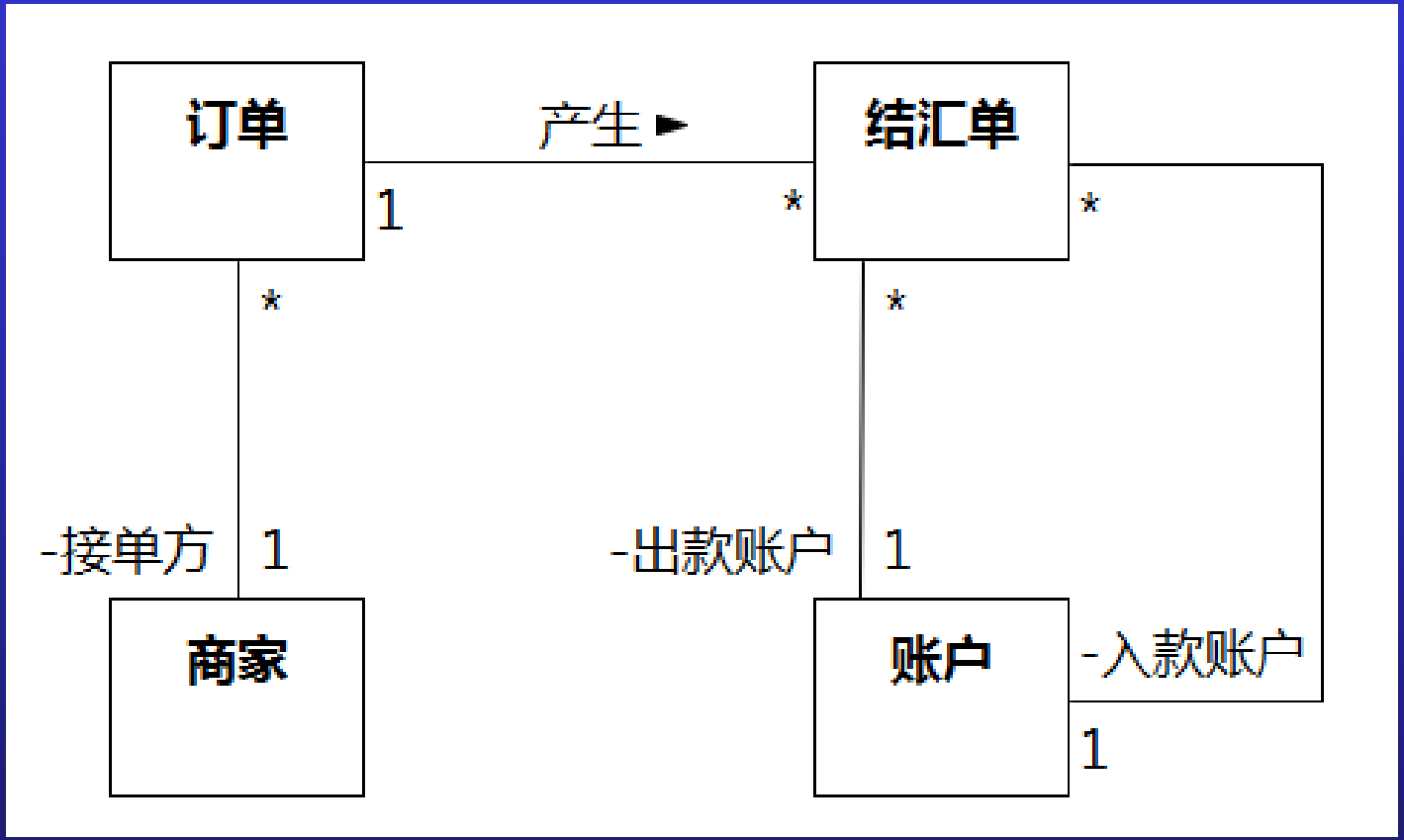
订单和订单项

聚合 vs. 组合：松散和紧密

关联

- 逐一考虑类图上各类之间关系，以及类与类自身的关系

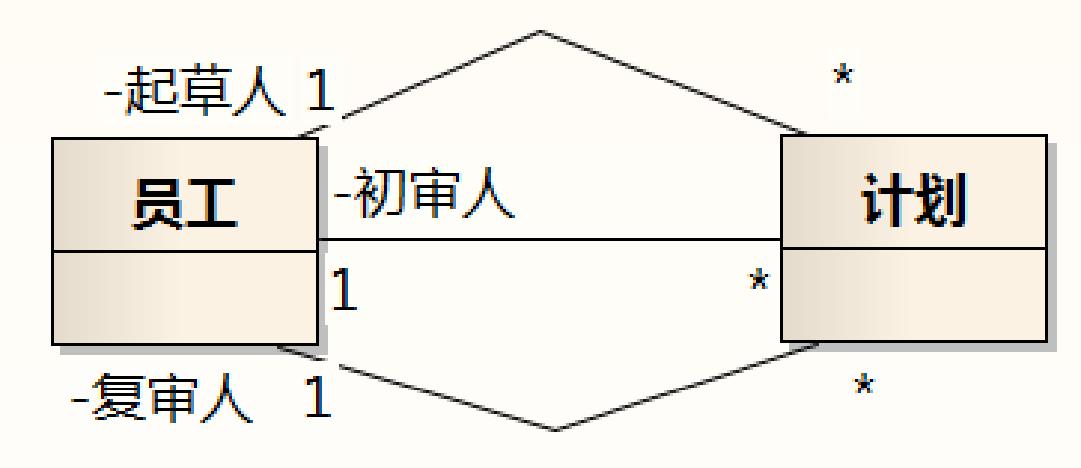
关联



多种关联
角色名优先

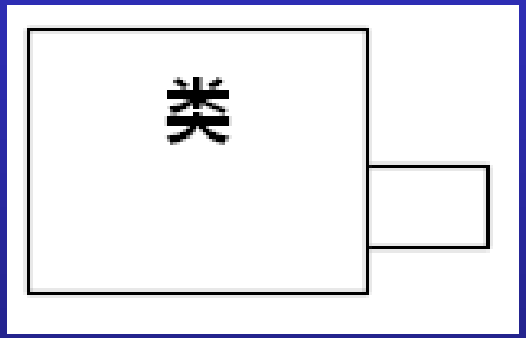
尽量给关联和角色起名——模型要能讲故事

关联



区分“多种关联”和“多重性”

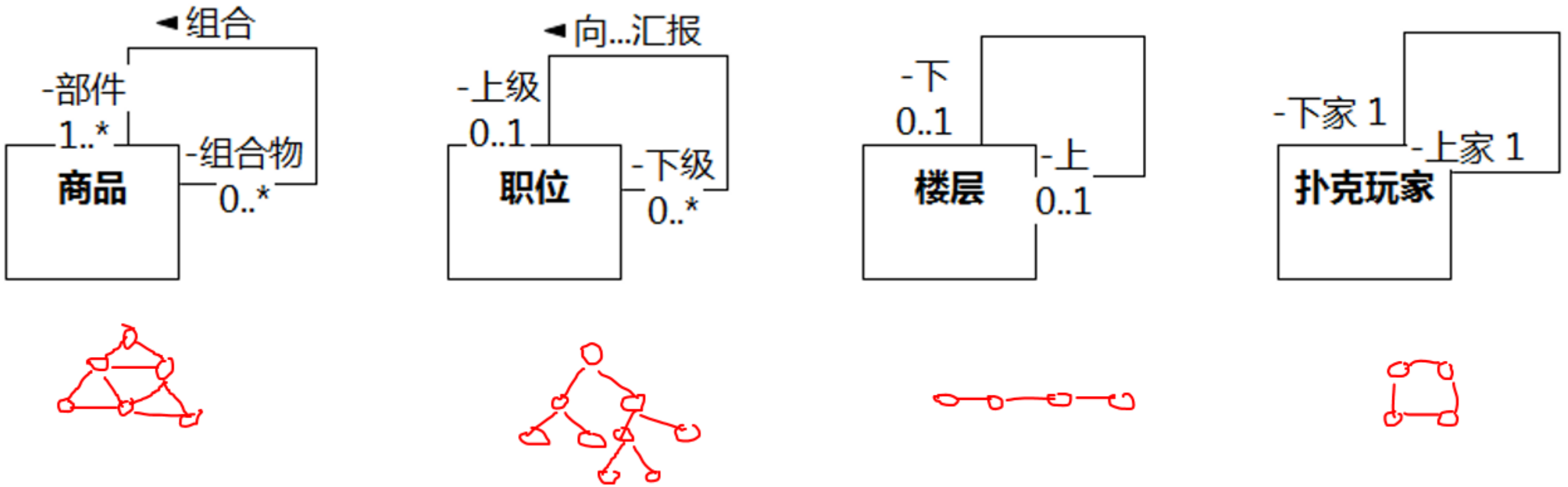
关联



多重性	形状
多对多	网络
1对多	树
1对0. . 1	队列
1对1	环

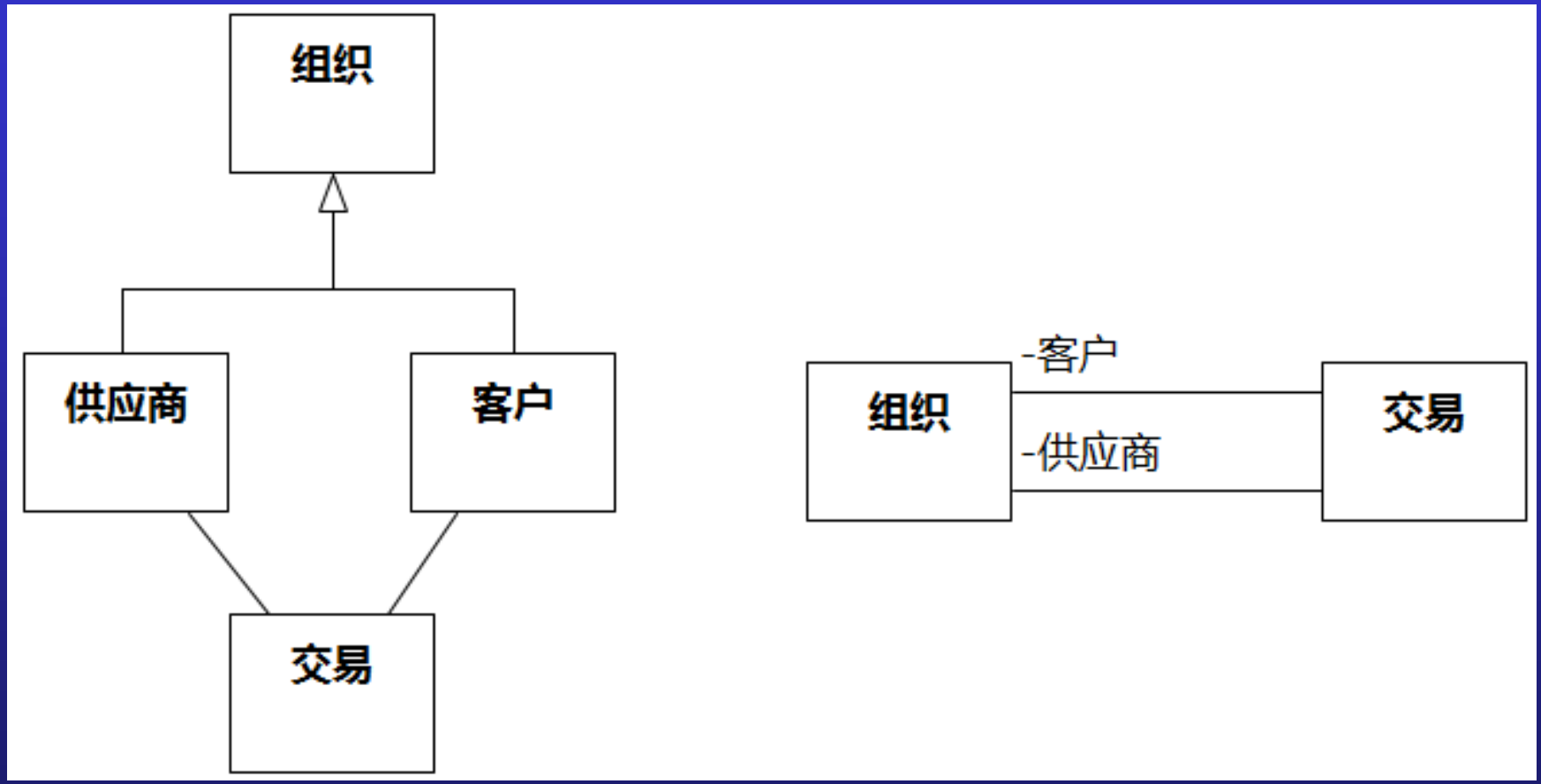
自反关联

关联



自反关联——不同形状

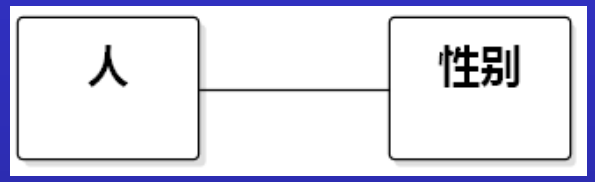
泛化和关联



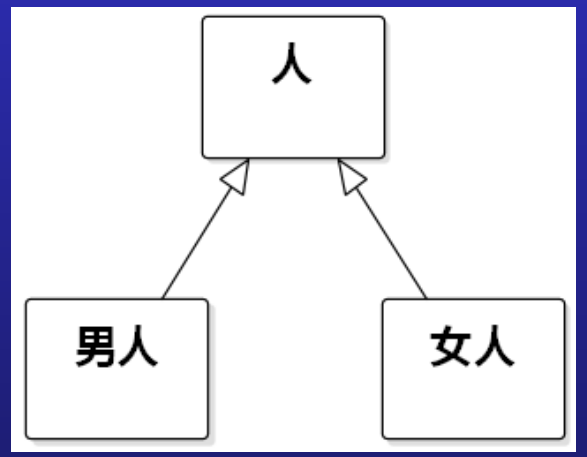
泛化变为角色

泛化和关联

- 共享数据——关联优先



- 行为变异——泛化优先



选择

建模工作流

*业务建模
- 愿景
- 业务用例图
- 现状业务序列图
- 改进业务序列图

*需求
- 系统用例图
- 系统用例规约

需求

提升销售

---

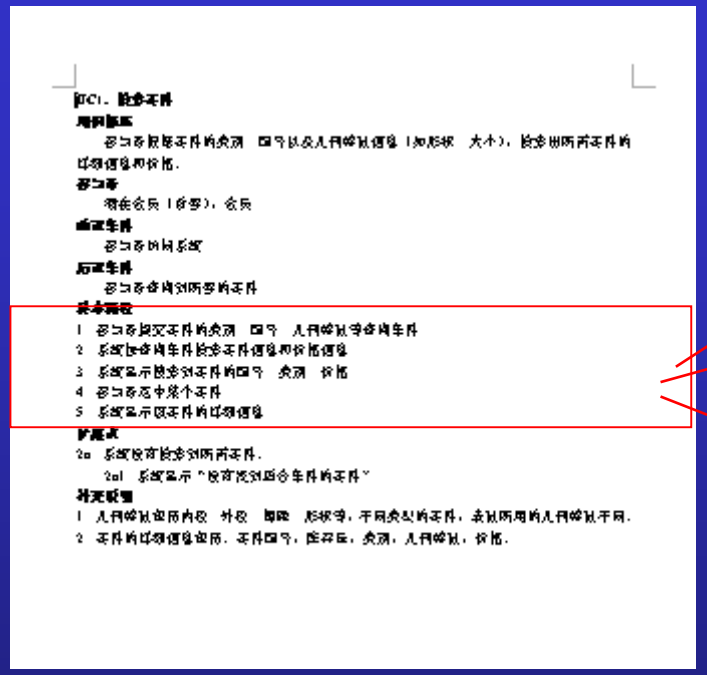
*分析
- 分析类图
- 分析序列图
- 分析状态机图

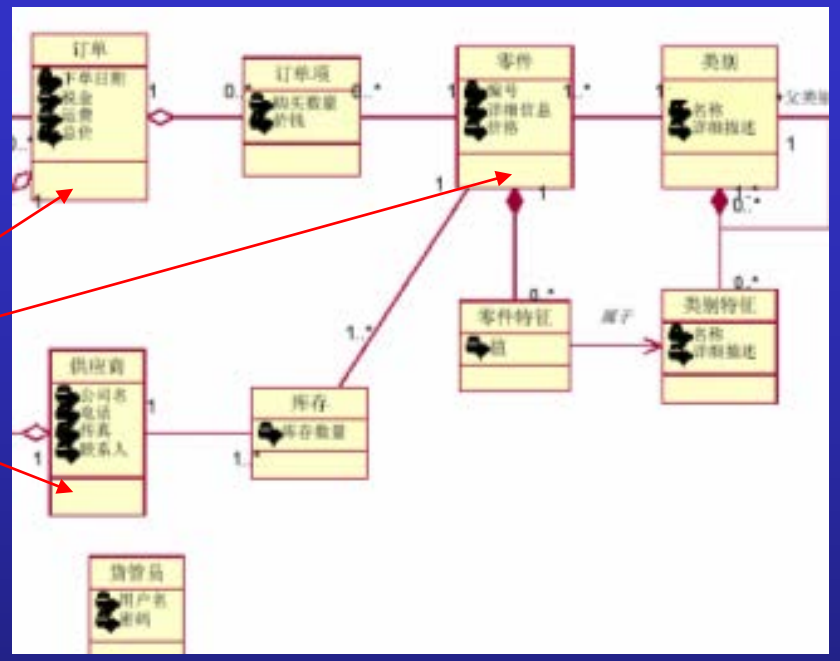
*设计
- 建立数据层
- 精化业务层
- 精化表示层

设计

降低成本

序列图





用例规约

类图

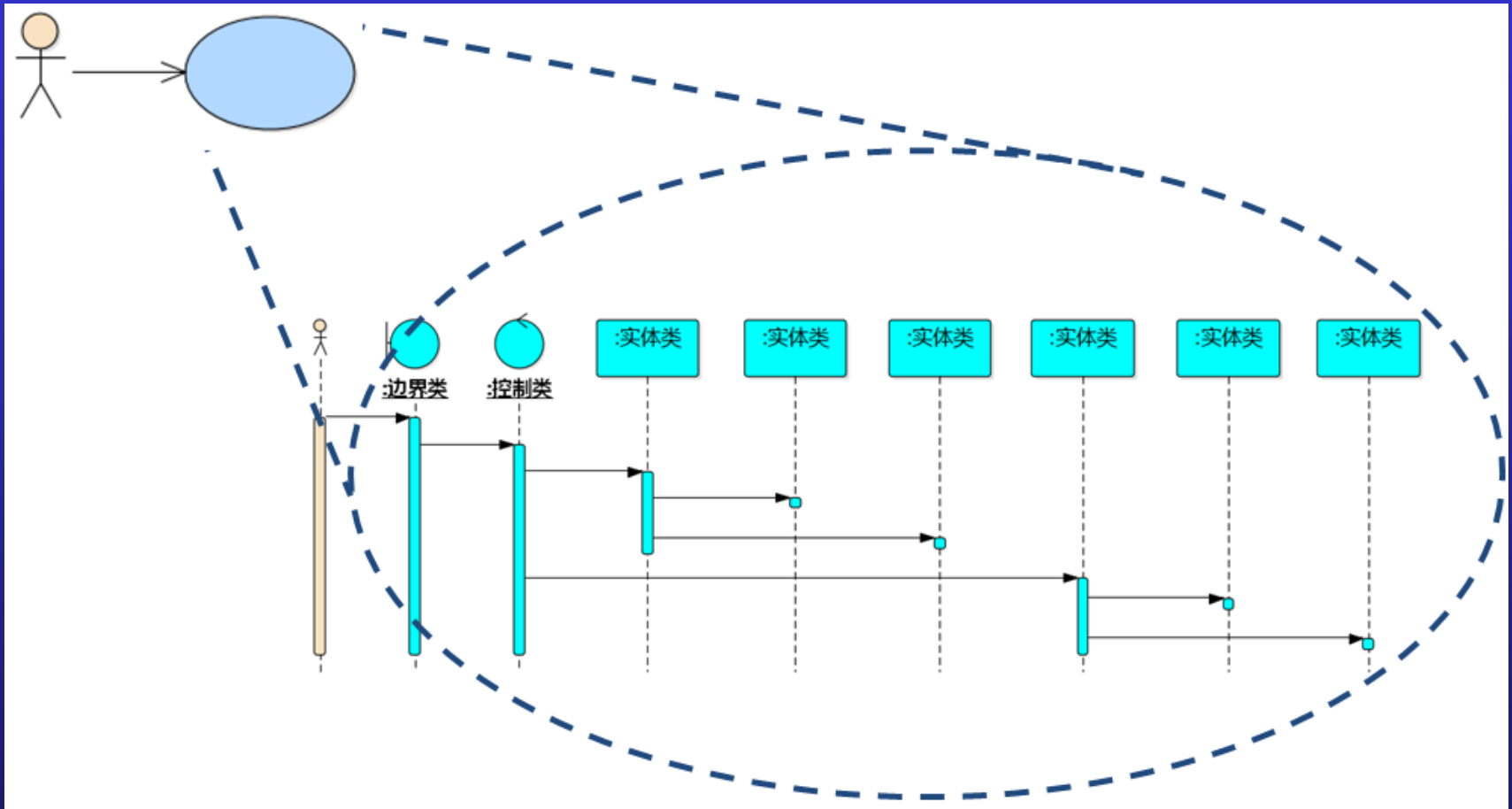
通过序列图完成责任分配

序列图

	研究对象	最小颗粒
业务序列图	组织内各系统之间	系统（人肉、电脑）
分析序列图	系统内各类之间	软件类

业务序列图 vs. 分析序列图

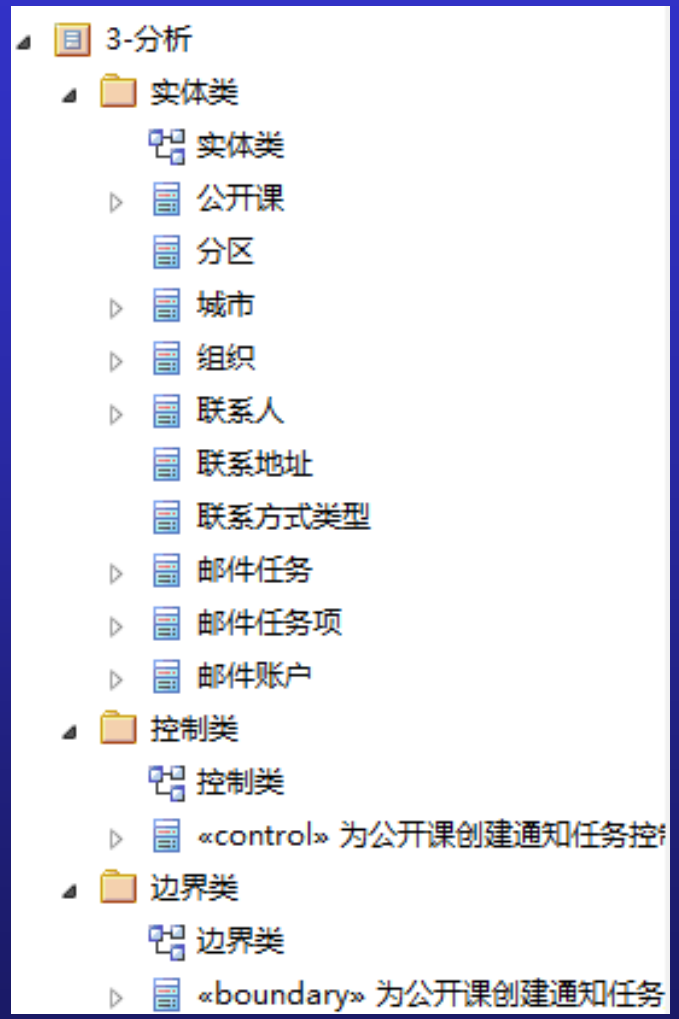
序列图



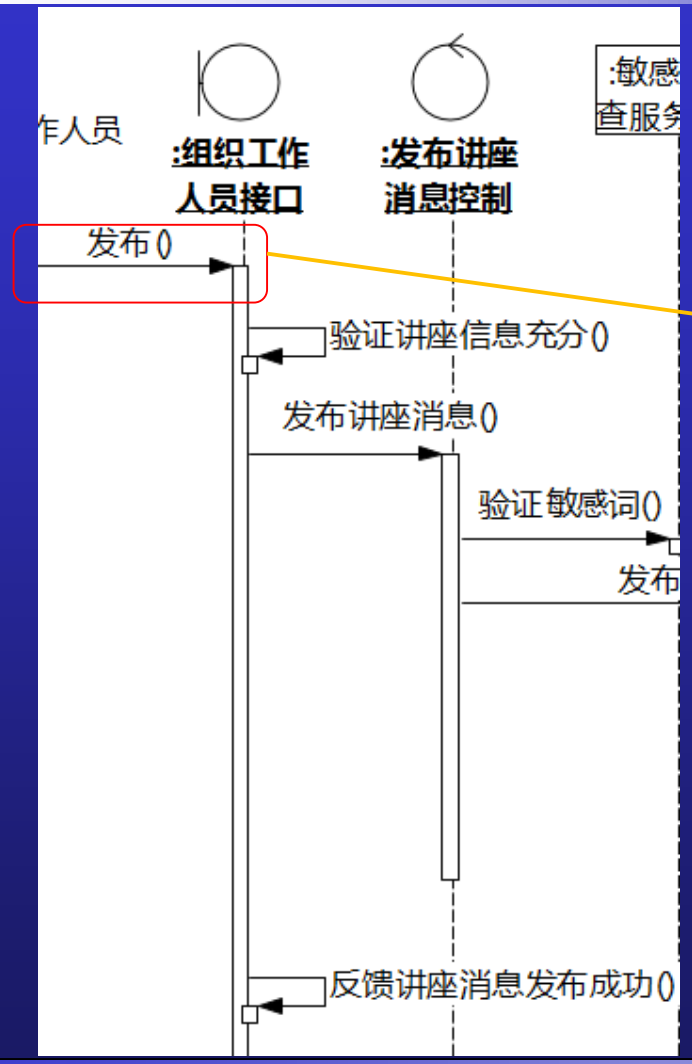
想象：系统内的交互模式

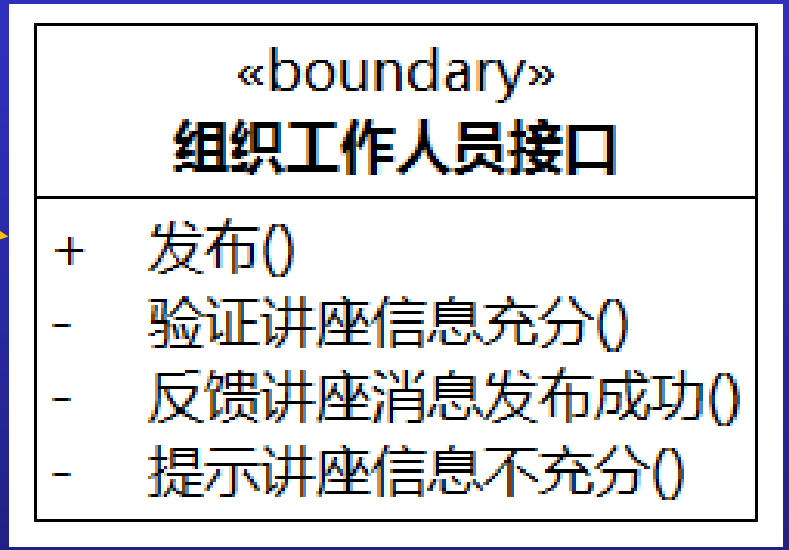
序列图

- 边界类：执行者对应边界类
  - 责任：输入、输出、过滤
- 控制类（可选）：用例对应控制类
  - 责任：控制事件流，负责为实体类分配责任
- 实体类：一个用例有多个实体类参与，一个实体类可以参与多个用例
  - 责任：业务行为的主要承载体



