服务端渲染的好处有哪些?

1、服务端和客户端可以共享某些代码，避免重复定义，这样可以使结构清晰，增加可维护性

2、首次加载页面的速度加快

3、SEO，服务器渲染可以让搜索引擎更容易读取页面的meta信息以及其他SEO相关信息，大大增加网站在搜索引擎中的可见度

从输入URL到页面加载发生了什么?

1. DNS解析：DNS解析是一个递归查询的过程所有网址真正的解析过程为: . -> .com -> google.com. -> www.google.com
2. TCP连接
3. 发送HTTP请求
4. 服务器处理请求并返回HTTP报文
5. 浏览器解析渲染页面
6. 连接结束

```
//a++ 是先返回a的值，再执行++运算。
var a=2
var b=a++;
console.log(a,b)// 3 2
//++a 是先执行++运算，在返回a的值
var a=2
var b=++a;
console.log(a,b)//3,3
var a=b=2
console.log(a,b)//2 2 b没有声明，只取默认值
```

在标准的 JavaScript 中， Ajax 异步执行调用基于下面哪一个机制才能实现

Event和callback

优化首屏加载速度

1、JavaScript外联文件引用放在html文档底部；CSS外联文件引用在html文档头部，位于header内；

2、http静态资源尽量用多个子域名；

3、服务器端提供html文档和http静态资源时，尽量开启gzip压缩；

4、在js，css，image等资源响应的http headers里，设置expires，last-modified；

5、尽量减少HTTP Requests的数量；

6、js/css的minify：可统一通过combo handler做到压缩加合并；

7、减少不必要的301/302跳转：别让页面打开时间浪费在302多次跳转上（每次可能几十毫秒）；
8、请大量使用CSS Spirtes：这样做可以大大地减少CSS背景图片的HTTP请求次数；
9、首屏不需要展示的较大尺寸图片，请使用**懒加载(lazyload)**；
10、图片无损压缩的优化；
11、避免404错误：尽量减少外联js；
12、减少cookies的大小：尽量减少cookies的体积对减少用户获得响应的时间十分重要；
13、使用无cookies的域；
14、减少DOM Elements的数量；
15、引入textarea/script元素做延迟解析异步渲染

JS性能优化方法：

https://www.jianshu.com/p/ba9759384ecf

1、动态加载

```
function loadJS(src) {
  const script = document.createElement('script');
  script.src = src;
  document.getElementsByTagName('head')[0].appendChild(script);
}
loadJS('http://example.com/scq000.js');
```

2、异步加载

```
<script type="text/javascript" src="./a.js" async></script>
<script type="text/javascript" src="./b.js" defer></script>
```

3、懒加载的原理：

原理：当访问一个页面的时候，先把img元素或是其他元素的背景图片路径替换成一张大小为1*1px图片的路径（这样就只需请求一次，俗称占位图），只有当图片出现在浏览器的可视区域内时，才设置图片正真的路径，让图片显示出来。这就是图片懒加载。

4、webpack实现脚本加载优化

import()方法、require.ensure()

```
//或者更优雅的写法
$('#button').click(async function() {
    const dialog = await import('./dialog.js');
  //do something with dialog

});
```

```
const routes = [{path: '/comment', component: r => require.ensure([], r(require('./Comment')), 'comment')}];
```

5、预加载:**提前加载图片，当用户需要查看时可直接从本地缓存中渲染**
6、节流函数延时请求

```
//简单的节流函数
function throttle(func,wait,mustRun){
    var timeout,startTime=new Date();
    return function(){
        var context=this,
        args=arguments,
        curTime=new Date();

        clearTimeout(timeout);
        //如果达到了规定的触发时间间隔，触发handler
        if(curTime-startTime>=mustRun){
            func.apply(contentx,args);
            startTime=curTime;
            //没达到触发问题，重新设定定时器

        }else{
            timeout=setTimeout(func,wait);
        }
    }
}
```

为什么把外链的script放在body后面，放在前面不行吗，为上面把link标签放在head标签里面，放在body后面不行吗，window.onload为什么不放在body后面的script标签里面

这样处理的好处是无需担心因页面未完成加载，造成DOM节点获取不到，使脚本报错的问题，而且能避免因脚本运行缓慢造成页面卡死的问题，所以一般要用window.onload来判断页面是否载入完成。

window.onload和$.ready()的区别。

$(document).ready:是DOM结构绘制完毕后就执行，不必等到加载完毕。意思就是DOM树加载完毕，就执行，不必等到页面中图片或其他外部文件都加载完毕。并且可以写多个.ready。

window.onload:是页面所有元素都加载完毕，包括图片等所有元素。只能执行一次。

script 的位置是否会影响首屏显示时间?

- 在解析 HTML 生成 DOM 过程中，js 文件的下载是并行的，不需要 DOM 处理到 script 节点。因此，script的位置不影响首屏显示的开始时间。
- 浏览器解析 HTML 是自上而下的线性过程，script作为 HTML 的一部分同样遵循这个原则
- 因此，script 会延迟 DomContentLoad，只显示其上部分首屏内容，从而影响首屏显示的完成时间

答案是：不影响但有可能截断首屏的内容，使其只显示上面一部分

异步加载和延迟加载

1.异步加载的方案： 动态插入script标签

2.通过ajax去获取js代码，然后通过eval执行

3.script标签上添加defer或者async属性

4.创建并插入iframe，让它异步执行js

5.延迟加载：有些 js 代码并不是页面初始化的时候就立刻需要的，而稍后的某些情况才需要的。

defer和async

- defer并行加载js文件，会按照页面上script标签的顺序执行async并行加载js文件，下载完成立即执行，不会按照页面上script标签的顺序执行

js延迟加载的方式有哪些?

- defer和async、动态创建DOM方式（用得最多）、按需异步载入js

异步加载JS的方式有哪些?

- defer，只支持IE
- async：
- 创建script，插入到DOM中，加载完毕后callBack

async都有哪些常用方法，分别是怎么用?

async是一个js类库，它的目的是解决js中异常流程难以控制的问题．async不仅适用在node.js里，浏览器中也可以使用． 1) async.parallel并行执行完多个函数后，调用结束函数

```
async.parallel([
    function () {},
    function () {}
], callback);
```

async.series串行执行完多个函数后，调用结束函数

```
    async.series([
    function(){  },
    function(){  }
]);
```

async.waterfall依次执行多个函数，后一个函数以前面函数的结果作为输入参数

```
async.waterfall([
    function (callback) {
        callback(null, 'one', 'two');
    },
    function (arg1, arg2, callback) {
        // arg1 now equals 'one' and arg2 now equals 'two'
        callback(null, 'three');
    },
    function (arg1, callback) {
        // arg1 now equals 'three'
        callback(null, 'done');
    }
], function (err, result) {
    // result now equals 'done'
});
```

async.map异步执行多个数组，返回结果数组

```
async.map(['file1', 'file2', 'file3'], fs.stat, function (err, results) {
    // results is now an array of stats for each file
});
```

async.filter异步过滤多个数组，返回结果数组

```
 async.filter(['file1', 'file2', 'file3'], fs.exists, function (results) {
     // results now equals an array of the existing files
 });
```

express项目的目录大致是什么样子的

!QAZ2wsx

app.js, package.json, bin/www, public, routes, views.

express常用函数

express.Router路由组件,app.get路由定向，app.configure配置，app.set设定参数,app.use使用中间件

mongodb有哪些常用优化措施

类似传统数据库，索引和分区

redis支持哪些功能

set/get, hset/hget, publish/subscribe, expire

redis最简单的应用

```
var redis = require("redis"),
    client = redis.createClient();
client.set("foo_rand000000000000", "some fantastic value");
client.get("foo_rand000000000000", function (err, reply) {
    console.log(reply.toString());
});
client.end();
```

apache,nginx有什么区别?

二者都是代理服务器，功能类似．apache应用简单，相当广泛．nginx在分布式，静态转发方面比较有优势．

documen.write和 innerHTML的区别

- document.write只能重绘整个页面
- innerHTML可以重绘页面的一部分

***defineProperty, hasOwnProperty, isEnumerable*都是做什么用的?**

Object.defineProperty(obj, prop, descriptor)用来给对象定义属性,有value,writable,configurable,enumerable,set/get等.

hasOwnProerty用于检查某一属性是不是存在于对象本身，继承来的父亲的属性不算.

isEnumerable用来检测某一属性是否可遍历，也就是能不能用for..in循环来取到.

