

NEW PLAN

软考高级架构师

• — 段 新 征 程 •





层次架构风格



两层C/S架构：客户端和服务端都有处理功能，现在已经不常用，原因有：开发成本较高、客户端程序设计复杂、信息内容和形式单一、用户界面风格不一、软件移植困难、软件维护和升级困难、新技术不能轻易应用、安全性问题、服务器端压力大难以复用。



三层C/S架构：**将处理功能独立出来**，表示层和数据层都变得简单。表示层在客户机上，功能层在应用服务器上，数据层在数据库服务器上。即将两层C/S架构中的数据从服务器中独立出来了。其优点下面四点：

- **各层在逻辑上保持相对独立**，整个系统的逻辑结构更为清晰，能提高系统和软件的可维护性和可扩展性；
- 允许灵活有效的选用相应的平台和硬件系统，具有良好的可升级性和开放性；
- 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；
- 功能层有效的隔离表示层与数据层，为严格的安全管理奠定了坚实的基础，**整个系统的管理层次也更加合理和可控制**。



三层C/S架构**设计的关键在于各层之间的通信效率**，要慎重考虑三层间的通信方法、通信频度和数据量，否则即使分配给各层的硬件能力很强，性能也不高。



层次架构风格



三层B/S架构： 是三层C/S架构的变种，将客户端变为用户客户端上的浏览器，将应用服务器变为网络上的WEB服务器，又称为0客户端架构，虽然不用开发客户端，但有很多缺点：

- 使用浏览器作为客户端的话安全性难以控制；
- 在数据查询等响应速度上，要远远低于C/S架构，因为C/S架构有部分数据存储在本地；
- 数据提交一般以页面为单位，数据的动态交互性不强。

混合架构风格



- 内外有别模型：企业内部使用C/S,外部人员访问使用B/S。
- 查改有别模型：采用B/S查询，采用C/S修改。
- 混合架构实现困难，且成本高



层次架构风格



富互联网应用RIA：弥补三层B/S架构存在的问题，**RIA是一种用户接口**，比用HTML实现的接口更加健壮，且有可视化内容，本质还是网站模式，其优点如下：

- **RIA结合了C/S架构反应速度快、交互性强的优点与B/S架构传播范围广及容易传播的特性；**
- **RIA简化并改进了B/S架构的用户交互；**
- **数据能够被缓存在客户端**，从而可以实现一个比基于HTML的响应速度更快且数据往返于服务器的次数更少的用户界面。
- **本质还是O客户端**，借助于高速网速实现必要插件在本地的快速缓存，增强页面对动态页面的支持能力，典型的如小程序。



层次架构风格-MVC架构



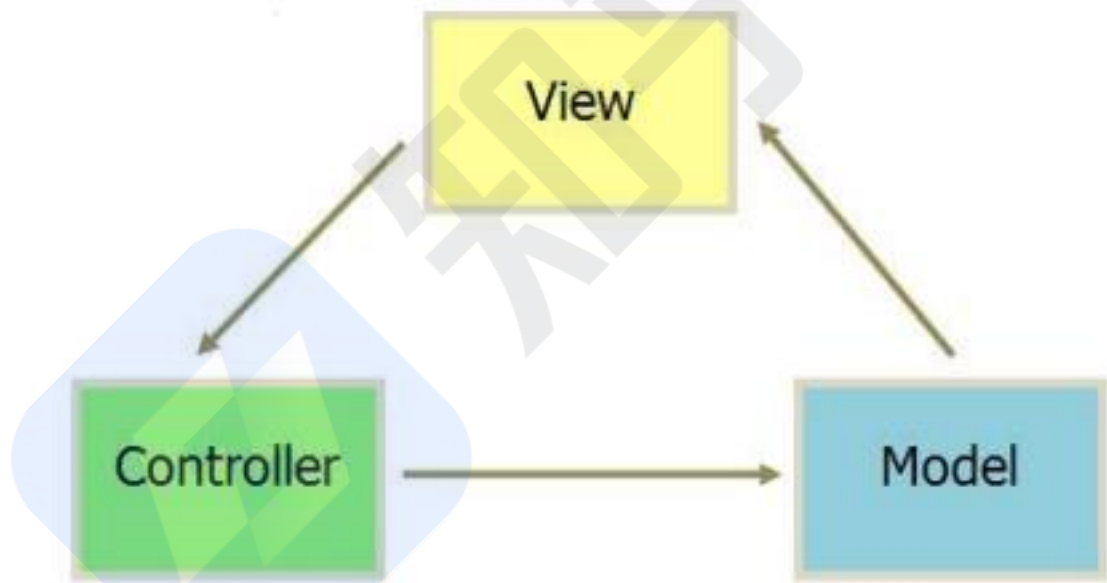
控制器(Controller)：是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入,并向模型发送数据。



