正则表达式从入门到实战

> 在开发的过程中，字符串处理往往很频繁。比如我们经常会对用户输入做校验：手机号，身份证号，邮箱，密码，域名，IP 地址，URL 或者其他与字符串相关校验的业务场景。正则表达式就是一种强大而灵活的文本处理工具，正则可以很好的解决这类字符串校验问题。掌握正则表达式，就能大大提高开发过程的效率。

正则表达式（Regular Expression）在代码中常常简写为regex。正则表达式通常被用来检索、替换那些符合某个规则的文本，它是一种强大而灵活的文本处理工具。

本场Chat将从2个方面入手：

- 学习正则表达式的语法规则，并介绍开发过程中使用正则表达式的流程，提供一款正则工具。
- 通过实现 5 个小功能练习使用正则，然后解决 2 个实际开发中遇到的问题。

通过本场 Chat 的学习，从零开始轻松掌握正则表达式，并且具备解决实际项目问题的能力。

一款好用的正则工具

给大家推荐一个正则工具**“RegexBuddy”**，你可以点此下载，密码：c509

说明：

为了便于理解，文章所有示例的正则表达式用**“regex=正则”**表示，**“=”**号后面就是正则

学习正则表达式语法，主要就是学习元字符以及它们在正则表达式上下文中的行为。

元字符包括：普通字符、标准字符、特殊字符、限定字符（又叫量词）、定位字符（也叫边界字符）。下面分别介绍不同元字符的用法。

普通字符

字母[a-zA-Z]、数字[0-9]、下划线[-]、汉字，标点符号：

- 匹配字母a可以regex=a
- 匹配字母b可以regex=b
- 匹配字母a或者b可以regex=a|b，这个正则**引入一个特殊字符“|”**，专业名称为“或”，你也可以叫它“竖线”，它表示“或”的意思。
- 匹配字母a或者b或者c可以regex=a|b|c
- 匹配字母a或者b或者c或者d可以regex=a|b|c|d
- 如果匹配所有26个字母，这种写法明显很二了。

这里引入两个特殊字符方括号**“[ ]”**和中划线**“-”***“[ ]”**，专业名称为“字符集合”，你也可以叫它“方括号”。**“-”*，表示“范围”，你也可以叫它“到”，regex=[A-Z]匹配从A到Z26个字母中的任意一个。

那么匹配字母a或者b或者c或者d可以regex=[abcd]。匹配数字1到8的任意数字可以regex=[1-8]，这样就不会匹配到0与9这2个数字了，如下：


```
0123456789
```

标准字符集合

标准字符集合是能够与“多种普通字符”匹配的简单表达式，比如：\d、\w、\s。

匹配数字0到9的任意数字可以regex=[0-9]也可以regex=**\d。标准字符集要注意区分大小写，**大写是相反的意思。regex=**\D，**则匹配非数字字符，即不能匹配数字0到9，如下：

标准字符	含义
\d	匹配0-9中的任意一个数字，等效于[0-9]
\D	匹配非数字字符，等效于[^0-9]
\w	匹配任意一个字母、数字或下划线，等效于[^A-Za-z0-9_]
\W	与任何非字母、数字或下划线字符匹配，等效于[^A-Za-z0-9_]
\s	匹配任何空白字符，包括空格、制表符
\S	匹配任何非空白字符
\n	匹配换行符
\r	匹配一个回车符
\t	匹配制表符

特殊字符

特殊字符在正则表达式中表示特殊的含义，比如：*****，**+**，**?** ，****，等等。

- “\”是转义字符，用于匹配特殊字符
- 匹配反斜杠“\”可以regex=\\，因为“\”是特殊字符，所以需要在它前边再加一个“\”进行转义
- 匹配星号“*”，可以regex=*，因为“*”是特殊字符，所以需要在它前边再加一个“\”进行转义

