

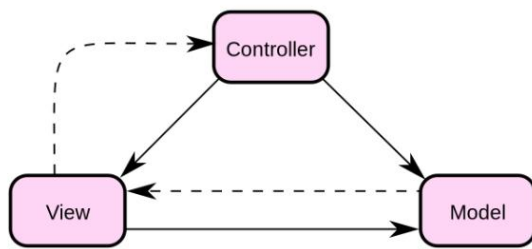
# MVC(Model-View\_Controller)

## 1. MVC 模式

### MVC 模式

MVC 模式代表 Model-View-Controller (模型-视图-控制器) 模式。这种模式用于应用程序的分层开发。

- **Model (模型)** - 模型代表一个存取数据的对象或 JAVA POJO。它也可以带有逻辑，在数据变化时更新控制器。
- **View (视图)** - 视图代表模型包含的数据的可视化。
- **Controller (控制器)** - 控制器作用于模型和视图上。它控制数据流向模型对象，并在数据变化时更新视图。它使视图与模型分离开。



## 2. MVC\_Student\_Implementation

### 2.1. Class(Student & StudentView & StudentController)

```
public class Student {
    private String rollNo;
    private String name;

    public String getRollNo() {
        return rollNo;
    }

    public void setRollNo(String rollNo) {
        this.rollNo = rollNo;
    }

    public String getName() {
        return name;
    }
}

private Student model;
private StudentView view;

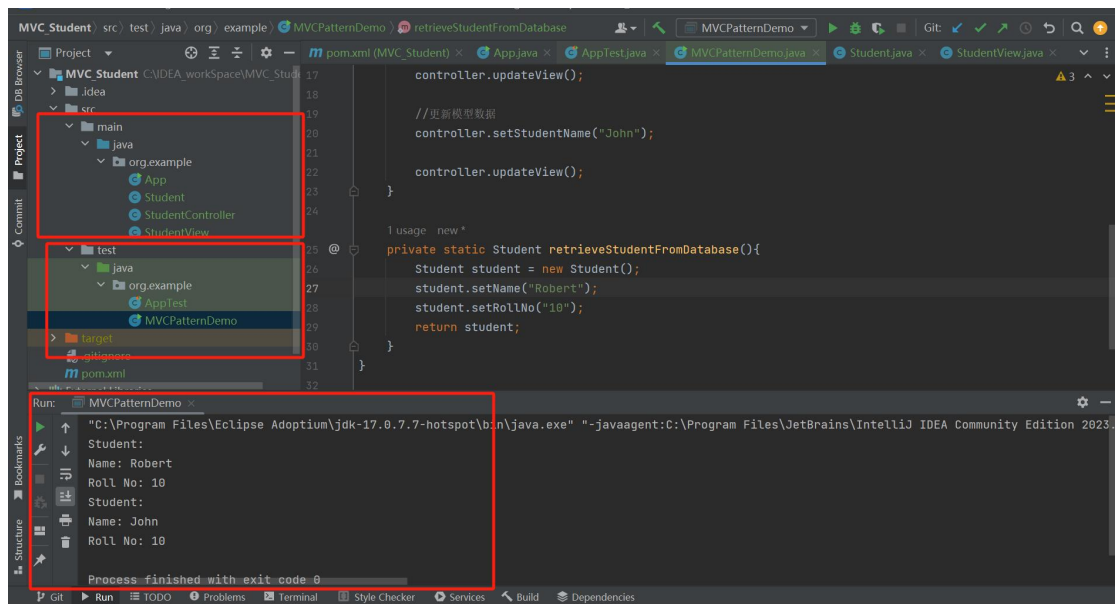
public StudentController(Student model, StudentView view){
    this.model = model;
    this.view = view;
}

public void setStudentName(String name){
    model.setName(name);
}

package org.example;

public class StudentView {
    public void printStudentDetails(String studentName, String studentRollNo){
        System.out.println("Student: ");
        System.out.println("Name: " + studentName);
        System.out.println("Roll No: " + studentRollNo);
    }
}
```

## 2.2. Output



Done :D

## 3. MPC 模式

### 3.1. 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)。

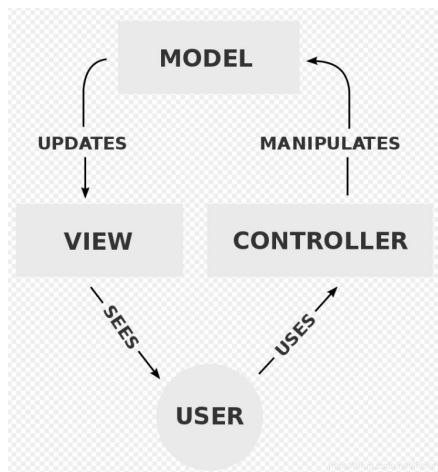
MVC 模式的目的是实现一种动态的程序设计，简化后续对程序的修改和扩展，并且使程序某一部分的重复利用成为可能。除此之外，MVC 模式通过对复杂度的简化，使程序的结构更加直观。软件系统在分离了自身的基本部分的同时，也赋予了各个基本部分应有的功能。专业人员可以通过自身的专长进行相关的分组：

