实验十 软件体系结构设计（二）

实验目的：

1. 体系结构风格和视图特点

2. 研究经典软件体系结构案例

3. 继续补充和修改自己项目的SAD

实验内容：

1. 对比书上各种软件体系结构风格和视图特点，思考自己项目属于哪种设计风格？

网上搜索最新的软件体系结构资料，如MVC、Kruchten 4+1视图等。

2. 参阅课本和网上资料，研究经典软件体系结构案例KWIC。

An Introduction to Software Architecture，4.1节

On-the-Criteria-To-Be-Used-in-Decomposing-Systems-into-Modules（Example System 1）

http://www.cs.cmu.edu/~ModProb/index.html

针对KWIC和自己项目，参考课本ch5 表5-3，小组成员每人给几种不同的体系结构风格设计打分，评最佳。

3. 补充和修改自己项目的SAD

记录项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。

# 1.C2 体系结构风格

C2结构是一个层次网络，包括构件和连接件两种软件元素。构件和连接键都是包含顶部和底部的软件元

素。构件和构件之间只能通过连接件进行连接，而连接件之间则可以直接进行连接。构件的顶部、底部

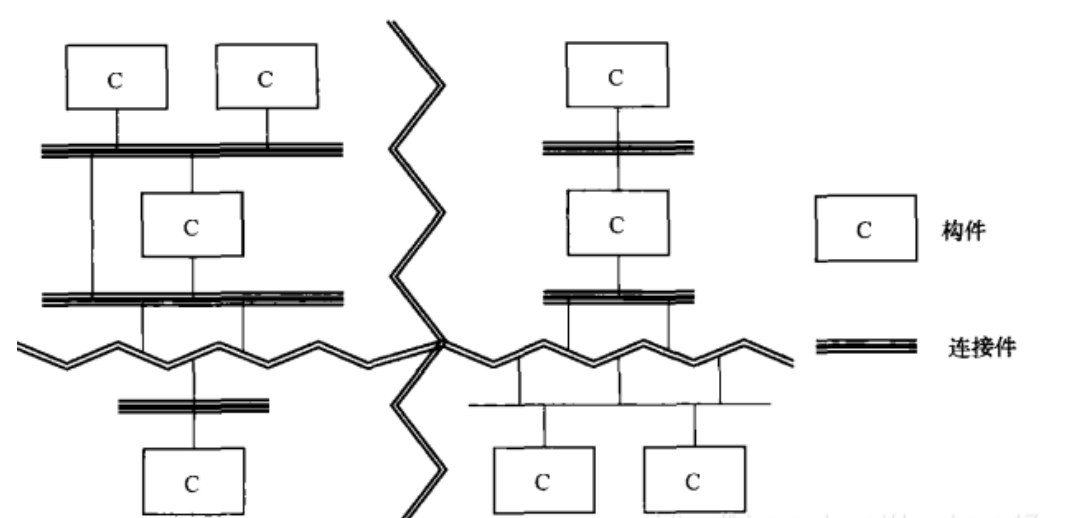
分别与连接件的底部、顶部连接，连接件的顶部、底部也分别与连接件的底部、顶部连接。

在C2体系结构中，构件之间的所有通信必须使用消息传递机制来实现。构件之间所有传递的信息可以分

为两种，一种是向上层构件发出服务请求的请求消息，另一种是向下层构件发出指示状态变化的通知消

息。连接件负责消息的过滤、路由、广播、通信和相关处理。

原理图：



优点：

1. 可以使用任何编程语言来开发构件，构件重用和替换比较容易实现
2. 具有一定的扩展能力，可以有多种不同粒度的构件
3. 构件不需要共享地址空间，避免了共享全局变量所造成的复杂关系
4. 具有良好的适应性
5. 在C2体系结构中，可以使用多个工具集和多种媒体类型，能够动态地更新系统的框架结构

