# 旅游景点售票系统软件体系结构文档

## 引言

随着我国经济的愈加繁荣，旅游产业迅猛发展，我国国内旅游消费及旅游业总收入的增长速度一直高位运行于居民消费支出和国内生产总值之上，人们对于旅行体验的要求也逐步提升。当前许多旅行相关的软件存在以下一些问题：

1. 功能过于繁杂，广大商家与用户之间缺乏交流
2. 热门景点过于优先，小众景点缺乏，不利于小众景点与新景点的推广
3. 平台干预过多
4. 用户评价较少

因此，我们希望开发一款主要面向商家和用户的软件，实现景点全方位覆盖与自由变更，提升商家和用户的体验。

### 1.1 标识

本系统适用于chrome 90.0，firefox 82.0等浏览器使用。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文件状态：  [ ]草稿  [ √ ]正式发布  [ ]正在修改 | 报告编号 | 旅游景点售票系统概要设计说明书  Travel-Mate | | |
| 当前版本 | V1.0.0 | | |
| 编写人 |  | 编写日期 |  |
| 审批人 |  | 审批日期 |  |
| 保密级别 | …… | | |

### 1.2 系统概述

针对目前旅行软件仍存在的许多问题，我们主要提供如下功能：

1. 景点分类管理（可以涉及多级分类 城市 景点or酒店）
2. 商家需要必要营业信息注册
3. 商家可以上传景点信息、添加景点
4. 顾客需要注册账号
5. 顾客可以根据城市、景点名称、景点评分查询，选择购买门票，可以对景点进行评价or打分
6. 系统能够对指定时间、城市、景点名字进行统计分析，以便分析出受欢迎的景点

与现有旅行软件相比，我们的软件主要面向商家和用户，能实现景点全方位覆盖与自由变更，提升商家和用户的体验。

### 1.3 文档概述

通过本项目的系统分析，可以深入了解软件系统的开发内容和开发过程。将所学的前端开发技术HTML、CSS、JavaScript等运用到本项目中去，后台使用数据库MySQL xx.xx.xx。同时开发过程要求运用软件工程方法和Case工具，运行环境：chrome 90.0，firefox 82.0等浏览器

### 1.4 基线

**1**. 旅游景点售票系统软件开发计划书 V1.0

2. 旅游景点售票系统需求分析报告 V1.0

3. 旅游景点售票系统软件测试计划 V1.0

## 2 引用文件

1.中华人民共和国国家标准GB T-8567-2006

## 3总体设计

### 3.1 基本功能

#### 3.1.1用户登录

用户登录注册: 对用户输入的用户名，密码进行验证，提交注册申请，申请账号.对用户输入的用户名，密码进行验证，验证通过后，该用户可以使用旅游景点售票系统中自己拥有权限的那部分功能，否则拒绝使用。

#### 3.1.2系统管理员功能

1. 商家管理

管理员修改，删除，新增或查询商家相关数据，系统根据管理员的操作，对商家资料进行更新或显示。

1. 顾客管理

管理员修改，删除，新增或查询顾客相关数据，系统根据管理员的操作，对顾客资料进行更新或显示。

1. 商家统计

统计商家售票收益，统计商家浏览量等，管理员可以将商家资料信息导出到Excel中。

1. 顾客统计

根据年龄、消费等情况统计顾客基本信息，管理员可以将顾客资料导出到Excel中。

#### 3.1.3顾客相关功能

1. 设置个人基本信息

顾客可以选择设置基本信息，例如性别、年龄、偏好、个人介绍等，系统根据顾客的相关操作，对该顾客的相关资料进行更新或显示。

1. 景点搜索查询

顾客可以选择城市，进行搜索查询景点，系统根据顾客的相关操作，对查询景点的相关资料进行显示。

1. 购票

顾客选择景点之后，可以浏览票价信息，进行购票操作，系统会根据顾客的操作，对购票情况、结果进行相关操作和显示。

1. 评价

顾客可以根据自己的选择对相关景点进行评价，与系统上的其他顾客进行交流分享，系统会根据用户的相关操作，更新或展示景点评价。

#### 3.1.4商家相关功能

1. 基本信息管理

商家可以上传、查看、更新营业证明，公司名称，公司注册资料等

1. 景点信息管理

商家可以查看、更新当前上传景点信息，或新增景点信息，包括介绍、图片等，系统会根据商家的操作，进行相关的更新和展示。商家可以查看、更新当前售票的票价、优惠、组合套票等相关信息，系统会根据商家的操作，进行相关的更新和展示。

1. 售票统计

商家可以选择导出在本景点消费的顾客的年龄、消费等基本信息统计。

商家可以选择导出本天、本月、本季度、本年的售票情况，收益金额等，也可统计每个时间段对景点的浏览量等相关信息，并可以选择导出到相关文件，系统会根据商家的相关操作进行对应的处理。

### 3.2 附加功能

1. 账号管理功能：系统管理人员能够列表显示商家和顾客的账号信息、禁用选定的用户帐号，但不得删除用户帐号。
2. 顾客订票功能：能实时更新和显示门票剩余数量，对零数量的门票，只能显示在界面，但顾客不能购买。
3. 订单管理功能：顾客订单列表显示所有订单，其中可分为：已付款、未使用、已使用三种状态。
4. 电子邮件服务：系统管理员能够通过及时有效的错误信息邮件，进行维护，使软件安全运行。

### 3.3运行环境

硬件要求：Cup Pentium 2.0GHZ 以上、512M内存以上，至少10G硬盘

系统运行环境：Windows server 2003、Windows XP SP2、IIS6、.NetFramework 2.0

### 3.4 基本设计概念和处理流程

旅游售票管理系统包括以下几大模块：

* 注册/登录
* 系统管理员功能
  + - 商家管理
    - 顾客管理
    - 商家统计
    - 顾客统计
* 顾客相关功能
  + - 设置个人基本信息
    - 景点搜索查询
    - 购票
    - 评价
* 商家相关功能
  + - 基本信息管理
    - 景点信息管理
    - 售票统计

## 4各模块结构

整个项目结构如下图4－1所示：

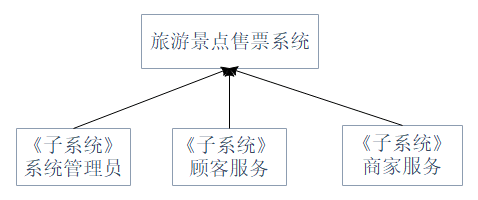


图4-1 网上购书系统项目结构图

### 4.1用户登录管理模块

用户登录管理模块图如图4-2所示。

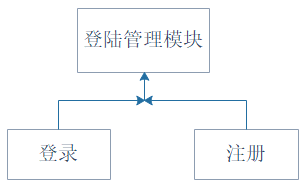


图4-2 用户管理模块图

### 4.2系统管理员

**（1）商家管理模块**

管理员修改，删除，新增或查询商家相关数据，系统根据管理员的操作，对商家资料进行更新或显示。如图4-3所示：

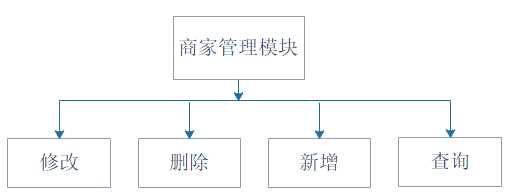


图4-3 商家管理模块

**（2）顾客管理模块**

管理员修改，删除，新增或查询顾客相关数据，系统根据管理员的操作，对顾客资料进行更新或显示。如图4-4所示：

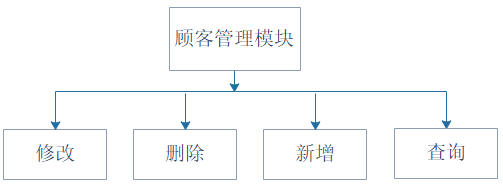


图4-4 顾客管理模块

**（3）商家统计模块**

通过选择统计条件，统计商家售票收益，统计商家浏览量等信息，管理员可以将商家资料信息导出到Excel中。如图4-5所示：

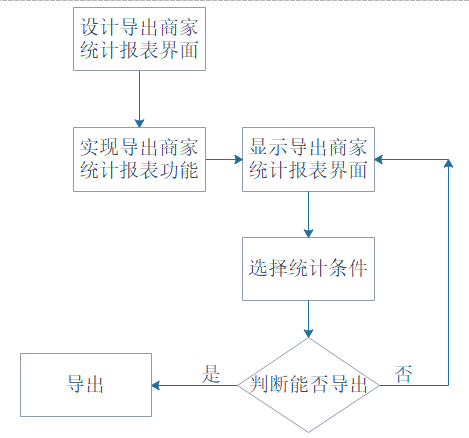


图4-5 商家统计模块

**（4）顾客统计模块**

通过选择统计条件，根据年龄、消费等情况统计顾客基本信息，管理员可以将顾客资料导出到Excel中。

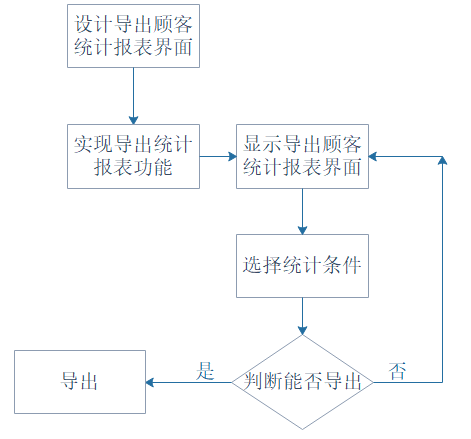


图4-6 顾客统计模块

### 4.3顾客服务

**（1）个人信息模块**

顾客可以选择设置基本信息，例如性别、年龄、偏好、个人介绍等，系统根据顾客的相关操作，对该顾客的相关资料进行更新或显示。如图4-7所示：

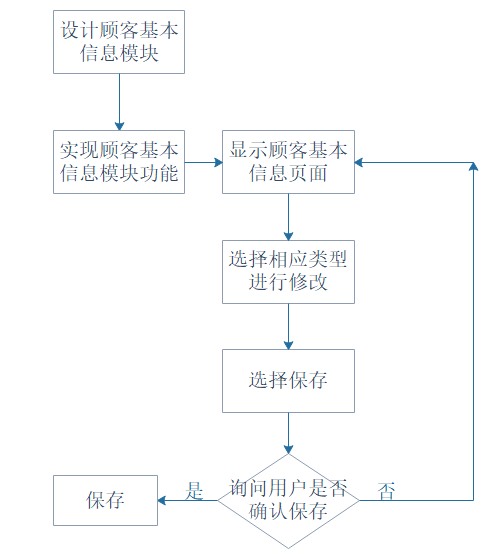
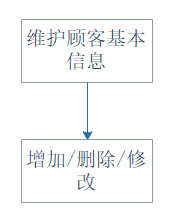


图4-7 顾客基本信息模块

**（2）景点搜索模块**

顾客可以选择城市，进行搜索查询景点，系统根据顾客的相关操作，对查询景点的相关资料进行显示。如图4-8所示：

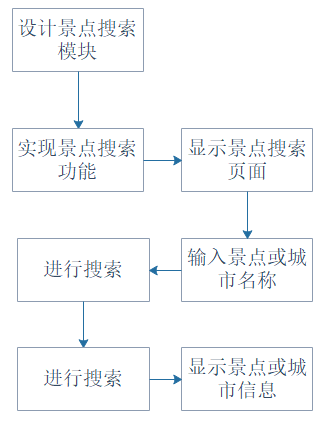


图4-8 景点搜索模块

**（3）购票模块**

顾客选择景点之后，可以浏览票价信息，进行购票操作，系统会根据顾客的操作，对购票情况、结果进行相关操作和显示。如图4-9所示：

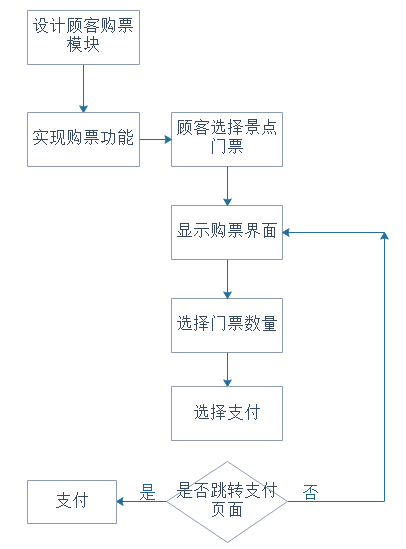


图4-9 顾客购票模块

**（4）评价模块**

顾客可以根据自己的选择对相关景点进行评价，与系统上的其他顾客进行交流分享，系统会根据用户的相关操作，更新或展示景点评价。如图4-10所示：

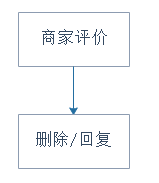
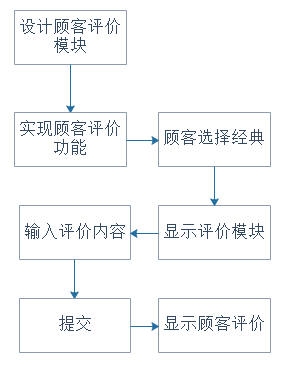


图4-10 顾客评价模块

### 4.4商家服务

**（1）基本信息管理模块**

商家可以上传、查看、更新营业证明，公司名称，公司注册资料等。如图4-11所示：

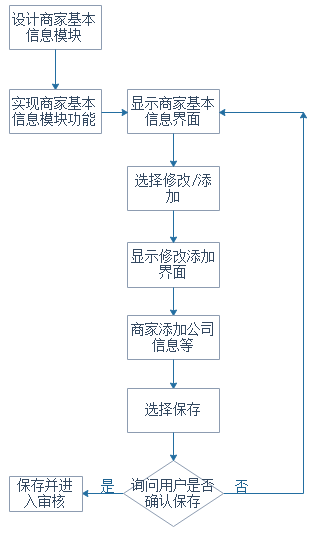


图4-11 商家基本信息模块

**（2）景点信息管理模块**

商家可以查看、更新当前上传景点信息，或新增景点信息，包括介绍、图片等，系统会根据商家的操作，进行相关的更新和展示。商家可以查看、更新当前售票的票价、优惠、组合套票等相关信息，系统会根据商家的操作，进行相关的更新和展示。如图4-12所示：

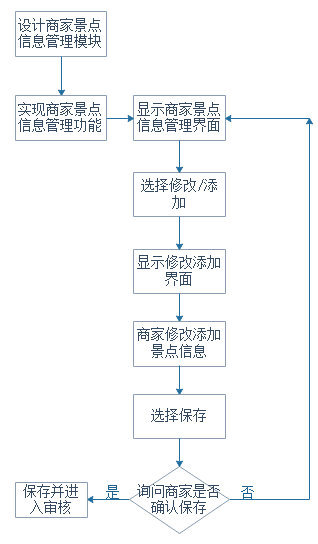


图4-12 商家景点信息管理模块

**（3）售票统计模块**

商家可以选择导出在本景点消费的顾客的年龄、消费等基本信息统计。商家可以选择导出本天、本月、本季度、本年的售票情况，收益金额等，也可统计每个时间段对景点的浏览量等相关信息，并可以选择导出到相关文件，系统会根据商家的相关操作进行对应的处理。如图4-13所示：

