[1、对象的概念](#3059-1585239257139)

[2、面向对象和面向过程的区别](#8484-1585239301962)

[面向过程：](#8479-1585239320451)

[面向对象：](#8473-1585239410062)

[3、三大核心特性](#7270-1585239649995)

[（1）、继承](#7015-1585239327499)

[概念：](#5671-1585239688491)

[好处：](#2020-1585240008036)

[语法规则：](#3333-1585240011235)

[（2）、封装](#8350-1585240578420)

[概念：](#2862-1585240632700)

[优点：](#4653-1585240636820)

[实现步骤：](#5828-1585240776884)

[（3）、多态](#7878-1585239704195)

[概念：](#1969-1585263270393)

[好处：](#1737-1585263286521)

[Java中的多态：](#5474-1585263414033)

**1、对象的概念**

对象具有属性和行为。

对象具有变化的状态。

对象具有唯一性。

对象都是某个类别的实例。

一切皆为对象，真实世界中的所有事物都可以视为对象。

**2、面向对象和面向过程的区别**

**面向过程：**

强调的就是功能行为，功能的执行过程，即先后顺序。

面向过程性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源。所以当性能是最重要的考量因素的时候，比如单片机、嵌入式开发的Linux / Unix等会使用面向过程。

但是，面向过程没有面向对象易维护，易替换，易扩展。

**面向对象：**

把多个功能合理放到不同对象里，强调的是具备某些功能的对象。

面向对象易维护，易复用，易扩展。因为面向对象有封装，继承，多态性的特性，所以可以使工程更加模块化，实现更低的耦合和更高的内聚，使系统更加灵活，更加易于维护。

但是，面向对象性能比面向过程低。

**PS：**Java性能差的首要并不是因为它是面向对象语言，而是Java是半编译语言，最终的执行代码不一定可以直接被CPU执行的二进制机械码。

**3、三大核心特性**

**（1）、继承**

**概念：**

子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。

**好处：**

子类拥有父类的所有属性和方法（除了private修饰的属性不能拥有）从而实现了实现代码的复用；

**语法规则：**

**A、方法的重写**

　 子类如果对继承的父类的方法不满意（不适合），可以自己编写继承的方法，这种方式就称为方法的重写。当调用方法时会优先调用子类的方法。

　 重写要注意：

　 a、返回值类型

　　 b、方法名

　　 c、参数类型及个数

　 都要与父类继承的方法相同，才叫方法的重写。

　 **重载和重写的区别：**

　　方法重载：在同一个类中处理不同数据的多个相同方法名的多态手段。

　　方法重写：相对继承而言，子类中对父类已经存在的方法进行区别化的修改。

**B、继承的初始化顺序**

　　1、初始化父类再初始化子类

　　2、先执行初始化对象中属性，再执行构造方法中的初始化。

　 基于上面两点，我们就知道实例化一个子类，java程序的执行顺序是：

　 父类对象属性初始化---->父类对象构造方法---->子类对象属性初始化--->子类对象构造方法

**C、final关键字**

　使用final关键字做标识有“最终的”含义。

　　1. final 修饰类，则该类不允许被继承。

　　2. final 修饰方法，则该方法不允许被覆盖(重写)。

　　3. final 修饰属性，则该类的该属性不会进行隐式的初始化，所以 该final 属性的初始化属性必须有值，或在\*\*构造方法中赋值(但只能选其一，且必须选其一，因为没有默认值！)，且初始化之后就不能改了，只能赋值一次。

　　4. final 修饰变量，则该变量的值只能赋一次值，在声明变量的时候才能赋值，即变为常量。

**D、super关键字**

　在对象的内部使用，可以代表父类对象。

　　1、访问父类的属性：super.age

　　2、访问父类的方法：super.eat()

　super的应用：

　首先我们知道子类的构造的过程当中必须调用父类的构造方法。其实这个过程已经隐式地使用了我们的super关键字。

　这是因为如果子类的构造方法中没有显示调用父类的构造方法，则系统默认调用父类无参的构造方法。

　那么如果自己用super关键字在子类里调用父类的构造方法，则必须在子类的构造方法中的第一行。

　要注意的是：如果子类构造方法中既没有显示调用父类的构造方法，而父类没有无参的构造方法，则编译出错。

（补充说明：虽然没有显示声明父类的无参的构造方法，系统会自动默认生成一个无参构造方法，但是，如果你声明了一个有参的构造方法，而没有声明无参的构造方法，这时系统不会动默认生成一个无参构造方法，此时称为父类有没有无参的构造方法。）

**（2）、封装**

**概念：**

指一种将抽象性函式接口的实现细节部份包装、隐藏起来的方法。封装可以让程式码更容易理解与维护，也加强了程式码的安全性。

**优点：**

良好的封装能够减少耦合。

类内部的结构可以自由修改。

可以对成员变量进行更精确的控制。

隐藏信息，实现细节。

**实现步骤：**

A、this关键字

　1.this关键字代表当前对象；this.属性 操作当前对象的属性；this.方法 调用当前对象的方法。

　2.封装对象的属性的时候，经常会使用this关键字。

　3.当getter和setter函数参数名和成员函数名重合的时候，可以使用this\*\*\*\*区别。如：

B、Java 中的内部类

　内部类（ Inner Class ）就是定义在另外一个类里面的类。与之对应，包含内部类的类被称为外部类。

　那么问题来了：那为什么要将一个类定义在另一个类里面呢？清清爽爽的独立的一个类多好啊！！

　答：内部类的主要作用如下：

　　1. 内部类提供了更好的封装，可以把内部类隐藏在外部类之内，不允许同一个包中的其他类访问该类。

　　2. 内部类的方法可以直接访问外部类的所有数据，包括私有的数据。

　　3. 内部类所实现的功能使用外部类同样可以实现，只是有时使用内部类更方便。

　　内部类可分为以下几种：

成员内部类

静态内部类

方法内部类

匿名内部类

**（3）、多态**

**概念：**

多态是同一个行为具有多个不同表现形式或形态的能力。

**好处：**

**可替换性（substitutability）**：多态对已存在代码具有可替换性。例如，多态对圆Circle类工作，对其他任何圆形几何体，如圆环，也同样工作。

**可扩充性（extensibility）**：多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。

**接口性（interface-ability）**：多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的。

**灵活性（flexibility）**：它在应用中体现了灵活多样的操作，提高了使用效率。

**简化性（simplicity）**：多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。

**Java中的多态：**

A、引用多态

父类的引用可以指向本类的对象；

　　 父类的引用可以指向子类的对象；

B、方法多态

创建本类对象时，调用的方法为本类方法；

　　 创建子类对象时，调用的方法为子类重写的方法或者继承的方法；

　　 使用多态的时候要注意：如果我们在子类中编写一个独有的方法（没有继承父类的方法），此时就不能通过父类的引用创建的子类对象来调用该方法！！！

　　 注意： 继承是多态的基础。

C、引用类型转换

向上类型转换(隐式/自动类型转换)，是小类型转换到大类型

向下类型转换(强制类型转换)，是大类型转换到小类型(有风险,可能出现数据溢出)。

instanceof运算符，来解决引用对象的类型，避免类型转换的安全性问题。

采用instanceof和getClass两种方法通过两者是否相等来判断，但是两者在判断上面是有差别的。Instanceof进行类型检查规则是:你属于该类吗？或者你属于该类的派生类吗？而通过getClass获得类型信息采用==来进行检查是否相等的操作是严格的判断,不会存在继承方面的考虑；