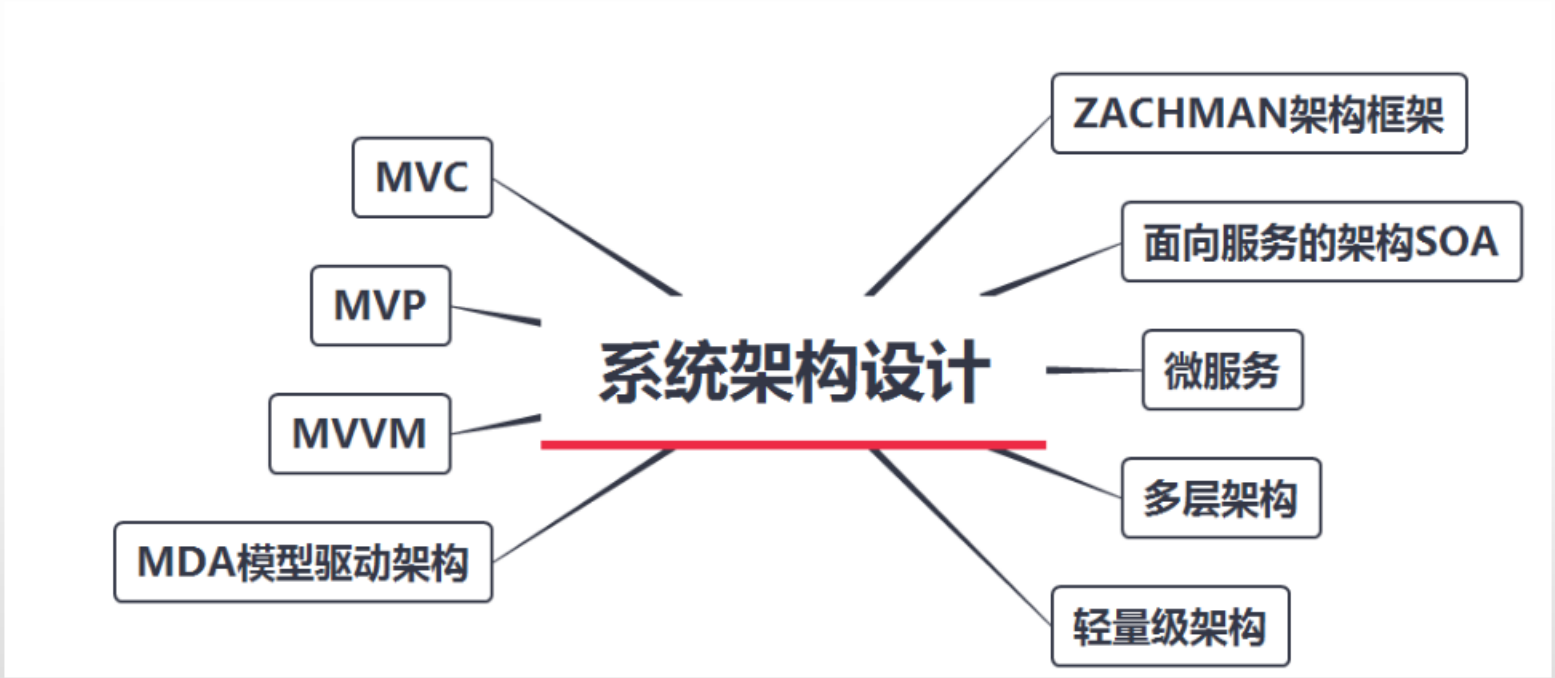
全国计算机技术与软件专业技术资格（水平）考试

架构设计

讲师：邹月平

51CTO学堂

系统架构设计



Zachman架构框架

Zachman框架综合考虑企业业务架构中不同角色的不同观点，提出了一个多视角、多维度的企业架构，是许多大公司用来理解、表述企业信息基础设施的一种可以理解的信息表述，为企业现在以及未来的信息基础设施建设提供蓝图和架构。

——

纵向的功能视图包括目标范围、业务模型、系统模型、技术模型、详细展现和功能系统，横向的关注点包括数据、功能、网络、人员、时间和动机。

Zachman架构框架

元 素	WHAT 数据	HOW 功能	WHERE网络	WHO人员	WHEN 时间	WHY 动机
目标范围规划者	对业务重要的事务列表	业务执行过程列表	业务操作地点列表	对业务重要的组织列表	业务关键事件周期列表	业务目标策略列表
业务模型业务拥有者	语义模型	业务过程模型	物流网络	工作流程模型	总进度表	业务计划
系统模型系统设计者	逻辑数据模型（实体关系图）	应用程序架构、关键数据流程图	分布式系统架构	人机界面架构	处理结构	业务规则模型
技术模型技术设计者	物理数据模型	系统设计（结构图、伪代码）	技术架构（硬件、软件类型）	表示层架构	控制结构	规则模型
详细展现系统开发者	数据定义	软件程序设计	网络架构	安全性架构	定时定义	规则规约
功能系统	转化后的数据	可执行程序	通信网络	业务组织	业务安排	业务战略

典型真题

阅读以下关于系统开发的叙述，在答题纸上回答问题1至问题3。

【说明】

某集团下属煤矿企业委托软件公司开发一套煤炭运销管理系统，该系统属于整个集团企业信息化架构中的业务层，系统针对煤矿企业开发，包括合同管理、磅房管理、质检化验、运费结算等功能。部分业务详细描述如下：

