软件设计课堂笔记

> @Date:2019年08月26日
>
> @Author:YeLihu
>
> @Email:87160265@qq.com

介绍

> 课程时间：2019-08-26

Q：软件设计和程序设计有什么不一样?

Q：软件包含什么

和其他课程的关系

1. 需求：**前导课程**
2. 数据库：
    1. 数据库的类型是什么?
    2. 数据库的数据量有多大？需不需要分库分表的操作?
    3. 数据的更新频率是多大?
    4. 范式/性能?
3. 测试：**测试是保证软件质量的重要环节**
    1. 测试用例怎么设计?
    2. **测试的依据是分析的文档**
4. 构造和演化：**软件设计的后续课程**
    1. 编码，单元测试，集成测试和性能测试形成了软件的构造
    2. 设计模式?
5. 软工综合实践：第七个学期
6. 毕设：第八个学期

第一讲：软件体系结构

基本概念

软件体系结构出现的原因

- 上个世纪90年代，网络的发展，Web Server的出现导致。
- 软件具有**抽象**的特点，不像建筑工程等，只需做好设计，画好图纸即可。

> **什么是Web Server**?
>
> Web服务器可以解析HTTP协议。当Web服务器接收到一个HTTP请求(request)，会返回一个HTTP响应(response)。
>
> 例如送回一个HTML页面。为了处理一个请求(request)，Web服务器可以响应(response)一个静态页面或图片，进行页面跳转(redirect)，或者把动态响应的产生委托给一些其它的程序例如JSP(JavaServer Pages)脚本，servlets，ASP(Active Server Pages)脚本，服务器端(server-side)JavaScript，或者一些其它的服务器端(server-side)技术。

软件危机：

需求的时候讲过，软件危机发生的原因：需求不明确，软件项目管理不当、技术...

原因

- 软件的规模越来越大，项目进度已经无法控制
- 成本(谁出钱？)
- 软件质量差

如何解决

- 需求方面进行改革(解决需求的问题)
- 项目管理，引入体系结构(解决规模的问题)

> **维基百科**：https://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E5%8D%B1%E6%9C%BA

体系架构的兴起

- 随着软件系统规模越来越复杂，整个系统的结构和规格说明显得越来越重要。
- 对于大规模的复杂软件系统来说，对**总体的系统结构**设计和规格说明比起**算法和数据结构**的选择重要得多。
- 对软件体系结构的研究将会成为提高软件生产率和解决软件**维护问题**的有效途径。(DevOPs、以产品管理为核心)
- 事实上，软件总是有体系结构的，**不存在没有体系结构的软件**。
- 软件体系结构虽起源于软件工程，但其形成同时借鉴了**计算机体系结构**和**网络体系结构**中很多宝贵的思想和方法，成为软件工程研究的重要分支

体系结构的定义

从构成的角度

注重区分处理构件、数据构件和连接构件，这一方 法在其它的定义和方法中基本上得到保持

从工程过程的角度

这一过程在的算法设计和数据结构设计之上

处理的是整体的结构，构件、协议...

> 爷问：什么是协议?
>
> 一种关于内容和顺序的约定，比如我们要求数据接口的参数、返回值、取值范围等，这就是一个协议。

从审视的视角

体系结构是复杂的，我们怎么去观察体系结构呢？下面四个视角。

- 概念角度描述系统的主要构件及它们之间的关系；
- 模块角度包含功能分解与层次结构；
- 运行角度描述了一个系统的动态结构
- 代码角度描述了各种代码和库函数在开发环境中的组

从重用的角度

代码重用的好处如下：

1. 提高效率
2. 节约时间（比如框架）。

软件体系结构级的重用意味着体系结构的决策能在具有相似需 求的多个系统中发生影响，这比代码级的重用要有更大的好处。

从维护的角度

软件体系结构是一个程序 / 系统各构件的结构、它们之间的相互关系以及进行设计的原则和随时间演化的指导方针。

从工程的作用

软件体系结构包括一个软件和系统构件、互联及约束的集合一个系统需求说明的集合

软件体系结构的意义

体系结构是风险承担者（Stick Holder）进行交流的手段

从对象的角度来看：软件 = 对象 + 消息

软件体系结构，用户要参与评审(怎么参与评审？)

