1、面向对象基础

1.1、面向对象思想

1.1.1、概述

面向对象(Object Oriented)是软件开发方法。面向对象的概念和应用已超越了程序设计和软件开发，是一种对现实世界理解和抽象的方法，是计算机编程技术发展到一定阶段后的产物。

面向对象是相对于面向过程来讲的，指的是把 相关的数据和方法组织为一个整体 来看待，从更高的层次来进行系统建模，更贴近事物的自然运行模式。

面向过程到面向对象思想层面的转变：

面向过程关注的是执行的过程，面向对象关注的是具备功能的对象。

面向过程到面向对象，是程序员思想上 从执行者到指挥者的转变。

此概念如果直接去理解的话可能会比较抽象，因为大家缺少对原始的面向过程的开发语言的了解，下面我们举一个栗子

1.1.2、举一个栗子

我们通过生活中的一个脑筋急转弯， 来理解这个概念。

问：

把大象装进冰箱 ， 需要分几步?

回答：

面向过程回答：

3步：1把冰箱门打开， 2把大象装进去 ， 3把冰箱门关闭

面向对象回答：

2步：1招一个能操作冰箱的工人（对象），2指挥工人装大象

思考：

如果问题改成： 把100只大象依次关进冰箱，共分为几步?

面向过程的回答： 此处需要省略N字。。。

面向对象的回答还是2步：

1招一个能操作冰箱的工人（对象） ， 2指挥工人把大象依次装进去。

结论：

从上述的栗子中， 我们发现面向过程很死板 ，是很难适应变化的 。 而面向对象更灵活，可复用性更高。

1.1.3、栗子好吃， 我们再举一个

我们再描述一个生活的场景：

场景：

当我们独自生活时， 我们经常纠结一日三餐怎么吃。

面向过程：
每天亲力亲为： 买菜 - 做饭 - 吃饭 - 洗碗 的过程。
面向对象：
招聘一个保姆，每天等吃即可。

场景升级：
假设你是一个富豪， 拥有一座占地3000亩地的庄园 ， 不再是只关注吃饭问题 ， 还有花草树木修剪，泳池维护清洗，卫生打扫，洗衣做饭。。。。。。
面向过程：
此处省略看着就累的N字。
面向对象：
招聘一个管家， 然后让管家招聘 园丁、泳池维护工、保姆等等。

结论：
从上述的栗子中， 我们发现面向过程，我们需要关注很繁琐的过程 。
而面向对象不用关注具体的细节，更关注的是统筹架构的问题。

其实我们进行大型应用开发时， 就如上述的例子一样， 如果我们写程序只关注过程的话， 代码量达到一定层次以后， 就很难再编写下去了。
如果采用面向对象的思想来设计编写程序 ， 我们

1.1.4、三大思想

面向对象思想从概念上讲分为以下三种：OOA、OOD、OOP
OOA：面向对象分析（Object Oriented Analysis）
OOD：面向对象设计（Object Oriented Design）
OOP：面向对象程序（Object Oriented Programming

1.1.5、三大特征

封装性：所有的内容对外部不可见
继承性：将其他的功能继承下来继续发展
多态性：方法的重载本身就是一个多态性的体现

1.2、类与对象

1.2.1、两者关系

类表示一个共性的产物，是一个综合的特征，而对象，是一个个性的产物，是一个个体的特征。
（类似生活中的图纸与实物的概念。）

类必须通过对象才可以使用，对象的所有操作都在类中定义。
类由属性和方法组成：
- 属性：就相当于人的一个个的特征
- 方法：就相当于人的一个个的行为，例如：说话、吃饭、唱歌、睡觉

1.2.2、类的定义格式

```
class 类名称{
    成员属性
    成员方法
}
```

1.2.3、属性与方法

```
属性定义格式：
    数据类型 属性名;
属性定义并赋值的格式：
    数据类型 属性名 = 初始化值;


方法定义格式：
    权限修饰符 返回值类型 方法名(形式参数列表){
        //方法体
        return 返回值;
    }
```

1.2.4、对象的创建与使用

```
一个类要想真正的进行操作，则必须依靠对象，对象的定义格式如下：
    类名称 对象名称 = new 类名称() ;

如果要想访问类中的属性或方法（方法的定义），则可以依靠以下的语法形式：
    访问类中的属性：    对象.属性 ;
    调用类中的方法：    对象.方法(实际参数列表) ;
```