一个纯JS日历控件的原理和流程

1、先计算闰年的自编函数

- **2、计算处理月第一天是星期几**

Js获取当前日期时间及其它操作

```
var myDate = new Date();
myDate.getYear();        //获取当前年份(2位)
myDate.getFullYear();    //获取完整的年份(4位,1970-????)
myDate.getMonth();       //获取当前月份(0-11,0代表1月)
myDate.getDate();        //获取当前日(1-31)
myDate.getDay();         //获取当前星期X(0-6,0代表星期天)
myDate.getTime();        //获取当前时间(从1970.1.1开始的毫秒数)
myDate.getHours();       //获取当前小时数(0-23)
myDate.getMinutes();     //获取当前分钟数(0-59)
myDate.getSeconds();     //获取当前秒数(0-59)
myDate.getMilliseconds();    //获取当前毫秒数(0-999)
myDate.toLocaleDateString();     //获取当前日期
var mytime=myDate.toLocaleTimeString();     //获取当前时间
myDate.toLocaleString( );        //获取日期与时间
```

日期时间脚本库方法列表

Date.prototype.isLeapYear 判断闰年
Date.prototype.Format 日期格式化
Date.prototype.DateAdd 日期计算
Date.prototype.DateDiff 比较日期差
Date.prototype.toString 日期转字符串
Date.prototype.toArray 日期分割为数组
Date.prototype.DatePart 取日期的部分信息
Date.prototype.MaxDayOfDate 取日期所在月的最大天数
Date.prototype.WeekNumOfYear 判断日期所在年的第几周
StringToDate 字符串转日期型
IsValidDate 验证日期有效性
CheckDateTime 完整日期时间检查
daysBetween 日期天数差

```
function is_leap(year) {
    return (year%100==0?res=(year%400==0?1:0):res=(year%4==0?1:0));
} //是否为闰年
var nstr=new Date(); //当前Date资讯
var ynow=nstr.getFullYear(); //年份
var mnow=nstr.getMonth(); //月份
var dnow=nstr.getDate(); //今日日期
var n1str=new Date(ynow,mnow,1); //当月第一天Date资讯
var firstday=n1str.getDay(); //当月第一天星期几
var m_days=new Array(31,28+is_leap(ynow),31,30,31,30,31,31,30,31,30,31); //各月份的总天数
var tr_str=Math.ceil((m_days[mnow] + firstday)/7); //表格所需要行数
//打印表格第一行（有星期标志）
document.write ("<table border='1' align='center' width='220' cellspacing='0'>" +
    "<tr>" +
    "<td align='center'>日</td>" +
    "<td align='center'>一</td>" +
    "<td align='center'>二</td>" +
    "<td align='center'>三</td>" +
    "<td align='center'>四</td>" +
    "<td align='center'>五</td>" +
    "<td align='center'>六</td>" +
    "</tr>");
for(i=0;i<tr_str;i++) { //表格的行
    document.write("<tr>");
    for(k=0;k<7;k++) { //表格每行的单元格
        idx=i*7+k; //单元格自然序列号
        date_str=idx-firstday+1; //计算日期
        (date_str≤0 || date_str>m_days[mnow]) ? date_str=" " : date_str=idx-firstday+1; //过滤无效日期（小于等于零的、大于月总天数的
        //打印日期：今天底色为红
        date_str==dnow ? document.write ("<td align='center' bgcolor='red'>" + date_str + "</td>")
            : document.write ("<td align='center'>" + date_str + "</td>");
    }
    document.write("</tr>"); //表格的行结束
}
document.write("</table>"); //表格结束
```

一个局部滚动的实现方法？：给一个固定的高度div，overflow-y:scroll，

-webkit-overflow-scrolling: touch;

::webkit-scrollbar{

Width:0;height:0;color:transparent;

}

浏览器从加载到渲染的过程，比如输入一个网址到显示页面的过程。（考察频率:高）

加载过程：

- 浏览器根据 DNS 服务器解析得到域名的 IP 地址
- 向这个 IP 的机器发送 HTTP 请求
- 服务器收到、处理并返回 HTTP 请求
- 浏览器得到返回内容

渲染过程：

- 根据 HTML 结构生成 DOM 树
- 根据 CSS 生成 CSSOM
- 将 DOM 和 CSSOM 整合形成 RenderTree
- 根据 RenderTree 开始渲染和展示
- 遇到<script>时，会执行并阻塞渲染

HTTPS原理：

HTTPS能够加密信息，以免敏感信息被第三方获取。所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用HTTPS协议。HTTPS其实是有两部分组成：HTTP + SSL / TLS，也就是在HTTP上又加了一层处理加密信息的模块。服务端和客户端的信息传输都会通过TLS进行加密，所以传输的数据都是

加密后的数据

HTTPS要使客户端与服务器端的通信过程得到安全保证，必须使用的对称加密算法，但是协商对称加密算法的过程，需要使用非对称加密算法来保证安全，然而直接使用非对称加密的过程本身也不安全，会有中间人篡改公钥的可能性，所以客户端与服务器不直接使用公钥，而是使用数字证书签发机构颁发的证书来保证非对称加密过程本身的安全。这样通过这些机制协商出一个对称加密算法，就此双方使用该算法进行加密解密。从而解决了客户端与服务器端之间的通信安全问题。

重绘和回流（考察频率：中）

- 重绘：当页面中元素样式的改变并不影响它在文档流中的位置时（例如：color、background-color、visibility等），浏览器会将新样式赋予给元素并重新绘制它，这个过程称为重绘。
- 回流：当Render Tree（DOM）中部分或全部元素的尺寸、结构、或某些属性发生改变时，浏览器重新渲染部分或全部文档的过程称为回流。
- 回流要比重绘消耗性能开支更大。
- 回流必将引起重绘，重绘不一定会引起回流。

重排和重绘

部分渲染树（或者整个渲染树）需要重新分析并且节点尺寸需要重新计算。这被称为重排。注意这里至少会有一次重排-初始化页面布局。
由于节点的几何属性发生改变或者由于样式发生改变，例如改变元素背景色时，屏幕上的部分内容需要更新。这样的更新被称为重绘。

什么情况会触发重排和重绘

添加、删除、更新 DOM 节点
通过 display: none 隐藏一个 DOM 节点-触发重排和重绘
通过 visibility: hidden 隐藏一个 DOM 节点-只触发重绘，因为没有几何变化
移动或者给页面中的 DOM 节点添加动画
添加一个样式表，调整样式属性
用户行为，例如调整窗口大小，改变字号，或者滚动。

上拉刷新和下拉加载的实现逻辑

```
        document.addEventListener('scroll', this.handleScroll);
      }
    }
  }
}

handleScroll() {
  const {
    match, pageNum, allowRequest
  } = this.props;
  const bodyScrollHeight = document.body.scrollTop || document.documentElement.scrollTop;
  const documentHeight = document.body.offsetHeight || document.documentElement.offsetHeight;
  const winHeight = document.documentElement.clientHeight;
  if (documentHeight < bodyScrollHeight + winHeight + 100) {
    // 100是离底部的距离值，满足条件后去请求分页接口
    if (allowRequest) {
      this.props.getCountryDetail({
        pageNum,
        pageSize: 10,
        extraInfo: {
          nationNameCn: match.params.nationName
        }
      });
      this.countIspage();
```

下拉刷新：(scrollTop+clientHeight)-offsetHeight<100
上拉加载：(scrollTop+clientHeight)-offsetHeight>100
height clientHeight offsetHeight?

对于上面的div，它的border的height有多高？答案是200+10*2+1*2=222。

clientHeight: 可理解为内部可视区高度，样式的height+上下padding(本例即220）。
scrollHeight: 内容的实际高度+上下padding（如果没有限制div的height，即height是自适应的，一般是scrollHeight==clientHeight）
offsetHeight:可理解为div的可视高度，样式的height+上下padding+上下border-width。（本例是222）
height 这个变量在几个浏览器中都是undefined
style.height这个变量在本例中是'200px'，不过遗憾的是只有将高度定义在元素的style属性中这个变量才有效，如果是抽取到了样式表中是无法取到的。

clientTop: 容器内部相对于容器本身的top偏移，实际就是 上border-width（本例是1）
scrollTop: Y轴的滚动条没有，或滚到最上时，是0；y轴的滚动条滚到最下时是 scrollHeight-clientHeight（很好理解）
offsetTop: 可以理解为容器相对于document的top的绝对偏移。等于top+margin-top

滚动时通常我们只能scrollTop，当scrollTop为 0 到 scrollHeight-clientHeight 是正常的滚动距离，否则就是滚动过头了（手机上的上下拉取）！

ejs是什么?

