02-第二章 ECMAScript6 字符串及数值的拓展

ECMAscript 6

> # 一、字符串拓展
>
> ES6 加强了对 `Unicode` 的支持，并且扩展了字符串对象。

一、字符串的Unicode 表示法

JavaScript 允许采用`\uxxxx`形式表示一个字符，其中xxxx表示字符的 Unicode 码点。

```
var str = '\u0061';
        console.log(str);//a
```

但是，这种表示法只限于码点在`\u0000~\uFFFF`之间的字符。超出这个范围的字符，必须用两个`双字节`的形式表示。

```
var str = '\uD842\uDFB7';
        console.log(str);//吉
```

```
var str = '\u20BB7';
        console.log(str);//₻7
```

上面代码表示，如果直接在\u后面跟上超过`0xFFFF`的数值（比如`\u20BB7`），JavaScript会理解成`\u20BB+7`。由于`\u20BB`是一个不可打印字符，所以只会显示一个空格，后面跟着一个7。

ES6 对这一点做出了改进，只要将码点放入`{}`大括号，就能正确解读该字符
var str = '\u{20BB7}';
console.log(str);//吉

二、codePointAt() 返回十进制码点

> JavaScript 内部，字符以 UTF-16 的格式储存，每个字符固定为2个字节。对于那些需要4个字节储存的字符（Unicode 码点大于0xFFFF的字符），JavaScript 会认为它们是两个字符。

```
var s = '吉';
console.log( s.length);//2
console.log(  s.charAt(0) );//
console.log(  s.charAt(1) );//
console.log(  s.charCodeAt(0) );//55362
console.log(  s.charCodeAt(1) );//57271
```

> ES6 提供了codePointAt方法，`能够正确处理 4 个字节储存的字符`，返回一个字符的码点。

```
 var s = '吉a';
         console.log( s.length );//3
        console.log( s.codePointAt(0) );//134071
        console.log( s.codePointAt(0).toString(16) );//20bb7
         console.log( '\u{20bb7}' );//吉
        console.log( s.codePointAt(1) );//57271
        console.log( s.codePointAt(1).toString(16) );//dfb7
        console.log( '\u{dfb7}' );//
        console.log( s.codePointAt(2) );//97
        console.log( s.codePointAt(2).toString(16) );//61
        console.log( '\u{61}' );//a
```

上面代码中，字符a在字符串s的正确位置序号应该是 1，但是必须向codePointAt方法传入 2。解决这个问题的一个办法是使用`for...of`循环
因为它会正确识别 32 位的 UTF-16 字符。

```
let s = '吉a';

        for(let ch of s){
            console.log(ch); //吉a
            console.log(ch.codePointAt(0).toString(16));//20bb7
61
        }
```

codePointAt方法是测试一个字符由两个字节还是由四个字节组成的最简单方法。

```
function is32Bit(c){
            return c.codePointAt(0) > 0xFFFF;

        }
        var r = is32Bit('吉');
        console.log( r );//true
        var r = is32Bit('a');
        console.log( r );//true
```

三.String.fromCodePoint()

> ES6 提供了`String.fromCodePoint`方法，可以识别大于0xFFFF的字符，弥补了String.fromCharCode方法的不足。在作用上，正好与codePointAt方法相反。

```
        var r = String.fromCharCode(0x20BB7);
        console.log(r);//𠮷
        var r = String.fromCodePoint(0x20bb7);
         console.log(r);//吉
```

tring.fromCharCode不能识别大于0xFFFF的码点，所以0x20BB7就发生了溢出，最高位2被舍弃了，最后返回码点U+0BB7对应的字符，而不是码点U+20BB7对应的字符。

四、for of字符串的遍历器

ES6 为字符串添加了遍历器接口,使得字符串可以被`for...of`循环遍历

```
let text = String.fromCodePoint(0x20BB7);
console.log(text.length);//2
for(var i=0;i< text.length;i++){
    console.log(text[i]);//
}
for(let codePoint of text){
    console.log(codePoint);//吉
}
```

五、includes(),startsWidth(),endsWith()

