**PO 是** Persistant Object 的缩写，用于表示数据库中的一条记录映射成的 java 对象。PO 仅仅用于表示数据，没有任何数据操作。通常遵守 Java Bean 的规范，拥有 getter/setter 方法。

**DAO 是** Data Access Object 的缩写，用于表示一个数据访问对象。使用 DAO 访问数据库，包括插入、更新、删除、查询等操作，与 PO 一起使用。DAO 一般在持久层，完全封装数据库操作，对外暴露的方法使得上层应用不需要关注数据库相关的任何信息。

**VO 是** Value Object 的缩写，用于表示一个与前端进行交互的 java 对象。有的朋友也许有疑问，这里可不可以使用 PO 传递数据？实际上，这里的 VO 只包含前端需要展示的数据即可，对于前端不需要的数据，比如数据创建和修改的时间等字段，出于减少传输数据量大小和保护数据库结构不外泄的目的，不应该在 VO 中体现出来。通常遵守 Java Bean 的规范，拥有 getter/setter 方法。

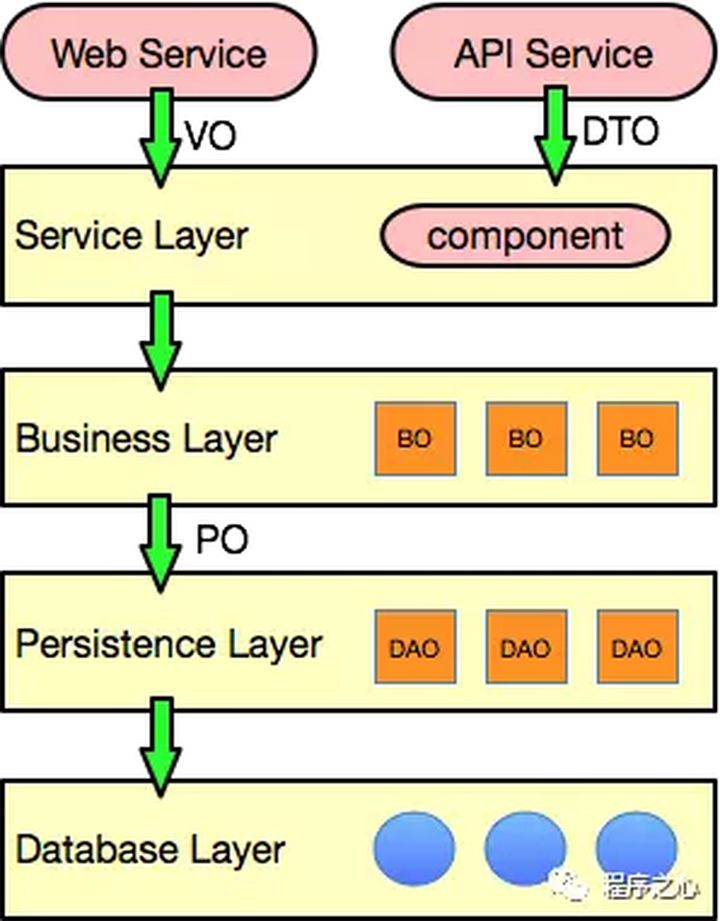
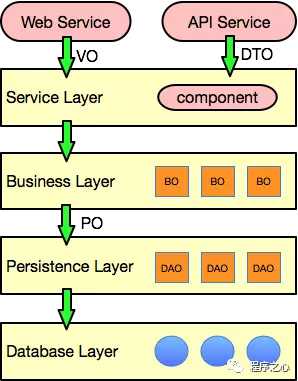
**DTO 是** Data Transfer Object 的缩写，用于表示一个数据传输对象。DTO 通常用于不同服务或服务不同分层之间的数据传输。DTO 与 VO 概念相似，并且通常情况下字段也基本一致。但 DTO 与 VO 又有一些不同，这个不同主要是设计理念上的，比如 API 服务需要使用的 DTO 就可能与 VO 存在差异。通常遵守 Java Bean 的规范，拥有 getter/setter 方法。

**BO 是** Business Object 的缩写，用于表示一个业务对象。BO 包括了业务逻辑，常常封装了对 DAO、RPC 等的调用，可以进行 PO 与 VO/DTO 之间的转换。BO 通常位于业务层，要区别于直接对外提供服务的服务层：BO 提供了基本业务单元的基本业务操作，在设计上属于被服务层业务流程调用的对象，一个业务流程可能需要调用多个 BO 来完成。

**POJO 是** Plain Ordinary Java Object 的缩写，表示一个简单 java 对象。上面说的 PO、VO、DTO 都是典型的 POJO。而 DAO、BO 一般都不是 POJO，只提供一些调用方法。

**应用**

不同类型的对象在架构设计中用于不同的用途，如下的分层架构表示了各个 POJO 的用途。为什么要在分层架构中，定义这些 POJO 对象呢？主要是为了确保各个分层能够很好地封装自己的服务，有效地控制信息的传播。



试想一下，如果没有 VO 和 PO 的区别，那么数据库表结构的所有字段就一览无余地展示到了前端，给后台安全带来很大的隐患，并且无法在网络传输中剥离冗余信息提高了用户的带宽成本。

**实例**

以一个实例来探讨下 POJO 的使用。假设我们有一个面试系统，数据库中存储了很多面试题，通过 web 和 API 提供服务。可能会做如下的设计：

数据表：表中的面试题包括编号、题目、选项、答案、创建时间、修改时间；

PO：包括题目、选项、答案、创建时间、修改时间；

VO：题目、选项、答案、上一题URL、下一题URL；

DTO：编号、题目、选项、答案、上一题编号、下一题编号；

DAO：数据库增删改查方法；

BO：业务基本操作。

可以看到，进行 POJO 划分后，我们得到了一个设计良好的架构，各层数据对象的修改完全可以控制在有限的范围内。本文首发于公众号“程序之心”，欢迎关注。

一些简单Java类的概念：（用于信息进行信息存储的类，并且不涉及复杂的逻辑处理结构Java实体bean）

JavaBean规范：

1.类名称要求有意义，可以明确描述某一类的实体。

2.类中所有的属性必须使用private进行封装并且要提供相应的setter和getter方法。

3.类中必须提供一个无参构造方法，并且多个构造方法必须按照参数的个数进行升序或者降序排列。

4.类中不允许有任何的System.out.print()的直接输出，所有结果必须返回给调用处才能直接输出。

5.类中必须提供一个获取类信息的方法，即是获取类所有的属性的方法，即是必须要重写toString（）方法。