## Java有没有有顺序的Map实现类，如果有，他们是怎么实现有序的？

Hashmap和Hashtable 都不是有序的。

TreeMap和LinkedHashmap都是有序的。（TreeMap默认是key升序，LinkedHashmap默认是数据插入顺序）

TreeMap是基于比较器Comparator来实现有序的。

LinkedHashmap是基于链表来实现数据插入有序的。

## 抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口么,类可以实现多个接口么。

1、接口可以继承接口，抽象类不可以继承接口，但可以实现接口。

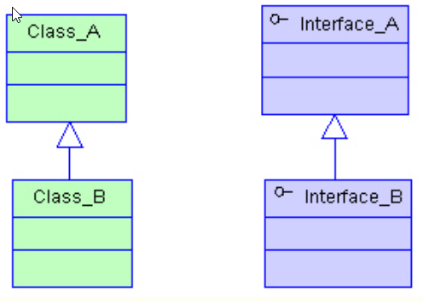
2、抽象类可以继承实体类。抽象类可以实现(implements)接口，抽象类是否可继承实体类，但前提是实体类必须有明确的构造函数。

3.抽象类可以继承实体类，就是因为抽象类的可以继承性和有方法。

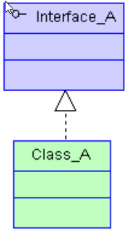
4、一个接口可以继承多个接口. interface C extends A, B {}是可以的. 一个类可以实现多个接口: class D implements A,B,C{} 但是一个类只能继承一个类,不能继承多个类 class B extends A{} 在继承类的同时,也可以继承接口: class E extends D implements A,B,C{} 这也正是选择用接口而不是抽象类的原因。

## 继承、实现、依赖、关联、聚合、组合的联系与区别

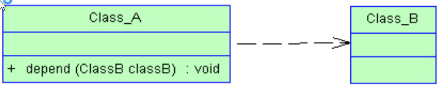
### 继承

指的是一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）的功能，并可以增加它自己的新功能的能力，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，在设计时一般没有争议性；

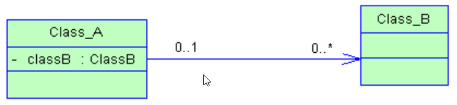
### 实现

指的是一个class类实现interface接口（可以是多个）的功能；实现是类与接口之间最常见的关系；在Java中此类关系通过关键字implements明确标识，在设计时一般没有争议性；

### 依赖

可以简单的理解，就是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、、临时性的、非常弱的，但是B类的变化会影响到A；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，为类B作为参数被类A在某个method方法中使用；

### 关联

他体现的是两个类、或者类与接口之间语义级别的一种强依赖关系，比如我和我的朋友；这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时性的，一般是长期性的，而且双方的关系一般是平等的、关联可以是单向、双向的；表现在代码层面，为被关联类B以类属性的形式出现在关联类A中，也可能是关联类A引用了一个类型为被关联类B的全局变量；

### 聚合

聚合是关联关系的一种特例，他体现的是整体与部分、拥有的关系，即has-a的关系，此时整体与部分之间是可分离的，他们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享；比如计算机与CPU、公司与员工的关系等；表现在代码层面，和关联关系是一致的，只能从语义级别来区分；

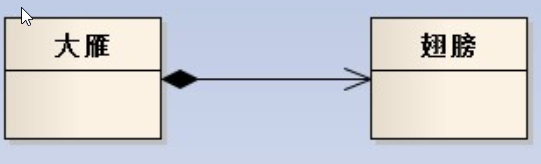
### 组合 (a拥有b，a没了b也就没了，实心)

组合也是关联关系的一种特例，他体现的是一种contains-a的关系，这种关系比聚合更强，也称为强聚合；他同样体现整体与部分间的关系，但此时整体与部分是不可分的，整体的生命周期结束也就意味着部分的生命周期结束；比如你和你的大脑；表现在代码层面，和关联关系是一致的，只能从语义级别来区分；

