# HTML

## **一. 什么是meta标签？**

* ****meta 标签提供各种元数据，从而提供各种文档信息****
* ****元数据****就是指****非显示型数据，机器可读型数据****

meta标签有四大属性：

1.****charset属性：**** 用于声明页面的字符编码，一定要写在第一行，否则将会出现乱码

1. ****http-equiv属性：**** 用于添加http头部内容，要与content属性结合使用
2. ****name属性****：常用与解决浏览器兼容问题，要与content属性结合使用
3. ****content属性：**** 通常与有http-equiv或name属性结合使用，为这两种属性加以说明

四大属性的使用原则： 除了charset属性，http-equiv属性和name属性都要结合content属性来使用

2. 实现左侧固定，右侧自适应两栏布局的方法

HTML布局：

<div class="outer">

<div class="sidebar">固定宽度区(sideBar)</div>

<div class="content">自适应区(content)</div>

</div>

<div class="footer">footer</div>复制代码

方法：

1、将左侧div浮动，右侧div设置margin-left

/\*方法1\*/

.outer{overflow: hidden; border: 1px solid red;}

.sidebar{float: left;width:200px;height: 150px; background: #BCE8F1;}

.content{margin-left:200px;height:100px;background: #F0AD4E;}复制代码

1. flex

/\*方法2\*/

.outer7{display: flex; border: 1px solid red;}

.sidebar7{flex:0 0 200px;height:150px;background: #BCE8F1;}

.content7{flex: 1;height:100px;background: #F0AD4E;}复制代码

flex可以说是最好的方案了，代码少，使用简单。但存在兼容性，有朝一日，大家都改用现代浏览器，就可以使用了。

需要注意的是，flex容器的一个默认属性值:align-items: stretch;。这个属性导致了列等高的效果。 为了让两个盒子高度自动，需要设置: align-items: flex-start;

1. float + BFC方法

/\*方法3\*/

.outer6{overflow: auto; border: 1px solid red;}

.sidebar6{float: left;height:150px;background: #BCE8F1;}

.content6{overflow:auto;height:100px;background: #F0AD4E;}复制代码

这个方案同样是利用了左侧浮动，但是右侧盒子通过overflow: auto;形成了BFC，因此右侧盒子不会与浮动的元素重叠。

延伸问题：因为代码中用了flex布局, 问了flex布局相关知识

Flex是Flexible Box的缩写，意为”弹性布局”，用来为盒状模型提供最大的灵活性。 任何一个容器都可以指定为Flex布局。容器分为两种,块flex和行内flex.

.box{

display: flex; /\*webkit需要加前缀\*/

/\*display:inline-flex;\*/

}复制代码

Flex布局有两层,采用flex布局的元素称为flex容器,其子元素则自动成flex item,即项目. 注:flex不同于block,flex容器的子元素的float,clear,vertical-align属性将失效.