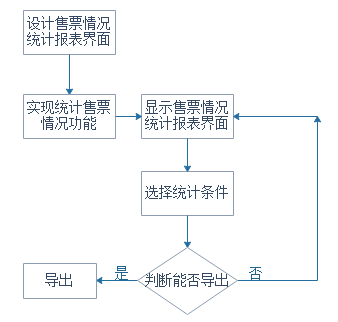


图4-13 售票统计模块

## 5 利用MySQL设计数据库

### 5.1创建数据库：MySQL

### 5.2创建表

表4－1 Scenic spot

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **列名** | **数据类型** | **长度** | **小数位** | **标识** | **主键** | **允许空** | **默认值** | **说明** |
| 1 | Scene\_id | int |  |  |  | √ |  |  |  |
| 2 | Scene\_name | string |  |  |  |  |  |  |  |
| 3 | Price | double |  | 2 |  |  |  |  |  |
| 4 | Address | string |  |  |  |  |  |  |  |
| 5 | Scene\_picture |  |  |  |  |  |  |  |  |
| 6 | Remaining\_tickets | int |  |  |  |  |  |  |  |
| 7 | View\_count | int |  |  |  |  |  |  |  |
| 8 | Favorite\_count | int |  |  |  |  |  |  |  |
| 9 | Mechant\_id | int |  |  |  |  |  |  |  |
| 10 | City\_name | string |  |  |  |  |  |  |  |
| 11 | City\_id | int |  |  |  |  |  |  |  |

表4－2 mechant

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **列名** | **数据类型** | **长度** | **小数位** | **标识** | **主键** | **允许空** | **默认值** | **说明** |
| 1 | Mechant\_ID | int |  |  |  | √ |  |  |  |
| 2 | Mechant\_name | string |  |  |  |  |  |  |  |
| 3 | password | string |  |  |  |  |  |  |  |
| 4 | Phone\_number | string |  |  |  |  | √ |  |  |

表4－3 order

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **列名** | **数据类型** | **长度** | **小数位** | **标识** | **主键** | **允许空** | **默认值** | **说明** |
| 1 | Order\_id | int |  |  |  | √ |  |  |  |
| 2 | User\_id | int |  |  |  |  |  |  |  |
| 3 | Mechant\_id | int |  |  |  |  |  |  |  |
| 4 | Total\_price | double |  | 2 |  |  |  |  |  |
| 5 | Total\_num | int |  |  |  |  |  |  |  |
| 6 | Scene\_id | int |  |  |  |  |  |  |  |

表4－4 user

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **列名** | **数据类型** | **长度** | **小数位** | **标识** | **主键** | **允许空** | **默认值** | **说明** |
| 1 | User\_id | int |  |  |  | √ |  |  |  |
| 2 | User\_name | string |  |  |  |  |  |  |  |
| 3 | password | string |  |  |  |  |  |  |  |
| 4 | gender | bool |  |  |  |  |  |  |  |
| 5 | caption | string |  |  |  |  |  |  |  |
| 6 | age | int |  |  |  |  |  |  |  |

表4－5 shopping cart

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **列名** | **数据类型** | **长度** | **小数位** | **标识** | **主键** | **允许空** | **默认值** | **说明** |
| 1 | User\_id | int |  |  | √ |  |  |  |  |
| 2 | Scene\_id | sysname |  |  | √ |  |  |  |  |
| 3 | Total\_num | int |  |  |  |  | √ |  |  |
| 4 | Total\_price | int |  | 2 |  |  |  |  |  |

表4－6 post

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **列名** | **数据类型** | **长度** | **小数位** | **标识** | **主键** | **允许空** | **默认值** | **说明** |
| 1 | Post\_id | int |  |  |  | √ |  |  |  |
| 2 | content | 富文本形式 |  |  |  |  |  |  |  |
| 3 | User\_id | int |  |  |  |  |  |  |  |
| 4 | Publish\_time | time |  |  |  |  |  |  |  |
| 5 | Order\_id | int |  |  |  |  |  |  |  |

表4－7 comment

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **列名** | **数据类型** | **长度** | **小数位** | **标识** | **主键** | **允许空** | **默认值** | **说明** |
| 1 | Comment\_id | int |  |  |  | √ |  |  |  |
| 2 | Comment\_content | string | 50 |  |  |  |  |  |  |
| 3 | Post\_id | int |  |  |  |  |  |  |  |
| 4 | Comment\_time | time |  |  |  |  |  |  |  |
| 5 | Order\_id | int |  |  |  |  |  |  |  |

## 6 数据库安全性

### 6.1概述

我们采用了MariaDB数据库，它是MySQL关系数据库管理系统的一个分支，相较于 MySQL 本身，具有更丰富的功能及更优越的性能。

对于安全性，我们采用了以下措施：

* 为数据库root用户设置或重置密码
* 禁止匿名用户登录
* 禁止root用户的远程访问，只允许本地localhost访问
* 删除test数据库（任何人都可以访问的数据库）
* 激活以上1～4选项

### 6.2 数据库验证方式

应用程序端连接数据库时使用的数据库验证方式为SQL Server验证。数据库连接串将使用CAPICOM技术加密存储在系统配置文件中。

## 7 数据库管理和设置

### 7.1 概述

本节内容与数据库功能无关，将从系统的数据库运行角度出发，对本数据库的属性设置和备份恢复策略提出指导性说明。

另外，我们在数据库的物理结构上还做了优化：数据库表设计满足 3NF，减少了冗余数据。表中的所有数据元素不但要能唯一地被主关键字所标识,而且它们之间还必须相互独立，不存在其他的函数关系接口。

### 7.2 数据库属性设置

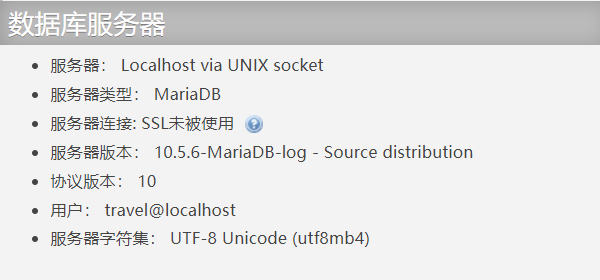


图4－11 数据库属性设置图

### 7.3 数据库备份恢复策略

目前的方案是使用定时脚本进行数据库的定期自动备份和事件清理。

如果时间允许，我们考虑部署多台服务器，同时部署MariaDB Galera Cluster（MariaDB 同步多主机集群），实现主从备份，保证数据的稳定性。

主要功能:

* 同步复制
* 真正的 multi-master，即所有节点可以同时读写数据库
* 自动的节点成员控制，失效节点自动被清除
* 新节点加入数据自动复制
* 真正的并行复制，行级
* 用户可以直接连接集群，使用感受上与MySQL完全一致

优势:

* 因为是多主，所以不存在Slavelag(延迟)
* 不存在丢失事务的情况
* 同时具有读和写的扩展能力
* 更小的客户端延迟
* 节点间数据是同步的,而 Master/Slave 模式是异步的,不同 slave 上的 binlog 可能是不同的

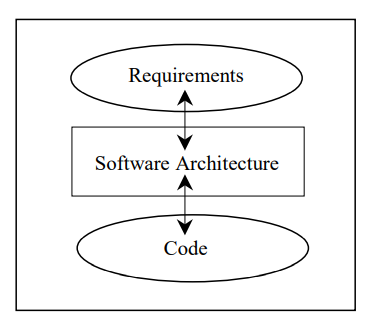
### 7.4 数据库运行日常维护

数据库维护是指当一个数据库被创建以后的工作都叫做数据库维护。包括备份系统数据、恢复数据库系统、产生用户信息表，并为信息表授权、监视系统运行状况，及时处理系统错误、保证系统数据安全，周期更改用户口令，数据库维护比数据库的创建和使用更难。

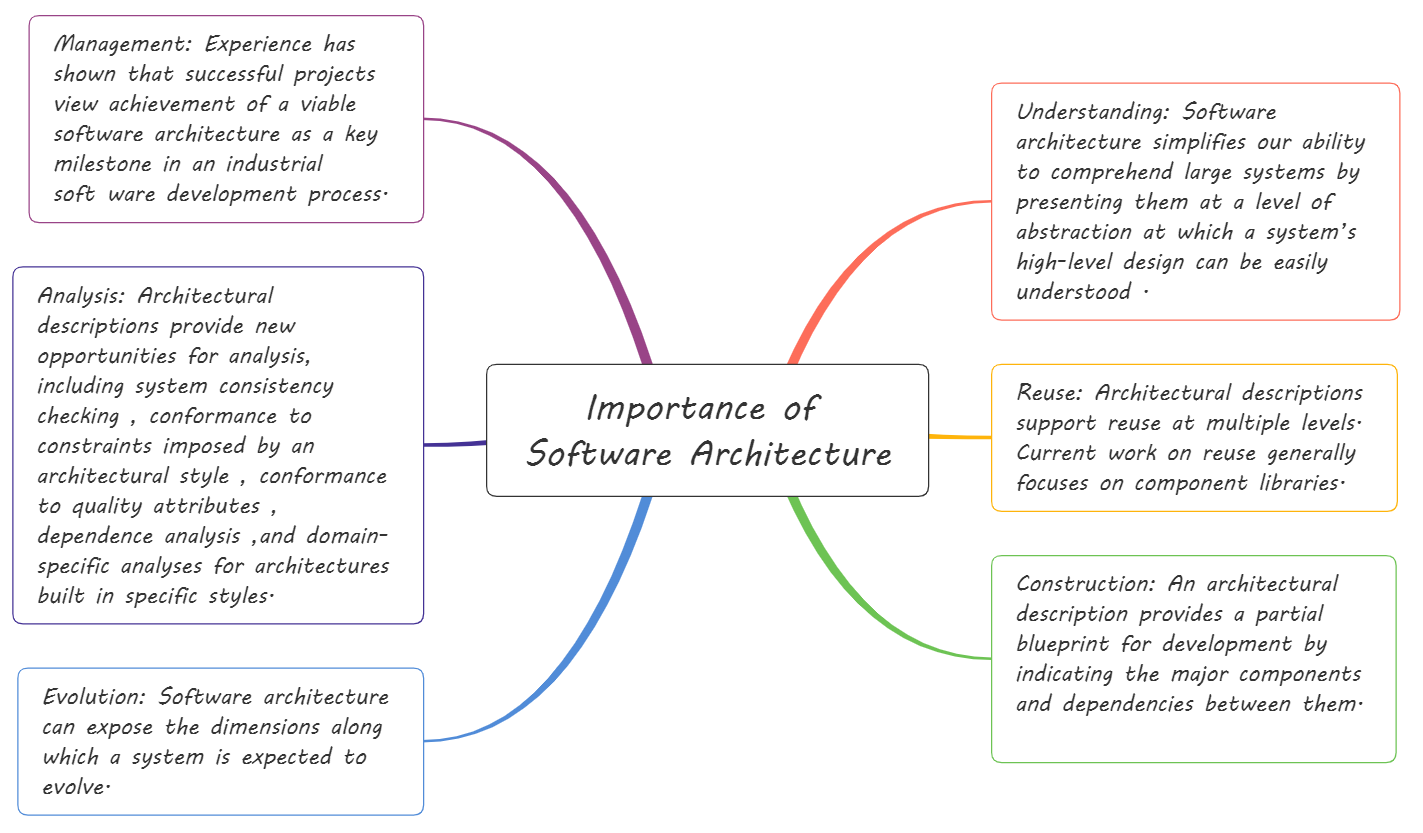
## 8成员学习报告

### 8.1 杜雅莉

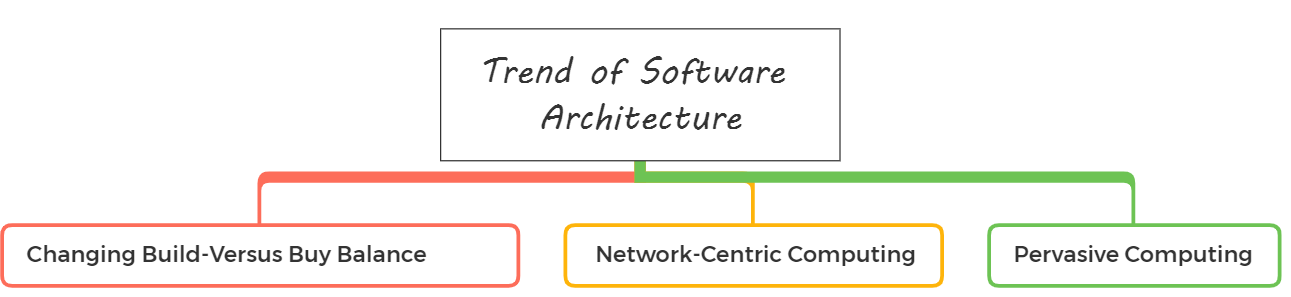
1. 软件体系结构设计的一个核心问题是能否使用重复的体系结构模式，即能够达到体系结构级的复用。软件体系结构是连接需求和代码的桥梁：



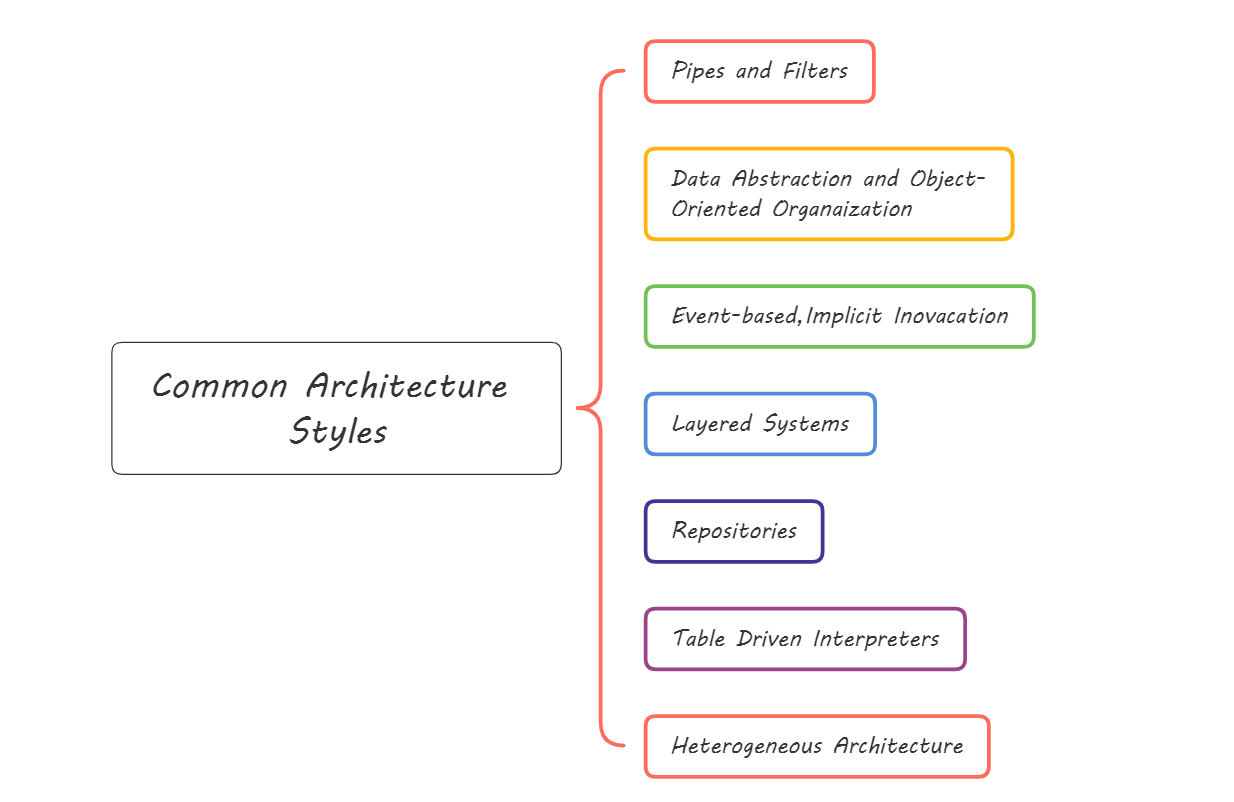
1. 软件体系结构的重要性



1. 软件体系结构未来发展趋势及方向



1. 软件体系结构从架构风格上可以分为以下几类：



* 管道和过滤器风格

在管道/过滤器风格的软件体系结构中，每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。因此，这里的构件被称为过滤器，这种风格的连接件就像是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。此风格特别重要的 过滤器必须是独立的实体，它不能与其它的过滤器共享数据，而且一个过滤器不知道它上游和下游的标识。一个管道/过滤器网络输出的正确性并不依赖于过滤器进 行增量计算过程的顺序。

