## 架构风格

数据流风格：批处理序列、管道-过滤器

调用/返回风格：主程序/子程序、面向对象、层次结构

独立构建风格：进程通信、事件驱动系统（隐式调用）

虚拟机风格：解释器、基于规则的系统

仓库风格：数据库系统、超文本系统、黑板系统

#### 软件架构风格 – 数据流风格

**批处理序列：**

构件为一系列固定顺序的计算单元，构件之间只通过数据传递交互。每个处理步骤是一个独立的程序，每一步必须在其前一步结束后才能开始，数据必须是完整的，以整体的方式传递。

**管道-过滤器：**

1. 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。这个过程通常是通过对输入数据流的变换或者计算来完成的，包括通过计算和增加信息以丰富数据、通过浓缩和删除以精简数据、通过改变记录方式以转化数据和递增转化数据等。这里的构建称为过滤器，连接件就是数据传输的管道，将一个过滤器的输出传到另一个过滤器的输入。
2. 早期编译器就是采用这种架构，要一步一步处理的，均可考虑采用此架构风格。

#### 软件架构风格 – 调用/返回风格

**主程序/子程序：**

单线程控制，把问题划分为若干个处理步骤，构件即为主程序和子程序，子程序通常可合并为模块。过程调用作为交互机制，既充当连接件的角色。调用关系具有层次性，其语义逻辑表现为主程序的正确性取决于它调用的子程序的正确性。

**面向对象：**

构建是对象，对象是抽象数据类型的实例。在抽象数据类型中，数据的表示和他们的相应操作被封装起来，对象的行为体现在其接受和请求的动作。连接件即是对象间交互的方式，对象是通过函数和过程的调用来交互的。

**层次结构：**

构件组织成一个层次机构，连接件通过决定层间如何交互的协议来定义。每层为上一层提供服务，使用下一层的服务，只能看到与自己相邻的层，通过层次结构，可以将大的问题分解为若干个渐进的小问题逐步解决，可以隐藏问题的复杂度，修改某一层，最多影响其相邻的两层（通常只能影响上层）

优点：

1. 这种风格支持基于可增加抽象层的设计，允许将一个复杂的问题分解成一个增量步骤序列的实现。
2. 不同的层次处于不同的抽象级别：越靠近底层，抽象级别越高；越靠近顶层，抽象级别越低。
3. 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，同样为软件复用提供了强大的支持。

缺点：

1. 并不是每个系统都可以很容易的划分为分层的模式。
2. 很难找到一个合适的、正确的层次抽象方法。

#### 软件架构风格 – 独立构件风格

**进程通信：**

构件是独立的过程，连接件是消息传递。构件通常是命名过程，消息传递的方式可以是点对点、异步或者同步方法，以及远程过程（方法）调用等。

**事件驱动系统（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。构件中的过程在一个或多个事件中注册，当某个事件触发时，系统自动调用在这个事件中注册的所有过程，一个事件的触发导致了另一个模块的过程调用，这种风格中的构件是匿名的过程，他们之间交互的连接件往往是以过程之间的隐式调用来实现的。

主要优点是为软件复用提供了强大的支持，为构建的维护和演化带来了方便；

其缺点是构建放弃了对系统计算的控制。

#### 软件架构风格 – 虚拟机风格

**解释器：**

解释器通常包含一个完成解释工作的解释引擎、一个包含将被解释的代码的存储区、一个记录解释引擎当前工作状态的数据结构，以及一个记录源代码被解释执行的进度的数据结构。具有解释器风格的软件中包含有一个虚拟机，可以仿真硬件的执行过程和一些关键应用，其缺点是执行效率比较低。

**基于规则的系统：**

基于规则的系统包含规则集、规则解释器、规则/数据选择器和工作内存，一般用在人工智能领域和DSS中。

#### 软件架构风格 – 仓库风格（以数据为中心的风格）

仓库风格中构件分两种：一种是中央数据结构，保存系统当前的状态。另一种是独立构件，对中央数据存储进行操作。

**数据库系统**

**黑板系统**

包含知识源、黑板和控制三部分。知识源包括若干独立计算的不同单元，提供解决问题的知识。知识源响应黑板的变化，也只修改黑板；黑板是一个全局数据库，包含问题域解空间的全部状态，是知识源相互作用的唯一媒介；知识源响应是通过黑板状态的变化来控制的，黑板系统通常应用在对于解决问题没有确定性算法的软件中（信号处理、问题规划和编译器优化等）

