# oneday

## MVVM

model-view-viewmodel

MVVM最早由微软提出，借鉴桌面应用程序MVC思想，在前端页面中，把Model用纯JavaScript对象表示，View负责显示，两者做到了最大限度的分离。

把Model和View关联起来就是ViewModel，负责把Model的数据同步到View显示出来。还负责把View的修改同步回Model。

设计思想：关注Model的变化，让MVVM框架自动更新DOM的状态，从而把开发者从操作DOM的繁琐步骤中解脱出来。

## 常用ES6用法

### let和const

对于变量来说，var定义的变量会提升到作用域中所有的函数和语句前面，let定义的变量会在其相应的代码块中建立一个暂时性死区，不会变量提升，直至变量被声明。

function test() {

console.log(a); // undefined

let a = 1;

}

const声明变量，一旦声明，不可更改，而且常量必须初始化。const虽然是常量，不允许修改默认赋值，但如果定义的是对象Object，那么可以修改对象内部的属性值。

const type = {

a: 1,

}

type.a = 2;

相同点：当前块内有效，执行到块外被销毁；不存在变量提升；不能重复声明。

不同点：const不能再赋值，let可重复赋值；const保证的实际上不是变量的值不得改动，而是变量指向的内存地址所保存的数据不得改动。

典型问题，for循环中使用var声明的循环变量，会跳出循环体污染当前的函数。

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

setTimeout(() => {

console.log(i); // 5,5,5,5,5

})

}

console.log(i) // 5

### symbol

ES6新增数据类型，每一个symbol类型都是独一无二的。

var symbol = Symbol();

typeof symbol // ‘symbol’

### 模板字符串

反引号``

换行

es5

console.log(‘hello world 1\n\

hello you.’)

es6

console.log(`hello world

hello you.`)

${}模板占位符

let name = ‘ben’;

let age = 27;

console.log(`my name is ${ name }, ${ age } years old.`)

### includes(str, index)

在字符串中检测指定文本，返回Bool值。

let str = ‘abcdef’;

if (str.includes(‘cde’)) {

…

}

### startsWith(str, index)

在字符串起始部分检测指定文本，返回Bool值。

### endsWith(str, index)

在字符串结束部分检测指定文本，返回Bool值。

## title属性

规定关于元素的额外信息，当鼠标移到元素上时显示提示信息。

<p title="this is title">Hello Moto</p>



## axios

axios是一个基于promise的http库，可以用在浏览器和node.js中。

### 特性

1. 从浏览器中创建XMLHttpRequests
2. 从node.js创建http请求
3. 支持Promise API
4. 拦截请求和响应
5. 转换请求数据和响应数据
6. 取消请求
7. 自动转换JSON数据
8. 客户端支持防御XSRF

### example

axios.get(‘url’)

.then()

.catch()

axios.post(‘url’, params)

.then()

.catch()

axios.all([url0, url1])

.then(axios.spread(res0, res1) => {…})

## 常用正则

https://www.cnblogs.com/zhangjianbing/p/7538178.html

只含有汉字、数字、字母、下划线不能以下划线开头和结尾

/^(?!\_)(?!.\*?\_$)[a-zA-Z0-9\_\u4e00-\u9fa5]+$/

只能输入英文下划线且不能以下划线开头和结尾

/^(?!\_)(?!.\*?\_$)[A-Za-z\_]+$/

只能输入1个数字

/^\d$/

replace去除逗号的方法

data.toString().replace(/,/g, '')

### 创建实例

axios.create([config])

var instance = axios.create({

baseURL: “”,

timeout: 1000,

headers: “”,

})

实例方法是axios都支持的方法。

### 响应结构

{

// 由服务器提供的响应

data: {},

// 来自服务器响应的HTTP状态码

status: 200,

statusText: ‘OK’,

// 服务器响应的头

headers: {},

// 为请求提供的配置信息

config: {},

}

### 拦截器

// 添加请求拦截器

axios.interceptors.request.use((config) => {

// 在发送请求之前做些什么

return config;

}, error => {

// 对请求错误做些什么

return Promise.reject(error);

