概念

同步|异步

- 同步 sync：按顺序执行，前面的内容执行结束后，才会执行后面的部分
  - 例：js代码阻塞运行
- 异步：不需要等待某一部分执行，可以同时执行
  - 例：有回调函数(事件、定时器)、Ajax

框架与库

- 框架(framework)：

  > 框架规定了自己的编程方式，是一套完整的解决方案，
  >
  > 大部分的逻辑在框架内部已经被确定，
  >
  > 使用时：需要根据规则填充自己的内容，具有一定的限制，但很强大（类似完形填空）

  - 例子：Vue.js
- 库(Library)：

  > 提供了一系列方法的集合，可以调用方法且程序逻辑由自己掌握，而并不是在库中定好的。
  >
  > 本质：一些函数的集合，每次调用函数实现一个特定的功能，只是一个工具
  >
  > 使用时：更自由，可以随意调用或不调用

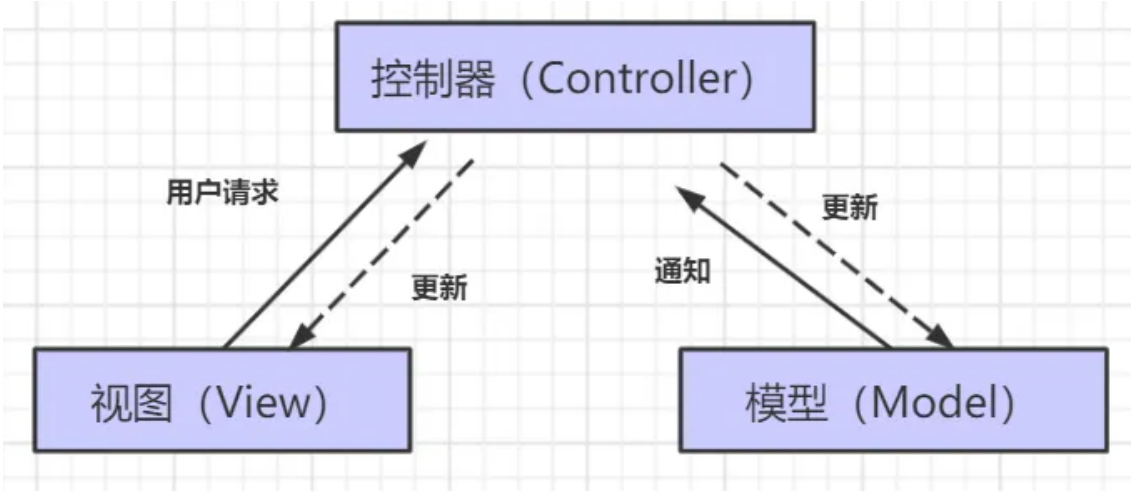
  - 例子：jQuery

MVC与MVVM

- MVC：

  > 在传统的非前后端分离项目中，后端需要处理大量的内容，如果不按照一定的模式就是"大乱炖"
  >
  > 模型 M - 视图(用户界面) V - 控制器 C

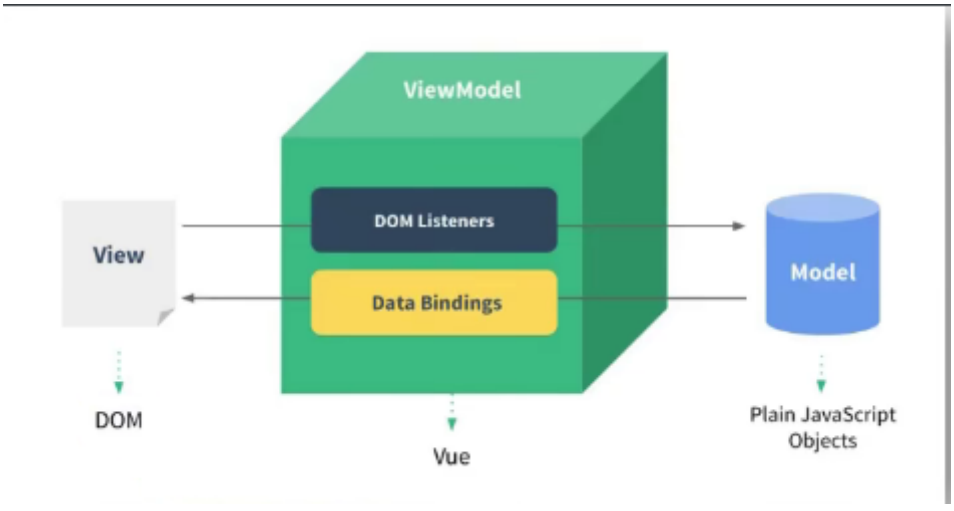
  - C即controller控制器：接受用户的输入并调用模型和视图去完成用户的需求，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。
  - V即View视图：指用户看到并与之交互的界面。比如由html元素组成的网页界面，或者软件的客户端界面。MVC的好处之一在于它能为应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，它只是作为一种输出数据并允许用户操作的方式。
  - M即model模型：指业务规则。在MVC的三个部件中，模型拥有最多的处理任务。被模型返回的数据是中立的，模型与数据格式无关，这样一个模型能为多个视图提供数据，由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。



- MVVM

> 由后端的MVC架构演化而来，传统的MVC并不符合前端的实际需求
>
> 划分代码职责：原本需要发数据、存数据、拼模板、渲染DOM...（大乱炖）
>
> - Model 模型：对应Vue data中的数据
> - View 视图：模板
> - ViewModel 视图模型：vue实例对象， 是View与Model的结合

  - 优点：
    - 将视图 UI 和业务逻辑分开
