■输入网址按下回车后，浏览器做了什么

●第一步：解析URL

输入的URL也叫“统一资源定位符”俗称“网址”

URL只是IP地址的映射

拿到输入的URL后，浏览器会先读取本地浏览器缓存，如果没有对应的IP，就去DNS域名系统去匹配真实的IP地址，并存到本地浏览器缓存，第二次访问就可以直接读取了。

●第二步：建立连接

拿到IP地址后，建立连接 TCP三次握手

●第三步：拿到数据，渲染页面

HTML → Dom树

CSS → CSS结构体

JS → 脚本

通过以上拿到的数据，然后以下步骤

Render Tree 渲染树

计算布局信息

UI引擎渲染

用户所见页面

●第四步：断开连接

TCP四次挥手

■TCP协议UDP协议 三次握手与四次挥手

|  |
| --- |
| 应用层 |
| 传输层 |
| 网络层 |
| 数据链路层 |
| 物理层 |

TCP和UDP协议都工作在传输层，目标都是在程序之间传输数据

TCP和UDP的区别在于一个基于连接，一个基于非连接

UDP基于非连接

TCP基于连接

TCP由于是基于非连接的，因此有信息传输安全性问题，例如：

对方是否收到

内容是否完整

顺序是否正确

等

**为了保证传输的信息安全无误，TCP有三个步骤：**

**三次握手**

**传输确认**

**四次挥手**

**三次握手是建立连接的过程**

**四次挥手是关闭连接的过程**

两者区别

TCP 稳定可靠（适用于传输文件，发送邮件，浏览网页等对于丢包要求敏感的场景）

UDP 性能损耗少，资源占用少，但稳定性弱（适用于对实时性要求较高，对丢包不敏感的场景。比如域名查询，语音通话，视频直播，隧道网络等）

■前端性能优化

●页面加载性能

提高加载速度，提高用户体验

减少http请求（雪碧图/精灵图，文件的合并）

减小文件大小（资源压缩，图片压缩，代码压缩）

CDN（第三方库、大文件、大图等，通过链接来引用）

SSR服务端渲染（预渲染）

懒加载

分包（一个包太大，为了性能优化，分开，让一开始不用加载那么大）

●性能优化（动画与操作性能）

目前浏览器性能都是超过项目需求的。但也可以通过以下，来再次提高浏览器性能：

减少dom操作，避免回流，文档碎片

●内存占用

内存占用过大，浏览器崩掉等

即时通讯等场景。暂时不用太考虑

●电量消耗

游戏方面，暂时不考虑

■跨域

●浏览器的同源策略

协议、域名、端口，3个都相同就是同源

CORS策略 （Cross Origin Resource Sharing）

注意：图片、视频文件不存在跨域。只有通过ajax请求的数据存在跨域问题

因此，同源策略不会限制<link><img><script><iframe>这些标签加载第三方同源

●解决方法

1、后端修改相应头

“Access-Control-Allow-Origin”

2、JSONP

后端开个后门，前端通过script标签来请求

3、Ngax反向代理

**◎目前vue等主流框架都使用代理来解决跨域问题**

**前端配置代理来中转请求**

**前端请求中转服务器，**

■前端安全问题

●前端安全的类型：

XSS攻击：跨站脚本攻击，主要实用javascript向网站注入代码，以实现盗号、窃取资料等目的。

csrf攻击：跨站伪造请求，冒充用户进行一些操作，甚至完成银行的转账。

sql注入：直接攻击系统的数据库，可以实现数据级别的操作，盗取数据。

文件上传漏洞：上传自定义的php代码并执行。

●XSS的防范

过滤：对提交内容中的非法标签和属性（例如<script> onclick等）进行过滤，从而避免XSS攻击。

转义：通过实用htmlspecialcharts等函数，将提交内容中的字符”<” 和”>”转换为HTML实体。

●CSRF的防范

referer防范：通过判断请求头中的referer头，确定请求的来源，从而避免CSRF攻击。

Token防范：通过在表单请求中添加token，标识表单的合法性，来避免CSRF攻击。

■session 和token和 cookie

●什么是session

Session存在服务器端。服务器为了保存用户状态而创建的一个特殊对象。

当浏览器第一次访问服务器时，服务器创建一个session对象（该对象有一个唯一的id，一般称之为session id），服务器会将session id以cookie的方式发送给浏览器。