})

// 添加响应拦截器

axios.interceptors.response.use(

response => {

return response;

},

error => {

return Promise.reject(error);

}

)

## Promise

Promise对象用于表示一个异步操作的最终状态（完成或失败），以及其返回的值。

const promise = new Promise((resolve, reject) => {

// 做一些异步操作 最终会调用以下两者之一

// resolve()

// 或

// reject()

})

## Math.random()

一个浮点型的伪随机数在[0, 1)之间。

## z-index

设置元素的堆叠顺序，可为负数。设置一个定位元素沿Z轴的位置，z轴定义为垂直延伸到显示区的轴，如果为正数，则离用户更近，反之更远。

## 安装node模块

安装node模块时，如果指定—save参数，那么此模块将会被添加到package.json文件中的dependencies依赖列表中，然后通过npm install命令即可自动安装依赖列表中所列出的所有模块。

## window.open()

let window = window.open(strUrl, strWindowName, [strWindowFeatures]);

strUrl === 要在新窗口打开的URL

strWindowName === 新窗口打开的名称

strWindowFeatures === 可选参数，新窗口的特征

调用window.open()方法后，远程URL不会被立即载入，过程是异步的。实际加载这个URL的时间推迟到当前脚本执行结束以后，窗口的创建和相关资源的加载时异步的。

如果已经存在以strWindowName命名的窗口，则不会再加载新窗口，而是将strUrl加载到这个窗口。如果要每次调用window.open()时都打开一个新窗口，则要把strWindowName设置为\_blank。

let windowObjectReference;

let strWindowFeatures = `

menubar=yes,

location=yes,

resizable=yes,

scrollbars=yes,

status=yes

`;

function openRequestedPopup() {

windowObjectReference =

window.open(

"http://www.cnn.com/",

"CNN\_WindowName",

strWindowFeatures

);

}

## 本地启node服务

npm run build 打包生成的文件目录下，启动bash，然后node server.js。

访问目录<http://localhost:8070/gyyt/suzhou#/>或者本地ip，<http://10.0.0.42:8070/gyyt/suzhou#/，需要注意的是在prod.env.js中的ip>改为可访问的后台ip。

## json数据转换

转换方法写在trans.js文件中，打开cmd（bash下有权限问题），执行node trans.js，在控制台会打印trans.js中console.log(features)输出的内容。但这样只会输出部分features内容，如需输出全部json格式内容，需console.log(JSON.stringify(features))。重定向到某一文件中可执行：

node trans.js >> 1.json。



## 动态绑定style（背景图片）

:style="{ background: 'url(' + mainBg + ') no-repeat' }"

根据不同的url切换背景图片

## node部署代理转发

安装http-proxy-middleware

npm install --save-dev http-proxy-middleware

在server.js中引用

const proxy = require('http-proxy-middleware');

let options = {

target: 'http://172.19.18.158:83', // target host 代理指向的路径

changeOrigin: true, // needed for virtual hosted sites

ws: true, // proxy websockets

pathRewrite: { // 重写

"^/file": '/mapJson/', // 所有带file前缀的请求url 均重写为/mapJson

},};

// create the proxy (without context)

let exampleProxy = proxy(options);

app.use('/file', exampleProxy); // 使用 所有带file前缀的请求url使用代理

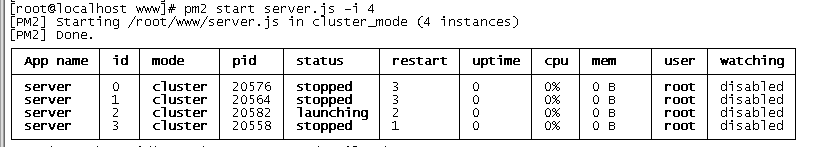
在其他文件中请求

let api = window.location.origin + '/file/' + local.get('map-json'); // 带‘/file’前缀

axios.get(api).then(…)

## pm2 kill

如遇如下情况，可先pm2 kill，然后再启动pm2 start server.js -i 4



## router key

需求是从/post-page/a，跳转到/post-page/b。然后我们惊人的发现，页面跳转后数据竟然没更新？！原因是vue-router”智能地”发现这是同一个组件，然后它就决定要复用这个组件，所以你在created函数里写的方法压根就没执行。

<router-view :key="$route.fullpath"></router-view>

created () {

this.getPost(this.$route.params.id)

},

methods () {

getPost(postId) {

// ...

