正则表达式入门使用参考

前言

- 什么是正则表达式

  正则表达式（Regular Expression）是对字符串的模式的定义。正如英文单词“Regular”所言，通俗来讲，是一个有着特定书写规范的表达式。我们平时使用的语言所造的句子也可以称之为表达式，但是这种自然语言的表达式并没有一种特定的形式化表述方式（虽然它由一种语言的文法产生，但是表达方式多种多样）。而接下来介绍的正则表达式则是一种有着明确的形式化书写规范的表达式，我们不妨将其看作对于一般表达式的共性抽象与内在组织方式的描述。

- 正则表达式的用途

  正因为正则表达式有着前面所述的形式化书写规范，它常被用来对字符串或文本进行搜索匹配、查找替换、内容提取等。

- Java的正则表达式

  正则表达式并不局限于某一种变成语言，大家所熟悉的诸如Java、Python，常用于web开发的JavaScript、Ruby，以及Linux的Shell等诸多语言都对正则表达式有着不同的支持，彼此之间虽然有细微的差别，但是整体上大同小异。接下来，我们主要向大家介绍Java的正则表达式，了解之后，在网上简单查阅资料，便可以在不同语言中融会贯通。

Java正则表达式基础

首先，我们先来了解正则表达式的基本字符，这里主要介绍一些基本的、最常用的字符，旨在帮助大家能够快速入门正则表达式。正确理解其含义并对它们了然于心是使用正则表达式的基础。除此之外还有一些在大家的作业中使用相对较少的，在后面我会列举出来，大家也可以据此查阅资料学习。

- 基本字符

  我们所使用的任何英文字符，甚至其他语言字符，例如汉字，都可以作为正则表达式的字符。例如`hello`、`你好`，都是正则表达式。

- \d \D
  - `\d`匹配一个数字字符。例如，使用`\d`可以匹配`0`或者`1`等其他单个数字字符。等效于`[0-9]`（我们后边会介绍）。
  - `\D`是`\d`的补集，匹配一个非数字字符。等效于`[^0-9]`（我们后边会介绍）。
- \s \S
  - `\s`匹配一个空白字符，包括空格、制表符、换页符等。与`[\f\n\r\t\v]`等效（我们后边会介绍）。
  - `\S`是`\s`的补集，匹配一个非空白字符，与`[^\f\n\r\t\v]`等效（我们后边会介绍）。
- \w \W
  - `\w`匹配英文字母、数字、下划线组成的字符集中的字符。与`[A-Za-z0-9_]`等效（我们后边会介绍）。
  - `\W`是`\w`的补集，匹配英文字母、数字、下划线组成的字符集之外的字符。与`[^A-Za-z0-9_]`等效。
- \f \n \r \t \v

  这些字符与大家在C语言课程中学习的转义字符具有相同的含义，分别表示匹配换页符、换行符、回车符、水平制表符、垂直制表符。

知道以上这些，我么就可以开始写一些简单的正则表达式了，比如，我们要匹配一个四位无符号整数（允许有前导0），可以这样写`\d\d\d\d`，但是，我们看到这里面有很多重复的`\d`，而且对于上面的基本字符，使用它们构造正则表达式匹配固定格式的有穷字符串还好（最多是写的长同时很丑陋），如果要匹配一些格式可以变化、长度没有限制的字符串，比如任意长度的无符号整数，只知道这些显然还是不够的。接下来，我们再学习一些符号，通过它们，可以让正则表达式表现得内容更灵活。

- ^ $

  这两个字符用于指定匹配位置。

  - `^`匹配字符串的起始位置。比如，我们想要匹配以一个数字开头的字符串，可以在正则表达式开头写`^\d`。
  - `$`正好相反，匹配字符串的结尾位置，以一个数字结尾的字符串，可以写成`\d$`。

- ? * +

  这三个字符用于对其前面的表达式内容的出现次数进行描述。

  - `?`，零次或一次匹配前面的字符或子表达式。比如要匹配一个单词`apple`的单数和复数形式，可以写成`apple(s)?`。
  - `*`，零次或多次匹配前面的字符或子表达式。例如，`a*`可以匹配空串（即没有任何字符的表达式），`a`，`aa`，`aaa`等等。
  - `+`，一次或多次匹配前面的字符或子表达式。比如我们上面的例子，匹配任意长度的无符号整数（长度为不小于1的整数，且允许前导0），就可以写成`\d+`。