常用的特殊字符说明标黄的要熟记。

特殊字符	含义
\	转义字符，将下一个字符标记为一个特殊字符
^	匹配字符串开始的位置
$	匹配字符串结尾的位置
*	零次或多次匹配前面的字符或子表达式
+	一次或多次匹配前面的字符或子表达式
?	零次或一次匹配前面的字符或子表达式
.	“点” 匹配除“\r\n”之外的任何单个字符
|	或
[ ]	字符集合
( )	分组，要匹配圆括号字符，请使用 “\(” 或 “\)”

限定字符

```
123
1234
12345
123456
```

- 匹配任意8位数字可以regex=\d{8}
- 匹配任意8位以上的数字可以regex=\d{8,}
- 匹配任意1到8位以上的数字可以regex=\d{1,8}

```
123
12345abc
12345678abc
1234567890abc
```

从上图，我们可以看到regex=\d{1,8}，可以匹配到任意1-8个数字，超过8位数字后，从新开始匹配。

匹配次数中的“贪婪模式”与“非贪婪模式”：

正则的匹配默认是贪婪模式，即匹配的字符越多越好，而**非贪婪模式**是匹配的字符越少越好，在修饰匹配字数的量词后再加上一个问号**“?”**即可。

那么同样是上面的字符串，regex=\d{1,8}?匹配到什么呢？

```
123
12345abc
12345678abc
1234567890abc
```


因为在{1,8}这个量词后面加上了问号**“?”**，表示非贪婪模式，所以只能匹配到1个数字，即匹配的字符越少越好。

常用的限定字符说明 标黄的要熟记。

限定字符	含义
*	零次或多次匹配前面的字符或子表达式
+	一次或多次匹配前面的字符或子表达式
?	零次或一次匹配前面的字符或子表达式
{n}	n是一个非负整数，匹配确定的n次
{n,}	n是非负整数，至少匹配n次

定位字符也叫字符边界，标记匹配的不是字符而是符合某种条件的位置，所以定位字符是“零宽的”。

常用的定位字符：

定位字符	含义
^	匹配字符串开始的位置，表示开始
$	匹配字符串结尾的位置，表示结尾
\b	匹配一个单词边界

匹配以Hello开头的字符串可以regex=^Hello

```
Hello
Hello,regex.
regex,Hello.
```

匹配以Hello结尾的字符串可以regex=Hello$，如下：

```
Hello
Hello,regex.
regex,Hello.
```

匹配以H开头以o结尾的任意长度字符串可以regex=^H.*o$，如下：

```
Hello
Hello,regex.
regex,Hello.
Hi,wto
Hello,wto.
```


\b匹配这样一个位置：前面的字符和后面的字符不全是\w。如果在“hello,hello1 helloregex,hello regexhello.”这个字符串里匹配regex=hello\b，匹配到的结果如下：

```
hello,hello1 helloregex,hello regexhello.
```

分析一下：为什么**hello1**匹配不了**“hello\b”**这个正则？

首先\b是一个定位字符，它是零宽的，标识一个位置，这个位置的前面和这个位置的后面不能全是\w，即不能全是字母数字和下划线[A-Za-z0-9_]，而hello1的o与1之间的位置前面是o后面是1，前后全是\w，不符合\b匹配的含义，因此**hello1**不能匹配正则表达

但是，特殊字符（**除了小尖角“^”和中划线“-”外**）被包含到方括号中，就会失去特殊意义，只代表其字符本身。

regex=[abc+*?]匹配“a”,“b”,“c”任意一个字符或者**“+”,“*”,“?”**，即包含在自定义集合中的特殊字符**“+”,“*”,“?”**失去了特殊含义，只表示其字符本身的意思。

特殊字符**小尖角“^”**，原本含义是匹配字符串的开始位置，如果包含在自定义集合[ ]中，则表示**取反**的意思。比如：regex=[^aeiou]匹配“a”,“e”,“i”,“o”,“u”之外的任意一个字符。

中划线“-”，在自定义集合[ ]中，表示“范围”，而不是字符**“-”**本身，regex=[a-z]，匹配从a到z中26个字母中的任意一个。

除小数点**“.”**外，标准字符集合包含在方括号中，仍然表示集合范围。regex=**[\d.+]**匹配**0-9的任意一个数字或者小数点“.”或者加号“+”**

