# Web前端面试技巧

面试一般分为技术面和hr面，形式的话很少有群面，少部分企业可能会有一个交叉面，不过总的来说，技术面基本就是考察你的专业技术水平的，hr面的话主要是看这个人的综合素质以及家庭情况符不符合公司要求，一般来讲，技术的话只要通过了技术面hr面基本上是没有问题（也有少数企业hr面会刷很多人）

那我们主要来说技术面，技术面的话主要是考察专业技术知识和水平，我们是可以有一定的技巧的，但是一定是基于有一定的能力水平的。

所以也慎重的告诉大家，技巧不是投机取巧，是起到辅助效果的，技术面最主要的还是要有实力，这里是基于实力水平之上的技巧。

## 面试中的技巧

1、**简历上做一个引导**：在词汇上做好区分，比如熟悉Java，了解python，精通c语言

这样的话对自己的掌握程度有个区分，也好让面试官有个着重去问，python本来写的也只是了解，自然就不会多问你深入的一些东西了。

2、**在面试过程中做一个引导**：

面试过程中尽量引导到自己熟知的一个领域，比如问到你说一下DNS寻址，然后你简单回答（甚至这步也可以省略）之后，可以说一句，自己对这块可能不是特别熟悉，对计算机网络中的运输层比较熟悉，如果有具体的，甚至可以再加一句，比如TCP和UDP

这样的话你可以把整个面试过程往你熟知的地方引导，也能更倾向于体现出你的优势而不是劣势，但是此方法仅限于掌握合适的度，比如有的知识点是必会的而你想往别处引就有点说不过去了，比如让你说几个html5的新特性，你一个也说不上来，那可能就真的没辙了。

3、**在自我介绍中做一个引导**：

一般面试的开头都会有一个自我介绍，在这个位置你也可以尽情的为自己的优势方面去引导。

4、**面试过程中展示出自信**：

面试过程中的态度也要掌握好，不要自卑，也不要傲娇，自信的回答出每个问题，尤其遇到不会的问题，要么做一些引导，实在不能引导也可以先打打擦边球，和面试官交流一下问题，看起来像是没听懂题意，这个过程也可以再自己思考一下，如果觉得这个过程可以免了的话也直接表明一下这个地方不太熟悉或者还没有掌握好，千万不要强行回答。

## 面试前的准备

最重要的肯定是系统的学习了，有一个知识的框架，基础知识的牢靠程度等。

其中算法尤其重要，越来越多公司还会让你现场或者视频面试中手写代码；

另一大重要的和加分项就是项目，在面试前，一定要练习回答自己项目的三个问题：

这是一个怎样的项目

用到了什么技术，为什么用这项技术（以及每项技术很细的点以及扩展）

过程中遇到了什么问题，怎么解决的。

### 富文本编辑器保留原样式

·在项目中富文本编辑器粘贴后需要保留(word、html)原有的样式，也就是富文本粘贴功能。但是我们使用quill富文本编辑器并没有这个功能，就重写了粘贴响应事件。

需要先获取到剪切板内容，复制过来的数据有些情况会被转义，需要再次转义回来，单双引号全部转为单引号比较可靠

保留word内容原样式要分析出word到剪切版特有的内容，然后用正则表达式判断要粘贴的内容是否来自word，如果是word的内容，就将cm/pt单位转换为px，并将html,body等无用标签替换为空，还有处理其他样式问题即可保留word原样式。

保留其他网页原样式就先过滤掉无用的html标签，还要处理stlye属性，才能还原网页原样式。难点在于分析剪切板得到的内容，只要分析好之后，用正则表达式和replace基本就能实现该功能。还有比较费劲的是要获取到光标的位置，并将处理好的文本插入到其中，也废了很多时间。

# 前端基础

## HTTP/HTML/浏览器

### 说一下http和https

https的SSL加密是在传输层实现的。

#### http和https的基本概念

http:

- **超文本传输协议**

**-** 应用最为广泛的一种网络协议

- 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

- 可以使浏览器更加高效，使网络传输减少。

https:

- 以安全为目标的HTTP通道

- 是**HTTP的安全版**

**- HTTPS的安全基础是SSL**，即HTTP下加入SSL层，因此加密的详细内容就需要SSL。

- 用于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。

#### http和https的区别

·http传输的数据都是明文的未加密的

·SSL协议可对http协议传输的数据进行加密处理

·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：

费用、传输协议、端口、连接方式、安全性

|  |  |
| --- | --- |
| Https协议 | http协议 |
| 需要ca证书，费用高 | 不需要ca证书 |
| 具有安全性的ssl加密传输协议 | 超文本传输协议，信息明文传输 |
| 端口为443 | 端口为80 |
| SSL+HTTP，可进行加密传输、身份认证 | 无状态的连接 |
| 安全 | 不安全 |

#### 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。

1.客户使用https url访问服务器，要求web 服务器建立ssl链接。

2.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）传输给客户端。

1. 客户端和web服务器端开始协商SSL链接的安全（加密）等级。

4.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。

5.web服务器通过自己的私钥解密出会话密钥。

6.web服务器通过会话密钥加密与客户端之间的通信。

#### https协议的优点

·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；

·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，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。

·现行架构下最安全的解决方案，大幅增加了攻击的成本。

·采用HTTPS加密的网站在谷歌搜索结果中的排名将会更高。

#### https协议的缺点

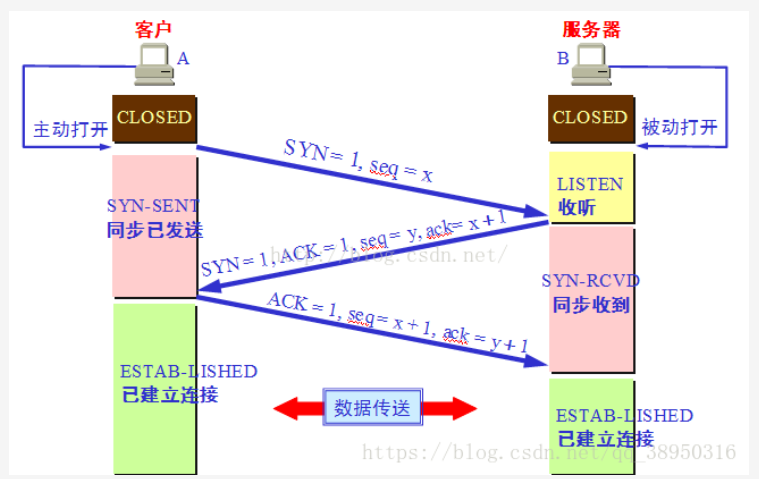
·握手阶段比较费时，页面加载时间延长50%，增加10%~20%的耗电。

·缓存不如http高效，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，不能在同一个ip上绑定多个域名，ipv4资源支持不了这种消耗。

### tcp三次握手



·第一次握手

- 主机向服务器发送SYN建立同步

·第二次握手

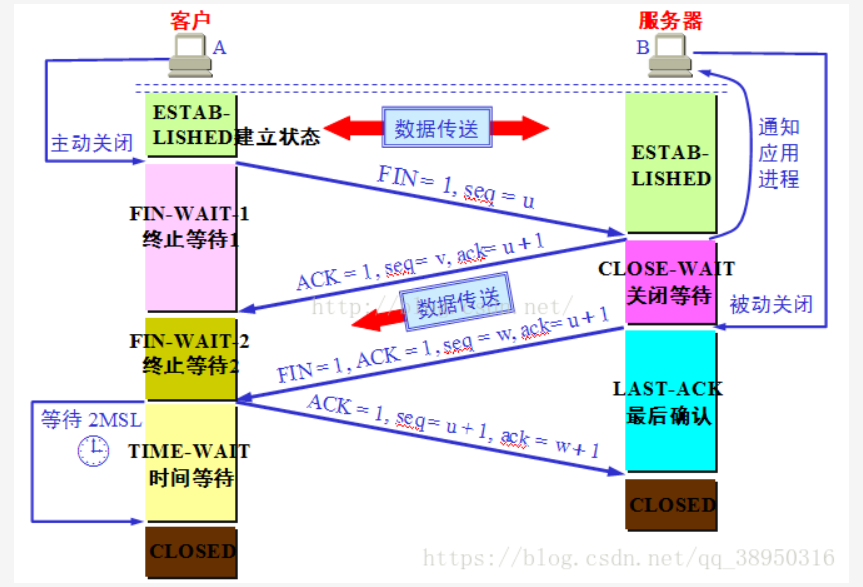
- 服务器向主机发送SYN和ACK建立同步并确认连接

·第三次握手

- 主机向服务器发送ACK确认连接

### tcp四次挥手

释放连接时



·第一次挥手

- 主机向服务器发送FIN断开连接

·第二次挥手

- 服务器向主机发送ACK等待数据包发完

·第三次挥手

- 服务器向主机发送FIN断开连接，并发送ACK确定断开

·第四次挥手

- 主机向服务器发送ACK确定断开

### TCP和UDP的区别

可靠、连接、数据、保障、发送速率、使用场景、首部、握手、面向、对

|  |  |
| --- | --- |
| **TCP** | **UDP** |
| 可靠的 | 不可靠的 |
| 有连接 | 无连接 |
| 数据无差错、不丢失、不重复、按序到达 | 尽最大努力交付 |
| 有可靠性保障:超时重发、流量控制 | 无可靠性保障 |
| 网络拥塞会使发送速率降低 | 网络拥塞不会使发送速率降低 |
| 适用于高安全性环境 | 适用于IP电话和视频会议等 |
| 首部为20字节 | 首部只有8字节 |
| 三次握手 | 无三次握手 |
| 面向字节流 | 面向报文 |
| 1对1 | 1对1,1对多 |

### WebSocket的实现和应用

#### 什么是WebSocket?

·HTML5中的协议，支持持久连续连接

·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

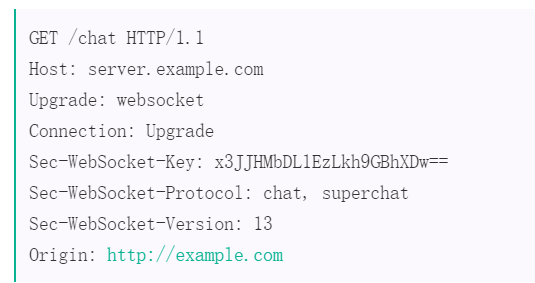
#### WebSocket的实现？

·WebSocket基于Http协议来完成一部分握手

·在握手阶段与Http是相同的

·看一个websocket握手协议的实现，基本是2个属性，upgrade，connection。

基本请求如下：



多了下面2个属性：



告诉服务器发送的是websocket



### 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

1. x-oss-storage-class:Standard

### 说一下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网站。

·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

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

### 几个很实用的BOM属性对象方法?

·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?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http:

- location.assign

-- 设置当前文档的URL

-- 可实现网页跳转

- 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

·history对象

- history.go()

-- 前进或后退指定的页面数

-- history.go(2);

- history.back()

-- 后退一页

- history.forward()

-- 前进一页

·Navigator对象

- navigator.userAgent

-- 返回用户代理头的字符串表示(包含浏览器版本信息)

- navigator.cookieEnabled

-- 返回浏览器是否支持(启用)cookie

### 说一下HTML5 drag api

·dragstart

- 在开始拖放被拖放元素时触发

·darg

- 在正在拖放被拖放元素时触发

·dragenter

- 在被拖放元素进入某元素时触发

·dragover

- 在被拖放在某元素内移动时触发

·dragleave

- 在被拖放元素移出目标元素时触发

·drop

- 在目标元素完全接受被拖放元素时触发

·dragend

- 在整个拖放操作结束时触发

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>测试</title>

</head>

<style>

html, body {

margin: 0;

padding: 0;

width: 100%;

height: 100%;

}

.tuo {

width: 100px;

height: 100px;

display: inline-block;

}

#one{

background-color: aqua;

}

#two{

background-color: blueviolet;

}

.drop {

width: 500px;

height: 500px;

border: 1px solid #000;

}

</style>

<body>

<div class="box">

<div id="one" draggable="true" class="tuo"></div>

<div id="two" draggable="true" class="tuo"></div>

<div class="drop"></div>

</div>

<script>

// 获取父元素

var ***box*** = ***document***.getElementsByClassName('box')[0];

// 获取可拖入元素

var ***drop*** = ***document***.getElementsByClassName('drop')[0];

// 当子元素的可拖动元素开始拖动时触发

***box***.ondragstart = function (e) {

***console***.log('开始拖动');

// 设置被拖动数据的数据类型和值

e.dataTransfer.setData('id',e.target.id);

}

// 当子元素的可拖动元素正在拖动时触发

***box***.ondrag = function (e) {

***console***.log('正在拖动',e.target.id);

}

// 当子元素的可拖动元素拖动结束时触发

***box***.ondragend = function (e) {

***console***.log('拖动结束');

}

// 可拖动元素进入可拖入元素时触发

***drop***.ondragenter = function (e) {

***console***.log('进入可拖入元素');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

}

// 可拖动元素在可拖入元素拖动时触发

***drop***.ondragover = function (e) {

***console***.log('在可拖入元素拖动');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

}

// 可拖动元素离开可拖入元素时触发

***drop***.ondragleave = function (e) {

***console***.log('离开可拖入元素');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

}

// 可拖动元素放入可拖入元素时触发

***drop***.ondrop = function (e) {

***console***.log('放入可拖入元素');

// 阻止默认事件，使元素可拖入其他元素

e.preventDefault();

