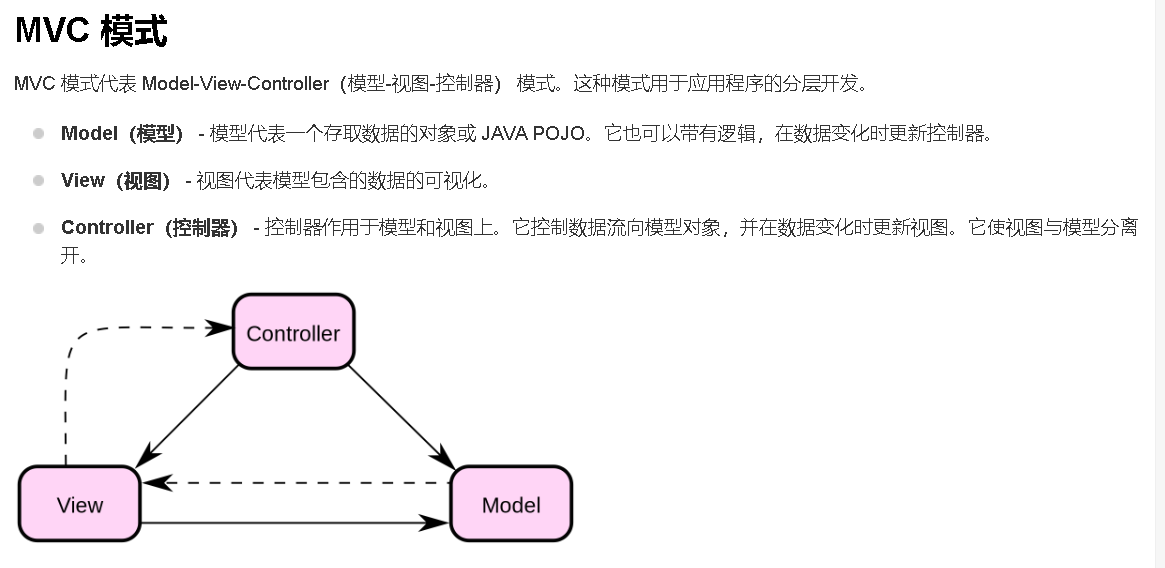
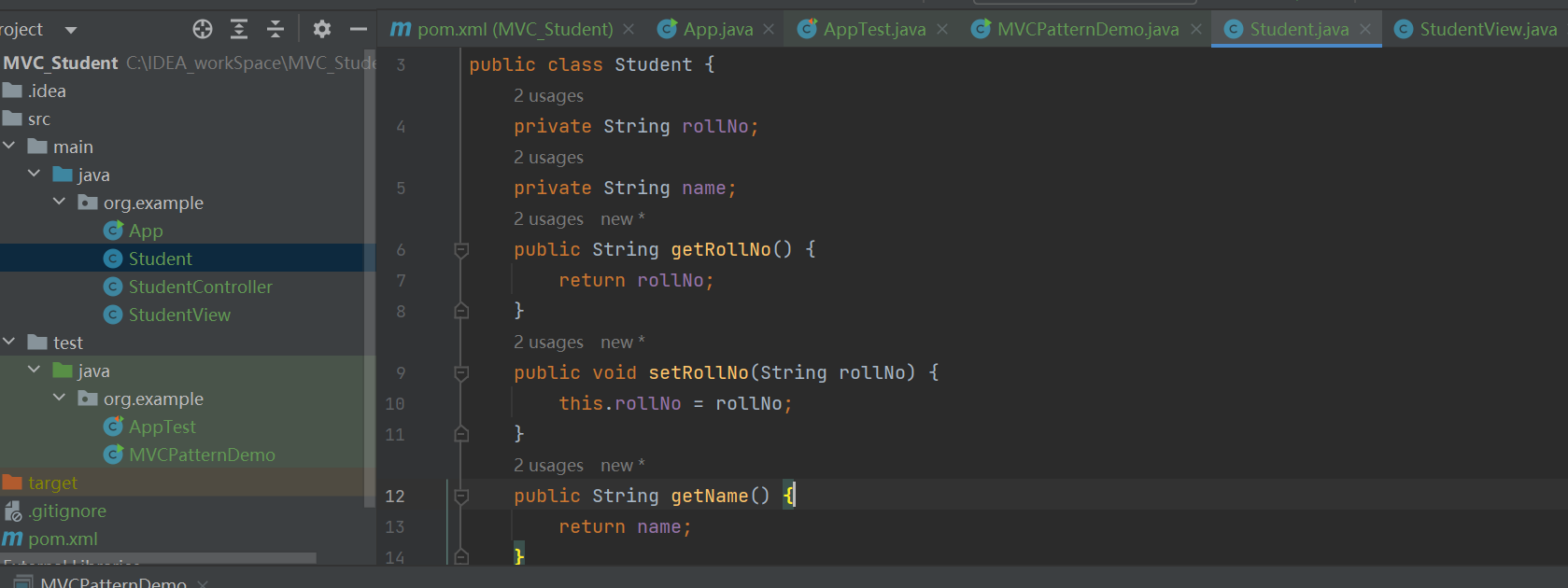
