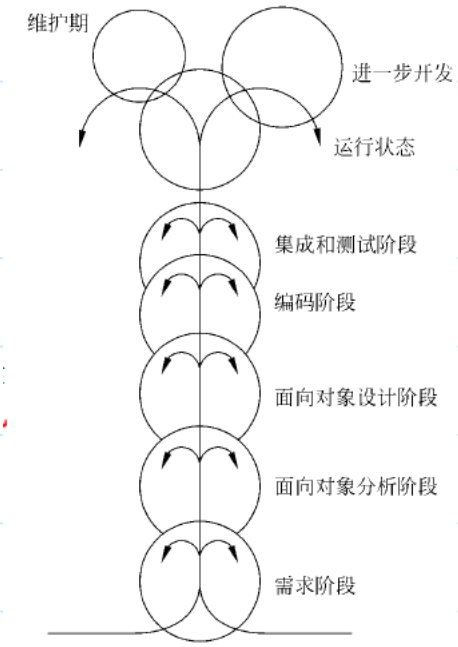
6-3 面向对象的设计

2019年5月8日 22:11

1 面向对象设计概述
2 系统设计
3 包的设计：从逻辑角度
4 数据库设计

1 面向对象设计概述

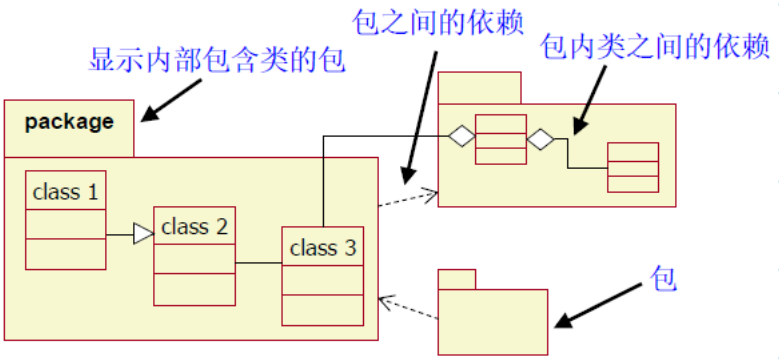
面向对象的设计

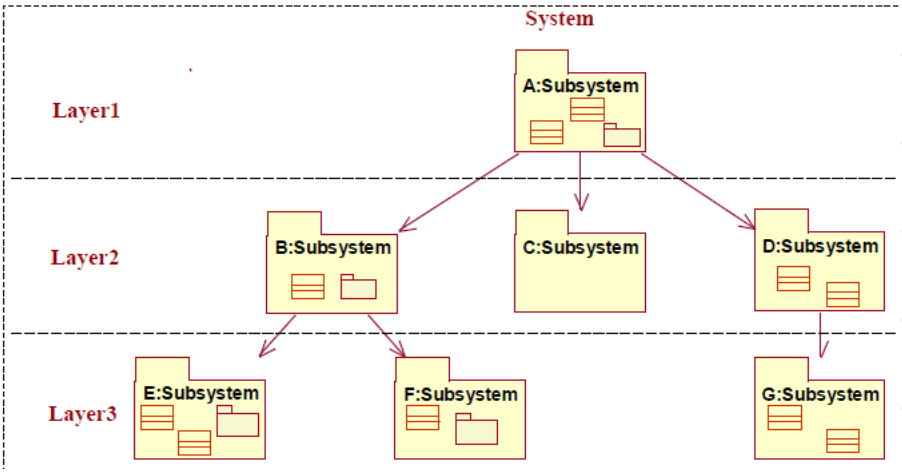
- 传统的结构化方法：分析阶段与设计阶段分得特别清楚，分别使用两套完全不同的建模符号和建模方法；
- 面向对象的设计(OOD)：OO各阶段均采用统一的“对象”概念，各阶段之间的区分变得不明显，形成“无缝”连接。
- 因此，OOD中仍然使用“类、属性、操作”等概念，是在OOA基础上的进一步细化，更加接近底层的技术实现。
- 

面向对象设计中的基本元素

- 基本单元：设计类(designclass)—对应于OOA中的“分析类”
- 为了系统实现与维护过程中的方便性，将多个设计类按照彼此关联的紧密程度聚合到一起，形成大粒度的“包”(package)；

包之间的依赖

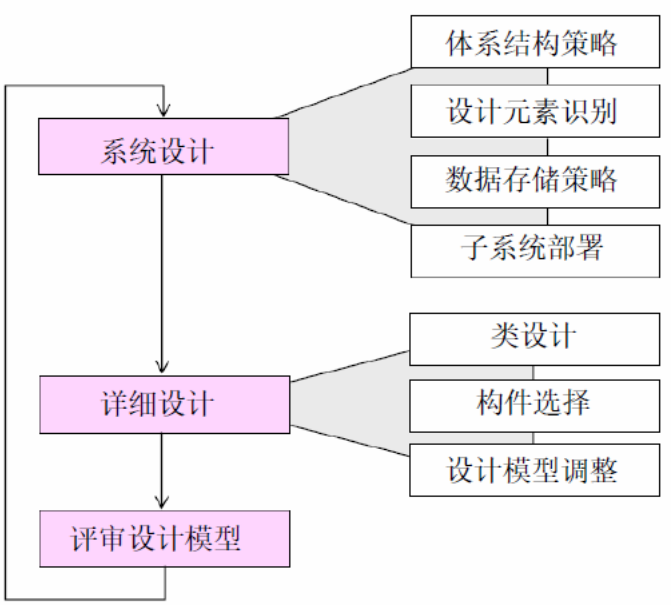
- 

- 一个或多个包聚集在一起，形成“子系统”(subsystem)
- 多个子系统，构成完整的“系统”(system)
- 

面向对象的设计的两个阶段

- 系统设计(SystemDesign)
  - 相当于概要设计（即设计系统的体系结构）；
  - 选择解决问题的基本途径；
  - 决策整个系统的结构与风格；
- 对象设计(ObjectDesign)
  - 相当于详细设计（即设计对象内部的具体实现）；
  - 细化需求分析模型；
  - 识别新的对象；
  - 在系统所需的应用对象与可复用的商业构件之间建立关联；
    - 识别系统中的应用对象；
    - 调整已有的构件；
    - 给出每个子系统/类的精确规格说明。

