软件体系结构

第一章 软件体系结构的基本概念

1.软件体系结构的概念

软件体系结构由构件，连接件与约束三大要素构成

构件：构件可以是一组代码，如程序的模块，也可以是一个可以独立运行的程序

连接件：连接件表示构件之间的相互作用，可以是过程调用，管道，消息

约束：一般为构件连接的条件

2.软件架构结构（重点是前两种，第三种了解即可）

（1）模块结构

模块结构所体现的是系统如何被构造成一组代码或者数据单元的决策

1）分解结构

这些单元是通过“子模块”关系将彼此相互关联起来的模块，展示了如何将大的模块，递归的分解为小的模块，直到它们足够小且容易理解

2）使用结构

单元之间以使用的这种依赖关系进行彼此关联，即一个单元取得正确结果，必须由另一个单元来确定，这就能说这两个单元存在使用关系

3）分层结构

一层就是功能一致的组合，在一个严格分层的系统中，第n层的模块可能仅使用第n-1层提供的服务

4）类的泛化结构

就是类于类之间的继承关系

5）数据模型

描述数据静态实体的静态信息以及它们之间的关系

（2）构件与连接件结构

构件与连接件的设计体现了系统如何设计为一组具有运行时行为（构件）与交互（连接件）的元素

1）服务结构

此处的结构单元是服务，服务通过一定的协作机制进行交互

2）客户机/服务器

构件是客户机与服务器，连接件是协议与消息

3）共享数据的存储结构

构件是作用在数据上的存储单元与计算单元。连接件则是提供数据存储机制

4）并发结构

这种结构使得架构师能够决定系统并行的时机以及可能会发生资源占用的位置。此处的结构单元是组件，连接件是它们的通信机制

（3）分配结构

分配结构体现了**系统**将如何在其环境中**与非软件结构关联**起来

1）部署结构

部署结构显示了软件是如何分配到硬件处理以及通讯元素上的

2）实现结构

该结构展示了软件元素如何映射到系统开发、集成或配置控制环境中的文件结构上

3）工作分配结构

该结构将**实现与集成模块**责任分给适当的开发小组

3.软件架构与视图模型

（1）视图的定义

在软件体系结构中，一个架构视图是对于从某一视角或某一点上看到了系统所做的简化描述，描述涵盖了系统的一个特定方面，而省略了与此方面无关的元素

（2）架构模型

一个架构模型可以看做软件架构中视点的一种表达。例如，在软件设计中所使用的UML图，如果是其架构意义上的图（**组件图，部署图**），就可称为架构模型

（3）视图模型

一个视图模型是指一组用来构建系统或软件架构的相关视图的集合，这样一组从不同的视角表达系统组合在一起构成对系统比较完整的表达

例如：Kruchten 4+1视图

4.软件体系结构核心元模型

（1）构件

构件是**具有某种功能的可复用的软件结构单元**，任何在系统中承担一定功能，发挥一定作用的软件体都可看作构件，如程序中的函数，模块，对象，类，甚至是文件

（2）连接件

连接是构件之间建立和维护行为关联与信息传递的途径，需要两方面的支持：首先是连接发送与维护的机制，其次是连接正确有效的进行信息交换的规则，这两方面简称为，"规则"，"协议"

5.其他相关概念

（1）架构模式

架构模式表示软件系统基本结构化的图式，其所代表的是在软件系统中最高等级的模式，常用大粒度体系架构元素设计

（2）设计模式

提供了一个用于细化软件系统的子系统或组件，或它们之间关系的图式，其所代表的是软件系统中中等粒度的模式

常用于更小粒度体系结构元素设计

第二章 软件质量属性

1.质量属性的说明

质量属性场景的描述，一般包括6个方面

1. 刺激源：某个生成刺激的实体（人，计算机系统）
2. 刺激：当到达引起系统进行响应的条件
3. 环境：刺激到达时，某系统所处的状态，如系统可能出于过载，或者正在运行的其他情况
4. 制品：被刺激的部分
5. 响应：表示系统在刺激到达后所采取的行动或者措施（做完工作的结果）
6. 相应度量：当响应发送时，我们以某种方式对其进行衡量（结果的评价或评价结果的标准）

2.质量属性及其定义

（1）概念完整性

概念完整性指设计架构的一致性，要求整个架构中同一件事情按照同样的方法被执行，例如构件之间发送消息有很多种方式，如消息，数据结构，时间信号等。按照概念一致性，我们只能选用一种方式

