4-2 SaaS与云端软件部署

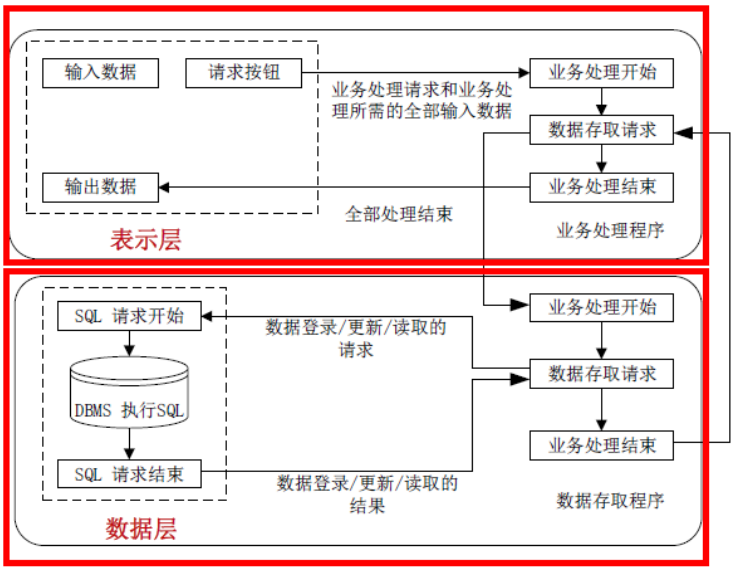
2019年4月14日　　10:27

1 软件架构初探：超越程序和代码

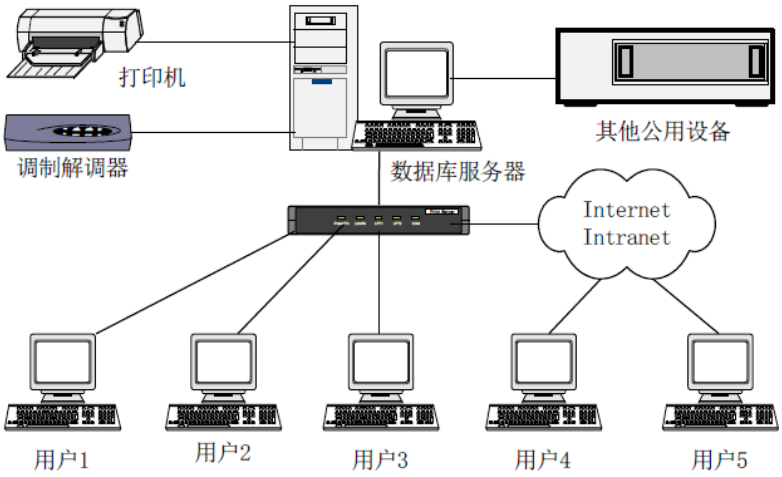
软件架构

- 软件越来越复杂，组成部分越来越多
  - 多个源文件
  - 多种类型的文件：用户界面、算法、数据层程序、配置文件、etc
- 不是单纯的代码，还涉及到所依赖的硬件和网络环境
  - 物理位置：单机、服务器、手机端、可穿戴硬件、etc
  - 网络支持：有线网络、3G/4G、WiFi等
- 多个软件实体之间如何组织起来?
- 软件和硬件之间的关系如何?
- 此即“软件架构”(SoftwareArchitecture)所关注的内容。

一个例子：两层客户端/服务器结构(C/S)

- 软件实体之间的关系：
  - 

- 硬件和网络环境之间的关系：

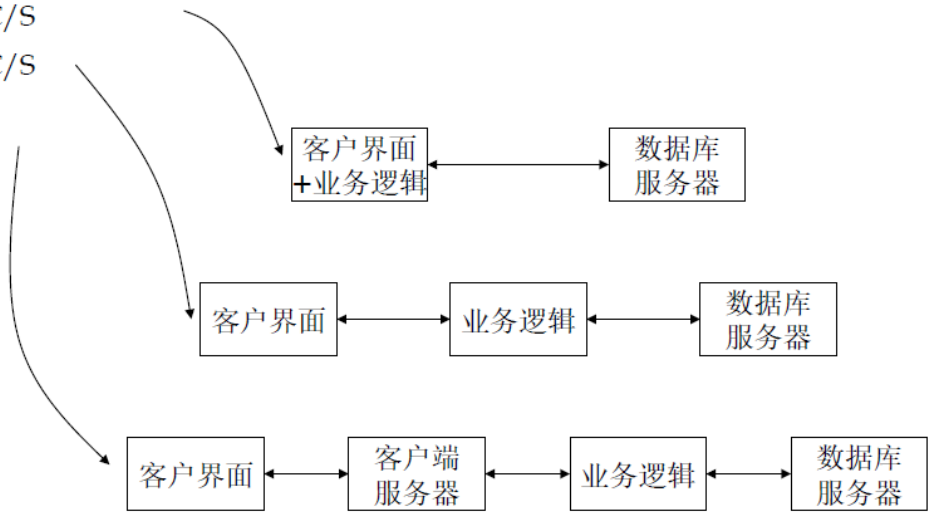


2 C/S B/S M/C

“客户机服务器”体系结构

- 客户机/服务器(Client/Server,C/S)：一个应用系统被分为两个逻辑上分离的部分，每一部分充当不同的角色、完成不同的功能，多台计算机共同完成统一的任务。
  - 客户机(前端，frontend)：用户交互、业务逻辑、与服务器通讯的接口；
  - 服务器(后端，backend)：与客户机通讯的接口、业务逻辑、数据管理。
- 一般的，
  - 客户机为完成特定的工作向服务器发出请求；
  - 服务器处理客户机的请求并返回结果。

