J2EE 3大框架

J2EE三大框架Struts、hibernate和spring，struts 主要负责表示层的显示，spring 利用它的IOC和AOP来处理控制业务（负责对数据库的操作），hibernate 主要作用是数据的持久化到数据库。

Spring是一个解决了许多在J2EE开发中常见的问题的强大框架。 Spring提供了管理业务对象的一致方法并且鼓励了注入对接口编程而不是对类编程的良好习惯。光谷校区专业老师指出，Spring的架构基础是基于使用JavaBean属性的Inversion of Control容器。然而，这仅仅是完整图景中的一部分：Spring在使用IoC容器作为构建完关注所有架构层的完整解决方案方面是独一无二的。Spring提供了唯一的数据访问抽象，包括简单和有效率的JDBC框架，极大的改进了效率并且减少了可能的错误。Spring的数据访问架构还集成了Hibernate和其他O/R mapping解决方案。Spring还提供了唯一的事务管理抽象，它能够在各种底层事务管理技术，例如JTA或者JDBC事务提供一个一致的编程模型。Spring提供了一个用标准Java语言编写的AOP框架，它给POJOs提供了声明式的事务管理和其他企业事务--如果你需要--还能实现你自己的aspects。这个框架足够强大，使得应用程序能够抛开EJB的复杂性，同时享受着和传统EJB相关的关键服务。Spring还提供了可以和IoC容器集成的强大而灵活的MVC Web框架。

说到Struts框架，它是一个基于Sun J2EE平台的MVC框架，主要是采用Servlet和JSP技术来实现的。由于Struts能充分满足应用开发的需求，简单易用，敏捷迅速，在过去的一年中颇受关注。Struts把Servlet、JSP、自定义标签和信息资源(message resources)整合到一个统一的框架中，开发人员利用其进行开发时不用再自己编码实现全套MVC模式，极大的节省了时间，所以说Struts是一个非常不错的应用框架。

Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序实用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的J2EE架构中取代CMP，完成数据持久化的重任。

MVC 框架

MVC指MVC模式的某种框架，它强制性的使应用程序的输入、处理和[输出](https://baike.baidu.com/item/%E8%BE%93%E5%87%BA)分开。使用MVC应用程序被分成三个核心部件：模型、[视图](https://baike.baidu.com/item/%E8%A7%86%E5%9B%BE)、控制器。它们各自处理自己的任务。

视图是用户看到并与之交互的界面。对老式的Web应用程序来说，视图就是由[HTML](https://baike.baidu.com/item/HTML)元素组成的界面，在新式的Web应用程序中，[HTML](https://baike.baidu.com/item/HTML)依旧在视图中扮演着重要的角色，但一些新的技术已层出不穷，它们包括[Adobe Flash](https://baike.baidu.com/item/Adobe%20Flash)和像[XHTML](https://baike.baidu.com/item/XHTML)，[XML](https://baike.baidu.com/item/XML)/[XSL](https://baike.baidu.com/item/XSL),[WML](https://baike.baidu.com/item/WML)等一些标识语言和[Web services](https://baike.baidu.com/item/Web%20services). MVC好处是它能为应用程序处理很多不同的[视图](https://baike.baidu.com/item/%E8%A7%86%E5%9B%BE)。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

模型表示企业数据和业务规则。在MVC的三个部件中，模型拥有最多的处理任务。例如它可能用像[EJB](https://baike.baidu.com/item/EJB)s和ColdFusion Components这样的构件对象来处理数据库，被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据，由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

控制器接受用户的输入并调用模型和视图去完成用户的需求，所以当单击Web页面中的超链接和发送[HTML表单](https://baike.baidu.com/item/HTML%E8%A1%A8%E5%8D%95)时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