当浏览器再次访问服务器时，会将session id发送过来，服务器依据session id 就可以找到对应的session对象。

●什么是token

Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌。当第一次登录后，服务器生成一个token便将此返回给客户端，以后客户端只需带上这个token前来请求数据，即可无需再次带上用户名和密码。

●Token和cookie的区别

Cookie有储存上限，4KB

COOKIE是浏览器默认存储和发送的行为

Token是程序员自己存储和发送的行为,存到localStorage中

■兼容问题

浏览器默认的margin和padding不同。解决方法是加一个全局的\*{margin:0; padding:0;}来统一样式

Chrome中文界面下默认会将小于12px的文本强制按照12px显示，可通过加入CSS属性

-webkit-text-size-adjust:none;来解决

IE浏览器的img会有个边框。可以通过全局取消图片边框来解决。

■MVVM开发模式的理解

MVVM为Model、View、ViewModel

Model代表数据模型。数据和业务逻辑都在Model层中定义。

View代表UI视图，负责数据的展示。

ViewModel负责监听Model中数据的改变并控制视图的更新，处理用户交互操作。

Model和View并无直接关联，而是通过ViewModel来进行联系的，Model和ViewModel之间有着双向数据绑定的联系。因此当Model中的数据改变时会出发View层的刷新，View中由于用户交互操作而改变的数据也会在Model中同步。

这种模式实现了Model和View的数据自动同步，因此开发者只需要专注对数据的维护操作即可，而不需要自己操作dom。

■用递归函数写1到100求和

function add(num1, num2) {

    const num = num1 + num2;

    if (num2 === 100) {

        return num;

    } else {

        return add(num, num2 + 1)

    }

}

var sum = add(1, 2);

console.log(sum)

■作用域链和闭包

●每个执行环境（execution context）都有一个与之关联的变量对象（variable object），该执行环境中定义的所有变量和函数都保存在这个变量对象中。

●执行环境有全局执行环境和局部执行环境：

◎全局执行环境被认为是window对象，因此全局变量和函数都是作为window的属性和方法创建的。全局环境直到应用程序退出时才被销毁。

◎局部执行环境中的所有代码执行完毕后，该环境被销毁，里面的所有变量和函数也随之被销毁。

●函数：

◎每个函数都有自己的执行环境。

◎当创建一个函数时（也可以理解成当预处理程序读到一个函数时），其作用域链被保存在内部属性[[Scope]]中。（Scope中保存的是创建该函数时的变量对象。比如在全局作用域下创建一个函数a，Scope保存的就是全局变量对象，即window对象。然后在a里又创建一个函数b，那么b的Scope保存的就是a的Scope加上a的活动对象）

◎当调用这个函数时，首先会创建一个当前函数的执行环境及活动对象（activation object）（我理解活动对象就是当前函数的变量对象）。然后从[[Scope]]中读取作用域链，并将新创建的活动对象加到作用域链的最前端。组成目前执行环境的完整的作用域链。

◎当这个函数执行完毕后，局部活动对象被销毁，内存中只保留原本它的[[Scope]]

●闭包是指有权访问另一个函数作用域中的变量的函数。也就是，在一个函数内部创建另一个函数。

●闭包的作用

减少全局变量（私有的方法和变量，避免变量的污染。）

减少传递给函数的参数数量

封装

●闭包的缺点是会常驻内存，会增大内存实用量。解决方法如下，实用完后释放内存

var q = 1;

function a(num) {

    for (var i = 0; i < 10; i++) {

    }

    return function b(nn) {

        return nn + num + i; //在闭包b中可以访问其父级a函数的活动对象,如num和i,注意,访问的是最终结果,比如for循环的i是最终循环完的值10

