得记住啊

面试题链接 https://juejin.im/post/5b44a485e51d4519945fb6b7

webpack考点

webpack是收把项目当作一个整体，通过一个给定的的主文件，webpack将从这个文件开始找到你的项目的所有依赖文件，使用loaders处理它们，最后打包成一个或多个浏览器可识别的js文件

- 一、npm install webpack webpack-dev-server --save-dev
  webpack是我们需要的模块打包机，webpack-dev-server用来创建本地服务器，监听你的代码修改，并自动刷新修改后的结果。下面是devserver的配置

```
contentBase,  // 为文件提供本地服务器
port, // 监听端口，默认8080
inline, // 设置为true,源文件发生改变自动刷新页面
historyApiFallback  // 依赖HTML5 history API,如果设置为true,所有的页面跳转指向index.html
devServer:{
    contentBase: './src' // 本地服务器所加载的页面所在的目录
    historyApiFallback: true, // 不跳转
    inline: true // 实时刷新
}
// 然后我们在根目录下创建一个'webpack.config.js'，在'package.json'添加两个命令用于本地开发和生产发布
"scripts": {
            "start": "webpack-dev-server",
            "build": "webpack"
        }
```

- 二、entry: 用来写入口文件，它将是整个依赖关系的根

```
var baseConfig = {
        entry: {
            main: './src/index.js'
        }
    }
```

- 三、output: 即使入口文件有多个，但是只有一个输出配置

```
var path = require('path')
    var baseConfig = {
        entry: {
            main: './src/index.js'
        },
        output: {
            filename: 'main.js',
```

```
            path: path.resolve('./build')
        }
    }
    module.exports = baseConfig
```

- 四、loader的作用： 1、实现对不同格式的文件的处理，比如说将scss转换为css，或者typescript转化为js2、转换这些文件，从而使其能够被添加到依赖图中loader是webpack最重要的部分之一，通过使用不同的Loader，我们能够调用外部的脚本或者工具，实现对不同格式文件的处理，loader需要在webpack.config.js里边单独用module进行配置，配置如下：

```
test: //匹配所处理文件的扩展名的正则表达式（必须）
    loader:// loader的名称（必须）
    include/exclude: //手动添加处理的文件，屏蔽不需要处理的文件（可选）
    query:// 为loaders提供额外的设置选项
    ex:
        var baseConfig = {
            // ...
            module: {
                rules: [
                    {
                        test: /*匹配文件后缀名的正则*/,
                        use: [
                            loader: /*loader名字*/,
                            query: /*额外配置*/
                        ]
                    }
                ]
            }
        }
        // babel-loader： 让下一代的js文件转换成现代浏览器能够支持的JS文件。babel有些复杂，所以大多数都会新建一个.babelrc进行配置css-loader,style-loader:两个建议配合使用，用来解析css文件，能够解释@import,url()如果需要解析less就在后面加一个less-loaderfile-loader: 生成的文件名就是文件内容的MD5哈希值并会保留所引用资源的原始扩展名url-loader: 功能类似 file-loader,但是文件大小低于指定的限制时，可以返回一个DataURL事实上，在使用less,scss,stylus这些的时候，npm会提示你差什么插件，差什么，你就安上就行了
```

- 四、Plugins

1. ExtractTextWebpackPlugin它会将入口中引用css文件，都打包都独立的css文件中，而不是内嵌在js打包文件中。
2. HtmlWebpackPlugin依据一个简单的index.html模版，生成一个自动引用你打包后的js文件的新index.html
3. HotModuleReplacementPlugin: 它允许你在修改组件代码时，自动刷新实时预览修改后的结果注意永远不要在生产环境中使用HMR loaders负责的是处理源文件的如css、jsx，一次处理一个文件。而plugins并不是直接操作单个文件，它直接对整个构建过程起作用下面列举了一些我们常用的plugins和他的用法ExtractTextWebpackPlugin: 它会将入口中引用css文件，都打包都独立的css文件中，而不是内嵌在js打包文件中。下面是他的应用

```
var ExtractTextPlugin = require('extract-text-webpack-plugin')
        var lessRules = {
            use: [
                {loader: 'css-loader'},
                {loader: 'less-loader'}
            ]
        }

        var baseConfig = {
            // ...
            module: {
                rules: [
                    // ...
                    {test: /\.less$/, use: ExtractTextPlugin.extract(lessRules)}
                ]
            },
            plugins: [
                new ExtractTextPlugin('main.css')
            ]
        }
```

- 五、产品阶段的构建 需要对资源进行别的 处理，例如压缩，优化，缓存，分离css和js。

```
var ENV = process.env.NODE_ENV
    var baseConfig = {
        // ...
        plugins: [
            new webpack.DefinePlugin({
                'process.env.NODE_ENV': JSON.stringify(ENV)
            })
        ]
    }
```

- 修改我们的script命令

```
"scripts": {
            "start": "NODE_ENV=development webpack-dev-server",
            "build": "NODE_ENV=production webpack"
        }
```

- process.env.NODE_ENV 将被一个字符串替代，它运行压缩器排除那些不可到达的开发代码分支。 当你引入那些不会进行生产的代码，下面这个代码将非常有用。

```
if (process.env.NODE_ENV === 'development') {
            console.warn('这个警告会在生产阶段消失')
        }
```

- 六、 优化插件 OccurenceOrderPlugin: 为组件分配ID,通过这个插件webpack可以分析和优先考虑使用最多 的模块，然后为他们分配最小的IDUglifyJsPlugin: 压缩代码下面是他们的使用方法var baseConfig = { // ... new webpack.optimize.OccurenceOrderPlugin() new webpack.optimize.UglifyJsPlugin()}然后在我们使用npm run build会发现代码是压缩的

斐波那契数列 使用递归

```
function fibonacci(n){
    if(n === 1 || n === 0 ) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

1. promise 承诺,就是说promise的状态一旦发生改变就永远不可逆。

- promis允许我们通过链式调用的方式来解决'回调地狱(由于回调函数是异步的，回调的代码中每一层的回调函数都需要依赖上一层的回调执行完，所以形成了层层嵌套的关系最终形成类似上面的回调地狱，此种形式展示时不利于我们阅读和维护)'的问题，通过Promise可以保证代码的整洁性和可读性。
- 创建一个promise实例

```
//resolve：表示异步操作执行成功后的回调函数
//reject：表示异步操作执行失败后的回调函数。

var p = new Promise(function(resolve,reject){
    setTimeout(function(){
        resolve("success");
    },1000);
    console.log("创建一个新的promise");
})
p.then(function(x){
    console.log(x);
})
//输出 延迟1000ms后输出success
创建一个新的promise
success
```

- 从上速可以看出promise方便处理异步操作，此外promise还可以链式调用

```
var p = new Promise(function(resolve,reject){resolve()})
p.then(..).then(..).then(..)
```

- 此外Promise除了then方法外，还提供了Promise.resolve、Promise.all、Promise.race等等方法 a. promise是一个对象或者函数，该对象或者函数都有一个then方法 b. thenable是一个对象或者函数，用来定义then方法 c. value是promise状态成功时的值 d. reason是promise状态失败时的值

要求

(1)一个promise必须有3个状态，pending，fullfilled(resolved),rejected当处于pending状态的时候，可以转移到fulfilled（resolved）,rejected状态。当处于fulfilled(resolved)状态或者rejected状态时就不可以改变。 (2) 一个promise必须有一个then方法，then方法必须接受这两个参数

```
promise.then(onFulfilled,onRejected)
```

其中onFulfilled方法表示状态从pending==》fulfilled(resolved)时所执行的方法，而onRejected表示状态从pending===》rejected所执行的方法。 (3) 为了实现链式调用，then方法必须返回一个promise

```
promise2=promise1.then(onFulfilled,onRejected)
```

###实现一个符合Promise/A+规范的Promise

```
function myPromise(constructor){
    let self=this;
    self.status="pending";//定义状态改变前的初始状态
    self.value=undefined;//定义状态为resolved的时候的状态
    self.reason=undefined;//定义状态为rejected的时候的状态
    function resolve(value){
        //两个==="pending",保证了状态的改变是不可逆的
        if(self.status==="pending"){
            self.value=value;
            self.status="resolved";
        }
    }
    function reject(reason){
        //两个==="pending"，保证了状态的改变是不可逆的
        if(self.status==="pending"){
            self.reason=reason;
            self.status="rejected";
        }
    }
    //捕获构造异常
    try{
        constructor(resolve,reject);
    }catch(e){
        reject(e);
    }
}
```

同时，需要在myPromise的原型上定义链式调用的then方法

```
myPromsie.prototype.then=function(onFullfilled,onRejected){
    let self=this;
```