管道/过滤器风格的软件体系结构具有许多很好的特点：

(1)使得软构件具有良好的隐蔽性和高内聚、低耦合的特点；

(2)允许设计者将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成；

(3)支持软件重用。重要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来；

(4)系统维护和增强系统性能简单。新的过滤器可以添加到现有系统中来；旧的可以被改进的过滤器替换掉；

(5)允许对一些如吞吐量、死锁等属性的分析；

(6)支持并行执行。每个过滤器是作为一个单独的任务完成，因此可与其它任务并行执行。

但是，这样的系统也存在着若干不利因素。

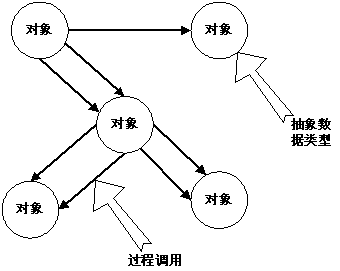
(1)通常导致进程成为批处理的结构。这是因为虽然过滤器可增量式地处理数据，但它们是独立的，所以设计者必须将每个过滤器看成一个完整的从输入到输出的转换。

(2)不适合处理交互的应用。当需要增量地显示改变时，这个问题尤为严重。

(3)因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性

* 数据抽象与面向对象风格

抽象数据类型概念对软件系统有着重要作用，目前软件界已普遍转向使用面向对象系统。这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。这种风格的构件是对象，或者说是抽象数据类型的实例。对象是一种被称作管理者的构件，因为它负责保持资源的完整性。对象是通过函数和过程的调用来交互的。



面向对象的系统的优点：

(1) 因为对象对其它对象隐藏它的表示，所以可以改变一个对象的表示，而不影响其它的对象。

(2) 设计者可将一些数据存取操作的问题分解成一些交互的代理程序的集合。

但是，面向对象的系统也存在着某些问题：

(1)为了使一个对象和另一个对象通过过程调用等进行交互，必须知道对象的标识。只要一个对象的标识改变了，就必须修改所有其他明确调用它的对象。

(2)必须修改所有显式调用它的其它对象，并消除由此带来的一些副作用。例如，如果A使用了对象B，C也使用了对象B，那么，C对B的使用所造成的对A的影响可能是料想不到的。

* 基于事件的隐式调用风格

基于事件的隐式调用风格的思想是构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

从体系结构上说，这种风格的构件是一些模块，这些模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。

基于事件的隐式调用风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式。

支持基于事件的隐式调用的应用系统很多。例如，在编程环境中用于集成各种工具，在数据库管理系统中确保数据的一致性约束，在用户界面系统中管理数据，以及在编辑器中支持语法检查。例如在某系 统中，编辑器和变量监视器可以登记相应Debugger的断点事件。当Debugger在断点处停下时，它声明该事件，由系统自动调用处理程序，如编辑程 序可以卷屏到断点，变量监视器刷新变量数值。而Debugger本身只声明事件，并不关心哪些过程会启动，也不关心这些过程做什么处理。

隐式调用系统的主要优点有：

(1)为软件重用提供了强大的支持。当需要将一个构件加入现存系统中时，只需将它注册到系统的事件中。

(2)为改进系统带来了方便。当用一个构件代替另一个构件时，不会影响到其它构件的接口。

隐式调用系统的主要缺点有：

(1)构件放弃了对系统计算的控制。一个构件触发一个事件时，不能确定其它构件是否会响应它。而且即使它知道事件注册了哪些构件的构成，它也不能保证这些过程被调用的顺序。

(2)数据交换的问题。有时数据可被一个事件传递，但另一些情况下，基于事件的系统必须依靠一个共享的仓库进行交互。在这些情况下，全局性能和资源管理便成了问题。

(3)既然过程的语义必须依赖于被触发事件的上下文约束，关于正确性的推理存在问题。

* 层次系统风格

层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机(在另一些层次系统中层是部分不透明的)。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。

这种风格支持基于可增加抽象层的设计。这样，允许将一个复杂问题分解成一个增量步骤序列的实现。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，同样为软件重用提供了强大的支持。

层次系统有许多可取的属性：

(1)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；

(2)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；

(3)支持重用。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。

但是，层次系统也有其不足之处：

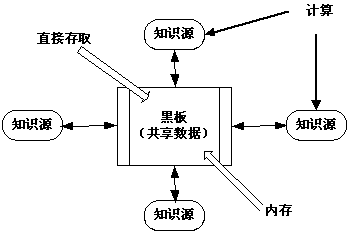
(1)并不是每个系统都可以很容易地划分为分层的模式，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或高级的功能综合起来；

(2)很难找到一个合适的、正确的层次抽象方法。

* 仓库风格

在仓库风格中，有两种不同的构件：中央数据结构说明当前状态，独立构件在中央数据存贮上执行，仓库与外构件间的相互作用在系统中会有大的变化。

控制原则的选取产生两个主要的子类。若输入流中某类时间触发进程执行的选择，则仓库是一传统型数据库；另一方面，若中央数据结构的当前状态触发进程执行的选择，则仓库是一黑板系统。



黑板系统主要由三部分组成：

(1)知识源。知识源中包含独立的、与应用程序相关的知识，知识源之间不直接进行通讯，它们之间的交互只通过黑板来完成。

(2)黑板数据结构。黑板数据是按照与应用程序相关的层次来组织的解决问题的数据，知识源通过不断地改变黑板数据来解决问题。

(3)控制。控制完全由黑板的状态驱动，黑板状态的改变决定使用的特定知识。

* C2风格

C2体系结构风格可以概括为：通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2风格中的系统组织规则如下：

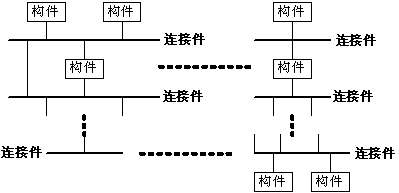
(1)系统中的构件和连接件都有一个顶部和一个底部；

(2)构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；

(3)一个连接件可以和任意数目的其它构件和连接件连接；

(4)当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

如下图所示，图中构件与连接件之间的连接体现了C2风格中构建系统的规则。



C2风格是最常用的一种软件体系结构风格。从C2风格的组织规则和结构图中，我们可以得出，C2风格具有以下特点：

(1)系统中的构件可实现应用需求，并能将任意复杂度的功能封装在一起；

(2)所有构件之间的通讯是通过以连接件为中介的异步消息交换机制来实现的；

(3)构件相对独立，构件之间依赖性较少。系统中不存在某些构件将在同一地址空间内执行，或某些构件共享特定控制线程之类的相关性假设。

* 三层C/S结构

三层C/S结构是将应用功能分成表示层、功能层和数据层三部分。其解决方案是：对这三层进行明确分割，并在逻辑上使其独立。原来的数据层作为DBMS已经独立出来，所以关键是要将表示层和功能层分离成各自独立的程序，并且还要使这两层间的接口简洁明了。三层C/S结构各层间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说也达不到所要求的性能。此外，设计时必须慎重考虑三层间的通信方法、通信频度及数据量。这和提高各层的独立性一样是三层C/S结构的关键问题。

**三层C/S的功能：**

1.表示层

表示层是应用的用户接口部分，它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据，显示应用输出的数据。为使用户能直观地进行操作，一 般要使用图形用户接口(GUI)，操作简单、易学易用。在变更用户接口时，只需改写显示控制和数据检查程序，而不影响其他两层。检查的内容也只限于数据的 形式和值的范围，不包括有关业务本身的处理逻辑。

图形界面的结构是不固定的，这便于以后能灵活地进行变更。例如，在一个窗口中不是放入几个功能，而是按功能分割窗口，以便使每个窗口的功能简洁单纯。在这层的程序开发中主要是使用可视化编程工具。

2. 功能层

功能层相当于应用的本体，它是将具体的业务处理逻辑地编入程序中。例如，在制作订购合同的时要计算合同金额，按照定好的格式配置数据、打印订购合同，而 处理所需的数据则要从表示层或数据层取得。表示层和功能层之间的数据交往要尽可能简洁。例如，用户检索数据时，要设法将有关检索要求的信息一次传送给功能 层(参见图2)，而由功能层处理过的检索结果数据也一次传送给表示层。在应用设计中，一定要避免进行一次业务处理，在表示层和功能层间进行多几次数据交换 的笨拙设计。

通常，在功能层中包含有：确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。这层的程序多半是用可视化编程工具开发的，也有使用COBOL和C语言的。

3. 数据层

数据层就是DBMS，负责管理对数据库数据的读写。DBMS必须能迅速执行大量数据的更新和检索。现在的主流是关系数据库管理系统(RDBMS)。因此，一般从功能层传送到数据层的要求大都使用SQL语言。

**三层C/S结构的优点：**

1。 具有灵活的硬件系统构成

对于各个层可以选择与其处理负荷和处理特性相适应的硬件。这是一个与系统可缩放性直接相关的问题。例如，最初用一台Unix工作站作为服务器，将数据层 和功能层都配置在这台服务器上。随着业务的发展，用户数和数据量逐渐增加，这时就可以将Unix工作站作为功能层的专用服务器，另外追加一台专用于数据层 的服务器。若业务进一步扩大，用户数进一步增加，则可以继续增加功能层的服务器数目，用以分割数据库。清晰、合理地分割三层结构并使其独立，可以使系统构 成的变更非常简单。因此，被分成三层的应用基本上不需要修正。

2。 提高程序的可维护性

三层C/S结构中，应用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。

3。 利于变更和维护应用技术规范

因为是按层分割功能，所以各个程序的处理逻辑变得十分简单。

4。 进行严密的安全管理

越关键的应用，用户的识别和存取权限设定愈重要。在三层C/S结构中，识别用户的机构是按层来构筑的，对应用和数据的存取权限也可以按层进行设定。例如，即使外部的入侵者突破了表示层的安全防线，若在功能层中备有另外的安全机构，系统也可以阻止入侵者进入其他部分。

此外，系统管理简单，可支持异种数据库，有很高的可用性。

C/S和B/S 的优缺点比较：

C/S和B/S是当今世界开发模式技术架构的两大主流技术。C/S是美国 Borland公司最早研发，B/S是美国微软公司研发。

1、C/S架构软件的优势与劣势

（1）、应用服务器运行数据负荷较轻。

最简单的C/S体系结构的数据库应用由两部分组成，即客户应用程序和数据库服务器程序。二者可分别称为前台程序与后台程序。运行数据库服务器程序 的机器，也称为应用服务器。一旦服务器程序被启动，就随时等待响应客户程序发来的请求；客户应用程序运行在用户自己的电脑上，对应于数据库服务器，可称为 客户电脑，当需要对数据库中的数据进行任何操作时，客户程序就自动地寻找服务器程序，并向其发出请求，服务器程序根据预定的规则应答，送回结果，应用 服务器运行数据负荷较轻。

（2）、数据的储存管理功能较为透明。

在数据库应用中，数据的储存管理功能，是由服务器程序和客户应用程序分别独立进行的，前台应用可以违反的规则，并且通常把那些不同的（不管是已知 还是未知的）运行数据，在服务器程序中不集中实现，例如访问者的权限，编号可以重复、必须有客户才能建立定单这样的规则。所有这些，对于工作在前台程序上 的最终用户，是“透明”的，他们无须过问（通常也无法干涉）背后的过程，就可以完成自己的一切工作。在客户服务器架构的应用中，前台程序不是非常“瘦小 ”，麻烦的事情都交给了服务器和网络。在C/S体系的下，数据库不能真正成为公共、专业化的仓库，它受到独立的专门管理。

（3）、C/S架构的劣势是高昂的维护成本且投资大。

首先，采用C/S架构，要选择适当的数据库平台来实现数据库数据的真正“统一”，使分布于两地的数据同步完全交由数据库系统去管理，但逻辑上两地 的操作者要直接访问同一个数据库才能有效实现，有这样一些问题，如果需要建立“实时”的数据同步，就必须在两地间建立实时的通讯连接，保持两地的数据库服 务器在线运行，网络管理工作人员既要对服务器维护管理，又要对客户端维护和管理，这需要高昂的投资和复杂的技术支持，维护成本很高，维护任务量大。

其次，传统的C/S结构的软件需要针对不同的操作系统系统开发不同版本的软件，由于产品的更新换代十分快，代价高和低效率已经不适应工作需要。在JAVA这样的跨平台语言出现之后，B/S架构更是猛烈冲击C/S，并对其形成威胁和挑战。

2、B/S架构软件的优势与劣势

（1）、维护和升级方式简单。

目前，软件系统的改进和升级越来越频繁，B/S架构的产品明显体现着更为方便的特性。对一个稍微大一点单位来说，系统管理人员如果需要在几百甚至 上千部电脑之间来回奔跑，效率和工作量是可想而知的，但B/S架构的软件只需要管理服务器就行了，所有的客户端只是浏览器，根本不需要做任何的维护。无论 用户的规模有多大，有多少分支机构都不会增加任何维护升级的工作量，所有的操作只需要针对服务器进行；如果是异地，只需要把服务器连接专网即可，实现远程 维护、升级和共享。所以客户机越来越“瘦”，而服务器越来越“胖”是将来信息化发展的主流方向。今后，软件升级和维护会越来越容易，而使用起来会越来越简单，这对用户人力、物力、时间、费用的节省是显而易见的，惊人的。因此，维护和升级革命的方式是“瘦”客户机，“胖”服务器。

（2）、成本降低，选择更多。

大家都知道windows在桌面电脑上几乎一统天下，浏览器成为了标准配置，但在服务器操作系统上windows并不是处于绝对的统治地位。现在的趋势是凡使用B/S架构的应用管理软件，只需安装在Linux服务器上即可，而且安全性高。所以服务器操作系统的选择是很多的，不管选用那种操作系统都 可以让大部分人使用windows作为桌面操作系统电脑不受影响，这就使的最流行免费的Linux操作系统快速发展起来，Linux除了操作系统是免费的 以外，连数据库也是免费的，这种选择非常盛行。

（3）、应用服务器运行数据负荷较重。

由于B/S架构管理软件只安装在服务器端（Server）上，网络管理人员只需要管理服务器就行了，用户界面主要事务逻辑在服务器 （Server）端完全通过WWW浏览器实现，极少部分事务逻辑在前端（Browser）实现，所有的客户端只有浏览器，网络管理人员只需要做硬件维护。 但是，应用服务器运行数据负荷较重，一旦发生服务器“崩溃”等问题，后果不堪设想。因此，许多单位都备有数据库存储服务器，以防万一。

