MVC

2017年6月29日 13:26

按职责划分，将代码分层
抽离出业务逻辑和UI逻辑

MVC

Model模型
存放所有数据对象（数据和相关逻辑），真正的逻辑处理
实现其他组件功能，如popup，form等

View视图
呈现给用户，与用户产生交互

View和Model不能直接沟通

Controller控制器
从视图获得事件和输入，并及时处理回调，以及和模型进行必要的对接
进行页面节点事件的注册和控制，以及页面加载性能的实现
最好只做以下三件事【

1. 初始化时，构造响应的view和model
2. 监听model层，将model层的数据传递到view层
3. 监听view层的事件，并且将view层事件转发到model层

例子：

```
<input type="button" value="baidu" id="baidu"/>

function G(id){    // 属于Model
  return document.getElementById(id);
}
var UI = new Object();  // Model
UI.register = function(id,even,fun,arr){
  if(G(id))
    G(id)["on"+even] = function(){ fun(arr); };
}
UI.register("baidu","click",tipInfo,[1,2]); // Controller
```

说明：
一个按钮
Html代码与js代码完全分离 -> 分离出View层
Js中，按钮对象封装成类，封装绑定事件的操作 ->分离出Model层
Control只负责给Model提交事件，触发Model中的操作

// MVC的弊端与MVVM的优点
当开发者在代码中调用大量相同的DOM API，代码难以维护

当大量DOM操作是页面渲染性能降低，加载速度变慢，影响用户体验
当Model变化频繁，需要主动更新到View，
当用户操作频繁，需要主动同步到Model，
难以维护复杂多变的数据状态
MVVM解决了这个问题

MVVM
Model-View-ViewModel
Model<=>ViewModel<=>View

Model 中定义数据修改和操作的业务逻辑

View 负责将数据模型转化成UI展现出来

ViewModel：就是与界面(view)对应的Model。因为，模型往往是不能直接跟视图控件一一对应上的，所以，需要再定义一个数据对象专门对应view上的控件。而ViewModel的职责就是把Model对象封装成可以显示和接受输入的界面数据对象。
通过双向数据绑定，连接View层和Model层，View数据的变化会同步到Model中，而Model数据的变化也会立即反应到View上。跳过了DOM操作。

MVVM的核心思想：不用再关注数据如何呈现到页面，由框架更新Model和View。

ViewModel框架的实现？

Vue.js 是采用 Object.defineProperty 的 getter 和 setter，并结合观察者模式来实现数据绑定的。