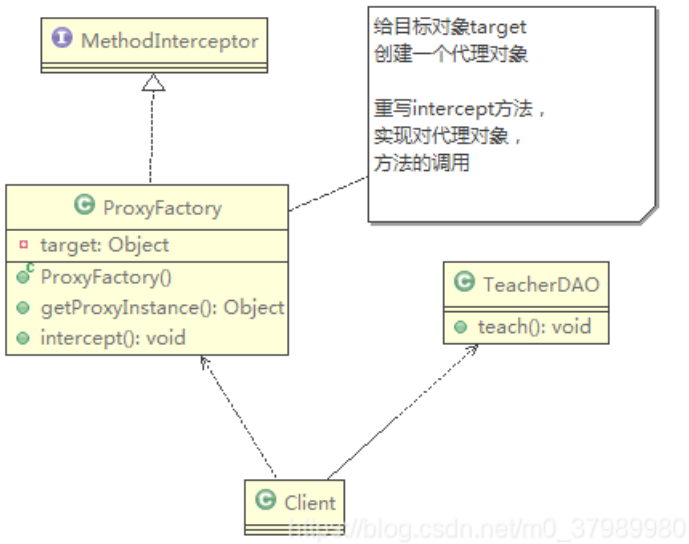
目录

> **MySQL笔记: B站宋红康最新教程 (持续更新中)**

UML 类图

> 浅谈UML中常用的几种图

- UML——Unified modeling language UML(统一建模语言)，是一种用于软件系统分析和设计的语言工具，它用于帮助软件开发人员进行思考和记录思路的结果
- UML本身是一套符号的规定，就像数学符号和化学符号一样，这些符号用于描述软件模型中的各个元素和他们之间的关系，比如**类、接口、实现、泛化、依赖、组合、聚合**等，如下图
- 使用UML来建模，常用的工具有 RationalRose , 也可以使用一些插件来建模



UML 图

UML 图

画UML图与写文章差不多，都是把自己的思想描述给别人看，关键在于思路和条理，UML图分类：

- 用例图(use case)
- 静态结构图： 类图、对象图、包图、组件图、部署图
- 动态行为图：交互图（时序图与协作图）、状态图、活动图
- **类图是描述类与类之间的关系的，是UML图中最核心的**

UML 类图

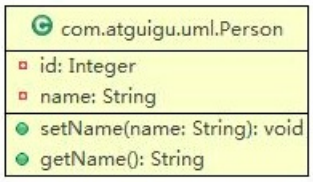
- 用于描述系统中的类(对象)本身的组成和类(对象)之间的各种静态关系。
- 类之间的关系： **依赖、泛化（继承）、实现、关联、聚合与组合**