面向对象设计的过程

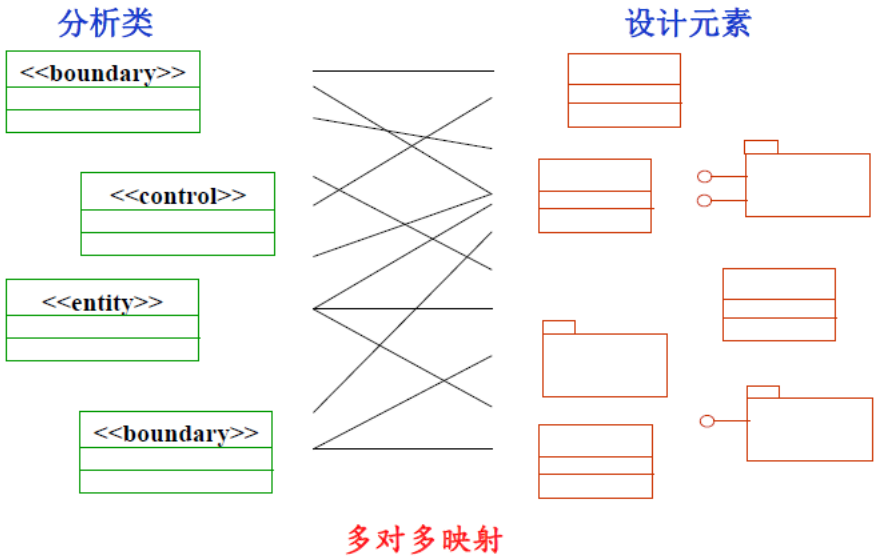
- 

2 系统设计

- 设计系统的体系结构 详见3-1和3-2
  - 选择合适的分层体系结构策略，建立系统的总体结构：分几层？每层的功能分别是什么？
- 识别设计元素
  - 识别“设计类”(designclass)、“包”(package)、“子系统”(subsystem)
- 部署子系统 详见3-3
  - 选择硬件配置和系统平台，将子系统分配到相应的物理节点，绘制部署图(deploymentdiagram)
- 定义数据的存储策略
- 检查系统设计

3 包的设计：从逻辑角度

识别设计元素

- 

确定设计元素的基本原则

- 如果一个“分析类”比较简单，代表着单一的逻辑抽象，那么可以将其一对一的映射为“设计类”；
  - 通常，主动参与者对应的边界类、控制类和一般的实体类都可以直接映射成设计类。
- 如果“分析类”的职责比较复杂，很难由单个“设计类”承担，则应该将其分解为多个“设计类”，并映射成“包”或“子系统”；
- 将设计类分配到相应的“包”或“子系统”当中；
  - 子系统的划分应该符合高内聚低耦合的原则。

图书管理系统：识别设计元素

- 

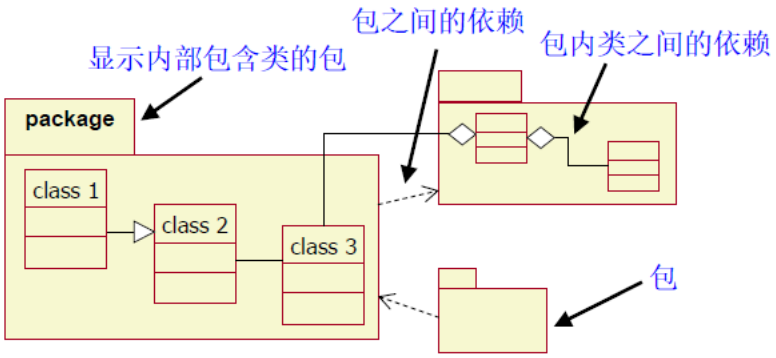
类型	分析类	设计元素
	LoginForm	“设计类” *LoginForm*
	BrowseForm	“设计类” *BrowseForm*
	……	……
	MailSystem	“子系统接口” *IMailSystem*
	BrowseControl	“设计类” *BrowseControl*
	MakeReservationControl	“设计类” *MakeReservationControl*
	……	……
	BorrowerInfo	“设计类” *BorrowerInfo*
	Loan	“设计类” *Loan*
	……	……

绘制包图(package diagram)