客户机/服务器的层次性

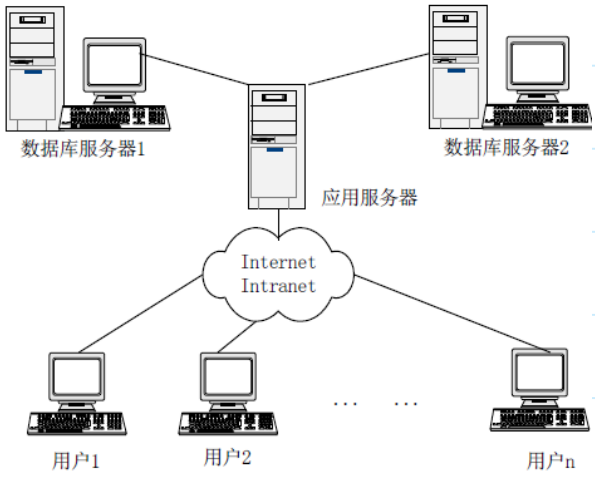
- “客户机-服务器”结构的发展历程：
  - 两层C/S（仓库体系风格）
  - 三层C/S
  - 多层C/S
- 

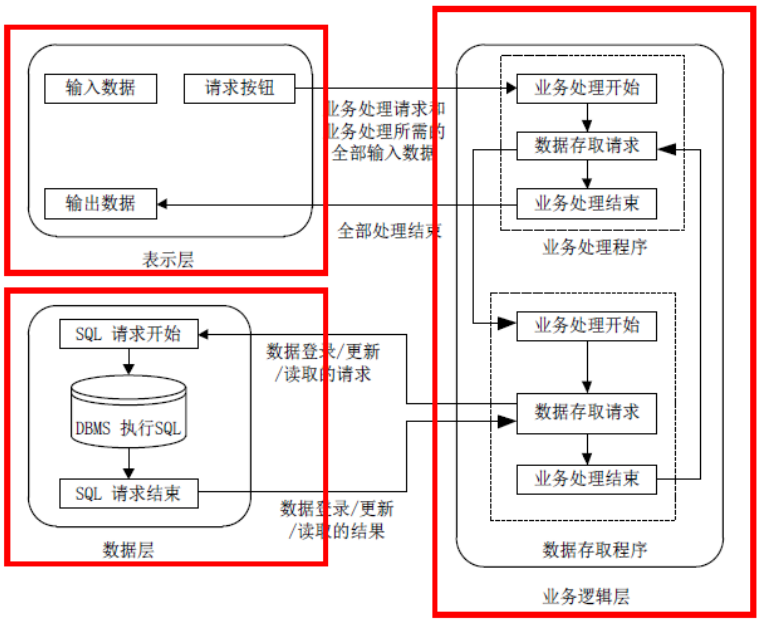
胖客户端与瘦客户端

- 业务逻辑的划分比重：在客户端多一些还是在服务器端多一些?

- 胖客户端：客户端执行大部分的数据处理操作
- 瘦客户端：客户端具有很少或没有业务逻辑

三层C/S体系结构

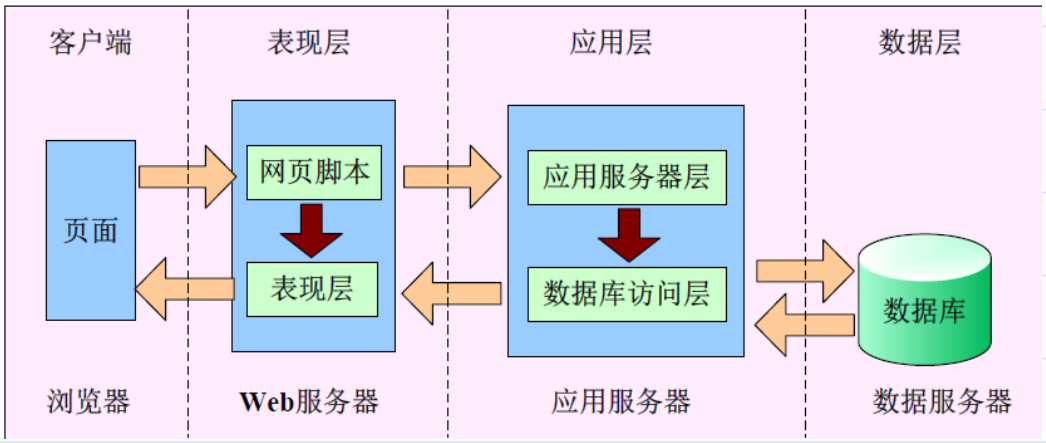
- 在客户端与数据库服务器之间增加了一个中间层
  - 表示层：用户界面—界面设计
  - 业务逻辑层：业务处理—程序设计
  - 数据层：数据存储—数据库设计
  - 

  - 

B/S结构

- 浏览器/服务器(B/S)是三层C/S风格的一种实现方式。
  - 表现层：浏览器
  - 逻辑层：
    - Web服务器
    - 应用服务器

    - 数据层：数据库服务器
- B/S与三层C/S结构的区别：
    - C/S：表现层仍部署在客户端；
    - B/S：客户端除了浏览器之外无任何程序需要部署。
-



超文本标记语言HyperTextMarkupLanguage(HTML)

- Web页面文档(Document)=Hierarchicalcollectionofelements
    - inline(headings,tables,lists)
    - embedded(images,JavaScriptcode)
    - forms:用于向服务端提交数据(text,radio/checkbuttons,dropdownmenus)
- 每个element具有特定的属性(attributes)和内容(content)
- 可采用大量的可视化HTML编辑软件编写

层叠样式表CascadingStyleSheets(CSS)