模型(Model)：是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在数据库中存取数据。模型表示业务数据和业务逻辑。



视图(View)：是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。是用户看到并与之交互的界面。视图向用户显示相关的数据，并能接收用户的输入数据，但是它并不进行任何实际的业务处理：





层次架构风格-MVP架构

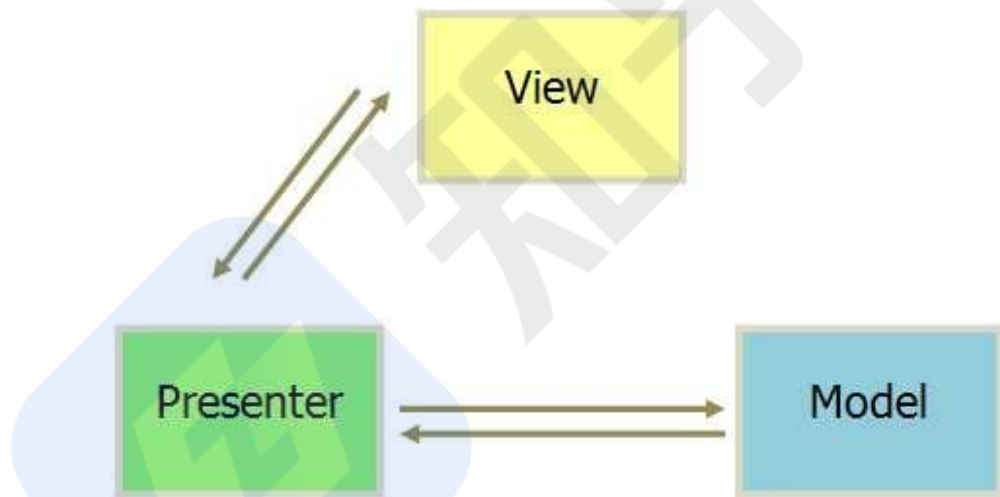


MVP是把MVC中的Controller换成了Presenter(呈现), 目的就是为了完全切断View跟Model之间的联系, 由Presenter充当桥梁, 做到View-Model之间通信的完全隔离。



MVP特点:

- M、V、P之间双向通信。
- View 与Model不通信, 都通过Presenter传递。Presenter完全把Model和View进行了分离, 主要的程序逻辑在Presenter里实现。
- View非常薄, 不部署任何业务逻辑, 称为”被动视图”(PassiveView), 即没有任何主动性, 而Presenter非常厚, 所有逻辑都部署在那里。
- Presenter与具体的View是没有直接关联的, 而是通过定义好的接口进行交互, 从而使得在变更View时候可以保持Presenter的不变, 这样就可以重用