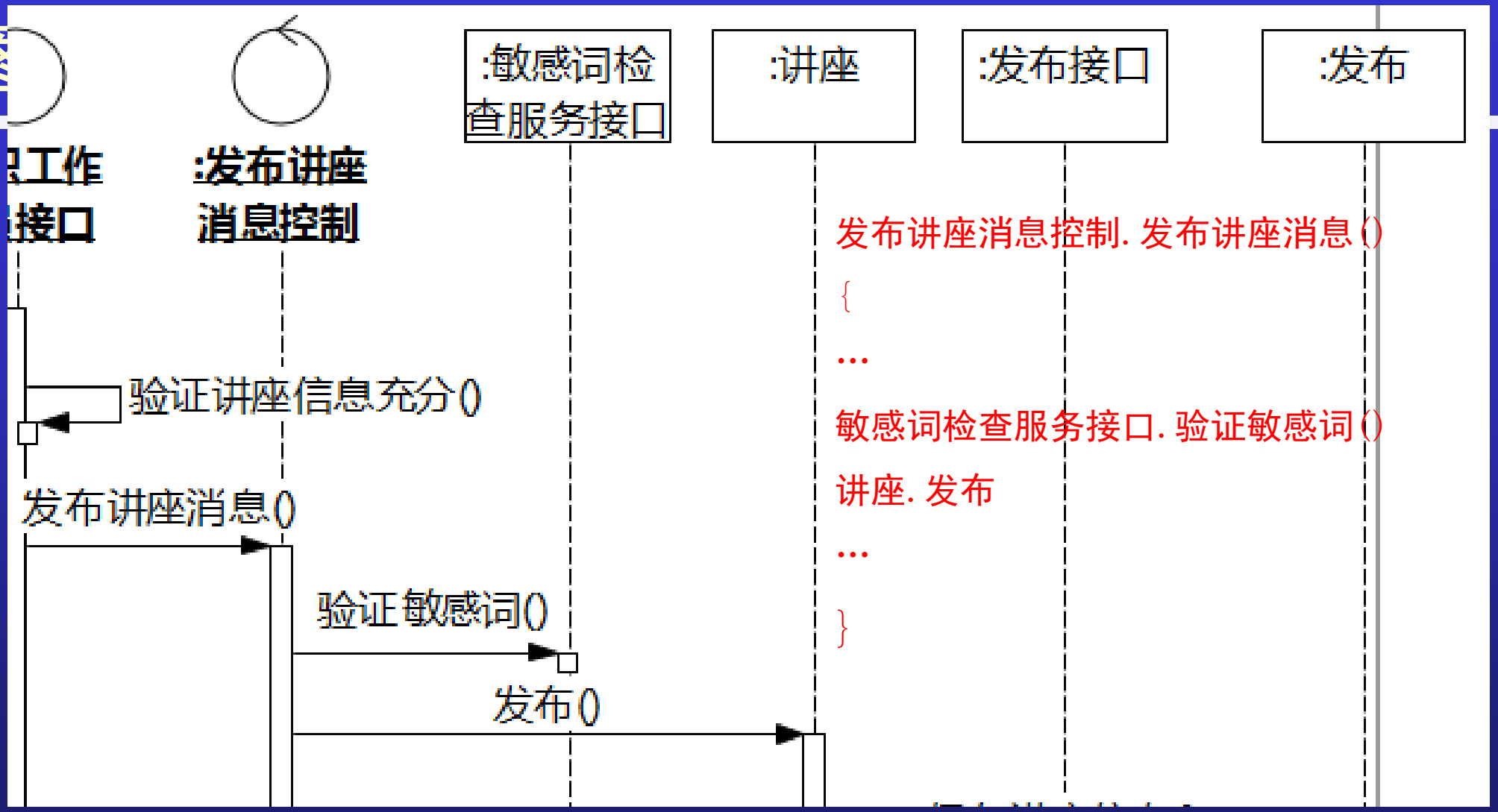
序列图





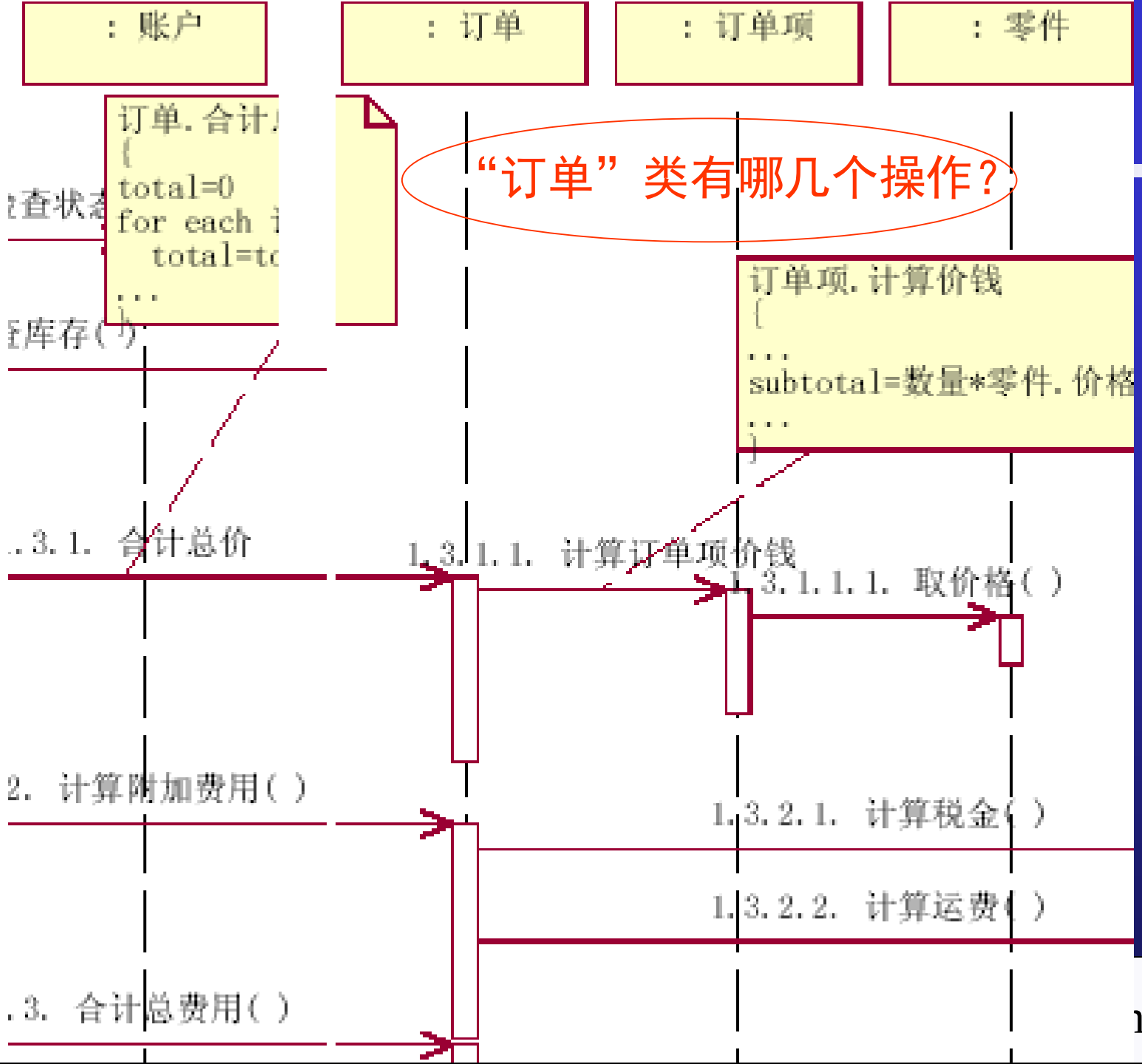
消息传入：类的操作——责任

序列图



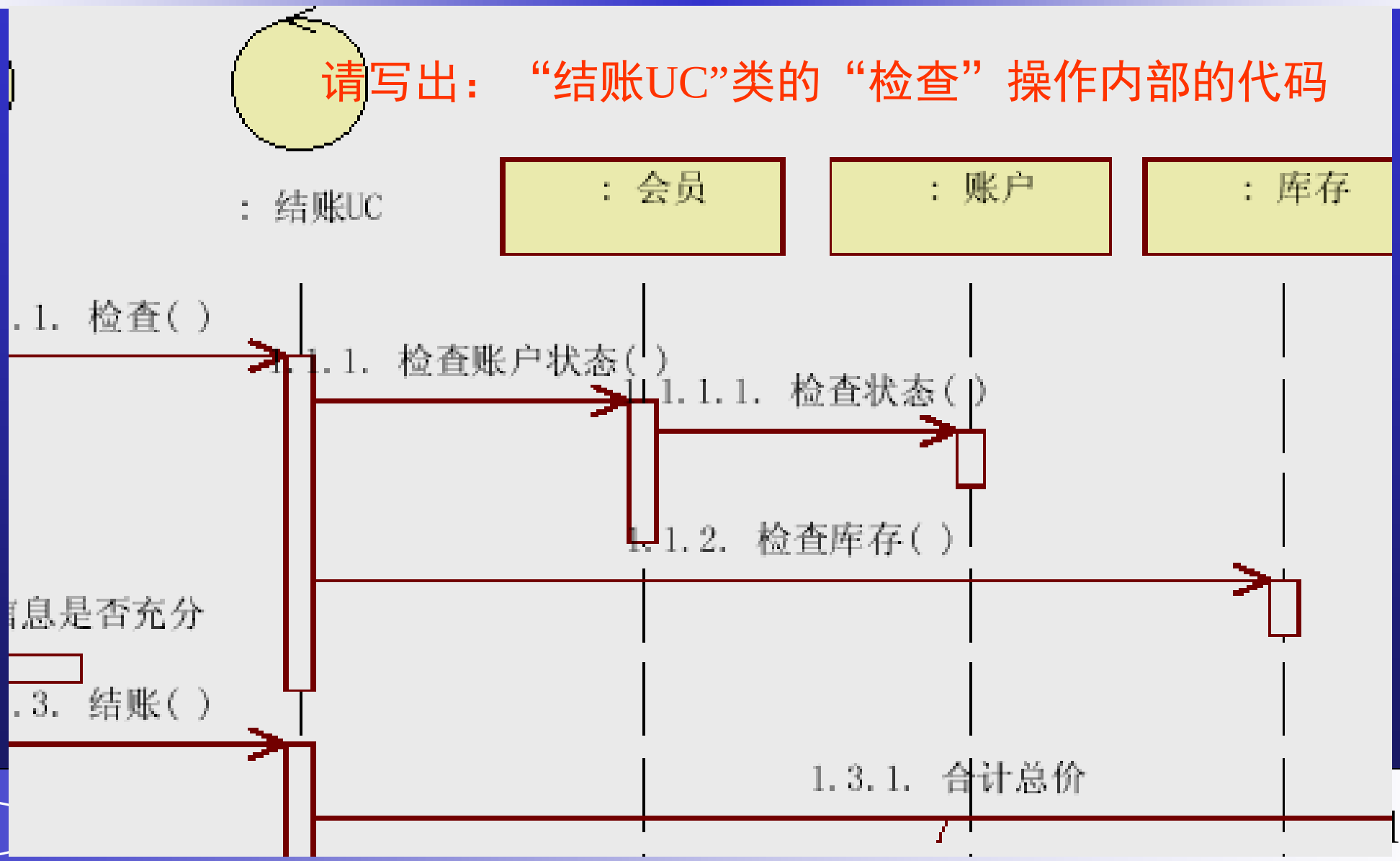
消息传出：类完成操作所需合作——协作

序列图

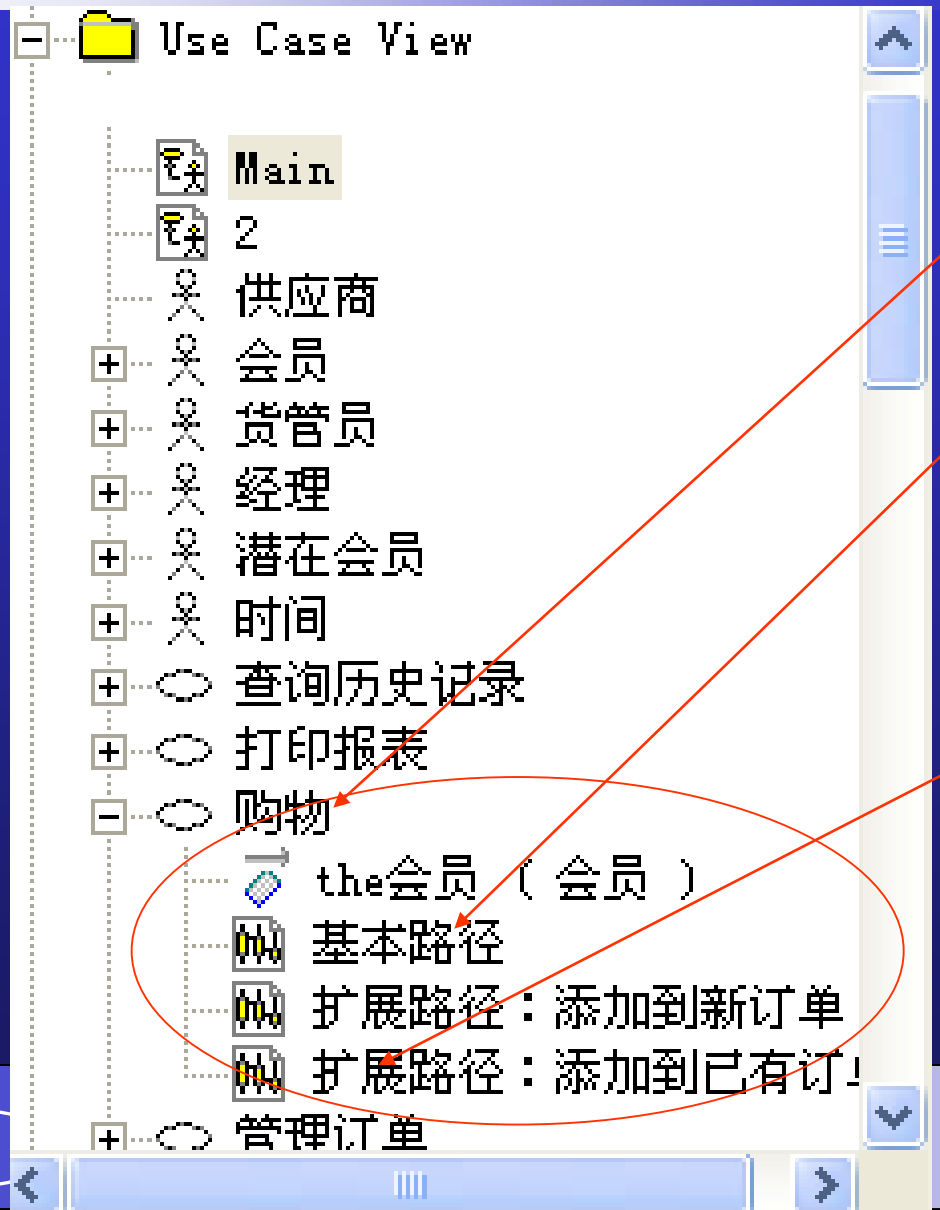

：账户
：订单
：订单项
：零件
订单.合计
{
total=0
for each
total=t
...
}
“订单”类有哪几个操作？
订单项.计算价钱
{
...
subtotal=数量*零件.价格
...
}
.3.1. 合计总价
1.3.1.1. 计算订单项价钱
1.3.1.1.1. 取价格( )
2. 计算附加费用( )
1.3.2.1. 计算税金( )
1.3.2.2. 计算运费( )
.3. 合计总费用( )