1.3、创建对象内存分析

1.3.1、栈

```
Java栈的区域很小 ， 大概2m左右 ， 特点是存取的速度特别快

栈存储的特点是，先进后出
```

存储速度快的原因:

栈内存，通过 '栈指针' 来创建空间与释放空间 ！

指针向下移动，会创建新的内存，向上移动，会释放这些内存 ！

这种方式速度特别快 ，仅次于PC寄存器 ！

但是这种移动的方式，必须要明确移动的大小与范围 ，

明确大小与范围是为了方便指针的移动 ，这是一个对于数据存储的限制，存储的数据大小是固定的 ，影响了程序的灵活性 ~

所以我们把更大部分的数据 存储到了堆内存中

存储的是:

基本数据类型的数据 以及 引用数据类型的引用!

例如:

```
int  a  =10;

Person p = new Person();
```

10存储在栈内存中 ， 第二句代码创建的对象的引用(p)存在栈内存中

1.3.2、堆

存放的是类的对象 .

Java是一个纯面向对象语言，限制了对象的创建方式:

所有类的对象都是通过new关键字创建

new关键字，是指告诉JVM ，需要明确的去创建一个新的对象 ，去开辟一块新的堆内存空间:

堆内存与栈内存不同，优点在于我们创建对象时 ，不必关注堆内存中需要开辟多少存储空间 ，也不需要关注内存占用时长 ！

堆内存中内存的释放是由GC(垃圾回收器)完成的

垃圾回收器 回收堆内存的规则:

当栈内存中不存在此对象的引用时,则视其为垃圾 ，等待垃圾回收器回收 ！

例如:

```
Person p0 = new Person();
Person p1 = p0;
Person p2 = new Person();
```

1.3.3、方法区

存放的是
- 类信息
- 静态的变量
- 常量
- 成员方法

方法区中包含了一个特殊的区域 (常量池)(存储的是使用static修饰的成员)

1.3.4、PC寄存器

PC寄存器保存的是 当前正在执行的 JVM指令的 地址 !

在Java程序中，每个线程启动时，都会创建一个PC寄存器 !

1.3.5、本地方法栈

保存本地(native)方法的地址 !

1.4、构造方法(构造器)

1.4.1、回顾对象创建

```
Person p = new Person();
```

在右侧Person后面出现的小括号，其实就是在调用构造方法 !

1.4.2、概述

作用:
    用于对象初始化。
执行时机:
    在创建对象时,自动调用
特点:
     所有的Java类中都会至少存在一个构造方法
     如果一个类中没有明确的编写构造方法， 则编译器会自动生成一个无参的构造方法，构造方法中没有任何的代码!

     如果自行编写了任意一个构造器，则编译器不会再自动生成无参的构造方法。

1.4.3、定义格式

定义的格式：
    与普通方法基本相同，区别在于： 方法名称必须与类名相同，没有返回值类型的声明 ！
案例：

```
 public class Demo3{
    public static void main(String[] args){
        Person p = new Person();
        p = new Person();
        p = new Person();
        p = new Person();
    }
}
class Person{
    public Person(){
        System.out.println("对象创建时,此方法调用");
    }
}
```

1.4.4、构造方法设计

建议自定义无参构造方法，不要对编译器形成依赖，避免错误发生。
当类中有非常量成员变量时，建议提供两个版本的构造方法，一个是无参构造方法，一个是全属性做参数的构造方法。
当类中所有成员变量都是常量或者没有成员变量时，建议不提供任何版本的构造。

1.5、方法的重载

方法名称相同，参数类型或参数长度不同，可以完成方法的重载 ！ 方法的重载与返回值无关!
方法的重载 ,可以让我们在不同的需求下，通过传递不同的参数调用方法来完成具体的功能。

1.6、构造方法的重载

一个类，可以存在多个构造方法 ：
参数列表的长度或类型不同即可完成构造方法的重载 ~

构造方法的重载 ,可以让我们在不同的创建对象的需求下，调用不同的方法来完成对象的初始化!

1.7、匿名对象

没有对象名称的对象 就是匿名对象。

匿名对象只能使用一次，因为没有任何的对象引用，所以将称为垃圾，等待被G·C回收。
只使用一次的对象可以通过匿名对象的方式完成，这一点在以后的开发中将经常使用到。