Web开发演变过程

——以Java EE平台为例

404 We Found 后端组 Leslie

阶段〇：在很久以前，还是静态的HTML网页。要想显示每天的天气，就必须每天手动更新HTML网页。

阶段一：为了让静态网页“动起来”，就需要在HTML中引入服务端语言元素。例如：Java，PHP，ASP.NET等

×1： 显示当前北京时间（这里假设本地的不是北京时间，正确的北京时间在服务器。）

PHP版本：

```
<html>
<head>
    <title>PHP</title>
</head>
<body>
    <?php
        echo("Hello World! 今天是: " . date('Y-m-d h:i:s', time()));
    ?>
</body>
</html>
```

Hello World! 今天是: 2020-05-16 03:39:20

Java版本：

```
<%@ page import="java.util.Date" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Java</title>
</head>
<body>
    <%
        out.print("Hello World! 今天是: " + new Date());
    %>
</body>
</html>
```

Hello World! 今天是: Sat May 16 15:40:52 CST 2020

但是他们本质上都还是HTML文件，只是经过服务器端的“翻译”！

在这里以Java EE平台为例，
介绍一下MVC，MVVM，及其两者关系。

×2：显示数据库表goods的内容到页面中。

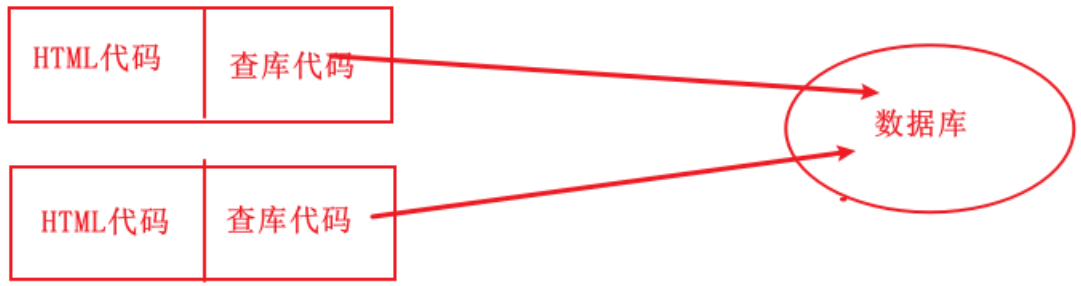
数据库表：

开始事务 文本 筛选

id	name	money
1	面包	1
2	葡萄	12
3	苹果	3
4	香蕉	1
5	白菜	1
6	糯米鸡	10
7	叉烧包	10
8	咸猪骨粥	30

网页效果：

==========
id：1name：面包money：1.0
==========
id：2name：葡萄money：12.0
==========
id：3name：苹果money：3.0
==========
id：4name：香蕉money：1.0
==========
id：5name：白菜money：1.0
==========
id：6name：糯米鸡money：10.0
==========
id：7name：叉烧包money：10.0
==========
id：8name：咸猪骨粥money：30.0



存在问题：查库代码不能复用，代码冗余度高，维护成本高。

阶段二：JSP + Java Bean　　名称中的“Bean”是用于Java的**可重用软件组件**的惯用叫法。

Q1：怎么样算是Java Bean？

要成为JavaBean类，则必须遵循关于命名、构造器、方法的特定规范。有了这些规范，才能有可以使用、复用、替代和连接JavaBeans的工具。

规范如下：

- 有一个public的无参数构造器。
- 属性可以通过*get*、*set*、*is*（可以替代get，用在布尔型属性上）方法或遵循特定命名规范的其他方法访问。
- 可序列化。