- 软件体系结构代表了系统的公共的高层次的抽象
- 软件体系结构对项目最终的质量和使用有极大的影响

体系结构是早期设计决策的体现

软件体系结构明确了对系统实现的约束条件

软件体系结构影响了开发组织的组织结构

软件体系结构制约着系统的质量属性

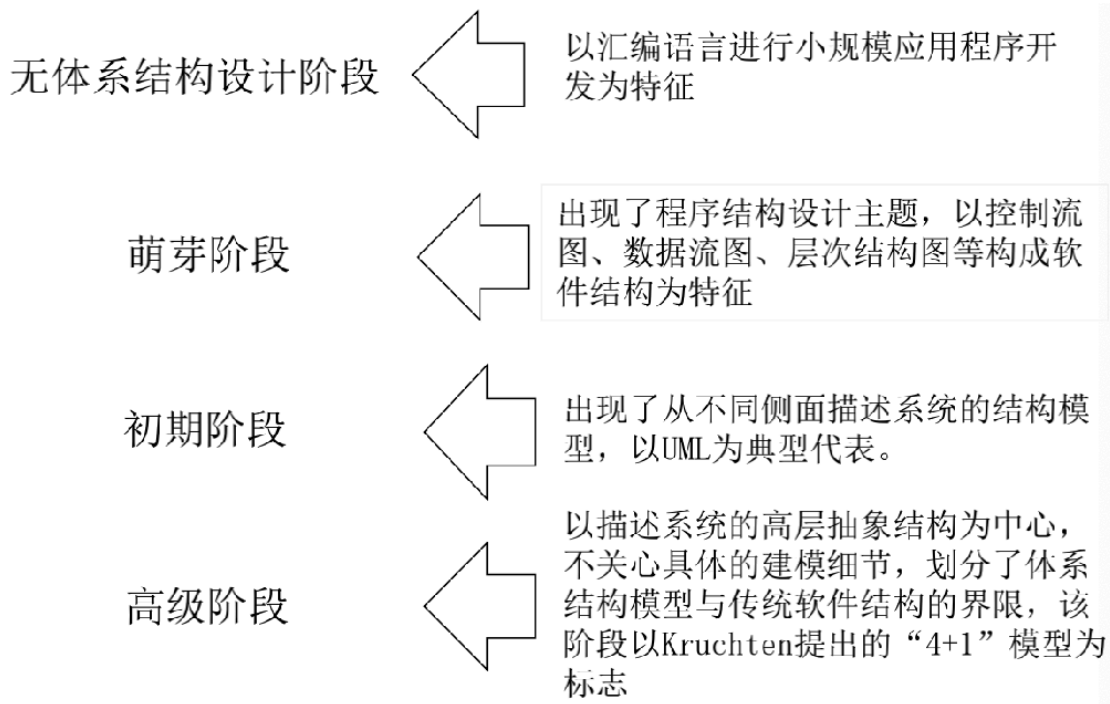
通过研究软件体系结构可能预测软件的质量(预测每千行代码的数量...)