EJS是一个JavaScript模板库，用来从JSON数据中生成HTML字符串。

单页应用的原理：

通过监听页面地址变化来调用前端MVC模块，渲染不同页面。当然，前端这时也拥有了修改URL的能力。管理好前端视图与URL，整个单页面就已经架构完成。(关键点:hmtl5使用historyAPI的支持，低级浏览器采用hash段的管理或无视)。

实现一个函数clone，可以对JavaScript中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制

```
Object.prototype.clone = function(){

            var o = this.constructor === Array ? [] : {};

            for(var e in this){

                    o[e] = typeof this[e] === "object" ? this[e].clone() : this[e];

            }

            return o;
    }
```

解释0.1 + 0.2 !== 0.3，如何解决

原因： 二进制来表示浮点数的，无法准确表示一个浮点数

解决： **转化成整数，结束后按比例相除 (1+2)/10=0.3**

怎么判断两个对象相等

```
JSON.stringify(obj)==JSON.stringify(obj);//true
```

封装一个函数，参数是定时器的时间，.then执行回调函数

```
function sleep (time) {
    return new Promise((resolve) => setTimeout(resolve, time));
}
```

HTML语义化:

1、什么是HTML语义化：

<基本上都是围绕着几个主要的标签，像标题（H1~H6）、列表（li）、强调（strong em）等等>根据内容的结构化（内容语义化），选择合适的标签（代码语义化）便于开发者阅读和写出更优雅的代码的同时让浏览器的爬虫和机器很好地解析。

2、**为什么要语义化?**

- 为了在没有CSS的情况下，页面也能呈现出很好地内容结构、代码结构:为了裸奔时好看；
- 用户体验：例如title、alt用于解释名词或解释图片信息、label标签的活用；
- 有利于SEO：和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息：爬虫依赖于标签来确定上下文和各个关键字的权重；
- 方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；
- 便于团队开发和维护，语义化更具可读性，是下一步吧网页的重要动向，遵循W3C标准的团队都遵循这个标准，可以减少差异化。

Doctype作用？标准模式与兼容模式各有什么区别？

- <!DOCTYPE>声明位于位于HTML文档中的第一行，处于 <html> 标签之前。告知浏览器的解析器用什么文档标准解析这个文档。DOCTYPE不存在或格式不正确会导致文档以兼容模式呈现
- 标准模式的排版 和JS运作模式都是以该浏览器支持的最高标准运行。在兼容模式中，页面以宽松的向后兼容的方式显示,模拟老式浏览器的行为以防止站点无法工作

行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？

- 行内元素有：a b span img input select strong（强调的语气）
- 块级元素有：div ul ol li dl dt dd h1 h2 h3 h4...p
- 常见的空元素:
 <hr> <img> <input> <link> <meta>

页面导入样式时，使用link和@import有什么区别?

- link属于XHTML标签，除了加载CSS外，还能用于定义RSS,定义rel连接属性等作用；而@import是CSS提供的，只能用于加载CSS
- 页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载
- import是CSS2.1 提出的，只在IE5以上才能被识别，而link是XHTML标签，无兼容问题

常见的浏览器内核有哪些？

- Trident内核：IE,MaxThon,TT,The World,360,搜狗浏览器等。[又称MSHTML]
- Gecko内核：Netscape6及以上版本，FF,MozillaSuite/SeaMonkey等
- Presto内核：Opera7及以上。[Opera内核原为：Presto，现为：Blink;]
- Webkit内核：Safari,Chrome等。[Chrome的Blink（WebKit的分支）]

html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分 HTML 和 HTML5？

- HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加
  - 绘画 canvas
  - 用于媒介回放的 video 和 audio 元素
  - 本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失
  - sessionStorage 的数据在浏览器关闭后自动删除

    - 语意化更好的内容元素，比如 article、footer、header、nav、section
    - 表单控件，calendar、date、time、email、url、search
    - 新的技术webworker, websocket, Geolocation
- 移除的元素：

- 纯表现的元素：basefont，big，center，font, s，strike，tt，u

- 对可用性产生负面影响的元素：frame，frameset，noframes

- 支持HTML5新标签：
    - IE8/IE7/IE6支持通过document.createElement方法产生的标签
    - 可以利用这一特性让这些浏览器支持HTML5新标签
    - 浏览器支持新标签后，还需要添加标签默认的样式
- 当然也可以直接使用成熟的框架、比如html5shim

```
<!--[if lt IE 9]>
<script> src="http://html5shim.googlecode.com
/svn/trunk/html5.js"</script><![endif]-->
```

- 如何区分HTML5： DOCTYPE声明\新增的结构元素\功能元素

HTML5的离线储存怎么使用，工作原理能不能解释一下？

- 在用户没有与因特网连接时，可以正常访问站点或应用，在用户与因特网连接时，更新用户机器上的缓存文件

- 原理：HTML5的离线存储是基于一个新建的.appcache文件的缓存机制(不是存储技术)，通过这个文件上的解析清单离线存储资源，这些资源就会像cookie一样被存储了下来。之后当网络在处于离线状态下时，浏览器会通过被离线存储的数据进行页面展示

- 如何使用：
    - 页面头部像下面一样加入一个manifest的属性；
    - 在cache.manifest文件的编写离线存储的资源
    - 在离线状态时，操作window.applicationCache进行需求实现

```
CACHE MANIFEST
  #v0.11
  CACHE:
  js/app.js
  css/style.css
  NETWORK:
  resourse/logo.png
  FALLBACK:
  / /offline.html
```

HTML5的form如何关闭自动完成功能?

- 给不想要提示的 form 或某个 input 设置为 autocomplete=off。

如何在页面上实现一个圆形的可点击区域?

- map+area或者svg
- border-radius
- 纯js实现 需要求一个点在不在圆上简单算法、获取鼠标坐标等等

实现不使用 border 画出1px高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果

iframe 有那些优点和缺点?

- 优点：
  - 用来加载速度较慢的内容（如广告）
  - 可以使脚本可以并行下载
  - 可以实现跨子域通信
- 缺点：
  - iframe 会阻塞主页面的 onload 事件
  - 无法被一些搜索引擎索识别
  - 会产生很多页面，不容易管理

label 的作用是什么？怎么使用的?

- label标签来定义表单控件的关系：
  - 当用户选择label标签时，浏览器会自动将焦点转到和label标签相关的表单控件上
- 使用方法1：
  - <label for="mobile">Number:</label>
  - <input type="text" id="mobile"/>
- 使用方法2：
  - <label>Date:<input type="text"/></label>

如何实现浏览器内多个标签页之间的通信?

- iframe + contentWindow
- postMessage
- SharedWorker(Web Worker API)
- storage 事件(localStorge API)
- WebSocket

如何实现浏览器内多个标签页之间的通信?(阿里)

- WebSocket、SharedWorker
- 也可以调用localstorge、cookies等本地存储方式

webSocket 如何兼容低浏览器?

- Adobe Flash Socket
- ActiveX HTMLFile (IE)
- 基于 multipart 编码发送 XHR
- 基于长轮询的 XHR

websocket长连接原理是什么

含义:Websocket是一个持久化的协议，相对于HTTP这种非持久的协议来说。

原理:类似长轮循长连接；发送一次请求；源源不断的得到信息

web开发中会话跟踪的方法有哪些

- cookie
- session
- url重写
- 隐藏input
- ip地址

title 与 h1 的区别、b 与 strong 的区别、i 与 em 的区别?

- **title** 表示是整个页面标题，**h1** 则表示层次明确的标题，对页面信息的抓取有很大的影响
- **strong** 标明重点内容，有语气加强的含义，使用阅读设备阅读网络时，**strong** 会重读，而 **b**是展示强调内容
- **i** 内容展示为斜体，**em** 表示强调的文本
- 自然样式标签：**b, i, u, s, pre**
- 语义样式标签：**strong, em, ins, del, code**
- 应该准确使用语义样式标签, 但不能滥用。如果不能确定时，首选使用自然样式标签

js基础数据类型，如何判断一个值的数据类型

1.基本数据类型（自身不可拆分的）：Undefined、Null、Boolean、Number、String

2.引用数据类型（对象）：Object （Array，Date，RegExp，Function）

```
function isArray(arr){
   return Object.prototype.toString.call(arr) === '[Object Array]';
}
```

```
function getType(nm){
    return Object.prototype.toString.call(nm);
}
```

js的两种回收机制

1 标记清除（mark and sweep）

2 引用计数（reference counting）

如何获得一个DOM元素的绝对位置?