缺点 ：

1. 构件和构件之间不允许直接相连
2. 与某一个连接件相关联的构件和连接件的数目没有限制

# 数据共享 体系结构风格

数据共享风格也成为仓库风格。

在这种风格中，有两种不同类型的软件元素：一种是中央数据单元，也成为资源库，用于表示系统的当

前状态；另一种是相互依赖的构件组，这些构件可以对中央数据单元实施操作。中央数据单元和构件之

间可以进行信息交换，这是数据共享体系结构的技术实现基础。

根据所使用的控制策略不同，数据共享体系结构可以分为两种类型，一种是传统的数据库，另一种是黑

板。

如果由输入流中的事件来驱动系统进行信息处理，把执行结构存储到中央数据单元，则这个系统就是数

据库应用系统。

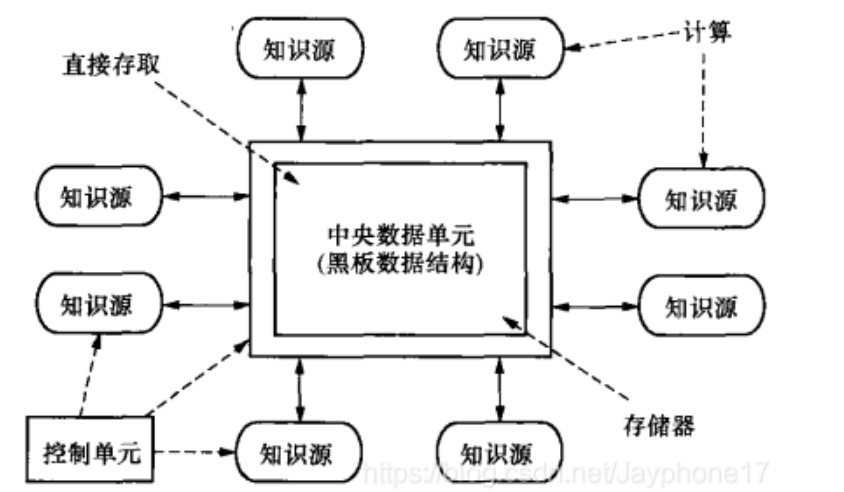
如果由中央数据单元的当前状态来驱动系统运行，则这个系统就是黑板应用系统。

黑板是数据共享体系结构的一个特例，用以解决状态冲突并处理可能存在的不确定性知识源。黑板常用于信号处理，如语音和模式识别，同时在自然语言处理领域中也有广泛的应用，如机器翻译和

句法分析。

原理图：

一个典型的黑板系统主要包括知识源、中央控制单元、控制单元



优点：

1. 便于多客户共享大量数据，而不必关心数据是何时产生的、由谁提供的及通过何种途径来提供

2. 便于将构件作为知识源添加到系统中来

缺点：

1. 对共享数据结构，不同知识源要达成一致

2. 需要同步机制和加锁机制来保证数据的完整性和一致性，增大了系统设计的复杂度

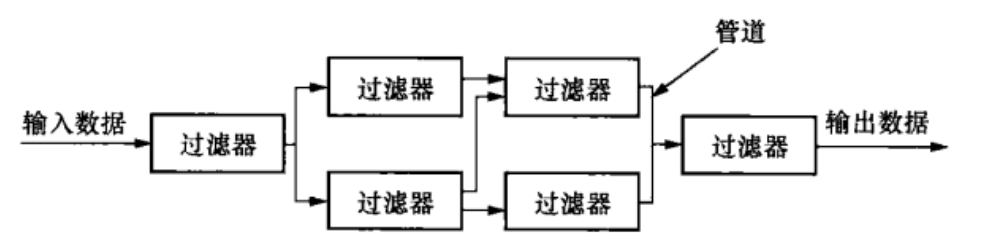
# 管道/过滤器 体系结构风格

主要包括过滤器和管道两种元素。在这种结构中，构件被称为过滤器，负责对数据进行加工处理。每个

过滤器都有一组输入端口和输出端口，从输入端口接收数据，经过内部加工处理之后，传送到输出端口

上。数据通过相邻过滤器之间的连接件进行传输，连接件可以看作输入数据流和输出数据流之间的通

路，这就是管道。



优点：

1. 简单性。
2. 支持复用。
3. 系统具有可扩展性和可进化型。
4. 系统并发性（每个过滤器可以独立运行，不同子任务可以并行执行，提高效率）。
5. 便于系统分析。

缺点：

1. 系统处理工程是批处理方式。
2. 不适合用来设计交互式应用系统。
3. 由于没有通用的数据传输标准，因此每个过滤器都需要解析输入数据和合成数据。
4. 难以进行错误处理。

