架构风格与模式

薛安振

架构风格

Java

什么是架构风格

一组原则。你可以把它看成是一组为系统家族提供抽象框架的粗粒度模式架构风格能改进分块，还能为频繁出现的问题提供解决方案，以此促进设计重用。

分类	架构风格
通信	SOA，消息总线，管道和过滤器
部署	客户端/服务器，三层架构，N层架构
领域	领域模型，网关
交互	分离表现层
结构	基于组件的架构，面向对象，分层架构

架构风格	描述
客户端-服务器	将系统分为两个应用，其中客户端向服务器发送服务请求。
基于组件的架构	把应用设计分解为可重用的功能、逻辑组件，这些组件的位置相互透明，只暴露明确定义的通信接口。
分层架构	把应用的关注点分割为堆栈组（层）。
消息总线	指接收、发送消息的软件系统，消息基于一组已知格式，以便系统无需知道实际接收者就能互相通信。
N层/三层架构	用与分层风格差不多一样的方式将功能划分为独立的部分，每个部分是一个层，处于完全独立的计算机上。
面向对象	该架构风格是将应用或系统任务分割成单独、可重用、可自给的对象，每个对象包含数据，以及与对象相关的行为。
分离表现层	将处理用户界面的逻辑从用户界面（UI）视图和用户操作的数据中分离出来。
面向服务架构（SOA）	是指那些利用契约和消息将功能暴露为服务、消费功能服务的应用。

什么是架构模式

由于解决系统构件定义、关系及系统运行方式的可复用方法集合。

MVC架构

分层架构

元模型架构

.管道-过滤器架构

微内核架构

云计算架构

REST架构

SOA架构

……

分层架构简介

分层架构：分层架构是使用最多的架构模式，通过分层使各个层的职责更加明确，通过定义的接口使各层之间通讯，上层使用下层提供的服务。分层分为：严格意义上的分层，一般意义的分层。严格意义的分层是n+1层使用n层的服务。而一般意义的分层是上层能够使用它下边所有层的服务。领域驱动设计的分层定义：UI层，UI控制层，服务层，领域层，基础设施层。

MVC架构简介

MVC架构：MVC架构相信做软件的都听说，主要是为了让软件的各部分松耦合，现在好多根据MVC思想构建的框架如：Spring MVC,Structs2,ASP.Net MVC等。MVC是Model View Control的简写，他的原理是什么那，比如拿web来举例吧。当一个web请求来了以后View接收这个请求，随即把请求转发给Control进行处理，Control通过分析请求的类型等信息决定加载哪些Model，当Model加载完成以后Control通知Model已经加载完毕，这是View就去读取Model数据进行显示自己。MVC还有一个衍生架构叫MVP,因为MVC的View跟Control和Model都有耦合关系所以为了解除View和Model之间的关系，View不直接读取Model而是通过Control来转发View需要的数据。还有一个衍生架构叫MVVP,就是增加了一个ViewControl的层，用来辅助视图的生成，这样View的功能更加简单只是用来显示不包含其它的功能，而且有了ViewControl使多视图或替换视图很方便。MVP微软的WPF就是使用这种架构。

微内核架构简介

微内核架构：微内核架构就是做一个稳定通用的内核，也就是给软件设计一个强劲的心脏。如果需要更多功能通过在内核外部再封装一层对软件进行扩充，微内核提供基本的接口供外部调用，这些接口一定要通用，并且提供事件的机制告诉外部内部发生的事件，这样就是内核与外部完全隔离。微软操作系统就是按照微内核设计的。我之前做了一个Gis组件当初思想也是这个样子的，但是当初不知道还有微内核架构，有了对微内核的深入理解会进一步完善那个Gis组件。

元模型架构简介

元模型架构：元模型架构就是有元数据支撑的架构，现在使用的也很广泛，比如：ORM,.Net 类的设计等都是元数据支持的。元数据有自我描述性比如ORM会描述类对应数据库中的表属性对应数据库里的字段，还有IOC类中的引用需要注入哪个类等等都会通过元数据的形式实现。IOC框架通过解析元数据信息使注入和被注入类只通过接口依赖，这样替换注入类很方便。元数据架构是很灵活的架构，可发展空间非常大，元数据架构会经常用反射技术或者动态代码生成技术。我之前做了一个ORM就是用到的元数据架构，我还想给ORM添加依赖注入面向切面编程等特性都很方便的。

