# HTML

## 行内元素和块元素的区别

## 行内元素：（a,span,labal,img,input,select）

内容多少就占多大的空间

与其他行内元素并排

不能设置宽高，默认宽度就是文字的宽度

Padding，margin只有左右生效

## 块元素：（h系列，ul，li，p，button）

不管内容多少，总是在浏览器中独占一行。

可设置宽高，如果不设置宽高那宽度是100%

行内块元素：共享一行，可设置宽高，多个行内元素排列在一起

# TS

## 为什么

TypeScript 是 JS 静态检查的工具，**在代码运行之前测试代码的正确性**，将问题暴露在IDE

## 类的特性

1. public 说明属性或方法是公有的，在任何地方被访问到，在 ts 中默认的是 public
2. private 说明属性或方法是私有的，不能在类的外部访问
3. protected 说明属性或方法是受保护的，和 private 类似，但是 protected 也可以在子类中访问

## 对比

Interface和class：

interface:接口只声明成员方法，不做实现。class:类声明并实现方法。

Interface和type：

相同点：都可以描述一个对象或者函数，**都允许拓展（extends）**

不同点： type可以声明基本类型别名还可以使用typeof，interface能够声明合并jinterface同一个对象

## 高级特性

Partial 将属性变为可选属性。

Required和Partial方法正好相反，是将属性变成必须。

Readonly是将属性变成只读。

Pick顾名思义，就是把一些属性挑选出来。效果如下：

Record用于创建一个具有同类型属性值的对象。

Exclude从类型 T 中剔除所有可以赋值给 U 的属性，然后构造一个类型。主要用于联合类型。

Extract功能与 Exclude相反

# CSS

## 层叠上下文

## 层叠顺序

background-> z-index<0-> block-> flout ->inline-> z-index=0 ->z-index>0

## 开启方法

position：relative | absolute

display：flex|inline-flex

opacity：不为1

transfrom：不为none

## 文本超出省略号

## text-overflow

text-overflow：ellipsis

verflow：hidden

white-space：nowrap

## 伪类(根据position定位，将内容挡住)

:after{

context:'...'

}

## CSS优化

1.CSS文件合并

2.减少层级嵌套

3.雪碧图

4.减少CSS表达式

5.使用link不使用import

6.使用动画开启GPU加速

## CSS不重名

## 命名规范

## 使用JS控制CSS

## CSS Modul

webpack中css-loader，module开启true，将名字转换为hash值

## REM布局

## meta标签属性

1. width

2. initial-scale

3.maximum-scale

4..minimum-scale

5.user-scalable

## 监听窗口

resize事件监听窗口大小改变

orientationchange事件监听手机是否旋转（orientation=180||0为竖屏 orientation=90||-90为横屏）

## rem

根据设计稿

设计稿的fontsize\*获取到的clientwidth/设计稿初始设备width

## 1px

1.将 `<meta name='viewport'>` 中所有的scale设为1/dpr

2.通过媒体查询改变不同dpr时的transform的缩放

3.border-imge，border-shadow

## 元素的隐藏

可通过`display` `visibility` `opacity` `z-index`解决

`display:none`相当于删除元素，位置消失，事件消失

`visibility:hidden` 位置不变，事件消失

`opacity`位置不变，事件不变

## CSS选择器及其顺序

### 种类

ID选择器（#），类选择器（.），伪类选择器（:），属性选择器（[]），标签选择器，紧邻同胞（+），一般同胞（~），子选择器（>）,后代选择器（ ），通配选择器（\*）

#### 权重

行间样式

ID选择器

类选择器，伪类选择器，属性选择器

标签选择器，伪元素选择器

通配选择器

## 会按照从右向左的顺序进行解析

## position定位

## relative

相对定位，相对于自身定位

## ablsolute

绝对定位，脱离文档流，相对于离自身最近的的relative | fixed | absolute 定位，默认是body

#### width和height百分比相对于最近的position元素，默认是html

## fixed

脱离文档流，相对于视口定位

## stricty

相对于最近元素，必须有top，left，right，bottom之一

## BFC

### 是一种块级作用域，在BFC内部的元素不会影响到外部

### 开启方法

#### body

#### float不为none

#### overflow不为visible

#### position为absolute | fixed

#### display为inline-block | table-cell | flex

### 应用

#### 清除浮动

#### 解决margin重叠问题

#### 自适应布局

## 回流和重绘

回流必将引起重绘，重绘不一定引起回流

### 回流

当元素的大小，位置等属性发生改变，窗口大小发生改变，或者获取clientwidth，offsetwidth这样的属性时，会引起回流

改变的元素及其所有父元素，子元素都会重新渲染

###重绘

当元素的颜色，背景颜色发生改变时，元素会被重新绘制，不会影响其他元素

### 比较

回流比重绘影响性能，浏览器将会引起回流和重绘的操作放到一个队列中，当队列满了，或者设定时间到了，会触发，将多次变为一次

当获取clientwidth等数值时会立即执行，获取最新数据

### 优化

1.避免多层级嵌套

2.避免table布局

3.尽量使用脱离文档流的方法，position：absolute

4.将clientwidth的值赋值给别人变量，避免多次获取

5.先将元素设置为display：none再操作DOM

## CSS动画/JS动画

### transition

时间（duration），延迟（delay），属性（property），变换速度（timing-function）

### animation

动画名（name），时间（duration），延迟（delay），次数（iteration-count），变换速度（timing-function），动画播放方向（direction）

#### 通过keyframes定义动画（from...to）

### 区别

transition必须触发，只能执行一次，只有第一帧和最后一帧，无法复用（只能用一次）

### requestAnimationFrame

根据浏览器的刷新间隔走，不会有掉帧问题

比CSS的多属性（srollTop）

# JS

## Symbol

属于JS的基本数据类型之一

常应用于：

1. 作为对象的Key值，可以保证其唯一性
2. 因为无法被Object.keys，for…in/of获取到，所以在写库的时候如果有不想暴露给别人的属性，可以使用Symbol
3. 可以注册全局Symbol，保证统一，Symbol.for可以查找是否注册过，如没注册就创建

## for...in与for...of区别

## for in得到的是key 可以获取到原型链上的方法

## for of 得到的是value

## 作用域

JS采用静态作用域，根据定义位置向上查找

## 闭包问题

## 常见形式

在函数内部创建一个变量和另一个函数，在另一个函数中使用刚定义的变量

## 什么是闭包

像刚才例子中的形式就叫闭包，在函数内部的函数可以访问函数内部的变量

## 优点

可以是变量存储在内存中，且不被外界所污染

## 缺点

存储在内存中非常消耗资源，容易造成内存泄露

## 应用

节流，防抖，计数器

## 禁止对象修改的方法

## defineProptype禁止修改，重写，删除

writable:false

configurable:false

## Object.preventExtensions()禁止拓展

## Object.seal()

## Object.freeze()

## 对象中引用对象依然可变

## ES6+

1.var let const

2.解构赋值[ ]{ }

3拓展运算符...

