# 第五章 引用类型

## object 类型

- 创建实例的方式

```
//1
var person = new Object();
person.name = 'nick';
person.age = 18;

//2 字面量表示法
var person = {
    name:'nick',
    age:18
}
```

## Arrary 类型

- 创建方式

```
//1
var colors = new Arrary();//可传参，表示数组的大小

var colors = new Array('red','blue','green');

var colors = Array(3);
var colors = Array('nick');//在数组构造函数中，可省略new操作符


//2 字面量表示法
var colors = ['red','blue','green'];//三个字符串的数组
var name = [];//空数组
var values = [1,2];
```

- 读取

```
var colors = ['red','blue','green'];
alert(colors[0]);//red
color[1] = 'black';//第二项被修改了
color[3] = 'yellow';//新增第四项
```

- Array 的 length 属性不是只读的，可用通过修改 length 来修改数组

```
var colors = ['red','blue','green'];
colors.length = 2;
//此时数组的第三项将会变成undefined,同理可通过此方法增加数组内的项
color[color.length] = 'white';
```

- 转换方法

```
<script>
    var colors = ['red','blue','green'];
    alert(colors.toString());//red,blue,green
    alert(colors.valueOf());//red,blue,green
    alert(colors);//red,blue,green
</script>
```

- 栈方法 LIFO（先进后出）

```
//push方法接收任何数量的参数，逐个添加到数组末尾，pop方法则从数组末尾移除最后一项
<script>
    var colors = ['red','blue','green'];
    // alert(colors.toString());//red,blue,green
    // alert(colors.valueOf());//red,blue,green
    // alert(colors);//red,blue,green

    colors.push('brown');
    alert(colors.length);//4

    var item = colors.pop();
    alert(item);//brown
</script>
```

- 队列方法 FIFO（先进先出）

```
<script>
    var colors = ['red','blue','green'];

    colors.push('brown');
    alert(colors.length);//4

    var item2 = colors.shift();//插入到尾部，此外unshift是插入到头部
    alert(item2);//red
    alert(colors);//blue,green,brown

</script>
```

- 重排列方法

```
<script>
    //反向数组
    var values = [0,1,2,3,4,5];
    values.reverse();
    alert(values);//5,4,3,2,1,0

    //升序排列 先将每一项转换为字符串再作比较，从小到大排列
    var values2 = [2,1,5,9,7,10];
    values2.sort();
    alert(values2);//1,10,2,5,7,9
    //10排在前面是因为，按字符串来排列的。

    //比较函数compare(value1,value2)，如果参数一小于参数二，返回-1；参数一大于参数二，返回1；相等返回零
    function compare(val1,val2){
        if(val1 < val2){
            return -1;
        }else if(val1 > val2){
            return 1;
        }else{
            return 0;
        }
    }
    //可将比较函数作为参数传入sort中即可正确升序比较

    values2.sort(compare)
    alert(values2);//1,2,5,7,9,10
</script>
```

- 操作方法

```
    //concat()不传参的情况下，只会返回一个原数组的副本；如果传入一个或多个数组，则会将每一项都添加到结果数组中；
    //如果传的不是数组，则直接简单添加到数组末尾
    var colors = ['red','blue','green'];
    var colors2 = colors.concat('yellow',['black','pink']);

    alert(colors2);//red,blue,green,yellow,black,pink

    //slice() 基于当前数组中一个或多个项创建一个新数组；参数1：指定位置开始的位置到末尾项；参数二：参数一所在位置到参数二位置的项的前一个（即不包括参数二位置的项）
    //该方法不影响原始数组
    var colors3 = colors2.slice(1);
    var colors4 = colors2.slice(1,3);

    alert(colors3);//blue,green,yellow,black,pink
    alert(colors4);//blue,green
```

```
    //splice() 删除，参数一：要删除的起始位置；参数二：要删除的项数
    //          插入，参数一：要插入的起始位置；参数二：0（表示不删除任何项）；参数三：要
插入的项；
    //          替换，参数一：起始位置；参数二：要被替换的项数；参数三：替换后的项；

    var colors = ['red','green','blue'];
    var removed = colors.splice(0,1);
    alert(colors);//green,blue
    alert(removed);//red

    removed = colors.splice(1,0,'yellow','pink');
    // alert(colors);//green,yellow,pink,blue
    // alert(removed);//  为空，原因是没有删除任何项

    removed = colors.splice(1,1,'red','purlple');
    alert(colors);//green,red,purlple,pink,blue
    alert(removed);//yellow
```