管道过滤器架构简介

管道-过滤器架构：这个模式就像是工厂的流水线，生产原料通过流水线经过很多环节进行处理变成产品。软件也是一样的，网络OSI7层就是消息通过管道内部的很多步处理对消息进行加工过滤转换。再举一个例子，两家企业需要信息交换，但是企业的信息格式和描述规则都不相同，如果想达到交换必须经过处理，所以我们就得用管道过滤器模式，通过管道过滤器模式信息进入管道我们会在管道里添加各种处理功能，比如：数据验证，信息加密，信息解密，信息压缩，信息解压缩，格式转换等功能，对消息进行处理以符合我们要求的消息格式，而且如果需要添加一个新的处理只要把处理的功能插入到管道中即可，这样达到最大的灵活性。应用此模式的有：**ASP.net**请求模型，**Spring** 对象构造，**Structs** 数据请求等。

REST架构简介

REST是英文Representational State Transfer的缩写，中文翻译为“表述性状态转移”，REST在原有的web架构上增加了三个新规范：

- 统一接口

REST世界里，网络上所有的事物都被抽象为资源，而REST就是通过通用的链接器接口对资源进行操作。这样设计的好处是保证系统提供的服务都是解耦的，改善了系统的交互性和可重用性。并且REST针对Web的常见情况做了优化，使得REST接口被设计为可以高效的转移大粒度的超媒体数据。

- 分层系统

分层系统规则的加入提高了各种层次之间的独立性，为整个系统的复杂性设置了边界，通过封装遗留的服务，使新的服务器免受遗留客户端的影响，这也就提高了系统的可伸缩性。

- 按需代码

REST允许对客户端功能进行扩展。比如，通过下载并执行applet或脚本形式的代码，来扩展客户端功能。但这在改善系统可扩展性的同时，也降低了可见性。所以它只是REST的一个可选的约束。

SOA架构简介

SOA服务具有平台独立的自我描述XML文档。Web服务描述语言（WSDL， Web Services Description Language）是用于描述服务的标准语言。

SOA 服务用消息进行通信，该消息通常使用XML Schema来定义。消费者和提供者或消费者和服务之间的通信多见于不知道提供者的环境中。服务间的通讯也可以看作企业内部处理的关键商业文档。

在一个企业内部，SOA服务通过一个扮演目录列表（directory listing）角色的登记处（Registry）来进行维护。应用程序在登记处（Registry）寻找并调用某项服务。统一描述，定义和集成（UDDI， Universal Description， Definition， and Integration）是服务登记的标准。

每项SOA服务都有一个与之相关的服务品质（QoS， quality of service）。QoS的一些关键元素有安全需求（例如认证和授权），可靠通信（译注：可靠消息是指，确保消息“仅且仅仅”发送一次，从而过滤重复信息。），以及谁能调用服务的策略。

风格与模式的细微差别

- 架构风格是系统主要的、组织性的设计。
- 架构模式从子系统或模块、及其之间的关系层次上描述了粗粒度的解决方案。
- 系统隐喻则更为概念化，比起软件工程概念，它更多地涉及现实世界的概念。