- 对一个复杂的软件系统，要使用大量的设计类，这时就必要把这些类分组进

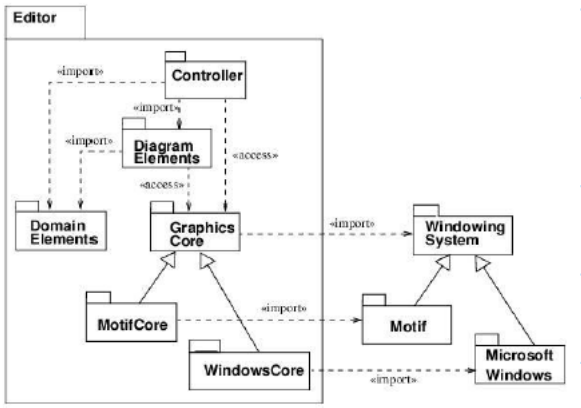
行组织；

- 把在语义上接近且倾向于一起变化的类组织在一起形成“包”，既可控制模型的复杂度，有助于理解，而且也有助于按组来控制类的可见性；
- 结构良好的包是松耦合、高内聚的，而且对其内容的访问具有严密的控制。



包之间的关系

- 类与类之间的存在的“聚合、组合、关联、依赖”关系导致包与包之间存在依赖关系，即“包的依赖”(dependency)；
- 类与类之间的存在的“继承”关系导致包与包之间存在继承关系，即“包的泛化”(generalization)；



模型管理视图—包图

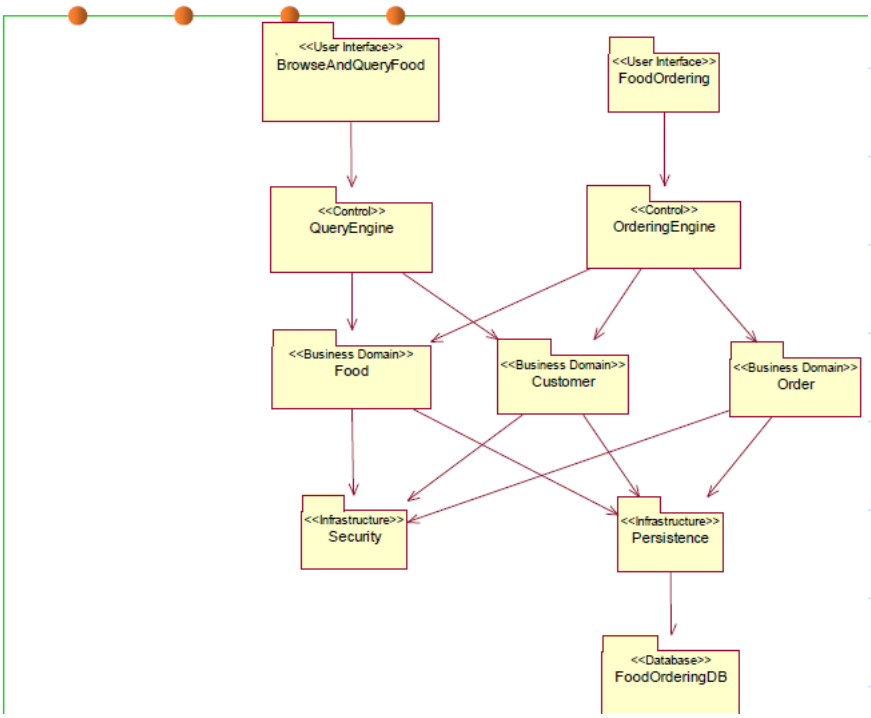
- 包图(PackageDiagram)是在UML中用类似于文件夹的符号表示的模型元素的组合。
- UML包图提供了组织元素的方式，包能够组织任何事物：类、其它包、用例等，系统中的每个元素都只能为一个包所有，一个包可嵌套在另一个包中。注：UML中的包为广义概念，不等同于Java包的狭义概念。
  - 类
  - 接口
  - 组件
  - 节点