```
        switch(self.status){
            case "resolved":
            onFullfilled(self.value);
            break;
            case "rejected":
            onRejected(self.reason);
            break;
            default;
        }
    }
```

2. vue 更新机制，双向绑定 https://juejin.im/post/5bb6db14f265da0abf7cfc4a

3. react 更新机制，diff算法

diff算法

React diff 是react性能的基础。 是在原diff算法上结合web特性的大胆改进。例如忽略跨层级的移动操作，用key让同一层级的节点进行区分，较少重新渲染的次数。 在tree层次上，相同颜色方框内的 DOM 节点进行比较。 在组件层次上，同一类型的组件，按照原策略继续比较 virtual DOM tree，如果不是同一类型组件，则替换整个组件下的所有子节点。

更新机制

4. 前端安全：XSS（跨站脚本攻击），CSRF跨站请求伪造）

- 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。

结果应该是这样的：Set-Cookie=.....

xss 跨站脚本攻击(Cross Site Scripting),为了不和层叠样式表（Cascading Style Sheets,CSS）缩写混淆，所以将跨站脚本攻击缩写为xss。Xss是攻击者在web页面插入恶意的代码。当用户浏览该页面的时候，代码执行，从而实现攻击目的。对受害用户可能采取Cookie资料窃取、会话劫持、钓鱼欺骗等各种攻击。

XSS跨站脚本攻击分为： 反射型XSS 反射性XSS，也就是非持久性XSS。用户点击攻击链接，服务器解析后响应，在返回的响应内容中出现攻击者的XSS代码，被浏览器执行。一来一去，XSS攻击脚本被web server反射回来给浏览器执行，所以称为反射型XSS。

- 如何防范： 1、正确使用GET,POST和Cookie；2、在非GET请求中增加伪随机数。 1.对于关键操作我们应该采用post方法。

2.CSRF在攻击的时候往往是在用户不情的情况下提交的，我们可以使用验证码来强制跟用户交互，但是太多强制性的操作对用户来说体验感不好，所以要权衡利弊。

3.在重要的请求中添加Token，目前主流的做法是使用Token抵御CSRF攻击。CSRF攻击成功的条件在于攻击者能够预测所有的参数从而构造出合法的请求，所以我们可以加大这个预测的难度，加入一些黑客不能伪造的信息。我们在提交表单时，保持原有参数不变，另外添加一个参数Token，该值可以是随机并且加密的，当提交

表单时，客户端也同时提交这个token，然后由服务端验证，验证通过才是有效的请求。但是由于用户的Cookie很容易由于网站的XSS漏洞而被盗取，所以这个方案必须要在没有XSS的情况下才安全。

4.检测Referer.所谓Referer，就是在一个网络请求头中的键值对，标示着目前的请求是从哪个页面过来的。服务器通过检查Referer的值，如果判断出Referer并非本站页面，而是一个外部站点的页面，那么我们就可以判断出这个请求是非法的。与此同时，我们也就检测到了一次csrf攻击。但是，服务器有时候并不能接收Referer值，所以单纯地只通过Referer来防御是不太合理的，它因此经常用于csrf的检测。

5. 前端优化

1.尽量减少HTTP请求 6. webpack babel配置中的stage-0是什么意思？ babel是把es6代码编译为es5, stage-0是对es7一些提案的支持，Babel通过插件的方式引入，让Babel可以编译ES7代码。当然由于ES7没有定下来，所以这些功能随时肯能被废弃掉的。现在我们来一一分析里面都有什么。“stage-0” 法力无边呢，因为它包含stage-1, stage-2以及stage-3的所有功能，同时还另外支持如下两个功能插件： transform-do-expressions transform-function-bind stage-2而是为了增强代码的可读性和可修改性而提出的 stage-3支持大名鼎鼎的async和await, 7. react生命周期 React规定constructor有三个参数，分别是props、context和updater。

props是属性，它是不可变的。 context是全局上下文。 updater是包含一些更新方法的对象，this.setState最终调用的是this.updater.enqueueSetState方法，this.forceUpdate最终调用的是this.updater.enqueueForceUpdate方法，所以这些API更多是React内部使用，暴露出来是以备开发者不时之需。

angular声明周期

- ngOnChanges() 当Angular（重新）设置数据绑定输入属性时响应。 该方法接受当前和上一属性值的SimpleChanges对象 当被绑定的输入属性的值发生变化时调用，首次调用一定会发生在ngOnInit()之前。
- ngOnInit() 在Angular第一次显示数据绑定和设置指令/组件的输入属性之后，初始化指令/组件。 在第一轮ngOnChanges()完成之后调用，只调用一次。
- ngDoCheck() 检测，并在发生Angular无法或不愿意自己检测的变化时作出反应。 在每个Angular变更检测周期中调用，ngOnChanges()和ngOnInit()之后。
- ngAfterContentInit() 当把内容投影进组件之后调用。 第一次ngDoCheck()之后调用，只调用一次。 只适用于组件。
- ngAfterContentChecked() 每次完成被投影组件内容的变更检测之后调用。 ngAfterContentInit()和每次ngDoCheck()之后调用 只适合组件。
- ngAfterViewInit() 初始化完组件视图及其子视图之后调用。 第一次ngAfterContentChecked()之后调用，只调用一次。 只适合组件。
- ngAfterViewChecked() 每次做完组件视图和子视图的变更检测之后调用。 ngAfterViewInit()和每次ngAfterContentChecked()之后调用。 只适合组件。
- ngOnDestroy() 当Angular每次销毁指令/组件之前调用并清扫。 在这儿反订阅可观察对象和分离事件处理器，以防内存泄漏。 在Angular销毁指令/组件之前调用。

8. js原型链 我们声明的构造函数 afun 是由另一个构造函数 Function 生成的 由于_proto_是任何对象都有的属性，而js里万物皆对象，所以会形成一条_proto_连起来的链条，递归访问_proto_必须最终到头，并且值是null。 当js引擎查找对象的属性时，先查找对象本身是否存在该属性，如果不存在，会在原型链上查找，但不会查找资深的prototype
9. service work Service Worker作用 web worker 在HTML页面中，如果在执行脚本时，页面的状态是不可相应的，直到脚本执行完成后，页面才变成可相应。web worker是运行在后台的js，独立于其他脚本，不会影响页面你的性能。并且通过postMessage将结果回传到主线程。这样在进行复杂操作的时候，就不会阻塞主线程了。 如何创建web worker：

检测浏览器对于web worker的支持性 创建web worker文件（js，回传函数等） 创建web worker对象 1.网络代理，转发请求，伪造响应

网络代理

转发请求

伪造响应

2.离线缓存 service worker与离线缓存 What's Service Workers? 小试 Service Workers。 调试 Service Workers。 通过 postMessage 与主窗口通信。 为应用添加离线缓存。 Service workers 的生命周期与更新。 3.消息推送 推送的到达率（到没到）， 推送的实时性（什么时候到）， 推送的表现形式（以什么样的 ui 展示）， 推送的后台耗电（后台是用户无感知，但是系统很敏感）， 推送内容（该推的不推，不该推的瞎推）。 4.后台消息传递 10. 闭包

闭包的含义：闭包就是能够读取其他函数内部变量的函数。

闭包之变量的作用域：函数内部声明变量的时候，一定要使用var命令。如果不用的话，你实际上声明了一个全局变量

使用闭包的注意点：

1）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。

2）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。

闭包的用途

一个是前面提到的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。 11. v8引擎区别浏览器做的优化 V8引擎是由Google开发的开源产品，使用C++开发。该引擎在Google Chrome浏览器中使用。和其他的引擎不同，V8还被流行的Node.js runtime使用。 最初，V8被设计用于提升web浏览器内部的JavaScript运行的性能。为了提升速度，V8把JavaScript代码翻译成执行效率更高的机器码（不使用解释器来做这件事）。在执行JavaScript代码时，V8像很多的现代JavaScript引擎——如SpiderMonkey或Rhino（Mozilla）——一样，实现了一个JIT编译器（即时编译器），从而把JavaScript代码编译成机器语言。和其他引擎最主要的差别在于，V8不会生成任何字节码或是中间代码。 12. 编译型和解释型语言的区别 机器语言： 优点是最底层，速度最快，缺点是最复杂，开发效率最低 汇编语言： 优点是比较底层，速度最快，缺点是复杂，开发效率最低 高级语言： 【编译型： 1、把源代码编译成机器语言的可执行程序 2、执行可执行程序文件 优点： 1、程序执行时，不再需要源代码， 2、执行速度快，因为你的程序代码已经翻译成了是计算机可以理解的机器语言。 缺点： 1、每次修改源代码，都要重新编译，生成机器码文件 2、跨平台性不好，不同操作系统，调用底层的机器指令不同，需为不同平台生成不同的机器码文件 解释型： 1、用户调用解释器，执行源代码文件 2、解释器把源代码文件边解释成机器语言边交给CPU执行 优点： 1、天生跨平台，因为解释器已经做好了对不同平台的交互处理，用户写的源代码不需要再考虑凭条差异性，可谓，一份源代码，所有平台都可以直接执行 2、随时修改，立刻生效，改完源代码后，直接运行看效果 缺点： 1、运行效率低，所有的代码均需经过解释器边解释变执行，速度比编译型慢很多 2、代码是明文 13. 跟缓存相关的配置 14. 如何提高webpack打包的速度

- a.区分编译目标，使用cross-env这个插件

```
{
    "script":{
        "build:js":"cross-env ctarget=all webpack",
        "build:js+css":"cross-env ctarget=js webpack"
    }
}
```

这样我们在webpack配置文件中通过process.env.ctarget区分当前编译目标了。

- b.分类js和css 通过webpack的loader实现，使用现有的轮子string-replace-loader

这个loader可以在编译阶段修改代码，并且可以使用正则表达式进行替换。可以使用他加载js文件，然后删除部分代码 (1)定位体积大的模块，使用webpack插件webpack-bundle-analyzer来查看整个项目的体积结构对比，他是以treemap的形式展现出来，很形象直观 (2)提取公共模块，使用webpack自带的插件CommonsChunPlugin,传入对象参数分别为names(打包文件的名字)、minChunks(如果是数字表示该模块被其他模块引用的次数，如果是函数返回值必须是布尔类型的值指明是否应该被打包进公共模块))、chunks则会指定一个字符串数组，如果设置了该参数，则打包的时候只会从其中指定的模块中提取公共子模块