也就是说\d在自定义集合中仍然表示数字，但是小数点在字符集合中只表示小数点本身，而不是除“\r\n”之外的任何单个字符。

选择符和分组

表达式	作用
pattern1|pattern2	或的关系，匹配左边的pattern1或右边的pattern2
(pattern)	匹配pattern并获取这一匹配，并存储
(?:pattern)	匹配pattern但不获取匹配结果，也就是不进行存储

regex=**x|y**，匹配字符x或y。**( )**表示捕获组，**( )**的作用如下：

1. 括号中的表达式可以作为整体被修饰，用来表示匹配括号中表达式的次数，regex=(abc){2,3}，可以匹配连续的2个或3个abc，如下：

   abcabc abc abcabcabc abcabcabcabc

2. 括号中的表达式匹配到的内容会存储起来，并可以获取到括号中表达式匹配到的内容

3. 每一对括号会分配一个编号，使用( )的捕获根据左括号的顺序从1开始自动编号，

```
abcabc abc abcabcdabc abcabcabcdabc
```

(?:pattern)表示非捕获组，匹配括号中表达式匹配到的内容，但是不进行存储匹配到的内容。这在使用 “或” 字符?**(|)**?来组合一个正则的各个部分是很有用的。

例如：匹配字符“story”或者“stories”，regex=stor(?:y|ies)就是一个比 regex=story|stories更简略的表达式。

预搜索

预搜索，又叫零宽断言，又叫环视，它是对位置的匹配，与定位字符（边界字符）类似。

表达式	作用
(?=pattern)	断言此位置的后面能匹配表达式pattern
(?<=pattern)	断言此位置的前面能匹配表达式pattern
(?!pattern)	断言此位置的后面不能匹配表达式pattern
(?<!pattern)	断言此位置的前面不能匹配表达式pattern

regex=love (?=story)匹配的结果如下（匹配“love?”后面是story）：

```
I love story!
I love stories!
```


regex=love (?!story)匹配的结果如下（匹配“love ”后面不能是story）：

```
I love story!
I love stories!
```

运算符的优先级

正则表达式从左到右进行计算，并遵循优先级顺序，这与算术表达式非常类似。下表的优先级从高到低排序。

运算符	描述
\	转义字符

r
loom
room

开发过程中使用正则表达式的流程

1. 分析所要匹配的数据特点，模拟各种测试数据；
2. 利用正则工具，写正则表达式与测试数据进行匹配，从而验证你写的正则；
3. 在程序里调用在正则工具中验证通过的正则表达式。

练习使用正则实现 5 个小功能

电话号码的正则

1. 电话号码由数字和“-”组成
2. 如果包含区号，那么区号为三位或四位，首位是0
3. 区号用“-”和其他数字分割
4. 除了区号，电话号码为7到8位
5. 手机号码为11位
6. 11位手机号码的前2位为“13”，“14”，“15”，“17”，“18”

分析

电话号码分为固话和手机号，首先匹配固话，然后匹配手机号。

- 固话的正则：regex=0\d{2,3}-\d{7,8}
- 手机号的正则：regex=1[34578]\d{9}

所以电话号码的正则：

```
regex=(0\d{2,3}-\d{7,8})|(1[34578]\d{9})
```

2. 如果是15位，则都是数字
3. 如果是18位，最后一位可能为数字或字母X

分析

- 15位数字：regex=\d{15}
- 18位数字：regex=\d{18}
- 17位数字+X：regex=\d{17}X|x

所以省份证号码的正则：

```
regex=(^\d{15}$)|(^\d{18}$)|(^\d{17}X|x$)
```

身份证号码匹配的结果如下：

```
110812198812121234
11081219881212123X
130503670401123
12312321232121212
```