序列图

请写出：“结账UC”类的“检查”操作内部的代码

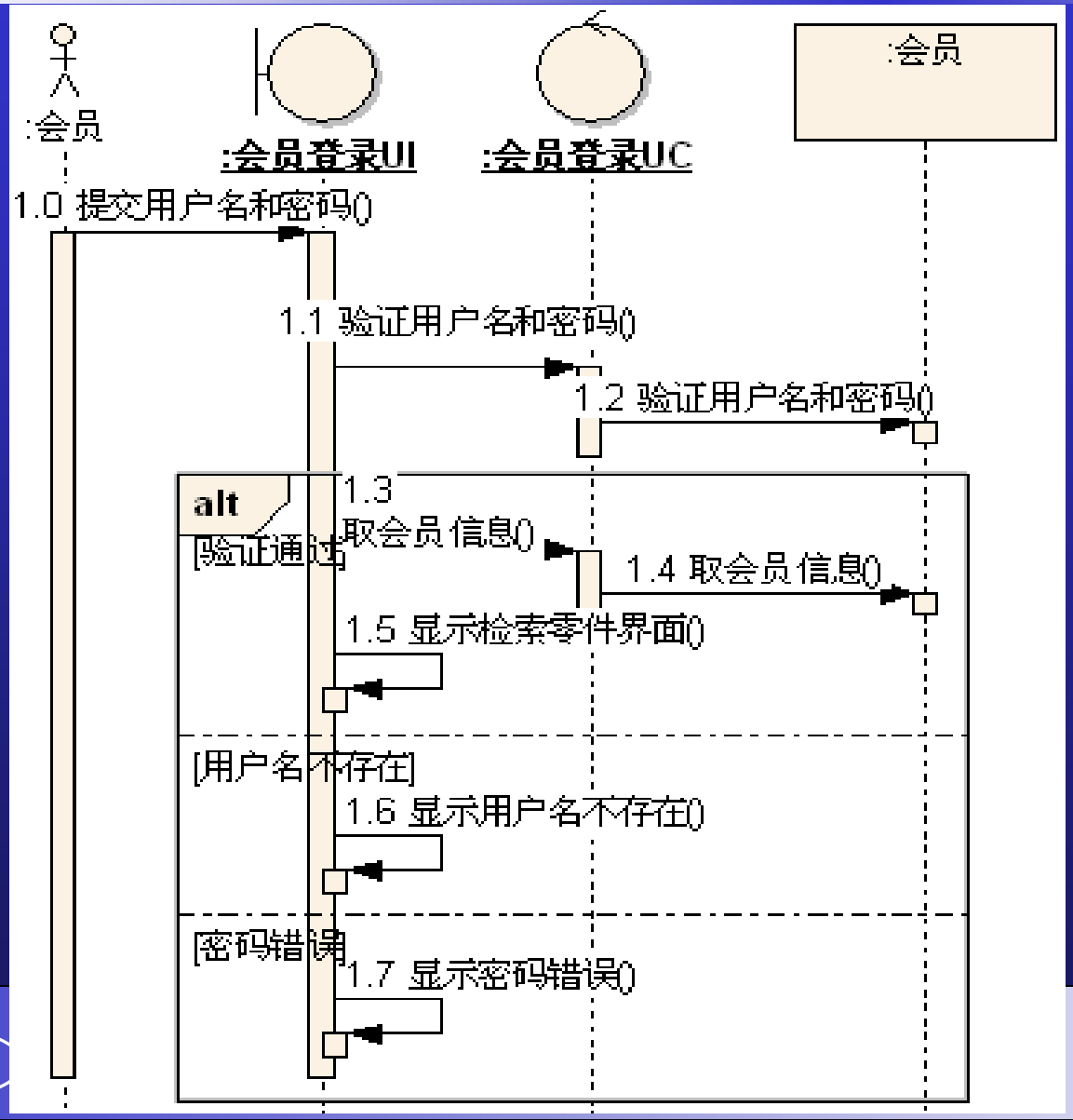


序列图



- 位置：每个用例下面，对应用例的路径
- 基本路径：一张图
- 简单的扩展点：可以合并到基本路径图
- 复杂扩展点：单独一张图，和基本路径图间链接

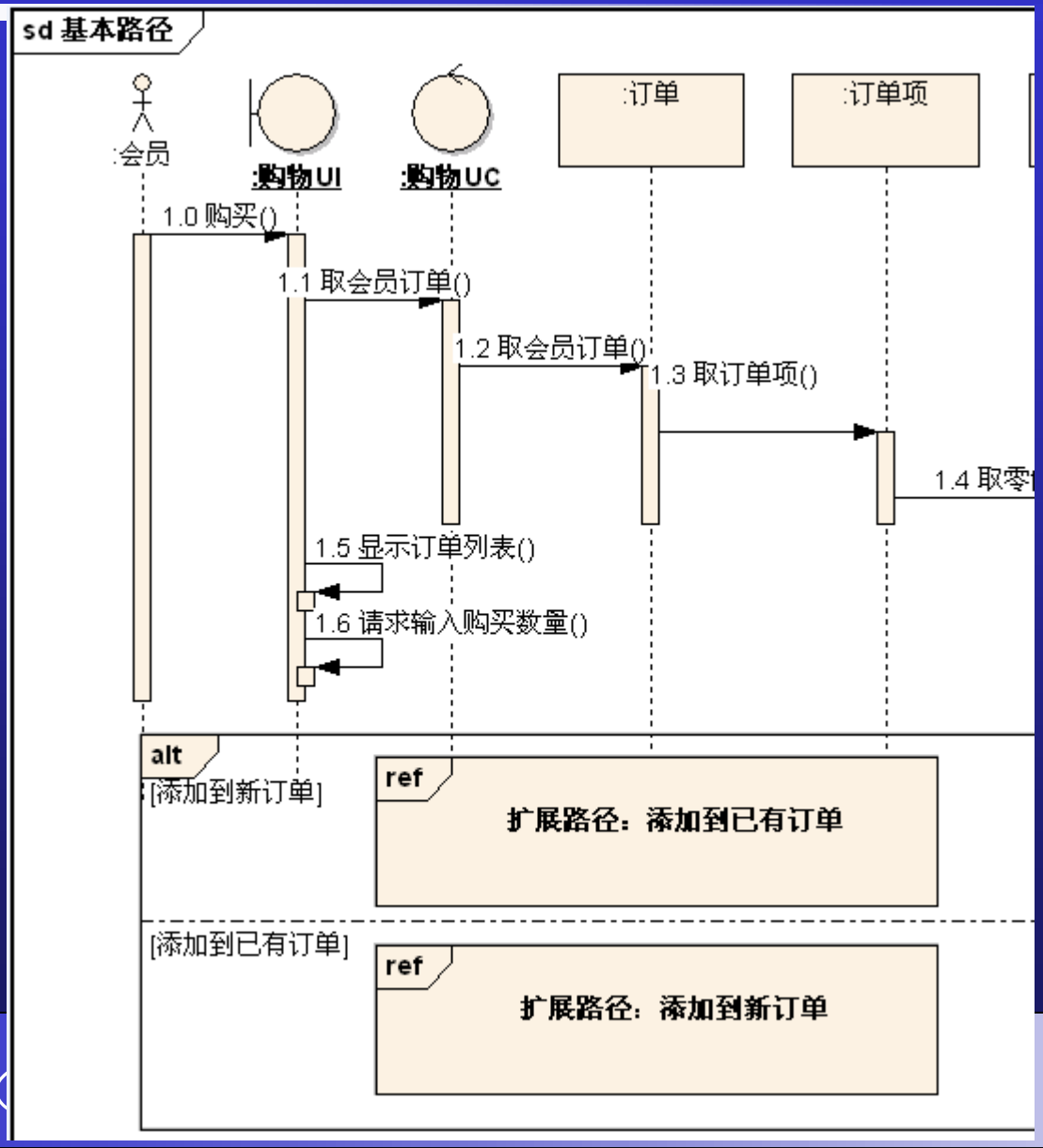
序列图



简单扩展点：

可以合并到基本路径图

序列图

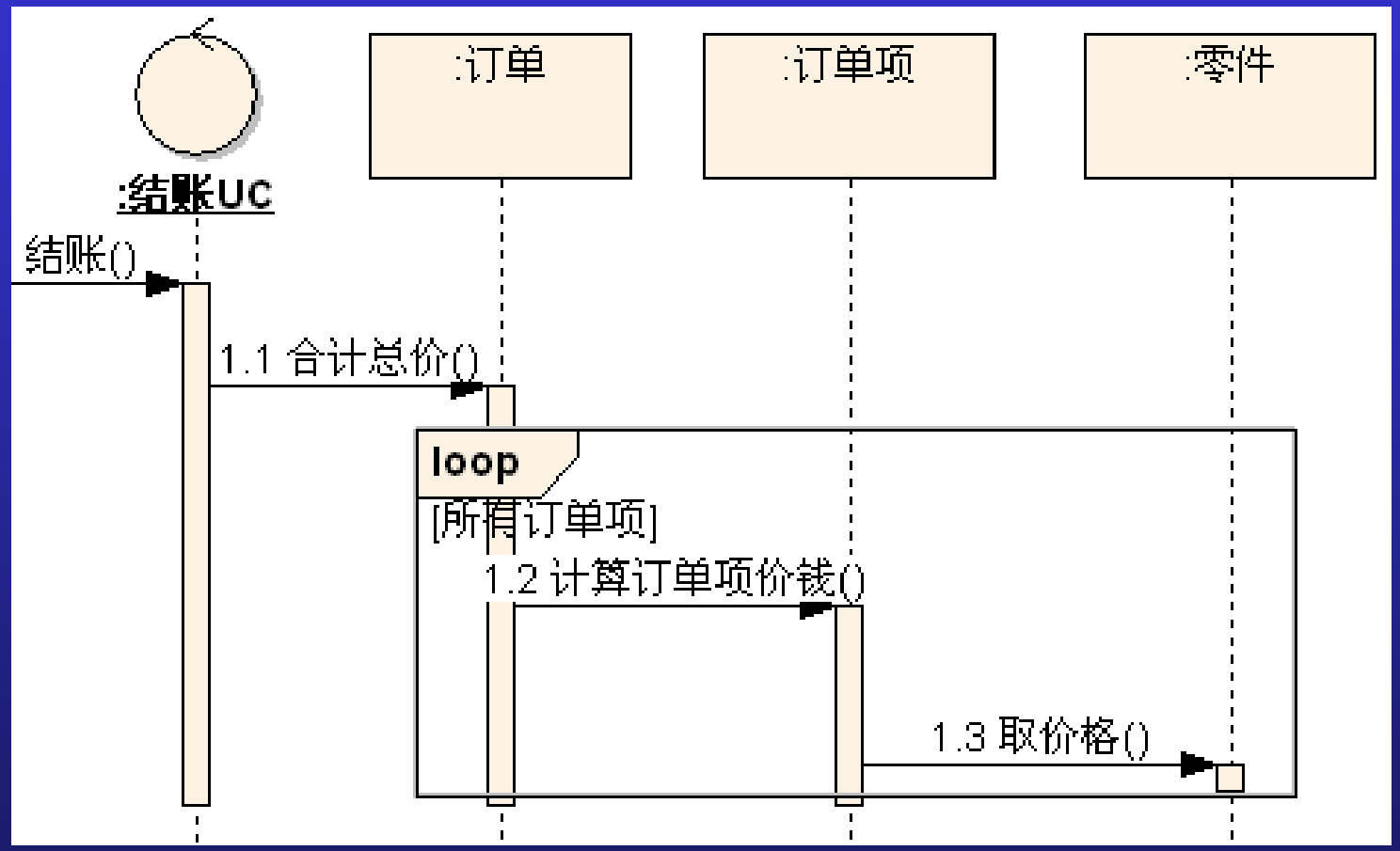


复杂扩展点：

单独一张图，在基本路径图引用

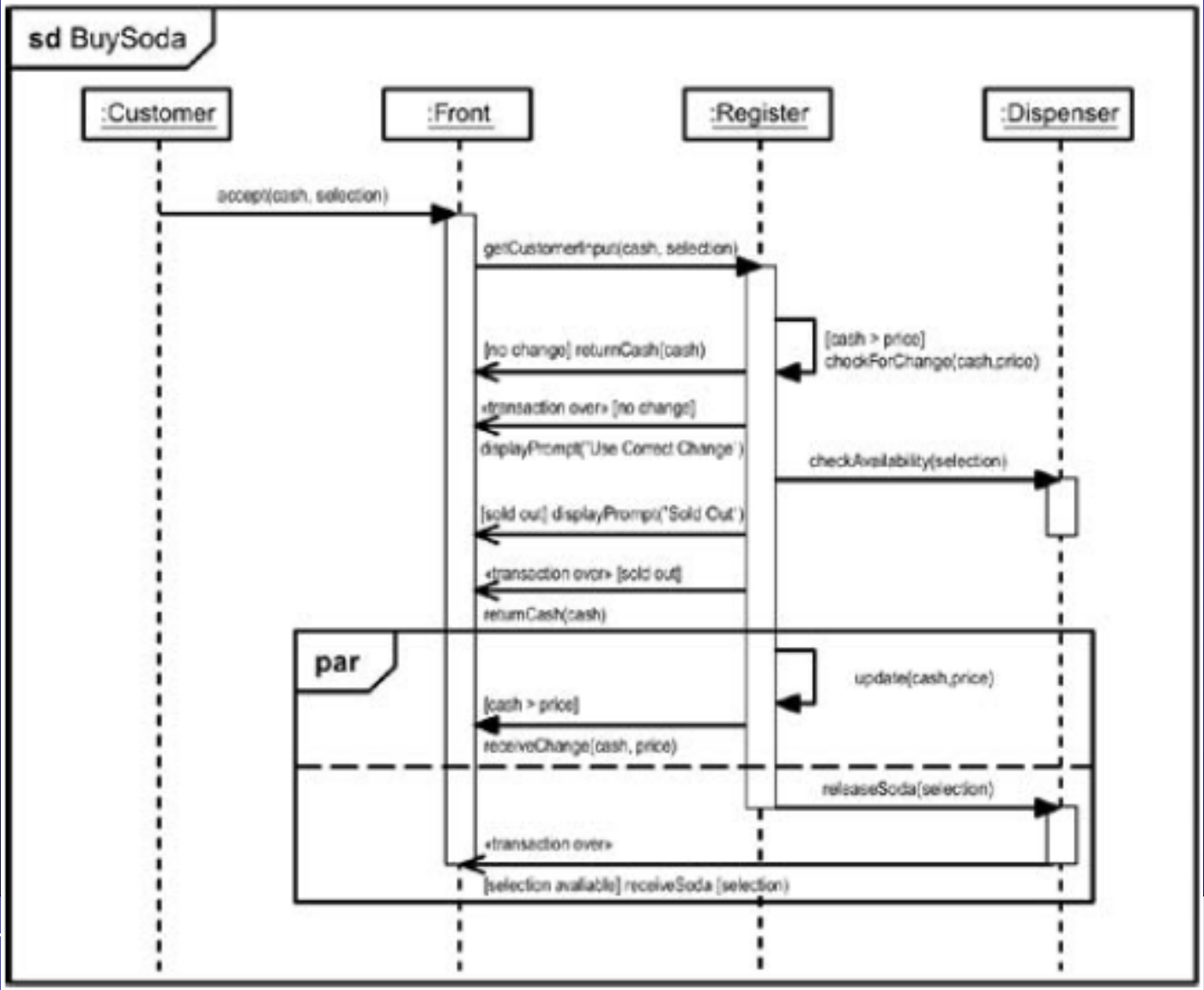
Include、Extend用例也适用

序列图



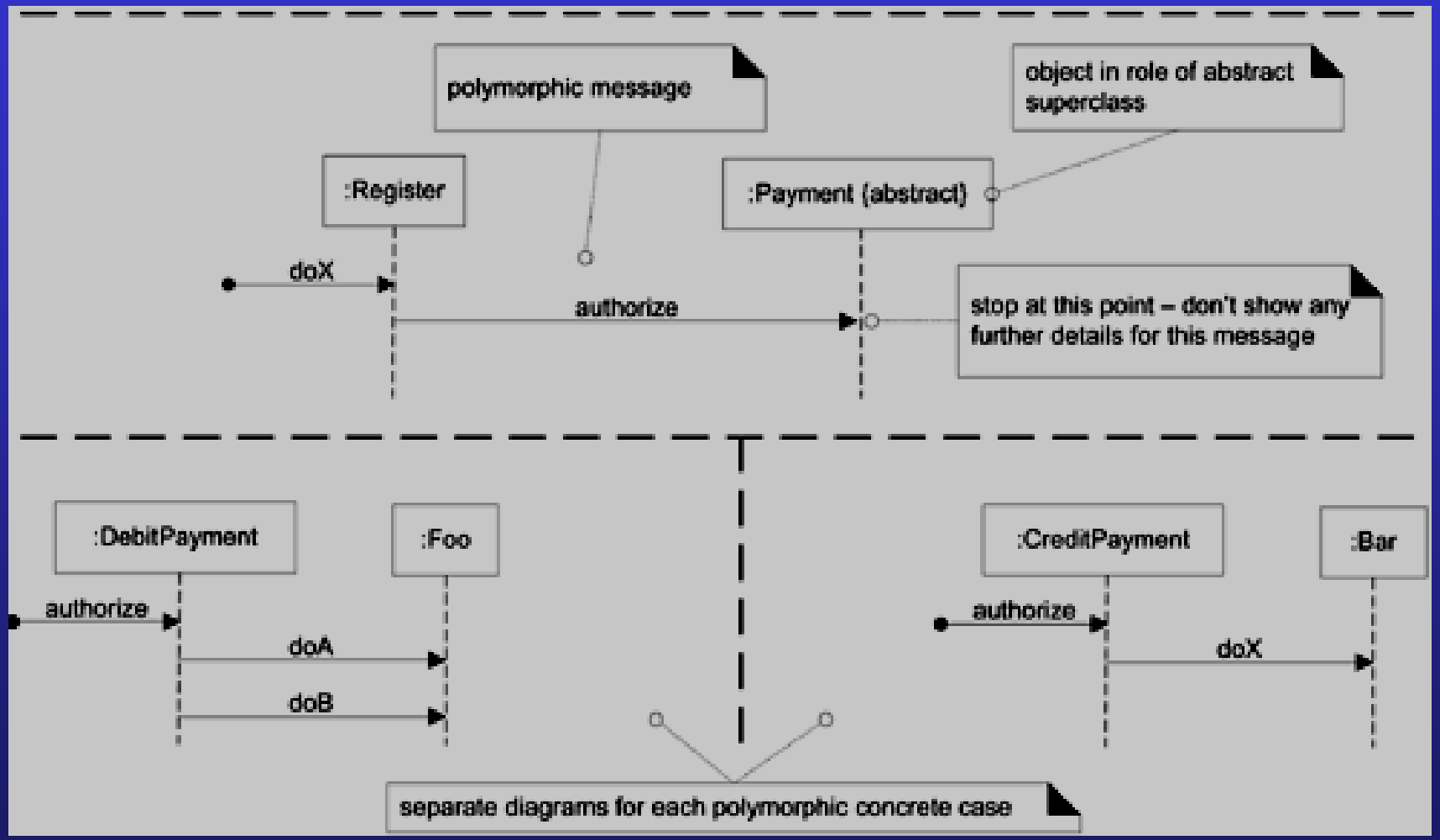
循环

序列图


sd BuySoda
:Customer
:Front
:Register
:Dispenser
accept(cash, selection)
getCustomerInput(cash, selection)
[cash > price]
checkForChange(cash,price)
[no change] returnCash(cash)
«transaction over» [no change]
displayPrompt("Use Correct Change")
checkAvailability(selection)
[sold out] displayPrompt("Sold Out")
«transaction over» [sold out]
returnCash(cash)
par
update(cash,price)
[cash > price]
receiveChange(cash, price)
releaseSoda(selection)
«transaction over»
[selection available] receiveSoda (selection)

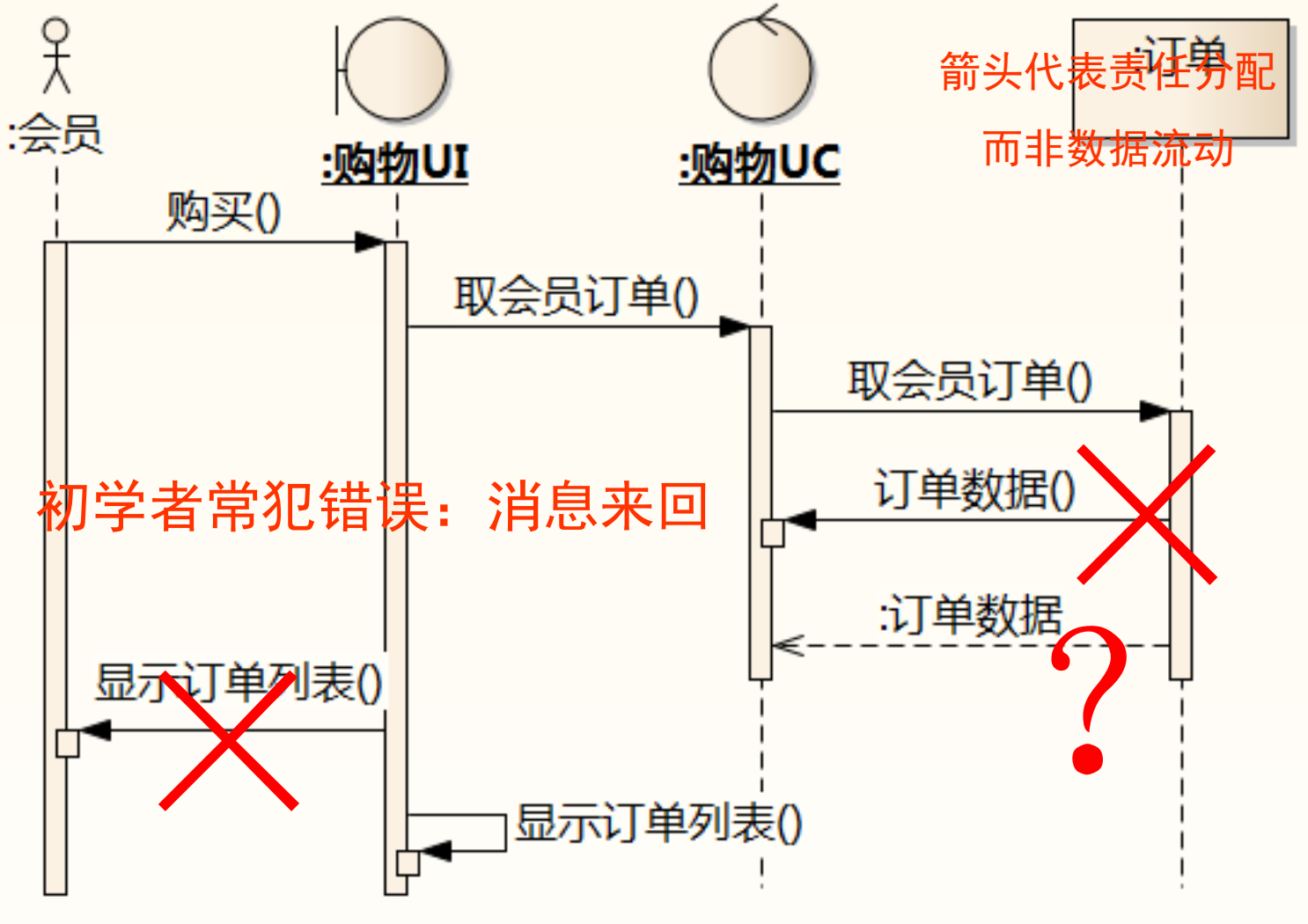
并行

序列图

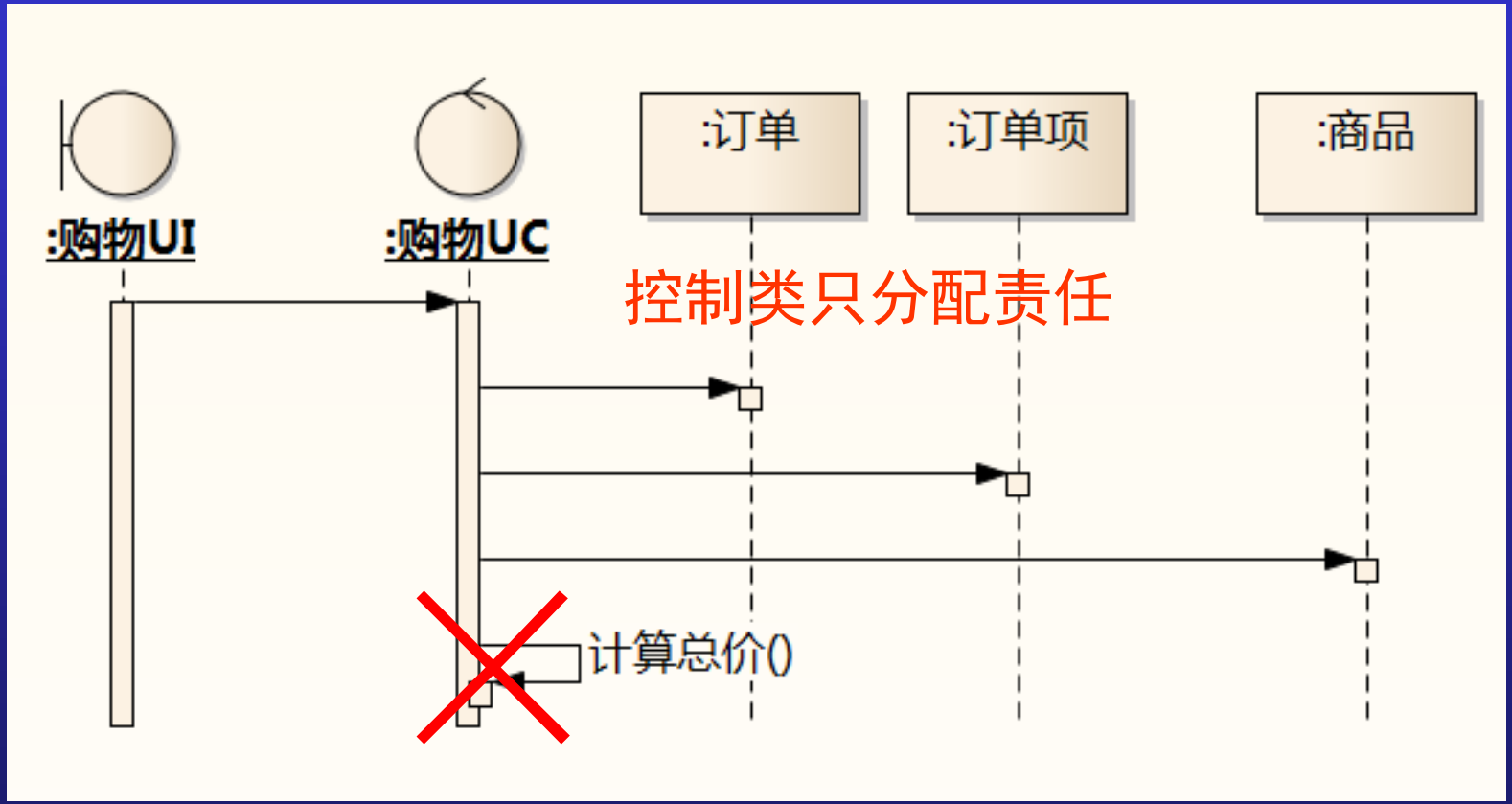

polymorphic message
object in role of abstract superclass
:Register
:Payment (abstract)
doX
authorize
stop at this point – don't show any further details for this message
:DebitPayment
:Foo
:CreditPayment
:Bar
authorize
doA
doB
authorize
doX
separate diagrams for each polymorphic concrete case

多态

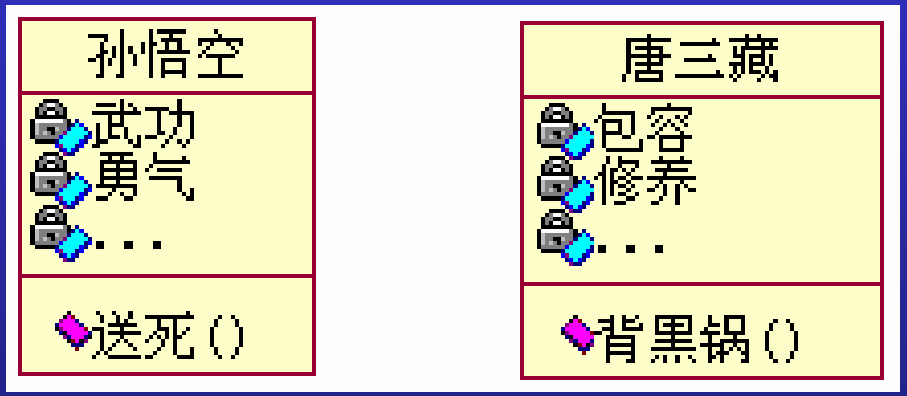
序列图


:会员
:购物UI
:购物UC
:订单
箭头代表责任分配
而非数据流动
购买()
取会员订单()
取会员订单()
订单数据()
:订单数据
显示订单列表()
显示订单列表()
初学者常犯错误：消息来回

序列图



责任分配



背黑锅我来，送死你去……

责任分配

低耦合，高内聚

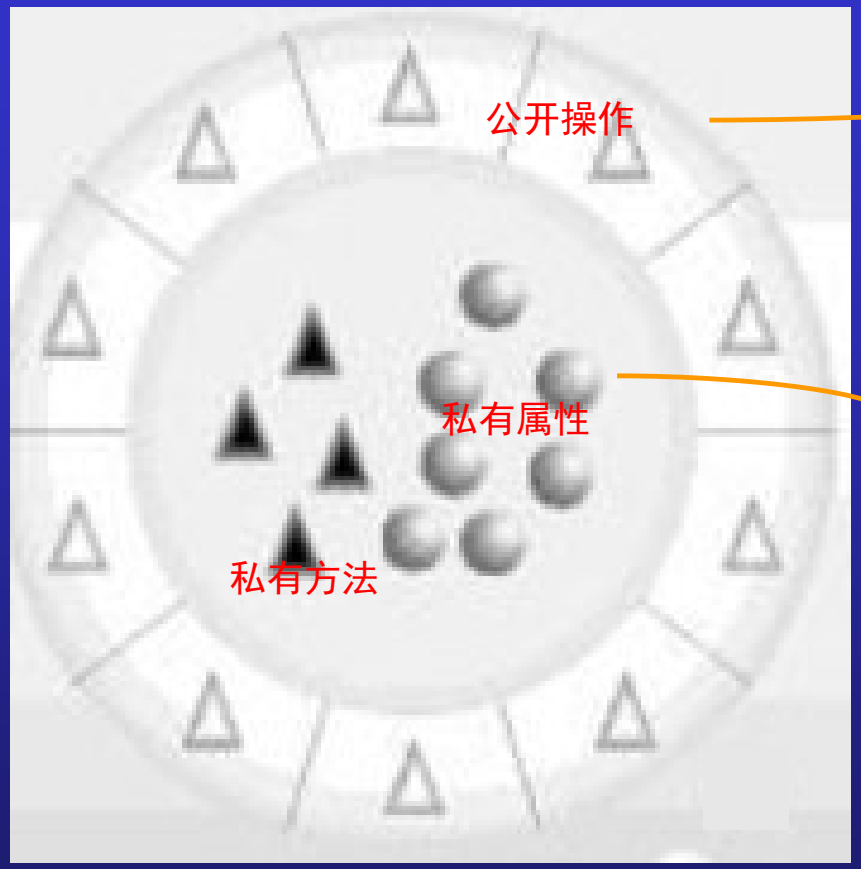
总原则

责任分配

- 专家原则——资源决定消息内容
- 老板原则——由老板发送消息给我
- 可视(Demeter)原则——只发消息给朋友

交互原则

专家原则



公开的契约——卖

使用者假定其不会变化

内部的实现——做

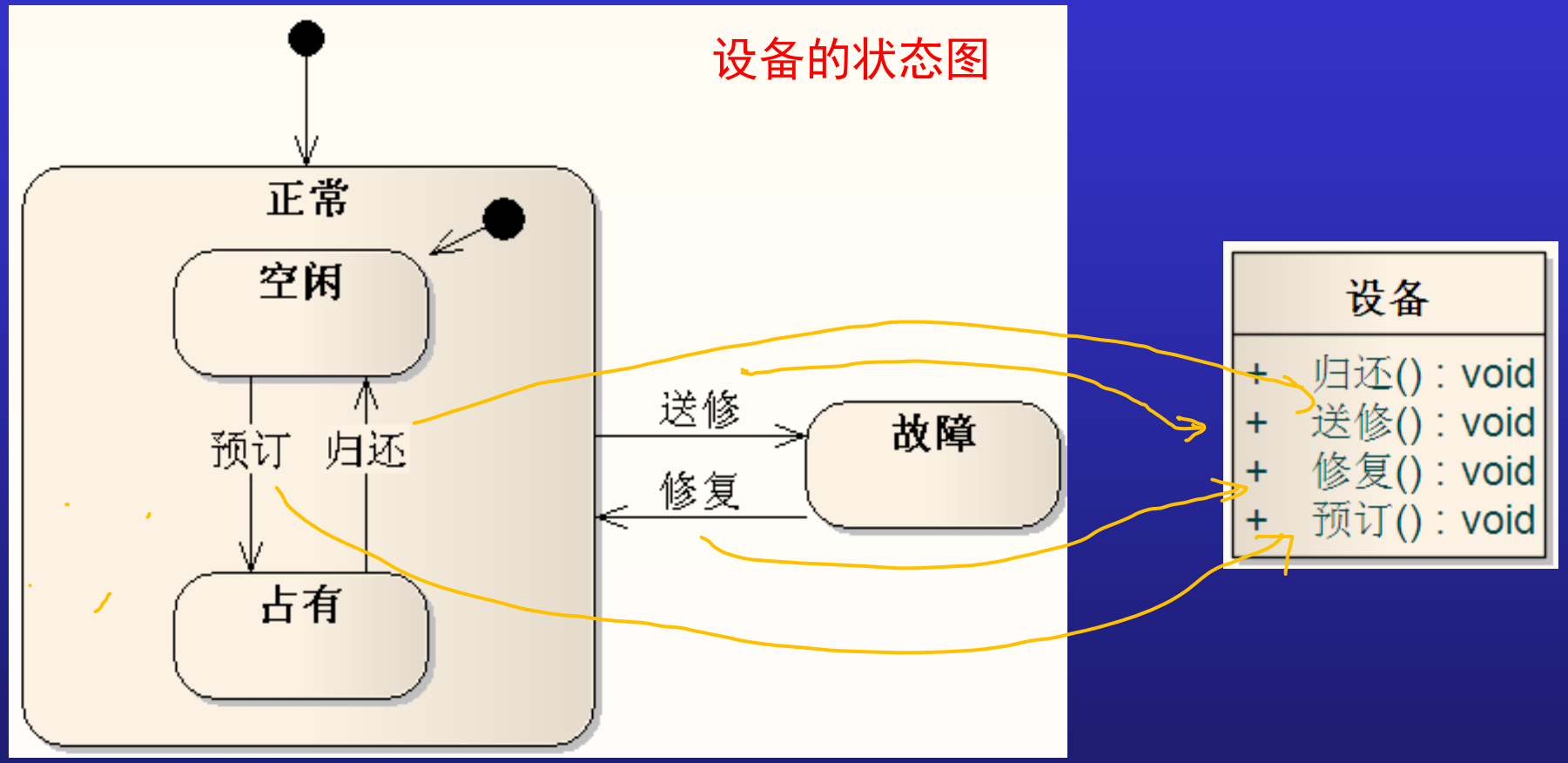
制造者可以随意抽换

“做”污染“卖”