对于继承、实现这两种关系没多少疑问，他们体现的是一种类与类、或者类与接口间的纵向关系；其他的四者关系则体现的是类与类、或者类与接口间的引用、横向关系，是比较难区分的，有很多事物间的关系要想准备定位是很难的，前面也提到，这几种关系都是语义级别的，所以从代码层面并不能完全区分各种关系；

但总的来说，后几种关系所表现的强弱程度依次为：组合>聚合>关联>依赖；

### 聚合关系图：



### 组合关系图：

聚合关系的类里含有另一个类作为参数。

雁群类（GooseGroup）的构造函数中要用到大雁（Goose）作为参数把值传进来 大雁类（Goose）可以脱离雁群类而独立存在 。

组合关系的类里含有另一个类的实例化 。

大雁类（Goose）在实例化之前 一定要先实例化翅膀类（Wings） 两个类紧密耦合在一起 它们有相同的生命周期 翅膀类（Wings）不可以脱离大雁类（Goose）而独立存在 。

信息的封装性不同 。

在聚合关系中，客户端可以同时了解雁群类和大雁类，因为他们都是独立的 。

而在组合关系中，客户端只认识大雁类，根本就不知道翅膀类的存在，因为翅膀类被严密的封装在大雁类中。

### 聚合与组合

（1）聚合与组合都是一种结合关系，只是额外具有整体-部分的意涵。

（2）部件的生命周期不同

聚合关系中，整件不会拥有部件的生命周期，所以整件删除时，部件不会被删除。再者，多个整件可以共享同一个部件。

组合关系中，整件拥有部件的生命周期，所以整件删除时，部件一定会跟着删除。而且，多个整件不可以同时间共享同一个部件。

（3）聚合关系是“has-a”关系，组合关系是“contains-a”关系。

### 关联和聚合

（1）表现在代码层面，和关联关系是一致的，只能从语义级别来区分。

（2）关联和聚合的区别主要在语义上，关联的两个对象之间一般是平等的，例如你是我的朋友，聚合则一般不是平等的。

（3）关联是一种结构化的关系，指一种对象和另一种对象有联系。

（4）关联和聚合是视问题域而定的，例如在关心汽车的领域里，轮胎是一定要组合在汽车类中的，因为它离开了汽车就没有意义了。但是在卖轮胎的店铺业务里，就算轮胎离开了汽车，它也是有意义的，这就可以用聚合了。

### 关联和依赖

（1）关联关系中，体现的是两个类、或者类与接口之间语义级别的一种强依赖关系，比如我和我的朋友；这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时性的，一般是长期性的，而且双方的关系一般是平等的。

（2）依赖关系中，可以简单的理解，就是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、临时性的、非常弱的，但是B类的变化会影响到A。

综合比较

这几种关系都是语义级别的，所以从代码层面并不能完全区分各种关系；但总的来说，后几种关系所表现的强弱程度依次为：

组合>聚合>关联>依赖；

## IO模型有哪些，讲讲你理解的nio ，他和bio，aio的区别是啥，谈谈reactor模型。

### 什么是BIO,NIO,AIO

JAVA BIO：

同步并阻塞，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器端就需要启动一个线程并处理，如果这个连接不做任何事情会造成不必要的开销，当然可以通过线程池机制改善

JAVA NIO：

同步非阻塞，服务器实现模式为一个请求一个线程，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有IO请求时才启动一个线程进行处理

JAVA AIO(NIO2)：

异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的I/O请求都是由OS先完成了再通知服务器应用去启动线程进行处理