- 位置方法

1. indexOf() 参数一：要查找的项；参数二：查找的起点位置的索引（可选） 从数组开头开始查找
2. lastIndexOf() 参数一：要查找的项；参数二：查找的起点位置的索引（可选） 从数组尾部开始查找

```
var numbers = [1,2,3,4,5,4,3,2,1];

alert(numbers.indexOf(4));//3 因为4的索引是3

alert(numbers.lastIndexOf(4));//5 因为从后往前遍历出现的第一个4的索引是5

alert(numbers.indexOf(4,4));//5

alert(numbers.lastIndexOf(4,4))//3

var person = {name:'nick'};
var people = [{name:'nick'}];

var morePeople = [person];

alert(people.indexOf(person));//-1 虽然值相同但不是同一个引用，因此为-1（即这两个方法
都是全等比较）
alert(morePeople.indexOf(person));//0
```

- 迭代方法

1. every() 参数一：每一项上要运行的函数； 参数二：运行该函数的作用域对象（可选）---影响 this 的值
    - 对数组的每一项运行指定函数，若数组中的每一项都返回 true 则该函数返回 true
2. filter() 参数一：每一项上要运行的函数； 参数二：运行该函数的作用域对象（可选）---影响 this 的值
    - 对数组的每一项运行指定函数，返回数组中返回 true 的项

3. forEach() 参数一：每一项上要运行的函数； 参数二：运行该函数的作用域对象（可选）---影响 this 的值
    - 对数组的每一项运行指定函数，没有返回值
4. map() 参数一：每一项上要运行的函数； 参数二：运行该函数的作用域对象（可选）---影响 this 的值
    - 对数组的每一项运行指定函数，返回每次函数调用返回结果的组成数组
5. some() 参数一：每一项上要运行的函数； 参数二：运行该函数的作用域对象（可选）---影响 this 的值
    - 对数组的每一项运行指定函数，若函数中任意一项返回 true，则返回 true
6. 以上方法都不会修改数组中的值
7. 传入该方法中的函数会接收三个参数：数组的项的值，数组中项的索引位置，数组对象本身；

```
//every 和 some 的比较
var numbers = [1,2,3,4,5,4,3,2,1];

var everyResult = numbers.every(function(item,index,arry){
    return (item > 2);
});

alert(everyResult);//false

var someResult = numbers.some(function(item,index,arry){
    return (item > 2);
});

alert(someResult);//true
```

```
//filter 和 map
var numbers = [1,2,3,4,5,4,3,2,1];

var filterResult = numbers.filter(function(item,index,arry){
    return (item > 2);
});

alert(filterResult);//3,4,5,4,3


var mapResult = numbers.map(function(item,index,arry){
    return item * 2;
})

alert(mapResult);//2,4,6,8,10,8,6,4,2
```

- 缩小方法

1. reduce() 参数一：在数组中每一项要调用的函数；参数二：作为缩小基准的初始值（可选）
2. reduceRight() 参数一：在数组中每一项要调用的函数；参数二：作为缩小基准的初始值（可选）

3. 传入该方法的函数接收四个参数：前一个值，当前值，项的索引值，数组对象；该函数的返回值会作为第一个参数传给下一次循环；

```
var values = [1,2,3,4,5];

var sum = values.reduce(function(prev,cur,index,array){
    return prev + cur;
});

alert(sum);//15 第一次：prev=1，cur=2；第二次prev=3 (1+2) ，cur=3...
//从右边开始遍历
var sum2 = values.reduceRight(function(prev,cur,index,array){
    return prev + cur;
});

alert(sum2);//15 第一次：prev=5，cur=4；第二次prev=9 (4+5) ，cur=3...
```

## Date 类型

- 创建日期对象,不传参的情况下自动获取当前日期和时间

```
//创建日期对象
    var now = new Date();
```

- Date.parse() 和 Date.UTC()

1. Date.parse() 接收一个表示日期的字符串，尝试将这个字符串返回相应日期的毫秒数

## RegExp 类型

- ECMAScript 通过 RegExp 类型来支持正则表达式
- 创建正则表达式 `var expression = /pattern / flages;`
    - 其中 pattern 部分，可是任何简单或复杂的正则表达式， flag 标志用来标明正则表达式的行为：
    - g:表示全局 即模式将应用到所有字符串
    - i:biaos 不区分大小写，但只匹配第一个
    - m:表示多行模式

1. 字面量形式定义正则表达式

```
var pattern1 = /at/g; //匹配所有包含at的实例

var pattern2 = /[bc]at/i; //匹配第一个bat或cat

var pattern3 = /.at/gi; //匹配以at结尾的三个字母的组合 不区分大小写
```

2. RegExp 函数定义正则表达式

```
var pattern1 = /[bc]at/i;

//等价于

var pattern2 = new RegExp('[bc]at','i');
//注意所传入的参数为字符串
```

3. RegExp 实例方法

- exec() 接受一个参数 即要应用模式的字符串，然后返回一个包含第一个匹配信息的数组；无匹配的情况下返回 null

```
    var text = 'mom and dad and baby';
    var pattern = /mom( and dad( and baby)?)?/gi;

    var matches = pattern.exec(text);

    alert(matches.index);//0
    alert(matches.input);//mom and dad and baby
    alert(matches[0]);//mom and dad
    alert(matches[1]);// and dad
    alert(matches[2]);// and baby
```

