面试题

CSS

1. css中盒子怎么实现水平居中和垂直居中的?

水平居中:

1. 对于文本,图片 行内元素,在父元素设置 text-align:center
2. 对于块级元素,子元素设置margin:0 auto
3. 父元素display:flex;justify-content:center
4. 子绝父相
   - 子元素有宽度
     - 法一:在子元素设置left:50%;margin-left:-0.5*宽度
     - 法二:left:0;left:0;margin:0 auto
   - 子元素没有宽度,在子元素设置left:50%;transform:translate(-50%,0)

垂直居中:

1. 父元素display:flex;align-items:center
2. 子绝父相
   - 子元素有高度
     - 法一:在子元素设置top:50%;margin-top:-0.5*高度
     - 法二: top:0;bottom:0;margin:auto 0
   - 子元素没有宽度,在子元素设置top:50%;transform:translate(0,-50%)

水平垂直居中 四个方法:

1. 父元素 flex布局,justify-content:center和align-items:center
2. 子绝父相三个.一个transform和top.另外两个要有高度和宽度,上下左右为0,margin:auto auto 还有一个top和left为50%,margin-left和margin-top为-0.5*宽度/高度

2. 怎么实现移动端适配?为什么会自适应?

rem :它是相对于跟元素的font-size

通过媒体查询,每个屏幕的 html的font-size不同,rem也会随之变化.也可以通过js动态 计算.可以直接引入代码

3. 行内元素间隙有3px,为什么会有?怎么去除?

因为代码之间有空格,会被解析成空格

解决方法:

- 给元素加float(最好)
- 注释掉空格
- 元素代码之间不要留空格
- 设置父元素font-size为0,这样空格大小为0.然后在元素中重新设置font-size(safari不适用)

4. 为什么清除浮动?怎么清除浮动?

因为如果不清除浮动的话,浮动层后面的非浮动的内容会被浮动层所覆盖,造成版面错乱,所以必须清除浮动

三种方法去浮动:

1. 将浮动层设置一个固定的高度,这样不管他是否浮动,都有一个固定的区域
2. 在父元素最后加上
3. 父元素添加伪类::after,为了让父元素的后面插入一行,不让下面元素向上移动,将display:block,同时设置高度为0 ,隐藏以来

```
    .box::after {
            content: "";
            clear: both;
            display: block;
            height: 0;
            visibility: hidden;
        }
```

5. 高度坍塌是什么情况?以及如何解决高度塌陷?

高度坍塌指的是 父元素的高度是子元素撑开的,若子元素设置浮动了,则导致子元素无法撑开父元素,父元素的高度坍塌,

造成的后果是:如果父元素高度坍塌,则父元素之下的所有元素都会向上移动,这样会导致页面混乱,不利于页面布局.

解决方法:(也是清浮动,见四)

6. css动画特性可以用js完成,那为什么用CSS实现?

让你的动画在移动设备上移动的更快一点.减少了重绘回流

1.不占用js的主线程.2. 浏览器可以对动画进行优化 3.可以利用硬件加速

7. 怎么实现文本不自动换行,超出部分用省略号替代?

```
            white-space: nowrap;
             overflow: hidden;
             text-overflow: ellipsis;
```

8. css选择器的优先级,子代和后代怎么实现?

为了方便比较css属性的优先级,可以定义一个权重

- !important:10000
- 内联样式:1000
- id选择器:100
- 类选择器,属性选择器,伪类:10
- 元素选择器,伪元素:1
- 通配符:0

  比较优先级的方法: 从权值最大的开始比较每一种权值的数量有多少,数量多的优先级高

后代选择器 div(空格)span 指的是后代中,不管是直接的还是间接的

子代选择器 div > span 指的是直接的后代

9. 左侧固定,右侧滑动怎么实现?

法一:float+bfc,左侧宽度需固定

```
        .container-left {
            float: left;
            width: 100px;
        }

        .container-right {
            overflow: hidden;
        }

        .container::after {
            content: "";
            clear: both;
            height: 0;
            overflow: hidden;
            visibility: hidden;
            display: block;
        }
```

法二:float+margin-left，左侧宽度需固定

将法一的右侧栏的overflow改成 margin-left:100px . 100px是左侧的固定宽度

法三:table-cell，表格布局,左侧宽度不需固定

将父元素的display:table.然后左右的display:table-cell

法四:flex方法,左侧宽度不需固定

父元素display:flex,右边设置flex-grow:1

法五:grid网格,左侧宽度不需固定

父元素display:grid,左边设置 grid-column:1,右边 grid-column:2

10 flex:1表示什么的缩写

flex:flex-grow|flex-shrink|flex-basis

flex:1表示flex-grow:1,如果父元素还有空间,就扩展放大

11.h5新标签有哪些？为什么要加强语义化?

