# vue与jQuery

1.vue比jQuery减少了DOM操作（当dom操作影响到布局时浏览器的渲染引擎会重新计算然后渲染，越多的dom操作就会导致越多的计算，会影响页面性能）vue使用虚拟DOM，会尽可能的复用DOM，减少了DOM操作。（可以提一下vue的diff算法）

2.支持数据双向绑定，在MVVM框架中viewmodel作为中介使view和model双向通信。双向数据绑定使数据能够自动更新。

3.vue支持**组件化，组件能够复用**，提高开发效率便于协同开发。

vue的两个核心：组件系统（模板，初始数据data，props，方法methods，生命周期钩子函数等）和数据双向绑定。

# 前后端分离

优点：

1.前端js可以做大部分的数据处理工作，对服务器压力减小到最小。

2.前后端流量大幅减少，因为前端静态化，在第一次获取之后静态资源会被浏览器缓存。前端/web服务器放css js图片等静态资源。

3.前后端工作的分离。前端专注于UI，后端专注于数据层业务处理仅为前端提供接口，可以用任何语言实现只要提供合适的restful接口即可可以为多个客户端使用。前后端可以并行开发，出现bug也可以快速定位。

4.~~安全性提高~~，~~一些注入式攻击在分离模式下被很好的规避~~。后端安全问题集中化处理restful接口上。（安全性能有没有提高不清楚）

5.高并发情况下，可以同时水平扩展前后端服务器。

6.服务器端出现问题只会让前端页面刷不出来数据，页面其实能正常访问的。

|  |  |  |
| --- | --- | --- |
|  | 服务器端渲染 | 客户端渲染 |
| 利 | 1.首屏渲染快，客户端只负责解析HTML  2.有利于搜索引擎优化(SEO)  3.节能 | 1.前后端分离解耦，前端专注于UI，后端专注于业务逻辑。前端/web服务器放css js图片等静态资源，异步调用数据。  2.局部刷新，无需每次请求完整页面，体验更好  3.减少后端服务器的负载压力。  4.交互性好，可以实现各种效果 |
| 弊 | 耦合性太强，服务器压力大，一旦服务器出现问题，前后台都受影响，用户体验差；  不容易维护，前端修改了部分HTML或者CSS后端也要相应修改 | SEO问题  首屏渲染慢（但其实没有慢到无法接受）  耗电 |

开发模式：

|  |  |
| --- | --- |
| 旧 | 新 |
| 1、产品经理/领导/客户提出需求  2、UI做出设计图  3、前端工程师做html页面  4、后端工程师将html页面套成jsp页面（前后端强依赖，后端必须要等前端的html做好才能套jsp。如果html发生变更，就更痛了，开发效率低）  5、集成出现问题  6、前端返工  7、后端返工  8、二次集成  9、集成成功  10、交付 | 1、产品经理/领导/客户提出需求  2、UI做出设计图  3、前后端约定接口&数据&参数  4、前后端并行开发（无强依赖，可前后端并行开发，如果需求变更，只要接口&参数不变，就不用两边都修改代码，开发效率高）  5、前后端集成  6、前端页面调整  7、集成成功  8、交付 |

小问题：

## vue组件中的data为什么必须是函数

因为一个组件是可以共享的，但是它们各自的data是**私有**的，所以每个组件都要return一个新的data对象，返回一个唯一的对象，不能跟其他组件共用一个对象。

# 后端路由、前端路由以及vue中的路由

经典面试题：从浏览器地址栏输入网址到你看到网页的过程中经历了什么？（其实讲的类似于后端路由的过程）

后端路由指的一般是服务端渲染；

前端路由：页面跳转的URL规则匹配由前端来控制。应用：SPA的web项目。（最大优势：无缝跳转？）

hash路由：兼容性高，URL带有#不好看

history路由：不带#好看，但是需要浏览器支持也**需要后端服务器支持**。

只要在浏览器地址栏输入URL再回车，是一定会去后端服务器请求一次的。而如果是在页面里通过点击按钮等操作，利用router库的api来进行的URL更新是不会去后端服务器请求的。

**hash模式：**

**hash模式利用的是浏览器不会对#号后面的路径对服务端发起路由请求。**也即在浏览器里输入如下这两个地址：http://localhost/#/user/1和http://localhost/其实到服务端都是去请求http://localhost这个页面的内容。

而前端的router库通过捕捉#号后面的参数、地址，来告诉前端库（比如Vue）渲染对应的页面。这样，不管是我们在浏览器的地址栏输入，或者是页面里通过router的api进行的跳转，都是一样的跳转逻辑。所以这个模式是不需要后端配置其他逻辑的，只要给前端返回http://localhost对应的html，剩下具体是哪个页面，就由前端路由去判断便可。