Q2：Java Bean长什么样？

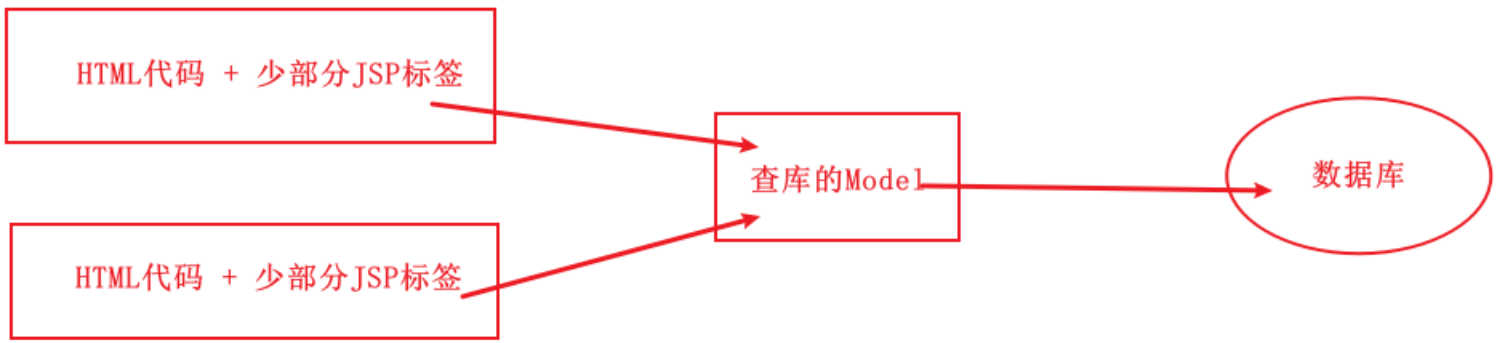
```java
/*实现序列化接口*/
public class PersonBean implements java.io.Serializable {
    private String name = null;
    private boolean deceased = false;

    /** 无参构造器(没有参数) */
    public PersonBean() { }

    /**getter和setter**/
    public String getName() { return name; }
    public void setName(String value) { name = value; }
    public boolean isDeceased() {  return deceased;  }
    public void setDeceased(boolean value) { deceased = value; }
}
```

把查询数据库的代码，单独拎出来，作为一个JavaBean供JSP调用。
当数据库连接信息修改时，只需要修改一次。

嗯，解决了阶段一存在的问题：查库代码不能复用，代码冗余度高，维护成本高。



但是，真的没有问题了吗？

×3：计算器案例。

计算器和前面查询数据库最大的不同是：计算器需要请求参数！

* 请求参数需要认证的时候非常不方便，JSP页面本身就是显示数据用的。

* 因为他本质上是HTML，只是为了解决HTML只能是静态的弊端。

* 处理业务逻辑不属于他业务范围。如果都堆在一起，会造成视图和业务逻辑的耦合。

```jsp
<%@ page language="java" pageEncoding="UTF-8"%>
<%--使用CalculatorBean --%>
<jsp:useBean id="calcBean" class="com.isleslie.mvc.j

<%--接收参数
    ！！！不是jsp应该干的活！！！
    假如我要进行请求参数的验证？？？
    ....
    ....                    业务逻辑代码
    ....
--%>
<jsp:setProperty name="calcBean" property="*"/>

<%
    //使用CalculatorBean进行计算
    calcBean.calculate();
%>
<!DOCTYPE HTML>
```

阶段三：JSP + JavaBean + Servlet

×4：登录案例。登录需要进行请求参数非空验证！

如果在阶段二的方式，需要放到JSP页面进行验证。在阶段三，使用Servlet进行验证，也就是说业务逻辑的处理。

```java
@WebServlet("/requestServlet_v1")
public class RequestServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String username = request.getParameter( s: "username");
        String password = request.getParameter( s: "password");
        if("".equals(username) || "".equals(password)){
            response.sendRedirect( s: "jsp-javabean-servlet/error.jsp?msg=" + URLEncoder.encode( s: "帐号或者密码不能为空", enc: "utf-8"));
            return;
        }
        User user = new User();
        user.setUsername(username);
        user.setPassword(password);
        if(!user.login()){
            response.sendRedirect( s: "jsp-javabean-servlet/error.jsp?msg=" + URLEncoder.encode( s: "帐号或者密码错误", enc: "utf-8"));
        }else{
            response.sendRedirect( s: "jsp-javabean-servlet/success.jsp");
        }

    }
```

请求参数接收

判断请求参数是否空（业务逻辑）

封装对象

调用JavaBean方法并响应（以重定向的方式）

阶段二存在问题也解决了，但是还存在问题。

对于User类这个Java Bean，业务方法(login)和数据(用户的信息)耦合了，需要进一步拆分。

用户信息：

```java
public class User {
    private String username;
    private String password;

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}
```