C/S 与 B/S 区别：

Client/Server是建立在局域网的基础上的，Browser/Server是建立在广域网的基础上的。

（1）硬件环境不同：

C/S 一般建立在专用的网络上， 小范围里的网络环境， 局域网之间再通过专门服务器提供连接和数据交换服务。

B/S 建立在广域网之上的， 不必是专门的网络硬件环境，例如电话上网， 租用设备， 信息自己管理， 有比C/S更强的适应范围， 一般只要有操作系统和浏览器就行。

（2）、对安全要求不同

C/S 一般面向相对固定的用户群， 对信息安全的控制能力很强。 一般高度机密的信息系统采用C/S 结构适宜，可以通过B/S发布部分可公开信息。

B/S 建立在广域网之上， 对安全的控制能力相对弱， 面向是不可知的用户群。

（3）、对程序架构不同

C/S 程序可以更加注重流程，可以对权限多层次校验，对系统运行速度可以较少考虑。

B/S 对安全以及访问速度的多重的考虑， 建立在需要更加优化的基础之上。 比C/S有更高的要求，B/S结构的程序架构是发展的趋势，从MS的。Net系列的BizTalk 2000 Exchange 2000等，全面支持网络的构件搭建的系统。SUN和IBM推的JavaBean构件技术等，使B/S更加成熟。

（4）、软件重用不同

C/S 程序可以不可避免的整体性考虑，构件的重用性不如在B/S要求下的构件的重用性好。

B/S 对的多重结构，要求构件相对独立的功能。能够相对较好的重用。就如买来的餐桌可以再利用，而不是做在墙上的石头桌子。

（5）、系统维护不同

系统维护是软件生存周期中，开销大，相当重要。C/S 程序由于整体性，必须整体考察，处理出现的问题以及系统升级难，可能是再做一个全新的系统。 B/S 构件组成方面构件个别的更换，实现系统的无缝升级。系统维护开销减到最小，用户从网上自己下载安装就可以实现升级。

（6）、处理问题不同

C/S 程序可以处理用户面固定，并且在相同区域， 安全要求高的需求，与操作系统相关， 应该都是相同的系统。 B/S 建立在广域网上，面向不同的用户群，分散地域， 这是C/S无法作到的，与操作系统平台关系最小。

（7）、用户接口不同

C/S 多是建立在Window平台上，表现方法有限，对程序员普遍要求较高。 B/S 建立在浏览器上， 有更加丰富和生动的表现方式与用户交流， 并且大部分难度减低，降低开发成本。

（8）、信息流不同

C/S 程序一般是典型的中央集权的机械式处理，交互性相对低。B/S 信息流向可变化， B－B、 B－C、 B－G等信息流向的变化，更像交易中心。

* 基于层次消息总线的架构风格

JB/HMB风格的基本特征

目前对软件体系结构的研究集中在以下方面：各种体系结构风格的汇编和总结、体系结

构描述语言(architectural description languages，简称ADLS)、体系结构的形式化基础、体系结构分析技术、基于体系结构的软件开发、体系结构恢复和再工程、支持体系结构设计的工具和环境及特定领域的软件体系结构等。 青鸟工程在“九五”期间，对基于构件构架模式的软件工业化生产技术进行了研究，并实现了青鸟软件生产线系统151。以青鸟软件生产线的实践为背景，提出了基于层次消息总线的软件体系结构(Jade bird hierarchical message bus based style，以下简称JB/HMB风格)，设计了相应的体系结构描述语言，开发了支持软件体系结构设计的辅助工具集，并研究了采用JB/HMB风格进行应用系统开发的过程框架。

JB/HMB风格的提出基于以下的实际背景：

(1) 随着计算机网络技术的发展，特别是分布式构件技术的日渐成熟和构件互操作标准的出现，如CORBA，DCOM和EJB等，加速了基于分布式构件的软件开发趋势，具有分布和并发特点的软件系统已成为一种普遍的应用需求。

(2) 基于事件驱动的编程模式已在图形用户界面程序设计中获得广泛应用。在此之前的

程序设计中，通常使用一个大的分支语句(switch Statement)控制程序的转移，对不同的输人情况分别进行处理，程序结构不甚清晰。基于事件驱动的编程模式在对多个不同事件响应的情况下，系统自动调用相应的处理函数，程序具有清晰的结构。

(3) 计算机硬件体系结构和总线的概念为软件体系结构的研究提供了很好的借鉴和启发，

REST架构风格

首先，REST是Web自身的架构风格。REST也是Web之所以取得成功的技术架构方面因素的总结。REST是世界上最成功的分布式应用架构风格。它是为运行在互联网环境 的 分布式超媒体系统量身定制的。互联网环境与企业内网环境有非常大的差别，最主要的差别是两个方面：

可伸缩性需求无法控制：并发访问量可能会暴涨，也可能会暴跌。

安全性需求无法控制：无法控制客户端发来的请求的格式，很可能会是恶意的请求。而所谓的“超媒体系统”，即，使用了超文本的系统。可以把“超媒体”理解为超文本+媒体内容。

### 8.2 王炳烨

关于软件体系结构的研究工作主要在国外展开的，国内到目前为止对于软件体系结构的研究尚处在起步阶段。软件体系结构在国内未引起人们广泛注意的原因主要有两点：

1. 软件体系结构从表面上看起来是一个老话题，似乎没有新东西。
2. 与国外相比，国内对大型和超大型复杂软件系统开发的经历相对较少，对软件危机的灾难性体会没有国外深刻，因而对软件体系结构研究的重要性和必要性的认识还不很充分。
3. 应用现状
4. 形成研究热点，仍处于非形式化水平

自20世纪90年代后期以来，软件体系结构的研究成为一个热点。广大软件工作者已经认识到软件体系结构研究的重大意义和它对软件系统设计开发的重要性，开展了很多研究和实践工作。

从软件体系结构研究的现状来看，当前的研究和对软件体系结构的描述，在很大程度上来说还停留在非形式化的基础上。软件构架师仍然缺乏必要的工具，这种工具应该是显式描述的、有独立性的形式化工具。

在目前通用的软件开发方法中，其描述通常是用非形式化的图和文本，不能描述系统期望的存在于构件之间的接口，不能描述不同的组成系统的组合关系的意义。难以被开发人员理解，更不能用来分析其一致性和完整性等特性。

当一个软件系统中的构件之间几乎以一种非形式化的方法描述时，系统的重用性也会受到影响，在设计一个系统结构过程中的努力很难移植到另一个系统中去。对系统构件和连接关系的结构化假设没有得到显式的、形式化的描述时，把这样的系统构件移植到另一个系统中去将是有风险的，甚至是不可能的。

1. 软件体系结构的形式化方法研究

软件体系结构研究如果仅仅停留在非形式化的框图阶段，已经难以适应进一步发展的需要。为支持基于体系结构的开发，需要有形式化建模符号、体系结构说明的分析与开发工具。从软件体系结构研究的现状来看，在这一领域近来已经有不少进展，其中比较有代表性的是美国卡耐基梅隆大学的Robert J．A11en于l997年提出的Wright系统。Wright是一种结构描述语言，该语言基于一种形式化的、抽象的系统模型，为描述和分析软件体系结构和结构化方法提供了一种实用的工具。Wright主要侧重于描述系统的软件构件和连接的结构、配置和方法。它使用显式的、独立的连接模型作为交互的方式，这使得该系统可以用逻辑谓词符号系统，而不依赖特定的系统实例来描述系统的抽象行为。该系统还可以通过一组静态检查来判断系统结构规格说明的一致性和完整性。从这些特性的分析来说，Wright系统的确适用于对大型系统的描述和分析。

1. 软件体系结构的建模研究

研究软件体系结构的首要问题是如何表示软件体系结构，即如何对软件体系结构建模。根据建模的侧重点的不同，可以将软件体系结构的模型分为5种：结构模型、框架模型、动态模型、过程模型和功能模型。在这5个模型中，最常用的是结构模型和动态模型。

1. 结构模型

这是一个最直观、最普遍的建模方法。这种方法以体系结构的构件、连接件和其他概念来刻画结构，并力图通过结构来反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质。研究结构模型的核心是体系结构描述语言。

1. 框架模型

框架模型与结构模型类似，但它不太侧重描述结构的细节而更侧重于整体的结构。框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构。

1. 动态模型

动态模型是对结构或框架模型的补充，研究系统的"大颗粒"的行为性质。例如，描述系统的重新配置或演化。动态可能指系统总体结构的配置、建立或拆除通信通道或计算的过程。这类系统常是激励型的。

1. 过程模型

过程模型研究构造系统的步骤和过程。因而结构是遵循某些过程脚本的结果。

1. 功能模型

该模型认为体系结构是由一组功能构件按层次组成，下层向上层提供服务。它可以看作是一种特殊的框架模型。

这5种模型各有所长，也许将5种模型有机地统一在一起，形成一个完整的模型来刻画软件体系结构更合适。例如，Kruchten在1995年提出了一个“4+1”的视角模型。“4+1”模型从5个不同的视角包括逻辑视角、过程视角、物理视角、开发视角和场景视角来描述软件体系结构。每一个视角只关心系统的一个侧面，5个视角结合在一起才能够反映系统的软件体系结构的全部内容。

1. 发展基于体系结构的软件开发模型

软件开发模型是跨越整个软件生存周期的系统开发、运行、维护所实施的全部工作和任务的结构框架，给出了软件开发活动各阶段之间的关系。目前，常见的软件开发模型大致可分为三种类型：

1. 以软件需求完全确定为前提的瀑布模型。
2. 在软件开发初始阶段只能提供基本需求时采用的渐进式开发模型,如螺旋模型等。
3. 以形式化开发方法为基础的变换模型。

所有开发方法都是要解决需求与实现之间的差距。但是，这三种类型的软件开发模型都存在这样或那样的缺陷，不能很好地支持基于软件体系结构的开发过程。因此，研究人员在发展基于体系结构的软件开发模型方面做了一定的工作。例如，为了形象地表示体系结构的生命周期，北京邮电大学的周莹新博士建立了一个软件体系结构的生命周期模型。

1. 软件产品线体系结构的研究

软件体系结构的开发是大型软件系统开发的关键环节。体系结构在软件生产线的开发中具有至关重要的作用，在这种开发生产中，基于同一个软件体系结构，可以创建具有不同功能的多个系统。在软件产品族之间共享体系结构和一组可重用的构件，可以增加软件工程和降低开发和维护成本。

一个产品线代表着一组具有公共的系统需求集的软件系统，它们都是根据基本的用户需求对标准的产品线构架进行定制，将可重用构件与系统独有的部分集成而得到的。采用软件生产线式模式进行软件生产，将产生巨型编程企业。但目前生产的软件产品族大部分是处于同一领域的。

1. 研究热点

当前，体系结构仍是一个非常新的研究领域，其概念还相当模糊。但软件体系结构作为软件工程领域中的一个组成部分，已经取得了长足的发展，受到大多数软件系统设计和研究人员的重视。

软件体系结构目前较活跃的研究方向包括：（1）软件体系结构形式基础的研究；（2）针对软件体系结构描述中特有的问题研究新的专门的高级语言；（3）建立用于度量和评价软件体系结构的模型和方法；（4）建立面向专门领域的软件体系结构范型库。（5）把软件体系结构从目前的直觉和经验状态过渡到理论。

1. 提供新的软件体系结构描述语言

在提高软件工程师对软件系统的描述和理解能力中，虽然软件体系结构描述起着重要作用，但这些抽象的描述通常是非形式化的和随意的。体系结构设计经常难以理解，难以适于进行形式化分析和模拟，缺乏相应的支持工具帮助构架师完成设计工作。为了解决这个问题，用于描述和推理的形式化语言得以发展，这些语言就叫做体系结构描述语言，ADLs寻求增加软件体系结构设计的可理解性和重用性。

ADL是这样一种语言，系统构架师可以利用它所提供的特性进行软件系统概念体系结构建模。ADL提供了具体的语法与刻画体系结构的概念框架。ADLs使得系统开发者能够很好地描述他们设计的体系结构，以便与人交流，能够用提供的工具对许多实例进行分析。

这种描述语言的目的就是提供一种规范化的体系结构描述，从而使得体系结构的自动化分析变得可能。研究人员已经设计出了近二十种ADLs，比较有影响力的有C2、UniCon、MetaH、Aesop、SADL、Rapide、Wright等。这些语言能够对体系结构连接器进行第一级抽象，同时还能描述模型的结构和内部构件之间的交互作用，并且还引入了一些新的系统分析模式。

1. 对软件体系结构的专门知识的整理

这方面的工作主要是对软件工程师在软件开发实践中得来的各种体系结构的原则、模式的整理和分类。例如，对软件体系结构风格的分类和比较，对体系结构描述语言的综合分析等。就目前看来，国内对软件体系结构的研究主要集中在对软件体系结构的专门知识的整理上。

1. 提供特定领域的体系结构框架

最近，开发特定的领域去为产品提供可重用框架日益受到关注。这些开发基于这样的想法：可以提取相关系统中的共同方面，以便可以通过低成本地把这些共同的设计实例化来构筑新系统。常见的例子有：

1. 编译器的标准分解。这个方法可以使一个本科生在一个学期时间内构造一个新的语言编译系统。
2. 标准化的通讯协议。这个可以使厂家通过在不同层次的抽象上提供服务来互相操纵。
3. 第四代语言。利用4GL开发出商务信息处理的通用范式。
4. 用户界面工具和框架。这个方法为开发者提供了一个可重用框架以及象菜单、对话框这样的可重用构件的集合。

软件体系结构充当一个理解系统构件和它们之间关系的框架，特别是那些始终跨越时间和实现的属性。这个理解对于现在系统的分析和未来系统的综合很有必要。在分析和支持下，体系结构抓住领域知识和实际的一致，促进设计的评估和构件的实施，减少仿真和构造原型。在综合的支持下，体系结构提供了建立系列产品的基础，以可预测的方式利用领域知识构造和维护模块、子系统和系统。

1. 提供软件体系结构的形式化基础

对体系结构设计的推理的形式化表示使得体系结构级的设计更好地被理解、被实现。它的目的是对体系结构设计人员在实践过程中总结出来的一些设计的经验和方法加以总结、概括，从而形成一个形式化的描述，形成一定的理论基础(以代替当前的不精确的研究)。已提出一些形式化机制，如过程代数、偏序集合、化学抽象机等。希望对系统的非功能特性如性能、可维护性等给出形式特征，同时给出软件体系结构的理论。

1. 建立评价软件体系结构的方法

通过分析来预见软件的质量，通过分析来创建、选择、评估与比较不同的体系结构。例如，Kazman等人在2000年提出的ATAM（Architectural Tradeoff Analysis Method）方法。ATAM方法不但能够揭示体系结构如何满足特定的质量需求（例如性能和可修改性），而且还提供了分析这些质量需求之间交互作用的方法。使用ATAM方法评价一个软件体系结构的目的是理解体系结构设计满足系统质量需求的结果。采用ATAM方法的步骤如图3所示，限于篇幅，在此不再详细介绍具体步骤，有兴趣的读者可查阅有关ATAM方法的文献。

1. 发展方向
2. 各种ADLs之间的信息互换

现有的ADLs大多是与领域相关的，所以不利于对不同领域体系结构的说明。但这些针对不同领域的ADLs在某些方面又大同小异，造成资源的冗余。其实，大多数ADLs具有一系列的共同概念。如何用一种公共形式把各种语言综合起来，使得能够交换各种体系结构描述信息，将是今后软件体系结构研究和实践的重点之一。

1. 设计工具和环境

软件体系结构设计既然作为软件工程的一部分，它的计算机辅助实现手段是相当重要的。我们应当开发出一些软件工具来实现体系结构的描述和分析，开发阶段转换工具，以实现阶段成果的自动转换，例如，把需求规格说明自动转换为构件等。目前关于这方面的研究成果很少，特别是可以应用到实际项目开发中的工具和环境就更少。

1. 体系结构再工程

当今软件系统的规模变得越来越大，结构也越来越复杂，同时从头开始构建的大系统数量在急剧地减少，因而很多遗留系统正在被逐步地利用。从遗留系统软件代码和系统中抽取结构信息，经过描述、统一、抽象、一般化与实例化等处理，可总结出系统的体系结构。