软件体系结构使推理和控制更改更简单

软件体系结构有助于循序渐进的原型设计(先做核心的功能)

软件体系结构可以作为培训的基础(新的人员可以根据文档快速的了解)

软件体系结构的发展史



体系结构描述构造与表示

描述体系结构的过程称为体系结构构造

"4+1"模型包括

- 设计视图
- 进程视图
- 实现视图
- 部署视图
- 用例视图

IEEE于1995年成立了体系结构工作组，起草了体系结构描述框架标准IEEE P1471

> 这是一种严格的国际标准，不同于国际标准的还有组织标准和国家标准，行业标准和企业标准。比如TCP/IP和UML就是一种行业标准

体系结构分析、设计与验证

验证

ATAM-Architecture Tradeoff Analysis Method

上面这个是软件体系结构评审的标准。

基于体系结构的软件开发方法

引入体系结构之后，软件系统的开发过程变为“问题定义—>软件需求—>软 件体系结构—>软件设计—>软件构造”

软件体系结构架起了软件需求与软件设计之间的一座桥梁

在基于构件和基于体系结构的软件开发逐渐成为主流情况下，已经出现了基于构件的软件工程。

> 构件类似于service，可以举例为第三方支付，地图等第三方接口。除了service，还可以通过网络协议等实现。

特定领域的体系结构框架

体系结构设计是可重用的，特定领域下，软件体系结构是相似的。

DSSA：面向某一个领域的体系结构。

本课程的体系结构

- 软件体系结构：体系结构是一种构件的层次化结构，包含构件之间的交互方式、

  构件使用的数据结构
- 构件：指语义完整、语法正确和有可重用价值的软件单元。比如属性要定义取值类型，范围等、方法要指明参数和功能。

构件模型

面向构件的模型历史上有：CORBA,EJB（SUN公司的Enterprise Java Bean）,COM/COM+/DCOM。

但是他们彼此之间互不兼容，各家有各家的标准，所以并没有流行开来。他们被SOA取代。

> SOA粗暴理解：把系统按照实际业务，拆分成刚刚好大小的、合适的、独立部署的模块，每个模块之间相互独立
>
> 比如现我有一个数据库，一个JavaWeb（或者PHP等）的网站客户端，一个安卓app客户端，一个IOS客户端。
>
> 现在要从这个数据库中获取注册用户列表，如果不用SOA的设计思想，那么就会这样：JavaWeb里面写一个查询方法从数据库里面查数据然后在网页显示，安卓app里面写一个查询方法查询后在app上显示，IOS同样如此。这里就会出现查询方法重叠了，这样的坏处很明显了，三个地方都有相同的业务代码，要改三个地方都要改，而且要改的一模一样。当然问题不止这一个。
>
> 于是乎出现了这样的设计思想，比如用Java（或者是其他语言皆可）单独创建一个工程部署在一台服务器上，并且写一个方法（或称函数）执行上述查询操作，然后使其他人可以通过某种途径（可以是http链接，或者是基于socket的RPC调用）访问这个方法得到返回数据，返回的数据类型是通用的json或者xml数据，就是说把这个操作封装到一个工程中去，然后暴露访问的方式，形成“服务”。比如这里就是注册用户服务，而关于注册用户的所有相关增删改查操作这个服务都会提供方法。
>
> 这样一来，JavaWeb这边可以访问这个服务然后得到数据使用，安卓和IOS这里也可以通过这个服务得到数据。而且最重要的是，要修改关于注册用户的业务方法只要改这个服务就好了，很好的解耦。同理，其他业务比如商品、广告等业务都可以单独形成服务部署在单独服务器上。