危害大

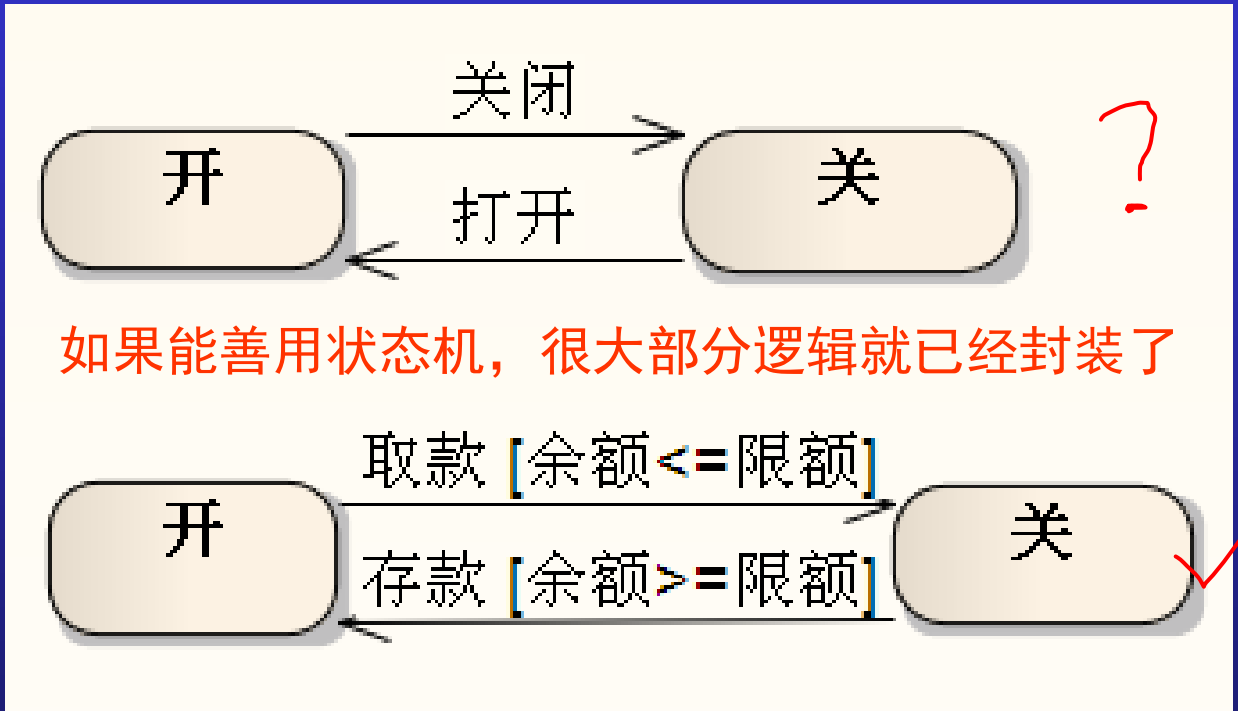
类的责任

专家原则



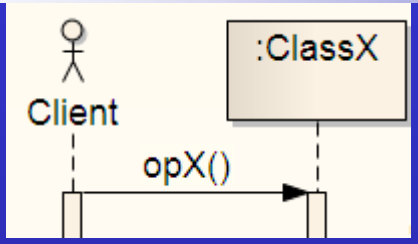
状态机有助于定义恰当的契约

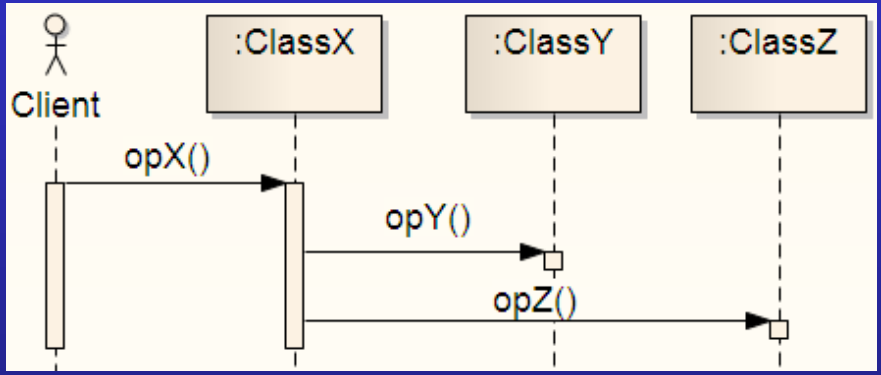
专家原则

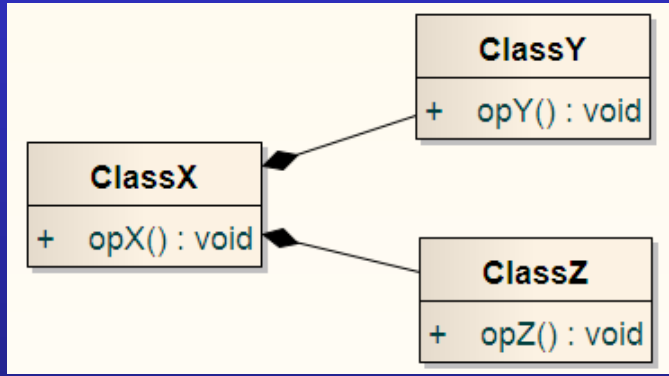


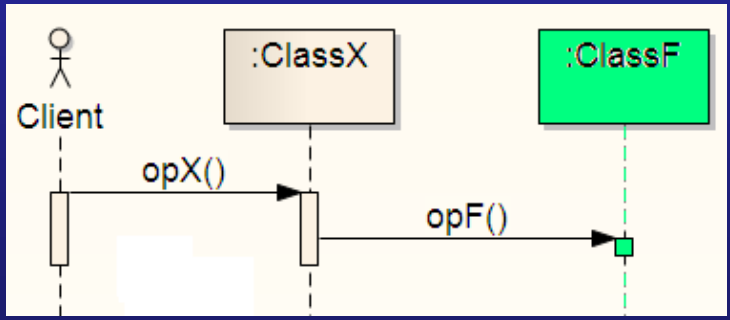
状态图

专家原则





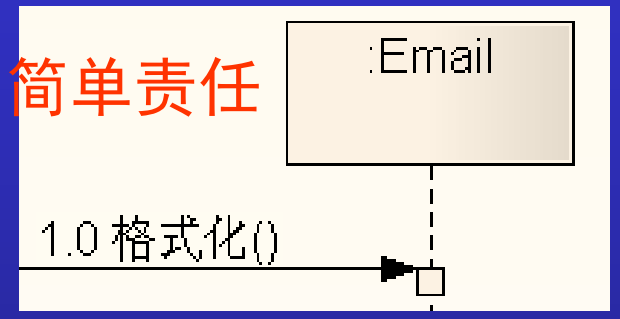


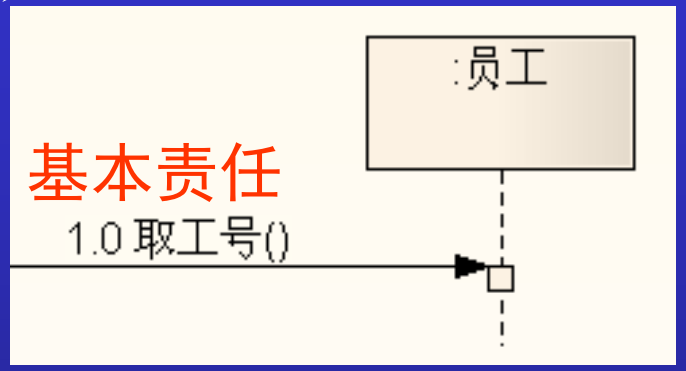


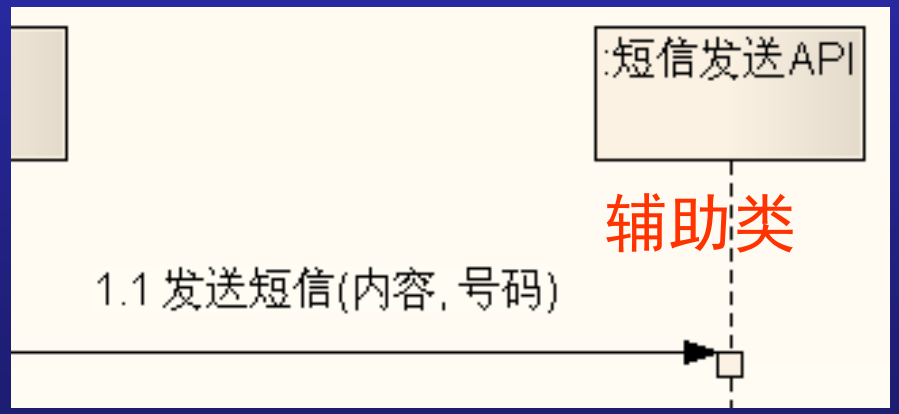
类适合接受的责任

专家原则

考虑到基础设施，独立也只是某个层面上的说法

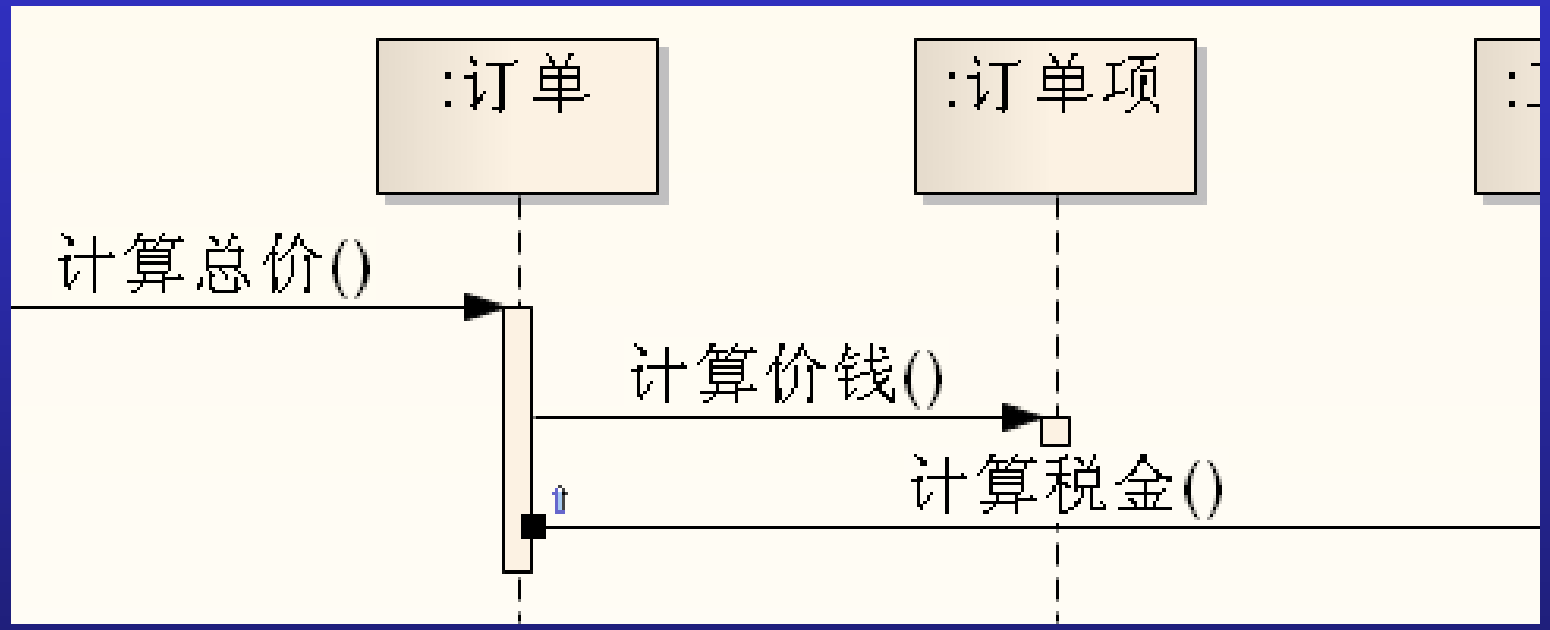






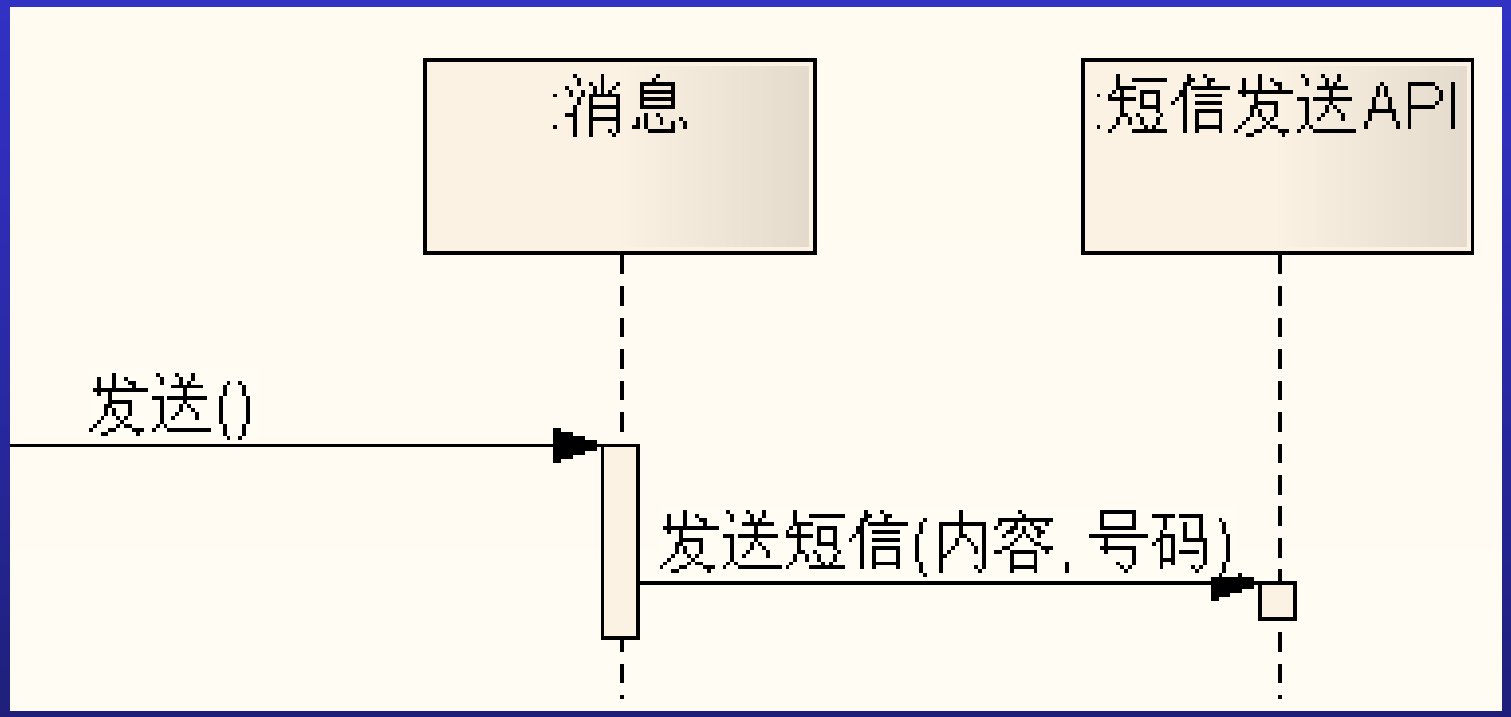
独立完成

专家原则



分解大责任

专家原则



委托给辅助类

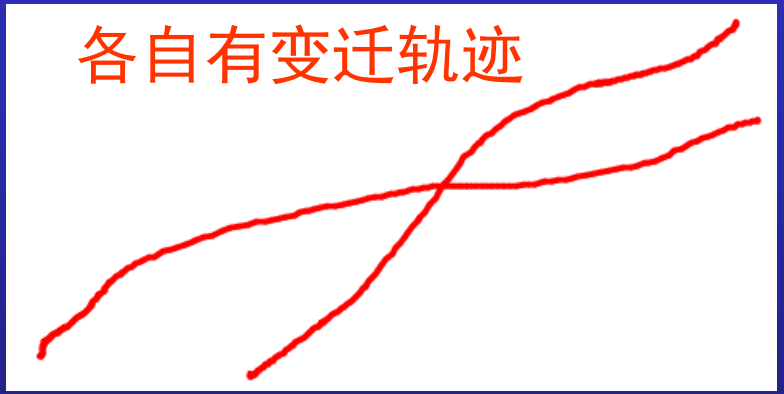
专家原则

表 1　固体的线胀系数 α(℃$^{-1}$)

物　质	t(℃)	α($\times10^{-6}$)
铝	25	25
金	25	14.2
银	25	19
铜	25	16.6
钨	0～100	4.5
铁	25	12.0
铂	25	9.0
黄铜(68 Cu, 32 Zn)	25	18～19
殷钢(36 Ni, 64 Fe)	0～100	0.8～12.8

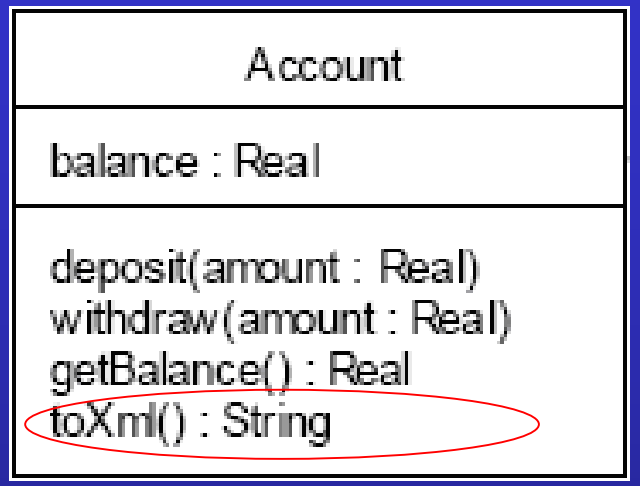
不同的膨胀系数

导致结合不能长久



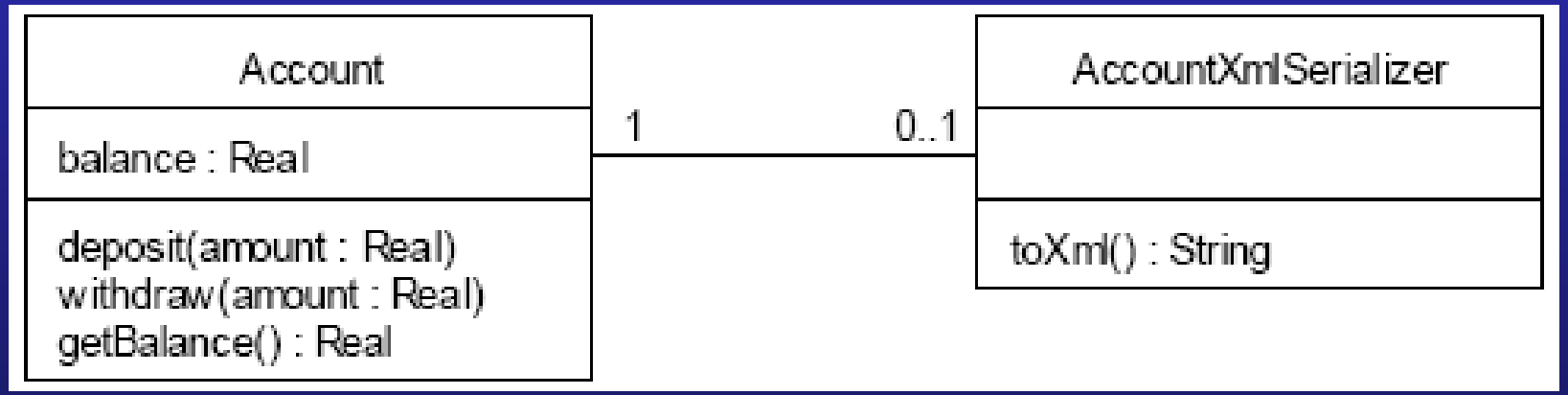
专家要专——SRP（单一责任原则）

专家原则



一个类只有一个变化原因

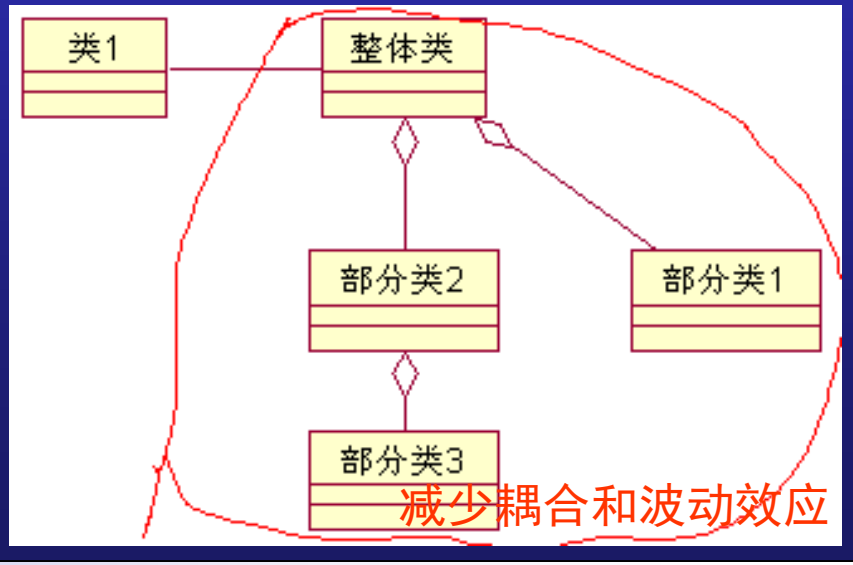
领域知识是瓶颈



SRP

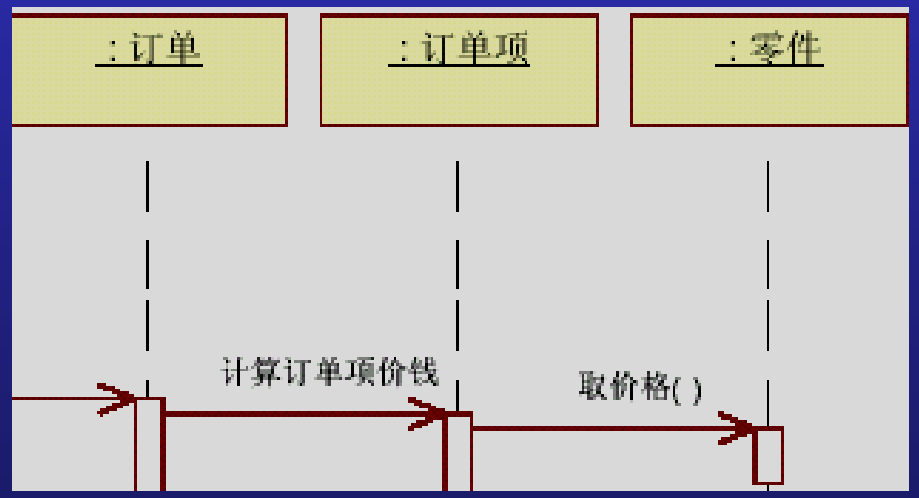
老板原则

- 由老板传递消息
- 当出现以下情况时，发给A的消息先通过B处理和中转
  - B聚合A（Aggregation）
  - B组合A（ Composition ）

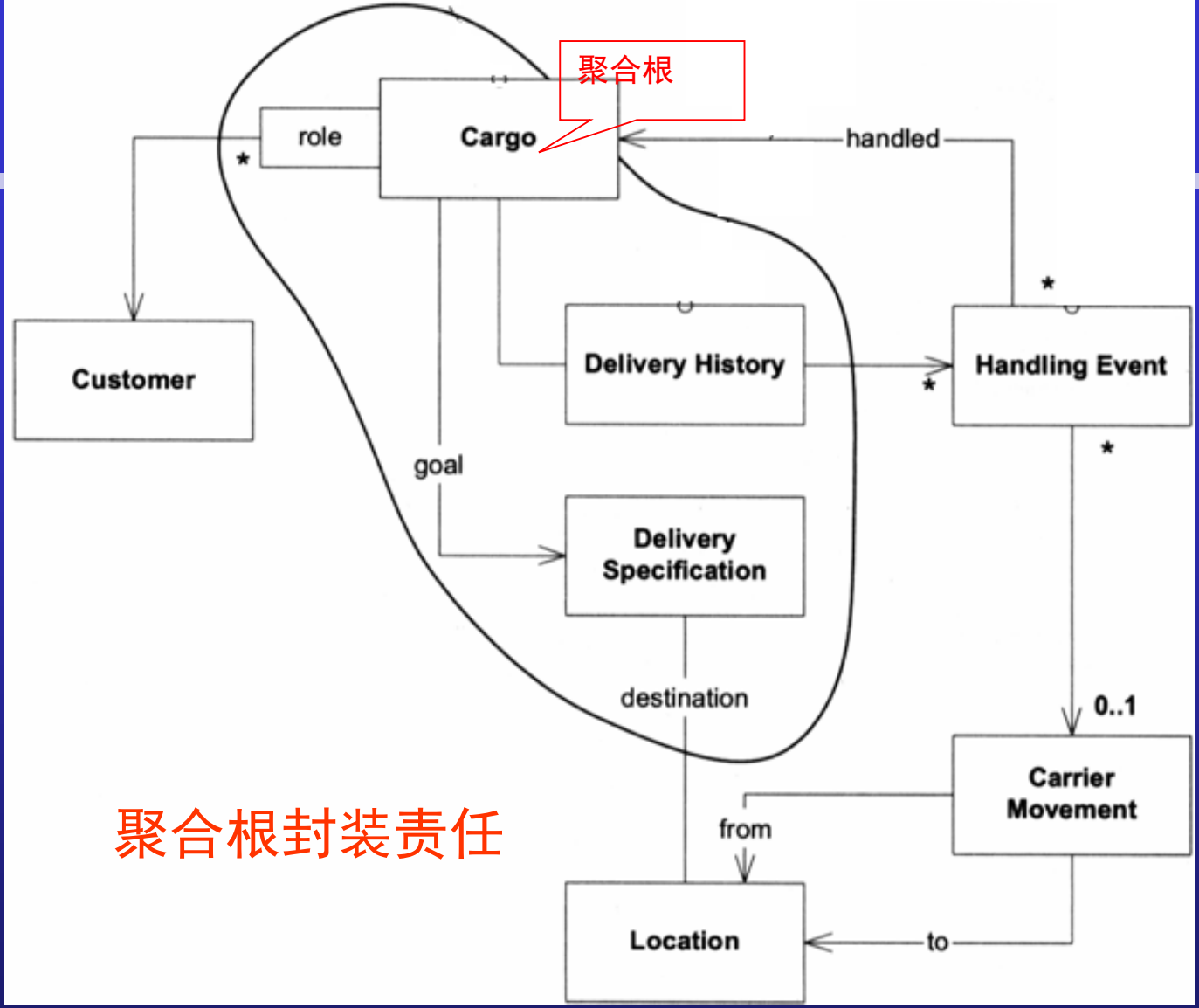


老板原则


订单
下单日期
税金
运费
总价
1
0..*
订单项
购买数量
价钱
计算价钱()
0..*
1
零件
编号
详细信息
价格
取价格()

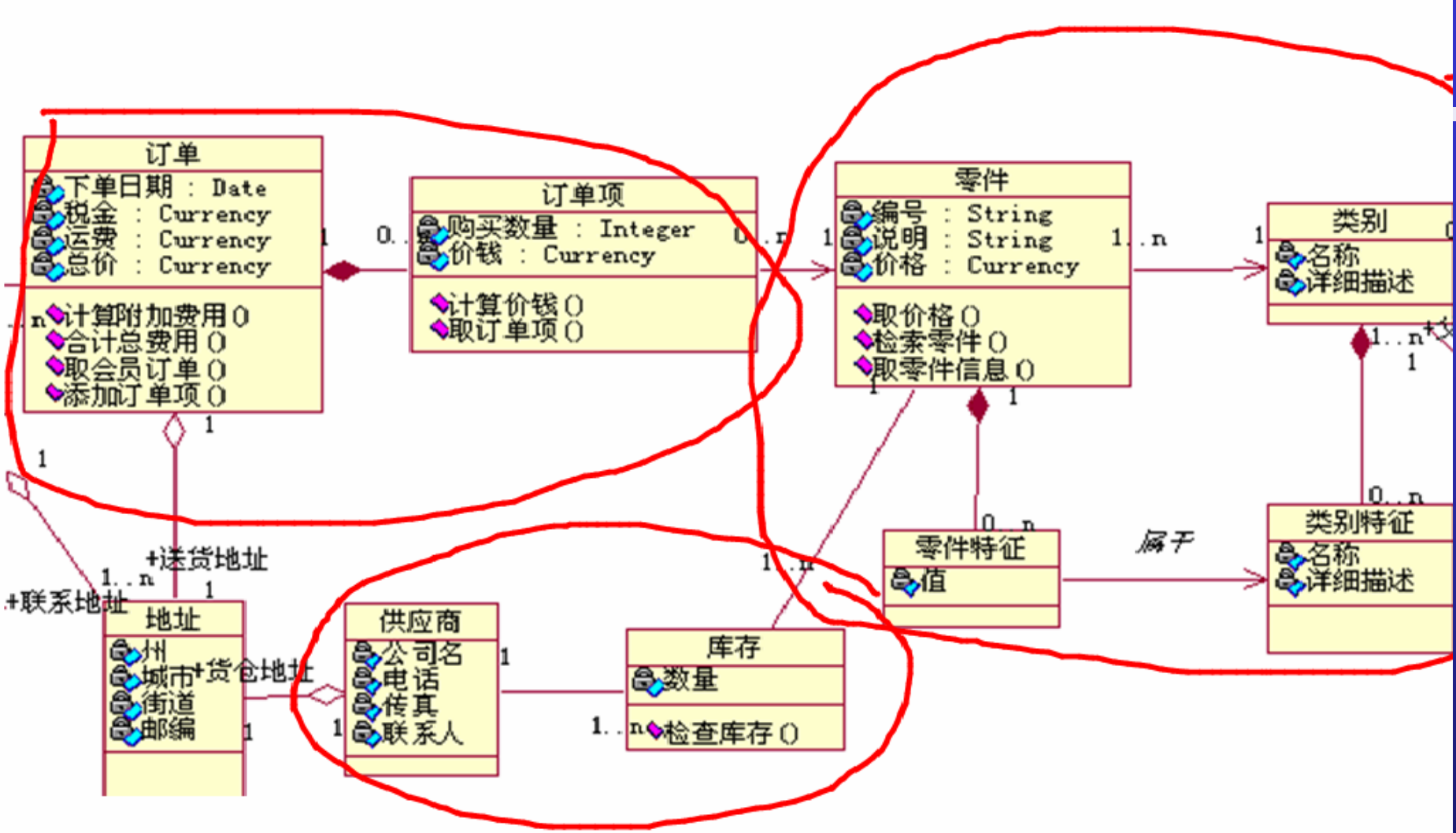

:订单
:订单项
:零件
计算订单项价钱
取价格( )

老板原则


聚合根
role
Cargo
handled
Customer
Delivery History
Handling Event
goal
Delivery Specification
destination
0..1
Carrier Movement
聚合根封装责任
from
Location
to

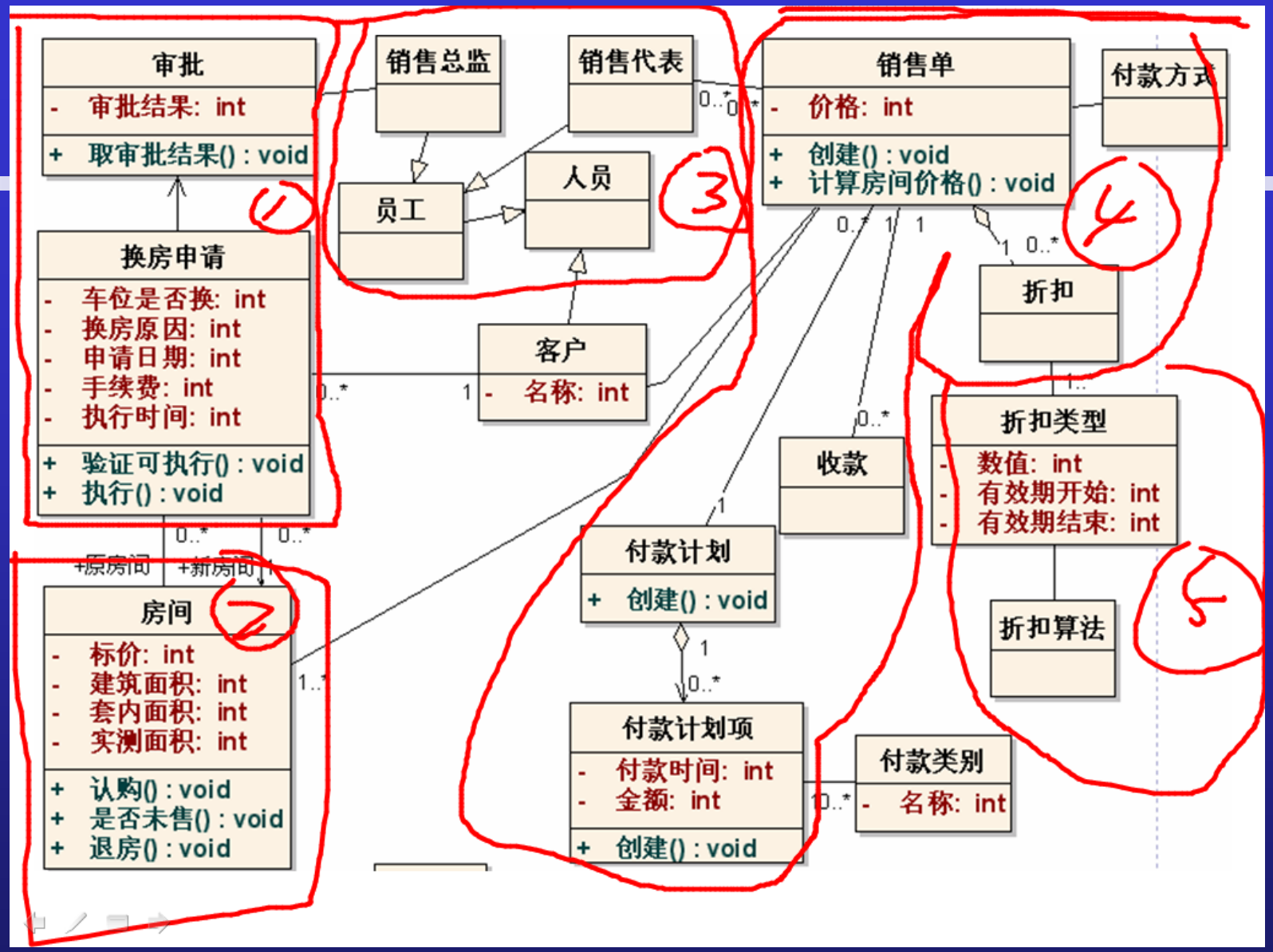
聚合根

老板原


订单
下单日期 : Date
税金 : Currency
运费 : Currency
总价 : Currency
计算附加费用()
合计总费用()
取会员订单()
添加订单项()
订单项
购买数量 : Integer
价钱 : Currency
计算价钱()
取订单项()
零件
编号 : String
说明 : String
价格 : Currency
取价格()
检索零件()
取零件信息()
类别
名称
详细描述
零件特征
值
属于
类别特征
名称
详细描述
+送货地址
+联系地址
地址
州
城市
街道
邮编
+货仓地址
供应商
公司名
电话
传真
联系人
库存
数量
检查库存()