J2EE 和 MVC 的联系

首先说三层架构：  
UI(.aspx)---------> BLL(业务处理)------> DAL(数据处理)----> 永久存储（数据库）   
  
MVC：  
MVC(Model View Controller)模型－视图－控制器   
  
  
很明显都是从整体上“策划”一个web项目的实现逻辑  
  
共同点：三层架构的UI层相当于MVC中的View层，作为视图，再说白一点，都是页面  
  
区别：BLL+DAL相当于MVC中的Model层， Model层实现系统中的业务逻辑，当然也包含了数据访问的逻辑  
  
      三层”中典型的Model层是已实体类构成的，而MVC里，Model则是由业务逻辑与访问数据组成的，  
  
Model层又分为不同的层（个人认为就是三层架构的DAL+BLL），它的分层也是为了结构清晰和低耦合，  
  
区别比较大的就是三层架构中没有Control层，而是由单个页面上的控件的事件处理页面与业务逻辑之间  
  
，而MVC中control层是作为联系视图层和Model的纽带，使得整个项目的结构更加清晰，降低了耦合性。  
  
  
举例说明这两种方法不同的实现思路：A在上海的浦东区逛街，有人要抢劫他，打110报警了，B在闵行区也被劫持，他也打110报警了，他们打110的时候，接电话的是上海市公安局总部指挥中心，对于A，来解救他是浦东分局的警察，对于B，解救他的是闵行分局的警察，对于AB来说，他们不需要关心到底是谁来解救他的，他们只管打110报警（类似于页面数据由action提交到控制器），由110指挥中心确定他的位置然后派出具体的地方警局去营救（控制器根据需求调用model层去完成对应的数据处理）。而三层架构在这个过程中就像A或B被劫持了，他们直接找到当地警（调用BLL层方法）的警察来处理，  
  
  
这些东西都是源于接手的破项目，不太规范的架构和编码看的我很难受，不管MVC还是三层架构，  
  
只要严格遵循起来，系统的扩展性和课维护性都不会很差。  
  
其抱怨人家留下来的系统烂，其实自己的也好不到那去，只不过是自己写的自己思路上清楚点而已。所以感觉以后要严格要求自己，争取写出的代码清晰一点，规范一点。

三层架构(3-tier application)   
通常意义上的三层架构就是将整个业务应用划分为：表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）

概念简介   
　　１、表现层（UI）：通俗讲就是展现给用户的界面，即用户在使用一个系统的时候他的所见所得。 　　２、业务逻辑层（BLL）：针对具体问题的操作，也可以说是对数据层的操作，对数据业务逻辑处理。 　　３、数据访问层（DAL）：该层所做事务直接操作数据库，针对数据的增添、删除、修改、更新、查找等   
  
概述   
　　在软件体系架构设计中，分层式结构是最常见，也是最重要的一种结构。微软推荐的分层式结构一般分为三层，从下至上分别为：数据访问层、业务逻辑层（又或成为领域层）、表示层。 　　三层结构原理： 　　3个层次中，系统主要功能和业务逻辑都在业务逻辑层进行处理。 　　所谓三层体系结构，是在客户端与数据库之间加入了一个“中间层”，也叫组件层。这里所说的三层体系，不是指物理上的三层，不是简单地放置三台机器就是三层体系结构，也不仅仅有B/S应用才是三层体系结构，三层是指逻辑上的三层，即使这三个层放置到一台机器上。 　　三层体系的应用程序将业务规则、数据访问、合法性校验等工作放到了中间层进行处理。通常情况下，客户端不直接与数据库进行交互，而是通过COM/DCOM通讯与中间层建立连接，再经由中间层与数据库进行交互。   
  