新标签:

文档类型设定：`<!doctype html>`;

字符设定：`<meta charset="utf-8>`;

常用新标签:

`header`，一般作为网页的**头部**使用，可以多个;

`footer`，底部，**不一定是文档最底部**，可以多个;

`aside`，**侧边**栏；

`nav`，**导航**栏；

`article`，独立内容区域，与session类似，**用于文章blog、帖子、短文或者回复、评论**等；

`section`，代表某一个区域/分区/页面/文档的**一部分区域**，有独立的内容，但结构相近，就可以用section，范围比div大，语义比div更强，可以包含header、`h1-h6`.....凸显语义的标签；

新增了许多input的属性：`placeholder`占位符，默认文字、`autofocus`页面加载时自动获得焦点、`multiple`多文件上传、`autocomplete`、`required`必填项、`accesskey`规定激活元素的快捷键

多媒体标签：`embed`定义嵌入的内容、`audio`播放音频、`video`播放视频；

`src`导入，`autoplay`自动播放、`controls`是否默认显示播放件、`loop`循环播放

原因：1.默认样式不一样； 2.有SEO优化作用；

JS

1. js绑定DOM元素方法?三种

1. 直接在html标签内绑定,onclick="alert("hello word")"
2. 在js中获取相应的dom元素后绑定
3. 在js实现addEventListener()实现绑定

2. 数组中遍历的方法

1. for循环
2. for .. in 效率比较低
3. for .. of(ES6)
4. map

```
//  遍历每一个元素并且返回对应的元素(可以返回处理后的元素) (map 映射 一一 对应)
//   返回的新数组和旧数组的长度是一样的,使用比较广泛，但其性能还不如 forEach
var arr = [1, 2, 3, 4, 5, 6]
var newArr = arr.map(function (item, idnex) {
    return item * item
})

console.log(newArr)    // [1, 4, 9, 16, 25, 36]
```

5. foreach

```
//  数组里的元素个数有几个，该方法里的回调就会执行几次
//  第一个参数是数组里的元素，第二个参数为数组里元素的索引，第三个参数则是它自己
var arr = [1, 2, 3, 4, 5, 6]
arr.forEach(function (item, idnex, array) {
    console.log(item)     // 1 2 3 4 5 6
    console.log(array)    // [1, 2, 3, 4, 5, 6]
})
```

3. 数组常用的方法?splice和slice的区别

1. 读取数据for循环

2. 添加push var arr9=arr8.push("k");在结尾添加
3. 删除pop,var arr7=arr6.pop(); 删除数组arr6的最后一个
4. 反转数组reverse,var arr10=arr1.reverse()
5. 数组截取 slice 和 splice

   slice使用的方法:arr.slice(m,n) 其中m,n 为下标,从m开始,n结束,其中不包括n.

   slice使用的方法:arr.splice(m,n,index1..index x),其中m为下标,n为个数,index 是要添加的元素

   在不添加元素的前提下,只有m是必须的.要是添加元素,三个元素都是必须的

4. js执行机制

js是单线程语言,js的eventLoop(事件循环)是js的执行机制.js任务分为同步任务和异步任务

js的执行机制是

- js首先判断是同步任务还是异步任务.同步任务放在主线程执行,异步任务进入event table,
- 异步任务在eventTable中注册回调函数,满足触发条件后进入event queue
- 同步任务进入主线程后,一直被执行,直到主线程空闲时,会去event queue 查看是否有可执行异步任务,如果有,推到主线程执行

所谓"回调函数"（callback），就是那些会被主线程挂起来的代码。异步任务必须指定回调函数，当主线程开始执行异步任务，就是执行对应的回调函数。

5. js垃圾回收机制

js最常见的垃圾回收方式是标记清除

工作原理:当变量进入环境时,将这个变量标记为"进入环境",当它离开后,将它标记为"离开环境",标记"离开环境"的就回收内存.

垃圾收集器在运行的时候会给存储在内存中的所有变量都加上标记。然后，它会去掉环境中的变量以及被环境中的变量引用的标记。而在此之后再被加上标记的变量将被视为准备删除的变量，原因是环境中的变量已经无法访问到这些变量了。最后。垃圾收集器完成内存清除工作，销毁那些带标记的值，并回收他们所占用的内存空间。

还有个回收方式:引用计数

6. 什么是闭包?闭包的优缺点,举例说明

闭包就是能够读取其他函数内部变量的函数

优点:

- 希望一个变量长期存储在内存中。
- 避免全局变量的污染。
- 私有成员的存在。

缺点:

- 常驻内存，增加内存使用量。
- 使用不当会很容易造成内存泄露。

```
function outer() {
  var name = "jack";
  function inner() {
    console.log(name);
  }
  return inner;
```

```
}
outer()(); // jack

function sayHi(name) {
  return () => {
    console.log(`Hi! ${name}`);
  };
}
const test = sayHi("xiaoming");
test(); // Hi! xiaoming
```

虽然 sayHi 函数已经执行完毕，但是其活动对象也不会被销毁，因为 test 函数仍然引用着 sayHi 函数中的变量 name，这就是闭包。
但也因为闭包引用着另一个函数的变量，导致另一个函数已经不使用了也无法销毁，所以闭包使用过多，会占用较多的内存，这也是一个副作用。