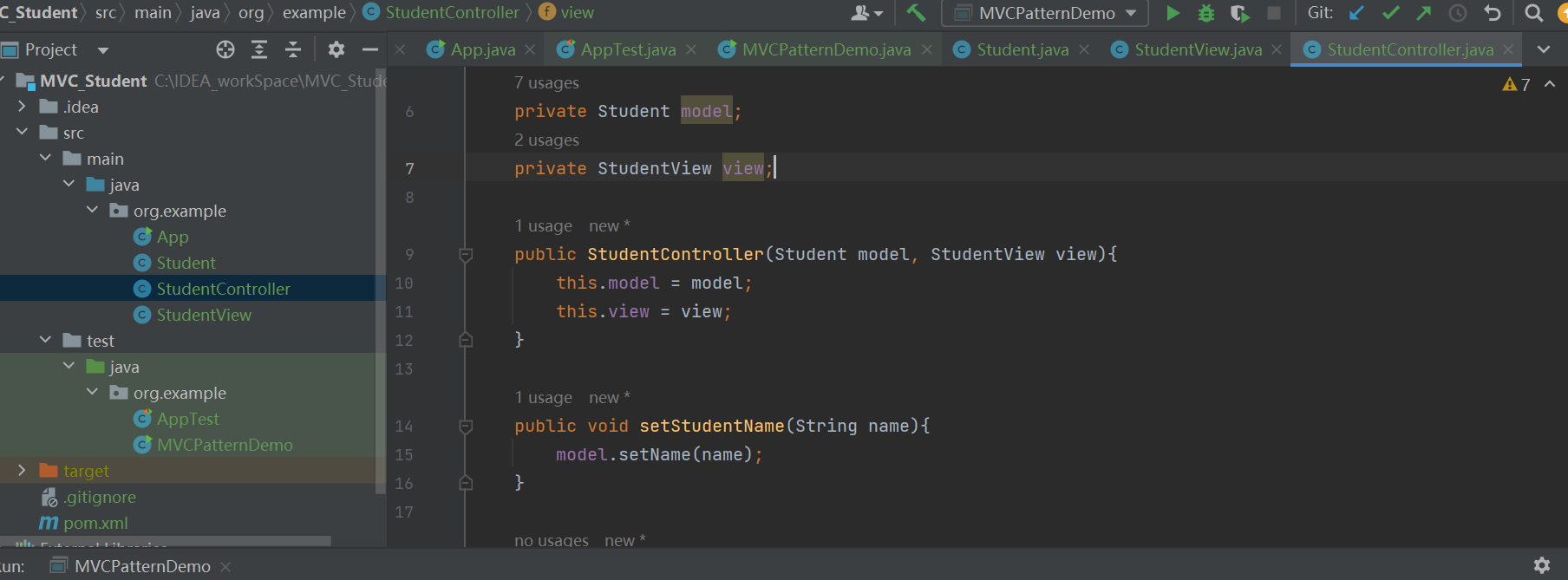
MVC(Model-View\_Controller)

