正则表达式

正则表达式简介

正则表达式（Regular expression，RegExp）的作用是对字符串执行**模式匹配**（pattern）。我们通常使用正则表达式来解决下列问题：

1. 规则验证（例如：验证手机号、邮箱等）
2. 提取子串（从某个字符串中提取出符合某个条件的子串）
3. 检查一个字符串是否含有某种子串
4. 将匹配到的子串进行替换（例如：将敏感词汇替换为*）

各种编程语言的正则表达式实现基本相同，不同的语言的实现可能会有一些细小的差别，本文着重讲解 JavaScript 中的正则表达式。

JavaScript 示例：

```
const content = '上海头等仓网'
const pattern = new RegExp('头等仓')
pattern.test(content)
```

Java 示例：

```java
import java.util.regex.*;

class RegexExample {
   public static void main(String args[]) {
      String content = "上海头等仓网";
      String pattern = ".*头等仓.*";
      boolean isMatch = Pattern.matches(pattern, content);

      System.out.println("字符串中是否包含了“头等仓”子串? " + isMatch);
   }
}
```

JavaScript RegExp 对象

在 JavaScript 中，RegExp 内置对象是对正则表达式的实现，它是对字符串执行模式匹配的强大工具。

创建 RegExp 对象的两种方式

1、构造函数语法

```
new RegExp(pattern, attributes)
```

2、直接量语法（**推荐**）

```
/pattern/attributes
```

参数：

- 参数 pattern 是一个字符串，指定了正则表达式的模式
- 参数 attributes 是一个可选的字符串，表示修饰符

返回值：

- 一个新的 RegExp 对象，具有指定的模式和修饰符。

RegExp 对象方法

- `test(string)`：检索字符串中指定的子串。返回 true 或 false。
- `exec(string)`：检索字符串中指定的子串。返回找到的子串，并返回其索引。
- `compile(regexp, modifier)`：用于改变和重新编译正则表达式。

String 对象对正则表达式的支持

- `search(regexp)`：检索字符串中指定的子串，返回第一个匹配到的子串的起始位置索引。
- `match(substr/regexp)`：检索字符串中指定的子串，返回存放匹配结果的数组。
- `replace(regexp/substr, replacement)`：将匹配到的子串进行替换。
- `split(separator,howmany)`：把字符串分割为字符串数组。

正则表达式语法

修饰符

修饰符用于指定正则表达式的匹配模式

修饰符	描述
g	执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。（global）
i	执行对大小写不敏感的匹配。（ignoreCase）
m	执行多行匹配。（multiline）

元字符

元字符是拥有特殊含义的字符

元字符	描述
\	转义符
|	选择匹配符，可以匹配多个规则
.	查找单个字符，除了换行和行结束符。
\w	查找单词字符。（word）
\W	查找非单词字符。
\d	查找数字。（digit）
\D	查找非数字字符。
\s	查找空白字符。（space）
\S	查找非空白字符。
\b	匹配单词边界。（boundary）
\B	匹配非单词边界。
\0	查找 NUL 字符。
\n	查找换行符。（newline）
\f	查找换页符。
\r	查找回车符。
\t	查找制表符。
\v	查找垂直制表符。
\xxx	查找以八进制数 xxx 规定的字符。
\xdd	查找以十六进制数 dd 规定的字符。
\uxxxx	查找以十六进制数 xxxx 规定的 Unicode 字符。

量词

量词	描述
n+	匹配任何包含至少一个 n 的字符串。
n*	匹配任何包含零个或多个 n 的字符串。
n?	匹配任何包含零个或一个 n 的字符串。
n{X}	匹配包含 X 个 n 的序列的字符串。
n{X,Y}	匹配包含 X 至 Y 个 n 的序列的字符串。
n{X,}	匹配包含至少 X 个 n 的序列的字符串。
n$	匹配任何结尾为 n 的字符串。
^n	匹配任何开头为 n 的字符串。
?=n	匹配任何其后紧接指定字符串 n 的字符串。
?!n	匹配任何其后没有紧接指定字符串 n 的字符串。

方括号（字符簇）

方括号用于查找某个范围内的字符

表达式	描述
[abc]	查找方括号之间的任何字符。
[^abc]	查找任何不在方括号之间的字符。
[0-9]	查找任何从 0 至 9 的数字。
[a-z]	查找任何从小写 a 到小写 z 的字符。
[A-Z]	查找任何从大写 A 到大写 Z 的字符。
[A-z]	查找任何从大写 A 到小写 z 的字符。

括号（子表达式）

在正则表达式中，使用括号括起来的内容是一个子表达式。

子表达式匹配到的内容会被系统捕获至缓冲区，使用`\n`（n表示数字）来反向引用（Back reference）系统的第 n 号缓冲区的内容。

需求例题

1. 校验手机号
2. 校验身份证号
3. 校验QQ号
4. 校验邮箱
5. 校验日期
6. 对手机号、身份证号进行隐私保护
7. 把句子分割成单词，例句 `How are you doing today`
8. 查找连续相同的四个数字 `1212ab45677778cd`

工具函数

将常用的正则表达式进行封装

`utils/regexp.js`:

```
/*
 * 正则表达式相关操作
 * @Author: xiaoming.bai
 * @Date: 2019-12-03 10:19:41
 * @Last Modified by: xiaoming.bai
 * @Last Modified time: 2019-12-04 15:55:05
 */

/**
 * 规则校验
 * @param  {String} str  待校验字符串
 * @param  {String} type 检测类型
 */
export const regExpTest = (str, type) => {
  const typeMap = {
    // 手机号
    mobile: /^(0|86|17951)?1[3-9]\d{9}$/,

    // QQ号
    qq: /[1-9][0-9]{4,9}/,

    // 邮箱
    email: /^[a-zA-Z0-9\._-]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+$/,

    // 微信环境
    wechat: /MicroMessenger/i,

    // 百度爬虫
    baiduSpider: /Baiduspider/i,
  }
  return typeMap[type].test(str)
}

/**
 * 手机号隐私处理
 * @param {String} phoneStr
 */
export const secretPhone = phoneStr => {
  const regexp = /^(\d{3})(\d{4})(\d{4})$/g
  const replacement = '$1****$3'
  return phoneStr.replace(regexp, replacement)
}

/**
 * 身份证号隐私处理
 * @param {String} id 身份证号
 */
export const secretIDCard = id => {
  const regexp = /^(\d{6})(\d{8})(\d{3}[0-9xX])$/g
  const replacement = '$1********$3'
  return phoneStr.replace(regexp, replacement)
```

```
}
```