**MVC和Kruchten 4+1视图**

**MVC**

经典MVC模式中，M是指模型，V是视图，C则是控制器，使用MVC的目的是将M和V的实现代码分离，从而使同一个程序可以使用不同的表现形式。其中，View的定义比较清晰，就是[用户界面](https://baike.baidu.com/item/用户界面/6582461?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)。

**编程模式**

M即model模型是指[模型表示](https://baike.baidu.com/item/模型表示/53432073?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)[业务规则](https://baike.baidu.com/item/业务规则/10826802?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)。在MVC的三个部件中，模型拥有最多的处理任务。被模型返回的数据是中立的，模型与[数据格式](https://baike.baidu.com/item/数据格式/5198733?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)无关，这样一个模型能为多个视图提供数据，由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的[重复性](https://baike.baidu.com/item/重复性/9226009?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)。

V即View视图是指用户看到并与之交互的界面。比如由html元素组成的网页界面，或者软件的客户端界面。MVC的好处之一在于它能为应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，它只是作为一种输出数据并允许用户操作的方式。

C即controller控制器是指控制器接受用户的输入并调用模型和视图去完成用户的需求，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

**框架内容**

MVC指MVC模式的某种框架，它强制性地使应用程序的输入、处理和输出分开。使用MVC应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务。最典型的MVC就是JSP + servlet + [javabean](https://baike.baidu.com/item/javabean/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)的模式。

**视图**

视图是用户看到并与之交互的界面。对老式的Web应用程序来说，视图就是由[HTML](https://baike.baidu.com/item/HTML/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)元素组成的界面，在新式的Web应用程序中，[HTML](https://baike.baidu.com/item/HTML/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)依旧在视图中扮演着重要的角色，但一些新的技术已层出不穷，它们包括[Adobe Flash](https://baike.baidu.com/item/Adobe Flash/7172106?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)和像[XHTML](https://baike.baidu.com/item/XHTML/316621?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)，XML/XSL,[WML](https://baike.baidu.com/item/WML/2891370?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)等一些标识语言和Web services.

MVC好处是它能为应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

**模型**

模型表示[企业数据](https://baike.baidu.com/item/企业数据/5514237?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)和业务规则。在MVC的三个部件中，模型拥有最多的处理任务。例如它可能用像[EJB](https://baike.baidu.com/item/EJB/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)s和ColdFusion Components这样的构件对象来处理数据库，被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据，由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的[重复性](https://baike.baidu.com/item/重复性/9226009?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)。

**控制器**

控制器接受用户的输入并调用模型和视图去完成用户的需求，所以当单击Web页面中的[超链接](https://baike.baidu.com/item/超链接/97857?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)和发送[HTML表单](https://baike.baidu.com/item/HTML表单/11046337?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

**Kruchten 4+1视图**

“4+1”视图是对逻辑架构进行描述，最早由 Philippe Kruchten 提出，他在1995年的《IEEE Software》上发表了题为《The 4+1 View Model of Architecture》的论文，引起了业界的极大关注，并最终被 [RUP](https://baike.baidu.com/item/ RUP/8924595?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank) 采纳，现在已经成为[架构设计](https://baike.baidu.com/item/架构设计/3603366?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)的结构标准。

该模型五个主要的视图

逻辑视图（Logical View），设计的对象模型（使用[面向对象](https://baike.baidu.com/item/面向对象/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)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](https://baike.baidu.com/item/开发环境/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)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，从而形成了第五个视图。

**逻辑架构**

面向对象的分解

逻辑架构主要支持功能性需求――系统应该为用户提供哪些服务。系统分解为一系列的关键抽象，（大多数）来自于问题域，表现为对象或对象类的形式。它们采用抽象、封装和继承的原理。分解并不仅仅是为了功能分析，而且用来识别遍布系统各个部分的通用机制和设计元素。我们使用 Rational/Booch 方法来表示逻辑架构，借助于类图和[类模板](https://baike.baidu.com/item/类模板/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)的手段。类图用来显示一个类的集合和它们的逻辑关系：关联、使用、组合、继承等等。相似的类可以划分成类集合。[类模板](https://baike.baidu.com/item/类模板/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)关注于单个类，它们强调主要的类操作，并且识别关键的对象特征。如果需要定义对象的内部行为，则使用[状态转换图](https://baike.baidu.com/item/状态转换图/4299880?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)或[状态图](https://baike.baidu.com/item/状态图/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)来完成。公共机制或服务可以在类功能 （class utilities）中定义。对于数据驱动程度高的应用程序，可以使用其他形式的逻辑视图，例如 E-R 图，来代替面向对象的方法（OO approach）。