    };

}

var n1 = a(3);  //a执行完毕后，其活动对象不会被销毁，因为b的作用于链仍然再引用这个活动对象。

var n2 = n1(2);

console.log(n2);  //返回15  nn(2)+num(3)+i(10)

n1 = null;  //解除对b的引用，已释放内存，销毁a的活动对象。

■原型与原型链

●我们创建的每个函数都有一个prototype属性，这个属性是一个指针，指向一个对象。如果使用new操作符来把这个函数当作构造函数调用，那么prototype属性指向的对象就叫做这个构造函数创造出来的对象实例的原型对象。

●Prototype属性指向的对象最初只有一个constructor属性，该属性也是一个指针，指向prototype属性所在的函数。

●

可以给构造函数的prototype对象定义方法和属性，那么它new出来的实例对象就都会继承这些方法和属性（也就是它们自己的原型对象中的方法和属性）

//创建一个构造函数Person

function Person() {

}

//定义通过Person构造函数来创建的对象实例的原型对象的属性和方法

Person.prototype.name = "Tom";

Person.prototype.age = 29;

Person.prototype.job = "worker";

Person.prototype.sayName = function () {

    alert(this.name);

};

var person1 = new Person();

var person2 = new Person();

console.log(person1.job == person2.job);  //会输出true，因为person1和person2引用同一个原型对象。

■This和Call 和apply

This直向当前执行环境的变量

Call和apply用来改变this的指向

Call和apply的区别是，call接受单个的参数，apply接受数组参数

function log(a, b) {

    console.log(this[a])  //不能使用this.a因为a是变量

    console.log(this[b])

}

var a = 'globalA'

var b = 'globalB'

var c = {

    a: 'localA',

    b: 'localB'

}

// 这里直接调用log时，this指向的是window对象，此JS需要在HTML中运行，才有WINDOW对象

log('a', 'b')

// globalA

// globalB

// call和apply可以改变this的指向

// 函数.call(对象（要使this指向这个对象）, 参数1, 参数2, ...)

// 函数.apply(对象（要使this指向这个对象）, 数组（数组里是接受的参数）)

log.call(c, 'a', 'b')

// localA

// localB

log.apply(c, ['a', 'b'])

// localA

// localB

■Promise和async和await

●Promise是异步编程的一种解决方案。其实是一个构造函数，自己身上有resolve、reject等方法，原型上有then、catch等方法。

通过new Promise创建一个它的实例对象，创建时，会将resolve或reject的值保存在这个实例对象中。实例对象通过.then取得resolve的值，或通过.catch取得reject的值

let state = Math.ceil(Math.random()\*20); //生成1-10的随机数

let p1 = new Promise (function (resolve, reject) {

    var data = {

        name: 'joe',

        age: 22,

        say: function () {

            console.log(123)

        }

    }

    if (state > 10) {

        resolve(data)

    } else {

        reject('失败啦')

    }

})

// 方法二：使用.then只接收成功的回调，使用.catch来捕获失败的回调

p1.then(function (data) {

    console.log(data)

    console.log(data.name)

    data.say()

}).catch(function (reson) {

    console.log(reson)

})

●async定义的函数会返回一个Promise的实例对象

let state = Math.ceil(Math.random()\*20); //生成1-10的随机数

// async定义的函数返回的是一个Promise的实例对象

// 示例如下：

// async function timeout() {

//     if (state > 10) {

//         return 'hello world'

//     } else {

//         throw new Error('has Error')

//     }

// }

// timeout().then(function (data) {

//     console.log(data)

// }).catch(function (reson) {

//     console.log(reson)

// })

// await 关键字只能在async 函数里面使用。

// 它的目的是如果一个async函数里还包括promise函数，就等待其内部的promise函数都执行完才改变父async函数的promise状态

// 案例如下

function aAdd (num) {

    return new Promise((resolve, reject) => {

        setTimeout(() => {

            resolve(num + 1)

        }, 500)

    })