    - **低耦合。**视图（View）可以独立于Model变化和修改，一个ViewModel可以绑定到不同的"View"上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。
    - **可重用性。**你可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑。
    - 实现数据的双向绑定：
      - 修改数据 M => 视图自动改变
      - 修改视图 V => 数据自动改变



数据代理

- 概念：通过一个对象代理对另一个对象中属性的操作（读/写）
- ```
  // vue2: 借助Object.defineProperty为对象追加属性
  let obj = { x:100 }
  let obj2 = { y:200 }

  //通过obj2中的x对obj中的x进行读写操作
  Object.defineProperty( obj2,'x'{
     get(){
         return obj.x
     },
     set(value){
         obj.x = value
     }
  })
  ```

防抖节流

- 为了限制S频繁的执行一段代码
- 例：scroll事件，只是轻微滚动一下滚动条就触发多次事件，由于过于频繁地DOM操作和资源加载，严重影响了网页性能，甚至会造成浏览器崩溃
- 此时：用 debounce（防抖）和 throttle（节流）的方式来减少调用频率，同时又不影响实际效果
- **防抖**：
  - 连续的多次触发，固定的时间间隔内，存在新的触发，就清除之前的重新记时，满足时间执行一次——**最新一次触发**（只保留 新事件）
  - 手段1：通过设置setTimeout定时器的方式延迟执行，当快速多次点击的时候，每一次都会重置定时器，只有你一段时间都不点击时定时器才能到达条件并执行事件函数。即如果触发事件后在 n 秒内又触发了事件，则会重新计算函数延执行时间。
  - 模拟一个表单提交的例子，多次快速点击提交后只会执行一次
- **节流（节流阀）**：
  - 连续的多次触发，每一段固定的时间间隔内，我们只去执行一次——**固定频率触发**（忽略新产生的同类事件）
  - 确保某个事件，在同一时间只能有一个
  - 如果已经存在，就保留原来的，不再触发
  - 与防抖最大的区别就是，无论事件触发多么频繁，都可以保证在规定时间内可以执行一次执行函数
  - 例：用户在滚动页面时，每隔一段时间发一次 ajax 请求，而不是在用户停下滚动页面操作时才去请求数据

