# JAVA基础细节拾遗

## Comparator接口

通常被一些排序框架所用

排序框架会写好排序算法流程，但是其中一定缺不了比较两个数据大小的逻辑，框架开发者是无法确定这个比大小的逻辑的，所以只好定义一个接口Comparator，以让使用者在用框架对自己的数据进行排序时，传递一个具体的Comparator对象进去，框架就能调Comparator上的int compareTo(o1,o2)方法来比较两个数据的大小

## Comparable接口

通常被一些排序框架所用

排序框架会写好排序算法流程，但是其中一定缺不了比较两个数据大小的逻辑，框架开发者是无法确定这个比大小的逻辑的，使用者可以让自己的数据类型实现Comparable接口，这样，框架在排序时就能调用元素上的int compareTo(other)方法来比较两个数据的大小

## 反射

最直观感受：通过一个字符串类名，就可以通过jdk的反射api来获得这个类的对象instance

还可以通过方法名和参数类型名 来反射出Method

然后可以用method.invok(instance,args)来调用用户所指定的方法

## 动态代理

jdk中写了一个框架，叫做Proxy框架

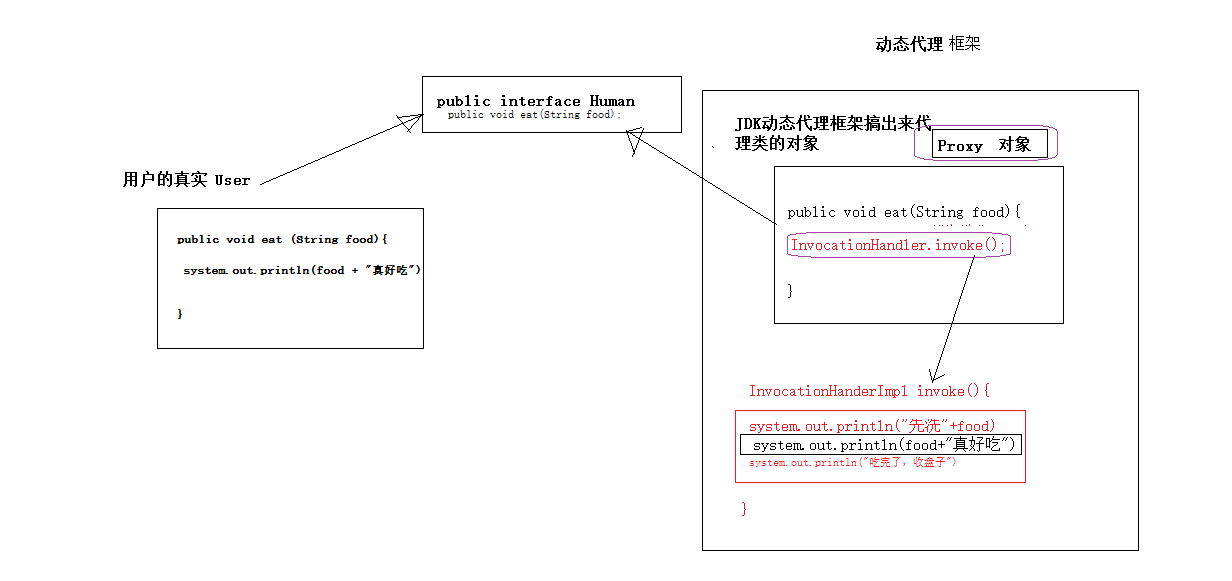
它能根据用户给定的接口，生成一个代理对象，代理对象中具备接口上定义好的方法

只不过，代理对象中的这些业务方法，其具体逻辑由Invocationhandler中的invoke()所决定

而invoke()中的逻辑是可以有用户来自定义的

这样一来，用户就可以根据接口动态地造出一个新对象（动态代理对象），其意义在于，用户可以临时为某个接口生成一个对象，来加强（或者彻底改造）之前的业务实现类的功能

示意图：



## 动态代理的应用示例：RPC

分布式系统中的底层技术

rpc是一种远程过程调用的实现机制：

调用端：通过rpc工具类来获得服务端的某个业务接口的动态代理对象

然后调用该代理对象上的方法时，rpc框架就会执行invoke，在invoke中会执行网络通信的逻辑，来向远端服务器发送调用请求信息，并接受响应结果

服务端： 通过一个serversocket来监听端口，接受客户端的请求，然后解析请求参数，然后根据客户端的请求参数调用 指定业务类的指定方法，得到结果后通过socket发送给客户端

具体代码见工程(rpc工程：rpcserver/rpcclient 用户测试工程：usertest)

## 修饰符

### 类的修饰符

public 在本包和其他包都可以引用

default 在本包可以引用

abstract

### 成员修饰符

功能：声明该成员变量或者成员方法能够在哪里引用；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 访问权限 | 本类 | 本包 | 子类 | 其他 |
| **public** | √ | √ | √ | √ |
| protect | √ | √ | √ | × |
| default | √ | √ | × | × |
| **private** | √ | × | × | × |

## 继承

使用场景： 大量事物如果有共同属性和共同的方法，就可以将这些共同属性和共同方法封装在一个所谓的父类中

然后其他的具体事物就可以通过继承父类，来直接获取公共的属性和方法

语法：

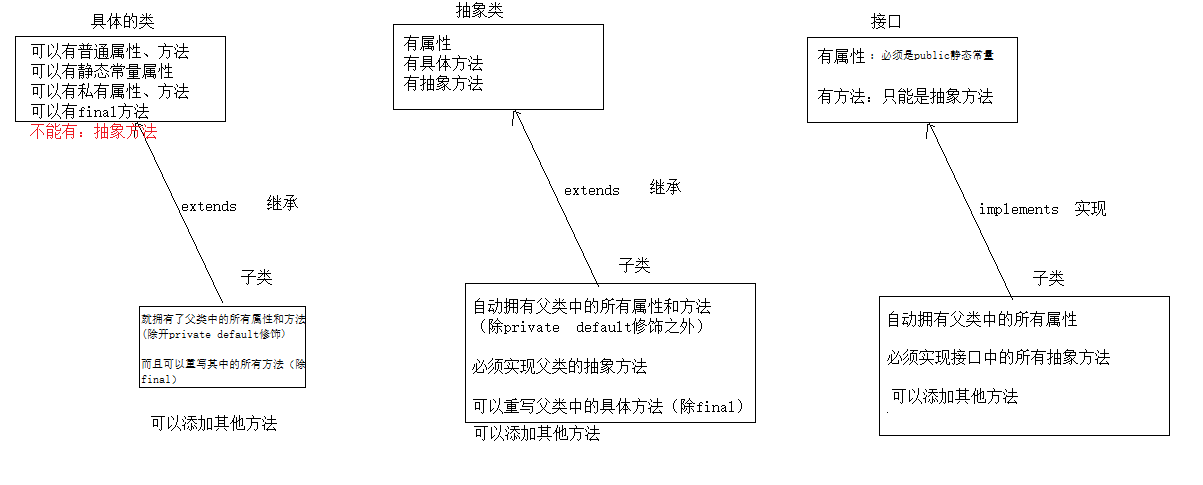
Class Student extends Person{

}

重写父类中的方法：子类中可以重写父类中的方法，调用子类对象的该方法时，执行的是子类中重写过的逻辑；

注意：如果父类中的某个方法被final所修饰，则子类不能再重写该方法

## 抽象类



抽象类  
public abstract class Human{

}

## 单例设计模式

## 工厂模式