在这种情况下，软件工程变得越来越重要，因为它提供了一条把遗留系统转换为可进化系统的现实可行的途径，是一种可以改进人们对软件的理解和改进软件本身的活动。这类研究的目的是为一些特定的应用领域的软件系统提供一些体系结构框架，如控制系统、移动机器人和用户接口界面等。通过这些框架可以很方便地构造一个新的软件系统。

### 8.3 张杏

#### 定义

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组组合连接起来。这一定义注重区分处理构件、数据构件和连接构件，这一方法在其他的定义和方法中基本上得到保持。

通俗理解，就是把一个系统更加有调理的结构化起来，通过运用5个面向对象设计原则、分包原则及23种设计模式，让系统的结构清晰可见。类比建造房屋，就是为软件提供一个好的建筑结构，而不是想到一砖就添一砖，未来想要移动这块儿砖头，就要牵扯到整个房屋。好的设计模式可以有效的将每一块砖合理分配到应该的位置，高内聚，低耦合，让每一块砖之间的影响达到最小，但整个房屋却变得更加坚固。

#### 产生

软件体系结构的产生与软件行业的发展有关，随着软件变得越来越大、越来越复杂，一个良好的结构设计，往往成为了这个软件系统成功的重要因素之一。体系结构设计在数据结构与程序算法中变得越来越重要。软件体系结构很好的把软件的各个元素组合起来，函数、结构、类、层、子系统等等。

体系结构会受到系统相关联系人（用户、客户、项目经理、维护者、开发者）、开发部门、技术环境、架构师经验等影响。

#### 作用

良好的设计结构，可以提高软件的效率，减少冗余的软件结构设计，方便进行功能扩展，易于修改，可维护，可复用，灵活性好。

体系结构可以很好的把整个系统架构抽象出来，以一个整体的形式让人们直观的了解，隐藏了具体的细节。这样，体系结构可以作为相关联系人之间沟通的工具。

#### 应用场景

首先需要承认的是，在软件设计的不同阶段需要不同的设计方法。面对不同粒度，不同复杂度的系统，你需要使用不同的思考框架。

体系结构设计与算法设计就需要不同的思考框架。

在算法设计中，动态规划、贪心算法等这些算法思想是非常重要的思考方式思考框架，他们在一定程度上能够保证快速得到计算的结果。它们面临的场景更多类似于“快速在某个字符串中快速寻找某个子串”，这样一些小问题。

算法虽然能够攻克粒度较小的问题，但是在构建系统的时候，往往面临的问题是：软件系统规模大且复杂性高。这时对系统的全局结构设计和规划更加重要。比如说，给你一个需求，设计一个图书管理系统，这时候动态规划这些思考框架就显得有些力不从心了。这时候，体系结构设计就应运而生了，体系结构设计是一种构建复杂系统时的思考框架，它服务的对象是粗粒度的复杂系统。

#### 组成

软件体系结构组成部分：

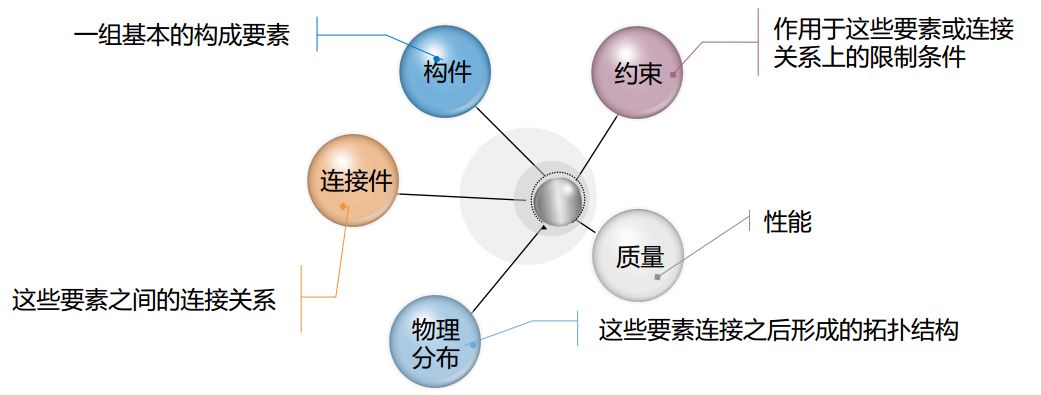
1、构件（Component）：一组代码，或程序块或一个独立的程序。（如SQL服务器）

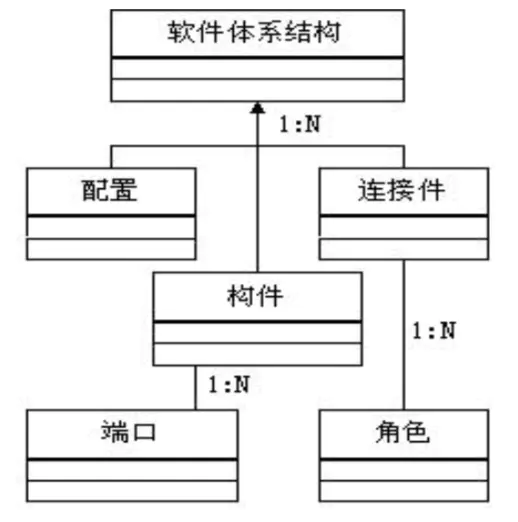
2、连接件（Connector）：关系的抽象，用以表示构件之间的相互作用。（如过程调用、管道等）

3、配置（Configuration）：用于对构件与连接件的语义说明。

4、端口：表示构件与外部环境的交互点。

5、角色：连接件的接口。



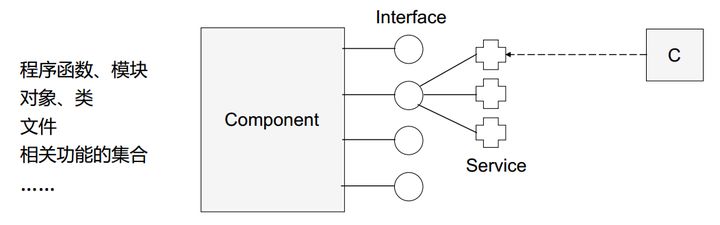


软件体系结构核心模型

**构件**

构件是具有某种功能的可复用的软件结构单元，表示系统中主要的计算元素和数据存储。

构件是一个抽象的概念，在程序中可以指程序函数、模块、对象、类等。

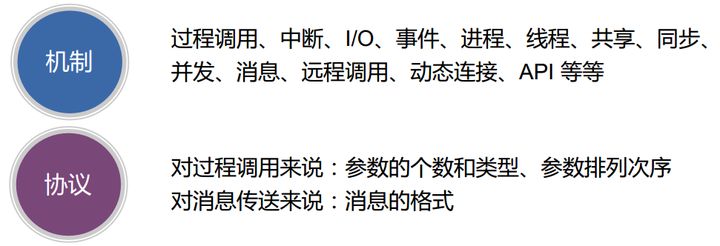


**连接件**

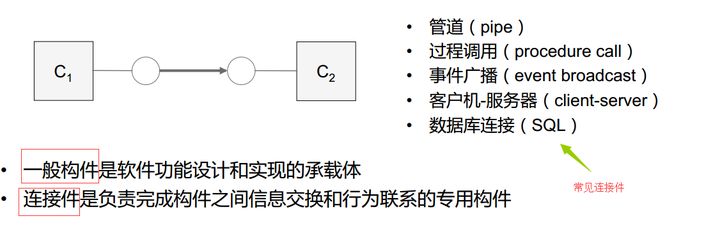
连接是构件间建立和维护行为关联与信息传递的途径。连接包含下面两种要素：

其中，机制指的实际中的消息传递方式。

而协议则决定了消息的语义理解。



连接件表示构件之间的交互并实现构件之间的连接。



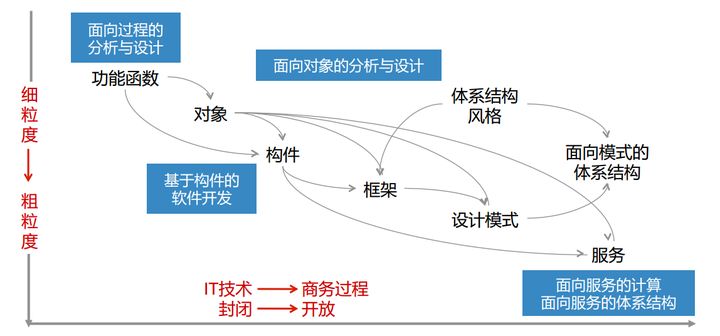
#### 软件体系结构目标

所有的设计原则等理论基本上都可以映射到下面一个或几个目标上。



#### 体系结构的发展

现在软件的复杂性及多变性，导致了软件粒度越来越粗，越来越开放。



#### 软件设计的原则

设计原则是系统分解和模块设计的基本标准，应用这些原则可以使代码更加灵活、易于维护和扩展。这里提到的设计原则包括以下几点：



需要指出的是，这些原则有一些交叉的部分，并非完全独立没有交集的，理解这一点，将能更清楚细微的区别。比如：

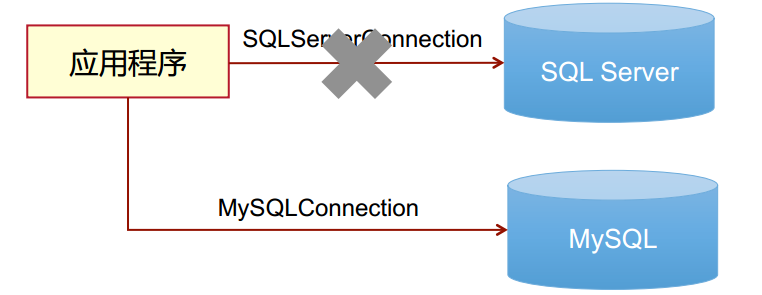
1. 通过封装来实现模块化
2. 通过将抽象程度相同的模块放在同一层次，形成层次化的结构。
3. 另外，层次化可以理解为模块化的特例

接下来分别介绍上述原则：

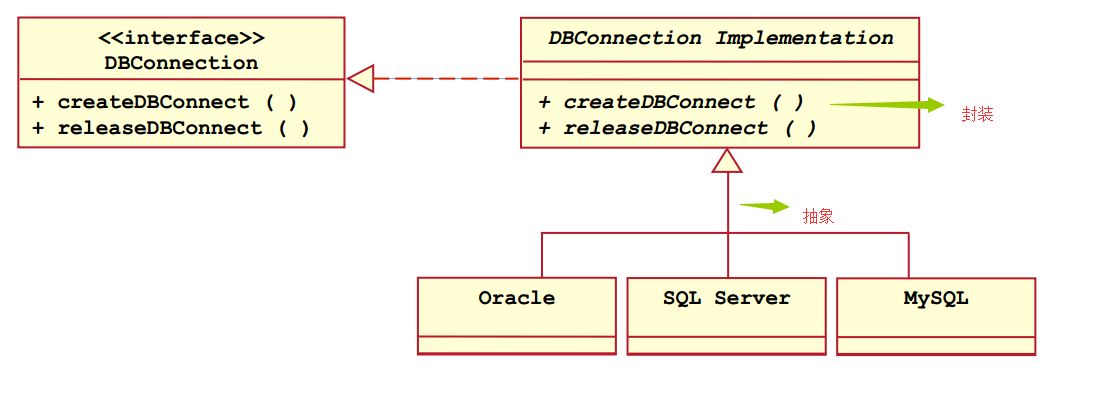
**抽象**

抽象是关注事物中与问题相关部分而忽略其他无关部分的一种思考方法。

待抽像的例子：



抽象后：

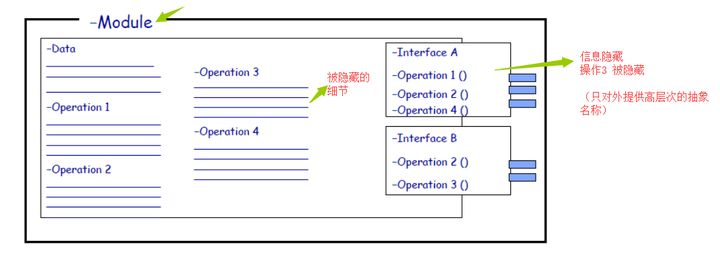


**封装**

封装和信息隐藏是指每个软件单元对其他所有单元都隐藏自己的设计决策，各个单元的特性通过其外部可见的接口来描述。

【要求】：应将单元接口设计得尽可能简单，并将单元对于环境的假设和要求降至最低。

举例：



**模块化**

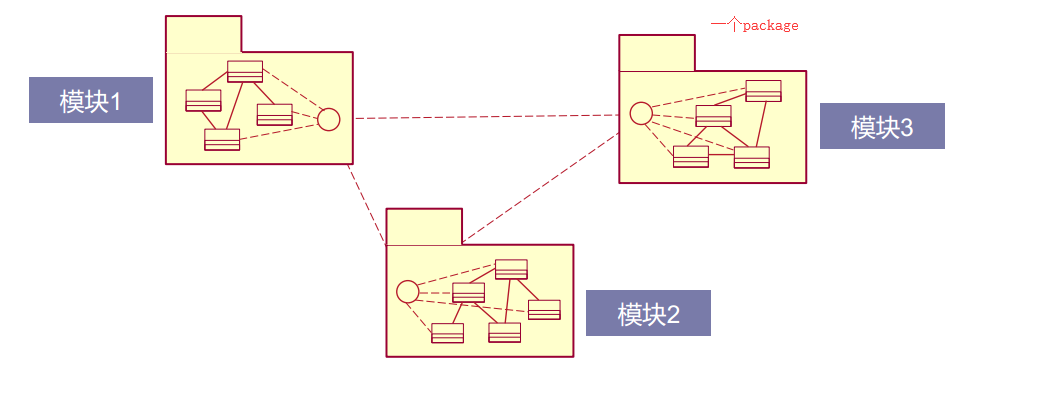
模块化是在逻辑和物理上将整个系统分解成多个更小的部分，其实质是“分而治之” ，即将一个复杂问题分解成若干个简单问题，然后逐个解决。

（一般是先进行逻辑模块化，随后进行物理分配）

举例：

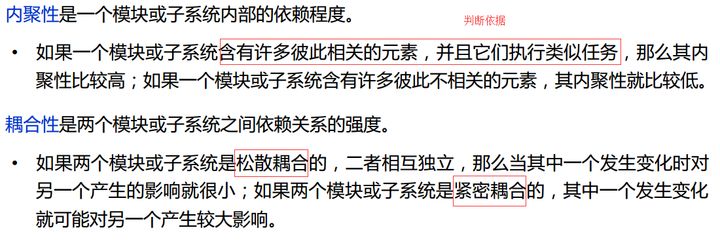
下面的模块化，主要是指逻辑上的模块化，并为涉及到如何将这些模块如何部署到物理机上的过程。

（在物理模块化的时候，可能将模块1和模块3放在同一台机器上，而模块2放在另一台机器上）



**系统分解**

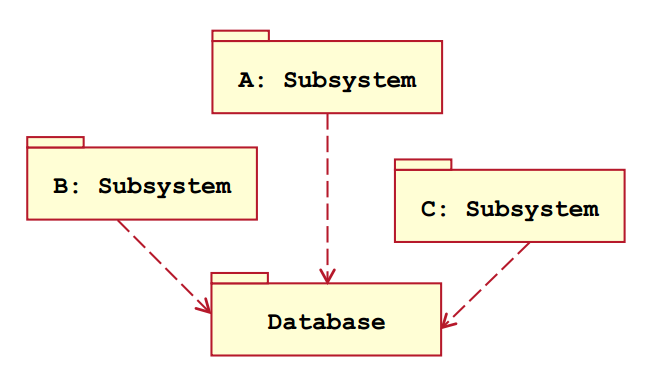
模块化，不可避免的一个问题是，对于系统如何进行分解。在大量实践中，人们总结出了两个衡量的原则：高内聚、 低耦合。