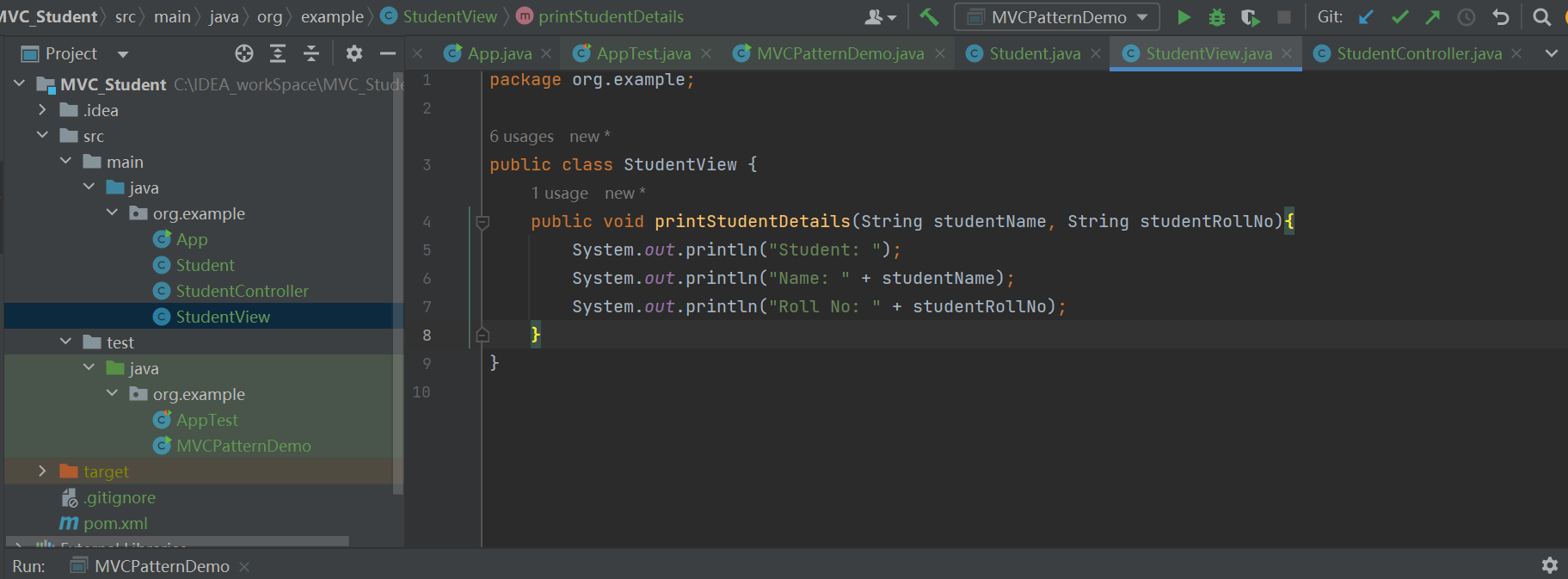
1. MVC模式



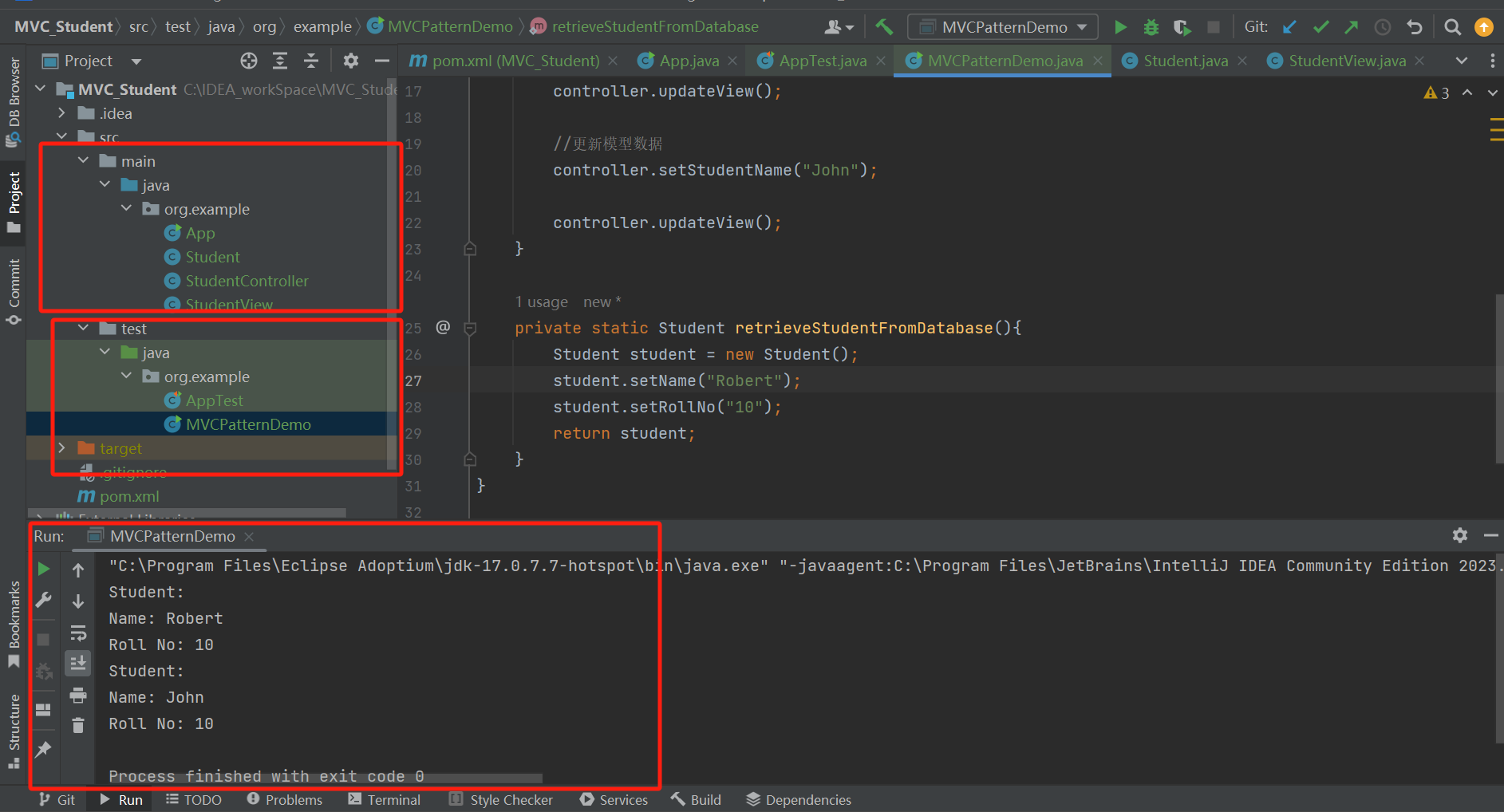
1. MVC\_Student\_Implementation
   1. Class(Student & StudentView & StudentController)







2.2. Output



Done :D

1. MPC模式
   1. What？

MVC（Model–View–Controller）模式是软件工程中的一种软件架构模式，它把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。