```
// a.js
require('./chunk1');
require('./chunk2');
require('jquery');
// b.js
require('./chunk1');
require('./chunk2');
require('vue');
// webpack配置如下
module.exports = {
  entry: {
    main: './main.js',
    main1: './main1.js',
    jquery:["jquery"],
    vue:["vue"]
  },
  output: {
    path: __dirname + '/dist',
    filename: '[name].js'
  },
  plugins: [
    new CommonsChunkPlugin({
      name: ["common","jquery","vue","load"],
      minChunks:2
    })
  ] };
```

(3)移除不必要的文件，webpack自带的两个库可以实现这个功能IgnorePlugin和ContextReplacementPlugin

```
// 插件配置
plugins: [
  // 忽略moment.js中所有的locale文件
  new webpack.IgnorePlugin(/^\.\/locale$/, /moment$/),
],
// 使用方式
const moment = require('moment');
// 引入zh-cn locale文件
require('moment/locale/zh-cn');
moment.locale('zh-cn');
```

(4)模块化引入 (5)通过CDN引入, (6)开启Gzip压缩 (7)压缩混淆代码

15. nginx转发的配置 nginx实现端口转发，实现原理是用Nginx监听80端口，当有HTTP请求到来时，将HTTP请求的HOST等信息与其配置文件进行匹配并转发给对应的应用。 端口转发指的是由软件统一监听某个域名上的某个端口(一般是80端口),当访问服务器的域名和端口符合要求时，就按照配置转发给指定的Tomcat服务器处理，我们常用的Nginx也有端口转发功能。

16. node 的一些特点 (1)异步I/O。当同时发起多个I/O请求时，异步I/O能极大的提升效率 (2)事件与回调函数。事件的编程方式具有轻量级、松耦合、只关注事务点等优势，缺点是多个异步任务的场景下，事件与事件之间各自独立，如何协作是一个问题。 (3)单线程。优点是不用像多线程那样处处在意状态的同步问题，没有死锁的存在，也没有线程上下文交换所带来的性能上的开销。 单线程的弱点有3方面，一是无法利用多核CPU。二是错误会引起整个应用退出，应用的健壮性值得考验。三是大量占用CPU导致无法继续调用异步I/O. 解决办法是：使用child_process模块，启用多进程。 (4)跨平台 window和*nix

17. node对于字节流的控制

18. https 的过程和服务器搭建

- 搭建https服务器。一、SSL证书申请，确认要申请证书的域名，生成私钥。生成证书后，证书颁发机构提供两张SSL证书，一个是站点的证书，一个是服务器的根证书，将生成的私钥文件“server.key”和服务器证书“server.crt”拷贝到服务器指定的目录，接着就可以配置HTTPS服务器了。二、配置HTTPS服务，1.基于Nginx的HTTP服务配置，2.基于Tomcat的HTTPS服务配置

19. http的无状态

- 无状态是指当浏览器发送请求给客户端请求，但是当同一个浏览器再次给服务器发送请求的时候，服务器并不知道它就是刚才的那个浏览器。简单的说，服务器不会记得你，所以就是无状态协议。

20. 如何处理js的错误:js eslint语法规范错误提示代码
21. node垃圾回收
22. js内存溢出详解链接 https://blog.csdn.net/vuturn/article/details/45097353 一、常见的内存泄露类型 1.造成内存泄露的代码： （1）循环引用 （2）自动类型装箱转换 （3）某些DOM 操作 二、内存泄露的解决方案 1.显示类型转换 对于类型转换的错误，我们可以通过显示类型转换的方式来避免。 2.避免事件导致的循环引用。 三、内存泄露的一些疑问

1、内存泄露是内存占用很大吗？ 不是，即使是1byte的内存，也叫内存泄露。

2、程序中提示内存不足，是内存泄露吗？ 不是，着一般是无限递归函数调用，导致栈内存溢出。

3.内存泄露是哪个区域？

堆区。栈区不会泄露

4.window对象时DOM对象吗？

不是，window对象参与的循环引用是不会内存泄露。

5.内存泄露的后果？

大多数情况下，后果不是很严重。但是过多的DOM操作会使网页执行速度变慢。

6.跳转网页，内存泄露仍然存在吗？

仍然存在，直到浏览器关闭。 23. http优缺点

优点

1.支持客户/服务器模式。 2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。 由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。 3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。 4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。 5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。以上虽为复制之作，但无可否认的是，这就是最适当的回答了，网友自己写的虽有益互索，但参考性太弱。如果楼主对本人的回答不满意，可以百度一下，一大堆。如有比如上答案更精确的，请楼主采纳它。

缺点详解链接

https://blog.csdn.net/qq_33301113/article/details/78725951 通信使用明文，内容可能会被窃听； 不验证通信方的身份，因此有可能遭遇伪装； 无法证明报文的完整性，有可能已遭篡改 24. es6 阮一峰的ES6 http://es6.ruanyifeng.com/ 25. webpack 26. 组件封装 27. 协商缓存和强缓存 http://www.cnblogs.com/wonyun/p/5524617.html

- 描述：缓存这东西，第一次必须获取到资源后，然后根据返回的信息来告诉如何缓存资源，可能采用的是强缓存，也可能告诉客户端浏览器是协商缓存，这都需要根据响应的header内容来决定的。下面用两幅图来描述浏览器的缓存是怎么玩的，让大家有个大概的认知。
- 《强缓存》：200 from cache 直接从缓存取,不发送服务器。(header字段:expires和cache-control：max-age=number，其中no-cache：不使用本地缓存，no-store：直接禁止游览器缓存数据，public：可以被所有的用户缓存，包括终端用户和CDN等中间代理服务器，private：只能被终端用户的浏览器缓存，不允许CDN等中继缓存服务器对其缓存。) 浏览器在请求某一资源时，会先获取该资源缓存的header信息，判断是否命中强缓存（cache-control和expires信息），若命中直接从缓存中获取资源信息，包括缓存header信息；本次请求根本就不会与服务器进行通信；在firebug下可以查看某个具有强缓存资源返回的信息，例如本地firebug查看的一个强缓存js文件
- 《协商缓存》 304 not modified）通过服务器来告知缓存是否可用(header字段包括Last-Modified/If-Modified-Since和Etag/If-None-Match) 如果没有命中强缓存，浏览器会发送请求到服务器，请求会携带第一次请求返回的有关缓存的header字段信息（Last-Modified/If-Modified-Since和Etag/If-None-Match），由服务器根据请求中的相关header信息来比对结果是否协商缓存命中；若命中，则服务器返

回新的响应header信息更新缓存中的对应header信息，但是并不返回资源内容，它会告知浏览器可以直接从缓存获取；否则返回最新的资源内容

> 既生Last-Modified何生Etag HTTP1.1中Etag的出现主要是为了解决几个Last-Modified比较难解决的问题： 一些文件也许会周期性的更改，但是他的内容并不改变(仅仅改变的修改时间)，这个时候我们并不希望客户端认为这个文件被修改了，而重新GET；

某些文件修改非常频繁，比如在秒以下的时间内进行修改，(比方说1s内修改了N次)，If-Modified-Since能检查到的粒度是s级的，这种修改无法判断(或者说UNIX记录MTIME只能精确到秒)；

某些服务器不能精确的得到文件的最后修改时间。

- 《Last-Modified与ETag是可以一起使用的，服务器会优先验证ETag，一致的情况下，才会继续比对Last-Modified，最后才决定是否返回304》
- 强缓存如何重新加载缓存缓存过的资源? 通过更新页面中引用的资源路径，让浏览器主动放弃缓存，加载新资源。

28. 浏览器渲染原理链接 https://juejin.im/entry/590bebfa128fe1005836e9e7
29. 面向对象 https://blog.csdn.net/jerry11112/article/details/79027834 万物皆对象，把一个对象抽象成类,具体上就是把一个对象的静态特征和动态特征抽象成属性和方法,也就是把一类事物的算法和数据结构封装在一个类之中,程序就是多个对象和互相之间的通信组成的.

面向对象具有封装性,继承性,多态性。 封装:隐蔽了对象内部不需要暴露的细节,使得内部细节的变动跟外界脱离,只依靠接口进行通信.封装性降低了编程的复杂性. 通过继承,使得新建一个类变得容易,一个类从派生类那里获得其非私有的方法和公用属性的繁琐工作交给了编译器. 而继承和实现接口和运行时的类型绑定机制 所产生的多态,使得不同的类所产生的对象能够对相同的消息作出不同的反应,极大地提高了代码的通用性. 30. 原型 实现继承和共享数据节省内存空间 getPrototypeOf指的是获取原型

31. 快速排序 快速排序 http://viviier.com/2017/01/20/%E7%AE%97%E6%B3%95%20-%20%E5%86%92%E6%B3%A1%E6%8E%92%E5%BA%8F%20%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F/
32. 异步 关于异步考点分析 https://juejin.im/post/5a41e11651882572ed55d4fd
33. 为什么用angular angular内部有什么值得学习的地方
34. 为什么说https安全 HTTPS是http的安全版，是建立在密码学基础之上的一种安全通信协议，严格来说是基于 HTTP 协议和 SSL/TLS 的组合。理解 HTTPS 之前有必要弄清楚一些密码学的相关基础概念，比如：明文、密文、密码、密钥、对称加密、非对称加密、信息摘要、数字签名、数字证书。接下来我会逐个解释这些术语，文章里面提到的『数据』、『消息』都是同一个概念，表示用户之间通信的内容载体，此外文章中提到了以下几个角色： Alice：消息发送者 Bob：消息接收者 Attacker：中间攻击者 Trent：第三方认证机构
35. nginx反向代理 反向代理（Reverse Proxy）方式是指以代理服务器来接受Internet上的连接请求，然后将请求转发给内部网络上的服务器；并将从服务器上得到的结果返回给Internet上请求连接的客户端，此时代理服务器对外就表现为一个服务器。 通常的代理服务器，只用于代理内部网络对Internet的连接请求，客户机必须指定代理服务器,并将本来要直接发送到Web服务器上的http请求发送到代理服务器中。当一个代理服务器能够代理外部网络上的主机，访问内部网络时，这种代理服务的方式称为反向代理服务。
36. git的各种操作

