## 基础篇

**四大编程思想**

1.POP-面向过程编程（Process Oriented Programming）

2.OOP-面向对象编程（Object Oriented Programming）

3.SOA-面向服务架构（service-oriented architecture）

4.AOP-面向方面(切面)编程（Aspect Oriented Programming）

详细：

1:面向过程编程是以功能为中心来进行思考和组织的一种编程方法，它强调的是系统的数据被加工和处理的过程，在程序设计中主要以函数或者过程为程序的基本组织方式，系统功能是由一组相关的过程和函数序列构成。面向过程强调的是功能（加工），数据仅仅是作为输入和输出存在。

2:面向对象编程的组织方式更加贴近现实世界。面向对象以对象为中心，将对象的内部组织与外部环境区分开来，将表征对象的内部属性数据与外部隔离开来，其行为与属性构成一个整体，而系统功能则表现为一系列对象之间的相互作用的序列，能更加形象的模拟或表达现实世界。在编程组织中，对象的属性与方法不再像面向过程那样分开存放，而是视为一个整体（程序的最终实现其实还是分离的，但这仅仅是物理实现上的，不影响将对象的这两个部分视为一个整体），因此具有更好的封装性和安全性（表征内部的属性数据需要通过对象的提供的方法来访问）。面向对象强调的是整体性，因此面向对象与面向过程在很多方面是可以互补的。同时由于对象继承和多态技术的引入，使得面向对象具有更强、更简洁的对现实世界的表达能力。从而增强了编程的组织性，重用性和灵活性。 面向对象依然保留着面向过程的特性，面向过程中的功能变成了对象的方法，加工处理功能变成了对象的服务性方法，而这部分方法依然需要外界的输入，同时也对外界进行输出，只是输入和输出也变成了对象。从方法论来讲，我们可以将面向过程与面向对象看做是事物的两个方面--局部与整体（注意：局部与整体是相对的），在实际应用中，两者方法都同样重要。面向过程和面向对象是编程方法中最基本的两种方法，处于编程方法体系的底层。

3:面向服务与面向过程、面向对象本质上没有什么不同，区别就在于考虑问题的层面不同。面向对象和面向过程多用于系统内部的组织和管理，而面向服务主要用于系统间的组织和管理。面向服务是更大的对象或者过程。面向服务设计的三大原则是无状态、单一实例和明确的服务接口。明确的服务接口是强制和必须的，但无状态和单一实例则不属于强制性原则，虽然说服务提供状态管理会增加服务的复杂性，多实例也一样会增加服务的复杂性（需要增加同步并发处理等，而且会导致访问不确定性），但很多情况下这又是无法避免的。现在的面向服务架构，主要用于系统间的交互和集成，有一系列的标准(XML,SOAP,WSDL,XSD,WS-policy,WS-BPEL等)。

4:面向方面应该属于面向对象的范畴，从对象组织角度来讲，我们一般采用的分类方法都是使用类似生物学分类的方法，以“继承”关系为主线，我们称之为纵向。但事实上，对象之间除了这种纵向分类之外，我们同样可以从横向的角度去观察这些对象，这就是面向方面（切面）编程的基本出发点。原来要解决这类问题，我们一般是采用接口来完成，但这有两个问题，一是对象设计的时候一般都是纵向思维，如果这个时候需要就需要考虑这些不同类的对象的这些共性，不仅会增加设计的难度和复杂性，还会造成类的接口过多而难以维护，二是需要对现有的对象动态增加这种行为或者责任的时候非常困难。现在很多程序的都是以中间语言存在，执行的时候是解释执行或者即时编译执行，这也为增加这种切面行为或者责任提供了比较好的切入口。面向方面跟Api hook很类似。

**Java基本数据类型**

boolean（布尔类型）、byte（字节类型）、char（字符类型）、short、int、long、double、float八种。

类型转换：char-->    自动转换：byte-->short-->int-->long-->float-->double

强制转换：①会损失精度，产生误差，小数点以后的数字全部舍弃。②容易超过取值范围。

记忆：8位：Byte（字节型）

16位：short（短整型）、char（字符型）

32位：int（整型）、float（单精度型/浮点型）

64位：long（长整型）、double（双精度型）

最后一个：boolean(布尔类型)

使用包装类

Character: **public final class Character implements java.io.Serializable, Comparable<Character>**

public static final int *MIN\_RADIX* = 2;

public static final int *MAX\_RADIX* = 36;

public static final char *MIN\_VALUE* = '\u0000';

public static final char *MAX\_VALUE* = '\uFFFF';

Boolean : **public final class Boolean implements java.io.Serializable, Comparable<Boolean>**

public static final Boolean *TRUE* = new Boolean(true)

public static final Boolean *FALSE* = new Boolean(false);

Byte: **public final class Byte extends Number implements Comparable<Byte>**

public static final byte *MIN\_VALUE* = -128;

public static final byte *MAX\_VALUE* = 127;

Short: **public final class Short extends Number implements Comparable<Short>**

public static final short *MIN\_VALUE* = -32768;

public static final short *MAX\_VALUE* = 32767;

Integer:  **public final class Integer extends Number implements Comparable<Integer>**

@Native public static final int *MIN\_VALUE* = 0x80000000;

@Native public static final int *MAX\_VALUE* = 0x7fffffff;

Long: **public final class Long extends Number implements Comparable<Long>**

@Native public static final long *MIN\_VALUE* = 0x8000000000000000L;

@Native public static final long *MAX\_VALUE* = 0x7fffffffffffffffL;

Double: **public final class Double extends Number implements Comparable<Double>**

public static final double *MAX\_VALUE* = 0x1.fffffffffffffP+1023;

public static final double *MIN\_VALUE* = 0x0.0000000000001P-1022; // 4.9e-324

Float: **public final class Float extends Number implements Comparable<Float>**

public static final float *MAX\_VALUE* = 0x1.fffffeP+127f; // 3.4028235e+38f

public static final float *MIN\_VALUE* = 0x0.000002P-126f; // 1.4e-45f

**short s1 = 1; s1 = s1 + 1;会报错 等价于short s1=(int) s1+1**

**short s1 = 1; s1 +=1;没错 等价于s1=(Short)s1+1**

**byte a = 127; a+=5; System.out.println(a); 结果为-124**