举个简单的例子：

在一个软件系统中，有三个子系统A、 B、 C都要访问一个关系数据库。

设计方案1：（高耦合方案）



设计方案2：（增加隔离层）

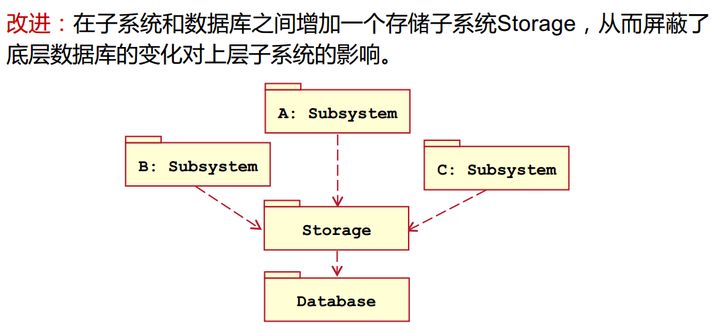
说说这种方案的好处。

设计1中，如果数据库结构发生变化，那么你需要分别在A，B，C中分别进行更改，这样分散的修改，特别容易造成错误。而在设计2中，只需要修改Storage层就可以了。

另外，在A，B，C系统中可能有一些共同的读数据库操作，比如三个系统中都有ReadAnimal（）这个操作，设计1中需要分别在A,B，C中进行修改，而设计2中将该共用方法提到Storage模块中，从而修改时，只需要进行一次修改。

PS:

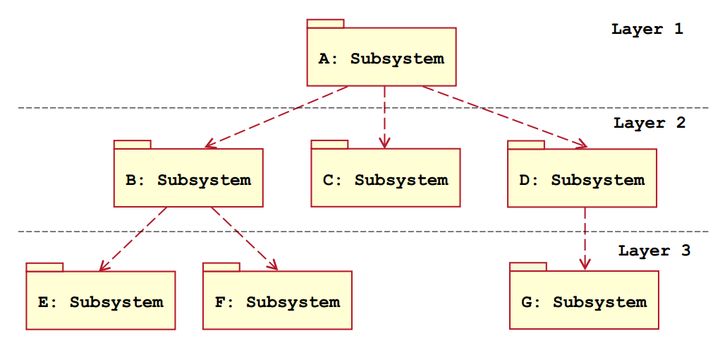
这里加入Storage，是因为假设3个系统中有一些共同的部分。但是如果子系统没有任何共享的数据库数据（每个表被一个系统所独占），那么去掉Storage层也没有关系，因为数据库某个表的变化只会影响到一个子系统。



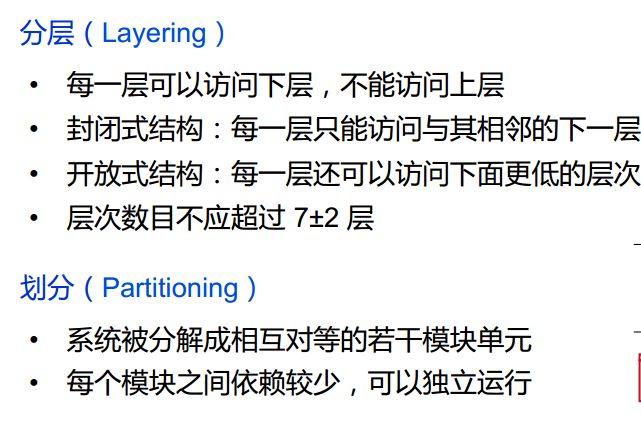
**层次化**

先给出层次化的表现形式：

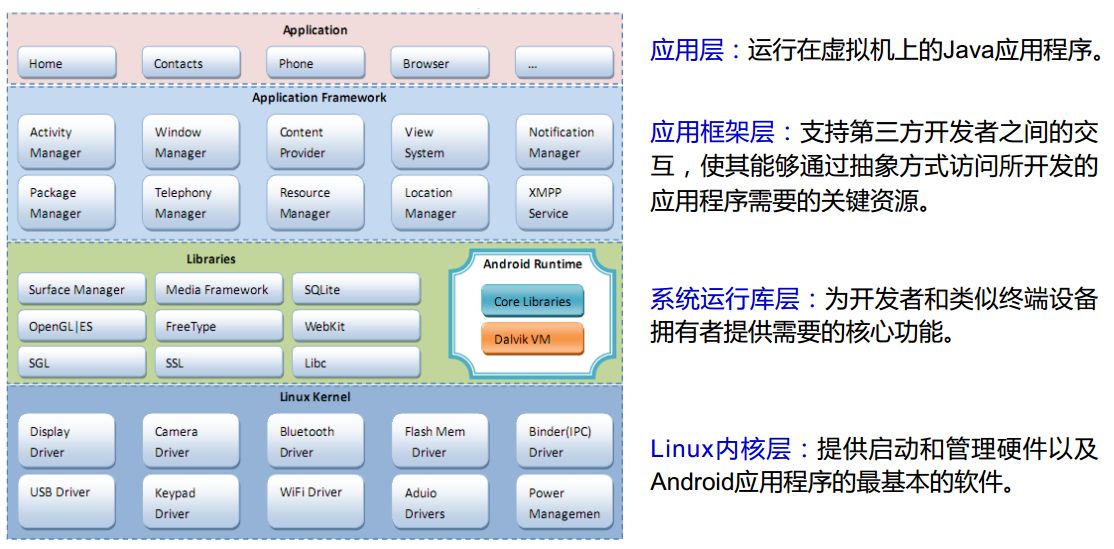
层次化是模块化的一种特例。在模块化的基础上，加入了一些限制。在层次化结构中，layer可以作为一种特殊的模块（包含特定的子系统集合），在这类模块（layer）的联系中，规定了layer模块只能被设计成类似于双向链表的顺序结构，而不能出现树状结构或者图状结构。



层次化的一些说明：



举例：Android操作系统



**复用**

复用（Reuse）是利用某些已开发的、 对建立新系统有用的软件元素来生成新的软件系统，其好处在于提高生产效率，提高软件质量。

源代码复用：对构件库中的源代码构件进行复用

• 软件体系结构复用：对已有的软件体系结构进行复用

• 框架复用：对特定领域中存在的一个公共体系结构及其构件进行复用

• 设计模式：通过为对象协作提供思想和范例来强调方法的复用

#### 10种常见的软件体系架构模式

架构模式是一个通用的、可重用的解决方案，用于在给定上下文中的软件体系结构中经常出现的问题。架构模式与软件设计模式类似，但具有更广泛的范围。

**一. 分层模式**

这种模式也称为多层体系架构模式。它可以用来构造可以分解为子任务组的程序，每个子任务都处于一个特定的抽象级别。每个层都为下一个提供更高层次服务。

一般信息系统中最常见的是如下所列的4层。

• 表示层(也称为UI层)

• 应用层(也称为服务层)

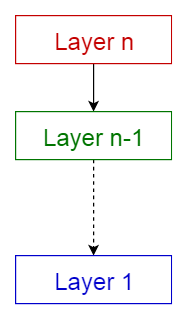
• 业务逻辑层(也称为领域层)

• 数据访问层(也称为持久化层)

使用场景：

• 一般的桌面应用程序

• 电子商务Web应用程序

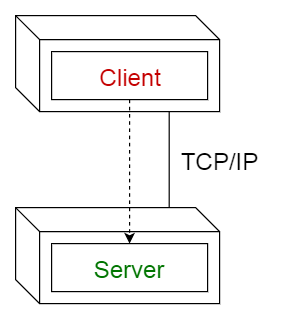


**二. 客户端-服务器模式**

这种模式由两部分组成：一个服务器和多个客户端。服务器组件将为多个客户端组件提供服务。客户端从服务器请求服务，服务器为这些客户端提供相关服务。此外，服务器持续侦听客户机请求。

使用场景：

• 电子邮件，文件共享和银行等在线应用程序



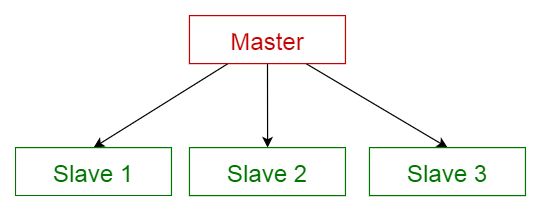
**三. 主从设备模式**

这种模式由两方组成;主设备和从设备。主设备组件在相同的从设备组件中分配工作，并计算最终结果，这些结果是由从设备返回的结果。

使用场景：

• 在数据库复制中，主数据库被认为是权威的来源，并且要与之同步

• 在计算机系统中与总线连接的外围设备(主和从驱动器)



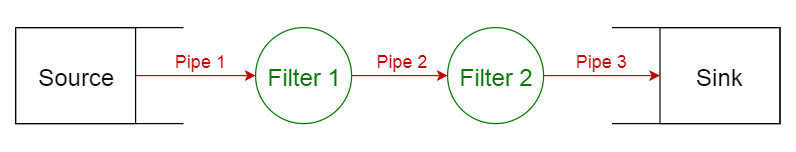
**四. 管道-过滤器模式**

此模式可用于构造生成和处理数据流的系统。每个处理步骤都封装在一个过滤器组件内。要处理的数据是通过管道传递的。这些管道可以用于缓冲或用于同步。

使用场景：

• 编译器。连续的过滤器执行词法分析、解析、语义分析和代码生成

• 生物信息学的工作流



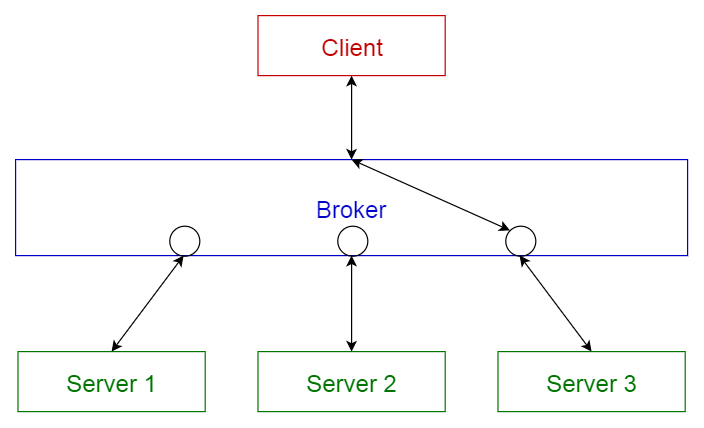
**五. 代理模式**

此模式用于构造具有解耦组件的分布式系统。这些组件可以通过远程服务调用彼此交互。代理组件负责组件之间的通信协调。

服务器将其功能(服务和特征)发布给代理。客户端从代理请求服务，然后代理将客户端重定向到其注册中心的适当服务。

使用场景：

• 消息代理软件，如Apache ActiveMQ，Apache Kafka，RabbitMQ和JBoss Messaging



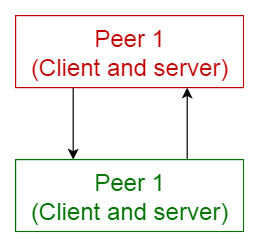
**六. 点对点模式**

在这种模式中，单个组件被称为对等点。对等点可以作为客户端，从其他对等点请求服务，作为服务器，为其他对等点提供服务。对等点可以充当客户端或服务器或两者的角色，并且可以随时间动态地更改其角色。

使用场景：

• 像Gnutella和G2这样的文件共享网络

• 多媒体协议，如P2PTV和PDTP• 像Spotify这样的专有多媒体应用程序



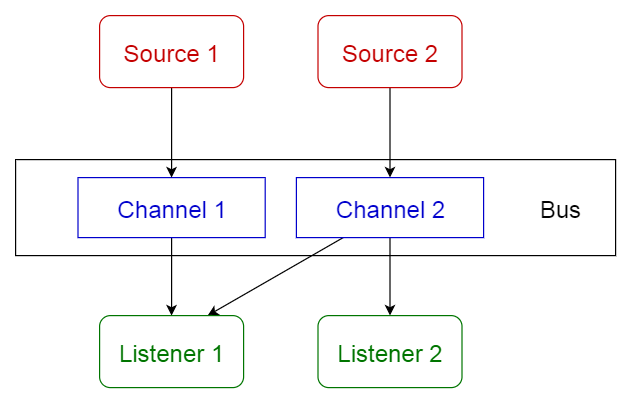
**七. 事件总线模式**

这种模式主要是处理事件，包括4个主要组件：事件源、事件监听器、通道和事件总线。消息源将消息发布到事件总线上的特定通道上。侦听器订阅特定的通道。侦听器会被通知消息，这些消息被发布到它们之前订阅的一个通道上。

使用场景：

• 安卓开发

• 通知服务



**八. 模型-视图-控制器模式**

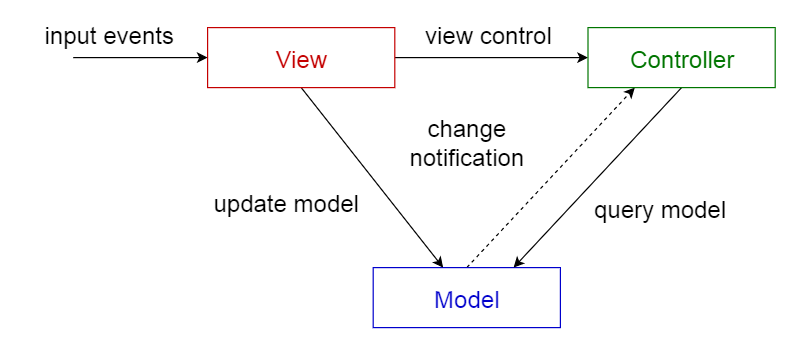
这种模式，也称为MVC模式，把一个交互式应用程序划分为3个部分，• 模型：包含核心功能和数据• 视图：将信息显示给用户(可以定义多个视图)• 控制器：处理用户输入的信息

这样做是为了将信息的内部表示与信息的呈现方式分离开来，并接受用户的请求。它分离了组件，并允许有效的代码重用。

使用场景：

• 在主要编程语言中互联网应用程序的体系架构

• 像Django和Rails这样的Web框架



**九. 黑板模式**

这种模式对于没有确定解决方案策略的问题是有用的。黑板模式由3个主要组成部分组成。

• 黑板——包含来自解决方案空间的对象的结构化全局内存

• 知识源——专门的模块和它们自己的表示

• 控制组件——选择、配置和执行模块所有的组件都可以访问黑板。组件可以生成添加到黑板上的新数据对象。组件在黑板上查找特定类型的数据，并通过与现有知识源的模式匹配来查找这些数据。