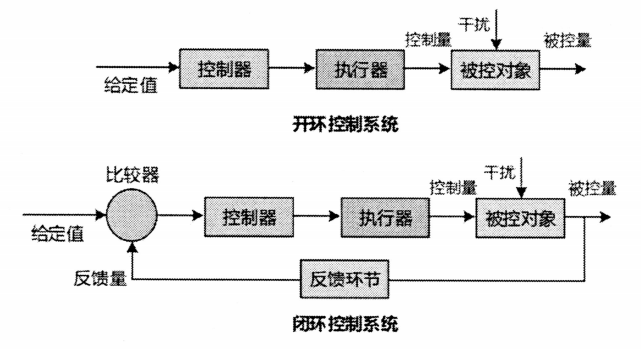
**超文本系统**

构件以网格连接方式相互连接，用户可以在构件之间进行按照人类的联想思维方式任意跳转到相应构件，超文本是一种非线性的网状信息组织方式，他以节点为基本单位，链作为节点之间的联想式关联。超文本系统通常应用在互联网领域。

**现代集成编译环境一般采用这种架构风格**

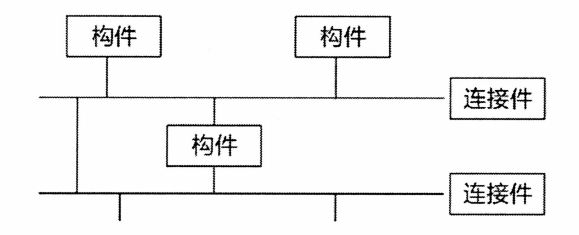
#### 软件架构风格 – 闭环控制架构（过程控制）

当软件被用来操作一个物理系统时，软件与硬件之间可以粗略地表示为一个反馈循环，这个反馈循环通过接受一定的输入，确定一系列的输出，最终使环境达到一个新的状态。适合于嵌入式系统，涉及连续的动作和状态。