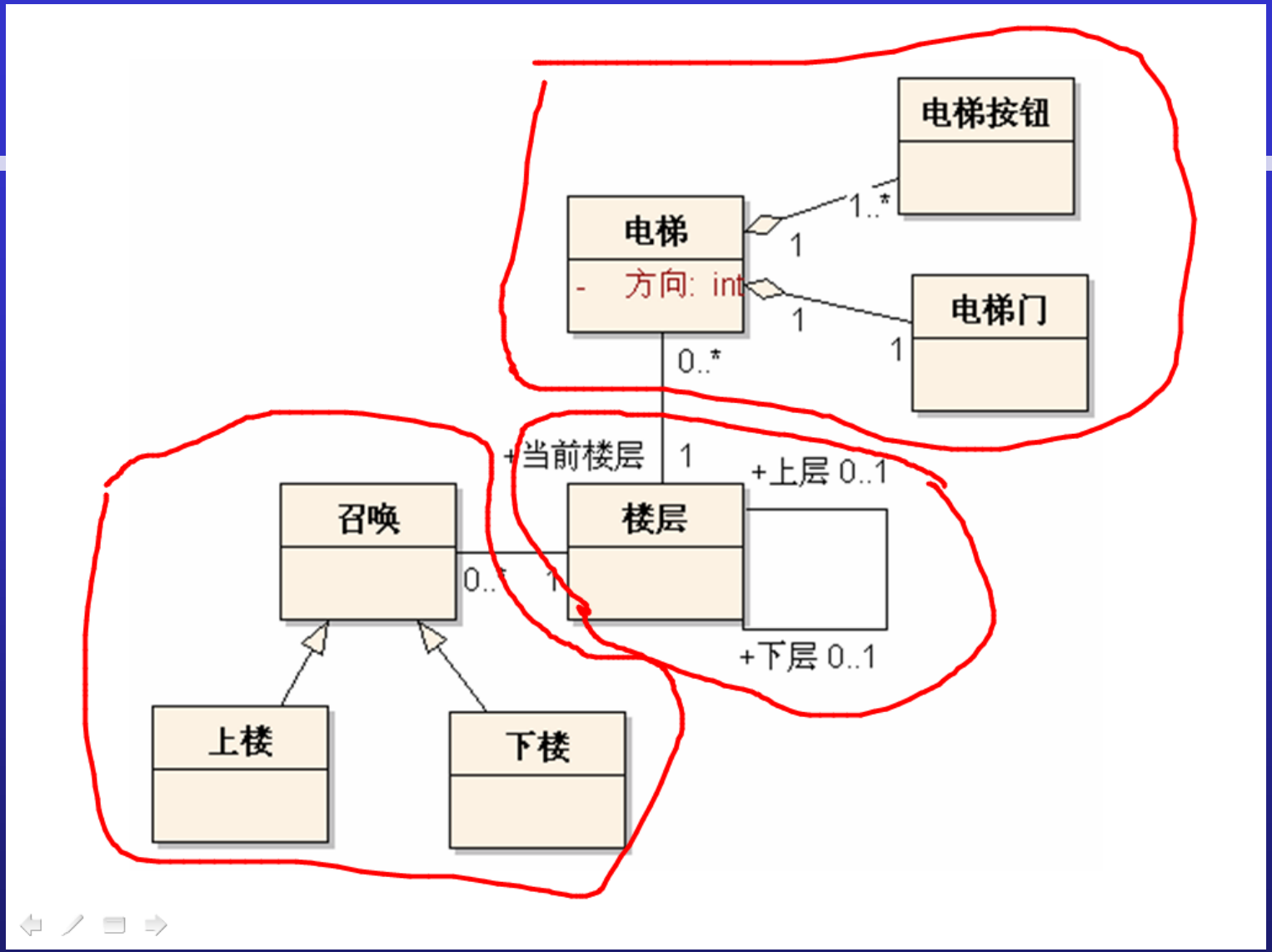
知识的分区

老板原则


审批
- 审批结果: int
+ 取审批结果() : void
换房申请
- 车位是否换: int
- 换房原因: int
- 申请日期: int
- 手续费: int
- 执行时间: int
+ 验证可执行() : void
+ 执行() : void
房间
- 标价: int
- 建筑面积: int
- 套内面积: int
- 实测面积: int
+ 认购() : void
+ 是否未售() : void
+ 退房() : void
+原房间
+新房间
销售总监
销售代表
员工
人员
客户
- 名称: int
销售单
- 价格: int
+ 创建() : void
+ 计算房间价格() : void
付款方式
折扣
折扣类型
- 数值: int
- 有效期开始: int
- 有效期结束: int
折扣算法
收款
付款计划
+ 创建() : void
付款计划项
- 付款时间: int
- 金额: int
+ 创建() : void
付款类别
- 名称: int
1
2
3
4
5

知识的分区

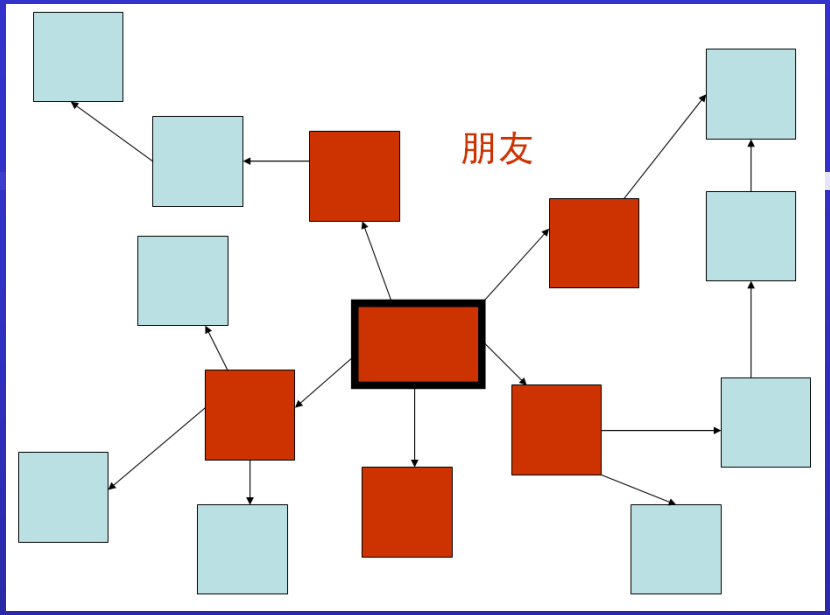
老板原则



知识的分区

可视原则

不要和陌生人说话



只能发消息给：

- 自己
- 方法参数中的对象
- 属性引用的对象（关联）
- 你创建的对象

Demeter法则

可视原则

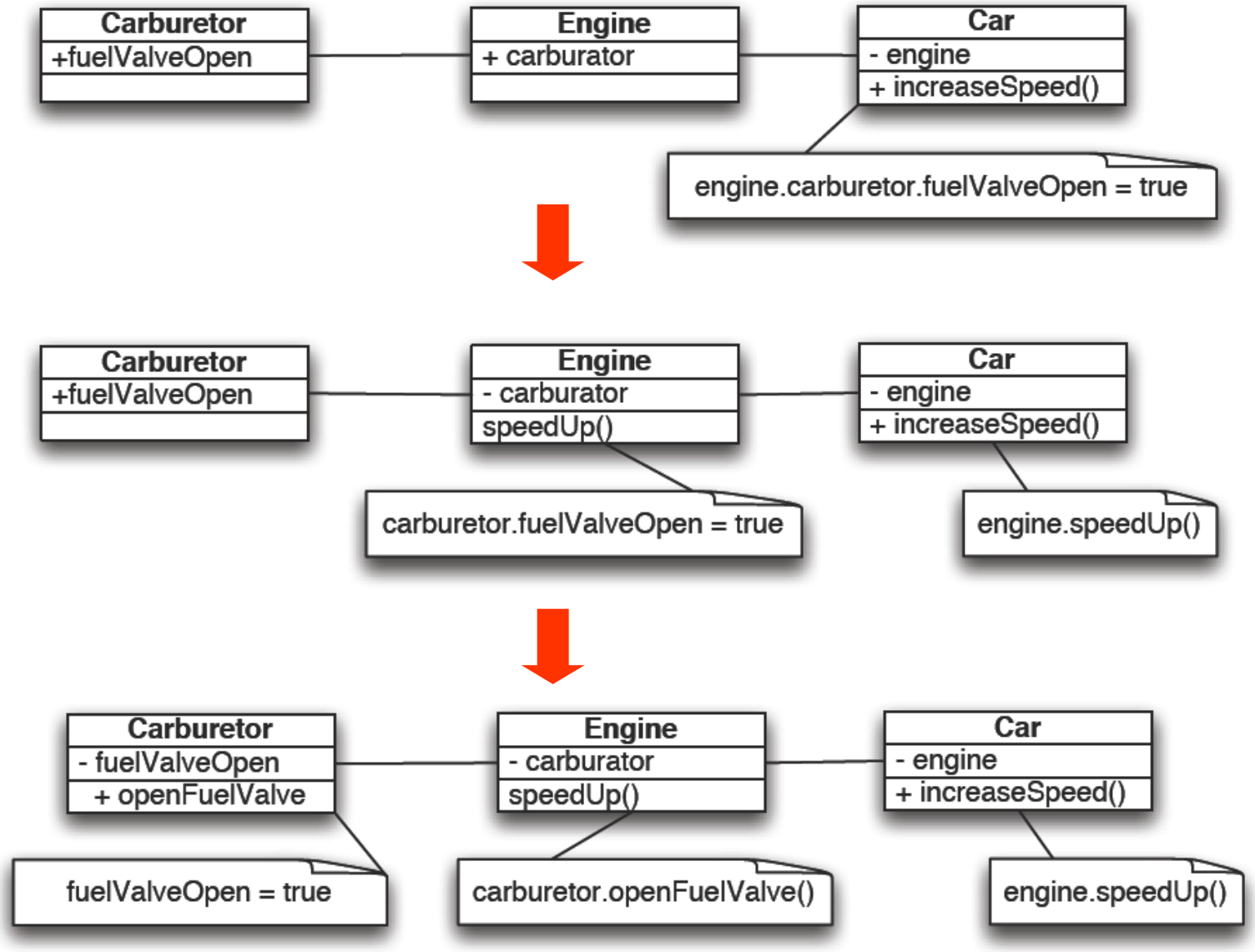
```
public class sample {
  private ObjectA a;
  private int function();
  public void example(ObjectB b) {
  ObjectC c;
  in f = function();
  b.invert();
  a = new.ObjectA();
  a.setActive();
  c.print();
  }
}
```

The Law of Demeter for functions states that any method of an object should call only methods belonging to:

- *itself*
- *any parameters that were passed in to the method*
- *any objects it created*
- *any directly held component object*

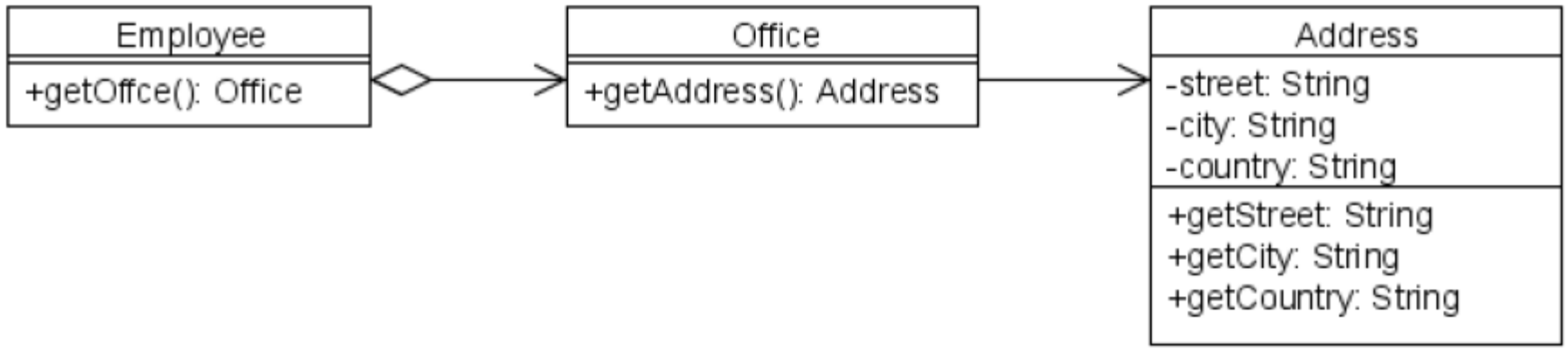
Demeter法则

可视原则



Demeter法则

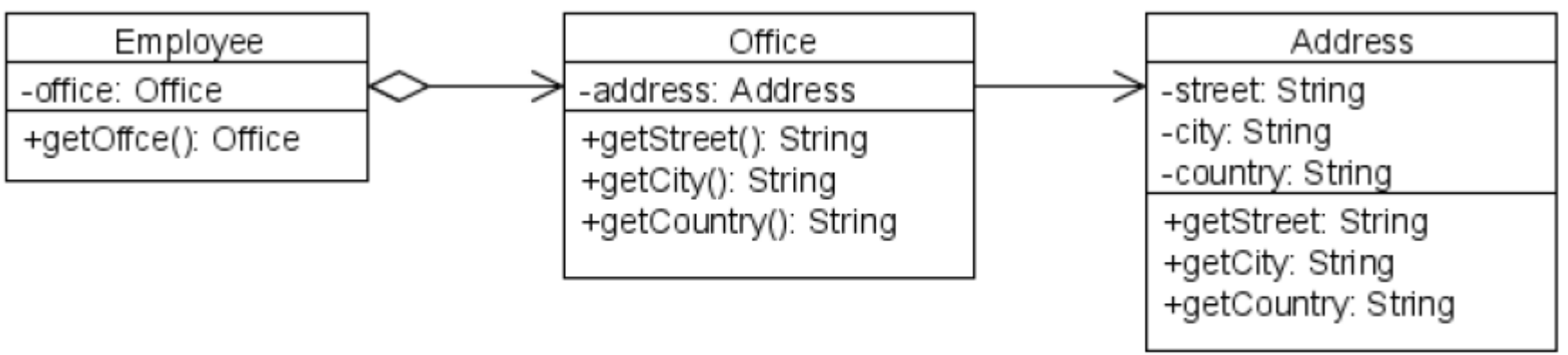
可视原则




```
String employeeStreet = this.office.getAddress().getStreet();
```

Demeter法则

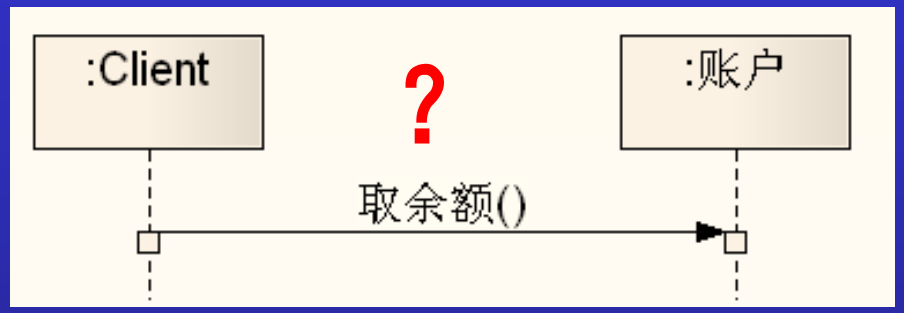
可视原则

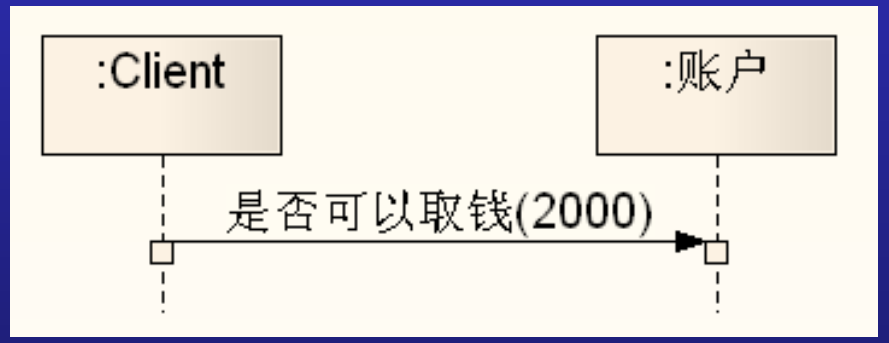


String employeeStreet = this.office.getStreet();