```
git reset <file> #撤销指定的文件
git reset #撤销所有的文件
git rm --cached .   #删除文件
git rm -r --cached . #删除文件和目录
.gitignore: 把不需要提交的文件增加到这个文件
git add : 增加指定的文件，少用点号
git branch -d branch_name 删除本地的分支
git checkout master 切换分支
git reset --hard HEAD~1 撤销commit操作，删除变化
```

37. 三次握手 客户端和服务端都需要直到各自可收发，因此需要三次握手。 C发起请求连接S确认，也发起连接C确认我们再看看每次握手的作用：第一次握手：S只可以确认 自己可以接受C发送的报文段第二次握手：C可以确认 S收到了自己发送的报文段，并且可以确认 自己可以接受S发送的报文段第三次握手：S可以确认 C收到了自己发送的报文段
38. 渲染路径 渲染路径 https://segmentfault.com/a/1190000008984446 什么是 CRP ？ 览器从开始请求 HTML 文档， 到首次渲染到屏幕上（首屏）， 背后需要做很多的事情， 这一连串事情就是 CRP 。 开发 app 的时候很多优化都是和缩短 CRP 有关的。 缩短 CRP 的长度能够有效降低首屏时间。 这也是理解 CRP 最大的好处。 游览器解析 HTML 形成 DOM 树。 游览器解析 CSS 代码输出 CSSOM 。 DOM 和 CSSOM 一起构造出一颗 Render Tree 。 游览器对 Render Tree 上的节点计算精确位置（布局）。 使用 Render Tree 绘制 (Painting) 到屏幕上。
39. 缓存 缓存详解 https://juejin.im/post/5a6c87c46fb9a01ca560b4d7
40. webpack的优化做过哪些？用过哪些api，打包公共文件应该怎么做？入口文件怎么配置的？自己写过loader或者plugin吗？

http、html和浏览器篇

1.http和https https的SSL加密是在传输层实现的。 (1)http和https的基本概念 http: 超文本传输协议，是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。 https: 是以安全为目标的HTTP通道，简单讲是HTTP的安全版，即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL。 https协议的主要作用是：建立一个信息安全通道，来确保数组的传输，确保网站的真实性。 (2)http和https的区别？ http传输的数据都是未加密的，也就是明文的，网景公司设置了SSL协议来对http协议传输的数据进行加密处理，简单来说https协议是由http和ssl协议构建的可进行加密传输和身份认证的网络协议，比http协议的安全性更高。 主要的区别如下：

Https协议需要ca证书，费用较高。 http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。 使用不同的链接方式，端口也不同，一般而言，http协议的端口为80，https的端口为443 http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

(3)https协议的工作原理 客户端在使用HTTPS方式与Web服务器通信时有以下几个步骤，如图所示。

客户使用https url访问服务器，则要求web 服务器建立ssl链接。 web服务器接收到客户端的请求之后，会将网站的证书（证书中包含了公钥），返回或者说传输给客户端。 客户端和web服务器端开始协商SSL链接的安全等级，也就是加密等级。 客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站。 web服务器通过自己的私钥解密出会话密钥。 web服务器通过会话密钥加密与客户端之间的通信。

(4)https协议的优点

使用HTTPS协议可认证用户和服务器，确保数据发送到正确的客户机和服务器； HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。 HTTPS是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。 谷歌曾在2014年8月份调整搜索引擎算法，并称“比起同等HTTP网站，采用HTTPS加密的网站在搜索结果中的排名将会更高”。

(5)https协议的缺点

https握手阶段比较费时，会使页面加载时间延长50%，增加10%~20%的耗电。 https缓存不如http高效，会增加数据开销。 SSL证书也需要钱，功能越强大的证书费用越高。 SSL证书需要绑定IP，不能再同一个ip上绑定多个域名，ipv4资源支持不了这种消耗。 41. 手写轮播图效果

```
//此时我们用js代码生成小圆点
var scroll = document.getElementById("scroll"); // 获得大盒子
        var ul = document.getElementById("ul"); // 获得ul
        var ulLis = ul.children;// 获得ul的盒子 以此来生成ol中li的个数
        var liWidth = ul.children[0].offsetWidth;// 获得每个li的宽度
        // 操作元素
        // 因为要做无缝滚动，所以要克隆第一张，放到最后一张后面
        // 1. 克隆元素
        ul.appendChild(ul.children[0].cloneNode(true));

        // 2.创建ol 和li
        var ol = document.createElement("ol");//创建ol元素
        scroll.appendChild(ol);// 把ol放到scroll盒子里面去
        for (var i=0;i<ulLis.length-1;i++) {
            var li = document.createElement("li");// 创建li元素
            li.innerHTML = i + 1;// 给li里面添加文字  1 2 3 4 5
            ol.appendChild(li);// 将li元素添加到ol里面
        }
        ol.children[0].className = "current";// ol中的第一个li背景色为purple
// 此时ol li元素即小圆点创建完毕 我们接着用js做动画
// 动画部分包括：
// 1.鼠标经过第几个小圆点，就要展示第几张图片，并且小圆点的颜色也发生变化
// 2.图片自动轮播，（这需要一个定时器）
// 3.鼠标经过图片，图片停止自动播放（这需要清除定时器）
// 4.鼠标离开图片，图片继续自动轮播 （重新开始定时器）
// 这里我们封装了一个animate()动画函数
// 动画函数 第一个参数，代表谁动  第二个参数 动多少
        // 让图片做匀速运动，匀速动画的原理是 当前的位置 + 速度  即 offsetLeft + speed
        function animate(obj,target){
            // 首先清除掉定时器
            clearInterval(obj.timer);
            // 用来判断 是+ 还是 -  即说明向左走还是向右走
            var speed = obj.offsetLeft < target ? 15 : -15;
            obj.timer = setInterval(function(){
                var result = target - obj.offsetLeft;//它们的差值不会超过speed
                obj.style.left = obj.offsetLeft + speed + "px";
                // 有可能有小数的存在，所以在这里要做个判断
                if (Math.abs(result) <= Math.abs(speed)) {
```

```
                    clearInterval(obj.timer);
                    obj.style.left = target + "px";
                }
            },10);
        }
        // 定时器 函数
        var timer = null; // 轮播图的定时器
        var key = 0;// 控制播放的张数
        var circle = 0;// 控制小圆点

        timer = setInterval(autoplay,1000);// 自动轮播
        function autoplay(){
            /*自动轮播时,要对播放的张数key进行一个判断,如果播放的张数超过ulLis.length-
1,
            就要重头开始播放.  因为我们克隆了第一张并将其放在最后面,所以我们要从第二张图
片开始播放*/
            key++; // 先++
            if(key > ulLis.length-1){// 后判断

                ul.style.left = 0; // 迅速调回
                key = 1; // 因为第6张是第一张，所以播放的时候是从第2张开始播放
            }
            // 动画部分
            animate(ul,-key*liWidth);
            // 小圆点circle    当显示第几张图片是，对应的小圆点的颜色也发生变化

            /*同理,对小圆点也要有一个判断*/
            circle++;
            if (circle > olLis.length-1) {
                circle = 0;
            }
            // 小圆点颜色发生变化
            for (var i = 0 ; i < olLis.length;i++) {
                // 先清除掉所用的小圆点类名
                olLis[i].className = "";
            }
            // 给当前的小圆点 添加一个类名
            olLis[circle].className = "current";

        }
```

寄生组合继承

```
// 寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不
会初始化两次实例方法/属性
function Cat(name){
  Animal.call(this);
  this.name = name || 'Tom';
}
(function(){
  // 创建一个没有实例方法的类
```

```
  var Super = function(){};
  Super.prototype = Animal.prototype;
  //将实例作为子类的原型
  Cat.prototype = new Super();
})();
// Test Code
var cat = new Cat();
console.log(cat.name);
console.log(cat.sleep());
console.log(cat instanceof Animal); // true
console.log(cat instanceof Cat); //true
```

事件委托

```
window.onload = function(){
    var oUl = document.getElementById("ul1");
    oUl.onclick = function(ev){
        var ev = ev || window.event;
        var target = ev.target || ev.srcElement;
        if(target.nodeName.toLowerCase() == 'li'){
                alert(123);
               alert(target.innerHTML);
        }
    }
}
```

a数组包含b数组