（2）可维护性

可维护性指的是系统承受一定修改的能力，当增加或修改功能、修复错误，以及满足新业务需求时，针对应用程序可能进行的修改可能会影响组件，服务，特征，接口。还包括系统中断或移除某个操作时系统要花费多长时间才能正常运行

（3）可重用性

可重用性定义了组件和子系统在其他应用程序或应用场景中重用的能力

（4）可用性（需计算）

$$a = \frac{MTBF}{MTBF + MTTR}$$

MTBF代表的是平均故障时间，**MTTR**代表平均修复时间，计算方式如下所示：

$$MTBF = \frac{\text{无故障工作时间}}{\text{故障次数}}$$

$$MTTR = \frac{\text{总的宕机时间}}{\text{故障次数}}$$

（5）互操作性

互操作性指的是两个或多个构件协作的能力，即不同来源的构件能相互协调、通信，共同完成更复杂的功能的能力

（6）可管理性

可管理性是查看与修改执行系统或软件状态的能力，描述了系统管理员管理应用程序的难易程度

（7）性能

性能与时间有关。性能描述的是系统的响应速度，也就是要在一定时间间隔内完成指定行为的能力

（8）可靠性（需会计算）

是指一切旨在避免、减少、处理、质量软件故障的分析、设计、测试方法、技术和实践活动。可靠性函数描述，即软件在[0,t]时间内，不发生失效的概率

$$R(t) = e^{-(t/\theta)}$$

t为总时间，θ为MTBF

（9）可伸缩性

可伸缩性是一衡量软件系统适应系统规模增长（用户数量，数据量，网络节点）的能力

（10）安全性

指软件在其生命周期内，系统向合法用户提供服务的同时，防止恶意行为或系统设计的使用方式之外的能力

（11）可支持性

可支持性是指系统不正常工作的时候，提供信息与解决问题的能力

（12）可测试性

软件的可测试性是指通过测试（通常是基于运行的测试）揭示软件缺陷的容易程度

（13）易用性（通俗来讲就是好不好用，功能是否齐全）

易用性关注的是对用户来说完成某个期待的任务的难易程度以及系统提供给用户支持的种类。

（14）可变性

可变性表示系统支持的制品（需求，测试计划，配置说明书等）适应变化的能力

（15）可移植性

可移植性指构建于任何一个平台上的软件能够运行到另一个不同平台上的难易程度

（16）开发的可分布性

开发的可分布性指设计的软件支持分布式软件部署的能力

（17）可部署性

可部署考虑一个可执行文件到达主机平台，以及如何依次被调用

（18）移动性

移动性处理平台的移动与功能可提供性问题，尺寸，显示类型，输入输出设备，带宽可用以及容量等

（19）多语言适应性（略）

（20）可定制性

对全球化应用而言，项目团队需要考虑定制需求，即系统在多大程度上由总部控制和多大程度上进行本地化管理

第三章—软件体系结构风格及案例

1.概念

软件架构风格是一个面向一类给定环境的架构设计决策的集合，这些设计决策形成了一种特定的模式，为一族系统提供粗粒度的抽象框架

2.数据流风格

数据流风格的基本构件是**处理**，构件接口包括**输入端口与输出端口**，构件在工作时从输入端口读取数据，经过计算，处理向输出端口写数据

数据流风格的连接件是数据流，它是一个特殊的构件，具有输入与输出两种接口，数据流负责把一个处理的输出端口传到另一个处理的输入端口，这种传送是单向的，异步的，有缓冲

（1）管道—过滤器

一个步骤的输出是另一个步骤的输入，每个步骤由一个过滤器实现，处理步骤之间的数据传输由管道负责，**管道**即是连接件

关键特征：整个管道过滤器网络持续的、增量的处理数据

（2）批处理

将管道过滤器风格中的过滤器的输入输出限制为单一的，则退化为批处理，每一步都是独立的，顺序执行的，组件间只通过数据传输交互

3.过程调用风格

过程调用风格，即通过子程序之间的传参与返回值实现数据共享进行交互，通过对子程序的不同调用顺序实现程序流程控制

（1）主程序/子程序体系结构风格

像C语言的编程，所有程序代码都在一个主程序中。使用单线程控制，把问题处理为若干步骤，按照自上而下的方法，基于"定义—使用"的关系，构件则是主程序与子程序，子程序通常可合成为模块