**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。

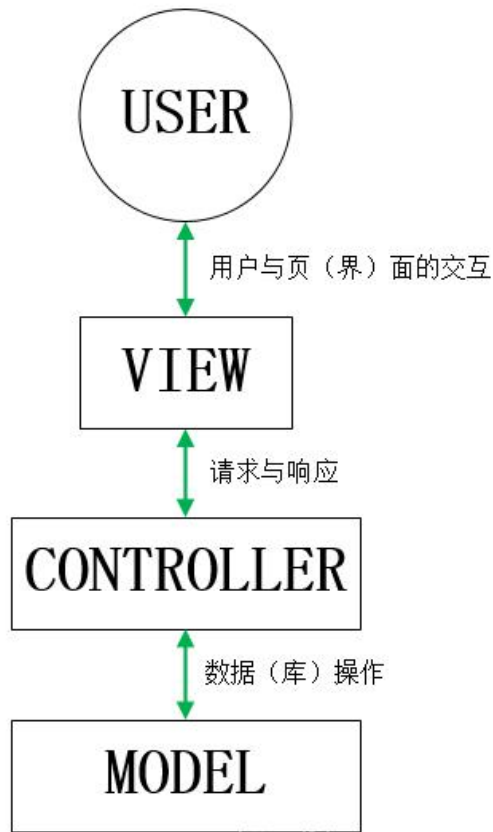
MVC 模式的描述如下图所示：



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：



<https://blog.csdn.net/11itdar>

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(^-^).

### 3.2. 优点与缺点:

3.2.1. 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;

3.2.2. 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

## 4. Report\_MVC

### 4.1. 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。

## HTTP协议

在Web应用中，浏览器请求一个URL，服务器就把生成的HTML网页发送给浏览器，而浏览器和服务器的传输协议是HTTP，所以：

- HTML是一种用来定义网页的文本，会HTML，就可以编写网页；
- HTTP是在网络上传输HTML的协议，用于浏览器和服务器的通信。

HTTP协议是一个基于TCP协议之上的请求-响应协议，它非常简单，我们先使用Chrome浏览器查看新浪首页，然后选择View - Developer - Inspect Elements就可以看到HTML：

## • 编写 HTTP Server

### 编写HTTP Server

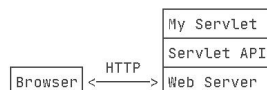
我们来看一下如何编写HTTP Server。一个HTTP Server本质上是一个TCP服务器，我们先用TCP编程的多线程实现的服务端框架：

```
public class Server {  
    public static void main(String[] args) throws IOException {  
        ServerSocket ss = new ServerSocket(8080); // 监听指定端口  
        System.out.println("server is running...");  
        for (;;) {  
            Socket sock = ss.accept();  
            System.out.println("connected from " + sock.getRemoteSocketAddress());  
            Thread t = new Handler(sock);  
            t.start();  
        }  
    }  
}  
  
class Handler extends Thread {
```

## • Servlet

这些基础工作需要耗费大量的时间，并且经过长期测试才能稳定运行。如果我们只需要输出一个简单的HTML页面，就不得不编写上千行底层代码，那就根本无法做到高效而可靠地开发。

因此，在JavaEE平台上，处理TCP连接，解析HTTP协议这些底层工作统统扔给现成的Web服务器去做，我们只需要把自己的应用程序跑在Web服务器上。为了实现这一目的，JavaEE提供了Servlet API，我们使用Servlet API编写自己的Servlet来处理HTTP请求，Web服务器实现Servlet API接口，实现底层功能：



我们来实现一个最简单的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：

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：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.itranswarp.learnjava</groupId>
  <artifactId>web-servlet-hello</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <java.version>17</java.version>
  </properties>
</project>
```

## • Tomcat

## • JSP

只不过，用 `PrintWriter` 输出HTML比较痛苦，因为不但要正确编写HTML，还需要插入各种变量。如果想在Servlet中输出一个类似新浪首页的HTML，写对HTML基本上不太可能。

那有没有更简单的输出HTML的办法？

有！

我们可以使用JSP。

JSP是Java Server Pages的缩写，它的文件必须放到 `/src/main/webapp` 下，文件名必须以 `.jsp` 结尾，整个文件与HTML并无太大区别，但需要插入变量，或者动态输出的地方，使用特殊指令 `<% ... %>`。

我们来编写一个 `hello.jsp`，内容如下：

## • MVC

## • Web 开发

Web开发

Web基础

Servlet入门

Servlet开发

Servlet进阶

JSP开发

MVC开发

MVC高级开发

使用Filter

使用Listener

我们可以使用JSP。

JSP是Java Server Pages的缩写，它的文件必须放到 `/src/main/webapp` 下，文件名必须以 `.jsp` 结尾，整个文件与HTML并无太大区别，但需要插入变量，或者动态输出的地方，使用特殊指令 `<% ... %>`。

我们来编写一个 `hello.jsp`，内容如下：

- 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：

假设我们已经编写了几个JavaBean：

