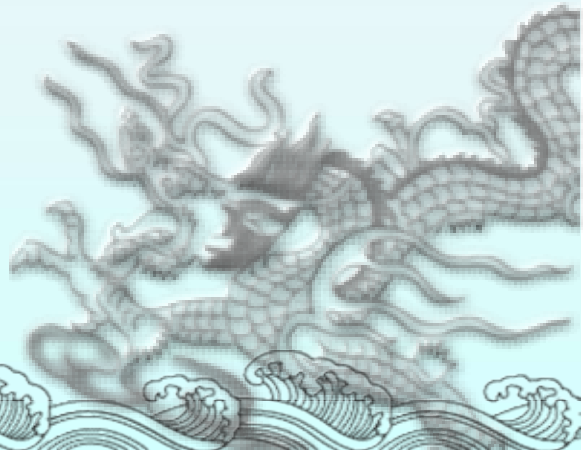
组件图、部署图和包图

内容提纲

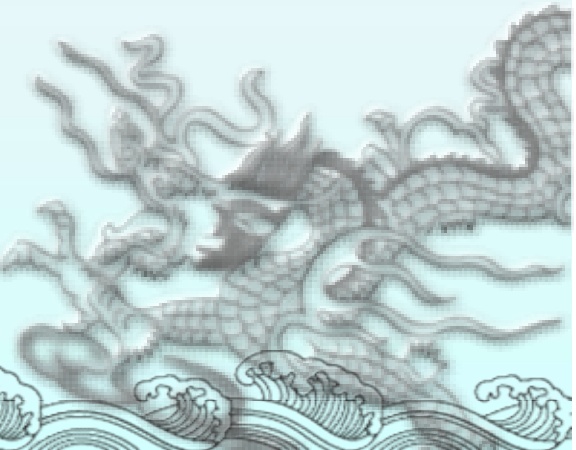
- 组件图
  - 相关概念
  - 实例讲解
- 部署图
  - 相关概念
  - 实例讲解
- 包图
  - 相关概念
  - 实例讲解



组件图

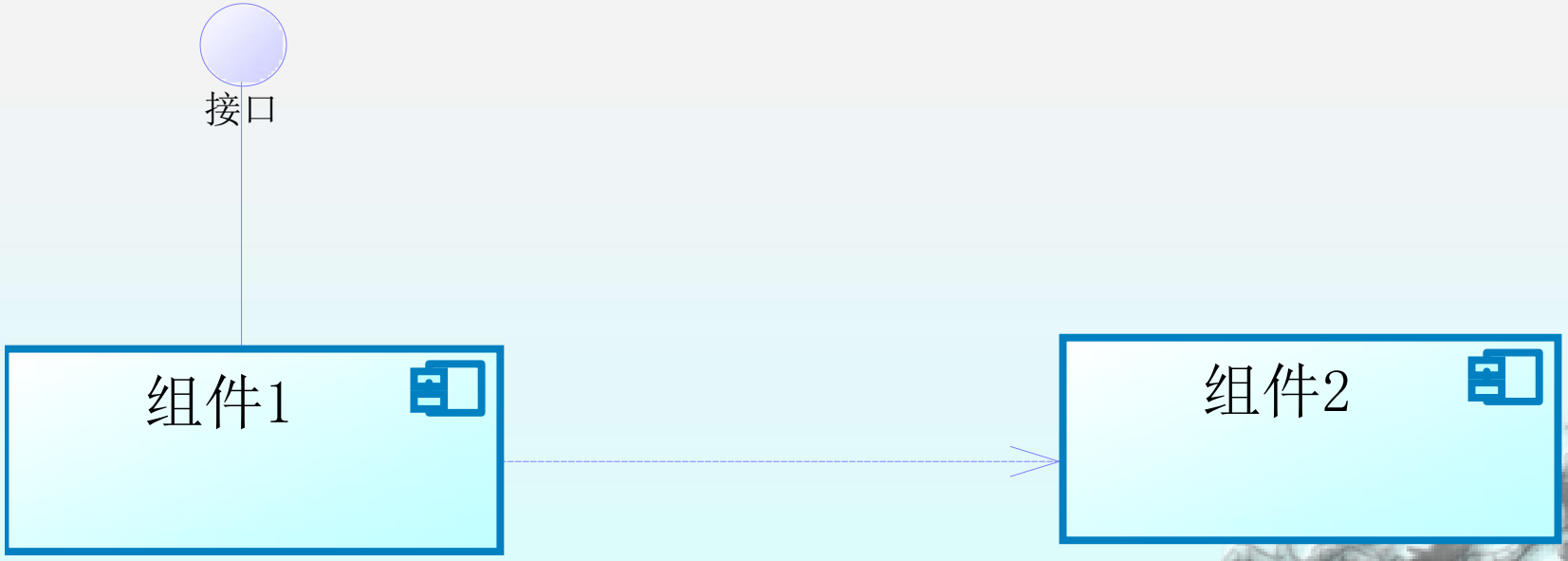
组件图定义

- 组件图又称为构件图(Component Diagram)。组件图中通常包括组件、接口，以及各种关系。组件图显示组件以及它们之间的依赖关系，它可以用来显示程序代码如何分解成模块或组件。一般来说，组件就是一个实际文件，可以有以下几种类型：
  - 源代码组件：一个源代码文件或者与一个包对应的若干个源代码文件。
  - 二进制组件：一个目标码文件，一个静态的或者动态的库文件。
  - 可执行组件：在一台处理器上可运行的一个可执行的程序单位，即所谓可执行程序。



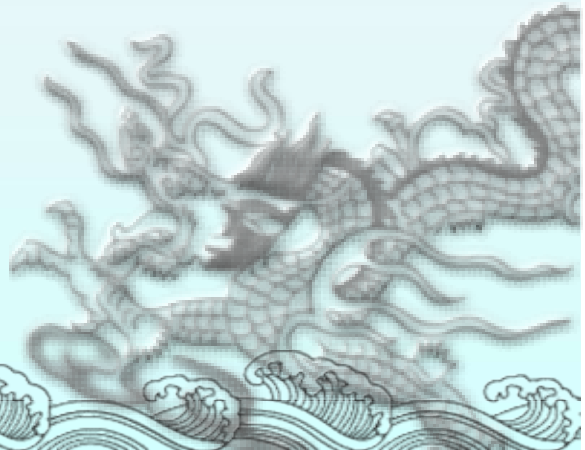
组件图

- 组件图定义
  - 组件图可以用来显示编译、链接或执行时组件之间的依赖关系，以及组件的接口和调用关系。



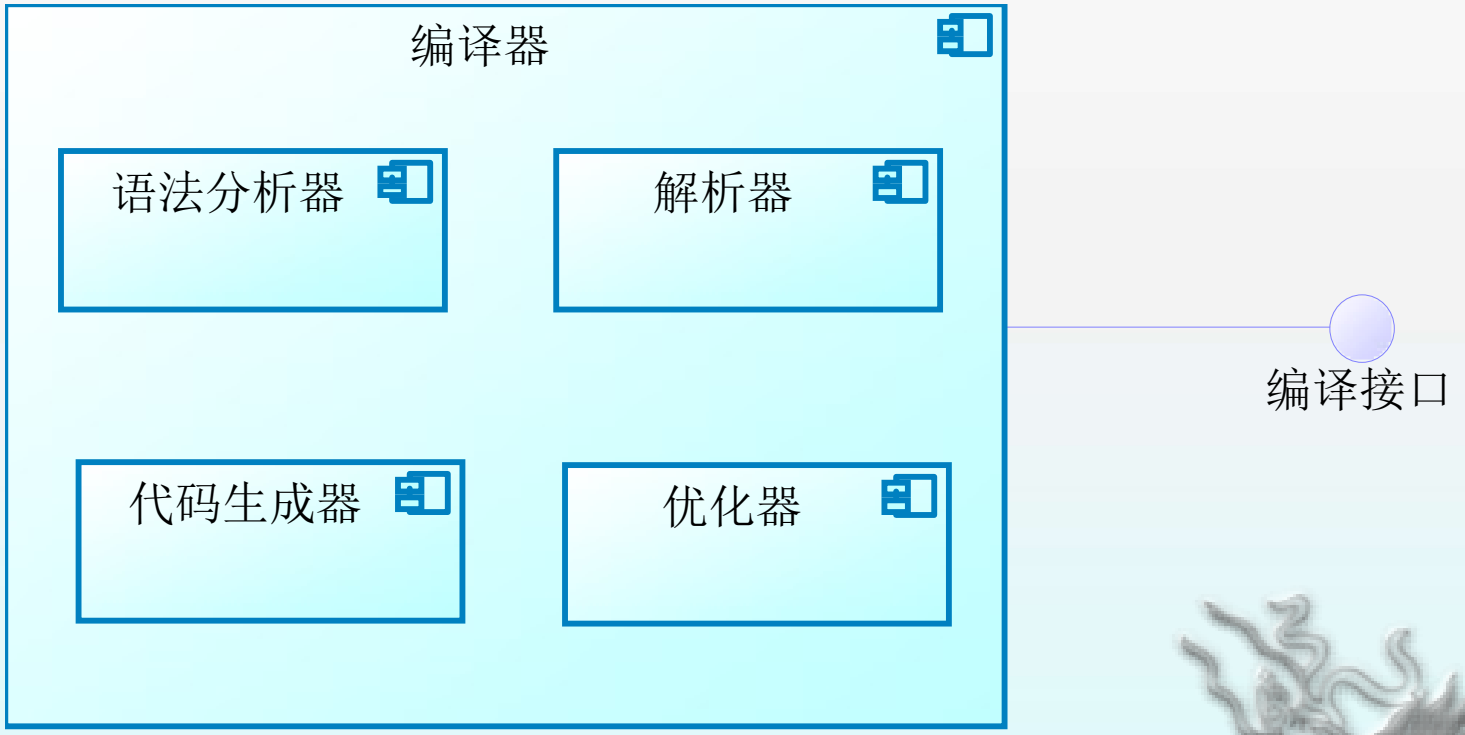
组件图

- 组件图定义
  - 组件间的关系有两种：泛化关系和依赖关系，如果两个不同组件中的类存在泛化关系或依赖关系，那么两个组件之间的关系就表示为泛化关系或依赖关系。
  - 对于由多个组件组成的大系统来说，组件图非常重要。



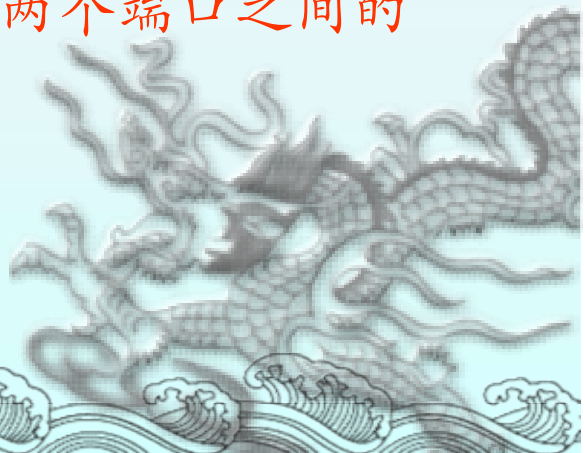
组件图

- 组件图实例



组件图

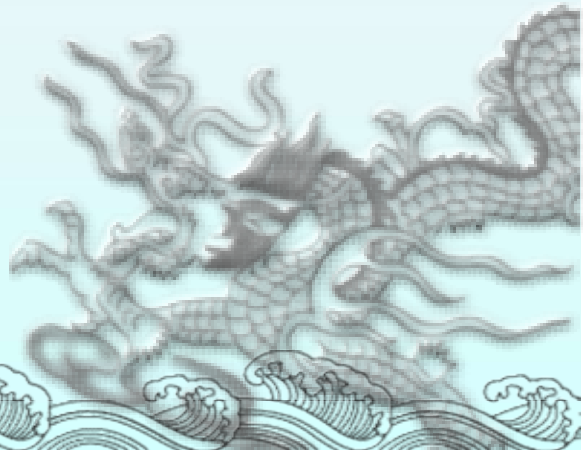
- 组件图组成元素
  - 组件：系统中可以替换的部分，一般对应一个实际文件，如exe、jar、dll等文件，它遵循并提供了一组接口的实现。
  - 接口：一组操作的集合，它指明了由类或组件所请求或者所提供的服务。
  - 部件：组件的局部实现。
  - 端口：被封装的组件与外界的交互点，遵循指定接口的组件通过它来收发消息。
  - 连接件：在特定语境下组件中两个部件之间或者两个端口之间的通信关系。



组件图

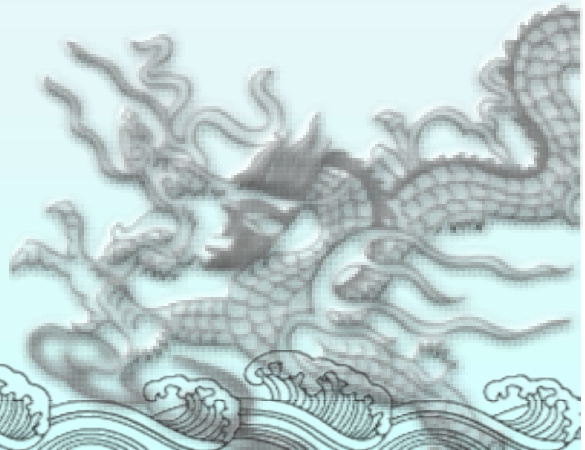
组件图绘制技巧

- 当需要把系统分成若干组件（构件），希望借助接口或组件将系统分解为低层结构并表示其相互关系时需要使用组件图。
- 在绘制组件图时，应该注意侧重于描述系统的静态实现视图的一个方面，图形不要过于简化，应该为组件图取一个直观的名称，在绘制时避免产生线的交叉。
- 注意组件的粒度，粒度过细的构件将导致系统过于庞大，会给版本管理带来问题。