> 还有就是一旦哪天突然有一堆人要注册，假设这堆人仅仅只是注册而不做其他事情，其他业务比如商品、广告服务等都不忙，唯独注册这个功能压力很大，而原有的一台部署了注册服务的服务器已经承受不了这么高的并发，这时候就可以单独集群部署这个注册服务，提供多几台服务器提供注册服务，而其他服务还不忙，那就维持原样。
>
> 当然，还有很多其他好处。
>
> 什么是服务治理，就是当服务越来越多，调用方也越来越多的时候，它们之间的关系就变得非常混乱，需要对这些关系进行管理。举例，还是上面的例子，假如我有一个用户服务，一开始有调用方1和调用方2来使用这个服务，后来越来越多，将近上百个调用方，这个时候作为服务方，它只知道提供服务，却不知道具体为谁提供了服务。而对于开发者来说，知道这N多调用方和N多服务方之间的关系是非常重要的。
>
> 所以这个时候就需要能进行服务治理的框架，比如dubbo+zookeeper，比如SpringCloud，有了服务治理功能，我们就能清晰地看到服务被谁谁谁调用，谁谁谁调用了哪些服务，哪些服务是热点服务需要配置服务器集群，而对这个服务集群的负载均衡也是服务治理可以完成的重要功能之一。
>
> 实际上SOA只是一种架构设计模式，而SOAP、REST、RPC就是根据这种设计模式构建出来的规范，其中SOAP通俗理解就是http+xml的形式，REST就是http+json的形式，RPC是基于socket的形式。上文提到的CXF就是典型的SOAP/REST框架，dubbo就是典型的RPC框架，而SpringCloud就是遵守REST规范的生态系统。

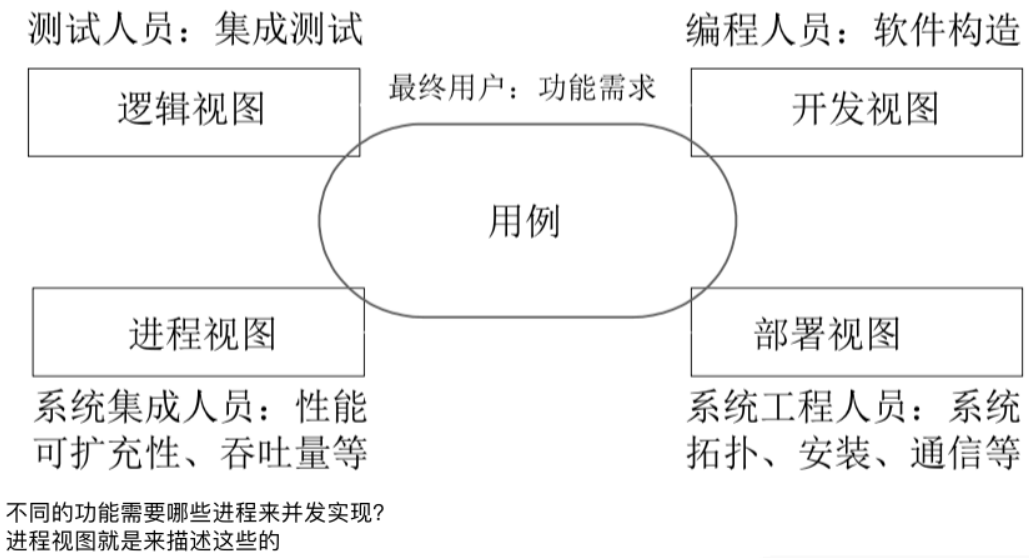
软件体系建模

"4+1"模型-概述

"4+1"模型包括

- 设计视图
- 进程视图
- 实现视图
- 部署视图
- 用例视图

"4+1"模型-逻辑视图



OS/CPU/网络等等都与软件的架构息息相关。

"4+1"模型-逻辑视图解析

与需求上相同的是，都是使用UML语言来描述，但是在设计中，不包括一些物理实体。

- 逻辑视图主要支持系统的功能需求，即系统提供给最终用户的服务。
- 在面向对象技术中，通过抽象、封装和继承，可以用对象模型来代表 逻辑视图，用类图来描述逻辑视图

"4+1"模型-开发视图解析

开发人员(程序员)使用

描述一个类的名称方法属性等。

开发视图当中是分层描述的(controller、service、dao……)

"4+1"模型-进程视图描述

几个进程可能运行在不同的机器上。

比进程更小的单位是线程和协程，因为当前cpu是多核的，能够更好的提高并发的表现。

"4+1"模型-部署视图/物理视图

描述软件和硬件的配合

为了满足软件对于硬件的需要：

- 软件需要多少带宽？(网络)
- cpu需要多少核心的？需不需要集群?
- ...

