正则表达式（Regex Expression）

@M了个J
李明杰

https://github.com/CoderMJLee
http://cnblogs.com/mjios

码拉松

字符串的合法验证

- 在开发中，经常会对一些字符串进行合法验证


```java
// 6~18个字符，可使用字母、数字、下划线，需以字母开头
public static boolean validate(String email) {
    // 验证逻辑...
}
```

小码哥教育 SEEMYGO

自己编写验证逻辑

```
if (email == null) {
    System.out.println("不能为空");
    return false;
}
```

```
char[] chars = email.toCharArray();
if (chars.length < 6 || chars.length > 18) {
    System.out.println("必须是6~18个字符");
    return false;
}
```

```
if (!isLetter(chars[0])) {
    System.out.println("必须以字母开头");
    return false;
}
```

```
for (int i = 1; i < chars.length; i++) {
    char c = chars[i];
    if (isLetter(c) || isDigit(c) || c == '_') continue;
    System.out.println("必须由字母、数字、下划线组成");
    return false;
}
return true;
```

小码哥教育 SEEMYGO

自己编写验证逻辑

```
private static boolean isLetter(char c) {
    return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
}

private static boolean isDigit(char c) {
    return c >= '0' && c <= '9';
}
```

```
// 必须是6~18个字符 false
System.out.println(validate("12345"));
// 必须以字母开头 false
System.out.println(validate("123456"));
// true
System.out.println(validate("vv123_456"));
// 必须由字母、数字、下划线组成
System.out.println(validate("vv123+/?456"));
```

小码哥教育 SEEMYGO

使用正则表达式

```
String regex = "[a-zA-Z]\\w{5,17}";
// false
12345".matches(regex)
// false
"123456".matches(regex)
// true
"vv123_456".matches(regex)
// false
"vv123+/?456".matches(regex)
```

- [a-zA-Z]\\w{5,17} 是一个正则表达式
  - 用非常精简的语法取代了复杂的验证逻辑
  - 极大地提高了开发效率

- 正则表达式的英文
  - Regex Expression

- 正则表达式是一种通用的技术，适用于绝大多数流行编程语言

```
// JavaScript中的正则表达式
const regex = /[a-zA-Z]\w{5,17}/;
regex.test('12345') // false
regex.test('123456') // false
regex.test('vv123_456') // false
regex.test('vv123+/?456') // false
```

小码哥教育 SEEMYGO

单字符匹配

语法	含义
[abc]	a、b、c
[^abc]	除了 a、b、c 以外的任意字符
[a-zA-Z]	从 a 到 z、从 A 到 Z
[a-d[m-p]]	[a-dm-p]（并集）
[a-z&&[def]]	d、e、f（交集）
[a-z&&[^bc]]	[ad-z]（差集，从 [a-z] 中减去 [bc]）
[a-z&&[^m-p]]	[a-lq-z]（差集，从 [a-z] 中减去 [m-p]）

小码哥教育 SEEMYGO

单字符匹配 – 示例

```
// 等价于[b|c|r]at、(b|c|r)at
String regex = "[bcr]at";
"bat".matches(regex) // true
"cat".matches(regex) // true
"rat".matches(regex) // true
"hat".matches(regex) // false
```

```
String regex = "[^bcr]at";
"bat".matches(regex) // false
"cat".matches(regex) // false
"rat".matches(regex) // false
"hat".matches(regex) // true
```

```
String regex = "foo[1-5]";
"foo3".matches(regex) // true
"foo6".matches(regex) // false
```

```
String regex = "foo[^1-5]";
"foo3".matches(regex) // false
"foo6".matches(regex) // true
```

```
String regex = "foo1-5";
"foo1-5".matches(regex) // true
"foo1".matches(regex) // false
"foo5".matches(regex) // false
```

小码哥教育 SEEMYGO

单字符匹配 – 示例

```
String regex = "[0-4[6-8]]";
"5".matches(regex) // false
"7".matches(regex) // true
"9".matches(regex) // false
```

```
String regex = "[0-9&&[345]]";
"2".matches(regex) // false
"3".matches(regex) // true
"4".matches(regex) // true
"5".matches(regex) // true
"6".matches(regex) // false
```

```
String regex = "[0-9&&[^345]]";
"2".matches(regex) // true
"3".matches(regex) // false
"4".matches(regex) // false
"5".matches(regex) // false
"6".matches(regex) // true
```

