**底层原理**

# Js

## 闭包

什么是闭包？闭包的特性？

1、函数嵌套函数

2、函数内部可以引用外部的参数和变量

3、参数和变量不会被垃圾回收机制回收

闭包的缺点：变量长期存在占用内存，在不使用时要做到变量及时释放

**Demo1**

function couter() { var num = 0;

return { // 增 up: function() { num++ },

down: function() { num--; }// 获取

getNum: function() {console.log(num);}

}

}

var c1 = couter();

c1.up();

c1.up();

c1.getNum(); // 2

**Demo2**

for (var i = 0; i < 5; ++i) {

func[i] = function() {

console.log(i);

}

(function(i) {

func[i] = function() {

console.log(i);

}

})(i);

}

func[3](); // 3

## 高阶函数

什么是高阶函数？高阶函数的特点？

1、函数可以作为参数被传递

2、函数可以作为返回值输出

常见的高阶函数Debounce和Throttle

**Debounce（防抖）**

function debounce(fn,delay){

delay = delay || 1000

var timer = null

return function() {

var arg = arguments

clearTimeout(timer)

timer = null

timer = setTimeout(function(){

fn.apply(this,arg)

},delay)

}

}

Var num=0

window.onscroll = debounce(function(e){

console.log(e,++num)

},2000)

**Throttle(节流)**

function(fn,delay){

var timer = null

var lastTime = 0

return function(){

var arg = arguments

clearTimeout(timer)

timer = null

var now = new Date().getTime()

if(lastTime === 0) {

lastTime = now

}

if(now – lastTime >= delay) {

fn.apply(this, arg)

lastTime = now

} else {

timer = setTimeout(function(){

fn.apply(this, arg)

lastTime = 0

},delay)

}

}

}

Var num=0

window.onscroll = debounce(function(e){

console.log(e,++num)

},2000)

## 柯里化（currying）

什么是柯里化？柯里化的特点

又称为部分求值，是把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回一个新的函数的技术，新函数接受余下参数并返回运算结果

**Demo**

function verify(reg){

return function(text){

return reg.test(text)

}

}

const isNum = verify(/\d/g)

const isString = verify(/[a-z]/g)

console.log(isNum(1))

console.log(isString(1))

## 原型，原型链，继承

Prototype就是调用构造行数所创建的实例对象的原型

多个原型通过\_proto\_链接来就是原型链

普通继承缺点，引用类型被所有子类共享，不能向父类传参

组合继承

function Vehicle (name){

this.name = name

} // 创建构造函数

Vehicle.prototype.show = function () {

alert(this.name);

}// 为原型Vehicle 添加一个方法

function Car (name) {

Vehicle.call(this, name)

} // 新建一个原型对象

Car.prototype = new Vehicle() // 将vehicle对象的原型链继承到Car上

var car = new Car(‘Benz’) //实例化Car

console.log(car.name)

car.show()

组合寄生继承

function Parent(name) { this.name = name }

Parent.prototype.show = function() { console.log(this.name) }

function Child(name) { Parent.call(this, name) }

Child.prototype = Object.create(Pranet.prototype, {

constructor: { value: Child, enumerable: false, writable: true, configurable: true }

})

const child = new Child(‘izaya’)

child.show()

ES6的关键字 class, constructor,static,extends,super

class Parent {

constructor(value) { this.value = value }

getVal() { console.log(this.val) }

}

class Child extends Parent { constructor(value) { super(value); this.val = value }}

const clild = new Child(‘fff’)

child.getVal()

## 跨域

什么是跨域：在协议、主机或端口不一致的时候，即判定为跨域

解决方案

Jsonp：仅支持get请求

CORS：在服务端添加Access-Control-Allow-Origin:\*

postMessage+ifrme： 这个也是需要目标服务器或者说是目标页面写一个postMessage，主要侧重于前端通讯

nginx反向代理：搭建一个中转nginx服务器，用于转发请求

vue内置的代理

## new 实现原理

1，创建个类的实例，然后将空对象的\_proto\_设置为类的prototype

2，初始化实例：调用传入参数，修改this指向该空实例

3，返回实例

function create() {

let obj = {}

let Con = [].shift.call(arguments)

obj.\_\_proto\_\_ = Con.prototype

let result = Con.apply(obj, arguments)

return result instanceof Object ? result : obj

}

## this

多情况优先级 new > bind > obj.fn() > fn

## dom0、dom2、dom3事件

dom0：就是onclick直接绑定

dom2：就是addeventListener

dom3： 新增鼠标键盘等事件

## call、bind、apply用法与区别

fn.call(this, arg1, arg2);

func.apply(this, [arg1, arg2])

foo.getX.bind(obj)()

foo.getX.call(obj)

foo.getX.apply(obj)

bind 是返回对应函数，便于稍后调用；apply 、call 则是立即调用 。

bind传入参数跟call一样

## EventLoop

### 线程与进程

两个都是CPU工作时间片的描述，进程描述CPU在运行指令或加载和保存上下文所需要的时间，放在应用上来说就是个程序。线程是进程的更小单位，描述执行一段指令所需要的时间

