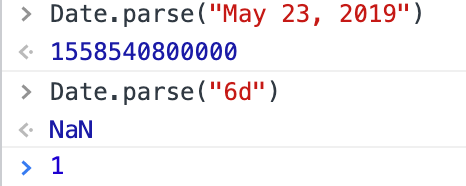
# 第五章 基本引用类型

## 1、Date

自协调世界时(UTC，Universal Time Coordinated)

格林威治时间 Greenwich Mean Time (GMT)

#### Date.parse()



如果直接把表示日期的字符串传给 Date 构造函数，那么 Date 会在后台先调用Date.parse()。

#### Date.UTC()

计时

// 起始时间  
 let start = Date.now();  
  
// 调用函数 doSomething();  
  
// 结束时间  
 let stop = Date.now(), result = stop - start;

日期比较会调用valueOf()方法，这个方法在Date被重写，返回的是日期的毫秒，与getTime()相同

## 2、RegExp

粘附标记 y，每次调用 exec()就只会在 lastIndex 的位置上寻找匹配项。

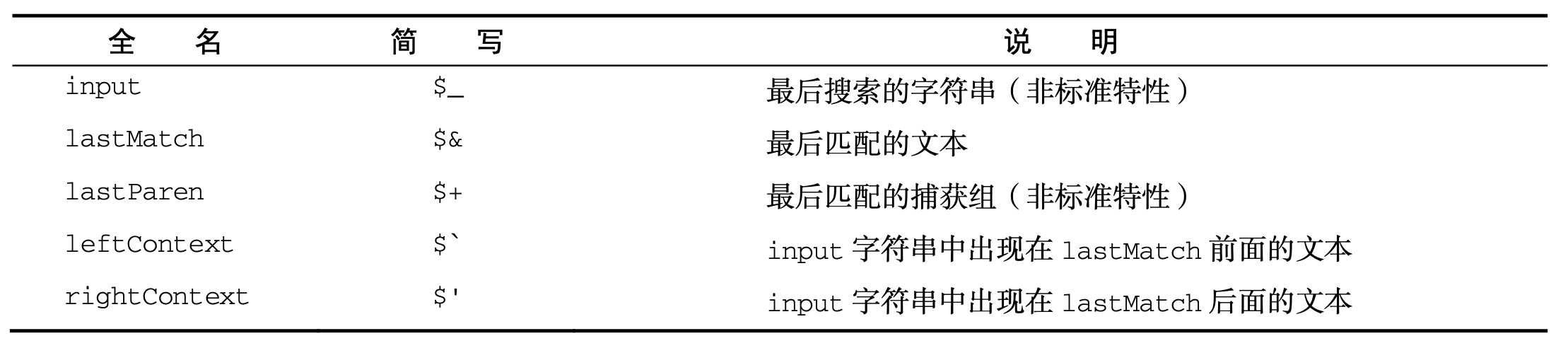
let text = "cat, bat, sat, fat";  
let pattern = /.at/y;  
let matches = pattern.exec(text);  
console.log(matches.index); // 0   
console.log(matches[0]); // cat   
console.log(pattern.lastIndex); // 3  
// 以索引3对应的字符开头找不到匹配项，因此exec()返回null   
// exec()没找到匹配项，于是将lastIndex设置为0  
matches = pattern.exec(text); console.log(matches);//null  
console.log(pattern.lastIndex); // 0  
  
  
// 向前设置lastIndex可以让粘附的模式通过exec()找到下一个匹配项:   
 pattern.lastIndex = 5;  
 matches = pattern.exec(text);  
 console.log(matches.index); //5  
 console.log(matches[0]); //bat  
 console.log(pattern.lastIndex); //8

无论正则表达式是怎么创建的，继承的方法 toLocaleString()和 toString()都返回正则表达式的字面量表示。

let pattern = new RegExp("\\[bc\\]at", "gi");  
console.log(pattern.toString()); // /\[bc\]at/gi  
console.log(pattern.toLocaleString()); // /\[bc\]at/gi

正则表达式的valueOf()方法返回正则表达式本身。(正则对象)

RegExp的一些属性与简写



let text = "this has been a short summer";

let pattern = /(.)hort/g;  
 if (pattern.test(text)) {  
 console.log(RegExp.input); // this has been a short summer  
 console.log(RegExp.leftContext); // this has been a  
 console.log(RegExp.rightContext); // summer  
 console.log(RegExp.lastMatch); // short  
 console.log(RegExp.lastParen); // s  
 }

if (pattern.test(text)) {  
 console.log(RegExp.$\_); // this has been a short summer  
 console.log(RegExp["$`"]); // this has been a  
 console.log(RegExp["$'"]); // summer  
 console.log(RegExp["$&"]); // short  
 console.log(RegExp["$+"]); //s  
}

RegExp. $1~RegExp.$9分别包含第 1~9 个捕获组的匹配项。在调用 exec()或 test()时，这些属性就会被填。.