    - 协作
    - 用例图
    - 以及其他包
- 使用包图可以将相关元素归入一个系统。一个包中可包含附属包、图表或单个元素。

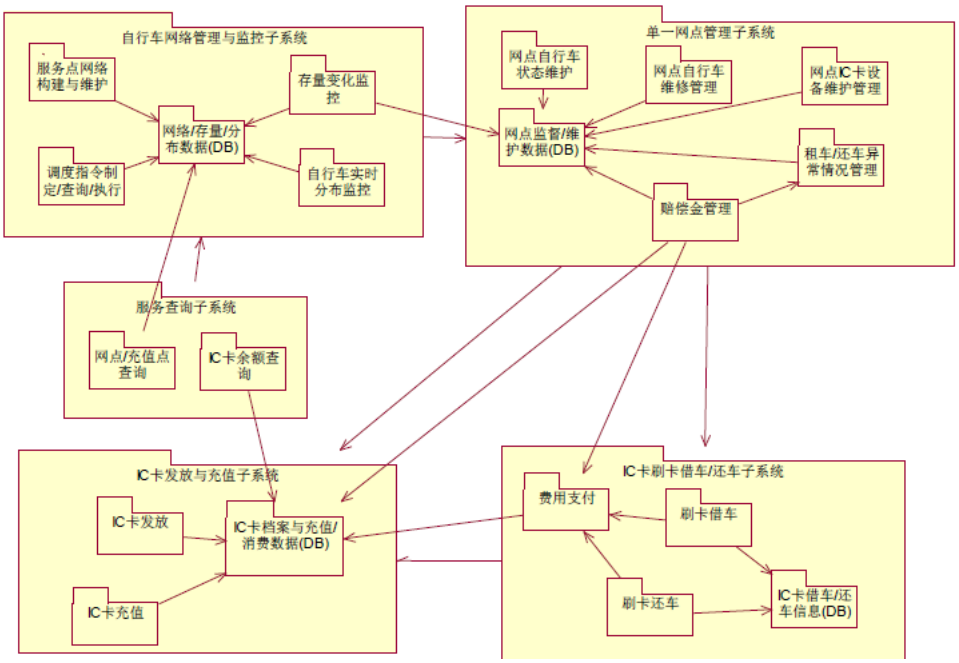
绘制包图的方法

- 分析设计类，把概念上或语义上相近的模型元素纳入一个包。
- 可以从类的功能相关性来确定纳入包中的类：
    - 如果一个类的行为和/或结构的变更要求另一个相应的变更，则这两个类是功能相关的。
    - 如果删除一个类后，另一个类便变成是多余的，则这两个类是功能相关的，这说明该剩余的类只为那个被删除的类所使用，它们之间有依赖关系。
    - 如果两个类之间大量的频繁交互或通信，则这两个类是功能相关的。
    - 如果两个类之间有一般/特殊关系，则这两个类是功能相关的。
    - 如果一个类激发创建另一个类的对象，则这两个类是功能相关的。
- 确定包与包之间的依赖关系(<<import>>、<<access>>等)；
- 确定包与包之间的泛化关系；
- 绘制包图。

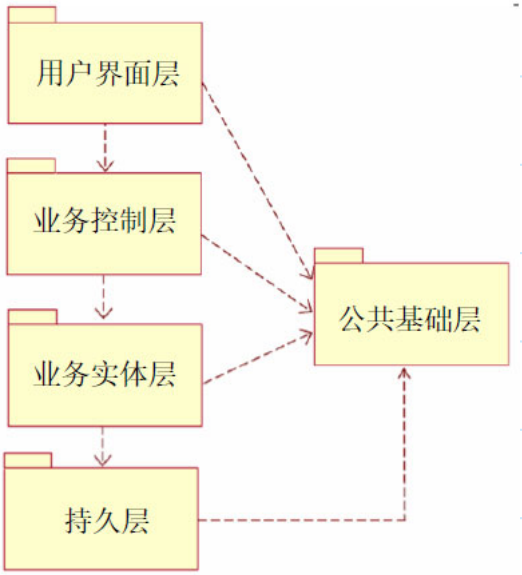
包图示例1



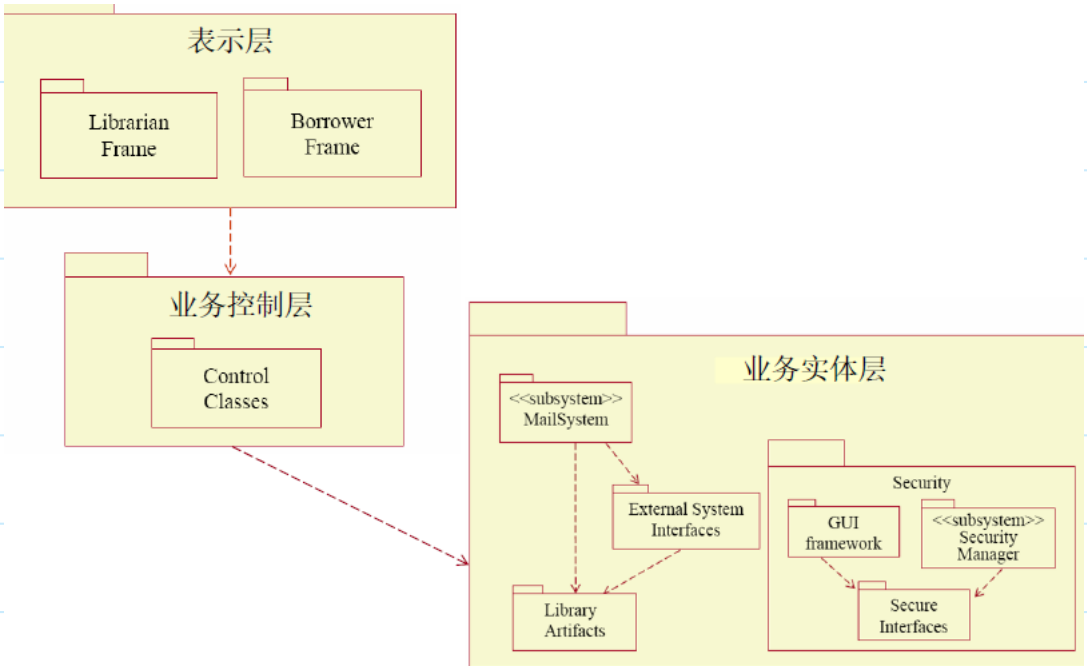
包图示例2



按实现技术划分包的分层结构



图书管理系统：软件体系结构

- 

JAVA中的“package”

java.io.InputStream is = java.lang.System.in;

java.io.InputStreamReader isr= new java.io.InputStreamReader(is);

java.io.BufferedReader br = new java.io.BufferedReader(isr);

- import java.lang.System;
- import java.io.InputStream;
- import java.io.InputStreamReader;
- import java.io.BufferedReader;

InputStream = System.in;