解析：

由于在 ECMA2015 中，只有函数才能分割作用域，函数内部可以访问当前作用域的变量，但是外部无法访问函数内部的变量，所以闭包可以理解成“定义在一个函数内部的函数，外部可以通过内部返回的函数访问内部函数的变量“。在本质上，闭包是将函数内部和函数外部连接起来的桥梁。

7. ES6列举常用的,新增有哪些?

1. Promises　解决异步问题
2. 块作用域构造Let and Const
3. 解构赋值

```
var arr=[1,2]

const [a,b]=arr

console.log(a,b)  // 输出 a=1  b=2
```

4. 箭头函数

```
var fn=()=>{

   console.log('11111')
}
```

8. ES6 中var let const区别?

1. var定义的变量，没有块的概念，可以跨块访问, 不能跨函数访问。
2. let定义的变量，只能在块作用域里访问，不能跨块访问，也不能跨函数访问。
3. const用来定义常量，使用时必须初始化(即必须赋值)，只能在块作用域里访问，而且不能修改。

9. const定义的对象是否可以改变?

可以改变.因为对象是引用类型的,P中保存的仅是对象的指针,这就意味着,const仅保证指针不发生改变,修改对象的属性不会改变对象的指针,所以是被允许的。也就是说const定义的引用类型只要指针不发生改变,其他的不论如何改变都是允许的。

10. ES6的promise有哪两个参数?

resolve rejected

11. get 和 post 的区别?

- GET 参数通过 url 传递，POST 放在 body 中。（http 协议规定，url 在请求头中，所以大小限制很小）
- GET 请求在 url 中传递的参数是有长度限制的，而 POST 没有。原因见上↑↑↑
- GET 在浏览器回退时是无害的，而 POST 会再次提交请求
- GET 请求会被浏览器主动 cache，而 POST 不会，除非手动设置
- GET 比 POST 更不安全，因为参数直接暴露在 url 中，所以不能用来传递敏感信息
- 对参数的数据类型，GET 只接受 ASCII字符，而 POST 没有限制
- GET 请求只能进行 url(x-www-form-urlencoded)编码，而 POST 支持多种编码方式
- **GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包。**对于 GET 方式的请求，浏览器会把 http 的 header 和 data 一并发送出去，服务器响应200（返回数据）。而对于 POST，浏览器先发送 header，服务器响应100 continue，浏览器再发送 data，服务器响应200 ok（返回数据）

12.js数据类型有哪些?如何判断数据类型?

JavaScript中有8种数据类型, 包括基本数据类型(number, string, boolean, null, undefined, symbol, bigint)和引用数据类型object

typeof 和instance of

13. js数组如何去重?

1.ES6 的 Set

```
let arr = [1,1,2,3,4,5,5,6]
let arr2 = [...new Set(arr)]
```

2.reduce()

```
let arr = [1,1,2,3,4,5,5,6]
let arr2 = arr.reduce(function(ar,cur) {
  if(!ar.includes(cur)) {
    ar.push(cur)
  }
  return ar
},[])
```

3.filter()

```
// 这种方法会有一个问题：[1,'1']会被当做相同元素，最终输入[1]
let arr = [1,1,2,3,4,5,5,6]
let arr2 = arr.filter(function(item,index) {
  // indexOf() 方法可返回某个指定的 字符串值 在字符串中首次出现的位置
  return arr.indexOf(item) === index
})
```

14. 深拷贝和浅拷贝是什么?怎么进行深拷贝?

深拷贝和浅拷贝最根本的区别在于是否是真正获取了一个对象的复制实体，而不是引用，
深拷贝在计算机中开辟了一块内存地址用于存放复制的对象，而浅拷贝仅仅是指向被拷贝的内存地址，如果原地址中对象被改变了，那么浅拷贝出来的对象也会相应改变。

JSON.parse(JSON.stringify())

递归方法:

```
function cloneObject(obj) {
  var newObj = {} //如果不是引用类型，直接返回
  if (typeof obj !== 'object') {
    return obj
  }
  //如果是引用类型，遍历属性
  else {
    for (var attr in obj) {
      //如果某个属性还是引用类型，递归调用
      newObj[attr] = cloneObject(obj[attr])
    }
  }
  return newObj
}
```

15.new date()获取的是哪里的时间?

获取的是本机时间,系统时间

16. promise await async 如何使用?如果按序分别请求三个数据,一个请求完后再请求另一个,怎么实现?

async 和 await 用了同步的方式去做异步，async 定义的函数的返回值都是 promise，await 后面的函数会先执行一遍，然后就会跳出整个 async 函数来执行后面js栈的代码

```
async function A(){}
async function B(){}
function C(){}
Promise.all([A(),B()]).then(C)
```

或者

```
const A = async () =>  await 'A';
const B = async () =>  await 'B';
const C = () =>  'C';
(async function All() {
    await Promise.all([A(), B()]);
    C();
})();
```

或者

```
var promise1 = new Promise((resolve, reject) => {
    setTimeout(() => {
        console.log(1);
        resolve()
    }, 0);
});
var promise2 = new Promise((resolve, reject) => {
    setTimeout(() => {
        console.log(2);
```

```
        resolve()
    }, 0);
});
Promise.all([promise1, promise2]).then(function(res) {
  console.log(3)
});
```