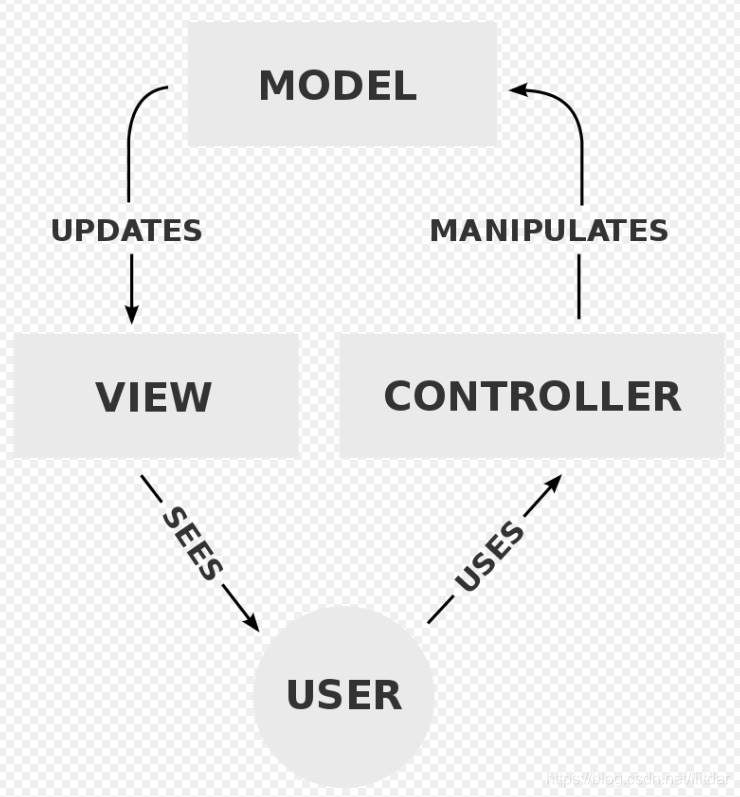
MVC 模式的目的是实现一种动态的程序设计，简化后续对程序的修改和扩展，并且使程序某一部分的重复利用成为可能。除此之外，MVC 模式通过对复杂度的简化，使程序的结构更加直观。软件系统在分离了自身的基本部分的同时，也赋予了各个基本部分应有的功能。专业人员可以通过自身的专长进行相关的分组：

模型（Model）：程序员编写程序应有的功能（实现算法等）、数据库专家进行数据管理和数据库设计（可以实现具体的功能）；

控制器（Controller）：负责转发请求，对请求进行处理；

视图（View）：界面设计人员进行图形界面设计。

MVC 模式的描述如下图所示：



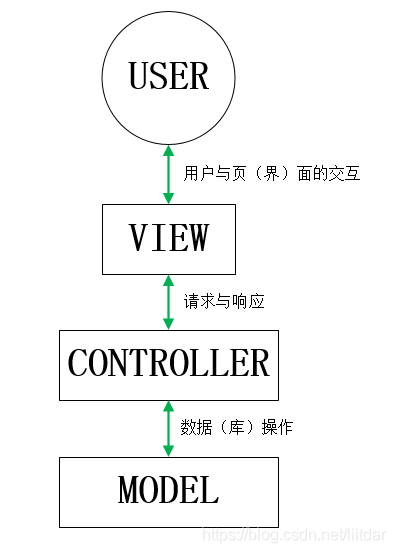
MVC 模式中三个组件的详细介绍如下：

• 模型（Model）：用于封装与应用程序业务逻辑相关的数据以及对数据的处理方法。Model 有对数据直接访问的权力，例如对数据库的访问。Model 不依赖 View 和 Controller，也就是说， Model 不关心它会被如何显示或是如何被操作。但是 Model 中数据的变化一般会通过一种刷新机制被公布。为了实现这种机制，那些用于监视此 Model 的 View 必须事先在此 Model 上注册，由此，View 可以了解在数据 Model 上发生的改变。（如，软件设计模式中的“观察者模式”）；

• 视图（View）：能够实现数据有目的的显示（理论上，这不是必需的）。在 View 中一般没有程序上的逻辑。为了实现 View 上的刷新功能，View 需要访问它监视的数据模型（即 Model），因此应该事先在被它监视的数据那里注册；

• 控制器（Controller）：起到不同层面间的组织作用，用于控制应用程序的流程。它处理事件并作出响应。“事件”包括用户的行为和数据 Model 上的改变。

从 MVC 模式的一般理解来看，视图层与模型层是存在直接联系的，并且模型层的变化会通过视图层反映出来，这确实是 MVC 模式的标准理解，不过在我目前接触到的实际应用中，更多的情况是，视图层与模型层是通过控制层联系起来的，两者之间并无直接的联系，三者之间的关系更类似下图所示：



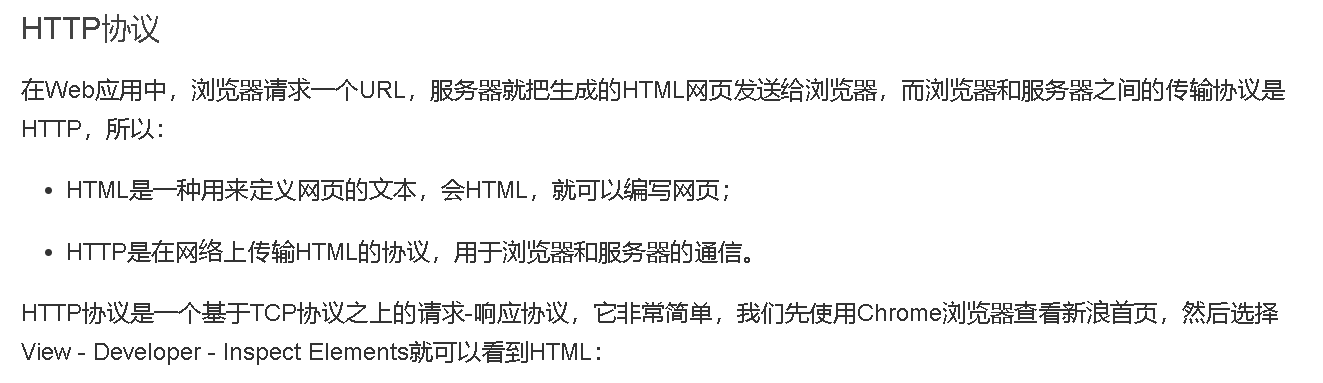
结合 MVC 模式的标准解释来看，上述模式可能是 MVC 模式的一种变型使用，这只是个人的武断猜测，并无实据，待以后对设计模式有深入了解后，可能就会有准确答案了(^-^)。

