你知道几种编程思想?

四大编程思想简述

1)POP--面向过程编程(Process-oriented programming):

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），数据仅仅作为输入和输出存在。这种过程化的思想是一种很朴素和普遍的思想和方法，人类很多活动都是这种组织模式，比如工厂生产，企业服务等。面向过程以数据的加工处理过程为主线，忽略了过程的所属、边界和环境，混淆了服务功能和自我功能（比如人可以砍树，这就是一种服务功能，有输入也有输出；它可以提供给外部，而行走，则是自我功能，没有输入也没有输出），外部环境和内部组织，以及环境数据和原料性数据之间的区别。从思维上来讲，面向过程更强调细节，忽视了整体性和边界性，但这与现实世界有很大的出入，因为现实世界中，这种过程都不是孤立存在的，而是从属于某个对象，因此，面向过程虽然反映了现实世界的而一个方面（功能），但无法更加形象的模拟或者表示现实世界。

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

世界是由一个个对象组成的，因此面向对象的思维方式更加接近现实世界，面向对象编程的组织方式也更加贴近现实世界。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。在面向对象编程中，大多时候，我们并不需要关心一个对象对象的方方面面，有些对象在整个系统中都是充当“原料”和“成品”的角色，其本身的行为并不在我们关心的范围，而另外有些对象处于一种加工厂地位，我们也仅关心这些对象的服务性功能，不需要太多关注对象内部属性和自我行为，针对这些对象关注点的不同会对对象进行分类，比如前面提到的两类对象，就是从在系统中所处的角色不同而分类，前者叫实体对象，后者称为操作对象。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)SOA--面向服务架构

面向服务以服务为出发点，组织和协调相关的对象来提供目标服务，对外提供必要的参数输入接口，将服务的结果作为输出，而“服务”本身的计算过程和组织则被封装在一起，对用户透明。其实面向服务也是以功能（服务）为中心，但其强调的是功能的整体性，封装性、自包性，而不是过程性和协作性，整体性指的是服务对外是作为一整体来体现的；封装性指的是服务完成的计算和处理过程、自有属性都不直接暴露给外部，除了通过公共的服务接口进行交互外，用户无法也不用知道内部的具体组织和协调的；自包性指的是服务的完成不依赖于服务的调用方，服务系统的本身就可以完成服务所需的功能；因此面向服务在程序组织上处于更高的层次，是一种粗粒度的组织方法。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)AOP--面向方面.

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。