ES6 入门

ECMAScript 6（以下简称ES6）是JavaScript语言的下一代标准，已经在2015年6月正式发布了。它的目标，是使得JavaScript语言可以用来编写复杂的大型应用程序，成为企业级开发语言。

1. ECMAScript和JavaScript的关系

一个常见的问题是，ECMAScript和JavaScript到底是什么关系？

要讲清楚这个问题，需要回顾历史。1996年11月，JavaScript的创造者Netscape公司，决定将JavaScript提交给国际标准化组织ECMA，希望这种语言能够成为国际标准。次年，ECMA发布第一版，规定了浏览器脚本语言的标准，并将这种语言称为ECMAScript，这个版本就是1.0版。

该标准从一开始就是针对JavaScript语言制定的，但是之所以不叫JavaScript，有两个原因。一是商标，Java是Sun公司的商标，根据授权协议，只有Netscape公司可以合法地使用JavaScript这个名字，且JavaScript本身也已经被Netscape公司注册为商标。二是想体现这门语言的制定者是ECMA，这样有利于保证这门语言的开放性和中立性。因此，ECMAScript和JavaScript的关系是，前者是后者的规则，后者是前者的一种实现。

2. ECMAScript的历史

ECMAScript 1.0是1997年发布的，接下来的两年，连续发布了ECMAScript 2.0（1998年6月）和ECMAScript 3.0（1999年12月）。3.0版是一个巨大的成功，在业界得到广泛支持，成为通行标准，奠定了JavaScript语言的基本语法，以后的版本完全继承。

2000年，ECMAScript 4.0开始酝酿。这个版本最后没有通过，但是它的大部分内容被ES6继承了。因此，ES6制定的起点其实是2000年。

为什么ES4没有通过呢？因为这个版本太激进了，对ES3做了彻底升级，导致标准委员会的一些成员不愿意接受。ECMA的第39号技术专家委员会（Technical Committee 39，简称TC39）负责制订ECMAScript标准，成员包括Microsoft、Mozilla、Google等大公司。

2007年10月，ECMAScript 4.0版草案发布，本来预计次年8月发布正式版本。但是，各方对于是否通过这个标准，发生了严重分歧。以Yahoo、Microsoft、Google为首的大公司，反对JavaScript的大幅升级，主张小幅改动；以JavaScript创造者Brendan Eich为首的Mozilla公司，则坚持当前的草案。

2008年7月，由于对于下一个版本应该包括哪些功能，各方分歧太大，争论过于激烈，ECMA开会决定，中止ECMAScript 4.0的开发，将其中涉及现有功能改善的一小部分，发布为ECMAScript 3.1，而将其他激进的设想扩大范围，放入以后的版本。会后不久，ECMAScript 3.1就改名为ECMAScript 5。

3 let和const指令

Let类似var，只在命令所在代码块内有效，是局部变量，因此binding优先级会更高，var和let取let。

注意：在let命令声明变量之前，该变量是不可用的，否则会形成暂时性死区（temporal dead zone）。

可用环境：for循环，迭代函数。

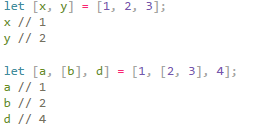
作用：增加显式报错，避免意料之外的行为。避免循环变量泄露为全局变量。

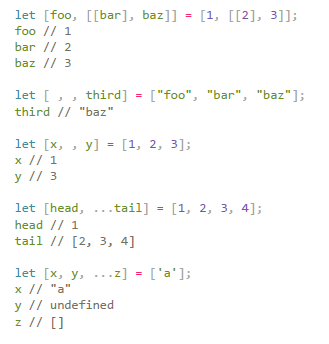
变化：取代了立即执行匿名函数（IIFE）(function(){ var tmp=…;…}();

Const 声明的变量值不能改变（read-only）。和let一样不能重复声明。如果声明一个数组，可以push值（添加元素），但不允许直接赋值。（即不允许指针地址改变）。如果想强制冻结，使用const foo = Object.freeze({});

若想在其他模块使用，可以用Import {A,b} from ‘./constants’; 或者 import \* as constants from ‘./constants/;

和C++非global定义即为局部变量相反，JS从前是只有全局变量（var=global），现在多了块约束。

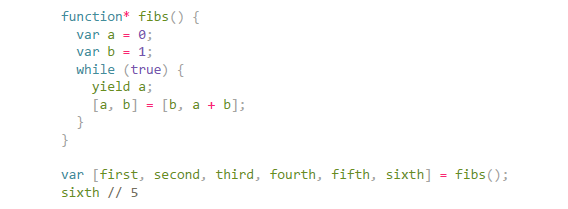
4. 变量的解构赋值

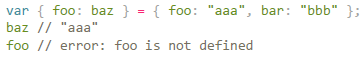


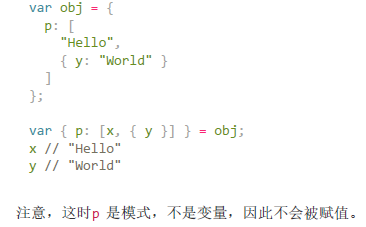
Undefined 对应于没有定义的情况，null对应于没有值得情况，两者有区别！如果赋值，null可以，undefined不行。

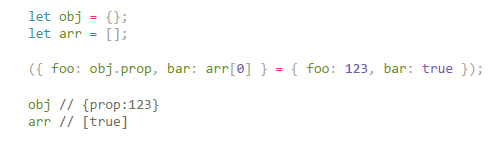
Yield用法：return告诉程序要返回什么值，而yield告诉程序当前的状态值，而且你运行到这里给我暂停一下