}

async function testResult() {

    let first = await aAdd(30);

    let second = await aAdd(50);

    let third = await aAdd(20);

    // 等待所有await的函数都得到结果后，才执行后面的代码

    let result = first + second + third

    if (state > 10) {

        return result

    } else {

        throw new Error('has Error')

    }

}

testResult().then(function (data) {

    console.log(data)

}).catch(function (reson) {

    console.log(reson)

})

■数组和字符串方法

|  |  |
| --- | --- |
| **数组** | **字符串** |
| 实例.join() 数组转字符串 | |
| 实例.split() 字符串转数组 | |
| 实例.concat() 合并 | |
| 实例.slice() 切片 实例.slice(start, end) 从start的位置，截取到end位置之前一位 | |
| 实例.splice() 替换 array.splice(start, deleteCount, item...) 从数组中删除一个或多个元素，并用新的item代替被删除的元素。参数deleteCount是要被删除的个数（从start开始） | 实例.replace() 替换 varstr13d=str13.replace('at', 'ond');  使用字符串'at'只能替换第一个匹配的位置 varstr13e=str13.replace(/at/g, 'xxx');  将第一个参数修改为带有全局标志的正则表达式，就将全部"at"都替换成了"ond"。 |
| 实例.push() | 实例.indexOf() 查找字符串返回位置 |
| 实例.pop() | 实例.lastIndexOf() 倒叙查找字符串返回位置 |
| 实例.unshift() | 实例.charAt() 查找位置返回字符串 |
| 实例.shift() | 实例.charCodeAt() 查找位置返回字符串编码 |
| 实例.reverse() 反转 | 实例.toLowerCase() 转小写 |
| 实例.sort() | 实例.toUpperCase() 转大写 |
|  | 实例.trim() 删前后空格 |
|  | 实例.match() 返回一个对象 |
|  | 实例.search() 返回第一个匹配元素的位置 |
| es6 | |
| Array.form() | `${变量}` |
| Array.of() | 实例.includes() |
| 实例.find() | 实例.startsWith() |
| 实例.fill()  类似splice | 实例.endWith() |
| for of 循环 | 实例.repeat() |
| 实例.entries() |  |
| in 判断对象或数组中是否存在某个值 |  |

■vuex

Vuex 状态管理模式。

Vuex的核心就是store（仓库），包含着应用中大部分的state（状态）

◎Vuex包括以下几个模块：

State 基本数据。

Getter 从基本数据派生的数据。

Mutation 是唯一更改store中状态的方法，且必须是同步函数。

Action 像一个装饰器，包裹mutations，使之可以异步。用于提交mutation而不是直接变更状态，可以包含任何异步操作。

Module 模块化vuex，允许将单一的store拆分为多个store且同时保存在单一的状态树种。

■vue的生命周期和钩子函数

生命周期就是各个阶段，比如created、mounted、updated等

钩子函数是在各个生命周期所执行的命令。

■DOM和BOM

●BOM（浏览器对象模型）

|  |  |  |
| --- | --- | --- |
| window对象 | BOM的核心对象 | alert() confirm()对话框等 |
| location对象 | 对浏览器URL的相关操作 | P207 |
| navigator 对象 | 客户端浏览器的信息 | P210 |
| screen 对象 | 客户端的信息，例如客户端的显示器 | P214 |
| history 对象 | 上网历史记录 | P215 |

●DOM（文档对象模型）

Node节点

getElementById

■离职原因

2022年5月转到智联电商。

主：

办公环境（拥挤，工位小主要还是在打印机旁边）

工作方面（换项目，混乱）

次：

工作氛围（没有事还要提加班）

领导管理风格

不是说智联不好，只是不适合我。

■前端问题汇总

https://blog.csdn.net/q95548854/article/details/98617043

■flex布局

<https://tool.4xseo.com/a/20.html>

display:flex;

●主轴方向

flex-direction: row;(水平) colum;(垂直)

●是否换行

flex-wrap:nowarp;(不换行，默认) warp;(换行)