- {}

  对于上面的`? * +`三个符号的进一步扩充，对其前面的表达式内容的出现次数进行描述，让表现更加灵活！

  - `{n}`，正好匹配n次，还是之前那个例子，匹配四位无符号整数（允许有前导0），这时就不用写`\d\d\d\d`了，`\d{4}`即可。
  - `{n,}`，至少匹配n次。之前的`a*`可以写作`a{0,}`，而之前的`\d+`可以写成`\d{1,}`。
  - `{n,m}`，匹配至少n次，至多m次，要求n<=m。还记得使用Google进行搜索时，页面底部的`Gooooooooooogle`么，我们现在匹配`Google`，要求`G`与`gle`中间的o个数大于等于2，小于等于20，那么可以写成`Go{2,20}gle`。

- |

  类似于逻辑运算中的或，比如`ClassA|ClassB`，可以匹配`ClassA`或者`ClassB`。

- []

  再灵活一点！我们上面都是将一个固定的字符或者表达式片段重复一定的次数，现在，我们要让我们选择的那个字符或者表达式片段不再固定，而是可以由各种元素组成。

  - `[xyz]`，字符集，用于匹配任意一个在字符集中的字符，比如上面Class的例子，可以改写成`Class[AB]`。
  - `[^xyz]`，反向字符集，与上面正好相反，用于匹配任意一个字符集之外的字符。
  - `[a-z]`，字符范围，匹配任意一个属于这个范围的字符，上面Class的例子，如果想要匹配`ClassA`到`ClassZ`，可以写成`Class[A-Z]`。
  - `[^a-z]`，反向字范围，匹配任意一个不属于这个范围的字符。

学习了这么多的字符，有没有感觉到我们能用正则表达式匹配很多字符串了呢？但是这里还差一些。请大家回想一下正则表达式的用途，我们使用正则不仅是搜索匹配，同时还有内容提取。这一点用途很广泛（大家的作业中也会经常遇到~~*奸笑*~~）。接下来，我们最后说一下正则表达式的捕获。

- ()

这不是用于指定优先级的括号么！？没错，它不仅能够指定优先级，同时还能够将其中的内容捕获出来用于提取。比如对于日期YYYY/MM/DD，我们要想提取年月日，使用正则的话，可以写成`(\d+)/(\d+)/(\d+)`（我们假定输入日期字符合法），这样匹配后结合Java的`group`方法即可提取，这一点我们稍后介绍。

最后，说一句**注意**，在很多其他语言的**正则表达式**中，使用两个`\`表示不转义，比如`\d`匹配数字字符，而使用`\\d`匹配字符`\d`，但是Java的**正则表达式**与此不同，`\\`表示后面的字符有特殊含义，因此在Java中，匹配一个数字字符，要写成`\\d`。同理，在其他语言中使用一个\取消转义，在Java中就要使用`\\`。

此外，在阅读上面举的例子中，有同学可能意识到了正则表达式不同符号的优先级的问题，在这里做一个补充说明。

运算符	说明
\	转义符
(), (?:), (?=), []（关于捕获我们稍后会详细介绍）	圆括号和方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, $, 任何元字符（即上述的\d\w等）、任何字符	定位点和序列
|	或

以上优先级从上到下由高到低。比如之前我们举过的Google的例子，我们在匹配时使用了`Go{2,20}gle`，这里`{}`的优先级比字符`o`要高，所以没有必要写成`G(o{2,20})gle`。

以上就是Java正则表达式的基础，此外，一些高级的或者使用相对不那么频繁的，鉴于只是一个入门教程，目标在于帮大家建立最基本的认识，同时抛砖引玉，我们略去了这一部分，大家可以参考这个网站，或者搜集其他资料。

正则表达式层次化表示

相信在计组课程中，同学们已经充分理解了层次化设计CPU的可以使设计逻辑更加清晰、易于实现和维护。好的正则表达式也需要层次化划分，这样可以使正则表达式更明晰易读、出现错误更好定位，同时也避免了过度匹配带来的爆栈等风险。

正则表达式层次化表示总体思路

1. 分析字符串的层次结构
2. 从底层开始写，不断使用前一层的“子表达式”，（即使用有意义的变量名代替单纯的表达式，类似于主函数调用子函数）
3. 对于有捕获组的表达式，对捕获组进行命名