### 使用场景

BIO方式

适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4以前的唯一选择，但程序直观简单易理解。

NIO方式

适用于连接数目多且连接比较短（轻操作）的架构，比如聊天服务器，并发局限于应用中，编程比较复杂，JDK1.4开始支持。

AIO方式

使用于连接数目多且连接比较长（重操作）的架构，比如相册服务器，充分调用OS参与并发操作，编程比较复杂，JDK7开始支持。

### BIO 同步并阻塞

tomcat采用的传统的BIO（同步阻塞IO模型）+线程池模式，对于十万甚至百万连接的时候，传统BIO模型是无能为力的：

①线程的创建和销毁成本很高，在linux中，线程本质就是一个进程，创建销毁都是重量级的系统函数

②线程本身占用较大的内存，像java的线程栈一般至少分配512K-1M的空间，如果系统线程过高，内存占用是个问题

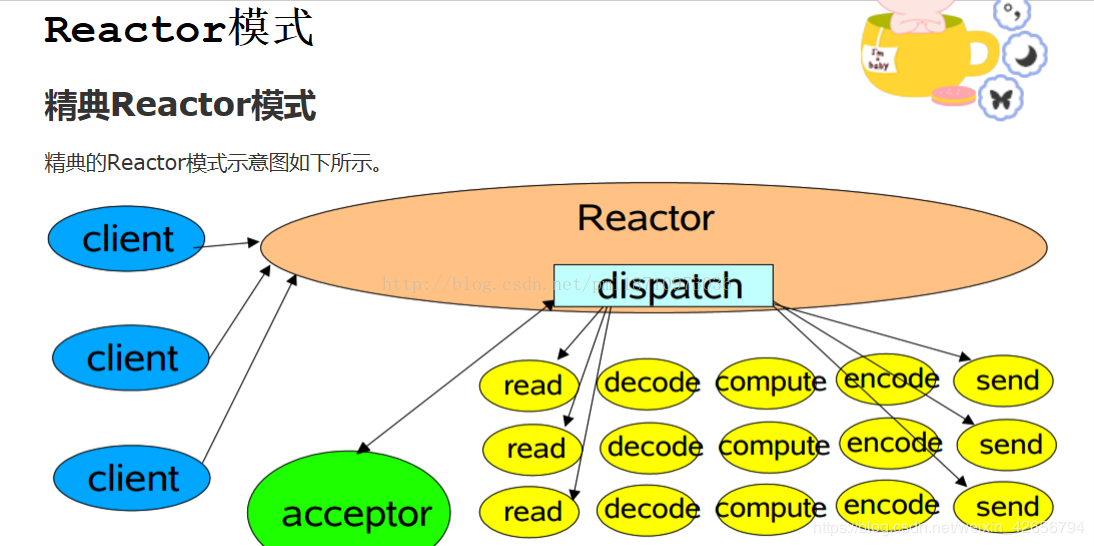
③线程的切换成本高，操作系统发生线程切换的时候，需要保留线程的上下文，然后执行系统调用，如果线程数过高可能执行线程切换的时间甚至大于线程执行的时间，这时候带来的表现是系统load偏高，CPUsy使用率很高

④容易造成锯齿状的系统负载。系统负载是用活动线程数或CPU核心数，一旦线程数量高但外部网络环境不是很稳定，就很容易造成大量请求的结果同时返回，激活大量阻塞线程从而使系统负载压力过大。

