# [为什么JavaWeb项目要分层](https://www.cnblogs.com/vicky-li/p/10615670.html)

首先让我们坐着时光机回到n年前的[web开发](https://www.baidu.com/s?wd=web%E5%BC%80%E5%8F%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)。  
那个时候最早都是静态的html页面，后来有了数据库，有了所谓的[动态页面](https://www.baidu.com/s?wd=%E5%8A%A8%E6%80%81%E9%A1%B5%E9%9D%A2&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)，  
然后[程序猿](https://www.baidu.com/s?wd=%E7%A8%8B%E5%BA%8F%E7%8C%BF&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)在编码的时候，会把所有的代码都写在页面上，包括数据库连接，包括事务控制，接收参数，各种校验，各种逻辑，各种html/js/css代码等等  
怎么样？够乱吧？像一坨那什么一样，这个页面可能有成千上万行？  
  
那么好，问题来了，回头需要修改的时候，你怎么办？  
你找个东西找半天，好不容易找到了，还不敢改，怕被其他地方用了，改出连带问题。  
页面一出错，定位不准到底是哪里的问题，从头到尾的挨个排查。  
等等等等。  
  
这就是大家常说的什么叫可维护性，这也是为什么越来越多的公司的规范要求不能写复杂sql。  
  
还记得之前在东软的时候，一哥们写了一个80多行的大sql来完成一个核心的查询。  
试问这个大sql天天在数据库里run，还有性能可言？  
再试问谁敢改？  
后来项目要改需求还是出现bug了，那个sql要改动，写sql的哥们改了好久才改好，因为时间长他也忘了，  
再后来他离职了。。。  
  
有人问，那简单sql实现不了我的功能呀，怎么办？  
从[数据库设计](https://www.baidu.com/s?wd=%E6%95%B0%E6%8D%AE%E5%BA%93%E8%AE%BE%E8%AE%A1&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)层面开始下手，要允许适当的冗余，把表弄好，就迎刃而解了，这也是数据库层面的一种解耦吧。  
  
后来。。。  
进入第二阶段，大家痛定思痛，决定要把页面和逻辑拆开，页面只是负责显示，逻辑都在后台。  
这就出现了短暂的，在jsp里使用标签调用bean的用法。bean里耦合了除了页面之外的所有东西。  
  
再后来。。。  
进入了第三阶段，大家又痛定思痛，决定要拆成三部分，就是大名鼎鼎的MVC。  
  
再再后来。。。  
衍生出来了类似于struts/springmvc等等的[mvc框架](https://www.baidu.com/s?wd=mvc%E6%A1%86%E6%9E%B6&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)  
---------------  
JavaWeb项目的层有2个维度。  
  
第一个维度是MVC的三层：  
M：model，模型层，包括了你的[业务逻辑](https://www.baidu.com/s?wd=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)和数据库操作，封装好给视图层使用的。  
V：view，视图层，仅仅做的是展示数据，不包含[业务逻辑](https://www.baidu.com/s?wd=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)，主要是jsp/html等等  
C：controller，控制层，负责接收请求，调用模型层处理[业务逻辑](https://www.baidu.com/s?wd=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)并返回给视图层。  
  
第二个维度是java代码里的三层：  
controller：控制层，负责接收参数/解析参数/封装参数，调用serivce，将service方法的返回值进行封装（如果需要），返回数据/返回页面，路由。  
service：负责业务逻辑，事务控制在这层里做，被controller调用，以及调用dao。  
dao：持久层，负责数据库交互，被service调用。  
  
这2个维度别弄混了哟。我今天主要说的是第二个维度的层哟。  
我认为，第二个维度是第一个维度的延伸，其实第二个维度再加上一个表现层就完美了，这就为什么有人说是4层架构。  
---------------  
前戏结束，步入正题：  
  
有些学生朋友可能会问为什么要分层呢？我本来可以在一个地方写完的东西，非要散落在各个层中，都在一起不是挺好的吗？  
开发效率高呀~  
跳来跳去的我脑子都晕啦。。。  
  
这就是为什么有人会把所有的东西都扔在一个层里，比如controller层。。。  
  
其实我们可以在jsp上把所有的逻辑以及数据库操作,数据展示全部写在一起，耦合在一起，这样开发起来貌似更快，  
但是维护性非常差，有朝一日想改一个小地方，牵一发而动全一身，隐患很高，无法快速定位问题。  
因此我们需要分层。  
分层了之后，你理论上改了持久层的东西，逻辑层是不用变动的。  
  