内容提纲

- 组件图
  - 相关概念
  - 实例讲解
- 部署图
  - 相关概念
  - 实例讲解
- 包图
  - 相关概念
  - 实例讲解

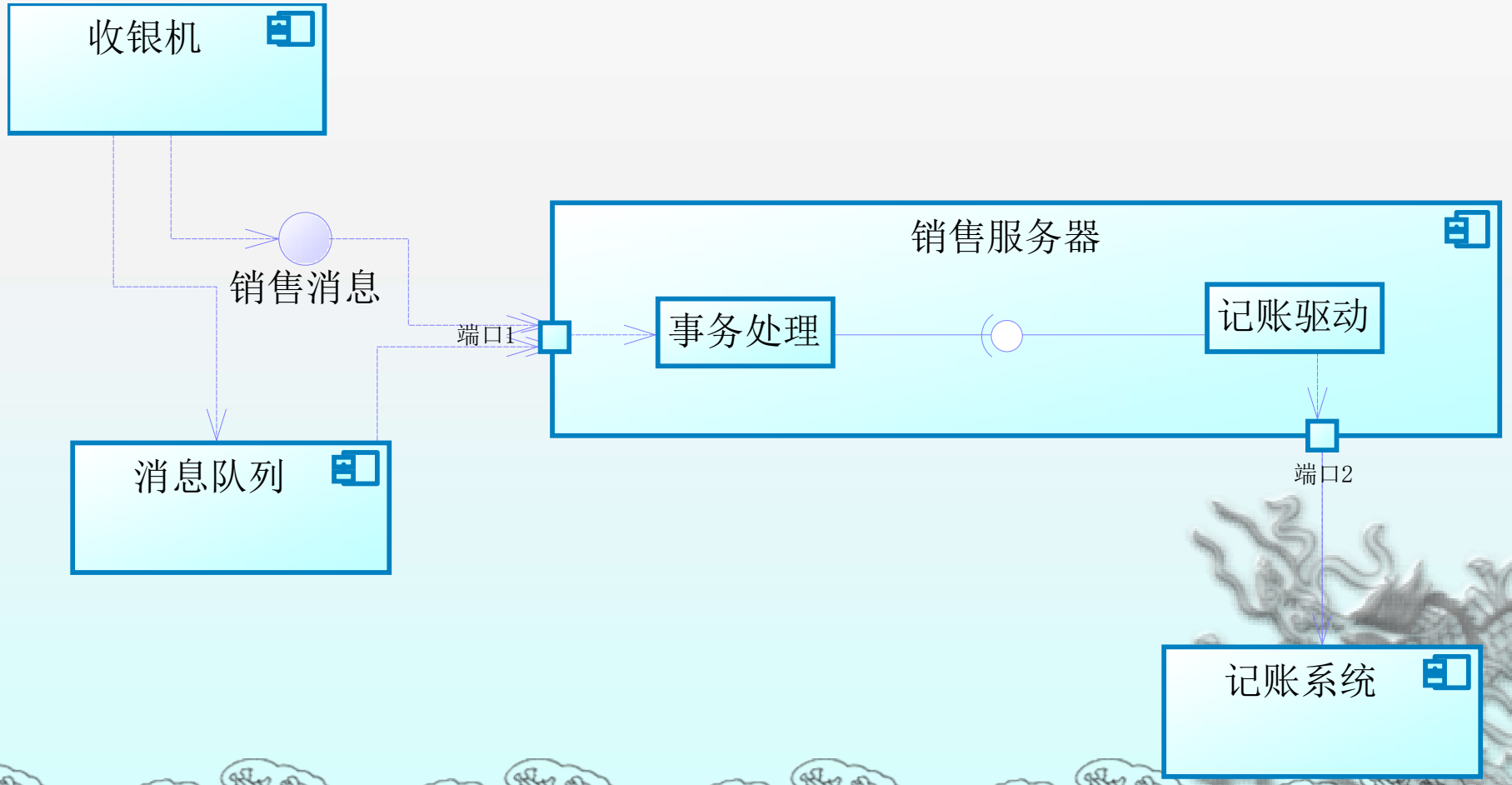


组件图

- 组件图实例分析
  - 在某销售终端系统中，客户端收银机可以通过销售消息接口与销售服务器相连。考虑到网络可能不可靠，需要提供一个消息队列组件。在网络环境畅通时收银机直接与服务器相连；如果网络不可靠则与消息队列交互，当网络可用时队列再与服务器交互。服务器分解为两个主要组件，主要包括事务处理组件和记账驱动组件，记账驱动组件需要和记账系统交互。绘制系统组件图。

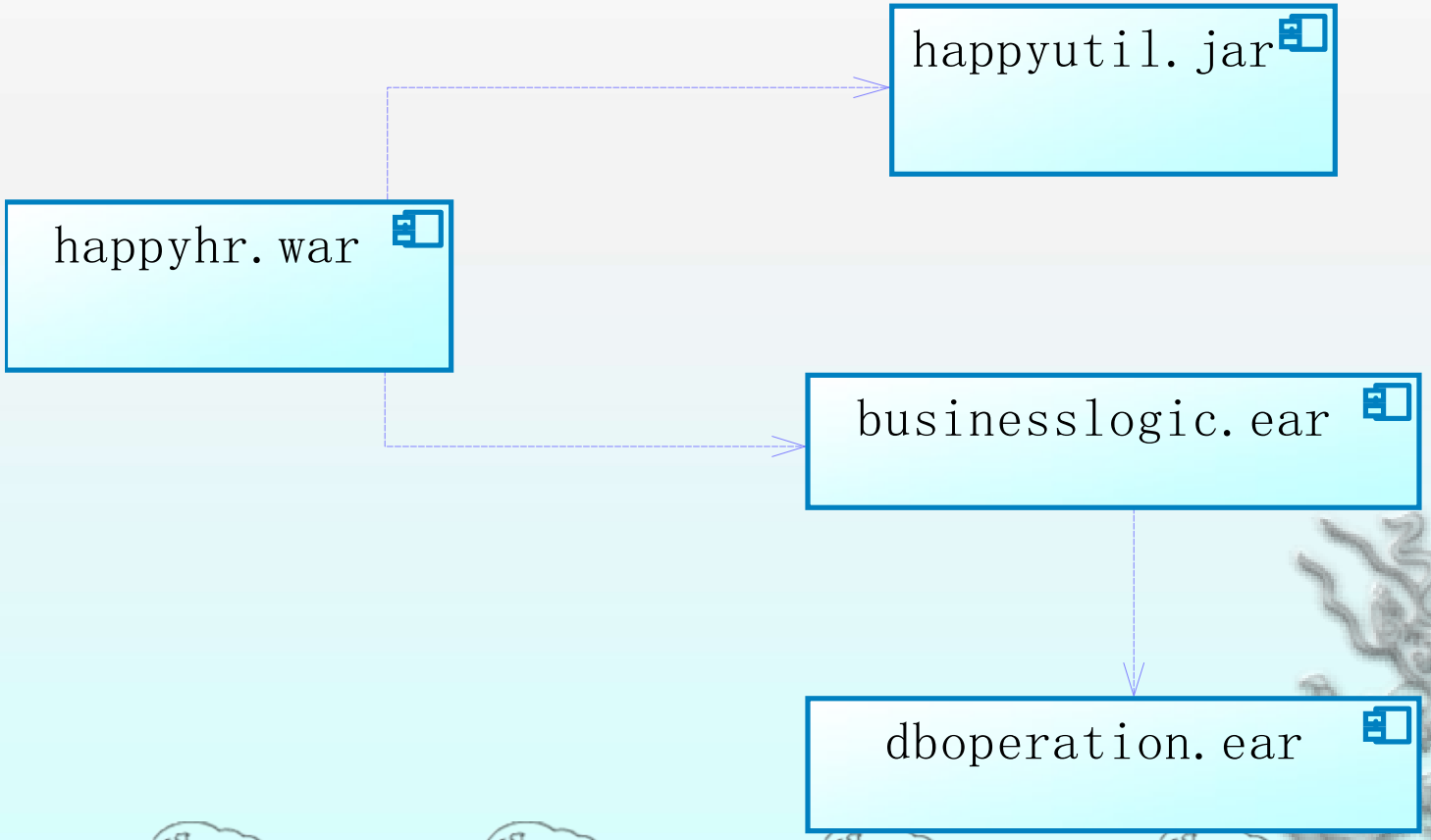
组件图

- 组件图实例分析



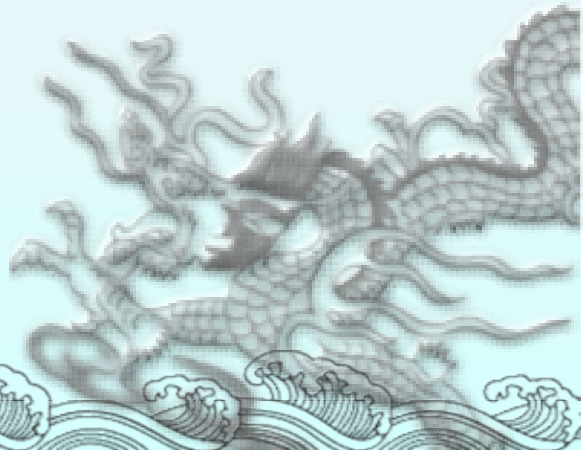
组件图

- 组件图实例分析
  - Java EE项目组件图



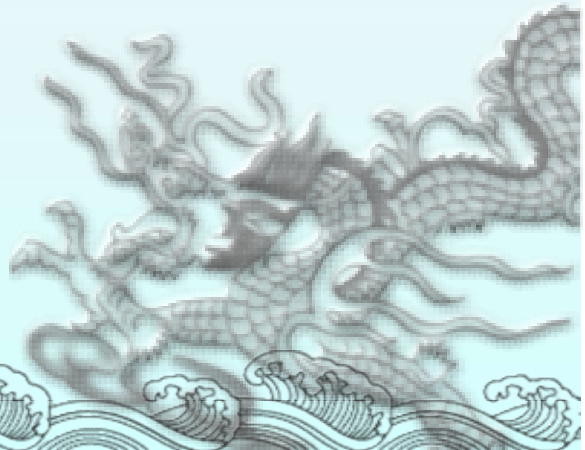
内容提纲

- 组件图
  - 相关概念
  - 实例讲解
- 部署图
  - 相关概念
  - 实例讲解
- 包图
  - 相关概念
  - 实例讲解



部署图

- 部署图又称为环境视图，它用于描述系统所使用的不同元素的物理布局，部署视图通过部署图来表示。在部署图中，系统所涉及的物理硬件称为节点，如计算机和打印机就是系统中典型的节点。

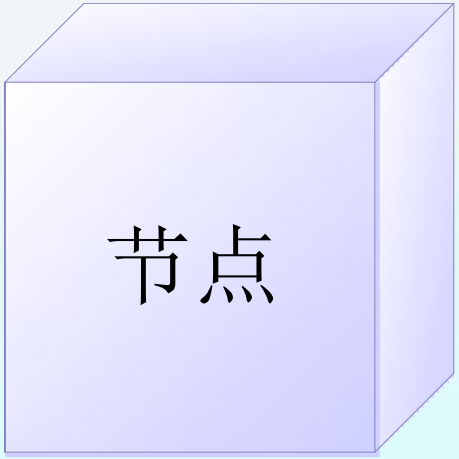


部署图

- 部署图定义
  - 部署图(Deployment Diagram)，也称为实施图，它和组件图一样，是面向对象系统的物理方面建模的两种图之一。组件图是说明组件之间的逻辑关系的，而部署图则是在此基础上更进一步，描述系统硬件的物理拓扑结构及在此结构上执行的软件。部署图可以显示计算节点的拓扑结构和通信路径、节点上运行的软件组件，常常用于帮助理解分布式系统。

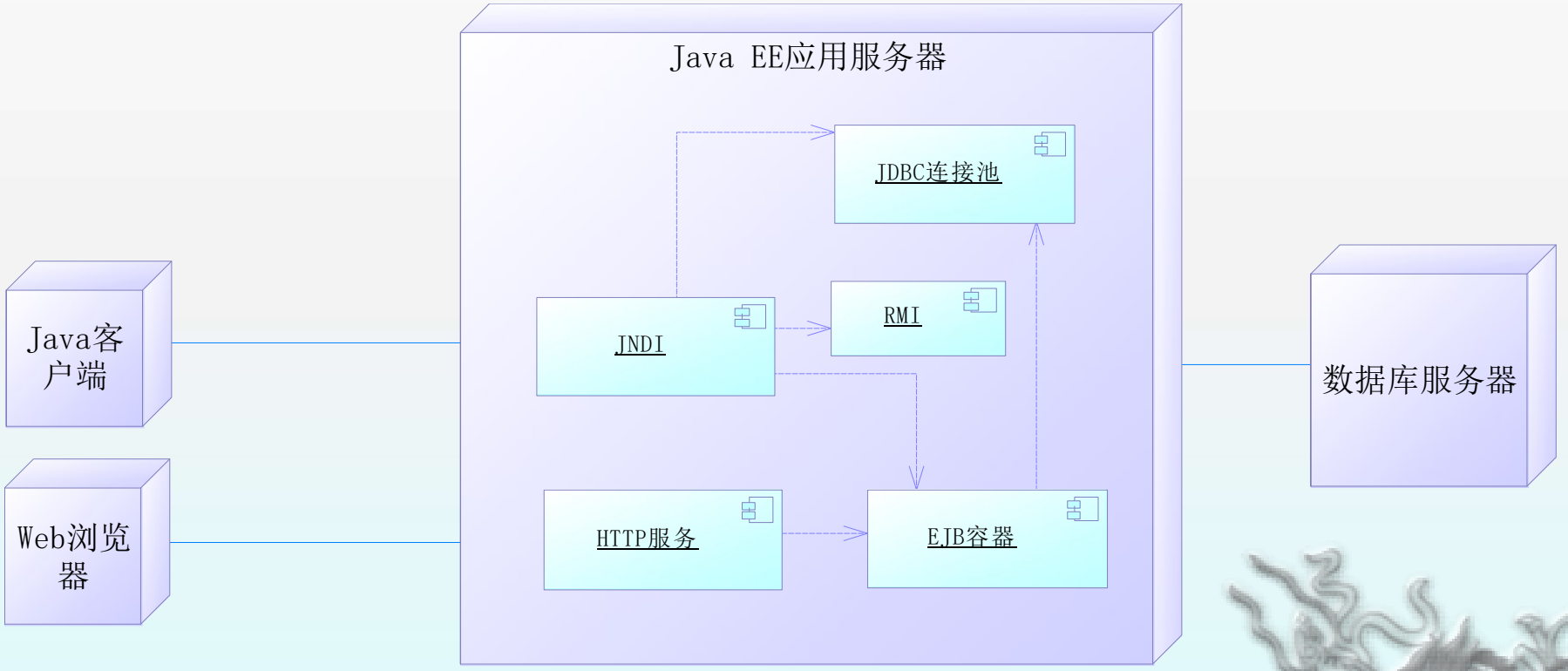
部署图

- 部署图定义
  - 在UML中，部署图显示了系统的硬件和安装在硬件上的软件，以及用于连接异构计算机之间的中间件。部署图通常被认为是一个网络图或者技术架构图。



部署图

- 部署图实例


Java EE应用服务器
JDBC连接池
JNDI
RMI
HTTP服务
EJB容器
Java客户端
Web浏览器
数据库服务器

部署图

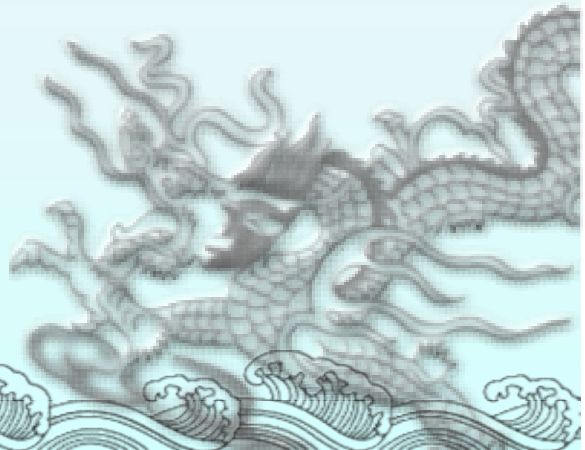
- 部署图组成元素
  - 节点和连接：节点(Node)代表一个物理设备。在 UML 中，使用一个立方体表示一个节点。节点之间的连线表示系统之间进行交互的通信路径，在 UML 中被称为连接。
  - 组件与接口：在部署图中，组件代表可执行的物理代码模块，如一个可执行程序。逻辑上它可以与类或包对应。在面向对象方法中，类和组件等元素并不是所有的属性和操作都对外可见，它们对外提供了可见操作和属性，称为类和组件的接口。

