**Week10-2:**

经典的软件体系结构风格主要有以下几种：

1)数据流风格: 批处理序列; **管道/过滤器**。

应用场景：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。

优点：a.设计者可以将整个系统的输入、输出特性简单的理解为各个过滤器功能的合成。 b.具有较强的可维护性和可扩展性c.支持一些特定的分析，如吞吐量计算和死锁检测等d.数据流风格具有并发性

缺点：a.交互式处理能力弱b.具体实现比较复杂：数据流同步问题；数据加密与解析。

2)调用/返回风格：主程序/子程序；面向对象风格；**层次结构**。

优点：a.支持系统设计过程中的逐级抽象 b.具有较好的可扩展性 c.支持软件复用

缺点：a.并不是每个系统都可以很容易地划分为分层的模式b.很难找到一个合适的、正确的层次抽象方法。

3)独立构件风格：进程通讯；事件系统。

4)虚拟机风格：解释器；基于规则的系统。

5)仓库风格：数据库系统；超文本系统；黑板系统。

6)过程控制环路

7)C/S风格

基本概念：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。C/S体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络。

优点： a.界面和操作可以很丰富b.安全性高c.响应速度快

缺点： a.适用面窄b.用户群固定c.维护成本高

8)B/S风格

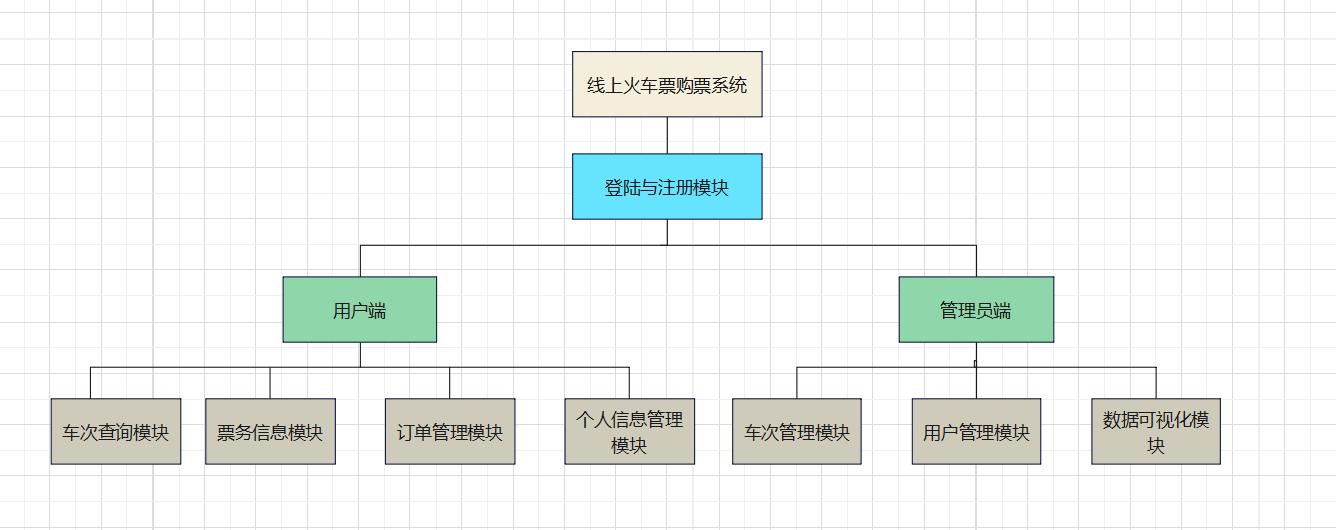
基本概念：B/S体系结构主要利用不断成熟的WWW浏览器技术，特别是浏览器嵌入的多种脚本语言。B/S体系结构有三个主要组成部分：浏览器、 Web服务器、数据库服务器。

优点：a.维护和升级方式简单b.交互性较强

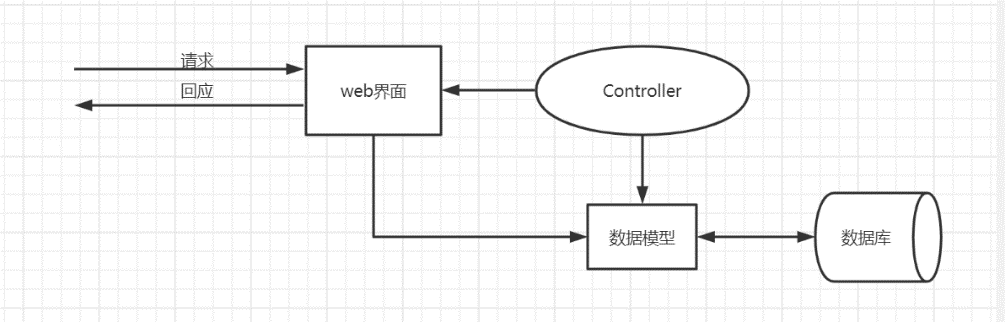
缺点：a.在速度和安全性上需要花费巨大的设计成本b.通常需要刷新页面（Ajax等可以一定程度缓解该问题）c.通信开销大