基本上，你可以认为风格=模式！

准确的说，风格是模式的外在表现！

分层架构风格

分层架构风格

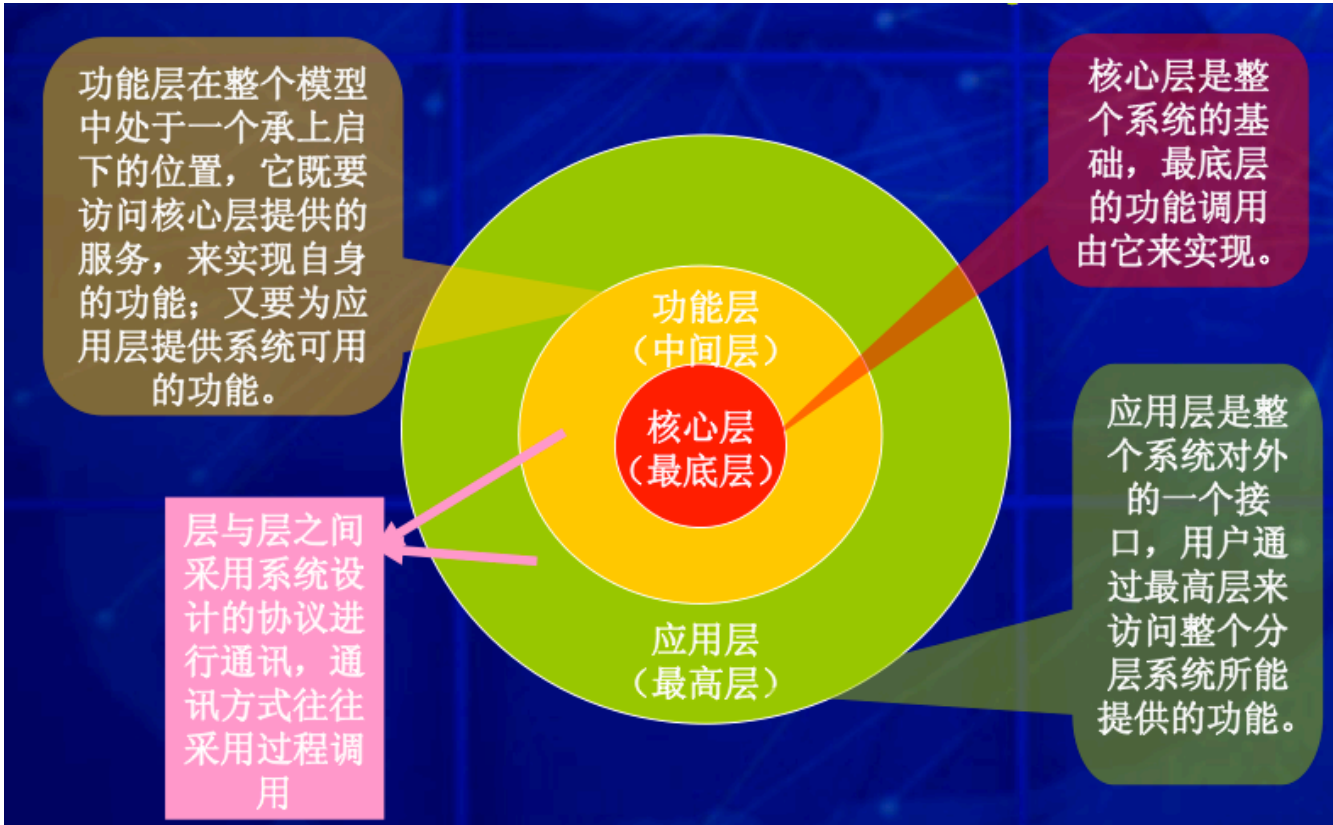
1.层次系统
2.分层模式
3.分层系统的优缺点分析
4.案例分析

分层系统

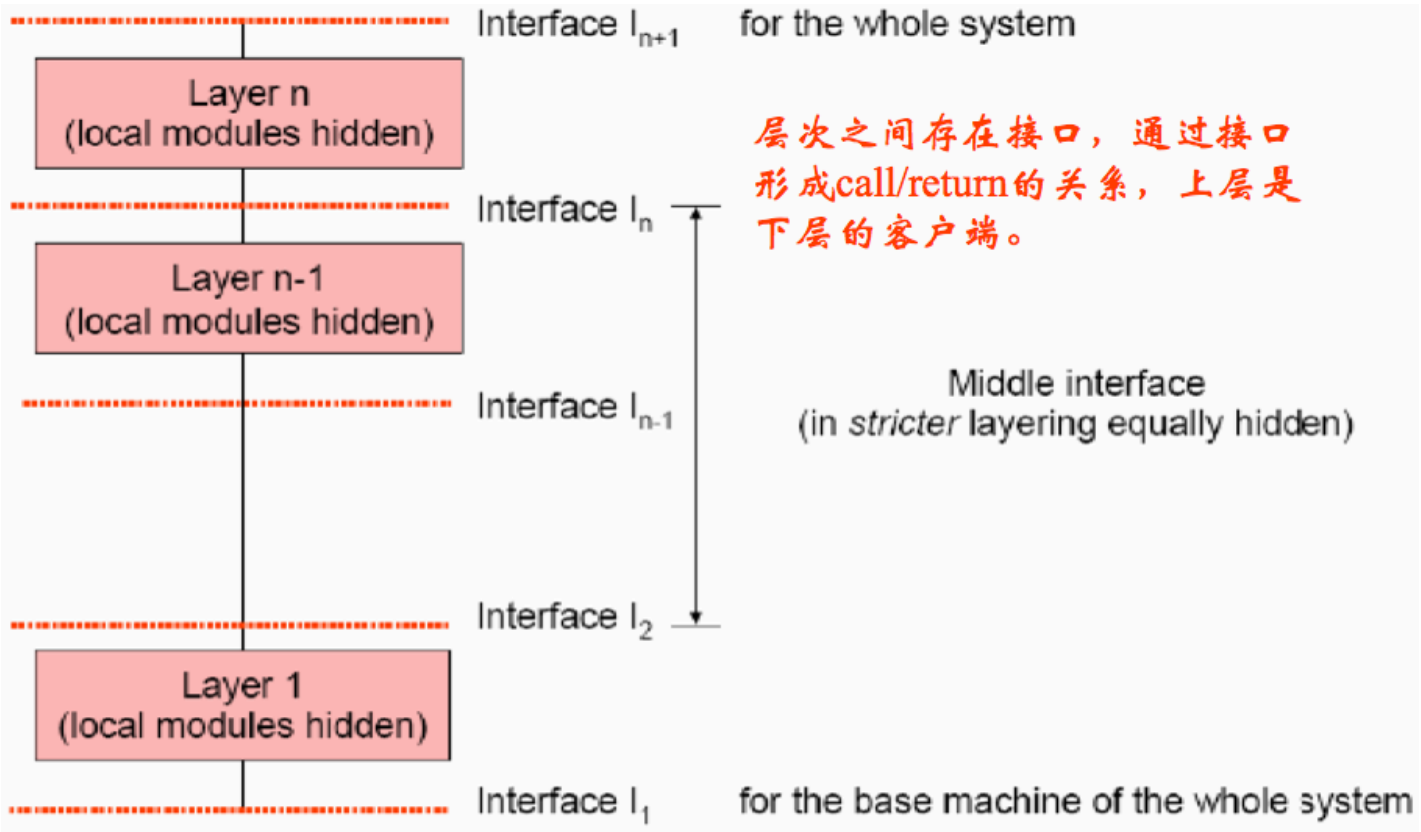
- A layered system is a system in which components are grouped, i.e., layered, in a hierarchical arrangement (在层次系统中,系统被组织成若干个层次,每个层次由一系列构件组成)
- lower layers provide functions and services that support the functions and services of higher layers (下层构件向上层构件提供服务)
- higher layers serves as a client to the layer below (上层构件被看作是下层构件的客户端)

分层系统示意图1

每个层次内部可以包含很多的功能组件，在实际应用中功能层都不是由一个层次实现的。



分层系统示意图2


Interface I_{n+1} for the whole system
Layer n
(local modules hidden)
Interface I_n
Layer n-1
(local modules hidden)
Interface I_{n-1}
Interface I_2
Layer 1
(local modules hidden)
Interface I_1 for the base machine of the whole system
层次之间存在接口，通过接口形成call/return的关系，上层是下层的客户端。
Middle interface
(in *stricter* layering equally hidden)