我们下面通过两个例子来更具体说明，如何实现正则表达式层次化表示。

电子邮箱

使用正则表达式匹配合法的电子邮箱，并提取邮箱的用户名、域名和顶级域名。约定：

1. 允许邮箱名称与域名有汉字（\u4e00-\u9fa5）
2. 顶级域名要么为全部为汉字，要么全部为英文字母

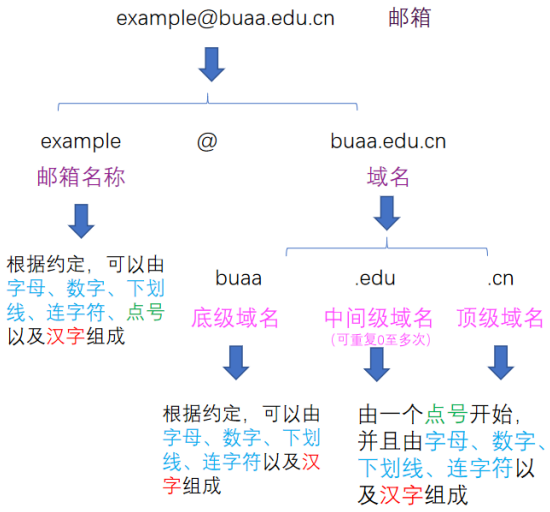
- 不好的写法

```
^([A-Za-z_\\d\\-.\\u4e00-\\u9fa5]+)@([A-Za-z_\\d\\-.\\u4e00-\\u9fa5]+(\\.[A-Za-z_\\d\\-\\u4e00-\\u9fa5]+)*\\.([A-Za-z]+|[\\u4e00-\\u9fa5]+))$
```

我们首先来观察一下，这个写法有如下缺点:

- 存在重复表达式
- 层次不清楚
- 过于冗长，不好维护

- 层次化表示



- 首先，由第一层分解邮箱的表达式

```
String email = username + "@"+ domain;
```

- 然后，对邮箱名称和域名进行拆解

```
String username = "\\.(?<name>["+ ch +"]+)";  // 对捕获组命名
String domain ="(?<domain>"+ buttomLevelDomain + middleLevelDomain +
topLevelDomain + ")"
```

- 之后，对域名进行拆解

```
String topLevelDomain = "(?<tld>["+ en +"]+|["+ zh + "]+)";
String middleLevelDomain = "(\\.["+ ch +"]+)*";
String buttomLevelDomain = "["+ ch +"]+";
```

- 最后，对最基本的组成单元：字母、数字、下划线、连字符以及汉字进行拆解

```
String zh = "\\u4e00-\\u9fa5";
String en = "\\w\\-";  // 其中\w与[A-Za-z0-9_]等效
String ch = zh + en;
```

日期匹配

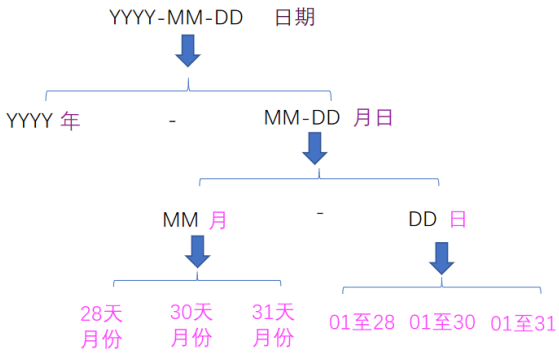
约定:

1. 日期满足格式YYYY-MM-DD
2. 对于年月日，允许有前导0
3. 为了简化处理，规定2月为28天

- 不好的写法

```
([0-9]{3}[1-9]|[0-9]{2}[1-9][0-9]{1}|[0-9]{1}[1-9][0-9]{2}|[1-9][0-9]{3})-(((0[13578]|1[02])-(0[1-9]|[12][0-9]|3[01]))|((0[469]|11)-(0[1-9]|[12][0-9]|30))|(02-(0[1-9]|[1][0-9]|2[0-8])))
```

- 层次化表示



  - 首先拆分日期：

```
String date = year + "-"+ monthAndDay;
```

  - 然后拆分年和月日

```
String year = "([0-9]{3}[1-9]|[0-9]{2}[1-9][0-9]{1}|[0-9]{1}[1-9][0-9]{2}|[1-9][0-9]{3})";

String monthAndDay ="(" + febMonth + "-" + febDay + ")|(" + bigMonth + "-" + bigMonthDay + ")|("+ smallMonth + smallMonthDay + ")";
```