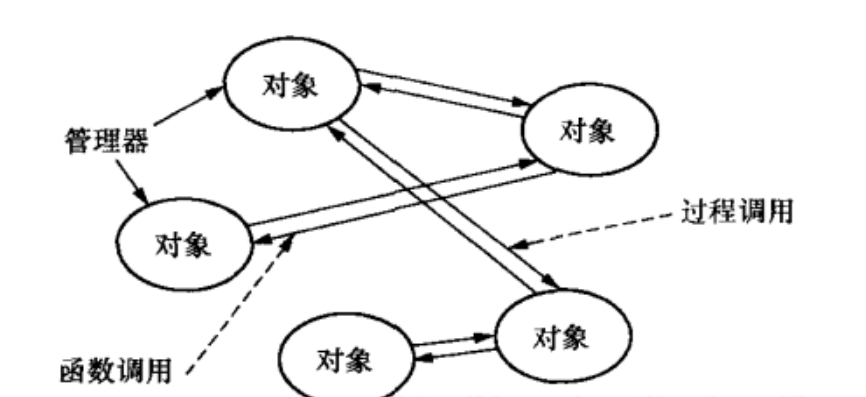
# 面向对象 体系结构风格

在面向对象体系结构中，软件工程的模块化、信息隐藏、抽象和重用原则得到了充分的体现。在这种体

系结构中，数据表示和相关原语操作都被封装在抽象数据类型中。在这种风格中，对象是构件，也成为

抽象数据类型的实例。对象与对象之间，通过函数调用和过程调用来进行交互。

原理图：



优点：

1. 一个对象对外隐藏了自己的详细信息
2. 对象将数据和操作封装在一起
3. 继承和封装方法为对象服用提供了技术支持

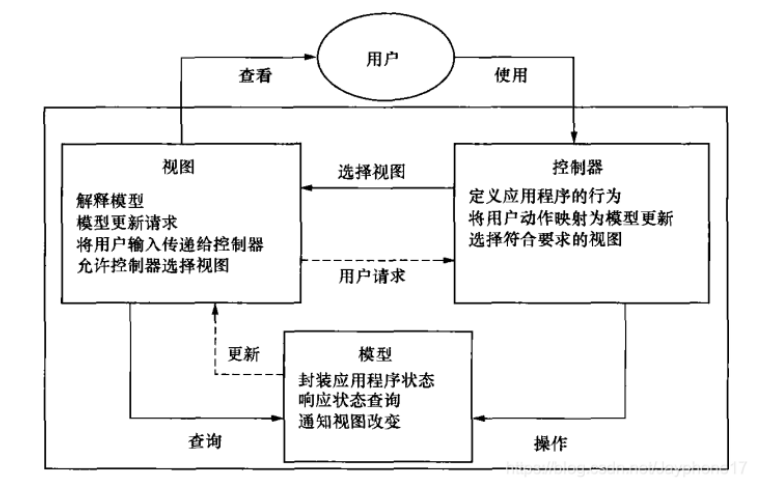
缺点：

1. 如果一个对象要调用另一个对象，则必须知道它的标识和名称
2. 会产生连锁反应

# MVC 体系结构风格

模型-视图-控制器（Model-View-Controller，MVC）是一种常见的体系结构风格。MVC被广泛应用与用户交互程序的设计中。

原理图：



优点：

1. 多个视图与一个模型相对应

2. 具有良好的移植性。

3. 系统被分割为三个独立的部分，当功能发生变化时，改变其中的一个部分就能够满足要求。

缺点：

1. 增加了系统设计和运行复杂性。

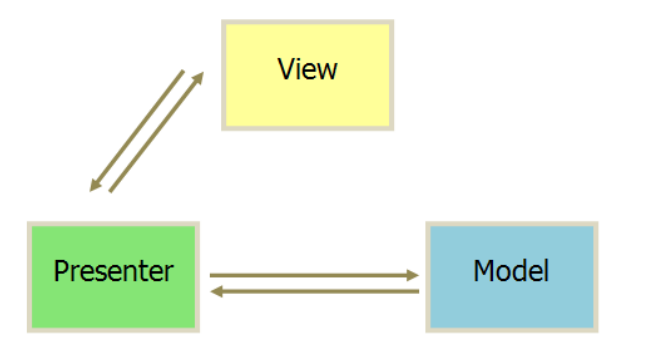
2. 视图与控制器连接过于紧密，妨碍两者的独立复用。

3. 视图访问模型的效率比较低。

**改进**：

MVP、MVVM：

MVP 模式将 Controller 改名为 Presenter，同时改变了通信方向。



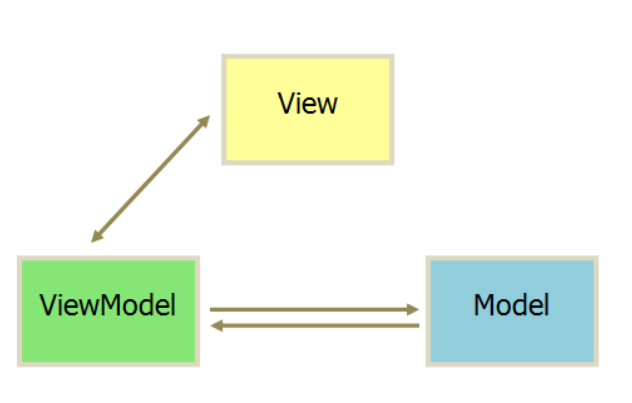
特点：各部分之间的通信，都是双向的。

View 与 Model 不发生联系，都通过 Presenter 传递。

View 非常薄，不部署任何业务逻辑，称为"被动视图"（Passive View），即没有任何主动性，而

Presenter非常厚，所有逻辑都部署在那里。

MVVM 模式将 Presenter 改名为 ViewModel，基本上与 MVP 模式完全一致。



唯一的区别是，它采用双向绑定（data-binding）：View的变动，自动反映在 ViewModel，反之亦然。

Angular 和 Ember 都采用这种模式。

# Kruchten 4+1视图

软件架构用来处理软件高层次结构的设计和实施。它以精心选择的形式将若干结构元素进行装配，从而

满足系统主要功能和性能需求，并满足其他非功能性需求，如可靠性、可伸缩性、可移植性和可用性。

Perry 和 Wolfe 使用一个精确的公式来表达，该公式由 Boehm 做了进一步修改：