常见的体系结构视图有以下几种：

1.分解视图: 分解视图用软件模块勾划出系统结构，通过不同抽象层级的软件模块形成层次化的结构，呈现为较为明晰的分解结构。

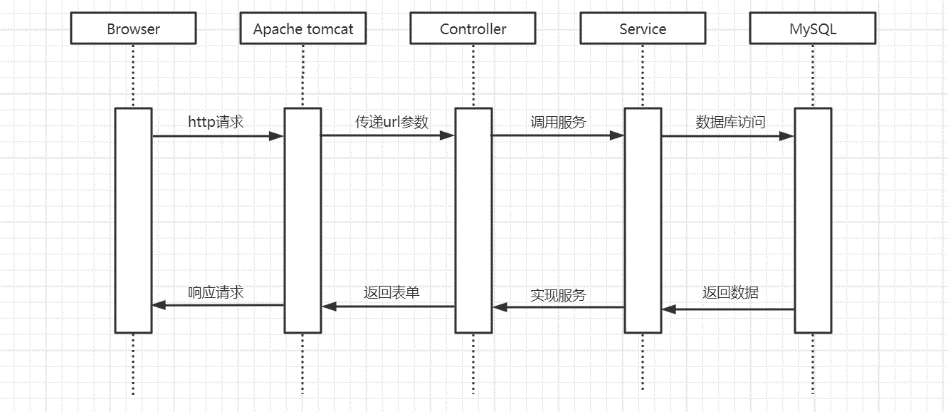


2.依赖视图: 依赖视图展现了软件模块之间的依赖关系，数据模型封装数据和功能，web界面显示信息，获得来自模型的数据，决定模型展示给用户的方式，依赖于数据模型。控制器作用于视图和模型上，空子数据流向模型对象，并在数据变化时更新视图，依赖于视图和模型。



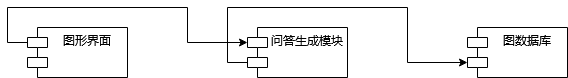
3.泛化视图: 泛化视图展现了软件模块之间的一般化或具体化的关系，典型的例子就是面向对象分析和设计方法中类之间的继承关系。值得注意的是，采用对象组合替代继承关系，并不会改变类之间的泛化特征。因此泛化是指软件模块之间的一般化或具体化的关系，不能局限于继承概念的应用。泛化视图有助于描述软件的抽象层次，从而便于软件的扩展和维护。比如通过对象组合或继承很容易形成新的软件模块与原有的软件架构兼容。

4.执行视图: 执行视图展示了系统运行时的时序结构特点，比如流程图、时序图等。执行视图中的每一个执行实体，一般称为组件，都是不同于其他组件的执行实体。如果有相同或相似的执行实体那么就把它们合并成一个。执行实体可以最终分解到软件的基本元素和软件的基本结构，因而与软件代码具有比较直接的映射关系。在设计与实现过程中，我们一般将执行视图转换为伪代码之后，再进一步转换为实现代码。

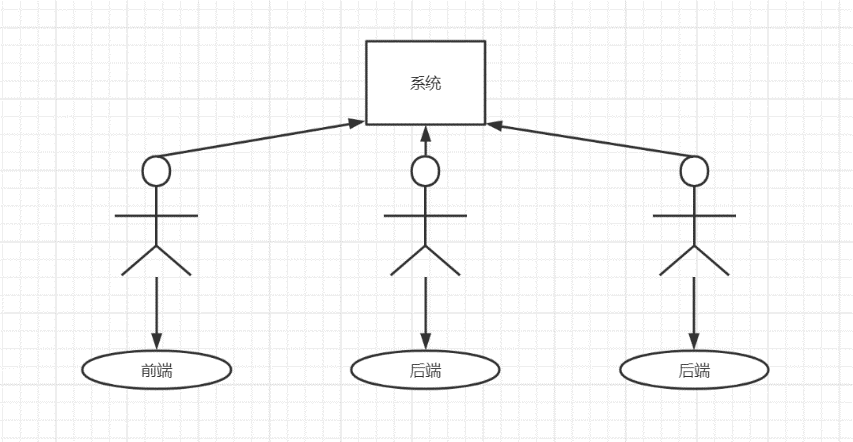


5.实现视图: 实现视图是描述软件架构与源文件之间的映射关系。通过目录和源文件的命名来对应软件架构中的包、类等静态结构单元，这样实现视图就可以由软件项目的源文件目录树来呈现。