重排(回流)、重绘

> DOM性能 浏览器的性能大部分都是被这两个问题所消耗

- 重绘不一定需要回流（比如颜色的改变），回流必然导致重绘（比如改变网页位置）
- 重绘：元素中的背景、透明度、颜色发生变化后，浏览器针对某一元素进行单独的渲染
  ```

- 重排（回流/重构）：DOM位置、大小或结构、定位发生变化；导致浏览器重新渲染整个DOM树、非常耗性能
    - 添加、删除可见的dom
    - 元素的位置改变
    - 元素的尺寸改变(外边距、内边距、边框厚度、宽高、等几何属性)
    - 页面渲染初始化
    - 浏览器窗口尺寸改变
    - 获取某些属性：offsetTop、offsetLeft、 offsetWidth、offsetHeight、scrollTop、scrollLeft、scrollWidth、scrollHeight、clientTop、clientLeft、clientWidth、clientHeight、getComputedStyle() (currentStyle in IE)。在多次使用这些值时应进行缓存
- 优化：
    - 不要一条一条地修改 DOM 的样式，可以先定义好 css 的 class，然后修改 DOM 的 className
    - 不要把 DOM 结点的属性值放在一个循环里当成循环里的变量
    - 获取浏览器重排DOM节点的属性值，存储到变量中
    - 避免使用 table 布局，很小的改动会造成整个 table 的重新布局

SPA单页应用

> 单页面应用指：只有一个web页面的应用。

- 特点：浏览器一开始直接加载必须的HTML、CSS、JS，所有的操作都在这一个页面上完成，有JavaScript控制交互和局部刷新
- 优点：
    - 有利于前后端分离
    - 良好的用户体验，不刷新界面，显示更流畅
    - 减轻服务器压力，不需要频繁请求界面
- 缺点：
    - 初次加载比较耗时
    - 不利于SEO优化

内存泄露与内存溢出

原生事件机制

1. 到目前为止,一共出现多少种事件机制?
    1. 一共存在3种
    2. 事件捕获机制
    3. 事件冒泡机制
    4. 标准事件机制
2. 标准事件机制,一共分为几个阶段?
    1. 捕获阶段

1. 从最外层的document元素开始向内逐层传递,触发**同类型**事件,直到找到目标元素为止
2. 目标阶段
    1. 捕获阶段结束之后,将目标元素身上所有的**同类型**事件全部触发
3. 冒泡阶段
    1. 目标阶段结束之后,从目标元素开始向外逐层传递,触发**同类型**事件,直到最外层document元素为止

3. 如何绑定事件监听?
    1. 举例:现在需要给div节点,绑定click事件监听
    2. div.onclick=function(){}
        1. 本质:是对div对象的onclick属性进行赋值
        2. 该方法对于每个节点的每个事件,都只能绑定一个回调函数,后绑定的会覆盖之前绑定的
        3. 该方法只能绑定冒泡事件
    3. div.addEventListener('click',function(){})
        1. 本质:是调用div对象身上的addEventListener方法,并传入事件回调函数
        2. 该方法对于每个节点的每个事件,都可以绑定多个回调函数
        3. 第三个实参:
            1. 数据类型:
                1. 布尔值
                    1. true->将当前事件存放于捕获阶段触发
                    2. false->将当前事件存放于冒泡阶段触发
                2. 对象
                    1. capture属性
                        1. true->将当前事件存放于捕获阶段触发
                        2. false->将当前事件存放于冒泡阶段触发
                    2. passive属性
                        1. 前言:部分浏览器存在调用event.preventDefault()方法默认无效的情况
                        2. true->事件回调函数中的event.preventDefault()方法**生效**
                        3. false->事件回调函数中的event.preventDefault()方法**不生效**
    4. 如何阻止事件冒泡?
        1. event.stopPropagation();
        2. event.cancelBubble=true;
        3. 扩展:如何阻止事件捕获?
            1. event.stopPropagation方法在冒泡阶段事件中使用,就是阻止冒泡,捕获阶段事件中使用就是阻止捕获
    5. 问题:请问我们绑定的是事件还是事件的回调函数?
        1. 简单说法:给div绑定click事件
        2. 完整说法:给div绑定click事件的回调函数
        3. 绑定的是事件的回调函数,不是事件,每个标签具有什么事件都是由W3C规范制定的,不是我们绑定的

常识

端口

- 端口范围：0~65535
- 知名端口：0~1024
- 常见端口：80、Mysql3306、mongodb27017