- CSS：将HTML文档中各elements的可视化显示样式与待显示的内容分开，在单独的文档(stylesheet)中加以定义，从而将页面设计师和开发者的工作分开
- <link rel=“stylesheet"href=“http://..."/>，在<head>元素内定义，用于指明该HTML页面使用哪个stylesheet；或者直接在<head>元素内定义
- 在每个HTML要素内，使用id和class属性来指向CSS中的相关定义：
    - id：页面范围内的唯一标识；
    - class：一个class可用于页面内的多个要素；
    -

```
<div id="right" class="content">
  <p>
   I'm Rainy. I teach Software Engineering and do
   research in the ICES Lab of CS/HIT.
```

动态产生HTML内容

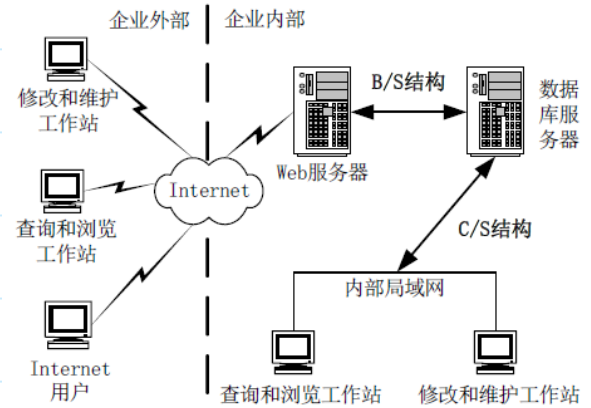
- 最初的大部分HTML页面都是静态的：直接从服务端获取，直接在HTML中展示；
- 目前的HTML：其中大部分内容是动态产生的，根据用户请求，服务端的程序执行之后产生内容，填充到HTML中，交付浏览器展示；

- 实现方式：在HTML中嵌入动态代码，如JSP、ASP、PHP、Ruby等，在服务器端编译/解释后传输到浏览器端执行。
- 此外，为了扩充浏览器中HTML的能力，可使用JavaScript、...

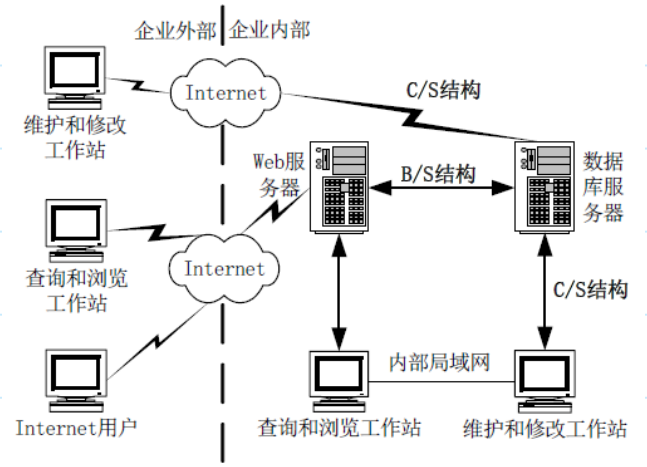
- B/S结构

  基于B/S体系结构的软件，系统安装、修改和维护全在服务器端解决，系统维护成本低：
  - 客户端无任何业务逻辑，用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级
  - 良好的灵活性和可扩展性：对于环境和应用条件经常变动的情况，只要对业务逻辑层实施相应的改变，就能够达到目的。
- B/S成为真正意义上的“瘦客户端”，从而具备了很高的稳定性、延展性和执行效率。
- B/S将服务集中在一起管理，统一服务于客户端，从而具备了良好的容错能力和负载平衡能力。

C/S+B/S混合模式

- 为了克服C/S与B/S各自的缺点，发挥各自的优点，在实际应用中，通常将二者结合起来；
- 遵循“内外有别”的原则：
  - 企业内部用户通过局域网直接访问数据库服务器
    - C/S结构；
    - 交互性增强；
    - 数据查询与修改的响应速度高；
  - 企业外部用户通过Internet访问Web服务器/应用服务器
    - B/S结构；
    - 用户不直接访问数据，数据安全；
  - 

- 遵循“查改有别”的原则：
  - 不管用户处于企业内外什么位置(局域网或Internet)，凡是需要对数据进行更新操作的(Add,Delete,Update)，都需要使用C/S结构；
  - 如果只是执行一般的查询与浏览操作(Read/Query)，则使用B/S结构

- 

M/C结构

- 移动端/云端结构(Mobile/Cloud)：
  - 客户端不是传统的客户机，而是各类移动终端设备，如智能手机、平板、智能家电、可穿戴设备等。
  - 服务端也不是传统的服务器，而是扩展到云环境下，支持高可伸缩性、按需付费等特性。
- 可以看作是C/S结构的扩展。
- 优势：移动，可以做到anytime&anywhere使用软件的功能。
- 客户端程序的体现形式：各类App

3 主流软件形态：SaaS

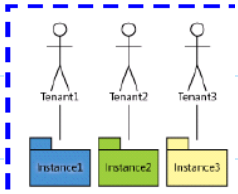
什么是SaaS?