- elem.offsetLeft：返回元素相对于其定位父级左侧的距离
- elem.offsetTop：返回元素相对于其定位父级顶部的距离
- elem.getBoundingClientRect()：返回一个DOMRect对象，包含一组描述边框的只读属性，单位像素

分析 ['1', '2', '3'].map(parseInt) 答案是多少?

- 答案:[1, NaN, NaN]

- parseInt(string, radix) 第2个参数 radix 表示进制。省略 radix 或 radix = 0，则数字将以十进制解析
- map 每次为 parseInt 传3个参数(elem, index, array)，其中 index 为数组索引
- 因此，map 遍历 ["1", "2", "3"]，相应 parseInt 接收参数如下

parseInt('1', 0); // 1
parseInt('2', 1); // NaN
parseInt('3', 2); // NaN

- 所以，parseInt 参数 radix 不合法，导致返回值为 NaN

var a=[1,2,3] a.psuh(4).reverse()===[4,3,2,1]错误的表达式，因为push()无返回值，不能链式调用
不改变原来数组的方法:
concat() slice() toString() join() join() 方法用于把数组中的所有元素放入一个字符串。
如何判断一个对象是否属于某个类?

```
if(a instanceof Person){
    alert('yes');
  }
```

讲一下事件循环机制
执行上下文(Execution context)
函数调用栈(call stack)
队列数据结构(queue)
Promise
JavaScript事件循环机制相关问题

- 事件循环机制的概念

关键字：单线程非阻塞、执行栈、事件队列、宏任务(setTimeout()、setInterval())、微任务(new Promise())
1行代码输出中文星期几

```
new Date().getDay();
```

不用循环 填充伪数组

```
var arrayLike={
    '0':'zhaishuang1',
    '1':'zhaishuang2',
    '2':'zhaishuang3',
    length:3
}
let tobeArray=Array.from(arrayLike)
```

数据类型

var foo = 10 + '20'; //foo=1020 number+string
(window.foo || (window.foo = 'bar')); //window.foo=bar

var foo = { n: 1 };
var bar = foo;
foo.x = foo = { n: 2 };
//foo.x=???
//bar.x=???

```
var foo={n:1};
var bar=foo;
foo.x=foo={n:2}
console.log(foo)//{n:2}
console.log(bar)//{n:1,x:{n:2}
console.log(foo.x)//undefined
console.log(bar.x)//{n:2}
```

var foo = "hello";
(function() {
  var bar = " world";
  console.log(foo + bar);
})();
console.log(foo + bar);

```
var foo="hello";
(function(){
    var bar="zhaishuang"
    console.log(foo+bar)//hello zhaishuang
})();
console.log(foo+bar)//hello[object Object]
```

for (let i = 0; i < 5; i++) {
  setTimeout(()=>{console.log(i)},3000)
}

使用promise实现异步

```
for(let i=0;i<5;i++){
        const p2=a=> new Promise(resolve=>setTimeout(()=>{
            console.log(a)
        },3000))
        p2(i)
    }
```

```
▶ Promise {<pending>}
0
1
2
3
4
```

```
function timeout(ms) {
    return new Promise((resolve, reject) => {
        setTimeout(resolve, ms, 'done');
    });
}

timeout(100).then((value) => {
    console.log(value);
});
```

实现add(2, 5); //7 add(2)(5); //7

原理是：链式调用

```
function addNumber(x){
    var sum=x;
    var tmp=(y)=>{
        sum=sum+y;
        return tmp;
    }
    tmp.toString=()=>{
        return sum;
    }
    return tmp;
}
```

下面UI，如何点击每一列的时候alert其index？（闭包）

```
<ul id="test">
    <li>一</li>
    <li>一</li>
    <li>一</li>
    <li>一</li>
</ul>
<script>
    var lis=document.getElementsByTagName('li')
    for(var i=0;i<3;i++){
        lis[i].index=i;
        lis[i].onclick=function(){
            alert(this.index)
        }
    }
</script>
```

变量提升

1、函数及变量的声明都将被提升到函数的最顶部。变量可以在使用后声明，也就是变量可以先使用再声明(函数声明和变量声明总是会被解释器悄悄地被"提升"到方法体的最顶部)

2、JavaScript 只有声明的变量会提升，初始化的不会。

```
var x='zhaishuang'
var getValue=function(){
    alert(x)
    var x='chenshuang'
    alert(x)
}
getValue();
//undefined
//chenshuang变量提升，内部作用域被浏览器解析成这样
// var x;alert(x);x='chenshuangshuang'
```

快速排序

```
//快速排序
function quickSort(arr){
    if(arr.length≤1){
        return arr;
    }
    var num=Math.floor(arr.length/2);//找到中间数的索引值，如果是浮点数，则向下取整
    var numValue=arr.splice(num,1);//找到中间数的值
    var left=[];
    var right=[];
    for(var i=0;i<arr.length;i++){
        if(arr[i]<numValue){
            left.push(arr[i]);//基准点的左边数传到左边数组
        }else{
            right.push(arr[i])//基准点的右边的数传到右边数组
        }
    }
    return quickSort(left).concat([numValue],quickSort(right));//递归不断重复比较
}
alert(quickSort([32,45,66,67,89]))
```

请简述JS中如何合并2个对象；

1、object.assign(obj1,obj2)

2、遍历赋值法： 循环对象n中的每一个对应属性。确认对象n中存在该属性确认对象o中不存在该属性

```
var extend=function(o,n){
    for(var p in n){
        if(n.hasOwnProperty(p)&&(!o.hasOwnProperty(p))){
            o[p]=n[p]
        }
    }
}
```

3、不同类型合成一个对象:forEach映射

```
const obj1 = {"name":"","age":"","salary":""};
const obj2 = [
    {"desc":"年龄",content:"25"},
    {"desc":"薪资",content:"3000"},
    {"desc":"姓名",content:"李杰"}
];
//汉英字典映射
const dict = {
    "年龄":"age",
    "薪资":"salary",
    "姓名":"name"
}
const obj3 = {...obj1};
obj2.forEach(item=>{
    if(item.desc in dict){
        const key = dict[item.desc];
        const value = item.content;
        obj3[key] = value;
    }
});
console.log(obj3);
```

```
▼ {name: "李杰", age: "25", salary: "3000"}
    age: "25"
    name: "李杰"
    salary: "3000"
  ▶ __proto__: Object
```

已知有一个长度未知的一维数值数组，请用最优雅的方式计算出其中前10项大于5的元素之和；

```
const testArray=[3,9,8,7667,84,91,6,11,21,31,41,51].slice(0,10)
const resultArray=testArray.filter((value) => { return value ≥ 5})
const sumArray=resultArray.reduce((a,b)=>{return a+b})
```

caller和callee的作用?

caller是返回一个对函数的引用，该函数调用了当前函数；
callee是返回正在被执行的function函数，也就是所指定的function对象的正文
如果一对兔子每月生一对兔子；一对新生兔，从第二个月起就开始生兔子；假定每对兔子都是一雌一雄，试问一对兔子，第n个月能繁殖成多少对兔子？（使用callee完成）

```
//典型的斐波那契数列
var result=[];
function fn(n){  //典型的斐波那契数列
    if(n===1||n===2){
        return 1
    }else{
        return result[n]?result[n]:arguments.callee(n-1)+arguments.callee(n-2);
    }
}
```

call和apply作用和区别。传null有没有影响,Function.call(null)，Function.apply(null)

apply和call都能继承另外一个对象的方法和属性；

Apply巧妙之处：可以将一个数组默认的转换为一个参数列表

[param1,param2,param3]（apply会将一个数组装换为一个 参数接一个参数的传递给方法）

Call巧妙之处:1、**直接用A对象方法来替换B对象 2、直接用B对象来执行A对象的方法 3、可以用 call 来实现继承**

使用apply的情况：在给对象参数的情况下,如果参数的形式是数组的时候,比如apply示例里面传递了参数arguments,这个参数是数组类型。

使用call的情况： 如果我的Person的参数列表是这样的(age,name),而Student的参数列表是(name,age,grade),这样就可以用call来实现了,也就是直接指定参数列表对应值的位置(Person.call(this,age,name,grade));