（1）合同管理：合同签订、合同查询、合同跟踪等。

（2）磅房管理：系统可以从所有类型的电子磅自动读数；可以自动从电子磅上读取车辆皮重、毛重，计算出净重；可根据合同内容自动减少相应提货单剩余数量，如果实际发货量超过合同额则拒绝发货。

（3）质检化验：根据过磅单、车号，生成化验分析委托单，而后生成化验分析报告。

（4）运费结算：依据过磅单上的净重、化验单、合同规定，自动计算出原料结算单、运费结算单。

煤矿企业根据集团的工作计划制订本企业的业务计划，煤矿企业根据集团划拨指标和提供的原料生产煤炭，所生产的煤炭交由集团统一管理并销售给客户。软件公司采用Zachman框架对企业业务架构和业务过程进行分析，结果如表2-1所示。

典型真题

表 2-1 煤炭运销管理系统 Zachman 框架分析

	（a）	（b）	（c）	（d）	时间	（e）
目标范围	A11	A12	A13	计划部、财务部、运销部	A15	A16
企业模型	A21	A22	A23	A24	A25	企业业务计划
系统模型	A31	A32	A33	合同界面、过磅界面、质检界面...	企业计划处理结构	A36
技术模型	A41	系统层、数据层 功能层、决策层	系统架构 软硬件配置	A44	A45	A46
详细展现	数据定义 Car、User...	A52	A53	A54	A55	程序逻辑 规格说明
功能系统	A61	A62	A63	A64	A65	A66

典型真题

【问题1】

Zachman框架是什么？请在表2-1中（a）-（e）位置补充企业业务架构中的信息类别。

【问题2】

项目组在该煤炭企业业务架构分析中完成了四项主要工作：数据流图、实体联系图、网络拓扑结构和计划时间表。这四项工作在表2-1中处于什么位置？请用表2-1中的位置编号表示。

【问题3】

据题目所述业务描述，请分别给出表2-1中A11和A23位置应该填入的内容。（物流关系用“→”表示）。

典型真题

参考答案

【问题1】

（1）Zachman框架综合考虑企业业务架构中不同角色的不同观点，提出了一个多视角、多维度的企业架构，是许多大公司用来理解、表述企业信息基础设施的一种可以理解的信息表述，为企业现在以及未来的信息基础设施建设提供蓝图和架构。

（2）　（a）What/数据　（b）How/功能/行为　（c）Where/位置/网络

（d）Who/人员/组织　（e）Why/动机

典型真题

【问题2】

（1） 数据流图：A32　　（2） 实体联系图：A31

（3） 网络拓扑结构：A53　（4） 计划时间表：A25

【问题3】

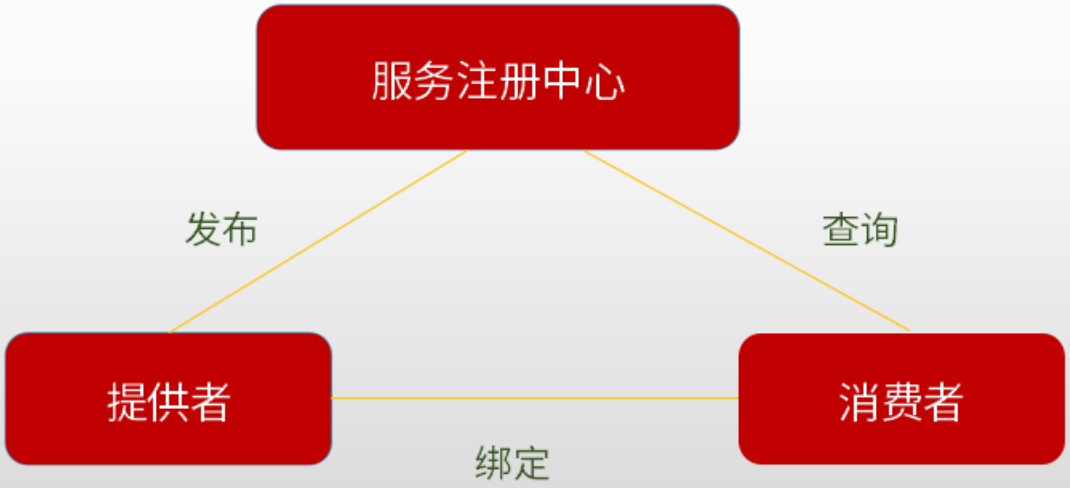
（1） A11项目关键元素：合同/合同管理、过磅/磅房管理、质检/质检化验、结算/运费结算。

（2） A23业务物流网络：煤矿企业←→集团→客户。

面向服务的架构SOA

SOA是一种在计算环境中设计、开发、部署和管理离散逻辑单元（服务）模型的方法。关于服务，一些常见的设计原则有：明确定义的接口、自包含和模块化、粗粒度、松耦合、互操作性。