// 获得拖动元素的数据

let data = e.dataTransfer.getData("id");

// 将可拖动元素变为可拖入元素的子元素

e.target.appendChild(***document***.getElementById(data));

}

</script>

</body>

</html>

### HTTP1.0和HTTP1.1的区别

**·长连接**

- 在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，在HTTP1.1中默认开启长连接keep-alive，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点

- HTTP1.0需要使用keep-alive参数来告知服务器端要建立一个长连接。

**·节约带宽**

- HTTP1.1支持只发送header信息（不带任何body信息）

- 如果服务器认为客户端有权限请求服务器，则返回100

- 客户端接收到100才开始把请求body发送到服务器

- 如果返回401，客户端就可以不用发送请求body了节约了带宽

**·HOST域**

- 在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址

- 请求消息中的URL并没有传递主机名（hostname）

- HTTP1.1的请求消息和响应消息都支持host域，且请求消息中如果没有host域会报告一个错误（400 Bad Request）。

**·缓存处理**

- 在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准

- HTTP1.1则引入了更多的缓存控制策略例如Entity tag，If-Unmodified-Since, If-Match, If-None-Match等更多可供选择的缓存头来控制缓存策略。

**·错误通知的管理**

- 在HTTP1.1中新增了24个错误状态响应码

### 说一下http2.0

·内容安全

·可以避免单纯使用https的性能下降

·请求资源所需时间更少，访问速度更快

**·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

- 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**

- 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**

### 补充400和401、403状态码

#### 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

#### 401状态码

·当前请求需要用户验证

#### 403状态码

服务器已经得到请求，但是拒绝执行

### fetch发送2次请求的原因

·fetch 第一次发送了一个Options请求，询问服务器是否支持修改请求头

·如果服务器支持，则在第二次中发送真正的请求

### Cookie、sessionStorage 、localStorage区别

·cookie数据始终在同源的http请求中携带(即使不需要)，cookie保存过多数据会带来性能问题，应该尽可能的少用

·sessionStorage 、localStorage仅在客户端即浏览器中保存，不参与和服务器的通信

·cookie数据可限制路径（path）

·cookie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

·sessionStorage 、localStorage的数据可以达到5M或更大

·sessionStorage仅在当前的浏览器页面关闭前有效

·localStorage始终有效，除非手动清除数据

·cookie只在设置的cookie过期时间之前一直有效，即使窗口和浏览器关闭

·sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面

·localStorage和cookie在所有同源窗口都是共享

·Cookie需要程序员自己封装，原生的cookie接口不友好

·sessionStorage 、localStorage既可采用原生接口，亦可再次封装

·cookie需要指定作用域，不可以跨域调用，限制多

·识别用户登陆，cookie比storage好用

·其他情况下可以用storage

·localstorage可以用来在页面传递参数

·sessionstorage可以用来保存一些临时的数据，防止用户刷新页面后丢失了一些参数

### cookie session区别

·都可用来存储用户信息

·cookie数据存放在客户的浏览器上

·session数据放在服务器上。

·cookie不安全，可以分析存放在本地的cookie进行cookie欺骗

·考虑到安全应当使用session。

·cookie一般用来存放不敏感的信息，比如用户设置的网站主题

·session一般用来存放敏感的信息，比如用户的登陆信息

·session可以存放于文件，数据库，内存中都可以

·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。

·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

### 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

·设置用户是否某段时间内免登录

·跟踪用户行为

- 例如在商城搜索框记录用户历史搜索记录

- 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

### 说一下web worker

·H5规范提供了JS分线程的实现，取名为：Web Workers，我们可以将大计算量的代码交由web Worker运行而不冻结用户界面

·相关API

- Worker

-- 构造函数。加载分线程执行的JS文件

- Worker.prototype.onmessage

-- 用于接受另一个线程的回调函数

- Worker.prototype.postMessage

-- 向另一个线程发生信息

·缺点

- 慢

- 子线程完全受主线程控制

- Worker内代码不能操作DOM（更新UI）

- 不能跨域加载JS

- 并没有改变JS单线程的本质

- 不是每个浏览器都支持这个新特性

·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

**主线程的JS**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<body>

<input type="text" placeholder="数值" id="number">

<button id="btn">计算</button>

</body>

<script type="text/javascript">

**var** input = document.getElementById("number");

**var** btn = document.getElementById("btn");

btn.onclick = **function** () {

**var** number = input.value;

//创建一个worker对象，并将js文件传入

**var** worker = **new** Worker("worker.js");

//绑定接收消息的监听

worker.onmessage = **function** (ev) {

console.log("主线程接收分线程返回的数据" + ev.data);

alert(ev.data);

};

//向分线程发送消息

worker.postMessage(number);

console.log("主线程向分线程发生数据：" + number);

}

</script>

</html>

**分线程的JS**

**var** onmessage = **function** (ev) {

**var** number = ev.data;

console.log("分线程接收到主线程发送的数据：" + number);

//计算

**var** result = fibonacci(number);

postMessage(result);

console.log("分线程向主线程返回数据：" + result);

alert(result);//报错，alert是window的方法，在分线程不能调用

};

**function** fibonacci(n) {//递归调用消耗内存，效率低，计算慢，计算过程不能操作其他东西

**if** (n === 0) {

**return** 0;

}

**return** n<=2 ? 1 : fibonacci(n-1) + fibonacci(n-2);

}

console.log(**this**);//返回的对象是DedicatedWorkerGlobalScope

//分线程中的全局对象不再是window，所以在分线程中不可能更新界面

### 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？

·iframe元素会创建包含另一个文档的内联框架

·可以将提示文字放在<iframe></iframe>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

·Doctype声明于文档最前面，告诉浏览器以何种方式来渲染页面

·有两种模式，严格模式和混杂模式。

·严格模式

- 排版和JS 运作模式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。

·如果不写文档声明，则会导致有些浏览器进入怪异模式，浏览器解析页面无法正常显示

### 浏览器内核

·Trident

- IE

·Blink

- 新Chrome和新

·Gecko

- Firefox

·Webkit

- Safari和旧Chrome

·Presto

- 旧Opera，已淘汰

·Trident+Webkit

- 大多数国内浏览器

### 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：

- set-cookie=httponly

- 这个属性可以防止XSS,它会禁止javascript脚本来访问cookie

- set-cookie=secure

- 这个属性告诉浏览器仅在请求为https的时候发送cookie

### 一句话概括RESTFUL

·用URL定位资源，用HTTP描述操作

### click在ios上有300ms延迟，原因及如何解决？

·原因：ios是可以双击缩放的，300ms延迟可以判断用户是否进行双击操作。

·禁用缩放

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

*<!-- 禁止移动端缩放 -->*

    <meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0,user-scalable=0">

·利用FastClick

- 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

安装fastclick

npm install fastclick --save

在main.js里面添加

import  FastClick  from  'fastclick'

### addEventListener参数

·addEventListener(event, function, useCapture)

- 绑定响应事件函数的第二个方法

- 可以为该元素添加响应事件

- 参数：

-- 事件名，不要on，如：“click”

-- 回调函数，当事件触发时该函数会被调用

-- 是否在捕获阶段触发事件，需要一个布尔值，一般传false

- 可以同时为一个元素的相同事件同时绑定多个响应事件，当事件被触发时，响应函数将会按照函数的绑定顺序执行。

btn01.addEventListener("click",**function** () {

alert("1");

},**false**);

btn01.addEventListener("click",**function** () {

alert("2");

},**false**);