软件架构 ＝ {元素，形式，关系/约束}

软件架构涉及到抽象、分解和组合、风格和美学。我们用由多个视图或视角组成的模型来描述它。为了

最终处理大型的、富有挑战性的架构，该模型包含五个主要的视图（请对照图 1）：

逻辑视图（Logical View），设计的对象模型（使用面向对象的设计方法时）。

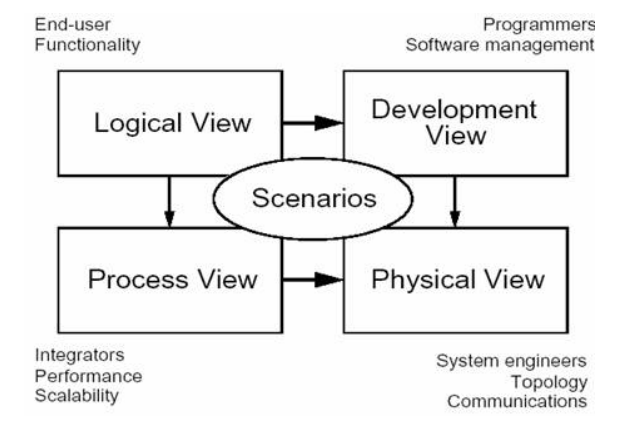
过程视图（Process View），捕捉设计的并发和同步特征。

物理视图（Physical View），描述了软件到硬件的映射，反映了分布式特性。

开发视图（Development View），描述了在开发环境中软件的静态组织结构。

架构的描述，即所做的各种决定，可以围绕着这四个视图来组织，然后由一些用例 （use cases）或场

景(scenarios)来说明，从而形成了第五个+1视图。



# 项目风格：

## RESTful

REST是Representational State Transfer(在表现层上的状态转化)的缩写，这个词的意思要在文章的后面

才能解释清楚。REST是一种WEB应用的架构风格，它被定义为6个限制，满足这6个限制，能够获得诸多

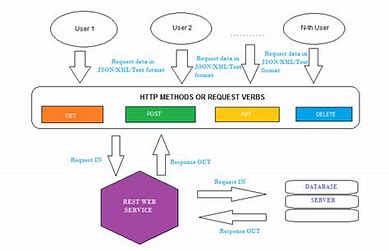
优点（详细优点在文章最后总结）。

先用一句话来概括RESTful API(具有REST风格的API): 用URL定位资源，用HTTP动词

（GET,HEAD,POST,PUT,PATCH,DELETE）描述操作，用状态码表示操作结果。

RESTful API有助于客户端和服务端的功能分离，服务器完全扮演着一个“资源服务商”的角色。各种不同

的客户端都可以通过同一套API与这个“资源服务商”交流，从而与资源进行互动。



REST的规则概括起来有以下五个方面：

1. 以资源为核心

什么是资源？在web应用中，什么都是资源（或者抽象成资源）。资源可以是一个实体，也可以是一个

过程，或者也可以是一个版本。比如，商品是资源，库存是资源，价格是资源等等。

REST没有“服务”概念，那是SOA的事情。

2. 每个资源分配唯一一个URL

每一个资源一个URL，而且是唯一的URL，比如我们使用http://www.haodaquan.com/v0/product 这

个URL标识商品资源。

3. 通过标准的HTTP（HTTPS）方法操作资源。

操作（调用）资源使用HTTP中的标准方法，比如常见的GET（获取资源），POST（存储资源），