SOA紧密相关的技术主要有UDDI、WSDL、SOAP和REST等，而这些技术都是以XML为基础而发展起来的。

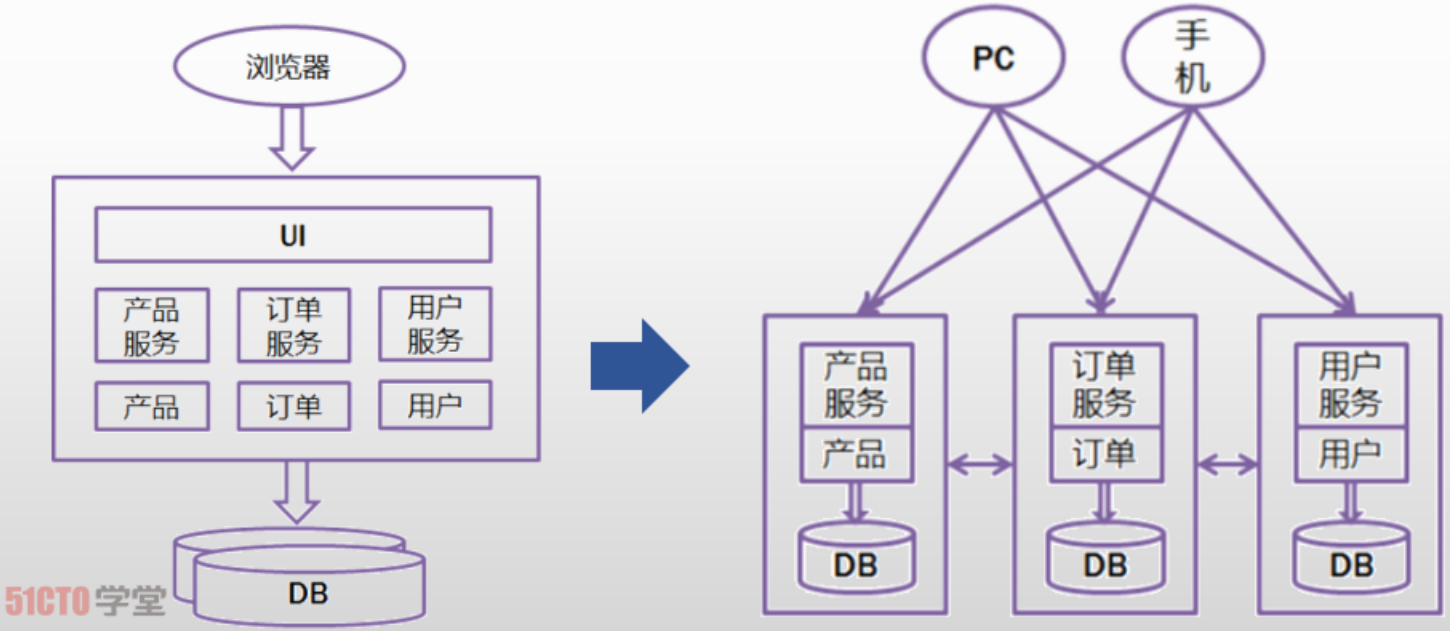


面向服务的架构SOA

UDDI	统一描述、发现和集成，提供了一种服务发布、查找和定位的方法，是服务的信息注册规范，以便被需要该服务的用户发现和使用它。
WSDL	Web服务描述语言是对服务进行描述的语言。
SOAP	简单对象访问协议定义了服务请求者和服务提供者之间的消息传输规范。SOAP用XML来格式化消息，用HTTP来承载消息。通过SOAP，应用程序可以在网络中进行数据交换和远程过程调用RPC。
REST	表述性状态转移是一种只使用HTTP和XML进行基于Web通信的技术，可以降低开发的复杂性，提高系统的可伸缩性。REST提出了如下一些设计概念和准则：（1）网络上的所有事物都被抽象为资源。（2）每个资源对应一个唯一的资源标识。（3）通过通用的连接件接口对资源进行操作。（4）对资源的各种操作不会改变资源标识。（5）所有的操作都是无状态的。

微服务

微服务是一种架构风格，将单体应用划分成一组小的服务，服务之间相互协作，实现业务功能每个服务运行在独立的进程中，服务间采用轻量级的通信机制协作（通常是HTTP/JSON)，每个服务围绕业务能力进行构建，并且能够通过自动化机制独立地部署。



