什么是正则表达式

- 正则表达式（Regular Expression）是用于描述字符排列和匹配模式的一种语法规则
- 在javascript中正则表达式是一个对象
- 主要用于字符串的查找、匹配、替换、分割操作

```
// 要求：查找所有apple , 不区分大小写
var str = "apple pear watermelon Apple banana";
var regexp = /apple/gi ;
console.log( str.match(regexp) );

// 结果：["apple", "Apple"]
```

正则表达式的格式

- /表达式/[模式修正符]

  比如：/apple\d{3}/i

  定界符：两个斜线"/"。
  原子：普通字符a、 p、 l、 e
  元字符：\d 、 {3}等具有特殊含义的字符的组合
  模式修正符：结束定界符之后的字符i，对正则表达式进行修正

正则表达式的构成

定界符

- 正则表达式放在起始定界符和结束定界符之间
- javascript中定界符使用斜线"/"

原子

- 原子是正则表达式的最基本的组成单元
- 原子是由打印和非打印字符组成：
  - 普通字符作为原子： 如 a~z、A~Z、0~9 、_等
  - 一些非打印字符作为原子： 如：\n \r \t 等
  - 一些特殊字符和转义后元字符作为原子
  - 所有标点符号，但语句特殊意义的符号需要转义后才可作为原子，如： * \+ \? \. 等

元字符

- 元字符为具有特殊含义的字符
- 主要用途：
  - 定义原子的筛选方式
  - 定义原子的集合
  - 定义原子数量限定
  - 定义边界控制
  - 定义子模式单元

元字符之原子的筛选方式

- | 元字符“|”又称模式选择符,匹配两个或更多的选择之一
  例如：/apple|pear|banana/表示匹配apple或者pear
- [] 匹配方括号中的任意一个原子
  例如：[fhy]our 匹配 four、hour或your
- [^] 匹配除方括号中原子之外的任意一个字符
  例如：/[^高]三/ 匹配张三、李三、王三都可以，但不能匹配高三
- [-] 用于连接一组按ASCII码顺序排列的原子，简化书写。
  例如：/[0123456789]/可以写成/[0-9]/
  /[a-zA-Z]/可以匹配所有大小写字母

元字符之原子的集合

- . 匹配换行符之外的任意字符 [^\n\r]
- \s 匹配一个空白字符，相当于[\f\n\r\t\v]
- \S 匹配一个非空白字符，相当于[^\f\n\r\t\v]
- \d 匹配0-9的数字，相当于[0-9]
- \D 匹配除0-9之外的任意字符，相当于[^0-9]
- \w 匹配任意一个数字、大小写字母和下划线字符，相当于[0-9a-zA-Z_]
- \W 匹配任意一个非数字、大小写字母和下划线字符，相当于[^0-9a-zA-Z_]

元字符之原子数量限定

- {n} 其前面的原子连续出现n次
- {n,} 其前面的原子至少连续出现n次
- {n,m} 其前面的原子连续出现n到m次
- * 其前面的原子至少出现0次，相当于{0,}
- + 其前面的原子至少出现1次，相当于{1,}
- ? 其前面的原子出现0次或1次，相当于{0,1}

> 贪婪匹配：每次匹配最大个数
> 取消贪婪匹配：数量控制符后加上?

```
var numstr = "123445566abc"
 console.log(numstr.match(/\d+/))
 console.log(numstr.match(/\d+?/))
```

元字符之边界控制

- ^ 写在正则表达式最前面表示必须以什么开始
- $ 写在正则表达式最后表示必须以什么结束

元字符之子模式单元

- ()
  可以将小原子变成大原子
  小括号中的部分为一个子模式
  子模式可以被反向引用 $1 $2 .....

```
var str="You veryvery good,very beautiful!";
   var regex1=/(very)+/g;
   console.log(str.match(regex1))
```

```
var username = "Doe , John";
console.log( username.replace(/(\w+)\s*, \s*(\w+)/, "$2 $1") );
```

模式修正符

- i 执行对大小写不敏感的匹配
- 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）
- m 执行多行匹配，即在到达一行文本末尾时还会继续查找下一行中是否存在与模式匹配的项，**只有当目标字符串含有\n，而且正则表达式中含有^或$的时候，/m修饰符才有作用**

```
var str = `apple pear 123
Apple melon 456
apple789`;

console.log( str.match(/^apple[\w]*\d$/gmi) )
```

如果 设置m，那么“^”与字符串的开始位置相匹配，而“$”与字符串的结束位置相匹配。 如果不设置m，那么“^”与字符串开始位置以及“\n”或“\r”之后的位置相匹配，而“$”与字符串结束位置以及“\n”或“\r”之前的位置相匹配。

```
var mutiline = /^abc/m;
var singleline = /^abc/;
var target = "ef\r\nabcd";
alert(mutiline.test(target));//true
alert(singleline.test(target));//false
```

\r\n在windows下代表换行，如果只有1个\n也是一样的效果。由于target不是以abc开头的字符串，所以匹配singleline的结果是false；由于target是多行字符串(含有\n)，而第2行是以abc开头，所以匹配multiline结果是true。

练习

- 邮箱
- 手机号
- URL地址
- 中文
  /[\u4e00-\u9fa5]/
- 身份证号码
- IP地址
- QQ
- 以字母开始包含字母、数字、下划线的用户名

JS中正则表达式的应用

正则表达式.test(字符串)

检索字符串是否匹配。返回 true 或 false

```
var reg = /^1[3-8]\d{9}$/;
 console.log(reg.test(13837160809));
```

字符串.match(正则表达式)

查找字符串以找到一个或多个与 regexp 匹配的文本。 regexp 具有标志 g则全局匹配，否则只匹配一次

```
var str = "1 plus 2 equal 3";
console.log(str.match(/\d+/g));
```

字符串.replace(正则表达式,新的字符串)

将正则表达式匹配到的子字符串替换为新的字符串

```
var str = 'apple Apple pear APPLE';
 console.log( str.replace(/apple/i, '苹果'));
```

字符串.split(正则表达式)

用正则表达式匹配到的字符作为分割符将字符串分割成数组

```
var reg = /^1[3-8]\d{9}$/;
var str="13837160809";
 console.log(str.test(reg));
```

练习

- 封装一个函数用来去除字符串两边空白字符
- 表单验证功能
    - 要求:
      用户名由字母数字下划线组成且长度在5-15个字符之间
      用户名也可以使用手机号或邮箱
      密码要求由字母、数字、特殊字符中至少两种组合而成
      密码长度要求在8-20个字符之间
        - /(?!^([a-zA-Z]+|\d+|[~!@#$%^&*?]+)$)^[\w~!@#$%^&*?]{8,20}$/
        - (?!正则表达式1)正则表达式2
            - 表示匹配不满足表达式1但满足正则表达2的字符串