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

package cn.edu.hit.cs;

public class A{}

---

package cn.edu.hit.cs;

public class B{}

---

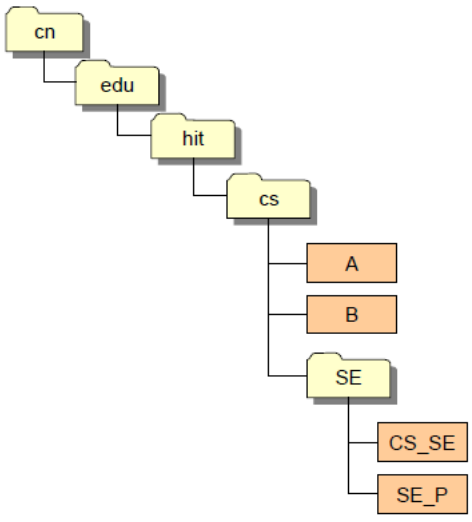
package cn.edu.hit.cs.se;

public class CS_SE{}

---

package cn.edu.hit.cs.se;

public class CS_SE_P{}



4 数据库设计

类和关系数据表的关系

- OOP以class为基本单位，所有的object都是运行在内存空间当中；
- 若某些object的信息需要持久化存储，那么就需要用到database，将object的属性信息写入关系数据表；
    - 假如淘宝没有“保存购物车内容”的功能（意即若不购买，下次进入之后购物车中的内容就被清空），那么“购物车”这个实体的属性就不需要关系数据表。
- 在系统执行某些功能的时候，需要首先从database中将信息取出，使用各实体类的new操作构造相应的object，在程序运行空间中使用(充分利用继承/组合/聚合/关联/依赖关系在各object之间相互导航)。
- 在进行OO分析和设计时完全可以将database忘掉，后续再加以考虑。

数据存储设计

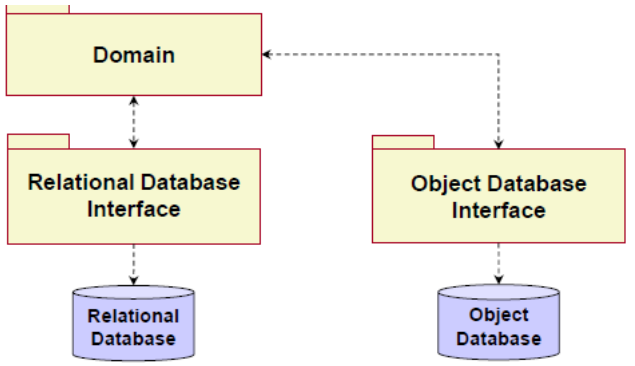
- “对象”只是存储在内存当中，而某些对象则需要永久性的存储起来；——持久性数据(persistentdata)
- 数据存储策略
    - 数据文件：由操作系统提供的存储形式，应用系统将数据按字节顺序存储，并定义如何以及何时检索数据。
    - 关系数据库：数据是以表的形式存储在预先定义好的成为Schema的类型中。
    - 面向对象数据库：将对象和关系作为数据一起存储。
    - 存储策略的选择：取决于非功能性的需求；

数据存储策略的tradeoff

- 何时选择文件?
    - 存储大容量数据、临时数据、低信息密度数据
- 何时选择数据库?
    - 并发访问要求高、系统跨平台、多个应用程序使用相同数据
- 何时选择关系数据库?
    - 复杂的数据查询
    - 数据集规模大
- 何时选择面向对象数据库?

- 数据集处于中等规模
- 频繁使用对象间联系来读取数据

数据存储策略

- 如果使用OO数据库，那么数据库系统应提供一个接口供应用系统访问数据
- 如果使用关系数据库，那么需要一个子系统来完成应用系统中的对象和数据库中数据的映射与转换。
- 

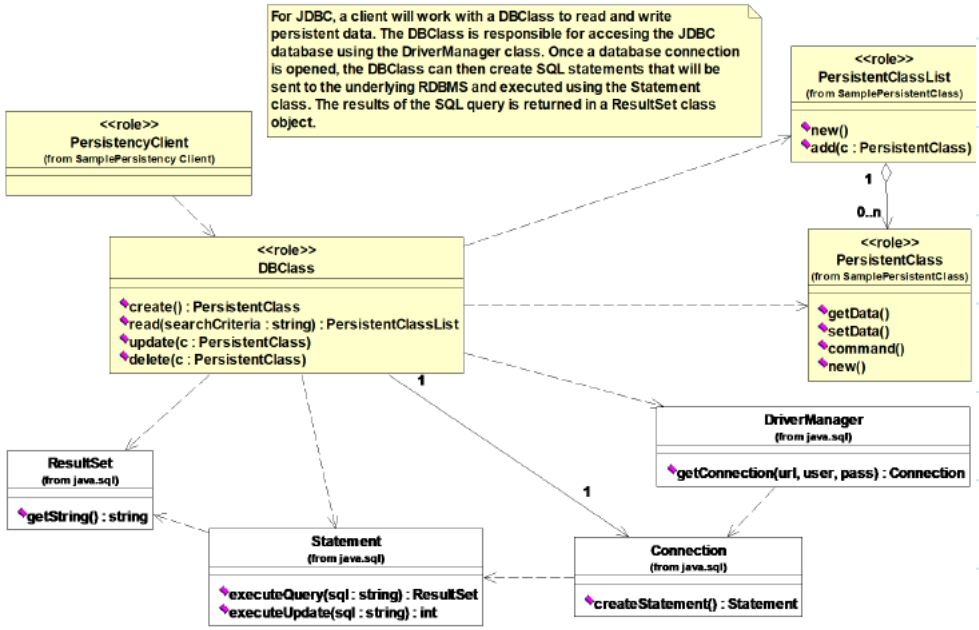
OO设计中的数据库设计

- 核心问题：对那些需要永久性存储的数据，如何将UML类图映射为数据库模型。
- 本质：把每一个类、类之间的关系分别映射到一张表或多张表；
- UML class diagram ->Relation DataBase(RDB)
- 两个方面：
  - 将类(class)映射到表(table)
  - 将关联关系(association)映射到表(table)