，也就是说每次return一个值，最后可以形成一个数组（也可以为空）。



上面代码中，真正被赋值的是变量baz ，而不是模式foo 。注意，采用这种写法时，变量的声明和赋值是一体的。对于let和const来说，变量不能重新声明，所以一旦赋值的变量以前声明过，就会报错。

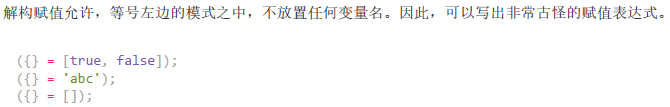






上面代码的写法会报错，因为JavaScript引擎会将{x}理解成一个代码块，从而发生语法错误。只有不将大括号写在行首，避免JavaScript将其解释为代码块，才能解决这个问题。





变量的解构赋值，很大的作用来自:

1. 将现有对象的方法，赋值到某个变量。Let{log,sin,cos}=Math;
2. 交换变量的值.[x,y] = [y,x];
3. 从函数返回多个值. Return [1,2,3];
4. 函数参数的定义. F({x:1,y:2,z:3})
5. 提取JSON数据. Var jsonData = {id:3,status:”OK”,data:[23,334]}; let{id,status,data:number} = jsonData;
6. 函数参数的默认值。 JQuery.ajax = function(url,{async = true, beforeSend=function(){}, cache=true,global = true,…}) {// do stuff};
7. 遍历map结构. For(let [key,value] of map){// do stuff}

但是对于函数调用会有一些tricky的地方：

Function move({x,y} = {x:0,y:0}){

Return [x,y];

}

Move({x:3}); // [3,undefied]

Move({}); // [ndf,ndf]

Move(); //[0,0]

在变量声明语句中，模式不能带有圆括号，例如{o : ({p:p})},[(z)]

可以使用圆括号的情况：赋值语句的非模式部分。

[(b)] = [3]

({ p : (d) } = {});

[(parseInt.prop)] = [3];

5. 字符串的扩展

* \u{20BB7} = “吉”
* For( let I of text){

Console.log(i);

}可以识别中文

* Repeat重复字符串
* ‘[结尾字符串]’.Padstart([字符串长度],’[重复字符串]=‘ ’’)
* 模板字符串：使用`…${变量}…`连接，减少’+’的使用

6. 正则的扩展

* /^.$/u.test(s) 可以识别中文（u修饰符）；粘连y修饰符以及全局匹配g修饰符.

Var s=”aaa\_aa\_a”;

Var r1 = /a+/g

Var r2 = /a+/y

R1.exec(s) // [‘aaa’]

R2.exec(s) // [‘aaa’]

R1.exec(s) // [‘aa’]

R2.exec(s) // null – 从剩余位置第一个（\_aa\_a的\_）开始

7. 数值的扩展

* 二进制&八进制： 0b11011,0o77266
* 转化为int: Number.parseInt()
* 增加小量：Number.EPSILON
* Math.trunc: 去除小数部分，否则返回NaN

8. 数组的扩展

* 字符串转char数组：Array.from(‘hello’)
* 还可以接受第二个参数，类似map方法:x => x\*x
* Array.of(3)=[3], Array(3)=[ , , ]
* [array].copyWithin(target,start=0,end=this.length)
* [array].find([map版](n)=>n<0 / [function版] function(value,index,arr){return …})寻找用第一个返回值为true的成员
* New Array(3).fill(number, start, end) 填充数组（可用于初始化）
* [array].keys(),values(),entries()可用for…of进行遍历
* [array].includes(number, start)用于寻找是否包含;Map.prototype.has(key)用于查找键名
* [,,,]是空位，[undefined,undefined]是算值的，空位可用filter过滤，许多方法（forEach,every,map,some）会自动跳过空位，ES6将其作为undefined处理，而for,fill,copyWithin会保留空位。
* 数组推导：var a2=[for(I of a1) if(i>0) i\*2];可以代替map: [1,2,3].map(function(i) {return i\*i});

8. 函数的扩展

* 允许设定函数默认值 function log(x,y=’world’){}
* Function foo({x,y=5}){}不允许使用foo()，需要使用foo({}),因为参数不是对象，x，y就不会被生成！
* Function m1({x=0, y=0}={}{}和function m2({x,y}={x:0,y:0})={}的区别在于前者函数参数的默认值是空对象，但是设置了对象解构赋值的默认值，即可以单独赋值；后者参数的默认值是一个有具体属性的函数，但是没有设置对象解构赋值的默认值，即只能整个赋值。当x有值,y无值时，m2的y为undefined,而m1的y为0.
* 另外如果function f(x, y=5, z)使用f(1,,2)调用时会报错，y的位置需要置为undefined，而null没有这个效果。
* 函数的长度问题：为无默认值的参数个数
* 作用域问题：如果function f(x,y=x){}此时x已经生成，y是f的第一个参数,如果 function f(y=x){}此时x为当下作用域最内部变量。
* 异常处理：

Funcrion throwIfMissing(){throw new Error(‘Missing Parameter’);}

function foo(para = throwIfMissing()){}

* 多余参数：”…变量名”用于获取所有剩下的参数，因此可以不限定输入参数个数。
* 多余参数逆运算：在函数调用的时候”…数组”可以将数组拆分成单个参数；或者可以合并数组。[…arr1,…arr2,…arr3];或者分解字符串[…”hello”]=[“h”,”e”,”l”,”l”,”o”]
* 箭头函数：快速映射->var f = (a,b)=>a+b
* 函数绑定：foo::bar == bar.bind(foo); foo::bar(…arguments); == bar.apply(foo,arguments);