#### 软件架构风格 – C2风格

C2风格是一种并行构件网络。如下图所示：



C2构件的基本规则：

构件和连接件都有一个顶部和底部

构件的顶部要连接到连接件的底部，构件的顶部要连接到连接件的顶部，构件之间不允许直连

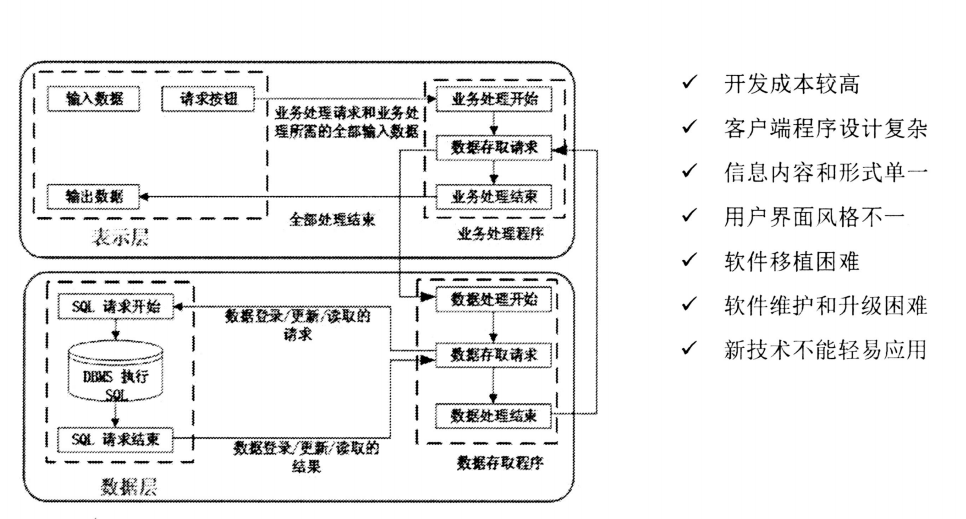
一个连接件可以和任意数目的其他构件和连接件连接

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

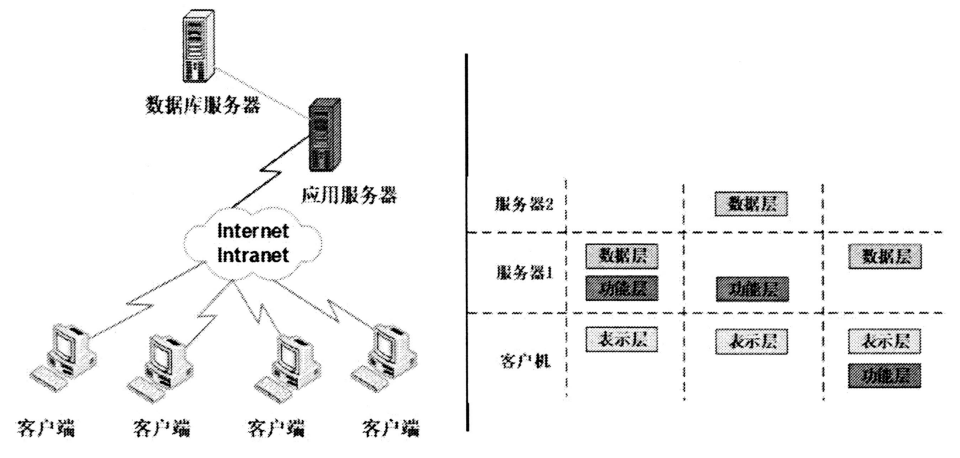
#### 软件架构风格 – 层次架构风格

两层C/S —》三层C/S —》三层B/S —》混合架构

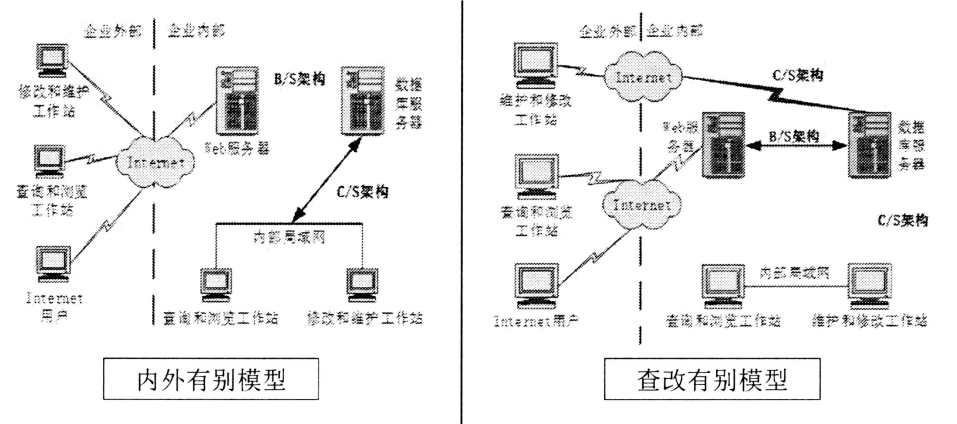
#### 软件架构风格 – 两层C/S架构



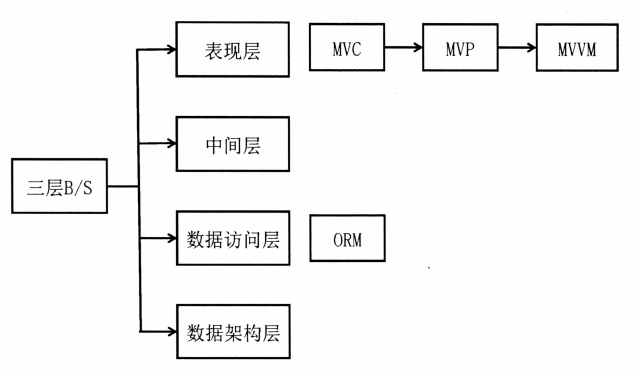
#### 软件架构风格 – 三层C/S架构、三层B/S架构



#### 软件架构风格 – 混合架构风格



#### 软件架构风格 – 层次架构风格



#### 软件架构风格 – MVC架构风格

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。通常模式对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