Demeter法则—修改

可视原则

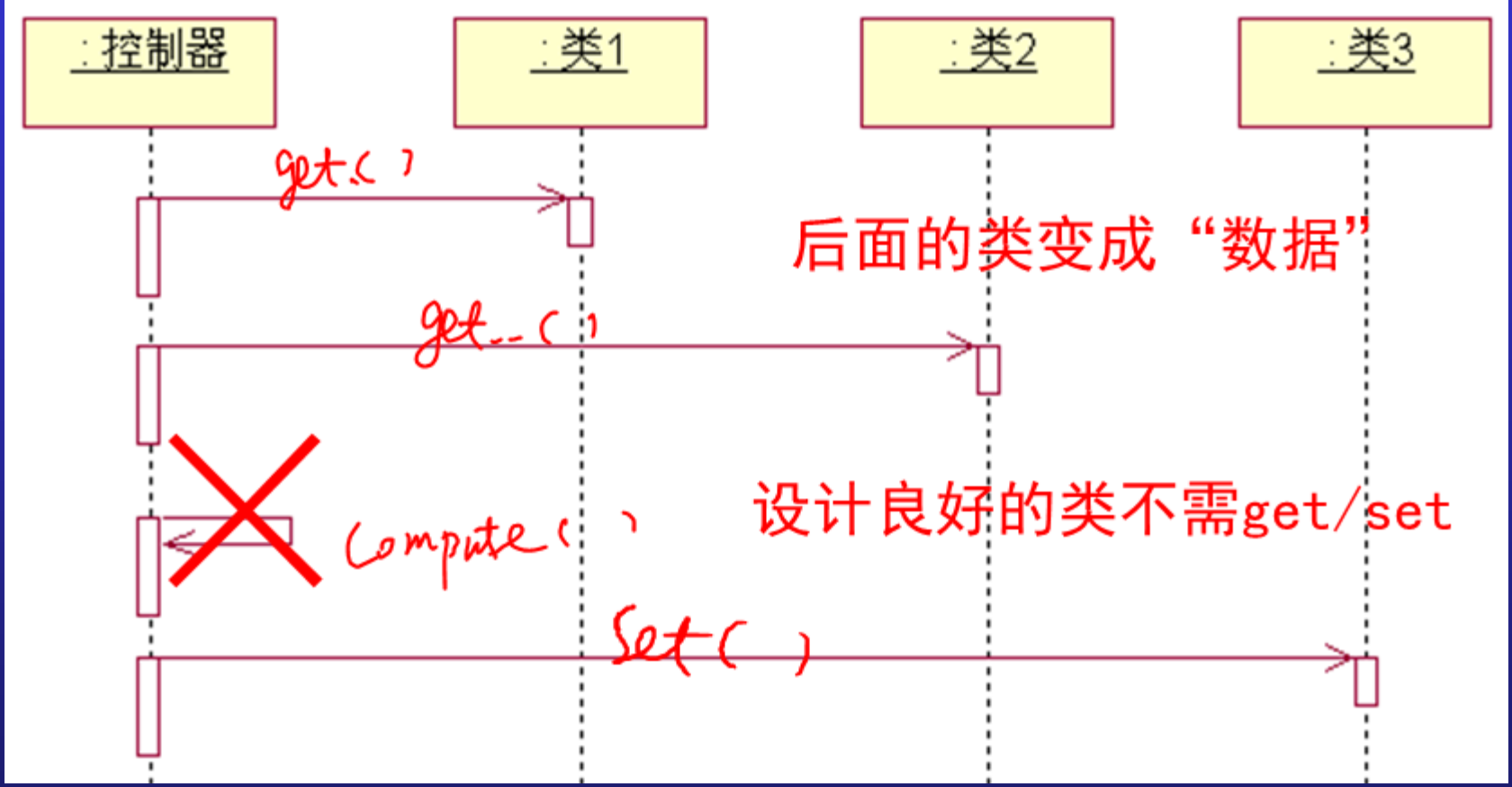




对不起，有事找我帮忙直说，
请不要拿我的东西去做事情

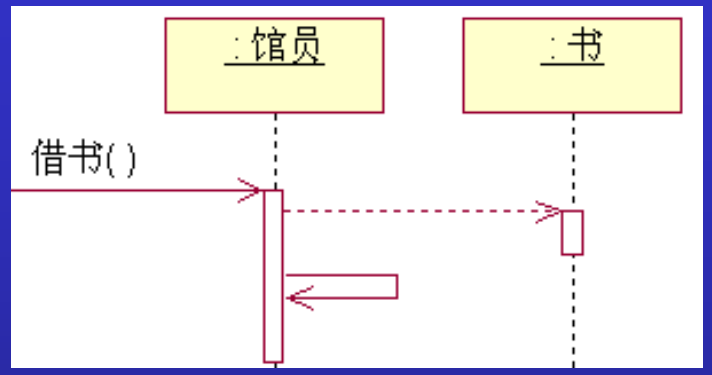
警惕Get/Set

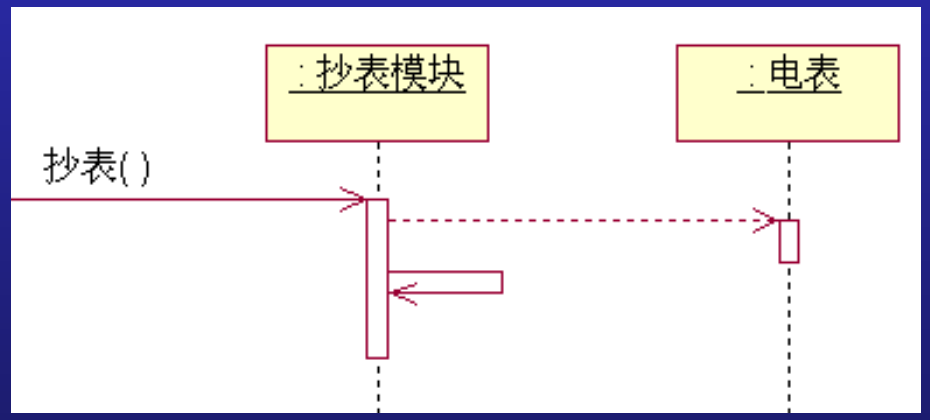
可视原则

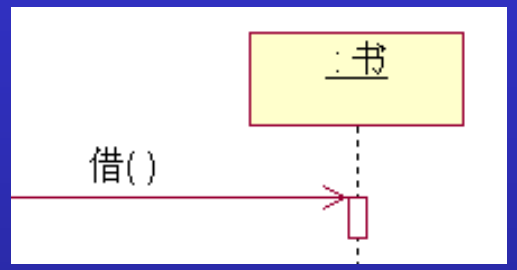


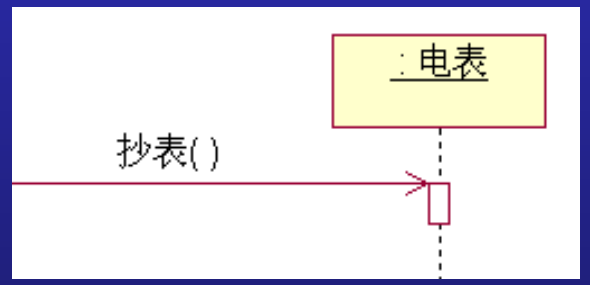
“控制器”——违反Demeter

可视原则









自治—逻辑分散在实体类中

建模工作流

*业务建模
- 愿景
- 业务用例图
- 现状业务序列图
- 改进业务序列图

*需求
- 系统用例图
- 系统用例规约

需求

提升销售

---

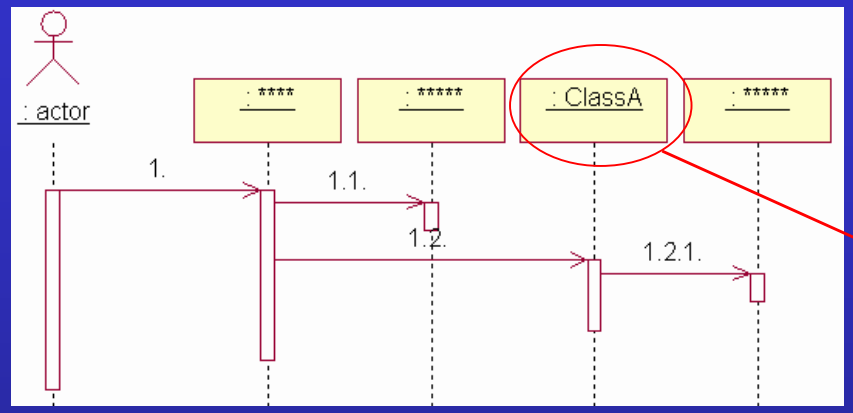
*分析
- 分析类图
- 分析序列图
- 分析状态机图

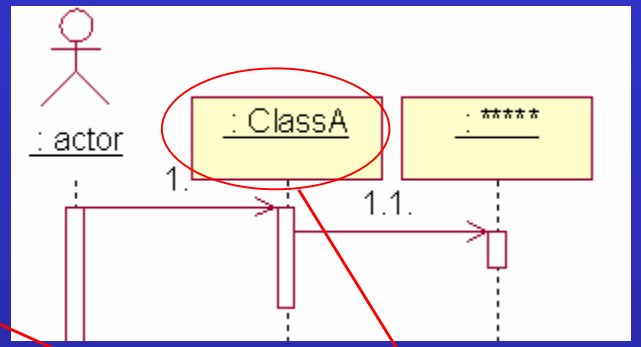
*设计
- 建立数据层
- 精化业务层
- 精化表示层

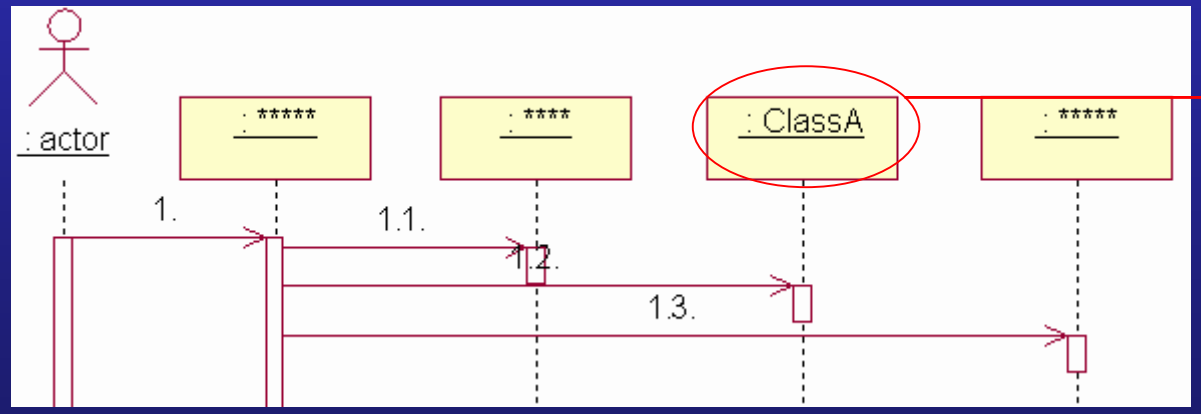
设计

降低成本

状态机图

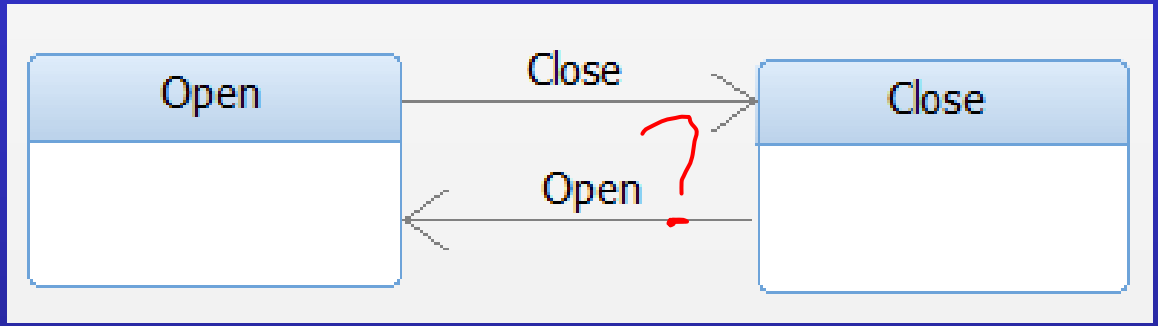


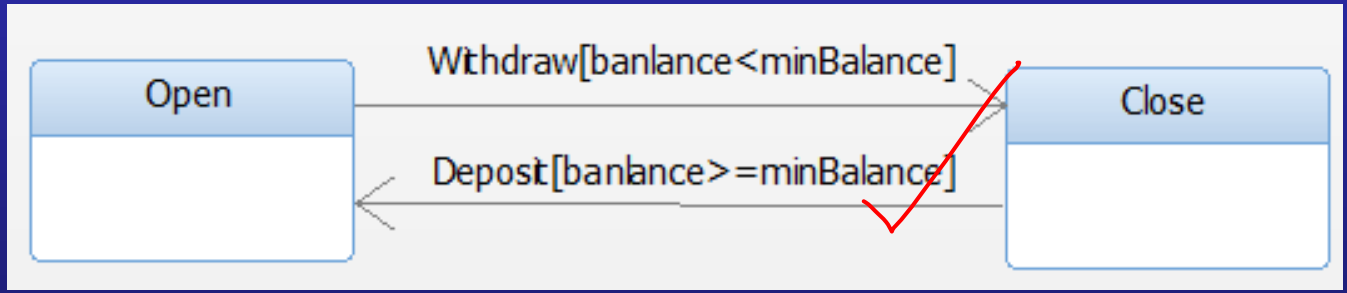




把对象从所有的序列图中单独拿出来考察

状态机图

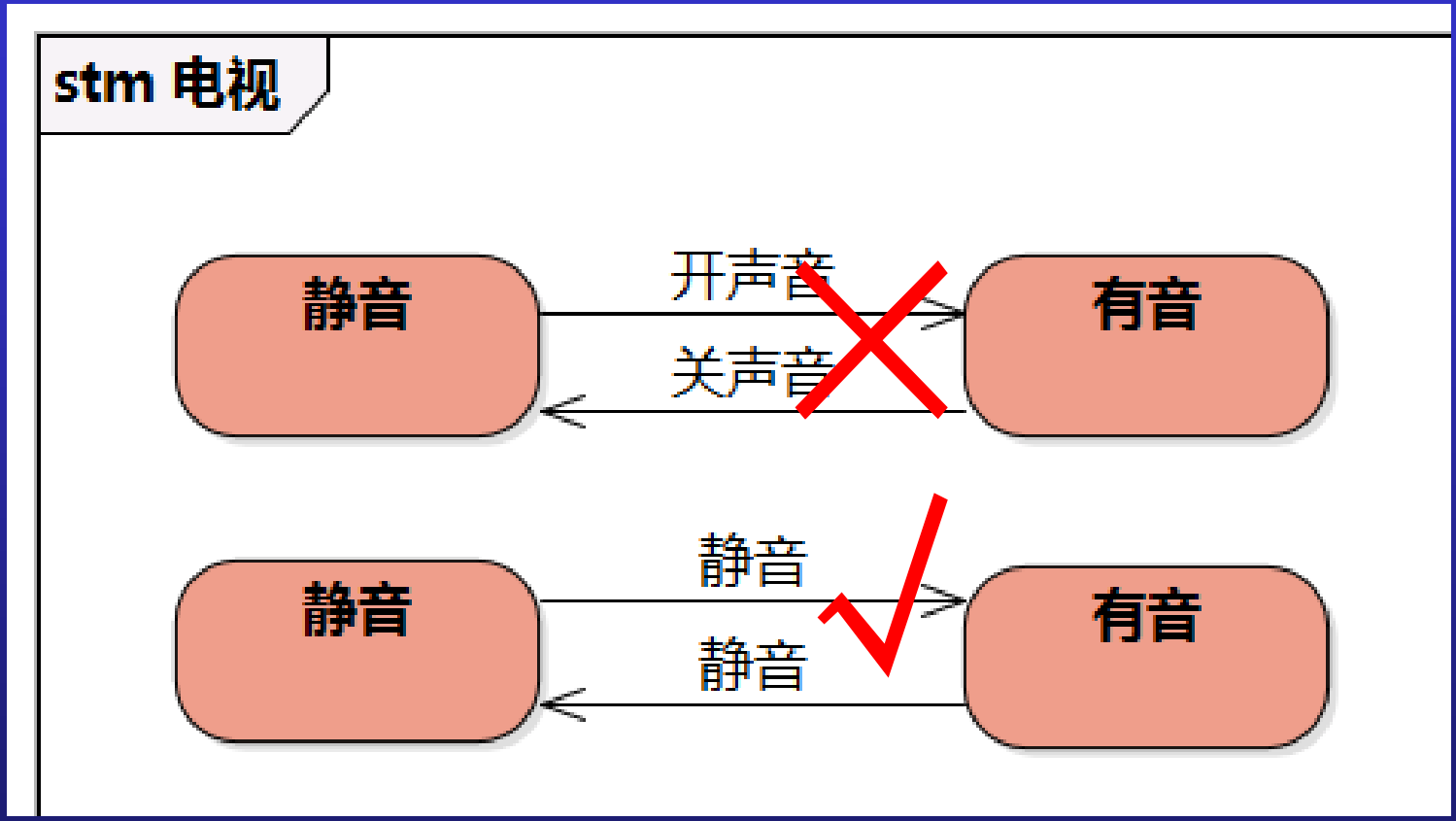




银行账户
BankAccount

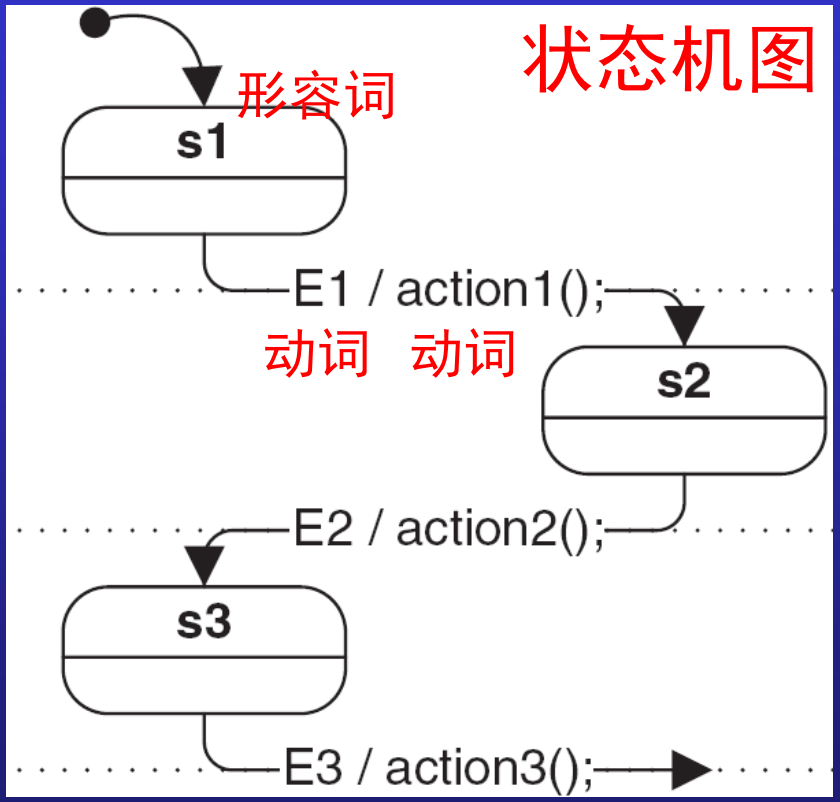
帮助定义恰当的责任

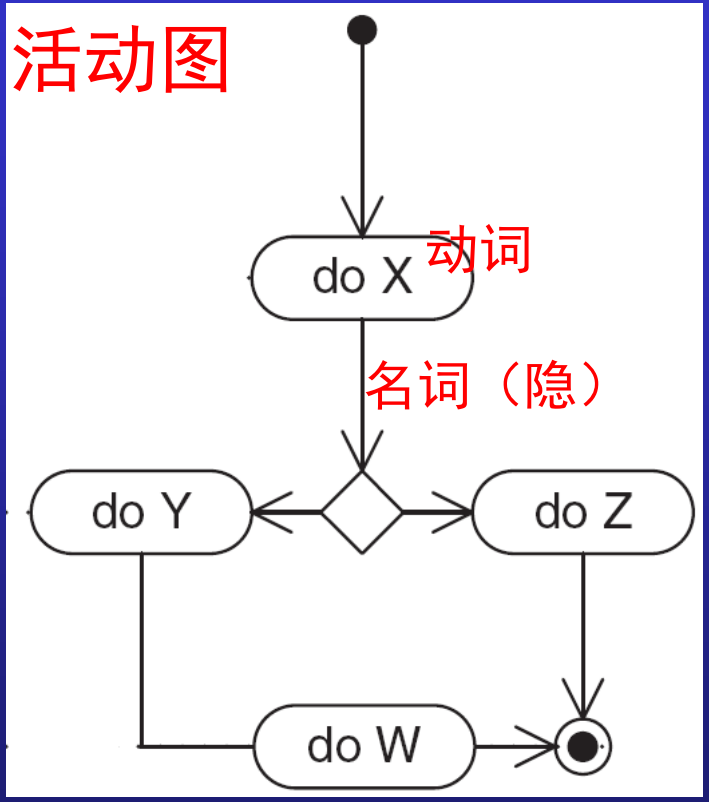
状态机图



帮助缩窄接口

状态机图





顶点和边的含义相反
不同的思考范型

状态机图 vs. 活动图

状态机图

她没有说：不要去…

对象能运转自如吗——开发人员的一天

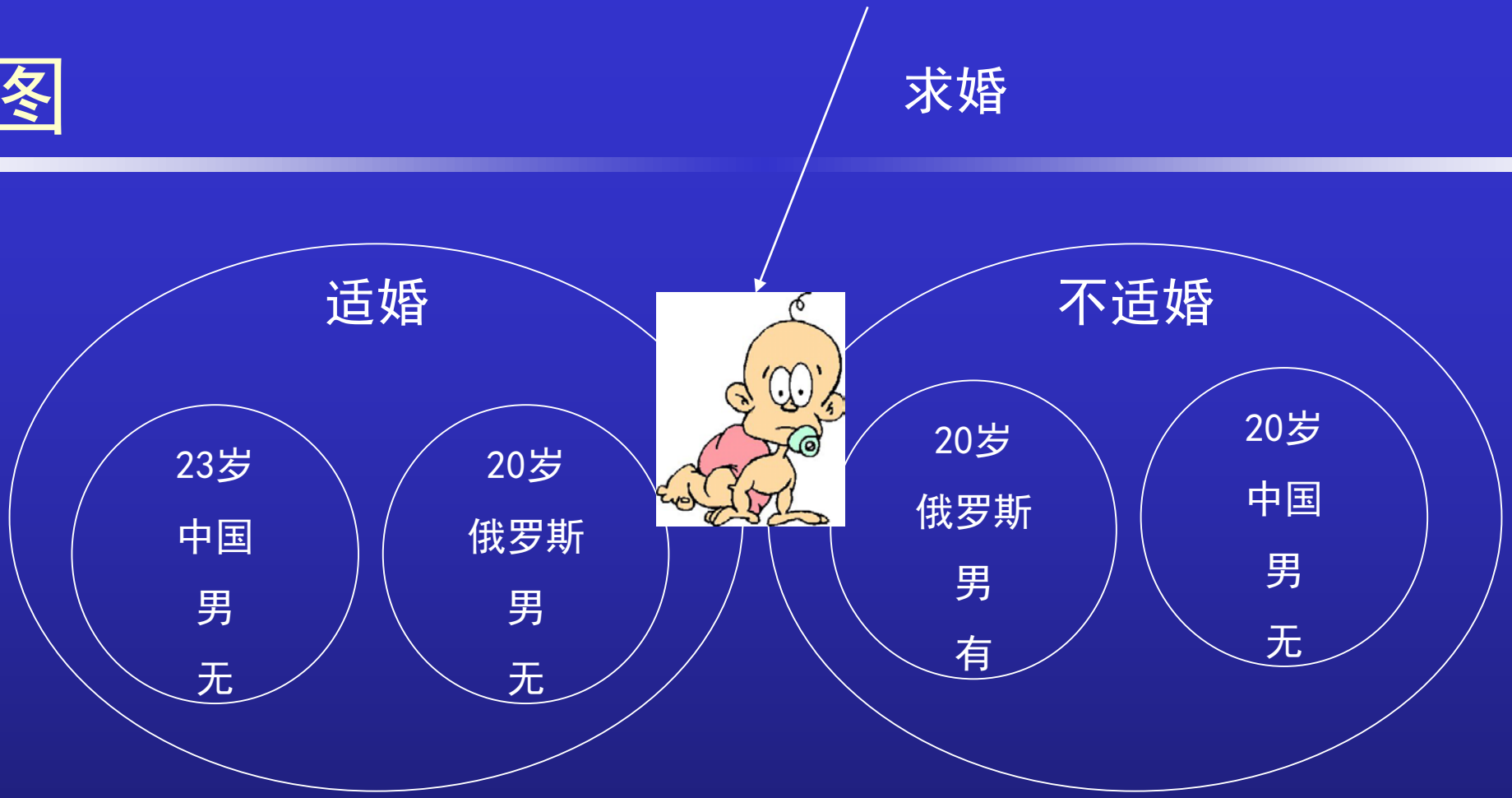
状态机图

- 年龄：…6, 7, 8, 9…17, 18…21, 22…34, 35…
- 国籍：…中国，美国，俄罗斯，阿联酋…
- 性别：…男，女…
- 配偶：…有，无，多个…