部署图

- 部署图绘制技巧
  - 部署图用于表示何者部署于何处，任何复杂的部署都可以使用部署图描述。
  - 一个部署图只是系统静态部署视图的一个图形表示，在单个部署图中不必捕获系统部署视图的所有内容。
  - 部署图一般以如下三种方式出现：
    - 对嵌入式系统建模（硬件之间的交互）
    - 对客户/服务器系统建模（用户界面与数据的分离）
    - 对全分布式系统建模（多级服务器）

内容提纲

- 组件图
  - 相关概念
  - 实例讲解
- 部署图
  - 相关概念
  - 实例讲解
- 包图
  - 相关概念
  - 实例讲解

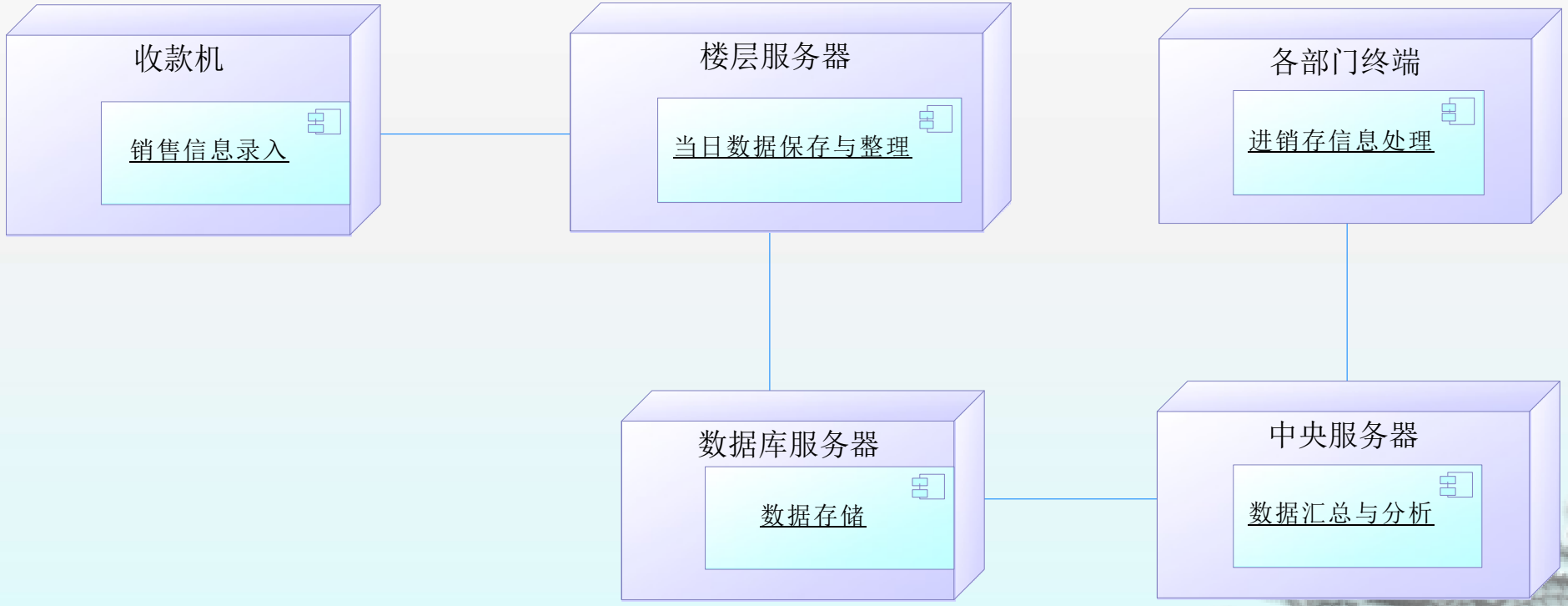


部署图

- 部署图实例分析
  - 某大型商场的信息管理系统是由一个数据库服务器、中央服务器、每个楼层的楼层服务器、各柜台的收款机和各个部门的计算机终端组成的局域网络，它们分别负责商场数据存储、数据的汇总与分析、当日数据的保存与整理、销售信息录入和进销存信息处理等各种业务处理。用部署图描述该系统在不同硬件上的配置情况。

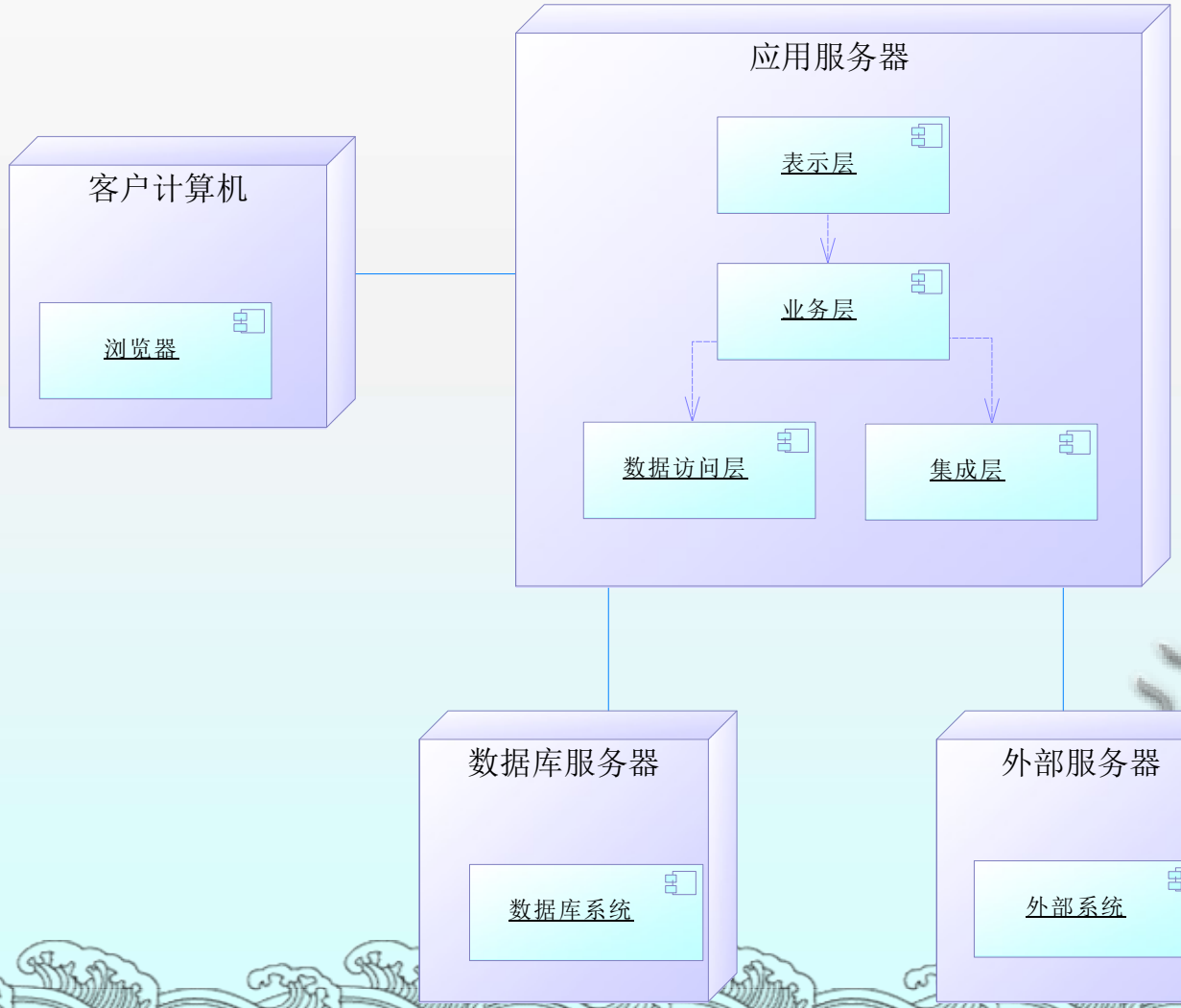
部署图

- 部署图实例分析


收款机
销售信息录入
楼层服务器
当日数据保存与整理
各部门终端
进销存信息处理
数据库服务器
数据存储
中央服务器
数据汇总与分析

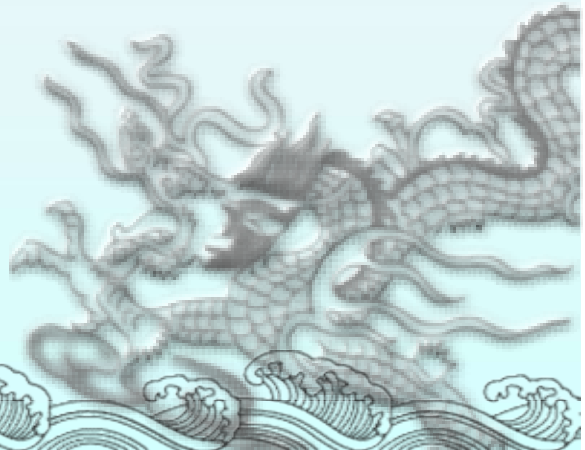
部署图

- 部署图实例分析
  - 典型B/S系统部署图



内容提纲

- 组件图
  - 相关概念
  - 实例讲解
- 部署图
  - 相关概念
  - 实例讲解
- 包图
  - 相关概念
  - 实例讲解

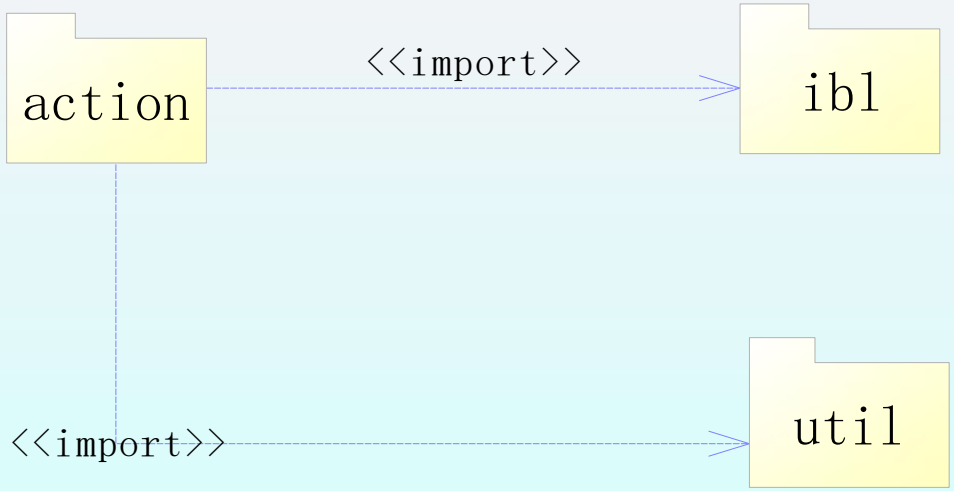


包图

- 概念
  - 包是一种把元素组织到一起的通用机制，包可以嵌套于其他包中。
  - 包图用于描述包与包之间的关系，包的图标是一个带标签的文件夹。

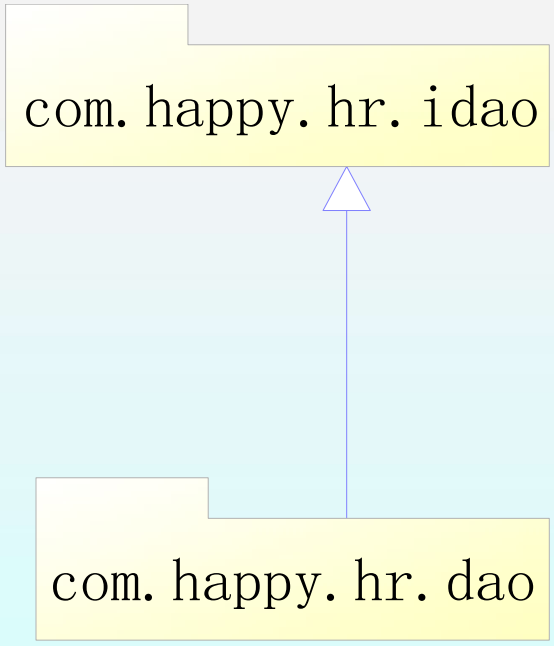
包图

- 包之间的关系
  - 引入关系：一个包中的类可以被另一个指定包（以及嵌套于其中的那些包）中的类引用。
  - 引入关系是依赖关系的一种，需要在依赖线上增加一个<<import>>衍型，包之间一般依赖关系都属于引入关系。