微服务

- 微服务有以下优势：

（1） 通过分解巨大单体式应用为多个服务方法解决了复杂性问题。它把庞大的单一模块应用分解为一系列的服务，同时保持总体功能不变。

（2） 让每个服务能够独立开发，开发者能够自由选择可行的技术，提供API服务。

（3） 微服务架构模式是每个微服务独立的部署。开发者不再需要协调其他服务部署对本服务的影响。这种改变可以加快部署速度。

（4） 微服务使得每个服务独立扩展。你可以根据每个服务的规模来部署满足需求的规模。甚至可以使用更适合于服务资源需求的硬件。

微服务

- 微服务架构带来的挑战如下：

（1） 并非所有的系统都能转成微服务。

（2） 部署较以往架构更加复杂：系统由众多微服务搭建，每个微服务需要单独部署，从而增加部署的复杂度，容器技术能够解决这一问题。

（3） 性能问题：由于微服务注重独立性，互相通信时只能通过标准接口，可能产生延迟或调用出错。

（4） 数据一致性问题：作为分布式部署的微服务，在保持数据一致性方面需要比传统架构更加困难。

多层架构

二层 C/S 结构为单一服务器且以局域网为中心，所以难以扩展至大型企业广域网或 Internet；软、硬件的组合及集成能力有限；它的缺点主要有：

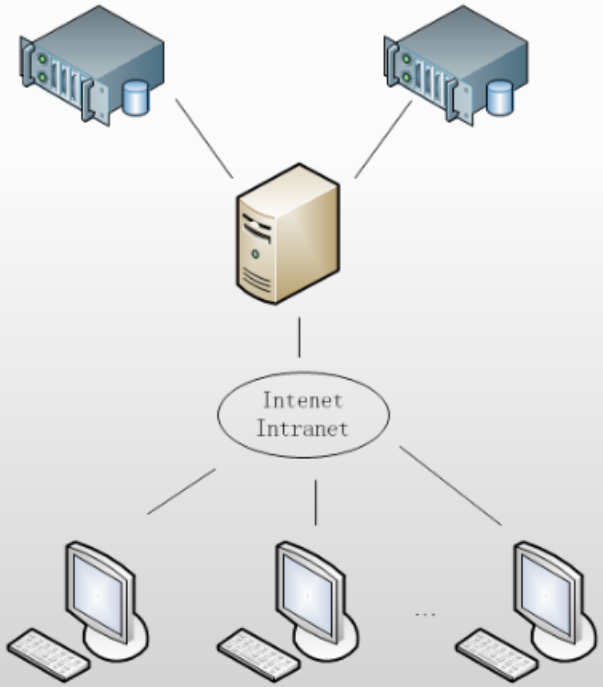
（1）服务器的负荷太重，难以管理大量的客户机，系统的性能容易变坏。

（2）数据安全性不好。因为客户端程序可以直接访问数据库服务器，那么，在客户端计算机上的其他程序也可想办法访问数据库服务器，从而使数据库的安全性受到威胁。

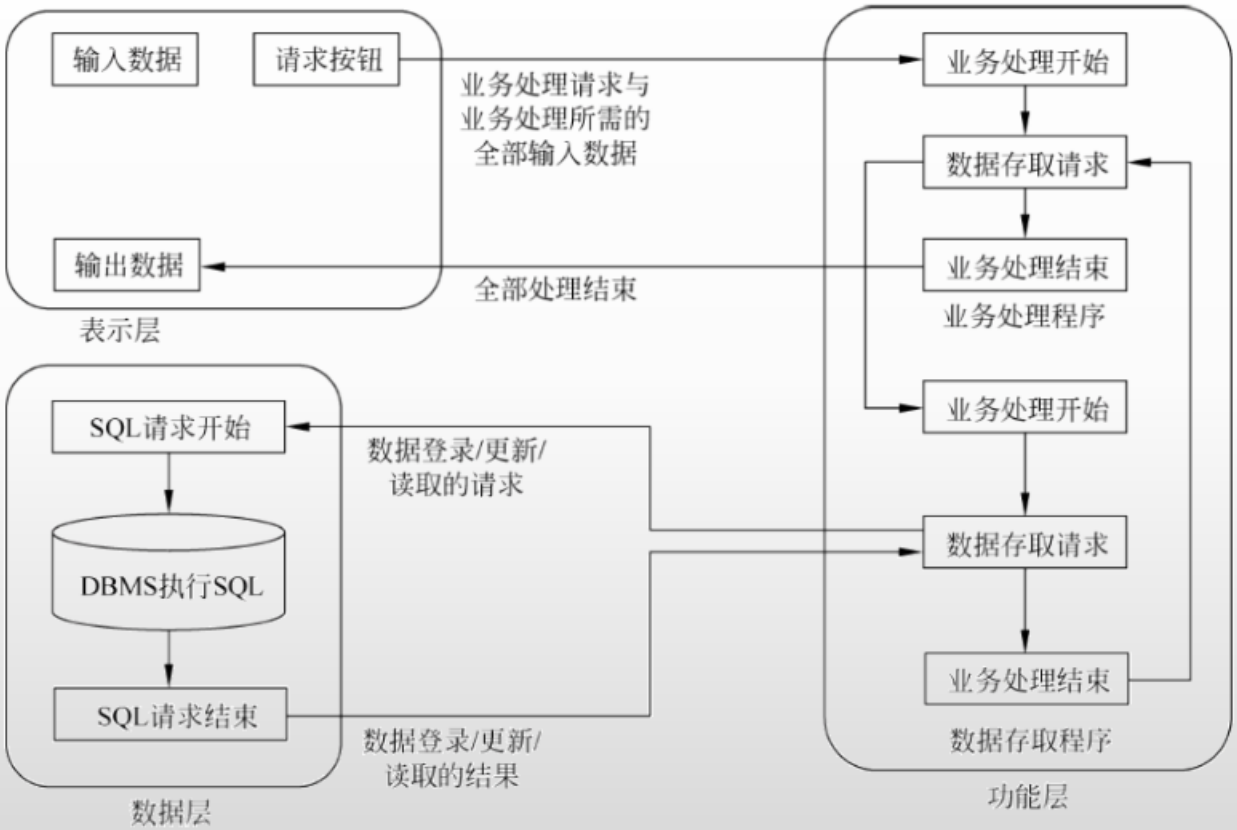
多层架构

与二层C/S架构相比，在三层C/S架构中，增加了一个应用服务器。可以将整个应用逻辑驻留在应用服务器上，而只有表示层存在于客户机上。这种客户机称为瘦客户机。三层C/S架构将应用系统分成表示层、功能层和数据层三个部分。

