█基础

// 正则表达式两种定义方法：

// var re = new RegExp("a","i")

// var re = /a/i

// 正则表达式对象的方法：

// exec(); 执行匹配，返回找到值的数组，没找到返回null

// test(); 测试匹配，返回bool值

// toSource(); 返回RegExp对象源代码

// toString(); 返回RegExp对象字符串

var a = /abc/ //定义一个正则表达式对象a

var b = 'skj6abcifl'

var c = a.test(b) //使用正则表达式对象的test方法来测试字符串b

console.log(c) //true

█匹配单个字符

|  |  |
| --- | --- |
| **字符** | **功能** |
| . | 匹配任意1个字符（除了\n） |
| [ ] | 匹配[ ]中列举的字符 |
| \d | 匹配数字，即0-9 |
| \D | 匹配非数字，即不是数字 |
| \s | 匹配空白，即 空格，tab键 |
| \S | 匹配非空白 |
| \w | 匹配单词字符，即a-z、A-Z、0-9、\_ |
| \W | 匹配非单词字符 |

●匹配单个字符的概念，以上表格中的字符均代表一位字符

如下面的\d的用法

// \d代表的是一位数字

var a = /\d/

var b = /\d\d/

var c = 'abc2'

var d = 'abc22'

var e = a.test(c)

var f = a.test(d)

var g = b.test(c)

var h = b.test(d)

console.log(e) //true

console.log(f) //true

console.log(g) //false 字符串c没有两位连续的数字，所以不匹配，返回false

console.log(h) //true

●[] 的用法

// []表示一位字符中，可以匹配[]列举的字符

// 例如下面的变量a，可以匹配包含字符串 'a1b' 或者 'a2b' 或者 'a3b'

var a = /a[123]b/

var b = 'a2b'

var c = 'hha3bwww'

var d = 'a5b'

console.log(a.test(b)) //true

console.log(a.test(c)) //true

console.log(a.test(d)) //false

● \d和[]的组合运用

// 匹配速度与激情后面加一位任意数字的字符串

var a = /速度与激情\d/

console.log(a.test('速度与激情1')) //true

console.log(a.test('速度与激情2')) //true

console.log(a.test('速度与激情315')) //true

// 匹配速度与激情后面一位数字是1-8的字符串

var b = /速度与激情[12345678]/

console.log(b.test('速度与激情1')) //true

console.log(b.test('速度与激情3')) //true

console.log(b.test('速度与激情9')) //false

// 以上还可以简写为[1-8]

var c = /速度与激情[1-8]/

console.log(c.test('速度与激情2')) //true

console.log(c.test('速度与激情5')) //true

console.log(c.test('速度与激情9')) //false

●\d和[]的组合运用2

// 匹配速度与激情后跟一位1-8的数字或者a-d的字母

var a = /速度与激情[1-8a-d]/

console.log(a.test('速度与激情2')) //true

console.log(a.test('速度与激情b')) //true

console.log(a.test('速度与激情9')) //false

console.log(a.test('速度与激情f')) //false

// 匹配速度与激情后跟一位任意数字或者任意字母(大写和小写都行)

var b = /速度与激情[\da-zA-Z]/

console.log(b.test('速度与激情2')) //true

console.log(b.test('速度与激情9')) //true

console.log(b.test('速度与激情x')) //true

console.log(b.test('速度与激情w')) //true

console.log(b.test('速度与激情F')) //true

console.log(b.test('速度与激情K')) //true

●\w的用法

// \w可以匹配所有英文字母大写和小写，所有数字，和下划线

// 注意\w的匹配范围一般还可以匹配其他语言，比如汉字。但是在JS里，不能匹配其他语言

var a = /速度与激情\w/

console.log(a.test('速度与激情2')) //true

console.log(a.test('速度与激情h')) //true

console.log(a.test('速度与激情Q')) //true

console.log(a.test('速度与激情\_')) //true

console.log(a.test('速度与激情哈')) //false

console.log(a.test('速度与激情#')) //false 除下划线外的其他符号均不匹配

●\s的用法，用于匹配空格

// 匹配速度与激情加空格加数字

var a = /速度与激情\s\d/

console.log(a.test('速度与激情 2')) //true

●匹配任意单个字符

// 匹配任意单个字符

var a = /速度与激情./

console.log(a.test('速度与激情2')) //true

console.log(a.test('速度与激情a')) //true

console.log(a.test('速度与激情B')) //true

console.log(a.test('速度与激情\_')) //true

console.log(a.test('速度与激情&')) //true

console.log(a.test('速度与激情 w')) //true

█匹配多个字符

|  |  |
| --- | --- |
| **字符** | **功能** |
| \* | |  |  | | --- | --- | |  | 匹配前一个字符出现0次或者无限次，即可有可无 | |
| + | 匹配前一个字符出现1次或者无限次，即至少有1次 |
| ? | 匹配前一个字符出现1次或者0次，即要么有1次，要么没有 |
| {m} | |  |  | | --- | --- | |  | 匹配前一个字符出现m次 | |
| {m,n} | |  |  | | --- | --- | |  | 匹配前一个字符出现从m到n次 | |