# Function 类型（实际上就是一个对象）

- 每个函数都是 Fuction 对象的一个实例
- 函数名仅仅是指向函数的指针
- 没有重载
- 函数的声明是可以在调用之后的，因为 js 引擎会把函数的声明提升至顶部，但是如果函数的初始化是在一个语句中，则不可行

```
alert(sum(10,10));
 function sum(a,b){
     return a+b;
 }//20


  alert(sum(10,10));
  var sum = function(a,b){
      return a+b;
  }//sum is not a function
```

- 函数的内部属性
    - this 函数执行的环境对象

- arguments

```
//阶乘函数
function factorial(num){
if(num <= 1){
    return 1;
}else{
    return num*factorial(num-1);
    }
}
//为了解决递归时的高耦合（即else情况下再次调用同名的函数）
function factorial(num){
if(num <= 1){
    return 1;
}else{
    return num*arguments.callee(num-1);
    }
}
//用argments.callee 来代替递归时的函数调用 可以使得无论函数名改变成什么 都能完成递归
var anotherfunc = factorial;
alert(factorial(3));//6
alert(anotherfunc(3));//6
```

**函数的属性和方法**

- 属性： length 表示函数希望接收的参数的数量 prototype 保存实例方法的真正所在（toString啊 valueOf啊） 该属性不可枚举，因此用for-in无法遍历出来
- 方法：（非继承而来的) apply() 参数1：在其运行函数的作用域 参数2：参数数组 call() 参数1：在其运行函数的作用域 后面的参数要列举所以参数 不能以数组形式
  - 使用call可以扩充函数的作用域

```
var o = {
    color:'blue'
}

window.color = 'red';

function getColor(){
    alert(this.color);
}

getColor.call(this);//red
getColor.call(window);//red
getColor.call(o);//blue
```

bind() 创建一个函数的实例，其this的值会被绑定到传给bind函数的值

```
 window.color = 'red';
    var o = {color:'blue'};

    function sayColor(){
        alert(this.color);
    }

    var objSayColor = sayColor.bind(o);
    objSayColor();//blue 即函数内this指向改成了 对象o
```

# 基本包装类型

- Boolean类型 是与布尔值对应的引用类型
    - 创建布尔对象：`var booleanObj = new Boolean(true);`

```
    var falseObj = new Boolean(false);
    var result = falseObj && true;
    alert(result);//true 原因是 falseObj是一个对象 当对象进行逻辑与操作时 会默认转换成true
    //要区分 基本类型 引用类型的两种布尔值
```

    - 不建议使用Boolean对象！！！
- Number类型 是与数字值对应的引用类型
    - 创建对象： `var numberObj = new Number(10);`
    - toString() 可以传入一个参数告诉他字符串返回出来的进制

```
var num = 10;
alert(num.toString());//"10"
alert(num.toString(2));//"1010"
alert(num.toString(8));//"12"
```

    - toFixed() 将数值转为字符串，并指定返回小数位数

```
var num = 10;
alert(num.toFixed(2))//"10.00"
```

- String类型 字符串的对象包装类型
    - 创建对象： `var strObj = new String('hello world');`
    - charAt() `alert(strObj.charAt(1));//f返回的是下标为1的那个字符` charCodeAt() `alert(strObj.charCodeAt(1)//返回的是下表为1的那个位置上的字符的字符编码`
    - concat() 字符串拼接 方法应用的字符串不发生改变

```
var strVal = 'hello ';
var result = strVal.concat('world');
alert(result);//'hello world'
alert(strVal);//'hello'
```

- slice()
- substr()
- substring()
- indexOf() 从头开始遍历字符串 返回字符所在字符串的下标 indexOf('o',6) 第二个参数为搜索开始的位置（包括）
- lastIndexOf() 从尾开始遍历字符串 返回字符所在字符串的下标
- trim() 创建一个删除前置和后缀所有空格的副本

```
var strVal = '  hello world   ';
var trimStrVal = strVal.trim();
alert(trimStrVal);//'hello world'
alert(strVal);'  hello world   '
```

- 大小写转换 toLowerCase() toUpperCase()
- 字符串的模式匹配方法
  - match() 接收一个正则表达式或者是一个RegExp对象

```
var text = 'cat bat sat fat';
var pattern = /.at/;

var matches = text.match(pattern);
alert(matches.index);//0
alert(matches[0]);//cat
alert(pattern.lastIndex);//0
```

  - search() 有匹配就返回1 没有就-1
  - replace() 正则表达式+g可以全部替换

```
  var text =  'cat bat sat fat';
  var result = text.replace('at','ond');
  alert(result);//cond bat sat fat

  result = text.replace(/at/g,'ond');
  alert(result);//cond bond sond fond
```

    - localeCompart() 如果字符串在排在参数字符串之前则返回一个复数（多数情况下是-1） 等于则返回0 之后则返回正数（多数情况下是1）

# 单体内置对象

- Global对象
- Math对象