传统上，JavaScript 只有`indexOf`方法，可以用来确定一个字符串是否包含在另一个字符串中。ES6 又提供了三种新方法。

- `includes()`：返回布尔值，表示是否找到了参数字符串。
- `startsWith()`：返回布尔值，表示参数字符串是否在原字符串的头部。
- `endsWith()`：返回布尔值，表示参数字符串是否在原字符串的尾部。

```
let s = 'hello world';
console.log( s.startsWith('ok') );//false
console.log( s.endsWith('d') );//true
console.log(s.includes('o'))//true


```

六、 repeat() 定义字符串出现次数

```
var str = 'window';
console.log( str.repeat(2) );//windowwindow
console.log( str);//window
```

如果repeat的参数是字符串，则会先转换成数字。

```
var str = 'window';
console.log( str.repeat('3') );//windowwindowwindow

```

七、padStart() ,padEnd()

str.padStart(num, '添加字符')
str不足num长度，在str添加字符

```
var s = 'x'.padStart(5,'ab');
console.log( s );//ababx
var s = 'x'.padStart(6,'ab');
console.log( s );//ababax
```

```
var str = 'x'.padEnd(4,'666');
console.log(str);//x666
```

padStart的常见用途是为数值补全指定位数。下面代码生成 10 位的数值字符串。

```
var str = '1'.padStart(10,'0');
console.log( str );//0000000001
```

八、模板字符串

传统的 JavaScript 语言，输出模板通常是这样写的。

```
 var num =  2;
    var content = 'today is 星期'+num+'了吗？';
     console.log( content);//today is 星期2了吗？
```

模板字符串（template string）是增强版的字符串，用反引号（ ` ）标识。它可以当作普通字符串使用，也可以用来定义多行字符串，或者在字符串中嵌入变量。
模板字符串中嵌入变量，需要将变量名写在`${}`之中。

```
 var num =  2;
    var content = `today is 星期${num}了吗？`;
    console.log( content);//today is 星期2了吗？
```

```
var str = '1'.padStart(10,'0');
console.log( str );//0000000001
```

九、标签模板

模板字符串的功能，不仅仅是上面这些。它可以紧跟在一个函数名后面，该函数将被调用来处理这个模板字符串。这被称为“标签模板”功能（`tagged template`）

```
// alert(123);
    alert `123`;
```

```
fn`hello ${123} world ${456}s`;
function fn(arr,value1,value2){

        console.log( arr );

        console.log(value1,value2);//123 456

    }
//等价于
    function fn(arr,...agr){//...agr剩余参数

        console.log( arr );
        console.log(...agr);//123 456

    }


```

fn函数所有参数的实际值如下。

> 第一个参数：[‘Hello ‘, ’ world ‘, ‘s’]
> 第二个参数: 123
> 第三个参数：456
> 第一个参数是标签模板中非变量的字符串，第二个参数是所有的保留

也就是说，tag函数实际上以下面的形式调用。

```
fn(['Hello ', ' world ', 's'], 123, 456);
```

二、数值拓展

一、Number.`isFinite()` Number.`isNaN()`

> Number.isFinite() 用来检查一个数值 是否为有限的（ finite ）

```
var r = Number.isFinite(1);
    console.log(r);//true
    var r = Number.isFinite(0.8);
    console.log(r);//true
    var r = Number.isFinite(Infinity);//无穷大
    console.log(r);//false
    var r = Number.isFinite(NaN);
    console.log(r);//false
    var r = Number.isFinite('ok');
    console.log(r);//false
```

Number.isFinite()对于`非数值一律返回false`

> Number.isNaN()用来检查一个值 `是否为NaN`

```
var r = Number.isNaN(NaN);
    console.log(r);//true
    var r = Number.isNaN(2);
    console.log(r);//false
    var r = Number.isNaN(true);
    console.log(r);//false
    var r = Number.isNaN(undefined);
    console.log(r);//false
    var r = Number.isNaN('ok');
    console.log(r);//false
```

Number.isNaN()只有对于NaN才返回true，`非NaN一律返回false`

它们与传统的全局方法isFinite()和isNaN()的区别在于，传统方法先调用`Number()`将非数值的值转为数值，再进行判断

二、Number.parseInt() Number.parseFloat()