（2）数据抽象与面向对象体系结构风格

这种风格的构件是对象，对象是抽象数据类型的实例。数据的表示与他们的操作应该封装起来

（3）层次结构风格

这个风格的特点是每一层为上一层提供服务，使用下一层服务，只能见到自己邻接的层。大的问题逐步转化为小问题，逐步解决，隐藏了很多复杂度，连接件通过决定层间如何交互来定义

4.独立构件风格

该风格能很好的支持软件重用与演化，很容易将一个构件集成到系统之中

（1）进程间的通信风格

构件是独立的进程，连接件是消息传递，消息传递为了实现进程间的同步与对共享资源的互斥操作

（2）基于事件的隐式调用

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，系统中的构件只需在一个或者多个事件中注册，当一个事件触发时，系统自动调用在这个事件中注册的所有过程

事件分发到已注册模块中有两种策略：**一种是没有派遣模块的时间管理器，一种有带派遣模块的事件管理器**

在无调度模块的事件系统中，各模块被称为观察者/被观察者，每一个模块都允许其他模块向自己发送的某些消息表明兴趣（通过观察者模式来实现）

在有独立调度模块的系统中，事件派遣模块负责接收到来的事件并派遣到其他模块，有两种策略，一种是广播式，即派遣模块将事件广播到所有模块，但只有感兴趣的构件才去响应事件，触发自身的行为，另一种为选择广播式只将消息传递到已经注册的模块中，选择发布包括两种派遣策略，点对点与发布—订阅模式

点对点：使用一个队列存储这些消息，直到接收端应用连接到队列，取回这些消息被唯一的消费者消费后即销毁消息

发布—订阅：一个事件可以被多个消费者消费，事件发送给订阅者之后，不会马上从topic删除，topic会在事件过期之后将其删除

5.层次风格

层次风格有两种：严格分层不允许进行跨层交互，松散分层允许跨层交互。由下层构件向上层构件提供服务1

（1）逻辑分层

按照相关功能与组件进行分组，例如MVC

（2）物理分层

物理部署，联系部署图

6.虚拟机风格

虚拟机风格也称为解释器风格，解释器的核心是虚拟机，包含三个被动数据组件与一个主动组件，分别是被解释执行的程序，用于保存程序当前执行状态的数据组件（内存），保存当前解释器引擎状态的组件（指令计数器），以及虚拟机解释引擎（CPU）

虚拟机风格存在两种子风格：解释器风格与基于规则的系统

（1）解释器风格

解释器是一个用来执行其他程序的程序，它针对不同的硬件平台实现了一个虚拟机，高抽象的层次翻译为低抽象层次所能理解的指令，以消除在程序语言与硬件之间的语义差异。一个解释器由**伪程序**与**解释引擎**二部分组成

（2）基于规则的系统

现实里的业务需求频繁的发生变化，软件系统也要随之适应，最好的办法是把频繁变化复杂的业务逻辑抽取出来，形成独立的规则库。

7.客户机/服务器风格

1.胖/瘦客户端

根据客户端所执行操作的量来决定，胖瘦客户端的特点如下：

1. 较低的服务器需求
2. 离线工作
3. 更好的多媒体性能
4. 更灵活
5. 充分利用已有资源
6. 更高的服务器容量

瘦：

1. 单点故障，服务器承担了大量业务，易于维护
2. 廉价的客户端硬件，活都让服务器干了，客户端低配就够用
3. 有限的显示性能，配置都不高，要啥自行车

2.传统的C/S风格

典型的胖客户端

3.多层C/S风格

把业务交给业务服务器，数据交给数据服务器，本地只负责显示与发送请求

4.B/S风格

大部分服务都在服务器执行，是一种典型的胖服务器风格

8.表示分离风格

（1）MVC设计模式

略！

（2）MVP设计模式

MV层不发生变化，引入主持者层来替换控制层，主持者层的功能如下：

主持者控制着视图的所有执行逻辑，还负责视图之间的同步，当用户完成了一些操作后，**由视图通知主持者，然后由主持者更新模型，并与视图信息同步**

9.SOA风格（了解定义即可）

面向服务架构（SOA）是一种软件架构的设计模式，用于将软件的不同功能以服务的方式提供给其他应用程序使用，使各种按照这种方式构建的各类系统中的服务可以以一种统一的方式进行交互