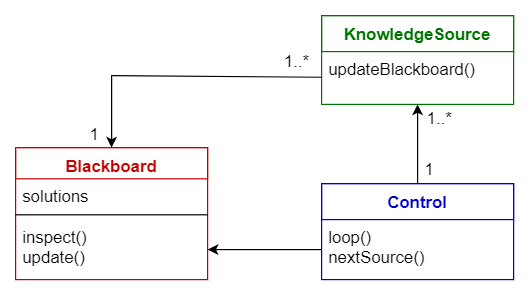
使用场景：

• 语音识别

• 车辆识别和跟踪

• 蛋白质结构识别

• 声纳信号的解释



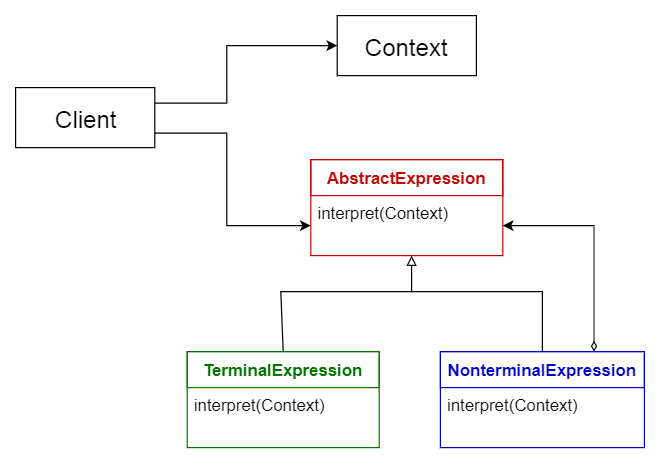
**十. 解释器模式**

这个模式用于设计一个解释用专用语言编写的程序的组件。它主要指定如何评估程序的行数，即以特定的语言编写的句子或表达式。其基本思想是为每种语言的符号都有一个分类。

使用场景：

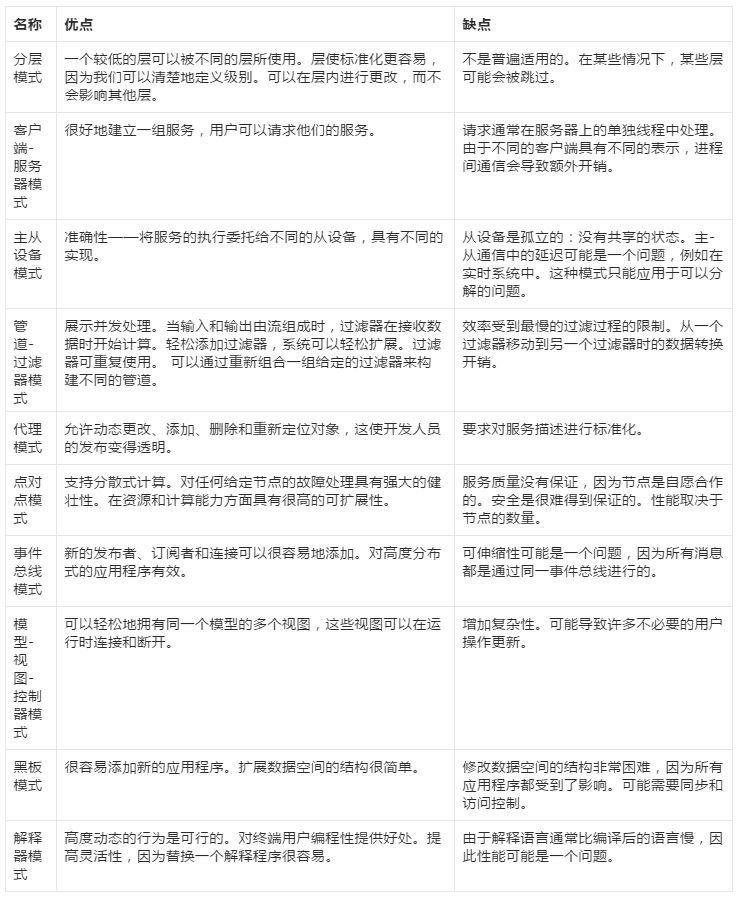
• 数据库查询语言，比如SQL

• 用于描述通信协议的语言



#### 体系架构模式的比较

下面给出的表格总结了每种体系架构模式的优缺点。



### 8.4 赵子涵

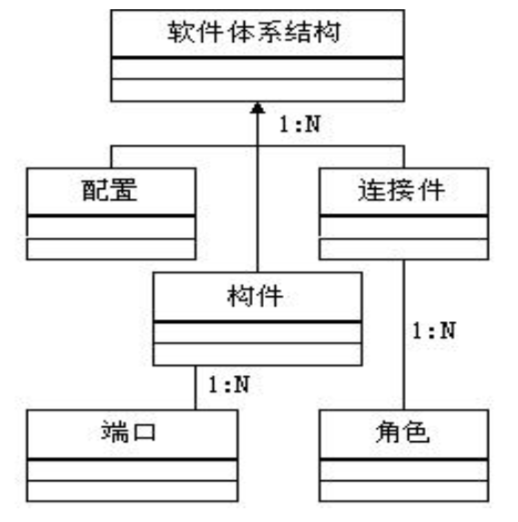
**软件体系结构**

1. 概念

是在处理算法与数据结构之上的，关于整体系统结构设计和描述方面的问题。是软件系统的一个结构、行为和熟悉的高级抽象。（除了概念的组织结构与系统的拓扑结构外，还有系统需求与元素的对应关系。）

重设计重用。

1. 软件体系结构组成部分
2. **构件**（Component）：一组代码，或程序块或一个独立的程序。（如SQL服务器）
3. **连接件**（Connector）：关系的抽象，用以表示构件之间的相互作用。（如过程调用、管道等）
4. **配置**（Configuration）：用于对构件与连接件的语义说明。
5. **端口**：表示构件与外部环境的交互点。
6. **角色**：连接件的接口。

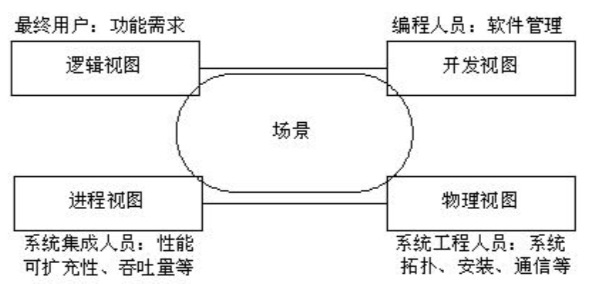


1. 软件体系结构模型
2. **结构模型**：最直观最普遍的建模方法，通过体系结构的构件、连接件和其他概念来刻画结构；力图通过结构，来反映系统的重要语义内容。

包括：系统的配置，约束，隐含的假设条件，风格，性质。

1. **框架模型**：与结构模型类似，但不侧重结构的细节，而侧重整体的结构。主要以一些特殊的问题作为目标，建立只针对和适应这类问题的结构。
2. **动态模型**：对结构或框架模型的补充，研究系统的“大粒度”的行为及其性质。如系统总体结构的配置或演化、通信通道或计算过程的建立或拆除等方面的行为。
3. **过程模型**：研究构造系统的步骤与过程，因而其机构是遵循某些脚本的结果。
4. **功能模型**：体系机构的一组功能构件，按照层次组成，下层向上层提供服务，可以看作为一种特殊的框架模型。
5. “4+1”视图模型

5个角度：逻辑视图，开发视图，进程视图，物理视图，场景视图。



### 8.5 徐容

**定义**

“软件体系结构包括有关软件系统组织的一系列重要决策，包括对构成系统的结构要素及其接口的选择；这些要素之间协作规定的行为；将这些结构和行为要素组成更大的子系统；以及指导该组织的建筑风格。软件体系结构还涉及功能，可用性，弹性，性能，重用性，可理解性，经济和技术约束，折衷和美学方面的考虑。”

**质量属性：**

开发期质量:可扩展性，可复用性，可维护性等；

运行期质量：正确性，健壮性，性能，可靠性，容错性，易用性，安全性，可移植性，兼容性。

**设计原则：**

面向接口编程（Program to interfaces, not to implementations）

多用组合，少用继承（Favor composition over inheritance）

Principle of Least Knowledge(Law of Demeter)

单一职责原则（Single Responsibility Principle）：就一个类而言，应该仅有一个引起它变化的原因。

开闭原则（Open-Closed Principle）：软件实体对扩展是开放的，但对修改是关闭的，即在不修改一个软件实体的基础上去扩展其功能。抽象化是开闭原则的关键

里氏代换原则（Liskov Substitution Principle）：在软件系统中，一个可以接受基类对象的地方必然可以接受一个子类对象。里氏代换原则是实现开闭原则的重要方法之一。

依赖倒置原则（Dependency Inversion Principle）：要针对抽象层编程，而不要针对具体类编程。实现开闭原则的关键是抽象化，并且从抽象导出具体化实现，如果说开闭原则是面向对象设计的目标的话，那么依赖倒置原则就是面向对象设计的主要手段。依赖注入：构造注入，设置注入，接口注入

接口隔离原则（Interface Segregation Principle）：使用多个专门的接口来取代一个统一的接口。

分离关注点（Principle of Separation of Concerns）:"Organize software into separate components(pieces) that are as independent as possible."

软件体系结构包括构成系统的设计元素的描述，设计元素之间的交互，设计元素的组合模式以及这些模式中的约束。

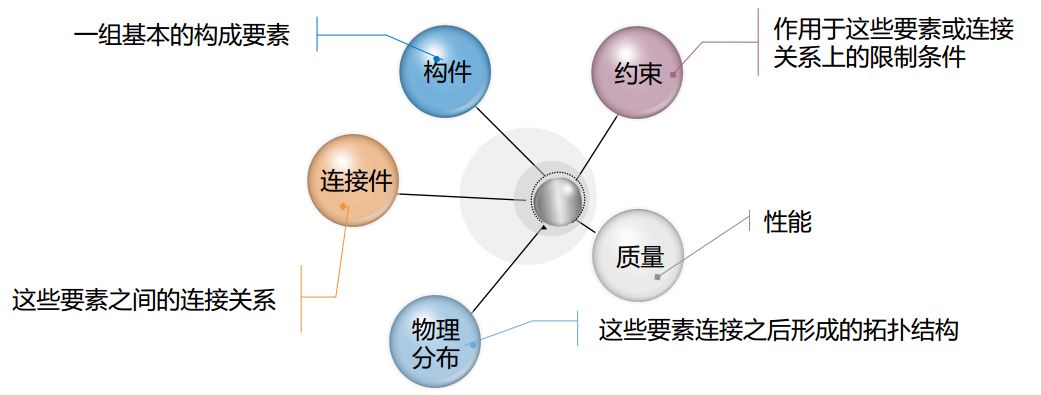
**构件：**一组基本的构成要素。

**连接件：**这些要素之间的连接关系。

**约束：**作用于这些要素或者连接关系上的限制条件。

**质量：**性能。

**物理分布：**这些要素连接之后的拓扑结构。



软件体系结构=构件+连接件+约束

**框架和体系结构的关系**

体系结构的呈现方式是一个设计规约，而框架则是“半成品”的软件。

体系结构的目的是指导软件系统的开发，而框架的目的是设计复用。

**软件体系结构风格**

1. 数据流风格：批处理和管道/过滤器。
2. 调用/返回风格：主程序/子程序、层次结构和C/S。
3. 面向对象风格。
4. 独立部件风格：进程通信和事件驱动。
5. 虚拟机风格：解释器和基于规则的系统。
6. 数据共享风格：数据库系统和黑板系统。

软件体系结构应用现状  
1、**ADL（Architecture Description Language）软件体系结构描述语言**：描述体系结构的概念框架。（C2，Wright，Aesop，Unicon等）  
优秀的ADL的特性：组装性，抽象性，重用性，可配置性，异构性，可分析性等。  
2、**软件体系结构描述构造与表示**：用一定描述方法，对体系结构进行描述。  
eg. “4+1”模型：由逻辑视图，开发视图，过程视图，和物理视图组成。用应用场景将这4个有机结合起来。比较细致地描述了需求和体系结构之间的关系。  
3、**体系结构的设计、分析与验证**：如何将系统分解成相应的组成成分（构件，连接件）  
eg. 风格设计。结构分析，功能分析，非功能分析。  
4、**体系结构的发现、演化与重用**：对已有的系统，提取体系结构等。  
5、**基于体系结构等软件开发**；  
6、**特定领域的体积结构**；  
7、**软件体系结构的支持工具**：软件体系结构的支持工具。  
8、**软件产品线体系结构**：提高重用性。  
9、**建立软件体系结构的方法**；

软件体系结构的研究与应用之中的不足之处：  
1、缺乏统一的软件体系结构概念，导致软件体系结构的研究范围模糊。  
2、ADL繁多，缺乏统一的ADL标准。  
3、软件体系结构研究缺乏统一的理论模型支持。  
4、市面上的多种标准规范或建议标准不易于操作。  
5、软件体系结构性质的研究尚不充分，不能明确给出一个良好的体系极高的属性或评断标准，也没有给出良好体系就高的设计指导原则，因此对于软件开发实践缺乏有力的促进作用。  
6、缺乏有效的支持环境，以及软件体系结构理论研究与环境支持不同步，缺乏有效的体系结构分析、设计、方针和验证工具支持，导致体系结构应用困难。  
7、缺乏有效的体系结构复用方案。  
8、体系结构发现方法研究相对欠缺。

**架构风格**

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。

### 8.6 杨纪元

目录

[一. 软件构架 2](#_Toc71665442)

[二. 软件设计 3](#_Toc71665443)

[三. 体系结构的目标 3](#_Toc71665444)

[四. 软件架构师的角色 4](#_Toc71665445)

[设计专长 4](#_Toc71665446)

[领域专长 4](#_Toc71665447)

[技术专长 4](#_Toc71665448)

[方法学专长 4](#_Toc71665449)

[软件架构师的隐藏角色 5](#_Toc71665450)

[体系结构工程师的任务 5](#_Toc71665451)

[五. 质量属性 5](#_Toc71665452)

[6.1 静态质量属性 5](#_Toc71665453)

[6.2 动态质量属性 5](#_Toc71665454)

[六. 质量方案 6](#_Toc71665455)

[通用质量属性 6](#_Toc71665456)

[七. 体系结构的设计风格 7](#_Toc71665457)

[八. 普通结构设计 8](#_Toc71665458)

[九. 体系结构类型 9](#_Toc71665459)

[十. 体系结构设计过程 9](#_Toc71665460)

[了解问题 9](#_Toc71665461)

[识别设计元素及其关系 9](#_Toc71665462)

[评估架构设计 10](#_Toc71665463)

[十一. 关键架构原则 10](#_Toc71665464)

[为改变而建，而不是为之持久 10](#_Toc71665465)

[降低风险并进行模型分析 10](#_Toc71665466)

[将模型和可视化用作沟通和协作工具 11](#_Toc71665467)

[使用增量迭代方法 11](#_Toc71665468)

[十二. 关键设计原则 11](#_Toc71665469)

[关注点分离 11](#_Toc71665470)

[单一责任原则 11](#_Toc71665471)

[最少知识原理 11](#_Toc71665472)

[最大限度地减少前期的大型设计 11](#_Toc71665473)

[不要重复功能 12](#_Toc71665474)

[重用功能时要优先考虑组合（而不是继承） 12](#_Toc71665475)

[识别组件并将它们分组在逻辑层中 12](#_Toc71665476)

[定义层之间的通信协议 12](#_Toc71665477)

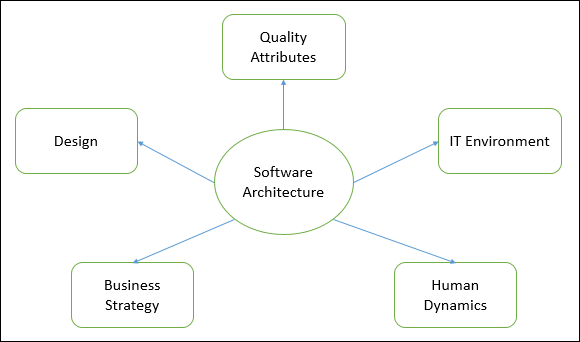
[定义图层的数据格式 12](#_Toc71665478)

[系统服务组件应该是抽象的 12](#_Toc71665479)

[设计异常和异常处理机制 12](#_Toc71665480)

[命名约定 13](#_Toc71665481)

软件系统的体系结构描述了其主要组件，它们之间的关系（结构）以及它们之间如何交互。软件体系结构和设计包括几个促成因素，例如业务战略，质量属性，人员动态，设计和IT环境。



我们可以将软件体系结构和设计分为两个不同的阶段：软件体系结构和软件设计。在**体系结构中**，非功能性决策由功能性需求强制转换和分离。在设计中，功能要求得以实现。

#### 一. 软件构架

体系结构是**系统**的**蓝图**。它提供了一种抽象方法来管理系统复杂性，并在组件之间建立通信和协调机制。

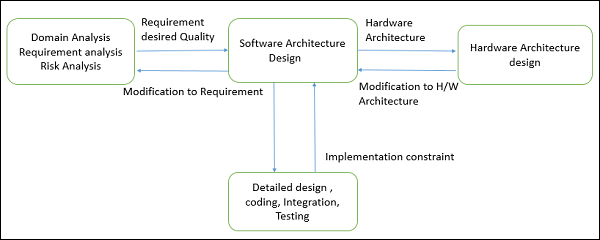
* 它定义了一种**结构化解决方案**，可以满足所有技术和运营要求，同时优化性能和安全性等常见质量属性。
* 此外，它涉及与软件开发有关的一组重要的组织决策，并且这些决策中的每一个都会对最终产品的质量，可维护性，性能和整体成功产生重大影响。这些决定包括-
  + 选择组成系统的结构元素及其界面。
  + 在这些元素之间的协作中指定的行为。
  + 将这些结构和行为元素组成大型子系统。
  + 体系结构决策与业务目标保持一致。
  + 体系结构风格指导组织。

#### 二. 软件设计

软件设计提供了一个**设计计划**，该**计划**描述了系统的元素，它们如何适合并一起工作以满足系统的需求。制定设计计划的目标如下-

* 与系统需求进行协商，并与客户，市场营销和管理人员建立期望。
* 在开发过程中充当蓝图。
* 指导实施任务，包括详细的设计，编码，集成和测试。

它在详细的设计，编码，集成和测试之前，以及在域分析，需求分析和风险分析之后。



#### 三. 体系结构的目标

该体系结构的主要目标是确定影响应用程序结构的需求。布局合理的体系结构可减少与构建技术解决方案相关的业务风险，并在业务和技术要求之间架起一座桥梁。

其他一些目标如下-

* 公开系统的结构，但隐藏其实现细节。
* 实现所有用例和方案。
* 尝试解决各种利益相关者的要求。
* 处理功能和质量要求。
* 降低所有权目标并改善组织的市场地位。
* 改善系统提供的质量和功能。
* 提高外部对组织或系统的信心。

但是软件体系结构仍然是软件工程中的新兴学科。它具有以下限制-

* 缺乏用于表示架构的工具和标准化方法。
* 缺乏分析方法来预测体系结构是否会导致实现符合要求的分析方法。
* 缺乏对体系结构设计对软件开发的重要性的认识。
* 缺乏对软件架构师角色的了解，以及利益相关者之间的沟通不畅。
* 缺乏对设计过程的了解，设计经验和设计评估。

#### 四. 软件架构师的角色

软件架构师提供了一个解决方案，技术团队可以为整个应用程序创建和设计该解决方案。软件架构师应在以下领域具有专业知识-

##### 设计专长

* 软件设计专家，包括多种方法和方法，例如面向对象的设计，事件驱动的设计等。
* 领导开发团队并协调开发工作，以确保设计的完整性。
* 应该能够审查设计建议并在它们之间进行权衡。

##### 领域专长

* 正在开发的系统专家以及软件开发计划。
* 协助需求调查过程，确保完整性和一致性。
* 协调要开发的系统的域模型的定义。

##### 技术专长

* 有助于系统实施的可用技术专家。
* 协调编程语言，框架，平台，数据库等的选择。

##### 方法学专长

* SDLC（软件开发生命周期）中可能采用的软件开发方法专家。
* 选择对整个团队有帮助的合适的开发方法。

##### 软件架构师的隐藏角色

* 促进团队成员之间的技术工作，并增强团队中的信任关系。
* 共享知识并拥有丰富经验的信息专家。
* 保护团队成员免受外力的干扰，这些外力会分散他们的注意力，并给项目带来较少的价值。

##### 体系结构工程师的任务

* 清晰，完整，一致且可实现的功能目标集
* 系统的功能描述，至少有两层分解
* 系统的概念
* 系统形式的设计，至少具有两层分解
* 时间，操作员属性以及实施和操作计划的概念
* 确保遵循功能分解并控制接口形式的文档或过程

#### 五. 质量属性

质量是衡量卓越水平或无缺陷或无缺陷状态的一种度量。质量属性是与系统功能分开的系统属性。

实施质量属性使区分好系统和坏系统变得更加容易。属性是影响运行时行为，系统设计和用户体验的总体因素。

他们可以分类为-

##### 6.1 静态质量属性

反映与体系结构，设计和源代码直接相关的系统和组织的结构。它们对于最终用户是不可见的，但是会影响开发和维护成本，例如：模块化，可测试性，可维护性等。

##### 6.2 动态质量属性

反映系统在执行过程中的行为。它们与系统的体系结构，设计，源代码，配置，部署参数，环境和平台直接相关。它们对于最终用户是可见的，并且在运行时存在，例如吞吐量，健壮性，可伸缩性等。

#### 六. 质量方案

质量方案指定了如何防止故障成为故障。根据其属性规范，它们可以分为六个部分-

* **来源**-产生刺激的内部或外部实体，例如人员，硬件，软件或物理基础结构。
* **刺激**-到达系统时需要考虑的条件。
* **环境**-刺激在一定条件下发生。
* **工件**-整个系统或其中的一部分，例如处理器，通信通道，持久性存储，进程等。
* **响应**-刺激到达后进行的活动，例如检测故障，从故障中恢复，禁用事件源等。
* **响应措施**-应该测量已发生的响应，以便可以测试需求。

##### 通用质量属性

下表列出了软件体系结构必须具有的常见质量属性：

|  |  |  |
| --- | --- | --- |
| 类别 | 质量属性 | 描述 |
| 设计品质 | 概念完整性 | 定义整体设计的一致性和连贯性。这包括组件或模块的设计方式。 |
| 可维护性 | 系统轻松进行更改的能力。 |
| 可重用性 | 定义组件和子系统的功能，使其适合在其他应用程序中使用。 |
| 运行时质量 | 互通性 | 一个系统或不同系统通过与外部各方编写和运行的其他外部系统进行通信和交换信息来成功运行的能力。 |
| 可管理性 | 定义系统管理员管理应用程序的难易程度。 |
| 可靠性 | 系统随时间推移保持运行的能力。 |
| 可扩展性 | 系统在不影响系统性能的情况下处理负载增加的能力或易于扩展的能力。 |
| 安全 | 系统具有防止超出设计用途的恶意或意外行为的功能。 |
| 表现 | 指示系统在给定时间间隔内执行任何操作的响应能力。 |
| 可用性 | 定义系统正常运行和工作的时间比例。可以将其衡量为预定时间段内系统总停机时间的百分比。 |
| 系统品质 | 可支持性 | 系统提供信息的能力，有助于在系统无法正常工作时识别和解决问题。 |
| 可测性 | 衡量为系统及其组件创建测试标准的难易程度。 |
| 用户素质 | 易用性 | 通过直观定义应用程序满足用户和消费者要求的程度。 |
| 体系结构质量 | 正确性 | 满足系统所有要求的责任。 |
| 非运行时质量 | 可移植性 | 系统在不同计算环境下运行的能力。 |
| 整体性 | 使系统的单独开发的组件能够正确一起工作的能力。 |
| 可修改性 | 每个软件系统都可以轻松地适应其软件的更改。 |
| 业务质量属性 | 费用和时间表 | 与上市时间，预期项目寿命和遗留资源利用有关的系统成本。 |
| 适销性 | 关于市场竞争的系统使用。 |

#### 七. 体系结构的设计风格

软件体系结构的设计风格，也称为**架构模式**，是一组其形状的应用原理。它根据结构组织的模式定义了一个系统族的抽象框架。

体系结构的设计风格是指：

* 为组件和连接器的词典提供如何组合的规则。
* 通过为经常发生的问题提供解决方案，改进分区并允许设计的重用。
* 描述一种特定的方式来配置组件（具有良好定义的接口，可重用和可替换的模块）和连接器（模块之间的通信链接）的集合。

为基于计算机的系统而构建的软件表现出许多体系结构样式之一。每种样式都描述一个系统类别，其中包括：

* 一组由系统执行所需功能的组件类型。
* 一组连接器（子例程调用，远程过程调用，数据流和套接字），使不同组件之间能够进行通信，协调和协作。
* 语义约束，定义了如何集成组件以形成系统。
* 组件的拓扑布局，指示它们的运行时相互关系。

#### 八. 普通结构设计

下表列出了可以按其主要关注领域进行组织的体系风格-

|  |  |  |
| --- | --- | --- |
| 类别 | 体系设计 | 描述 |
| 沟通 | 消息总线 | 规定使用可以使用一个或多个通信通道接收和发送消息的软件系统。 |
| 面向服务的架构（SOA） | 定义使用合同和消息将功能公开和使用为服务的应用程序。 |
| 部署方式 | 客户端服务器 | 将系统分为两个应用程序，客户端在其中向服务器发出请求。 |
| 3层或N层 | 将功能分成单独的部分，每个部分是位于物理上分开的计算机上的一层。 |
| 领域 | 域驱动设计 | 专注于对业务域进行建模，并基于业务域内的实体定义业务对象。 |
| 结构 | 基于组件 | 将应用程序设计分解为可重用的功能或逻辑组件，以暴露定义明确的通信接口。 |
| 分层的 | 将应用程序的关注点分为堆叠的组（层）。 |
| 面向对象 | 基于将应用程序或系统的职责划分为对象，每个对象包含与对象相关的数据和行为。 |

#### 九. 体系结构类型

从企业的角度来看，有四种类型的体系结构，这些体系结构统称为**企业体系结构**。

* **业务架构**-定义企业内业务，治理，组织和关键业务流程的策略，并专注于业务流程的分析和设计。
* **应用程序（软件）架构**-用作各个应用程序系统，它们之间的交互以及与组织业务流程的关系的蓝图。
* **信息架构**-定义逻辑和物理数据资产以及数据管理资源。
* **信息技术（IT）架构**-定义构成组织整体信息系统的硬件和软件构建块。

#### 十. 体系结构设计过程

架构设计过程着重于将系统分解为不同的组件及其交互，以满足功能和非功能需求。软件架构设计的关键输入是-

* 由分析任务产生的需求。
* 硬件架构（软件架构师又向配置硬件架构的系统架构师提供要求）。

体系结构设计过程的结果或输出是**体系结构描述**。基本架构设计过程包括以下步骤-

##### 了解问题

* 这是最关键的步骤，因为它会影响后续设计的质量。
* 如果没有清楚地了解问题，就不可能创建有效的解决方案。
* 许多软件项目和产品被认为是失败的，因为它们实际上并未解决有效的业务问题或具有可识别的投资回报率（ROI）。

##### 识别设计元素及其关系

* 在此阶段，建立基线以定义系统的边界和上下文。
* 根据功能需求将系统分解为主要组件。可以使用设计结构矩阵（DSM）对分解建模，该结构结构矩阵显示设计元素之间的依赖性，而无需指定元素的粒度。
* 在此步骤中，通过描述许多系统实例来完成对体系结构的第一次验证，并且此步骤称为基于功能的体系结构设计。

##### 评估架构设计

* 每个质量属性都有一个估计值，因此为了收集定性度量或定量数据，需要对设计进行评估。
* 它涉及评估体系结构是否符合体系结构质量属性要求。
* 如果所有估计的质量属性均符合要求的标准，则架构设计过程已完成。
* 如果不是，则进入软件体系结构设计的第三阶段：体系结构转换。如果观察到的质量属性不符合其要求，则必须创建一个新设计。

#### 十一. 关键架构原则

以下是设计架构时要考虑的关键原则-

##### 为改变而建，而不是为之持久

考虑一下应用程序可能需要随着时间的变化而变化，以解决新的要求和挑战，并建立支持它的灵活性。

##### 降低风险并进行模型分析

使用设计工具，可视化效果，建模系统（例如**UML**）来捕获需求和设计决策。影响也可以分析。不要对模型进行形式化，以免抑制其轻易迭代和修改设计的能力。

##### 将模型和可视化用作沟通和协作工具

设计，决策和设计的有效更改之间的有效沟通对于良好的体系结构至关重要。使用模型，视图和体系结构的其他可视化效果与所有涉众有效地交流和共享设计。这样可以快速传达设计变更。

识别并了解关键的工程决策和最容易出错的领域。在第一时间投资于正确的关键决策，以使设计更加灵活，并且不会因更改而被破坏。

##### 使用增量迭代方法

从基线体系结构开始，然后通过迭代测试改进候选体系结构以改进体系结构。通过多次遍历将迭代细节添加到设计中，以获取大图或正确的图景，然后将注意力集中在细节上。

#### 十二. 关键设计原则

以下是在最大程度地降低成本，维护要求并最大程度地提高体系结构的可扩展性和可用性时应考虑的设计原则-

##### 关注点分离

将系统的组件划分为特定的功能，以使组件功能之间没有重叠。这将提供高内聚力和低耦合度。这种方法避免了系统组件之间的相互依赖性，这有助于简化系统。

##### 单一责任原则

系统的每个模块都应负一个特定的责任，这有助于用户清楚地了解系统。它还应有助于组件与其他组件的集成。

##### 最少知识原理

任何组件或对象都不应该了解其他组件的内部细节。这种方法避免了相互依赖，并有助于可维护性。

##### 最大限度地减少前期的大型设计

如果应用程序的需求不清楚，则最大程度地减少前期的大型设计。如果有可能修改需求，则应避免对整个系统进行大型设计。

##### 不要重复功能

“不重复功能”指定不应重复组件的功能，因此仅应在一个组件中实现一段代码，即组件可重用。应用程序中功能的重复会使其难以实施更改，降低清晰度并引入潜在的不一致之处。

##### 重用功能时要优先考虑组合（而不是继承）

继承会在子类和父类之间建立依赖关系，因此会阻止子类的自由使用。相反，该组合提供了很大的自由度并减少了继承层次结构。

##### 识别组件并将它们分组在逻辑层中

系统中满足要求所需的标识组件和关注区域。然后将这些相关组件分组在一个逻辑层中，这将帮助用户从较高的层次上理解系统的结构。避免在同一层中混合不同类型关注点的组件。

##### 定义层之间的通信协议

了解组件之间如何通信，这需要对部署方案和生产环境有完整的了解。

##### 定义图层的数据格式

各种组件将通过数据格式相互交互。不要混用数据格式，以使应用程序易于实现，扩展和维护。尝试使层的数据格式相同，以便各个组件在相互通信时无需对数据进行编码/解码。它减少了处理开销。

##### 系统服务组件应该是抽象的

与安全性，通信或系统服务（如日志记录，概要文件和配置）相关的代码应在单独的组件中抽象出来。不能将此代码与业务逻辑混合使用，因为扩展设计和维护很容易。

##### 设计异常和异常处理机制

预先定义异常，有助于组件以优雅的方式管理错误或意外情况。整个系统中的异常管理都是相同的。

##### 命名约定

命名约定应事先定义。它们提供了一个一致的模型，可以帮助用户轻松理解系统。团队成员更容易验证其他人编写的代码，因此会增加可维护性。