**软件体系结构：**

软件架构设计：软件架构设计就是从宏观上说明一套软件系统的组成与特性，是一系列有层次地决策。

**软件架构设计的方法：**

1.五视图法：（软件系统的不同的角色会站在不同的角度上提出的问题，我们就得从不同的视角来看待软件架构设计这项工作）

2.五大视图：

|  |  |  |  |
| --- | --- | --- | --- |
| 视图 | 考虑的方面 | 工具 | 说明 |
| 逻辑架构 | 重点关注的是软件功能性需求。  系统功能的划分、  如何进行界面与用户的交互、  设计接口等 | 结构图  UML用例图  用例描述表  鲁棒图  UML类图等  （UML 已经成为对软件架构进行文档化的事实上的标准表示法。在视图文档的组织结构中， UML 主要用于表示元素或元素组的行为。） | 逻辑架构从系统用户的角度考虑问题，设计出的架构能够满足业务逻辑的需求；  逻辑架构关注功能，不仅包括用户可见的功能，还包括为实现用户功能而必须提供的“辅助功能模块” |
| 开发架构 | 重点关注的是开发编码实现方面的问题。  分层结构设计、  开发技术选项、  模块划分、  开发规范、  软件质量属性 | 绘图工具 | 从系统开发人员的角度来考虑，设计的架构要易于理解、开发、单元测试等，  开发架构关注程序包，不仅包括要编写的源程序，还包括可以直接使用的第三方SDK 和现场框架、类库，以及开发的系统将运行于其上的系统软件或中间件。关注编译时刻的静态依赖关系。 |
| 数据结构 | 数据架构不仅仅要考虑开发中涉及到的数据库，实体模型，也要考虑物理架构中数据存储的设计。  数据如何存储、  数据库表结构关系的设计、  如何设计实体、  使用关系型数据库还是非关系型 | Power Designer  Visio | 数据架构关注持久化数据的存储方案，不仅包括实体及实体关系的存储格式、还包括数据传递，数据复制，数据同步等策略。 |
| 运行架构 | 运行架构关注的不再是全局而是局部，着重关注那些关键点与难点，常常需要技术攻关与预研。主要考虑控制流、通讯机制、资源争用、锁机制、同步异步、并发、串行，同时也要考虑质量属性。  同步运行还是异步、  并发还是串行、  对象之间的交互、  安全性、可靠性、性能等 |  | 运行架构主要关注非功能需求；  运行架构关注进程、线程、对象等运行时概念，以及相关的并发，同步，通信等问题。运行架构关注运行期间各个单元的交互。 |
| 物理结构 | 物理架构主要考虑硬件选择和拓扑结构，软件到硬件的映射，软硬件的相互影响。网络方面、性能方面、部署方面 |  | 物理架构关注系统安装和部署在什么样的环境上。  物理架构关注“目标程序及其依赖的运行库和系统软件”最终如何安装或部署到物理机器，以及如何部署机器和网络来配合软件系统的可靠性，可伸缩性等要求。 |

3.五视图法分析的意义：

1）全面分析软件系统方方面面的问题

2）尽早地发现和排除项目风险与不确定因素

3）从不同角度去展现要设计的软件系统

4）为项目进行中不同的干系人提供指导：

    -- 逻辑架构描述系统功能，并指导系统测试

    -- 开发架构规范软件的层次及代码风格

    -- 数据架构指导数据库的设计

    -- 运行架构定义了一些关键过程的设计

    -- 物理架构明确软件如何部署与实施

**经典的体系结构风格：**

1）数据流风格: 批处理序列; 管道/过滤器。

实例：编译器、

优点：

* 设计者可以将整个系统的输入、输出特性简单的理解为各个过滤器功能的合成。
* 基于管道-过滤器风格的系统具有较强的可维护性和可扩展性
* 支持一些特定的分析，如吞吐量计算和死锁检测等
* 管道-过滤器风格具有并发性

缺点：

* 交互式处理能力弱
* 具体实现比较复杂
  + 数据流同步问题
  + 数据加密与解析

1. 调用/返回风格：主程序/子程序；面向对象风格；层次结构。

实例：ISO、OSI网络体系结构、

优点：

层次结构风格支持系统设计过程中的逐级抽象

* + 基于层次结构风格的系统具有较好的可扩展性
  + 层次结构风格支持软件复用

缺点：  
 并不是每个系统都可以很容易地划分为分层的模式

* + 很难找到一个合适的、正确的层次抽象方法。

1. 独立构件风格：进程通讯；事件系统。
2. 虚拟机风格：解释器；基于规则的系统。
3. 仓库风格：数据库系统；超文本系统；黑板系统。
4. 过程控制环路
5. C/S风格