**history模式：**

router库要实现这个功能通常是通过H5提供的history这个API，在页面中的跳转通常调用的就是history.pushState()这个api，所以和后端无关（一般页面里面的跳转url都有）。但是**一旦你从浏览器地址栏里输入一个地址，比如**http://localhost/user/1,这个URL是会向后端发起一个get请求的。后端路由表里如果没有配置相应的路由，那么自然就会返回一个404了。

# 移动端布局

有关于移动端适配问题的解决：

1.CSS3的媒体查询：查询设备宽度来执行不同的CSS代码

|  |  |
| --- | --- |
| @media screen and(max-width:600px){} | 当小于等于600px的时候使用下面的CSS样式 |

优点：方法简单，适合响应式布局（）

2.flex弹性布局：（天猫）---在我的音乐播放器中也是采用的这种方式

设置理想视口：

|  |
| --- |
| <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no"> |

高度用像素固定，宽度百分比自适应。也就是类似于PC端的流式布局。

3.rem布局（淘宝）  
html的font-size通过js计算出来，设计稿是基于横向分辨率750px做的，因为美工出图基本上都是基于750px。淘宝设计稿基于iphone6/7/8的横向分辨率做的，html的font-size是100px就是1rem=100px；同理换成plus之后横向分辨率是414相应的font-size: 110.4px;（414\*2/750\*100=110.4px）

这样整个网页在设备内显示时的页面宽度就会等于设备逻辑像素大小，也就是device-width。这个device-width的计算公式为：

设备的物理分辨率/(devicePixelRatio \* scale)，在scale为1的情况下，device-width = 设备的物理分辨率/devicePixelRatio

devicePixelRatio称为设备像素比，每款设备的devicePixelRatio都是已知，并且不变的，目前高清屏，普遍都是2，不过还有更高的，比如2.5, 3 等

4.vw vh视口宽度视口高度

# 各种布局

|  |  |  |  |
| --- | --- | --- | --- |
| 静态布局 | 所有元素一律按px作为定位  布局特点：常规的PC网站都是静态布局，就是设置了min-width，如果小于这个宽度就会出现滚动条，如果大于就内容居中外加背景 | PC端：居中布局，如果屏幕宽高调整则使用横纵滚动条来显示遮掩部分  ~~移动端：通过meta元信息标签设定~~ | 优点：简单，没有兼容性问题  缺点：不能根据用户屏幕作出不同表现。目前大部分门户网站都采用这种布局方式。 |
| 流式布局 | 屏幕分辨率改变时，页面元素大小会变化但是布局不会变。 | 使用%定义宽度，px定义高度 | 因为用百分比定义宽度，像素定义高度导致如果屏幕分辨率跨度大的话会把元素宽度拉长但是高度不变显得不协调 |
| 自适应布局 | 为不同的屏幕分辨率定义各自的静态布局 | 使用CSS3媒体查询 |  |
| 响应式布局 | 确保一个页面在所有终端都显示出令人满意的效果 | 媒体查询＋流式布局 | 响应式和自适应都是检测设备对不同的设备采用不同的css，且多采用百分比但是不同的是响应式布局会随着设备改变改变展示样式。流式布局偏向于解决类似设备不同分辨率之间的兼容。响应式偏向于解决不同设备（PC 手机）之间的兼容。 |
| 弹性布局（rem/em布局） | rem和em都是适应不同网页字体大小展现而产生的。em相对于父元素，rem相对于html页面素 | 包裹文字的各元素的尺寸采用em/rem作单位，但是页面主要划分区域的尺寸仍然使用百分比或者px做单位 | 优点：使用em/rem做单位，可以使包裹文字的元素随着文字的缩放而缩放。  页面中所有元素的高度都等比例缩放 |

# 音乐播放器

1.有关于$nextTick的使用：

vue的核心是数据驱动视图（DOM），一般不去做操作DOM的操作，但是使用第三方基于JavaScript的库的时候难免操作DOM，而vue在一个事件循环中将发生数据改动的事件放入队列中，因为vue总是希望能够避免不必要的计算尽可能的复用已经渲染的元素减少DOM操作，所以在下次事件循环（tick）时将去除重复的数据改动并进行DOM更新渲染，（diff算法的核心），而$nextTick就是帮助我们知道DOM更新的时刻。

在本音乐播放器中，因为要使用滚动的效果，头部有待下载的图片还没有渲染出来，所以用的setTimeout延迟初始化scroll组件，而nextTick可以当promise使用返回promise对象（前提是浏览器支持）（具体参看开发文档）

2.对于生命周期函数的使用：

created 在这个阶段获取的数据通过 写了api文件夹里的js获取数据函数 然后再created里调用函数获取数据。

mounted 使用1中的延迟初始化scroll组件

destoryed 移除了一些定时器。

3.移动端布局：主要是用的width：100%。元素的高度固定。