4.箭头函数

5.Set

6.Promise

7.async,await

8.iterator(迭代器)

9.ES6 Module

10.函数默认值

11.proxy

## typeof和instanceof区别以及原理

## typeof

在存储变量时，会在变量的机器码1-3存储类型

000：对象

010：浮点数

100：字符串

110：布尔

1： 整数

null的机器码均为0

undefined：用-2^30整数表示

## instanceof

只要右边的prototype在左边的原型链上即可

## Promse

## 三个状态

pending，fulfilled，rejected，一旦从pending态改变就不可再次改变

## Promise

传入一个函数作为参数，该函数有两个参数，resolve，reject

resolve将pending态转变为fulfilled态，并将成功返回的数据传入.then的第一个回调中

reject将pending态转变为rejected态，并将错误传到.then的第二个参数中

## then

有两个参数，第一个参数为成功时的回调，第二个参数为失败的回调，他会返回一个新的Promise，支持链式调用

##catch

类似于then的第二个参数，属于then的语法糖，同样支持链式调用

## all race

### all

传入一个人数组，数组中全是promise的实例，全部成功会执行成功回调，只要有一个失败，就会返回第一个失败的数据

### race

同样传入一个数组，返回第一个成功的数据

## finally

无论成功还是失败都会执行的回调函数

## 终止

1. 返回一个Pending态的promise

2.通过race阻止后续Promise

3.直接抛出异常

## async/await

## Async

返回一个Promise，可以用then方法添加回调函数

## Await

必须用在Async中，等待返回值之后才能执行后续代码

## 与Generator的区别

Async await 相当于Generator的语法糖，Async 等于 \* await 等于 yield

Await 后可以跟基本类型 yield不可以，不需要执行next（）自动执行

## 注意

1.有可能会出现rejected，所以尽量放在try-catch中或者直接.catch

2.希望并发执行，可以使用Promise.all或者for循环执行

## 函数作用域和块级作用域

## 函数作用域

函数体以及函数体内部嵌套的函数体都是有定义的

## 块级作用域

{}内有定义，外部不可见，包括for，if，while

## JS事件机制

## 事件捕获 （IE9+支持）

从document出发，执行指定事件，到目标节点（可通过addeventlistener第三个参数（false）开启）

## 事件冒泡

从目标节点出发，向上传递到document，执行指定事件

### 可以通过stopPropagation阻止冒泡

## class原理

## class方法

1.static：静态方法，只能调用类使用

2.constructor：继承给子类的属性，new时自动调用该方法

3.super()：继承构造函数的this指向

## 原理

通过classCallCheck检测是不是通过new调用，创建函数，将constructor中的属性挂载到函数上，通过createClass方法（根据defineProperties添加方法）将原型和对象上方法挂载到函数上

## extends原理

## 继承父类原型

## 继承父类对象属性

## 创建子类自己的方法

## 再返回该子类

## 继承的过程分为两步，用Object.create继承父类原型方法，用call继承父类的对象方法

## Array和ArrayBuffer的区别

1. ArrayBuffer中只允许存二进制数据

2.ArrayBuffer的长度不能修改

3.ArrayBuffer存储在栈中

## ajax步骤及其参数

## 获取AJAX对象

var xml=New XMLHttpRequest

## 创建HTTP请求

xml.open(method,url,flag,name,password)

1. method请求方法

2.url请求地址

3.是否异步

4.验证身份时的账号密码

## 设置HTTP状态函数

readyState:

0：初始状态，还没有调用send

1：已调用send，正在传输数据

2：send执行完毕，已经接收到服务器数据

3：正在解析内容

4：解析完成，可以在客户端调用

## 获取数据

## 发送请求

xml.send(date)

1. data:name=myName&password=myPassword

## load, ready, DOMContentLoaded白屏事件，首屏时间

## window.onload:  
当页面全部载入完成（包含全部资源）

## document.onload:  
当整个html文档载入的时候就触发了。也就是在body元素载入之前就開始运行了

## DOMContentLoaded:  
当页面的DOM树解析好而且须要等待JS运行完才触发  
DOMContentLoaded事件不直接等待CSS文件、图片的载入完成

## onreadytstatechange:  
当对象状态变更时触发这个事件，一旦document的readyState属性发生变化就会触发

白屏时间：浏览器开始显示内容的时间

首屏时间：浏览器渲染完成的时间

## JS和V8垃圾回收

## 垃圾回收

标记清除

>1.将所有变量进行标记

2.将环境中的变量以及被引用变量的标记清楚

3.在垃圾回收时还有标记的就被清除掉

引用计数

>每被赋值一次+1

被赋值变量被重新赋其他值-1

垃圾回收时数值为零的被清除

当循环引用时可能会内存泄漏

内存泄漏的种类

>1.全局变量不会被回收

2.闭包不会被回收

3.定时器中的不会被回收

4.引用DOM但是DOM被删除，变量不会被回收

V8垃圾回收

>首先分为新生代区和老生代区

新生代区的垃圾回收：

将内存分为from空间和to空间，将from中的对象复制到to中，再将非活跃对象释放，from和to交换

复制过程：先将能从根对象到达的对象复制到to，再遍历to中对象的指针，如果指向from，将对象复制到to

对象晋升：多次没有被清理会被移动到老生代

老生代的垃圾回收： 通过标记清除

## 富文本编辑器

## 让一个 div 成为可编辑状态，加入contenteditable="true" 属性即可。

## DOM 为可编辑区提供了 document.execCommand 方法，该方法允许运行命令来操纵可编辑区域的内容。

## 判断是否空对象

## for in

进入for返回false

## object.key

看返回值是否为0

## JSON.stringify

(JSON.stringify(data) == "{}")