区别在于 call 的第二个参数可以是任意类型，而apply的第二个参数必须是数组，也可以是arguments。

传null/undefined 返回值为window

```
> function fn(){console.log(this)} fn.call(null)
  ▶ Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}
<· undefined
>
```

单线程原理

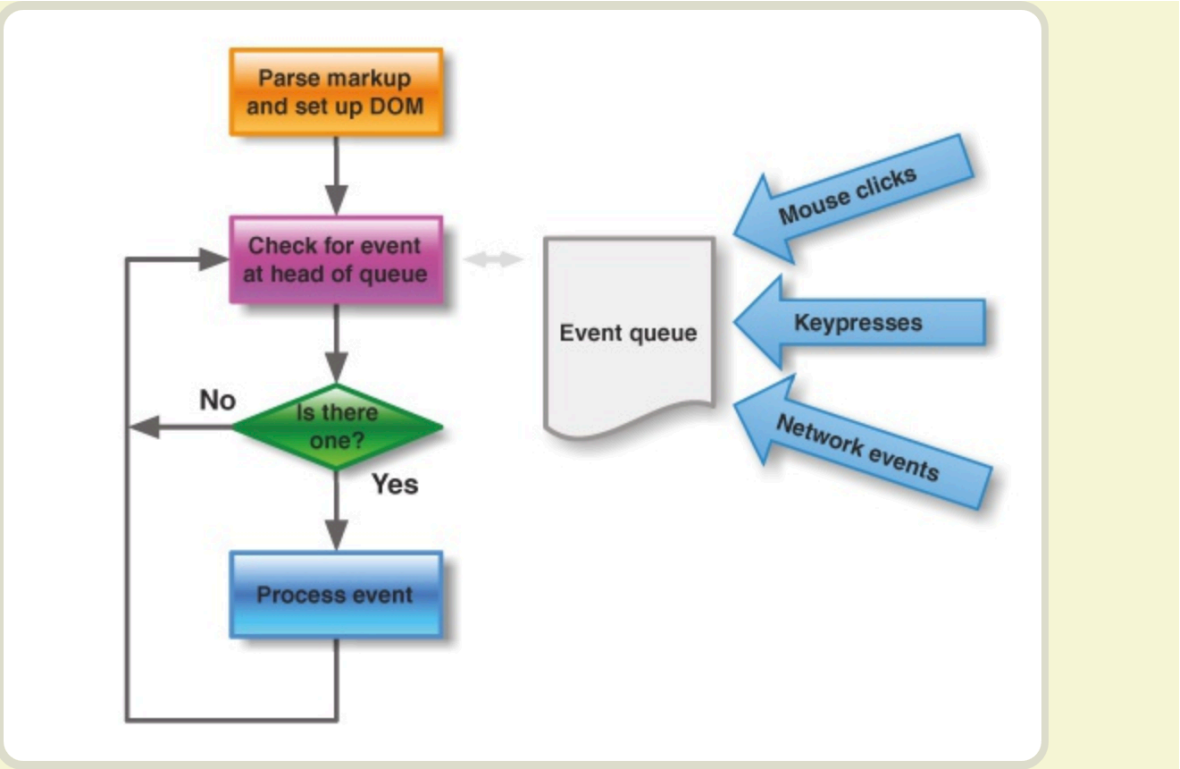
JS只有一个主线程，主线程执行完执行栈的任务后去检查异步的任务队列，如果异步事件触发，则将其加到主线程的执行栈

（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。



只要主线程空了，就会去读取"任务队列"，这就是JavaScript的运行机制。这个过程会不断重复。

关于Http 2.0？

HTTP/2引入了“服务端推（server push）”的概念，它允许服务端在客户端需要数据之前就主动地将数据发送到客户端缓存中，从而提高性能。
HTTP/2提供更多的加密支持
HTTP/2使用多路技术，允许多个消息在一个连接上同时交差。
它增加了头压缩（header compression），因此即使非常小的请求，其请求和响应的header都只会占用很小比例的带宽。

怎么解决跨域?

1、jsonp跨域、nginx反向代理、node.js中间件代理跨域、后端设置http header、后端在服务器上设置cors。
原理:动态插入script标签，通过script标签引入一个js文件，文件载入成功后悔执行URL参数中指定的函数，并且会把我们需要的json数据作为参数传入。
优点:兼容性好，简单易用，支持浏览器与服务器双向通信。
缺点：只支持GET请求
2、CORS是现代浏览器支持跨域资源请求的一种方式，全称是"跨域资源共享"（Cross-origin resource sharing），当使用XMLHttpRequest发送请求时，浏览器发现该请求不符合同源策略，会给该请求加一个请求头：Origin，后台进行一系列处理，如果确定接受请求则在返回结果中加入一个响应头：Access-Control-Allow-Origin;浏览器判断该相应头中是否包含Origin的值，如果有则浏览器会处理响应，我们就可以拿到响应数据，如果不包含浏览器直接驳回，这时我们无法拿到响应数据。
CORS与JSONP的使用目的相同，但是比JSONP更强大，CORS支持所有的浏览器请求类型，承载的请求数据量更大，开放更简洁，服务端只需要将处理后

的数据直接返回，不需要再特殊处理。

web storage两种方式(sessionStorage和localStorage)

sessionStorage用于本地存储的一个会话数据，这些数据只有在同一个会话中的页面才能访问并且会话结束后数据也随之销毁。因此seeionStorage不是一种持久化的储存，仅仅是会话级别的存储。

而localStorage用于持久化的本地存储，除非主动删除数据，否则数据是用于不会过期的

localstorage 与sessionStorage的区别

sessionStorage当前会话，当前访问的储存

localstorage服务端永久存在，除非手动清空

web storage 和cookie的区别

1、cookie的大小是受限的（不能超过4K，最多只有20条），并且每次你请求一个新的页面时，cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。Web Storage更大容量存储

2、web storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要前端开发者自己封装setCookie，getCookie。

3、cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

数组多类型去重（字符串、文字）

```
    /*去重*/
//     new Set(array)
   function delRep(array){
       var newArray=[]
       array.forEach(item=>{
           if(newArray.indexOf(item)==-1){
               newArray.push(item)
           }
       })
       return newArray
   }
   function delRep2(array){
       var newArray=[];
       newArray=array.filter((ele,i,arr)=>{
           return arr.indexOf(ele)===i
       })
       return newArray
   }
```

```
function deleteRepeat(array){
    var newArray=[];
    array.forEach((index)=>{
        if(newArray.indexOf(index)==-1){
            newArray.push(index);
        }
    });
    return newArray;
}
var repeatArray=['1','1','2','2']
deleteRepeat(repeatArray)
```

ES6 的const用es5实现

```
var CONST={};
Object.defineProperty(CONST,"a",{
    value:1,
    writable:false,//设置只读属性
    configurable:false,//设置属性不可配置
    enumerable:true
})
console.log(CONST.a)//1
CONST.a=2//错误，属性只读
```

那些操作会造成内存泄漏?

- 内存泄漏指任何对象在您不再拥有或需要它之后仍然存在
- 垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收
- setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏
- 闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

HTML中<meta>中必需元素是什么

<meta> 元素可提供有关页面的元信息（meta-information），比如针对搜索引擎和更新频度的描述和关键词。

<meta> 标签位于文档的头部，不包含任何内容。<meta> 标签的属性定义了与文档相关联的名称/值对。

必需元素：content　　some_text　　定义与 http-equiv 或 name 属性相关的元信息

必需的属性

属性	值	描述
content	some_text	定义与 http-equiv 或 name 属性相关的元信息

可选的属性

属性	值	描述
http-equiv	content-type expires refresh set-cookie	把 content 属性关联到 HTTP 头部。
name	author description keywords generator revised others	把 content 属性关联到一个名称。
scheme	some_text	定义用于翻译 content 属性值的格式。

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-scalable=no"/>
<meta http-equiv="X-UA-Compatible" content="ie=edge">
```

怎么判断设备类型，meta标签和js判断设备类型（navigator.userAgent）

<meta name="applicable-device" content="pc,mobile">

```
function IsPC() {
    var userAgentInfo = navigator.userAgent;
    var Agents = ["Android", "iPhone",
        "SymbianOS", "Windows Phone",
        "iPad", "iPod"];
    var flag = true;
    for (var v = 0; v < Agents.length; v++) {
        if (userAgentInfo.indexOf(Agents[v]) > 0) {
            flag = false;
            break;
        }
    }
    return flag;
}
var flag = IsPC(); //true为PC端, false为手机端
```

移动端怎么适应不同宽度页面