在系统中表现出相同行为的属性值和链接组合

状态！＝状态位

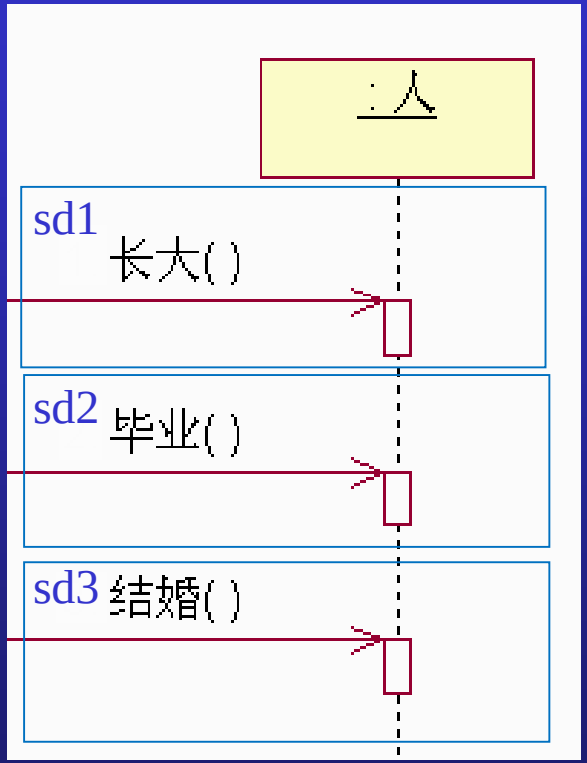
状态机图



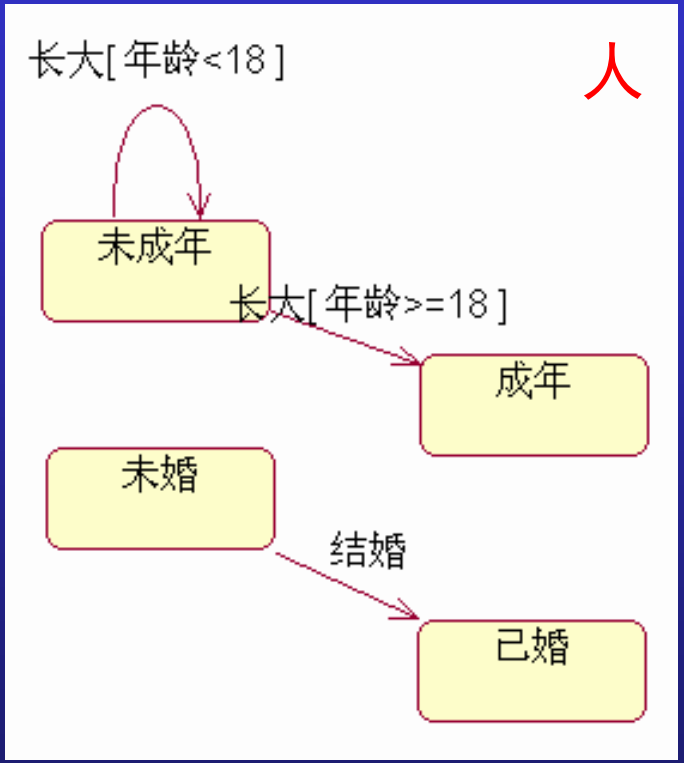
状态机图

类图

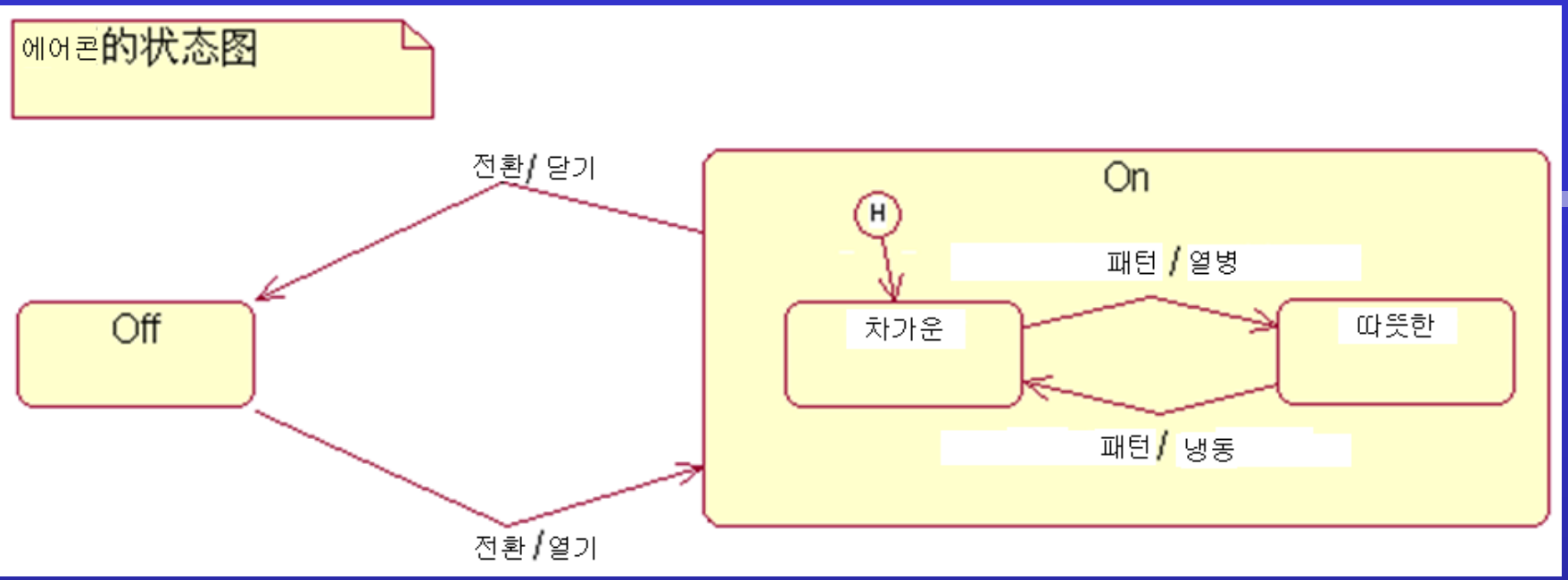
序列图

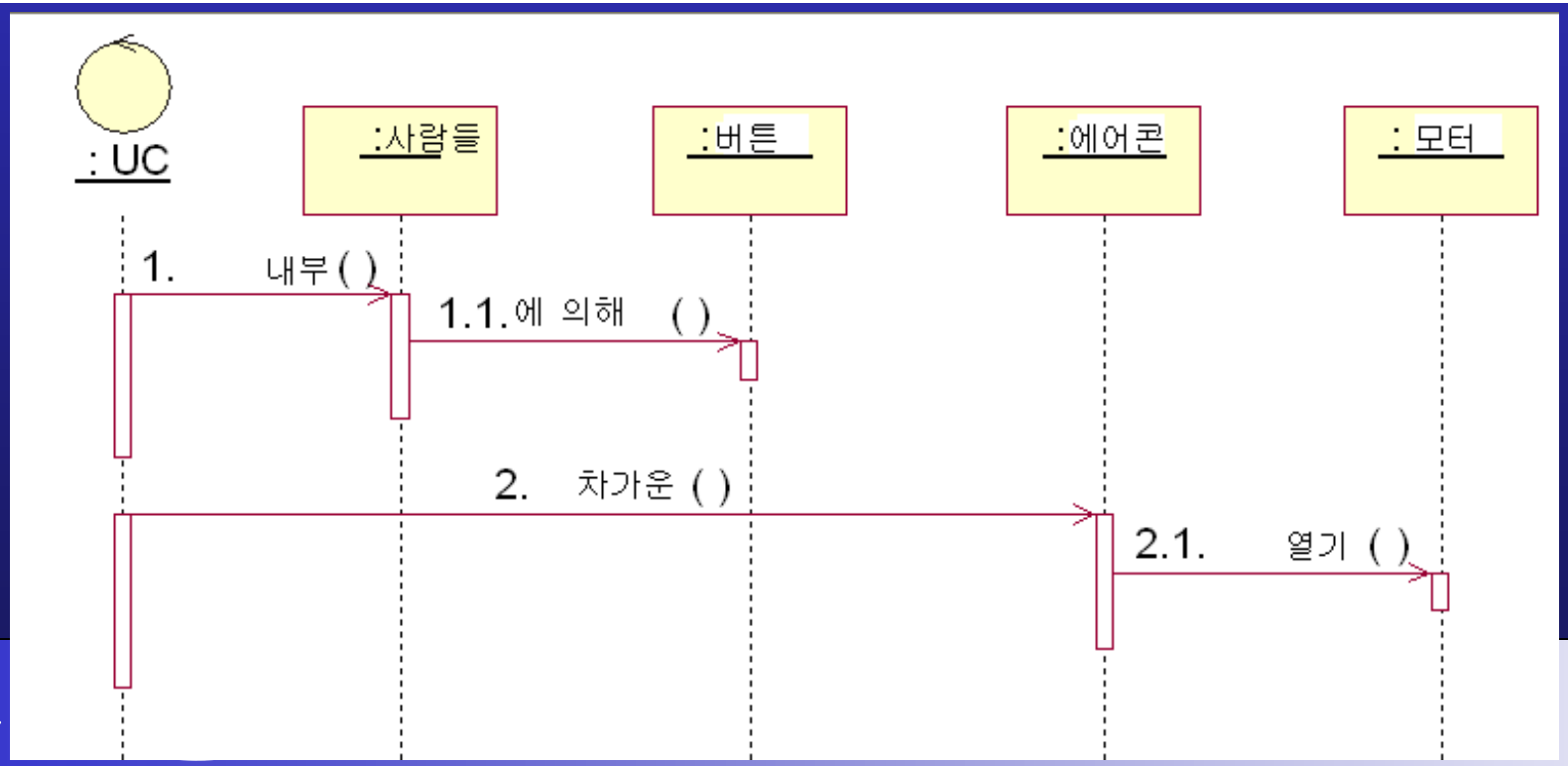


状态机图



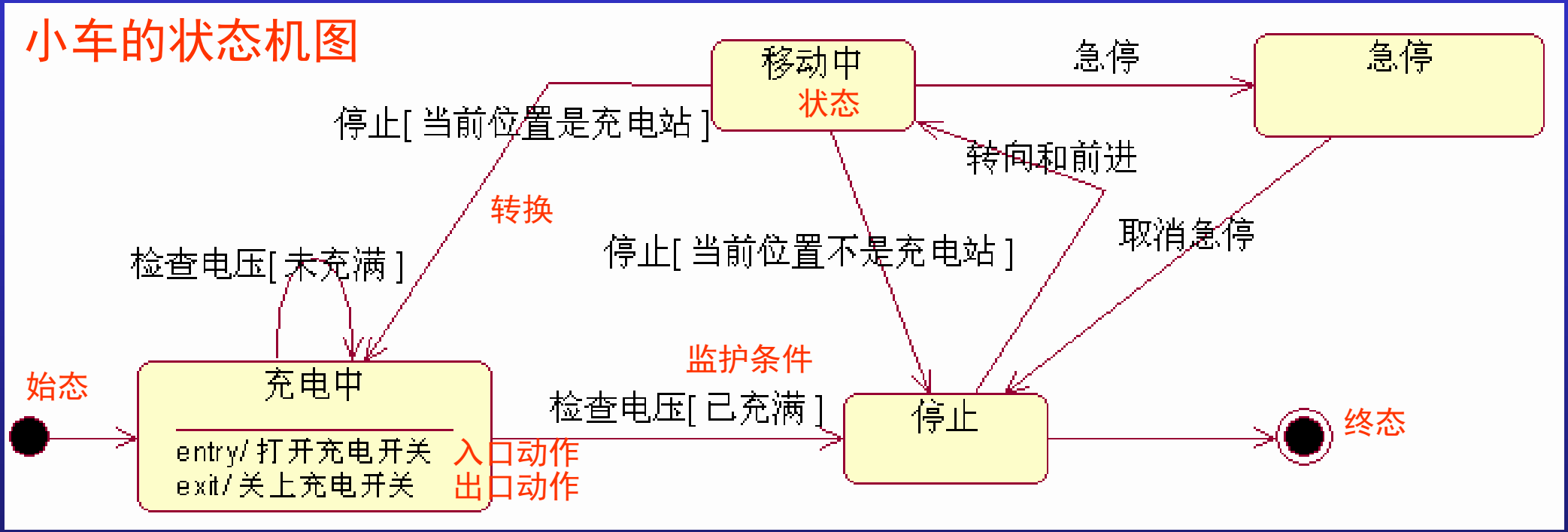
属性值变化导致行为发生变化—转换





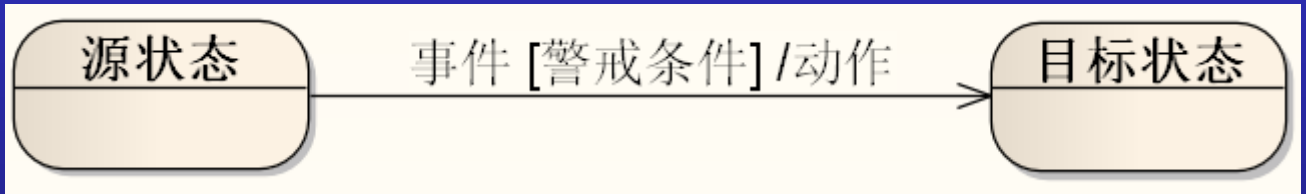
假设状态机图是对的
那么序列图错在哪

状态机图



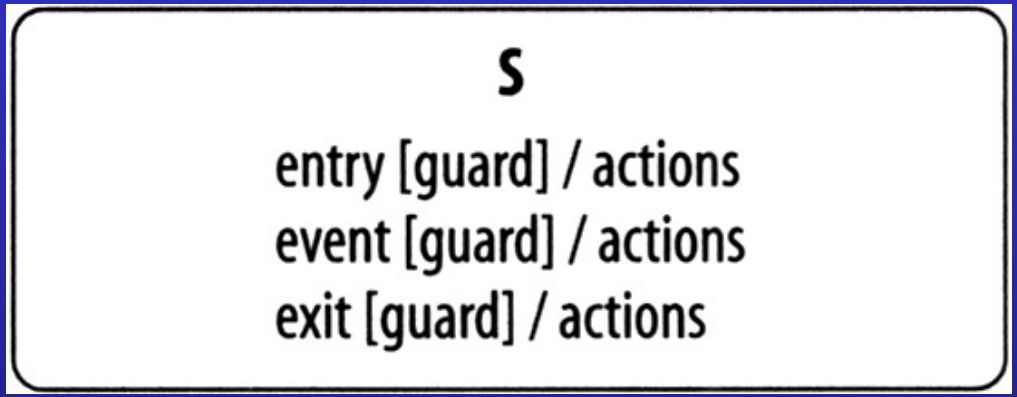
小车有哪几个操作？

状态机图



转换

状态机图



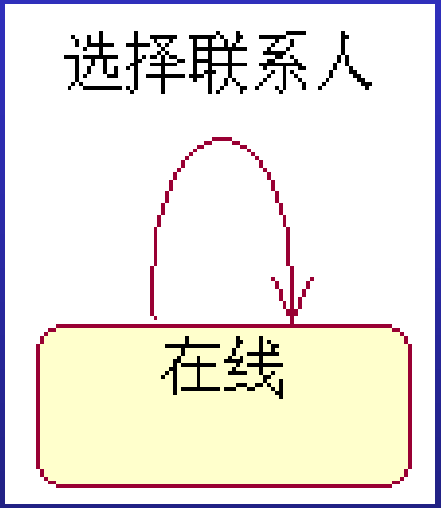
entry：进入时必须执行

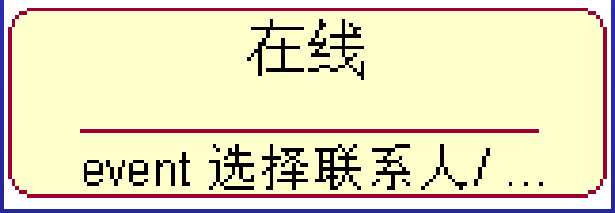
exit：离开时必须执行

event：发生event时内部执行

内部动作

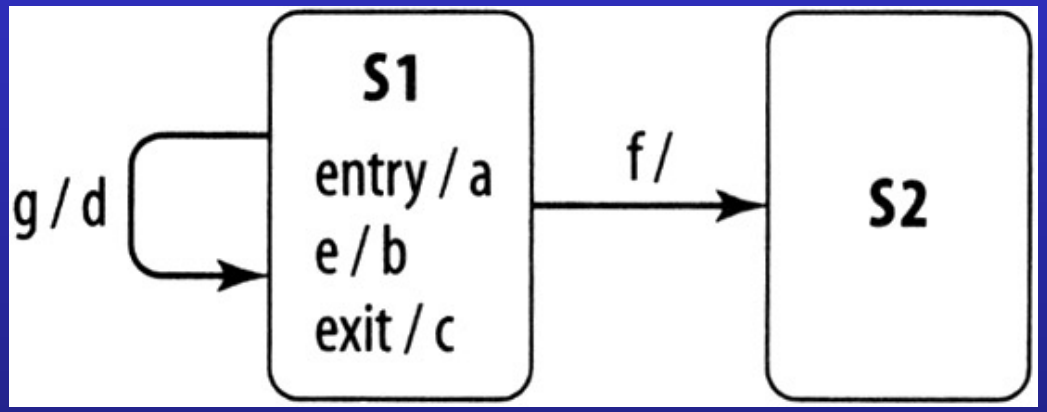
状态机图





此二者有何区别？

状态机图



当前S1：

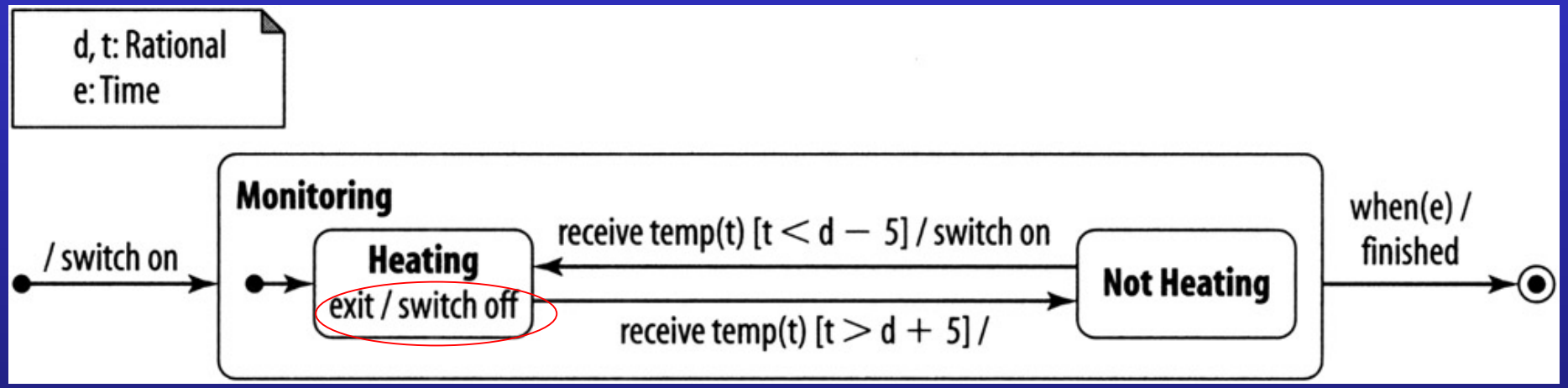
e发生：执行b，到S1

g发生：执行c, d, a，到S1

f发生：执行c，到S2

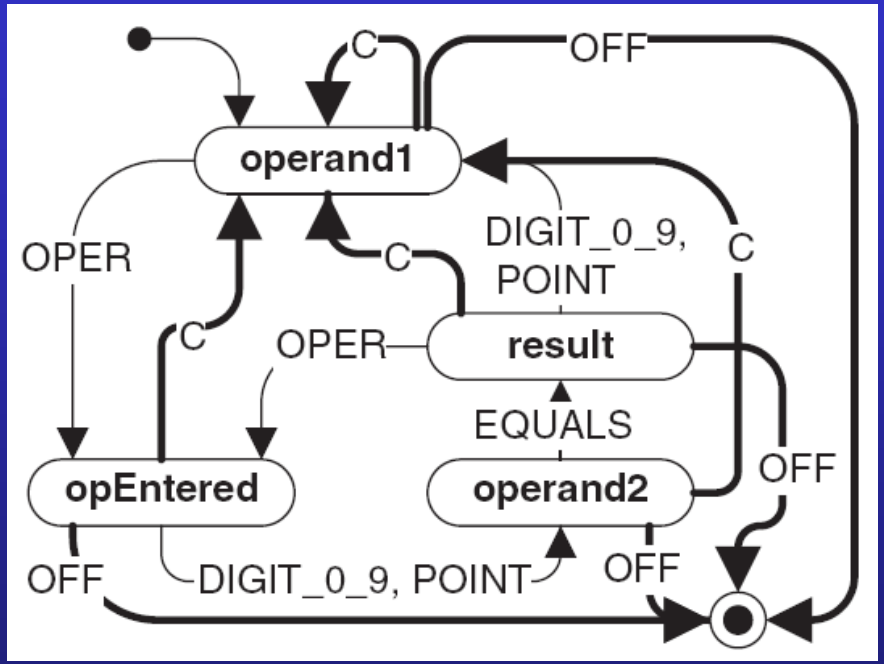
执行顺序

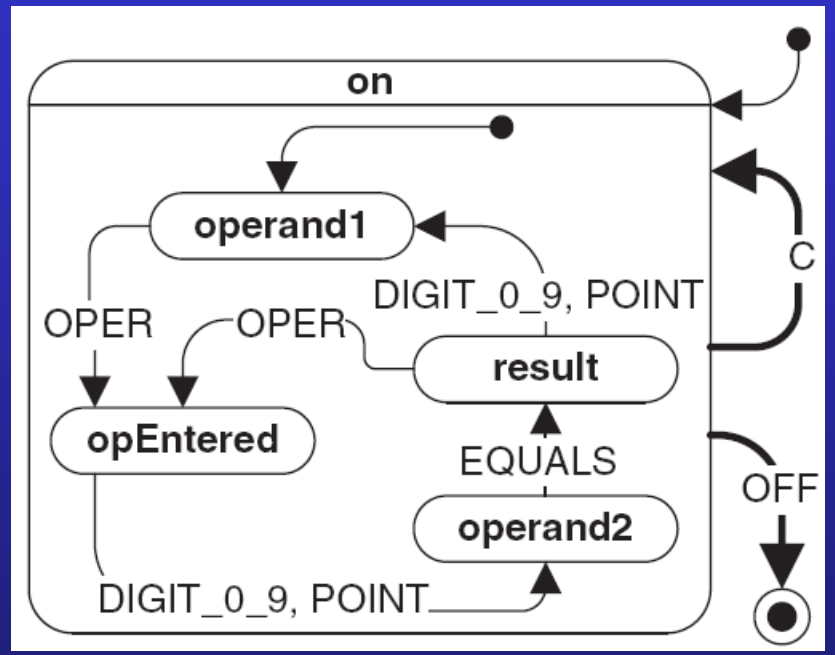
状态机图


d, t: Rational
e: Time
Monitoring
/ switch on
Heating
exit / switch off
receive temp(t) [t < d − 5] / switch on
receive temp(t) [t > d + 5] /
Not Heating
when(e) /
finished

嵌套状态

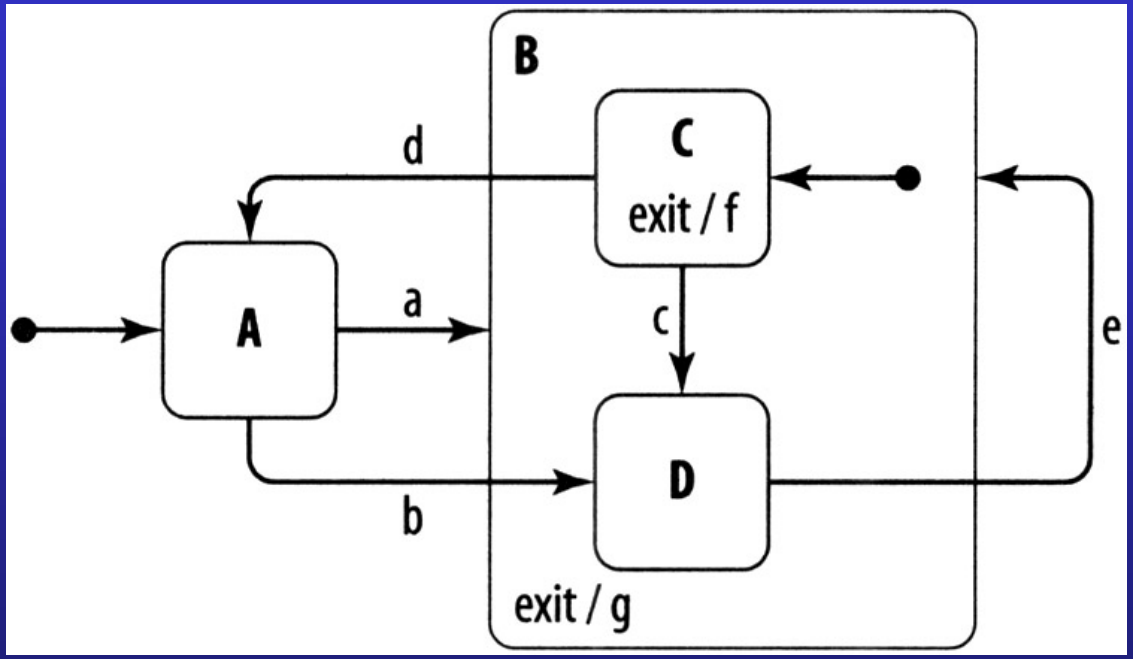
状态机图


C
OFF
operand1
OPER
C
DIGIT_0_9,
POINT
C
C
OPER
result
EQUALS
opEntered
operand2
OFF
OFF
DIGIT_0_9, POINT
OFF


on
operand1
C
DIGIT_0_9, POINT
OPER
OPER
result
opEntered
EQUALS
OFF
operand2
DIGIT_0_9, POINT

分层复用行为

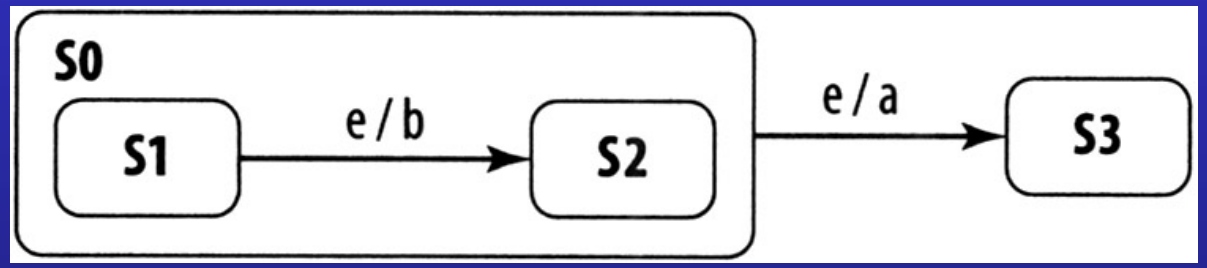
状态机图



源状态	发生	动作	到达状态
A	a		B, C
B, C	c	f	B, D
B, D	e	g	B, C
B, C	d		

如果状态C下d发生，会怎样？

状态机图



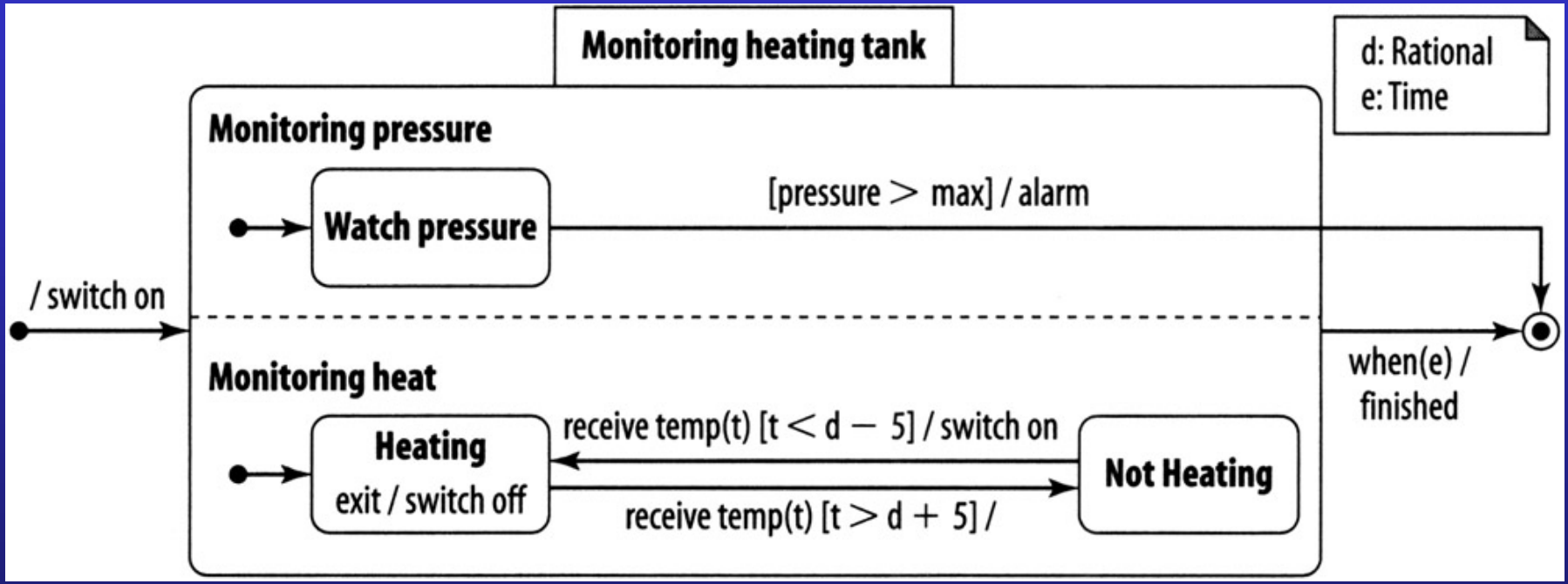
当前S1：

e发生会怎样？

子状态覆盖父状态，到S2

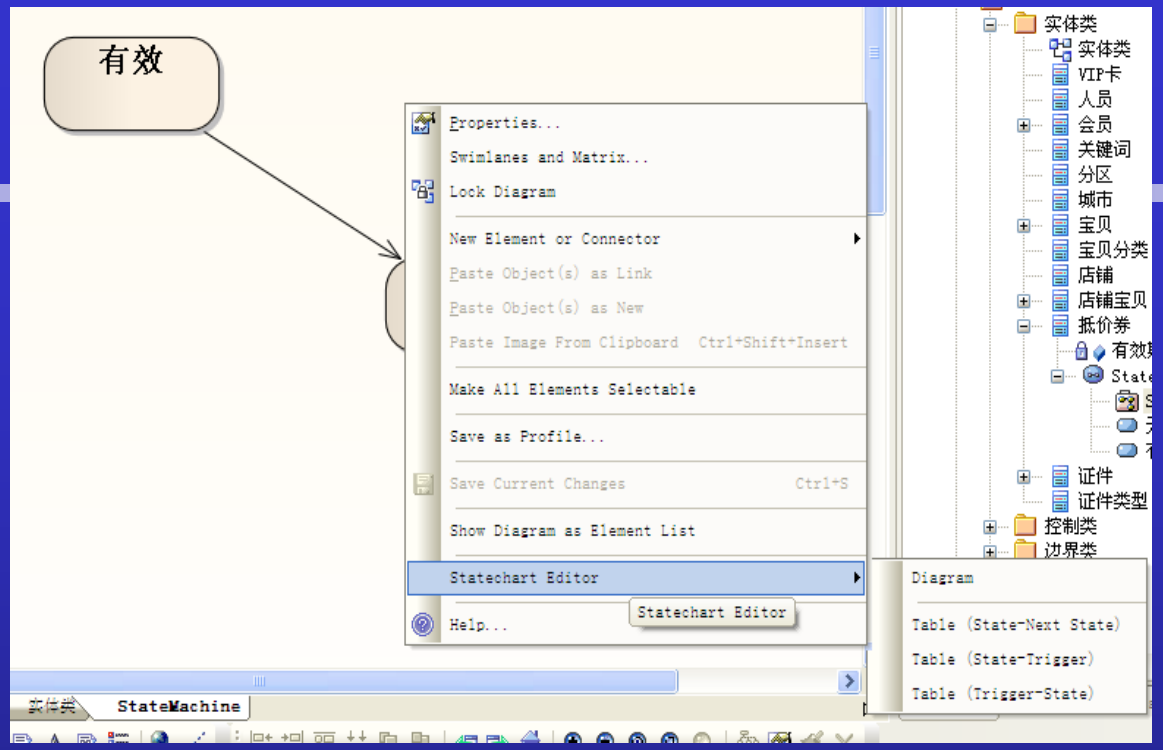
嵌套状态

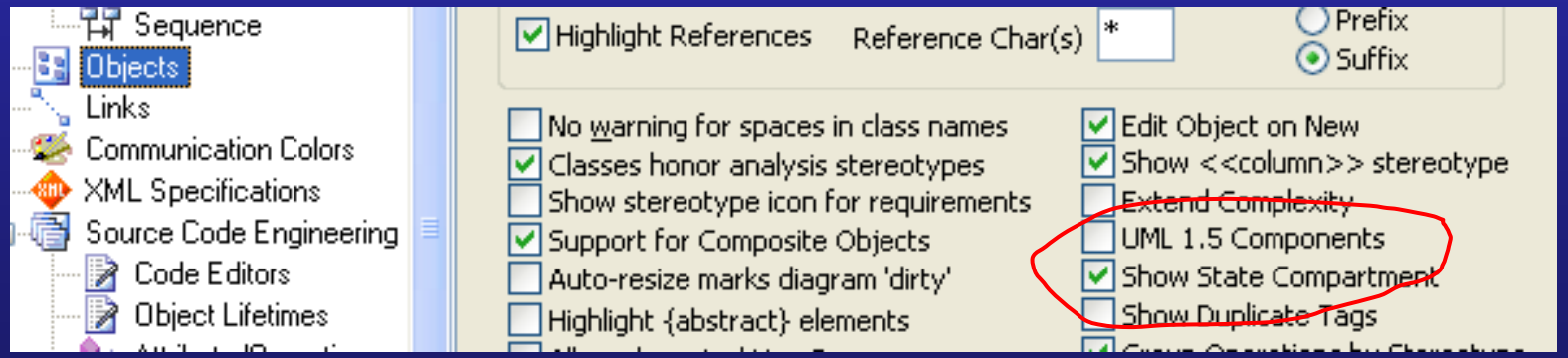
状态机图



并行状态—AND

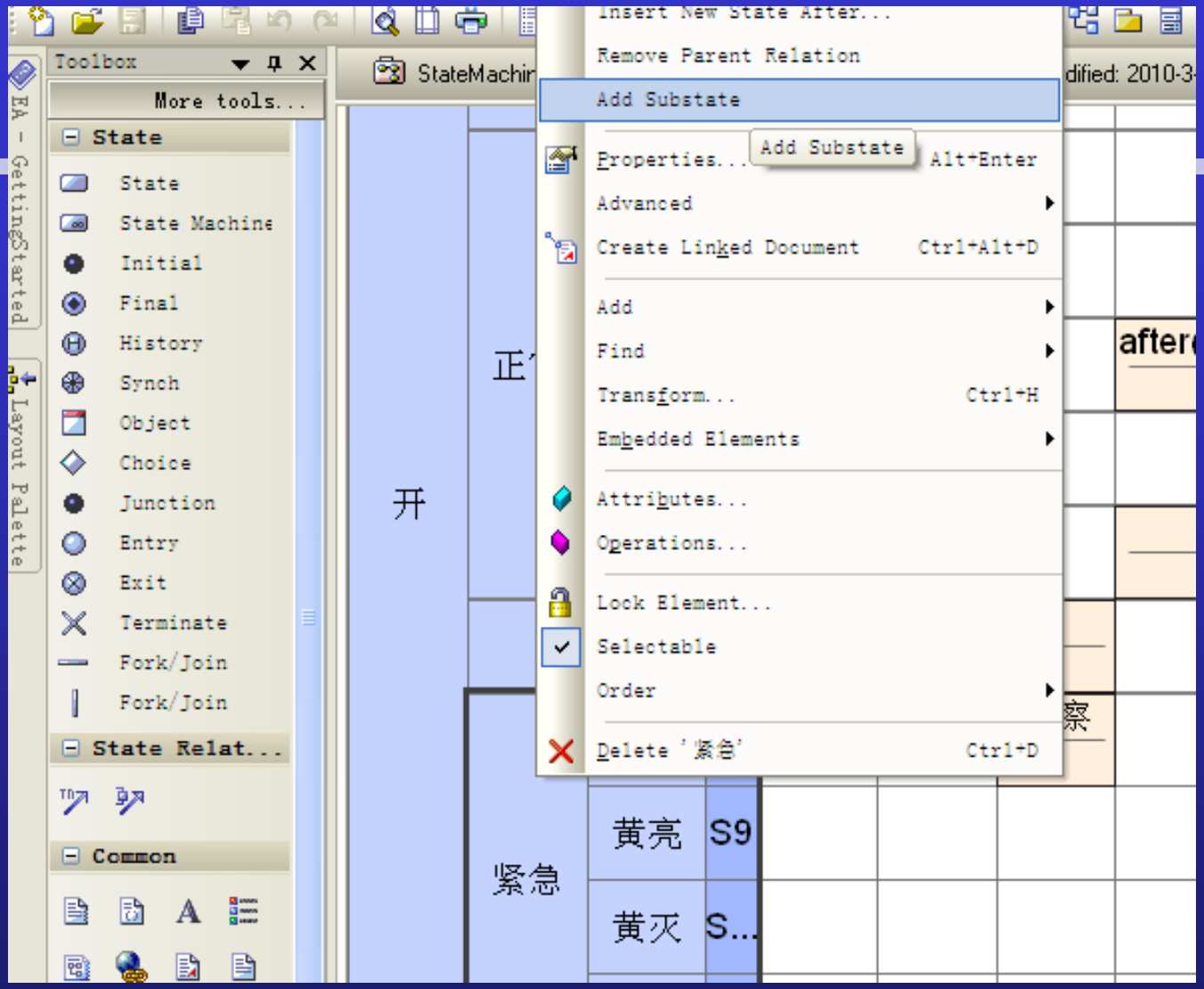
状态机图


有效
Properties...
Swimlanes and Matrix...
Lock Diagram
New Element or Connector
Paste Object(s) as Link
Paste Object(s) as New
Paste Image From Clipboard Ctrl+Shift+Insert
Make All Elements Selectable
Save as Profile...
Save Current Changes Ctrl+S
Show Diagram as Element List
Statechart Editor
Help...
Statechart Editor
Diagram
Table (State-Next State)
Table (State-Trigger)
Table (Trigger-State)
StateMachine
实体类
VIP卡
人员
会员
关键词
分区
城市
宝贝
宝贝分类
店铺
店铺宝贝
抵价券
证件
证件类型
控制类
边界类


Sequence
Objects
Links
Communication Colors
XML Specifications
Source Code Engineering
Code Editors
Object Lifetimes
Highlight References
Reference Char(s)
*
Prefix
Suffix
No warning for spaces in class names
Classes honor analysis stereotypes
Show stereotype icon for requirements
Support for Composite Objects
Auto-resize marks diagram 'dirty'
Highlight {abstract} elements
Edit Object on New
Show <<column>> stereotype
Extend Complexity
UML 1.5 Components
Show State Compartment
Show Duplicate Tags

状态机格式

状态机图


Toolbox
More tools...
State
State
State Machine
Initial
Final
History
Synch
Object
Choice
Junction
Entry
Exit
Terminate
Fork/Join
Fork/Join
State Relat...
Common
EA - Getting Started
Layout Palette
StateMachin
Insert New State After...
Remove Parent Relation
Add Substate
Properties...
Add Substate
Alt+Enter
Advanced
Create Linked Document
Ctrl+Alt+D
Add
Find
Transform...
Ctrl+H
Embedded Elements
Attributes...
Operations...
Lock Element...
Selectable
Order
Delete '紧急'
Ctrl+D
开
紧急
黄亮
S9
黄灭
S...
after

子状态

状态机图

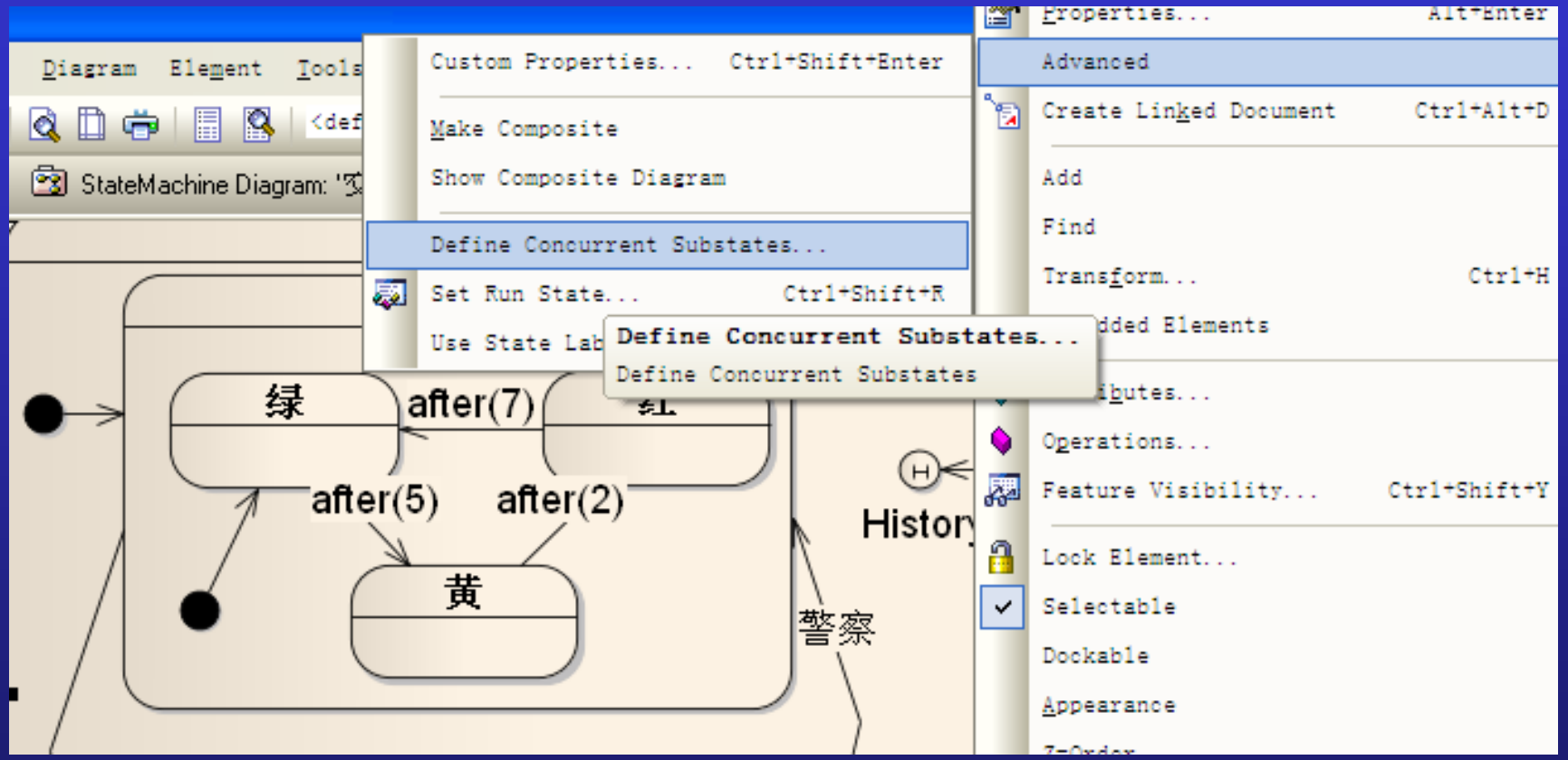
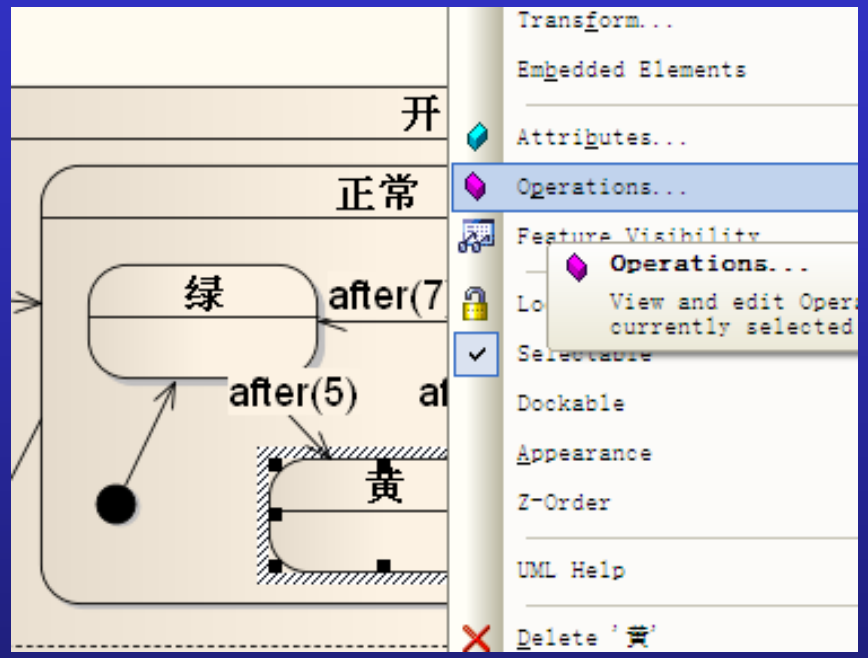
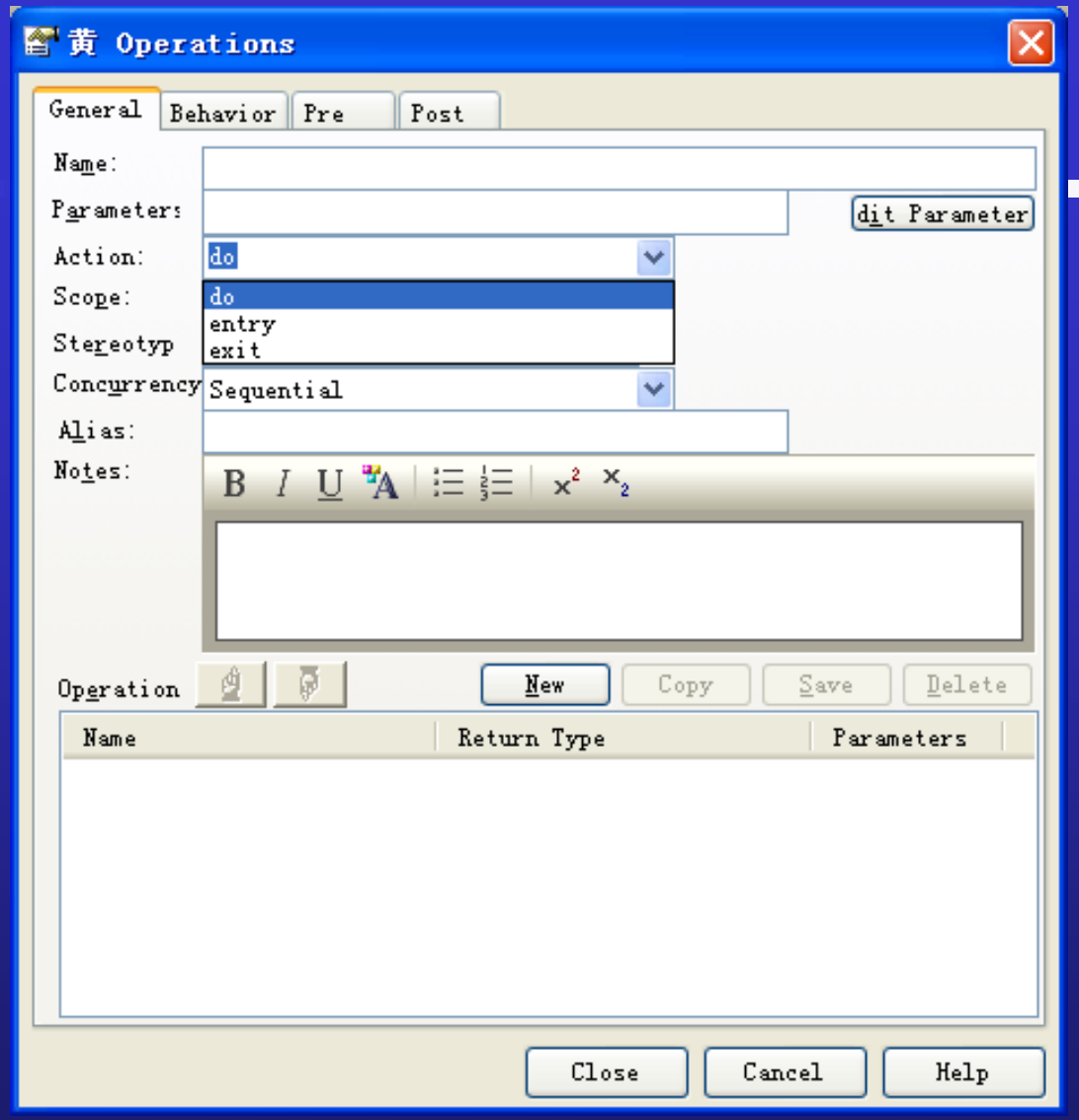