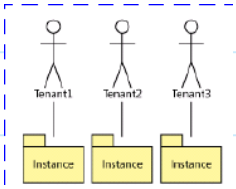
- SaaS(Softwareasaservice，软件即服务)
  - 一种通过Internet提供软件的模式，用户不用再购买软件，而改用向提供商租用基于web的软件，来管理企业经营活动，且无需对软件进行维护，服务提供商负责软件的可用性(软件维护、可扩展性、灾难恢复等)管理与支持；
- 企业采用SaaS服务模式，就像使用自来水和电能一样方便，从而大幅度降低了组织中应用先进技术的门槛与风险。
- 关键词：On-demand licensing and use

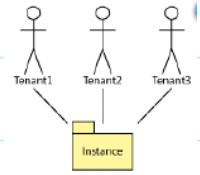
SaaS的特征

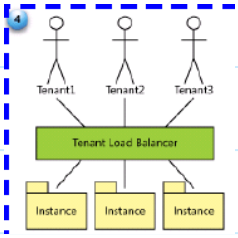
- SaaS：本质上属于B/S结构，对B/S的扩展：
  - 通过web来管理和使用软件；
  - 软件被集中式的部署与管理，统一升级和维护；
  - 单实例、多租户；
- SaaS与传统B/S的本质区别：多租户共享Server和软件实例。

从package-based software到SaaS的四个阶段

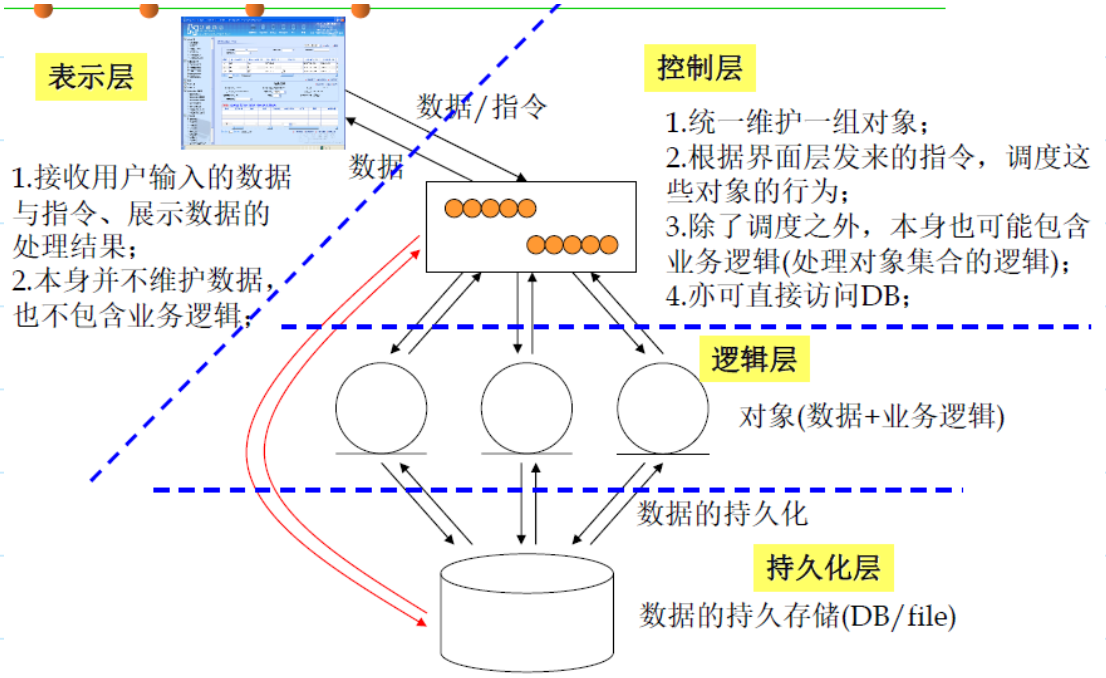
- Level1(AdHoc/Custom)定制开发
  - 将传统的软件系统迁移为基于网络的应用；
  - 每个用户具有自己的一套独立系统(DB、app、code)，在提供商的硬件环境下运行自己的实例。
  - 

- Level2(Configurable)可配置
  - 通过配置，一套相同的软件系统可以适应不同顾客的需求；
  - 运行时，该系统为不同顾客产生不同的运行实例；
  - 

- Level3(Configurable,MultiTenantEfficient)高性能的多租户架构
  - 在前一阶段的基础上增加了“多租户”特性；
  - 因而，多个顾客可以同时使用一个程序实例。
  - 优点：服务资源共享，利用效率高。
  - 

- Level4(Scalable,Configurable,MultiTenantEfficient)
  - 针对level3中可能出现的资源伸缩性问题，增加了负载平衡功能。
  - 在多台服务器上部署多个instance，顾客请求被分配到不同实例上。
  - 提供商可根据需求大小，动态调整资源，而无需改变软件本身。
  - 

SasS的层次划分



有关分层的补充说明

- 表示层、控制层、逻辑层、持久化层、数据层：这种划分是否一定有必要？——NO
- 这种划分的目的是为了“清晰分工”、“职责明确”，但同时也增加了系统运行时的性能代价；——为完成一项功能需要在多层之间多次调用
- 为简化起见，可将其中某些部分合并：
  - 边界类与控制类的合并：在用户界面中直接嵌入业务逻辑；
  - 控制类与实体类的合并：控制类中包含所有的业务逻辑，直接与持久化存储打交道。
  - 合三为一，不分层。
- 需根据系统的NFR需求，做出最佳的选择。

SaaS的惯用架构模式：MVC(模型-视图-控制器)

- 用户界面需要频繁的修改，它是“不稳定”的。
- 业务逻辑/数据与用户界面之间应避免/尽量少的直接通信。
- 目标：将承担不同职责的软件实体之间清晰的分离开来，降低耦合；
  - 让SaaS程序的用户界面与业务逻辑功能实现模块化，以便使程序开发人员可以分别开发、单独修改各个部分而不影响其他部分。