## Ajax

用户与客户端之间加了个中间层，通过XML对象向服务器发送异步请求，获取数据后通过JS操作DOM更新数据，使用户与服务器操作异步化

object.create

创建一个空对象，让空对象的\_\_proto\_\_等于第一个参数，将第二个参数的属性给空对象

## 数组的方法

1.join（\*）（将数据转换成字符串中间填入\*）

2.push，pop，shift，unshift

3.splice

4.slice

5.concat

6.every

7.map

8.foreach

9.filter

10.indexof/indexlastof

## set，weakset，map，weakmap

## Set和weakSet

>Set中成员的值都是唯一的，只有值没有键//add，delete，has，keys，values，foreach

> weakSet中的值必须是对象，且是弱引用（当此对象不再被其他引用时，会被垃圾回收）

## Map和weakMap

>Map以键值对形式存储，键名可以为任意类型

>weakMap只支持对象作为键名，弱引用（当此对象不再被其他引用时，会被垃圾回收）常用于对获取的DOM添加文字说明

## Event loop

1.JS分为主线程和执行栈，执行栈里的任务等待主线程调用

2. 任务分为同步任务和异步任务，同步任务在执行栈中等待执行，异步任务当有结果之后，将回调函数，放到任务队列中等待执行栈为空，执行

3.当同步任务执行完毕，执行栈为空时，检查微任务队列是否有微任务，一次执行完所有，否则检查宏任务队列，执行完一个宏任务之后，再次检查微任务队列

## node的event loop不同

每执行完一种宏任务，再去执行微任务（timer，pull，check）

JS异步操作

回调函数

事件监听

定时器

Promise

## 内存泄漏的原因

## 元素被移除，元素上的事件没有被移除

可以手动清除事件或事件委托

## 闭包

手动清除

## 定时器没有被清除

## 意外的全局变量

## 判断数组的方式

#### arr instanceof Array

#### arr.\_\_proto\_\_===Array.prototype

#### arr.\_\_proto\_\_.instructor===Array

#### Object.prototype.toString.call(arr)

#### Array.isArray(arr)

## typeof的用法详解

### 对于数字类型，返回number（NaN）

### 对于字符串类型，返回string

### 对于对象，数组，Null返回Object

### 对于布尔值，返回Boolean

### 对于函数，返回function

### 对于未定义，返回undefined（undefined）

## var let const区别

### var

有变量提升

###let，const

没有变量提升，会造成暂时性死区，const定义时必须赋值，赋值为常量是不可修改，为对象时可以修改

## JavaScript 是如何影响 DOM 生成的

渲染引擎内部有一个html解析器，它可以将字节流转换为dom结构，网络进程加载多少数据就解析多少数据（根据context-type判断文件类型），将碰到内嵌脚本时，解析器会暂停解析，JS引擎介入，执行脚本，之后再继续解析，遇到引入文件时，会先下载脚本，下载会阻碍解析

优化：chrome有预解析优化，使用CDN加载脚本，压缩JS要脚本大小，没有操作DOM的JS可以采用异步加载

## JS数字

JS采用64位双精度储存数字，看着有限的数字实际转换为是无限的，所以会京都缺失

## 异步有哪些实现方式？

## 回调函数(B函数被作为参数传递到A函数里，在A函数执行完后再执行B )

## 事件监听

## 发布订阅(根据调度中心引发事件执行)

## Promise

## 如何创建对象

## new Object

## 工厂模式（解决了创建多个相似对象,无法知道对象的类型）

## 构造函数（每个方法都会在每个实例创建一遍）

## 原型模式（属性共享，修改一个另一个也会改变）

## 构造加原型

## 如何下载文件

1. 利用a标签的download属性，使用blob进行解析，在使用URL.createObjectURL关联一个url，设置a.href为url，download为文件名，aclick（），再清除dom节点和url即可
2. window.open()方法，后端提供下载接口：

## offsetwidth

## MutationObserver

监听DOM的变化，在变化时异步触发回调，可以返回DOM的变化的类型和属性等

通过 new MutationObserver(callback) 进行注册返回一个实例，实例方法有observer(开启监听)、disconnect（停止监听）、takeRecords（清空数据）

## IntersectionObserver

观察目标元素与视口或指定根元素产生的交叉区的变化，在变化时触发回调，可以返回DOM在屏幕中的占比和宽高数据

通过 new IntersectionObserver(callback, option)进行注册返回一个实例，实例方法有observer(开启监听)、unobserver（停止监听）、disconnect（关闭观察器）

# 网络

## TCP和UDP的区别

## UDP

1. 不需要建立连接，知道ip和端口就可以直接连接

2.没有确认机制，没有重传机制，无法保证数据的完整性

3.传输时不会拆分，按原样发送

4.不会粘包

5.支持一对多

## TCP

1. 需要握手建立连接，需要挥手断开连接

2.通过ACK可以确认数据是否接受和完整

3.会造成粘包

4.支持一对一

### TCP如何保证数据完整

1. 三次握手四次挥手
2. 超时重传，再传数据时会，如果长时间没有收到ACK报文，就会重新传数据
3. 流量控制，如果发送方发送的过快，接收方处理不了，数据堆在缓存区，会造成丢包，根据接收方的处理数据的速度发送数据
4. 拥塞控制，有可能一开始网络就很堵，对导致大量数据在网络中，会产生大量的丢包和超时重传，所以采用了1. 慢启动机制，在最开始发送时，会发送少量数据探路，确保网络的状态，2.拥塞窗口，在发送之前将接收方提供的窗口和本身的窗口对比，取较小的窗口，窗口随增长，发生拥堵会/2

### 粘包以及解决办法

第一个数据的尾粘到了第二个数据的头，接收方无法明确指令

1. TCP会将多次数据整合到一个包发送

2.接收方将数据放到缓冲区，在进程从缓冲区取数据之前，后一包也没接收放到缓冲区

#### 1.强制将数据发送，不整合成一个包

#### 2.提高接受进程的优先级

## TCP三次握手和四次挥手

## 基本标识

ACK应答有效

FIN请求关闭

SYN建立连接

ack希望收到的位置

seq发送的位置

## 握手过程

客户端向服务器发送SYN段+seq =

服务器向客户端发送SYN段+seq+ack

客户端向服务器发送ACK段+ack

## 挥手过程

客户端向服务器发送FIN段+seq+ack

服务器向客户端发送ACK段+seq+ack

服务器继续传输数据

服务器向客户端发送FIN段+seq+ack

客户端向服务器发送ACK段+seq

## 强缓存和协商缓存

### 强缓存

在加载资源时，会先查看是否命中强缓存，命中的依据是expire（绝对时间）和cache-control（相对时间），指的是缓存过期的时间，如果没有过期，直接用强缓存，否则发送请求

### 协商缓存

客户端会将if-modified-since和Etag发送给服务器，如果Etag的值等于if-no-match的值或者if-modified-since的值等于Last-modified（Etag的优先级高于if-modified-since）说明命中协商缓存，返回304状态码，使用本地缓存，否则返回新数据和200状态码