```javascript
<span style="font-size:12px;"><meta name="viewport" content="width=device-width,initial-scale=1.0,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no" />
// width     设置viewport宽度，为一个正整数，或字符串‘device-width’
// height    设置viewport高度，一般设置了宽度，会自动解析出高度，可以不用设置
// initial-scale     默认缩放比例，为一个数字，可以带小数
// minimum-scale     允许用户最小缩放比例，为一个数字，可以带小数
// maximum-scale     允许用户最大缩放比例，为一个数字，可以带小数
// user-scalable     是否允许手动缩放</span>
```

传输数据的get 和post有什么区别

GET和POST的区别

1、最直观的区别就是GET把参数包含在URL中，POST通过request body传递参数。

2、GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

3、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 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）。也就是说，GET只需要汽车跑一趟就把货送到了，而POST得跑两趟，第一趟，先去和服务器打个招呼“嗨，我等下要送一批货来，你们打开门迎接我”，然后再回头把货送过去。

http请求 有哪几种?

HTTP协议中共定义了八种方法或者叫“动作”来表明对Request-URI指定的资源的不同操作方式，具体介绍如下：

Options：返回服务器针对特定资源所支持的HTTP请求方法。也可以利用向Web服务器发送'*'的请求来测试服务器的功能性。

Head：向服务器索要与GET请求相一致的响应，只不过响应体将不会被返回。这一方法可以在不必传输整个响应内容的情况下，就可以获取包含在响应消息头中的元信息。

Get：向特定的资源发出请求。

Post：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的创建和/或已有资源的修改。

Put：向指定资源位置上传其最新内容。

Delete：请求服务器删除Request-URI所标识的资源。

Trace：回显服务器收到的请求，主要用于测试或诊断。

Connect：HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

虽然HTTP的请求方式有8种，但是我们在实际应用中常用的也就是get和post，其他请求方式也都可以通过这两种方式间接的来实现。

http请求头部信息

```
Headers  Preview  Response  Cookies  Timing

▾ General
  Request URL: http://localhost:8080/api/product/safeMap/getNavigationDistance
  Request Method: POST
  Status Code: ● 200 OK
  Remote Address: [::1]:8080
  Referrer Policy: no-referrer-when-downgrade

▾ Response Headers    view source
  Connection: keep-alive
  Content-Length: 57
  Content-Type: application/json; charset=utf-8
  Date: Mon, 14 May 2018 02:42:42 GMT
  Vary: Accept-Encoding

▾ Request Headers    view source
  Accept: application/json
  Accept-Encoding: gzip, deflate, br
  Accept-Language: zh-CN,zh;q=0.9
  Cache-Control: no-cache
  Connection: keep-alive
  Content-Length: 63
  Content-Type: application/json
  Cookie: io=MAjBiSO5DvkeazBzAAAB; Webstorm-e3e7eb91=b52b5b06-96a5-4e7a-8d43-5dde02b6aef1; __za__session__id__=15262657
  141187658902
  Host: localhost:8080
  Origin: http://localhost:8080
  path: /api/product/safeMap/getNavigationDistance
  Pragma: no-cache
  Referer: http://localhost:8080/product/safeMap/%E6%97%A5%E6%9C%AC
  User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11
  15A372 Safari/604.1

▾ Request Payload    view source
  ▾ {id: 8002, latitude: 35.7019167329, longitude: 139.7378540039}
      id: 8002
      latitude: 35.7019167329
      longitude: 139.7378540039
```

Accept：浏览器可接受的MIME类型。
Accept-Charset：浏览器可接受的字符集。
Accept-Encoding：浏览器能够进行解码的数据编码方式，比如gzip。Servlet能够向支持gzip的浏览器返回经gzip编码的HTML页面。许多情形下这可以减少5到10倍的下载时间。
Accept-Language：浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到。
Authorization：授权信息，通常出现在对服务器发送的WWW-Authenticate头的应答中。
Connection：表示是否需要持久连接。如果Servlet看到这里的值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。要实现这一点，Servlet需要在应答中发送一个Content-Length头，最简单的实现方法是：先把内容写入ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。
Content-Length：表示请求消息正文的长度。
Cookie：这是最重要的请求头信息之一
From：请求发送者的email地址，由一些特殊的Web客户程序使用，浏览器不会用到它。
Host：初始URL中的主机和端口。
If-Modified-Since：只有当所请求的内容在指定的日期之后又经过修改才返回它，否则返回304“Not Modified”应答。
Pragma：指定“no-cache”值表示服务器必须返回一个刷新后的文档，即使它

是代理服务器而且已经有了页面的本地拷贝。
Referer：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。**Referer** 表示这个请求是从哪个URL过来的，假如你通过google搜索出一个商家的广告页面，你对这个广告页面感兴趣，鼠标一点发送一个请求报文到商家的网站，这个请求报文的Referer报文头属性值就是http://www.google.com。
User-Agent：浏览器类型，如果Servlet返回的内容与浏览器类型有关则该值非常有用。
UA-Pixels，UA-Color，UA-OS，UA-CPU：由某些版本的IE浏览器所发送的非标准的请求头，表示屏幕大小、颜色深度、操作系统和CPU类型。

HTTP状态码

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息
200 OK 正常返回信息
201 Created 请求成功并且服务器创建了新的资源
202 Accepted 服务器已接受请求，但尚未处理
301 Moved Permanently 请求的网页已永久移动到新位置。
302 Found 临时性重定向。
303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。
304 Not Modified 自从上次请求后，请求的网页未修改过。
400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。
401 Unauthorized 请求未授权。
403 Forbidden 禁止访问。
404 Not Found 找不到如何与 URI 相匹配的资源。
500 Internal Server Error 最常见的服务器端错误。
502 HTTP 502 – 网关错误
503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

AJAX的工作原理：

1.创建ajax对象（XMLHttpRequest/ActiveXObject(Microsoft.XMLHttp)）
2.判断数据传输方式(GET/POST)
3.打开链接 open()
4.发送 send()
5.当ajax对象完成第四步（onreadystatechange）数据接收完成，判断http响应状态（status）200-300之间或者304（缓存）执行回调函数

http请求和响应的报文中都会包含些什么信息。

请求头	说明
Host	接受请求的服务器地址，可以是IP:端口号，也可以是域名
User-Agent	发送请求的应用程序名称
Connection	指定与连接相关的属性，如Connection:Keep-Alive
Accept-Charset	通知服务端可以发送的编码格式
Accept-Encoding	通知服务端可以发送的数据压缩格式
Accept-Language	通知服务端可以发送的语言

响应头	说明
Server	服务器应用程序软件的名称和版本
Content-Type	响应正文的类型（是图片还是二进制字符串）
Content-Length	响应正文长度
Content-Charset	响应正文使用的编码
Content-Encoding	响应正文使用的数据压缩格式
Content-Language	响应正文使用的语言

URL中含有中文字符编码和解码

encodeURI编码
decodeURi解码

比较typeof与instanceof?

对于 Array,Null 等特殊对象使用 typeof 一律返回 object，这正是 typeof 的局限性。
Instanceof定义和用法：instanceof 用于判断一个变量是否属于某个对象的实例。

点击穿透的问题原因是什么，怎么解决。

原因：
混用touch和click肯定会导致点透问题
点击穿透现象有3种：
1、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
2、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
3、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
详细解决方案：

1. 只用touch
   最简单的解决方案，完美解决点击穿透问题
   把页面内所有click全部换成touch事件（touchstart、’touchend’、’tap’），需要特别注意a标签，a标签的href也是click，需要去掉换成js控制的跳转，或者直接改成span + tap控制跳转。如果要求不高，不在乎滑走或者滑进来触发事件的话，span + touchend就可以了，毕竟tap需要引入第三方库
   不用a标签其实没什么，移动app开发不用考虑SEO，即便用了a标签，一般也会去掉所有默认样式，不如直接用span

局部滚动穿透的原因和解决方案

原因：底部的页面理论上是不可滚动的，但是当滑动弹窗时，底部页面会跟随滚动，这就是所谓的页面滚动穿透的问题
解决：
1、position：fixed；width：100%；配合Js监听scroll事件，需要滚动的时候就还原，所以如果需要保持滚动条的位置需要用 js 保存滚动条位置关闭的时候还原滚动位置
2、父容器overflow:hidden来禁用滚动，配合JS进行还原
3、在弹窗打开方法中,增加bodyfixed属性并记录滚动的位置