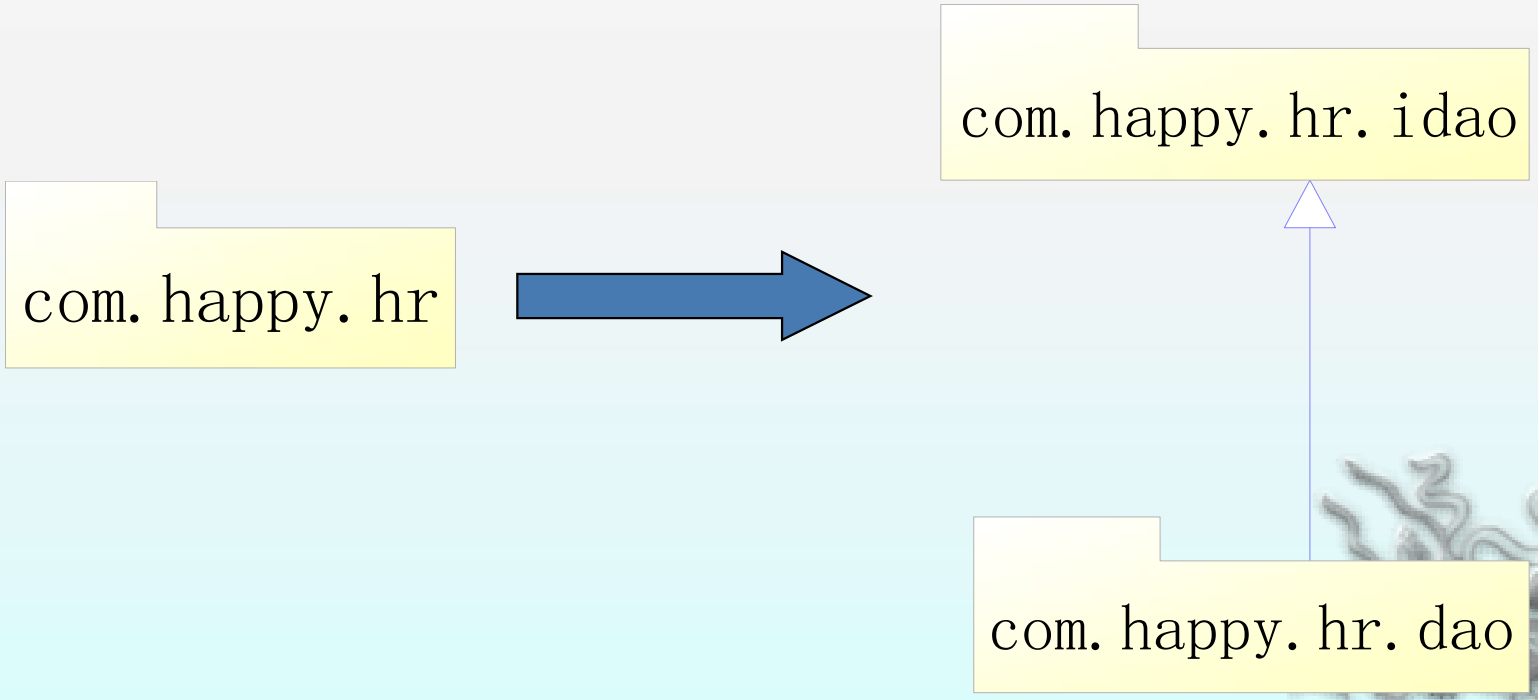
包图

- 包之间的关系
  - 泛化关系：表示一个包继承了另一个包的全部内容，同时又补充自己增加的内容。



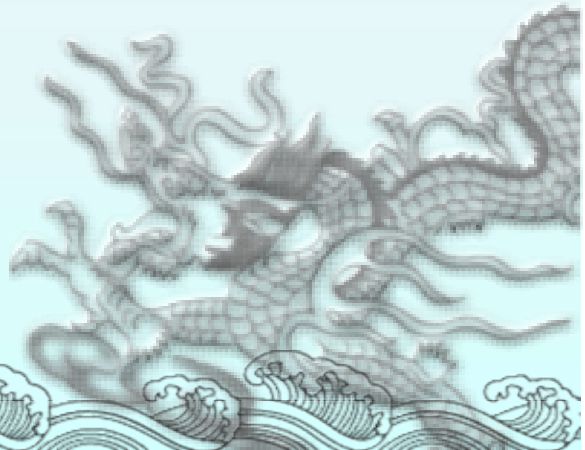
包图

- 包之间的关系
  - 嵌套关系：一个包中可以包含若干个子包，构成了包的嵌套层次结构。



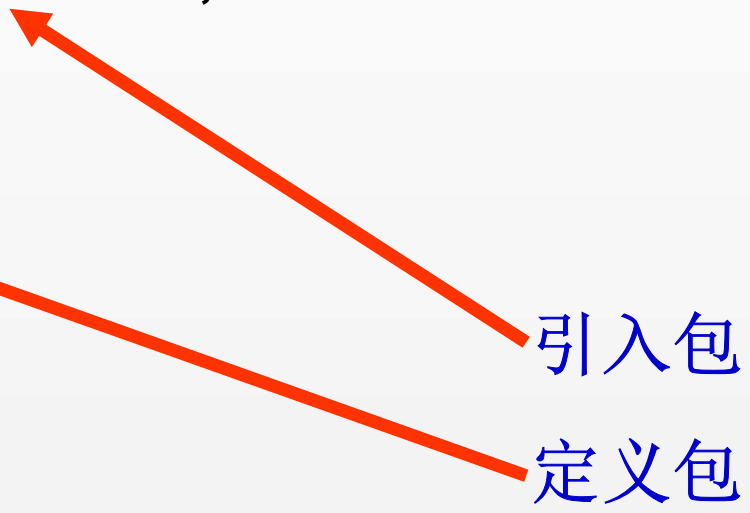
包图

- 包图绘制技巧
  - 两种组包方式:
    - 根据系统分层架构组包（推荐使用）；
    - 根据系统业务功能模块组包。
  - 参照类之间的关系确定包之间的关系；
  - 减少包的嵌套层次，一般不超过三层；
  - 每个包的子包控制在7±2个；
  - 如果几个包有若干相同组成部分，可优先考虑将它们合并；
  - 可通过包图来体现系统的分层架构。



包的基本概念

```
using System;
using System.Collections.Generic;
using System.Text;

namespace scoreA
{
    class A
      {
         ……
      }
    class B
      {
         ……
      }
   class C
      {
         ……
      }
}
```



在**windows**中文件夹有什么作用？

•对文件进行分类管理

•避免了命名冲突

包的作用

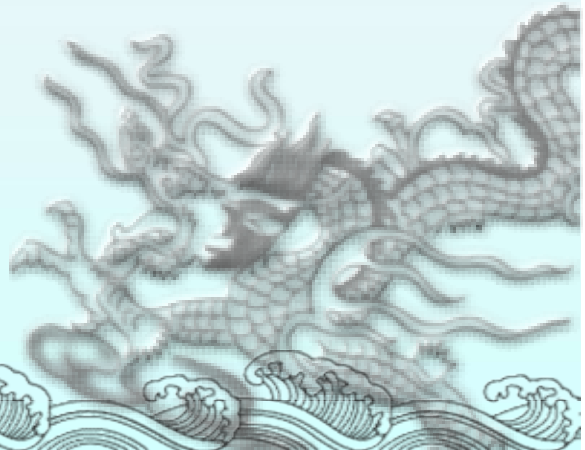
一个程序往往包含了很多个类，那么如何管理这些类就成了一个需要解决的问题（分组机制）

这些类可能由不同的程序员进行建立，当把这些类合并成一个大系统时，往往会产生命名冲突（类名冲突）

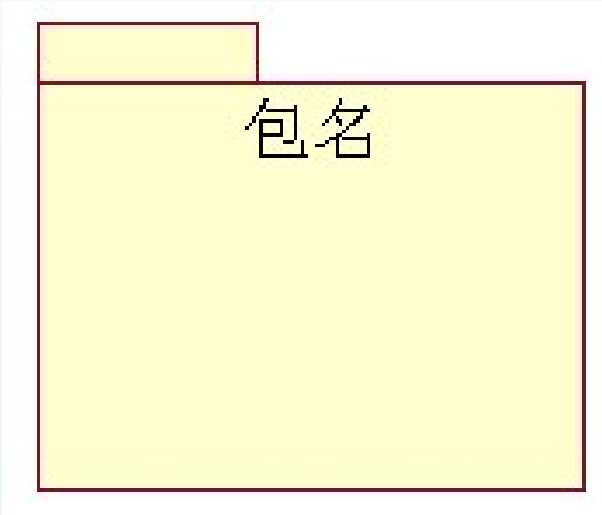
包的两个作用：分组机制，命名空间。

Java: 包**(package)**

C#: 命名空间**(namespace)**

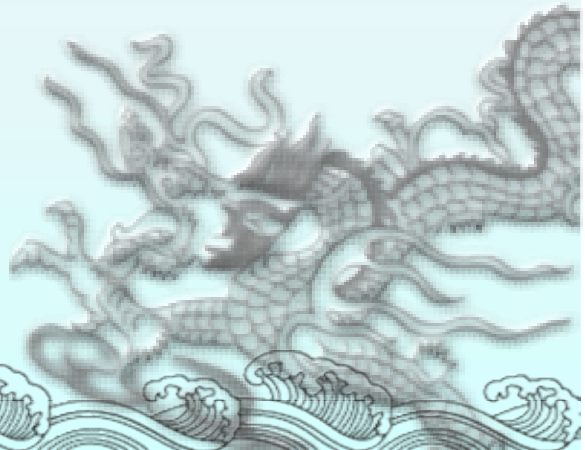


包的符号



包内的元素：

- 类
- 接口
- 构件
- 结点
- 协作
- 用例
- 图



包的特征

包与构成元素之间的关系：

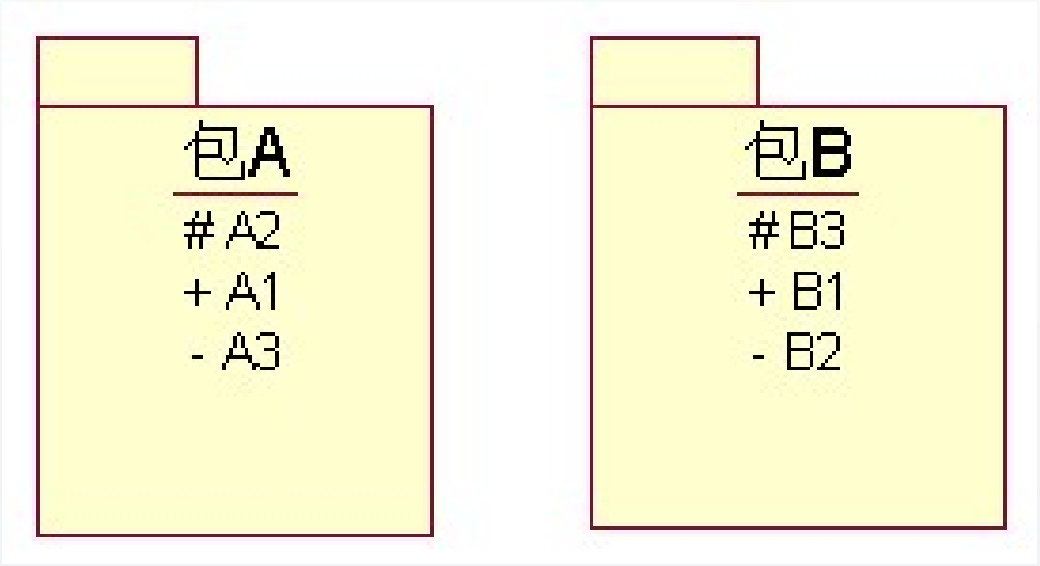
- 类似组成关系，即一个包被撤销，那么其封装的元素也随之被撤销；

包所构成的名字空间及其要求：

- 一个包构成了一个名字空间，即在一个包的语境中同一种元素的名字必须唯一；
- 一个元素只能被一个包所拥有；
- 建模人员对包中的所有元素均唯一命名；

建模人员可显式地以文字方式或图形方式显示包的内容。

包内元素的可见性



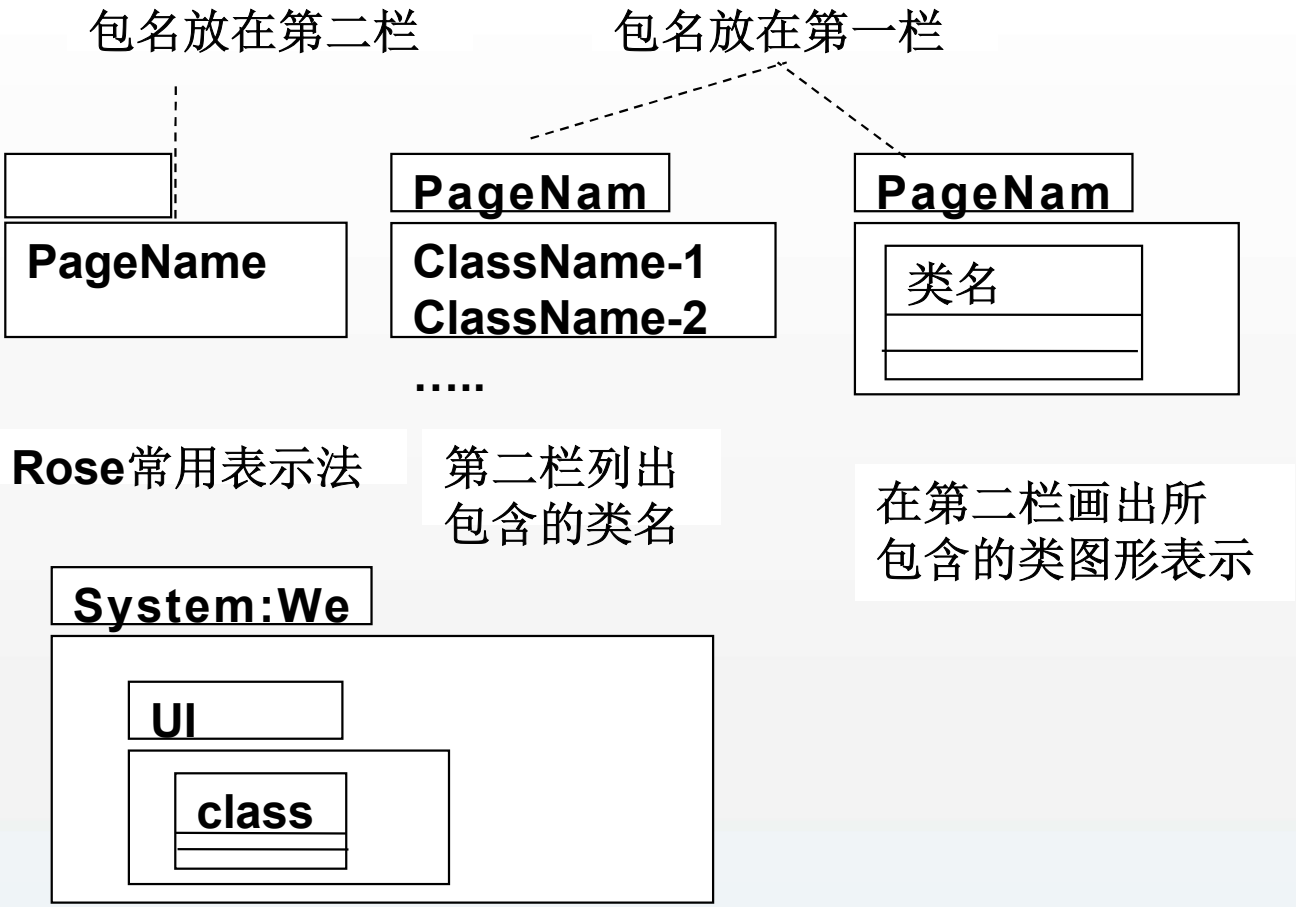
Public(+): 可在其他任何包中使用

Private(-): 只能在该包中使用

Protected(#): 可以在该包和该包的子包中使用

包的表示

- UML中，用文件夹符号来表示一个包。包由两个矩形表示，它包含2栏。下面是最常见的几种包的表示法，如图所示。


包名放在第二栏
包名放在第一栏
PageName
PageNam
ClassName-1
ClassName-2
.....
PageNam
类名
Rose常用表示法
第二栏列出
包含的类名
在第二栏画出所
包含的类图形表示
System:We
UI
class
嵌套包

包的表示法

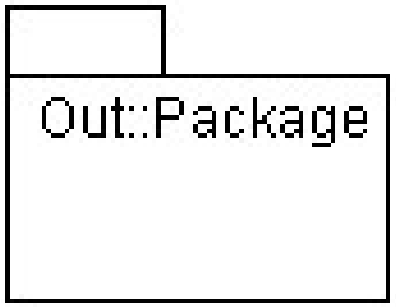
包的命名有**2**种方式: 简单包名和路径包名.

简单包名: **Vision**

路径包名: **Sensors::Vision**

路径包名中位于前面的是外围包, 后面的是内部包.
注意包的嵌套层数不应过多.