客户机/服务器（C/S）体系结构是基于资源不对等，且为实现共享而提出来的。

C/S体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络。

优点：

* + 界面和操作可以很丰富
  + 安全性高
  + 响应速度快

缺点：

* + 适用面窄
  + 用户群固定
  + 维护成本高

1. B/S风格
2. B/S体系结构主要利用不断成熟的WWW浏览器技术，特别是浏览器嵌入的多种脚本语言

优点：

* 1. 维护和升级方式简单
  2. 交互性较强

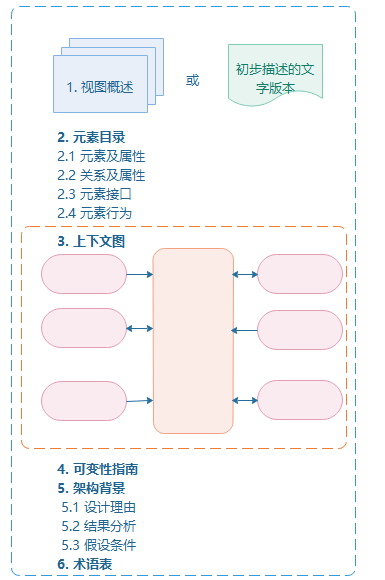
缺点：

* 1. 在速度和安全性上需要花费巨大的设计成本
  2. 通常需要刷新页面（Ajax等可以一定程度缓解该问题）
  3. 通信开销大

**软件架构文档化：（阅读**Documenting Software Architectures ，2nd，Paul Clements https://wiki.sei.cmu.edu/sad 笔记）

目的：架构的主要用途是充当项目关系人之间进行交流的工具，文档则促进了这种交流 —— 架构项目关系人希望从架构文档中获得自己所关心的架构信息。

视图编档：



**（1）视图概述**

* 通常是图形的，也可以是表格、文本地形式，使用UML语言描述
* 应该包括一个解释符号的键
* 显示元素和它们之间的关系
* 首先显示您要传达的有关视图的信息
* 应该确定视图范围之外的元素
  + 如果图中未明确标记外部实体，考虑添加上下文图

**（2）元素目录**

* 解释主要演示文稿中描述的元素及其属性
* 通常是带有元素名称和文字说明的表格
* 可能包含接口文档
* 可能包含行为文档

**（3）上下文图**

用图形展示系统如何与其环境相关

**（4）可变性指南**

描述架构的可变化点：如

* 可以对系统进行参数设置或重新配置的点。
  + 池中的实例数
  + 支持插件或附加组件
  + 支持不同版本的OS，数据库服务器或运行时环境
* 也许该视图是参考架构
  + 提供实例化它的准则

**（5）架构背景**

为架构的合理性提供足够的 、 令人信服的论据。包括：基本原理 、 分析结果及设计中所反映的假定。

（6）其他资讯

* + 重要设计决策的描述和依据（包括相关的被拒绝的替代方案）
  + 分析，原型和实验的结果

（7）父视图

如果当前视图是另一种视图的细化，请指明哪个视图

**UML建模**

UML 通过图形化的表示机制从多个侧面对系统的分析和设计模型进行刻画。

　　10种视图，四类：

　　1、用例图

　　2、静态图，包括 包图、类图、对象图。

　　类图的边表示类之间的联系，包括 继承、关联、依赖、聚合 等。

　　对象图描述在某种状态下或某一时间段，系统中 活跃的对象及其关系。

　　包由子包、类 组成。

　　3、行为图，包括 状态图、交互图、活动图，他们从不同的侧面刻画系统的动态行为。

　　交互图分为顺序图、合作图。顺序图强调对象之间 消息发送的时序。合作图更强调对象间的动态协作关系。

　　状态图描述对象的动态行为。

　　活动图描述操作序列，这些操作序列可以并发、同步，包含控制流、信息流。

　　4、实现图，包括 构件图、部署图。描述组成和分布情况。

部署图节点表示实际的计算机和设备，边表示节点之间的物理连接，也可以显示连接的类型及节点之间的依赖性。

**系统架构的评估：**

评估可以只针对一个体系结构，也可以针对一对一组体系结构。关注的是质量属性。

　　1、性能，是指系统的响应能力，多长时间对某个事件做出响应，或者某段时间内系统所能处理的事件的个数。

　　2、可靠性，是最重要的软件特性，平均失效等待时间MTTF，平均失效间隔时间MTBF

　　3、健壮性，不受错误使用和错误输入的影响。

　　4、可用性，正常运行的时间比例。经常用两次故障之间的时间长度或恢复正常的速度来表示。

　　5、安全性，阻止非授权用户。分为机密性、完整性、不可否认性、可控性 等特性。

　　6、可修改性，通过考察变更的代价衡量可修改性。

　　7.可扩展性，新特性来扩展软件系统，改进版本来替换构件并删除不需要的特性构件，需要松散耦合的构件。

8.可移植性。

**主要评估方法：**

　1、SAAM 非功能质量属性的体系结构分析方法，是最早形式成文档并得到广泛使用的分析方法。最初它用于比较不同的软件体系结构，以分析SA的可修改性。

　2、ATAM