* 1. 优点与缺点：
     1. 优点： 低耦合；重用性高； 生命周期成本低； 部署快； 可维护性高； 有利软件工程化管理；
     2. 缺点： 没有明确的定义； 不适合小、中型应用程序； 增加系统结构和实现的复杂性； 视图对模型数据的低效率访问

1. Report\_MVC
   1. MVC\_开发

今天我们访问网站，使用App时，都是基于Web这种Browser/Server模式，简称BS架构，它的特点是，客户端只需要浏览器，应用程序的逻辑和数据都存储在服务器端。浏览器只需要请求服务器，获取Web页面，并把Web页面展示给用户即可。

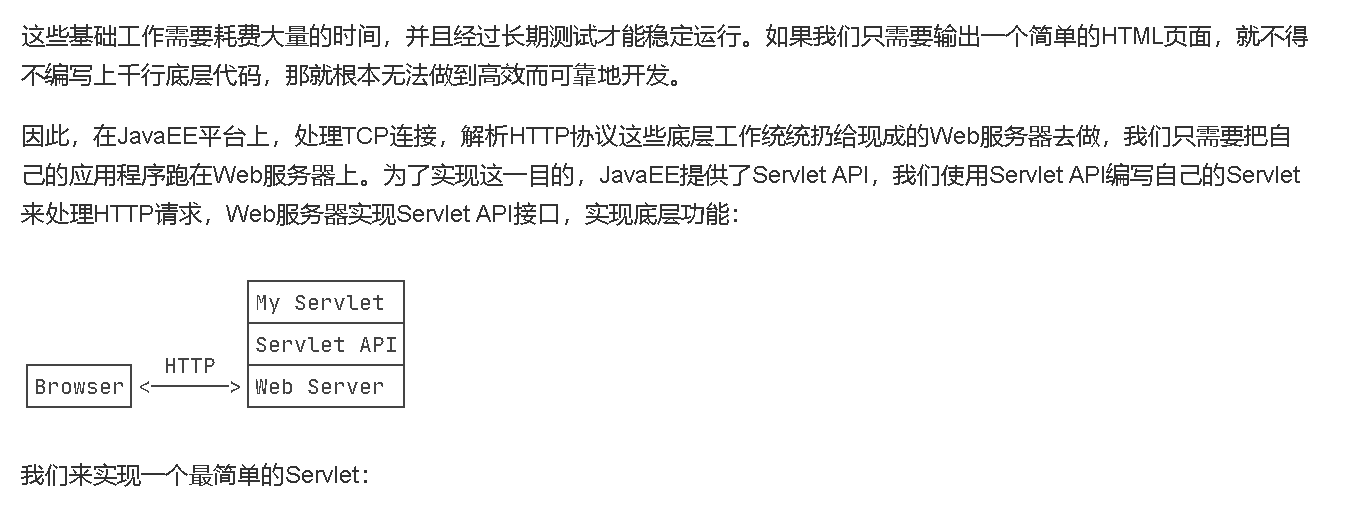
Web页面具有极强的交互性。由于Web页面是用HTML编写的，而HTML具备超强的表现力，并且，服务器端升级后，客户端无需任何部署就可以使用到新的版本，因此，BS架构升级非常容易。