### 执行栈

是一个储存函数调用的栈结构，遵循先进后出的原则。当我们执行js时会先执行一个main函数然后执行代码，并遵循先进后出的原则，由于执行栈的数量有限，所以如果递归过多，会导致爆栈溢出

### 浏览器

中的任务源分macrotask（宏任务）、microtask（微任务）。执行的顺序是

1执行所有同步代码，这是宏任务

2执行同步代码完成之后，执行栈为空，查询是否有异步代码需要执行

3执行所有微任务

4执行完微任务后，如有必要则刷新页面

5开始下一轮eventloop并执行宏任务中的异步代码

### Node中的Event Loop

**:**分为6个阶段，它们会按照顺序反复执行，每当进入某个阶段都会从对应的回调队列中取函数来执行。当队列为空，或者处理的数量达到系统阈值时，则进入一下个阶段。

1，timer阶段会执行setTimeout与setInterval，并且是由poll阶段控制的

2，I/O阶段：队列执行几乎所有的异常行为的回调函数，处理上一个循环中少数未执行的的I/O回调

3，idle，prepare：内部调用

4，poll阶段：1、回到timer阶段执行回调，2、执行I/O回调。如果poll队列不为空，会遍历队列并同步执行，知道为空或达到阈值。如果队列为空则判断setImmediate有回调需要执行。如果有poll阶段会停止并且进入到check阶段执行回调。如果没有会等到回调被加入队列，等待会设置超时时间，防止一直等待。

5 check执行setImmediate

6 close callbacks 执行close事件

## ES6

### reduce

两个参，第一个回调函数，第二个默认第一次的上次值

计算累加回调函数有四个参，上次计算值，当前值，下标，循环的数组，

const sum = arr.reduce((previousValue, current, index, arr) => {

console.log(previousValue, current, index, arr)

return previousValue + current

}, 0)

# vue

## Vue实现这种数据双向绑定原理

1、Observer：能够对数据对象的所有属性进行监听，如有变动可拿到最新值并通知订阅者。

核心是通过Obeject.defineProperty()来监听数据的变动，这个函数内部可以定义setter和getter，每当数据发生变化，就会触发setter。

2、Compile：是解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图3、Watcher（观察者）：作为连接Observer和Compile的桥梁，能够订阅并收到每个属性变动的通知，执行指令绑定的相应回调函数，从而更新视图。（1）在自身实例化时往属性订阅器(dep)里面添加自己（2）自身必须有一个update()方法（3）待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调

## Template编译过程

先转化为AST树，在得到的render函数中返回VNode（vue的虚拟DOM节点）

首先，通过compile编译器把template编译成AST语法树（abstract syntax tree，即源代码的抽象语法结构的树状表现形式），compile是creatCompiler（createCompiler是用以创建编译器的）的返回值，compile负责合并option

然后，AST会经过generate（将AST语法树转化成render function字符串的过程）得到render函数，render的返回值是VNode，VNode是Vue的虚拟DOM节点，里面包含（标签名、子节点、文本等）

# Webpack

## 核心：

entry入口，output出口，module模块，chunk代码块，loader文件转译器，pulgin插件

## 构建流程：

1，初始化参数，从配置文件与sell脚本读取合并参数。

2，开始编译，用参数初始化cpmpiler对象加载所有插件，执行run方法开始编译

3，确定入口

4，编译模块，从入口文件出发然后找到依赖模块使用loader翻译后编译，递归执行，直到所有文件都执行过一次

5，输出资源：根据 依赖组装chunk，再把chunk转换成文件

6，输出完成：确定好输出内容后根据配置把文件写入到文件系统中

## 性能优化

优化打包时间

1：优化loader搜索范围

2: HappyPack插件使loader的单线程执行变为并行，提高打包销量

3：DllPlugin插件将一些特定的类提前打包引入，当有更新是才会重新打包

减少打包体积

1，按需加载

2，ScopeHoisting将有依赖的文件尽可能打包到一个文件中，webpack4中会在生产环境自动执行

# **CommonJs规范与AMD/CMD规范**

## CommonJS

模块规范主要分为三部分：模块引用、模块定义、模块标识。其加载模块是同步的，只有加载完成，才能执行后面的操作。

## AMD/CMD：

1、对于依赖的模块AMD是提前执行，CMD是延迟执行。不过RequireJS从2.0开始，也改成可以延迟执行（根据写法不同，处理方式不通过）。

2、AMD推崇依赖前置（在定义模块的时候就要声明其依赖的模块），CMD推崇依赖就近（只有在用到某个模块的时候再去require——按需加载）。

3、AMD的api默认是一个当多个用，CMD严格的区分推崇职责单一。例如：AMD里require分全局的和局部的。CMD里面没有全局的 require,提供 seajs.use()来实现模块系统的加载启动。CMD里每个API都简单纯粹。