> ES6 将全局方法parseInt()和parseFloat()，移植到Number对象上面，行为完全保持不变。
> 这样做的目的，是逐步减少全局性方法，使得语言逐步模块化。

```
var r = parseInt('12.34');
console.log( r );//12
var r = parseFloat('12.345#');
console.log( r );//12.345

//--------------es6写法

var r = Number.parseInt('12.34');
console.log( r );//12
var r = Number.parseFloat('12.345#');
console.log( r );//12.345
```

三、`Number.isInteger()` 是否为整数

> 用来判断一个值是否为整数。需要注意的是，在 JavaScript 内部，整数和浮点数是同样的储存方法，所以 3 和 3.0 被视为同一个值。

```
var r = Number.isInteger(8);
    console.log(r);//true
    var r = Number.isInteger(8.0);
    console.log(r);//true
    var r = Number.isInteger(8.1);
    console.log(r);//false
    var r = Number.isInteger(-8);
    console.log(r);//true
    var r = Number.isInteger('18');
    console.log(r);//false
    var r = Number.isInteger(true);
    console.log(r);//false
```

四、`Number.isSafeInteger()` 安全整数

> JavaScript 能够准确表示的整数范围在-2^53到`2^53`之间（`不含两个端点`），超过这个范围，无法精确表示这个值。

```
var a = Math.pow(2,53);
    console.log(a);//9007199254740992
    console.log(a+1);//9007199254740992
```

上面代码中，超出 2 的 53 次方之后，一个数就不精确了

`Number.MAX_SAFE_INTEGER` `Number.MIN_SAFE_INTEGER` 这两个常量，用来表示这个范围的上下限

```
var r = Number.MAX_SAFE_INTEGER == Math.pow(2,53)-1;
console.log( r );//true
var r = Number.MAX_SAFE_INTEGER == 9007199254740991;
console.log( r );//true
var r = Number.MAX_SAFE_INTEGER == -Number.MIN_SAFE_INTEGER;
console.log( r );//true
```

五、`Math.sign()` 正数 负数 0 -0 NaN

Math.sign方法用来判断一个数到底是正数、负数、还是零。对于非数值，会先将其转换为数值。

它会返回五种值。

参数为正数，返回`+1`；
参数为负数，返回`-1`；
参数为 0，返回`0`；
参数为-0，返回`-0`;
其他值，返回`NaN`。

```
var r = Math.sign(8);
   console.log( r );//1
   var r = Math.sign(-8);
   console.log( r );//-1
   var r = Math.sign('-8');
   console.log( r );//-1
   var r = Math.sign(0);
   console.log( r );//0
   var r = Math.sign(-0);
   console.log( r );//-0
   var r = Math.sign('ok');
   console.log( r );//NaN
```

@(总结：)

```
一、字符串的Unicode表示法
        '\u0000'--'\uFFFF'
        '\u0061';
        '\u{20BB7}'


    charAt(0)
    charCodeAt(0)
二、codePointAt(0).toString(16); 十进制代码

    String.fromCharCode();
三、String.fromCodePoint(0x20BB7);
四、for of

五、includes(),startsWidth(),endsWith();
六、repeat();
    str.repeat();
七、padStart(num,'字符串') padEnd(num,'字符串');

八、模板字符串
    `hello world`;
     `hello ${num } world`

九、标签模板
    fn `123`;
    fn ` hello ${123} world ${456}`;


十、 Number.isFinite() Number.isNaN()
        isFinite()  isNaN()

十一、Number.parseInt() number.parFloat()
         parseInt() number.parseFloat();

十二、Number.isInteger();是否为整数
    ---javascript内部3 等同于 3.0
十三、Number.isSafeInteger()是否为安全整数
    -Math.pow(2.53) ~ Math.pow(2.53)不包含两端点

    Number.MAX_SAFE_INTEGER 常量 Number.MIN_SAFE_INTEGER 常量
十四、Math.sign(); 正值 负值 0 -0 NaN
```