```
.stopScroll{position:fixed;width:100%}
this.setState({scrollLocation: document.documentElement.scrollTop})
document.body.classList.add(style.stopScroll);
```

关闭弹窗的方法中,移除属性，恢复滚动位置

```
document.body.classList.remove(style.stopScroll);
document.documentElement.scrollTop = this.state.scrollLocation;
```

h5调用手机摄像头的原理是HTML5 The Media Capture API提供了对摄像头的可编程访问，用户可以直接用getUserMedia获得摄像头提供的视频流

Event有哪些属性：

事件句柄 （Event Handlers）
onchange
onclick
onfocus
Onkeydown
onBlur
onkeyup
onMousemove
鼠标属性

button
clientX
clinetY
screenX
screenY
IE属性
cancelBubble
keyCode
offsetX
标准Event属性
Bubbles

简述一下src与href的区别

href 是指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，用于超链接。
src是指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求src资源时会将其指向的资源下载并应用到文档内，例如js脚本，img图片和frame等元素。

浏览器的缓存机制。

- 强缓存
  用户发送的请求，直接从客户端缓存中获取，不发送请求到服务器，不与服务器发生交互行为。
- 协商缓存
  用户发送的请求，发送到服务器后，由服务器判定是否从缓存中获取资源。
- 两者共同点：客户端获得的数据最后都是从客户端缓存中获得。
- 两者的区别：从名字就可以看出，强缓存不与服务器交互，而协商缓存则需要与服务器交互。

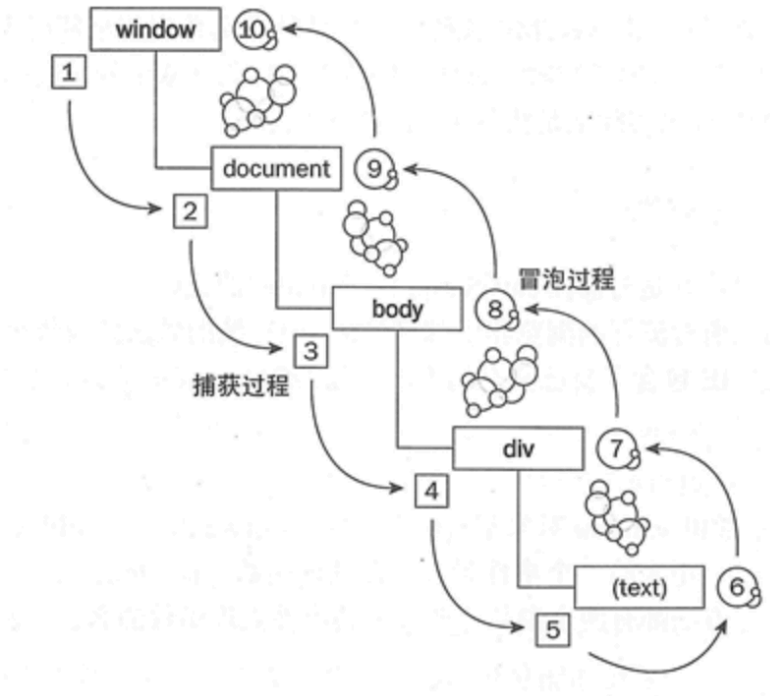
流程：**（a）浏览器判定是否有缓存=>（b）缓存是否过期=>（c）跟服务器协商是否使用缓存=>（d）协商缓存**

cache的作用：

1、减少延迟，让你的网站更快，提高用户体验。
2、避免网络拥塞，减少请求量，减少输出带宽。

js事件流，什么是事件委托

JS事件流原理图如下：



从图中我们可以知道

1、一个完整的JS事件流是从window开始，最后回到window的一个过程

2、事件流被分为三个阶段(1~5)捕获过程、(5~6)目标过程、(6~10)冒泡过程

3、在冒泡过程中6比7早触发，也就解释了上面那题，为什么btn1,会比content先触发

一个DOM元素绑定两个事件，一个冒泡，一个捕获，则事件会执行多少次，执行顺序如何？

执行次数：绑定了几个事件便执行几次。

所有事件的顺序是：其他元素捕获阶段事件 -> 本元素代码顺序事件 -> 其他元素冒泡阶段事件 。

事件委托就是利用事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件

优点

1.可以大量节省内存占用，减少事件注册。比如ul上代理所有li的click事件就很不错。

2.可以实现当新增子对象时，无需再对其进行事件绑定，对于动态内容部分尤为合适

缺点

事件代理的常用应用应该仅限于上述需求，如果把所有事件都用事件代理，可能会出现事件误判。即本不该被触发的事件被绑定上了事件。

事件是？IE与火狐的事件机制有什么区别？ 如何阻止冒泡？

- 我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击

一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为
- 事件处理机制：IE是事件冒泡、Firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件
- ev.stopPropagation();（旧ie的方法 ev.cancelBubble = true;）

使用addEventListener点击li弹出内容，并且动态添加li之后有效

```
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
</ul>
<script>
    var ulNode = document.getElementById("ul");
    ulNode.addEventListener('click', function (e) {
        if (e.target && e.target.nodeName.toUpperCase() == "LI") {
            alert(e.target.innerHTML);
        }
    }, false);
</script>
```

bind与on的区别就在于–事件冒泡:

on(events,[selector],[data],fn)

events:一个或多个用空格分隔的事件类型和可选的命名空间，如”click”或”keydown.myPlugin”。

selector:一个选择器字符串用于过滤器的触发事件的选择器元素的后代.

data:当一个事件被触发时要传递event.data给事件处理函数。

fn:该事件被触发时执行的函数。 false 值也可以做一个函数的简写，返回false。

bind(type,[data],fn)

为每个匹配元素的特定事件绑定事件处理函数。

jQuery 3.0中已弃用此方法，请用 on()代替。

参数类型跟前面那个on一样.

bind与on的区别就在于–事件冒泡:

```
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
</ul>
<script>
    (function () {
        $('ul').on('click','li',function(){
        alert($(this).text())
        });
        var ok = $('<li>4</li>');
        $('ul').last().append(ok);
})()

</script>
<!——事件委托的好处——>
<!——万一子元素非常多，给每个子元素都添加一个事件，会影响到性能；——>
<!——为动态添加的元素也能绑上指定事件；——>
```

setTimeout 和 setInterval 细谈

常问的点，前者是在一定时间过后将函数添加至执行队列，执行时间=延迟时间+之前函数代码执行时间+执行函数时间。
后者是不管前一次是否执行完毕，每隔一定时间重复执行，用于精准执行互相没有影响的重复操作。
如果需要控制前后执行顺序，最好使用setTimeout模拟setInterval