第四章 软件体系结构的描述与建模

软件体系结构描述语言分为三大类：非正式的**框线图**，**正式的体系结构描述语言ADL**，与基于**UML**的符号化表达

1.Kruchten 4+1 视图

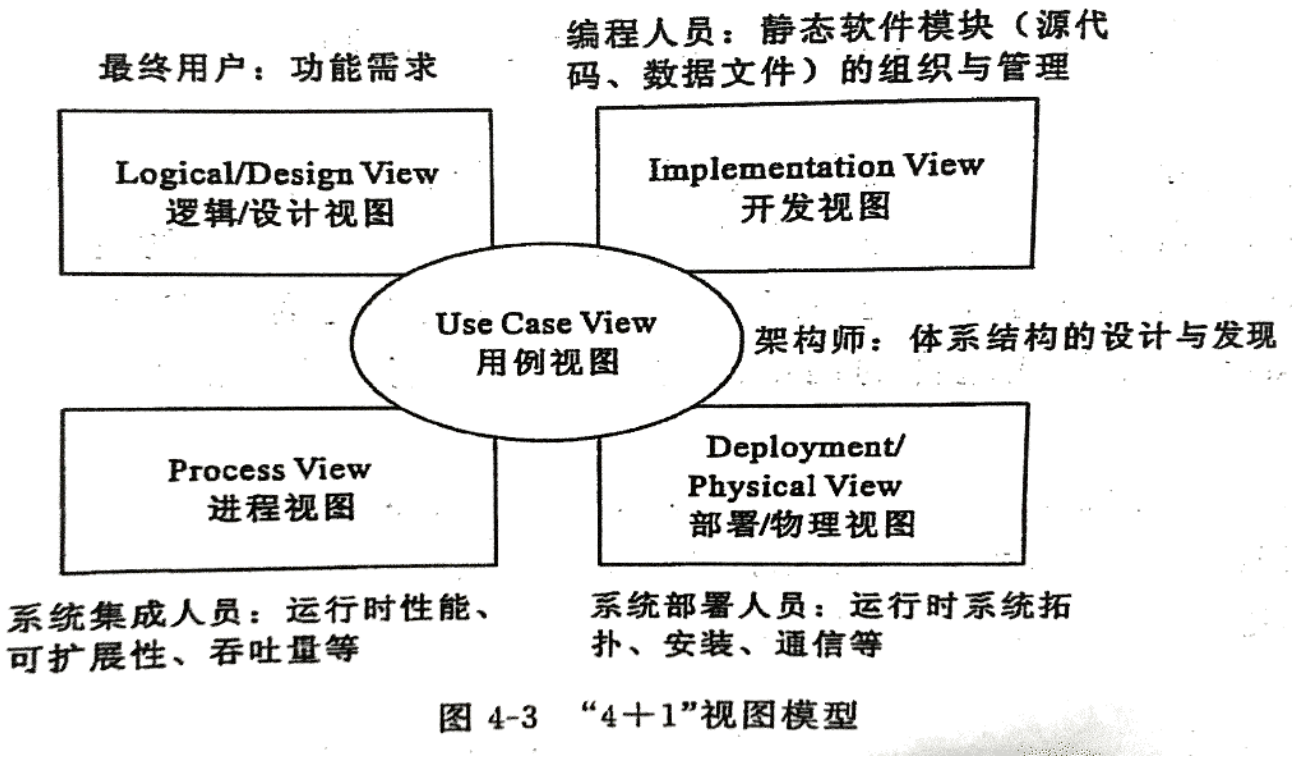

图 4-3 “4＋1”视图模型

（1）逻辑视图

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，它通过结构元素，核心抽象和机制，关注点的分离和职责的划分来实现系统功能。

常用来描述的UML图：类图，包图，组合结构图，状态图

（2）开发视图

开发视图也称**模块视图**（画这个重点单纯是为了方便理解），主要是侧重与软件模块的组织与管理。开发视图要考虑软件开发的容易性，软件的重用与软件的通用性，要充分考虑不同开发工具所带来的局限性