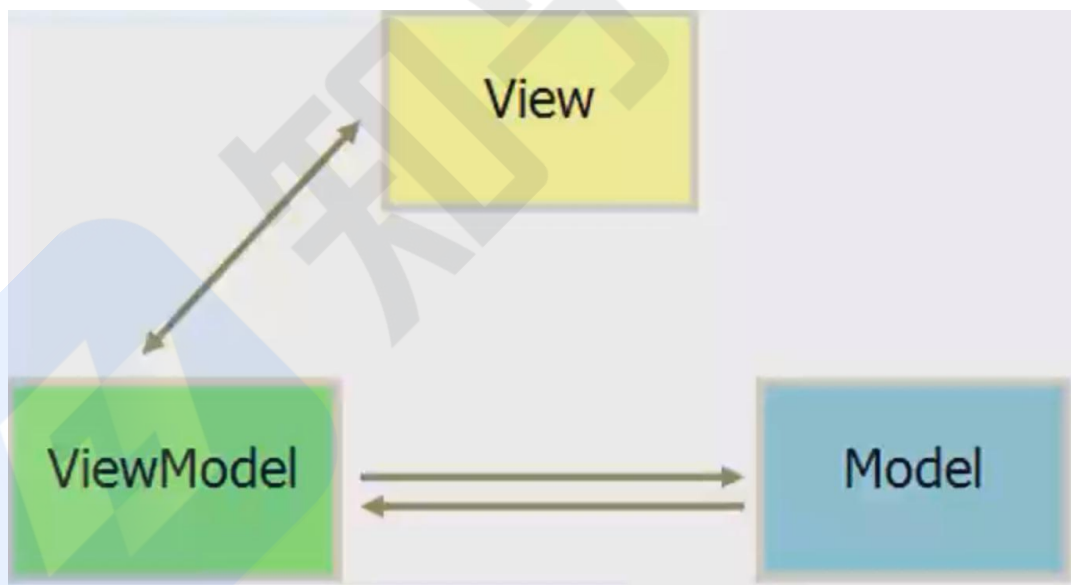


层次架构风格-MVVM架构



MVVM: MVVM模式和MVC模式类似，主要目的是分离视图(View)和模型(Model)，实现双向绑定，有几点优点：

- 低耦合，视图(View)可以独立于Model变化和修改，一个ViewModel可以绑定到不同的“View”上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。
- 可重用性，可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑。
- 独立开发，开发人员可以专注于业务逻辑和数据的开发(ViewModel)，设计人员可以专注于页面设计。
- 可测试，界面向来是比较难于测试的，而现在测试可以针对ViewModel来写。





面向服务的架构风格



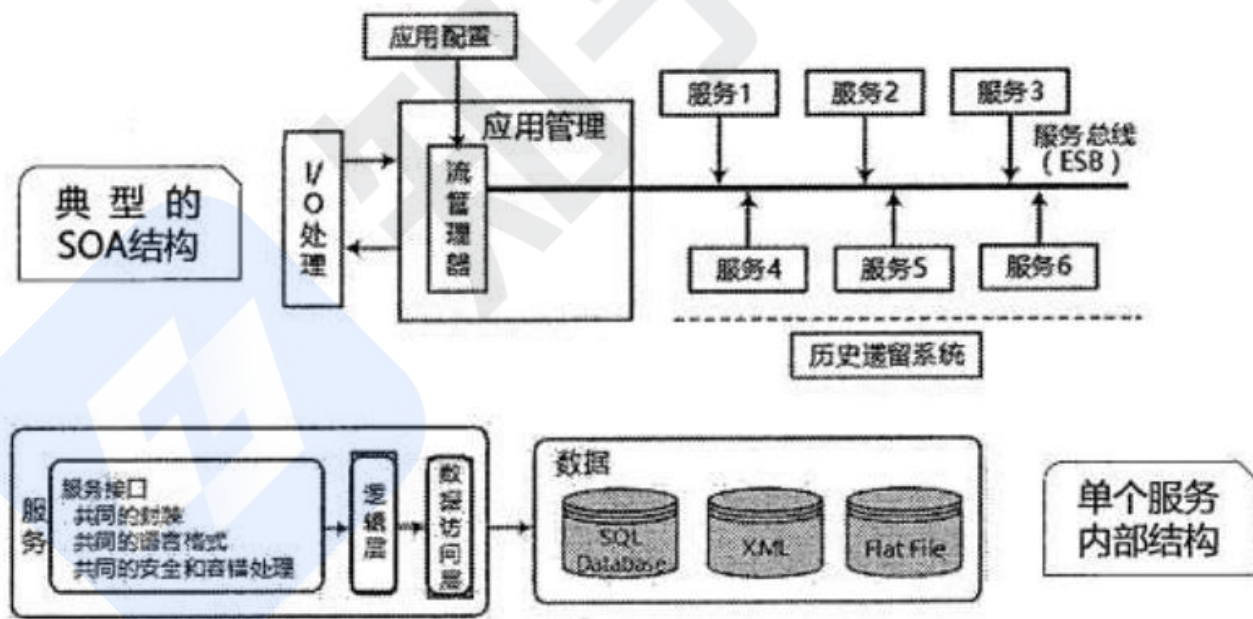
SOA是一种**粗粒度、松耦合服务架构**，服务之间通过简单、精确定义接口进行通信，不涉及底层编程接口和通信模型。