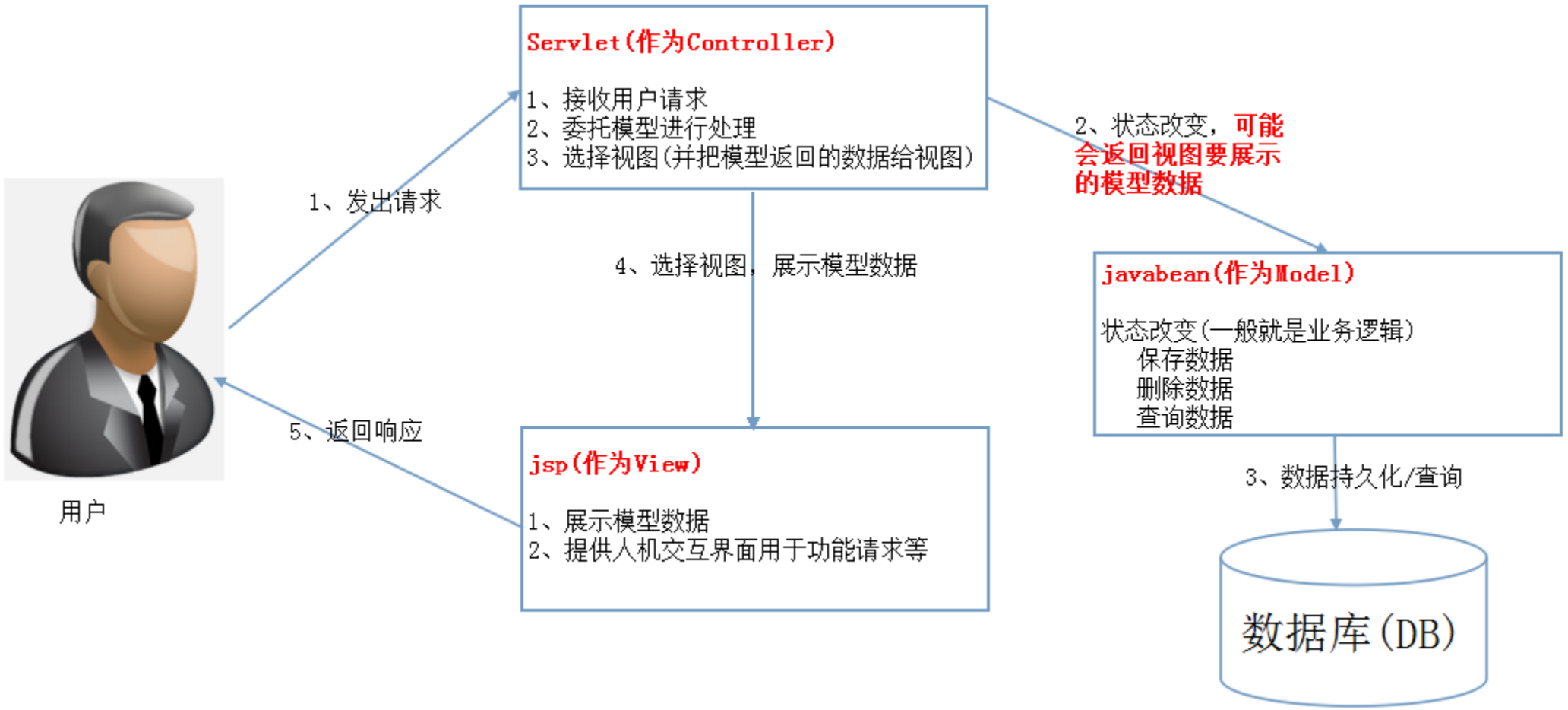
业务逻辑：

```java
public class UserDao {
    public boolean login(User user){
        System.out.println(user);
        boolean flag = false;
        if("admin".equals(user.getUsername()) && "admin".equals(user.getUsername())){
            flag = true;
        }
        return flag;
    }
}
```

拆出来的业务逻辑就是我们熟悉的Dao，用于访问数据库。

所以说，MVC是什么？

M：Model，模型　　（User类，UserDao类）　　Bool flag = userDao.select();
V：View，视图　　（View：JSP）
C：Controller，控制器　　（Controller：Servlet）



维基百科定义：

将应用程序划分为三种组件，模型 - 视图 - 控制器（MVC）设计定义它们之间的相互作用。

•**模型（Model）** 用于封装与应用程序的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。

•**视图（View）**能够实现数据有目的的显示（理论上，这不是必需的）。

•**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。

From：https://zh.wikipedia.org/wiki/MVC

阶段四：HTML + JavaBean + Servlet（前后端分离）

继续前面的登录案例。在前面不是前后端分离的情况下，是使用重定向的方式进行响应。
如果前后端分离，使用Ajax通过Json进行交互。

由重定向👇：

```
if(!new UserDao().login(user)){
    response.sendRedirect( s: "jsp-javabean-servlet/error.jsp?msg=" + URLEncoder.encode( s: "帐号或者密码错误", enc: "utf-8"));
}else{
    response.sendRedirect( s: "jsp-javabean-servlet/success.jsp");
}
```