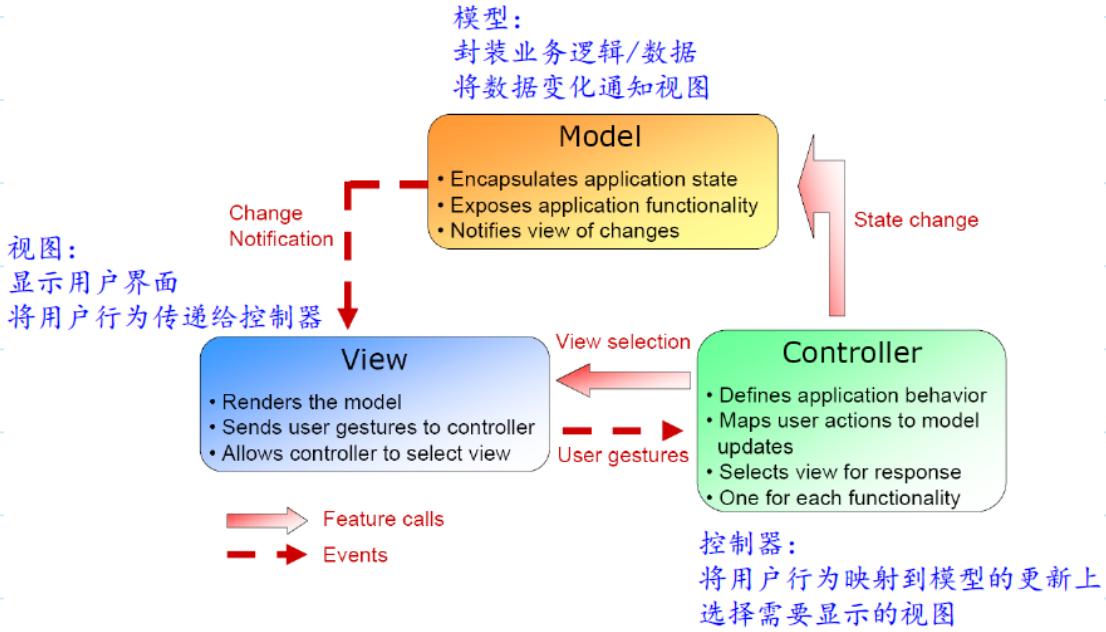
解决方案：ModelViewController(MVC)

- MVC是一种软件体系结构，它将应用程序的数据模型/业务逻辑、用户界面分别放在独立的构件中，从而对用户界面的修改不会对数据模型/业务逻辑造成很大影响。
  - 在传统的B/S体系结构的基础上加入了一个新的元素：控制器，由控制器来决定视图与模型之间的依赖关系。
  - 模型(Model,M)：用于管理应用系统的行为和数据，并响应为获取其状态信息(通常来自视图)

而发出的请求，还会响应更改状态的指令(通常来自控制器)；——对应于传统B/S中的业务逻辑和数据

- 视图(View,V)：用于管理数据的显示；——对应于传统B/S中的用户界面
- 控制器(Controller,C)：用于解释用户的鼠标和键盘输入，以通知模型和视图进行相应的更改。——在传统B/S结构中新增的元素

MVC运行机制



MVC各层次的实现技术

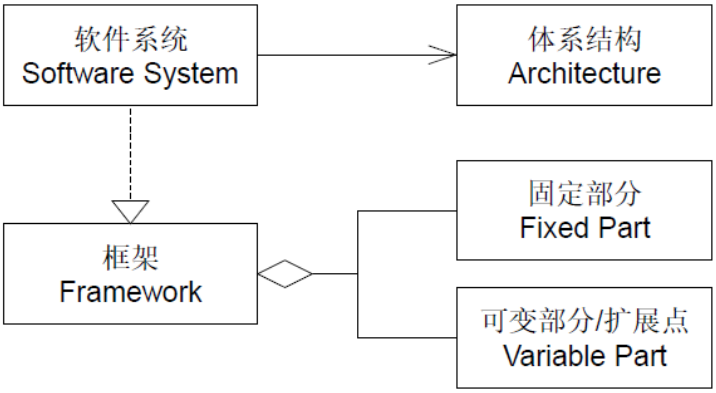
- 针对不同层次，采用不同的实现技术：
  - 用户界面层

    HTML/JavaScript/CSS、jQuery、JSP、AJAX、Flex、HTML5、Dojo、Bootstrap、Node js…
  - 控制层：PHP、Python、Servlet、Ruby、...
  - 业务逻辑层：JavaBean、Pojo、...
  - 持久化层：JDBC、JDO、Hibernate、iBatis、...
- Struts、Django、CI、Rails等以不同的编程语言(Java、Python、PHP、Ruby)分别实现了这一架构，提供了一个半成品，帮助开发人员迅速地开发符合MVC架构的应用程序，它们都是“框架Framework”。

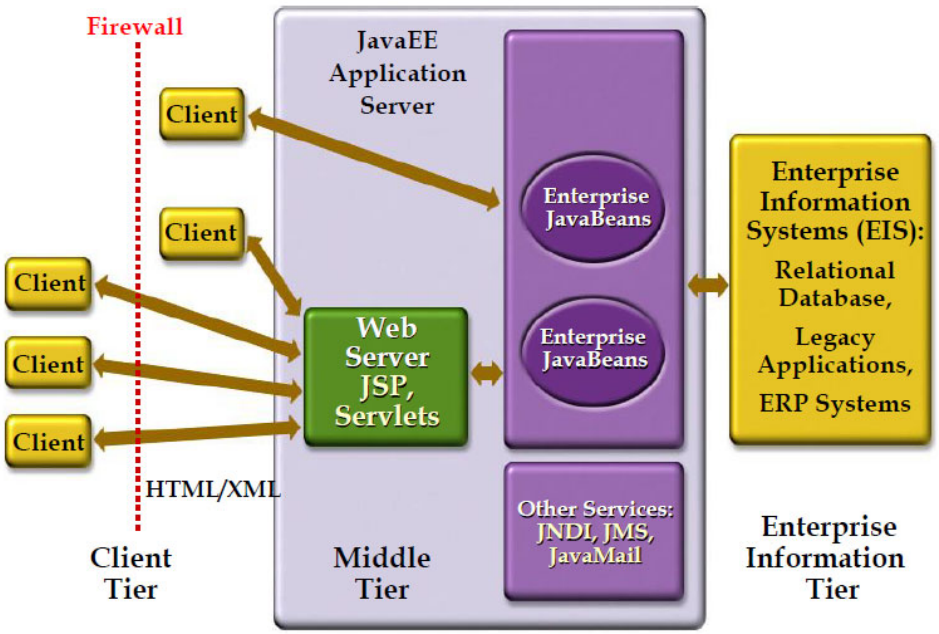
Framework vs Architecture(框架和体系结构)