小码哥教育 SEEMYGO

预定义字符

语法	含义
.	任意字符
\d	[0-9]（数字）
\D	[^0-9]（非数字）
\s	[\t\n\f\r]（空白）
\S	[^\s]（非空白）
\w	[a-zA-Z_0-9]（单词）
\W	[^\w]（非单词）

- 以 1 个反斜杠（\）开头的字符会被当做转义字符处理
- 因此，为了在正则表达式中完整地表示预定义字符，需要以 2 个反斜杠开头，比如 "\\d"

小码哥教育 SEEMYGO

预定义字符 – 示例

```
String regex = ".";
"@".matches(regex) // true
"c".matches(regex) // true
"6".matches(regex) // true
".".matches(regex) // true
```

```
String regex = "\\.";
"@".matches(regex) // false
"c".matches(regex) // false
"6".matches(regex) // false
".".matches(regex) // true
```

```
String regex = "\\[123\\]";
"1".matches(regex) // false
"2".matches(regex) // false
"3".matches(regex) // false
"[123]".matches(regex) // true
```

```
String regex = "\\d";
"c".matches(regex) // false
"6".matches(regex) // true
```

```
String regex = "\\D";
"c".matches(regex) // true
"6".matches(regex) // false
```

小码哥教育 SEEMYGO

预定义字符 – 示例

```
String regex = "\\s";
"\t".matches(regex) // true
"\n".matches(regex) // true
"\f".matches(regex) // true
"\r".matches(regex) // true
" ".matches(regex) // true
"6".matches(regex) // false
```

```
String regex = "\\S";
"\t".matches(regex) // false
"\n".matches(regex) // false
"\f".matches(regex) // false
"\r".matches(regex) // false
" ".matches(regex) // false
"6".matches(regex) // true
```

```
String regex = "\\w";
"_".matches(regex) // true
"c".matches(regex) // true
"6".matches(regex) // true
"+".matches(regex) // false
```

```
String regex = "\\W";
"_".matches(regex) // false
"c".matches(regex) // false
"6".matches(regex) // false
"+".matches(regex) // true
```

小码哥教育 SEEMYGO

量词（Quantifier）

贪婪（Greedy）	勉强（Reluctant）	独占（Possessive）	含义
X{n}	X{n}?	X{n}+	X 出现 n 次
X{n,m}	X{n,m}?	X{n,m}+	X 出现 n 到 m 次
X{n,}	X{n,}?	X{n,}+	X 出现至少 n 次
X?	X??	X?+	X{0,1}（X 出现 0 次或者 1 次）
X*	X*?	X*+	X{0,}（X 出现任意次）
X+	X+?	X++	X{1,}（X 至少出现 1 次）

小码哥教育 SEEMYGO

量词 - 示例

```
String regex = "6{3}";
"66".matches(regex) // false
"666".matches(regex) // true
"6666".matches(regex) // false
```

```
String regex = "6{2,4}";
"6".matches(regex) // false
"66".matches(regex) // true
"666".matches(regex) // true
"6666".matches(regex) // true
"66666".matches(regex) // false
```

```
String regex = "6{2,}";
"6".matches(regex) // false
"66".matches(regex) // true
"666".matches(regex) // true
"6666".matches(regex) // true
"66666".matches(regex) // true
```

```
String regex = "6?";
"".matches(regex) // true
"6".matches(regex) // true
"66".matches(regex) // false
```

```
String regex = "6*";
"".matches(regex) // true
"6".matches(regex) // true
"66".matches(regex) // true
```

```
String regex = "6+";
"".matches(regex) // false
"6".matches(regex) // true
"66".matches(regex) // true
```

小码哥教育 SEEMYGO

Pattern、Matcher

- String 的 matches 方法底层用到了 Pattern、Matcher 两个类

```
// java.lang.String
public boolean matches(String regex) {
    return Pattern.matches(regex, this);
}
```

```
// java.util.regex.Pattern
public static boolean matches(String regex, CharSequence input) {
    Pattern p = Pattern.compile(regex);
    Matcher m = p.matcher(input);
    return m.matches();
}
```

小码哥教育 SEEMYGO

Matcher 常用方法

```
// 如果整个 input 与 regex 匹配，就返回 true
public boolean matches();
```

```
// 如果从 input 中找到了与 regex 匹配的子序列，就返回 true
// 如果匹配成功，可以通过 start、end、group 方法获取更多信息
// 每次的查找范围会先剔除此前已经查找过的范围
public boolean find();
```

```
// 返回上一次匹配成功的开始索引
public int start();
```

```
// 返回上一次匹配成功的结束索引
public int end();
```

```
// 返回上一次匹配成功的 input 子序列
public String group();
```