**过程架构**

过程分解

过程架构考虑一些非功能性的需求，如性能和可用性。它解决并发性、分布性、系统完整性、[容错性](https://baike.baidu.com/item/容错性/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)的问题，以及逻辑视图的主要抽象如何与过程结构相配合在一起－即在哪个控制线程上，对象的操作被实际执行。

过程架构可以在几种层次的抽象上进行描述，每个层次针对不同的问题。在最高的层次上，过程架构可以视为一组独立执行的通信程序（叫作“processes”）的[逻辑网络](https://baike.baidu.com/item/逻辑网络/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)，它们分布在整个一组硬件资源上，这些资源通过 LAN 或者 WAN 连接起来。多个[逻辑网络](https://baike.baidu.com/item/逻辑网络/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)可能同时并存，共享相同的物理资源。例如，独立的[逻辑网络](https://baike.baidu.com/item/逻辑网络/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)可能用于支持离线系统与在线系统的分离，或者支持软件的模拟版本和测试版本的共存。

过程是构成可执行单元任务的分组。过程代表了可以进行策略控制过程架构的层次（即：开始、恢复、重新配置及关闭）。另外，过程可以就处理负载的分布式增强或可用性的提高而不断地被重复。

软件被划分为一系列单独的任务。任务是独立的控制线程，可以在处理节点上单独地被调度。

接着，我们可以区别主要任务、次要任务。主要任务是可以唯一处理的架构元素；次要任务是由于实施原因而引入的局部附加任务（周期性活动、缓冲、暂停等等）。它们可以作为 Ada Task 或轻量线程来实施。主要任务的通讯途径是良好定义的交互任务通信机制：基于消息的同步或异步通信服务、[远程过程调用](https://baike.baidu.com/item/远程过程调用/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)、事件广播等。次要任务则以会见或[共享内存](https://baike.baidu.com/item/共享内存/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)来通信。在同一过程或处理节点上，主要任务不应对它们的分配做出任何假定。

消息流、过程负载可以基于过程视图来进行评估，同样可以使用哑负载来实现“中空”的过程架构，并测量在目标系统上的性能。正如 Filarey et al. 在他的 Eurocontrol 实验中描述的那样。

**开发架构**

子系统分解

开发架构关注[软件开发环境](https://baike.baidu.com/item/软件开发环境/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)下实际模块的组织。软件打包成小的程序块（[程序库](https://baike.baidu.com/item/程序库/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)或子系统），它们可以由一位或几位开发人员来开发。子系统可以组织成分层结构，每个层为上一层提供良好定义的接口。

系统的开发架构用模块和子系统图来表达，显示了“输出”和“输入”关系。完整的开发架构只有当所有软件元素被识别后才能加以描述。但是，可以列出控制开发架构的规则：分块、分组和可见性。

大部分情况下，开发架构考虑的内部需求与以下几项因素有关：开发难度、软件管理、重用性和通用性及由工具集、[编程语言](https://baike.baidu.com/item/编程语言/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)所带来的限制。开发架构视图是各种活动的基础，如：需求分配、团队工作的分配（或团队机构）、成本评估和计划、项目进度的监控、[软件重用](https://baike.baidu.com/item/软件重用/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)性、移植性和安全性。它是建立产品线的基础。

**物理架构**

软件至硬件的映射

物理视图主要描述硬件配置。在UML中通常被称为部署视图，它主要考虑如何把软件映射到硬件上。通常需要考虑到解决系统拓扑结构、系统安装和通信等问题。我们希望使用不同的物理配置：一些用于开发和测试，另外一些则用于不同地点和不同客户的部署。因此软件至节点的映射需要高度的灵活性及对[源代码](https://baike.baidu.com/item/源代码/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)产生最小的影响。

**场景**

综合所有的视图

四种视图的元素通过数量比较少的一组重要场景（更常见的是用例）进行无缝[协同工作](https://baike.baidu.com/item/协同工作/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)，我们为场景描述相应的脚本（对象之间和过程之间的交互序列）。

在某种意义上场景是最重要的需求抽象，它们的设计使用对象场景图和对象交互图来表示。

该视图是其他视图的冗余，但它起到了两个作用：

作为一项驱动因素来发现[架构设计](https://baike.baidu.com/item/架构设计/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)过程中的架构元素。

作为[架构设计](https://baike.baidu.com/item/架构设计/0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%224+1%22%E8%A7%86%E5%9B%BE/_blank)结束后的一项验证和说明功能，既以视图的角度来说明又作为架构原型测试的出发点。