或者

```
var index = 0
function C(){
    console.log(3);
}
setTimeout(() => {
    console.log(1);
    index++;
    if(index === 2){
        C()
    }
}, 0);
setTimeout(() => {
    console.log(2);
    index++;
    if(index === 2){
        C()
    }
}, 0);
```

17. eval函数是什么?是否有安全性问题?效率问题?

eval通常用在一些需要动态执行字符串，或将字符串转为javascript对象的场景，比如将json字符串转为javascript对象。

未写完

18. 原生js继承怎么实现?prototype怎么实现

1. **原型继承**核心就是让自定义的构造器的prototype对象**指向**父类构造器生成的对象：

```
//Person是个构造函数，Per是自定义的构造器
function Per(){}
Per.prototype = new Person('Alice');
const person = new Per()//继承父类实例定义的所有属性以及父类构造器原型上的属性
```

2. **借用函数继承**：通过函数对象本身的 `call` 和 `apply` 来显示的指定函数调用时必备的参数；

```
function Per(name){
    Person.call(this,name)//当成了普通函数来使用
}
const person = new Per('Alice')
//只能调用Person中定义的属性和函数，无法调用Person定义在prototype上的属性和方法
```

19. js三大事件冒泡、捕获是怎么执行的

事件冒泡的概念下在p元素上发生click事件的顺序应该是**p -> div -> body -> html -> document**；

事件捕获的概念下在p元素上发生click事件的顺序应该是**document -> html -> body -> div -> p**；

接下来是摘自《红宝书》的总结：

事件流描述的是从页面中接收事件的顺序。P348

IE的事件流叫做**事件冒泡**：就是指事件开始时由最具体的元素（文档中嵌套层次最深的的那个节点）接收，然后逐级向上传播到较为不具体的节点（文档）。

div–>body–>html–>document

事件捕获：不太具体的节点应该更早接收到事件，而最具体的节点应该最后接收到事件。（用意在于在事件到达预定目标之前捕获它）

document–>html–>body–>div

20. js中的this的指向问题,如何改变this的指向?

this的最终指向的是那个**调用它的对象。**

改变this指向的方法：

1. 使用**箭头函数**；

> 箭头函数中的this指向**定义时**当前周围的作用域；

1. 在函数内部使用`_this=this`;
2. 使用**apply、call、bind**

> `call`是立即执行传递this的；`bind`不是立即执行的，返回的是函数的副本；

1. new**实例化一个对象**；

在非严格模式下，如果如果函数没有用作构造函数，而是仅仅作为普通函数使用的话，那么函数中的this是指向window的。在严格模式下，this的值就是undefined。

21. for ...in和for .. of的区别

for...in循环

- 性能：

  效率最低，不推荐用于数组遍历，一般用于对象循环

  ```
  for(j in arr) {}
  ```

for...of循环

- 性能：性能好于for...in，但仍**低于普通for循环**

  不仅可以遍历数组，还可以遍历Map、Set这两种ES6新推出的数据结构。

22. 轮播图实现的时候是怎么考虑的?

四种方式实现：

swiper插件实现轮播图；

JS实现轮播图；

jQuery实现轮播图；

css3实现轮播图；

23.箭头函数和一般函数有什么区别?

定义箭头函在数语法上要比普通函数简洁得多。箭头函数省去了`function`关键字，采用箭头`=>`来定义函数。

基本语法：关于箭头函数的参数：

① 如果箭头函数**没有参数**，直接写一个**空括号**即可。

② 如果箭头函数的**参数只有一个**，也可以**省去包裹参数的括号。**

③ 如果箭头函数有多个参数，将参数依次用逗号(,)分隔，包裹在括号中即可。

基本语法：关于箭头函数的函数体：

① 如果箭头函数的函数体只有一句代码，就是简单返回某个变量或者返回一个简单的JS表达式，可以省去函数体的大括号{ }。

② 如果箭头函数的函数体只有一句代码，就是返回一个对象，用小括号包裹要返回的对象，不报错

```
let getTempItem = id => ({ id: id, name: "Temp" });
```

③ 如果箭头函数的函数体只有一条语句并且不需要返回值（最常见是调用一个函数），可以给这条语句前面加一个`void`关键字

```
let fn = () => void doesNotReturn();
//用来简化回调函数：
[1,2,3].map(function (x) {return x * x;});// 正常函数写法
[1,2,3].map(x => x * x);// 箭头函数写法
var result = [2, 5, 1, 4, 3].sort(function (a, b) {  return a - b;});// 正常函数写法
var result = [2, 5, 1, 4, 3].sort((a, b) => a - b);// 箭头函数写法
```

区别:

1、语法更加简洁、清晰

2、箭头函数不会创建自己的this，它会捕获自己在**定义时**（注意，是定义时，不是调用时）所处的**外层执行环境的this**，并继承这个`this`值。所以，箭头函数中`this`的指向在它**被定义的时候就已经确定**了，之后永远不会改变。

3、箭头函数继承而来的**this指向永远不变**

4、.call()/.apply()/.bind()**无法改变箭头函数中this的指向**（但是也不会报错）

5、箭头函数不能作为构造函数使用：因为箭头函数没有自己的`this`，它的`this`其实是继承了外层执行环境中的`this`，且`this`指向永远不会随在哪里调用、被谁调用而改变，所以箭头函数不能作为构造函数使用，或者说构造函数不能定义成箭头函数，否则用`new`调用时会报错!

6、箭头函数没有自己的arguments：在箭头函数中访问`arguments`实际上获得的是外层局部（函数）执行环境中的值。**可以在箭头函数中使用rest参数代替arguments对象，来访问箭头函数的参数列表**

7、箭头函数没有原型prototype

```
let sayHi = () => {    console.log('Hello World !')};
console.log(sayHi.prototype); // undefined
```

HTTP 浏览器

1. https和http的区别? S代表什么?

HTTPS 与 HTTP 相比

- HTTPS协议需要到CA申请证书，一般免费证书很少，需要交费。
- HTTP协议运行在TCP之上，所有传输的内容都是明文，HTTPS运行在SSL/TLS之上，SSL/TLS运行在TCP之上，所有传输的内容都经过加密的。
- HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
- HTTPS可以有效的防止运营商劫持，解决了防劫持的一个大问题。

HTTPS 介绍：HTTPS在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。TLS/SSL协议不仅仅是一套加密传输的协议，TLS/SSL中使用了非对称加密，对称加密以及HASH算法。

HTTP 协议中的内容都是明文传输，HTTPS 的目的是将这些内容加密，确保信息传输安全。最后一个字母 S 指的是 SSL/TLS 协议，它位于 HTTP 协议与 TCP/IP 协议中间。

2. cookie和session的区别?

1. 存在的位置：
   cookie 存在于客户端，临时文件夹中；session 存在于服务器的内存中，一个 session 域对象为一个用户浏览器服务
2. 安全性
   cookie 是以明文的方式存放在客户端的，安全性低，可以通过一个加密算法进行加密后存放；
   session 存放于服务器的内存中，所以安全性好
3. 生命周期(以 20 分钟为例)
   cookie 的生命周期是累计的，从创建时，就开始计时，20 分钟后 cookie 生命周期结束；
   session 的生命周期是间隔的，从创建时，开始计时如在 20 分钟，没有访问 session，那么 session 生命周期被销毁。但是，如果在 20 分钟内（如在第 19 分钟时）访问过 session，那么，将重新计算 session 的生命周期。关机会造成 session 生命周期的结束，但是对 cookie 没有影响
4. 访问范围
   cookie 为多个用户浏览器共享；session 为一个用户浏览器独享

3.TCP三次握手和四次挥手的过程?

4.当输入url后发生什么?

1. DNS 域名解析（域名解析成ip地址，走UTP协议，因此不会有握手过程）：浏览器将 URL 解析出相对应的服务器的 IP 地址（1. 本地浏览器的 DNS 缓存中查找 2. 再向系统DNS缓存发送查询请求 3. 再向路由器DNS缓存 4. 网络运营商DNS缓存 5. 递归搜索），并从 url 中解析出端口号
2. 浏览器与目标服务器建立一条 TCP 连接（三次握手）
3. 浏览器向服务器发送一条 HTTP 请求报文
4. 服务器返回给浏览器一条 HTTP 响应报文
5. 浏览器进行渲染
6. 关闭 TCP 连接（四次挥手）

5.浏览器的本地存储有哪些?

cookie ,localstorage sessionstorage indexedDB

6. 怎么实现跨域?jsonp的原理

实现跨域有几种方法：

1) JSONP

原理:利用script src 标签没有跨域限制请求资源的特点,网页可以得到从其他来源动态获得JSON数据.JSONP一定需要对方服务器做支持才可以

2) CORS

原理:根据浏览器需要遵循同源策略(即协议,域名,端口号相同),浏览器会自动进行CORS通信.此时需要服务器端 设置"Access-control-allow-origin",开启CORS,就可实现跨域.该属性可以表示哪些域名可以访问资源,如果设置通配符则表示所有网站都可以访问资源.

3)node

原理:是浏览器需要遵循同源策略,服务器向服务器请求,则不需要.所以根据node写一个中间件代理服务器.代理服务器需要做四个步骤:

- 接受客户端请求
- 将请求转发给服务器
- 拿到服务器响应数据
- 将响应转发给浏览器

4) nginx(反向代理)

原理:也是设置服务器,需要进行Linux部署,它是反向代理

5) POSTMessage

postmessage 采用不同源的脚本采取异步方式进行有限的通信,可以实现跨域信息传递.它是为数不多的跨域操作的window属性之一.

适用于 多窗口信息传递,页面与其他窗口的数据传递,页面与嵌套的iframe信息传递

6) Websocket

原理:websocket协议本质上是一个基于TCP协议的协议,为了建立websocket连接,浏览器向服务器发送HTTP请求,这个请求与普通的HTTP请求不同,它包含了一些附加头信息,其中附加头信息"upgrade:websocket",表明这是一个申请协议升级的HTTP请求.然后服务器解析附加头信息并产生应答请求返回给浏览器,客户端和服务器之间就建立websocket连接.可以在这个连接里自由的传递消息

7.状态码

2xx（成功）表示成功处理了请求的状态码

3xx（重定向）表示要完成请求，需要进一步操作；通常，这些状态代码用来重定向

4xx（请求错误）这些状态码表示请求可能出错，妨碍了服务器的处理

5xx（服务器错误）这些状态码表示服务器在尝试处理请求时发生内部错误。这些错误可能是服务器本身的错误，而不是请求出错

8. 如何实现前端性能优化? 如果打开网页 加载过慢,如何解决?

1. 减少请求数量
2. 减小资源大小
3. 优化网络连接
4. 优化资源加载
5. 减少重绘回流
6. 性能更好的API
7. webpack优化

9. seo搜索引擎优化?

1.创建唯一且准确的**网页标题title**

```
<title>前端搜索引擎优化的技巧</title>
```

2.使用 **的 keywords 元数据**来提炼网页重要关键字，以及 `description` 元数据准确总结网页内容。

```
<meta name='keywords' content='SEO,title,meta,语义化,alt'>
<meta name='description' content='介绍搜索引擎优化的技巧:语义化标签、img的alt属性等。'>
```

3.使用**语义化元素**： em或strong

4.利用**img中的`alt`属性**

5.设置`rel='nofollow'`忽略跟踪

6.尽量让结构（HTML）、表现（CSS）及行为（JavaScript）**三者分离。**如果在一个 HTML 页面中，编写大量的 CSS 样式或脚本，会拖慢其加载速度，此外，如果不为 `` 定义宽高，那么会引起页面重新渲染，同样也会影响加载速度。一旦加载超时，“蜘蛛”就会放弃爬取。如果这个 HTML 文档内容比较独特丰富（合理插入图片说明）等，会被认为质量较高符合用户需求，从而提高 SEO 的排名。

VUE

1.vue中v-for中的key是什么作用?若没有key,会发生什么?

当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用“**就地复用**”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序， 而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。**key 的作用主要是为了高效的更新虚拟DOM。**

没有key,控制台会出现警告

2. v-if和v-show的区别

v-if和 v-show指令都可以控制一个元素的显示和隐藏

如果v-if和v-show的值同时为假时，v-if和v-show绑定的元素都不会在页面中显示，

对于v-show而言，当取值为假时，生成了这个元素，但却将这个元素通过 display:none 的方式隐藏了这个元素，所以在页面显示的时候就不显示

对于v-if而言，当取值为假时，不会生成这个元素，自然而然的也就不会在页面中显示

频繁的需要隐藏和出现就用v-show

3.vue兄弟组件,父子组件是怎么通信的?

1. 父传子： 先在**父组件中绑定变量``**，parent是定义在父组件中的变量/值； 再在**子组件中添加props属性接收**父组件传递过来的变量`props:['msg']`； 最后就可以在子组件中使用``来表示父组件中parent变量中的值了。
2. 子传父： 先在子组件中**绑定事件**`@change="sendChild"`，触发的时候在`setChild`事件中**用`$emit()`触发父组件中的函数**，并将子组件中的变量作为参数传递；

```
methods：{
    sendChild:function(){
        this.$emit('transparent',this.msg)
    }
}
```

在父组件中绑定事件``，当子组件触发这个事件的时候，就可以**调用getChild方法获取到传递过来的参数**；

```
methods:{
    getChild(msg){
        this.user=msg;
    }
}
```

1.兄弟组件互相传值，**通过Vuex状态管理传值**： 先通过npm加载vuex，创建store.js文件

```
//store.js
   import Vue from 'vue'
   import Vuex from 'vuex'
   Vue.use(Vuex);
   const state={name:'Alice'};
   const mutations={
       newName(state,message){
           state.name=message
       }
   }
   export default new Vuex.Store({state,mutations})
```

2.兄弟组件互相传值，引入bus.js文件，发布者订阅者模式：

```
import Bus from './bus.js'
//一个子组件触发
methods:{
  Bus.$emit('触发的方法名', 需要传递的值);
}
//一个子组件监听
   mounted:{
       bus.$on("方法名",(传递的值)=>{ })
   }
```

3.兄弟组件互相传值`$root`

```
//一个子组件触发
this.$root.$emit('触发的方法名', 需要传递的值);
//一个子组件监听
this.$root.$off("方法名");//每次进入先关闭一下
this.$root.$on("方法名",(传递的值)=>{ })
```

4. vue 生命周期有哪些?分别有什么作用?什么时候用created和mouted?

beforeCreate（初始化界面前）created（初始化界面后）

beforeMount（渲染dom前）mounted（渲染dom后）

beforeUpdate（更新数据前）updated（更新数据后）

beforeDestroy（卸载组件前）destroyed（卸载组件后）

`beforeCreate`：初始化了部分参数，如果有相同的参数，做了参数合并，执行 `beforeCreate`；

`created`：初始化了 `Inject`、`Provide`、`props`、`methods`、`data`、`computed` 和 `watch`，执行 `created`；

`beforeMount`：检查是否存在 `el` 属性，存在的话进行渲染 `dom` 操作，执行 `beforeMount`；

`mounted`：实例化 `Watcher`，渲染 `dom`，执行 `mounted`；

`beforeUpdate`：在渲染 `dom` 后，执行了 `mounted` 钩子后，在数据更新的时候，执行 `beforeUpdate`；

`updated`：检查当前的 `watcher` 列表中，是否存在当前要更新数据的 `watcher`，如果存在就执行 `updated`；

`beforeDestroy`：检查是否已经被卸载，如果已经被卸载，就直接 `return` 出去，否则执行 `beforeDestroy`；

`destroyed`：把所有有关自己痕迹的地方，都给删除掉；

created和mounted之间的区别：

- created和mounted中**ajax请求**的区别：created的时候视图未出现，请求较多的情况下，会出现**白屏**；
- created 是在模板渲染成html前调用，即通常初始化某些属性值，然后再渲染成视图，比如初始化、获取屏幕高度调整、赋值等等；

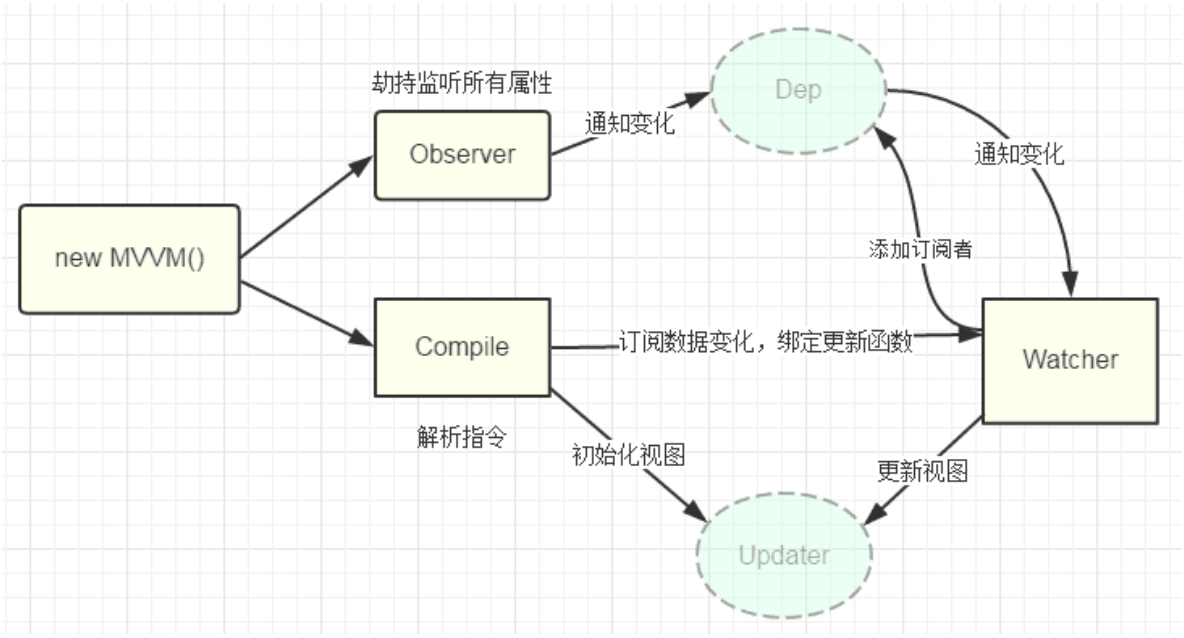
  而mounted是在模板渲染成html后调用，通常是初始化页面完成后，再对html的dom节点进行一些需要的操作；

5. vue中的data为什么是函数?

为了保证组件的独立性 和 可 复用性，data 是一个函数，组件实例化的时候这个函数将会被调用，返回一个对象，计算机会给这个对象分配一个内存地址，你实例化几次，就分配几个内存地址，他们的地址都不一样，所以每个组件中的数据不会相互干扰，改变其中一个组件的状态，其它组件不变。

6. vue双向绑定的原理? vue 数据劫持

mvvm 双向绑定，采用**数据劫持结合发布者-订阅者模式**的方式，通过 `Object.defineProperty()` 来劫持各个属性的 setter、getter，在数据变动时发布消息给订阅者，触发相应的监听回调。



具体步骤：

1. 需要 observe 的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter 和 getter 这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化
2. compile 解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图
3. Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁，主要做的事情是:
    - 在自身实例化时往属性订阅器(dep)里面添加自己
    - 自身必须有一个 update() 方法
    - 待属性变动 dep.notice() 通知时，能调用自身的 update() 方法，并触发 Compile 中绑定的回调，则功成身退。
4. MVVM 作为数据绑定的入口，整合 Observer、Compile 和 Watcher 三者，通过Observer来监听自己的 model 数据变化，通过 Compile 来解析编译模板指令，最终利用 Watcher 搭起 Observer 和 Compile 之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据 model 变更的双向绑定效果。

7. keep-alive是什么?有什么作用?

keep-alive用来缓存组件,避免多次加载相应的组件,减少性能消耗，简单一点来说就是从页面1链接到其他页面后回退到页面1不用在重新执行页面1的代码，只会从缓存中加载之前已经缓存的页面1，这样可以减少加载时间及性能消耗，提高用户体验性。

通过设置了keep-alive，可以简单理解为从页面1跳转到页面2后，然后后退到页面1，只会加载缓存中之前已经渲染好的页面1，而不会再次重新加载页面1，及不会再触发页面一种的created等类似的钩子函数，除非自己重新刷新该页面1。

8. 多页应用与单页应用的区别？以及优缺点

多页应用：

- 定义：每一次页面的跳转，后端都会返回一个新的html文件。

优点：1. **首屏事件快**（只请求 html 文件就可以展示页面了，只经历了一个http请求）；2. SEO（**搜索引擎优化**，可以识别 html 内的内容）效果好。

缺点：页面切换**慢**（每次跳转都要发送一个http请求）

单页应用：

- 定义：每一次页面的跳转，都是使用`JS`渲染。页面跳转不使用 a 标签，而是使用，不请求html文件，通过`JS`感知到`URL`的变化，可以用`JS`动态得把当前页面清除掉，再把下一个页面挂载到当前页面上。

优点：页面切换**快。**不需要http请求。

缺点：首屏时间稍慢（除了请求一个html文件，还要请求一个 JS 文件）；SEO差（不认识JS文件中的内容）

9. vue的computed 和 watch的区别以及应用场景?mouted里存放什么?

computed 计算属性	**watch 观察的动作**	**methods**
1. 数据会被缓存，只要依赖不发生改变，即使页面重新渲染，该方法也不会被调用 2.computed中的函数必须用return返回	1. 直接监测一个值的变化，监测值不发生变化，该方法就不会调用； 2. watch只会监听数据的值是否发生改变，而不会去监听数据的地址是否发生改变。也就是说，watch想要监听引用类型数据的变化，需要进行深度监听。 3.watch中的函数有两个参数，前者是newVal，后者是oldVal。	每次页面发生渲染，都会被重新调用
在computed中不要对data中的属性进行赋值操作。如果对data中的属性进行赋值操作了，就是data中的属性发生改变，从而触发computed中的函数，形成死循环了。	监听复杂数据类型需用深度监听（在被监听对象中使用`handler`）；特殊情况下，watch无法监听到数组的变化，特殊情况就是说更改数组中的数据时，数组已经更改，但是视图没有更新。**更改数组必须要用splice()或者$set。**	
使用场景：**当一个值受多个属性影响的时候————购物车商品结算**	使用场景：**当一条数据的更改影响到多条数据的时候———搜索框**	

computed和watch区别：

1、 功能上：computed是计算属性，也就是依赖其它的属性计算所得出最后的值，是用于定义**基于数据之上的数据。**watch是监听一个值的变化，然后执行对应的回调，是在**某个数据变化时做一些事情。**

2、 **是否调用缓存**：computed中的函数所依赖的属性没有发生变化，那么调用当前的函数的时候会从缓存中读取，而watch在每次监听的值发生变化的时候都会执行回调。

3、 **是否调用return**：computed中的函数必须要用return返回，watch中的函数不是必须要用return。

4、 如果一个值依赖多个属性（多对一），用`computed`肯定是更加方便的。如果一个值变化后会引起一系列操作，或者一个值变化会引起一系列值的变化（一对多），用`watch`更加方便一些。

10. vuex的作用?

vuex专为 Vue.js 应用程序开发的**状态管理模式。它采用集中式存储管理应用的所有组件的状态**，并以相应的规则保证状态以一种可预测的方式发生变化。

主要用于管理vue中的数据，可以兄弟组件互相传值；

11. vue中插槽的作用?

插槽就是Vue实现的一套**内容分发的API**，将``元素作为承载分发内容的出口，没有插槽的情况下在组件标签内些一些内容是不起任何作用的。

插槽内可以是任意内容。在`你好`内放置一些内容，输出内容还是在组件中的内容，直接在父组件的`标签中定义的内容不会被渲染。**在子组件template中加入**`元素占位，便能渲染父组件`**标签下的内容。**

具名插槽，当需要多个插槽时，可以使用``的特性：name。这个特性可以用来定义额外的插槽。

插槽默认内容，插槽可以提供一个默认内容，如果如果父组件没有为这个插槽提供了内容，会显示默认的内容。如果父组件为这个插槽提供了内容，则默认的内容会被替换掉。

作用域插槽，作用域插槽就是父组件在**调用子组件的时候给子组件传了一个插槽**，这个插槽为作用域插槽，该插槽必须放在template标签里面，同时声明从子组件接收的数据放在一个自定义属性内，并定义该数据的渲染方式。