## cache-control字段

no-store禁止缓存

no-cache代理服务器不可使用缓存，需要请求

public任何情况都可以缓存

prvate指定用户可以缓存

max-age缓存存活时间

## 跨域问题

## 原因

浏览器有同源策略，既协议，域名，端口号完全相同才可以请求数据，link，img，script除外

## jsonp

利用script没有同源策略的漏洞，在本地创建方法，通过问号传参的方式，向服务器发送数据和方法，在本地执行方法

## cors

在后端开启即可，简单请求和复杂请求

简单请求：get，post，head和text，multipart，application

## nginx

服务器之间没有同源策略，先向nginx请求，nginx再向服务器请求

## cors头部字段

Access-Control-Allow-Origin指定了允许访问该资源的外域 URI

Access-Control-Allow-Methods指明了实际请求所允许使用的 HTTP 方法

Access-Control-Allow-Headers其指明了实际请求中允许携带的首部字段

Access-Control-Expose-Headers果要访问其他头，则需要服务器设置本响应头

Access-Control-Max-Age结果能够被缓存多久

Origin

origin 参数的值为源站 URI。它不包含任何路径信息，只是服务器名称。

Access-Control-Request-Method

用于预检请求。其作用是，将实际请求所使用的 HTTP 方法告诉服务器。

Access-Control-Request-Headers

用于预检请求。其作用是，将实际请求所携带的首部字段告诉服务器。

## 正向代理与反向代理

## 正向 将请求发给代理服务器，代理服务器收到返回的数据，再返回给我们

## 反向 代理服务器接受请求，再将请求转发给服务器，再将数据返回给我们

正向就是客户端代理，反向就是服务器代理

## Referrer

防盗链，代表着链接的来源

在地址框直接输入会是空referer

## 网络分层

## OSI七层

物理层-数据链路层-网络层-传输层-会话层-表示层-应用层

## TCP四层

数据链路层-网络层-传输层-应用层

>应用层：http，ftp，DNS，telnet，websocket

传输层：TCP，UDP

网络层：IP

## HTTP请求方式及其区别

## 请求方式

1. get

2. post

3.delete

4.put

5.head

6.options

7.trace

8.connect

## OPTIONS

在使用CORS跨域时，采用非简单请求之前先用OPTIONS查看服务器是否支持方法

## GET和POST

GET

>用于获取数据，数据在URL处可见，有长度限制，相对于POST不够安全，只支持ASCII字符，可缓存（数据在URL中），幂等

POST

>用于发送数据，数据在URL处不可见，没有长度限制，相对于PUT安全，支持任意类型，不可缓存（数据在请求体中），不幂等

## POST和PUT

POST

>无法指定具体路径，只能指定到集合路径，不幂等，多次操作会产生多次变化

PUT

> 可以指定具体路径，幂等，多次请求与一次请求结果相同

## 状态码

### 1XX

100客户端应继续发送请求

101需要切换协议

### 2XX

200成功

201请求成功，建立新的资源

202服务器接受请求，但未处理

### 3XX

301永久重定向

302临时重定向

303用GET请求另一个URL

304协商缓存可用

### 4XX

400代码错误

401需要用户验证

403权限不够，组织访问

404资源丢失

405请求方法不对

413数据大小超出限制

### 5XX

500代码错误

503服务器超负荷

504使用代理时，响应超时

## HTTP2/3

## HTTP2

### 原因

随着互联网的发展，打开页面需要下载的资源越来越多，HTTP1.1的缺点就暴露出来

1. 队头堵塞，效率低

2. 采用明文传输不够安全

3.服务器无法自主推送消息

4.报文头有时很大，报文体却很小

### 机制

1. 二进制传输

将数据拆分成采取二进制编码的帧，可以乱序传输，根据标识进行组装

2. 多路复用

采取一条通道传输，可以双向传输数据

3. 压缩

采用算法在服务器和客户端建立字典，对于相同的元素，不需要多次请求和响应

4.服务器可以自主向客户端发送数据

5.HTTP2支持HTTPS，更加安全

### 解析多路复用

HTTP1 每次请求都需要连接->传数据->断开->连接

HTTP1.1 添加了keep Alive 每次连接不需要断开，下次可以继续使用该通道，但由于数据传输采用文本分割得形式，是串行的，只节约了连接的事件，并没有节约传数据的时间

HTTP2 采用帧传输，根据顺序标识，可以组装，是并行的。不管多少请求都采用一条通道，服务器的并发也会提升

### 缺点

1. 当发生丢包时，整个数据需要等待重传

2.tcp连接时需要先和TSL层建立联系，握手会有延时

## HTTP3

将TCP改为UDP，使用QUIC协议，将帧传输改为流传输，使用流ID代替IP，可以保证切换网络时继续使用连接

## HTTPS

## 为什么使用HTTPS

1. HTTP是明文传输，不够安全

2.无法发验证对方的身份是否合法

3.无法验证数据的完整性

## 解决办法

1. 在HTTP上建立一个SSL层，通过向SSL层通信，再由SSL曾向TCP通信

2. 通过对称加密和非对称加密的方法

3.使用数字签名验证数据的完整性

4.使用数字证书验证身份

## 具体过程

1. 服务器向客户端发送一个公钥证书，证书上有着公钥和一些信息，可以验证身份

2.客户端速记生成一个对称密钥由公钥加密，发送给服务器

3.服务器通过私钥解密，得到对称密钥，将数据通过hash简报到一个简报，将数据和简报进行加密，发送给客户端

4.客户端通过对称密钥解密，得到数据和简报，将数据通过hash得到另一个简报，与得到简报对比，确认数据完整性

## cookie localstorage session storage的区别与用法

`cookie`

1. 服务器给客户端发送`setCookie`通知客户端生成`cookie`，每次请求会发送给服务器

2. `cookie`有时间限制，时间到会自动失效

3. `cookie`最大4KB

4. `cookie`没有自己的方法，需要封装

`local storage`

1. 不会发送给服务器

2. 除非主动删除，否者不会失效

3. 有自己的方法`getItem` `setItem`

4. 最大5M

`session storage`

1. 不会发送给服务器

2. 最大5M

3. 仅本次会话有效，关闭网页自动失效

4. 有自己的方法`getItem` `setItem`

注：`local storage`和`session storage`在无痕模式下默认失效

## Cookie于Session的创建流程

当客户端访问服务器时，服务器会先查看是否有cookie（JSESSIONID）字段，如果没有生成一个Session对象，并生成一个id赋给session，之后服务器发回客户端一个setcookie字段，之后浏览器会将setcookie中的内容记录在cookie中

当客户端再次请求时，会先查看是否有cookie，会将cookie放到请求头中，服务器会根据cookie字段中的sessionID找到相应session

当session销毁时，会生成一个新的sessionid发回客户端，cookie将sessionID更新，下次请求时服务器根据ID找不到session，会建立一个新的session

## Token

服务器在验证用户信息后，会通过算法和密钥进行加密，将数据和密钥一起作为token返回给客户端，客户端存在cookie（跨域）或local storage中，下次发送请求时在请求头 Authorization 字段加上token，客户端根据对数据再次加密对比token是否相同，相同则验证通过

* 用户使用用户名密码来请求服务器
* 服务器进行验证用户的信息
* 服务器通过验证发送给用户一个token
* 客户端存储token，并在每次请求时附送上这个token值
* 服务端验证token值，并返回数据