- 框架(Framework)：可实例化的、部分完成的软件系统或子系统，它为一组系统或子系统定义了统一的体系结构(architecture)，并提供了构造系统的基本构造块(buildingblocks)，还为实现具体功能定义了扩展点(extendingpoints)。

- 框架实现了体系结构级别的复用。

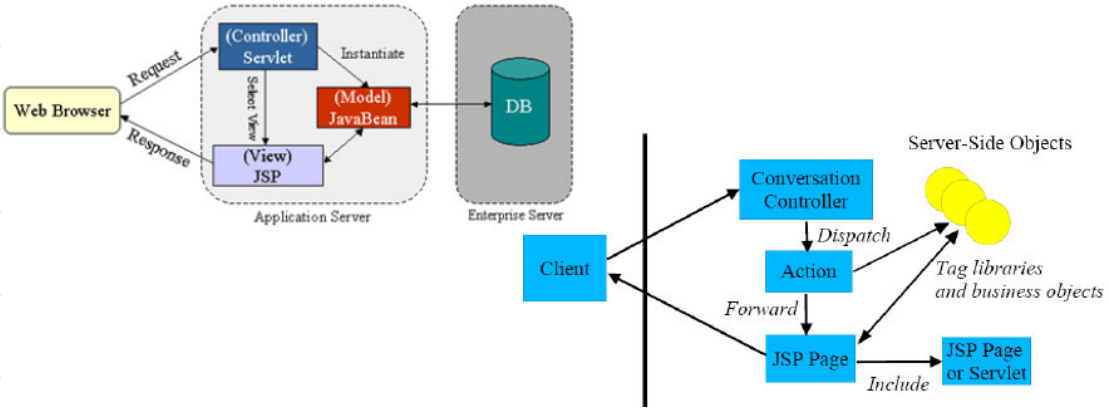


以JavaEE和Struts为例

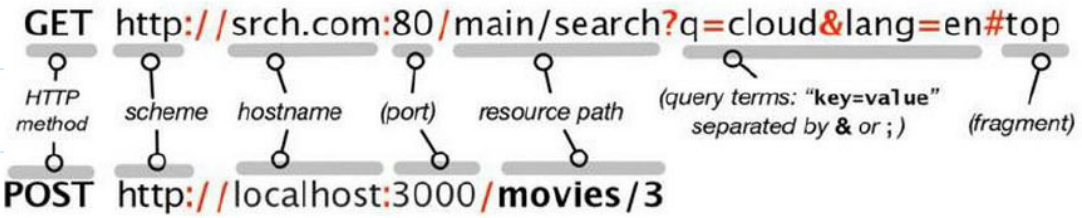


JavaEE MVC Model

- Servlet作为控制器，负责处理用户请求，并将其转发给扮演Model角色的JavaBean，而JSP作为纯粹的View：
- Servlet还决定在处理完该请求之后，将向用户显示哪个JSP页面(View)以显示处理结果；
- JSP页面获取Servlet的处理结果并动态显示给用户。



前端请求的URI(UniformResourceIdentifier)

- 用户通过浏览器发出的请求，通过HTTP协议传递到服务器端。
- 具体形式即为URI
- 

- HTTP是无状态协议，通过浏览器端的cookies保持同一用户的多次请求，通过服务器端的session保持与用户的连接。

FilterDispatcher：前端控制器/调度器

- 前端请求的URI中包含了一个resourcepath，代表着用户的请求。
- 通过HTTPRequest发送至MVC的frontcontroller，它相当于服务端的入口、总调度。——在Struts中，实现为FilterDispatcher。
- FilterDispatcher接收到请求之后，根据配置文件将请求转发到具体执行动作的Model(Struts中称之为action)。
- 配置文件strutsxml：
- ```
<struts>
  <package name="01" extends="struts-default">
    <action name="submit" class="SubmitAction" method="submit">
        <param name="param1">value1</param>
        <result>result.jsp</result>
    </action>
  </package>
</struts>
```

Action(model)和JSP(View)

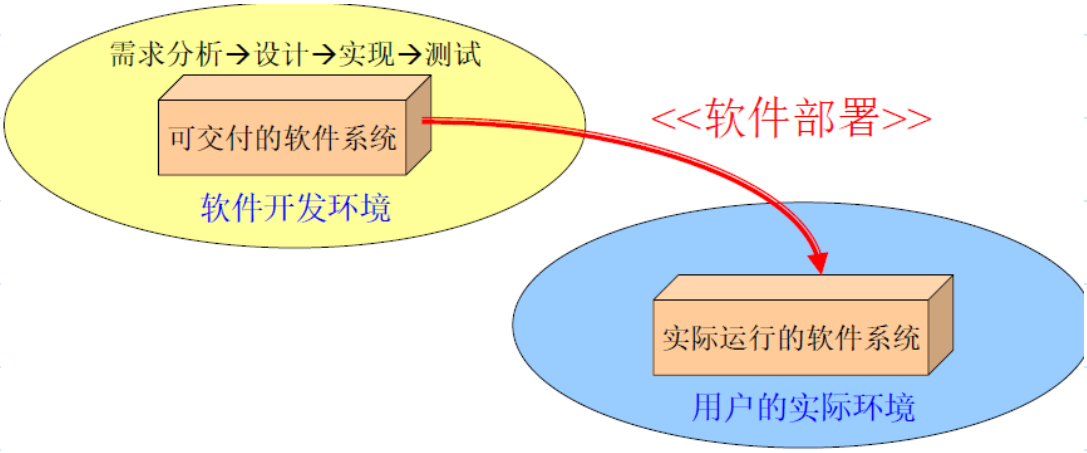
- ```
public class SubmitAction extends ActionSupport {
     private String param1;
     public String execute() throws Exception{
       ...
     }
     public void submit(String param1){
       ...
     }
}
```

```
<s:form action="submit" method="true">
  <s:textfield label="Message" name="Param1" />
  <s:submit value="submit" />
</s:form>
```