Diagram
Element
Tools
StateMachine Diagram:
Custom Properties... Ctrl+Shift+Enter
Make Composite
Show Composite Diagram
Define Concurrent Substates...
Set Run State... Ctrl+Shift+R
Define Concurrent Substates...
Define Concurrent Substates
绿
after(7)
after(5)
after(2)
黄
警察
History
Properties... Alt+Enter
Advanced
Create Linked Document Ctrl+Alt+D
Add
Find
Transform... Ctrl+H
Operations...
Feature Visibility... Ctrl+Shift+Y
Lock Element...
Selectable
Dockable
Appearance

正交（分区）

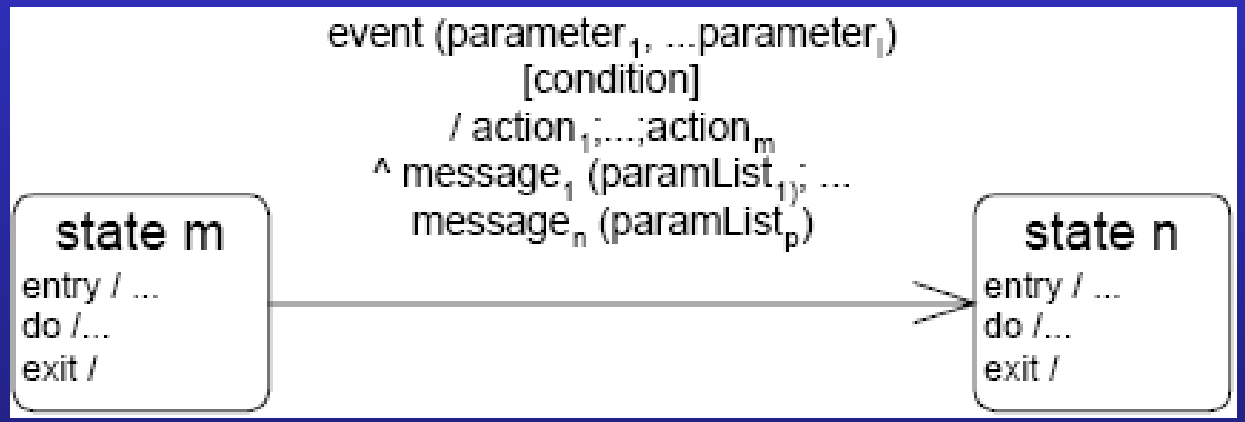
状态机图


开
正常
绿
after(7
after(5)
黄
Transform...
Embedded Elements
Attributes...
Operations...
Feature Visibility
Operations...
View and edit Oper
currently selected
Dockable
Appearance
Z-Order
UML Help
Delete '黄'


黄 Operations
General
Behavior
Pre
Post
Name:
Parameters:
dit Parameter
Action:
do
Scope:
do
entry
exit
Stereotyp
Concurrency
Sequential
Alias:
Notes:
Operation
New
Copy
Save
Delete
Name
Return Type
Parameters
Close
Cancel
Help

添加动作

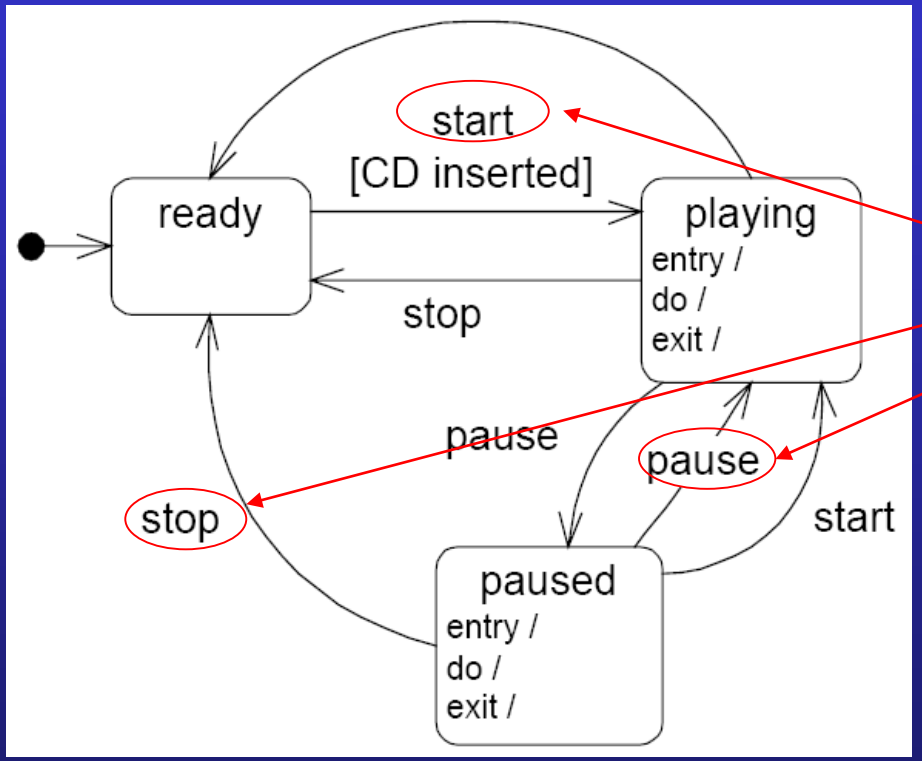
状态机图

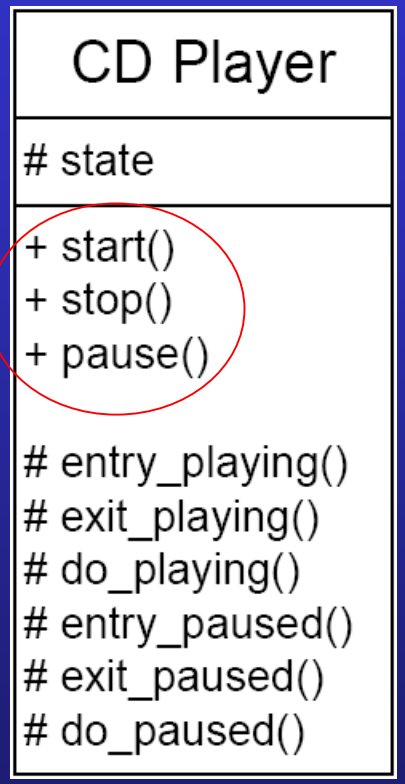


1. 事件
2. 检查当前状态是否能接受事件
3. 如果可以，检查转换条件
4. 如果条件为真
   - 状态m的出口活动
   - action1 – actionm
   - 发送message1 – messagen
   - 改变状态
   - 状态n的入口活动
   - 状态n的do活动

代码映射

状态机图


start
[CD inserted]
ready
playing
entry /
do /
exit /
stop
pause
pause
stop
start
paused
entry /
do /
exit /


CD Player
state
+ start()
+ stop()
+ pause()
# entry_playing()
# exit_playing()
# do_playing()
# entry_paused()
# exit_paused()
# do_paused()

以CD机为例

状态机图

```
class CD_Player {
protected:
enum {INITIAL, READY, PLAYING, PAUSED, FINAL}   stateValue;
stateValue  m_state;
virtual void setState (stateValue newValue)
{
    if ((newValue >= INITIAL) && (newValue <= FINAL))
    {
    m_state = newValue
    }
    else
    {
    throw badStateChange;    // exception
    }
};
```

状态和设置状态

状态机图

```
virtual void stop (void)
{
        switch (m_state)
        {
                case PLAYING:
                        exit_playing();
                        setState (READY);
                        break;

                case PAUSED:
                        exit_paused();
                        setState (READY);
                        break;

                default:    break;  // event ignored

        }  // switch
};
```

Stop

状态机图

```
virtual void start (void)
{
      switch (m_state)
      {
            case READY:
                  if (CD inserted)
                  {
                        exit_ready();
                        setState (PLAYING);
                        entry_playing();
                        do_playing();
                  }
            break;

            case PAUSED:
                  exit_paused();
                  setState (PLAYING);
                  entry_playing();
                  do_playing();

            default:    break;  // event ignored

      }  // switch
};
```

Start

状态机图

```
virtual void pause (void)
{
        switch (m_state)
        {
                case PLAYING:
                        exit_playing();
                        setState (PAUSED);
                        entry_paused();
                        do_paused();
                break;

                case PAUSED:
                        exit_paused();
                        setState (PLAYING);
                        entry_playing();
                        do_playing();
                break;

                default:    break;  // event ignored

        }  // switch
};
```

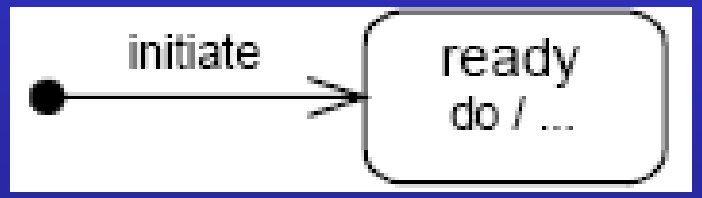
Pause

状态机图

```
CDPlayer::CDPlayer (void)     // constructor
{
      setState (INITIAL);
      // user code
};
```

```
void CDPlayer::initiate (void)
{
      switch (m_state)
      {
            case INITIAL:
                  setState (READY);
                  do_ready();
                  break;

            default:    break; // event ignored
      }
};
```

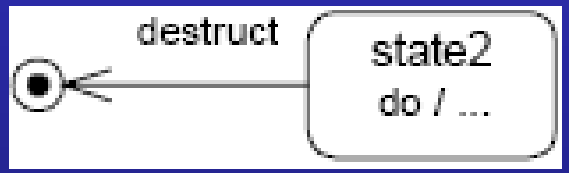


初始

状态机图

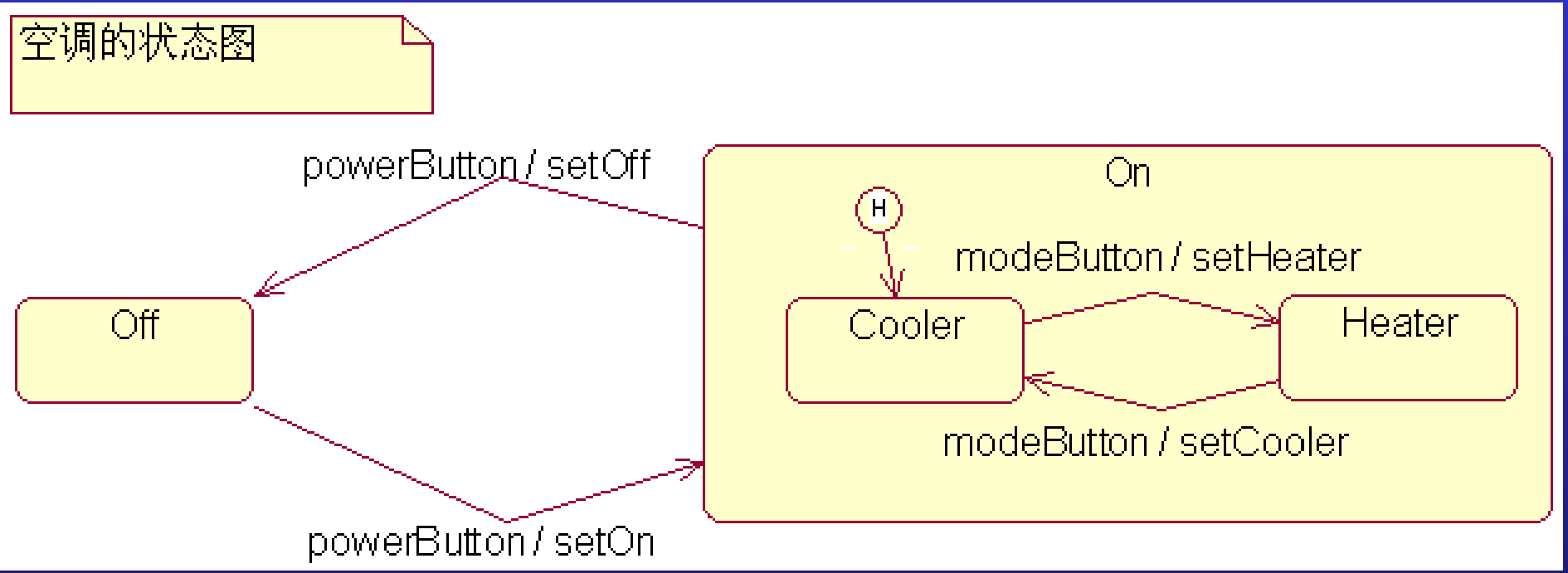
```
void CDPlayer::destruct (void)
{
        switch (m_state)
        {
                case STATE2:
                        setState (FINAL);
                        delete this;
                        break;

                default:    break; // event ignored
        }
};
```



终止

状态机图



请写出powerButton()的代码

状态机图

```
class AirCon {
    public static final int off = 1;
    public static final int on = 2;
    public static final int cooler
    = 3;
    public static final int heater
    = 4;
    public int state; // state
    variable
    public int on_subState;

    ……
```

```
public void powerButton() {
    switch (__①__ ) {
        case __②__ :
            setOn;
            __③__ =
on_subState;
            break;
        case __④__ :
            setOff;
            state = __⑤__ ;
            break;
    ……
```

powerButton()的代码

状态机图

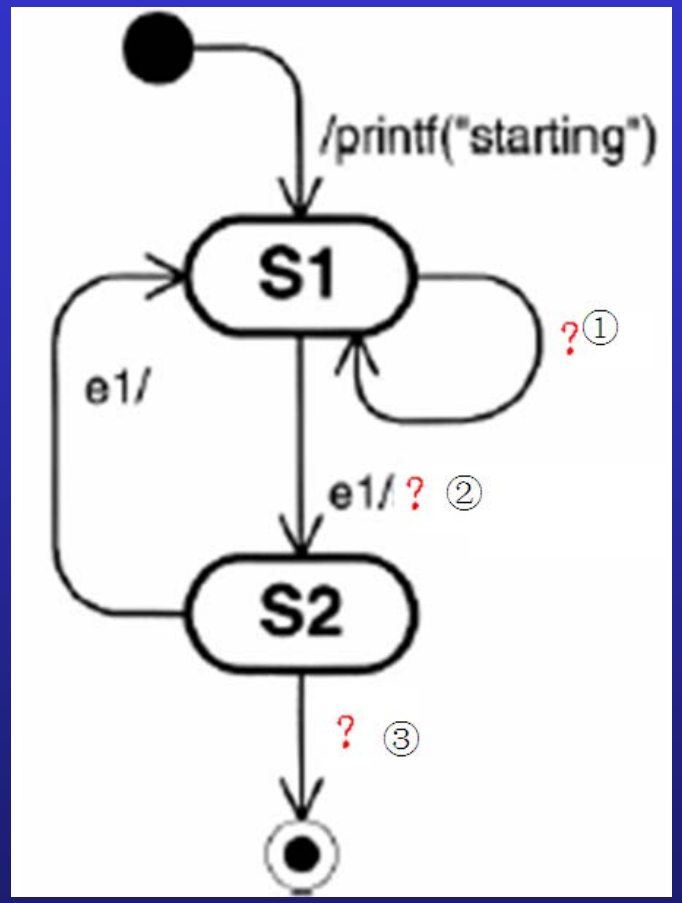
```
AirCon() {  //constructor
  state = off;
  on_subState = cooler;
 }
public void modeBut() {  // event method
  switch (state) {
        case off :
              break;
        case cooler :
              setHeater;    // action
              // exit actions
              on_subState = Heater;
              state = on_subState;
              // entry actions
              break;
        case heater :
              setCooler;   // action
              // exit actions
              on_subState = Cooler;
              state = on_subState;
              // entry actions
              break;
        default :
              break;
 }
```

初始状态

modeButton()
的代码？

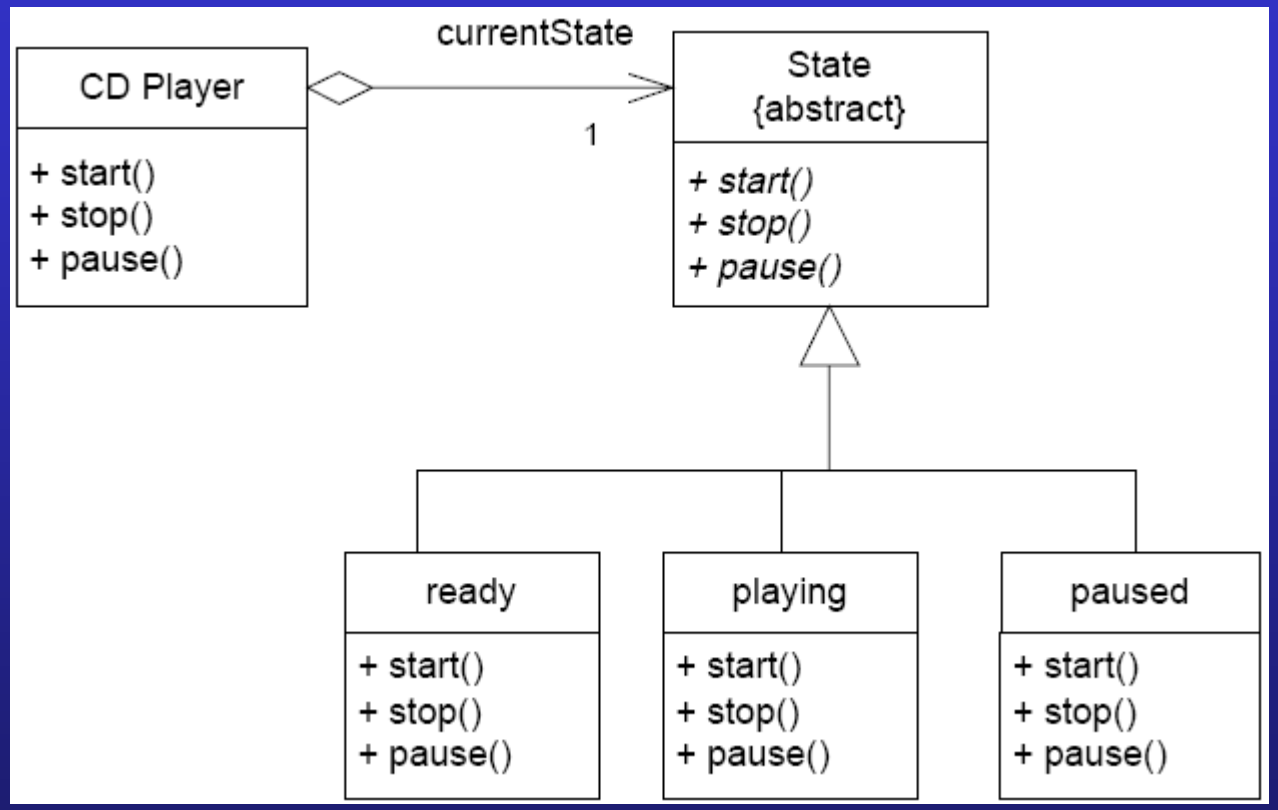
状态机图

```
int xs = 0;
do {
    receiveEvent(ev);
    switch (xs) {
        case 0:  printf ("starting");
                 break;
        case 1:  switch (ev) {
                     case e1: send(oa,5);
                              xs = 2;
                              break;
                     case e2: xs = 1;
                              break;
                 }
        case 2:  switch (ev) {
                     case e1: xs = 1;
                              break;
                     case e2: xs = 999;
                              break;
                 }
} while (xs < 999);
```



代码倒推回状态机图？

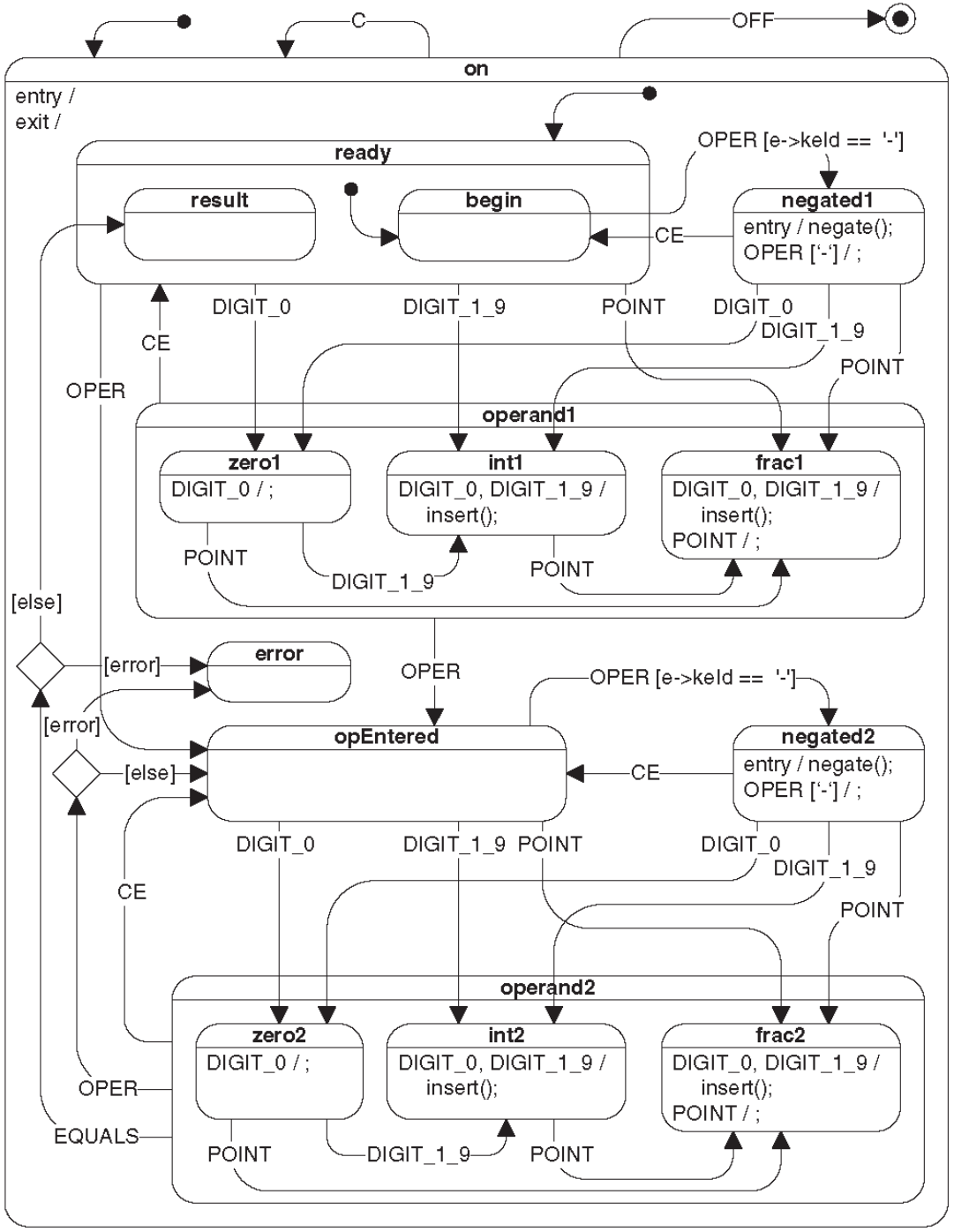
状态机图



GoF的State模式

进一步重构

状态机


C
OFF
on
entry /
exit /
ready
result
begin
OPER [e->keld == '-']
negated1
entry / negate();
OPER ['-'] / ;
CE
DIGIT_0
DIGIT_1_9
POINT
DIGIT_0
DIGIT_1_9
POINT
CE
OPER
operand1
zero1
DIGIT_0 / ;
int1
DIGIT_0, DIGIT_1_9 /
insert();
frac1
DIGIT_0, DIGIT_1_9 /
insert();
POINT / ;
POINT
DIGIT_1_9
POINT
[else]
[error]
error
OPER
OPER [e->keld == '-']
[error]
opEntered
negated2
entry / negate();
OPER ['-'] / ;
[else]
CE
DIGIT_0
DIGIT_1_9
POINT
DIGIT_0
DIGIT_1_9
CE
POINT
operand2
zero2
DIGIT_0 / ;
int2
DIGIT_0, DIGIT_1_9 /
insert();
frac2
DIGIT_0, DIGIT_1_9 /
insert();
POINT / ;
OPER
EQUALS
POINT
DIGIT_1_9
POINT

示例


qp
Quantum
Platform