

01-let关键字

笔记本: ES6

创建时间: 2022/4/11 14:53

更新时间: 2022/4/12 9:48

作者: r37qboxc

let a;
let a,b,c;

```
script>  
//声明变量  
let a;  
let b,c,d;  
let e = 100;  
let f = 521, g = 'iloveyou', h = [];
```

let 特性

- 1: 不能重复使用
- 2: 块级作用域(只在代码块有效, 代码块外无效)
- 3: 不存在变量提升(不能在声明之前使用变量)
- 4: 不影响模块链(区别于2)

```
//4. 不影响作用域链  
{  
  let school = '尚硅谷';  
  function fn(){  
    console.log(school);  
  }  
  fn();  
}
```

可以打印出来

作用域: (全局, 函数, eval(es5))

var是全局的, let是只在模块作用

02-const声明常量

笔记本: ES6

创建时间: 2022/4/11 16:23

更新时间: 2022/4/11 16:28

作者: r37qboxc

1. 一定要赋初始值
2. 一般常量用大写
3. 常量的值不能修改
4. 块级作用域
5. 对于对数组和对象做出的修改, 不算对常量的修改, 不会报错

```
const TEAM = ['UZI', 'MXLG', 'Ming', 'Letme'];  
// TEAM.push('Meiko');    不会报错  
TEAM = 100;               会报错
```

03-变量结构赋值

笔记本: ES6

创建时间: 2022/4/11 16:33

更新时间: 2022/4/11 16:38

作者: r37qboxc

允许按照一定模式从数组和对象中**提取**值, 对变量进行赋值

数组结构

```
//ES6 允许按照一定模式从数组和对象中提取值，对变量进行赋值，
//这被称为解构赋值。
//1. 数组的结构
const F4 = ['小沈阳', '刘能', '赵四', '宋小宝'];
let [xiao, liu, zhao, song] = F4;

console.log(xiao);
console.log(liu);
console.log(zhao);
console.log(song);
```

对象结构

```
//2. 对象的解构
const zhao = {
  name: '赵本山',
  age: '不详',
  xiaopin: function(){
    console.log("我可以演小品");
  }
};

let {name, age, xiaopin} = zhao;
console.log(name);
console.log(age);
console.log(xiaopin);

xiaopin();
```



04-模板字符串&&简化对象的声明

笔记本: ES6

创建时间: 2022/4/11 16:39

更新时间: 2022/4/11 16:48

作者: r37qboxc

1. ES6引入单引号字符串

```
9 script>
0 // ES6 引入新的声明字符串的方式 『`』 '' ""
1 //1. 声明
2 // let str = `我也是一个字符串哦!`;
3 // console.log(str, typeof str);
4
5 //2. 内容中可以直接出现换行符
6 let str = `


7     <li>沈腾</li>
8     <li>玛丽</li>
9     <li>魏翔</li>
0     <li>艾伦</li>
1     </ul>`;
2
```

如果用单引号就需要用 + 号连接换行

2. 可以进行变量拼接: \${} 格式

简化对象的声明

```
//ES6 允许在大括号里面，直接写入变量和函数，作为对象的属性和方法。
//这样的书写更加简洁
let name = '尚硅谷';
let change = function(){
    console.log('我们可以改变你!!');
}

const school = {
    name,
    change,
    improve(){
        console.log("我们可以提高你的技能");
    }
}
```



05-箭头函数与函数默认值相关

笔记本: ES6

创建时间: 2022/4/11 16:48

更新时间: 2022/4/13 14:52

作者: r37qboxc

let fn = () => {}

1. this是静态的, 始终指向声明时的状态(什么时候声明, this就是什么)
2. 箭头函数不能作为构造函数

```
//2. 不能作为构造实例化对象
// let Person = (name, age) => {
//     this.name = name;
//     this.age = age;
// }
// let me = new Person('xiao',30);
// console.log(me);
```

3. 不能使用arguments 变量

```
//3. 不能使用 arguments 变量
// let fn = () => {
//     console.log(arguments);
// }
// fn(1,2,3);
```

4. 箭头函数的简写

```
//4. 箭头函数的简写
//1) 省略小括号, 当形参有且只有一个的时候
// let add = n => {
//     return n + n;
// }
// console.log(add(9));
//2) 省略花括号, 当代码体只有一条语句的时候, 此时 return 必须省
// 而且语句的执行结果就是函数的返回值
let pow = (n) => n*n;
```

例子

```
let ad = document.getElementById('ad');

//绑定事件
ad.addEventListener("click", function(){
    //保存 this 的值
    // let _this = this;
    //定时器
    setTimeout(() => {
        //修改背景颜色 this
        // console.log(this);
        // this.style.background = 'pink';
        this.style.background = 'pink';
    }, 2000);
});
```

此处的箭头函数 指向 ad 也就是在声明的时候定义的

例子2

```
//需求-2 从数组中返回偶数的元素
const arr = [1,6,9,10,100,25];
// const result = arr.filter(function(item){
//     if(item % 2 === 0){
//         return true;
//     }else{
//         return false;
//     }
// });

const result = arr.filter(item => item % 2 === 0);
```

总结:

使用场景: 适合于this无关的设置, 特别不使用与this相关的, 特别DOM, 不然就会指向外层

```
// 箭头函数适合与 this 无关的回调. 定时器, 数组的方法回调
// 箭头函数不适合与 this 有关的回调. 事件回调, 对象的方法

{
    name: '尚硅谷',
    getName: () => {
        this.name;
    }
}
```

如果是 function 写法, this 指的是对象
如果是箭头函数, 就指到外层去了

函数参数的默认值赋值

1. 赋予初始值, 如果传了就赋值, 没有就用默认的

```
//ES6 允许给函数参数赋值初始值
//1. 形参初始值
function add(a,b,c=10) {
    return a + b + c;
}
let result = add(1,2,3);
console.log(result);
```

2. 与结构赋值相结合(也可赋予初始值)

```
//2. 与解构赋值结合
function connect({host, username, password, port}){
    console.log(host);
    console.log(username);
    console.log(password);
    console.log(port);
}
connect({
    host: 'localhost',
    username: 'root',
    password: 'root',
```


06-rest参数 和 扩展符的使用

笔记本: ES6

创建时间: 2022/4/11 17:21

更新时间: 2022/4/13 9:28

作者: r37qboxc

用来获取函数的实参, 用来代替arguments

运用3个.

function(..args){}

```
// ES6 引入 rest 参数, 用于获取函数的实参, 用来代替 arguments
// ES5 获取实参的方式
// function date(){
//     console.log(arguments);
// }
// date('白芷','阿娇','思慧');

// rest 参数
function date(...args){
    console.log(args); // filter some every map
}
date('阿娇','柏芝','思慧');
```

他是一个数组, 可以用数组的api

rest参数必须放到最后!!!

扩展符

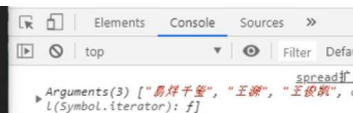
把 ...用在数组 就是把数组扩展开来

eg

```
const tfboys = ['易烊千玺','王源','王俊凯'];
// => '易烊千玺','王源','王俊凯'

// 声明一个函数
function chunwan(){
    console.log(arguments);
}

chunwan(...tfboys); // chunwan('易烊千玺','王源','王俊凯')
```



如果不加三个点, arguments就是一个值存的数组

扩展运算符的运用

1. 拼接

```
const kuaizi = ['王太利','肖央'];
const fenghuang = ['曾毅','玲花'];
// const zuixuanxiaopingguo = kuaizi.concat(fenghuang);
const zuixuanxiaopingguo = [...kuaizi, ...fenghuang];
console.log(zuixuanxiaopingguo);
```