到输出Json👉：

```
if(!new UserDao().login(user)){
    //response.sendRedirect("jsp-javabean-servlet/error.jsp?msg=" + URLEncoder.encode("帐号或者密码错误","utf-8"));
    res.put("code",1);
    res.put("message","帐号或者密码错误");
}else{
    //response.sendRedirect("jsp-javabean-servlet/success.jsp");
    res.put("code",0);
    res.put("message","登录成功！");
}
//响应
mapper.writeValue(response.getWriter(),res);
```

这时候前端就可以使用JQ封装的AJAX进行请求，并把响应的JSON通过操作Dom元素给显示到页面上。

```
<div>
    用户名: <label><input type="text" id="username"></label><br>
    密码: <label><input type="password" id="password"></label>
    <button type="submit" onclick="login()">登录</button>
    <p id="msg" style="color: red"></p>
</div>
<script>
    function login() {
        let username = $('#username').val();
        let password = $('#password').val();
        $.ajax({
            url:"/what_is_mvc_war_exploded/user/loginServlet",
            data:{"username":username,"password":password},
            dataType:"json",
            success: function(data) {
                console.log(data)
                $("#msg").html(data.message);
            }
        });
    }
</script>
```

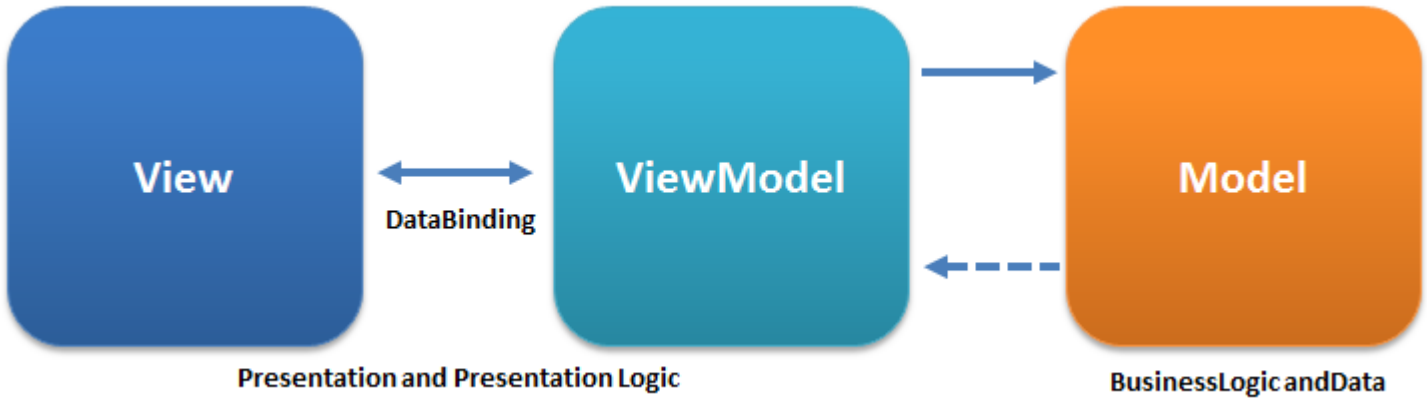
但是直接操作DOM元素十分麻烦。

我们关注的本身是怎么把响应的数据显示到页面上，而不是怎么去找DOM元素。

这个阶段前端的视图（HTML）和数据（Model）还是耦合在一起的。

把HTML和Model联系起来需要手动获取DOM元素，然后赋值。

阶段五：HTML + JavaBean + Servlet（前后端分离，前端数据和视图解耦）



前端数据和视图通过ViewModel连接实现解耦。

ViewModel相当于View和Model的桥梁，当Model修改，ViewModel会自动修改并同步到View。

也就是熟悉的MVVM。

```html
<div id="login">
    用户名：<label><input type="text" v-model="username" ></label><br>
    密码：<label><input type="password" v-model="password"></label>
    <button type="submit" v-on:click="login">登录</button>
    <p style="color: red">{{ message }}</p>
</div>
```

使用Vue.js进行双向绑定

Axios.js作为Ajax请求

```html
<script>
    let login = new Vue({
        el:"#login",
        data:{
            username:'', password:'', message:''
        },
        methods:{
            login: function () {
                axios.get('/what_is_mvc_war_exploded/user/loginServlet', {
                    params: {
                        username:this.username,
                        password:this.password
                    }
                })
                .then(response => {
                    this.message = response.data.message;
                })
                .catch(error => {
                    alert(error);
                });
            }
        }
    })
</script>
```