●基本概念

匹配多个字符，如上表里的符号，是辅助匹配其前一个字符的

以{ m} 和{m,n}的用法为例

// 比如匹配一位数字可以用\d

// 匹配两位数字可以用\d\d

// 但是既可以一位也可以两位，怎么匹配？

// 下面是匹配速度与激情后跟2-5位数字（2位或3位或4位或5位数字）

var a = /速度与激情\d{2,5}/

console.log(a.test('速度与激情2')) //false

console.log(a.test('速度与激情22')) //true

console.log(a.test('速度与激情222')) //true

console.log(a.test('速度与激情2222')) //true

console.log(a.test('速度与激情22222')) //true

console.log(a.test('速度与激情222222')) //true

// 再比如匹配电话号码的位数（11位连续的数字）

var b = /\d{11}/

console.log(b.test('22222222222')) //true

console.log(b.test('22222')) //false

console.log(b.test('222222222222222222222')) //true

●\*的用法

// 匹配全部内容

var a = /.\*/

console.log(a.exec('dklasjdfklasjklf'))

// 输出：

// [ 'dklasjdfklasjklf',

// index: 0,

// input: 'dklasjdfklasjklf',

// groups: undefined ]

// 注，但是不能匹配换行

console.log(a.exec('akldfjk\nkljkljjl'))

// 输出：匹配了换行（\n）之前的

// [ 'akldfjk',

// index: 0,

// input: 'akldfjk\nkljkljjl',

// groups: undefined ]

●+的用法

// +匹配的是它前面的字符至少得出现一次

var a = /w+/

console.log(a.test('aaawaa')) //true

console.log(a.test('aaawwwwwaa')) //true

console.log(a.test('aaaaa')) //false

█匹配开头和结尾

|  |  |
| --- | --- |
| 字符 | 功能 |
| ^ | |  |  | | --- | --- | |  | 匹配字符串开头 | |
| $ | 匹配字符串结尾 |

// 判断变量名是否符合规格即（英文字母数字和下划线，开头不能是数字）

// 必须已英文字母或下划线开头，加^是限定被判断的字符串必须以其后面的字符为开头，^[a-zA-Z\_]

// 后面的内容可以是英文字母下划线和数字，使用\*表示可以没有也可以出现任意多次，使用$表示判断完必须正好是被匹配字符串的结尾，[a-zA-Z0-9\_]\*$

var a = /^[a-zA-Z\_][a-zA-Z0-9\_]\*$/

console.log(a.test('d')) //true

console.log(a.test('abcs999')) //true

console.log(a.test('ab\_kk98')) //true

console.log(a.test('99abc')) //false

console.log(a.test('99abc#')) //false

●小例子：匹配开头和结尾，还有转译

// 匹配一个163邮箱

// @前需要4到20位，{4，20}

// 这里的.与正则的符号冲突了，所以需要转译，\.

var a = /^[a-zA-Z\_0-9]{4,20}@163\.com$/

console.log(a.test('abcd@163.com'))

█分组

|  |  |
| --- | --- |
| 字符 | 功能 |
| | | 匹配左右任意一个表达式 |
| (ab) | 将括号中字符作为一个分组 |
| \num | 引用分组num匹配到的字符串 |
| (?P<name>) | 分组起别名 |
| (?P=name) | |  |  | | --- | --- | |  | 引用别名为name分组匹配到的字符串 | |

// 拿上面的邮箱的例子

// |代表或，括号里符合126或163或icloud都成立

// 括号还有取值的作用，a.exec('abcd@163.com')[1]

var a = /^([a-zA-Z\_0-9]{4,20})@(126|163|icloud)\.com$/

console.log(a.exec('abcd@163.com'))

// 输出：

// [ 'abcd@163.com',

// 'abcd',

// '163',

// index: 0,

// input: 'abcd@163.com',

// groups: undefined ]

console.log(a.exec('abcd@163.com')[1]) //abcd

●分组的命名和引用

// 测试一个HTML元素标签是否闭合

// 注意第二个标签里的/要用转译为\/，因为和JS表示正则的/冲突了

// 注意第二个标签里的\1代表对分组()里的内容的引用。如果有多个分组，就是用\1 \2 \3等来引用

var a = /^<(\w)>.\*<\/\1>$/

console.log(a.test('<p>abc</p>')) //true

console.log(a.test('<p>abc</a>')) //false

// 还可以给分组起名字

// 命名分组：(?P<分组的名称>分组的内容)

// 引用分组：(?P=分组的名称)

var b = /^<(?P<name1>\w)>.\*<\/(?P=name1)>$/

console.log(b.test('<p>abc</p>')) //true

console.log(b.test('<p>abc</a>')) //false