2. 克隆

//2. 数组的克隆

```
// const sanzhihua = ['E', 'G', 'M'];  
// const sanyecao = [...sanzhihua];  
// console.log(sanyecao);
```

3. 将数组对象(含有原型的_proto_)转为一个真正的数组[]

//3. 将伪数组转为真正的数组

```
const divs = document.querySelectorAll('div');  
const divArr = [...divs];  
console.log(divArr);
```

3里面有原型元素_proto_

07-Symbol - git

笔记本: ES6

创建时间: 2022/4/11 17:45

更新时间: 2022/4/12 16:41

作者: r37qboxc

第7种数据类型(其他number, string, Boolean, null, object, undefined)

2.9.1.Symbol 基本使用

ES6 引入了一种新的原始数据类型 Symbol，表示独一无二的值。它是 JavaScript 语言的第七种数据类型，是一种类似于字符串的数据类型。

Symbol 特点

- 1) Symbol 的值是唯一的，用来解决命名冲突的问题
- 2) Symbol 值不能与其他数据进行运算
- 3) Symbol 定义的对象属性不能使用 for...in 循环遍历，但是可以使用 Reflect.ownKeys 来获取对象的所有键名

Symbol 不能运算, 拼接等等

```
//创建Symbol
let s = Symbol();
// console.log(s, typeof s);
let s2 = Symbol('尚硅谷');
let s3 = Symbol('尚硅谷');
//Symbol.for 创建
let s4 = Symbol.for('尚硅谷');
let s5 = Symbol.for('尚硅谷');
```

通过for创建, 参数是相等的 故 s4===s5 输出为 true

第二种方式给对象添加属性

```
//
let youxi = {
  name: "狼人杀",
  [Symbol('say')]: function(){
    console.log("我可以发言")
  },
  [Symbol('zibao')]: function(){
    console.log('我可以自爆');
  }
}
```

Symbol控制对象的表现

isConcatSpreadable 是否可展开

The image shows a code editor on the left and a browser console on the right. The code in the editor is as follows:

```
// let o = {};  
  
// console.log(o instanceof Person);  
  
const arr = [1,2,3];  
const arr2 = [4,5,6];  
arr2[Symbol.isConcatSpreadable] = false;  
console.log(arr.concat(arr2));  
</script>  
</body>
```

The browser console on the right displays the output of the code. It shows an array of length 4, which is the result of concatenating the two input arrays. The third element of the array is an array containing the elements 4, 5, 6, and a Symbol object. The Symbol object is of type Symbol(Symbol.isConcatSpreadable) and its value is false. A red arrow points from the `arr2` variable in the code to the Symbol object in the console output.

Console Output:

```
(4) [1, 2, 3, Array(3)]  
  0: 1  
  1: 2  
  2: 3  
  3: (3) [4, 5, 6, Symbol(Symbol.isConcatSpreadable):  
    length: 4  
    __proto__: Array(0)]
```

08-迭代器 - git

笔记本: ES6

创建时间: 2022/4/12 10:02

更新时间: 2022/4/12 16:41

作者: r37qboxc

1) ES6 创造了一种新的遍历命令 for...of 循环, Iterator 接口主要供 for...of 消费

2) 原生具备 iterator 接口的数据(可用 for of 遍历)

- a) Array
- b) Arguments
- c) Set
- d) Map
- e) String
- f) TypedArray
- g) NodeList

3) 工作原理

- a) 创建一个指针对象, 指向当前数据结构的起始位置
- b) 第一次调用对象的 next 方法, 指针自动指向数据结构的第一个成员
- c) 接下来不断调用 next 方法, 指针一直往后移动, 直到指向最后一个成员
- d) 每调用 next 方法返回一个包含 value 和 done 属性的对象

作用: 自定义遍历数组 可参考demo中的 iterator.html

```
//自定义遍历数组

const store = {
  name: 'my_store',
  goods: ['pen', 'notes', 'A4 paper'],

  //是一个迭代器, 而不是一个方法
  [Symbol.iterator]() {
    let index = 0;
    return {
      //因为使用箭头函数, 所以this直接拿的上一层的, 如果使用function函数 就要用_this去储存上一层的this
      next: () => {
        return {value: this.goods[index++], done: this.goods.length < index};
      }
    };
  }
};

let s1 = store;
for(let v of store){
  console.log(v);
}
```


09-生成器 - git

笔记本: ES6

创建时间: 2022/4/12 11:06

更新时间: 2022/4/12 16:41

作者: r37qboxc

生成器主要作用: 异步编程

以前使用的异步函数: Ajax MongoDB, timeont()

yield: 函数代码的分割符

他是个迭代器对象, 以下产生4块代码

```
//函数代码的分隔符
function * gen(){
    yield '一只没有耳朵';
    yield '一只没有尾部';
    yield '真奇怪';
}

let iterator = gen();
```



需要用 `let iterator = gen();`
`iterator.next();`

生成器函数参数

可以节省空间变量

10-Promise - git

笔记本: ES6

创建时间: 2022/4/12 13:46

更新时间: 2022/4/14 11:17

作者: r37qboxc

异步编程的新解决方案

Promise是一个 构造函数用来封装异步操作的成功或者失败的结果

其中涉及到then方法

相比传统的调用方法, 减少了缩进(横向)

//代码案例中体现出来

Promise.prototype的then返回参数

then返回的意思一个 Promise 对象, 但是他的 status是根据回调来决定的

例如

Promise里面调用 resolve(data), 那么 PromiseStatus就是 resolve

而返回的值(PromiseValue) 是在 then里面 return设置的值, 如果无 则为 undefined

```
▼ Promise {<pending>} ⓘ  
  ► __proto__: Promise  
    [[PromiseStatus]]: "resolved"  
    [[PromiseValue]]: undefined
```

```
const result = p.then(value => {  
  console.log(value);  
  return 123;  
}, reason=>{  
  console.warn(reason);  
});
```

因为 then方法的返回对象是 Promise 那么 then是可以进行链式调用的(类似linked node)

```
//链式调用  
p.then(value=>{}, reason=>{}).then(value=>{}, reason=>{})
```

Promise catch方法


```

const p = new Promise((resolve, reject)=>{
  setTimeout(()=>{
    //设置 p 对象的状态为失败，并设置失败的值
    reject("出错啦!");
  }, 1000)
});

// p.then(function(value){}, function(reason){
//   console.error(reason);
// });

p.catch(function(reason){
  console.warn(reason);
});

```



ES11中

allSettled([])可以放2个数组, 但都是以成功的Promise返回

```

//声明两个promise对象
const p1 = new Promise((resolve, reject)=>{
  setTimeout(()=>{
    resolve('商品数据 - 1');
  },1000)
});

const p2 = new Promise((resolve, reject)=>{
  setTimeout(()=>{
    // resolve('商品数据 - 2');
    reject('出错啦!');
  },1000)
});

//调用 allsettled 方法 any
const result = Promise.allSettled([p1, p2]);

```

11-set和 map

笔记本: ES6

创建时间: 2022/4/12 16:36

更新时间: 2022/4/12 16:56

作者: r37qboxc


ES6 提供了新的数据结构 Set (集合)。它类似于数组,但成员的值都是唯一的,集合实现了 `Iterator` 接口,所以可以使用『扩展运算符』和『`for...of...`』进行遍历,集合的属性和方法:

- 1) `size` 返回集合的元素个数
- 2) `add` 增加一个新元素,返回当前集合
- 3) `delete` 删除元素,返回 `boolean` 值
- 4) `has` 检测集合中是否包含某个元素,返回 `boolean` 值

Set 本质上是一个 对象 `let s = new Set();` 类似数组


因为 唯一性, 所以自动去重

```
let s = new Set();
let s2 = new Set(['大事儿', '小事儿', '好事儿', '坏事儿', '小事儿']);
console.log(s2);
```



交集

```
let arr = [1,2,3,4,5,4,3,2,1];
//1. 数组去重
// let result = [...new Set(arr)];
// console.log(result);
//2. 交集
let arr2 = [4,5,6,5,1];
let result = [...new Set(arr)].filter(item => {
  let s2 = new Set(arr2);
  if(s2.has(item)){
    return true;
  }else{
    return false;
  }
});
```



并集

```
//3. 并集
let union = [...new Set([...arr, ...arr2])];
console.log(union);
```

差集(以arr 为主体求与 arr2的差集,交集取反)

```
//4. 差集
let diff = [...new Set(arr)].filter(item => !(new Set(arr2).has(item)));
console.log(diff);
```

Map(升级版的set)

索引和值 都可以是 字符串, 对象, 数组

```
let m = new Map();

//添加元素
m.set('name', '尚硅谷');
m.set('change', function(){
    console.log("我们可以改变你!!");
});
let key = {
    school : 'ATGUGU'
};
m.set(key, ['北京', '上海', '深圳']);
```

map的方法

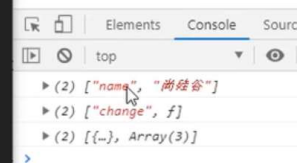
```
//size
// console.log(m.size);

//删除
// m.delete('name');

//获取
// console.log(m.get('change'));
// console.log(m.get(key));

//清空
// m.clear();

//遍历
for(let v of m){
    console.log(v);
}
```



12 - class - git

笔记本: ES6

创建时间: 2022/4/12 17:09

更新时间: 2022/4/14 11:07

作者: r37qboxc

ES5与ES6类集成关系的区别(原形与原型链)

ES5中

```
//ES 5 类继承 及构造方法 原型链实现
function Phone(brand, price){
  this.brand = brand;
  this.price = price;
}
Phone.prototype.callPhone = () =>{
  console.log("callPhone method");
}

function SmartPhone(brand, price, color){
  Phone.call(this, brand, price);
  this.color = color;
}
//设置子原形
SmartPhone.prototype = new Phone;
SmartPhone.prototype.constructor = SmartPhone;

SmartPhone.prototype.play = () =>{
  console.log('play method')
}

const newPhone = new SmartPhone('apple', 5000, 'black');
console.log(newPhone);
```

ES6(java化)

```
//ES6 类继承与构造方法
class Phone2{
  constructor(brand, price){
    this.brand = brand;
    this.price = price;
  }

  callPhone(){
    console.log("callPhone method");
  }
}

class SmartPhone2 extends Phone2{
  constructor(brand, price, color){
    super(brand, price);
    this.color = color;
  }

  play(){
    console.log('play method')
  }
}

const newPhone2 = new SmartPhone2('xiaomi', 800, 'color');
console.log(newPhone2);
```

使用get set时需要注意的点
get 方法 如果回传 this.price 会无限递归, 因为
this.price也是调用的get

```
get price(){
  console.log(`${this._price}被获取`)
  //此处注意 如果使用 this.price 会出现无限递归的情况, 在vue中已经解决, 因为this.price是调用的自身的 price 会无限调用
  return this._price;
}
//在构造器有该属性时 默认调用1次
set price(value){
  console.log(`${value}被设置成新的价格`)
  this._price = value;
}
```

ES11 私有属性

```
//私有属性
#age;
... #weight;
```

13-数值的扩展

笔记本: ES6

创建时间: 2022/4/13 10:28

更新时间: 2022/4/13 10:51

作者: r37qboxc

数值最小精度

```
//0. Number.EPSILON 是 JavaScript 表示的最小精度
//EPSILON 属性的值接近于 2.2204460492503130808472633361816E-16
// function equal(a, b){
//     if(Math.abs(a-b) < Number.EPSILON){
//         return true;
//     }else{
//         return false;
//     }
// }
// console.log(0.1 + 0.2 === 0.3);           false
// console.log(equal(0.1 + 0.2, 0.3))         true
```

进制数值 0b=>二进制, 0o=>八进制, 0x=>16进制

```
//1. 二进制和八进制
let b = 0b1010;    10(8+2)
let o = 0o777;      511
let d = 100;
let x = 0xff;        255
```

检测一个数值是否为有限数

```
//2. Number.isFinite 检测一个数值是否为有限数
// console.log(Number.isFinite(100));           true
// console.log(Number.isFinite(100/0));          false
// console.log(Number.isFinite(Infinity));       false
```

检测是否为NaN

```
//3. Number.isNaN 检测一个数值是否为 NaN
console.log(Number.isNaN(123));
```

返回为false

转整数或者转浮点数(可截断字符串)


```
//4. Number.parseInt Number.parseFloat字符串转整数
// console.log(Number.parseInt('5211314love'));
// console.log(Number.parseFloat('3.1415926神奇'));
```

抹掉小数

```
//6. Math.trunc 将数字的小数部分抹掉
// console.log(Math.trunc(3.5));
```

输出 3

判断正负数,0

```
//7. Math.sign 判断一个数到底为正数 负数 还是零
console.log(Math.sign(100));      1
console.log(Math.sign(0));        0
console.log(Math.sign(-20000));   -1
```


14-Object的三个扩展方法

笔记本: ES6

创建时间: 2022/4/13 10:56

更新时间: 2022/4/13 11:02

作者: r37qboxc


• Object的比较 Object.is()

```
//1. Object.is 判断 (method) ObjectConstructor.is(value1: any, value2: any): boolean
// console.log(Object.is(NaN, NaN)); // === true
console.log(Object.is(NaN, NaN)); // === true
console.log(NaN === NaN); // === false
// 任何类型用 === 比较NaN都是false
```

• Object合并(后覆盖前)

```
//2. Object.assign 对象的合并
const config1 = {
  host: 'localhost',
  port: 3306,
  name: 'root',
  pass: 'root',
  test: 'test'
};

const config2 = {
  host: 'http://atguigu.com',
  port: 33060,
  name: 'atguigu.com',
  pass: 'iloveyou'
};
```




```
{host: "http://atguigu.com", port: 33060, name: "atguigu.com", pass: "iloveyou", test: "test"}
```

• Object的原型设置

```
//3. Object.setPrototypeOf 设置原型对象 Object.getPrototypeOf
const school = {
  name: '尚硅谷'
};
const cities = {
  xiaoqu: ['北京', '上海', '深圳']
};

Object.setPrototypeOf(school, cities);
console.log(Object.getPrototypeOf(school));
```



```
{name: "尚硅谷"}
  name: "尚硅谷"
  __proto__: {xiaoqu: (3) ["北京", "上海", "深圳"]}
    constructor: f Object()
    hasOwnProperty: f hasOwnProperty()
    isPrototypeOf: f isPrototypeOf()
    propertyIsEnumerable: f propertyIsEnumerable()
    toLocaleString: f toLocaleString()
    toString: f toString()
    valueOf: f valueOf()
    __defineGetter__: f __defineGetter__()
    __defineSetter__: f __defineSetter__()
    __lookupGetter__: f __lookupGetter__()
```

15-模块化-git

笔记本: ES6

创建时间: 2022/4/13 11:04

更新时间: 2022/4/14 10:34

作者: r37qboxc

1.25. 模块化

模块化是指将一个大的程序文件,拆分成许多小的文件,然后将小文件组合起来。

1.25.1. 模块化的好处

模块化的优势有以下几点:

- 1) 防止命名冲突
- 2) 代码复用
- 3) 高维护性

ES6之前的模块化规范

1.25.2. 模块化规范产品

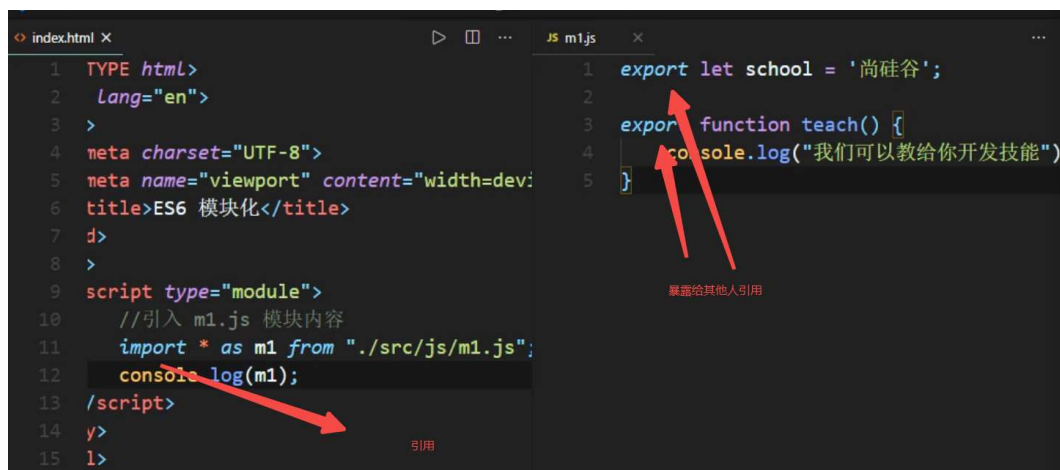
ES6 之前的模块化规范有:

- 1) CommonJS => NodeJS、Browserify
- 2) AMD => requireJS
- 3) CMD => seaJS

ES6之后的模块化

Export: 规定模块的对外接口, 向外边提供的方法, 变量
(被引用模块)

Import: 用于输入其他模块提供的功能



本地浏览器因为file协议chrome会出现跨域问题, vs code需要装live server插件
一般暴露(如果v 没有被 export, 那么getPrice是拿不到 v的而出现not define)

```
//一般暴露
export let v = 123;
export function getPrice(){
  return v;
};
```

统一暴露

```
//统一暴露
export {name, getName}
let name = 'Chris';
function getName(){
  return name;
}
```

默认暴露

```
//默认暴露, 以对象形式居多
export default {
  department: 'CR',
  getPower: (val) => val * val
}
```

模块化(类似代理模式)

16-Babel

笔记本: ES6

创建时间: 2022/4/13 14:28

更新时间: 2022/4/13 14:32

作者: r37qboxc

将新的写法转化为ES5以适应某些不兼容的浏览器(IE)

https://www.bilibili.com/video/BV1uK411H7on?p=46&spm_id_from=pageDriver

ES8-async-git

笔记本: ES6

创建时间: 2022/4/14 9:07

更新时间: 2022/4/14 10:32

作者: Chris吕

在函数前面加 async

无论函数内回传的是什么, 总是返回一个Promise对象
返回的值或者状态, 由async里面的函数决定 resolve, reject, value

```
</script>
    async function fn(){
      return 'result';
    }

    //结果总是一个 promise对象
    console.log(fn())
```

```
▼ Promise {<fulfilled>: 'result'} ⓘ
  ► [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "result"
```

因为是Promise对象, 所以都适用 then和catch函数

```
    async function fn(){
      return new Promise((resolve, reject) => {
        reject("warn reason");
      })
    }

    //结果总是一个 promise对象
    fn().then(val => {}, reason => {
      console.warn(reason);
    })
  </script>
```

ES8-await-git

笔记本: ES6

创建时间: 2022/4/14 9:07

更新时间: 2022/4/14 10:32

作者: Chris吕

async 和 await结合就把异步函数变得跟同步函数一样

1. await必须写在 async函数里面
2. await右侧的表达式一般为 Promise对象
3. await返回的值一般是 resolve 也就是Promise成功的值
4. await的Promise失败值(reject) 会抛出异常通过 try catch 处理

```
//await...总会返回一个Promise成功的 值, 失败用try catch处理
const p = new Promise((resolve, reject) => {
  // resolve('success');
  reject('fail');
})

async function fc(){
  try{
    let result = await p;
    console.log(result);
  }catch(reason){
    console.log(reason);
  }
}

fc();

</script>
```

ES11-可选链操作符-git

笔记本: ES6

创建时间: 2022/4/14 11:26

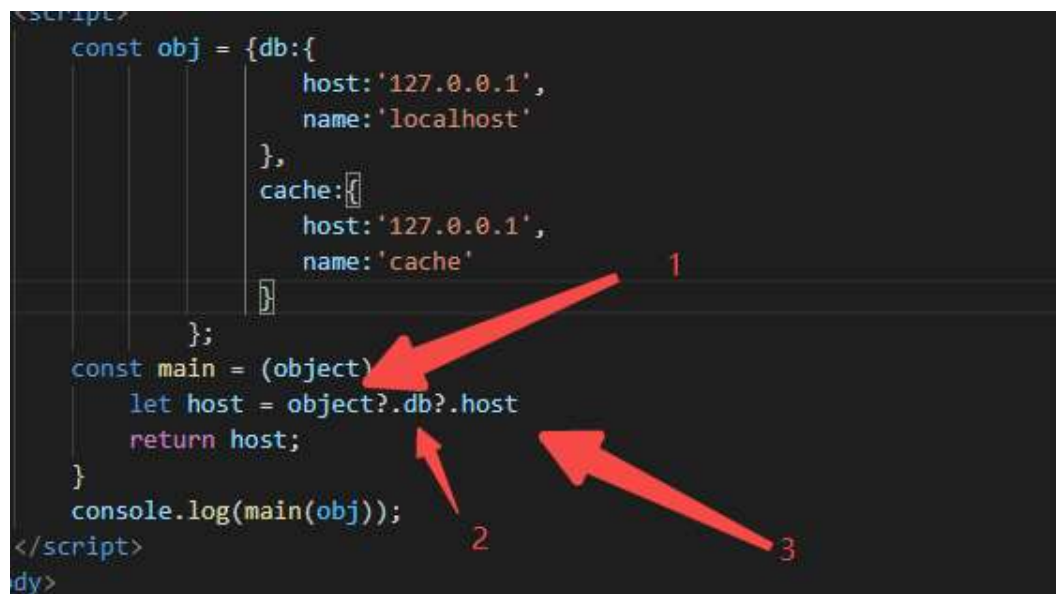
更新时间: 2022/4/14 11:40

作者: Chris吕

? 逐级验证

```
<script>
  const obj = {db:{
    host:'127.0.0.1',
    name:'localhost'
  },
  cache:{
    host:'127.0.0.1',
    name:'cache'
  }
};

const main = (object)
  let host = object?.db?.host
  return host;
}
console.log(main(obj));
</script>
</script>
```



ES11-动态-import-git

笔记本: ES6

创建时间: 2022/4/14 11:40

更新时间: 2022/4/14 13:56

作者: Chris吕

import的返回函数是一个 Promise对象 那么就可以用 then来捕捉成功的结果

```
1  const e = document.getElementById('btn');
2
3  e.onclick = ()=>{
4    //测试点击事件
5
6    //在使用的時候 才 import
7    import('./clickEvent.js').then(module => {
8      module.hello();
9    })
10 }
11
12
13
14
```

ES11-BigInt计算

笔记本: ES6

创建时间: 2022/4/14 13:59

更新时间: 2022/4/14 14:02

作者: Chris吕

普通整数值运算 出错

```
//大数值运算
let max = Number.MAX_SAFE_INTEGER;
console.log(max);
console.log(max + 1);
console.log(max + 2);
```

输出:



BigInt(): 注意 不能和普通整数进行计算

```
console.log(BigInt(max))
console.log(BigInt(max) + BigInt(1))
console.log(BigInt(max) + BigInt(2))
```