4 SaaS的部署环境：云平台

何谓“软件部署”

- 软件部署与实施(SoftwareDeployment&Implementation)：将系统设计方案与软件系统转换成实际运行系统的全过程。
-

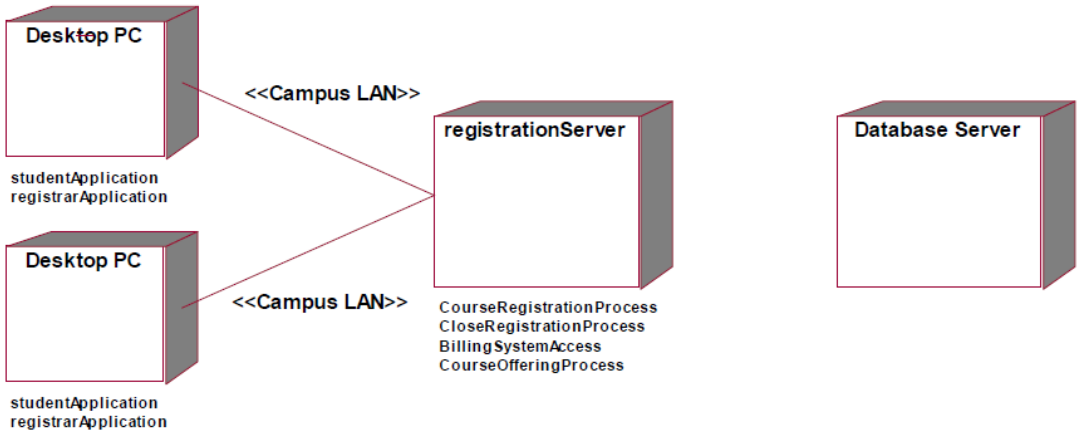


软件部署模型(DeploymentModel)

- 为系统选择硬件配置和运行平台；
- 将类、包、子系统分配到各个硬件节点上。
- 系统通常使用分布式的多台硬件设备，通过UML的部署图(deploymentdiagram)来描述；
  - 部署图反映了系统中软件和硬件的物理架构，表示系统运行时的处理节点以及节点中对象/子系统的分布与配置。
  - 部署对系统的性能和复杂度具有较大影响，需要在设计初期就要完成
- 描述系统中各硬件之间的物理通讯关系；
- 描述各软件实体被配置到哪个具体硬件上、这些软件实体之间的物理通讯关系；

部署子系统

-



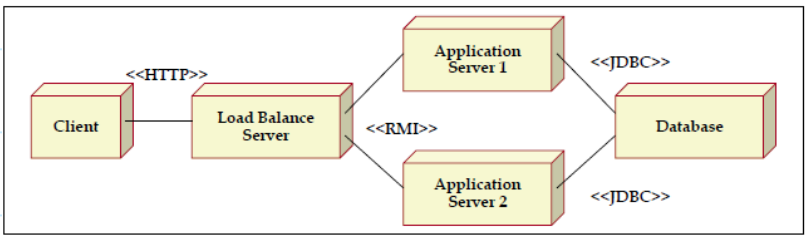
关于部署图

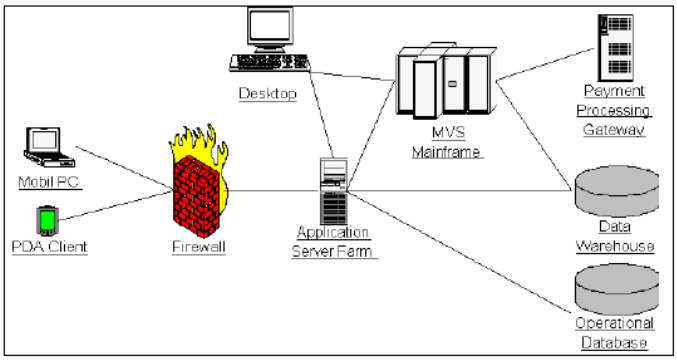
- 部署图(deploymentdiagram)：
  - 节点(node)：一组运行资源，如计算机、设备或存储器等。每个节点用一个立方体来表示，
  - 节点的命名：client、ApplicationServer、DatabaseServer、Mainframe等较通用的名字；

