# 设计模式

* MVVM 和 MVC

[MVC、MVVM模式的概念与区别 - 理想三旬· - 博客园 (cnblogs.com)](https://www.cnblogs.com/wenkangIT/p/15149077.html" \l ":~:text=mvc %E5%92%8C mvvm %E5%85%B6%E5%AE%9E%E5%8C%BA%E5%88%AB%E5%B9%B6%E4%B8%8D%E5%A4%A7%E3%80%82,%E9%83%BD%E6%98%AF%E4%B8%80%E7%A7%8D%E8%AE%BE%E8%AE%A1%E6%80%9D%E6%83%B3%EF%BC%8C%E4%B8%BB%E8%A6%81%E5%8C%BA%E5%88%AB%E5%A6%82%E4%B8%8B%EF%BC%9A 2.mvvm %E9%80%9A%E8%BF%87%E6%95%B0%E6%8D%AE%E6%9D%A5%E9%A9%B1%E5%8A%A8%E8%A7%86%E5%9B%BE%E5%B1%82%E7%9A%84%E6%98%BE%E7%A4%BA%E8%80%8C%E4%B8%8D%E6%98%AF%E8%8A%82%E7%82%B9%E6%93%8D%E4%BD%9C%E3%80%82 3.mvc%E4%B8%ADModel%E5%92%8CView%E6%98%AF%E5%8F%AF%E4%BB%A5%E7%9B%B4%E6%8E%A5%E6%89%93%E4%BA%A4%E9%81%93%E7%9A%84%EF%BC%8C%E9%80%A0%E6%88%90Model%E5%B1%82%E5%92%8CView%E5%B1%82%E4%B9%8B%E9%97%B4%E7%9A%84%E8%80%A6%E5%90%88%E5%BA%A6%E9%AB%98%E3%80%82)

|  |
| --- |
| MVC 是一种使用 MVC（Model View Controller 模型-视图-控制器）设计创建 Web 应用程序的模式：[1]  Model（模型）表示应用程序核心（如数据库）。  View（视图）显示效果（HTML页面）。  Controller（控制器）处理输入（业务逻辑）。  MVC 模式同时提供了对 HTML、CSS 和 JavaScript 的完全控制。  **Model（模型）**是应用程序中用于处理应用程序数据逻辑的部分。 　　通常模型对象负责在数据库中存取数据。  **View（视图）**是应用程序中处理数据显示的部分。 　　通常视图是依据模型数据创建的。  **Controller（控制器）**是应用程序中处理用户交互的部分。 　　通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。 |
| **MVVM模式的组成部分****模型** 模型是指代表真实状态内容的领域模型（面向对象），或指代表内容的数据访问层（以数据为中心）。 **视图** 就像在MVC和MVP模式中一样，视图是用户在屏幕上看到的结构、布局和外观（UI）。 **视图模型** 视图模型是暴露公共属性和命令的视图的抽象。MVVM没有MVC模式的控制器，也没有MVP模式的presenter，有的是一个绑定器。在视图模型中，绑定器在视图和数据绑定器之间进行通信。 |

* 发布订阅模式、观察者模式

[面试官：说说你对发布订阅、观察者模式的理解？区别？ · Issue #295 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/295)

* 设计模式

[面试官：说说对设计模式的理解？常见的设计模式有哪些？ · Issue #290 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/290)

|  |
| --- |
| 观察者模式：  class subject{      constructor(){          this.observerList = [];      }      addObserver(observer){          this.observerList.push(observer);      }      removeObserver(observer){          const index = this.observerList.findIndex(o => o.name === observer.name);          this.observerList.splice(index,1);      }      notifyObserver(message){          this.observerList.forEach(observer => observer.notified(message));      }  }  class Observer{      constructor(name,subject){          this.name = name;          if(subject){              subject.addObserver(this);          }      }      notified(message){          console.log(this.name,"know the message :",message);      }  }  var s = new subject();  var o = new Observer("name",s);  var oo = new Observer("namee");  s.addObserver(oo);  s.notifyObserver("message"); |