J2EE体系结构中：视图 JSP，控制 Servlet，模型 Entity Bean、Session Bean

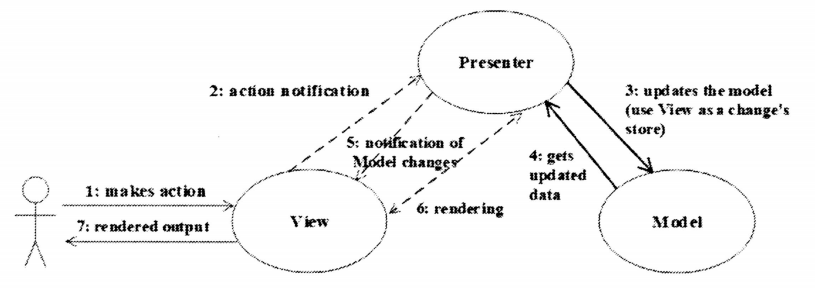
#### 软件架构风格 – MVP架构风格

MVP是MVC的变种

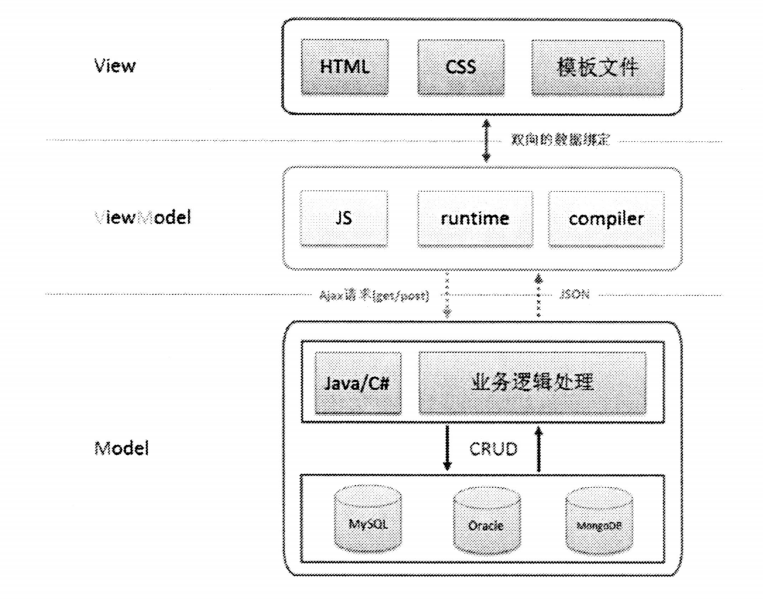
MVP实现了V与M之间的解耦（V不直接使用M，修改V不会影响M）

MVP更好的支持单元测试（业务逻辑在P中，可以脱离V来测试这些逻辑，可以将一个P用于多个V，而不需要修改P的逻辑）

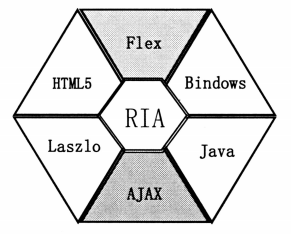
MVP中V要处理界面事件，业务逻辑在P中，MVC中界面事件有C处理。



#### 软件架构风格 – MVVM架构风格



#### 软件架构风格 – 富互联网应用（RIA）



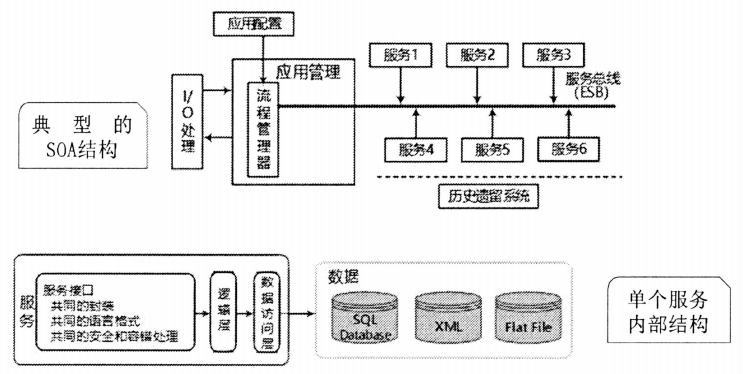
RIA结合了C/S架构反应速度快、交互性强的优点，以及B/S架构传播范围广及容易传播的特性