●左右对齐方式

justify-content:space-between;(中对齐) flex-start;(左对齐) flex-end;(右对齐)

●上下对齐方式

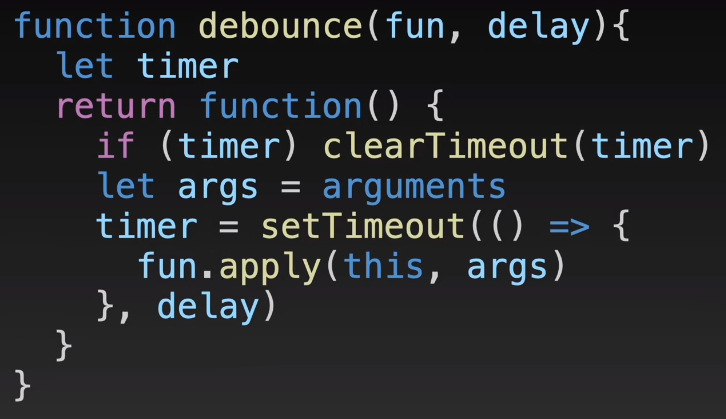
align-items: flex-start;(上对齐) flex-end;(下对齐) center;(居中对齐)

■防抖和节流

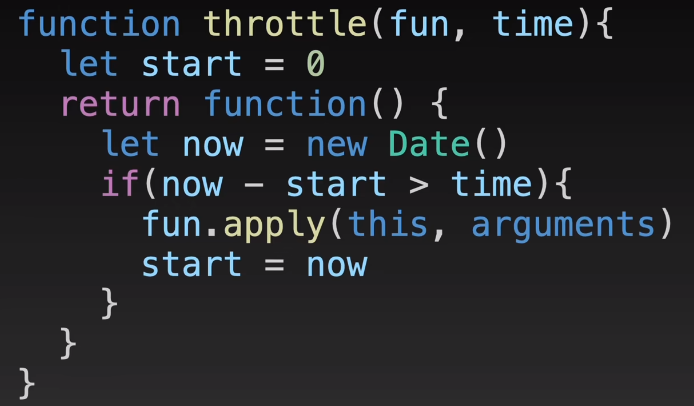
防抖是操作时不执行，确定不操作了才执行。（例如玩手机时不会黑屏，一段时间不操作了，才黑屏）

节流是到时间了即使还在操作，也执行。

防抖：



节流：



■箭头函数

箭头函数的this指向该函数所在对象的父级对象，而不是该函数所在的对象。

箭头函数的this定义时就确定了，且不能修改（不能用call apply等）

箭头函数只是语法糖，因此不能当作构造函数用new，也没有prototype

let bb = {

            name: 'jojo',

            init: function () {

                console.log(this)

            }

        }

        bb.init()

        // 输出：{name: 'jojo', init: ƒ}

        // 这个普通函数的this是该函数所在的对象

        let bb2 = {

            name: 'tom',

            init: () => {

                console.log(this)

            }

        }

        bb2.init()

        // 输出：Window {window: Window, self: Window, document: document, name: '', location: Location, …}

        // 箭头函数的this是该函数所在对象的父级对象，这里是window对象。

        // 因为箭头函数没有自己的作用域

■浅拷贝和深拷贝

JS数据类型：基本数据类型（Undefined, Null, Boolean, Number, String）和引用数据类型（Object）

将一个变量B赋值成变量A时，如果变量A是基本数据类型，没问题。

而如果变量A是引用类型，那么变量B的值是A的引用，因此修改A或B，都会互相关联。

let a = [1, 2, 3]

        let b = a

        b.push(5)

        console.log(a)

        // 输出：[1, 2, 3, 5]  a 也被影响了

要解决这种情况，就需要用深拷贝

let a = [1, 2, 3]

        // 建立b为一个新的空数组，再用ES6的展开运算符把a数组的元素都放进这个空数组里

        let b = [...a]

        b.push(5)

        console.log(a)

        // 输出：[1, 2, 3]  a 不会被影响了