PUT（重置资源），PATCH（局部更新资源）、DELET（删除资源），也可能会有HEADER和OPTIONS

方法。

4. 资源的表现层可以是xml、json或者其他

这里是指调用资源后返回的数据格式，一般以JSON和XML居多，特别json格式可以直接被js使用，爽的

很，经常被使用。

5. 操作是无状态的

REST架构是不存储调用者的状态的。换句话说，比如登陆才能调用资源，你可别指望登陆这个状态让

REST服务帮你存储。换句话说，业务层的活，REST不干。

遵守REST规则，可以让架构变得轻便，简单，安全性和可靠性高，更重要的是扩展性好。当然，利用应

用程序的多层架构可是实现更高的重用性、安全性和扩展性。有了 REST 服务，前端关注界面展现，后

端关注业务逻辑，分工明确，职责清晰。

从面向实用的角度来看，REST 架构风格可以为 Web 开发者带来三方面的利益：

简单性

采用 REST 架构风格，对于开发、测试、运维人员来说，都会更简单。可以充分利用大量 HTTP 服务器

端和客户端开发库、Web 功能测试 / 性能测试工具、HTTP 缓存、HTTP 代理服务器、防火墙。这些开

发库和基础设施早已成为了日常用品，不需要什么火箭科技（例如神奇昂贵的应用服务器、中间件）就

能解决大多数可伸缩性方面的问题。

可伸缩性

充分利用好通信链各个位置的 HTTP 缓存组件，可以带来更好的可伸缩性。其实很多时候，在 Web 前端

做性能优化，产生的效果不亚于仅仅在服务器端做性能优化，但是 HTTP 协议层面的缓存常常被一些资

深的架构师完全忽略掉。松耦合

统一接口 + 超文本驱动，带来了最大限度的松耦合。允许服务器端和客户端程序在很大范围内，相对独

立地进化。对于设计面向企业内网的 API 来说，松耦合并不是一个很重要的设计关注点。但是对于设计

面向互联网的 API 来说，松耦合变成了一个必选项，不仅在设计时应该关注，而且应该放在最优先位

置。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **属性** | **优先级** | **共享数据** | **面向对象** | **管道/过滤器** | **MVC** |
| 易于改变算法 | 1 | 1 | 2 | 3 | 5 |
| 易于改变数据表示 | 4 | 1 | 5 | 4 | 2 |
| 易于改变功能 | 3 | 4 | 3 | 3 | 4 |
| 好的性能 | 3 | 4 | 1 | 4 | 3 |
| 有效的数据表示 | 3 | 5 | 5 | 5 | 6 |
| 易于复用 | 5 | 1 | 3 | 3 | 2 |
| 总结 |  | 49 | 64 | 70 | 74 |