JavaScript 语言精粹六（JS 方法集）

年年有风 风吹年年 慢慢即漫漫 —— 夏吉尔硕

起步：

本章节为上章节补充，上个章节以数组为主题，介绍数组的常用方法集，👉查看数组

本章节用于简单描述 `JavaScript` 中数组以外的数据类型的常用方法集。

Number

number.toExponential(fractionDigits)

- `toExponential` 方法是把数字类型转化为**指数形**字符串
- `fractionDigits` 可选 20 以内**小数位数**
- 例如：

```
var nums = 1024;
var trs = nums.toExponential(3);
console.log(trs); // 1.024e+3--->1.024*10的3次方
```

number.toFixed(fractionDigits)

- `toFixed` 方法是把数字类型转化为**十进制形**字符串
- `fractionDigits` 可选 20 以内**小数位数**
- 例如：

```
var nums = Math.PI;
var trs = nums.toFixed(3);
console.log(trs); //3.142
```

number.toPrecision(precision)

- `toPrecision` 方法是把数字类型转化为十进制数形式的字符串
- `precision` 可选 21 以内**数字总数**
- 例如：

```
var nums = Math.PI;
var trs = nums.toPrecision(3);
console.log(trs); // 3.14
```

number.toString(radix)

- `toString` 方法是把数字类型转化为字符串
- `radix`控制基数， 可选 2-36 以内,默认为 10
- 也可简写成 `String(number)`
- 例如：

```
var nums = 1024,
  nums2 = 9,
  nums3 = 125;
var trs = nums.toString(2);
var trs2 = nums2.toString(2);
var trs3 = nums2.toString(5);
console.log(trs); // 10000000000-->1*2的10次方
console.log(trs2); //1001
console.log(trs3); //1000
```

Function

function.apply(this.Ary , argArray)

- `apply` 方法是调用 `function`,传递一个会绑定到 `this` 上的对象和一个可选数组作为参数
- `apply` 方法被用在 `apply` 调用模式 `(apply invocation pattern)` 中
- 例如：

```
Function.prototype.fakeBind = function (that) {
  var method = this,
    slice = Array.prototype.slice,
    args = slice.apply(arguments, [1]);
  return function () {
    return method.apply(that, args.concat(slice.apply(arguments, [0])));
  };
};
var test = function () {
  return this.value;
}.fakeBind({ value: 233 });
console.log(test()); // 233
```

Object

object.hasOwnProperty(name)

- `hasOwnProperty` 方法是对调用此方法的对象身上的属性进行检查
- 若 `object` 包含一个名为 `name` 的属性，则此方法返回 **true**
- 注：此方法只会检查当前对象，不会进行原型链上属性检查

```
var list = {
  name: "poo",
  gender: "boy",
};
Object.prototype.list = list;
var test = new Object();
console.log(list.hasOwnProperty("name")); // true
console.log(test.hasOwnProperty("name")); // false
console.log(test.list.name); // poo
```

String

string.charAt(pos)

- charAt 方法返回在 string 中 pos 位置处的字符
- 若 pos 小于 0 或者不在字符串长度范围内，则返回空字符串
- 注：JavaScript 无字符类型，所以此方法返回结果为字符串

```
let type = "food";
var pick = type.charAt(3); // d
/** 实现原理 */
String.prototype.fakeCharAt = function (pos) {
  return this.slice(pos, pos + 1);
};
var ahh = "yaHoo";
let pick = ahh.fakeCharAt(2); // H
console.log(typeof pick); // string
```

string.charCodeAt(pos)

- charCodeAt 与 charAt 相似，返回的是 string 中 pos 位置处的字符码位
- 若 pos 小于 0 或者不在字符串长度范围内，则返回 NaN

```
let type = "food";
var pick = type.charCodeAt(3); // 100
console.log(typeof pick); // number
```

string.concat(string...)