电子邮箱的正则

1. 邮箱格式：用户名@网址.域名
2. 用户名：字母、数字、下划线组成
3. 网址：字母、数字
4. 域名：2-4位字母组成，1-2个域名
5. 不区分大小写

分析

- 用户名：regex=\w+。
- 网址：regex=[a-zA-Z0-9]+。

所以电子邮箱的正则regex=\w+@[a-zA-Z0-9]+(.[a-zA-Z]{2,4}){1,2}。

电子邮箱匹配的结果如下：

- 1-255的正则regex=^([1-9]|[1-9]\d|1\d\d|2[0-5][0-5])。
- 0-255的正则regex=^(\d|[1-9]\d|1\d\d|2[0-5][0-5]) 。

所以 IP地址的正则regex=^([1-9]|[1-9]\d|1\d\d|2[0-5][0-5]).((\d|[1-9]\d|1\d\d|2[0-5][0-5]).){2}(\d|[1-9]\d|1\d\d|2[0-5][0-5])$

IP地址匹配的结果如下：

```
1.1.1.1
12.12.12.12
123.123.123.123
255.255.255.101
172.0.0.1
0.255.255.1
```

日期格式的正则

日期格式：yyyy-mm-dd

分析

- 4位的年，第一位只能是1或2，regex=^([12]\d{3})
- 一年的12个月(01～09和1～12)，regex=^(0?[1-9]|1[0-2])
- 一个月的31天(01～09和1～31)， regex=^((0?[1-9])|((1|2)[0-9])|30|31)

所以 格式为yyyy-mm-dd的日期正则 regex=^([12]\d{3})-(0?[1-9]|1[0-2])-(0?[1-9]|((1|2)[0-9])|30|31)$

yyyy-mm-dd的日期匹配的结果如下：

```
1988-12-12
1988-10-28
1988-10-31
1989-01-01
2008-08-08
0123-13-34
3223-12-09
```

Pattern类：

- Pattern是正则表达式regex的编译表示形式
- 代码：`Pattern pattern = Pattern.compile(regex);`

Matcher类：

- 通过解释Pattern对输入的字符串input执行匹配操作的引擎
- 代码：`Matcher matcher = pattern.matcher(input);`

注意：在Java代码中转义字符“\”要写成“\”才表示一个“\”。比如regex=\d，在Java代码中应该写成“\d”。

Java示例代码：

```
package regex;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class TestRegex {

    public static void main(String[] args) {
        String input = "Hello regex 666!";
        // java中要想表示\需要通过转义字符\进行转义
        String regex = "\\w+";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(input);
        // matches()方法，将输入的整个字符串与给定的正则匹配
        System.out.println(matcher.matches());
        // 结果为：false，因为"Hello regex 666!"不全是\w

        String regex1 = "\\d+";
        Pattern pattern1 = Pattern.compile(regex1);
        Matcher matcher1 = pattern1.matcher(input);
        // find()方法，从输入的字符串里找出与给定的正则匹配的子串
```

短信验证码在目前的互联网应用的非常广泛，在一些重要操作中都需要输入短信验证码来验证身份信息。

列举3条不同的验证码短信内容如下：

1. 【京东】尊敬的用户，634561是您本次的省份验证码，30分钟内有效，请完成验证。
2. 【滴滴】您的验证码是6678，请在页面中提交验证码完成验证。
3. 【百度】376687（动态验证码），请在30分钟内填写。

那么如何通过一个正则表达式来获取到3个不同类型的短信内容里的数字验证码呢？

首先分析以上3条短信内容，找出共同点：

1. 验证码都是数字，可以是4位数字，也可以是6位数字
2. 每条短信都包含“验证码”3个汉字
3. “验证码”3个字与数字的顺序关系，“验证码”3个字可以在数字前，也可以在数字后

按照以上的分析，我们就可以写在正则工具里写正则表达式进行验证了。

1. 4位数字或者6位数字，可以用“\d{4}|\d{6}”来匹配，我们使用捕获组( )来获取数字部分，即regex=(\d{4}|\d{6})