- 节点立方体之间的连接表示这些节点之间的通信关系，通常有以下类型：异步、同步；HTTP、SOAP；JDBC、ODBC；RMI、RPC；等等；

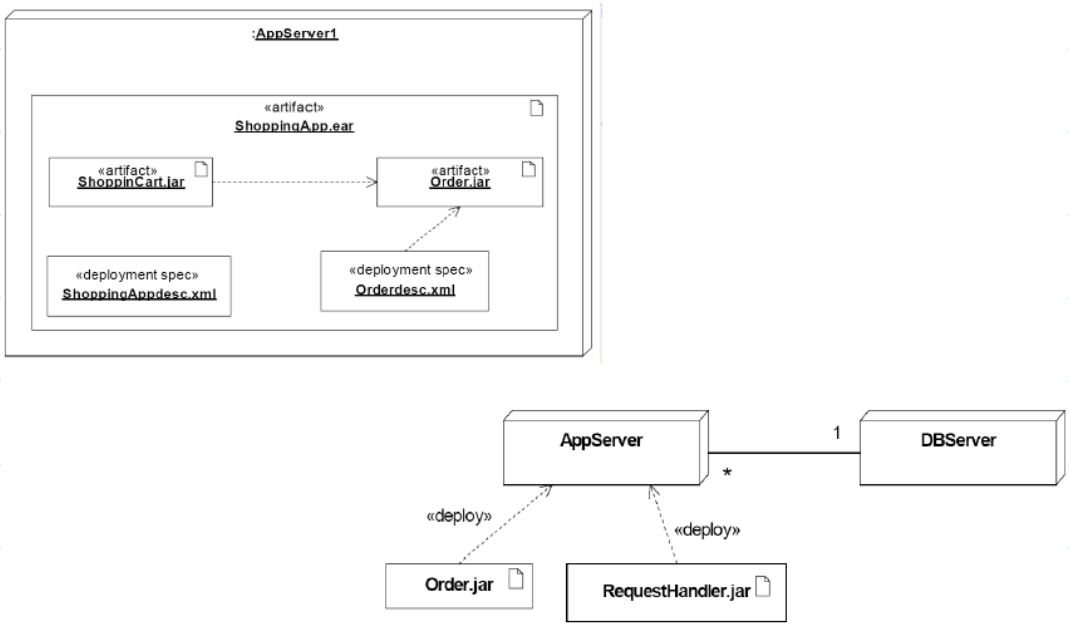
- 部署图在两个层面的作用：
  - High-level：描述系统中各硬件之间的物理通讯关系；
  - Low-level：描述各软件实体被配置到哪个具体硬件上、这些软件实体之间的物理通讯关系；

High-level Deployment Diagram



- 

Low-level Deployment Diagram

- 

绘制部署图(deploymentdiagram)

- 确定“节点(node)”：
  - 标识系统中的硬件设备，包括大型主机、服务器、前端机、网络设备、输入/输出设备等。

    - 一个处理机(processor)是一个节点，它具有处理功能，能够执行一个组件；
    - 一个设备(device)也是一个节点，它没有处理功能，但它是系统和外部世界的接口。
- 对节点加上必要的“构造型(stereotype)”
    - 使用UML的标准构造型或自定义新的构造型，说明节点的性质。
- 确定“连接(connection)”
    - 把系统的包、类、子系统、数据库等内容分配到节点上，并确定节点与节点之间、节点与内容之间、内容与内容之间的联系，以及它们的性质。
- 绘制配置图(deploymentdiagram)

架构设计思路小结

- 逻辑架构：只考虑如何分层、每个层次中的模块、层次内模块的关系、层次间模块的关系。主要是给开发者提供指南。
- 物理架构：考虑的是实际硬件/网络环境，以及如何将逻辑架构映射到硬件/网络环境上去。主要是给实施人员提供指南。
    - 通常，逻辑分层可以1:1映射到物理分层上；
    - 某些时候，多个逻辑层次可以部署在同一个物理层次上(n:1)；
    - 某些时候，同一个逻辑层次可以拆分为多个物理设施上(1:n)；
- 物理架构的设计思路：
    - 从逻辑架构入手，分别考虑每个逻辑层次在物理环境下是如何实现的；
    - 从简单到复杂，考虑每项设计决策对物理设施的要求，逐渐扩充物理架构。

将SaaS部署到哪里?

三种选择：

- 本机(主要用于开发环境)
- 自己搭建服务器(组织内部使用)
- 公共的服务器——云

什么是Cloud?

- 这是一种新的计算方式和共享基础架构的方法，IT相关的计算能力被作为“服务”，通过Internet向外部客户提供，但客户不需了解这些计算能力的物理来源及其分布。
- 目标：使IT计算能力(存储和计算)可以向电能一样提供给客户。

Cloud所能提供的三种典型服务

- IaaS(InfrastructureasaService，基础架构即服务)
    - 通过互联网提供了数据中心、基础架构硬件，可以提供服务器、操作系统、磁盘存储、数据库和/或信息资源。

    - AmazonEC2
- Paas(PlatformasaService，平台即服务)
    - 提供了软件基础架构，软件开发者可以在这个基础架构之上建设新的应用，或者扩展已有的应用。
    - cloud_stack
    - Salesforcecom的Forcecom、Google的AppEngine和微软的Azure、新浪的SAE、百度的BAE
- SaaS(SoftwareasaService，软件即服务)
    - Salesforcecom、NetSuite、Google的Gmail/Docs

基础设施即服务(IaaS)

- 简单的说，IaaS可看作物理服务器(裸机，CPU+内存+磁盘)的虚拟化；
- 用户可在上面安装操作系统、运行环境、装载数据，再在上面部署应用系统。
- 代表:
    - AmazonEC2
    - GoogleComputeEngine(GCE)
    - OpenStack
    - VMWare
    - Eucalyptus
- 哈工大的云空间http://s.hit.edu.cn/

平台即服务(PaaS)

- 在IaaS基础上，有了完整的运行环境和基础服务支持(例如OS、DB、应用服务器、MVC、Message等)；
- 将中间件环境作为了服务，向用户提供；
- 按照平台要求将程序部署到上面去。
- 代表:
    - GoogleAppEngine(GAE)
    - WindowsAzure
    - SinaAppEngine(SAE)
    - BaiduAppEngine(BAE)
    - Heroku