**ES6新特性**

# 01、箭头操作符

|  |
| --- |
| var arr = [1,2,3,4]; arr.forEach(function(v,i,a){  console.log(v,i,a); });  arr.forEach(v => console.log(v)); |

# 02、类的支持

|  |
| --- |
| //类的定义  class Animal {  //ES6中新型构造器  constructor(name) {  this.name = name;  }  //实例方法  sayName() {  console.log('My name is '+this.name);  }  }  //类的继承  class Programmer extends Animal {  constructor(name) {  //直接调用父类构造器进行初始化  super(name);  }  program() {  console.log("I'm coding...");  }  }  //测试我们的类  var animal=new Animal('dummy'),  wayou=new Programmer('wayou');  animal.sayName();//输出 ‘My name is dummy’  wayou.sayName();//输出 ‘My name is wayou’  wayou.program();//输出 ‘I'm coding... |

# 03、增强的对象字面量

对象字面量被增强了，写法更加简洁与灵活，同时在定义对象的时候能够做的事情更多了。具体表现在：

可以在对象字面量里面定义原型

定义方法可以不用function关键字

直接调用父类方法

|  |
| --- |
| //通过对象字面量创建对象  var human = {  breathe() {  console.log('breathing...');  }  };  var worker = {  \_\_proto\_\_: human, //设置此对象的原型为human,相当于继承human  company: 'freelancer',  work() {  console.log('working...');  }  };  human.breathe();//输出 ‘breathing...’  //调用继承来的breathe方法  worker.breathe();//输出 ‘breathing...’ |

# 04、字符串模板

字符串模板相对简单易懂些。ES6中允许使用反引号 ` 来创建字符串，此种方法创建的字符串里面可以包含由美元符号加花括号包裹的变量${vraible}。如果你使用过像C#等后端强类型语言的话，对此功能应该不会陌生。

|  |
| --- |
| //产生一个随机数  var num=Math.random();  //将这个数字输出到console  console.log(`your num is ${num}`); |

# 05、结构

自动解析数组或对象中的值。比如若一个函数要返回多个值，常规的做法是返回一个对象，将每个值做为这个对象的属性返回。但在ES6中，利用解构这一特性，可以直接返回一个数组，然后数组中的值会自动被解析到对应接收该值的变量中。

|  |
| --- |
| var [x,y]=getVal(),//函数返回值的解构  [name,,age]=['wayou','male','secrect'];//数组解构  function getVal() {  return [ 1, 2 ];  }  console.log('x:'+x+', y:'+y);//输出：x:1, y:2  console.log('name:'+name+', age:'+age);//输出： name:wayou, age:secrect |

# 06、参数默认值

现在可以在定义函数的时候指定参数的默认值了，而不用像以前那样通过逻辑或操作符来达到目的了。

|  |
| --- |
| function sayHello(name){  //传统的指定默认参数的方式  var name=name||'dude';  console.log('Hello '+name);  }  //运用ES6的默认参数  function sayHello2(name='dude'){  console.log(`Hello ${name}`);  }  sayHello();//输出：Hello dude  sayHello('Wayou');//输出：Hello Wayou  sayHello2();//输出：Hello dude  sayHello2('Wayou');//输出：Hello Wayou |

# 07、不定参数

不定参数是在函数中使用命名参数同时接收不定数量的未命名参数。这只是一种语法糖，在以前的JavaScript代码中我们可以通过arguments变量来达到这一目的。不定参数的格式是三个句点后跟代表所有不定参数的变量名。比如下面这个例子中，…x代表了所有传入add函数的参数。

|  |
| --- |
| //将所有参数相加的函数  function add(...x){  return x.reduce((m,n)=>m+n);  }  //传递任意个数的参数  console.log(add(1,2,3));//输出：6  console.log(add(1,2,3,4,5));//输出：15 |

# 08、拓展参数

拓展参数则是另一种形式的语法糖，它允许传递数组或者类数组直接做为函数的参数而不用通过apply。

|  |
| --- |
| var people=['Wayou','John','Sherlock'];  //sayHello函数本来接收三个单独的参数人妖，人二和人三  function sayHello(people1,people2,people3){  console.log(`Hello ${people1},${people2},${people3}`);  }  //但是我们将一个数组以拓展参数的形式传递，它能很好地映射到每个单独的参数  sayHello(...people);//输出：Hello Wayou,John,Sherlock  //而在以前，如果需要传递数组当参数，我们需要使用函数的apply方法  sayHello.apply(null,people);//输出：Hello Wayou,John,Sherlock |

# 09、let与const关键字

可以把let看成var，只是它定义的变量被限定在了特定范围内才能使用，而离开这个范围则无效。const则很直观，用来定义常量，即无法被更改值的变量

|  |
| --- |
| for (let i=0;i<2;i++)console.log(i);//输出: 0,1  console.log(i);//输出：undefined,严格模式下会报错 |

# 10、for of 值遍历

我们都知道for in 循环用于遍历数组，类数组或对象，ES6中新引入的for of循环功能相似，不同的是每次循环它提供的不是序号而是值。

|  |
| --- |
| var someArray = [ "a", "b", "c" ];  for (v of someArray) {  console.log(v);//输出 a,b,c  } |

