# Vue框架

1. **MVC、MVP、MVVM**

**MVC：**

1. View传指令到Controller
2. Controller完成业务逻辑后，要求Model改变状态
3. Model将新的数据发送到View，用户得到反馈

**MVP：**

1. （Model）是业务逻辑层，主要负责数据，网络请求等操作
2. （View）是视图层，负责绘制UI元素、与用户进行交互
3. （Presenter）是View与Model交互的中间纽带，处理与用户交互的逻辑

**MVVM：**

1. Model层代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑
2. View代表UI组件，它负责 将数据模型转换成UI展现出来
3. ViewModel是一个同步View和Model的对象

ViewModel通过双向数据绑定把View层和Model层连接了起来，而View和Model之间的同步工作完全是 自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM，不需要关注数据状态 的同步问题，复杂的数据状态维护完全由MVVM来统一管理

对于 MVVM 来说，其实最重要的并不是通过双向绑定或者其他的方式将 View 与 ViewModel 绑定起 来，而是通过 ViewModel 将视图中的状态和用户的行为分离出一个抽象，这才是 MVVM 的精髓

1. **虚拟DOM**

相较于 直接操作DOM 来说（各种浏览器的规范），操作 JS 原生对象会快很多，并且我们也可以通过 JS 来模拟 DOM（tag、props、children），这种模拟出来的DOM称为虚拟DOM。

虚拟 DOM 提升性能的点在于 DOM 发生变化的时候，通过 diff 算法比对 JavaScript 原生对象，计算出需要变更的 DOM，然后只对变化的 DOM 进行操作，而不是更新整个视图

1. **Diff算法**

由于需要对比前后两个虚拟DOM的对象树的不同，两种方法：

二叉树的遍历、**递归逐层比较**

由于递归遍历算法是O(n)级别，选用，并且具体判断算法如下：

1. 从根节点开始到枝叶节点，从左到右遍历其子节点，给每一步的节点一个索引，方便记录渲染差异
2. 一旦节点有子元素，就去判断子元素是否有不同，但是子元素情况很多
   1. 判断标签tagName是否发生变化，不同节点被替换，相同继续判断子节点
   2. 判断节点是否有移除，换位，新增；为了做到这一步，引入了key给每个节点打上标识
   3. 后再判断单个节点的props是否发生变化
3. 记录差异化，局部的渲染更新DOM页面
4. **双向绑定**

利用 Object.defineProperty() 对数据进行劫持，设置一个监听器 Observer ，用来监听所有属性，如果属性上发生变化了，就需要告诉订阅者 Watcher 去更新数据，最后指令解析器 Compile 解析对应的指令，进而会执行对应的更新函数，从而更新视图，实现双向绑定

1. **前端路由**

**前端路由的本质：**

监听 URL 的变化，然后匹配路由规则，显示相应的页面，并且无须刷新页面。目前 前端使用的路由就只有两种实现方式：

**Hash 模式  
 History 模式**

**Hash 模式：**

就是 Hash URL，当 # 后面的哈希值发生变化时，可以通过 hashchange 事件来 监听到 URL 的变化，从而进行DOM卸载更新，并且无论哈希值如何变化，页面都其实没有刷新，浏览器没有历史记录

**History 模式：**

History 模式是 HTML5 新推出的功能，主要使用 history.pushState 和 history.replaceState 改变 URL，通过 History 模式改变 URL 同样不会引起页面的刷新，只会更新浏览器的历史记录；当用户做出浏览器动作时，比如点击后退按钮时会触发 popState 事件

**两种模式的区别：**

1. Hash 模式只可以更改 # 后面的内容，History 模式可以通过 API 设置任意的同源 URL
2. History 模式可以通过 API 添加任意类型的数据到历史记录中，Hash 模式只能更改哈希值，也就 是字符串
3. Hash 模式无需后端配置，并且兼容性好。History 模式在用户手动输入地址或者刷新页面的时候 会发起 URL 请求，后端需要配置 index.html 页面用于匹配不到静态资源的时候