1.各种软件体系结构风格

**数据流风格**：

当输入数据经过一系列的计算和操作构件的变换形成输出数据时，可以应用这种体系结构。批处理序列; 管道/过滤器都属于数据流风格。

特点：数据流风格将系统视为一组数据流和转换器，其中数据流沿着转换器流动，转换器执行一些处理操作。这种风格侧重于数据的流动和转换，强调数据的处理过程。

视图特点：数据流图是数据流风格的主要视图，它显示了数据流的路径和转换器之间的关系。图中的箭头表示数据流的方向，节点表示数据转换器。

**调用/返回风格：**

特点：调用/返回风格将系统分解为一组相互独立的模块，每个模块都有自己的接口和功能。模块之间通过调用和返回来进行通信和协作。

视图特点：模块图是调用/返回风格的主要视图，它显示了模块之间的调用关系和依赖关系。通常使用**层次结构**来组织模块，顶层模块调用底层模块来实现功能。层次结构：整个系统被组织成一个分层结构，每一层为上层提供服务，并作为下一层的客户。

**独立构件风格**：

特点：独立构件风格将系统分解为一组相互独立的组件，每个组件都有自己的状态和行为，并且可以独立地部署和升级。

视图特点：组件图是独立构件风格的主要视图，它显示了组件之间的关系和依赖关系。通常使用接口和端口来定义组件之间的通信。

**仓库风格**：

特点：仓库风格将系统建模为一个数据仓库或中央数据库，所有的数据都存储在一个统一的位置，并且可以通过查询和更新来访问。

视图特点：数据模型图是仓库风格的主要视图，它显示了数据的结构和关系。通常使用实体-关系图来表示数据模型，其中实体表示数据对象，关系表示实体之间的关联。

**过程控制环路：**

特点：过程控制环路将系统建模为一组相互依赖的进程或任务，这些进程通过消息传递或共享状态来进行通信和协作。

视图特点：进程图是过程控制环路的主要视图，它显示了进程之间的调用关系和数据流动关系。通常使用有向图来表示进程之间的依赖关系。

**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。界面和操作可以很丰富，安全性高，响应速度快；但适用面窄、用户群固定、维护成本高

特点：客户端/服务器风格将系统分为客户端和服务器端两部分，客户端负责用户界面和用户交互，服务器端负责业务逻辑和数据处理。

视图特点：架构图是C/S风格的主要视图，它显示了客户端和服务器端之间的通信和协作关系。通常使用分层结构来组织客户端和服务器端的功能。

**B/S风格：**

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。B/S体系结构有三个主要组成部分：浏览器、 Web服务器、数据库服务器。

特点：浏览器/服务器风格将系统建模为一个Web应用程序，用户通过浏览器访问应用程序的前端界面，服务器端提供服务和处理业务逻辑。

视图特点：架构图是B/S风格的主要视图，它显示了浏览器和服务器之间的通信和交互关系。通常使用客户端和服务器端来表示浏览器和服务器之间的功能划分。

我们的网上书店项目可以归类为B/S风格（浏览器/服务器风格）。在这种架构中，用户通过浏览器访问网站的前端界面，而后端业务逻辑和数据处理则由服务器端提供和处理。用户与系统的交互主要通过HTTP请求和响应来完成，前端页面通过异步请求向后端发送数据请求，后端处理后返回相应的数据或页面。这种架构适用于Web应用程序的开发，能够实现跨平台访问，并且具有灵活性和可扩展性。