"4+1"模型-用例视图(4+1模型中的1)

将需求中用例图的system打破，拆分和展开。

体系结构设计过程

目标和实施者

将逻辑模型转化为实现模型。

实施者是：系统架构师

参与者

- 开发者(程序员、测试人员.....)
- 用户代表
- 硬件/网络架构师
- 安全架构师/咨询师、
- 操作人员

过程

主要输出

主要输出有如下几个文档

- 体系结构设计文档 (ADD-Architectural Design Document);
- 详细设计阶段的软件项目管理计划 (SPMP/DD);
- 详细设计阶段的软件配置管理计划 (SCMP/DD);
- 详细设计阶段的软件验证与确认计划 (SVVP/DD);
- 详细设计阶段的软件质量保证计划 (SQAP/DD);
- 集成测试计划 (SVVP/IT).
- 进度报告和配置状态审计报告

体系结构设计的基本原则

基本原则如下，后附解析(加粗)：

- 考虑多种设计方案：

  需要考虑平衡折中的方案(投资/时间/领域是否成熟/团队的能力)*

- 设计应该可跟踪到需求模型

  设计一定是基于需求而不是想象！跟踪矩阵可以很好的解决这个问题*

- 设计应尽可能地重用设计经验：

  为了提高质量和效率，既然可以重用，说明有了先前经验，质量上可以保证，重用能够节省时间。类似于需求的重用（概念模型可以重用,用户、功能...），设计也可以有重用(体系结构风格style、设计模式DP统称为可以重用)。

- 软件的结构应与问题域的结构相近。

  比如教务系统的设计上，用户角色和岗位的设置就要尽可能的和现实一样。

- 设计模型应该符合统一规范。

  不同岗位、前端后端，都需要遵循一定的规范。

- 设计不是编码，编码也不是设计。

  设计中的伪码比C/Java...编程语言更加抽象，并且注重逻辑。

- 对设计模型进行质量评审，而不是事后进行修改。

  找bug效率最高的方式、质量的保证是——评审。评审可以分为结对编程和正式评审，这个环节发生在正式提交编译之前。

> **如何将人工智能运用到软件工程?**
>
> 自动编码、项目管理

软件体系结构风格

软件体系结构风格的定义

- 描述**某一特定应用领域**中系统组织方式的**惯用模式**。 某一类特定的领域
- 体系结构风格定义了一个系统家族：即一个体系结构定义了：
  - **一个词汇表**：词汇表包含
    - **一些构件**：内存，数据库....
    - **连接件类型**：协议(HTTP/JDBC...)构件之间联系的方式。
  - **一组约束**：这组约束指出系统 是如何将上面这些构件和连接件组合起来的。
- 体系结构风格反映了领域中众多系统所共有的结构和语义特性，并指导 如何将各个模块和子系统有效地组织成一个完整的系统。

经典的软件体系结构风格

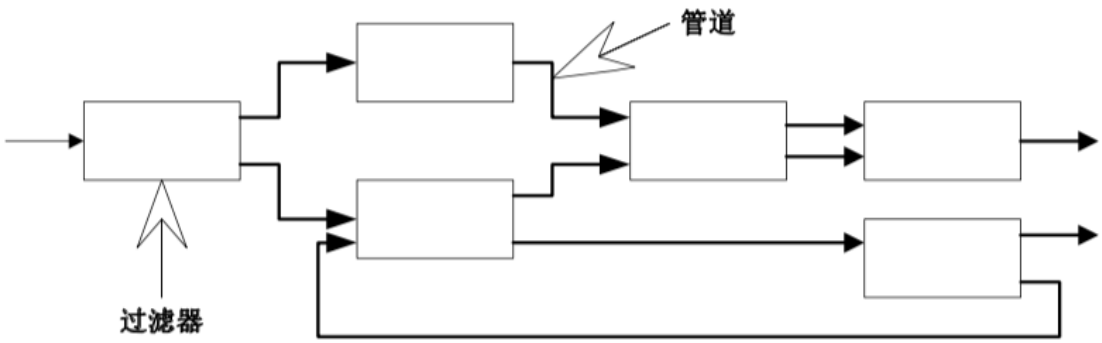
- 数据流风格:批处理序列;**管道**/过滤器**例子：Linux和Unix的非交互式系统，一个个命令按照输入的顺序执行。**
- 调用/返回风格:主程序/子程序;面向对象风格;**分层结构。对象→方法**
- 仓库风格:数据库系统;超文本系统;黑板系统。