```
public class User {
    public long id;
    public String name;
    public School school;
}

public class School {
    public String name;
    public String address;
}
```

在 `UserController` 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 `UserController` 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 `UserController` 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：

```
@WebServlet(urlPatterns = "/user")
public class UserController extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        // 假装从数据库读取：
        School school = new School("No.1 Middle School", "101 South Street");
        User user = new User(123, "Bob", school);
        // 放入Request中：
        req.setAttribute("user", user);
        // forward给用户.jsp：
        req.getRequestDispatcher("/WEB-INF/user.jsp").forward(req, resp);
    }
}
```

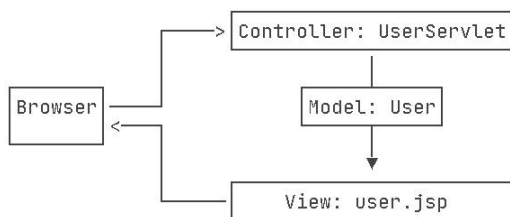
在 `user.jsp` 中，我们只负责展示相关 JavaBean 的信息，不需要编写访问数据库等复杂逻辑：

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：

```
<%@ page import="com.itranswarp.learnjava.bean.*"%>
<%
    User user = (User) request.getAttribute("user");
%>
<html>
<head>
    <title>Hello World - JSP</title>
</head>
<body>
    <h1>Hello <%= user.name %>!/</h1>
    <p>School Name:
    <span style="color:red">
        <%= user.school.name %>
    </span>
    </p>
    <p>School Address:
    <span style="color:red">
```

**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:

### 1.1. POM bug:

```
<version>${tomcat.version}</version>
<scope>provided</scope>
<!--<scope>compile</scope>-->
</dependency>
<dependency>
```

### 1.2. Solution:





廖雪峰

#4 Created at: 3/27/2020 18:18

那是idea的问题，如果你把provided改成compile，生成的war包会很大，因为把tomcat打包进去了

## 解决方案

打开idea的Run/Debug Configurations:

选择Application - Main

右侧Configuration: Use classpath of module

勾选Include dependencies with "Provided" scope

### 2.1. Bug:

```
Main x
"C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
Jan 16, 2024 3:12:35 PM org.apache.coyote.AbstractProtocol init
信息: Initializing ProtocolHandler ["http-nio-8080"]
Jan 16, 2024 3:12:35 PM org.apache.catalina.core.StandardService startInternal
信息: Starting service [Tomcat]
Jan 16, 2024 3:12:35 PM org.apache.catalina.core.StandardEngine startInternal
信息: Starting Servlet engine: [Apache Tomcat/10.1.1]
Jan 16, 2024 3:12:35 PM org.apache.catalina.startup.ContextConfig getDefaultWebXmlFragment
信息: No global web.xml found
```

### 2.2. Solution:

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

#### Outcome:

```
mon x Web x Spti x loCI x 8 Sp x spti x MV x MV x MV x MV x Disc x lck x
localhost:8080/user
Hello Bob!
School Name: No.7 Middle School
School Address: 101 North Street
```

## 4.2. MVC\_高级开发 (MVC\_Development\_Advanced)

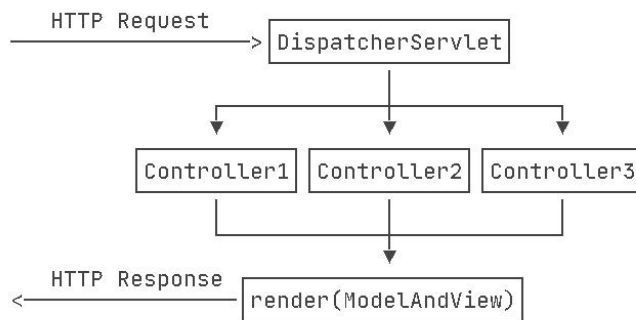
通过结合 Servlet 和 JSP 的 MVC 模式，我们可以发挥二者各自的优点：

- 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:

```
[WARNING] The requested profile "test" could not be activated because it does not exist.
PS C:\IDEA_workspace\MVC_Advanced_Development\web-mvc> tree
Folder PATH listing for volume Windows
Volume serial number is 2892-DDE2
C:..
  .idea
  src
    main
      java
        com
          itranswarp
            learnjava
              bean
              controller
              framework
      resources
        static
          css
          js
        WEB-INF
```

```
  WEB-INF
    templates
  webapp
    static
      css
      js
    WEB-INF
      templates
  target
    classes
      com
        itranswarp
          learnjava
            bean
            controller
            framework
    generated-sources
      annotations
  maven-archiver
```

```
maven-status
├── maven-compiler-plugin
│   └── compile
│       └── default-compile
└── web-mvc-1.0-SNAPSHOT
    ├── META-INF
    ├── static
    │   ├── css
    │   └── js
    ├── Tomcat
    │   ├── localhost
    │   └── ROOT
└── tomcat.8081
    ├── work
    │   ├── Tomcat
    │   ├── localhost
    │   └── ROOT
PS C:\IDEA_workSpace\MVC_Advanced_Development\web-mvc>
```

OUTPUT:

