

JavaScript进阶 (ES6)

- ES6对内置对象的扩展
- ES6对函数的扩展
- ES6新增数据类型和数据结构



河北师范大学软件学院
Software College of Hebei Normal University

JavaScript进阶 (ES6)

---ES6对内置对象的扩展



河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **ES6 对String和RegExp的扩展**
- **ES6 对Number和Math的扩展**
- **ES6 对Array和Object的扩展**



对String的扩展

- ES6中提供了字符串的遍历接口（可用for...of循环遍历）

```
for (let codePoint of 'foo') {  
  console.log(codePoint)  
} // "f", "o", "o"
```

- ES6中提供新的方法用于查找、判断和生成字符串

- indexOf (ES5)、includes、startsWith、endsWith、repeat

```
var s = 'Hello world!';  
s.startsWith('Hello') // true  
s.endsWith('!') // true  
s.includes('o') // true
```

思考：与正则的对比
如何使用正则完成类似的功能

对RegExp的扩展

- ES5、ES6中通过构造函数实例化正则对象的形式不同

```
var regex = new RegExp('xyz', 'i');  
var regex = new RegExp(/xyz/i);  
var regex = new RegExp(/xyz/, 'i');  
new RegExp(/abc/ig, 'i').flags//i
```

- ES6中为正则添加了y修饰符，（粘连sticky）修饰符

```
var s = 'aaa_aa_a';  
var r1 = /a+/g; var r2 = /a+/y;  
r1.exec(s); r2.exec(s); // ["aaa"] ["aaa"]  
r1.exec(s); r2.exec(s); //["aa"] null
```

- ES6中正则新的增对象属性（sticky属性、flags属性）

内容提纲

- ES6 对String和RegExp的扩展
- ES6 对Number和Math的扩展
- ES6 对Array和Object的扩展



对Number、Math的扩展

- ES6中Number新的静态方法

- Number.isFinite()、Number.isNaN()
- Number.parseInt()、Number.parseFloat() (减少全局性方法, 使得语言模块化)
- Number.isInteger() 具体参见Demo实例

- ES6中Math对象新增的方法

- Math.trunc()、Math.sign() 等

```
console.log(Math.trunc(4.9)); // 4
console.log(Math.trunc(-4.1)); // -4
console.log(Math.sign(-5)) // -1
console.log(Math.sign(5)) // +1
console.log(Math.sign(-0)) // +0
```

内容提纲

- ES6 对String和RegExp的扩展
- ES6 对Number和Math的扩展
- ES6 对Array和Object的扩展



对Array的扩展

- ES6中Array新增的静态方法

- Array.from() 可将类数组对象或可遍历的对象（包括Map）转换为数组
- Array.of() 可将一组值，转换为数组，弥补数组构造函数Array()的不足

- ES6中Array新增的原型方法

- Array.prototype.copyWithin()
- Array.prototype.find()、Array.prototype.findIndex()
- Array.prototype.fill()、Array.prototype.entries()
- Array.prototype.keys()、Array.prototype.values()
- Array.prototype.includes()

- ES6中空位数组（稀疏数组）部分

对Object的扩展

• 属性的简洁表示法

- ES6允许在对象之中，直接写入变量和函数，作为对象的属性和方法
- 只写属性名不写属性值时，属性值等于属性名所代表的变量，没有冒号了
- 简洁的对象表示法，使得创建对象和返回对象更为简洁（典型案例：get/set属性方法）

```
var name = "Jack";
var p1 = {
  name,
  hello() {
    console.log("Hi,I'm ", this.name);
  }
};
p1.hello();// Hi,I'm Jack
```

对Object的扩展

- ES6允许字面量定义对象时，用表达式作为对象的属性名

```
let propKey = 'foo';  
let obj = {  
  [propKey]: true,  
  ['a' + 'bc']: 123  
};
```

属性名表达式与简洁表示法，不能同时使用

```
console.log(obj); // {foo: true, abc: 123}
```

- Object新增的静态方法

- Object.is()、Object.assign()
- Object.setPrototypeOf()、Object.getPrototypeOf()
- Object.values()、Object.entries()

The background of the slide is decorated with various abstract shapes in shades of green and yellow. These shapes, which include circles, ovals, and irregular blobs, are scattered across the top and right sides of the slide, creating a modern, organic feel.

Have a Break!

JavaScript进阶 (ES6)

---ES6对函数的扩展



河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **ES6 新增的箭头函数**
- **ES6 对函数参数默认值的扩展**
- **ES6 中的Rest与Spread操作符**



ES6新增的箭头函数

- ES6中提供了新的语法规则来描述函数（箭头函数=>）

- 箭头函数语法简单地描述为：参数 => 函数体 或 （参数） => { 函数体 }
- 优点：可减少冗余的代码（如function关键字等）节省空间，避免this指向错误
- 如果箭头函数不需要参数或需要多个参数时，就使用一个圆括号代表参数部分

// ES5 的写法

```
var max= function (a,b) {  
    return a>b?a:b;  
};  
max(2,3);//3
```

// ES6 的写法

```
var max= (a,b) => a>b?a:b;  
max(2,3);//3
```

复合语句的话，需要使用大括号
和对应的return语句进行返回，
单语句可以不用return关键字

ES6新增的箭头函数

• 箭头函数需注意的几个点

- 函数内的 `this` 是与函数定义时所在的对象绑定，而不是使用时所在的对象（避免 `this` 缺陷）
- 大括号被解释为代码块，所以如果箭头函数直接返回一个对象，需在对象外面加上括号

```
var point = {  
  x:0,  
  y:0,  
  moveTo:function (x,y) {  
    //内部嵌套函数  
    var moveToX = ()=>this.x=x;//this正确绑定到point  
    var moveToY = ()=>this.y=y;//this正确绑定到point  
    moveToX();  
    moveToY();  
  }  
};  
point.moveTo(2,2);  
console.log(point);//{x:2,y:2}
```


内容提纲

- ES6 新增的箭头函数
- ES6 对函数参数默认值的扩展
- ES6 中的Rest与Spread操作符



ES5 函数参数默认值的实现方法

- ES5中不能直接为函数的参数指定默认值，需通过 || 来实现

```
var sum = function(a,b,c){  
    b = b || 4;  
    c = c || 5;  
    return a+b+c;  
}  
console.log(sum(1,2,3)); //1+2+3  
console.log(sum(1,2));   //1+2+5  
console.log(sum(1));      //1+4+5
```

案例中：未传实参的话，形参初始为undefined，undefined转为布尔类型为false，根据||短路原则直接返回右操作数，相当于给参数指定了默认值

问题：思考sum(1,0,0)返回多少？ 1还是10

ES6 对函数参数默认值的扩展

- ES6允许为函数的参数设置默认值

- 直接写在参数定义的后面，比ES5更加直观
- 不会出现ES5中实参转换为布尔类型的问题

```
var sum = function(a,b=4,c=5){  
    return a+b+c;  
}  
console.log(sum(1,2,3)); //1+2+3  
console.log(sum(1,2));   //1+2+5  
console.log(sum(1));     //1+4+5  
console.log(sum(1,0,0)); //1+0+0
```

使用Babel查看ES5是如何实现相应的功能

ES6 对函数参数默认值的扩展

• ES6函数的参数默认值注意事项

- 带默认值的参数变量是默认声明的，所以函数体内不能再用let或const重复声明
- 参数一般有顺序，有默认值的参数应该是尾参数，这样可以使有默认值的用默认值没有默认值的用传递的值

```
function foo(x=5){  
    let x = 1;    //报错  
    const x = 2; //报错  
}  
foo();
```

```
function f(x = 1,y) {  
    return [x,y];  
}  
f();//[1,undefined]  
f(2);//[2,undefined]  
f(,3)//报错，无法使x用1， y用3
```

Uncaught SyntaxError: Unexpected

内容提纲

- ES6 新增的箭头函数
- ES6 对函数参数默认值的扩展
- ES6 中的Rest与Spread操作符



ES6 中的Rest与Spread操作符

• ...Rest (剩余操作符)

- 主要用在函数参数的声明中，可获得隐含的实参，取代ES5中函数隐藏变量arguments
- arguments (获得所有实参) 是个类数组对象，缺点不能像操作数组那样直接操作
- ...Rest比arguments更灵活，...Rest操作符需放在了函数形参的最后，实例如下

```
function f(x, ...y) {  
    console.log(x,y);  
}  
f("a", "b", "c"); // "a", ["b", "c"]  
f("a"); // "a", []  
f(); // undefined, []
```


ES6 中的Rest与Spread操作符

- ...Spread (扩展操作符)

- 主要用在函数的调用中使用 (虽然也是..., 但使用的场景不同)
- Spread将一个数组转换为用逗号分隔的参数序列, 是...Rest的逆过程
- 在call和apply的转换过程中十分有用

```
function f(x, ...y) {  
    console.log(x,y);  
}  
f("a",...["b","c"]); //"a",["b","c"]  
f("a"); //"a",[]  
f(); //undefined,[]
```

总结

- **ES6 新增的箭头函数**
- **ES6 对函数参数默认值的扩展**
- **ES6 中的Rest与Spread操作符**



The background of the slide is decorated with various abstract shapes in shades of green and yellow. These shapes, which include circles, ovals, and teardrop-like forms, are scattered across the top and right sides of the slide, creating a modern and organic feel.