2.MVC和Kruchten 4+1

**MVC：**

MVC是一种常见的软件架构模式，用于构建用户界面应用程序。

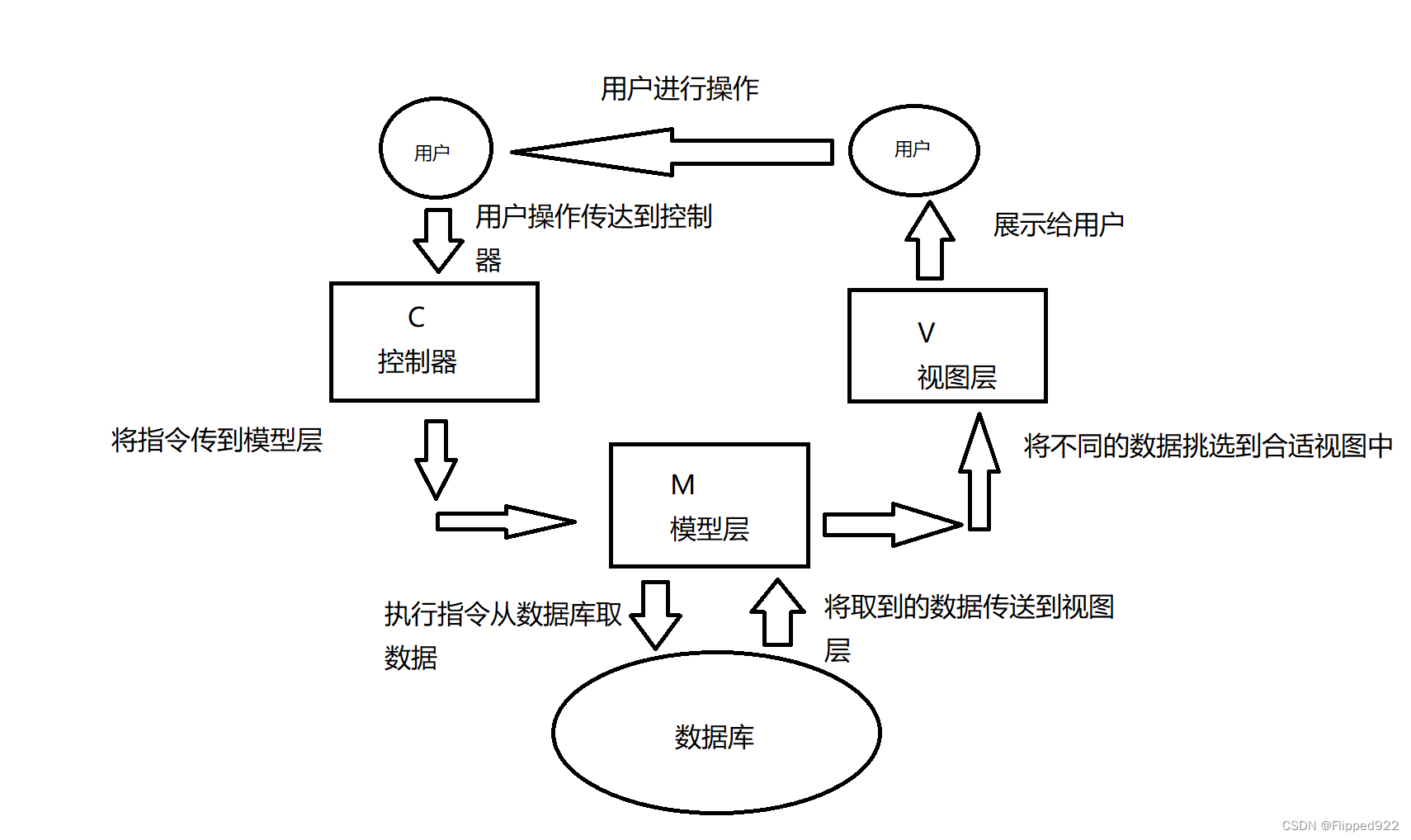
在MVC中，应用程序被分为三个核心组件：

Model（模型）：表示应用程序的数据和业务逻辑，负责处理数据的存储、检索和修改。

View（视图）：表示用户界面的展示层，负责呈现模型数据并与用户交互。

Controller（控制器）：作为模型和视图之间的中介，负责接受用户输入、调用相应的模型逻辑，并更新视图以反映模型的变化。

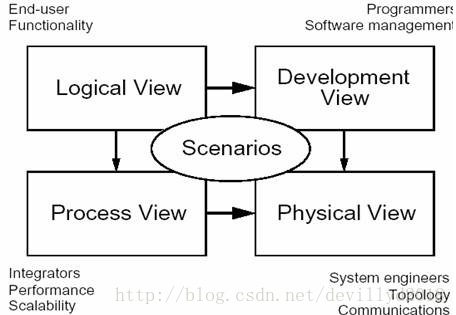
执行流程为由终端的客户端进行操作，控制器基于用户的操作将请求传达给M层，M层接收到请求去执行关于数据库的动作，从而调用数据库的数据展示到V层，也就是前端的页面展示出来。