```
isContained =(a, b)=>{
    if(!(a instanceof Array) || !(b instanceof Array)) return false;
    if(a.length < b.length) return false;
    var aStr = a.toString();
    for(var i = 0, i < b.length; i++){
        //-1相当于false,如果不包含返回false
      if(aStr.indexOf(b[i]) == -1) return false;
    }
    return true;
  }
```

web移动端click点透问题

点击穿透现象有3种： 点击穿透问题：点击蒙层（mask）上的关闭按钮，蒙层消失后发现触发了按钮下面元素的click事件跨页面点击穿透问题：如果按钮下面恰好是一个有href属性的a标签，那么页面就会发生跳转另一种跨页面点击穿透问题：这次没有mask了，直接点击页内按钮跳转至新页，然后发现新页面中对应位置元素的click事件被触发了。

- 三、解决方案

1. 对于B元素本身没有默认click事件的情况（无a标签等），应统一使用touch事件，统一代码风格，并且由于click事件在移动端的延迟要大很多，不利于用户体验，所以关于触摸事件应尽量使用touch相关事件。

2. 对于B元素本身存在默认click事件的情况,应及时取消A元素的默认点击事件，从而阻止click事件的产生。即应在上例的handle函数中添加代码如下：

if(eve == "touchend") e.preventDefault(); 3. 对于遮盖浮层，由于遮盖浮层的点击即使有小延迟也是没有关系的，反而会有疑似更好的用户体验，所以这种情况，可以针对遮盖浮层自己采用click事件，这样就不会出现点透问题。

深拷贝

不能实现例如包装对象Number,String,Boolean,以及正则对象RegExp和Date对象的克隆,应该怎么办呢？ 所有对象都有valueOf方法，valueOf方法对于：如果存在任意原始值，它就默认将对象转换为表示它的原始值。对象是复合值，而且大多数对象无法真正表示为一个原始值，因此默认的valueOf()方法简单地返回对象本身，而不是返回一个原始值。数组、函数和正则表达式简单地继承了这个默认方法，调用这些类型的实例的valueOf()方法只是简单返回这个对象本身。

```
function baseClone(base){
 return base.valueOf();
}

//Number
var num=new Number(1);
var newNum=baseClone(num);
//newNum->1

//String
var str=new String('hello');
var newStr=baseClone(str);
// newStr->"hello"

//Boolean
var bol=new Boolean(true);
var newBol=baseClone(bol);
//newBol-> true
```

```
function extend(a, b) {
       for (var key in a) {
           var item = a[key];
           //是否是对象
           if (item instanceof Object) {
               //如果是对象,在b对象中新增一个属性,并且这个属性是空对象
               b[key] = {};
               //递归,把a对象的对象属性的值一个一个的复制到b对象的对象属性中
               extend(item, b[key]);
           } else if (item instanceof Array) { //判断
               //是否是数组,在b对象中新增一个属性,并且这个属性也是数组
               b[key] = [];
```

```
                //递归,把a对象中的这个数组的值一个一个的复制到b对象这个数组的属性中
                extent(item, b[key]);
            } else {
                b[key] = a[key];
            }
        }
    }
    //被复制的对象
    var obj1 = {
        name: '红红',
        age: '24',
        color: ['blue', 'red', 'black'],
        obj: {
            book: 'javascript高级程序设计',
            dog: '小花'
        }
    }

    //另一个对象
    var obj2 = {};
    //深拷贝,obj1属性的值改变后不会影响到obj2属性的值
    extend(obj1, obj2);
```

对于正则表达式的克隆

```
RegExp.prototype.clone = function() {
var pattern = this.valueOf();
var flags = '';
flags += pattern.global ? 'g' : '';
flags += pattern.ignoreCase ? 'i' : '';
flags += pattern.multiline ? 'm' : '';
return new RegExp(pattern.source, flags);
};

var reg=new RegExp('/111/');
var newReg=reg.clone();
//newReg->  /\/111\//
```

手写Ajax

具体来说，AJAX 包括以下几个步骤。 创建 XMLHttpRequest 实例 发出 HTTP 请求 接收服务器传回的数据 更新网页数据

ajax的同源政策。AJAX 只能向同源网址（协议、域名、端口都相同）发出 HTTP 请求，如果发出跨域请求，就会报错。比如：https://www.github.com不能向http://www.github.com发送 AJAX 请求。 CORS CORS 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin resource sharing）。它允许浏览器向跨源服务器，发出 XMLHttpRequest请求，从而克服了 AJAX 只能同源使用的限制。 如果 A 想利用JavaScript向 B 网站发送 AJAX 请求，那么只需要在 B 的后端写上： response.setHeader('Access-Control-Allow-Origin',' A 网站的 协议 + 域名 + 端口号') 就可以实现 AJAX 的跨域请求。

```
button.addEventListener("click", e => {
  let request = new XMLHttpRequest();
  request.open("GET", "/yyzcl"); //配置request.open是用来设置 HTTP 请求的第一部分的
  request.setHeader("yyzcl","OK"); //设置请求头.是用来设置请求头的第二部分的，第二部分
有些信息是浏览器自动设置的
  request.send("Yyzcl");
  request.onreadystatechange = () => {
      //0==》unsent未打开    1==》opened未发送  2==》headers_received已获得响应头
3==》loading正在下载响应体  4==》done请求完成
    if (request.readyState === 4) {
      let string = request.responseText;  // 获取服务器的返回值，是一个字符串
      let obj = window.JSON.parse(string);  // 把符合JSON语法的字符串转换为JS对应的值
（可以是对象，字符串等）
    }
  };
});
```

- ajax的优点 无需刷新页面请求数据，可以使产品更快、更小、更友好 可以把以前服务端的任务转嫁到客户端来处理，减轻服务器负担，节省带宽 浏览器支持好，无需插件
- ajax的缺点 不支持浏览器的回退按钮 安全性存在问题，能够在用户不知情的情况下发送请求 暴露了http交互细节 对搜索引擎(网络爬虫)的支持比较弱 程序不容易调试

cookie有效期

expires/Max-Age 字段为此cookie超时时间。若设置其值为一个时间，那么当到达此时间后，此cookie失效。不设置的话默认值是Session，意思是cookie会和session一起失效。当浏览器关闭(不是浏览器标签页，而是整个浏览器) 后，此cookie失效。

Cookie和session的区别

HTTP是一个无状态协议，因此Cookie的最大的作用就是存储sessionId用来唯一标识用户.

自己实现一个bind函数

```
Function.prototype.bind=function(obj,arg){
  var arg=Array.prototype.slice.call(arguments,1);
  var context=this;
  return function(newArg){
    arg=arg.concat(Array.prototype.slice.call(newArg));
    return context.apply(obj,arg);
  }
}
```

用setTimeout实现setInterval

setInterval的缺陷，使用setInterval()创建的定时器确保了定时器代码规则地插入队列中。这个问题在于：如果定时器代码在代码再次添加到队列之前还没完成执行，结果就会导致定时器代码连续运行好几次。而之间没有间隔。不过幸运的是：javascript引擎足够聪明，能够避免这个问题。当且仅当没有该定时器的如何代码实例

时，才会将定时器代码添加到队列中。这确保了定时器代码加入队列中最小的时间间隔为指定时间。 这种重复定时器的规则有两个问题：1.某些间隔会被跳过 2.多个定时器的代码执行时间可能会比预期小。

```
function say(){
  //something
  setTimeout(say,200);
}
setTimeout(say,200)
//或者
```

代码的执行顺序

```
setTimeout(function(){console.log(1)},0);
new Promise(function(resolve,reject){
   console.log(2);//第一次执行
   resolve();
}).then(function(){console.log(3)//第四次
}).then(function(){console.log(4)});//第五次

process.nextTick(function(){console.log(5)});//第三次

console.log(6);//第二次
//输出2,6,5,3,4,1
```

js怎么控制一次加载一张图片，加载完后再加载下一张

```
<script type="text/javascript">
var obj=new Image();
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";
obj.onload=function(){
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);
document.getElementById("mypic").innnerHTML="<img src='"+this.src+"' />";
}
</script>
<div id="mypic">onloading……</div>
//或者
<script type="text/javascript">
var obj=new Image();
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";
obj.onreadystatechange=function(){
if(this.readyState=="complete"){
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);
document.getElementById("mypic").innnerHTML="<img src='"+this.src+"' />";
}
}
</script>
<div id="mypic">onloading……</div>
```

如何实现sleep的效果（es5或者es6）

(1)while循环的方式

```
function sleep(ms){
   var start=Date.now(),expire=start+ms;
   while(Date.now()<expire);//调用函数的时间小于 调用函数的时间+延迟多少ms
   console.log('1111');
   return;
}
//执行sleep(1000)之后，休眠了1000ms之后输出了1111。上述循环的方式缺点很明显，容易造成死循环。
```

(2)通过promise来实现

```
function sleep(ms){
  var temple=new Promise(
  (resolve)=>{
  console.log(111);
  setTimeout(resolve,ms)
  });
  return temple
}
sleep(500).then(function(){
   //console.log(222)
})
//先输出了111，延迟500ms后输出222
```

(3)通过async封装

```
function sleep(ms){
  return new Promise((resolve)=>setTimeout(resolve,ms));
}
async function test(){
  var temple=await sleep(1000);
  console.log(1111)
  return temple
}
test();
//延迟1000ms输出了1111
```

(4)通过generate来实现