```
setTimeout(()=> {
    //do something
    setTimeout(arguments.callee, 3000);
}, 3000)
```

定时器的执行顺序或机制

js是单线程的，浏览器遇到setTimeout或者setInterval会先执行完当前的代码块，在此之前会把定时器推入浏览器的待执行事件队列里面，等到浏览器执行完当前代码之后会看一下事件队列里面有没有任务，有的话才执行定时器的代码。 所以即使把定时器的时间设置为0还是会先执行当前的一些代码

$.fn是什么意思?

$.fn是指jquery的命名空间，加上fn上的方法及属性，会对jquery实例每一个有效。

Jquery中$与$.fn的区别是什么?

$拓展的方法是静态方法，可以使用$直接调用，其拓展的方式有两种，一般使用$.extend({});;

而$.fn拓展的方法是实例方法，必须由“对象”$("")来调用，其拓展的方式同样有两种，一般使用$.fn.extend({ })。

script标签中的defer\anysc属性是做什么用的，有什么区别?

用处：向html页面中插入javascript代码

script标签存在两个属性，defer和async，因此script标签的使用分为三种情况：

1.<script src="example.js"></script>

没有defer或async属性，浏览器会立即加载并执行相应的脚本。也就是说在渲染script标签之后的文档之前，不等待后续加载的文档元素，读到就开始加载和执行，此举会阻塞后续文档的加载；

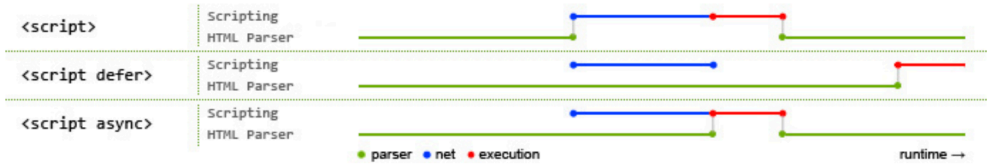
2.<script async src="example.js"></script>

有了async属性，表示后续文档的加载和渲染与js脚本的加载和执行是并行进行的，即异步执行；

3.<script defer src="example.js"></script>

有了defer属性，加载后续文档的过程和js脚本的加载(此时仅加载不执行)是并行进行的(异步)，js脚本的执行需要等到文档所有元素解析完成之后，DOMContentLoaded事件触发执行之前。

下图可以直观的看出三者之间的区别：



其中蓝色代表js脚本网络加载时间，红色代表js脚本执行时间，绿色代表html解析。

从图中我们可以明确一下几点：

1.defer和async在网络加载过程是一致的，都是异步执行的；

2.两者的区别在于脚本加载完成之后何时执行，可以看出defer更符合大多数场景对应用脚本加载和执行的要求；

3.如果存在多个有defer属性的脚本，那么它们是按照加载顺序执行脚本的；而对于async，它的加载和执行是紧紧挨着的，无论声明顺序如何，只要加载完成就立刻执行，它对于应用脚本用处不大，因为它完全不考虑依赖。

jquery和zepto的区别。

核心要点：

- Zepto更轻量级
- Zepto是jQuery的精简，针对移动端去除了大量jQuery的兼容代码
- 部分API的实现方式不同

详情

1. 针对移动端程序，Zepto有一些基本的触摸事件可以用来做触摸屏交互（tap事件、swipe事件），Zepto是不支持IE浏览器的。
2. DOM操作的区别：添加id时jQuery不会生效而Zepto会生效
3. 事件触发的区别：使用jquery时load事件的处理函数不会执行；使用zepto时load事件的处理函数会执行
4. 事件委托的区别：zepto中，选择器上所有的委托事件都依次放入到一个队列中，而在jquery中则委托成独立的多个事件
5. width() 与 height()的区别：zepto由盒模型（box-sizing）决定，用.width()返回赋值的width，用.css('width')返回border等的结果；jquery会忽略盒模型，始终返回内容区域的宽/高（不包含padding、border）.
6. offset()的区别：zepto返回{top,left,width,height}; jquery返回{width,height}。zepto无法获取隐藏元素宽高，jquery可以

7. zepto中没有为原型定义extend方法而jquery有
8. zepto的each方法只能遍历数组，不能遍历JSON对象。

设立"严格模式"的目的，主要有以下几个：

1、- 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;（函数重命名、变量保留字，this指向全局，**函数必须声明在顶层**等等）
   - 消除代码运行的一些不安全之处，保证代码运行的安全；
   - 提高编译器效率，增加运行速度；
   - 为未来新版本的Javascript做好铺垫。

2、规定只要函数参数使用了默认值、解构赋值、或者扩展运算符，那么函数内部就不能显式设定为严格模式，否则会报错。设定全局性的严格模式，这是合法的或者把函数包在一个无参数的立即执行函数里面。

webpack如何打包不同版本，如何做到热更新

如何打包不同版本：
通过 assets-webpack-plugin 生成一个记录了版本号的文件
在html中把入口文件引入即可，（html一般也引入入口文件和公共文件也就行了吧）剩下的模块加载时webpack的分内事了
如何做到热更新：
webpack-dev-server实现自动刷新
webpack-dev-server实现热更新(HMR)

```
plugins: [
    new HtmlWebpackPlugin({
        template: './index.html'
    }),
    new webpack.HotModuleReplacementPlugin()
]
```

git pull 会自动auto-merge，git fetch不会

正则表达式

表达式全集

字符	描述
\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个向后引用、或一个八进制转义符。例如，“n”匹配字符“n”。“\n”匹配一个换行符。串行“\\”匹配“\”而“\(”则匹配“(”。
^	匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。
$	匹配输入字符串的结束位置。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置。
*	匹配前面的子表达式零次或多次。例如，zo*能匹配“z”以及“zoo”。*等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“does”或“does”中的“do”。?等价于{0,1}。
{*n*}	*n*是一个非负整数。匹配确定的*n*次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。
{*n*,}	*n*是一个非负整数。至少匹配*n*次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o*”。
{*n*,*m*}	*m*和*n*均为非负整数，其中*n*<=*m*。最少匹配*n*次且最多匹配*m*次。例如，“o{1,3}”将匹配“fooooood”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。
?	当该字符紧跟在任何一个其他限制符（*,+,?，{*n*}，{*n*,}，{*n*,*m*}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o+?”将匹配单个“o”，而“o+”将匹配所有“o”。
.	匹配除“*n*”之外的任何单个字符。要匹配包括“*n*”在内的任何字符，请使用像“(.|\n)”的模式。
(pattern)	匹配pattern并获取这一匹配。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0…$9属性。要匹配圆括号字符，请使用“\(”或“\)”。
(?:pattern)	匹配pattern但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用或字符“(|)”来组合一个模式的各个部分是很有用。例如“industr(?:y|ies)”就是一个比“industry|industries”更简略的表达式。
(?=pattern)	正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”中的“Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
(?!pattern)	正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
(?<=pattern)	反向肯定预查，与正向肯定预查类拟，只是方向相反。例如，“(?<=95|98|NT|2000)Windows”能匹配“2000Windows”中的“Windows”，但不能匹配“3.1Windows”中的“Windows”。
(?<!pattern)	反向否定预查，与正向否定预查类拟，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows”。
x|y	匹配x或y。例如，“z|food”能匹配“z”或“food”。“(z|f)ood”则匹配“zood”或“food”。
[xyz]	字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。
[^xyz]	负值字符集合。匹配未包含的任意字符。例如，“[^abc]”可以匹配“plain”中的“p”。
[a-z]	字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。
[^a-z]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。

\b	匹配一个单词边界，也就是指单词和空格间的位置。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。
\B	匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。
\cx	匹配由x指明的控制字符。例如，\cM匹配一个Control-M或回车符。x的值必须为A-Z或a-z之一。否则，将c视为一个原义的“c”字符。
\d	匹配一个数字字符。等价于[0-9]。
\D	匹配一个非数字字符。等价于[^0-9]。
\f	匹配一个换页符。等价于\x0c和\cL。
\n	匹配一个换行符。等价于\x0a和\cJ。
\r	匹配一个回车符。等价于\x0d和\cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于[^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于\x09和\cI。
\v	匹配一个垂直制表符。等价于\x0b和\cK。
\w	匹配包括下划线的任何单词字符。等价于“[A-Za-z0-9_]”。
\W	匹配任何非单词字符。等价于“[^A-Za-z0-9_]”。
\x*n*	匹配*n*，其中*n*为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用ASCII编码。.
num	匹配*num*，其中*num*是一个正整数。对所获取的匹配的引用。例如，“(.)\1”匹配两个连续的相同字符。
n	标识一个八进制转义值或一个向后引用。如果*n*之前至少*n*个获取的子表达式，则*n*为向后引用。否则，如果*n*为八进制数字（0-7），则*n*为一个八进制转义值。
nm	标识一个八进制转义值或一个向后引用。如果*nm*之前至少有*nm*个获得子表达式，则*nm*为向后引用。如果*nm*之前至少有*n*个获取，则*n*为一个后跟文字*m*的向后引用。如果前面的条件都不满足，若*n*和*m*均为八进制数字（0-7），则*nm*将匹配八进制转义值*nm*。
nml	如果*n*为八进制数字（0-3），且*m和*均为八进制数字（0-7），则匹配八进制转义值*nm*l。
\u*n*	匹配*n*，其中*n*是一个用四个十六进制数字表示的Unicode字符。例如，\u00A9匹配版权符号（©）。

常用正则表达式

用户名	/^[a-z0-9_-]{3,16}$/
密码	/^[a-z0-9_-]{6,18}$/
十六进制值	/^#?([a-f0-9]{6}|[a-f0-9]{3})$/
电子邮箱	/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/ /^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/
URL	/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
IP 地址	/((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)/ /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
HTML 标签	/^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/
删除代码\\注释	(?<!http:|\S)//.*$
Unicode编码中的汉字范围	/^[\u2E80-\u9FFF]+$/

1、验证手机号

var myreg = /^[1][3,4,5,7,8][0-9]{9}$/;

myreg.test(15000796109)

2、去空格

去除所有空格：str =str.replace(\/s*/g,"")
去除两头空格：str=str.repalce(/^\s*/*$/g,"")
去除左空格：str = str.replace(/^\s*/, “”);
去除右空格：str = str.replace(/(\s*$)/g, "");