6.部署视图: 部署视图是将执行实体和计算机资源建立映射关系。这里的执行实体的粒度要与所部署的计算机资源相匹配，比如以进程作为执行实体那么对应的计算机资源就是主机，这时应该描述进程对应主机所组成的网络拓扑结构，这样可以清晰地呈现进程间的网络通信和部署环境的网络结构特点。当然也可以用细粒度的执行实体对应处理器、存储器等。部署视图有助于设计人员分析一个设计的质量属性，比如软件处理网络高并发的能力、软件对处理器的计算需求等。



7.工作分配视图: 工作分配视图将系统分解成可独立完成的工作任务，以便分配给各项目团队和成员。工作分配视图有利于跟踪不同项目团队和成员的工作任务的进度，也有利于在个项目团队和成员之间合理地分配和调整项目资源。

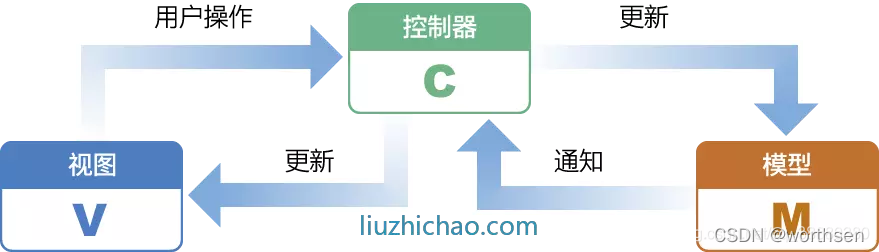


通过分析我们自己的项目，我们发现火车售票系统的设计风格属于**C/S风格。**由客户端应用程序、网络和数据库服务器三部分组成。

\*\*最新的软件体系结构资料：

(1)MVC体系结构：

MVC（视图/模型结构）把数据和视图组件分离，这使得我们可以在几个不同的试图组件中显示相同的数据，并且实现新类型的视图，并且不改变底层的数据结构。也可以根据控制响应的不同命令，在同一个组件区创建不同的视图样式，来加载不同的输入数据。为了更灵活的对用户输入进行处理，引入了delegate这个概念。它的好处是，数据项的渲染与编程可以进行定制。

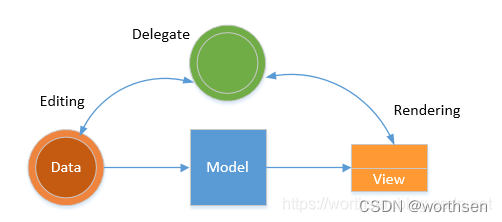


Model：model管理数据，负责数据的请求和更新

View：向用户展示数据

Controller：处理Model和View之间的交互。负责为View提供正确的数据，为Model提供来自View的操作请求。

该模式降低耦合，多个View可以显示同一数据，View层不用担心数据层Model发生变化。



(2)Kruchten 4+1视图：

Kruchten 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/%20RUP/8924595?fromModule=lemma_inlink" \t "_blank) 采纳，现在已经成为[架构设计](https://baike.baidu.com/item/%E6%9E%B6%E6%9E%84%E8%AE%BE%E8%AE%A1/3603366?fromModule=lemma_inlink" \t "_blank)的结构标准。该模型五个主要的视图:

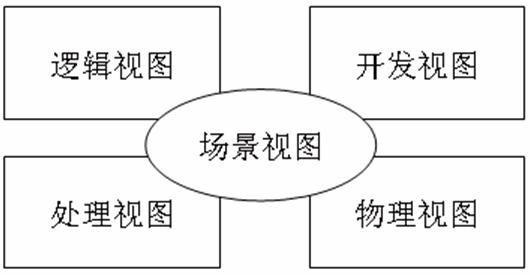
逻辑视图（Logical View），设计的对象模型（使用[面向对象](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1?fromModule=lemma_inlink" \t "_blank)的设计方法时）。

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/%E5%BC%80%E5%8F%91%E7%8E%AF%E5%A2%83?fromModule=lemma_inlink" \t "_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。



运用4+1视图方法：针对不同需求进行架构设计

如前文所述，要开发出用户满意的软件并不是件容易的事，软件架构师必须全面把握各种各样的需求、权衡需求之间有可能的矛盾之处，分门别类地将不同需求一一满足。Philippe Kruchten提出的4+1视图方法为软件架构师"一一征服需求"提供了良好基础，如下图所示。

