**数据流图**

DFD组成：

数据流，加工，外部实体，数据存储

结构化系统分析方法的主要表达工具

数据流图分层次的，平时也把数据流图称为“分层数据流图”的原因

分层数据流图很好的传承了结构化思想——“自顶 至下，逐步精化”。将图分成了若干个层次，首先绘制针对系统整体的顶层图，以说明系统与外界的交互，再将图层层细化

顶层图：将该系统看成了一个大加工，这种图能很好的体现出系统与外部实体之间的交互关系

0层图：将顶层图中的加工分解成若干个加工，并用数据流连接这些加工

数据字典：

DFD描述了系统的分解，即系统由哪几部分组成，各部分之间的联系等，

但是，对于数据的详细内容却无法在DFD中得到反映

数据字典是在DFD的基础上，对DFD中出现的所有命名元素都加以定义，使得每个图形元素的名字都有一个确切的解释

数据平衡原则：

保持一致并非指输 入输出流的数量与名称完全一样。下层输入输出流成分之和等与上层数据流

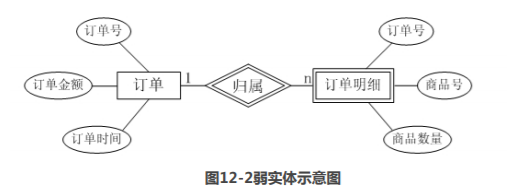
错题：

顶层数据流图描述了系统 的输入与输出

数据流图是结构化分析模型需求分析阶段得到的结果，描述了系统的功能，在进行接口设计 时，应以它为依据

**E-R**图：数据库建表

弱实体：实体需要依赖于另一个实体(自己的键有部分来自强实体)，若另一个实体不存在了，它也随 之消失。相对而言，弱实体所依赖的那个实体称为强实体



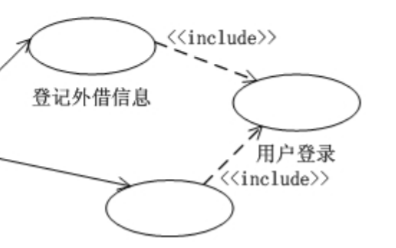
**UML建模**

UML由构造块、公共机制、规则三个部分组成

包含关系：可以从两个或两个以上的用例中提取公共行为时

“登记外借信息”和“查询外借信息”两个用例都需要登录（一定要做）

可以定义一个抽象用例“用户登录”

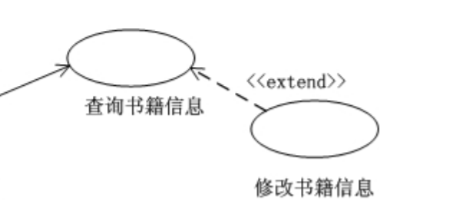


扩展关系：果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生 多种分支，则可以将这个用例分为一个基本用例和一个或多个扩展用例

扩展特定条件发生，扩展用例的行为才被执行

箭头指向基本用例

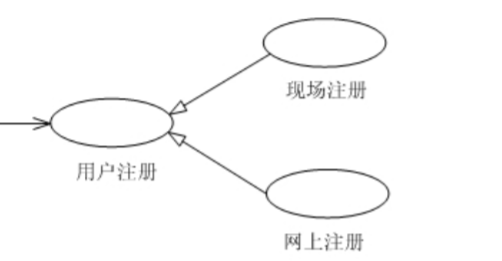
图书管理员进行“查询书箱信息”操作时，如果发现书箱信息有误，他可以 使用“修改书箱信息”用例来完成错误的修改



泛化关系：当多个用例共同拥有一种类似的结构和行为的时候，可以将它们的共性抽象成 为父用例，其他的用例作为泛化关系中的子用例。

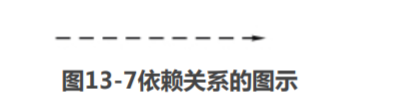
子用例是父用例的一种特 殊形式

箭头指向父用例



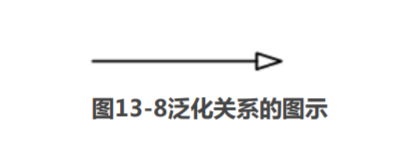
定类之间的关系

依赖关系



有两个元素X、Y，如果修改元素X的定义可能会引起对另一个元素Y的定义的修改，则称元素Y 依赖（Dependency）于元素X

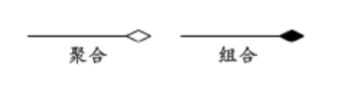
泛化关系：



类图中的泛化关系描述了一般事物与该事物中的特殊种类之间的关系

关联关系：表示两个类之间存在某种语义上的联系。例如，一个人为一家公司工作， 一家公司有许多办公室。我们就认为人和公司、公司和办公室之间存在某种语义上的联系

其中特殊：聚合和组合



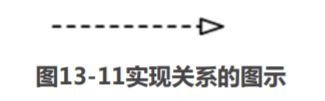
聚合关系：聚合（Aggregation）是一种特殊形式的关联。聚合表示类之间的关系是整体与部分的关系，“部分”可以独立于“整体”而存在

组合关系：如果发现“部分”类的存在，是完全依赖于“整体”类的

关联（长期）>依赖（临时，A方法用到B对象）

实现关系：

用来规定接口和实现接口的类或组件之间的关系



多重性关系：是用来说明关联的两个类之间的数量关系

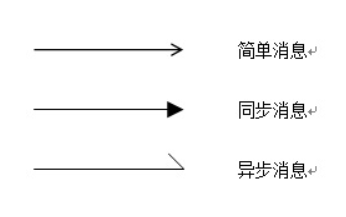
多重度表示为一个整数范围n..m，整数n定义所连接的最少对象的数目，而m则为最多对象数 （当不知道确切的最大数时，最大数用\*号表示）