在SOA中，**服务是一种为了满足某项业务需求的操作、规则等的逻辑组合**，它包含一系列有序活动的交互，为实现用户目标提供支持。



SOA并不仅仅是一种开发方法，还具有管理上的优点，**管理员可直接管理开发人员所构建的相同服务**。多个**服务通过企业服务总线(ESB)提出服务请求**，由应用管理来进行处理，如下：





面向服务的架构风格



实施SOA的关键目标是实现企业IT资产重用的最大化，在实施SOA过程中要牢记以下特征：可从企业外部访问、随时可用(服务请求能被及时响应)、粗粒度接口(粗粒度提供一项特定的业务功能，而细粒度服务代表了技术构件方法)、服务分级、松散耦合(服务提供者和服务使用者分离)、可重用的服务及服务接口设计管理、标准化的接口(WSDL、SOAP、XML是核心)、支持各种消息模式、精确定义的服务接口。



从基于对象到基于构件再到基于服务，架构越来越松散耦合，粒度越来越粗，接口越来越标准



基于服务的构件与传统构件的区别有四点：

- 服务构件粗粒度，传统构件细粒度居多；
- 服务构件的接口是标准的，主要是WSDL接口，而传统构件常以具体API形式出现；
- 服务构件的实现与语言是无关的，而传统构件常绑定某种特定的语言；
- 服务构件可以通过构件容器提供QoS的服务，而传统构件完全由程序代码直接控制。



面向服务的架构风格



SOA中应用的关键技术如下表。



UDDI：是一套基于WEB的、分布式的、为WebService提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的WebService注册，以使别的企业能够发现的访问协议的实现标准，**用于WEB服务注册统一描述、发现及集成。**



WSDL (Web Service描述语言)：**将Web服务描述定义为一组服务访问点**，客户端可以通过这些服务访问点对包含面向文档信息或面向过程调用的服务进行访问(类似远程调用)，**用于描述服务。**



SOAP(简单对象访问协议)：是用于交换XML编码信息的轻量级协议，**用于传递信息。**



XML (可扩展标记语言)：是WebService平台中表示数据的基本格式，**用于数据交换。**

功能	协议
发现服务	UDDI DISCO
描述服务	WSDL XML Schema
消息格式层	SOAP REST
编码格式层	XML (DOM, SAX)
传输协议层	HTTP、TCP/IP、SMTP等