### • 编写HTTP Server



• Servlet



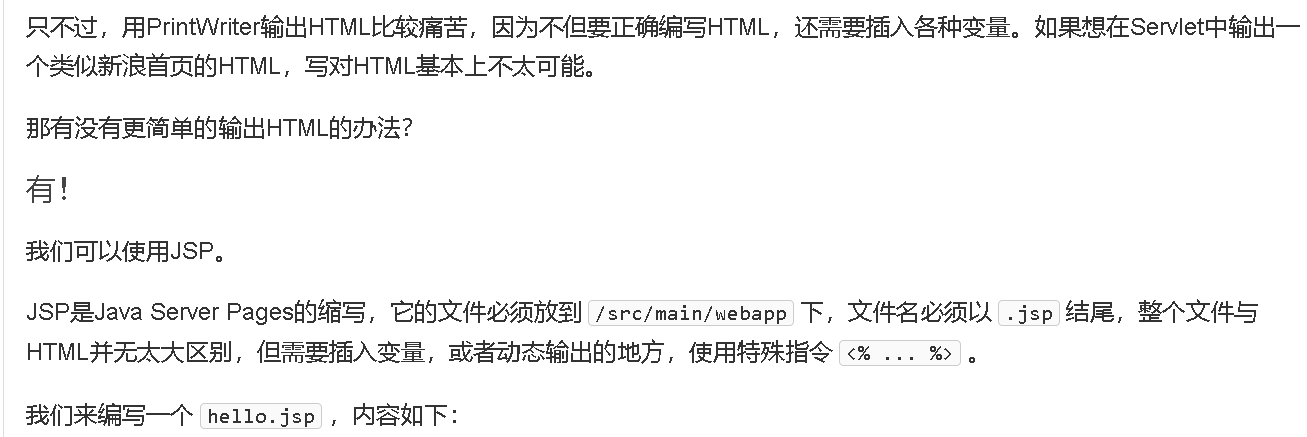
现在问题来了：Servlet API是谁提供？

Servlet API是一个jar包，我们需要通过Maven来引入它，才能正常编译。编写pom.xml文件如下：



• Tomcat

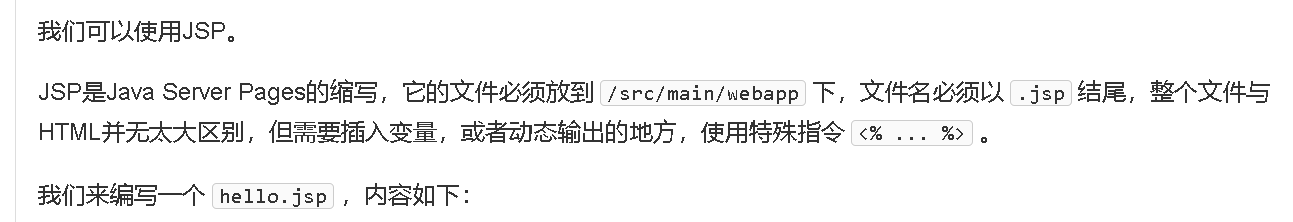
• JSP



• MVC

• Web开发





• Servlet适合编写Java代码，实现各种复杂的业务逻辑，但不适合输出复杂的HTML；

• JSP适合编写HTML，并在其中插入动态内容，但不适合编写复杂的Java代码。

可以将两者接和起来，发挥各自的优点，避免各自的缺点：

JavaBean —— 对象

• 例子：



在UserServlet中，我们可以从数据库读取User、School等信息，然后，把读取到的JavaBean先放到HttpServletRequest中，再通过forward()传给user.jsp处理：

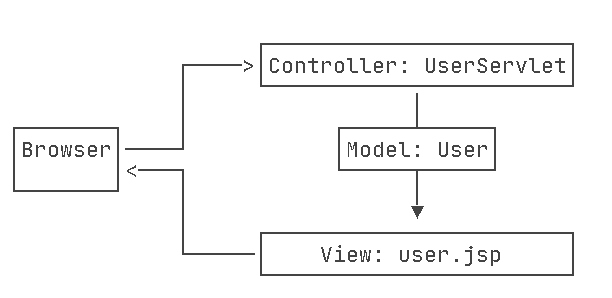


在user.jsp中，我们只负责展示相关JavaBean的信息，不需要编写访问数据库等复杂逻辑：



Servlet——尤指 Java 语言中在服务器上运行的）小型应用程序；小服务程序

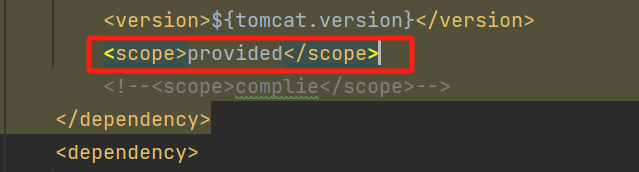
我们把UserServlet看作业务逻辑处理，把User看作模型，把user.jsp看作渲染，这种设计模式通常被称为MVC：Model-View-Controller，即UserServlet作为控制器（Controller），User作为模型（Model），user.jsp作为视图（View），整个MVC架构如下：



使用MVC模式的好处是，Controller专注于业务处理，它的处理结果就是Model。Model可以是一个JavaBean，也可以是一个包含多个对象的Map，Controller只负责把Model传递给View，View只负责把Model给“渲染”出来，这样，三者职责明确，且开发更简单，因为开发Controller时无需关注页面，开发View时无需关心如何创建Model。

MVC\_Development：

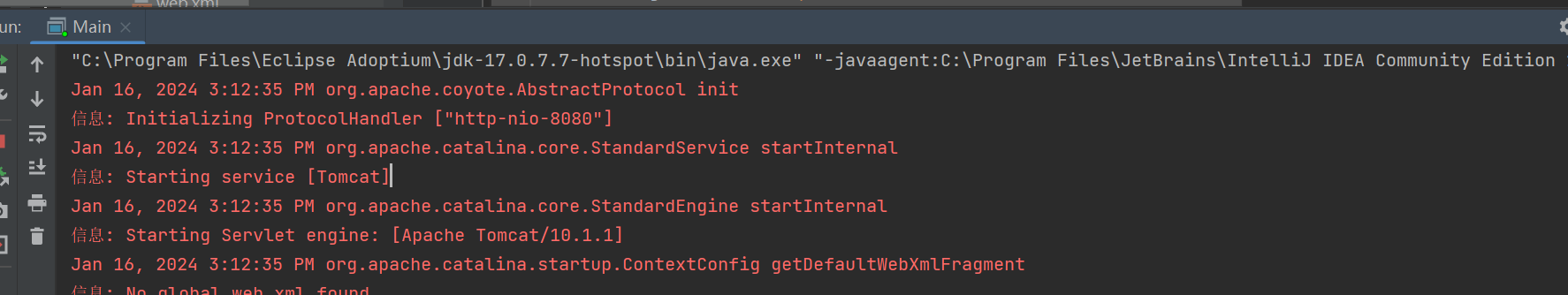
* 1. POM bug:



* 1. Solution:



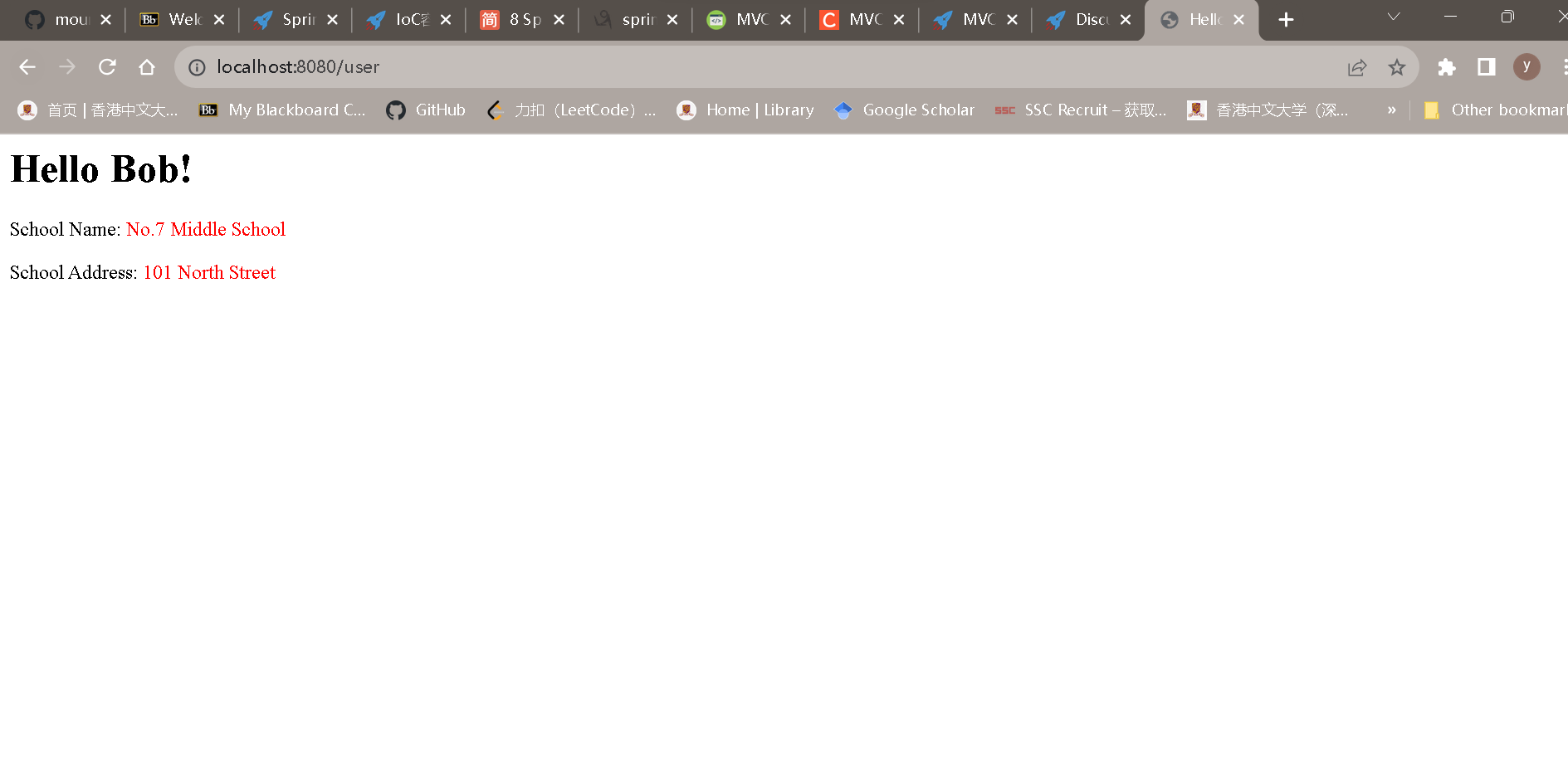
2.1. Bug:



2.2. Solution:

((( Do not need to solve it! Because the outcome is OK. (?) )))

Outcome:



* 1. MVC\_高级开发 (MVC\_Development\_Advanced)