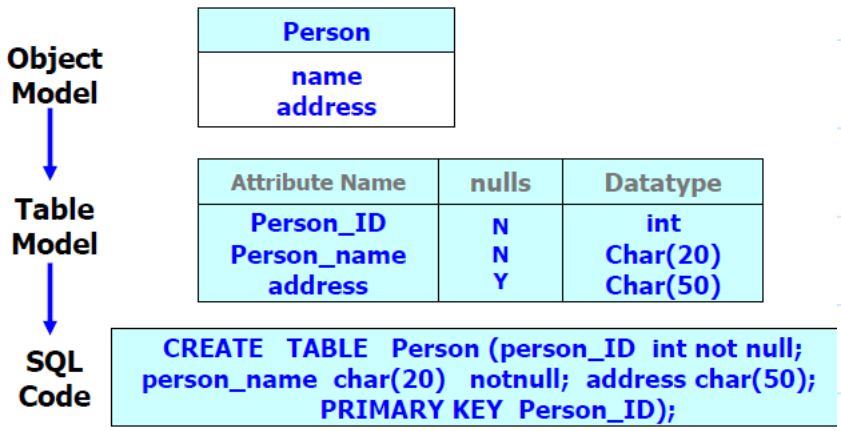
对象关系映射(ORM)

- 对象关系映射(ObjectRelationalMapping,ORM)：
  - 为了解决面向对象与关系数据库存在的互不匹配的现象；
  - 通过使用描述对象和数据库之间映射的元数据，将OO系统中的对象自动持久化到关系数据库中。
- 目前流行的ORM产品：
  - Apache OJB
  - Hibernate
  - JPA(Java Persistence API)
  - Oracle TopLink

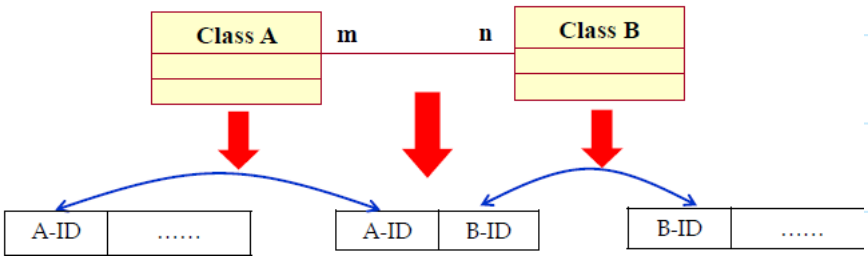
Example:Persistency:RDBMS:JDBC

- 

将对象映射到关系数据库

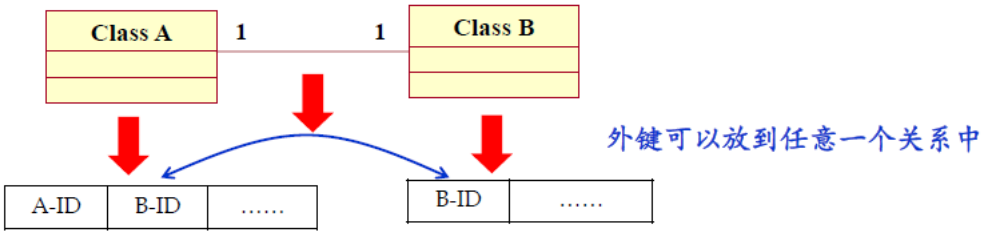
- 最简单的映射策略——“一类一表”：表中的字段对应于类的属性，表中的每一行数据记录对应类的实例(即对象)。
  - 

将关联映射到关系数据库(统一的、简单的途径)

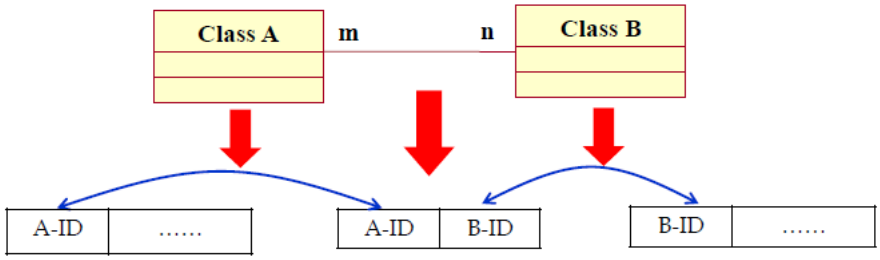
- 不管是1:1、1:n还是m:n的关联关系，均可以采用以下途径映射为关系数据表
  - A与B分别映射为独立的数据表，然后再加入一张新表来存储二者之间的关联；
  - 

将关联映射到关系数据库(1:1和m:n的关联关系)