- 以上方法不支持ie8及以下浏览器

- 在ie8中可使用attachEcent()来绑定多个事件，不支持其他浏览器

### 介绍知道的http返回的状态码

·100 Continue 继续

- 客户端应继续其请求

·101 Switching Protocols 切换协议

- 服务器根据客户端的请求切换协议

- 只能切换到更高级的协议

- 例如，切换到HTTP的新版本协议

·200 OK 请求成功

- 一般用于GET与POST请求

- 请求所希望的响应头或数据体将随此响应返回

- 如果文件不通过GZIP压缩，文件是多大，则要有多大传输量

·201 Created 已创建

- 成功请求并创建了新的资源

·202 Accepted 已接受

- 已经接受请求，但未处理完成

·203 Non-Authoritative Information 非授权信息

- 请求成功

- 但返回的meta信息不在原始的服务器，而是一个副本

·204 No Content 无内容

- 服务器成功处理，但未返回内容

- 在未更新网页的情况下，可确保浏览器继续显示当前文档

·205 Reset Content 重置内容

- 服务器处理成功，用户终端（例如：浏览器）应重置文档视图

- 可通过此返回码清除浏览器的表单域

·206 Partial Content 部分内容

- 服务器成功处理了部分GET请求

·300 Multiple Choices 多种选择

- 请求的资源可包括多个位置

- 响应可返回一个资源特征与地址的列表用于用户终端选择

·**301 Moved Permanently 永久移动**

- 永久重定向

- 请求的资源已被永久的移动到新URI

- 返回信息会包括新的URI，浏览器会自动定向到新URI

- 今后任何新的请求都应使用新的URI代替

- 除非额外指定，否则这个响应是可缓存的

- 常用的场景是域名跳转

·**302 Found 临时移动**

- 临时重定向

- 与301类似，但资源只是临时被移动

- 客户端应继续使用原有URI

- 只有在Cache-Control或Expires中进行了指定的情况下，这个响应才是可缓存的

- 临时跳转，比如未登陆的用户访问用户中心重定向到登录页面

·303 See Other 查看其它地址

- 与301类似

- 使用GET和POST请求查看

·**304** **Not Modified 未修改**

- 所请求的资源未修改

- 如果客户端发送了一个带条件的GET 请求且该请求已被允许，而文档的内容并没有改变，则服务器应当返回这个304状态码

- 服务器返回此状态码时，不会返回任何资源。

- 客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源

·305 Use Proxy 使用代理

- 所请求的资源必须通过代理访问

·306 Unused 已经被废弃的HTTP状态码

·307 Temporary Redirect 临时重定向

- 与302类似，使用GET请求重定向

·400 Bad Request 请求无效

·401 Unauthorized 当前请求需要用户验证

·402 Payment Required 保留，将来使用

·403 Forbidden 服务器已经得到请求，但是拒绝执行

·404 Not Found 服务器无法根据客户端的请求找到资源（网页）。

- 通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面

·405 Method Not Allowed 客户端请求中的方法被禁止

·406 Not Acceptable 服务器无法根据客户端请求的内容特性完成请求

·407 Proxy Authentication Required 请求要求代理的身份认证

- 与401类似，但请求者应当使用代理进行授权

·408 Request Time-out 服务器等待客户端发送的请求超时

·409 Conflict 服务器处理请求时发生了冲突

- 服务器完成客户端的PUT请求时可能返回此代码

·410 Gone 客户端请求的资源已经不存在

- 410不同于404，如果资源以前有现在被永久删除了可使用410代码

- 网站设计人员可通过301代码指定资源的新位置

·411 Length Required 服务器无法处理客户端发送的不带Content-Length的请求信息

·412 Precondition Failed 客户端请求信息的先决条件错误

·413 Request Entity Too Large 由于请求的实体过大，服务器无法处理，因此拒绝请求。

- 为防止客户端的连续请求，服务器可能会关闭连接

- 如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息

·414 Request-URI Too Large 请求的URI过长（URI通常为网址），服务器无法处理

·415 Unsupported Media Type 服务器无法处理请求附带的媒体格式

·416 Requested range not satisfiable 客户端请求的范围无效

·417 Expectation Failed 服务器无法满足Expect的请求头信息

·500 Internal Server Error 服务器内部错误，无法完成请求

·501 Not Implemented 服务器不支持请求的功能，无法完成请求

·502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应

·503 Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求

- 延时的长度可包含在服务器的Retry-After头信息中

·504 Gateway Time-out 充当网关或代理的服务器，未及时从远端服务器获取请求

·505 HTTP Version not supported 服务器不支持请求的HTTP协议的版本，无法完成处理

### http常用请求头

|  |  |
| --- | --- |
| 协议头 | 说明 |
| Accept | 可接受的响应内容类型（Content-Types）。 |
| **Accept-Charset** | 可接受的字符集 |
| **Accept-Encoding** | 可接受的响应内容的编码方式。 |
| **Accept-Language** | 可接受的响应内容语言列表。 |
| Accept-Datetime | 可接受的按照时间来表示的响应内容版本 |
| Authorization | 用于表示HTTP协议中需要认证资源的认证信息 |
| **Cache-Control** | 用来指定当前的请求/回复中的，是否使用缓存机制。 |
| Connection | 客户端（浏览器）想要优先使用的连接类型 |
| Cookie | 由之前服务器通过Set-Cookie（见下文）设置的一个HTTP协议Cookie |
| Content-Length | 以8进制表示的请求体的长度 |
| Content-MD5 | 请求体的内容的二进制 MD5 散列值（数字签名），以 Base64 编码的结果 |
| Content-Type | 请求体的MIME类型 （用于POST和PUT请求中） |
| Date | 发送该消息的日期和时间（以[RFC 7231](https://m.nowcoder.com/tutorial/96/24304825a0c04ea9a53cdb09cb664834" \l "section-7.1.1.1)中定义的"HTTP日期"格式来发送） |
| Expect | 表示客户端要求服务器做出特定的行为 |
| From | 发起此请求的用户的邮件地址 |
| Host | 表示服务器的域名以及服务器所监听的端口号。如果所请求的端口是对应的服务的标准端口（80），则端口号可以省略。 |
| If-Match | 仅当客户端提供的实体与服务器上对应的实体相匹配时，才进行对应的操作。主要用于像 PUT 这样的方法中，仅当从用户上次更新某个资源后，该资源未被修改的情况下，才更新该资源。 |
| If-Modified-Since | 允许在对应的资源未被修改的情况下返回304未修改 |
| If-None-Match | 允许在对应的内容未被修改的情况下返回304未修改（ 304 Not Modified ），参考 超文本传输协议 的实体标记 |
| If-Range | 如果该实体未被修改过，则向返回所缺少的那一个或多个部分。否则，返回整个新的实体 |
| If-Unmodified-Since | 仅当该实体自某个特定时间以来未被修改的情况下，才发送回应。 |
| Max-Forwards | 限制该消息可被代理及网关转发的次数。 |
| **Origin** | 发起一个针对[跨域资源共享](http://itbilu.com/javascript/js/VkiXuUcC.html)的请求（该请求要求服务器在响应中加入一个Access-Control-Allow-Origin的消息头，表示访问控制所允许的来源）。 |
| Pragma | 与具体的实现相关，这些字段可能在请求/回应链中的任何时候产生。 |
| Proxy-Authorization | 用于向代理进行认证的认证信息。 |
| Range | 表示请求某个实体的一部分，字节偏移以0开始。 |
| Referer | 表示浏览器所访问的前一个页面，可以认为是之前访问页面的链接将浏览器带到了当前页面。Referer其实是Referrer这个单词，但RFC制作标准时给拼错了，后来也就将错就错使用Referer了。 |
| TE | 浏览器预期接受的传输时的编码方式：可使用回应协议头Transfer-Encoding中的值（还可以使用"trailers"表示数据传输时的分块方式）用来表示浏览器希望在最后一个大小为0的块之后还接收到一些额外的字段。 |
| **User-Agent** | 浏览器的身份标识字符串 |
| Upgrade | 要求服务器升级到一个高版本协议。 |
| Via | 告诉服务器，这个请求是由哪些代理发出的。 |
| Warning | 一个一般性的警告，表示在实体内容体中可能存在错误。 |

### 强缓存和协商缓存

·缓存分为两种：强缓存和协商缓存，根据响应的header内容来决定

|  |  |  |  |
| --- | --- | --- | --- |
|  | 获取资源形式 | 状态码 | 发送请求到服务器 |
| 强缓存 | 从缓存取 | 200（from cache） | 否，直接从缓存取 |
| 协商缓存 | 从缓存取 | 304（not modified） | 是，通过服务器来告知缓存是否可用 |

·强缓存相关字段有expires，cache-control

- 如果cache-control与expires同时存在的话，cache-control的优先级高于expires。

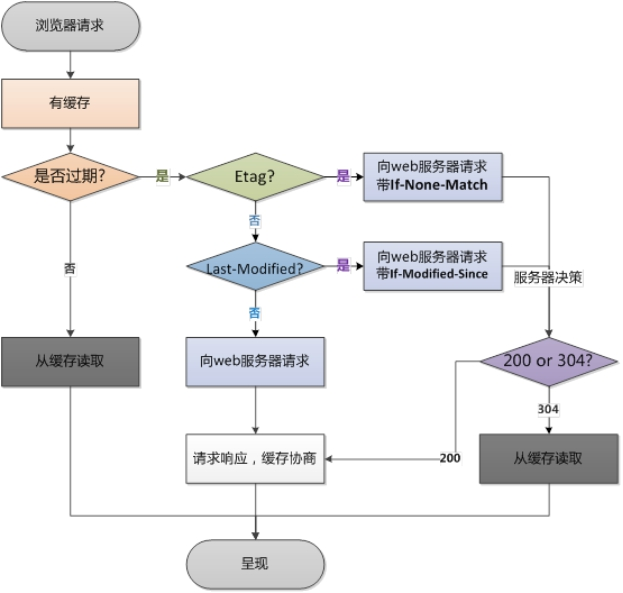
·协商缓存相关字段有Last-Modified/If-Modified-Since，Etag/If-None-Match

·因为服务器上的资源不是一直固定不变的，大多数情况下它会更新

·这个时候如果我们还访问本地缓存，那么对用户来说，那就相当于资源没有更新，用户看到的还是旧的资源

·所以我们希望服务器上的资源更新了浏览器就请求新的资源，没有更新就使用本地的缓存

·以最大程度的减少因网络请求而产生的资源浪费



web性能优化

·**降低请求量**

- 合并资源，减少HTTP 请求数，minify / gzip 压缩，webP，lazyLoad。

·**加快请求速度**

- 预解析DNS，减少域名数，并行加载，CDN 分发。

·**缓存**

- HTTP 协议缓存请求，离线缓存 manifest，离线数据缓存localStorage。

·**渲染**

- JS/CSS优化，加载顺序，服务端渲染，pipeline

### GET和POST的区别

|  |  |
| --- | --- |
| GET | POST |
| 参数通过url传递 | 参数在请求体中 |
| 参数有长度限制 | 没有长度限制 |
| 不安全，不能用来传递敏感信息 | 安全 |
| 只能进行url编码 | 支持多种编码方式 |
| 参数会被完整保留在浏览历史记录里 | 参数不会被保留 |
| 本质上是TCP连接 | |
| 产生一个TCP数据包 | 产生两个TCP数据包 |

### HTTP支持的方法

·get,post,head,options,put,delete,trace,connect

### 如何画一个三角形

·边框的均分原理

- 定义一个div

- 宽高设为0

- 三角形底边的边框为正常border+颜色

- 其余边框为border+transparent

div {

/\*绘制倒三角，transparent设置全透明色\*/

width:0px;

height:0px;

/\*设置底边\*/

border-top:100px solid red;

border-right:100px solid transparent;

border-bottom:100px solid transparent;

border-left:100px solid transparent;

}

·如果要画不等边三角形，将边框大小改成不同即可

### HTML5和CSS3的新特性

#### html5

·标签增删

- 8个语义元素

-- header、section、footer、aside、nav、main、article、figure

- 内容元素

-- mark高亮 progress进度

- 新的表单控件

-- calander、date、time、email、url、search、color、datetime、datetime-local

- 移除过时标签

-- big、font、frame、frameset

·canvas绘图

- 支持内联SVG。支持MathML

·拖放api

·多媒体

- audio、video、source、embed、track

·本地离线存储

- 把需要离线存储在本地的文件列在一个manifest配置文件

·web存储

- localStorage、SessionStorage

·多线程

- web worker、websocket

#### CSS3

CSS3边框

- border-radius，box-shadow等

·CSS3背景

- background-size，background-origin等

·CSS3 2D，3D转换

- transform

·CSS3动画

- animation

### 在地址栏里输入一个URL,到这个页面呈现出来，中间会发生什么

#### 完整回答

·**寻找url域名的服务器ip**,逐一查找浏览器缓存、系统缓存、路由器缓存、系统的hosts文件、DNS服务器，直到找到为止

·浏览器根据ip以及相应的端口号，**构造http请求**

·请求报文会包括这次请求的信息

·将http请求封装在一个tcp包中

·tcp包会依次经过**传输层，网络层，数据链路层，物理层**到达服务器

·服务器解析这个请求来作出响应，返回相应的html给浏览器

·浏览器根据html构建DOM树

·然后构建CSSOM树

·CSSOM树和DOM树合并为渲染树

·查看缓存里是否有所请求的资源

·渲染页面

#### 简单回答

·DNS解析

·TCP连接

·发送HTTP请求

·服务器处理请求并返回HTTP报文

·浏览器解析渲染页面

### 常见的HTTP的首部

·通用首部

- 表示一些通用信息，比如date表示报文创建时间

·请求首部

- 请求报文中独有的，如cookie，和缓存相关的如if-Modified-Since

·响应首部

- 响应报文中独有的，如set-cookie，和重定向相关的location

·实体首部

- 用来描述实体部分，如allow用来描述可执行的请求方法，content-type描述主题类型，content-Encoding描述主体的编码方式

### cache-control的值有哪些

·cache-control是一个通用消息头字段

·用于HTTP请求和响应中，通过指定指令来实现缓存机制

·这个缓存指令是单向的

·常见的取值有private、no-cache、max-age、must-revalidate等，默认为private

### 浏览器在生成页面的时候，会生成那两颗树？

·DOM树和CSSOM树

·当浏览器接收到服务器相应来的HTML文档后，会遍历文档节点，生成DOM树

·CSSOM规则树由浏览器解析CSS文件生成

### csrf和xss的网络攻击及防范

#### CSRF

·跨站请求伪造

·可以理解为攻击者盗用了用户的身份，以用户的名义发送了恶意请求

·在受害者访问一个网站时，其 Cookie 还没有过期的情况下，攻击者伪造一个链接地址发送受害者并欺骗让其点击，从而形成 CSRF 攻击

·防御方式

- 使用验证码、检查https头部的refer、token

#### XSS

·跨站脚本攻击

·当渲染DOM树的过程成发生了不在预期内执行的JS代码时，就发生了XSS攻击

·大多数XSS攻击的主要方式是嵌入一段远程或者第三方域上的JS代码

·实际上是在目标网站的作用域下执行了这段js代码

·比如获取cookie，或者其他用户身份信息，可以分为存储型和反射型

·防御方式

- 为cookie设置httpOnly属性，对用户的输入进行检查，进行特殊字符过滤

### 怎么看网站的性能如何

·被动检测

- 在被检测的页面置入脚本或探针，当用户访问网页时，探针自动采集数据并传回数据库进行分析

·主动监测

- 主动的搭建分布式受控环境，模拟用户发起页面访问请求，主动采集性能数据并分析

- 在检测的精准度上，专业的第三方工具效果更佳，比如说性能极客

### cookie有哪些字段可以设置

·name

- cookie的名称

·value

- cookie的值

·domain

- 可以访问此cookie的域名

- 非顶级域名，如二级域名或者三级域名

- 设置的cookie的domain只能为顶级域名或者二级域名或者三级域名本身

- 不能设置其他二级域名的cookie，否则cookie无法生成

- 顶级域名只能设置domain为顶级域名

- 不能设置为二级域名或者三级域名，否则cookie无法生成

- 二级域名能读取设置了domain为顶级域名或者自身的cookie

- 不能读取其他二级域名domain的cookie

- 要想cookie在多个二级域名中共享，需要设置domain为顶级域名

- 顶级域名只能获取到domain设置为顶级域名的cookie，其他domain设置为二级域名的无法获取。

·path

- 可以访问此cookie的页面路径

- 比如domain是abc.com,path是/test

- 那么只有/test路径下的页面可以读取此cookie。

·expires/Max-Age

- 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。

·Size

- cookie大小。

·http

- cookie的httponly属性

- 若此属性为true，则只有在http请求头中会带有此cookie的信息，而不能通过document.cookie来访问此cookie。

·secure

- 设置是否只能通过https来传递此条cookie

## CSS

### 说一下css盒模型

·用来装页面上的元素的矩形区域

·标准盒子模型

- 盒子宽度 = 左右border+左右padding+width

·IE盒子模型

- 盒子宽度 = width

·box-sizing属性

- content-box

-- 将元素设置为标准盒子模型

-- width = 内容区

- border-box

-- IE盒子模型

-- width = 边框+内边距+内容区

### 画一条0.5px的线

·采用meta viewport的方式

- 缩放到原来的0.5倍，如果是1px那么就会变成0.5px

- viewport只针对于移动端，只在移动端上才能看到效果

<meta name="viewport" content="width=device-width, initial-scale=0.5, minimum-scale=0.5, maximum-scale=0.5"/>

·采用transform: scale()的方式

- transform: scale(0.5,0.5);

### link标签和import标签的区别

·link属于html标签

·@import是css提供的

·页面被加载时，link会同时被加载

·@import引用的css会等到页面加载结束后加载

·link没有兼容性问题

·@import只有IE5以上才能识别

·link方式样式的权重高于@import

### 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

·transition需要触发一个事件才能改变属性

·animation不需要触发任何事件的情况下才会随时间改变属性值

·transition为2帧，从from .... to，

·animation可以一帧一帧的

### Flex布局

·Flex是Flexible Box的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性

·布局的传统解决方案，基于盒状模型，依赖display属性 + position属性 + float属性。它对于那些特殊布局非常不方便，比如，垂直居中就不容易实现。