上面三种结构风格优缺点整理如下：

	数据流风格	面向对象的体系	仓库风格
优点	重用、维护、并发性支持、大数据处理	交互性强，复用性好	数据和事务的处理
缺点	没有交互、设计复杂	必须知道谁(对象)提供服务(函数)，灵活性差	单一，优点也是缺点

> exp:
>
> H5中有内存数据库，将会话存放在内存的session中。IndexDB...

管道和过滤器



管道和过滤器的优点

适合大数据处理

管道和过滤器的缺点

没有交互

分层系统

> 计算机网络就是分层的体现，物理层/网络层/应用层/.....

分层系统的优点：

将大的系统降低复杂度。

可以协作、便于维护、便于重用。避免竞争。、

分层系统的缺点：

实时性、性能要求较高的系统支持较差

本来是直接简单的操作，现在需要在整个系统中层层传递，势必造成性能的下降，同时也加大的开发的复杂度。

仓库系统

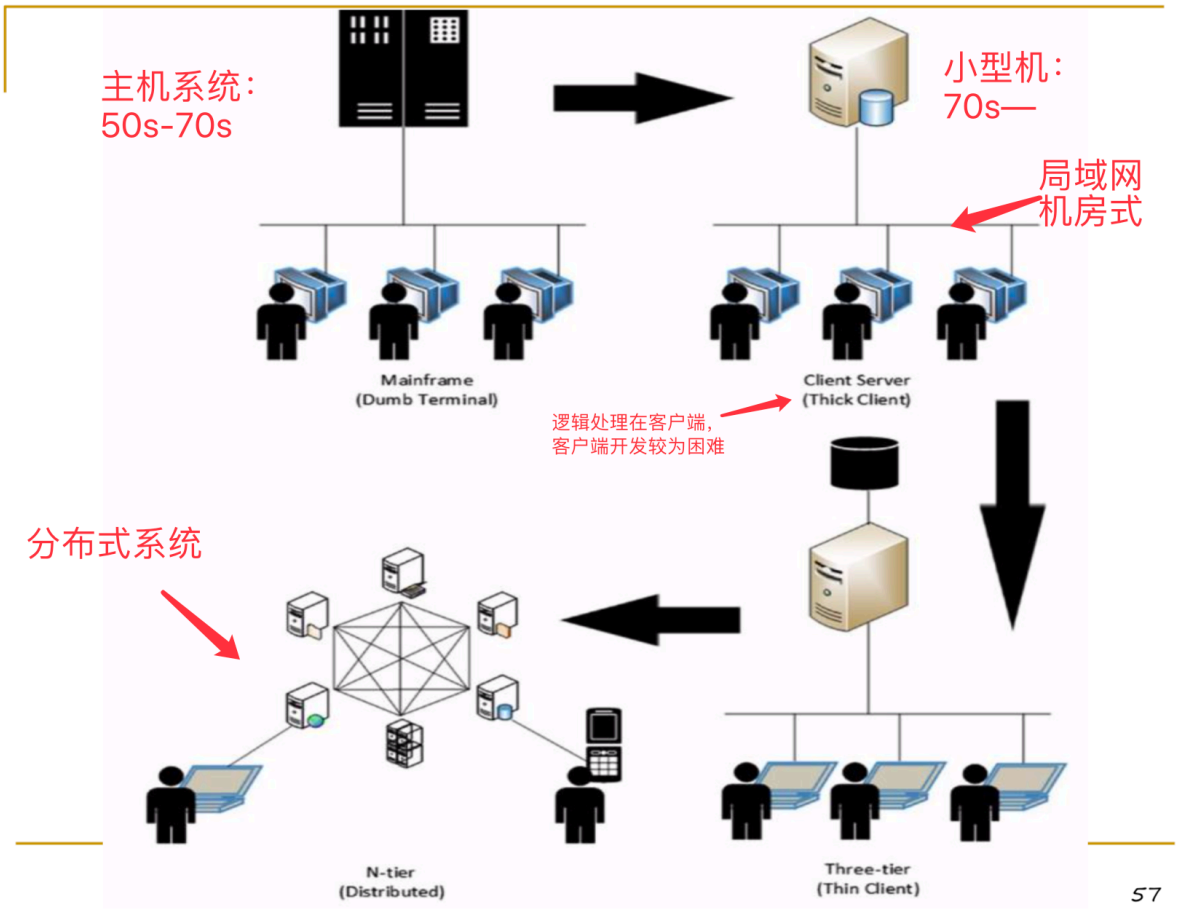
- 数据库
- 内存方式：session、indexDB.....