## XSS和CSRF

## XSS

是一种注入攻击，将恶意代码与正常代码混在一起，被浏览器一起执行

反射型

>通过构造恶意链接，点击时向网站注入脚本

DOM型

>通过在输入框输入恶意脚本，发生在客户端的攻击

存储型

>将恶意代码上传到服务器，每次请求数据时都返回脚本

#### 防范：

1.设置CSP白名单禁止外域加载

2.设置HttpOnly阻止Cookie

3.输入和输出检测，对内容进行转义，过滤

## CSRF

过程

>用户通过A网站的认证生成cookie，在未关闭A时打开B，B以用户的身份通过A的cookie放松请求

#### 防范：

1.使用referer字段检查请求来源

2.使用token

3.通过验证码

## 输入URL到渲染经历了什么

1.用户输入URL ,浏览器会进行判断，是搜索内容还是URL

> 如果是搜索内容，默认搜索引擎+搜索内容合成新的URL

2.将URL发送给网络进程，查看是否命中强缓存

> 命中，拦截请求，返回200；未命中，发送请求

3.进行DNS解析

>查看是否有DNS缓存，没有的话进行一级一级的查找

端口号：http默认80，https默认443

如果是HTTPS还需要建立SSL/TSL连接

4.TCP三次握手建立连接

5.判断状态码

6.根据content-type解析数据包，准备渲染

7.对文档进行解析

>HTML解析成DOM Tree

CSS解析成CSSOM Tree

两者合并成Render Tree

计算得出元素大小，位置，样式，开始渲染

## DNS解析过程

#### 查看浏览器缓存

#### 查看操作系统本地缓存

#### 查看路由器缓存

#### 向本地DNS服务器

#### 本地DNS服务器向根DNS服务器发送请求

返回顶级域服务器的IP地址

#### 本地服务器向顶级域服务器发送请求

要么返回查询IP，要么返回二级域DNS服务器IP地址

#### 以此类推，直到返回查询IP

# React

## hashhistory和browserhistory区别

## hashhistory

有个＃，当/#/变化时并不会发送请求，会自己根据路径render

## browserhistory

当/发生变化时会发送请求，所以需要后端支持

## Fiber

旧版react用的是JS调用栈，大量的同步任务会造成阻塞，Fiber实现了自己的调用栈，在虚拟DOM Tree生成之后，Fiber会以链表的形式遍历，生成Fiber树，每生成一个节点会将控制权交给浏览器查看是否有优先级更高的事件，如果有用先完成，之后再批量更新节点

## redux和react-redux

# redux

## store

用来储存state，可以通过createstore创建

## action

是一个对象，第一个属性为type，组件通过dispatch将当前state和action发出，store会自动调用reducer

## reducer

是一个纯函数，通过判断action的type属性进行操作，返回一个全新的state

之后store会调用监听函数，监听者会通过getstate获取当前状态并重新渲染

## 中间件（applyMiddleware）

### redux-thunk使dispatch可以接受函数作为参数

### redunx-promise使dispatch可以接受promise作为参数

# react-redux

## connect

通过connect可以将mapStateToProps和mapDispatchToProps传入组件

## provider

在根组件包裹，可以让所有子组件都拿到state

## Dva数据流向

组件dispatch一个action，如果是同步任务，就通过reducer改变state，如果是异步任务，就先通过effect，再通过reducer改变state

## react生命周期

## 初始化

1. constructor

2.getDefaultProps

3.getInitialState

## 挂载阶段

1. getDerivedStateFromProps

2.render

3.componmentDidMount

## 更新阶段

1. getDerviedStateFromProps

2.componmentShouldUpdata

3.render

4,getSnpshotBeforeUpdata

5.componmentDidMount

## 卸载阶段

1. componentWillUnmount

## 错误

1. componentDidCatch

## HOOKS相关

## 为什么使用

1. 解决了this的问题

2.更好的复用性

3.编译之后更小

## 为什么必须在函数组件顶部作用域调用Hooks API？

> 当用了Hooks API之后会创建一个Hooks实例，并存入Hooks链中，返回给组件一个state，setter，重新渲染时，组件并不知道setter对应哪个state，仅仅代表顺序，（所以需要按顺序在最顶级）

## useState如何更新数据

>内部有一个dispatchAction函数，他会生成一个update挂载到Hooks队列(回一次执行所有update)，理论上可以多次dispatch但只有最后一次生效，直接覆盖数据（setState会对比）

## 为什么要用setState

setState采用的是合并更新，将需要更新的State放到队列中，如果用直接赋值的话，不会放到队列中，下次setState获取不到之前修改的State

## setState是同步还是异步

在React控制时是异步，否则是同步

### 有一个变量控制是否异步（默认值为false，不异步），React调用之前会将值改为true

## v-dom和diff算法

## v-dom

React会将JS代码转变为对象，再去渲染真实DOM，这个对象就是v-dom，当代码发生改变时，会根据diff算法，对比新老v-dom，改变旧的v-dom，再去改变真实DOM

### 优点：减少内存消耗，更新更快

## diff算法

### 三个策源

1. 减少跨层级操作

2.同一类型的组件有相似的结构

3.同一层级可以根据唯一标识区分

### 具体过程

1. 对树的每一层级开始遍历，如果组件不存在直接销毁

2. 判断是否为同一类型的组件，是->根据 compontentshouldupdata判断是否需要更新，否->直接替换

3.根据唯一标识对节点进行区分，insert，move，remove

## react的组件传参

## 父传子

通过在子组件标签中传值，子组件用props获取

## 子传父

在父组件定义函数，传到子组件，子组件通过触发函数往父组件传值

## 兄弟间

在更高级别组件中定义方法，传给兄弟，通过触发传值

## context

在数据起始位置使用provider包裹，在后续需要使用数据的外面包裹consumer

## redux

## react路由传参的方法

### ：传参

可以通过route path=url：XXX传递

this.props.match.params.name查询

### query传参

再link标签中通过query属性传递对象

## vue和react的区别

## react是函数式思想，状态和逻辑通过参数传入，所以react是单向数据流，在setstate之后会重新渲染一便（shouldComponentUpdate返回true就继续渲染）

Vue数据可变，对每个属性建立监听，属性改变时更新对应的虚拟DOM

##react通过js生成html，所以设计了jsx，Vue将这些组合到了一起

## MVC和MVVM

##MVC

由三部分Model、View、Controller，用户对View（html）的操作交给了Controller（js）处理，在Controller中响应View的事件调用Model（请求到的数据）的接口对数据进行操作

##MVVM

它实现了View和Model的自动同步，也就是当Model的属性改变时，不用再自己手动操作Dom元素，来改变View的显示，而是改变属性后该属性对应View层显示会自动改变。

# Vue

## nextTick原理

Vue在数据更新后不会立即改变视图，而是将更新汇总一起执行，如有重复的更改还会替换，提高效率，所以在数据发生改变后想要获取最新的Dom属性需要用到nextTick。

一句话总结：

将回调函数变成异步任务，在Dom渲染后再执行回调函数 。

原理：

会维护一个callbacks数组，每次调用nextTick都会向数组内推入回调函数。之后再执行一个降级函数，promise ->MutationObserver -> setImmediate -> setTimeout 将遍历执行callbacks数组（通过slice复制一个新数组）内的方法包装成微任务，等调用栈执行完同步任务后，执行。