Thank You!



河北师范大学软件学院
Software College of Hebei Normal University

JavaScript进阶 (ES6)

---ES6新增数据类型和数据结构



河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- **新增数据类型 (Symbol)**
- **新增数据结构 (Set)**
- **新增数据结构 (Map)**



新增的数据类型Symbol

• 属性名的冲突问题，以及Symbol的提出

- ES5的对象属性名都是字符串，这容易造成属性名的冲突
- ES6引入了一种新的原始数据类型Symbol，表示独一无二的值
- Symbol变量属于**基本数据类型**（不是对象），Symbol函数前**不能使用new命令**
- Symbol函数可以接受一个字符串作为参数，表示对Symbol实例的描述，主要用于区分变量

```
let s = Symbol();  
typeof s; // "symbol"  
var s1 = Symbol('foo');  
var s2 = Symbol('bar');  
console.log(s1); // Symbol(foo)  
console.log(s2); // Symbol(bar)
```

新增的数据类型Symbol

• Symbol的特点

- Symbol函数的参数只是表示Symbol值的描述，相同参数的Symbol函数的返回值是不相等的
- Symbol变量不能与其他值进行运算，但可转换成字符串类型

```
var s1 = Symbol("foo");    var s1 = Symbol();  
var s2 = Symbol("foo");    var s2 = Symbol();  
s1 === s2 // false        s1 === s2 // false
```

```
var sym = Symbol('My symbol');  
// "your symbol is " + sym; // 报错  
var sym = Symbol('My symbol');  
String(sym); // 'Symbol(My symbol)'  
sym.toString(); // 'Symbol(My symbol)'
```


新增的数据类型Symbol

• 作为属性名的Symbol

- 由于每一个Symbol值都是不相等的，这意味着Symbol值可以作为标识符，用于对象的属性名，就能保证不会出现同名的属性。这对于一个对象由多个模块构成的情况非常有用，能防止某一个键被不小心改写或覆盖，最为对象属性的具体形式如下

```
var mySymbol = Symbol();  
var a = {};  
a[mySymbol] = 'Hello!'; // 第一种写法  
var a = {  
  [mySymbol]: 'Hello!' // 第二种写法  
};  
var a = {};  
Object.defineProperty(a, mySymbol, { value: 'Hello!' }); // 第三种写法  
// 以上写法都得到同样结果  
a[mySymbol] // "Hello!"
```

新增的数据类型Symbol

- 作为属性名的Symbol（注意访问属性的方法）

- 区分使用点操作符和中括号操作符时，访问对象属性的不同，Symbol需使用[]，而不是点

```
var mySymbol = Symbol();  
var a = {};  
a.mySymbol = 'Hello!';  
console.log(a[mySymbol]); // undefined  
console.log(a['mySymbol']); // "Hello!"
```

- 使用Symbol值定义属性时，Symbol值须放在方括号之中

```
let s = Symbol();  
let obj = {  
  [s]: function (arg) {console.log(arg);} //若不放中括号内会怎样?  
};  
obj[s](123);
```

新增的数据类型Symbol

• 作为属性名的Symbol的遍历特性

- Symbol作为属性名，该属性不会出现在for...in、for...of循环中
- 也不会被Object.keys()、Object.getOwnPropertyNames()返回，但它也不是私有属性
- 使用Object.getOwnPropertySymbols方法，可以获取指定对象的所有Symbol属性名

```
var obj = {}; var foo = Symbol("foo");  
Object.defineProperty(obj, foo, {  
    value: "foo bar",  
});  
for (var i in obj) {  
    console.log(i); // 无输出  
}  
console.log(Object.getOwnPropertyNames(obj)); // []  
console.log(Object.getOwnPropertySymbols(obj)); // [Symbol(foo)]
```