C/S架构

主要有三个组成部分：

1. 数据库服务器
2. 客户应用程序
3. 网络

CS发展历史



C/S风格的优点

- 数据管理和应用分开，安全性高。
- 数据的处理效率高，局域网内效果明显。
- 对于整个计算机系统的扩充较为灵活

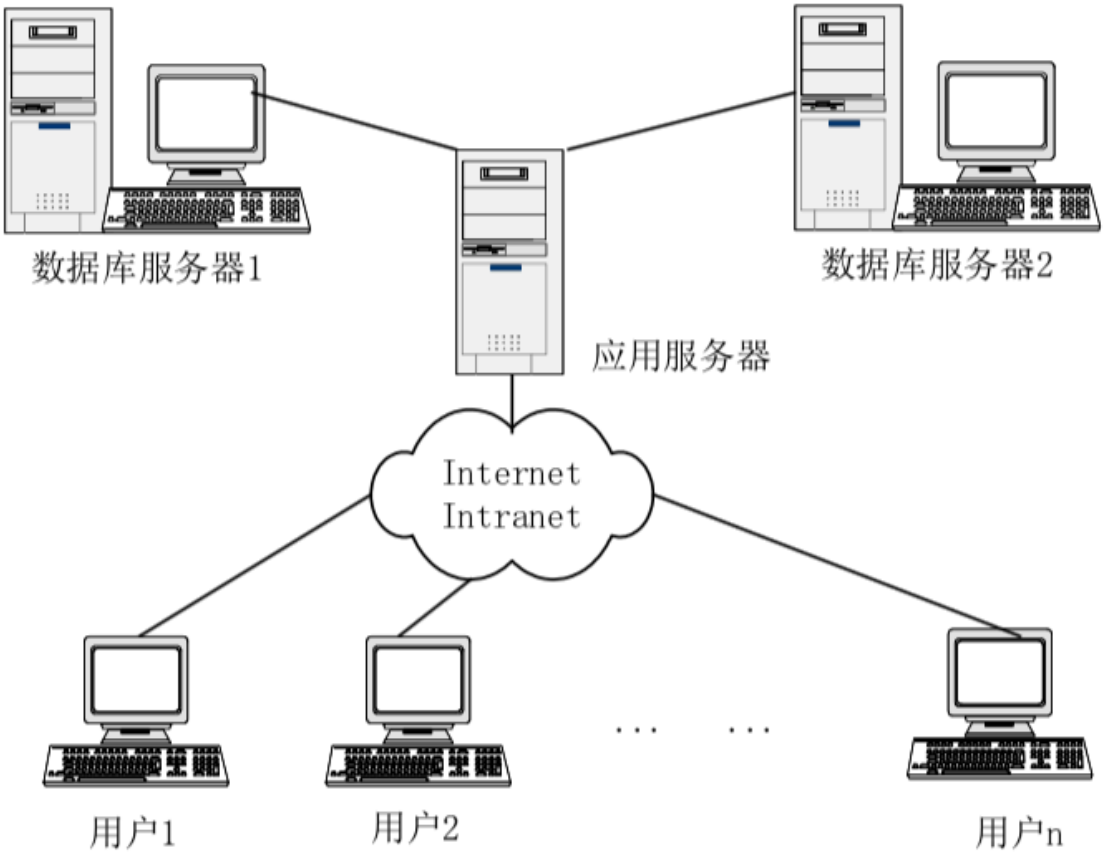
C/S风格的缺点