类图简单举例

1. 代码

```
public class Person { // 代码形式->类图
    private Integer id;
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

2. 类图



类图关系

跳转到目录

概述

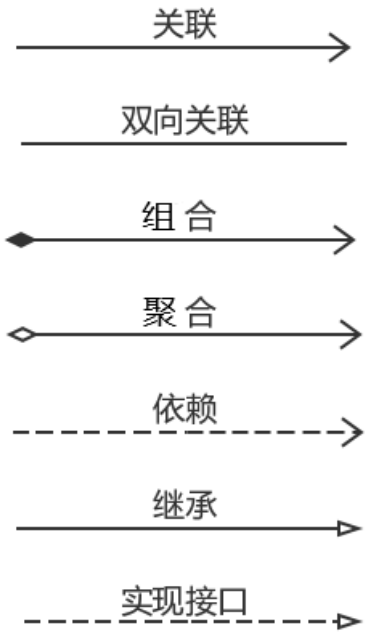
> - 设计一个类中的信息和行为要 高内聚
> - 设计多个类, 类之间要 低耦合

面向对象是符合人们对现实世界的思维模式，利用面向对象设计，特别是采用各种设计模式来解决问题时，会设计多个类，然后创建多个对象，一个设计良好的类，应该是**兼顾信息和行为**并且`高内聚`。而不同的类之间，应该做到`松耦合`。

当面对应用系统或者需要解决的问题经常是复杂的、高度抽象的，我们`创建的多个对象往往是有联系的`，通常对象之间的关系可以分为以下几类：

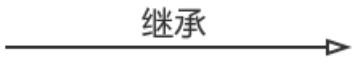
- **泛化关系**
- **实现关系**
- **依赖关系**
- **关联关系**
- **聚合关系**
- **组合关系**

对于`继承(泛化)、实现(实现)`这两种关系比较简单，它们体现的是一种类与类、或者类与接口之间的`纵向关系`。 其他的四种关系则体现的是类与类、或者类与接口之间的`引用/横向关系`。这四种关系所表现的强弱程度来看，从强到弱依次为：`组合>聚合>关联>依赖`。



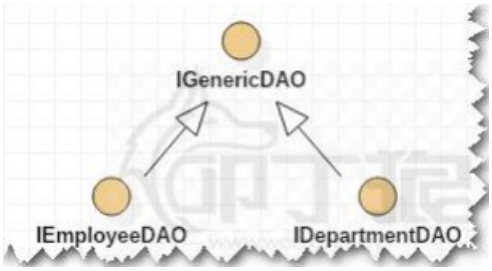
1、泛化关系（generalization）

跳转到目录

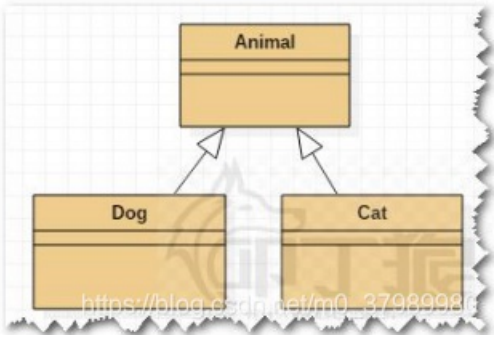


- 泛化关系其实就是`继承关系`：指的是一个类（称为子类、子接口）继承(`extends`)另外的一个类（称为父类、父接口）的功能，并可以增加自己额外的一些功能，**继承是类与类或者接口与接口之间最常见的关系；**
- 在Java中此类关系通过关键字`extends`明确标识。
- 在UML类图中，继承通常使用 **空心三角+实线** 表示

接口之间的泛化关系: 接口用圆心表示



类之间的泛化关系:



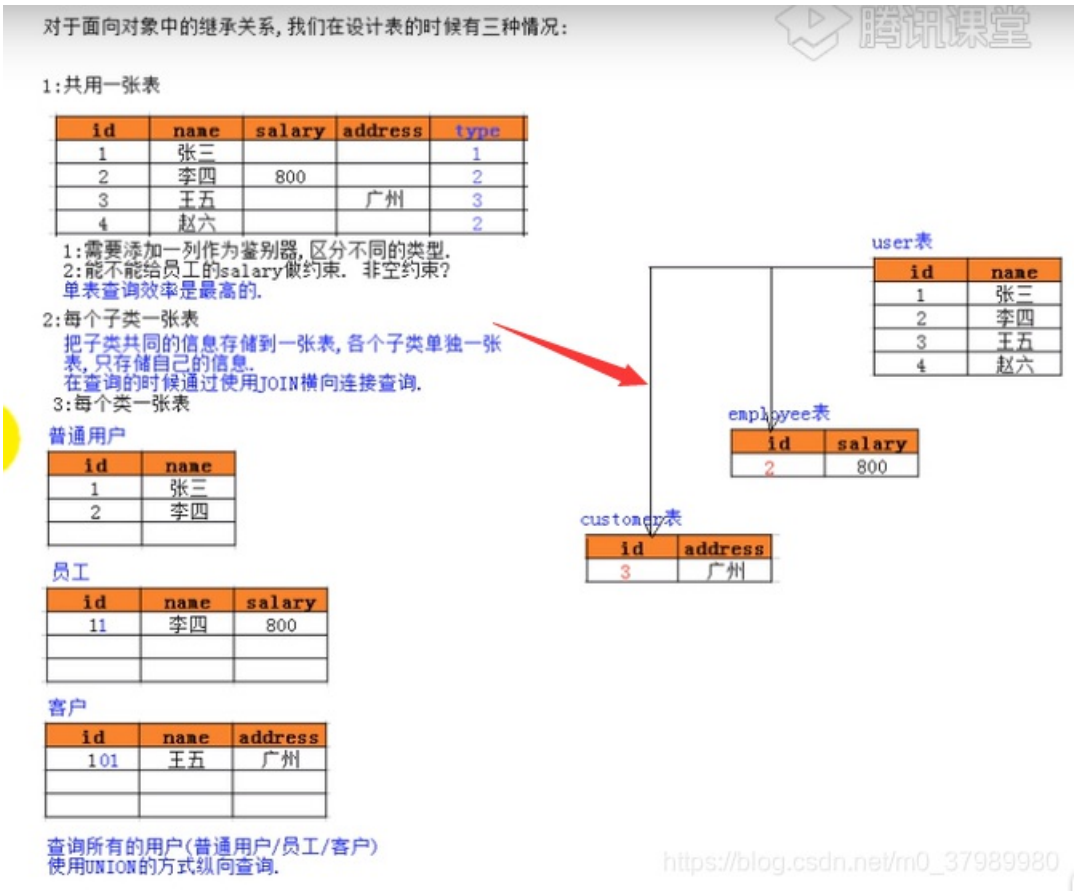
泛化关系的表设计

面向对象的设计：