# HTTP协议

HTTP 是基于 TCP/IP 协议的应用层协议。它不涉及数据包（packet）传输，主要规定了客户端和服务器之间的通信格式，默认使用80端口

## 状态码

301 永久重定向；

302 Found 临时性重定向

400 Bad Request 客户端请求有语法错误，不能被服务器所理解

401 Unauthorized 请求未经授权

403 Forbidden 服务器收到请求，但是拒绝提供服务

404 Not Found 请求资源不存在

500 Internal Server Error 服务器发生不可预期的错误

503 Server Unavailable 服务器当前不能处理客户端的请求，一段时间后可能恢复正常

## 请求方法

GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT

## 缓存

HTTP 缓存分为 2 种，一种是强缓存，另一种是协商缓存。主要作用是可以加快资源获取速度，提升用户体验，减少网络传输，缓解服务端的压力。

### 强缓存：

不需要发送请求到服务端，直接读取浏览器本地缓存，在 Chrome 的 Network 中显示的 HTTP 状态码是 200 ，在 Chrome 中，强缓存又分为 Disk Cache（存放在硬盘中）和 Memory Cache（存放在内存中），存放的位置是由浏览器控制的。是否强缓存由 Expires、Cache-Control 和 Pragma 3 个 Header 属性共同来控制

**Expires** 的值是一个 HTTP 日期，在浏览器发起请求时，会根据系统时间和 Expires 的值进行比较，如果系统时间超过了 Expires 的值，缓存失效。由于和系统时间进行比较，所以当系统时间和服务器时间不一致的时候，会有缓存有效期不准的问题。Expires 的优先级在三个 Header 属性中是最低的。

**Cache-Control** 是 HTTP/1.1 中新增的属性，在请求头和响应头中都可以使用，常用的属性值如有：

max-age：单位是秒，缓存时间计算的方式是距离发起的时间的秒数，超过间隔的秒数缓存失效

no-cache：不使用强缓存，需要与服务器验证缓存是否新鲜

no-store：禁止使用缓存（包括协商缓存），每次都向服务器请求最新的资源

private：专用于个人的缓存，中间代理、CDN 等不能缓存此响应

public：响应可以被中间代理、CDN 等缓存

must-revalidate：在缓存过期前可以使用，过期后必须向服务器验证

**Pragma** 只有一个属性值，就是 no-cache ，效果和 Cache-Control 中的 no-cache 一致

### 协商缓存

当浏览器的强缓存失效的时候或者请求头中设置了不走强缓存，并且在请求头中设置了If-Modified-Since 或者 If-None-Match 的时候，会将这两个属性值到服务端去验证是否命中协商缓存，如果命中了协商缓存，会返回 304 状态，加载浏览器缓存，并且响应头会设置 Last-Modified 或者 ETag 属性

**ETag/If-None-Match** 的值是一串 hash 码，代表的是一个资源的标识符，当服务端的文件变化的时候，它的 hash码会随之改变，通过请求头中的 If-None-Match 和当前文件的 hash 值进行比较，如果相等则表示命中协商缓存。ETag 又有强弱校验之分，如果 hash 码是以 "W/" 开头的一串字符串，说明此时协商缓存的校验是弱校验的，只有服务器上的文件差异（根据 ETag 计算方式来决定）达到能够触发 hash 值后缀变化的时候，才会真正地请求资源，否则返回 304 并加载浏览器缓存。

**Last-Modified/If-Modified-Since** 的值代表的是文件的最后修改时间，第一次请求服务端会把资源的最后修改时间放到 Last-Modified 响应头中，第二次发起请求的时候，请求头会带上上一次响应头中的 Last-Modified 的时间，并放到 If-Modified-Since 请求头属性中，服务端根据文件最后一次修改时间和 If-Modified-Since 的值进行比较，如果相等，返回 304 ，并加载浏览器缓存。