# 优化

## SEO优化

1.优化网站的分级结构，结构不要嵌套太深（面包屑导航）

2.使用语义化标签

3.mate标签定义一些关键词，描述

4.img添加alt属性

5.a标签添加ref=‘nofollow’属性（无需跟踪）

6.a标签使用title属性

7.内链采用绝对路径

8.将重要的内容放到最前面

9.避免使用iframe

10.重要内容避免JS输出

11.较少使用display：none

12.SSR

## 前端优化

## 减少请求次数

1.雪碧图，字体图标和base64

2.减少重定向

3.使用缓存

## 优化网络连接

1.使用CDN（通过综合考量将用户的请求转移到最近的服务器节点，解决拥挤）

2.DNS预解析（将以后可能会用到的DNS提前解析好，放到系统缓存中）

3.长连接（keep-alive）

## 模块资源加载

1.异步执行script

2.按需加载

3.懒加载和预加载

## 减少回流和重绘

1.避免table布局

2.避免层级嵌套太深

3.事件代理

4.节流防抖

5.及时清理环境（闭包，定时器，事件监听），防止内存泄漏

## 长列表优化

## 完整列表

尽量使用innerHTML

## 非完整列表

1. 懒加载

2.只在显示区域渲染

# 其他

## 轮询方式

## 短轮询

服务器收到请求，不管是否有数据，都立即响应请求，之后浏览器隔一段时间内在发送同样的请求

## 长轮询

服务器收到请求后，如果有数据，立即返回，如果没有，等待数据或到了时间限制，响应请求，浏览器收到响应后立即再次发送请求

## common.js、amd、cmd和ES6 module

在ES6之前，模块化一直使用的是common.js和amd，前者用于服务器，后者由于浏览器，modelu出现后服务器和浏览器都可以使用modelu

## common.js

modelu export导出，require导入

引入值得拷贝，使用引入的方法并不会改变引入值

## amd和cmd

define定义模块，require加载模块

amd是引入直接加载，cmd是使用才加载

## module

export导出 import导入

引入值的引用，使用引入的方法会改变引入值

## 栈和堆

# 程序内存中的栈和堆

## 栈

是系统自己分配的，顺序是从上到下，栈中存储的数据的 生命周期随着的函数结束而销毁

## 堆

是程序员分配的，若程序员不释放，顺序是从下往上，程序结束时由操作系统回收

## 不同

创建方式不同，生命周期不同，占用空间大小不同，生长方向不同，存放内容不同

# 操作系统中的栈和堆

## 栈

一种先进后出的数据结构，只允许从一端插入或者弹出

## 堆

是一种特殊的完全二叉树，二叉树满足根节点的是最大或最小时称为堆（最大称大根堆，最小称小根堆）

## Webpack

## 什么是webpack

一个打包工具，将浏览器不能直接运行的代码打包为可以运行的文件

## 过程

1. 先将配置文件和shell中的内容进行合并为参数

2.再将参数初始化为一个compiler对象，加载完所有插件，执行run方法开始编译

3.从entry出发，调用loader编译，遍历所有依赖编译

4.根据模块间的依赖关系组装成一个个chunk，再把每个chunk加到输出列表

5.根据配置的路径和文件名输出

### 插件会在相应的时间后执行相应的逻辑

## 热模块

服务器通过websocket将编译打包的信息传给客户端，客户端根据文件hash值进行替换

## 路由懒加载

如果没有运用懒加载，打包后文件会很大，进入首页时，需要加载的内容太多，路由懒加载可以按需加载页面 () => import('./testComponent')

可配合react路由懒加载中间件Loadable（loadable(() => import('../components/Menu/Menu'))

）

## 提取公共代码

CSS： miniCssExtractPlugin

JS：optimization中的splitChunks自己配置

框架代码： DLLPlugin

## webpack打包后文件还很大怎么办

1. 抽离出公共代码 （MiniCssExtractPlugin,optimization中配置splitChunk，DLLPlugin）

2.Tree Shaking(除去JS中用不上的代码)

3.UglifyJS压缩代码

4.路由懒加载

5.作用域提升（Scope Hoisting）

## loader

模块转换器，将文件转换为webpack认识的文件，在打包之前执行

## plugin

插件，在打包过程中，实现某些功能

## 常见loader

1. style-loader

2.css-loader

3.less-loader

4.url-loader

5.file-loader

6.babel-loader

7.eslint-loader

## 常见plugin

1. html-webpack-plugin 自动生成html

2.MiniCssExtractPlugin 抽离css代码

3.HappyPack（开启多线程）

4.DLLPlugin（抽离组件代码）

## 如何统计页面停留时长

## 监听进出

onload/onbeforeunload

## 监听路由前进后退

pageshow/pagehide

## 监听路由改变

1. browser

history.pushState/history.replaceState

2.hash

hashChange

## 监听页面状态

visibilitychange

## 上报时机

1. 离开时上报，刷新或关闭可能数据会丢失

2. 下次打开时上报，丢失最后一个页面数据（离开时将数据存入localstorage中，下次打开时上报）

## 进程与线程

## 进程

进程是操作系统的一次执行过程，是一个动态概念，是程序执行过程中分配和资源管理的基本单位，有五个状态，初始态，执行态，等待态，就绪态，终止态

## 线程

线程是CPU调度的基本单位，它与同属进程的线程共享资源，线程之间通过‘互斥锁’，‘信号量’联系

## 联系

线程是进程的一部分，一个线程只能属于一个进程，一个进程可以有多个线程

## 区别

1. 花销区别：进程之间的转换花销更大

2.位置区别：在操作系统中有多个进程，一个进程中有多个线程

3.内存分配区别：操作系统给线程分配内存，同属进程共享资源

## 数组，链表，栈，队列

## 数组

> 数组存储是连续的，所以读取数据比较方便，但是不利于插入和删除

适用于经常访问，但是不经常插入删除

## 链表

>链表有两块区域，一块区域存储数据，另一块区域用来存储下一个数据的位置，有利于插入和删除，改变指针指向即可

适用于元素个数不确定，而且经常插入和删除

## 栈

>先入后出

适用于关心最近一次操作

## 队列

>先入先出

适用于按顺序处理数据时

## 白屏问题

## 白屏时间

Date.now()-performance.timing.navigationsSart

## 异步IO和同步IO

数据传输路线：A->A缓冲区->B缓冲区->B

# 同步IO

## 阻塞IO

B向B缓冲区要数据，在B缓冲区没准备好时，B一直等待

## 非阻塞IO

B向B缓冲区要数据，B缓冲区没准备好立即告诉B，B不需要等待（反复请求）

## 复用IO

通过Select函数监控数据是否请求完成（轮询），完成返回可读状态，线程再去读取数据

## 信号驱动IO

建立一个练习，当数据准备好之后，会返回可读状态，线程再去读取数据

# 异步IO

向内核发送read请求，内核会自动将数据复制到用户空间

## 设计模式

## 工厂模式

主要用于创建同一类对象（权限管理）

## 单例模式

用一个变量标识确定是否创建过对象，如创建过，直接返回创建的对象（一些弹窗，多次点击只会被创建一次）

## 中介者模式

通过一个中介对象，可以让其他对象相互通信（购买商品时，有颜色，有数量都会改变，可以通过中介者来转发处理这些事件）

## 策略模式

在策略类（可变）中封装了一个个函数，在环境类（不可变）中可以执行这些函数

## 观察者模式

B把自己注册到A中，A的数据改变，让B更新（微信消息）

## 发布订阅模式

A改变，将数据发到调度中心，调度中心让B更新（redux）

## 顺序表和链表

## 顺序表

一般表现为数组，可以通过index访问

### 适用于频繁查询的情况

## 链表

表中每个节点，都保存下个节点的指针

### 适用于频繁增删的情况

## 浏览器的运行机制

## 浏览器是多进程的

1.GPU进程(用于3D绘制)