新增的数据类型Symbol

•与Symbol变量复用相关的静态方法

- Symbol.for()接受一个字符串作为参数，搜索有没有以该参数作为名称的Symbol值。如果有，就返回这个Symbol值，否则就新建并返回一个以该字符串为名称的Symbol值
- Symbol.keyFor()方法返回一个已登记的Symbol类型值的key，字符串类型

```
var s1 = Symbol.for('foo');  
var s2 = Symbol.for('foo');  
console.log(s1 === s2); // true
```

```
console.log(Symbol.for("bar") === Symbol.for("bar")); // true  
console.log(Symbol("bar") === Symbol("bar")); // false  
console.log(Symbol.for("bar") === Symbol("bar")); // false
```

内容提纲

- 新增数据类型 (Symbol)
- 新增数据结构 (Set)
- 新增数据结构 (Map)



新增数据结构 (Set)

- ES6提供了新的数据结构Set

- 它类似于数组，但是成员的值都是唯一的，没有重复的值
- 用Set构造函数来生成Set对象，用法类似实例化数组对象，通过new实例化Set对象
- 通过add方法向Set结构加入成员，Set结构不会添加重复的值

```
let s1 = new Set([1,2,3,4,5,5,6,2,2]);  
console.log(s1); //Set(6) {1, 2, 3, 4, 5...}
```

```
var s2 = new Set();  
[2, 3, 5, 4, 5, 2, 2].map(x => s2.add(x));  
for (let i of s2) {  
  console.log(i);  
} // 2 3 5 4
```

新增数据结构 (Set)

• Set的原型属性和方法

- Set.prototype.constructor、Set.prototype.size
- Set.prototype.add(value)、Set.prototype.delete(value)
- Set.prototype.has(value)、Set.prototype.clear()
- Set.prototype.keys() (注意返回的类型)、Set.prototype.values()、Set.prototype.entries()

```
var properties = new Set();
properties.add('width');
properties.add('height');
console.log(properties.size);
if (properties.has('width') && properties.has('height')) {
    console.log("do something!");
}
```

• WeakSet (成员只能是对象且都是弱引用, 参阅回收机制)

内容提纲

- 新增数据类型 (Symbol)
- 新增数据结构 (Set)
- 新增数据结构 (Map)



新增数据结构 (Map)

- ES6提供了新的数据结构Map

- 它类似于对象，也是键值对的集合，但是“键”的范围不限于字符串
- Object结构提供了“字符串-值”的对应，Map结构提供了“值-值”的对应
- Map也可以接受一个数组作为参数，组的成员是一个个表示键值对的数组

```
var m = new Map();  
var o = {p: 'Hello World'};  
m.set(o, 'content')  
m.get(o); // "content"  
m.has(o); // true  
m.delete(o); // true  
m.has(o); // false
```

```
var o = {}  
var map = new Map([  
  ['name', '张三'],  
  [o, 'Author']  
]);  
map.size // 2  
map.has('name'); // true  
map.get('name'); // "张三"  
map.has(o); // true  
map.get(o); // "Author"
```

新增数据结构 (Map)

• Map的原型属性和方法

- Map.prototype.size、Map.prototype.set(key、value)、Map.prototype.get(key)
- Map.prototype.has(key)、Map.prototype.delete(key)、Map.prototype.clear()
- Map.prototype.keys() (注意返回类型)、Map.prototype.values()、Map.prototype.entries()

```
let map = new Map()
    .set(1, 'a')
    .set(2, 'b');
// get方法读取key对应的键值，如果找不到key，返回undefined
var m = new Map();
var hello = function() {console.log("hello");}
m.set(hello, "Hello ES6!"); // 键是函数
m.get(hello); // Hello ES6!
```

• WeakMap (只接受对象作为键名，弱引用)



Thank You!



河北师范大学软件学院
Software College of Hebei Normal University

ES6 新增的遍历语法for...of

- 你是如何遍历数组中的元素的?
- 当 JavaScript 刚被发布的时候, 可能如下这么写

```
for (var index = 0; index < myArray.length; index++) {  
    console.log(myArray[index]);  
}
```

- 从 ES5 开始, 你可能使用内置的 forEach 方法

```
myArray.forEach(function (value) {  
    console.log(value);  
});
```

- 缺点: 不能通过使用 break 语句退出循环或者使用 return 语句从封闭函数中返回

ES6

- 当使用for...in会如何

```
for (var index in myArray) { // 不要用for...in遍历数组，for...in可用来遍历对象  
    console.log(myArray[index]);  
}
```

- 分配给索引的值为字符串 "0" , "1" 等等，不是真正的数字类型的数字。这样使用是不方便，不符合原意的，比如想的到第2个元素的下一个元素时，你可能不希望得到字符串运算 ("2" + 1 == "21") 的结果

-