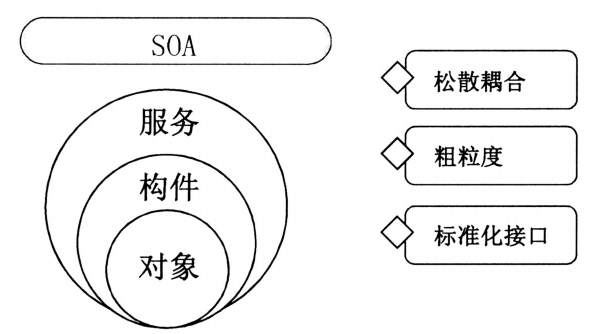
RIA简化并改进了B/S架构的用户交互

数据能够被缓存在客户端，从而可以实现一个比基于HTML的响应速度更快且数据往返于服务器的次数更少的用户界面。

#### 软件架构风格 – 基于服务的架构（SOA）

服务是一种为了满足某项业务需求的操作、规则等的逻辑组合，它包含一系列的有序活动的交互，为实现用户目标提供支持。





服务构件粗粒度，传统构件细粒度居多

服务构件的接口是标准的，主要是WSDL接口，传统构件常以具体API形式出现

服务构件的实现与语言无关，传统构件绑定某种特定语言

服务构件可以通过构件容器提供Qos的服务，传统构件完全有程序代码直接控制

**SOA的实现方式 – Web Service**

底层传输层

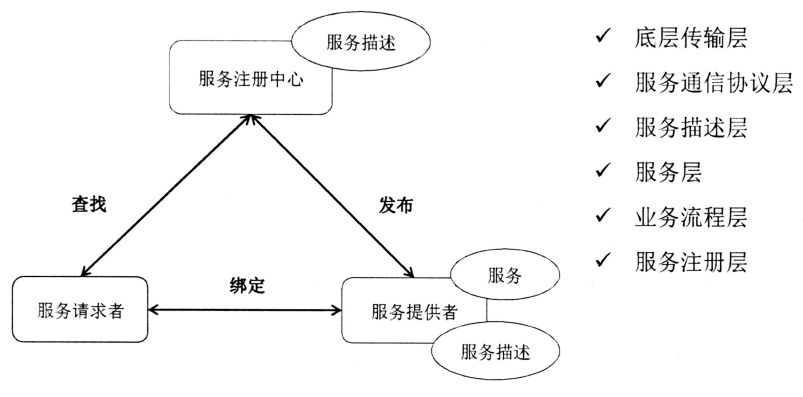
服务通信协议层

服务描述层

服务层

业务流程层

服务注册层



**SOA的实现方式 – ESB**

提供位置透明性的消息路由和寻址服务

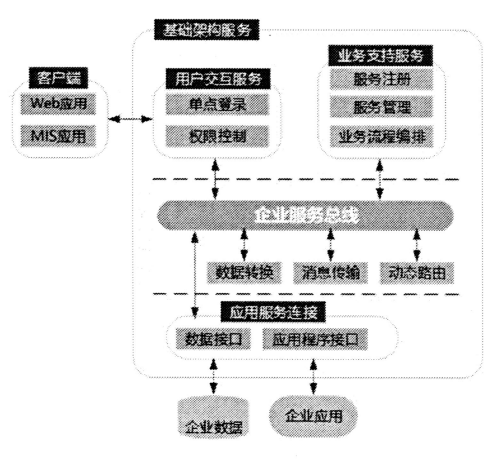
提供服务注册和命名的管理功能

支持多种信息的传递范式

支持多种可以广泛使用的传输协议

支持多种数据格式及其相互转换

提供日志和监控功能



**SOA的实现方式 – 服务注册表**

服务注册：应用开发者（服务提供者）想注册表公布服务的功能

服务位置：服务使用者（服务应用开发者），帮助他们查询注册服务，寻找符合自身要求的服务

服务绑定：服务使用者利用检索到的额服务接口来编写代码，所编写的代码将与注册的服务绑定、调用注册的服务，以及与他们实现互动

**SOA-关键技术**

**功能 协议**

发现服务 UDDI、DISCO

描述服务 WSDL、XML、Schema

消息格式层 SOAP、REST

编码格式层 XML（DOM、SAX）

传输协议层 HTTP、TCP/IP、SMTP

WSDL就是WebService接口对应的WSDL文件，该文件通过XML格式说明如何调用，可以看做WebService的接口文档（使用说明书）

SOAP，Simple Object Access Protocol，简单对象访问协议，用于访问网络服务的协议，它是基于XML的简单协议，可使应用程序在HTTP之上进行信息交换

SOAP是一种网络通信协议，用于网络上、不同平台、不同语言的应用程序间的通信

SOAP协议 = HTTP协议 + XML数据格式

**SOA-关键技术-REST**

HTTP + XML进行基于Web通信的技术

简单性，缺少严格配置文件