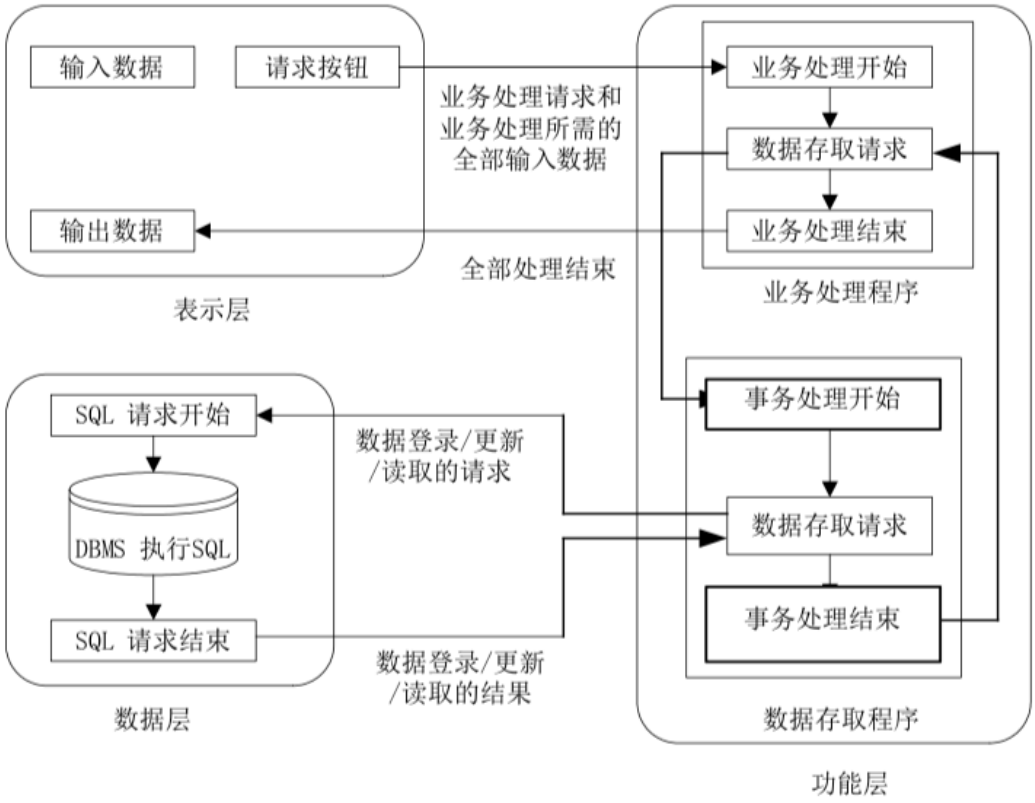
- 开发成本较高
- 客户端设计复杂
- 移植性、可维护性较差
- 软件升级困难
- 新技术不能轻易应用

三层CS架构——面向构件的软件开发

解决CS架构的缺点，远程调用类的实例，属于实例与实例之间的调用。

出现了应用服务器、对应的软件体系结构是三次C/S风格





这种结构的优点是

便于维护