# 媒体查询：

# BFC

### **如何创建BFC**

1、float的值不是none。

2、position的值不是static或者relative。

3、display的值是inline-block、table-cell、flex、table-caption或者inline-flex

4、overflow的值不是visible

### **BFC的作用**

###### **利用BFC避免margin重叠。**

###### **自适应两栏布局**

###### **3.清除浮动。**

当我们不给父节点设置高度，子节点设置浮动的时候，会发生高度塌陷，这个时候我们就要清除浮动。

# Js

## es6新增

## let 定义变量，变量不可以再次定义，但可以改变其值。

具有块级作用域。（即大括号）

没有变量提升，必须先定义再使用

let声明的变量不会压到window对象中，是独立的

****如果使用var声明了变量，也不能再次用let声明了，反之也是不行的。原因也是这个 变量已经被声明过了。****

**（2）const**

使用const关键字定义常量

常量是不可变的，一旦定义，则不能修改其值

* 初始化常量时，必须给初始值
* 具有块级作用域
* 没有变量提升，必须先定义再使用
* 常量也是独立的，定义后不会压入到window对象中
* 

## **（3）解构赋值**

## **（4）函数和参数**

## **1.箭头函数**

#### **（5）扩展运算符**

#### **（6）find()方法和findIndex()方法，includes()方法includes()方法**

## **（7）新增对象Set**

# 2.promise作用

## 3. 提交数据两种数据结构

URL传值和form表单提交的区别和原理

区别：

1、url传值就是get ,from表单就是post ，get是从服务器获取数据，post是向服务器传送数据

2、 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。

3、get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。

4、get安全性非常低，post安全性较高

## 4.性能优化

减少 HTTP 请求

使用 HTTP2

* 1. 使用服务端渲染
  2. 静态资源使用 CDN
  3. 将 CSS 放在文件头部，JavaScript 文件放在底部
  4. 使用字体图标 iconfont 代替图片图标
  5. 善用缓存，不重复加载相同的资源
  6. 压缩文件
  7. 图片优化
  8. 通过 webpack 按需加载代码，提取第三库代码，减少 ES6 转为 ES5 的冗余代码
  9. 减少重绘重排
  10. 使用事件委托
  11. 注意程序的局部性
  12. if-else 对比 switch
  13. 查找表
  14. 避免页面卡顿
  15. 使用 requestAnimationFrame 来实现视觉变化
  16. 使用 Web Workers
  17. 使用位操作
  18. 不要覆盖原生方法
  19. 降低 CSS 选择器的复杂性
  20. 使用 flexbox 而不是较早的布局模型
  21. 使用 transform 和 opacity 属性更改来实现动画
  22. 合理使用规则，避免过度优化

#### 何时使用 Map ，何时使用 Object？

当所要存储的是简单数据类型，并且 key 都为字符串或者整数或者 Symbol 的时候，优先使用 Object ，因为Object可以使用 字符变量 的方式创建，更加高效。

当需要在单独的逻辑中访问属性或者元素的时候，应该使用 Object

JSON 直接支持 Object，但不支持 Map

Map 是纯粹的 hash， 而 Object 还存在一些其他内在逻辑，所以在执行 delete 的时候会有性能问题。所以写入删除密集的情况应该使用 Map。Map 会按照插入顺序保持元素的顺序，而Object做不到。

Map 在存储大量元素的时候性能表现更好，特别是在代码执行时不能确定 key 的类型的情况

## new 一个函数发生了什么

构造调用：

* 创造一个全新的对象
* 这个对象会被执行 [[Prototype]] 连接，将这个新对象的 [[Prototype]] 链接到这个构造函数.prototype 所指向的对象
* 这个新对象会绑定到函数调用的 this
* 如果函数没有返回其他对象，那么 new 表达式中的函数调用会自动返回这个新对象

如果函数返回一个对象，那么new 这个函数调用返回这个函数的返回对象，否则返回 new 创建的新对象

## 事件循环机制 （Event Loop）

事件循环机制从整体上告诉了我们 JavaScript 代码的执行顺序 Event Loop即事件循环，是指浏览器或Node的一种解决javaScript单线程运行时不会阻塞的一种机制，也就是我们经常使用异步的原理。 先执行宏任务队列，然后执行微任务队列，然后开始下一轮事件循环，继续先执行宏任务队列，再执行微任务队列。

* 宏任务：script/setTimeout/setInterval/setImmediate/ I/O / UI Rendering
* 微任务：process.nextTick()/Promise

上诉的 setTimeout 和 setInterval 等都是任务源，真正进入任务队列的是他们分发的任务。

#### 优先级

* setTimeout = setInterval 一个队列
* setTimeout > setImmediate
* process.nextTick > Promise

执行微任务过程中产生的新的微任务并不会推迟到下一个循环中执行，而是在当前的循环中继续执行

## 垃圾回收：释放内存，提升浏览器页面性能

#### 1. JavaScript 的内存管理

栈内存中的基本类型，可以通过操作系统直接处理；而堆内存中的引用类型，正是由于可以经常变化，大小不固定，因此需要 JavaScript 的引擎通过垃圾回收机制来处理

#### 2.Chrome 内存回收机制

新生代内存回收: Scavenge 算法

老生代内存回收: Mark-Sweep（标记清除） 和 Mark-Compact（标记整理）

老生代内存的管理方式和新生代的内存管理方式区别还是比较大的。Scavenge 算法比较适合内存较小的情况处理；而对于老生代内存较大、变量较多的时候，还是需要采用“标记-清除”结合“标记-整理”这样的方式处理内存问题，并尽量避免内存碎片的产生