```
//普通用户
public class User {
   private Long id;
   private String name;
}


//员工
public class Employee extends User{
   private BigDecimal salary;
}


//客户
public class Customer extends User{
   private String address;
}
```

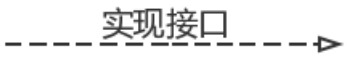
对于面向对象中的继承关系,设计表的时候有三种情况:

- 共用一张表
- 每个子类一张表
- 每个类一张表

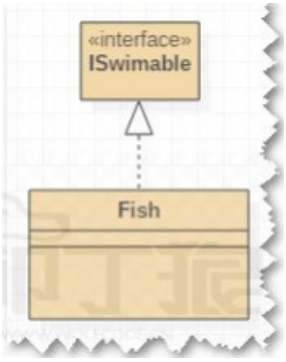


2、实现关系（realization）

跳转到目录



- 实现关系：指的是一个class类实现 interface接口（可以实现多个接口）的功能；实现是类与接口之间最常见的关系；
- 在Java中此关系通过关键字 `implements` 明确标识。
- 在UML类图中，实现通常使用 `空心三角+虚线` 表示



3、依赖关系（dependent）

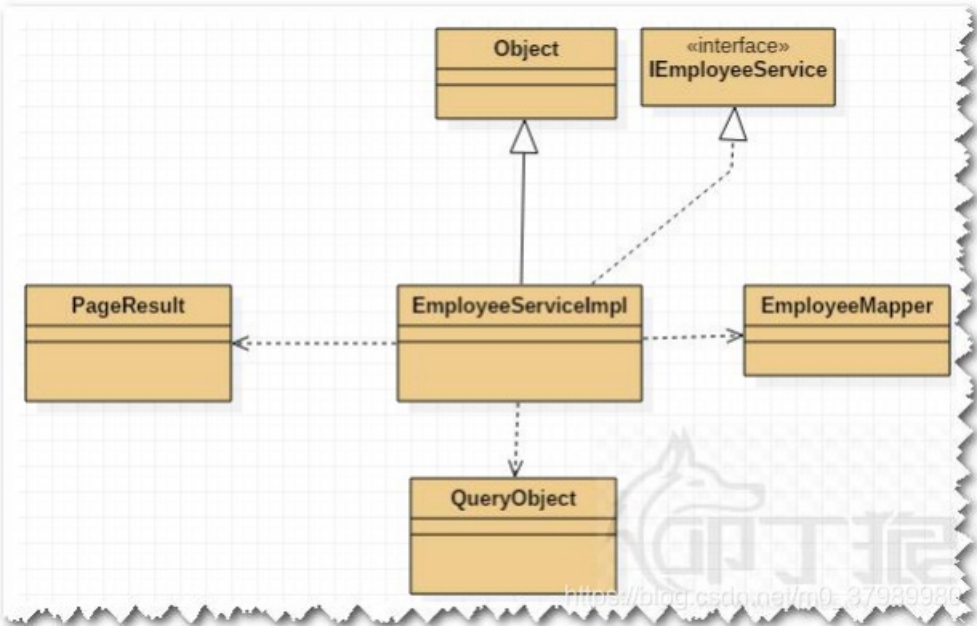
跳转到目录

- 依赖关系：指的是类与类之间的联接。依赖关系表示`一个类依赖于另一个类的定义`。一般而言，依赖关系在Java语言中体现为`成员变量、局域变量、方法的形参、方法返回值`，或者对**静态方法的调用。**
- 表示一个A类依赖于B类的定义,如果A对象离开B对象,A对象就不能正常编译,则A对象依赖于B对象(A类中使用到了B对象);
- **比如某人要过河，需要借用一条船，此时人与船之间的关系就是依赖；** 表现在代码层面，类B作为参数被类A在某个method方法中使用。
- 在UML类图中，依赖通常使用`虚线箭头`表示