只支持几个操作POST、GET、PUT、DELETE

强调信息本身，称为资源

资源：

网络上所有的事物都被抽象为资源

每个资源对应一个唯一的资源标识

通过通用的连接器接口对资源进行操作

对资源的各种操作不会改变资源标识

所有的操作都是无状态的

#### 软件架构风格 – 微服务

微服务顾名思义，就是很小的服务，所以他输入面向架构的一种

#### 软件架构风格 – 微服务与SOA

|  |  |
| --- | --- |
| **微服务** | **SOA** |
| 能拆分的就拆分 | 是整体的，服务能放在一起的都放在一起 |
| 纵向业务拆分 | 是水平多层 |
| 有单一组织负责 | 按层级划分不同部门的组织负责 |
| 细粒度 | 粗粒度 |
| 两句话能解释明白 | 几百字相当于SOA的目录 |
| 独立的子公司 | 类似大公司里面划分了一些业务单元（BU） |
| 组件小 | 存在较复杂的组件 |
| 业务逻辑存在于每个服务中 | 业务逻辑横跨多个业务领域 |
| 使用轻量级的通信方式，如HTTP | 企业服务生产总线（ESB）充当了服务之间通信的角色 |

|  |  |
| --- | --- |
| **微服务架构实现** | **SOA实现** |
| 团队级，自底向上开展实施 | 企业级，自顶向下开展实施 |
| 一个系统被拆分成多个服务，粒度细 | 服务有多个子系统组成，粒度大 |
| 无集中式总线，松散的服务架构 | 企业服务总线，集中式的服务架构 |
| 集成方式简单（HTTP、REST、JSON） | 集成方式复杂（ESB、WS、SOAP） |
| 服务能独立部署 | 单块框架系统，相互依赖，部署复杂 |

#### 软件架构风格 – MDA

模型驱动架构(MDA，Model Driven Architecture)。MDA是一种基于诸如统一建模语言(UML)、可扩展标记语言(XML)，和公共对象请求代理体系结构(CORBA)等一系列业界开放标准的框架，因此，它具备软件设计和模型的可视化、存储和交换的功能。

MDA能够创建出机器可读和高度抽象的模型，这些模型独立于实现技术，以标准化的方式储存。

MDA使用模型完成软件的分析、设计、构建、部署、维护等跟踪开发活动。

MDA的主要目标：可移植性、互通性、可重用性

MDA的3种核心模型：

平台独立模型（PIM）：具有高抽象层次、独立于任何实现技术的模型

平台相关模型（PSM）：为某种特定实现技术量身定做，让你用这种技术中可用的是实现构造来描述系统的模型。PIM会被换成一个或者多个PSM。

代码Code：用源代码对系统的描述（规约）。每个PSM都将被变换成代码