#### 3.内存泄漏与优化

场景1.过多的缓存未释放；

2.闭包太多未释放；

3.定时器或者回调太多未释放；

4.太多无效的 DOM 未释放；

5.全局变量太多未被发现。

## 浅谈前端模块化规范

#### CommonJs和Es Module的区别

1.CommonJs

CommonJs可以动态加载语句，代码发生在运行时

CommonJs混合导出，还是一种语法，只不过不用声明前面对象而已，当我导出引用对象时之前的导出就被覆盖了

CommonJs导出值是拷贝，可以修改导出的值，这在代码出错时，不好排查引起变量污染

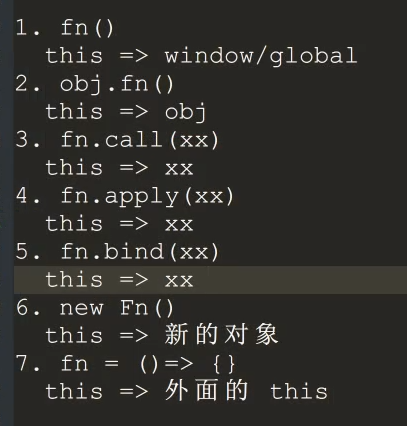
2.Es Module

* Es Module是静态的，不可以动态加载语句，只能声明在该文件的最顶部，代码发生在编译时
* Es Module混合导出，单个导出，默认导出，完全互不影响
* Es Module导出是引用值之前都存在映射关系，并且值都是可读的，不能修改

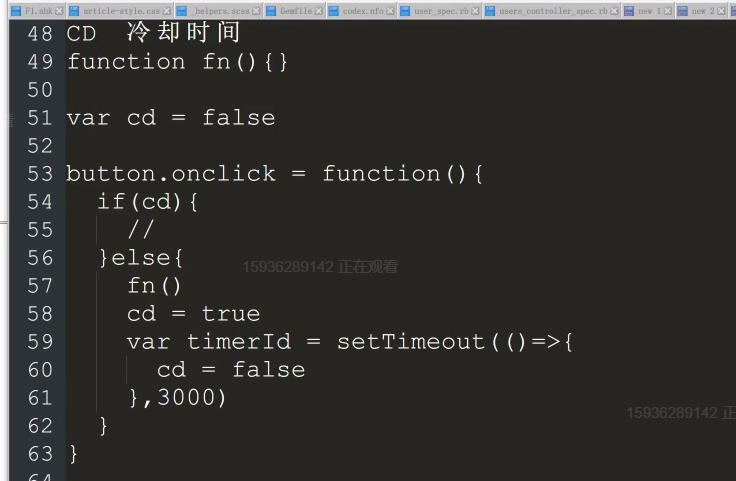
## for in 和 for of的区别， for of怎么遍历对象

* for in适合遍历对象
* for of适合遍历数组

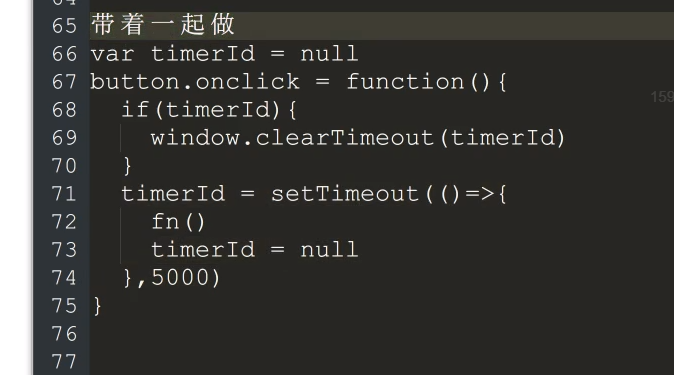
**This指向问题？**



### 节流



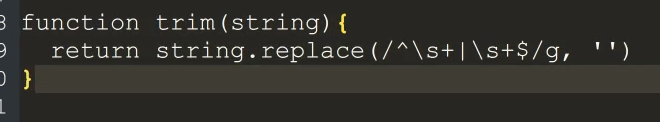
防抖



### 手写ajax



### 正则实现trim（）



# 服务器

## 1.https 原理（加密 证书）

1. 客户端使用https的url访问web服务器，要求与服务器建立ssl连接
2. web服务器收到客户端请求后，会将网站的证书（包含公钥）传送一份给客户端
3. 客户端收到网站证书后会检查证书的颁发机构以及过期时间，如果没有问题就随机产生一个秘钥
4. 客户端利用公钥将会话秘钥加密，并传送给服务端，服务端利用自己的私钥解密出会话秘钥
5. 之后服务器与客户端使用秘钥加密传输

### 2.XSS

XSS: Cross Site Scripting攻击,全称跨站脚本攻击.