·弹性容器

- 要使用弹性盒，必须先将一个元素设置为弹性容器

- 我们通过 display 来设置弹性容器

-- display:flex 设置为块级弹性容器

-- display:inline-flex 设置为行内的弹性容器

- 其子元素会自动变成弹性元素

·弹性元素

- 弹性容器的子元素是弹性元素（弹性项）

·弹性元素可以同时是弹性容器

·flex-direction

- 指定容器中弹性元素的排列方式

- row

-- 默认值，弹性元素在容器中水平排列（左向右）

-- 主轴 自左向右

- row-reverse

-- 弹性元素在容器中反向水平排列（右向左）

-- 主轴 自右向左

- column

-- 弹性元素纵向排列（自上向下）

- column-reverse

-- 弹性元素方向纵向排列（自下向上）

flex-direction: row;

·主轴

- 弹性元素的排列方向称为主轴

·侧轴

- 与主轴垂直方向的称为侧轴

·flex-grow

- 定义元素的放大比例

- 指定弹性元素的伸展的系数

- 当父元素有多余空间的时，子元素如何伸展

- 父元素的剩余空间，会按照比例进行分配

- 默认值为0，不伸展

·flex-shrink

- 定义了元素的缩小比例

- 当父元素中的空间不足以容纳所有的子元素时，对子元素进行收缩

- 默认值是1，等比例收缩

- 0是不收缩

- 值越大，收缩的越多

·flex-wrap

- 设置弹性元素是否在弹性容器中自动换行

- nowrap

-- 默认值，元素不会自动换行

- wrap

-- 元素沿着侧轴方向自动换行

- wrap-reverse

-- 元素沿着侧轴反方向换行

flex-wrap: wrap;

·flex-flow

- 可同时设置wrap 和 direction 的属性

flex-flow: row wrap;

·justify-content

- 如何分配主轴上的空白空间（主轴上的元素如何排列）

- flex-start

-- 元素沿着主轴起边排列

- flex-end

-- 元素沿着主轴终边排列

- center

-- 元素居中排列

- space-around

-- 空白分布到元素两侧

- space-between

-- 空白均匀分布到元素间

- space-evenly

-- 空白分布到元素的单侧

justify-content: center

·align-items

- 元素在辅轴上如何对齐

- 元素间的关系

- stretch

-- 默认值，将元素的长度设置为相同的值

- flex-start

-- 元素不会拉伸，沿着辅轴起边对齐

- flex-end

-- 沿着辅轴的终边对齐

- center

-- 居中对齐

- baseline

-- 基线对齐

align-items: center;

·垂直水平居中对齐

justify-content: center;

align-items: center;

·align-content

- 侧轴空白空间的分布

- flex-start

-- 元素沿着侧轴起边排列

- flex-end

-- 元素沿着侧轴终边排列

- center

-- 元素居中排列

- space-around

-- 空白分布到元素两侧

- space-between

-- 空白均匀分布到元素间

- space-evenly

-- 空白分布到元素的单侧

align-content: space-between;

·align-self

- 为某个子元素单独设置自身的align-items

align-self: stretch;

### BFC（块级格式化上下文）

·解决高度塌陷和外边距重叠问题

·高度塌陷

- 当为子元素设置浮动以后，子元素会完全脱离文档流

- 此时将会导致子元素无法撑起父元素的高度，从而导致父元素的高度塌陷

·外边距重叠

- 子元素和父元素相邻的垂直外边距会发生重叠，子元素的外边距会传递给父元素

·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

·哪些元素会生成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

### 水平垂直居中的方法

·margin:auto法

- 子绝父相

- 子元素外边距为auto

- 子元素偏移量全为0

- 脱离文档流

- 这种方式只适用于大小确定的元素

.d1{

width: 400px;

height: 400px;

position: relative;

border: 1px solid #465468;

}

.d2{

width: 200px;

height: 140px;

background-color: #0086b3;

position: absolute;

margin: auto;

top: 0;

left: 0;

right: 0;

bottom: 0;

}

·translate负50%法

- 子绝父相

- 子元素top和left为50%

- 子元素translate(-50%,-50%)

- 脱离文档流

.d1{

width: 500px;

height: 400px;

border: 2px solid #379;

position: relative;

}

.d2{

width: 200px;

height: 140px;

background-color: #746;

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%,-50%);

}

·table-cell

- 父元素display: table-cell;（设置成表格）

- 父元素vertical-align: middle;（单元格垂直居中）

- 父元素text-align: center;（字体水平居中）

- 如果子元素为div，需要margin: 0 auto;

- 未脱离文档流

.d1{

width: 300px;

height: 300px;

border: 3px solid #555;

display: table-cell;

vertical-align: middle;

text-align: center;

}

.d2{

width: 100px;

height: 80px;

background-color: #0086b3;

margin: 0 auto;

}

·利用flex

- 父元素display: flex;（设置为弹性盒）

- 父元素justify-content: center;（元素在主轴居中对齐）

- 父元素align-items: center;（元素在辅轴居中对齐）

.d1{

width: 300px;

height: 300px;

border: 3px solid #555;

display: flex;

justify-content: center;

align-items: center;

}

.d2{

width: 100px;

height: 80px;

background-color: #0086b3;

}

### 关于js动画和css3动画的差异性

·渲染线程分为main thread和compositor thread

·如果css动画只改变transform和opacity，这时整个CSS动画在compositor

trhead完成

·而js动画则会在main thread执行，然后触发compositor thread进行下一步操作

·特别注意的是如果改变transform和opacity是不会layout或者paint的。

·区别

- 功能涵盖面，js比css大

- 实现/重构难度不一，CSS3比js更加简单，性能调优方向固定

- 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

### 说一下块元素和行元素

·块元素

- 独占一行，并且会自动填满父元素

- 可以设置margin和pading以及高度和宽度

·行元素

- 不会独占一行，width和height会失效

并且在垂直方向的padding和margin会失效

·可通过display将元素变为块元素或行元素

### 多行元素的文本省略号

/\*将文本设置为不换行\*/

white-space: nowrap;

/\*隐藏溢出属性\*/

overflow: hidden;

/\*在溢出内容的后方显示省略号\*/

text-overflow: ellipsis;

### visibility=hidden、opacity=0、display:none的区别

·opacity=0

- 将元素隐藏起来，不会改变页面布局

- 如果该元素已绑定响应事件，会触发响应事件

·visibility=hidden

- 将元素隐藏起来了，不会改变页面布局

- 不会触发该元素已经绑定的响应事件

·display=none

- 将元素隐藏起来，会改变页面布局

- 可以理解成元素在页面中消失

- 元素的html结构还在

### 双边距重叠问题（外边距折叠）

·多个相邻（兄弟或者父子关系）普通流的块元素垂直方向marigin会重叠

·重叠的结果为：

- 两个相邻的外边距都是正数时，重叠结果是它们两者之间较大的值

- 两个相邻的外边距都是负数时，重叠结果是两者绝对值的较大值

- 两个外边距一正一负时，重叠结果是两者的相加的和

### position属性

·relative

- 相对定位是相对于元素在文档流中原来的位置进行定位

- 相对定位的元素不会脱离文档流

- 会使元素提升一个层级，可能覆盖原本层级的元素

- 通常偏移量只需要设置left和top

- 相对定位不会改变元素的性质，块还是块，内联还是内联

· absolute

- 开启绝对定位会使元素脱离文档流

- 相对于离他最近并开启了定位的祖先元素进行定位

- 如果所有的祖先元素都没有开启定位，则相对于浏览器窗口进行定位

- 一般开启了子元素的绝对定位，都会开启父元素的相对定位

- 会使元素提升一个层级，可能覆盖原本层级的元素

- 块元素的宽度和高度会被内容撑开，内联变为块元素

·fixed

- 固定定位也是一种绝对定位，它的大部分特点都和绝对定位一样

- 区别

- 永远都会相对于浏览器窗口进行定位

- 固定定位会使元素固定在浏览器窗口某个位置，不会随窗口的滚动条滚动

- IE6不支持固定定位

·sticky

- 兼容性并不是很好

- 粘滞定位和相对定位的特点基本一致

- 不同的是粘滞定位可以在元素到达某个位置时将其固定

·inherit

- 继承父元素

### 浮动清除

·使用带clear属性的空元素

- 在浮动元素后添加元素如<div class="clear"></div>

- 并在CSS中赋予.clear{clear:both;}属性

·使用CSS的overflow属性

- 给浮动元素的容器添加overflow:hidden或overflow:auto

- 浮动元素又会回到了容器层，把容器高度撑起

·给浮动的元素的容器添加浮动

- 会使其整体浮动，影响布局，不推荐使用。

·使用邻接元素处理

- 给浮动元素后面的元素添加clear属性。

·使用CSS的:after伪元素

- 给浮动元素的容器添加clearfix的class

- 给这个class添加一个:after伪元素，在末尾添加一个看不见的块元素

- 并赋予.clear{clear:both;}属性

- 可以完美兼容当前主流的各大浏览器

·IE6 中还需要 hasLayout ，例如为父元素设置 zoom:1

### Clearfix

·既可以解决高度塌陷，又可以确保父元素和子元素的垂直外边距不会重叠

.clearfix:before,

.clearfix:after{

content: "";

display: table;

clear: both;

}

.clearfix{

zoom: 1;

}

### CSS选择器有哪些，优先级呢

·id 选择器、class 选择器、标签选择器、属性选择器、伪元素选择器、伪类选择器等

·内联样式

- 1000

·id选择器

- 100

·类、伪类、伪元素和属性

- 10

·元素选择器

- 1

·通配\*

- 0

·继承的样式，没有优先级

·当选择器中包含多种选择器时，需要将多种选择器的优先级相加进行比较

·但是，选择器优先级计算不会超过他的最大数量级

·如果选择器的优先级一样，则选择靠后的样式

·并集选择器的优先级是单独计算的，不会相加

·可以在样式的最后，添加一个!important，则此时该样式将会获得最高的优先级，但是在开发中尽量避免使用

### css动画如何实现

·使用@keyframes设置一个关键帧，关键帧设置了动画执行每一个步骤

·通过animation属性为元素同时设置动画相关的所有属性

- 如关键帧名、时长、延时、动画时序函数等

### 隐藏页面中某个元素的方法

·display:none;

·visibility:hidden;

·opacity: 0;

·position将元素移到外部

·z-index涂层遮盖等

### 三栏布局的实现方式

·浮动布局： float + margin

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<style>

body,html{

height: 100%;

padding:0;

margin: 0;

}

.container{

width: 100%;

height: 100%;

}

/\*左边栏左浮动\*/

.left{

float:left;

height:100%;

width:100px;

background:#ff69b4;

}

/\*中间栏自适应\*/

.main{

height:100%;

margin:0 200px 0 100px;

background: #659;

}

/\*右边栏右浮动\*/

.right{

float:right;

height:100%;

width:200px;

background:#ff69b4;

}

</style>

</head>

<body>

<div class="container">

<div class="left">Left</div>

<div class="right">Right</div>

<div class="main">Main</div>

</div>

</body>

</html>

·绝对定位布局：position + margin

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<style>

body,html{

height: 100%;

padding: 0;

margin: 0;

overflow: hidden;

}

.container{

width: 100%;

height: 100%;

}

/\*左右进行绝对定位\*/

.left,.right{

position: absolute;

height:100%;

top: 0;

background: #ff69b4;

}

.left{

left: 0;

width: 100px;

}

.right{

right: 0;

width: 200px;

}

/\*中间用margin空出左右元素所占的空间\*/

.main{

height:100%;

margin: 0 200px 0 100px;

background: #659;

}

</style>

</head>

<body>

<div class="container">

<div class="left">Left</div>

<div class="right">Right</div>

<div class="main">Main</div>

</div>

</body>

</html>

·使用table实现

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<style>

.container{

display: table;

width:100%;

}

.container>div{

display: table-cell;

}

.left{

width: 100px;

background: red;

}

.main{

background: blue;

}

.right{

width: 200px;

background: red;

}

</style>

</head>

<body>

<div class="container">

<div class="left">left</div>

<div class="main">center</div>

<div class="right">right</div>

</div>

</body>

</html>

·flex实现

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<style>

.container{

display: flex;

}

.left{

width:200px;

background: red;

}

.main{

flex: 1;

background: blue;

}

.right{

width:200px;

background: red;

}

</style>

</head>

<body>

<div class="container">

<div class="left">left</div>

<div class="main">center</div>

<div class="right">right</div>

</div>

</body>

</html>

·grid实现

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<style>

.container{

display: grid;

width: 100%;

grid-template-rows: 100px; /\*设置行高\*/

grid-template-columns: 100px auto 200px; /\*设置列数属性\*/

}

.left{

background: red;

}

.main{

background: blue;

}

.right{

background:red;