```
function* sleep(ms){
   yield new Promise(function(resolve,reject){
             console.log(111);
             setTimeout(resolve,ms);
         })
}
sleep(500).next().value.then(function(){console.log(2222)})
```

简单实现Node的Events模块

简介：观察者模式或者说订阅模式，他定义了对象之间的一对多的关系。让多个观察者对象同时监听某一个主题对象，当一个对象发送改变时，所有依赖于它的对象都将得到通知。

- (1)订阅模式的实现(eventEmitter发出say事件，通过On接收，并且输出结果，这就是一个订阅模式的实现)

```
var events=require('events');
var eventEmitter=new events.EventEmitter();
eventEmitter.on('say',function(name){
    console.log('Hello',name);
})
eventEmitter.emit('say','Jony yu');
```

- (2)实现一个Events模块的EventEmitter。事件模块的监听

1. 实现简单的Event模块的emit和on方法

```
function Events(){
this.on=function(eventName,callBack){
  if(!this.handles){
    this.handles={};
  }
  if(!this.handles[eventName]){
    this.handles[eventName]=[];
  }
  this.handles[eventName].push(callBack);
}
this.emit=function(eventName,obj){
   if(this.handles[eventName]){
     for(var i=0;o<this.handles[eventName].length;i++){
       this.handles[eventName][i](obj);
     }
   }
}
return this;
}
//调用
var events=new Events();
```

```
events.on('say',function(name){
    console.log('Hello',nama)
});
events.emit('say','Jony yu');
//结果就是通过emit调用之后，输出了Jony yu
```

2. 每个对象是独立的.因为是通过new的方式，每次生成的对象都是不相同的

```
var event1=new Events();
var event2=new Events();
event1.on('say',function(){
    console.log('Jony event1');
});
event2.on('say',function(){
    console.log('Jony event2');
})
event1.emit('say');
event2.emit('say');
//event1、event2之间的事件监听互相不影响
//输出结果为'Jony event1' 'Jony event2'
```

对HTML语义化标签的理解

HTML5语义化标签是指正确的标签包含了正确的内容，结构良好，便于阅读，比如nav表示导航条，类似的还有article、header、footer等等标签。

iframe是什么？有什么缺点？

定义：iframe元素会创建包含另一个文档的内联框架 提示：可以将提示文字放在之间，来提示某些不支持iframe的浏览器 缺点：

会阻塞主页面的onload事件 搜索引擎无法解读这种页面，不利于SEO iframe和主页面共享连接池，而浏览器对相同区域有限制所以会影响性能。

Doctype作用? 严格模式与混杂模式如何区分？它们有何意义?

Doctype声明于文档最前面，告诉浏览器以何种方式来渲染页面，这里有两种模式，严格模式和混杂模式。

严格模式的排版和 JS 运作模式是 以该浏览器支持的最高标准运行。 混杂模式，向后兼容，模拟老式浏览器，防止浏览器无法兼容页面。

click在ios上有300ms延迟，原因及如何解决？

(1)粗暴型，禁用缩放

```
<meta name="viewport" content="width=device-width,user-scalable=no">
```

(2)利用FastClick，其原理是检测到touchend事件后，立刻触发模拟click事件，并且把浏览器300毫秒之后真正触发的事件给阻断掉

一句话概括RESTFUL

就是用url定位资源，用http描述操作

讲讲viewport和移动端布局

响应式布局的常用解决方案对比(媒体查询、百分比、rem和vw/vh） 《像素》分为两种类型：css像素和物理像素。 a.我们在js或者css代码中使用的px单位就是指的是css像素. b.物理像素也称设备像素，只与设备或者说硬件有关，同样尺寸的屏幕，设备的密度越高，物理像素也就越多。 《视口》浏览器显示内容的屏幕区域，狭义的视口包括了布局视口、视觉视口和理想视口 在不缩放的情况下，一个css像素就对应一个dpr，也就是说，在不缩放 1 CSS像素 = 物理像素 / 分辨率 在移动端的布局中，我们可以通过viewport元标签来控制布局，比如一般情况下，我们可以通过下述标签使得移动端在理想视口下布局： initial-scale表示不缩放 maximum-scale最大2缩放比例 device-width分辨率的宽

```
<meta id="viewport" name="viewport" content="width=device-width; initial-
scale=1.0; maximum-scale=1; user-scalable=no;">
```

清除浮动 一是利用 clear 属性 二是触发浮动元素父元素的 BFC (Block Formatting Contexts, 块级格式化上下文)，使到该父元素可以包含浮动元素

(1).clear{clear:both;} (2)给浮动元素的容器overflow:hidden; 可以清除浮动，另外在 IE6 中还需要触发 hasLayout ，例如为父元素设置容器宽高或设置 zoom:1。在添加overflow属性后，浮动元素又回到了容器层，把容器高度撑起，达到了清理浮动的效果。

```
.news {
  background-color: gray;
  border: solid 1px black;
  overflow: hidden;
  *zoom: 1;
  }
.news img {
  float: left;
  }
.news p {
  float: right;
  }
```

(3)使用邻接元素处理

什么都不做，给浮动元素后面的元素添加clear属性。 (4)使用CSS的:after伪元素

盒模型

标准盒模型 box-sizing:content-box 宽度为content+padding+border IE盒模型：box-sizing:border-box width

画一条0.5px的线

采用meta viewport的方式

采用 border-image的方式

采用transform: scale()的方式

link标签和import标签的区别

link属于html标签，而@import是css提供的 页面被加载时，link会同时被加载，而@import引用的css会等到页面加载结束后加载。 link是html标签，因此没有兼容性，而@import只有IE5以上才能识别。 link方式样式的权重高于@import的。

transition和animation的区别

Animation和transition大部分属性是相同的，他们都是随时间改变元素的属性值，他们的主要区别是transition需要触发一个事件才能改变属性，而animation不需要触发任何事件的情况下才会随时间改变属性值，并且transition为2帧，从from .... to，而animation可以一帧一帧的。

Flex布局(弹性布局) 圣杯布局

- 容器属性和元素属性 容器的属性 flex-direction：决定主轴的方向（即子item的排列方法） .box { flex-direction: row | row-reverse | column | column-reverse; } flex-wrap：决定换行规则 .box{ flex-wrap: nowrap | wrap | wrap-reverse; } flex-flow： .box { flex-flow: || ; } justify-content：对其方式，水平主轴对齐方式 align-items：对齐方式，竖直轴线方向
- 项目的属性（元素的属性）： order属性：定义项目的排列顺序，顺序越小，排列越靠前，默认为0 flex-grow属性：定义项目的放大比例，即使存在空间，也不会放大 flex-shrink属性：定义了项目的缩小比例，当空间不足的情况下会等比例的缩小，如果定义个item的flow-shrink为0，则为不缩小 flex-basis属性：定义了在分配多余的空间，项目占据的空间。 flex：是flex-grow和flex-shrink、flex-basis的简写，默认值为0 1 auto。 align-self：允许单个项目与其他项目不一样的对齐方式，可以覆盖align-items，默认属性为auto，表示继承父元素的align-items

BFC（块级格式化上下文，用于清楚浮动，防止margin重叠等）

直译成：块级格式化上下文，是一个独立的渲染区域，并且有一定的布局规则。

BFC区域不会与float box重叠 BFC是页面上的一个独立容器，子元素不会影响到外面 计算BFC的高度时，浮动元素也会参与计算

那些元素会生成BFC：

根元素 float不为none的元素 position为fixed和absolute的元素 display为inline-block、table-cell、table-caption，flex，inline-flex的元素 overflow不为visible的元素

关于js动画和css3动画的差异性

功能涵盖面，js比css大 实现/重构难度不一，CSS3比js更加简单，性能跳优方向固定 对帧速表现不好的低版本浏览器，css3可以做到自然降级 css动画有天然事件支持 css3有兼容性问题

margin塌陷如何解决

垂直之间塌陷的原则是以两盒子最大的外边距为准。 解决方法： （1）为父盒子设置border，为外层添加border后父子盒子就不是真正意义上的贴合。 （2）为父盒子添加overflow：hidden； （3）为父盒子设定padding值。

visibility=hidden, opacity=0，display:none的区别

opacity=0，该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定一些事件，如click事件，那么点击该区域，也能触发点击事件的visibility=hidden，该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件display=none，把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉一样。

多行元素的文本省略号

```
 display: -webkit-box
-webkit-box-orient:vertical
-web-line-clamp:3
overflow:hidden
```

居中

```
text-align: center;
margin:0 auto;
.flex-center{
    display:flex;
    justify-content:center;
}
```

- 有时候行内元素或者文字显示为垂直居中，仅仅是因为它们的上下内边距相等 如,适用于单行

```
.link{
    padding-top:30px;
    padding-bottom:30px;
}
<!-- 或者 -->
.center-text-trick {
  height: 100px;
  line-height: 100px;
  white-space: nowrap;
}
```

- 多行呢？可以使用table

```

```

```
.center-table {
  display: table;
  height: 250px;
  background: white;
  width: 240px;
  margin: 20px;
}
.center-table p {
  display: table-cell;
  margin: 0;
  background: black;
  color: white;
  padding: 20px;
  border: 10px solid white;
  vertical-align: middle;
}
.flex-center-vertically {
  display: flex;
  justify-content: center;
  flex-direction: column;
  height: 400px;
}
```

- 块级元素
- 高度知道时用margin负值法
- 元素高度未知用transform