```
public class BClass{
}

public class AClass{
 private BClass b1; // 依赖关系情况1:成员变量. 这也是关联关系
 public void doWork(BClass b2){ // 依赖关系情况2: 方法参数
 }
 public void doWork(){
  BClass b3; // 依赖关系情况3: 方法内的局部变量
 }
}
```

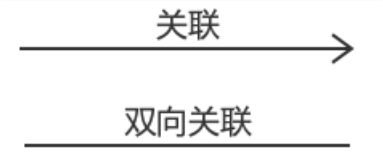
```
public class EmployeeServiceImpl implements IEmployeeService{
 private EmployeeMapper employeeMapper;
 public PageResult query(QueryObject qo){
  // TODO
  return null;
 }
}
```

上面代码折射为UML类图为:



4、关联关系（association）

跳转到目录



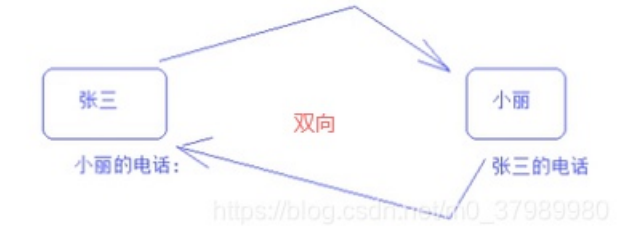
- 关联关系：指的是类与类之间的联接，**它使一个类知道另一个类的属性和方法（实例变量体现）**。A类依赖于B对象,并且把B作为A的一个成员变量, 则A和B存在关联关系.
- **关联可以是双向的，也可以是单向的**。两个类之前是一个层次的，不存在部分跟整体之间的关系。
- 在UML类图中，关联通常使用实线箭头表示

按照多重性分类：

- 一对一：一个A对象属于一个B对象，一个B对象属于一个A对象。
- 一对多：一个A对象包含多个B对象。
- 多对一：多个A对象属于一个B对象，并且多个A对象中的每个A对象只能属于一个B对象。
- 多对多：一个A对象属于多个B对象，一个B对象属于多个A对象。

按照导航性分类：

- 单向：只能从A通过属性导航到B，B不能导航到A。
- 双向：A可以通过属性导航到B，B也可以通过属性导航到A。



关联关系的判断方法：

- 判断都是从对象的实例上面来看的
- 判断关系必须确定一对属性
- 判断关系必须确定具体需求

4.1、一对一的关系

跳转到目录

一对一：一个A对象属于一个B对象，一个B对象属于一个A对象。

单向：Java类设计

```
@Data
public class QQNumber {
  private QQZone zone;
}
```

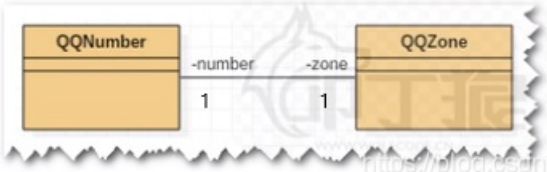
```
@Data
public class QQZone {
}
```

zone表示QQNumber中关联QQZone对象的名字

双向：Java类设计

```
@Data
public class QQNumber {
  private QQZone zone;
}
```

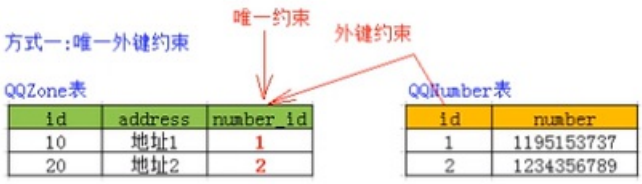
```
@Data
public class QQZone {
  private QQNumber number;
}
```



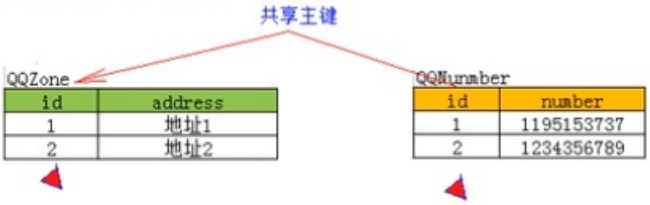
表的设计

one2one的表的设计：

方式一：唯一外键约束



方式二：共享主键(主外键)



在开发中，针对one2one情况，使用方式一。

但是我们不会使用外键约束来限定，而是通过业务代码来做到唯一。

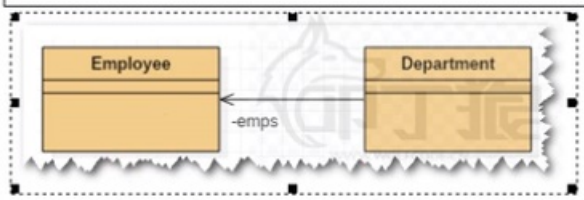
4.2、单向的一对多

跳转到目录

多对一：多个A对象属于一个B对象，并且每个A对象只能属于一个B对象。