顺序图：用来描述对象之间动态的交互关系，着重体现对象间消息传递的时间顺序

状态图：用来描述一个特定对象的所有可能状态及其引起状态转移的事件

顺序图将交互关系表现为一个二维图，纵向是时间轴，时间沿竖线向下延伸

同步消息：发送消息的对象要等到接收消息的对象执行完所有操作后，发送消息的对象才能继续执行自己的操作

存活时间

错题：

部署图 显示了系统的硬件，安装在硬件上的软件，以及用于连接异构的机器之间的中间件

采用UML进行面向对象开发时，部署图通常在实施阶段被使用

用例图（也可称用例建模）描述的是外部执行者（Actor）所理解的系统功能。用例图用于需求 分析阶段

业务用例不研究“软件系 统”需求，它关心是一个“业务组织”对外提供哪些服务，支持那些业务过程。业务用例描述的是 业务参与者如何使用业务组织提供的服务的过程。因此业务用例其实是一种业务流程

业务对象模型是描述业务用例实现的对象模型，即业务结构以及结构元素如何完成业务用例

业务用例和参与者一起描述组织支持的业务过程

UML的设计视图包含了类、接口和协作，设计视图的静态方面由类图，对象图表现，动态方面由交互图、状态图和活动图表现

**算法：**

特性：有穷性，确定性，输入，输出，可行性

复杂度：一个是算法效率的度量（时间复杂度），一个是算法运行所需要的 计算机资源量的度量（空间复杂度）

时间复杂度：用O()来表示

sum=0; //执行1次

for(i=1;i<=n;i++) //执行n次

for(j=1;j<=n;j++) //执行n 2次

sum++; //执行n 2次

本程序段的T(n)=2n 2+n+1，时间复杂度应取指数级别最高的，所以为O(n 2 )平方

现时间复杂度和空间复杂度之间有着微妙的关系，经常可以 相互转换，也就是可以利用空间来换时间，也可以用时间来换空间

O（f（n）），给出了算法运行时间的上界，一般用来表达最坏情况下的时间复杂度，这也是平 时最常见的一种表达表式；

Ω（f（n）），给出了算法运行时间的下界，一般用来表达最好情况下的时间复杂度；

Θ（f（n）），给出了算法运行时间的上界和下界

**设计模式**

分类：

创建性 模式处理的是对象的创建过程，结构性模式处理的是对象/类的组合，行为性模式处理类和对象间的 交互方式和任务分布

简单工厂模式：

缺点：工厂决定类的实例化，当有新产品要加入到系统中 时，必须修改工厂类，以加入必要的处理逻辑，一旦它无法确定要对哪个类进行实例化，就无法使用该模式

抽象工厂模式与工厂模式最大的区别在于：工厂模式针对的是一个产品的等级结构，而抽象工 厂模式则针对的是多个产品等级结构

单例模式的要点：一是某个类只能有一个 实例；二是它必须自选创建这个实例；三是它必须自行向整个系统提供这个实例

Bulider&&原型模式：用户不知道内部的具体构建细节



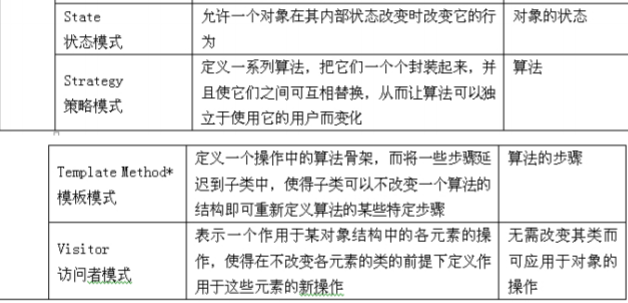
门面（Facade）：提供一个统一的接口去访问多个子系统的多个不同的接口。门面模式定义了一个高层次的接口，使得子系统更容易被使用

策略（Strategy）模式定义一系列的算法，将每一个算法封装起来，并让它们可以相互替换。 策略模式让算法独立于使用它的客户而变化

代理模式通过提供与对象相同的接口来控制对这个对象的访问



享元：系统只使用少量的对象， 而这些对象都相近、状态变化很小、对象使用次数较多



在面向对象软件开发过程中，采用设计模式目的就在于要重用成功的软件开发经验

采用设计模式能够复用相似问题的相 同解决方案，加快设计的速度

设计模式是用一种固定的解决方案来解决某一类问题

优点是方案出错的可能 性很小，因为这些方案都是经过很多人实践总结出来的；

第二是适应需求变化

面向对象技术：

面象分析：不注重系统实现和测试

设计：定义对象和对象如何协作来满足需求

实现：实现系统

测试：验证正确性

对象特性：清晰的边界、良好的定义、可扩展性

面向对象分析步骤：

1. 分析问题域，建立用例模型
2. 发现定义对象和类
3. 识别对象内部特征
4. 识别对象外部联系
5. 识别对象之间交互

面向对象分析活动：

认定对象

组织对象

描述对象间相互作用

定义对象的操作

定义对象内部信息

面向对象设计：

高层模块不应该依赖底层模块，两者都应该依赖抽象

抽象不应该依赖细节，细节应依赖抽象

错题：

继承 子类拥有父类所有属性和方法T

继承可以多重，不同程序

变量具有类型属性，常量没有（有数据类型）

指针变量：

存放数据地址，所用存储空间与内容不一样

指针变量加一相当于地址加一是有意义的

指针变量可以静态定义

基于构建的开发：

强调使用可复用的软件来设计和构建系统，对构建进行 合格性检验、适应性修改，再集成到系统