用来描述的UML图：组件图

（3）进程视图

主要关注一些非功能性的需求，如系统的性能与可用性，进程视图的并发性、分布性、系统的继承性与容错能力，以及逻辑视图中的主要抽象与如何结合进程结构

用来描述的UML图：**顺序图，通信图，活动图**，计时图，交互概览图

（4）部署视图

部署视图主要考虑如何将软件映射到硬件上，它通常要考虑系统的性能，规模，可靠性等。解决系统的拓扑结构，系统安装，通信问题等

用来描述的UML图：**部署图**

（5）用例视图（用来联系其余四个视图的视图）

在软件的开发周期中是最先创建的视图，用例视图用来捕获最终用户需求的功能性，使四个视图有机的联系起来，从某种意义上来说，用例视图是最重要的抽象

用来描述的UML图：用例图

2.视图间的关系

首先，逻辑视图与进程视图是概念层次的视图，一般是用于进行分析与设计的，开发视图与部署视图是在物理层面上的，一般用来部署与构建实际的应用程序组件。

其次，逻辑视图是和开发视图的功能性联系紧密，它们描述了功能性的是如何建模实现的。进程视图与部署视图则是通过使用行为和物理建模实现了非功能性的方面。

最后，用例视图使得结构元素在逻辑视图中进行分析并在开发视图中进行实现，用例视图的场景是在进程视图中实现的并是在物理视图中部署的

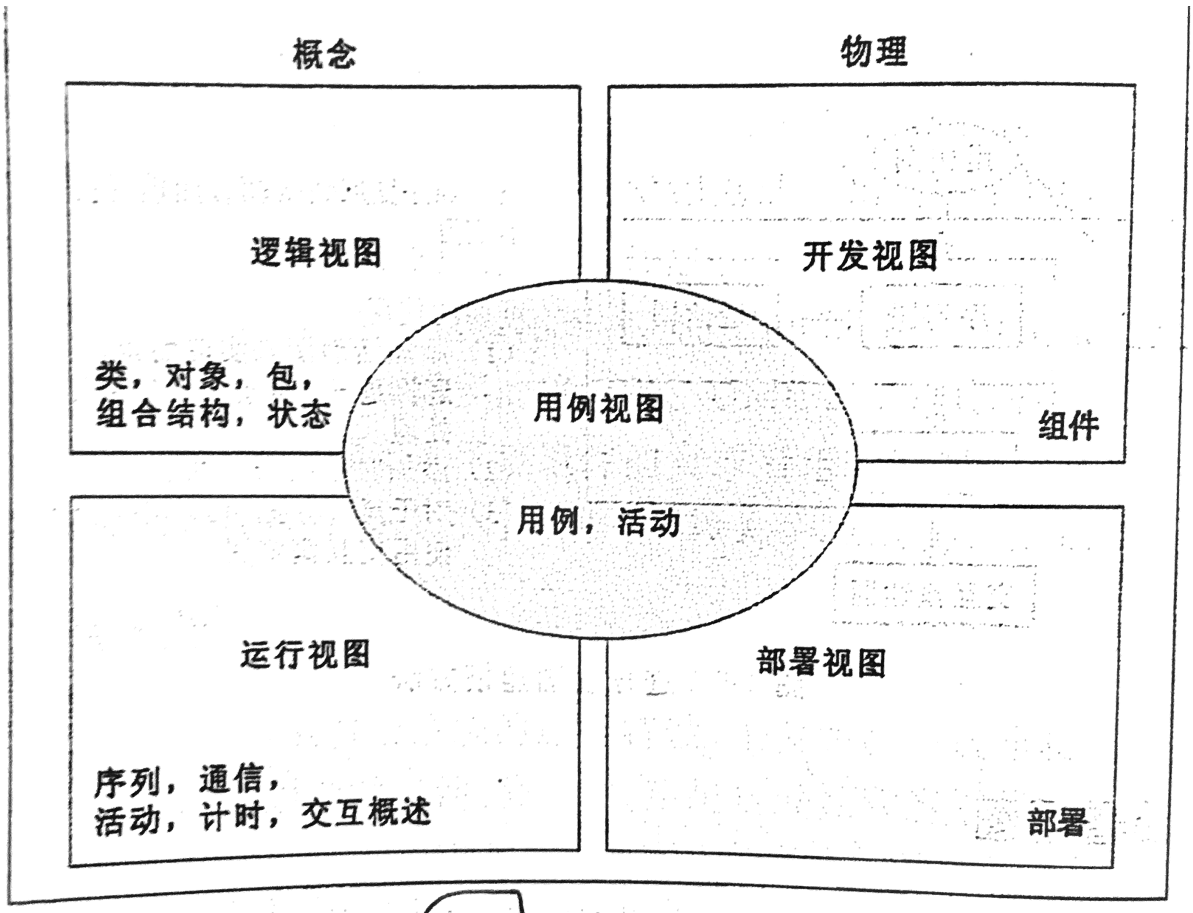
概念
物理
逻辑视图
类，对象，包，
组合结构，状态
开发视图
组件
用例视图
用例，活动
运行视图
序列，通信，
活动，计时，交互概述
部署视图
部署

图 4-7 某系统鸟瞰图