层次	功能
表示层	用户接口，检查用户输入的数据，显示输出数据。
功能层	业务逻辑层，是将具体的业务处理逻辑编入程序中。
数据层	对DBMS进行管理和控制。



多层架构



多层架构

- 与传统的二层架构相比，三层C/S架构具有以下优点：

（1）允许合理地划分三层的功能，使之在逻辑上保持相对独立性，从而使整个系统的逻辑结构更为清晰，能提高系统的可维护性和可扩展性。

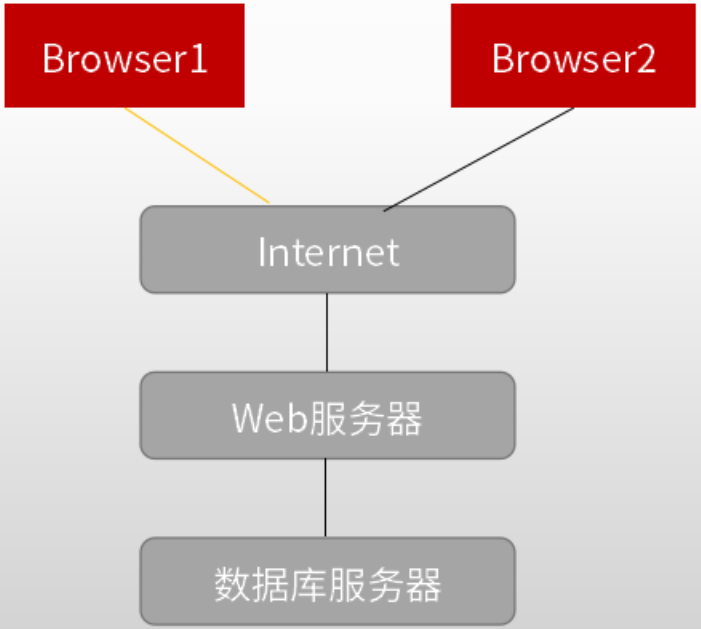
（2）允许更灵活、有效地选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层，并且这些平台和各个组成部分可以具有良好的可升级性和开放性。

（3）系统的各层可以并行开发，各层也可以选择各自最适合的开发语言，使之能并行且高效地进行开发，达到较高的性能价格比。对每一层的处理逻辑的开发和维护也会更容易些。

（4）利用功能层可以有效地隔离表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，这就为严格的安全管理奠定了坚实的基础。

多层架构

BS浏览器/服务器（Browser/Server, B/S）架构是三层C/S架构的一种实现方式，其具体结构为“浏览器/Web服务器/数据库服务器”。B/S架构利用WWW浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。



轻量级架构

➢表现层

用户界面的逻辑位于最顶层。表现层负责把用户要求的业务逻辑处理结果以可视化的友好的方式返回给用户，并提供接受用户命令的接口和表现层页面控制逻辑的代码。

➢业务逻辑层

业务逻辑层负责处理问题领域的业务规则和根据用户需求进行的业务处理以满足用户的功能需求。通常情况下，业务逻辑层处理使用的实体对象由持久层提供。

➢ 持久层

数据通过持久层进行持久化。所谓持久化，即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。

轻量级架构

持久层的设计，使得业务逻辑层只需要负责业务逻辑的实现，而把对数据的操作交给了持久层。持久层对数据及对数据操作的封装有以下几个优点：

（1）屏蔽数据库平台的变化对业务逻辑层的影响。当数据库变化时，只需修改持久层操作数据库的代码，而持久层提供给业务逻辑的对象模型没有变化，从而避免了业务逻辑的修改。

（2）通过持久层的封装处理，可以在持久层实现支持多种数据库平台，而对业务逻辑层提供统一的接口。

（3）代码可重用性高，能够完成所有的数据库访问操作。

通过持久层的设计，将复杂的业务逻辑和数据逻辑分离，降低系统的耦合程度，从而在开发时更明确地进行分工，维护工作也更容易进行，系统的体系结构也变得更加清晰。

轻量级架构

- SSH：指的是Struts2(做前端控制器)，Spring(管理各层的组件)，Hibernate(负责持久化层)

- SSM：指的是SpringMVC(做前端控制器)，Spring(管理各层的组件)，Mybatis(负责持久化层)

所在分层	SSH	SSM
页面层（View）	JSP	JSP
控制器层（Controler）	Struts2	SpringMVC
业务层（Service）	java	java
持久层（DAO）	Hibernate	MyBatis
数据库层（DB）	MySQL/Oracle	MySQL/Oracle
组件管理层（Bean）	Spring	Spring

轻量级架构

- Hibernate与Mybatis区别：

①开发方面：Hibernate开发中，sql语句已经被封装，直接可以使用；Mybatis 属于半自动化，sql需要手工完成。

②sql优化方面：对复杂查询的sql语句进行人工调优的时候，Mybatis更方便。

③可移植性方面：Hibernate使用时自动生成相应的sql语句，因此具备良好的数据库移植性，而 Mybatis 中手动编写的sql语句需要针对不同厂商的数据库进行修改。

