Java 与正则表达式

章铁飞

正则表达式是描述和处理字符串的工具，可以对字符串进行查找、替换、分割等操作。很多文本编辑器与编程语言都支持正则表达式，java 也不例外。虽然正则表达式并不复杂，但要灵活应用还需要基本知识与案例的操练。首先，来看两个应用正则表达式的简单案例。

案例 1：String s=132ffdsaf3242fdsfsadfsadf232,使用正则表达式从中找出 1323242232。
案例 1 主要目的是将字符串 s 中的英文字符删除，只剩下数字。一般的做法是依次处理 String s 中的每个字符，如果不是英文字符，则删除，最后剩下的就是数字。相比而言，采用正则表达式的方式就要简单很多，代码如下：

```
public class Demo1{
    static String s="132ffdsaf3242fdsfsadfsadf232";
    public static void main(String[] args){
        String str = s.replaceAll("[^0-9]", "");
        System.out.println(str);
    }
}
```

replaceAll()是一个替换操作，第二个参数表示空，也就把第一个参数表示的字符用空字符替换，即删除。第一个参数"[^0-9]"是一个正则表达式，什么意思呢？符号 0-9 表示 0 到 9 的数字，[]表示出现在括号中的任意字符，[^]表示括号中的任意字符的取反，最终"[^0-9]"表示出现的任意非数字字符。所以，语句 replaceAll("[^0-9]", "")表示将任意非数字的字符删除。

案例 2：String str="2008-03-23 12:30:01"，使用正则表达式将字符串变成 20080323123001。
案例 2 的目的是将其中任意非数字的字符删除，当然可用案例 1 中的正则表达式解决。我们尝试使用正则表达式中其他的表达方式来解决该问题，代码如下：

```
public class Demo2{
    public static void main(String[] args){
        String str = "2008-03-23 12:30:01";
        String str2 = "";
        String[] res = str.split("\\D");
        for(int i=0; i<res.length; i++){
            str2 += res[i];
        }
        System.out.println(str2);
    }
}
```

语句 str.split("\\D")中，表示 str 按给定的参数符号为界，分割(split)为多个子字符串。以字符串 s="2008-03-23"，如果调用 s.split("-")，字符串 s 就以"-"为界，将 s 分割为三个子字符串"2008","03"和"23"。但代码中的 str，很难以具体的字符为界，因为除了"-"间隔，还有空格和":"等。此时就需要使用正则表达式"\\D"，"\D"表示非数字字符，而"\"表示下一个字符，比如已知"\n"表示匹配字符 n，而"\\n"表示匹配换行符"\n"；所以"\\D"表示匹配非数字字符。

Construct	Description
[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z, or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z] (subtraction)

表 1 字符正则符

表 1 是简单的一类表达式，左侧是正则表达式，右侧是功能描述。

Construct	Description
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

表 2 字符正则符

Greedy	Reluctant	Possessive	Meaning
X?	X??	X?+	*X*, once or not at all
X*	X*?	X*+	*X*, zero or more times
X+	X+?	X++	*X*, one or more times
X{n}	X{n}?	X{n}+	*X*, exactly *n* times
X{n,}	X{n,}?	X{n,}+	*X*, at least *n* times
X{n,m}	X{n,m}?	X{n,m}+	*X*, at least *n* but not more than *m* times

表 3 数量正则符

Boundary Construct	Description
^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input

\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

表 4 边界正则符

案例 3：假设我们要搜索美国的社会安全号，格式是 999-99-9999，对应的正则表达式：
[0-9]{3}\-[0-9]{2}\-[0-9]{4}

[0-9]：0 到 9 的数字；
{3}：出现 3 次；
\-：匹配-；

如果"-"也可以不出现，比如 999999999，只需要增加"?"符，表示或者没有：
[0-9]{3}\-?[0-9]{2}\-?[0-9]{4}

案例 4：匹配生日格式“June 26, 1951”，对应的正则表达式：
([a-z]+) \s+[0-9]{1,2},\s*[0-9]{4}

Java 关于正则表达式的包 java.util.regex，主要包含三个类：Pattern，Matcher 和 PatternSyntaxException。

Pattern 类用于创建一个正则表达式，创建一个匹配模式，它的构造方法私有，不可以直接创建，需要通过 Pattern.compile 方法创建一个正则表达式。比如案例 1 为例：

```
import java.util.regex.*;
public class T1{
    public static void main(String[] args){
        Pattern p = Pattern.compile("\\D");
        System.out.println(p.pattern()); //输出为"\D"；
    }
}
```