```
.parent {
  position: relative;
}
.child {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
<!-- 或者用flex布局 -->
.parent {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Cookie、sessionStorage、localStorage的区别？

共同点：都是保存在浏览器端，并且是同源的

Cookie：cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递。而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下,存储的大小很小只有4K左右。（key：可以在浏览器和服务器端来回传递，存储容量小，只有大约4K左右）

sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持，localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。（key：本身就是一个回话过程，关闭浏览器后消失，session为一个回话，当页面不同即使是同一页面打开两次，也被视为同一次回话）

localStorage：localStorage 在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的。（key：同源窗口都会共享，并且不会失效，不管窗口或者浏览器关闭与否都会始终生效）

补充说明一下cookie的作用：

保存用户登录状态。例如将用户id存储于一个cookie内，这样当用户下次访问该页面时就不需要重新登录了，现在很多论坛和社区都提供这样的功能。 cookie还可以设置过期时间，当超过时间期限后，cookie就会自动消失。因此，系统往往可以提示用户保持登录状态的时间：常见选项有一个月、三个 月、一年等。

跟踪用户行为。例如一个天气预报网站，能够根据用户选择的地区显示当地的天气情况。如果每次都需要选择所在地是烦琐的，当利用了 cookie后就会显得很人性化了，系统能够记住上一次访问的地区，当下次再打开该页面时，它就会自动显示上次用户所在地区的天气情况。因为一切都是在后 台完成，所以这样的页面就像为某个用户所定制的一样，使用起来非常方便

定制页面。如果网站提供了换肤或更换布局的功能，那么可以使用cookie来记录用户的选项，例如：背景色、分辨率等。当用户下次访问时，仍然可以保存上一次访问的界面风格。

补充400和401、403状态码

(1)400状态码：请求无效 产生原因：

前端提交数据的字段名称和字段类型与后台的实体没有保持一致 前端提交到后台的数据应该是json字符串类型，但是前端没有将对象JSON.stringify转化成字符串。

解决方法：

对照字段的名称，保持一致性 将obj对象通过JSON.stringify实现序列化

(2)401状态码：当前请求需要用户验证 (3)403状态码：服务器已经得到请求，但是拒绝执行 12.fetch发送2次请求的原因 fetch发送post请求的时候，总是发送2次，第一次状态码是204，第二次才成功？ 原因很简单，因为你用fetch的post请求的时候，导致fetch 第一次发送了一个Options请求，询问服务器是否支持修改的请求头，如果服务器支持，则在第二次中发送真正的请求。

http2.0

首先补充一下，http和https的区别，相比于http,https是基于ssl加密的http协议 简要概括：http2.0是基于1999年发布的http1.0之后的首次更新。

提升访问速度（可以对于，请求资源所需时间更少，访问速度更快，相比http1.0） 允许多路复用：多路复用允许同时通过单一的HTTP/2连接发送多重请求-响应信息。改善了：在http1.1中，浏览器客户端在同一时间，针

对同一域名下的请求有一定数量限制（连接数量），超过限制会被阻塞。 二进制分帧：HTTP2.0会将所有的传输信息分割为更小的信息或者帧，并对他们进行二进制编码 首部压缩 服务器端推送

HTML5 drag api

dragstart：事件主体是被拖放元素，在开始拖放被拖放元素时触发，。 darg：事件主体是被拖放元素，在正在拖放被拖放元素时触发。 dragenter：事件主体是目标元素，在被拖放元素进入某元素时触发。 dragover：事件主体是目标元素，在被拖放在某元素内移动时触发。 dragleave：事件主体是目标元素，在被拖放元素移出目标元素是触发。 drop：事件主体是目标元素，在目标元素完全接受被拖放元素时触发。 dragend：事件主体是被拖放元素，在整个拖放操作结束时触发

几个很实用的BOM属性对象方法?bom是浏览器对象。包括location对象、navigator对象、history对象

二、location对象

location.href-- 返回或设置当前文档的URL location.search -- 返回URL中的查询字符串部分。例如 http://www.dreamdu.com/dreamdu.php?id=5&name=dreamdu 返回包括(?)后面的内容? id=5&name=dreamdu location.hash -- 返回URL#后面的内容，如果没有#，返回空 location.host -- 返回URL中的域名部分，例如www.dreamdu.com location.hostname -- 返回URL中的主域名部分，例如dreamdu.com location.pathname -- 返回URL的域名后的部分。例如 http://www.dreamdu.com/xhtml/ 返回/xhtml/ location.port -- 返回URL中的端口部分。例如 http://www.dreamdu.com:8080/xhtml/ 返回8080 location.protocol -- 返回URL中的协议部分。例如 http://www.dreamdu.com:8080/xhtml/ 返回(//)前面的内容 http: location.assign -- 设置当前文档的URL location.replace() -- 设置当前文档的URL，并且在history对象的地址列表中移除这个URL location.replace(url); location.reload() -- 重载当前页面

二、Navigator对象

navigator.userAgent -- 返回用户代理头的字符串表示(就是包括浏览器版本信息等的字符串) navigator.cookieEnabled -- 返回浏览器是否支持(启用)cookie

三、history对象

history.go() -- 前进或后退指定的页面数 history.go(num); history.back() -- 后退一页 history.forward() -- 前进一页

web Quality （无障碍）

能够被残障人士使用的网站才能称得上一个易用的（易访问的）网站。 残障人士指的是那些带有残疾或者身体不健康的用户。 如使用alt属性：

```
<img src="person.jpg"  alt="this is a person"/>
```

有时候浏览器会无法显示图像。具体的原因有： 用户关闭了图像显示 浏览器是不支持图形显示的迷你浏览器 浏览器是语音浏览器（供盲人和弱视人群使用） 如果您使用了 alt 属性，那么浏览器至少可以显示或读出有关图像的描述。

一个图片url访问后直接下载怎样实现？

请求的返回头里面，用于浏览器解析的重要参数就是OSS的API文档里面的返回http头，决定用户下载行为的参数。

下载的情况下：

```
1. x-oss-object-type:
        Normal
2. x-oss-request-id:
        598D5ED34F29D01FE2925F41
3. x-oss-storage-class:
        Standard
```

HTTP请求的方式，HEAD方式

head：类似于get请求，只不过返回的响应中没有具体的内容，用户获取报头 options：允许客户端查看服务器的性能，比如说服务器支持的请求方式等等。

WebSocket是什么样的协议，具体有什么优点？

(1)什么是WebSocket? WebSocket是HTML5中的协议，支持持久连续，http协议不支持持久性连接。Http1.0和HTTP1.1都不支持持久性的链接，HTTP1.1中的keep-alive，将多个http请求合并为1个 (2)WebSocket是什么样的协议，具体有什么优点？ HTTP的生命周期通过Request来界定，也就是Request一个Response，那么在Http1.0协议中，这次Http请求就结束了。在Http1.1中进行了改进，是的有一个connection：Keep-alive，也就是说，在一个Http连接中，可以发送多个Request，接收多个Response。但是必须记住，在Http中一个Request只能对应有一个Response，而且这个Response是被动的，不能主动发起。 WebSocket是基于Http协议的，或者说借用了Http协议来完成《一部分握手》，在《握手阶段》与Http是相同的。我们来看一个websocket握手协议的实现，基本是2个属性，upgrade，connection。

TCP和UDP的区别

（1）TCP是面向连接的，udp是无连接的即发送数据前不需要先建立链接。 （2）TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付。 并且因为tcp可靠，面向连接，不会丢失数据因此适合大数据量的交换。 （3）TCP是面向字节流，UDP面向报文，并且网络出现拥塞不会使得发送速率降低（因此会出现丢包，对实时的应用比如IP电话和视频会议等）。 （4）TCP只能是1对1的，UDP支持1对1,1对多。 （5）TCP的首部较大为20字节，而UDP只有8字节。 （6）TCP是面向连接的可靠性传输，而UDP是不可靠的。

冒泡排序

比较相邻的两个元素，如果前一个比后一个大，则交换位置。 第一轮的时候最后一个元素应该是最大的一个。 按照步骤一的方法进行相邻两个元素的比较，这个时候由于最后一个元素已经是最大的了，所以最后一个元素不用比较。

```
function bubble_sort(arr){
  for(var i=0;i<arr.length-1;i++){
    for(var j=0;j<arr.length-i-1;j++){
      if(arr[j]>arr[j+1]){
        var swap=arr[j];
```

```
        arr[j]=arr[j+1];
        arr[j+1]=swap;
      }
    }
  }
}

var arr=[3,1,5,7,2,4,9,6,10,8];
bubble_sort(arr);
console.log(arr);
```

如何实现函数节流和防抖

```
//节流
function throttle(fn, delay) {
  delay = delay || 50
  let statTime = 0
  return function () {
    statTime === 0 && fn.apply(this, arguments)
    let currentTime = new Date()
    if (currentTime - statTime > delay) {
      fn.apply(this, arguments)
      statTime = currentTime
    }
  }
}
let throttleFn = throttle(fn)
throttleFn()//只会执行一次
throttleFn()
throttleFn()
throttleFn()
//防抖

寒东设计师
2018年05月18日阅读 6589
前端基础面试题@JS篇
说说Js的数据类型都有哪些
基本类型
String
Number
Boolean
null
undefined
symbol
引用类型
object
说说Http状态码
1** 信息，服务器收到请求，需要请求者继续执行操作(101，升级为websocket协议)
2** 成功，操作被成功接收并处理(206,部分内容，分段传输)
3** 重定向，需要进一步操作以完成请求(301,302重定向；304命中缓存)
4** 客户端错误，请求包含语法错误或无法完成请求(401,要求身份验证；403，服务器理解客服端需
```