这里的V和M和MVC一样吗？也就是说MVC和MVVM有关系吗？

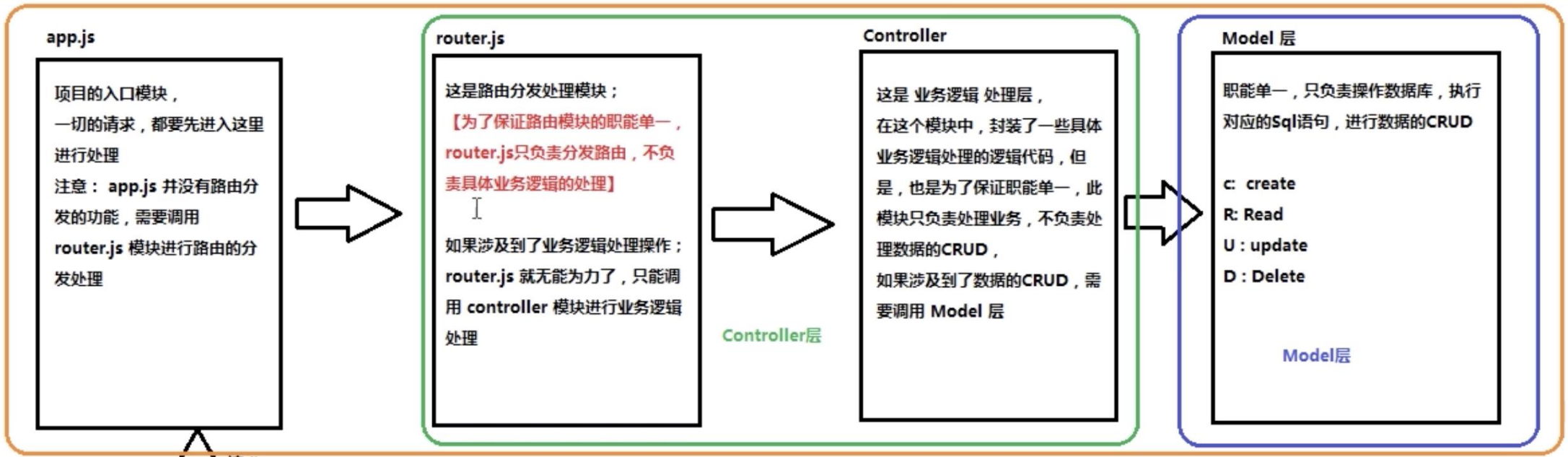
首先，弄清楚MVVM的V和M代表什么。V代表着HTML，M代表着后端传过来的数据。而MVC的V在前后端分离的模式下代表着Json，M代表着JavaBean。

因此，我认为两者关系不大。

在前后端分离的情况下，前端可以单独作为一个单独的应用；后端可以单独作为一个单独的应用，只不过V变成了JSON。

MVVM属于前端，MVC属于后端。

处理过程

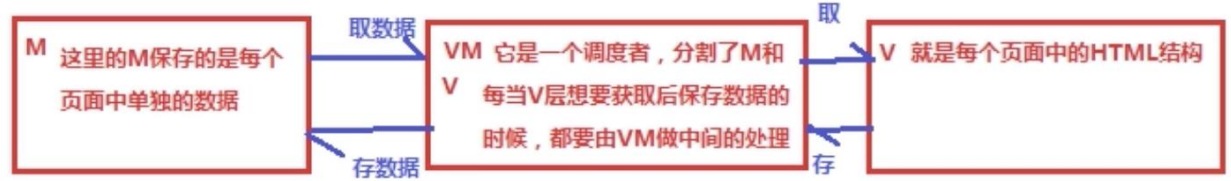


MVVM是前端视图层的分层开发思想，主要把每个页面，分成了 M 、 V 和 VM
其中，VM 是 MVVM 思想的核心；因为 VM是M和V之间的调度者

View 视图层

请求

每当用户操作了界面，如果需要进行业务的处理，都会通过网络请求，去请求后端的服务器，此时，我们的这个请求，就会被 后端的 App.js 监听到



前端页面中使用MVVM的思想，主要是为了让我们开发更加方便，因为 MVVM提供了数据的双向绑定；
注意：数据的双向绑定是由VM提供的；