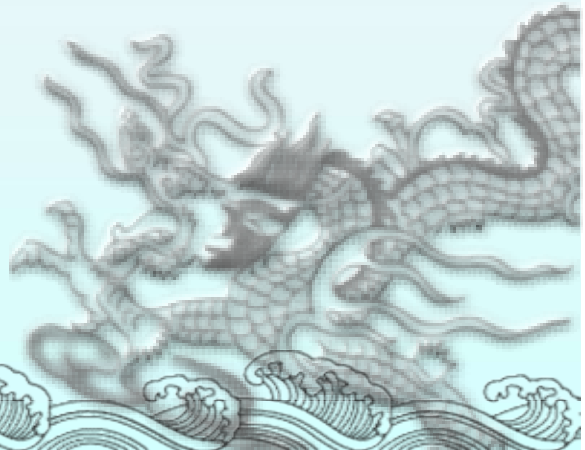
简单名



路径名

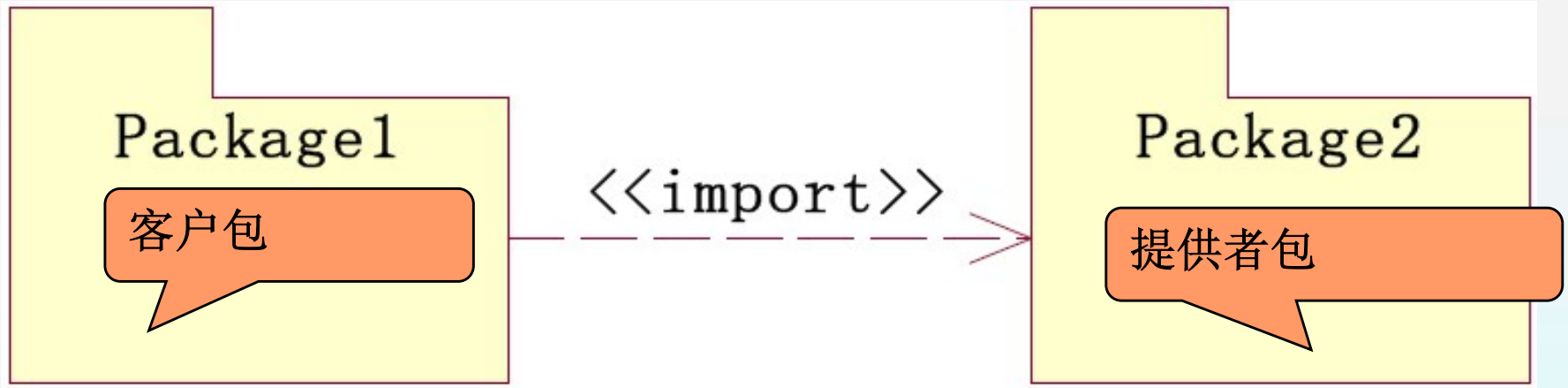
包间的依赖关系

- 为了使一个包能引用另一个包中的元素，建模人员必须使用版型为<<access>>或<<import>>的依赖。



引入依赖关系import是默认的关系

- 说明提供者包的命名空间将被添加到客户包的命名空间中，客户包中的元素也能够访问提供者包的所有公共元素
- 客户包可以存取提供者包中内容，并且引用提供者包时无需包名，直接用元素名称就可。

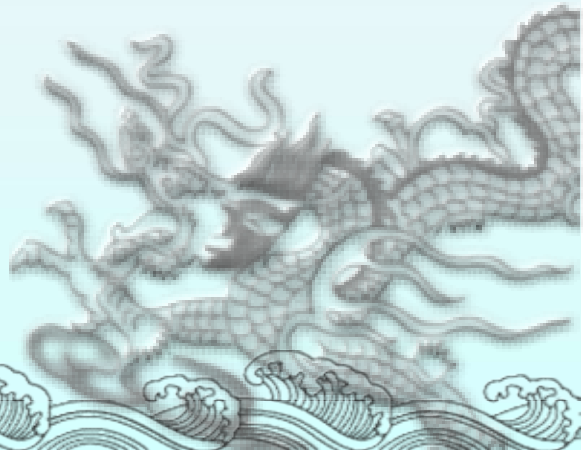


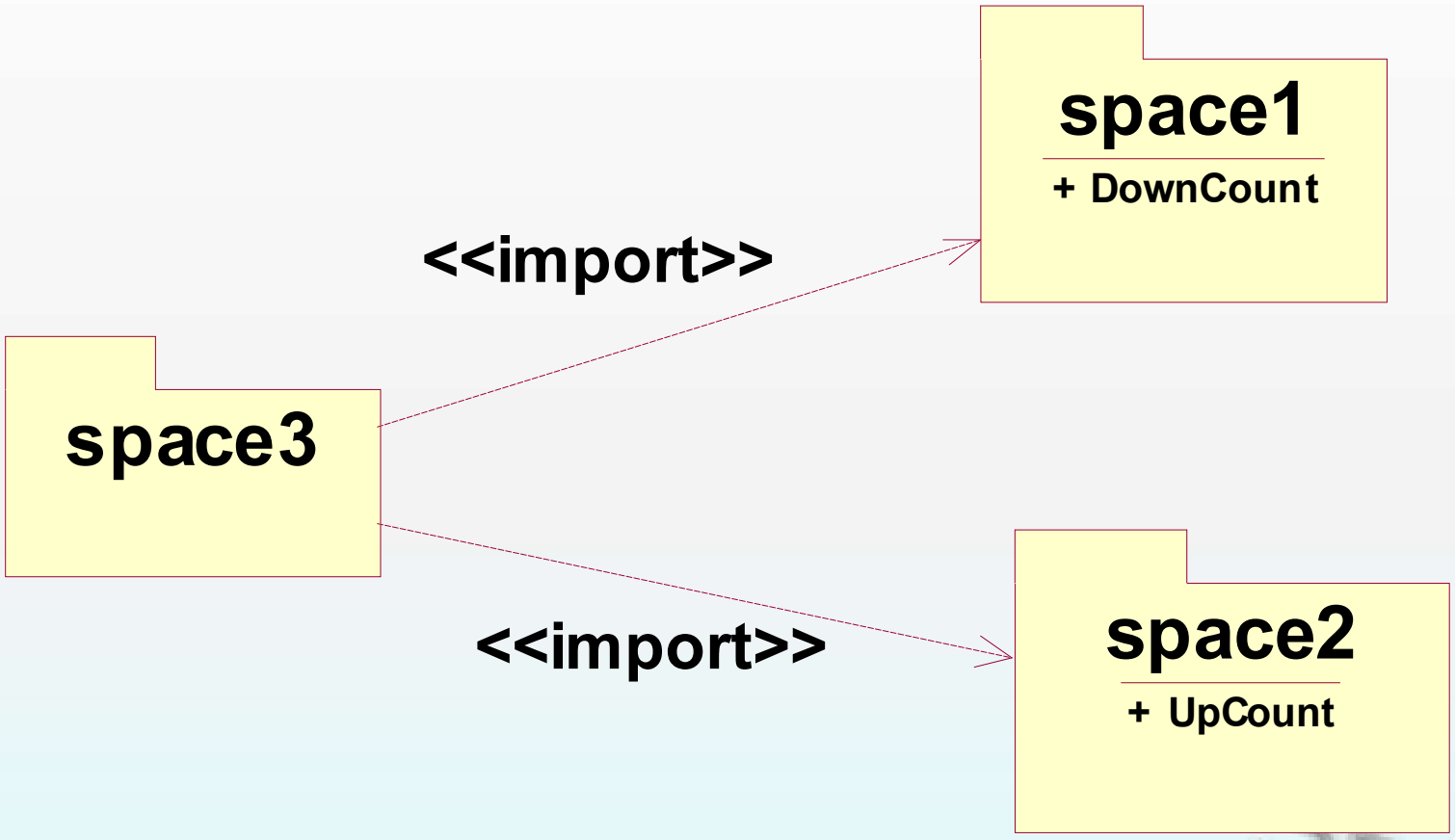
```
namespace space1   //第一个命名空间
{
   class DownCount  //倒数计数器
   {
      int v;
      public DownCount(int n)
      {
        v = n;
      }
      public void reset(int n) //计数器置初始值
      {
        v = n;
      }
      public int count()       //倒数计数方法
      {
         if (v > 0) return v--;
         else return 0;
      }
   }
   //在此可建立其他的类
}
```

```
namespace space2   //第二个命名空间
{
   class UpCount  //正数计数器
   {
      private int v;
      public int goal;
      public UpCount(int n)  //从0至n输出
      {
         goal= n;
         v = 0;
      }
      public void reset(int n)
      {
         goal = n;
         v = 0;
      }
      public int count()
      {
         if (v < goal)
            return v++;
         else
            return goal;
      }
   }
   //在此可建立其他的类
}
```

```
//using space1;
//using space2;

namespace space3   //第三个命名空间
{
……
space1.DownCount dc = new space1.DownCount(10);
space2.UpCount uc = new space2.UpCount(10);
……
}
```

```
using space1;
using space2;

namespace space3   //第三个命名空间
{
……

DownCount dc = new DownCount(10);
UpCount uc = new UpCount(10);
……
}
```

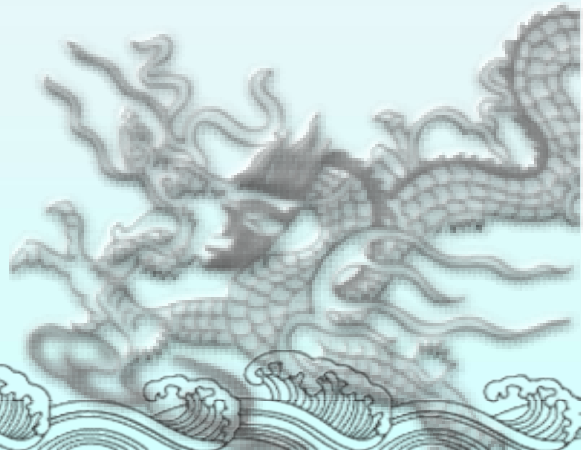



space1
+ DownCount
<<import>>
space3
<<import>>
space2
+ UpCount

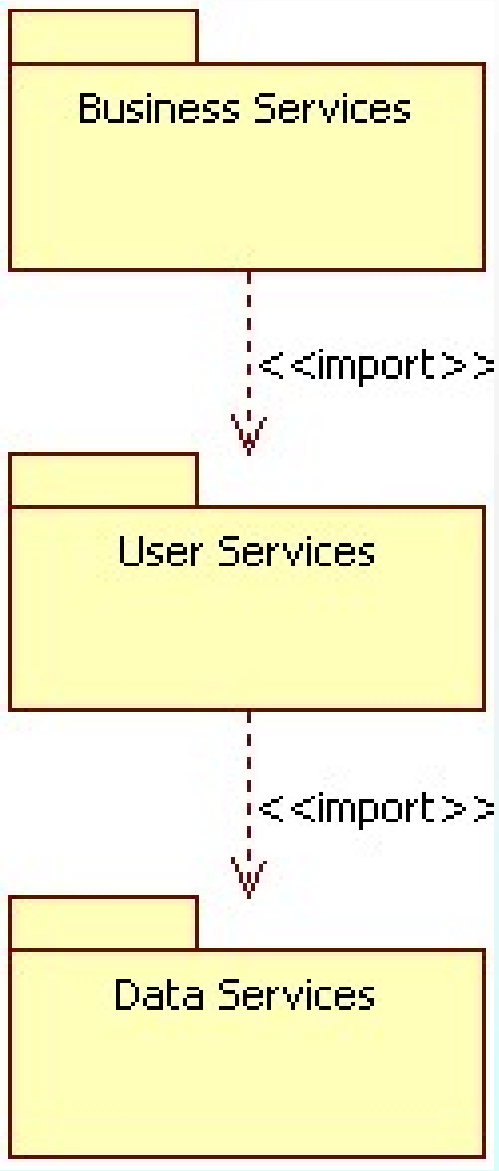
<<access>>和<<import>>的区别

- <<access>>依赖关系代表客户包访问提供者包，不增加客户包的内容;

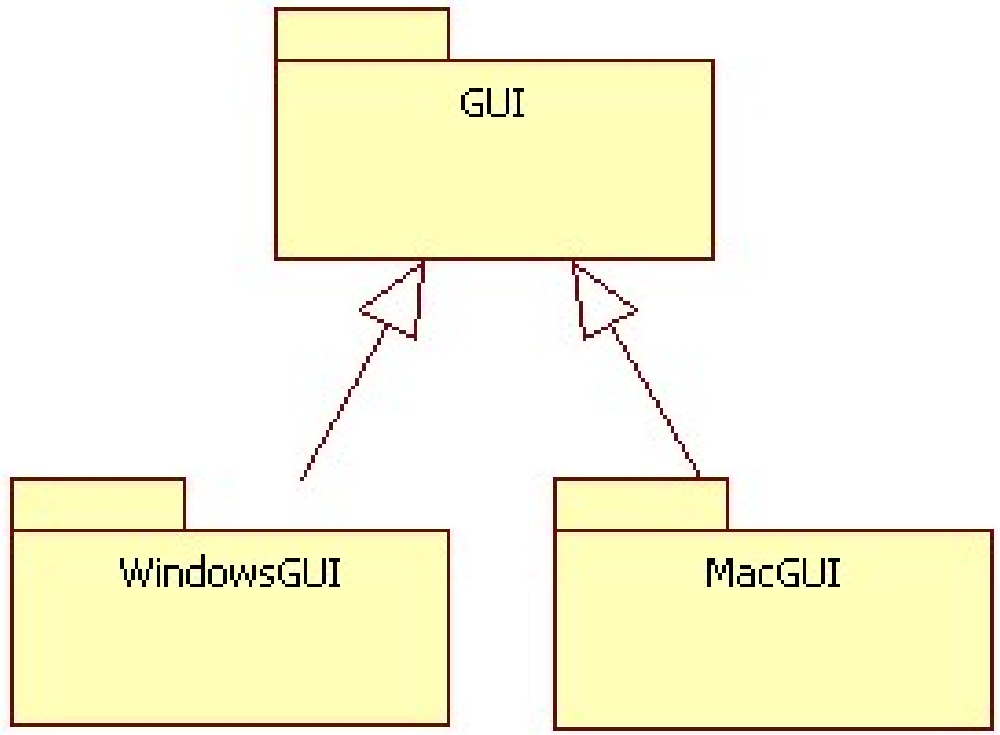
- <<import>>依赖关系代表提供者包的内容增加到客户包中。



包之间可以存在依赖关系. 这种依赖关系没有传递性.



包之间可以存在泛化关系.



设计包的原则

设计包时应遵循以下原则:

- 重用等价原则(Reuse Equivalency Principle, REP)
  - 把类放入包中时, 应考虑把包作为可重用的单元.
- 共同闭包原则(Common Closure Principle, CCP)
  - 把需要同时改变的类放在同一个包中.
- 共同重用原则(Common Reuse Principle, CRP)
  - 不会一起使用的类不要放在同一个包中.
- 非循环依赖原则(Acyclic Dependencies Principle, ADP)
  - 包之间的依赖关系不要形成循环

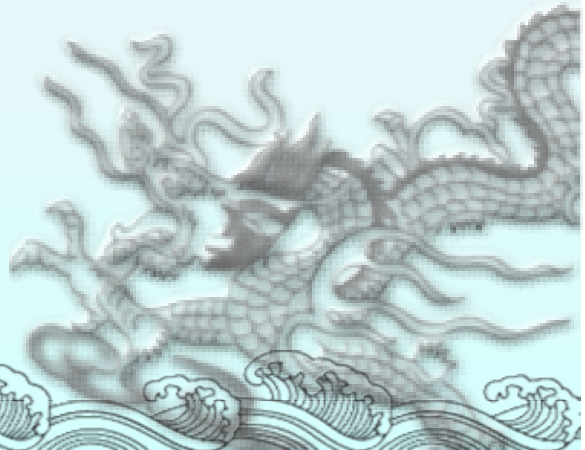
重用等价原则、共同闭包原则、共同重用原则这3个原则事实上是相互排斥的，不可能同时被满足。它们是从不同使用者的角度提出的，重用等价原则和共同重用原则是从重用人员的角度考虑的，而共同闭包原则是从维护人员的角度考虑的。共同闭包原则希望包越大越好，而共同重用原则却要求包越小越好。

包的应用

基本功能：对建模元素进行分组.