}

</style>

</head>

<body>

<div class="container">

<div class="left">left</div>

<div class="main">center</div>

<div class="right">right</div>

</div>

</body>

</html>

·双飞翼布局

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<style>

body,html,.container{

height: 100%;

padding:0;

margin: 0;

}

.col{

float: left; /\* 把left和right定位到左右部分 \*/

}

.main{

width:100%;

height:100%;

background: #659;

}

.main\_inner{ /\* 处理中间栏的内容被遮盖问题 \*/

margin:0 200px 0 100px;

}

.left{

width: 100px;

height: 100%;

margin-left: -100%;

background: #ff69b4;

}

.right{

height:100%;

width:200px;

margin-left: -200px;

background: #ff69b4;

}

</style>

</head>

<body>

<div class="container">

<div class="main col ">

<div class="main\_inner">Main</div>

</div>

<div class="left col ">Left</div>

<div class="right col ">Right</div>

</div>

</body>

</html>

·圣杯布局

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<style>

/\* 两边定宽，中间自适用 \*/

body,html,.container{

height: 100%;

padding:0;

margin: 0;

}

.col{

float: left; /\* 三个col都设置float: left,为了把left和right定位到左右部分 \*/

position:relative;

}

/\*父元素空出左右栏位子: 因为上一步中，左右栏定位成功了，但是中间栏的内容会被遮盖住\*/

.container{

padding:0 200px 0 100px;

}

/\*左边栏\*/

.left{

left:-100px;

width: 100px;

height:100%;

margin-left: -100%;

background: #ff69b4;

}

/\*中间栏\*/

.main{

width:100%;

height: 100%;

background: #659;

}

/\*右边栏\*/

.right{

right:-200px;

width:200px;

height:100%;

margin-left: -200px;

background: #ff69b4;

}

</style>

</head>

<body>

<div class="container">

<div class="main col">Main</div>

<div class="left col">Left</div>

<div class="right col">Right</div>

</div>

</body>

</html>

### display：table和table标签有什么区别

·table标签

- 具有语义化的表格元素

·display:table

- 能够让一个html元素和它的子节点像table元素一样

·可以使用基于表格的css布局，现在逐渐淘汰了table布局

·用div+css编写出来的文件比用table编写出来的文件小

·table必须在页面完全加载后才显示，div则是逐行显示

·table的嵌套性太多，没有div简洁

### z-index的定位方法

·设置元素的层级

·可以为z-index指定一个整数，该值会作为当前元素的层级，层级高的优先显示

·对于没有开启定位的元素不能使用z-index

·父元素的层级再高，也不会盖住子元素

### 如何不操作它本身改变一个DOM元素的字体颜色？

·更改父元素的color

### 用的最多的css属性是啥？

·flex属性，灵活但是兼容性方面不强

### line-height和height的区别

·line-height

- 指布局里面一段文字的行高，是针对字体来设置的

·height

- 指容器的整体高度

### 背景颜色会填充哪些区域？

·background-color设置的背景颜色会填充元素的content、padding、border区域

### inline-block、inline和block的区别？

·inline

- 将一个元素作为内联元素显示

- 不会独占一行

- 宽高无效

- 垂直外边距无效

·block

- 将一个元素作为块元素显示

- 独占一行

·inline-block

- 将一个元素转换为行内块元素

- 可以使一个元素既有内联元素的特点，又有块元素的特点

- 既可以设置宽高，又不会独占一行

### 为什么img是inline还可以设置宽高

·img是可替换元素

·通过src设置内容

·拥有内置的宽高

·性质与inline-block相同，可以设置宽高

### 两个嵌套的div，position都是absolute，子div设置top属性，那么这个top是相对于父元素的哪个位置定位的

·margin的外边缘

### css的常用选择器

·id选择器，类选择器，伪类选择器等

### css布局

·圣杯布局、双飞翼布局、Flex布局、绝对定位布局、表格布局、网格布局

·圣杯布局

- 布局从上到下分为header、container、footer

- 然后container部分定为三栏布局

- 这种布局方式同样分为header、container、footer

- center 是在 container 的padding中的，宽度小的时候会出现混乱

·双飞翼布局

- 给center 部分包裹了一个 main

- 通过设置margin主动地把页面撑开

·Flex布局

- 弹性盒布局

- 是由CSS3提供的一种灵活方便的布局

·绝对定位布局

- 是给container 设置position: relative和overflow: hidden

- 绝对定位的元素的参照物为第一个postion不为static的祖先元素

- 通过设置left和right并把两边撑开

- center 设置top: 0和bottom: 0使其高度撑开

·表格布局

- 能使三栏的高度统一

·网格布局

- 最强大的布局方式，使用起来极其方便

- 兼容性并不好

- 可以将页面分割成多个区域，或者用来定义内部元素的大小，位置，图层关系

### css预处理器有什么

·less，sass等

### 实现一个两列等高布局，讲讲思路

·为了实现两列等高，可以给每列加上 padding-bottom:9999px;

·同时父元素设置overflow:hidden;

## JavaScript

### get请求传参长度的误区

·误区：

- 们经常说get请求参数的大小存在限制，而post请求的参数大小是无限制的

·HTTP 协议未规定 GET 和POST的长度限制

·GET的最大长度显示是因为 浏览器和 web服务器限制了 URI的长度

·不同的浏览器和WEB服务器，限制的最大长度不一样

·要支持IE，则最大长度为2083byte，若只支持Chrome，则最大长度 8182byte

### 补充get和post请求在缓存方面的区别

·get

- 用于获取数据

- 用户获取数据可以使用缓存，不用每次都与数据库连接

·post

- 用于提交数据

- 一般是修改和删除的工作，必须与数据库交互

- 不能使用缓存

### 说一下闭包

·当一个嵌套的内部（子）函数引用了嵌套的外部（父）函数的变量（或函数）时，就产生了闭包

·使函数内部的变量在函数执行完后，仍然存活在内存中（延长了局部变量的生命周期）

·让函数外部可以操作（读写）到函数内部的数据（变量/函数）

### 说一下类的创建和继承

·类的创建（es5）

- Animal.prototype.eat = function(food) {};

- new一个function，在这个function的prototype里面增加属性和方法

- 这样就生成了一个Animal类，实例化生成对象后，有方法和属性

·类的继承——原型链继承

- Cat.prototype = new Animal();

- 将父类的实例对象赋值给子类的prototype

- 特点

-- 基于原型链，既是父类的实例，也是子类的实例

- 缺点

-- 无法实现多继承

·构造继承

- Animal.call(this);

- 使用父类的构造函数来增强子类实例

- 特点

-- 可以实现多继承

- 缺点

-- 只能继承父类实例的属性和方法

-- 不能继承原型上的属性和方法

·实例继承

- 为父类实例添加新特性，作为子类实例返回

·拷贝继承

- 拷贝父类元素上的属性和方法

·组合继承

- 相当于构造继承和原型链继承的组合体

- 调用父类构造，继承父类的属性并保留传参的优点

- 将父类实例作为子类原型，实现函数复用

function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

Cat.prototype = new Animal();

Cat.prototype.constructor = Cat;

·寄生组合继承

- 通过寄生方式，砍掉父类的实例属性

- 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

- 较为推荐

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();

})();

·es6有专门的类、构造函数、静态成员、继承、get和set的概念和使用

### 如何解决异步回调地狱

·promise、generator、async/await

### 说说前端中的事件流

·HTML中与javascript交互是通过事件驱动来实现的

- 例如鼠标点击事件onclick、页面的滚动事件onscroll等等

- 可以向文档或者文档中的元素添加事件侦听器来绑定事件

事件流

- 事件流描述的是从页面中接收事件的顺序

-- 事件捕获阶段

-- 处于目标阶段

-- 事件冒泡阶段

·addEventListener

- 接收3个参数：要处理的事件名、作为事件处理程序的函数和一个布尔值

- 最后这个布尔值参数如果是true，表示在捕获阶段调用事件处理程序

- 如果是false，表示在冒泡阶段调用事件处理程序，默认值

·IE只支持事件冒泡

### 如何让事件先冒泡后捕获

·监听到捕获事件时，先暂缓执行，直到冒泡事件被捕获后再执行捕获事件

### 说一下事件委托

·事件委托

- 不在事件的发生地（直接dom）上设置监听函数

- 是在其父元素上设置监听函数

- 通过事件冒泡，父元素可以监听到子元素上事件的触发，通过判断事件发生元素DOM的类型，来做出不同的响应

·最经典的就是ul和li标签的事件监听

- 比如我们在添加事件时候，采用事件委托机制

- 不会在li标签上直接添加，而是在ul父元素上添加

·比较合适动态元素的绑定

- 新添加的子元素也会有监听函数，也可以有事件触发机制

### 说一下图片的懒加载和预加载

·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

·懒加载

- 懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

·两种技术的本质

- 两者的行为是相反的，一个是提前加载，一个是迟缓甚至不加载。

- 懒加载对服务器前端有一定的缓解压力作用

- 预加载则会增加服务器前端压力

### mouseover和mouseenter的区别

·mouseover

- 当鼠标移入元素或其子元素都会触发事件

- 有一个重复触发，冒泡的过程

- 对应的移除事件是mouseout

·mouseenter

- 当鼠标进入元素本身（不包含元素的子元素）会触发事件

- 也就是不会冒泡

- 对应的移出事件是mouseleave

### js的new操作符做了哪些事情

·new 操作符新建了一个空对象

·这个对象原型指向构造函数的prototype

·执行构造函数后返回这个对象

### 改变函数内部this指针的指向函数（bind，apply，call的区别）

·通过apply和call改变函数的this指向

- 他们两个函数的第一个参数都是一样

- 表示要改变指向的那个对象

- 第二个参数，apply是数组，而call则是正常传参形式

·通过bind改变this作用域会返回一个新的函数，这个函数不会马上执行

### js的各种位置的区别？

·clientHeight

- 内容区+内边距，不包含border和滚动条

·offsetHeight

- 内容区+内边距+边框+滚动条，不包含因为滚动被隐藏的部分

·scrollHeight

- 表示了所有区域的高度，包含了因为滚动被隐藏的部分

·clientTop

- 表示边框border的厚度，在未指定的情况下一般为0

·scrollTop

- 滚动后被隐藏的高度，获取对象相对于由offsetParent属性指定的父坐标(css定位的元素或body元素)距离顶端的高度。

### js拖拽功能的实现

·onmousedown响应事件，鼠标被按下

·onmousemove响应事件，鼠标移动

·onmouseup响应事件，鼠标松开

·div的左偏移量 = 鼠标.clentX - 元素.offsetLeft

·div的上偏移量 = 鼠标.clentY - 元素.offsetTop

·当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容，此时会导致拖拽功能异常。可以通过return false来取消默认行为

·setCapture()设置元素捕获鼠标按下的事件，只有ie支持

### 异步加载js的方法

·defer

- 只支持IE

- 如果您的脚本不会改变文档的内容

- 可将 defer 属性加入到<script>标签中，以便加快处理文档的速度

·async

- HTML5属性仅适用于外部脚本

- 并且如果在IE中，同时存在defer和async，那么defer的优先级比较高，脚本将在页面完成时执行

- 创建script标签，插入到DOM中

### Ajax解决浏览器缓存问题

·在ajax发送请求前加上

- anyAjaxObj.setRequestHeader("If-Modified-Since","0")

·在ajax发送请求前加上

- anyAjaxObj.setRequestHeader("Cache-Control","no-cache")

·在URL后面加上一个随机数

- "fresh=" + Math.random()

·在URL后面加上时间戳

- "nowtime=" + new Date().getTime()。

·jQuery使用 $.ajaxSetup({cache:false})

- 不需要保存缓存记录

- 所有ajax都会执行这条语句

### 垃圾回收机制(GC)

·处理程序运行过程中产生的垃圾

·当一个对象没有任何的变量或属性对它进行引用，此时我们将永远无法操作该对象，这种对象称为垃圾，会占用大量的内存空间，导致程序运行变慢

·JS中拥有自动的垃圾回收机制，会自动将这些垃圾对象从内存中销毁，我们不需要也不能手动进行垃圾回收操作

·我们需要做的只是将不再使用的对象设置为null即可

### eval是做什么的

·该函数可以执行一段字符串形式的JS代码，并将执行结果返回

·在开发中尽量不要使用，首先它的执行性能比较差，然后它还具有安全隐患。

·如果要兼容ie7及以下浏览器的JSON操作，则可以引用外部文件json2.js来处理

### 如何理解前端模块化

·前端模块化就是复杂的文件编程一个一个独立的模块，比如js文件等等

·分成独立的模块有利于重用（复用性）和维护（版本迭代）

·会引来模块之间相互依赖的问题，所以有了commonJS规范，AMD，CMD规范等等，以及用于js打包（编译等处理）的工具webpack

### 说一下Commonjs、AMD和CMD

·一个模块是能实现特定功能的文件

·有了模块就可以方便的使用别人的代码

·想要什么功能就加载什么模块

·Commonjs

- 开始于服务器端的模块化，同步定义的模块化

- 每个模块都是一个单独的作用域

- 暴露模块，modules.exports

- 引入模块require()。

·AMD

- require（）函数在加载依赖函数的时候是异步加载的，这样浏览器不会失去响应

- 它指定的回调函数，只有前面的模块加载成功，才会去执行

- 网页在加载js的时候会停止渲染，因此我们可以通过异步的方式去加载js

·CMD

- 与AMD相似

- AMD依赖前置加载，提前加载执行

- CMD就近加载，延迟加载

### 对象深度克隆的简单实现

·ES5的常用的对象克隆的一种方式

·注意数组是对象，但是跟对象又有一定区别

function deepClone(obj){

//用来存放克隆的数组/对象，判断要克隆的是数组还是对象

var newObj= obj instanceof ***Array*** ? []:{};

