**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

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

var obj = new Base();

var obj = {};

obj.\_\_proto\_\_ = Base.prototype;

Base.call(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

# 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

# **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。