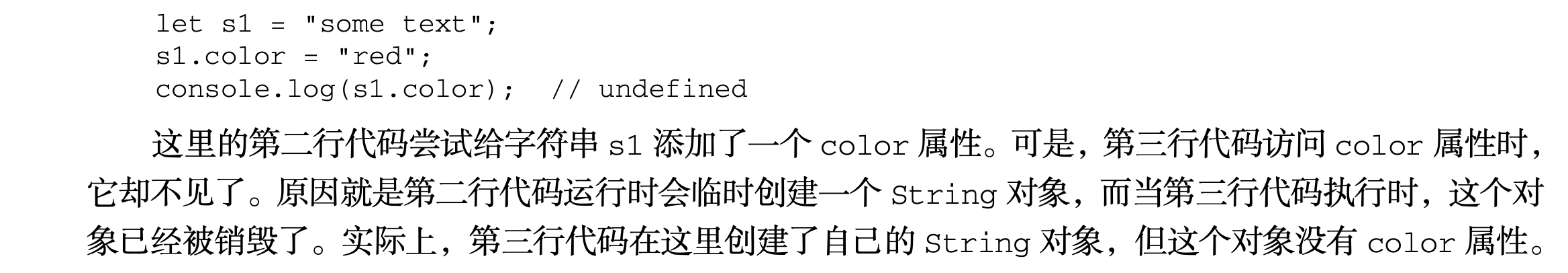
let text = "this has been a short summer";  
 let pattern = /(..)or(.)/g;  
 if (pattern.test(text)) {  
 console.log(RegExp.$1); // sh  
 console.log(RegExp.$2); // t  
 }

注意：RegExp 构造函数的所有属性都没有任何 Web 标准出处，因此不要在生产环境中使用它们。

### 3、原始值包装类型

①每当用到某个原始值的方法或属性时，后台都会创建一个相应原始包装类型的对象，从而暴露出操作原始值的各种方法。

②自动创建的原始值包装对象则只存在于访问它的那行代码执行期间。举例：



③Object 构造函数作为一个工厂方法，能够根据传入值的类型返回相应原始值包装类型的实例。

let obj = new Object("some text");  
console.log(obj instanceof String); // true

④Number的方法

num.toFixed(2)表示:保留两位小数（四舍五入）

num.toExponential(2)：科学计数表示，保留两位小数

Number.isInteger()方法，用于辨别一个数值是否保存为整数。

console.log(Number.isInteger(1)); // true  
console.log(Number.isInteger(1.00)); // true  
console.log(Number.isInteger(1.01)); // false

⑤String方法的规则

当某个参数是负值时，slice()方法将所有负值参数都当成length加上负参数值。

substr()方法将第一个负参数值当成length加上该值，将第二个负参数值转换为 0。

substring()方法会将所有负参数值都转换为 0

console.log(stringValue.substring(3, -4)); // "hel"  
//相当于调用 substring(3, 0)，等价于 substring(0, 3)  
//这是因为这个方法会将较小的参数作为起点，将较大的参数作为终点。

ES6新增的startsWith()、 endsWith()、includes()。

let message = "foobarbaz";  
console.log(message.endsWith("bar")); // false  
console.log(message.endsWith("bar", 6)); // true  
//endsWith第二个参数，默认是length。加了第二个参数后，就好像字符串的length被第二个参数替代了一样。

repeat()方法

let stringValue = "na ";  
console.log(stringValue.repeat(16)\*\* + "batman");  
// na na na na na na na na na na na na na na na na batman

