基本引用类型

ESQ缺少传统面向对象语言的某些结构，包括类和接口
对象被认为是某个特定引用类型的实例。
新对象通过使用new操作符后跟一个构造函数来创建

Date

库 date-format

ES的Date参考Java早期版本中的java.util.Date
日期保存为自协调世界时（UTC,Universal Time Coordinated）,1970.1.1零时所经过的毫秒数

```
// new调用Date构造函数
let now = new Date();
```

调用Date构造函数不传参，默认保存当前日期和时间。如果要为其他日期时间创建日期对象，必须传入毫秒级表示(虽然传日期字符串会后台调用Date.parse())

- Date.parse()
  接受表示日期的字符串参数，转成毫秒级数

  - 月/年/日 e.g., "5/23/2019"
  - 月 日，年 e.g.,"May 23,2019"
  - 周几 月名 日 年 时:分:秒 时区 e.g.,"Tue May 23 2019 00:00:00 GMT-0700"
  - ISO 8601扩展模式 "YYYY-MM-DDTHH:mm:ss:sssZ"(只适用兼容ES5的实现)

  如果Date.parse()字符串不表示日期，该方法返回NaN。如果直接把日期字符串给Date构造函数，Date会在后台调用Date.parse()

- Date.UTC()
  Date.UTC也返回日期的毫秒级表示
  传参 年、月、日、时、分、秒、毫秒
  月份0起点，如果不提供参数，会默认1日，其他都默认

- Date.now()
  返回方法执行时日期和时间的毫秒数

Date.parse()和Date.UTC()都会被Date 构造函数隐式调用

继承的方法

Date类型重写了toLocaleString()、toString()、valueOf()方法

- toLocaleString() 返回与浏览器运行的本地环境一致的日期和时间(AM,PM 但是没有时区，具体根据浏览器不同而不同)
- toString() 返回带时区信息的日期和时间，而时间也是以 24 小时制（0~23）表示的
- valueOf() 不返回字符串，返回的是日期的毫秒表示

```
toLocaleString() - 2/1/2019 12:00:00 AM
toString() - Thu Feb 1 2019 00:00:00 GMT-0800 (Pacific Standard Time)
```

日期格式化方法

返回值都是字符串

- toDateString()显示日期中的周几、月、日、年（格式特定于实现）
- toTimeString()显示日期中的时、分、秒和时区（格式特定于实现）
- toLocaleDateString()显示日期中的周几、月、日、年（格式特定于实现和地区）
- toLocaleTimeString()显示日期中的时、分、秒（格式特定于实现和地区）
- toUTCString()显示完整的 UTC 日期（格式特定于实现）

日期/时间组件方法

方　法	说　明
getTime()	返回日期的毫秒表示；与 valueOf()相同
setTime(*milliseconds*)	设置日期的毫秒表示，从而修改整个日期
getFullYear()	返回 4 位数年（即 2019 而不是 19）
getUTCFullYear()	返回 UTC 日期的 4 位数年
setFullYear(*year*)	设置日期的年（*year* 必须是 4 位数）
setUTCFullYear(*year*)	设置 UTC 日期的年（*year* 必须是 4 位数）
getMonth()	返回日期的月（0 表示 1 月，11 表示 12 月）
getUTCMonth()	返回 UTC 日期的月（0 表示 1 月，11 表示 12 月）
setMonth(*month*)	设置日期的月（*month* 为大于 0 的数值，大于 11 加年）
setUTCMonth(*month*)	设置 UTC 日期的月（*month* 为大于 0 的数值，大于 11 加年）
getDate()	返回日期中的日（1~31）
getUTCDate()	返回 UTC 日期中的日（1~31）
setDate(*date*)	设置日期中的日（如果 *date* 大于该月天数，则加月）
setUTCDate(*date*)	设置 UTC 日期中的日（如果 *date* 大于该月天数，则加月）
getDay()	返回日期中表示周几的数值（0 表示周日，6 表示周六）
getUTCDay()	返回 UTC 日期中表示周几的数值（0 表示周日，6 表示周六）
getHours()	返回日期中的时（0~23）
getUTCHours()	返回 UTC 日期中的时（0~23）
setHours(*hours*)	设置日期中的时（如果 *hours* 大于 23，则加日）

方　法	说　明
setUTCHours(*hours*)	设置 UTC 日期中的时（如果 *hours* 大于 23，则加日）
getMinutes()	返回日期中的分（0~59）
getUTCMinutes()	返回 UTC 日期中的分（0~59）
setMinutes(*minutes*)	设置日期中的分（如果 *minutes* 大于 59，则加时）
setUTCMinutes(*minutes*)	设置 UTC 日期中的分（如果 *minutes* 大于 59，则加时）
getSeconds()	返回日期中的秒（0~59）
getUTCSeconds()	返回 UTC 日期中的秒（0~59）
setSeconds(*seconds*)	设置日期中的秒（如果 *seconds* 大于 59，则加分）
setUTCSeconds(*seconds*)	设置 UTC 日期中的秒（如果 *seconds* 大于 59，则加分）
getMilliseconds()	返回日期中的毫秒
getUTCMilliseconds()	返回 UTC 日期中的毫秒
setMilliseconds(*milliseconds*)	设置日期中的毫秒
setUTCMilliseconds(*milliseconds*)	设置 UTC 日期中的毫秒
getTimezoneOffset()	返回以分钟计的 UTC 与本地时区的偏移量（如美国 EST 即“东部标准时间”返回 300，进入夏令时的地区可能有所差异）