}

}

## 动态修改浏览器title

document.title = ‘test’

## 判断一个变量是否为函数

const isFunction = variable => typeof variable === ‘function’

## created和mounted

created：在实例创建完成后立即被调用，在这一步，实例已完成一下配置：数据观测，属性和方法的运算，watch/event事件的回调。挂载阶段还没开始，$el属性不可见。视图中的html还没渲染，无法操作dom节点。

mounted：el被新创建的$el替换，并挂载到实例上去之后调用该钩子。mounted不会承诺所有的子组件也都被一起挂载，如果希望等到整个视图都渲染完毕，可以用$vm.nextTick替换mounted。

mounted() {

this.$nextTick( () => {

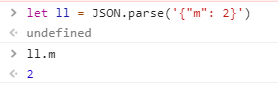
// …

})

}

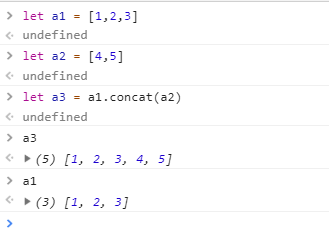
## JSON.parse()

将一个json字符串转换为对象。

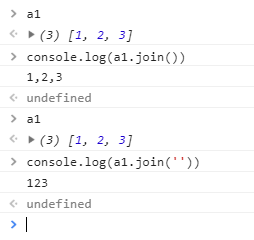


## JavaScript常用数组操作方法

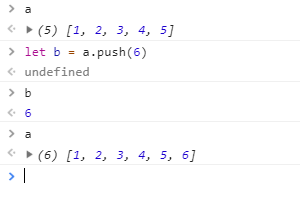
1. concat() 用于连接两个或多个数组，该方法不会改变现有数组，仅会返回被连接数组的一个副本。



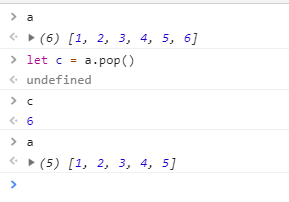
1. join() 用于将数组中的所有元素放入一个字符串，元素通过指定的分隔符进行分割，默认使用逗号‘，’，不改变原数组。



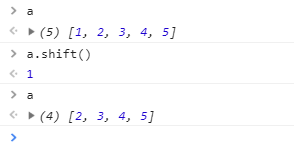
1. push() 向数组的末尾添加一个或多个元素，并返回新的长度。会改变原数组。



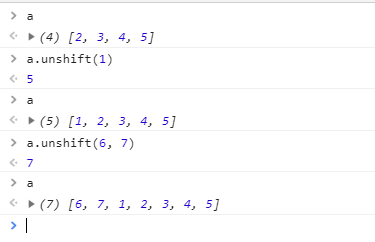
1. pop() 用于删除并返回数组的最后一个元素，会改变原数组。



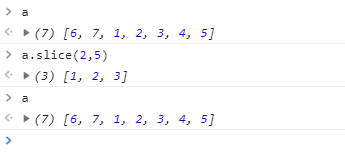
1. shift() 用于把数组的第一个元素删除，并返回第一个元素的值。改变原数组。



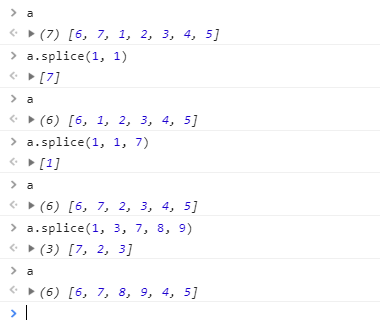
1. unshift() 可向数组的开头添加一个或多个元素，并返回新的长度。改变原数组。



1. slice() 返回一个新数组，包含从start到end（不包含该元素）的数组中的元素，返回选定的元素，不会修改原数组。

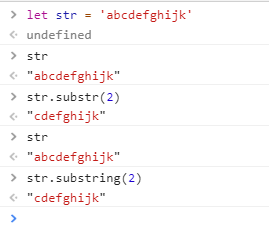


1. splice() 可删除从index处开始的零个或多个元素，并且用参数列表中声明的一个或多个值来替换那些被删除的元素。如果是删除元素，返回的则是含有被删除元素的数组，会改变原数组。



1. substring()和substr()

**相同点**：如果只是写一个参数，两者作用是一样的。都是截取从当前下标到字符串结尾的片段。

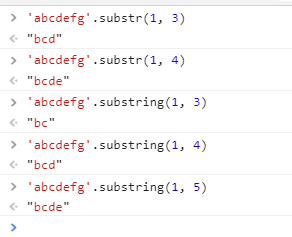


**不同点：**第二个参数

substr(startIndex, length) 第二个参数是截取字符串的长度。

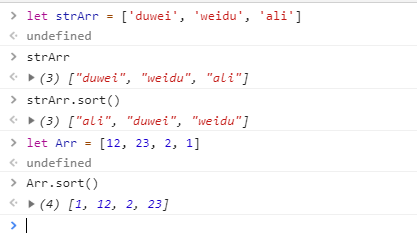
substring(startIndex, length) 第二个参数是截取字符串最终的下标（不含尾）。

都不改变原数组

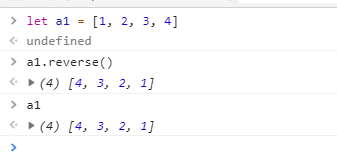


1. sort() 排序

按照Unicode code排序，默认升序。



1. reverse() 用于颠倒数组中元素的顺序，改变原数组。

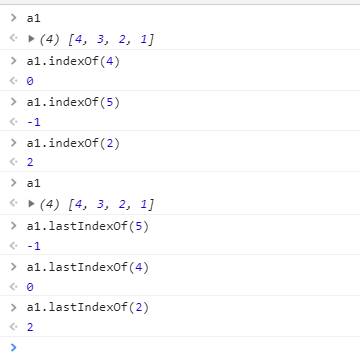


1. indexOf和lastIndexOf

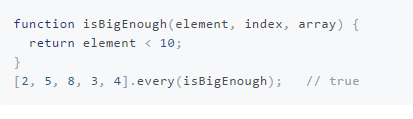
都接受两个参数，查找的值，查找的起始位置。

不存在，返回-1；存在，返回位置。

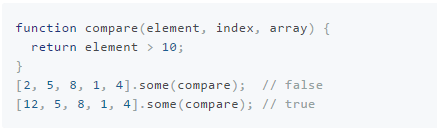
indexOf是从前面往后查找，lastIndexOf是从后面往前找。



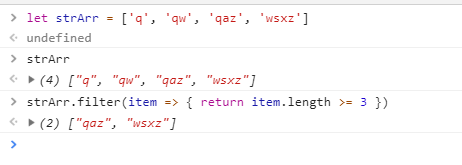
1. every 对数组的每一项都运行给定的函数，每一项都返回true，则返回true。



1. some对数组的每一项都运行给定的函数，任意一项返回true，则返回true。

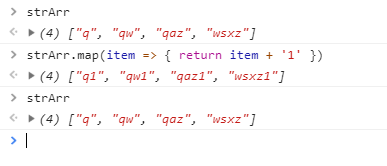


1. filter 对数组的每一项都运行给定的函数，返回结果为true的向组成的数组。



不会对空数组进行检测 不会改变原始数组

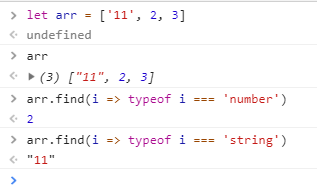
1. map 对数组的每一项都运行给定的函数，返回每次调用的结果组成的新数组。



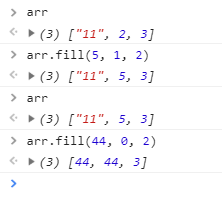
1. forEach 数组遍历



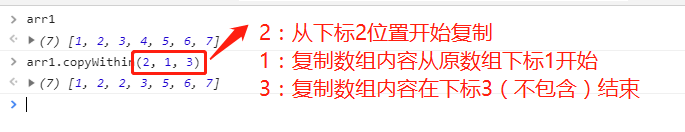
1. find() 传入一个回调函数，找到数组中符合当前搜索规则的第一个元素，返回它并且终止搜索。如果没找到返回undefined。



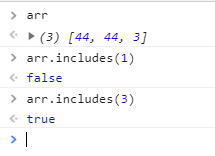
1. findIndex() 传入一个回调函数，找到数组中符合当前搜索规则的第一个元素，返回它的下标，并终止搜索。
2. fill() 用新元素替换掉组内的元素，可以指定替换下标范围。



1. copyWithin() 选择数组的某个下标，从该位置开始复制数组元素。也可以指定要复制的元素范围。



1. includes() 判断数组中是否存在该元素，参数：查找的值，起始位置。可以替换indexOf()。

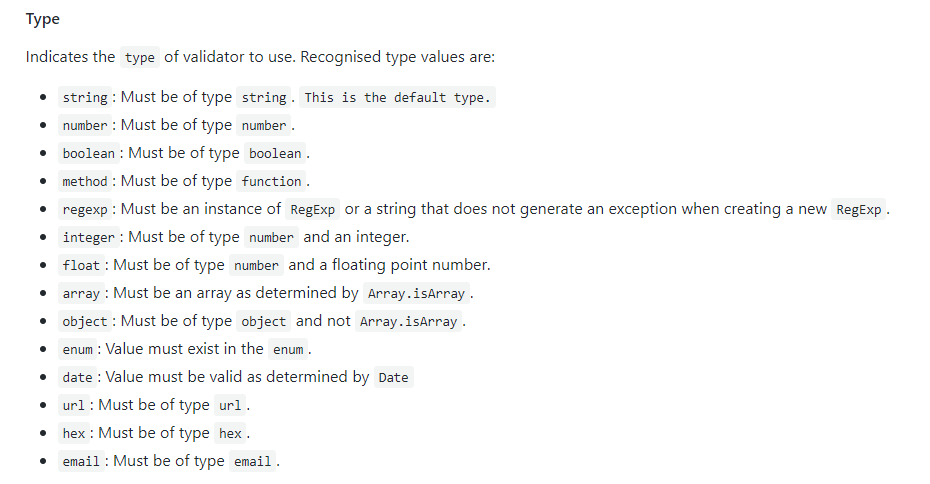


## form表单校验

<https://github.com/yiminghe/async-validator>

type：

eg： { required: true, message: "请输入密码", type:'string' },

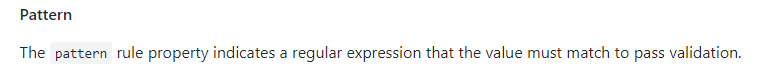


pattern：

eg:：newPassword: [

{ pattern: /^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]{6,32}$/,

message: "请输入6~32位的数字+字母的密码，谢谢！" }],



required：

The required rule property indicates that the field must exist on the source object being validated.

eg：{ required: true, message: "请输入密码", type:'string' },

## replace

replace()用于在字符串中用一些字符替换指定字符，或替换一个与正则表达式匹配的子串。返回一个替换后的字符串，不改变原字符串。



## require()基本用法

当Node遇到require(X)时，处理顺序

1. 如果是内置模块，比如require(‘http’)

返回该模块，不再执行。

1. 如果X是以”./”,”/”,”../”开头

2.1 根据X所在的父模块，确定X的绝对路径。

2.2 将X当成文件，依次查下面文件，只要有一个存在就返回该文件，不再执行。X，X.js，X.json, X.node

2.3 将X当成目录，依次查下面文件，只要有一个存在就返回该文件，不再执行。X/package.json，X/index.js，X/index.json, X/index.node

## readFile() readFileSync()

readFile(url, function()) 可以是绝对路径也可以是相对路径，相对路径是相对于当前进程的路径（process.cwd()）, function(err, buffer)参数，第一个是发生错误时的错误对象，第二个是代表文件内容的Buffer实例。

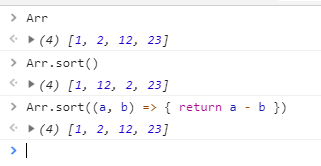
readFileSync(filename, ‘utf-8’) 用于同步读取文件，返回一个字符串。

如果第二个参数不指定编码，则返回一个Buffer实例，反之返回一个字符串。

## sort()

arrObject.sort(sortby)对数组元素进行排序， 在原数组上进行排序， 不生成副本。

如果没有参数，将按照字母顺序对数组中的元素进行排序。如果想按照其他标准排序，需要提供比较函数。



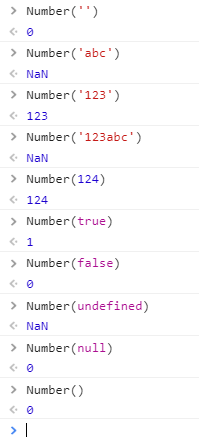
## 数据类型转换

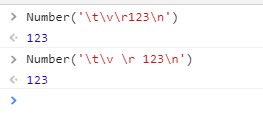
JS是一种动态类型语言，变量没有类型限制，可以随时赋予任何值。但各种运算符对数据类型是有要求的，如果运算符发现运算子的类型与预期不符，则会自动转换类型。’4’ – ‘3’ // 1

### 强制转换，Number(), String(), Boolean()

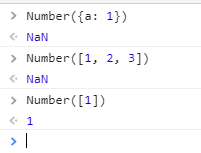
Number()

原始类型值



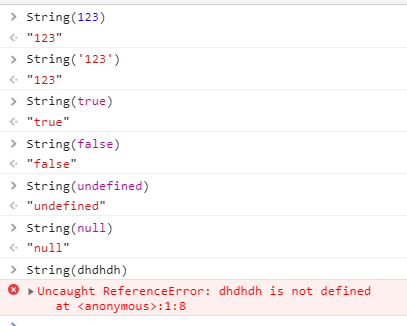


对象： 参数是对象时返回NaN，除非是包含单个数值的数组

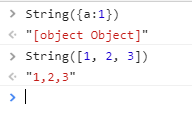


String()