每个Dao类是跟每个表走，Dao的每个方法里就一个个的简单sql，不包含任何业务逻辑，可以被不同的service复用和调用。  
  
经过抽象后基本上都是通用的，因而我们在定义DAO层的时候可以将相关的方法定义完毕，  
这样的好处是在对Service进行扩展的时候不需要再对DAO层进行修改，提高了程序的[可扩展性](https://www.baidu.com/s?wd=%E5%8F%AF%E6%89%A9%E5%B1%95%E6%80%A7&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)。  
  
提高了程序的[可扩展性](https://www.baidu.com/s?wd=%E5%8F%AF%E6%89%A9%E5%B1%95%E6%80%A7&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)。具体什么时候做这些操作，怎么做这些操作都由Service来处理。  
（就像[郭德纲](https://www.baidu.com/s?wd=%E9%83%AD%E5%BE%B7%E7%BA%B2&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)的相声里的一句话：“行了，你甭管了”）  
  
而Service则不是，一个Service里可以会调用多个不同的dao，完成特定的业务逻辑，事务控制，  
封装Service层的业务逻辑有利于通用的业务逻辑的独立性和重复利用性  
  
同时，一个Service的方法也有可能被多个Controller的方法来调用。  
  
逻辑出问题就在Service找问题，数据库，sql有问题就在Dao层找问题，  
参数解析错误，跳转错误，就在Controller上找问题。  
这样快速定位问题，互不干扰。  
  
---------------  
分层架构（这里会延伸到更广阔的架构）：  
  
回头项目玩大了，怎么办？拆！！！  
  
具体可以搜一下：maven分模块开发，怎么玩代码依赖，怎么玩微服务，怎么玩SOA，怎么玩RPC，怎么玩dubbo。  
  
web项目发展有几个阶段啊  
  
第一个阶段（单一应用架构）：  
所有代码都耦合在一个项目里，放在一台服务器上，这种all in one的方式是有好处的。  
创业初期，不用什么架构，走[敏捷开发](https://www.baidu.com/s?wd=%E6%95%8F%E6%8D%B7%E5%BC%80%E5%8F%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)，最快速的实现需求才是王道。  
你甭管我有多烂，我至少能跑起来，我至少能在外网上让你访问，让你使用。  
当然了，初期的[访问量](https://www.baidu.com/s?wd=%E8%AE%BF%E9%97%AE%E9%87%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)很少，节省部署和运维成本才是王道呀。  
听阿里的讲座，说淘宝的前期的版本用的就是一台PC机作为服务器，所有的功能耦合在一个项目里，  
每次往生产环境上发版本，要上传一个600mb的包，呵呵。  
  
第二个阶段（垂直应用架构）：  
哎哟，不错哦。自己的儿子被越来越多的人访问，[访问量](https://www.baidu.com/s?wd=%E8%AE%BF%E9%97%AE%E9%87%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://www.cnblogs.com/vicky-li/p/_blank)激增，一台服务器扛不住了，  
没事，我们可以玩负载均衡，玩集群。  
但是！这种性能加速度其实会变得越来越小，因为你的项目是耦合在一起的。  
这时，我们需要拆分项目，这里又有2个维度，按层拆，按模块拆。  
将拆好的不同项目分别部署在不同的服务器上，并且再分不同的小集群。  
  
第三个阶段（分布式服务架构）：  
唉呀妈呀，访问量陡增，到这步你创业应该算成功了，开始烧投资人的钱了吧。  
经过上面拆成了越来越多的模块，模块与模块交互越来越多，怎么办？  
这个时候我们需要把核心的业务抽出来，作为独立的服务，慢慢发展成稳定的服务中心，  
用来提升业务复用和整合。  
就像阿里的大牛说过，没有淘宝的积累，天猫能那么快的出来吗？  
这个时候，你的缓存，数据库，消息队列等服务都应该是分布式的。  
  
第四个阶段（流动计算架构）  
哎呀妈呀，访问量又上了一个台阶，翻了好几百倍吖，肿么办？  
这个时候服务越来越多，一些容量和资源的浪费问题凸显出来，  
这时我们需要一个调度中心来基于访问压力动态的管理集群容量，  
提高利用率。  
  
第五个阶段（云计算架构）  
抱歉，作者正在学习中，未完待续~

别处看的，优秀~