表示层   
　　　位于最外层（最上层），离用户最近。用于显示数据和接收用户输入的数据，为用户提供一种交互式操作的界面。   
　　   
业务逻辑层   
　　　业务逻辑层（Business Logic Layer）无疑是系统架构中体现核心价值的部分。它的关注点主要集中在业务规则的制定、业务流程的实现等与业务需求有关的系统设计，也即是说它是与系统所应对的领域（Domain）逻辑有关，很多时候，也将业务逻辑层称为领域层。例如Martin Fowler在《Patterns of Enterprise Application Architecture》一书中，将整个架构分为三个主要的层：表示层、领域层和数据源层。作为领域驱动设计的先驱Eric Evans，对业务逻辑层作了更细致地划分，细分为应用层与领域层，通过分层进一步将领域逻辑与领域逻辑的解决方案分离。 　　业务逻辑层在体系架构中的位置很关键，它处于数据访问层与表示层中间，起到了数据交换中承上启下的作用。由于层是一种弱耦合结构，层与层之间的依赖是向下的，底层对于上层而言是“无知”的，改变上层的设计对于其调用的底层而言没有任何影响。如果在分层设计时，遵循了面向接口设计的思想，那么这种向下的依赖也应该是一种弱依赖关系。因而在不改变接口定义的前提下，理想的分层式架构，应该是一个支持可抽取、可替换的“抽屉”式架构。正因为如此，业务逻辑层的设计对于一个支持可扩展的架构尤为关键，因为它扮演了两个不同的角色。对于数据访问层而言，它是调用者；对于表示层而言，它却是被调用者。依赖与被依赖的关系都纠结在业务逻辑层上，如何实现依赖关系的解耦，则是除了实现业务逻辑之外留给设计师的任务。 　   
　   
数据层   
　　　数据访问层：有时候也称为是持久层，其功能主要是负责数据库的访问，可以访问数据库系统、二进制文件、文本文档或是XML文档。 　　简单的说法就是实现对数据表的Select，Insert，Update，Delete的操作。如果要加入ORM的元素，那么就会包括对象和数据表之间的mapping，以及对象实体的持久化。   
优缺点   
优点   
　　1、开发人员可以只关注整个结构中的其中某一层； 　　2、可以很容易的用新的实现来替换原有层次的实现； 　　3、可以降低层与层之间的依赖； 　　4、有利于标准化； 　　5、利于各层逻辑的复用。   
缺点   
　　1、降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。 　　2、有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。   
[编辑本段]规则   
　　三层结构的程序不是说把项目分成DAL, BLL, WebUI三个模块就叫三层了, 下面几个问题在你的项目里面： 　　1. UILayer里面只有少量(或者没有)的SQL语句或者存储过程调用, 并且这些语句保证不会修改数据? 　　2. 如果把UILayer拿掉, 你的项目还能在Interface/API的层次上提供所有功能吗? 　　3. 你的DAL可以移植到其他类似环境的项目吗? 　　4. 三个模块, 可以分别运行于不同的服务器吗? 　　如果不是所有答案都为YES, 那么你的项目还不能算是严格意义上的三层程序. 三层程序有一些需要约定遵守的规则： 　　1. 最关键的, UI层只能作为一个外壳, 不能包含任何BizLogic的处理过程 　　2. 设计时应该从BLL出发, 而不是UI出发. BLL层在API上应该实现所有BizLogic, 以面向对象的方式 　　3. 不管数据层是一个简单的SqlHelper也好, 还是带有Mapping过的Classes也好, 应该在一定的抽象程度上做到系统无关 　　4. 不管使用COM+(Enterprise Service), 还是Remoting, 还是WebService之类的远程对象技术, 不管部署的时候是不是真的分别部署到不同的服务器上, 最起码在设计的时候要做这样的考虑, 更远的, 还得考虑多台服务器通过负载均衡作集群 　　所以考虑一个项目是不是应该应用三层/多层设计时, 先得考虑下是不是真的需要? 实际上大部分程序就开个WebApplication就足够了, 完全没必要作的这么复杂. 而多层结构, 是用于解决真正复杂的项目需求的。   
  
与MVC的区别   
　　MVC（模型Model-视图View-控制器Controller）是一种设计模式，我们可以用它来创建在域对象和UI表示层对象之间的区分。 　　同样是架构级别的，相同的地方在于他们都有一个表现层，但是他们不同的地方在于其他的两个层。 　　在三层架构中没有定义Controller的概念。这是我认为最不同的地方。而MVC也没有把业务的逻辑访问看成两个层，这是采用三层架构或MVC搭建程序最主要的区别。当然了。在三层中也提到了Model，但是三层架构中Model的概念与MVC中Model的概念是不一样的，“三层”中典型的Model层是以实体类构成的，而MVC里，则是由业务逻辑与访问数据组成的。