### NIO同步非阻塞

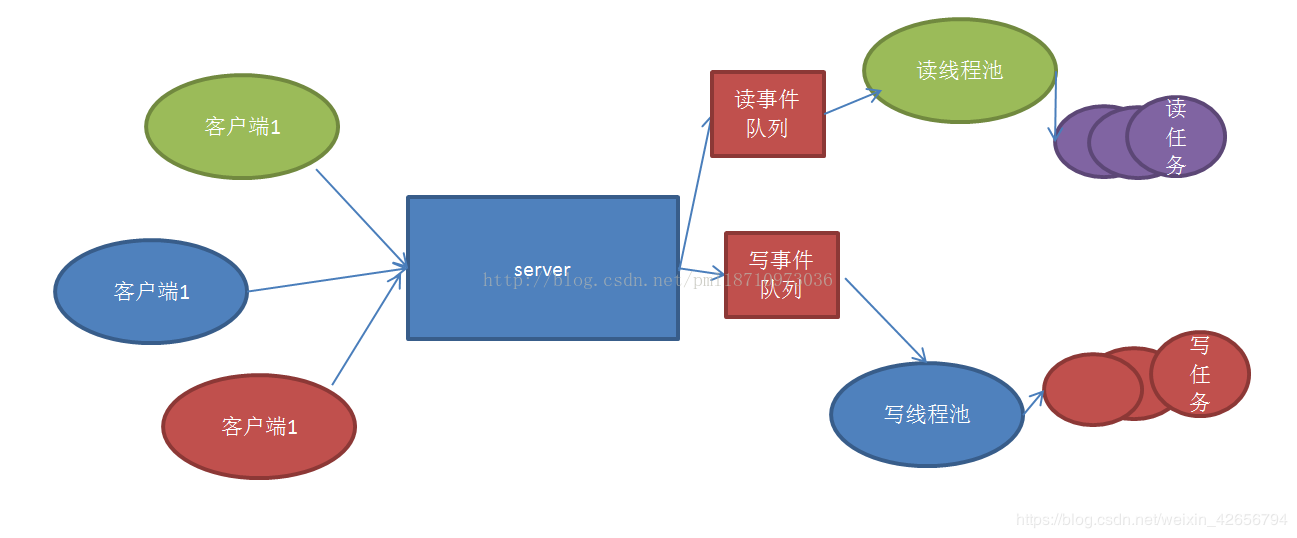
NIO基于Reactor，当socket有流可读或可写入socket，操作系统会相应的通知引用程序进行处理，应用再将流读取到缓冲区或写入操作系统。也就是，不是一个链接就要对应一个处理线程，而是一个有效请求对应一个线程，当连接没有数据时，是没有工作线程来处理的

Reactor模型



nio只有acceptor的服务线程是堵塞进行的，其他读写线程是通过注册事件的方式，有读写事件激活时才调用线程资源区执行，不会一直堵塞等着读写操作，Reactor的瓶颈主要在于acceptor的执行，读写事件也是在这一块分发

### AIO异步非堵塞IO



AIO需要一个链接注册读写事件和回调方法，当进行读写操作时，只须直接调用API的read或write方法即可，这两种方法均为异步，对于读操作而言，当有流可读取时，操作系统会将可读的流传入read方法的缓冲区，并通知应用程序；对于写操作而言，当操作系统将write方法传递的流写入完毕时，操作系统主动通知应用程序

即，read/write方法都是异步的，完成后会主动调用回调函数

## 反射的原理，反射创建类实例的三种方式是什么。

### 什么是Java反射机制

当程序运行时，允许改变程序结构或变量类型，这种语言称为动态语言。我们认为java并不是动态语言，但是它却有一个非常突出的动态相关机制，俗称：反射。

IT行业里这么说，没有反射也就没有框架，现有的框架都是以反射为基础。在实际项目开发中，用的最多的是框架，填的最多的是类，反射这一概念就是将框架和类揉在一起的调和剂。所以，反射才是接触项目开发的敲门砖！

IT行业里这么说，没有反射也就没有框架，现有的框架都是以反射为基础。在实际项目开发中，用的最多的是框架，填的最多的是类，反射这一概念就是将框架和类揉在一起的调和剂。所以，反射才是接触项目开发的敲门砖！

### 反射的应用及原理

我们可能听过，Java编写的程序，一次编译，到处运行。这也是Java程序为什么是无关平台的所在，原因在于，java的源代码会被编译成.class文件字节码，只要装有Java虚拟机JVM的地方（Java提供了各种不同平台上的虚拟机制，第一步由Java IDE进行源代码编译，得到相应类的字节码.class文件，第二步，Java字节码由JVM执行解释给目标计算机，第三步，目标计算机将结果呈现给我们计算机用户；因此，Java并不是编译机制，而是解释机制），.class文件畅通无阻。

Java的反射机制，操作的就是这个.class文件，首先加载相应类的字节码（运行eclipse的时候，.class文件的字节码会加载到内存中），随后解剖（反射 reflect）出字节码中的构造函数、方法以及变量（字段），或者说是取出，我们先来定义一个类Animal，里面定义一些构造函数，方法，以及变量：