```java
@Data
public class Department {
  private List<Employee> emps = new ArrayList<>();
}
```

```java
@Data
public class Employee {
}
```




```
Department d = new ...;


Employee e1 = new ...
Employee e2 = new ...
Employee e3 = new ...
                        单向一对多,关系在`一`的一

d.getEmps().add(e1);
d.getEmps().add(e2);
d.getEmps().add(e3);
```

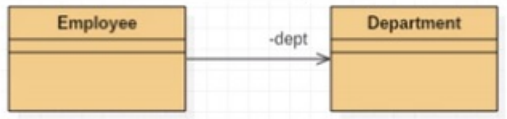
4.3、单向多对一

跳转到目录

多对一：多个A对象属于一个B对象，并且每个A对象只能属于一个B对象。

```java
@Data
public class Department {
}
```

```java
@Data
public class Employee {
  private Department dept;
}
```




```
Department d = new ...;


Employee e1 = new ...
Employee e2 = new ...
Employee e3 = new ...
                        单向多对一, 关系在`多`的一方

e1.setDept(d);
e2.setDept(d);
e3.setDept(d);
```

4.4、双向一对多、多对一

跳转到目录

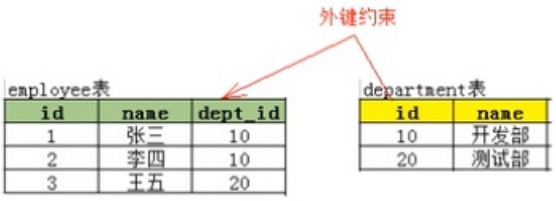
双向：A可以通过属性导航到B，B也可以通过属性导航到A，其实就是把两个单向关系组合起来。

```java
@Data
public class Department {
  private List<Employee> emps =
new ArrayList<>();
}
```

```java
@Data
public class Employee {
  private Department dept;
}
```

表的设计

外键在many方




```
保存的时候:
    先保存one方,再保存many方,会更加合理一些,性能更好(SQL要少些).

案例:保存一个部门,两个员工.

情况1:先保存部门,再保存员工:
INSERT INTO department (name) VALUES( ? );
INSERT INTO employee   (name,dept_id) VALUES( ?, ? ); 第二个?的值,就该是保存部门之后的ID值.
INSERT INTO employee   (name,dept_id) VALUES( ?, ? );

情况2:先保存员工,再保存部门:
INSERT INTO employee   (name,dept_id) VALUES( ?, NULL );此时没有对应部门的ID
INSERT INTO employee   (name,dept_id) VALUES( ?, NULL );
INSERT INTO department (name) VALUES( ? );
额外发送SQL来维护dept_id这一列的值.
UPDATE employee SET dept_id  = ? WHERE id = ?;
UPDATE employee SET dept_id  = ? WHERE id = ?;
```

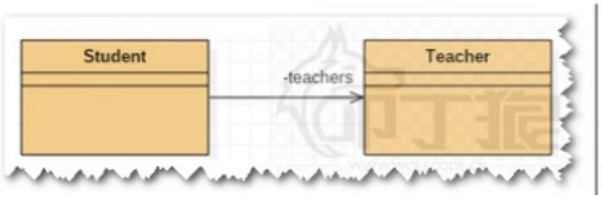

4.5、单向多对多

跳转到目录

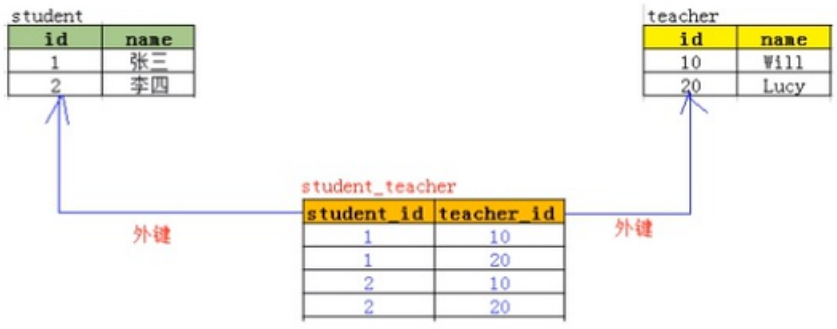
多对多：一个A对象属于多个B对象，一个B对象属于多个A对象。