求，但是禁止访问)
5** 服务器错误，服务器在处理请求的过程中发生了错误

说说ajax状态码，ajax一定是异步的吗？
ajax不一定是异步的，可以通过open方法的第三个参数来配置(默认为true，异步)

状态码：

0 - (未初始化)还没有调用send()方法
1 - (载入)已调用send()方法，正在发送请求
2 - (载入完成)send()方法执行完成
3 - (交互)正在解析响应内容
4 - (完成)响应内容解析完成，可以在客户端调用了

说说ajax是什么？优势？劣势？应该注意的问题？
ajax是一种和后台通信的标准。全称是Asynchronous Javascript And XML(异步javascript和XML)。

优势：

无需刷新页面请求数据，可以使产品更快、更小、更友好
可以把以前服务端的任务转嫁到客户端来处理，减轻服务器负担，节省带宽
浏览器支持好，无需插件
劣势：

不支持浏览器的回退按钮
安全性存在问题，能够在用户不知情的情况下发送请求
暴露了http交互细节
对搜索引擎(网络爬虫)的支持比较弱
程序不容易调试
注意的问题：

浏览器兼容性问题，这个问题jQuery等库已经帮我们封装好了
跨域问题，不同域之间不允许通过ajax进行访问，可以参考阮一峰老师的跨域资源共享 CORS 详解
为了更快的速度和对搜索引擎友好，首页尽量不要用ajax而是服务端渲染(当然这看分场景)
ajax适合增删改查操作
你把下面的表达式的打印结果写出来

```
1.toString()  //Uncaught SyntaxError: Invalid or unexpected token
true.toString()  //"true"
[].toString()  //""
{}.toString()  //Uncaught SyntaxError: Unexpected token .
null.toString()  //Uncaught TypeError: Cannot read property 'toString' of null
undefined.toString()  //Uncaught TypeError: Cannot read property 'toString' of undefined
NaN.toString()   //"NaN"
```

复制代码
这些需要刻意背一下，其中1和{}是语法错误。null和undefined是因为没有toString方法,可以使用call来借用(想详细了解，可以到评论区看我如何被骂的)：

```
1..toString()   //"1"
(1).toString()  //"1"
Number(1).toString()  //"1"
({}).toString()  //[object Object]
Object.prototype.toString.call(null)   //[object Null]
```

```
Object.prototype.toString.call(undefined)   //[object Undefined]
```

复制代码

前端性能优化你了解哪些

内容层面

使用CDN

单域名、多域名，单域名可以减少DNS查找次数，多域名可以增加浏览器并行下载数量，这需要权衡，一般同一个域下不要超过四个资源。

避免重定向(分场景)

避免404

网络层面

利用缓存，可以参考另一篇文章手写文件服务器，说说前后端交互

文件压缩(通过响应头Accept-Encoding: gzip, deflate, br告诉服务器你支持的压缩类型)

按需加载，提取公共代码，tree-shaking等(都可以通过webpack来实现)

减少cookie大小

文件合并，通过css雪碧图合并图片

文件预加载、图片懒加载

渲染层间

js放底部，css放顶部

减少reflow(回流)和repaint(重绘)

减少dom节点

代码层面

缓存dom节点，减少节点查找，css选择器层级优化

减少dom节点操作

合理使用break、continue、return等，优化循环

像react用到的事件委托、对象池等手段

说说浏览器的reflow和repaint

浏览器解析过程

解析html生成dom树

解析css

把css应用于dom树，生成render树(这里记录这每一个节点和它的样式和所在的位置)

把render树渲染到页面

reflow(回流)

reflow翻译为回流，指的是页面再次构建render树。每个页面至少发生一次回流，就是第一次加载页面的时候

此外，当页面中有任何改变可能造成文档结构发生改变(即元素间的相对或绝对位置改变)，都会发生reflow，常见的有：

添加或删除元素(opacity:0除外，它不是删除)

改变某个元素的尺寸或位置

浏览器窗口改变(resize事件触发)

repaint(重绘)

repaint翻译为重绘，它可以类比为上面的第四步，根据render树绘制页面，它的性能损耗比回流要小。每次回流一定会发生重绘。此外，以下操作(不影响文档结构的操作，影响结构的会发生回流)也会发生重绘：

元素的颜色、透明度改变

text-align等

浏览器优化

我们不太容易精确知道哪些操作具体会造成哪些元素回流，不同的浏览器都有不同的实现。但是确定是他们的的耗时是比较长的，因为涉及到大量的计算。

浏览器为了提升性能也对这个问题进行了优化。方案就是维护一个队列,把所有需要回流和重绘的操作都缓存起来，一段时间之后再统一执行。但是，有的时候我们需要获取一些位置属性，当我们一旦调用

这些api的时候，浏览器不得不立即计算队列以保证提供的数据是准确的。例如以下操作：

offsetTop, offsetLeft, offsetWidth, offsetHeight
scrollTop/Left/Width/Height
clientTop/Left/Width/Height
width,height
getComputedStyle或者IE的currentStyle
注意问题
批量处理
使用DocumentFragment进行缓存，这样只引发一次回流
把频繁操作的元素先display：null，只引发两次回流
cloneNode和replaceChild，只引发两次回流
不要频繁更改style，而是更改class
避免频繁调用offsetTop等属性，在循环前把它缓存起来
绝对定位具有复杂动画的元素，否则会引起父元素和后续大量元素的频繁回流
如何去除字符串首位空格？

```
//es6
' ab '.trim()      //"ab"
//正则
' ab '.replace(/^\s*|\s*$/g,'')  //"ab"
```

复制代码
如何获取url中的查询字符串

```
function queryUrlParameter(str) {
    let obj = {}
    let reg = /([^?=&#]+)=([^?=&#]+)/g;
    str.replace(reg, function () {
        obj[arguments[1]] = arguments[2]
    })
    //如果加上hash
    // reg = /#([^?&=#]+)/g
    // if (reg.test(str)) {
    //     str.replace(reg, function () {
    //         obj.hash = arguments[1]
    //     })
    // }
    return obj
}
console.log(queryUrlParameter('http://www.baidu.com?a=1&b=2#12222'))  //{ a: '1',
b: '2'}
```

复制代码
如何实现一个深拷贝、深比较
深拷贝

```
function clone(obj) {
  if (obj == null || typeof obj !== 'object') return obj

  let newObj = null

  // 时间对象有特殊性
  if (obj.constructor === Date) {
    newObj = new obj.constructor(obj)
  } else {
    newObj = obj.constructor()
  }
```

```
  for (let key in Object.getOwnPropertyDescriptors(obj)) {
    newObj[key] = clone(obj[key])
  }
  return newObj
}
```

复制代码

深比较

```
function deepCompare(a, b){
  if(a === null
    || typeof a !== 'object'
    || b === null
    || typeof b !== 'object'){
    return a === b
  }

  const propsA = Object.getOwnPropertyDescriptors(a)
  const propsB = Object.getOwnPropertyDescriptors(b)
  if(Object.keys(propsA).length !== Object.keys(propsB).length){
    return false
  }

  return Object.keys(propsA).every( key => deepCompare(a[key], b[key]))

}
```

复制代码

如何实现函数节流和防抖

节流

```
function throttle(fn, delay) {
  delay = delay || 50
  let statTime = 0
  return function () {
    statTime === 0 && fn.apply(this, arguments)
    let currentTime = new Date()
    if (currentTime - statTime > delay) {
      fn.apply(this, arguments)
      statTime = currentTime
    }
  }
}

let throttleFn = throttle(fn)

throttleFn()//只会执行一次
throttleFn()
throttleFn()
throttleFn()
```

复制代码

防抖

```
function debounce(fn, delay) {
  delay = delay || 50
  let timer = null
  return function () {
```

```
      let self = this
      clearTimeout(timer)
      timer = setTimeout(fn.bind(self, arguments), delay);
    }
  }
```

说说浏览器的（reflow回流）和（repaint重绘）

1. 浏览器解析过程

解析html生成dom树 解析css 把css应用于dom树，生成render树(这里记录这每一个节点和它的样式和所在的位置) 把render树渲染到页面

2. reflow(回流) reflow翻译为回流，指的是页面再次构建render树。每个页面至少发生一次回流，就是第一次加载页面的时候 此外，当页面中有任何改变可能造成文档结构发生改变(即元素间的相对或绝对位置改变)，都会发生reflow，常见的有：

添加或删除元素(opacity:0除外，它不是删除) 改变某个元素的尺寸或位置 浏览器窗口改变(resize事件触发)

3. repaint(重绘) repaint翻译为重绘，它可以类比为上面的第四步，根据render树绘制页面，它的性能损耗比回流要小。每次回流一定会发生重绘。此外，以下操作(不影响文档结构的操作，影响结构的会发生回流)也会发生重绘：

元素的颜色、透明度改变 text-align等

.ajax 和 jsonp 的区别？

ajax和jsonp的区别： 相同点：都是请求一个url 不同点：ajax的核心是通过xmlHttpRequest获取内容 jsonp的核心则是动态添加