小码哥教育 SEEMYGO

找出所有匹配的子序列

```
public static void findAll(String regex, String input) {
    findAll(regex, input, 0);
}

public static void findAll(String regex, String input, int flags) {
    if (regex == null || input == null) return;
    Pattern p = Pattern.compile(regex, flags);
    Matcher m = p.matcher(input);
    boolean found = false;
    while (m.find()) {
        found = true;
        System.out.format("\"%s\", [%d, %d)%n", m.group(), m.start(), m.end());
    }
    if(!found){
        System.out.println("No match.");
    }
}
```

小码哥教育 SEEMYGO

Matcher – 示例

```
String regex = "123";
findAll(regex, "123");
// "123", [0, 3)

findAll(regex, "6_123_123_123_7");
// "123", [2, 5)
// "123", [6, 9)
// "123", [10, 13)
```

```
String regex = "[abc]{3}";
findAll(regex, "abccabaaaccbbbc");
// "abc", [0, 3)
// "cab", [3, 6)
// "aaa", [6, 9)
// "ccb", [9, 12)
// "bbc", [12, 15)
```

```
String regex = "\\d{2}";
findAll(regex, "0_12_345_67_8");
// "12", [2, 4)
// "34", [5, 7)
// "67", [9, 11)
```

小码哥教育 SEEMYGO

Matcher – 示例

```
String input = "";
findAll("a?", input);
// "", [0, 0)

findAll("a*", input);
// "", [0, 0)

findAll("a+", input);
// No match.
```

```
String input = "a";
findAll("a?", input);
// "a", [0, 1)
// "", [1, 1)

findAll("a*", input);
// "a", [0, 1)
// "", [1, 1)

findAll("a+", input);
// "a", [0, 1)
```

```
String input = "abbaaa";
findAll("a?", input);
// "a", [0, 1)
// "", [1, 1)
// "", [2, 2)
// "a", [3, 4)
// "a", [4, 5)
// "a", [5, 6)
// "", [6, 6)

findAll("a*", input);
// "a", [0, 1)
// "", [1, 1)
// "", [2, 2)
// "aaa", [3, 6)
// "", [6, 6)

findAll("a+", input);
// "a", [0, 1)
// "aaa", [3, 6)
```

小码哥教育 SEEMYGO

Matcher – 贪婪、勉强、独占的区别

```
String input = "afooaaaaaafooa";
findAll(".*foo", input); // 贪婪
// "afooaaaaaafoo", [0, 13)

findAll(".*?foo", input); // 勉强
// "afoo", [0, 4)
// "aaaaaafoo", [4, 13)

findAll(".*+foo", input); // 独占
// No match.
```

- 贪婪
  - 先吞掉整个 input 进行匹配
  - ✓ 若匹配失败，则吐出最后一个字符
  - 然后再次尝试匹配，重复此过程，直到匹配成功

- 勉强
  - 先吞掉 input 的第一个字符进行匹配
  - ✓ 若匹配失败，则再吞掉下一个字符
  - 然后再次尝试匹配，重复此过程，直到匹配成功

- 独占
  - 吞掉整个 input 进行唯一的一次匹配

小码哥教育 SEEMYGO

捕获组（Capturing Group）

```
String regex1 = "dog{3}";
"doggg".matches(regex1) // true

String regex2 = "[dog]{3}";
"ddd".matches(regex2) // true
"ooo".matches(regex2) // true
"ggg".matches(regex2) // true
"dog".matches(regex2) // true
"gog".matches(regex2) // true
"gdo".matches(regex2) // true

String regex3 = "(dog){3}";
"dogdogdog".matches(regex3) // true
```

捕获组 – 反向引用（Backreference）

- 反向引用（Backreference）
  - 可以使用反斜杠（\）+ 组编号（从 1 开始）来引用组的内容

```
String regex = "(\\d\\d)\\1";
"1212".matches(regex) // true
"1234".matches(regex) // false
```

```
String regex = "([a-z]{2})([A-Z]{2})\\2\\1";
"mjPKPKmj".matches(regex) // true
"mjPKmjPK".matches(regex) // false
```

- ((A)(B(C))) 一共有 4 个组
  - 编号1：((A)(B(C)))
  - 编号2：(A)
  - 编号3：(B(C))
  - 编号4：(C)

```
String regex = "((I)( Love( You)))\\3{2}";
"I Love You Love You Love You".matches(regex) // true
```