MVC

- 模型：执行业务流程（不包括输入输出），存储业务数据。模型不依赖于视图和控制器，提高了架构的灵活性。
- 视图：展示模型中的数据，用户的同一份数据可以通过不同的视图以不同的方式展示。视图必须了解模型中的数据结构，对模型有很强的依赖性，但是模型对于视图则没有依赖性。
- 控制器：把模型接收的事件和用户输入的数据转化为对模型方法的调用。控制器对用户的行为作出解释，并决定调用模型的哪个方法。

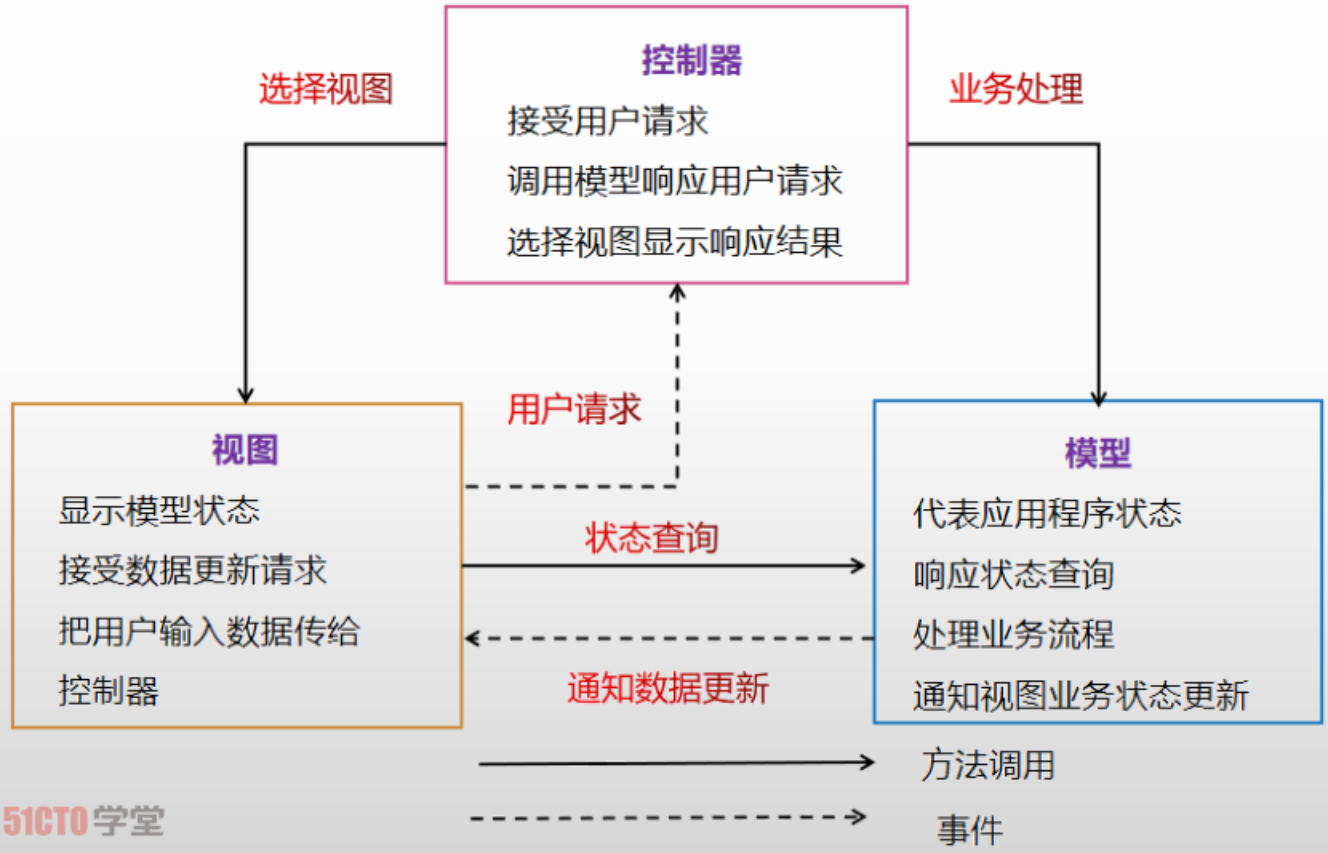
使用MVC模式来设计表现层，可以有以下的优点。

（1）允许多种用户界面的扩展。在MVC模式中，视图与模型没有必然的联系，都是通过控制器发生关系，这样如果要增加新类型的用户界面，只需要改动相应的视图和控制器即可，而模型则不需发生改动。

（2）易于维护。控制器和视图可以随着模型的扩展而进行相应的扩展，只要保持一种公共的接口，控制器和视图的旧版本也可以继续使用。

（3）功能强大的用户界面。用户界面与模型方法调用组合起来，使程序的使用更清 晰，可将友好的界面发布给用户。

MVC


控制器
接受用户请求
调用模型响应用户请求
选择视图显示响应结果
选择视图
业务处理
用户请求
视图
显示模型状态
接受数据更新请求
把用户输入数据传给
控制器
状态查询
通知数据更新
模型
代表应用程序状态
响应状态查询
处理业务流程
通知视图业务状态更新
方法调用
事件