## UDP与TCP的区别已经各自特点

UDP协议是面向无连接，不需要连接双方，不对数据进行解析，不保证有序且不丢失的传递。UDP协议也没有控制流量的算法。但相对于TCP，UDP更加轻便

**UDP特点：**

1面向无连接：发送端，应用层将数据传递给传输层UDP协议，UDP协议添加一个头标识传递给网络层。接收端，网络层将数据传给传输层，UDP将头标识去除，丢给应用层。

2不可靠性：通信无需链接，想发就发。接收到数据便发送数据，既不备份也不关心是否被收到。没有拥塞控制，以一定的速度发送数据，遇到网络不好使容易丢包

3高效：头部仅八字节，相对于TCP至少二十字节来说高很多。头部包含了两个十六位的端口号，即源端口和目标端口。数据报文长度。数据报文校验和IPV4字段。

4：传输方式： 支持一对一，一对多，多对多，多对一，单薄，多播，广播功能

**TCP**

头部：sequence number，保证传送报文是有序的。Acknowledgement Number表示下次期望接受序号以及当前已接受序号。Window size 用于流量控制。标识符

状态机：

三次握手。1.客户端向服务端发送链接请求报文，包含自身数据通讯初始序号。请求后客户端进入SYN-SENT状态。2服务端接收到请求报文段后，如同意链接则发送个应答，应答中包含自身数据通讯初始序号。发送后进入SYN-RECEIVED转态。3当客户端接收到同意应答后，向服务端发送确认报文，发送后进入ESTABLISHED状态，服务端接口到确认应答后也进入ESTABUSHED状态，此时连接建立成功。

四次挥手：1客户端认为数据发送完成，向服务端发送连接释放请求。2服务端接受到后会告诉应用层释放TCP连接，然后发送ACK包，然后进入CLOSE\_WAIT转态。此时表明服务端已不再接受客户端数据。但由于TCP是双向的，所以服务端仍可以给客户端发送数据。3如果此时还有未发完的数据会继续发送，发完后会向客户端发送连接释放请求，然后服务端进入LAST-ACK状态。4客户端接口到释放请求后，向服务端发送确认应答，然后进入TIME-WAIT状态，持续2MSL时间，期间未收到服务端重发请求则进入CLOSED状态。当服务端收到应答后也进入CLOSED状态

## http2优化

1引入**多路复用**技术，优化了http1的队头阻塞，可以一个tcp连接发所有的请求，提高了浏览效率。帧和流的概念：帧代表最小的数据单位，每个帧会标识出属于哪个流，流就是多个帧组成的数据流。多路复用就是一个tcp中存在多条流

2将所有的数据分隔，并采用**二进制编码**

3，**header压缩**：使用HPACK压缩格式对传输的header进行编码，并在两端维护索引表，用于记录出现过的header，后续发送就可以通过表进行接收。提高了头传送效率

4**服务端push**：通过push技术可以再客户端请求时主动推送必要资源，可以减少延迟时间，也可使勇prefetch

## http3优化

优化了由于丢包而导致的队头阻塞。使用QUIC来实现，QUIC是基于UDP协议，但实现了很多额外功能：1多路复用，由于http2的多路复用只能流的多路复用，无法做到tcp的复用，所以会出现丢包边阻塞的现象，QUIC不仅可以做到流多路复用，且原生上就支持tcp的复用。解决了这个问题。2，O-RTT，类似与tcp的快速打开技术，缓存当前上下文，下次会话时可以直接从缓存传递给服务端验证通过就可以传输了。3纠错机制，发送包时例如需发三个包，这是就会追加发送个校验包，如果丢失一个包，则可以用剩余三个包计算出丢失的包的内容。如果超过一个，则只能重发

# 性能优化

雪碧图，懒加载，预加载，DNS预解析、节流、防抖、预渲染、CDN, 代码压缩，按需加载，

# 安全

## XSS攻击：

将可执行代码注入到网页或服务端中，分持久型与非持久型。通过接口注入到服务端中的是持久型。通过修改URL参数的方式加入攻击代码，诱导用户访问链接从而攻击。

两种防御方式：1，转义字符，将用户的引号尖括号斜杠进行转译。2，CSP本质上是建立白名单，明确浏览器外部资源可以加载和执行，通过HTTP header中content-security-policy或者设置meta标签的方式<meta http-equiv="Content-Security-Policy">

## CSRF攻击

跨站请求伪造。攻击者构造一个后端请求地址，诱导用户点击或者自动发起请求。如果用户是登录状态下，后端就以为是用户操作，从而进行相应逻辑。

防御方法：1，get请求不对数据进行修改。2，不让第三方网站访问到cookie，3，阻止第三方网站请求接口。4，请求时附带验证信息如token。

# 设计模式

工厂模式：

# 面试题

**移动端一像素四种解决方案**1使用媒体查询设置小数像素（对设备有要求，兼容性差）

2viewport+rem+js：用js获取devicePixelRatio属性进行判断，然后设置根元素fontSize

3trasform：scale

4媒体查询+transform

**什么是BFC？**

BFC 全称为 块格式化上下文 (Block Formatting Context) 。

例如内部块元素浮动后，外部块元素无法维持高度，需要清除浮动。