正则表达式 RegExp

创建RegExp对象

使用RegExp构造函数

```
var patt1=new RegExp(pattern,flags);
```

直接通过 /正则表达式/ 写出来

```
var patt2= /pattern/ flags;
```

转义

元字符要进行转义

`( ) [ ] { } \ ^ $ | ? * + . _`

构造函数中某些情况要双重转义

`\` 被转义为 `\\`

正则表达式

直接给出字符，就是精确匹配

`\d` 可以匹配一个数字

`\w` 可以匹配一个字母或数字

`\s` 可以匹配一个空格

`\n` 可以匹配一个换行符

`.` 可以匹配任意字符

- 'js.'可以匹配'jsp'、'jss'、'js!'等等。

`*` 表示任意个字符（包括0个）

`+` 表示至少一个字符

`?` 表示0个或1个字符

`{n}` 表示n个字符

`{n,m}` 表示n-m个字符

`[ ]` 用于查找某个范围内的字符

- [0-9a-zA-Z_]可以匹配一个数字、字母或者下划线；
- [0-9a-zA-Z_]+可以匹配至少由一个数字、字母或者下划线组成的字符串，比如'a100'，'0_Z'，'js2015'等等；
- [a-zA-Z_\$][0-9a-zA-Z_\$]*可以匹配由字母或下划线、$开头，后接任意个由一个数字、字母或者下划线、$组成的字符串，也就是JavaScript允许的变量名；
- [a-zA-Z_\$][0-9a-zA-Z_\$]{0, 19}更精确地限制了变量的长度是1-20个字符（前面1个字符+后面最多19个字符）。

`A|B` 可以匹配A或B

`^` 表示行的开头

- ^\d表示必须以数字开头。

`$` 表示行的结束

- \d$表示必须以数字结束。
- js也可以匹配'jsp'，但是加上^js$就变成了整行匹配，就只能匹配'js'了。

匹配标志

g (globle)：全局匹配，被应用于在字符串中查找所有可能的匹配，匹配到目标串的结尾，返回的结果可以是多个。

i (ignorCase)：忽略大小写匹配

m (mutiple)：多行匹配

属性和方法

exec(字符串) 捕获组。匹配成功后，返回一个数组，匹配失败时返回null。

() 表示要提取的分组

```
var re = /^(\d{3})-(\d{3,8})$/;
re.exec('010-12345'); // ['010-12345', '010', '12345']
re.exec('010 12345'); // null
```

test(字符串) 判断正则表达式是否匹配。匹配则返回 true，否则返回 false。

贪婪匹配

正则匹配默认是贪婪匹配，也就是匹配尽可能多的字符。

```
var re = /^(\d+)(0*)$/;
re.exec('102300'); // ['102300', '102300', '']
```

由于\d+采用贪婪匹配，直接把后面的0全部匹配了，结果0*只能匹配空字符串了。

采用非贪婪匹配（也就是尽可能少匹配），加个 ? 就可以。

```
var re = /^(\d+?)(0*)$/;
re.exec('102300'); // ['102300', '1023', '00']
```

正则匹配案例

特殊字符集合，除去_

```
new RegExp("[`~!@#$^&*()=|{}':;',\\[\\].<>/?~！@#￥……&*（）&mdash;—|{}【】‘；：”“'。，、？]")
```