通过结合Servlet和JSP的MVC模式，我们可以发挥二者各自的优点：

• Servlet实现业务逻辑；

• JSP实现展示逻辑。

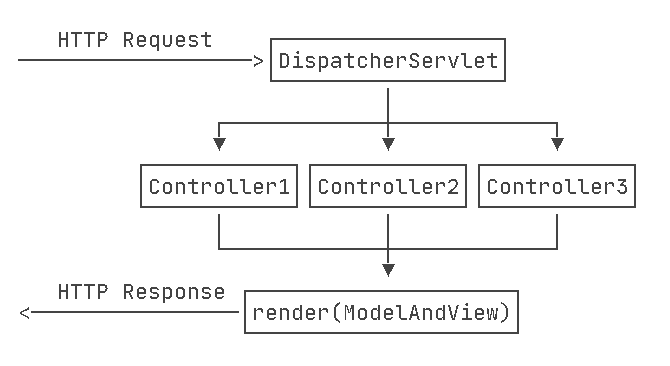
但是，直接把MVC搭在Servlet和JSP之上还是不太好，原因如下：

- Servlet提供的接口仍然偏底层，需要实现Servlet调用相关接口；

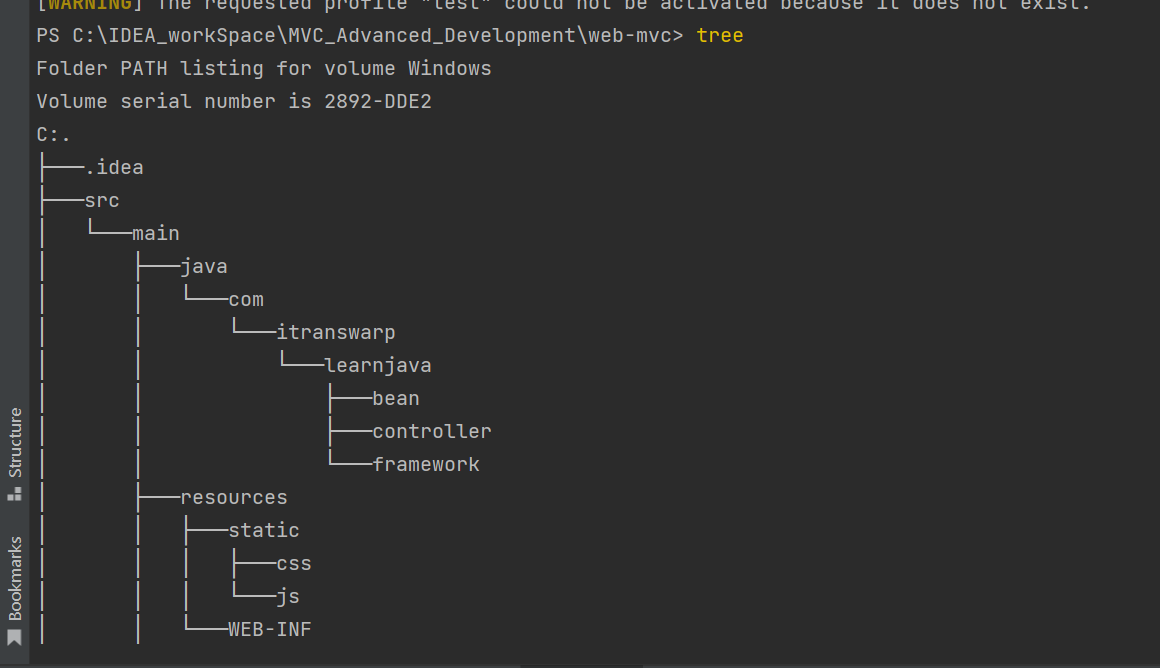
- JSP对页面开发不友好，更好的替代品是模板引擎；

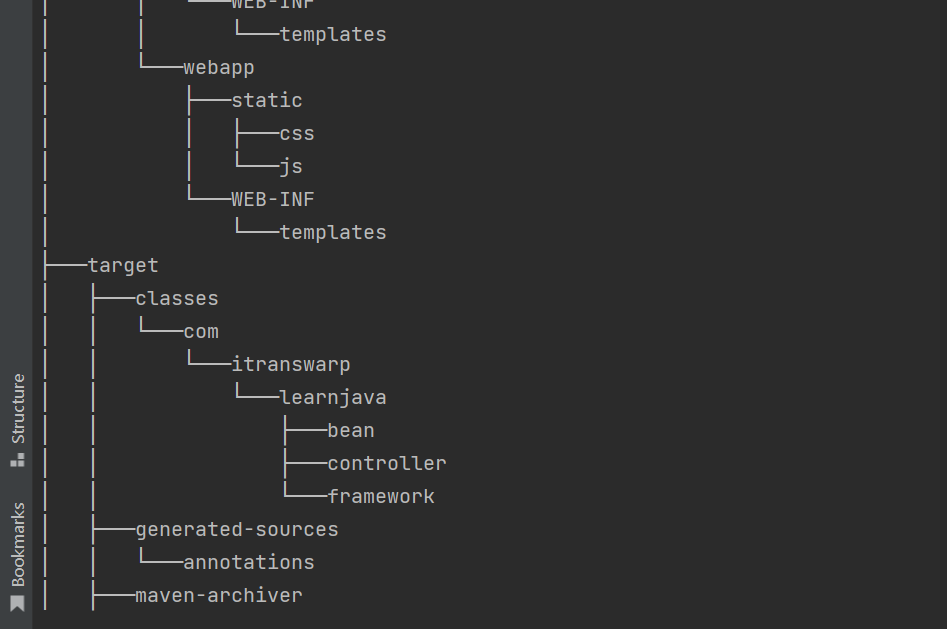
- 业务逻辑最好由纯粹的Java类实现，而不是强迫继承自Servlet。

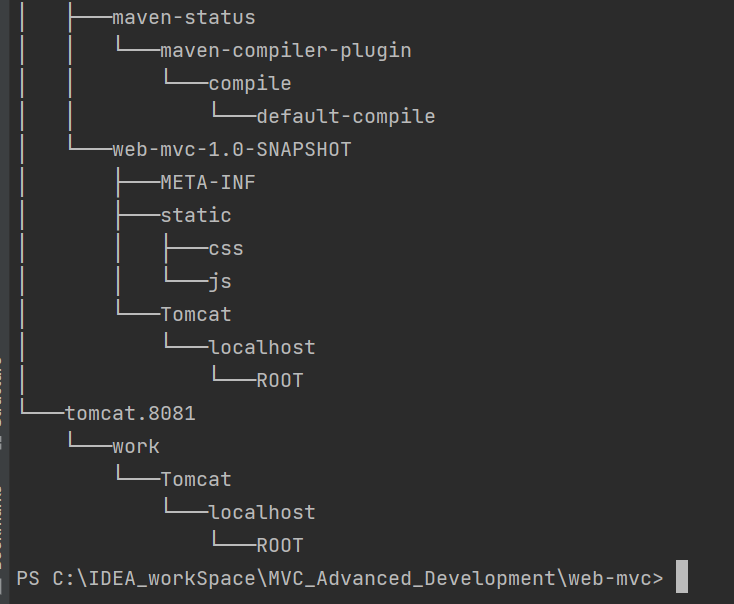
• 这个MVC的架构如下:



• Tree:







OUTPUT:

