1. 浏览器是怎样解析CSS选择器的?

> 解析:
>
> CSS选择器的解析是从右向左解析的。若从左向右的匹配，发现不符合规则，需要进行回溯，会损失很多性能。若从右向左匹配，先找到所有的最右节点，对于每一个节点，向上寻找其父节点直到找到根元素或满足条件的匹配规则，则结束这个分支的遍历。两种匹配规则的性能差别很大，是因为从右向左的匹配在第一步就筛选掉了大量的不符合条件的最右节点（叶子节点），而从左向右的匹配规则的性能都浪费在了失败的查找上面。 而在 CSS 解析完毕后，需要将解析的结果与 DOM Tree 的内容一起进行分析建立一棵 Render Tree，最终用来进行绘图。在建立 Render Tree 时（WebKit 中的「Attachment」过程），浏览器就要为每个 DOM Tree 中的元素根据 CSS 的解析结果（Style Rules）来确定生成怎样的 Render Tree。

2. 使用Promise实现红绿灯交替重复亮

红灯3秒亮一次，黄灯2秒亮一次，绿灯1秒亮一次；如何让三个灯不断交替重复亮灯？（用Promise实现）

三个亮灯函数已经存在：

```
function red() {
    console.log('red');
}
function green() {
    console.log('green');
}
function yellow() {
    console.log('yellow');
}
```

> 解析:
>
> 红灯3秒亮一次，绿灯1秒亮一次 ，黄灯2秒亮一次，意思就是3秒执行一次red函数，2秒执行一次green函数，1秒执行一次yellow函数，不断交替重复亮灯，意思就是按照这个顺序一直执行这3个函数，这步可以利用递归来实现。

```
function red() {
console.log('red');
}
function green() {
console.log('green');
}
function yellow() {
console.log('yellow');
}

var light = function (timmer, cb) {
return new Promise(function (resolve, reject) {
  setTimeout(function () {
      cb();
      resolve();
  }, timmer);
});
};
```

```
var step = function () {
Promise.resolve().then(function () {
  return light(3000, red);
}).then(function () {
  return light(2000, green);
}).then(function () {
  return light(1000, yellow);
}).then(function () {
  step();
});
}

step();
```

3. 为什么在 JavaScript中比较两个相似的对象时返回 false?

```
let a = { a: 1 };
let b = { a: 1 };
let c = a;
console.log(a === b); // 打印 false，即使它们有相同的属性
console.log(a == b);  // false
console.log(a === c); // true
```

> 解析：
>
> JavaScript 以不同的方式比较对象和基本类型。在基本类型中，JavaScript 通过值对它们进行比较，而在对象中，JavaScript 通过引用或存储变量的内存中的地址对它们进行比较。这就是为什么第一个和第二个 `console.log` 语句返回 `false`，而第二个 `console.log` 语句返回 `true`。`a` 和 `c` 有相同的引用地址，而 `a` 和 `b` 没有。

4. 为什么在调用这个函数时，代码中的 b 会变成一个全局变量?

```
function myFunc() {
  let a = b = 0;
}
myFunc();
```

> 解析：
>
> 原因是赋值运算符是从右到左的求值的。这意味着当多个赋值运算符出现在一个表达式中时，它们是从右向左求值的。所以上面代码变成了这样：
>
> ```
> function myFunc() {
> let a = (b = 0);
> }
> myFunc();
> ```
>
> 首先，表达式 `b = 0` 求值，在本例中 `b` 没有声明。因此，JavaScript 引擎在这个函数外创建了一个全局变量 `b`，之后表达式 `b = 0` 的返回值为 `0`，并赋给新的局部变量 `a`。
>
> 可以通过在赋值之前先声明变量来解决这个问题。

```
function myFunc() {
let a,b;
a = b = 0;
}
myFunc();
```

5. var,let和const的区别是什么?

解析:

(1) var声明的变量会挂载在window上，而let和const声明的变量不会:

```
var a = 100;
console.log(a,window.a);    // 100 100

let b = 10;
console.log(b,window.b);    // 10 undefined

const c = 1;
console.log(c,window.c);    // 1 undefined
```

(2) var声明变量存在变量提升，let和const不存在变量提升:

```
console.log(a); // undefined  ===>  a已声明还没赋值，默认得到undefined值
var a = 100;

console.log(b);
// 报错：Cannot access 'b' before initialization
// => 无法在初始化之前访问“b”
let b = 10;

console.log(c);
// 报错：Cannot access 'c' before initialization
const c = 10;
```

(3) let和const声明形成块作用域

```
if(1){
var a = 100;
let b = 10;
}

console.log(a); // 100
console.log(b)  // 报错：b is not defined  ===> 找不到b这个变量

-------------------------------------------------------------

if(1){
var a = 100;
const c = 1;
}
console.log(a); // 100
console.log(c)  // 报错：c is not defined  ===> 找不到c这个变量
```

(4) 同一作用域下let和const不能声明同名变量，而var可以

```
var a = 100;
console.log(a); // 100

var a = 10;
console.log(a); // 10
----------------------------------
let a = 100;
let a = 10;
//  控制台报错：Identifier 'a' has already been declared  ===> 标识符a已经被声明了。
```

(5) 暂存死区

```
var a = 100;

if(1){
a = 10;
// 在当前块作用域中存在a使用let/const声明的情况下，给a赋值10时，只会在当前作用域找变量a，
// 而这时，还未到声明时候，所以控制台Cannot access 'a' before initialization
let a = 1;
}
```

(6) const

```
/*
* 1、一旦声明必须赋值,不能使用null占位。
* 2、声明后不能再修改
* 3、如果声明的是复合类型数据，可以修改其属性
*
*/

const a = 100;

const list = [];
list[0] = 10;
console.log(list);   // [10]

const obj = {a:100};
obj.name = 'apple';
obj.a = 10000;
console.log(obj);  // {a:10000,name:'apple'}
```

6. 什么时候不使用箭头函数? 说出三个或更多的例子?

解析:

(1) 当想要函数被提升时(箭头函数是匿名的)

(2) 要在函数中使用`this/arguments`时，由于箭头函数本身不具有`this/arguments`，因此它们取决于外部上下文

(3) 使用命名函数(箭头函数是匿名的)

(4) 使用函数作为构造函数时(箭头函数没有构造函数)

（5）当想在对象字面是以将函数作为属性添加并在其中使用对象时，因为咱们无法访问 `this` 即对象本身。

7. Object.freeze() 和 const 的区别是什么?

解析：

`const` 和 `Object.freeze` 是两个完全不同的概念。

`const` 声明一个只读的变量，一旦声明，常量的值就不可改变：

```
const person = {
name: "Leonardo"
};
let animal = {
species: "snake"
};
person = animal; // ERROR "person" is read-only
```

`Object.freeze` 适用于值，更具体地说，适用于对象值，它使对象不可变，即不能更改其属性。

```
let person = {
name: "Leonardo"
};
let animal = {
species: "snake"
};
Object.freeze(person);
person.name = "Lima"; //TypeError: Cannot assign to read only property
'name' of object
console.log(person);
```

8. 如何在 JS 中创建对象?

解析：

（1）使用对象字面量：

```
let obj = {
    name:"张三",
}
console.log(obj);         // {name: "张三"}
```

（2）使用构造函数：

```
let obj = new Object();
obj.name = "张三";
console.log(obj);         // {name: "张三"}
```

（3）使用 Object.create 方法：

```
let obj = Object.create({
name:"张三",
});
console.log(obj.name);    // 张三
```

9. disabled 和 readonly 的区别?

> 解析:
>
> disabled 指当 input 元素加载时禁用此元素。input 内容不会随着表单提交。
>
> readonly 规定输入字段为只读。input 内容会随着表单提交。
>
> 无论设置 readonly 还是 disabled，通过 js 脚本都能更改 input 的 value。

10. {} 和 [] 的 valueOf 和 toString 的结果是什么?

> 解析:
>
> {} 的 valueOf 结果为 {}，toString 的结果为 "[object Object]"
>
> [] 的 valueOf 结果为 []，toString 的结果为 ""

11. webpack3和webpack4区别

1. mode

   webpack增加了一个mode配置，只有两种值development | production。对不同的环境他会启用不同的配置。

2. CommonsChunkPlugin

   CommonChunksPlugin已经从webpack4中移除。 可使用optimization.splitChunks进行模块划分（提取公用代码）。 但是需要注意一个问题，默认配置只会对异步请求的模块进行提取拆分，如果要对entry进行拆分 需要设置optimization.splitChunks.chunks = 'all'。

3. webpack4使用MiniCssExtractPlugin取代ExtractTextWebpackPlugin。

4. 代码分割。

   使用动态import，而不是用system.import或者require.ensure

5. vue-loader。

   使用vue-loader插件为.vue文件中的各部分使用相对应的loader，比如css-loader等

6. UglifyJsPlugin

   现在也不需要使用这个plugin了，只需要使用optimization.minimize为true就行，production mode下面自动为true

   optimization.minimizer可以配置你自己的压缩程序

---

12. 跨域

因为浏览器出于安全考虑，有同源策略。也就是说，如果协议、域名或者端口有一个不同就是跨域，Ajax 请求会失败。防止CSRF攻击。 1.JSONP JSONP 的原理很简单，就是利用 标签没有跨域限制的漏洞。 通过 标签指向一个需要访问的地址并提供一个回调函数来接收数据当需要通讯时。

```
<script>
    function jsonp(data) {
        console.log(data)
    }
</script>
```

JSONP 使用简单且兼容性不错，但是只限于 get 请求。

2. CORS CORS 需要浏览器和后端同时支持。IE 8 和 9 需要通过 XDomainRequest 来实现。
3. document.domain 该方式只能用于二级域名相同的情况下，比如 a.test.com 和 b.test.com 适用于该方式。

   只需要给页面添加 document.domain = 'test.com' 表示二级域名都相同就可以实现跨域
4. webpack配置proxyTable设置开发环境跨域
5. nginx代理跨域
6. iframe跨域
7. postMessage 这种方式通常用于获取嵌入页面中的第三方页面数据。一个页面发送消息，另一个页面判断来源并接收消息

---

13. 前端性能优化方案

三个方面来说明前端性能优化 一： webapck优化与开启gzip压缩 1.babel-loader用 include 或 exclude 来帮我们避免不必要的转译，不转译node_moudules中的js文件 其次在缓存当前转译的js文件，设置 loader: 'babel-loader?cacheDirectory=true' 2.文件采用按需加载等等 3.具体的做法非常简单，只需要你在你的 request headers 中加上这么一句： accept-encoding:gzip 4.图片优化，采用svg图片或者字体图标 5.浏览器缓存机制，它又分为强缓存和协商缓存 二：本地存储——从 Cookie 到 Web Storage、IndexedDB 说明一下SessionStorage和localStorage还有cookie的区别和优缺点 三：代码优化 1.事件代理 2.事件的节流和防抖 3.页面的回流和重绘 4.EventLoop事件循环机制 5.代码优化等等

---

14. 浏览器缓存

缓存可以减少网络 IO 消耗，提高访问速度。浏览器缓存是一种操作简单、效果显著的前端性能优化手段 很多时候，大家倾向于将浏览器缓存简单地理解为“HTTP 缓存”。 但事实上，浏览器缓存机制有四个方面，它们按照获取资源时请求的优先级依次排列如下：

- Memory Cache
- Service Worker Cache
- HTTP Cache
- Push Cache

缓存它又分为强缓存和协商缓存。优先级较高的是强缓存，在命中强缓存失败的情况下，才会走协商缓存

- 实现强缓存：过去我们一直用 expires。当服务器返回响应时，在 Response Headers 中将过期时间写入 expires 字段，现在一般使用Cache-Control 两者同时出现使用Cache-Control
- 协商缓存：Last-Modified 是一个时间戳，如果我们启用了协商缓存，它会在首次请求时随着 Response Headers 返回：每次请求去判断这个时间戳是否发生变化。从而去决定你是304读取缓存还是给你返回最新的数据。

---

15. IOS手机浏览器字体齿轮

```
修改-webkit-font-smoothing属性，结果是：

-webkit-font-smoothing: none: 无抗锯齿
-webkit-font-smoothing: antialiased | subpixel-antialiased | default: 灰度平滑
```

---

16. 优雅降级和渐进增强

渐进增强 progressive enhancement： 针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。 优雅降级 graceful degradation： 一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。 区别： + a. 优雅降级是从复杂的现状开始，并试图减少用户体验的供给 + b. 渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要 + c. 降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带

---

17. websocket和ajax轮询

- websocket是html5中提出的新的协议，可以实现客户端与服务器的通信，实现服务器的推送功能
- 优点是，只要简历一次连接，就可以连续不断的得到服务器推送消息，节省带宽和服务器端的压力。
- ajax轮询模拟常连接就是每隔一段时间（0.5s）就向服务器发起ajax请求，查询服务器是否有数据更新
- 缺点就是，每次都要建立HTTP连接，即使需要传输的数据非常少，浪费带宽

---

18. js垃圾回收方法

标记清除（mark and sweep）

- 这是JavaScript最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”。
- 垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

引用计数(reference counting)

- 在低版本IE中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个 变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加1，如果该变量的值变成了另外一个，则这个值得引用次数减1，当这个值的引用次数变为0的时 候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为0的值占用的空间。
- 在IE中虽然JavaScript对象通过标记清除的方式进行垃圾回收，但BOM与DOM对象却是通过引用计数回收垃圾的， 也就是说只要涉及BOM及DOM就会出现循环引用问题。

---

19. 对webpack的看法

- webpack是一个模块打包工具，可以使用webpack管理你的模块依赖，并编译输出模块们所需要的静态文件。能很好的管理、打包web开发中所用到的HTML、js、css以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源，webpack有对应的模块加载器。webpack模块打包器会分析模块间的依赖关系，最后生成了优化且合并后的静态资源。
- webpack两大特色：
  - code splitting（可以自动完成）
  - loader 可以处理各种类型的静态文件，并且支持串联操作 webpack是以commonJS的形式来书写脚本，但是AMD/CMD的支持也很全面，方便旧项目进行代码迁移
- webpck具有require和browserify的功能，但仍有很多自己的新特性：
  - 对 CommonJS、AMD、ES6的语法做了兼容
  - 对JS、css、图片等资源文件都支持打包

- 串联式模块化加载器以及插件机制，让其具有更好的灵活性和扩展性，例如提供对conffeescript、ES6的支持
- 有独立的配置文件webpck.config.js
- 可以将代码切割成不同的chunk，实现按需加载，降低了初始化时间
- 支持sourceUrls和sourceMaps，易于调试
- 具有强大的plugin接口，大多是内部插件，使用起来比较灵活
- webpack使用异步IO并具有多级缓存，在增亮编译上更加快

---

20. 移动端性能优化

- 尽量使用css3动画，开启硬件加速
- 适当使用touch时间代替click时间
- 避免使用css3渐变阴影效果
- 可以用transform: translateZ(0) 来开启硬件加速
- 不滥用float。float在渲染时计算量比较大，尽量减少使用
- 不滥用web字体。web字体需要下载，解析，重绘当前页面
- 合理使用requestAnimationFrame动画代替setTimeout
- css中的属性（css3 transitions、css3 3D transforms、opacity、webGL、video）会触发GUP渲染，耗电

21. 请实现一个方法将data结构转换为tree结构。

```
let data = [
  {"parent_id": null, "id": 'a', 'value': 'xxxx'},
  {"parent_id": 'a', "id": 'c', 'value': 'xxxx'},
  {"parent_id": 'd', "id": 'f', 'value': 'xxxx'},
  {"parent_id": 'c', "id": 'e', 'value': 'xxxx'},
  {"parent_id": 'b', "id": 'd', 'value': 'xxxx'},
  {"parent_id": 'a', "id": 'b', 'value': 'xxxx'},
];
let tree = {
  'a': {
    value: 'xxx',
    children: {
      'b': {
        value: 'xxx',
        children: {
          'd': {
            value: 'xxx',
            children: {
              'f': {
                value: 'xxx'
              }
            }
          }
        }
      },
      'c': {
        value: 'xxx',
        children: {
          'e': {
            value: 'xxx'
```

```
        }
      }
    }
  }
};
```

参考答案：

```
function transformToTree(data) {
        let result = {};
        let newData;
        data.sort((a, b) => a.id.codePointAt() - b.id.codePointAt());
        newData = data.filter(item => !item.parent_id);
        newData.forEach(item => {
            result[item.id] = {
                value: item.value
            };
        });
        function loop(result) {
            for (const key in result) {
                let newData = data.filter(item => item.parent_id === key);
                if (newData.length) {
                    newData.forEach(item => {
                        result[key].children = result[key].children || {};
                        result[key].children[item.id] = {
                            value: item.value
                        };
                        loop(result[key].children);
                    });
                }
            }
        }
        loop(result);
        return result;
    }
```

22. 下面代码的输出结果是什么?

```
function fn(n, o) {
  console.log(o);
  return {
    fn: function (m) {
      return fn(m, n);
    }
  }
}
fn(0).fn(1).fn(2).fn(3); // =>?
```

解析：

这块连续调用.fn只是一个迷惑作用，其实可以看做打印值都是上一次调用的传参，第一次调用没有第二个参数，所以首先打印一个undefined，之后每次打印都是前一次的参数，所以输出结果应该是undefined 0 1 2。

23. TCP的三次握手和四次挥手

解析：

> 三次握手

- 第一次握手：客户端发送一个SYN码给服务器，要求建立数据连接；
- 第二次握手： 服务器SYN和自己处理一个SYN（标志）；叫SYN+ACK（确认包）；发送给客户端，可以建立连接
- 第三次握手： 客户端再次发送ACK向服务器，服务器验证ACK没有问题，则建立起连接；

> 四次挥手

- 第一次挥手： 客户端发送FIN(结束)报文，通知服务器数据已经传输完毕；
- 第二次挥手: 服务器接收到之后，通知客户端我收到了SYN,发送ACK(确认)给客户端，数据还没有传输完成
- 第三次挥手： 服务器已经传输完毕，再次发送FIN通知客户端，数据已经传输完毕
- 第四次挥手： 客户端再次发送ACK,进入TIME_WAIT状态；服务器和客户端关闭连接；

---

24. 原型

解析：

所有的函数数据类型都天生自带一个prototype属性，该属性的属性值是一个对象 prototype的属性值中天生自带一个constructor属性，其constructor属性值指向当前原型所属的类 所有的对象数据类型，都天生自带一个*proto*属性，该属性的属性值指向当前实例所属类的原型

---

25. V-model的原理是什么?

解析：

Vue的双向数据绑定是由数据劫持结合发布者订阅者实现的。 数据劫持是通过Object.defineProperty()来劫持对象数据的setter和getter操作。 在数据变动时作你想做的事

- 原理 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 在初始化vue实例时，遍历data这个对象，给每一个键值对利用Object.definedProperty对data的键值对新增get和set方法，利用了事件监听DOM的机制，让视图去改变数据

---

26. vuex的流程

解析：

页面通过mapAction异步提交事件到action。action通过commit把对应参数同步提交到mutation。mutation会修改state中对于的值。 最后通过getter把对应值跑出去，在页面的计算属性中 通过mapGetter来动态获取state中的值

---

27. `$route`和`$router`的区别

解析：

- $route是“路由信息对象”，包括path，params，hash，query，fullPath，matched，name等路由信息参数。

- $router是“路由实例”对象包括了路由的跳转方法，钩子函数等。

---

28. react和vue的区别

解析：

=> 相同点： 1.数据驱动页面，提供响应式的试图组件 2.都有virtual DOM,组件化的开发，通过props参数进行父子之间组件传递数据，都实现了webComponents规范 3.数据流动单向，都支持服务器的渲染SSR 4.都有支持native的方法，react有React native， vue有wexx

=> 不同点： 1.数据绑定：Vue实现了双向的数据绑定，react数据流动是单向的 2.数据渲染：大规模的数据渲染，react更快 3.使用场景：React配合Redux架构适合大规模多人协作复杂项目，Vue适合小快的项目 4.开发风格：react推荐做法jsx + inline style把html和css都写在js了 vue是采用webpack + vue-loader单文件组件格式，html, js, css同一个文件

---

29. 移动端300ms延迟

解析：

300毫米延迟解决的是双击缩放。双击缩放，手指在屏幕快速点击两次。safari浏览器就会将网页缩放值原始比例。由于用户可以双击缩放或者是滚动的操作， 当用户点击屏幕一次之后，浏览器并不会判断用户确实要打开至这个链接，还是想要进行双击操作 因此，safair浏览器就会等待300ms，用来判断用户是否在次点击了屏幕 解决方案：1.禁用缩放，设置meta标签 user-scalable=no 2.fastclick.js 原理：FastClick的实现原理是在检查到touchend事件的时候，会通过dom自定义事件立即 发出click事件，并把浏览器在300ms之后真正的click事件阻止掉 fastclick.js还可以解决穿透问题

---

30. CDN（内容分发网络）

解析：

1.尽可能的避开互联网有可能影响数据传输速度和稳定性的瓶颈和环节。使内容传输的更快更稳定。 2.关键技术：内容存储和分发技术中 3.基本原理：广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对的地区或者网络中。当用户访问网络时利用全局负载技术 将用户的访问指向距离最近的缓存服务器，由缓存服务器直接相应用户的请求（全局负载技术）

41、vuex中state,getter,mutation,action,module,plugins各自的用途，和用法?

解析:

- State:{ count: 0 } 保存着所有的全局变量
- Getter： 对state中的数据派生出一些状态，例如对数据进行过滤。（可以认为是store中的计算属性），会对state中的变量进行过滤再保存，只要state中的变量发生了改变，它也会发生变化，不变化的时候，读的缓存。
- Mutation：更改 Vuex 的 store 中的状态的唯一方法是提交 mutation。
- 一条重要的原则就是要记住 mutation 必须是同步函数。
- Action: Action 类似于 mutation, 不同点在于，Action 提交的是 mutation，而不是直接变更状态。Action 可以包含任意异步操作，mutation只能是同步。
- 有点不同的是Action 函数接受一个与 store 实例具有相同方法和属性的 context 对象，因此你可以调用 context.commit 提交一个 mutation，或者通过 context.state 和 context.getters 来获取 state 和 getters。

- Module： //模块，可以写很多模块，最后都引入到一个文件。分散管理。生成实例的时候 都放在Store的modules中
- plugins：插件（Plugins）是用来拓展webpack功能的，它们会在整个构建过程中生效，执行相关的任务。

42、Vue 2.0不再支持在v-html中使用过滤器怎么办?

解析:

①全局方法（推荐）

```
Vue.prototype.msg = function（msg）{
  return msg.replace（"\n", "<br>"）
}
 <div v-html="msg(content)"></div>
```

②computed方法

```
computed: {
 content: function(msg){
  return msg.replace("\n", "<br>")
 }
}
<div>{{content}}</div>
```

③$options.filters(推荐)

```
filters: {
 msg: function(msg){
  return msg.replace(/\n/g, "<br>")
 }
},
data: {
 content: "XXXX"
}
<div v-html="$options.filters.msg(content)"></div>
```

43、怎么解决vue动态设置img的src不生效的问题?

解析:

```
<img class="logo" :src="logo" alt="公司logo">
  data() {
    return {
      logo:require("./../assets/images/logo.png"),
    };
  }
//因为动态添加src被当做静态资源处理了，没有进行编译，所以要加上require
```

44、vue中怎么重置data?

解析:

Object.assign() Object.assign（）方法用于将所有可枚举属性的值从一个或多个源对象复制到目标对象。它将返回目标对象。

```
var o1 = { a: 1 };
var o2 = { b: 2 };
var o3 = { c: 3 };
var obj = Object.assign(o1, o2, o3);
console.log(obj); // { a: 1, b: 2, c: 3 }
console.log(o1); // { a: 1, b: 2, c: 3 }, 注意目标对象自身也会改变。
```

注意：具有相同属性的对象，同名属性，后边的会覆盖前边的。

由于Object.assign()有上述特性，所以我们在Vue中可以这样使用：

Vue组件可能会有这样的需求：在某种情况下，需要重置Vue组件的data数据。此时，我们可以通过this.获取当前状态下的，通过options.data()获取该组件初始状态下的data。

然后只要使用Object.assign(this.options.data())就可以将当前状态的data重置为初始状态。

45、vue怎么实现强制刷新组件?

解析：

① v-if

```
当v-if的值发生变化时，组件都会被重新渲染一遍。因此，利用v-if指令的特性，可以达到强制
<comp v-if="update"></comp>
<button @click="reload()">刷新comp组件</button>
data() {
  return {
    update: true
  }
},
methods: {
  reload() {
     // 移除组件
    this.update = false
      // 在组件移除后，重新渲染组件
      // this.$nextTick可实现在DOM 状态更新后，执行传入的方法。
    this.$nextTick(() => {
      this.update = true
    })
  }
}
```

② this.$forceUpdate

```
<button @click="reload()">刷新当前组件</button>
methods: {
  reload() {
    this.$forceUpdate()
  }
}
```

46、vuex组件中访问state报错怎么解决?

解析：

TypeError: Cannot read property ‘state’ of undefined”

在组件中使用this.$store.state.test访问state的属性报错，是因为store的实例并未注入到所有的子组件，需修改main.js

```
new Vue({

    el: '#app',
    store, //将store注入到子组件
    router,
    components: { App },
    template: ''

})
```

47、Vue 的父组件和子组件生命周期钩子函数执行顺序?

解析:

Vue 的父组件和子组件生命周期钩子函数执行顺序可以归类为以下 4 部分:

- 加载渲染过程 : 父 beforeCreate -> 父 created -> 父 beforeMount -> 子 beforeCreate -> 子 created -> 子 beforeMount -> 子 mounted -> 父 mounted
- 子组件更新过程 : 父 beforeUpdate -> 子 beforeUpdate -> 子 updated -> 父 updated
- 父组件更新过程 : 父 beforeUpdate -> 父 updated
- 销毁过程 : 父 beforeDestroy -> 子 beforeDestroy -> 子 destroyed -> 父 destroyed

48、虚拟 DOM 的优缺点?

解析:

- 优点:

保证性能下限： 框架的虚拟 DOM 需要适配任何上层 API 可能产生的操作，它的一些 DOM 操作的实现必须是普适的，所以它的性能并不是最优的；但是比起粗暴的 DOM 操作性能要好很多，因此框架的虚拟 DOM 至少可以保证在你不需要手动优化的情况下，依然可以提供还不错的性能，即保证性能的下限； 无需手动操作 DOM： 我们不再需要手动去操作 DOM，只需要写好 View-Model 的代码逻辑，框架会根据虚拟 DOM 和 数据双向绑定，帮我们以可预期的方式更新视图，极大提高我们的开发效率；跨平台： 虚拟 DOM 本质上是 JavaScript 对象,而 DOM 与平台强相关，相比之下虚拟 DOM 可以进行更方便地跨平台操作，例如服务器渲染、weex 开发等等。

- 缺点:

无法进行极致优化： 虽然虚拟 DOM + 合理的优化，足以应对绝大部分应用的性能需求，但在一些性能要求极高的应用中虚拟 DOM 无法进行针对性的极致优化。

49、TCP 传输的三次握手、四次挥手策略

解析:

- 三次握手： 为了准确无误地吧数据送达目标处，TCP协议采用了三次握手策略。用TCP协议把数据包送出去后，TCP不会对传送后的情况置之不理，他一定会向对方确认是否送达，握手过程中使用TCP的标志：SYN和ACK
- - 发送端首先发送一个带SYN的标志的数据包给对方
- - 接收端收到后，回传一个带有SYN/ACK标志的数据包 以示传达确认信息
- - 最后，发送端再回传一个带ACK的标志的数据包，代表“握手”结束
- 如在握手过程中某个阶段莫明中断，TCP协议会再次以相同的顺序发送相同的数据包

- 断开一个TCP连接需要“四次挥手”：
- 第一次挥手：主动关闭方发送一个FIN，用来关注主动方到被动关闭方的数据传送，也即是主动关闭方告诫被动关闭方：我已经不会再给你发数据了（在FIN包之前发送的数据，如果没有收到对应的ACK确认报文，主动关闭方依然会重发这些数据）。但是，此时主动关闭方还可以接受数据
- 第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号收到序号 +1（与SYN相同，一个 FIN占用一个序号）
- 第三次挥手：被动关闭方发送一个 FIN。用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会给你发送数据了
- 第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手

50、浏览器是如何渲染页面的?

解析：

渲染的流程如下： 1.解析HTML文件，创建DOM树。

自上而下，遇到任何样式（link、style）与脚本（script）都会阻塞（外部样式不阻塞后续外部脚本的加载）。

2.解析CSS。优先级：浏览器默认设置<用户设置<外部样式<内联样式<HTML中的style样式；

3.将CSS与DOM合并，构建渲染树（Render Tree）

4.布局和绘制，重绘（repaint）和重排（reflow）

60、calc, support, media各自的含义及用法?

> @support主要是用于检测浏览器是否支持CSS的某个属性，其实就是条件判断，如果支持某个属性，你可以写一套样式，如果不支持某个属性，你也可以提供另外一套样式作为替补。
>
> calc() 函数用于动态计算长度值。 calc()函数支持 "+", "-", "*", "/" 运算；
>
> @media 查询，你可以针对不同的媒体类型定义不同的样式。

61、1rem、1em、1vh、1px各自代表的含义?

> **rem**：是全部的长度都相对于根元素元素。通常做法是给html元素设置一个字体大小，然后其他元素的长度单位就为rem。

> **em:**
>
> - 子元素字体大小的em是相对于父元素字体大小
> - 元素的width/height/padding/margin用em的话是相对于该元素的font-size

> **vw/vh :**
>
> 全称是 Viewport Width 和 Viewport Height，视窗的宽度和高度，相当于 屏幕宽度和高度的1%，不过，处理宽度的时候%单位更合适，处理高度的 话 vh 单位更好。

> **px:**
>
> px像素（Pixel）。相对长度单位。像素px是相对于显示器屏幕分辨率而言的。
>
> 一般电脑的分辨率有{1920*1024}等不同的分辨率

62、怎么画一条0.5px的直线?

```
height: 1px;
transform: scale(0.5);
```

63、target、currentTarget的区别?

> currentTarget当前所绑定事件的元素
>
> target当前被点击的元素

64、Cookie如何防范XSS攻击?

> XSS（跨站脚本攻击）是指攻击者在返回的HTML中嵌入javascript脚本，为了减轻这些攻击，需要在HTTP头部配上，set-cookie：
>
> httponly-这个属性可以防止XSS,它会禁止javascript脚本来访问cookie。
>
> secure - 这个属性告诉浏览器仅在请求为https的时候发送cookie

65、简单说一下webpack 的原理

> 初始化参数：从配置文件和 Shell 语句中读取与合并参数，得出最终的参数；
>
> 开始编译：用上一步得到的参数初始化 Compiler 对象，加载所有配置的插件，执行对象的 run 方法开始执行编译；
>
> 确定入口：根据配置中的 entry 找出所有的入口文件；
>
> 编译模块：从入口文件出发，调用所有配置的 Loader 对模块进行翻译，再找出该模块依赖的模块，再递归本步骤直到所有入口依赖的文件都经过了本步骤的处理；
>
> 完成模块编译：在经过第4步使用 Loader 翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系；
>
> 输出资源：根据入口和模块之间的依赖关系，组装成一个个包含多个模块的 Chunk，再把每个 Chunk 转换成一个单独的文件加入到输出列表，这步是可以修改输出内容的最后机会；
>
> 输出完成：在确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统。

66、 简单说一下babel的原理

> babel的转译过程分为三个阶段：**parsing、transforming、generating**，以ES6代码转译为ES5代码为例，babel转译的具体过程如下：
>
> 1. ES6代码输入
> 2. babylon 进行解析得到 AST
> 3. plugin 用 babel-traverse 对 AST 树进行遍历转译,得到新的AST树
> 4. 用 babel-generator 通过 AST 树生成 ES5 代码

67、web上传漏洞原理？如何进行？防御手段?

> 如何进行：用户上传了一个可执行的脚本文件，并通过此脚本文件获得了执行服务器端命令的能力。

主要原理：当文件上传时没有对文件的格式和上传用户做验证，导致任意用户可以上传任意文件，那么这就是一个上传漏洞。

防御手段：

1、最有效的，将文件上传目录直接设置为不可执行，对于Linux而言，撤销其目录的'x'权限；实际中很多大型网站的上传应用都会放置在独立的存储上作为静态文件处理，一是方便使用缓存加速降低能耗，二是杜绝了脚本执行的可能性；

2、文件类型检查：强烈推荐白名单方式，结合MIME Type、后缀检查等方式；此外对于图片的处理可以使用压缩函数或resize函数，处理图片的同时破坏其包含的HTML代码；

3、使用随机数改写文件名和文件路径，使得用户不能轻易访问自己上传的文件；

4、单独设置文件服务器的域名

68、请介绍一下回流（Reflow）与重绘（Repaint）

回流：当我们对 DOM 的修改引发了 DOM 几何尺寸的变化（比如修改元素的宽、高或隐藏元素等）时，浏览器需要重新计算元素的几何属性（其他元素的几何属性和位置也会因此受到影响），然后再将计算的结果绘制出来。这个过程就是回流（也叫重排）。

重绘：当我们对 DOM 的修改导致了样式的变化、却并未影响其几何属性（比如修改了颜色或背景色）时，浏览器不需重新计算元素的几何属性、直接为该元素绘制新的样式（跳过了上图所示的回流环节）。这个过程叫做重绘。

重绘不一定导致回流，回流一定会导致重绘

硬要比较的话，回流比重绘做的事情更多，带来的开销也更大。但这两个说到底都是吃性能的，所以都不是什么善茬。我们在开发中，要从代码层面出发，尽可能把回流和重绘的次数最小化

69、什么是闭包，如何使用它，为什么要使用它?

闭包就是能够读取其他函数内部变量的函数。由于在Javascript语言中，只有函数内部的子函数才能读取局部变量，因此可以把闭包简单理解成“定义在一个函数内部的函数”。

它的最大用处有两个，一个是可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。

使用闭包的注意点：·由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在IE中可能导致内存泄露。

解决方法是，在退出函数之前，将不使用的局部变量全部删除。·

闭包会在父函数外部，改变父函数内部变量的值。所以，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），把内部变量当作它的私有属性（private value），这时一定要小心，不要随便改变父函数内部变量的值。

70、如何规避javascript多人开发函数重名问题?

命名空间 封闭空间 js模块化mvc（数据层、表现层、控制层） seajs（如果了解的呢，可以说）变量转换成对象的属性 对象化