# 11、模块

在ES6标准中，JavaScript原生支持module了。这种将JS代码分割成不同功能的小块进行模块化的概念是在一些三方规范中流行起来的，比如CommonJS和AMD模式。

将不同功能的代码分别写在不同文件中，各模块只需导出公共接口部分，然后通过模块的导入的方式可以在其他地方使用。

|  |
| --- |
| // point.js  module "point" {  export class Point {  constructor (x, y) {  public x = x;  public y = y;  }  }  }    // myapp.js  //声明引用的模块  module point from "/point.js";  //这里可以看出，尽管声明了引用的模块，还是可以通过指定需要的部分进行导入  import Point from "point";    var origin = new Point(0, 0);  console.log(origin); |

# 12、Map，Set 和 WeakMap，WeakSet

这些是新加的集合类型，提供了更加方便的获取属性值的方法，不用像以前一样用hasOwnProperty来检查某个属性是属于原型链上的呢还是当前对象的。同时，在进行属性值添加与获取时有专门的get，set 方法。

|  |
| --- |
| // Sets  var s = new Set();  s.add("hello").add("goodbye").add("hello");  s.size === 2;  s.has("hello") === true;  // Maps  var m = new Map();  m.set("hello", 42);  m.set(s, 34);  m.get(s) == 34; |

有时候我们会把对象作为一个对象的键用来存放属性值，普通集合类型比如简单对象会阻止垃圾回收器对这些作为属性键存在的对象的回收，有造成内存泄漏的危险。而WeakMap,WeakSet则更加安全些，这些作为属性键的对象如果没有别的变量在引用它们，则会被回收释放掉。

|  |
| --- |
| // Weak Maps  var wm = new WeakMap();  wm.set(s, { extra: 42 });  wm.size === undefined  // Weak Sets  var ws = new WeakSet();  ws.add({ data: 42 });//因为添加到ws的这个临时对象没有其他变量引用它，所以ws不会保存它的值，也就是说这次添加其实没有意思 |

# 13、Proxies

Proxy可以监听对象身上发生了什么事情，并在这些事情发生后执行一些相应的操作。一下子让我们对一个对象有了很强的追踪能力，同时在数据绑定方面也很有用处。

|  |
| --- |
| //定义被侦听的目标对象  var engineer = { name: 'Joe Sixpack', salary: 50 };  //定义处理程序  var interceptor = {  set: function (receiver, property, value) {  console.log(property, 'is changed to', value);  receiver[property] = value;  }  };  //创建代理以进行侦听  engineer = Proxy(engineer, interceptor);  //做一些改动来触发代理  engineer.salary = 60;//控制台输出：salary is changed to 60 |

# 14、Symbols

我们知道对象其实是键值对的集合，而键通常来说是字符串。而现在除了字符串外，我们还可以用symbol这种值来做为对象的键。Symbol是一种基本类型，像数字，字符串还有布尔一样，它不是一个对象。Symbol 通过调用symbol函数产生，它接收一个可选的名字参数，该函数返回的symbol是唯一的。之后就可以用这个返回值做为对象的键了。Symbol还可以用来创建私有属性，外部无法直接访问由symbol做为键的属性值。

|  |
| --- |
| (function() {  // 创建symbol  var key = Symbol("key");  function MyClass(privateData) {  this[key] = privateData;  }  MyClass.prototype = {  doStuff: function() {  ... this[key] ...  }  };  })();  var c = new MyClass("hello")  c["key"] === undefined//无法访问该属性，因为是私有的 |

# 15、Math，Number，String，Object 的新API

|  |
| --- |
| Number.EPSILON  Number.isInteger(Infinity) // false  Number.isNaN("NaN") // false  Math.acosh(3) // 1.762747174039086  Math.hypot(3, 4) // 5  Math.imul(Math.pow(2, 32) - 1, Math.pow(2, 32) - 2) // 2  "abcde".contains("cd") // true  "abc".repeat(3) // "abcabcabc"  Array.from(document.querySelectorAll('\*')) // Returns a real Array  Array.of(1, 2, 3) // Similar to new Array(...), but without special one-arg behavior  [0, 0, 0].fill(7, 1) // [0,7,7]  [1,2,3].findIndex(x => x == 2) // 1  ["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]  ["a", "b", "c"].keys() // iterator 0, 1, 2  ["a", "b", "c"].values() // iterator "a", "b", "c"  Object.assign(Point, { origin: new Point(0,0) }) |

# 16、Promises

Promises是处理异步操作的一种模式，之前在很多三方库中有实现，比如jQuery的[deferred](http://api.jquery.com/category/deferred-object/) 对象。当你发起一个异步请求，并绑定了.when(), .done()等事件处理程序时，其实就是在应用promise模式。

|  |
| --- |
| //创建promise  var promise = new Promise(function(resolve, reject) {  // 进行一些异步或耗时操作  if ( /\*如果成功 \*/ ) {  resolve("Stuff worked!");  } else {  reject(Error("It broke"));  }  });  //绑定处理程序  promise.then(function(result) {  //promise成功的话会执行这里  console.log(result); // "Stuff worked!"  }, function(err) {  //promise失败会执行这里  console.log(err); // Error: "It broke"  }); |