padStart()和 padEnd()方法

第一个参数是长度，第二个参数是可选的填充字符串，默认为空格

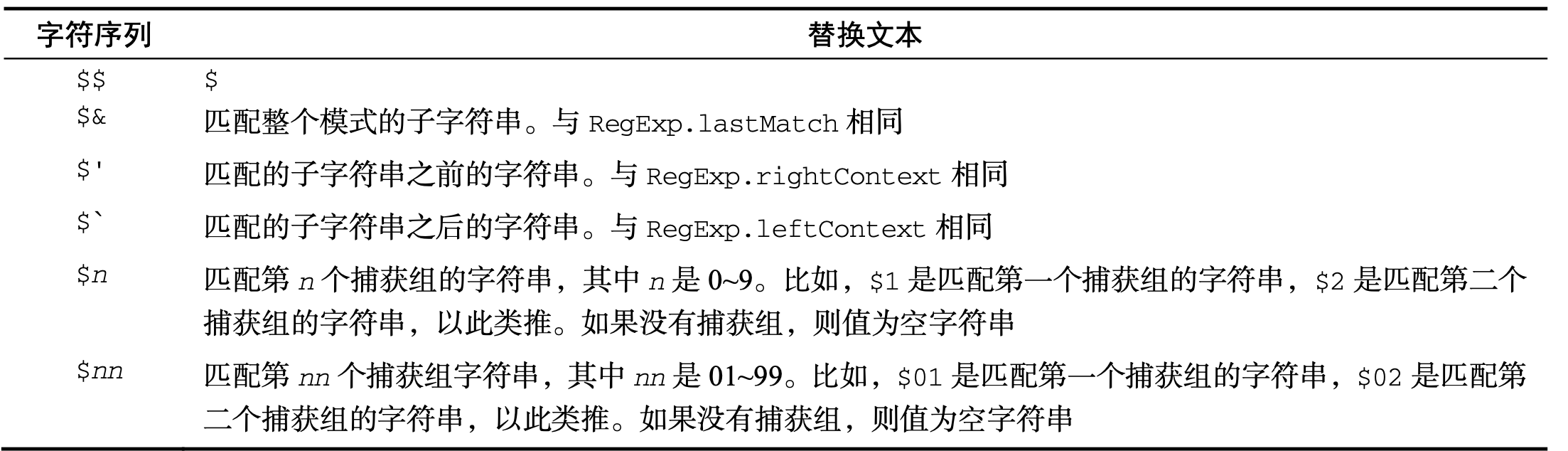
let stringValue = "foo";  
 console.log(stringValue.padStart(6)); // " foo"  
 console.log(stringValue.padStart(9, ".")); // "......foo"  
 console.log(stringValue.padEnd(6)); // "foo "  
 console.log(stringValue.padEnd(9, ".")); // "foo......"  
  
  
console.log(stringValue.padStart(8, "bar")); // "barbafoo"  
console.log(stringValue.padStart(2)); // "foo"  
console.log(stringValue.padEnd(8, "bar")); // "foobarba"  
console.log(stringValue.padEnd(2)); // "foo"，如果第一个参数小于字符串长度，则会返回原字符串

字符串的match方法跟 RegExp 对象的 exec()本质相同

search()传入的是唯一的参数也是正则，这个方法返回模式第一个匹配的位置索引，如果没找到则返回-1。

replace可以使用包含$的简写↓

let text = "cat, bat, sat, fat";  
result = text.replace(/(.at)/g, "word ($1)");  
console.log(result); // word (cat), word (bat), word (sat), word (fat)



replace,第二个参数使用函数的示例，可以作为一个经典replace例子参考

function htmlEscape(text) {  
 return text.replace(/[<>"&]/g, function(match, pos, originalText) {  
 switch(match) {  
 case "<":  
 return "&lt;";  
 case ">":  
 return "&gt;";  
 case "&":  
 return "&amp;";  
 case "\"":  
 return "&quot;";  
 }  
 }); }  
console.log(htmlEscape("<p class=\"greeting\">Hello world!</p>"));   
// "&lt;p class=&quot;greeting&quot;&gt;Hello world!</p>"

### 4、内置对象Global

①encodeURI()不会编码属于 URL 组件的特殊字符，比如冒号、斜杠、问号、 1 井号，而 encodeURIComponent()会编码它发现的所有非标准字符。

let uri = "http://www.wrox.com/illegal value.js#start";  
console.log(encodeURI(uri));  
// "http://www.wrox.com/illegal%20value.js#start"  
console.log(encodeURIComponent(uri));  
// "http%3A%2F%2Fwww.wrox.com%2Fillegal%20value.js%23start"   
  
  
//在decode时，decodeURI()只会decode出由encodeURI()转化的字符  
//decodeURIComponent()则解码所有被encodeURIComponent()编码的字符，基本上就是解码所有特殊值。  
  
  
let uri = "http%3A%2F%2Fwww.wrox.com%2Fillegal%20value.js%23start";  
// http%3A%2F%2Fwww.wrox.com%2Fillegal value.js%23start  
console.log(decodeURI(uri));  
// http:// www.wrox.com/illegal value.js#start  
console.log(decodeURIComponent(uri));

②eval

当解释器发现 eval()调用时，会将参数解释为实际的 ECMAScript 语句，然后将其插入到该位置。 通过 eval()执行的代码属于**该调用所在上下文**，被执行的代码与该上下文拥有相同的作用域链。

但是

eval("let msg = 'hello world';");  
console.log(msg); // Reference Error: msg is not defined  
   
eval("var msg = 'hello world';");  
console.log(msg); // hello world

在严格模式下，在 eval()内部创建的变量和函数无法被外部访问。

③虽然 ECMA-262 没有规定直接访问 Global 对象的方式，但浏览器将 window 对象实现为 Global 对象的代理。window对象在JavaScript中远不止实现了ECMAScript的Global对象那么简单。

④Math.random()

获取[lowerValue, upperValue]的整数

function selectFrom(lowerValue, upperValue) {  
 let choices = upperValue - lowerValue + 1;  
 return Math.floor(Math.random() \* choices + lowerValue);  
 }  
 let num = selectFrom(2,10);  
 console.log(num); //2~10范围内的值，其中包含2和10