Pattern 类中主要的方法有 matcher，并且需要搭配 Matcher 类一起使用，因为 matcher 方法返回 Matcher 类。而 Matcher 类中常用的方法是 find()，group(), start()和 end()，对应的说明读者可自行查阅 Java API 文档。下面是官方文档中的例子：

```
import java.io.Console;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class RegexTest{

    public static void main(String[] args){
        Console console = System.console();
        if (console == null) {
            System.err.println("No console.");
```

```
                System.exit(1);
            }
            while (true) {
                Pattern pattern =
                Pattern.compile(console.readLine("%nEnter your regex: "));
                //System.out.println(pattern.pattern());

                Matcher matcher =
                pattern.matcher(console.readLine("Enter input string to search: "));

                boolean found = false;
                while (matcher.find()) {
                    console.format("I found the text" +
                        " \"%s\" starting at " +
                        "index %d and ending at index %d.%n",
                        matcher.group(),
                        matcher.start(),
                        matcher.end());
                    found = true;
                }
                if(!found){
                    console.format("No match found.%n");
                }
            }
        }
}
```

主要可以划分为两个步骤：

1. 通过终端输入正则表达式：

Pattern pattern = Pattern.compile(console.readLine("%nEnter your regex: "));

2. 基于输入正则表达式的模式，查找输入字符串中是否出现对应的模式，如果出现则输出其对应的起始点和终点位置；如果多个匹配，则输出多次；

Matcher matcher = pattern.matcher(console.readLine("Enter input string to search: "));

matcher.find()用于查找下个匹配的字符串，matcher.group()用于输出匹配的字符串。以案例 2 为例，输出其中每个数字:

```
Enter your regex: \d+
Enter input string to search: 2008-03-23 12:30:01
I found the text "2008" starting at index 0 and ending at index 4.
I found the text "03" starting at index 5 and ending at index 7.
I found the text "23" starting at index 8 and ending at index 10.
I found the text "12" starting at index 11 and ending at index 13.
I found the text "30" starting at index 14 and ending at index 16.
I found the text "01" starting at index 17 and ending at index 19.
```

下面的数量正则表达式：

```
Enter your regex: a{2}
Enter input string to search: afdsadfafdaafd
I found the text "aa" starting at index 10 and ending at index 12.
```

因此上述程序可以作为正则表达式的测试工具。

案例 5：编写一 java 程序，将 C 语言的常数声明语句修改为对应的 java 常数声明语句，比如：

C 语言：#define PI 3.141592654

转化为：

Java：private static final double PI = 3.141592654

假设 cl.c 是包含常数的 C 语言文件，转化为对应的 java 版本 jl.java。

cl.c:

```
#define     PI          3.141592654
#define     EPOCH       85
#define     EPSILONg    279.611371   /*solar ecliptic long at EPOCH */
#define     RHOg        282.680403   /* solar ecliptic longof perigee at EPOCH */
#define     ECCEN       0.01671542   /* solar orbiteccentricity */
#define     lzero       18.251907    /* lunar meanlong at EPOCH */
#define     Pzero       192.917585 /* lunar mean long ofperigee at EPOCH */
#define     Nzero       55.204723    /* lunar meanlong of node at EPOCH */
```

以下是供参考主体代码行，请读者自己想出其他的正则表达式方案；另外，请读者补上打开文件 cl.c，逐行处理，并且写文件 jl.java 的辅助代码。

C2Java.java:

```
public class C2Java{
    public static void main(String[] args){
        String str = "#define Pzero 192.917585 /* lunar mean long ofperigee at EPOCH */";
        String jstr = "";

        String[] s = str.split("[\\s\\t]+");
        s[0] = "private static final double "; //overwrite #define;

        jstr = s[0] + s[1] +"="+s[2] +";   //";
        for(int i=3; i<s.length; i++)
        {
            jstr = jstr+" "+s[i];
        }
        System.out.println(jstr);
    }
}
```

练习 1：

读取一个文件，输出该文件中所有的 11 位手机号码。测试文件见附件。

http://www.cnblogs.com/0201zcr/p/4994724.html

练习 2：
读取一个文件，输出该文件中所有的电子邮箱地址。测试文件见附件。

练习 3：完成案例 5 的完整代码。