2.Browser进程（用于页面管理）

3.第三方插件进程

4.渲染进程（页面渲染，脚本执行）

## 渲染进程是多线程的

1.GUI渲染引擎线程（用于解析HTML，CSS，回流重绘）

2.JS引擎线程（用于解析JS，与渲染线程互斥）

3.定时触发器线程（将定时器中的回调放到事件队列中，JS是单线程的，处于堵塞状态会有误差）

4.异步HTTP请求线程（检测到XHR变化，将请求放到任务队列中）

## proxy（代理）

拦截你的操作，代替你执行新定义的操作

## SSR（服务器渲染）

## 什么是SSR，与客户端渲染的区别

客户端渲染是先接收一个空的HTML界面，之后通过下载解析JS进行渲染

服务器渲染是先通过服务器生成HTML，再发送给客户端

## 优点

有利于SEO,搜索引擎不会爬虫脚本（谷歌可以），而且不会等待加载完毕再爬取

有利于首屏加载，直接返回HTML，渲染速度加快

## 过程

SSR

客户端->node->后端->node->客户端

客户端渲染

客户端->node->页面->node->后端->node->页面

## 如何在浏览器不同的窗口进行通信

# 同源情况下

## cookie

通过获取cookie的值来进行通信

## localstorage

通过获取localstorage的值来进行通信

## websocket

http://39.106.156.135:3000/http/detailed?id=12323

## postmessage

可以通过window.open获取到A页面的window对象，之后通过该对象的postMessage方法发送数据

# 非同源情况下

## websocket

同上

## postmessage

同上

Websocket

websocket可以让服务器主动向客户端发送消息，也可以让客户端主动向服务器发送消息，告别了消耗大，效率低的轮询

## 元素是否在可视窗口内

通过getboundingclientrect获取属性

top：元素上边距距离视口顶的高度

bottom：元素下边距距离视口顶的高度

left：元素左边距距离视口左的宽度

top：元素右边距距离视口左的宽度

>rect.top < window.innerHeight && rect.bottom > 0

rect.left < window.innerWidth && rect.right > 0

## Webworker

### 解决了JS是单线程的问题，使浏览器可以是为其提供多线程

通过new Worker建立 ，onmessage监听，postmessage发布，terminate、close关闭

#### 大量数据请求，图像处理时使用

## 更新机制

小程序冷启动时如果发现有新版本，将会异步下载新版本的代码包，并同时用客户端本地的包进行启动，即新版本的小程序需要等下一次冷启动才会应用上。 如果需要马上应用最新版本，可以使用 [wx.getUpdateManager](https://link.zhihu.com/?target=https%3A//developers.weixin.qq.com/miniprogram/dev/api/wx.getUpdateManager.html" \t "_blank) API 进行处理。

## 如何监控网页崩溃？

p1：网页加载后，通过 postMessage API 每 5s 给 sw 发送一个心跳，表示自己的在线，sw 将在线的网页登记下来，更新登记时间；

p2：网页在 beforeunload 时，通过 postMessage API 告知自己已经正常关闭，sw 将登记的网页清除；

p3：如果网页在运行的过程中 crash 了，sw 中的 running 状态将不会被清除，更新时间停留在奔溃前的最后一次心跳；

sw：Service Worker 每 10s 查看一遍登记中的网页，发现登记时间已经超出了一定时间（比如 15s）即可判定该网页 crash 了。

## V8引擎是如何执行JS代码的

①AST(Abstract Syntax Tree):抽象语法树，一种数据结构

②HotSpot：热点代码，多次出现的代码

③Bytecode:字节码

④Machine code:机器码

⑤Ingition：解释器，将AST转换为Bytecode

⑥TurBoFan：编译器，将Bytecode转换为Machine code

⑦JIT(ust-in-time compilation):即时编译,运行的时候编译执行

## AST的解释以及生成过程：

### AST是很重要的一种数据结构，Babel和Eslint的工作原理就是先将源代码转换为AST，然后再进行其他操作

### AST的生成过程：先分词，再解析：先将源代码拆分为一个个token[{}]（不可再分的字符或者字符串），然后将token解析为AST

## js引擎理解:

### 是用来处理js代码的一段程序或者解释器，对原代码进行词法分析和语法分析后生成AST,基于AST再生成字节码，从而执行代码。

### 执行过程：

#### 语法分析阶段：对加载完成的代码进行语法检验，检验完成后进入预编译阶段；

####　预编译阶段：搜集函数名以及变量提升，确定this指向和作用域链

####　执行阶段：通过事件循环执行

## v8引擎：

### v8引擎是js引擎的一种，其特点是提高js执行性能

### v8执行代码过程：

####　将源代码生成AST：源代码进行词法分析和语法分析后生成AST

####　生成字节码：根据AST生成字节码，字节码通过Ingition转换为机器码才能执行

####　执行代码：如果一段字节码是第一次执行，解释器会逐行执行；如果在执行字节码过程中发现有HotSpot，JIT即时编译器就会把HotSpot编译为机器码，并保存起来，下次就可以直接使用了，省去了将字节码转换为机器码的过程，提升了执行效率；

## git revert 和 git reset 的区别

## revert 回滚提交，回到之前提交的版本，会重新commit一次

## reset 撤销提交，撤销之前的提交，之后的提交会放到暂存区

## git merge 与 git rebase的区别

## merge 会将其他分支的commit在自己分支后重新commit

优点：记录每次commit的情况

## rebase 会将整个分支移动到其他分支后面

优点：项目更加简洁

## 前端监控与埋点

## 监控分为三种：

数据监控（访问人数，用户行为），性能监控（白屏时间，加载完成时间），异常监控（一般try，catch就可以捕捉，也可以sentry进行监控）

## 埋点方式有三种：

1. 手动埋点，手写代码或调用第三方SDK（百度站长统计）上传给服务器，自由度高，工程量可能会大一些
2. 可视化埋点：将业务代码和埋点代码分开，通过可视化界面设置流程，进行埋点，效率高
3. 无埋点：前端收集所有事件，让后端自己进行过滤

## 上传的方式（需要上传事件的类型）

1. Img标签不需要跨域，可以通过imge.src属性将数据传给后端（[www.qq.com?type=login](http://www.qq.com?type=login)）
2. 通过请求上传，beacon会发送少量数据，当beacon进入消息队列中会返回true，没进入返回false，再之后会尽力发送，但不保证能否发送成功