Rose中，包可以提供其它功能：

- 数据建模中,包表示模式和域
- Web建模中,包可以表示一个虚拟目录
- 作为控制单元方便团队开发和配置管理



包图建模

包图主要用于两种不同层次的用途：一是对成组元素建模；二是对体系结构建模。

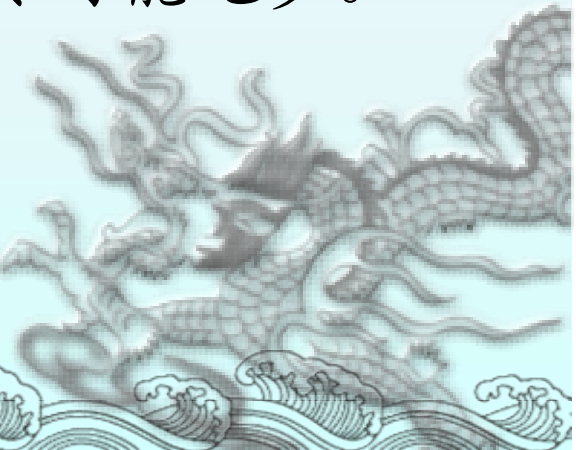
1 对成组元素建模

对成组元素进行建模可以说是包图最常见的用途，它将建模元素组织成组，然后对组进行命名，在对成组元素建模时，应遵循以下几个策略：

每个包都应该是由在概念上、语义上相互接近的元素组成。

对于每个包，找出应标记为公共的元素，但应尽可能地少。

采用泛化来对特殊包进行建模。



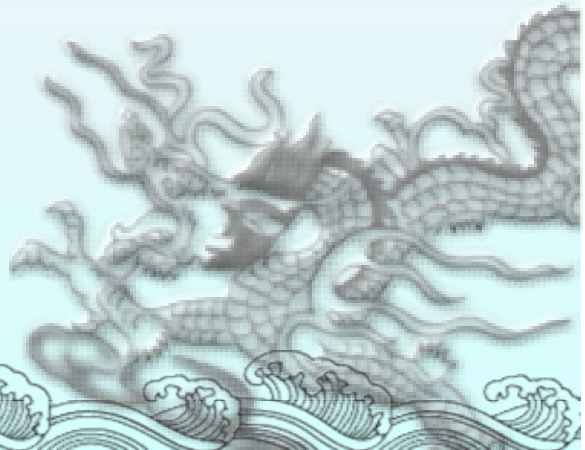
包图建模

在构建包模型时，注意，在包中只标明对每个包起核心作用的元素；另外也可以标识每个包的文档标记值，以使其更加清晰。

2 对体系结构建模

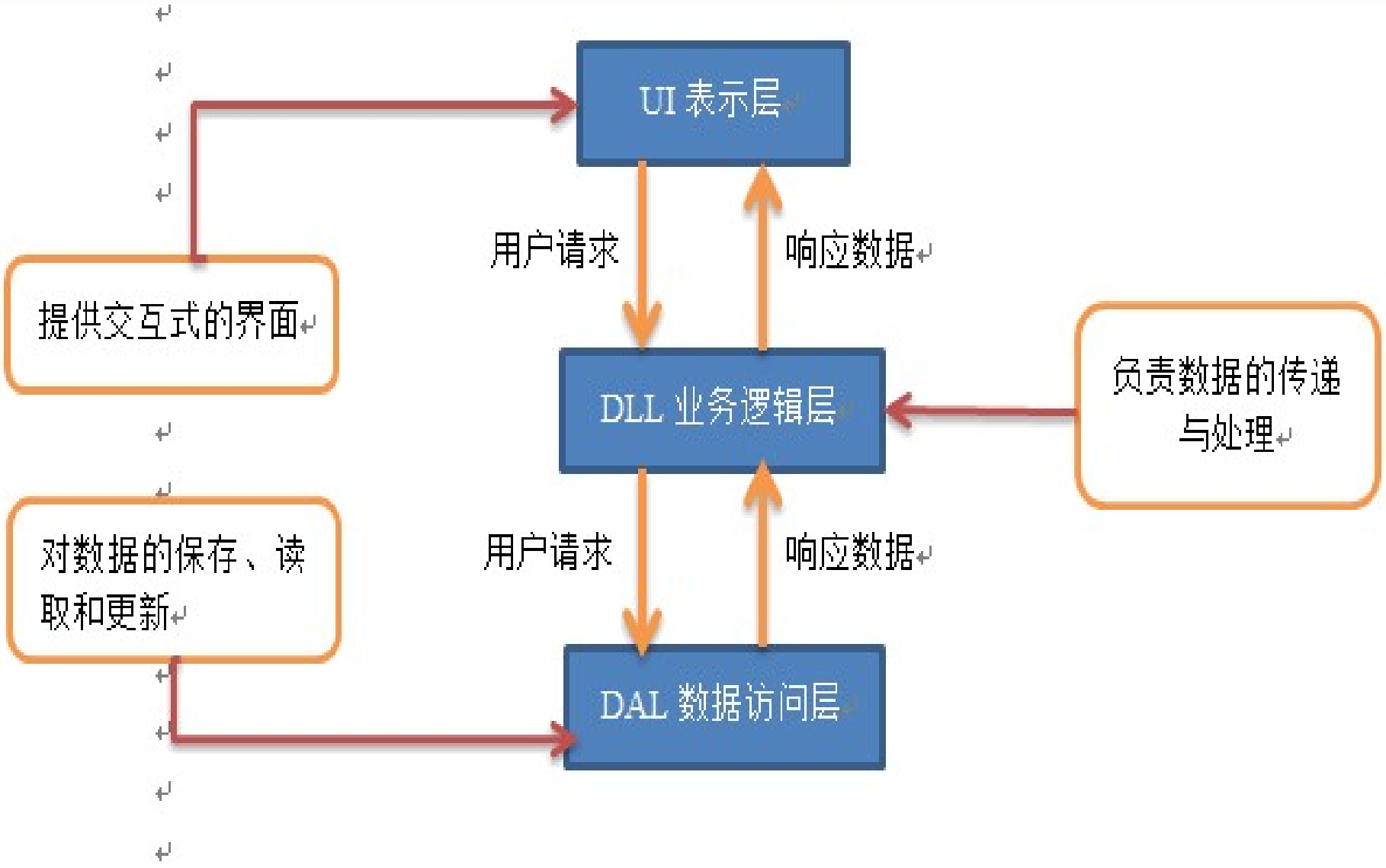
体系结构是一个软件系统的核心逻辑结构，在应用软件中，分层和MVC是最常见的两种结构。

在分层的体系结构中，最常见的划分是表示层、逻辑层、数据层。如果采用分层体系结构，我们就把每一层用一个包来表示。



什么是三层架构

“三层架构”中的“三层”是指：表示层（User Interface Layer-UI）、业务逻辑层(BussinessLogic Layer-BLL)、数据访问层（Data Access Layer-DAL)。三层架构的结构可以用如下图表示。

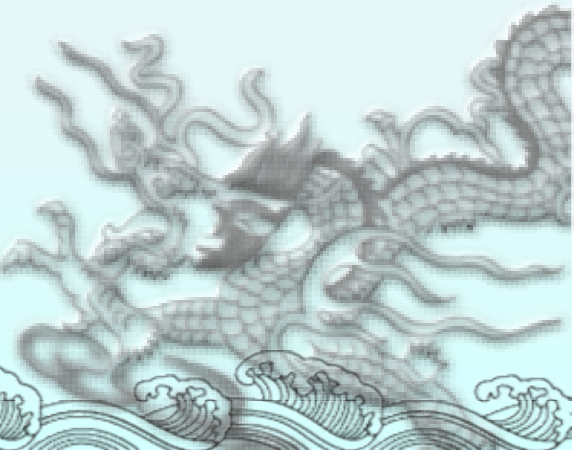

UI 表示层
用户请求
响应数据
提供交互式的界面
DLL 业务逻辑层
负责数据的传递与处理
对数据的保存、读取和更新
用户请求
响应数据
DAL 数据访问层

三层之间的结构关系（图）

表示层：位于系统的最外层（最上层），离用户最近。用于显示数据和接收用户输入的数据，只提供软件系统与用户交互的界面。

业务逻辑层：位于表示层和数据访问层之间，专门负责处理用户输入的信息，或者是将这些信息发送给数据访问层进行保存，或者是通过数据访问层从数据库读出这些数据。该层可以包括一些对“商业逻辑”描述的代码在里面。业务逻辑层是表示层和数据访问层之间的桥梁，负责数据处理和传递。

数据访问层：仅实现对数据的保存和读取操作。数据访问包括访问数据库系统、二进制文件、文本文档或是XML文档。数据访问层只负责对数据的访问存取工作。



优缺点

优点

1、开发人员可以只关注整个结构中的其中某一层；

2、可以很容易的用新的实现来替换原有层次的实现；

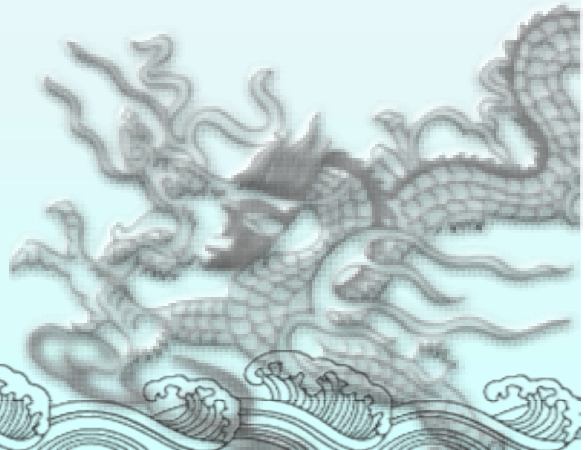
3、可以降低层与层之间的依赖；

4、有利于标准化；

5、利于各层逻辑的复用；

6、结构更加明确；

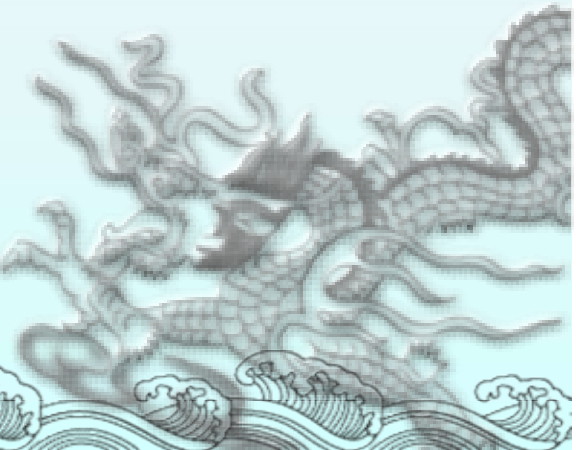
7、利于维护。



缺点

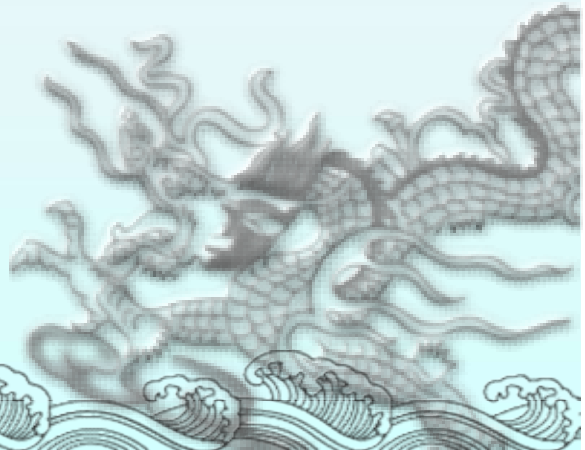
1、有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