MVP

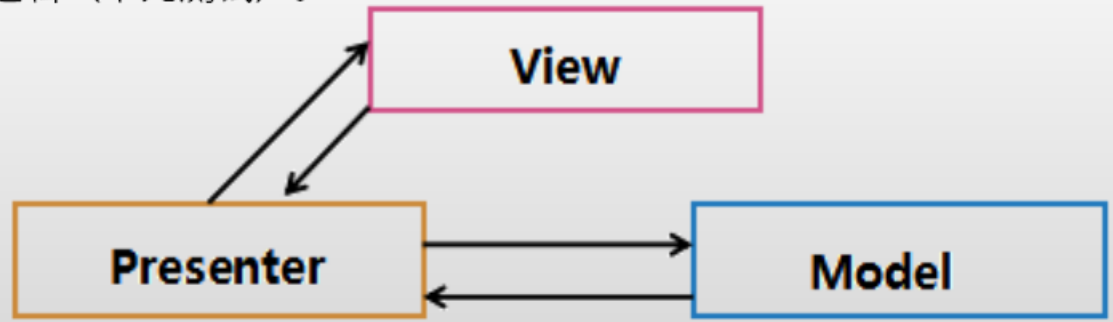
- MVP 的优点包括：

（1）低耦合。模型与视图完全分离，可以修改视图而不影响模型。

（2）可以更高效地使用模型，因为所有的交互都发生在一个地方—Presenter 内部。

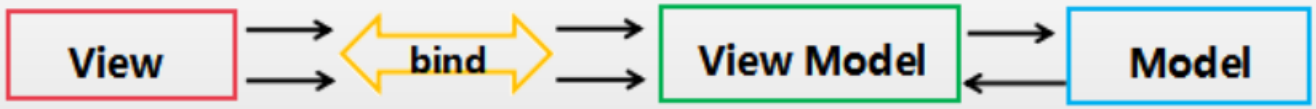
（3）复用性好。可以将一个 Presenter 用于多个视图，而不需要改变 Presenter 的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁。

（4）可测试性好。如果把逻辑放在 Presenter 中，就可以脱离用户接口来测试这些逻辑（单元测试）。



MVVM

MVVM是由MVP进化而来，MVVM模式基本上与MVP相同，只是把MVP中的P变成了VM，即ViewModel，MVVM中的数据可以实现双向绑定，当Model变化时，View-Model会自动更新，View也会自动变化。很好做到数据的一致性，不用担心，在模块的这一块数据是这个值，在另一块就是另一个值了。所以 MVVM模式有些时候又被称作：model-view-binder模式。因此MVVM框架比较适合逻辑复杂的前端项目，比如一些管理系统等。

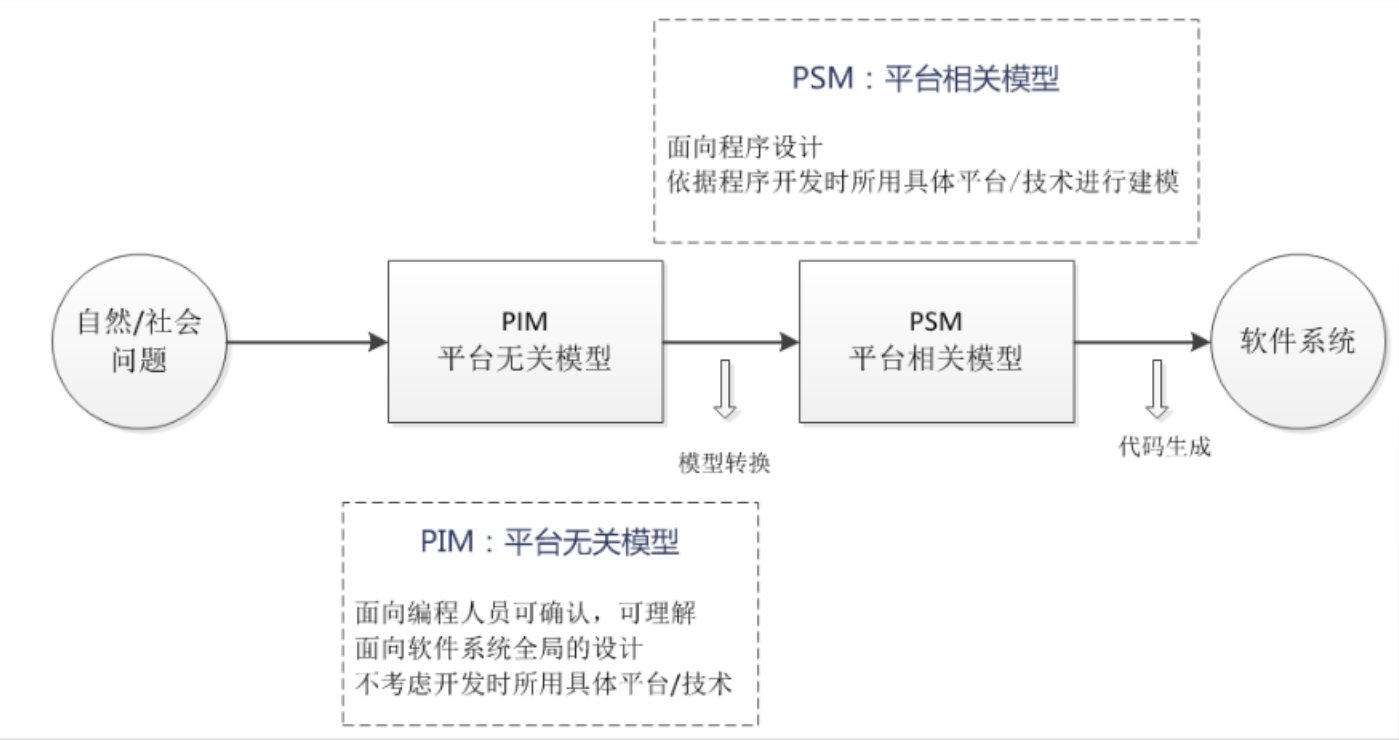


MDA模型驱动架构

MDA(Model Driven Architecture)是模型驱动架构，它是由OMG定义的一个软件开发框架。它是一种基于UML以及其他工业标准的框架，支持软件设计和模型的可视化、存储和交换。