//遍历数组或对象属性

for(var item in obj){

// 判断item是否还需要克隆，如果需要则递归函数

var temple= typeof obj[item] == 'object' ? deepClone(obj[item]):obj[item];

//将数据存入newObj

newObj[item] = temple;

}

return newObj;

}

### 实现一个once函数，传入函数参数只执行一次

function ones(func){

var tag=true;

return function(){

if(tag==true){

func.apply(null,arguments);

tag=false;

}

return undefined

}

}

### 将原生的ajax封装成promise

var myNewAjax=function(url){

//返回promise对象

return new ***Promise***(function(resolve,reject){

// 创建xhr

var xhr = new ***XMLHttpRequest***();

//初始化 设置请求方法和url

xhr.open('get',url);

//发送

xhr.send(***data***);

//事件绑定 处理服务端返回的结果

xhr.onreadystatechange=function(){

//xhr.readyState获取返回状态和状态码

//4表示服务端返回了所有结果

//响应状态码中2开头的表示成功，并非只有200

if(xhr.status==200&&readyState==4){

// 将得到的数据解析成js对象

var json=***JSON***.parse(xhr.responseText);

//返回数据

resolve(json)

}else if(xhr.readyState==4&&xhr.status!=200){

//返回错误信息

reject('error');

}

}

})

}

### js监听对象属性的改变

·我们假设这里有一个user对象

·在ES5中可以通过Object.defineProperty来实现已有属性的监听

- 缺点：如果id不在user对象中，则不能监听id的变化

- vue2的双向绑定实现原理

***Object***.defineProperty(user,'name',{

set：function(key,value){

}

})

·在ES6中可以通过Proxy来实现

- 即使有属性在user中不存在

- 通过user.id来定义也可以监听这个属性的变化

- vue3的双向绑定实现原理

var ***user*** = new ***Proxy***({}，{

set：function(target,key,value,receiver){

}

})

### 如何实现一个私有变量，用getName方法可以访问，不能直接访问

·通过defineProperty来实现

***obj***={

name:yuxiaoliang,

getName:function(){

return this.name

}

}

object.defineProperty(obj,"name",{

//不可枚举不可配置

});

·通过函数的创建形式

function product(){

var name='yuxiaoliang';

this.getName=function(){

return name;

}

}

var ***obj***=new product();

### ==和===、以及Object.is的区别

·==

- 只判断值是否相同，如果数据类型不同，则会转换成相同的进行比较

”1” == 1 //true

- undefined 衍生自 null,两值判断返回true。Undefined == null //true

- NaN不和任何值相等，包括它本身。可通过isNaN()函数判断b是否为NaN

·===

- 判断值和数据类型是否相同，类型不同直接返回false

·Object.is

- 判断两个值是否完全相等

- 类似于===

- 主要的区别就是+0！=-0 而NaN==NaN

### setTimeout、setInterval和requestAnimationFrame之间的区别

·requestAnimationFrame不需要设置时间间隔

·requestAnimationFrame会把每一帧中的所有DOM操作集中起来

·在一次重绘或回流中就完成，并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率

·在隐藏或不可见的元素中，requestAnimationFrame将不会进行重绘或回流

·这当然就意味着更少的CPU、GPU和内存使用量

·requestAnimationFrame是由浏览器专门为动画提供的API，在运行时浏览器会自动优化方法的调用

·并且如果页面不是激活状态下的话，动画会自动暂停，有效节省了CPU开销

### js怎么控制一次加载一张图片，加载完后再加载下一张

·使用onload实现

var ***obj***=new ***Image***();

***obj***.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";

//资源加载完成后执行

***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+"' />";

}

·使用onreadystatechange实现

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+"' />";

}

}

### 如何实现sleep的效果

·while循环的方式

- 容易造成死循环

function sleep(ms){

var start=***Date***.now(),expire=start+ms;

while(***Date***.now()<expire);

***console***.log('1111');

return;

}

·通过promise来实现

function sleep(ms){

//返回promise

return new ***Promise***(

(resolve)=>{

//延时一段时间,然后返回成功状态

setTimeout(resolve,ms);

}

);

}

sleep(5000).then(function(){

***console***.log(222)

})

·通过async封装

function sleep(ms){

// 返回promise

return new ***Promise***((resolve)=>setTimeout(resolve,ms));

}

async function test(){

var temple=await sleep(5000);

***console***.log(1111)

return temple

}

test();

·通过generate来实现

### 简单的实现一个promise

·初始版本myPromise

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);

}

}

### Function.\_proto\_(getPrototypeOf)是什么？

·获取一个对象的原型

·在chrome中可以通过\_proto\_的形式

·在ES6中可以通过Object.getPrototypeOf的形式。

·Function.\_\_proto\_\_==Object.prototype //false

·Function.\_\_proto\_\_==Function.prototype//true

·Function的原型也是Function

### ES6箭头函数的特性

·箭头函数没有this

- 需要通过查找作用域链来确定this的值

- this绑定的是最近一层非箭头函数的this

·箭头函数没有自己的arguments对象

- 但是可以访问外围函数的arguments对象

·不能通过new关键字调用

- 同样也没有new.target值和原型

### 箭头函数中this指向举例

·箭头函数的this会往外找

var ***a*** = 11;

function test2(){

this.a=22;

let b=()=>{

***console***.log(this.a);//22

}

b();

}

var ***x*** = new test2();

### js判断类型

·typeof()，instanceof，Object.prototype.toString.call()等

### 数组常用方法

·filter()、push()、pop()、shift()、unshift()、splice()、sort()、reverse()、map()等

### 数组去重

·indexOf循环去重

·ES6 Set去重

let ***arr*** = [1,2,3,4,5,4,3,2,1];

//数组去重,并扩展

let ***result*** = [...new ***Set***(***arr***)];

***console***.log(***result***);// [1, 2, 3, 4, 5]

·Object 键值对去重

- 把数组的值存成 Object 的 key 值，比如 Object[value1] = true

- 在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的

### 事件代理在捕获阶段的实际应用

·可以在父元素层面阻止事件向子元素传播，也可代替子元素执行某些操作

### 去除字符串首尾空格

·使用trim()

·使用正则(^\s\*)|(\s\*$)

### 性能优化

·减少HTTP请求

·使用内容发布网络（CDN）

·添加本地缓存

·压缩资源文件

·将CSS样式表放在顶部，把javascript放在底部（浏览器的运行机制决定）

·避免使用CSS表达式

·减少DNS查询

·使用外部javascript和CSS

·避免重定向

·图片lazyLoad

### 能来讲讲JS的语言特性吗

·运行在客户端浏览器上；

·不用预编译，直接解析执行代码；

·是弱类型语言，较为灵活；

·与操作系统无关，跨平台的语言；

·脚本语言、解释性语言

### 如何判断一个数组

·Object.prototype.call.toString()

·instanceof

### JS实现跨域

·**JSONP**

- 通过动态创建script，再请求一个带参网址实现跨域通信

·document.domain + iframe跨域

- 两个页面都通过js强制设置document.domain为基础主域，就实现了同域

·location.hash + iframe跨域

- a域与b跨域相互通信，通过中间页c来实现。

- 三个页面，不同域之间利用iframe的location.hash传值，相同域之间直接js访问来通信

·window.name + iframe跨域

- 通过iframe的src属性由外域转向本地域

- 跨域数据即由iframe的window.name从外域传递到本地域

·postMessage跨域

- 可以跨域操作的window属性之一。

**·CORS**

- 服务端设置Access-Control-Allow-Origin即可

- 前端无须设置，若要带cookie请求，前后端都需要设置。

·代理跨域

- 开启一个代理服务器，实现数据的转发

### 数据类型

#### Js基本数据类型

·undefined、null、number、boolean、string、symbol

#### Js引用数据类型

·Object、BigInt

#### 引用类型常见的对象

·Object、Array、RegExp、Date、Function

·特殊的基本包装类型(String、Number、Boolean)

·单体内置对象(Global、Math)

#### 基本数据类型和引用数据类型的区别

·基本数据类型的值不可变

·引用数据类型的值可变

·基本数据类型不能添加属性和方法

·引用数据类型可以添加属性和方法

·基本数据类型的赋值是简单赋值

·引用数据类型的赋值是对象引用

·基本数据类型的比较是值的比较

·引用类型的比较是内存地址的比较

·基本数据类型存放在栈区

·引用数据类型同时保存在栈区和堆区

#### NaN是什么

·Not a Number,表示不是数字

·数据类型是数值，它不等于任何值，包括自身

·在布尔运算时被当做false

·NaN与任何数运算得到的结果都是NaN

·当运算失败或者运算无法返回正确的数值的就会返回NaN

### js的命名方式

·命名时尽量使用英文，如果不会可以使用拼音，但是不要英文和拼音混用

·命名格式

- 驼峰命名法

-- 首字母小写，每个单词的开头字母大写

- 也可以所有字母都小写，单词之间用\_或-连接

- 公司让你怎么写你就怎么写

### 重排和重绘，讲讲看

#### 重绘（repaint或redraw）

·触发重绘的条件

- 改变元素外观属性。如：color，background-color等。

·当盒子的位置、大小以及其他属性（颜色、字体大小等）都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面上

·重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观

·注意

- table及其内部元素可能需要多次计算才能确定好其在渲染树中节点的属性值，比同等元素要多花两倍时间，这就是我们尽量避免使用table布局页面的原因之一

#### 重排（重构/回流/reflow）

·当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建, 这就称为回流(reflow)。

·每个页面至少需要一次回流，就是在页面第一次加载的时候。

#### 重绘和重排的关系

·在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树

·完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程称为重绘

·重排必定会引发重绘，但重绘不一定会引发重排

### 跨域的原理

·跨域是指浏览器不能执行其他网站的脚本

·由浏览器的同源策略造成，是浏览器对JavaScript实施的安全限制

·只要协议、域名、端口有任何一个不同，都被当作是不同的域

·跨域原理，即是通过各种方式，避开浏览器的安全限制

### null == undefined为什么

·要比较相等性之前，不能将null 和 undefined 转换成其他任何值

·但 null == undefined 会返回 true

·ECMAScript规范中是这样定义的

### this的指向有哪几种

·默认绑定

- 全局环境中，this默认绑定到window。

·隐式绑定

- 一般地，被直接对象所包含的函数调用时，也称为方法调用，this隐式绑定到该直接对象。

·隐式丢失

- 隐式丢失是指被隐式绑定的函数丢失绑定对象，从而默认绑定到window

·显式绑定

- 通过call()、apply()、bind()方法把对象绑定到this上，叫做显式绑定

·new绑定

- 如果函数或者方法调用之前带有关键字new，它就构成构造函数调用。对于this绑定来说，称为new绑定。

### 暂停死区

·在代码块内，使用let、const命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”

### AngularJS双向绑定原理

·Angular将双向绑定转换为一堆watch表达式

·然后递归这些表达式检查是否发生过变化

·如果变了，则执行相应的watcher函数

·等到model中的值不再发生变化，也就不会再有watcher被触发

·一个完整的digest循环就完成了

·Angular中在view上声明的事件指令

·如：ng-click、ng-change等

·会将浏览器的事件转发给$scope上相应的model的响应函数

·等待相应函数改变model，紧接着触发检查机制刷新view。

·watch表达式

- 可以是一个函数、可以是$scope上的一个属性名，也可以是一个字符串形式的表达式

- $watch函数所监听的对象叫做watch表达式

·watcher函数

- 指在view上的指令（ngBind，ngShow、ngHide等）以及{{}}表达式，他们所注册的函数。

- 每一个watcher对象都包括

-- 监听函数

-- 上次变化的值

-- 获取监听表达式的方法

-- 监听表达式

-- 是否需要使用深度对比（angular.equals()）

### 小明在玩Flappy Bird时候老是卡顿甚至崩溃，说出原因（3-5个）以及解决办法（3-5个）

·内存溢出问题

- 不断生成的钢管不及时清理容易导致内存溢出游戏崩溃

- 在钢管离开可视区域后，销毁钢管，让垃圾收集器回收钢管

·资源过大问题

- 绘制图片需要较大计算量

- 选择图片文件大小更小的图片格式，比如使用webp、png格式的图片

·资源加载问题

- 在可视区域生成钢管，用户的体验就认为钢管是卡顿后才生成的，不流畅

- 在可视区域之前就预加载好资源

·canvas绘制频率问题

- 大部分显示器刷新频率为60次/s

- 游戏的每一帧绘制间隔时间需要小于1000/60=16.7ms，才能让用户觉得不卡顿

（这是单机游戏，回答应该与网络无关）

### 什么是按需加载

·当用户触发了动作时才加载对应的功能

·触发的动作，是要看具体的业务场景而言

·包括但不限于以下几个情况

- 鼠标点击、输入文字、拉动滚动条，鼠标移动、窗口大小更改等

·加载的文件，可以是JS、图片、CSS、HTML等

### 说一下什么是virtual dom（虚拟DOM）

·用JavaScript 对象结构表示 DOM 树的结构；

·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

·用新的树和旧的树进行比较，记录两棵树差异

·把所记录的差异应用到所构建的真正的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

### webpack用来干什么的

· JavaScript 应用程序的静态模块打包器(module bundler)