2、增加了开发成本。



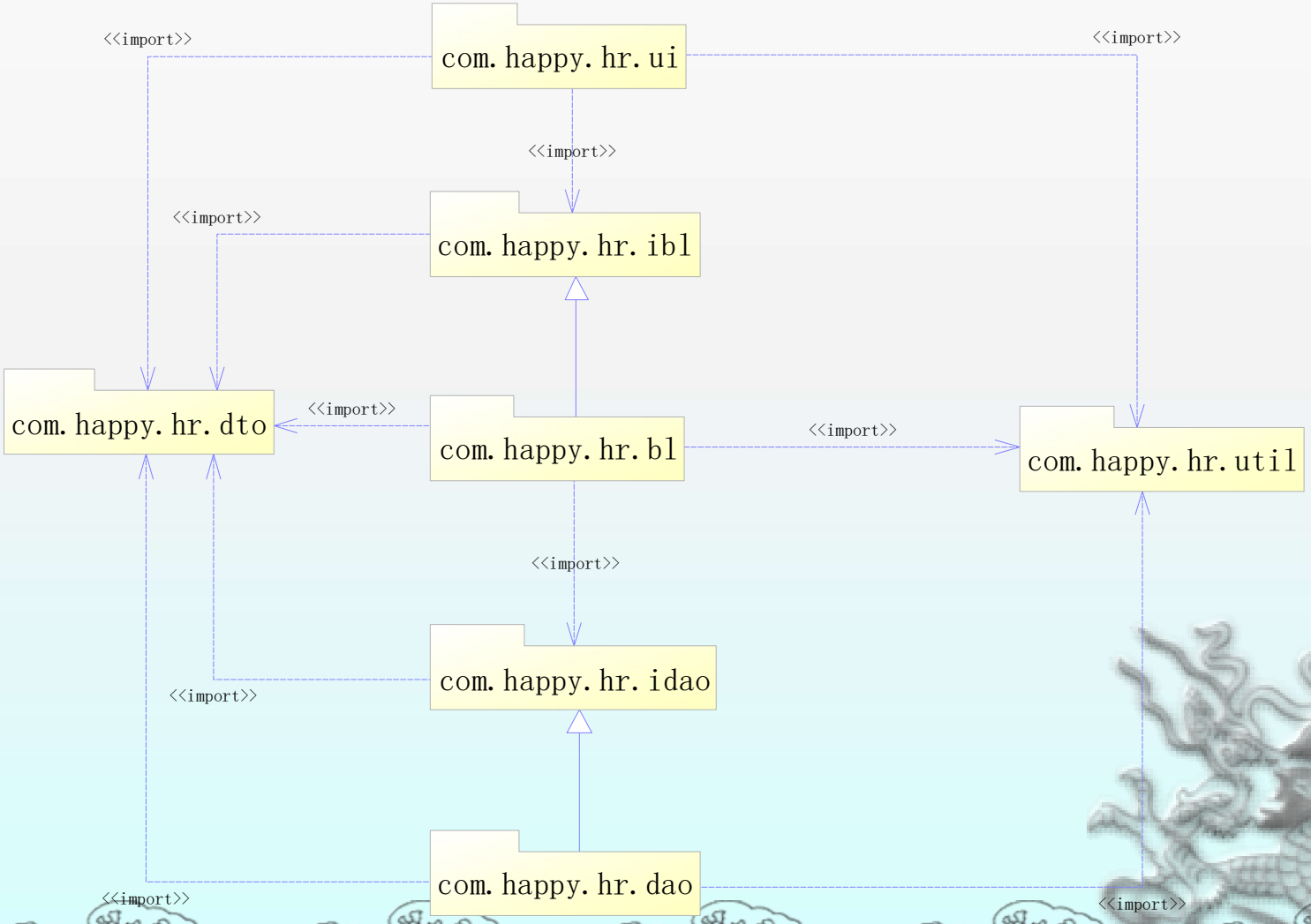
内容提纲

- 组件图
  - 相关概念
  - 实例讲解
- 部署图
  - 相关概念
  - 实例讲解
- 包图
  - 相关概念
  - 实例讲解



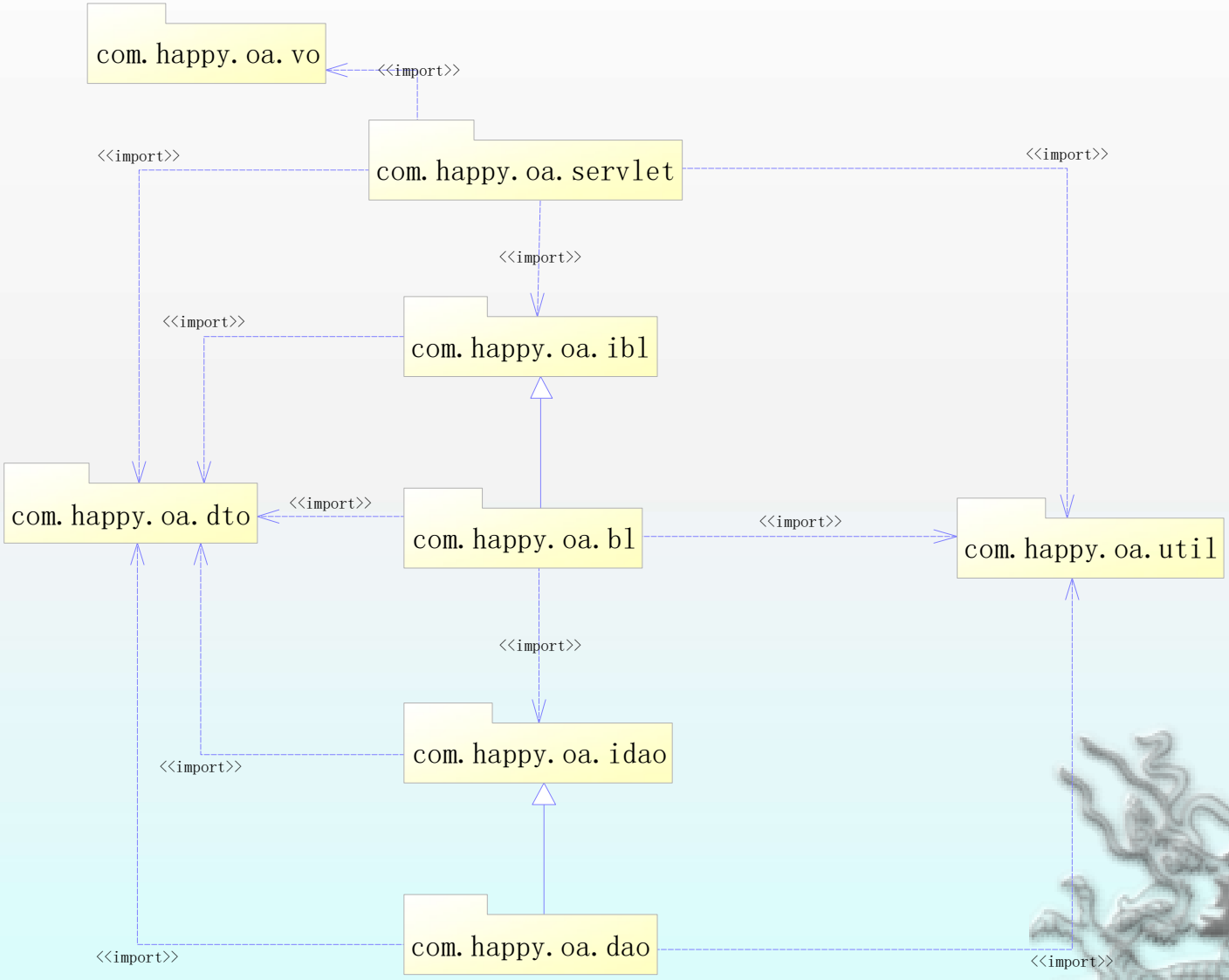
包图

- 包图实例（基于C/S的人力资源管理系统）

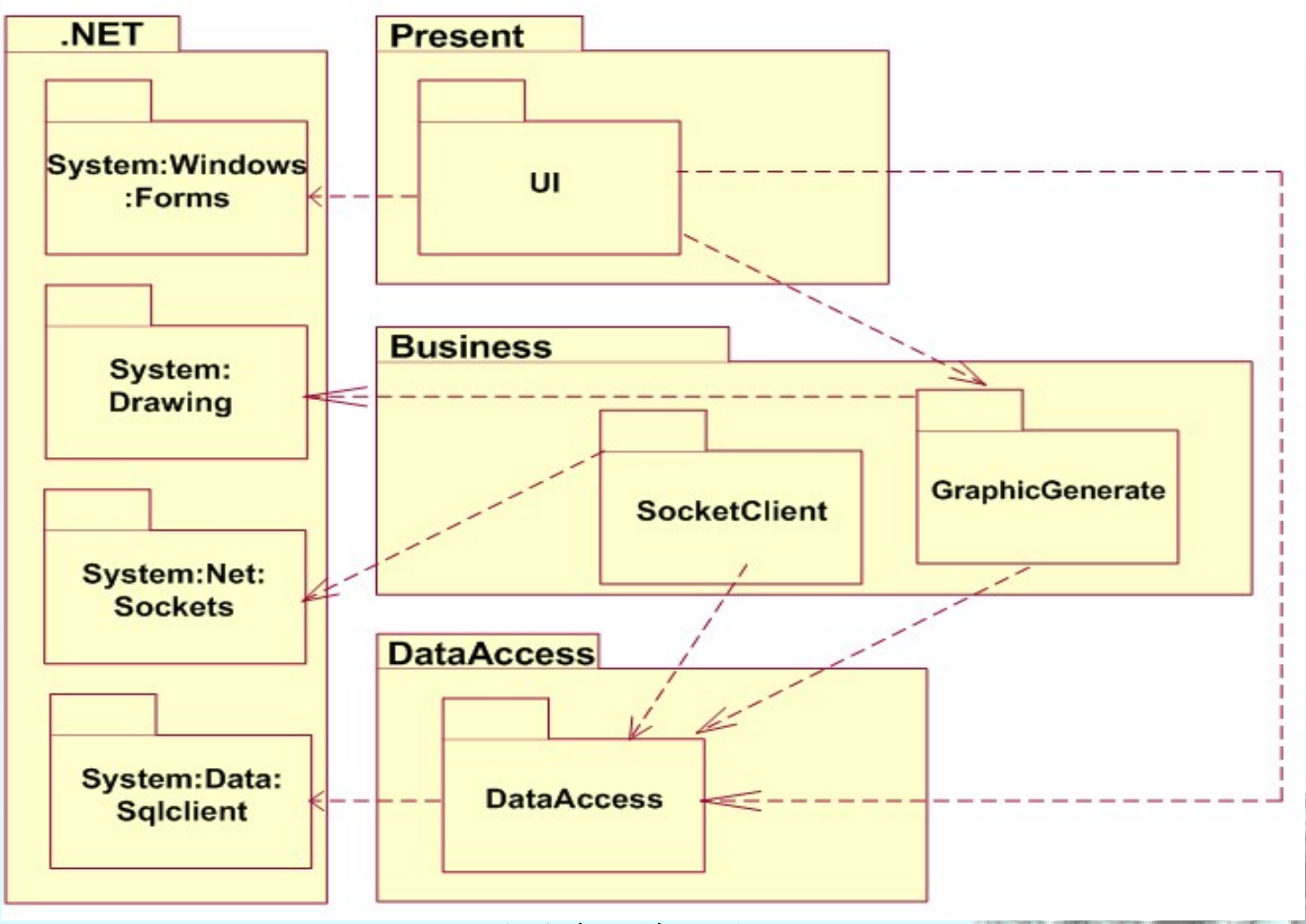


包图

- 包图实例（基于B/S的OA系统）



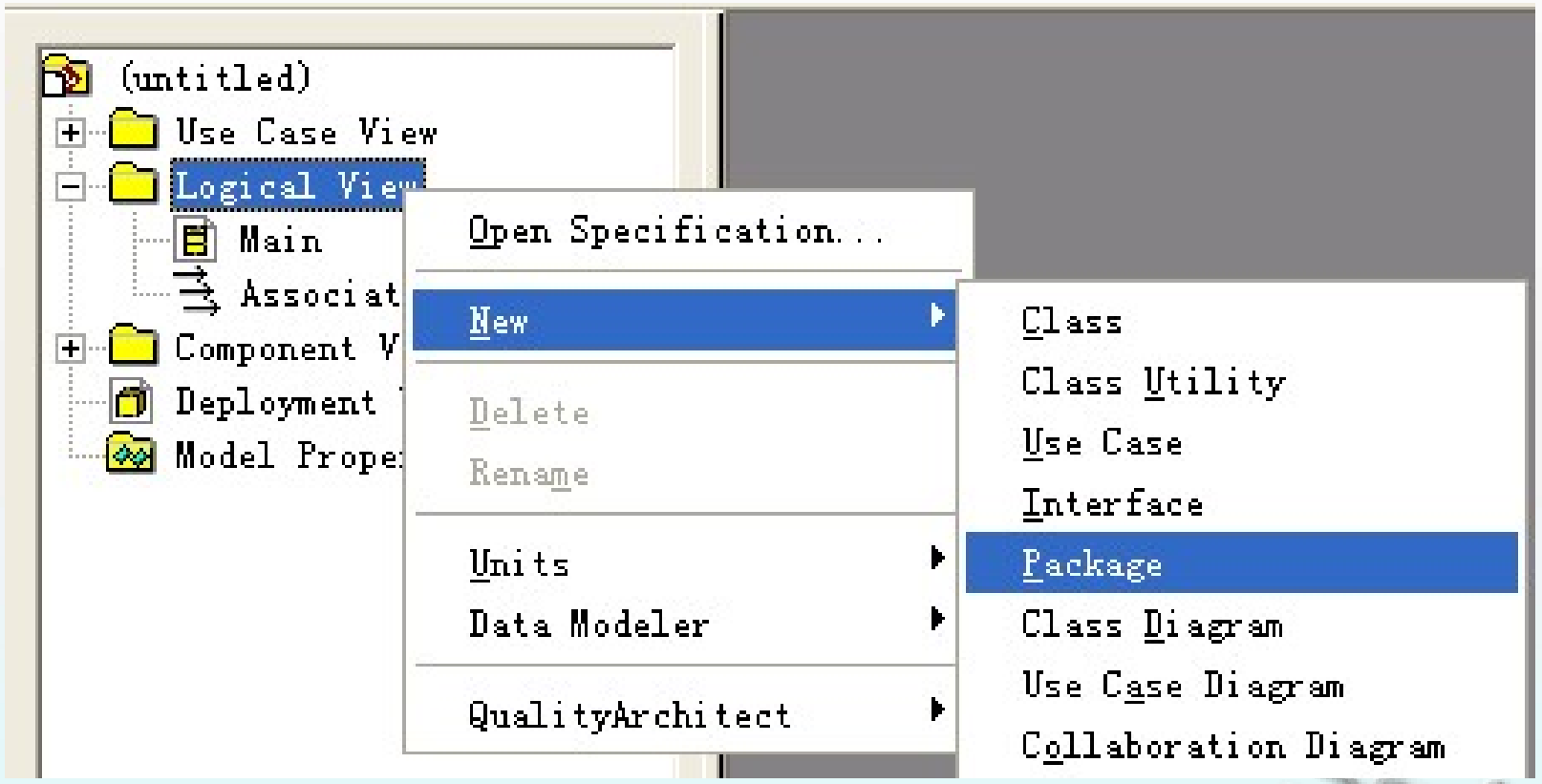
包图建模

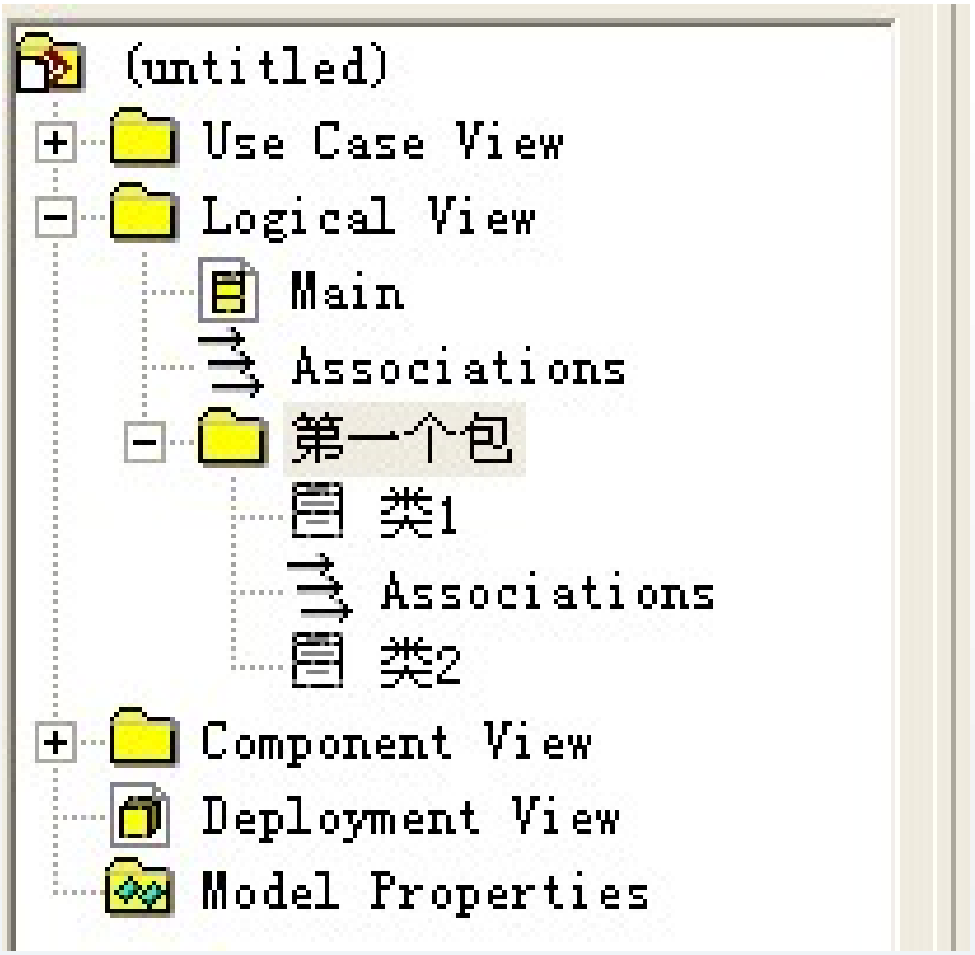

.NET
System:Windows
:Forms
System:
Drawing
System:Net:
Sockets
System:Data:
Sqlclient
Present
UI
Business
SocketClient
GraphicGenerate
DataAccess
DataAccess