分层系统组成

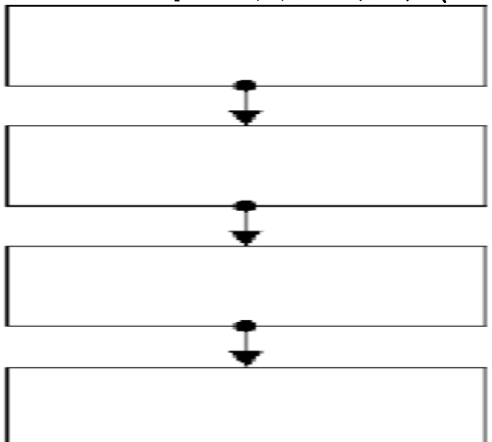
- The components are defined in each layer. (层次系统的基本构件:各层次内部包含的构件)
- The connectors are defined by the protocols that determine how the layers will interact. (连接件:层间的交互协议)
- Topology: layered structure. (拓扑结构:分层)
- Topological constraints include limiting interactions to adjacent layers.(拓扑约束:对相邻层间交互的约束)

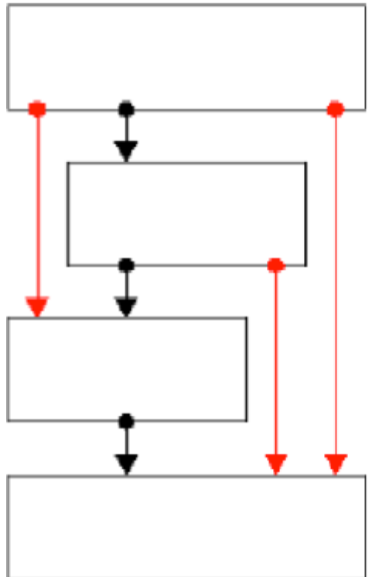
分层系统特征

- 这种风格支持基于可增加抽象层的设计，允许将一个 复杂问题分解成一个增量步骤序列的实现。
- 不同的层次处于不同的抽象级别： 越靠近底层，抽象级别越高；越靠近顶层，抽象级别越低；
- 由于每一层最多只影响两层，同时只要给相邻层提供 相同的接口，允许每层用不同的方法实现，同样为软件复用提供了强大的支持。

分层系统模式

- 某一层中的构件只能与同一级别中的对等实体或较低 级别中的构件交互,这有助于减少不同级别中的构件 之间的依赖性。
- 有两种通用的分层方法:
  - 严格分层(Strict System Layering)
  - 松散分层(Loosely System Layering)