```java
@Data
public class Student {
  private List<Teacher> teachers = new ArrayList<>();
}
```

```java
@Data
public class Teacher {
}
```



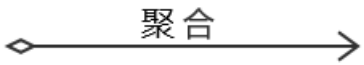
表的设计



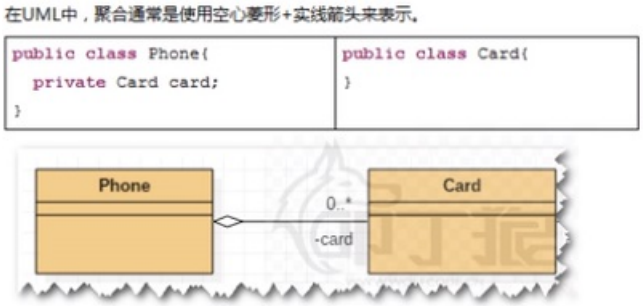

```
中间表里两个列都是外键,

需要为中间表设置主键吗?
    其实不需要;
    非要设置主键:  可以把 student_id 和 teacher_id作为一个联合主键.
```


5、聚合关系（aggregtion）

跳转到目录

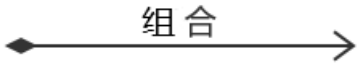


- 聚合关系是`关联关系的一种特例`，他体现的是`整体与部分`，是一种`“弱拥有”`的关系，即`has-a`的关系。聚合是`整体`和`个体`之间的关系。
- 例如，汽车类与引擎类、轮胎类，以及其它的零件类之间的关系便整体和个体的关系。
- 与关联关系一样，`聚合关系`也是通过`实例变量`实现的。**`但是关联关系所涉及的两个类是处在同一层次上的，而在聚合关系中，两个类是处在不平等层次上的，一个代表整体，另一个代表部分。`**
- 聚合关系表示整体和个体的关系，整体和个体可以相互独立存在，一定是有两个模块分别管理整体和个体。
- 在UML类图中，聚合通常使用`空心菱形+实线箭头`表示



6、组合关系（composition）

跳转到目录

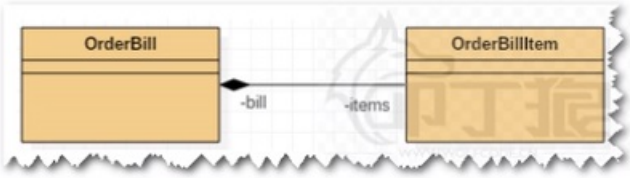


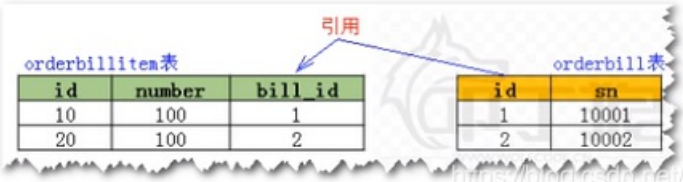
- 组合关系是`关联关系的一种特例`，他体现的是一种`contains-a`(包含)的关系，这种关系比聚合更强，也称为`强聚合`。
- 它要求普通的聚合关系中代表整体的对象负责代表部分对象的生命周期，组合关系是不能共享的。代表整体的对象需要负责保持部分对象和存活，在一些情况下将负责代表部分的对象湮灭掉。代表整体的对象可以将代表部分的对象传递给另一个对象，由后者负责此对象的生命周期。换言之，代表部分的对象在每一个时刻只能与一个对象发生组合关系，由后者排他地负责生命周期。部分和整体的生命周期一样。
- **`整体和个体不能独立存在，一定是在一个模块中同时管理整体和个体，生命周期必须相同(级联)。`**
- 在UML类图中，组合通常使用`实心菱形+实线箭头`表示

在UML中，组合通常是使用实心菱形+实线箭头表示。

```java
public class OrderBill {
  private List<OrderBillItem> items;
}
```

```java
public class OrderBillItem {
  private OrderBill bill;
}
```





相关参考: UML类图关系（泛化 、继承、实现、依赖、关联、聚合、组合）