·当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块

·然后将所有这些模块打包成一个或多个bundle

### 写一个函数，第一秒打印1，第二秒打印2

·使用let块级作用域实现

function StopWatch(s) {

//使用let后,会保存声明时的值,类似闭包,不能用var

for(let i=1;i<=s;i++){

//for循环的代码是瞬间执行的,需要设置定时器

//会瞬间同时设置s个定时器,从设置定时器开始计时

//分别表示s秒后执行

setTimeout(()=>{

***console***.log(i);

},1000\*i);

}

}

StopWatch(10);

·使用闭包实现

function StopWatch(s) {

for(var i=1;i<=s;i++){

(function(i){

setTimeout(function(){

***console***.log(i)

},1000\*i)

})(i)

}

}

StopWatch(10);

### vue的生命周期

·beforeCreate

- 数据检测和数据代理创建之前，无法通过vm访问到data中的数据、methods中的方法

·created

- 数据检测和数据代理创建完成，可以通过vm访问到data中的数据、methods中的方法，模板未解析

·beforeMount

- 挂载之前，模板解析完成，未转换成真实dom，此处对dom的操作都不生效

· mounted

- 挂载完成，已转换成真实dom，可对dom操作，但最好不要

- 一般在此处进行：开启定时器、发送网络请求、订阅消息、绑定自定义事件等初始化操作

- 已将真实dom赋值给vm.$el

·beforeUpdate

- 当data中的数据改变时,更新之前,此时数据是新的，但页面是旧的

·updated

- 数据是新的，页面也是新的

· beforeDestroy

- vm.$destroy()会完全销毁一个实例，清理它与其他实例的连接，解绑它的全部指令及事件监听器，click事件可用，但不会更新数据和页面

- 当vm.$destroy()开始调用时，销毁之前

- data、methods、指令等都处于可用状态，但是对数据的所有操作都不会触发页面更新

- 一般在此时关闭定时器、取消订阅消息、解绑自定义事件等收尾操作

·destroyed

- 销毁完成，完全销毁一个实例，清理它与其他实例的连接，解绑它的全部指令及事件监听器，click事件可用，但不会更新数据和页面

·NextTick

- 在下一次 DOM 更新结束后执行其指定的回调

·activated

- 路由组件被激活时触发

·deactivated

- 路由组件失活时触发

### 简单介绍一下symbol

·Symbol是ES6 的新增属性，代表用给定名称作为唯一标识

- 这种类型的值可以这样创建，let id=symbol(“id”)

·Symbl确保唯一，即使采用相同的名称，也会产生不同的值

- 我们创建一个字段，仅为知道对应symbol的人能访问

- 使用symbol很有用，symbol并不是100%隐藏

- Object.getOwnPropertySymbols(obj)可以获得所有的symbol

- Reflect.ownKeys(obj)返回对象所有的键，包括symbol

·并不是真正隐藏，但大多数库内置方法和语法结构遵循通用约定他们是隐藏的

### 什么是事件监听

·使用addEventListener绑定事件监听

·addEventListener(event, function, useCapture)

- event指定事件名

- function指定要事件触发时执行的函数

- useCapture指定事件是否在捕获或冒泡阶段执行

·事件传递有两个阶段，冒泡阶段和捕获阶段

·捕获阶段

- 在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件。

·冒泡阶段

- 事件从目标元素向他的祖先元素传递，依次触发祖先元素上的事件

### 介绍一下promise

·Promise是一个对象，保存着未来将要结束的事件

·特征

- 对象的状态不受外部影响

-- Promise对象代表一个异步操作，有三种状态

-- pending进行中

-- fulfilled已成功

-- rejected已失败

-- 只有异步操作的结果，才可以决定当前是哪一种状态

-- 任何其他操作都无法改变这个状态，这也就是promise名字的由来

- 一旦状态改变，就不会再变

-- promise对象状态改变只有两种可能

-- 从pending改到fulfilled或者从pending改到rejected

-- 只要这两种情况发生，状态就凝固了，不会再改变

-- 这个时候就称为定型resolved,

### 说C++、Java、JavaScript这三种语言的区别

#### 从静态类型还是动态类型来看

·静态类型

- 编译的时候就能够知道每个变量的数据类型

- 编程的时候也需要给定数据类型

- C、C++、Java都属于静态类型语言

- 在编译后会大量利用已知类型的优势

- 编译后的代码就可以用内存地址加偏移量的方法存取变量

- 而地址加偏移量的算法汇编很容易实现

·动态类型

- 运行的时候才知道每个变量的类型

- 编程的时候无需显示指定类型

- JavaScript、Ruby、Python都属于动态类型语言

- 会当做字符串通通存下来，之后存取就用字符串匹配

#### 从编译型还是解释型来看

·编译型语言

- 像C、C++，需要编译器编译成本地可执行程序后才能运行

- 由开发人员在编写完成后手动实施

- 用户只使用这些编译好的本地代码，这些本地代码由系统加载器执行

- 由操作系统的CPU直接执行，无需其他额外的虚拟机等

·解释性语言

- 像JavaScript、Python，开发语言写好后直接将代码交给用户

- 用户使用脚本解释器将脚本文件解释执行

- 对于脚本语言，没有开发人员的编译过程，当然，也不绝对

#### JavaScript与Java的不同

·Java

- 将源代码编译成字节码，这个同执行阶段是分开的

- 也就是从源代码到抽象语法树到字节码这段时间的长短是无所谓的

·JavaScript

- 在网页和JavaScript文件下载后同执行阶段一起在网页的加载和渲染过程中实施

- 处理时间有严格要求

### 原型链

·原型链的顶端

- Object的原型对象（Object.prototype）

·Object的原型的原型

- null

·函数的显示原型指向的对象默认是空Object实例对象（但Object不满足）

·所有函数都是Function的实例（包含Function本身）

### JS中string的startwith和indexof两种方法的区别

·Startwith

- 参数有3个

- stringObj

-- 要搜索的字符串对象

- str，

-- 搜索的字符串

- position

-- 可选，从哪个位置开始搜索

- 如果以position开始的字符串以搜索字符串开头，则返回true，否则返回false

·Indexof函数

- indexof函数可返回某个指定字符串在字符串中首次出现的位置

### js字符串转数字的方法

·使用parseInt（）

- 将一个字符串前面的整数返回

### Let、const、var的区别

·var声明的变量是全局或者整个函数块的

·let和const声明的变量是块级的变量

·var声明的变量存在变量提升

·let和const不存在变量提升

·let声明的变量允许重新赋值

·const不允许重新赋值

### 什么是块级作用域

·声明的变量只在块内有效

### 如何用ES5的方法实现块级作用域

·使用立即执行函数

### setTimeout和Promise的执行顺序

·先执行Promise构造函数内的代码

·然后执行promise.then

·再执行setTimeout

### 平时怎么调试JS

·用Chrome自带的控制台

### JS的作用域类型

·函数作用域

- 如果在函数内部我们给未定义的一个变量赋值，这个变量会转变成为一个全局变量

·块作用域

- 块作用域吧标识符限制在{}中

### 改变函数作用域的方法

·eval（）

- 执行一段字符串形式的JS代码

·with关键字

- 重复引用同一个对象的多个属性的快捷方式

### setTimeout(fn,100);100毫秒是如何权衡的

·setTimeout()函数只是将事件插入了任务列表

·必须等到当前代码执行完，主线程才会去执行它指定的回调函数

·有可能要等很久

·没有办法保证回调函数一定会在setTimeout指定的时间内执行

·100毫秒是插入队列的时间+等待的时间

### 怎么获得对象上的属性

·for（let I in obj）

- 该方法依次访问一个对象及其原型链中所有可枚举的类型

·object.keys

- 返回一个数组

- 包括所有可枚举的属性名称

·object.getOwnPropertyNames

- 返回一个数组

- 包含不可枚举的属性

### 简单讲一讲ES6的一些新特性

（以下回答夹杂着ES6-11的新特性）

·let、const声明变量

·模板字符串

·箭头函数

·局部变量

·解构赋值

·对字符串、 数组、正则、对象、函数等拓展了一些方法

·引入了新的数据类型symbol和BigInt

·新的数据结构set和map

·为解决异步回调问题，引入了promise和 generator

·有了Class和模块，实际开发中需要使用babel进行编译模块

·await/async

·重要的特性

- 块级作用域

-- 不再需要立即执行的函数表达式，循环体中的闭包不再有问题

- rest参数

-- 用于获取函数的多余参数，这样就不需要使用arguments对象了

- promise

-- 一种异步编程的解决方案

- 模块化

-- export命令用于规定模块的对外接口

-- import命令用于引入其他模块提供的功能

### 如何使不同页面之间进行通信

·两个tab页具有依赖关系

- window.postMessage

· 两个页面属于同源范畴

- storage事件

·两个互不相关的tab页面通信

- 结合window.postMessage与 storage事件

### 如果已经有三个promise，A、B和C，想串行执行，该怎么写？

// promise

A.then(B).then(C).catch(...)

// async/await

(async ()=>{

await a();

await b();

await c();

})()

### 知道private和public吗

·public

- 公有属性

- 所有用户都可以直接进行调用

·private

- 私有属性

- 除class自己之外，任何人都不能直接使用

- 可通过get和set方法使用

### async和await具体该怎么用？

(async () = > {

await new promise();

})()