- Implementing 1 to 1

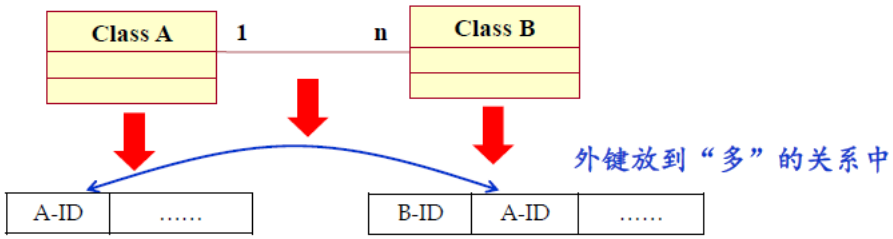


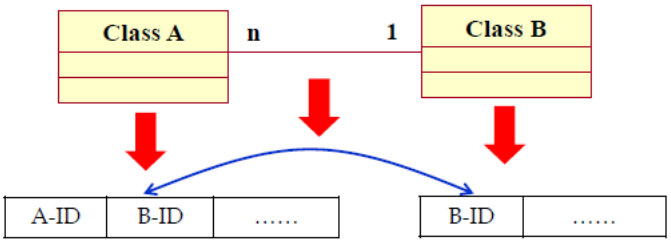
- Implementing m:n



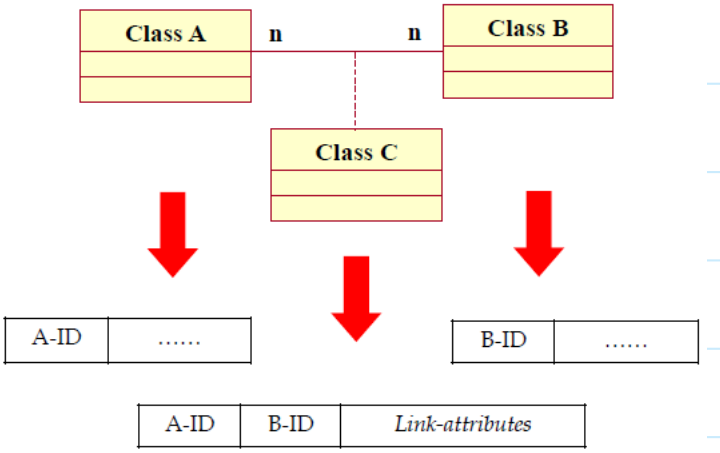
将关联映射到关系数据库(1:n的关联关系)

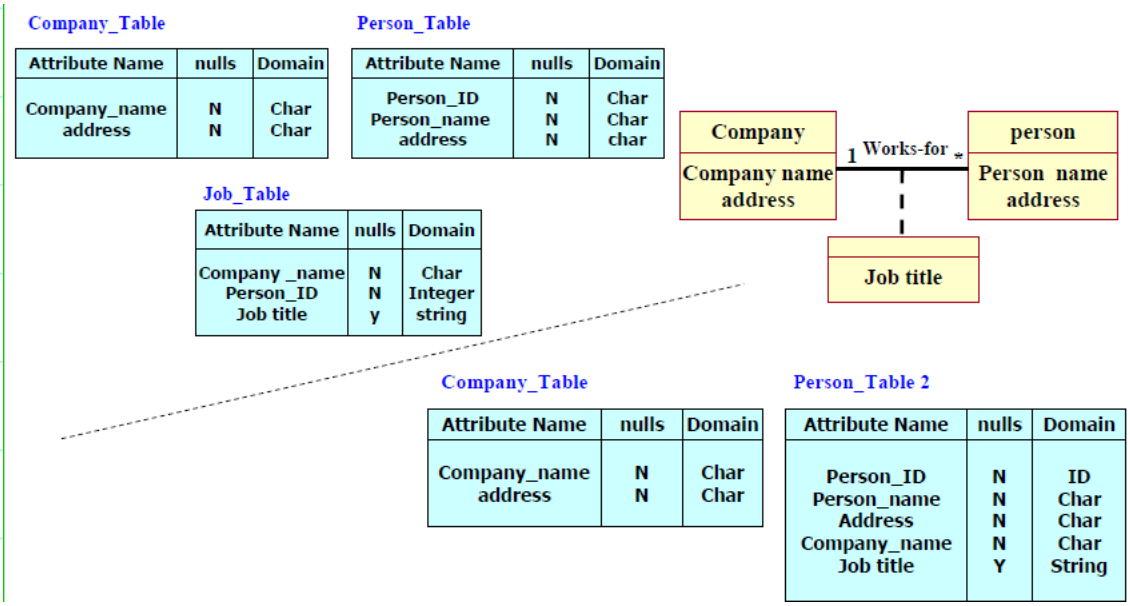
Implementing 1:n

- 



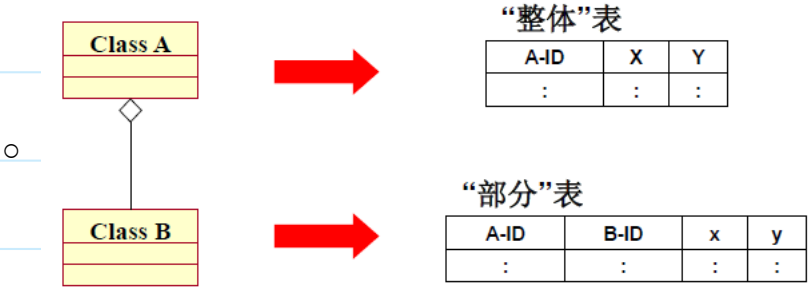
将关联映射到关系数据库(基于关联类的关联关系)