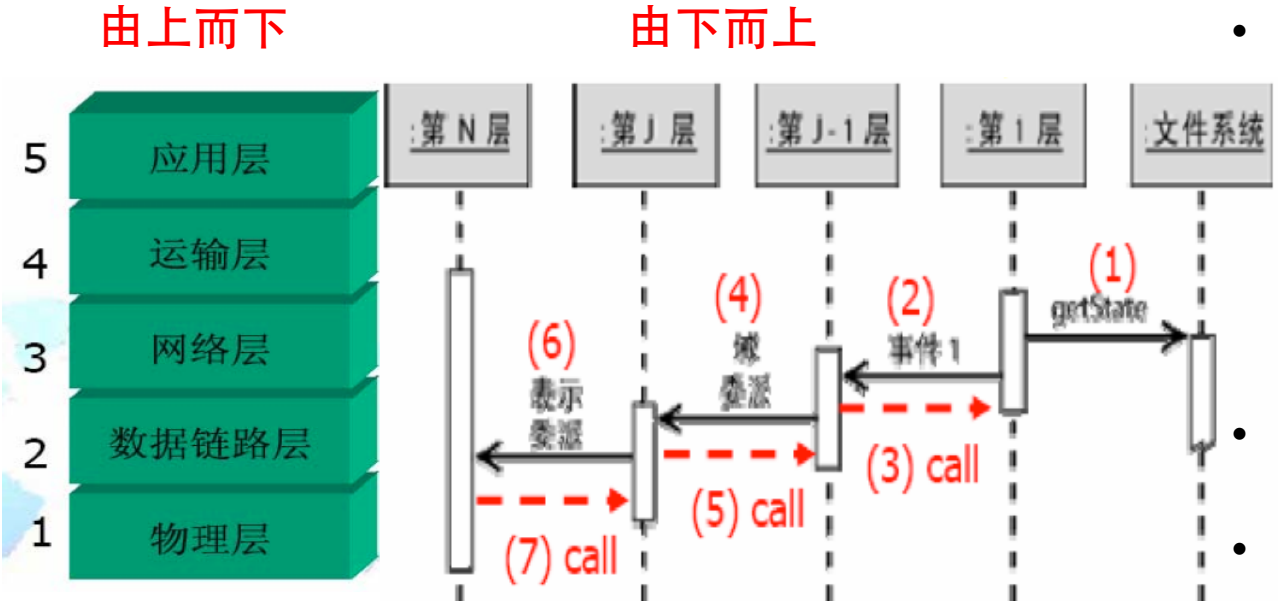




分层系统交互模式

- 严格和松散的分层模式是从系统的静态结构角度来划 分的
- 针对用户使用分层系统的动态行为,又存在四种的交 互模式:
  - 由上而下的交互模式
  - 由下而上的交互模式
  - 只使用所有分层的一个子集
  - 双向分层协议栈模式

分层系统交互模式

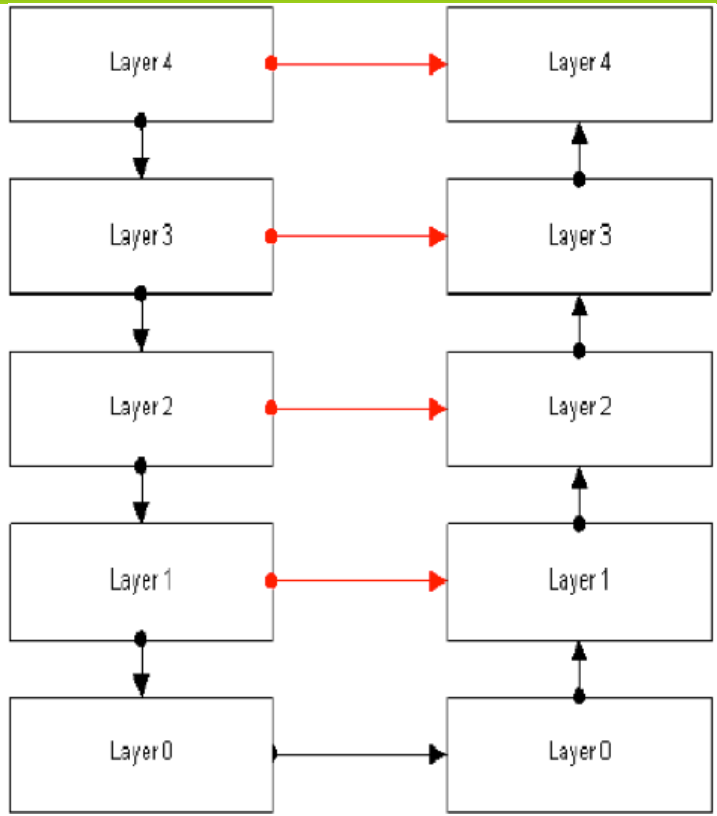


- 在由下而上方案中，较低层通过事件(event)、回调 和委派(callback delegate)来与较高层通信。

  目的：防止较低层依赖于较高层。为什么？」问题：什么叫做“回调”？
- 由上而下的信息和控制通常被描述成“请求”
- 由下而上的方式被描述为“通知”

双向分层协议栈模式

- 在某些网络/通讯系统里,信息必须在两个方向都可以传递
- 通信的双方各包括一个N层的协议栈 ,发送端系统中:高层 -> 低层
- 发送端与接收端系统在最底层的交互 ,接收端系统中:低层 -> 高层