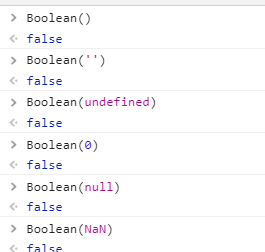
原始类型值



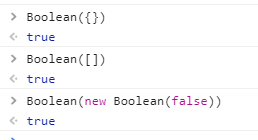
参数如果是对象，返回一个类型字符串，如果是数组，返回该数组的字符串形式。



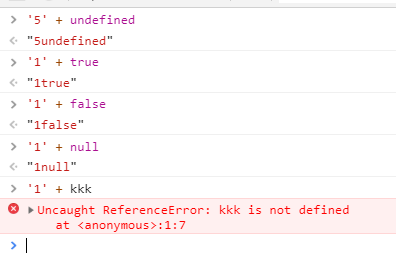
Boolean() 可以转换任意类型的值



所有对象包括空对象的布尔值都是true

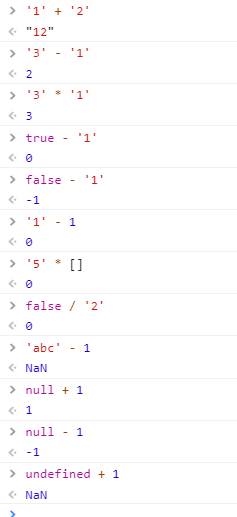


自动转换为字符串



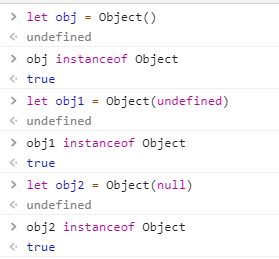
自动转换为数值

除了加法运算符(+)有可能转为字符串，其他运算符都会把运算子转换为数值

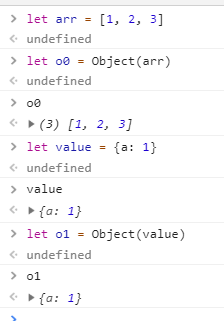


## Object()

Object本身是一个函数，可以当做工具方法使用，将任意值转为对象。常用于保证某个值一定是对象。



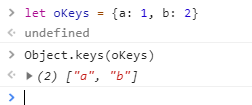
如果Object方法的参数是一个对象，返回该对象，不用转换。



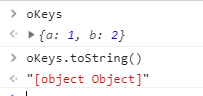
Object(value) 将value转换为对象

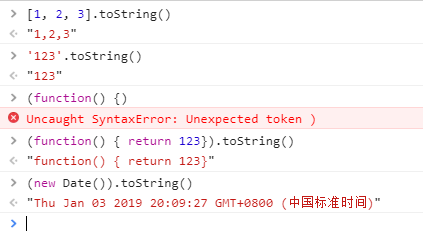
new Object(value) 新生成一个对象

Object.keys() 方法的参数是一个对象，返回一个数组，数组成员是对象自身（不是继承的）的所有属性名。



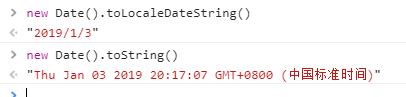
Object.toString() 返回一个对象的字符串形式，默认情况下返回类型字符串。



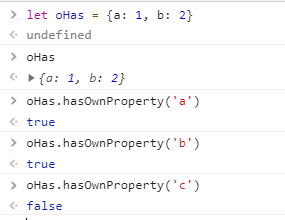




Object.toLocalString() 返回本地的字符串形式

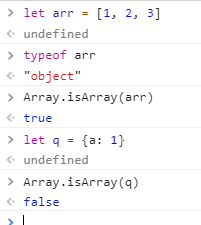


Object.hasOwnProperty() 接收一个字符串作为参数，返回一个布尔值，表示该实例对象是否具有该属性。

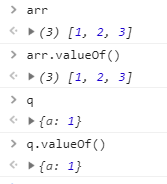


## 数组Array对象

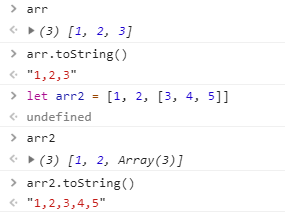
Array.isArray() 返回一个布尔值，表示参数是否为数组。



valueOf() 所有对象都拥有的方法，表示对该对象求值。数组的valueOf方法返回数组本身。



toString()



push() pop() 都会改变原数组。

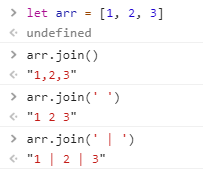
空数组使用pop()不会报错，返回undefined。



shift() 删除数组的第一个元素 并返回该元素 会改变原数组

unshift() 用于在数组的第一个位置添加元素 并返回新数组的长度 改变原数组

join() 以指定参数作为分隔符，将所有数组成员连接为一个字符串返回。如果不提供参数，默认以逗号分隔。



如果数组成员是undefined，null或者空位，会被转换为空字符串。



concat() 用于多个数组的合并，将新数组的成员添加到原数组的后面，然后返回一个新的数组，原数组不变。



## 表格 min-width 解决表格宽度自适应问题

对应列的最小宽度，与 width 的区别是 width 是固定的，min-width 会把剩余宽度按比例分配给设置了 min-width 的列

## **render-header**

列标题 Label 区域渲染使用的Function Function(h, { column, $index })

return h('span', [

h('span', {}, ['操作']),

h('span', {

style: 'margin-left: 10px;',

on: {

'click': this.modifyHeader

},

attrs: {

title: '编辑表头'

}

}, [

h('i', {

class: 'el-icon-setting'

})

])

])

## 设置小圆点

.equip-status {

display: inline-block; // 这个必须 不然不显示

width: 8px;

height: 8px;