XSS指恶意攻击者利用网站没有对用户提交数据进行转义处理或者过滤不足的缺点，进而添加一些 代码，嵌入到web页面中去。使别的用户访问都会执行相应的嵌入代码。 从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式。

XSS攻击的危害包括：

1. 盗取各类用户帐号，如机器登录帐号、用户网银帐号、各类管理员帐号
2. 控制企业数据，包括读取、篡改、添加、删除企业敏感数据的能力

3、盗窃企业重要的具有商业价值的资料

4、非法转账

5、强制发送电子邮件

6、网站挂马

7、控制受害者机器向其它网站发起攻击

## 4.cookie 和 token 都存放在 header 中，为什么不会劫持 token？

1、首先token不是防止XSS的，而是为了防止CSRF的；

2、CSRF攻击的原因是浏览器会自动带上cookie，而浏览器不会自动带上token

# Vue

## 1.Vue 的父组件和子组件生命周期钩子执行顺序是什么

* 1.加载渲染过程

父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount->子mounted->父mounted

* 2.子组件更新过程

父beforeUpdate->子beforeUpdate->子updated->父updated

* 3.父组件更新过程

父beforeUpdate->父updated

* 4.销毁过程

父beforeDestroy->子beforeDestroy->子destroyed->父destroyed

总结：从外到内，再从内到外

## v-for 中 key 的作用是什么？

Key是v-for渲染页面的节点标识。

使用key之后，当页面发生改变时，Vue会根据key的变化，重新渲染页面，并移除key不存在的元素。提升运行效率。

**3.MVVM是什么？**

响应式，双向数据绑定，即MVVM。

是指数据层（Model）-视图层（View）-数据视图（ViewModel）的响应式框架。

它包括：

1.修改View层，Model对应数据发生变化。

2.Model数据变化，不需要查找DOM，直接更新View。

**4. 简述 Vue.js 的优点**

* 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。
* 独立开发。开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。
* 方便测试。界面素来是比较难于测试的，开发中大部分 Bug 来至于逻辑处理，由于 ViewModel 分离了许多逻辑，可以对 ViewModel 构造单元测试。
* 易用 灵活 高效。

1. **vue响应式原理？**

[Object.defineProperty](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty" \t "https://cn.vuejs.org/v2/guide/_blank)

优点

缺点

措施

**6.为何vue采用异步渲染**

因为如果不采用异步更新，那么更新数据都会对当前组件进行重新渲染，

考虑性能问题，数据更新后，再去异步更新视图。

**7.vue.nextTick的原理**

1）所有同步任务都在主线程上执行，形成一个执行栈（execution context stack）。

2）主线程之外，还存在一个"任务队列"（task queue）。只要异步任务有了运行结果，就在"任务队列"之中放置一个事件。

3）一旦"执行栈"中的所有同步任务执行完毕，系统就会读取"任务队列"，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，进入执行栈，开始执行。

4）主线程不断重复上面的第三步。

1. **Vue组件的生命周期？**

## 9.VUE双向数据绑定原理

1.vue 双向数据绑定是通过 数据劫持 结合 发布订阅模式的方式来实现的， 也就是说数据和视图同步，数据发生变化，视图跟着变化，视图变化，数据也随之发生改变；

2.核心：关于VUE双向数据绑定，其核心是 Object.defineProperty  /ˈprɑːpərti/()方法；

3.介绍一下Object.defineProperty()方法

* （1）Object.defineProperty(obj, prop, descriptor) ，这个语法内有三个参数，分别为 obj （要定义其上属性的对象） prop （要定义或修改的属性） descriptor /dɪˈskrɪptər/

（具体的改变方法）

1. 简单地说，就是用这个方法来定义一个值。当调用时我们使用了它里面的get方法，当我们给这个属性赋值时，又用到了它里面的set方法；

## 在 Vue 中，子组件为何不可以修改父组件传递的 Prop，如果修改了，Vue 是如何监控到属性的修改并给出警告的

1.子组件为何不可以修改父组件传递的 Prop

单向数据流，易于监测数据的流动，出现了错误可以更加迅速的定位到错误发生的位置。

1. 如果修改了，Vue 是如何监控到属性的修改并给出警告的

# React

### Component和PureComponent的区别

1. React.PureComponent 与 React.Component 几乎完全相同，但 React.PureComponent 通过props和state的浅对比来实现 shouldComponentUpate()。

PureComponent缺点：

可能会因深层的数据不一致而产生错误的否定判断，从而shouldComponentUpdate结果返回false，界面得不到更新。

PureComponent优点：

不需要开发者自己实现shouldComponentUpdate，就可以进行简单的判断来提升性能。