2. 验证码3个字就用“验证码”来匹配，regex=验证码

3. “验证码”3个字在数字前，可以regex=验证码\D(\d{4}|\d{6})，“验证码”3个字在数字后，可以regex=(\d{4}|\d{6})\D验证码，这2个表达式是或的关系，需要用到括号来组织这2个表达式，然后再用或“|”来进行选择，即regex=(验证码\D(\d{4}|\d{6}))|((\d{4}|\d{6})\D验证码)

4. 由于要通过捕获组( )来获取数字内容，又要用括号来组织关系，因此需要把或“|”两边的表达式部分用非捕获组(?:)来标记，因为我们只需要获取数字部分的括号( )匹配到的数字。即regex=(?:验证码\D(\d{4}|\d{6}))|(?:(\d{4}|\d{6})\D验证码)

最后我们把分析到的表达式代入到Java代码完成功能。注意在Java中，反斜杠需要转

```
        inputList.add("【京东】尊敬的用户，634561是您本次的省份验证码，30
分钟内有效，请完成验证。");
        inputList.add("滴滴】您的验证码是6678，请在页面中提交验证码完成验
证。");
        inputList.add("【百度】376687（动态验证码），请在30分钟内填
写。");
        String regex = "(?:验证码\\D*(\\d{4}|\\d{6}))|(?:
(\\d{4}|\\d{6})\\D*验证码)";
        Pattern pattern = Pattern.compile(regex);
        System.out.println("一键获取到的验证码如下：");
        for (String input : inputList) {
            Matcher matcher = pattern.matcher(input);
            if (matcher.find()) {
                for (int i = 1; i <= matcher.groupCount(); i++) {
                    if (matcher.group(i) != null) {
                        System.out.println(matcher.group(i));
                    }
                }
            }
        }
    }
}
```

运行效果：


```
一键获取到的验证码如下：
634561
6678
376687
```

问题2：判断用户密码是否为强密码

用户设置的密码弱，会导致信息安全问题，一般的系统都要求设置强密码。下面是京东注册页面的截图：

以京东注册为例，京东建议使用字母、数字和符号两种及以上的组合，6-20个字符。

下面我们通过正则表达式来完成用户输入的密码是否符合密码规则的校验。首先分析密码要求，如下：

1. 密码包括字母、数字和符号3种字符
2. 必须包含2种及以上的字符
3. 密码长度6-20位
4. 字母包括：A-Za-z，数字包括：0-9，
5. 符号包括32个：`-=][';/.,~!@#$%^&()_+|}{":?><

需要注意的是如果使用32个符号，特殊字符"\"、"["、"]"是需要进行转义的，为了简单直

因此正则可以这么写：

```
regex=^(?![A-Za-z]+$)(?![0-9]+$)(?![@#$]+$)[A-Za-z0-9@#$]{6,20}$
```

解释：

- `^(?![A-Za-z]+$)` 表示从头到位不能全是字母
- `^(?![0-9]+$)` 表示从头到位不能全是数字
- `^(?![@#$]+$)` 表示从头到位不能全是符号@#$
- `^[A-Za-z0-9@#$]{6,20}$` 表示从头到位只能是字母数字符号@#$的集合

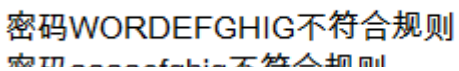
需要注意的是，开始符“^”和预搜索“(?!)”都是零宽的，表示位置，所以开始符“^”只需要在整个正则表达式的开始处写一个即可。

最后我们把分析到的表达式代入到Java代码完成功能。

Java代码片段：

```
String pwdRegex = "^(?![A-Za-z]+$)(?![0-9]+$)(?![@#$]+$)[A-Za-z0-
9@#$]{6,20}$";
        Pattern pattern = Pattern.compile(pwdRegex);
        Matcher matcher;
        for (String pwd : pwdList) {
            matcher = pattern.matcher(pwd);
            if (matcher.find()) {
                System.out.println("密码" + pwd + "符合规则");
            } else {
                System.out.println("密码" + pwd + "不符合规则");
            }
        }
```


密码校验效果：



到此，正则表达式的Chat就结束了，文章有点长，希望对你有用，咱们下次再见。