### js加载过程阻塞，解决方法。

·指定script标签的async属性

·脚本相对于页面的其余部分异步地执行

### JavaScript中的轮播实现原理？

·图片轮播的原理就是图片排成一行

·然后准备一个只有一张图片大小的容器

·对这个容器设置超出部分隐藏

·再控制定时器来让这些图片整体左移或右移

·如果有两个轮播，可封装一个轮播组件，供两处调用

### 怎么实现一个计算一年中有多少周？

·是平年还是闰年

- 如果是平年

·1月1号是周几

- 如果是周五，一年365天把1号 2号3号减去，也就是把第一个不到一周的天数减去等于362

·最后一天是周几

- 如果是周五，需要把周一到周五减去，也就是362-5=357

·正常情况 357这个数计算出来是7的倍数。357/7=51 。即为周数

### new操作符原理

·创建一个类的实例

·创建一个空对象obj

·把空对象的\_\_proto\_\_设置为构造函数的prototype

·初始化实例

·构造函数被传入参数并调用

·关键字this被设定指向该实例obj

·返回实例obj

### arguments

·arguments是伪数组对象

- 有length属性

- 不能调用数组方法

- 可用Array.from()转换

### 箭头函数获取arguments

·可用…rest参数获取

### Eventloop

·任务队列中，在每一次事件循环

·macrotask只会提取一个执行

·microtask会一直提取，直到microsoft队列为空为止

### 数组移除第一个元素的方法有哪些？

·splice和shift等

## jQuery

### 为什么用Jquery？

·强大选择器: 方便快速查找DOM元素

·隐式遍历(迭代): 一次操作多个元素

·读写合一: 读数据/写数据用的是一个函数

·链式调用: 可以通过.不断调用jQuery对象的方法

### jQuery核心对象

·执行jQuery/$函数返回的就是jQuery对象

·jQuery对象内部包含的是DOM元素对象的伪数组（可能只有一个元素）

·jQuery对象有很多有用的属性和方法

### jQuery的选择器有哪些？

·基本的css选择器

·层次选择器

·过滤选择器

·表单选择器

## BootStrap

### 容器有哪些

·流体容器

- container-fluid

- 容器宽度width = auto

- 左右padding值为15px

·固定容器

- 左右padding值为15px

### 栅格盒模型设计的精妙之处

·一行分为12列

·响应式布局，适应不同的窗口大小

·容器上两边具有15px的padding

·容器有box-sizing: border-box;样式

- 将容器的宽度和高度用来设置整个盒子可见框的大小

- 防止列溢出

·行两边具有-15px的margin

- 将行与容器两边的15px的padding

- 使整个容器完全包裹行

- 为了能使列嵌套行

·列两边具有15px的padding

- 为了维护槽宽的规则。列两边必须右15px的padding

### 栅格系统实现原理

·为流体容器和固定容器设置公共样式

- padding

- 左右margin auto

- 槽宽

- clearfix解决高度塌陷和外边距重叠

·为固定容器特定样式

- 使用媒体查询设置容器宽度

·通过递归为所有col选择器设置

- 向左浮动

- 宽度

- 列排序

- 列偏移

# 前端核心

## 服务端编程

### JSONP的缺点

·JSONP只支持get，因为script标签只能使用get请求

·JSONP需要后端配合返回指定格式的数据

### dom是什么，你的理解？

·文档对象模型（Document Object Model，简称DOM）

- 是W3C组织推荐的处理可扩展标志语言的标准编程接口

- 组织页面（或文档）的对象被组织在一个树形结构中

- 用来表示文档中对象的标准模型就称为DOM

### 关于dom的api有什么

·节点创建型api，页面修改型API，节点查询型API，节点关系型api，元素属性型api，元素样式型api等

## AJAX

### Ajax的返回状态

·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，已经接收到全部响应内容

·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

### 原生Ajax实现

·AJAX创建异步对象XMLHttpRequest

·xhr.open()设置请求方法和url

·xhr,send()发送

·处理服务端返回的结果

·readyState获取返回状态，若为4说明响应内容解析完成

·判断响应报文的状态，若为200说明服务器正常运行并返回响应数据

·读取响应数据，可以通过 responseText 属性来取回由服务器返回的数据

### 如果我想发出两个有顺序的ajax需要怎么做？

·发出两个有顺序的ajax

- 使用回调函数

- 使用Promise.then

- 使用async等

### Fetch和Ajax比有什么优缺点？

·promise方便异步

·在不想用jQuery的情况下，相比原生的ajax，也比较好写

### 发送ajax请求的方式

·原生js发送AJAX请求

·jQuery发送AJAX请求

·Axios发送AJAX请求

- 最常用

·fetch函数发送AJAX请求

### IE缓存问题解决

·在一些浏览器中(IE),由于缓存机制的存在，ajax 只会发送的第一次请求，剩 余多次请求不会在发送给浏览器而是直接加载缓存中的数据。

·可以传入t=时间戳解决问题

## 移动Web开发

### 讲讲viewport和移动端布局

·像素：

- 屏幕是由一个一个发光的小点构成，这一个个的小点就是像素

- 分辨率：1920 x 1080 说的就是屏幕中小点的数量

- 在前端开发中像素要分成两种情况讨论：CSS像素 和 物理像素

- 物理像素，上述所说的小点点就属于物理像素

- CSS像素，编写网页时，我们所用像素都是CSS像素

- 浏览器在显示网页时，需要将CSS像素转换为物理像素然后再呈现

- 一个css像素最终由几个物理像素显示，由浏览器决定

·视口（viewport）

- 视口就是屏幕中用来显示网页的区域

- 可以通过查看视口的大小，来观察CSS像素和物理像素的比值

·完美视口

- 移动端默认的视口大小是980px(css像素)

- 默认情况下，移动端的像素比就是  980/移动端宽度  （980/750）

- 如果我们直接在网页中编写移动端代码，这样在980的视口下，像素比是非常不好，导致网页中的内容非常非常的小

- 编写移动页面时，必须要确保有一个比较合理的像素比

- 将像素比设置为最佳像素比的视口大小我们称其为完美视口

- 可以通过meta标签设置完美视口

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

·禁止缩放

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">

·禁止滚动

- event.preventDefault();

·vw单位

- vw 表示的是视口的宽度（viewport width）

- vw这个单位永远相当于视口宽度进行计算

·媒体查询

- 网页可以根据不通的设备或窗口大小呈现出不同的效果

- 使用响应式布局，可以使一个网页适用于所有设备

- 响应布局的关键就是 媒体查询

- 通过媒体查询，可以为不同的设备，或设备不同状态来分别设置样式

### 知道PWA吗

·PWA全称Progressive Web App，即渐进式WEB应用

·一个 PWA 应用首先是一个网页,

·可以通过 Web 技术编写出一个网页应用.

·随后添加上 App Manifest 和 Service Worker 来实现 PWA 的安装和离线等功能

### 知道touch事件吗

·touchstart

- 指放到屏幕上时触发

·Touchmove

- 手指在屏幕上滑动时触发

·Touchend

- 手指离开屏幕时触发

·touchcancel

- 系统取消touch事件的时候触发

# 前端进阶

## 前端工作流

### 前端测试

### 作为一个项目负责人怎么协调多人协作

### 接口文档的制定

### 需求不明确，接口文档是不是越详细越好

### webpack和gulp区别（模块化与流的区别）

#### gulp

·强调的是前端开发的工作流程

- 我们可以通过配置一系列的task，定义task处理的事务

- 例如文件压缩合并、雪碧图、启动server、版本控制等

- 然后定义执行顺序，来让gulp执行这些task

- 从而构建项目的整个前端开发流程

#### webpack

·一个前端模块化方案，更侧重模块打包

- 可以把开发中的所有资源（图片、js文件、css文件等）都看成模块

- 通过loader（加载器）和plugins（插件）对资源进行处理

- 打包成符合生产环境部署的前端资源

## 流行框架

### redux用处

·在组件化的应用中，会有着大量的组件层级关系

·深嵌套的组件与浅层父组件进行数据交互，变得十分繁琐困难

·redux站在一个服务级别的角度

·可以毫无阻碍地将应用的状态传递到每一个层级的组件中

·redux就相当于整个应用的管家

### redux里常用方法

·提供 getState() 方法获取 state

·提供 dispatch(action) 方法更新 state

·通过 subscribe(listener) 注册监听器

### angularJs和react区别

·React对比Angular是思想上的转变

·它也并不是一个库，是一种开发理念

·组件化、分治的管理、数据与view的一体化

·它只有一个中心

- 发出状态，渲染view

·对于虚拟dom它并没有提高渲染页面的性能

·它提供更多的是利用jsx便捷生成dom元素

·利用组件概念进行分治管理页面每个部分(例如 header section footer slider)

### vue双向绑定原理

·vue数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的

·vue2通过 Object.defineProperty() 实现

- 重新定义了对象获取属性值(get)和设置属性值(set)

·vue3通过Proxy来实现

### 说说vue、react、angularjs、jquery的区别

·JQuery是事件驱动

·其他是数据驱动

·JQuery业务逻辑和UI更改该混在一起， UI里面还参杂这交互逻辑，让本来混乱的逻辑更加混乱。

·Angular，vue是双向绑定

·React不是双向绑定

## nodejs

### node的事件方法讲讲看

·emitter.addListener(eventName, listener)

·emitter.emit(eventName[, ...args])

·emitter.on(eventName, listener)

·emitter.removeListener(eventName, listener)

### node的特性，适合处理什么场景

·Node.js借助事件驱动

·非阻塞I/O模型变得轻量和高效

·非常适合运行在分布式设备的数据密集型实时应用

### 讲讲Express

·express 是一个简洁而灵活的 node.js Web应用框架

·提供了一系列强大特性帮助你创建各种 Web 应用和丰富的 HTTP 工具

## React

# 计算机基础

## 计算机网络

### cdn原理

·CDN的全称是Content Delivery Network，即内容分发网络

·CDN的基本原理是广泛采用各种缓存服务器

·将这些缓存服务器分布到用户访问相对集中的地区或网络中

·在用户访问网站时，利用全局负载技术

·将用户的访问指向距离最近的工作正常的缓存服务器上

·由缓存服务器直接响应

### 请求方法head特性

·head只请求页面的首部

·head方法和get方法相同，只不过服务器响应时不会返回消息体

·一个head请求的响应中，http头中包含的元信息应该和一个get请求的响应消息相同

·head可以用来获取请求中隐含的元信息，而不用传输实体本身

·经常用来测试超链接的有效性和可用性

Head请求有以下特点

- 只请求资源的首部

- 检查超链接的有效性

- 检查网页是否被修改

- 用于自动搜索机器人获取网页的标志信息，获取rss种子信息，或者传递安全认证信息等

### OSI七层模型

·应用层

- 文件传输

- 常用协议HTTP、snmp、FTP

·表示层

- 数据格式化、代码转换、数据加密

·会话层

- 建立或解除会话

·传输层

- 提供端对端的接口

- tcp和udp协议

·网络层

- 为数据包选择路由、IP、icmp

·数据链路层

- 传输有地址的帧

·物理层

- 二进制的数据形式在物理媒体上传输数据

### TCP/IP四层模型

·TCP/IP模型是一系列网络协议的总称

·这些协议的目的是使得计算机之间可以进行信息交换

·链路层

- 负责建立电路连接

- 是整个网络的物理基础

- 典型的协议包括以太网，ADSL等，

·网络层

- 负责分配地址和传送二进制数据

- 主要协议是IP协议

·传输层

- 负责传送文本数据

- 主要协议是TCP

·应用层

- 负责传送各种最终形态的数据

- 直接与用户信息打交道的层

- 主要协议是http、ftp等

### 接口特点

·所有的接口都是封闭的，只对公司内部的产品有效

·没有中间用户的授权过程

·有点接口需要用户登录才能访问

·有点接口不需要用户登录就可访问

### Token

·针对接口特点，移动端与服务端的通信就需要2个token

·第一个token是针对接口的（api\_token）

- 保持接口访问的隐蔽性和有效性

- 接口token生成规则参考如下

- api\_token = md5 ('模块名' + '控制器名' + '方法名' + '当天日期' + '加密密钥')

·第二个token是针对用户的（user\_token）

- 它的职责是保护用户的用户名及密码多次提交，以防密码泄露

## 操作系统

### 操作系统进程和线程的区别

·进程

- 是并发执行的程序在执行过程中分配和管理资源的基本单位

- 是一个动态概念，竞争计算机系统资源的基本单位。

·线程

- 是进程的一部分，一个没有线程的进程可以被看作是单线程的

- 线程有时又被称为轻权进程或轻量级进程

- 是 CPU 调度的一个基本单位

### 线程的哪些资源共享，哪些资源独享

#### 共享资源

·堆

- 由于堆是在进程空间中开辟出来的，所以它是理所当然地被共享的

- 因此new出来的都是共享的（16位平台上分全局堆和局部堆，局部堆是独享的）

·全局变量

- 与具体某一函数无关的

- 也与特定线程无关

·静态变量

- 对于局部变量来说，它在代码中是“放”在某一函数中的

- 但是其存放位置和全局变量一样，存于堆中开辟的.bss和.data段

·文件等公用资源

- 使用这些公共资源的线程必须同步

- Win32 提供了几种同步资源的方式

- 包括信号、临界区、事件和互斥体。

#### 独享资源

·栈

- 栈是独享的

·寄存器

- 电脑的寄存器是物理的

- 线程里存放的是副本，包括程序计数器PC

### Linux查询进程指令，查询端口，杀进程

·查询进程

- ps 命令

-- 用于查看当前正在运行的进程。

- grep

-- 搜索

- ps -ef | grep java

-- 表示查看所有进程里CMD是java的进程信息

- ps -aux | grep java

--aux 显示所有状态

·杀死进程

- kill -9[PID]

### 进程间的通信方式有哪些

·无名管道

- 半双工的通信方式

- 数据只能单向流动且只能在具有亲缘关系的进程间使用

·高级管道

- 将另一个程序当作一个新的进程在当前程序进程中启动

- 则这个进程算是当前程序的子进程

·有名管道

- 也是半双工的通信方式

- 但是允许没有亲缘进程之间的通信

·消息队列

- 消息队列是有消息的链表，存放在内核中，并由消息队列标识符标识

- 消息队列克服了信号传递信息少

- 管道只能承载无格式字节流以及缓冲区大小受限的缺点

·信号量

- 信号量是一个计数器，可以用来控制多个进程对共享资源的访问

- 它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源

·信号

- 用于通知接受进程某个事件已经发生

·共享内存

- 共享内存就是映射一段能被其他进程所访问的内存

- 这段共享内存由一个进程创建，但是多个进程可以访问

- 共享内存是最快的IPC 方式，往往与其他通信机制配合使用

·套接字

- 可用于不同机器之间的进程通信

## 数据库

### Redis和 mysql

·类型上

- mysql是关系型数据库，redis是缓存数据库

·作用上

- mysql用于持久化的存储数据到硬盘，功能强大，但是速度较慢

- redis用于存储使用较为频繁的数据到缓存中，读取速度快

·需求上

- mysql和redis因为需求的不同，一般都是配合使用

# 算法与数据结构

## 树

## 递归

## 数组

## 排序

# 设计模式

#### 单例，工厂，发布订阅

·单例模式

- 在它的核心结构中值包含一个被称为单例的特殊类

- 一个类只有一个实例，即一个类只有一个对象实例

·工厂模式

- 在创建对象时不会对客户端暴露创建逻辑

- 并且是通过使用一个共同的接口来指向新创建的对象

·发布订阅模式

- 在软件架构中，发布订阅是一种消息范式

- 消息的发送者（称为发布者）不会将消息直接发送给特定的接收者（称为订阅者）。

- 而是将发布的消息分为不同的类别，无需了解哪些订阅者（如果有的话）可能存在。

- 订阅者可以表达对一个或多个类别的兴趣，只接收感兴趣的消息，无需了解哪些发布者（如果有的话）存在