- 

- 

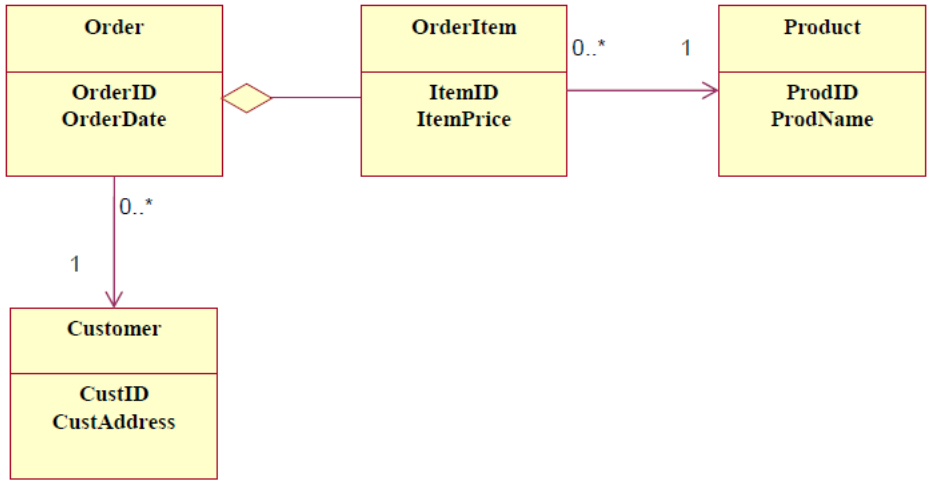
将组合/聚合关系映射到关系数据库

- 实现方法：类似于1:n的关联关系
  - 建立“整体”表
  - 建立“部分”表，其关键字是两个表关键字的组合

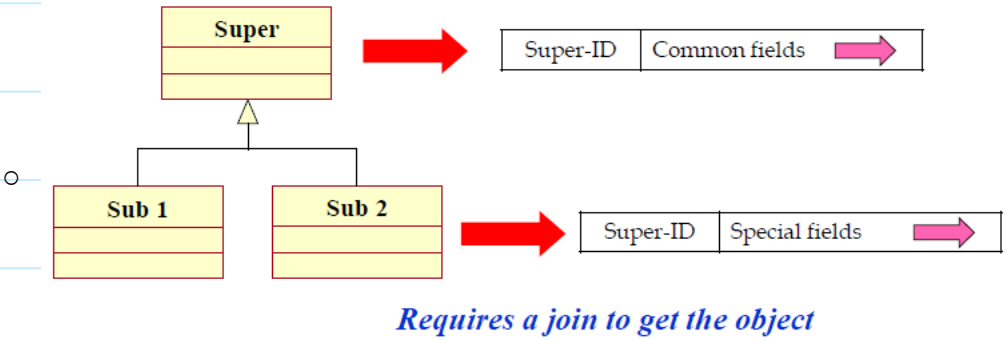
- 

OO到DB的映射

为以下类图设计关系数据表　　讨论

- 

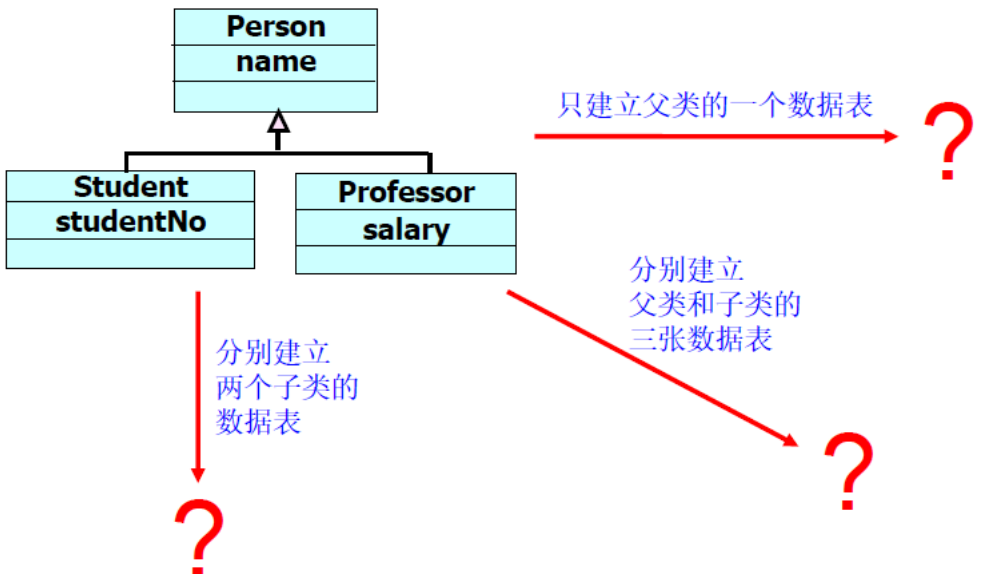
将继承关系映射到关系数据库

- 策略1：分别建立父类和子类的三张数据表
  - 

Requires a join to get the object

- 策略2：将子类的属性上移到父类所对应的数据表中，该表包括父类的属性、各子类的全部属性；
- 策略3：将父类的属性下移到各个子类所对应的数据表中

OO到DB的映射

- 

检查系统设计

- 检查“正确性”
  - 每个子系统都能追溯到一个用例或一个非功能需求吗?
  - 每一个用例都能映射到一个子系统吗?
  - 系统设计模型中是否提到了所有的非功能需求?
  - 每一个参与者都有合适的访问权限吗?
  - 系统设计是否与安全性需求一致?
- 检查“一致性”
  - 是否将冲突的设计目标进行了排序?
  - 是否有设计目标违背了非功能需求?
  - 是否存在多个子系统或类重名?
- 检查“完整性”
  - 是否处理边界条件?
  - 是否有用例走查来确定系统设计遗漏的功能?
  - 是否涉及到系统设计的所有方面(如硬件部署、数据存储、访问控制、遗留系统、边界条件)?
  - 是否定义了所有的子系统?
- 检查“可行性”
  - 系统中是否使用了新的技术或组件？是否对这些技术或组件进行了可行性研究?
  - 在子系统分解环境中检查性能和可靠性需求了吗?
  - 考虑并发问题了吗?