  - 之后拆分月和日

```
String febMonth = "02";
String bigMonth = "(0[13578]|1[02])";
String smallMonth = "(0[469]|11)";

String febDay = day0 + "|2[0-8]";
String bigMonthDay = day1 + "|3[01]";
String smallMonthDay = day1 + "|30";

String day0 = "0[1-9]|1[0-9]";  //01-19日
String day1 = "0[1-9]|[12][0-9]";  // 01-29日
```

Java正则表达式API的使用

我们前面学习了Java正则表达式的书写基础，接下来，我们在会书写基本的正则表达式的基础上学习Java中对于正则表达式支持的相关API的使用。

- `java.util.regex.Pattern`，Pattern类。Pattern类顾名思义，就是用于构造可以被使用的Java正则表达式的形式，通俗来讲，我们使用Pattern类去由我们写的正则表达式转化成Java可以使用的pattern，使用这个pattern来进行后面的匹配、替换或者捕获提取等操作。下面介绍具体用法。
  - `Pattern.compile()` 由于Pattern类**没有public的构造方法**，所以要想生成pattern对象实例，使用compile，返回值为pattern对象实例，参数即我们上面学过的正则表达式。
  - `Pattern.match()` 故名思义，使用match来进行匹配，第一个参数是我们上面生成的pattern，第二个参数是一个待匹配的字符串，返回值为`boolean`类型，表示是否成功匹

配。

- `java.util.regex.Matcher`，Matcher类。前面我们看到，使用Pattern类的方法即可做一些简单的匹配了，但是我们只能知道是否成功匹配，要想获取匹配的精确位置，以及进行捕获和替换，就需要使用到Matcher类。
  - Matcher类与Pattern类一样**没有public的构造方法**，构造matcher对象，需要使用之前我们构造的pattern对象实例的matcher()方法。
  - 索引方法

    `start()` `end()`，这两个方法分别返回正则表达式匹配的字符串在输入的字符串中的起始位置索引和结束位置的后一位的索引。**注意：在使用这两个方法时，要先调用matcher实例的find()方法，返回布尔值。**否则，会抛出无匹配异常。另外，如果有多个匹配，想要遍历全部匹配时，每调用一次find()方法，就会匹配下一个，因此可以使用一个while循环进行遍历。

    ```
    Pattern p = Pattern.compile("\\d+\\s\\d+");
    Matcher m = p.matcher("12 34 56 78");
    while (m.find()) {
         System.out.println("start:" + m.start());
         System.out.println("end:" + m.end());
    }
    ```

    ```
    // 如果没有调用find方法
    Exception in thread "main" java.lang.IllegalStateException: No match
    available
        at java.util.regex.Matcher.start(Matcher.java:343)
        at Main.main(Main.java:8)
    ```

  - 查找与替换方法

    `replaceFirst()` `replaceAll()`，这两个方法用于对matcher匹配的内容进行替换，字面意义上就可以知道，前者只替换初次匹配，后者替换全部匹配。参数为要替换为的字符串。

    ```
    String str0 = "bug0 bug1 bug233";
    Pattern p = Pattern.compile("bug\\d+");
    Matcher m = p.matcher(str0);
    String str1 = m.replaceFirst("fix");
    String str2 = m.replaceAll("fix");
    System.out.println(str0);
    System.out.println(str1);
    System.out.println(str2);

    // output
    // bug0 bug1 bug233
    // fix bug1 bug233
    // fix fix fix
    ```

    我们可以看到，这个替换是通过返回值进行替换的，不会对原字符串进行修改。

  - 捕获方法

    之前已经向大家提到过捕获的概念了，所谓捕获，就是我们希望能够从正则表达式中提取出我们想要的一部分，我们使用`()`来构造捕获组，捕获组是可以嵌套的。这样的功能，虽然前面的start和end方法可以一定程度上实现，但是它们是有很多局限于不方便之处的。提取一部分匹配的内容，捕获是更好的选择。

    - 简单的情况

我们先来看一个实例，这个是我们之前提到过的年月日提取的例子。

```
String str = "2019/09/27";
Pattern p = Pattern.compile("(\\d+)/(\\d+)/(\\d+)");
Matcher m = p.matcher(str);
if (m.find()) {
    System.out.println(m.group(1));
    System.out.println(m.group(2));
    System.out.println(m.group(3));
} else {
    System.out.println("Nothing matched");
}
```

```
Year: 2019
Month: 09
Day: 27
```