小码哥教育 SEEMYGO

边界匹配符（Boundary Matcher）

语法	含义
\b	单词边界
\B	非单词边界
^	一行的开头
$	一行的结尾
\A	输入的开头
\z	输入的结尾
\Z	输入的结尾（结尾可以有终止符）
\G	上一次匹配的结尾

小码哥教育 SEEMYGO

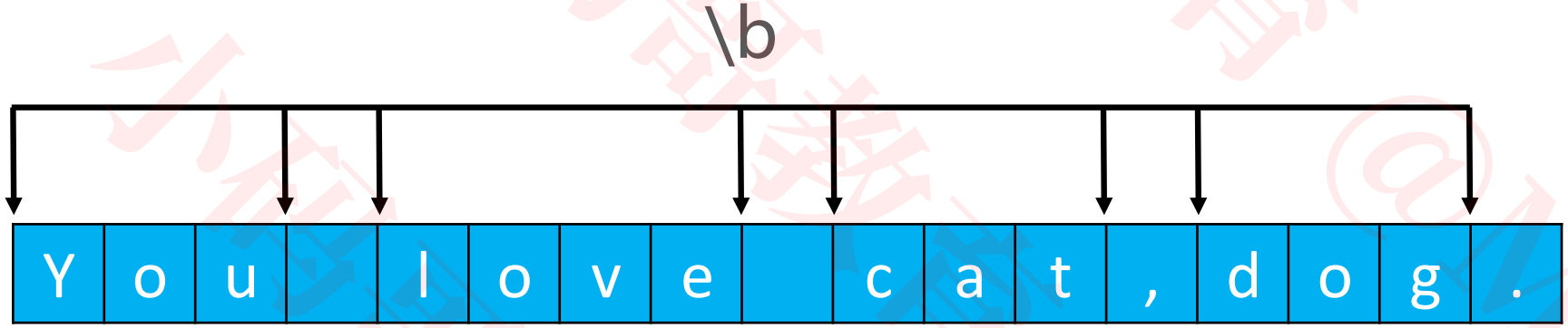
一些概念

- 终止符（Final Terminator、Line Terminator）
  - \r（回车符）、\n（换行符）、\r\n（回车换行符）

- 输入：整个字符串

- 一行：以终止符（或整个输入的结尾）结束的字符串片段
  - 如果输入是 `"dog\ndog\rdog"`
  - 那么 3 个 `dog` 都是一行

边界匹配符 – 单词边界




```
String regex = "\\bdog\\b";
findAll(regex, "This is a dog.");
// "dog", [10, 13)

findAll(regex, "This is a doggie.");
// No match.

findAll(regex, "dog is cute");
// "dog", [0, 3)

findAll(regex, "I love cat,dog,pig.");
// "dog", [11, 14)
```

```
String regex = "\\bdog\\B";
findAll(regex, "This is a dog.");
// No match.

findAll(regex, "This is a doggie.");
// "dog", [10, 13)

findAll(regex, "dog is cute");
// No match.

findAll(regex, "I love cat,dog,pig.");
// No match.
```

边界匹配符 - 示例

```
String regex = "^dog$";
findAll(regex, "dog");
// "dog", [0, 3)

findAll(regex, "    dog");
// No match.
```

```
findAll("\\s*dog$", "    dog");
// "    dog", [0, 9)

findAll("^dog\\w*", "dogblahblah");
// "dogblahblah", [0, 11)
```

```
String regex = "\\Gdog";
findAll(regex, "dog");
// "dog", [0, 3)

findAll(regex, "dog dog");
// "dog", [0, 3)

findAll(regex, "dogdog");
// "dog", [0, 3)
// "dog", [3, 6)
```

小码哥教育 SEEMYGO

常用模式

模式	含义	等价的正则写法
DOTALL	单行模式（.可以匹配任意字符，包括终止符）	(?s)
MULTILINE	多行模式（^、$ 才能真正匹配一行的开头和结尾）	(?m)
CASE_INSENSITIVE	不区分大小写	(?i)

小码哥教育 SEEMYGO

常用模式 – CASE_INSENSITIVE

```
String regex = "dog";
String input = "Dog_dog_DOG";
findAll(regex, input);
// "dog", [4, 7)

findAll(regex, input, Pattern.CASE_INSENSITIVE);
// "Dog", [0, 3)
// "dog", [4, 7)
// "DOG", [8, 11)

findAll("(?i)dog", input);
// "Dog", [0, 3)
// "dog", [4, 7)
// "DOG", [8, 11)
```

