4-26

正则表达式(对象)

> 定义：专门用来你**检索模糊范围**字符串的一种规则

- 火星文，难点在于找规律，自己写完之后都看不懂

语法：

- 简写：`/ /`
- 实例化：`new RegExp(有两个参数)`第一个参数是字符串，第二个参数是修饰符
- **修饰符**
- **g** - global （全局）的缩写
- **i** - ignore (忽略大小写)的缩写 比如：查找数据中忽略大小写`str.match(/x/ig)`在str中查找x；不区分大小写

`str.match(/\d/g)`找到str中所有的数字放在数组中

转义符 \ 加在需要转移的字符前边，可以把一些带有特殊含义的字符转义成正常的字符串输出

- 主要注意的是在写‘\’转义
- 当 \ +字母的时候会有特殊含义，这种带有特殊含义的字母叫做**元字符**
- \d 代表**一个**数字
- \D 代表一个非数字
  正则的方法：
  exec()语法：正则.exec（字符串）
  找到正则中第一个匹配的字符且放到数组中。
  返回的值是[字符、index、input、groups]
  test()语法：
  语法：/规则/.test(字符串)
  检测正则是否匹配字符串，成立就返回true，不成立就返回false；
  正则中的或者是`|`

match方法

match方法是字符串的方法；
语法：str.match(/正则/)
找到正则中所有匹配的字符且放到数组中；匹配不到返回`null`

`str.match(/\d/g)` 找到str中所有的数字放在数组中
量词 +号 最少一个最多不限 写在需要找的数据后
`str.match(/\d+/g)` 全局查找整个字符串，把一个数字或者连续多个数字存放在数组中；

replace (常用)替换

replace是字符串的方法：
语法：str.replace(两个参数)第一个是要替换的旧字符串或者是**正则**，第二个参数是替换成什么新的字符串或者是**函数**；
当第二个参数为函数的时候，函数的第一个参数就是每次匹配到的结果 `function（$0）{}` 第一个参数每匹配到一个结果就执行一次函数($0的长度就是前边匹配到的结果)；此函数必须要有return；不然就返回undefined；
默认：
函数的第一个参数就是每次匹配到的结果
函数的第二个参数就是index索引
函数的第三个参数就是input 全部字符串
函数的第四个及之后的参数就是undefined；

子项 （）小括号

也有提权的作用。
子项是从左往右数的，每有一个（）就是一个子项；
作用，放在正则中；能够在一个规则中提取某些（指定）字符
在replace中，如果第一个参数中有子项的时候，那么第二个函数的参数将改变（1，每次匹配的结果，2，第一个子项的值，3，第二个子项的值。4，第三个子项的值）
在最后一个子项之后的参数，就还是默认的index、input、undefined；
注意：子项如果包了一个规则，在规则之后跟着一个量词，那么结果是最后一个
`let str = '2019' (\d)+` 结果是9，而不是默认的2；

- 子项重复项：`\+数字`
- 比如 `/(b1)c1(d1)\1\2/` 结果是 `'b1c1d1b1d1'`

正则中的 [ ]

在[ ]中任意选择一个字符
比如：[12345]可以为1或者2或者3或者4或者5
也可以简写为[1-5] 1-5的意思（顺序是从unicode编码来的）
数字0-9，小写字母a-z;大写A-Z；
中文区间范围[\u4e00-\u9fa5]

正则中的^开头$结尾

整段字符串是否都满足正则的规则，而不是字符串中有一小段匹配正则的规则就返回；
比如

```
str.match(/^1[89]&/) 例如这个str只能是18或者19才能满足，188或者199前两位虽然满足，但加上开头^ 和$ 结尾后就不满足了；
```