RegExp

let expression = /pattern/flags;
pattern 模式：可以是任何简单或复杂的正则表达式，包括字符类、限定符、分组、向前查找和反向引用
每个正则表达式可以带零个或者多个flags，用于控制正则表达式的行为
flags 标记：

- g：全局模式，表示查找字符串的全部内容，而不是找到第一个匹配的内容就结束

- i：不区分大小写，表示在查找匹配时忽略 pattern 和字符串的大小写
- m：多行模式，表示查找到一行文本末尾时会继续查找
- y：粘附模式，表示只查找从 lastIndex 开始及之后的字符串
- u：Unicode 模式，启用 Unicode 匹配
- s：dotAll 模式，表示元字符.匹配任何字符（包括\n 或\r）

所有元字符在pattern中必须转义包括
([{ \ ^ $ |)] } ? * + .

字面量模式	对应的字符串
/\[bc\]at/	"\\[bc\\]at"
/\.at/	"\\.at"
/name\/age/	"name\\/age"
/\d.\d{1,2}/	"\\d.\\d{1,2}"
/\w\\hello\\123/	"\\w\\\\hello\\\\123"

RegExp可以基于已有的正则表达式实例，选择性地修改他们地标记：

```
const re1 = /cat/g;
console.log(re1); // "/cat/g"
const re2 = new RegExp(re1);
console.log(re2); // "/cat/g"
const re3 = new RegExp(re1, "i");
console.log(re3); // "/cat/i"
```

RegExp 实例属性

- global：布尔值，表示是否设置了 g 标记
- ignoreCase：布尔值，表示是否设置了 i 标记
- unicode：布尔值，表示是否设置了 u 标记
- sticky：布尔值，表示是否设置了 y 标记
- lastIndex：整数，表示在源字符串中下一次搜索的开始位置，始终从 0 开始
- multiline：布尔值，表示是否设置了 m 标记
- dotAll：布尔值，表示是否设置了 s 标记
- source：正则表达式的字面量字符串（不是传给构造函数的模式字符串），没有开头和结尾的斜杠
- flags：正则表达式的标记字符串。始终以字面量而非传入构造函数的字符串模式形式返回（没有前后斜杠）

RegExp 实例方法

实例的主要方法是exec()
只接受一个参数，即要应用模式的字符串
如果找到了匹配项，返回第一个匹配信息的数组，否则返回null
返回的数组有额外的属性：

- index 匹配模式的其实位置
- input 查找的字符串

```
let text = "mom and dad and baby";
let pattern = /mom( and dad( and baby)?)?/gi;
let matches = pattern.exec(text);
console.log(matches.index); // 0
console.log(matches.input); // "mom and dad and baby"
console.log(matches[0]); // "mom and dad and baby"
console.log(matches[1]); // " and dad and baby"
console.log(matches[2]); // " and baby"
```

模式上设置了 g 标记，则每次调用 exec()都会在字符串中向前搜索下一个匹配项

```
let text = "cat, bat, sat, fat";
let pattern = /.at/g;
let matches = pattern.exec(text);
console.log(matches.index); // 0
console.log(matches[0]); // cat
console.log(pattern.lastIndex); // 3
matches = pattern.exec(text);
console.log(matches.index); // 5
console.log(matches[0]); // bat
console.log(pattern.lastIndex); // 8
matches = pattern.exec(text);
console.log(matches.index); // 10
console.log(matches[0]); // sat
console.log(pattern.lastIndex); // 13
```

另一个方法是test()，接收字符串参数。
如果输入与模式匹配，返回true，否则返回false

RegExp 构造函数属性

RegExp构造函数本身也有几个属性 (其它语言中被称为静态属性)

全　名	简　写	说　明
input	$_	最后搜索的字符串（非标准特性）
lastMatch	$&	最后匹配的文本
lastParen	$+	最后匹配的捕获组（非标准特性）
leftContext	$`	input 字符串中出现在 lastMatch 前面的文本
rightContext	$'	input 字符串中出现在 lastMatch 后面的文本

- 属性适用于作用域中的所有正则表达
- 会根据最后执行的正则表达式操作而变化
- 通过两种不同的方式去访问
  全名/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
}
```

也可以替换成简写形式，只不过要使用中括号语法来访问，因为不是合法的ES标识符

```
let text = "this has been a short summer";
let pattern = /(.)hort/g;
/*
 * 注意：Opera 不支持简写属性名
 * IE 不支持多行匹配
 */
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
}
```

还有过 RegExp.$1~RegExp.$9，包含第1-9个匹配项

```
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
}
```

模式局限

- \A 和\Z 锚（分别匹配字符串的开始和末尾）
- 联合及交叉类
- 原子组
- x（忽略空格）匹配模式
- 条件式匹配
- 正则表达式注释