小码哥教育 SEEMYGO

常用模式 – DOTALL、MULTILINE

```
String regex = ".";
String input = "\r\n";
findAll(regex, input);
// No match.

findAll(regex, input, Pattern.DOTALL);
// "\r", [0, 1)
// "\n", [1, 2)

findAll(regex, input, Pattern.MULTILINE);
// No match.

findAll(regex, input, Pattern.MULTILINE | Pattern.DOTALL);
// "\r", [0, 1)
// "\n", [1, 2)

findAll("(?sm).", input);
// "\r", [0, 1)
// "\n", [1, 2)
```

小码哥教育 SEEMYGO

常用模式 – DOTALL、MULTILINE

```
String regex = "^dog$";
String input = "dog\ndog\rdog";

findAll(regex, input);
// No match.

findAll(regex, input, Pattern.DOTALL);
// No match.

findAll(regex, input, Pattern.MULTILINE);
// "dog", [0, 3)
// "dog", [4, 7)
// "dog", [8, 11)

findAll(regex, input, Pattern.DOTALL | Pattern.MULTILINE);
// "dog", [0, 3)
// "dog", [4, 7)
// "dog", [8, 11)
```

小码哥教育 SEEMYGO

边界匹配符 - \A、\z

```
String regex = "\\Adog\\z";

findAll(regex, "dog");
// "dog", [0, 3)

findAll(regex, "dog\n");
// No match.

findAll(regex, "dog\ndog\rdog");
// No match.

findAll(regex, "dog\ndog\rdog", Pattern.MULTILINE);
// No match.
```

小码哥教育 SEEMYGO

边界匹配符 – \A、\Z

```
String regex = "\\Adog\\Z";

findAll(regex, "dog");
// "dog", [0, 3)

findAll(regex, "dog\n");
// "dog", [0, 3)

findAll(regex, "dog\ndog\rdog");
// No match.

findAll(regex, "dog\ndog\rdog", Pattern.MULTILINE);
// No match.
```

常用正则表达式

- 正则表达式在线测试
- https://c.runoob.com/front-end/854

需求	正则表达式
18 位身份证号码	\d{17}[\dXx]
中文字符	[\u4e00-\u9fa5]

小码哥教育 SEEMYGO

String 类与正则表达式

- String 类中接收正则表达式作为参数的常用方法有

```
public String replaceAll(String regex, String replacement)

public String replaceFirst(String regex, String replacement)

public String[] split(String regex)
```

小码哥教育 SEEMYGO

练习 – 替换字符串中的单词

- 将单词 row 换成单词 line

```
String s1 = "The row we are looking for is row 8.";
// The line we are looking for is line 8.
String s2 = s1.replace("row", "line");
// The line we are looking for is line 8.
String s3 = s1.replaceAll("\\brow\\b", "line");
```

```
String s1 = "Tomorrow I will wear in brown standing in row 10.";
// Tomorline I will wear in blinen standing in line 10.
String s2 = s1.replace("row", "line");
// Tomorrow I will wear in brown standing in line 10.
String s3 = s1.replaceAll("\\brow\\b", "line");
```

练习 – 替换字符串的数字

- 将所有连续的数字替换为 **

```
String s1 = "ab12c3d456efg7h89i1011jk12lmn";
// ab**c**d**efg**h**i**jk**lmn
String s2 = s1.replaceAll("\\d+", "**");
```

小码哥教育 SEEMYGO

练习 – 利用数字分隔字符串

```
String s1 = "ab12c3d456efg7h89i1011jk12lmn";
// [ab, c, d, efg, h, i, jk, lmn]
String[] strs = s1.split("\\d+");
```

小码哥教育 SEEMYGO

练习 – 提取重叠的字母、数字

```
String input = "aa11+bb23-mj33*dd44/5566%ff77";
String regex = "([a-z])\\1(\\d)\\2";
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(input);
while (m.find()) {
    // a d f
    System.out.println(m.group(1));
    // 1 4 7
    System.out.println(m.group(2));
}
```

```
String input = "aa12+bb34-mj56*dd78/9900";
String regex = "[a-z]{2}\\d(\\d)";
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(input);
while (m.find()) {
    // 2 4 6 8
    System.out.println(m.group(1));
}
```