面向服务的架构风格



SOA的三种实现方式: **WEB Service**、**服务注册表**和**企业服务总线ESB**

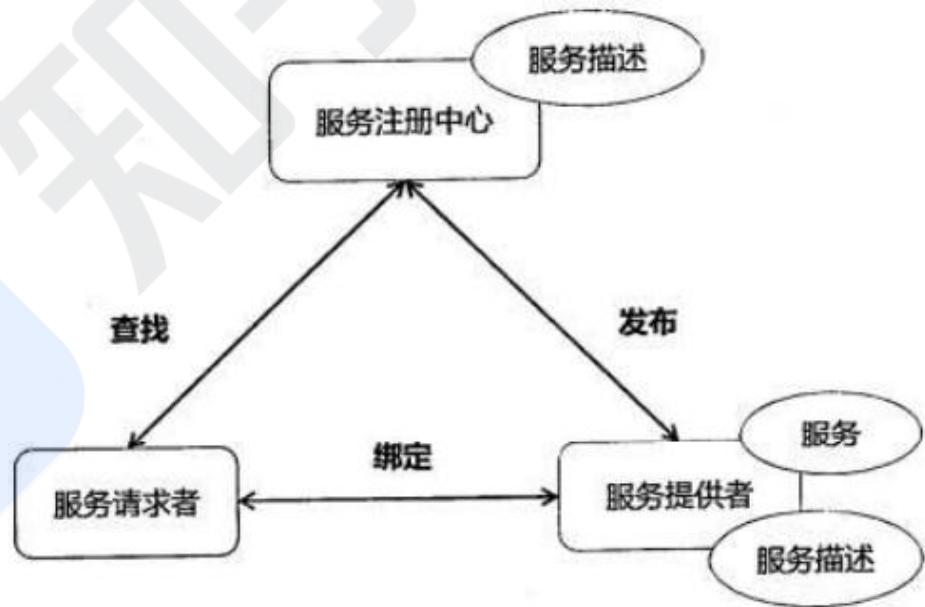


WEB Service: **服务提供者**、**服务注册中心**(中介, 提供交易平台, 可有可无)、**服务请求者**。服务提供者将服务描述发布到服务注册中心, 供服务请求者查找, 查找到后, 服务请求者将绑定查找结果。如图



服务注册表:

- **服务注册:** 应用开发者(服务提供者)在注册表中公布服务的功能。
- **服务位置:** 服务使用者(服务应用开发者), 帮助他们查询注册服务, 寻找符合自身要求的服务。
- **服务绑定:** 服务使用者利用检索到的服务接口来编写代码, 所编写的代码将与注册的服务绑定, 调用注册的服务, 以及与它们实现互动。





面向服务的架构风格



企业服务总线ESB: 简单来说是一根管道, 用来连接各个服务节点。
ESB的存在是为了集成基于不同协议的不同服务, ESB做了消息的转化、解释以及路由的工作, 以此来让不同的服务互联互通。



包括: 客户端(服务请求者)、基础架构服务(中间件)、核心集成服务(提供服务)。



SB特点:

- SOA的一种实现方式, ESB在面向服务的架构中起到的是总线作用, 将各种服务进行连接与整合;
- 描述服务的元数据和服务注册管理;
- 在服务请求者和提供者之间传递数据, 以及对这些数据进行转换的能力, 并支持由实践中总结出来的一些模式如同步模式、异步模式等;
- 发现、路由、匹配和选择的能力, 以支持服务之间的动态交互, 解耦服务请求者和提供者。高级一些的能力, 包括对安全的支持、服务质量保证、可管理性和负载平衡等。





面向服务系统构建过程中，()用于实现Web服务的远程调用，()用来将分散的、功能单一的Web服务组织成一个复杂的有机应用。

- A.UDDI(Universal Description,Discovery and Integration)
- B.WSDL (Web Service Description Language)
- C.SOAP (Simple Object Access Protocol)
- D.BPEL (Business Process Execution Language)

- A.UDDI (Universal Description,Discovery and Integration)
- B.WSDL (Web Service Description Language)
- C.SOAP (Simple Object Access Protocol)
- D.BPEL (Business Process Execution Language)



解析： UDDI (Universal Description, Discovery &Integration)， UDDI用于Web服务注册和服务查找;WSDL (Web Service Description Language),WSDL用于描述web服务的接口和操作功能;SOAP(SimpleObject Access Protocol) ,SOAP为建立Web服务和请求之间的通信提供支持。 BPEL (Business Process Execution Language For Web Services)翻译成中文的意思是面向Web服务的业务流程执行语言， 也有的文献简写成BPEL4Ws， 它是一种使用Web服务定义和执行业务流程的语言。使用BPEL， 用户可以通过组合、编排和协调web服务自上而下地实现面向服务的体系结构(SOA)。 BPEL提供了一种相对简单易懂的方法， 可将多个Web服务组合到一个新的复合服务(称作业务流程)中。

T H E E N D

功不唐捐，玉汝于成！

• 开 启 新 征 程 •