MVC模式的分离关注点使得开发人员能够更好地组织和维护代码，提高了应用程序的可维护性和可扩展性。MVC架构简化了开发的流程，提供了一种思想使开发和维护用户接口的技术含量降低。使开发时间得到相当大的缩减，它使程序员（Java开发人员）集中精力于业务逻辑，界面程序员（和JSP开发人员）集中精力于表现形式上。但是使用MVC也会使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

**Kruchten 4+1视图：**

Kruchten的4+1视图是一种用于描述软件体系结构的模型，它包括了四个视图和一个场景视图。每一个视图只关心系统的一个侧面，5 个试图结合在一起才能反映系统的软件体系结构的全部内容。



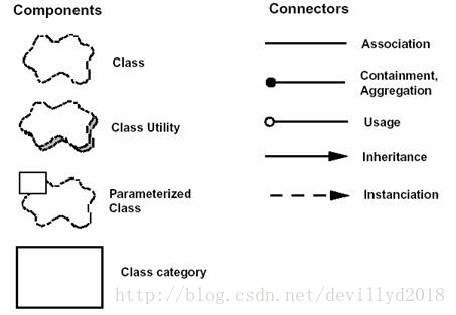
四个视图分别是：

**逻辑视图（Logical View）**：主要是用来 描述系统的功能需求，即系统提供给最终用户的服务. 在逻辑视图中，系统分解成一系列的功能抽象、功能分解与功能分析，这些主要来自问题领域（Problem Definition)。

在面向对象技术中，通过抽象、封装、继承,可以用对象模型来代表逻辑视图，可以用类图（Class Diagram）来描述逻辑视图。如下图:

构件(Components)：类、类服务、参数化类、类层次

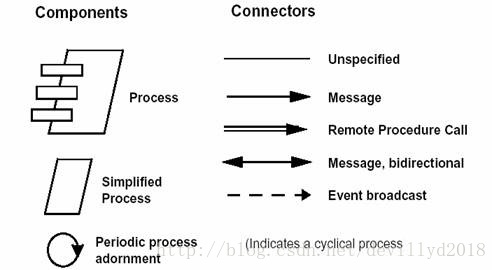
连接件(Connectors)：关联、包含聚集、使用、继承、实例化



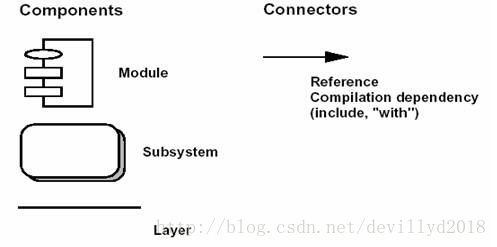
**进程视图（Process View）**：描述了软件系统的并发和通信机制，通常使用流程图、时序图等来表示。侧重系统的 运行特性，关注非功能性的需求（性能，可用性）。服务于系统集成人员，方便后续性能测试。强调并发性、分布性、集成性、鲁棒性（容错）、可扩充性、吞吐量等。定义逻辑视图中的各个类的具体操作是在哪一个线程（Thread）中被执行。

如下图: 构件：进程、简化进程、循环进程

连接件：未指定，消息、远程过程调用（RPC）、双向消息、事件广播



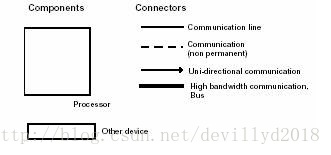
**开发视图（Development View）**：主要用来 描述软件模块的组织与管理（通过程序库或子系统）。服务于软件编程人员， 方便后续的设计与实现。它通过系统输入输出关系的模型图和子系统图来描述。要考虑软件的内部需求：开发的难易程度、重用的可能性，通用性，局限性等等。开发视图的风格通常是层次结构，层次越低，通用性越好（底层库：Java SDK，图像处理软件包）。如下图: 构件：模块、子系统、层 连接件：参照相关性、模块/过程调用



物理视图（Physical View）：主要 描述硬件配置。服务于 系统工程人员，解决系统的拓扑结构、系统安装、通信等问题。主要考虑如何把软件映射到硬件上，也要考虑系统性能、规模、可靠性等。可以与进程视图一起映射。如下图:

构件：处理器、计算机、其它设备

连接件：通信协议等



场景视图描述了软件系统的各种用例场景和交互情况，通常使用用例图、时序图等来表示。用于刻画构件之间的相互关系，将四个视图有机地联系起来。可以描述一个特定的视图内的构件关系，也可以描述不同视图间的构件关系。文本、图形表示皆可。