- concat 方法用于进行字符串拼接，且构造出新字符串
- 注：concat 性能和使用便捷性都不如直接使用运算符 + ，因此几乎不用

```
let name = "iPhone ";
let add = "Plus ";
var pick = name.concat(add); // iPhone Plus
```

```
/** 猜测实现原理 */
String.prototype.fakeConcat = function (add) {
  return this + add;
};
let name = "tomato";
var xxx = name.fakeConcat(" Plus"); // tomato Plus
```

string.indexOf(searchString , position)

- `indexOf` 方法用于在当前字符串中查找另一个字符串 `searchString`
- 找到则返回匹配字符串的位置，否则返回 **-1**
- `position` 为可选参数，可指定查找起始位置

```
let str = "xipengheng";
let pick1 = str.indexOf("eng"); // 3
let pick2 = str.indexOf("ning"); // -1
let pick3 = str.indexOf("eng", 4); // 7
```

string.lastIndexOf(searchString , position)

- `lastIndexOf` 与 `indexOf`相同，不过它是倒序查找自定字符串
- 找到则返回匹配字符串的位置，否则返回 **-1**
- `position` 为可选参数，可指定查找起始位置

```
let str = "xipengheng";
let pick1 = str.lastIndexOf("eng"); // 7
let pick2 = str.lastIndexOf("ning"); // -1
let pick3 = str.lastIndexOf("eng", 4); // 3
```

string.localeCompare(that)

- `localeCompare` 方法是比较两个字符串
- 若当前字符串小于 `that` 则为负，相等为 `0`
- 注：这方法没啥规则，也没啥用途，没学习的必要

string.replace(searchValue , replaceValue)

- `replace` 方法会对 `searchVale` 进行查找，并且使用 `replaceVale` 替换
- 注：只会替换首次出现的搜索词组，且返回的是一个新字符串

```
let str = "xipengheng";
let newStr = str.replace("eng", "a"); // xipaheng
```

- 此方法也可与正则组合，根据需要替换一个或多个字符

```
let str = "xipengheng";
let reg = /eng/;
let regGlobal = /eng/g;
let newStr = str.replace(reg, "a"); // xipaheng
let gloStr = str.replace(regGlobal, "a"); // xipaha
```

string.search(regexp)

- search 方法类似于 indexOf 方法，但它只接受一个正则对象
- 找到则返回首个匹配的字符位置，否则返回 **-1**
- 此方法会自动忽略全局标识符号 g ，且没有 position 参数

```
let str = "i miss you,and you ?";
let reg = /you/;
let regGlobal = /you/g;
let newStr = str.search(reg); // 7
let gloStr = str.search(regGlobal); // 7
```

string.slice(start,end)

- slice 方法复制 string 的一部分来构造一个新字符串
- 若 start 参数为负数，它将**与 string.length 相加**
- end 为要取的**最后一个字符的位置+1**

```
let str = "0123456789"; //length=10
let newStr = str.slice(0, 6); // 012345
let erroStr = str.slice(-3); // 789
```

string.split(separator,limit)

- split 方法把这个 string分隔成片段创建一个数组
- separator 允许是字符串或正则表达式，若为空则返回单字符数组
- limit 会限制被分割的片段数量
- 注：**此方法的使用频率挺高的**

```
let str = "abcde";
let newArr1 = str.split(); // [ 'abcde' ]
let newArr2 = str.split(""); // [ 'a', 'b', 'c', 'd', 'e']
let flagArr = str.split("", 3); // [ 'a', 'b', 'c' ]
let pickArr = str.split("b"); // [ 'a', 'cde' ]
```

string.substring(start,end)

- substring 方法与 slice 作用相同，只是它不接受负数
- 注：**因此平时使用 slice 更好一些**

string.toLocaleLowerCase()& string.toLocaleUpperCase()

- toLocaleLowerCase 方法返回一个全新全**小写字符串**
- toLocaleUpperCase 方法返回一个全新全**大写字符串**
- 冷知识：**toLocaleUpperCase** 是用来处理土耳其语
- 注： i 大写是-->İ 而不是 I

```
let str = "abCdE";
let upStr = str.toLocaleUpperCase(); // ABCDE
let lowerStr = str.toLocaleLowerCase(); // abcde
```

string.fromCharCode()

- fromCharCode 方法是通过字符编码找到对应字符并拼接
- 注： 该方法返回一个字符串

```
let char = String.fromCharCode(67, 97, 116); //Cat
console.log(typeof char); // string
```

总结：

> 本章节描述的是Function、Number、String、Object 等常用方法集，并且为便于理解，模拟出了官方 API 实现的原理（*大概思路是* 我不保证一定是，但能用，emm）。