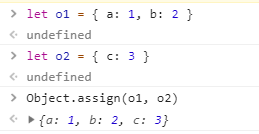
border-radius: 50%;

background: rebeccapurple;

}

## Object.assign

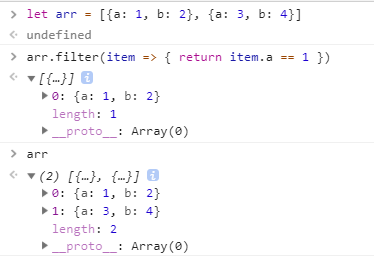
将所有可枚举属性的值从一个或多个源对象复制到目标对象，返回目标对象。



## Array filter()

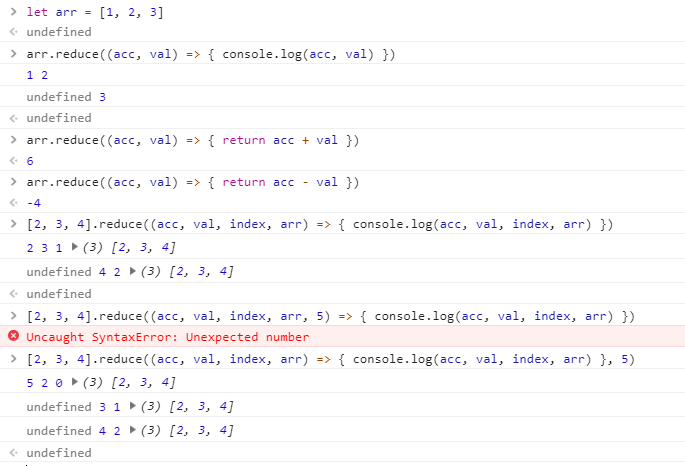
不会对空数组进行检测

不会改变原数组



## reduce

对数组的每一个元素执行自定义的reducer函数（升序执行），将结果汇总单个返回值。



## Cannot read property 'indexOf' of undefined 报错信息

新增和编辑公用一个表单弹窗，新增弹窗弹出后关闭，再点击编辑，弹窗弹出后关闭，报错。

this.$refs.addTaskForm.$refs.addForm.resetFields()

**原因**：弹窗表单中有一项form-item没有使用prop属性，导致resetFields()遍历时报错。

**解决方法**：在该form-item加上prop属性。使用resetFields()清空表单，前提是每一项表单都有prop属性。

## 转换Number类型

加法运算符 + ，将一个string类型的变量转回为number类型

let int = ‘15’

int = +int

console.log(int) // 15

console.log(typeof int) // ‘number’

## 快速Float转Integer 向下取整

| 位或运算符 该操作是直接删除小数点后的内容，将浮点数截断为整数

console.log(23.6 | 0) // 23

console.log(-23.6 | 0) // -23

console.log(1555 / 10 | 0) // 155

console.log(1555 / 100 | 0) // 15

let a = ~~2.33 // 2

let b = 2.33 >> 0 // 2

## 获取数组中最后的元素

数组方法slice()可以接受负整数，它将从数组的末尾开始截取数值。

let array = [1,2,3,4,5]

array.slice(-1) // [5]

array.slice(-2) // [4,5]

## 日历

[…Array(7).keys].map(days => new Date(Date.now() – 24 \* 60 \* 60 \* 1000 \* days))

## 生成随机ID

// 生成长度为11的随机字母数字字符串

Math.random().toString(36).substring(2)

## 获取URL查询参数

?foo=bar&baz=bing => {foo: bar, baz: bing}

let q = {}

location.search.replace(/([^?&=]+)=([^&]+)/g, (\_, k, v) => q[k] = v)

## 数组去重

let arr = […new Set([1, 2, 3, 2])]

arr = [1, 2, 3]

## 过滤空值

let res = ['', 1, 2, 0, undefined].filter(Boolean)

res = [1, 2]

## 创建空对象

let ob = Object.create(null)

## 合并对象

…延展操作符

let person = {a: 1}

let hello = {b: 2}

let oo = {…person, …hello}

oo = {a: 1, b: 2}