用包分层

包图的Rose建模

1、新建包





2、向包中添加类

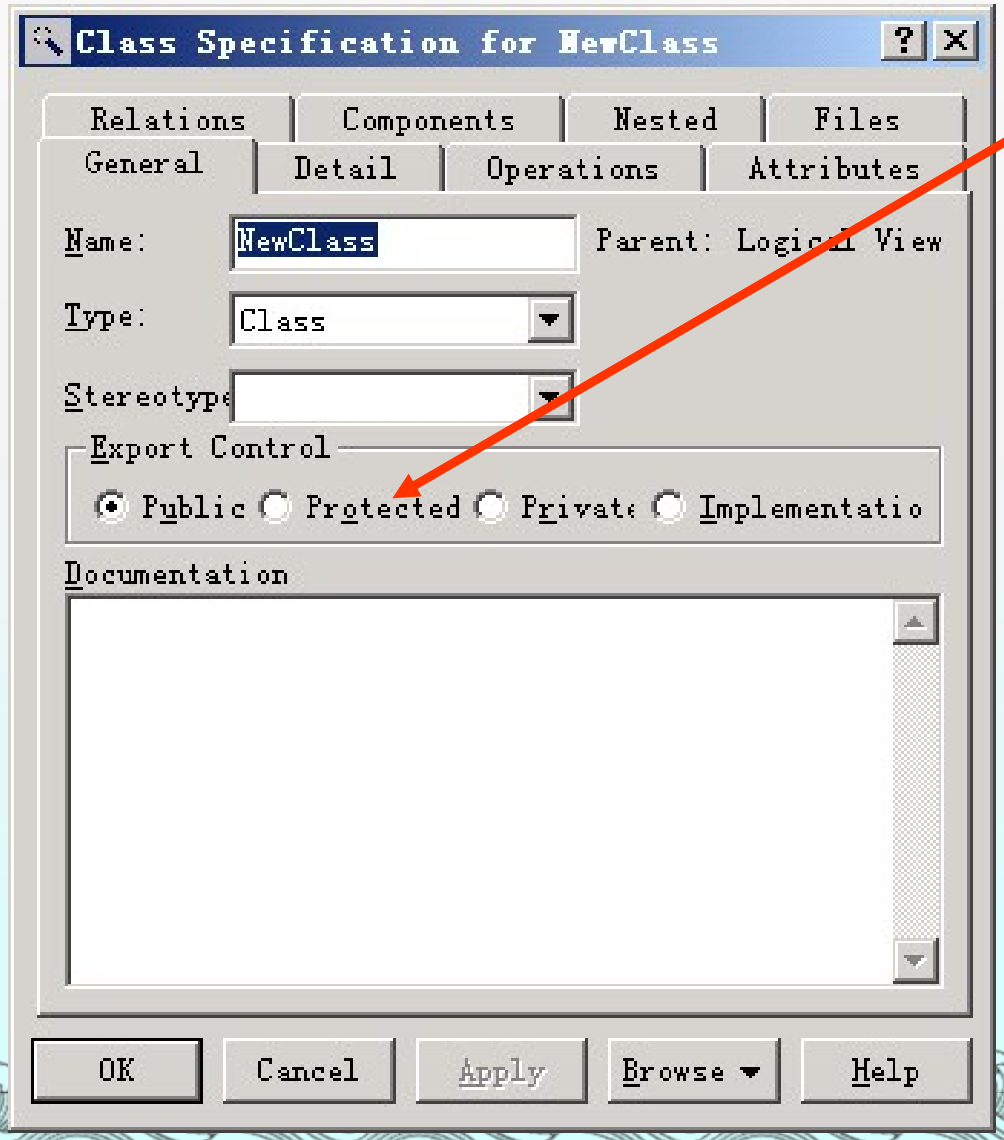
方法**1**：在浏览器中将已经存在的类拖到包中

方法**2**：在浏览器中新建类

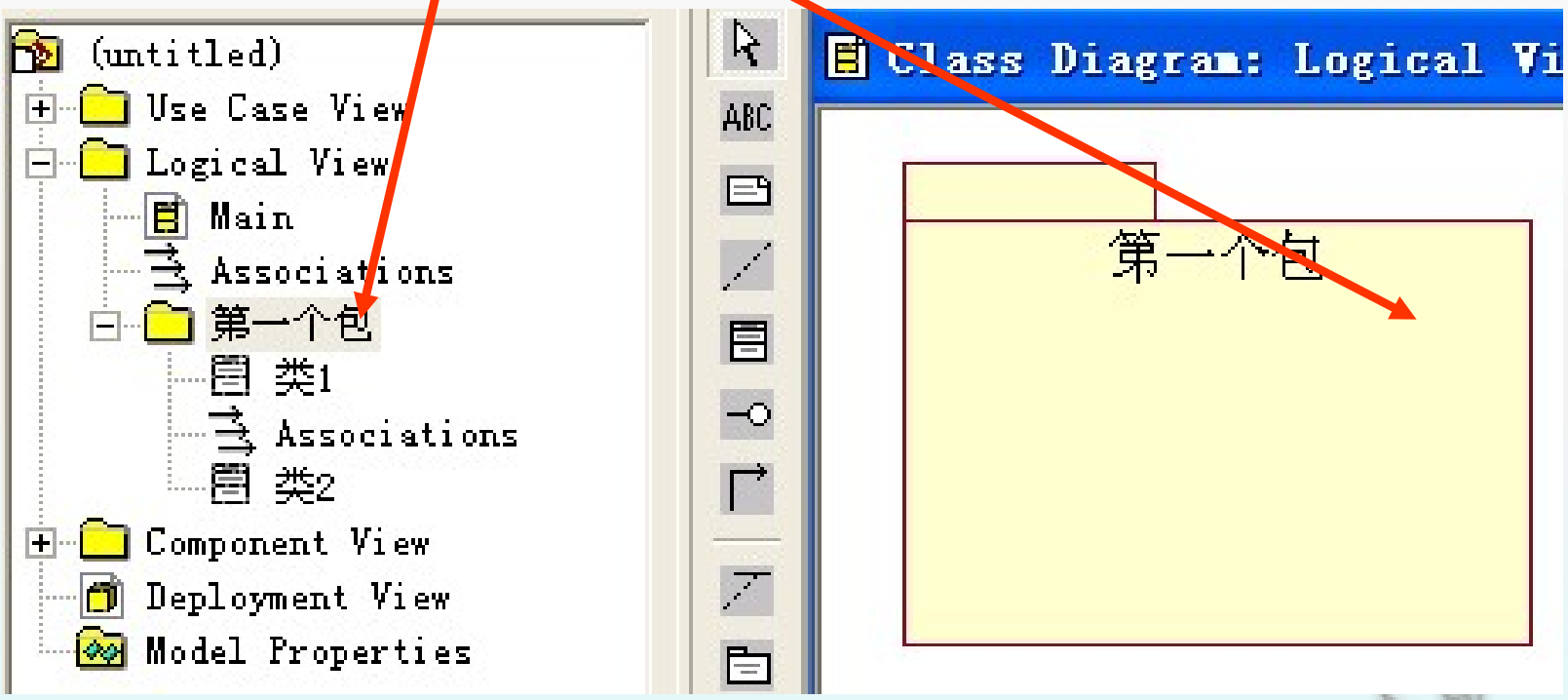
方法：右击包名**->new->class**

3、设置类在包中的可见性

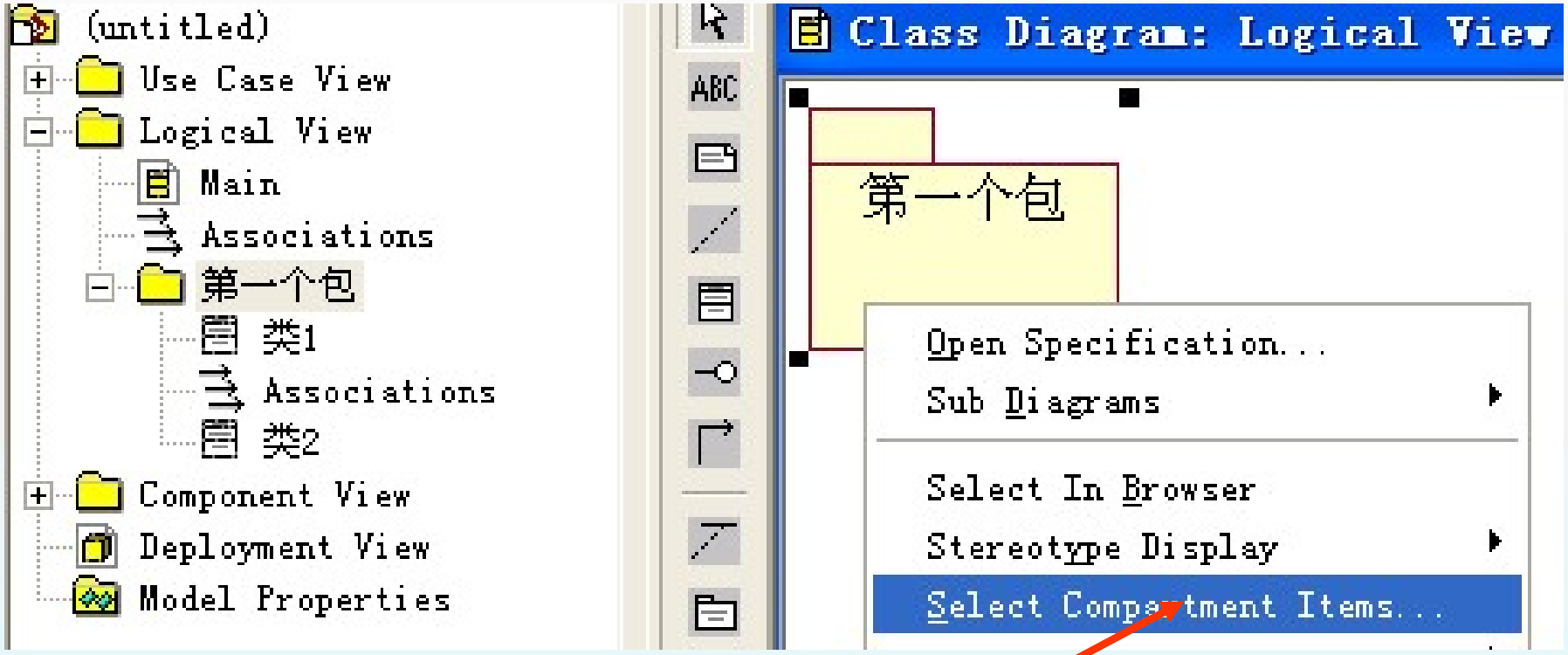
方法：在浏览器中双击类，弹出下窗，选择



4、将包拖到绘图区

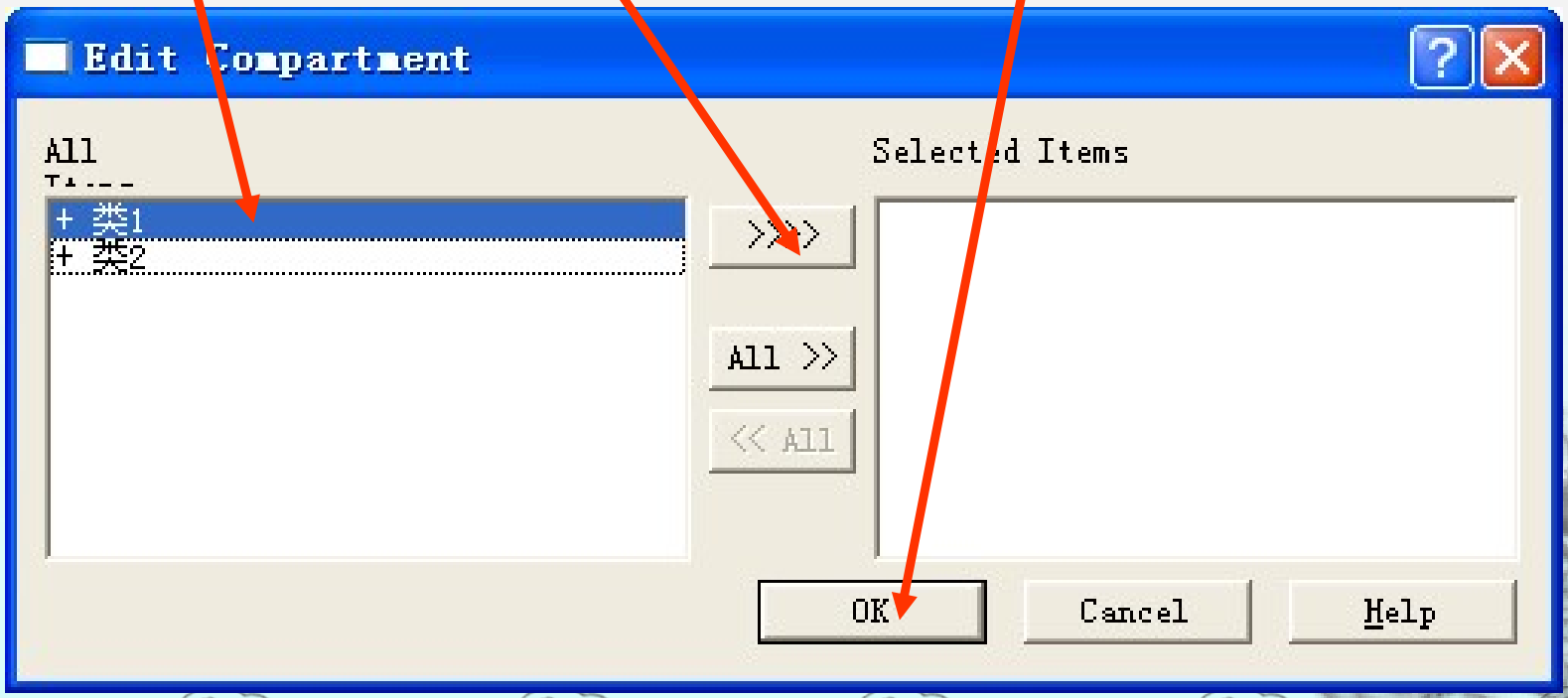

(untitled)
Use Case View
Logical View
Main
Associations
第一个包
类1
Associations
类2
Component View
Deployment View
Model Properties
ABC
Class Diagram: Logical Vi
第一个包

5、 显示包中的元素



在绘制区中右击包，选择此项，弹出下窗

选择元素，点击添加按纽，单击**OK**返回



使用Rational Rose绘制包图的步骤

1. 创建包
2. 修改包的属性
3. 增加包的信息
4. 添加包之间的关系