# Axios

入口文件有一个函数来创建axios实例，这个实例有axios一些方法，再返回出去

Axios最主要的是request方法，无论是get还是post都调用这个方法发送请求，request方法中有一个数组，数组里有一个[dispatchRequest]，利用foreach，如果是request就unshift fulfilled和rejected（返回config），如果是response就push fulfilled和rejected，相当于在请求之前和在响应之后用户可以做一些处理，之后只要数组length不等于0就一直while，每次将数组shift两次，等于在request的时候会将fulfilled返回的config，通过dispatchRequest请求，在response的时候会将dispatchRequest返回的数据传给fulfilled，dispatchRequest底层实际上就是对数据进行处理，没配置的话的就通过XHR请求，最后再返回结果

# 小程序

## 双线程

小程序采用双线程即渲染层和逻辑层是分开的，交予两个线程进行管理，视图层使用webView渲染，逻辑层采用jscore运行脚本，这样就会导致无法直接操作DOM，两个线程会通过微信客户端进行中转，网络请求也由Native转发，DOM的更新有点像V-Dom，用对象模拟DOM树，对比新老树的差异，渲染差异部分

## 运行机制

小程序启动会有两种情况，一种是「冷启动」，一种是「热启动」。假如用户已经打开过某小程序，然后在一定时间内再次打开该小程序，此时无需重新启动，只需将后台状态的小程序切换到前台，这个过程就是热启动；冷启动指的是用户首次打开或小程序被微信主动销毁后再次打开的情况，此时小程序需要重新加载启动。

# 新技术

## Serverless

## 前端的几个阶段

1. 通过jsp写页面

2. Ajax之后前后端分离

3.node.js之后工程化开发，全栈

4.下一个开发模式有可能是serverless

## serverless

是Faas（函数平台）和Baas（云服务平台）的结合，相当于在Faas中使用Baas的函数

## 优势

1. 无状态，执行函数即可

2.无运维

3.低成本，按函数执行次数收费

## 应用场景

除了传统的服务端，还可以开发一些小程序，将来物联网也可能会应用

## WebAssembly

## 可以使非JS语言能在浏览器上运行的技术

## 上一次性能提升是因为浏览器加入了即时编译器（JIT），下一次性能提升就是WebAssembly

## JIT：浏览器的JS引擎会将JS转成机器语言，解释器或者编译器，解释器就是一行行解析，编译器是全解析完再执行，解释器快，但是可能会重复执行（循环），编译器慢，但是效率高

## 请求更快，因为WebAssembly文件体积更小（采用二进制）

## 解析更快，因为WebAssembly已经是字节码，不需要在解析了，正常应该先解析成AST，在被转换字节码

## 编译和优化更快，LLVM之前已经优化过了

## 执行更快，因为更贴近机器

## 不支持垃圾回收

## 原理：将高级语言转换为一种低于高级语言但是高于汇编代码的一种语言（IR）

# 排序

## 冒泡排序（俩俩比较，如果比第二个大就交换）

平 n2 坏 n2 好 n 空 1 稳

## 算法描述

### 比较相邻的元素。如果第一个比第二个大，就交换它们两个；

### 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样在最后的元素应该会是最大的数；

### 针对所有的元素重复以上的步骤，除了最后一个；

### 重复步骤1-3直到排序完成

## 选择排序（找最小的与前面的交换）

平 n2 坏 n2 好 n2 空 1 不稳

## 算法描述 n个记录的直接选择排序可经过n-1趟直接选择排序得到有序结果。具体算法描述如下：

### 初始状态：无序区为R[1..n]，有序区为空；

### 第i趟排序(i=1,2,3…n-1)开始时，当前有序区和无序区分别为R[1..i-1]和R(i..n）。该趟排序从当前无序区中-选出关键字最小的记录 R[k]，将它与无序区的第1个记录R交换，使R[1..i]和R[i+1..n)分别变为记录个数增加1个的新有序区和记录个数减少1个的新无序区；

### n-1趟结束，数组有序化了。

## 插入排序（与前面排序过的作比较，看哪里合适插在哪里）

平 n2 坏 n2 好 n 空 1 稳

## 算法描述 一般来说，插入排序都采用in-place在数组上实现。具体算法描述如下：

### 从第一个元素开始，该元素可以认为已经被排序；

### 取出下一个元素，在已经排序的元素序列中从后向前扫描；

### 如果该元素（已排序）大于新元素，将该元素移到下一位置；

### 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置；

### 将新元素插入到该位置后；

### 重复步骤2~5。

## 归并排序（将序列拆分，再拆分，将排序好的子序列排序）

平 nlogn 坏 nlogn 好 nlogn 空 n 稳

## 算法描述

### 把长度为n的输入序列分成两个长度为n/2的子序列；

### 对这两个子序列分别采用归并排序；

### 将两个排序好的子序列合并成一个最终的排序序列。

## 快速排序（找一个基准值，小的放前面，大的放后面）

平 nlogn 坏 n2 好 nlogn 空 nlogn 不稳

##算法描述 快速排序使用分治法来把一个串（list）分为两个子串（sub-lists）。具体算法描述如下：

### 从数列中挑出一个元素，称为 “基准”（pivot）；

### 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作；

### 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

# 项目

## 下载文件

利用a标签的download属性，使用blob进行解析，在使用URL.createObjectURL关联一个url，设置a.href为url，download为文件名，再清除dom节点和url即可，但是在加密后会产生乱码，原因是responseType由blob变成了arraybuffer，blob是个构造函数，arraybuffer类似于数组，乱码大概率是因为blob主要解决媒体类型，arraybuffer主要针对数据本身，更改type之后问题解决

## 虚拟列表

为什么：

在权限系统内要录入每个系统的权限点，权限点又有层级关系，包括系统级，菜单级，按钮级，有一个一键展开所有权限点的需求，权限点很多，但是同一个系统的权限点分页又似乎不是很友好，所以想到了虚拟列表

有几个关键的数据：

1. 每个数据的高度
2. 虚拟列表总高度
3. 可视范围内可以展示数据内容及个数
4. 虚拟列表的偏移量

布局方面：

一个大div里包含两个子div，一个div渲染内容为了撑起布局，产生滚动条，另一个div才是真的列表，监听大div的滚动事件，通过计算列表的scrollTop改变偏移量，使列表可以跟着滚动条上下偏移达到效果。

改进1:

在实际开发中发现每一项的高度实际上是不固定的，所以无法用定高的方案。最后采用的方案是为每一条数据维护一个布局缓存，在刚获取到数据时先用默认设计好的高度初始化，在渲染完毕后，再根据其真实的高度去更改缓存（在updata生命周期内，nextTick方法内获取高度，改变偏移量并改变当前及以后所有的缓存）。

改进2：

当快速滚动时，因数据需要重新渲染所以会有短暂的白屏，因此在列表上下加入两个缓冲区，也就是说，真正渲染的内容实际上是大于屏幕内的内容的（偏移量的计算需要更改）。

改进3:

可以用getBoundingClientRect或者IntersectionObserver进行改造