这个功能在同学们接下来的作业中会非常实用，建议同学们一定要掌握。接下来我们再举一个例子，这个例子与同学们作业的主题有相关，但是为了保证同学们能够独立思考完成作业，我们只举一个简化版的例子：输入一个幂函数，提取并输出幂函数的指数。

```
Scanner input = new Scanner(System.in);
String expression = input.nextLine();
String sp = "[ \\t]*";
String num = "[+-]?\\d+";
String exp = sp + "\\*\\*" + sp + "(" + num + ")" + sp;
String powFunc = sp + "x" + sp + "(" + exp + ")?";
Pattern p = Pattern.compile(powFunc);
Matcher m = p.matcher(expression);
if (m.find()) {
    String result = m.group(2);
    if (result != null) {
        System.out.println(result);
    } else {
        System.out.println(1);
    }
} else {
    System.out.println("no valid expression");
}
```

针对这个例子简单说明一下。首先，在书写正则表达式的时候，希望同学们能够根据接下来同学们拿到的指导书所规定的格式，按照逻辑对正则表达式进行拆解和构造，提取它们的公共部分，使用不同的子成分连接成最后的正则表达式，而不是针对每种情况简单堆砌，不然程序不易被别人理解，同时会为自己debug造成困难。上面只是我的一种写法，仅做抛砖引玉。

其次，关于转义。前面说过，使用`\\`进行转义与取消转义，这里的`*`本身是正则表达式的字符，将其转义为普通字符，使用`\\*`。而对于上面`num`中，由于`+`在字符集中，这时不必使用`\\`即会被解释为`+`这个字符。

最后，关于捕获组中使用`group`方法中的参数，我这里使用了`group(2)`，指的是提取第二个捕获组。那么捕获组如何编号呢？非常简单，0是一个特殊的捕获组，代表整个正则表达式，从1往后，按照左括号的出现顺序依次标号，举个例子：

```
 A      (       ( B C ) ( D   ( E ) ) )
0(all)  1(BCDE) 2(BC)   3(DE) 4(E)
```

- 其他一些实用技巧

  相信大家看到了，上面的代码很丑陋。就捕获组而言：

  - 一，写一个`group(2)`，使用数字标号，不直观，而且同学们的作业不可能这么简单，要是捕获组多了，用数字标号，自己看着都很不清楚。
  - 二，对于`exp`外的括号，我纯粹是为了表示一个优先级，并不想捕获。

  针对上面两个问题，我们介绍两个使用的技巧。

  - 捕获组的命名

    如果大家看一下Java8的API文档中对于group参数的描述，则可以看到Matcher类的group方法是重载的，其参数可以是int，也可以是String：

    > `public String group(String name)`
    >
    > Returns the input subsequence captured by the given named-capturing group during the previous match operation.

    这就是我们要介绍的命名捕获组。使用非常简单，只要在将`(content)`改写为`(?<name>content)`即可，其中name不需要写成加双引号的字符串。在上面的示例中，可以将`String exp = sp + "\\*\\*" + sp + "(" + num + ")" + sp;`改写为`String exp = sp + "\\*\\*" + sp + "(?<exponent>" + num + ")" + sp;`，然后在使用group时，使用`m.group("exponent")`即可（这里要加双引号）。这样，正则表达式每一部分的含义就非常清楚了。

  - 非捕获组

    我们在这里先介绍一种最基本的非捕获组，即如果不需要捕获`(content)`中的内容，可以写作`(?:content)`。例如，上面的`String powFunc = sp + "x" + sp + "(" + exp + ")?";`可以改写为`String powFunc = sp + "x" + sp + "(?:" + exp + ")?";`。

    这里需要注意，使用非捕获组，对应的`()`不算入捕获组的编号中，对应的，我们上面的代码应当调整为`m.group(1)`。

    除此之外，非捕获组还有两组四个，有零宽度（正/负）（先行/后发）断言，它们主要用于限制待匹配字符串的前后的条件，由于是正则入门，我们在此不进行过多介绍，如果大家感兴趣，可以参考这篇博客，或者查找其他资料自行学习。

写在最后

以上就是Java正则表达式一些最基础的知识，希望大家能够通过自己去实践来掌握他们，然后多去查阅资料，以此为基础去学习更多的相关知识，为自己将接下来的OO作业以及将来实际参与开发打下坚实的基础。