分层系统的优点

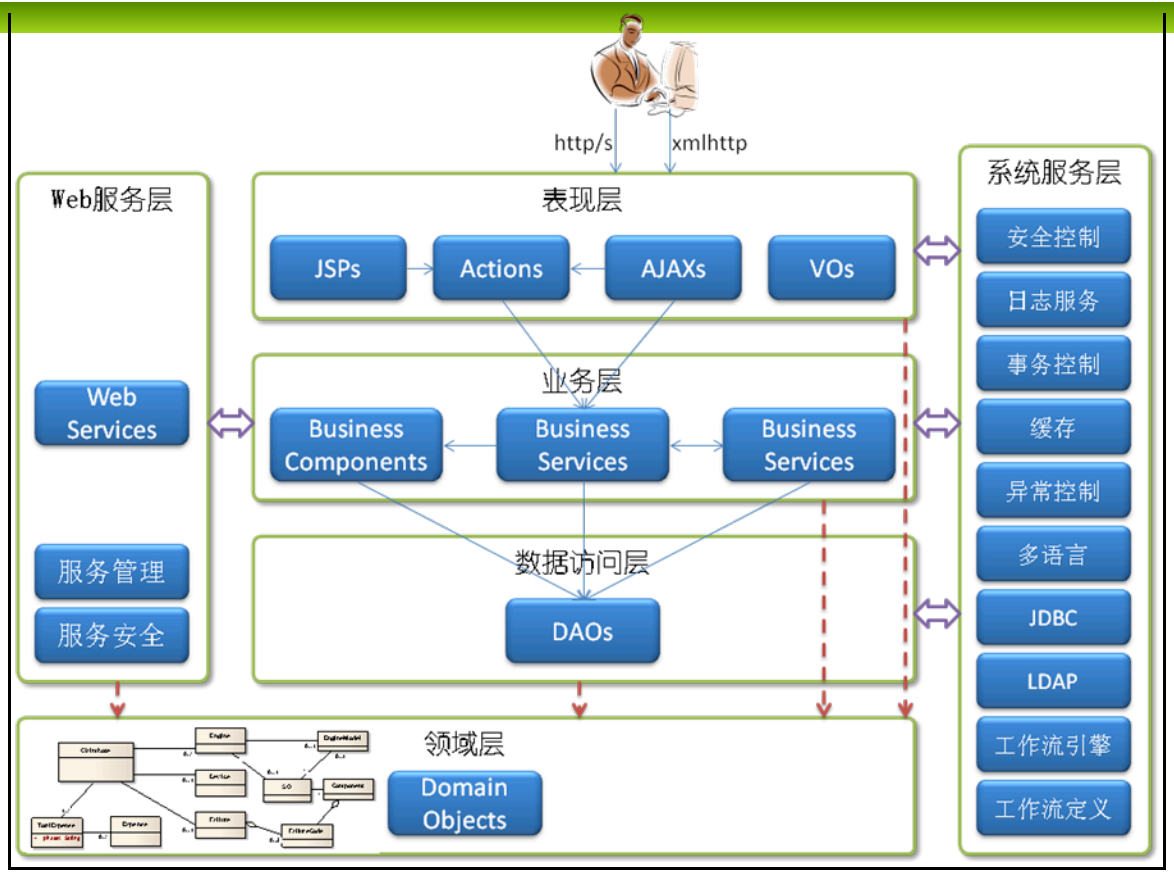
支持基于抽象程度递增的系统设计,使设计者可以把 一个复杂系统按递增的步骤进行分解;

- 。支持功能增强,因为每一层至多和相邻的上下层交 互,因此功能的改变最多影响相邻的上下层;
- 。支持复用。只要提供的服务接口定义不变,同一层的 不同实现可以交换使用。这样,就可以定义一组标准 的接口,而允许各种不同的实现方法。
- 。对标准化的支持。清晰定义并且广泛接受的抽象层次 能够促进实现标准化的任务和接口开发,同样接口的 不同实现能够互换使用。
- 。可测试性。具有定义明确的层接口以及交换层接口的 各个实现的能力提高了可测试性。

分层系统的缺点

- 并不是每个系统都可以很容易地划分为分层的模式，甚至即使 一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系 统设计师不得不把一些低级或高级的功能综合起来；
- 效率的降低：
  - 由分层风格构成的系统，运行效率往往低于整体结构。
  - 在上层中的服务如果有很多依赖于最底层，则相关的数据必须通过一 些中间层的若干次转化，才能传到；
- 很难找到合适的、正确的层次抽象方法：
  - 层数太少，分层不能完全发挥这种风格的可复用性、可更改性和可移植性上的潜力。
  - 层数过多，则引入不必要的复杂性和层间隔离冗余以及层间传输开销。
  - 目前，没有可行的广为人们所认可的层粒度的确定和层任务的分配方 法。

案例分析：一个web系统



谢谢！

Contact:

xue.anzhen@gmail.com