**程序设计架构分层演进**

1、MVC是web开发中常见的程序结构，一般结构如下：

（1）view层：视图层。

（2）controller层：业务层，集合了各种action。

（3）model层：模型层，一般和数据打交道，既存储数据，也包含了对数据的操作。简单的sample，一个表对应一个model类；一个展现视图也可以对应一个model类。

从MVC三层模型设计之初，就是为了将controller层、view层以及model层区分开来。Model集合了数据存储和数据操作功能，control层调用model层的方法，实现对数据的访问。

这里并没有数据库这个概念，所以模型层会有一些冗杂，两个表的联合查询出来的数据，会被封装成一个模型交给控制层；同样的，控制层因为没有服务的概念，如果项目比较大，也会变的有些冗余。

采用这样的结构在一定程度上，可以做到代码清晰，较容易扩展，代码的管理复杂度较低。但是如果是业务很多，逻辑又很复杂的网站，如果再加上开发人员的水平参差不齐，那必然会导致下面的情况：

（1）control层中的action的代码越来越长，逻辑越来越复杂，不同action之间看起来有很多可以重用的代码， 但是真要进行重构的话，又非常困难。

（2）model层中包含的方法越来越多，有些方法也过于复杂。甚至在不少方法中还包含了业务逻辑。

（3）代码的修改，还是牵一发而动全身。

（4）代码难以进行自动化测试。

本来以为引入了mvc，程序的管理复杂度问题就高枕无忧了，但现在又面临了相同的问题了。为了将数据、业务和操作分离，在mvc中再引入Entity层、Service层和Dao层，可以在很大程度上避免或者缓解上述问题。

原有的mvc结构改成如下：

（1）view层：显示层。

（2）control层：业务层，集合了各种action。

（3）entity层：实体层，集合了各种实体类。

（3）service层。

（4）DAO层。

原来的model层不见了，增加了entity层、service层和DAO层。DAO，即Data Access Object，数据访问接口，数据访问：顾名思义就是与数据库打交道。在这个结构中，control不直接和DAO联系，需要操作数据的时候，通过service层访问DAO层来实现。service层做的事情，不仅仅是调用DAO操作数据，还会包含了一定的业务逻辑。整个程序的设计，也变成了针对服务进行设计。不带@ResponseBody的 返回页面。带@ResponseBody的 返回数据。在controller中一般完成对对象的封装，这样在service中只需要处理对象。

这样做的好处是：

（1）control层中的action得以精简，因为action中的一些逻辑，被重构成一个个的服务。而不同的action也可以重用服务了。

（2）只负责和数据打交道的DAO层，相比之前的model层，也得以精简（DAO层尽量只做最原子的数据操作，不同数据操作之间的联系，这边不考虑，那是service层的事情）。

（3）service层可以实现很大程度上的代码复用，程序的功能封装更清晰了。在service中执行的是事务，（插入、更新、删除）出现错误可以回滚。只有查询可以放在controller，不过最好也放在service。

（4）由于service层更加清晰的定义了应用程序的边界，那么对于各个service函数（对应某个服务/应用），要做到自动化测试就方便多了。WEB程序如何做到能方便的进行单元测试，这是一直困扰我的难题，这样的设计似乎真的可行了~

（5）开发人员的工作分配，理论上真的可以按层次划分了。只是理论上~

同时，这样的设计模式也是存在一定的缺点的：

（1）层次太多，刚接触的开发人员理解起来比简单的mvc结构费时；

（2）service层的设计需要一定的功力，因为action中和model层的逻辑在很大程度上转移到这里了。

但整体上看，service层的引入，更加清晰的定义了应用程序的边界，提供了一系列可以重用的操作集合。这对于网站的可扩展性和可维护性是非常有帮助的。

当然，如果网站的业务逻辑并不复杂，完全没必要用这样的设计。过度设计是万恶之源~