基于 MDA 的软件开发方法的主要过程是抽象出与实现技术无关、完整描述业务功能的核心平台无关模型（Platform Independent Model，PIM），然后针对不同实现技术制定多个转换规则，通过这些转换规则及辅助工具将 PIM转换成与具体实现技术相关的平台相关模型（Platform Specific Model，PSM），最后将经过充实的 PSM转换成代码。

MDA模型驱动架构



MDA模型驱动架构

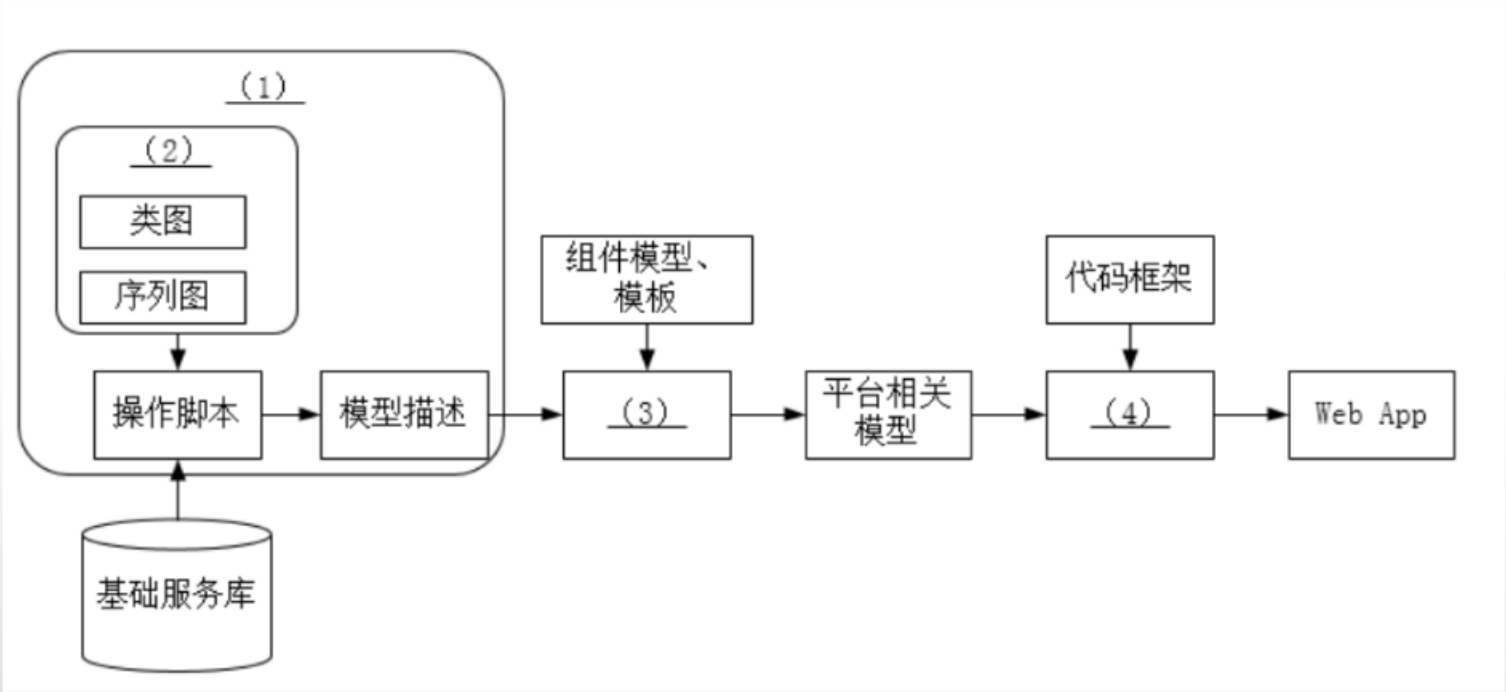
MDA的优点：

（1）使用平台无关的建模语言来搭建平台无关的模型PIM，然后根据特定平台和实现语言的映射规则，将PIM 转换以生成平台相关的模型PSM，最终生成应用程序代码和测试框架。因此MDA方法可移植性比较好。

（2）MDA方法中提供了模型转换标准，以及对象约束语言，工具厂商可以开发自动化的工具，开发人员只需关注于业务建模，开发PIM。从PIM到最后面向具体技术平台的可执行的应用程序，都由自动化的MDA工具来解决，很好地实现了平台互操作性。

（3）在MDA中代码是由模型生成的，可以保证文档和代码的一致性。

MDA模型驱动架构



MDA模型驱动架构

（1）平台无关模型（PIM）
（2）UML建模
（3）模型变换（映射）
（4）模型生成源码

技术成就梦想

51CTO学堂