

1、课程名称：正则表达式



魔乐科技**JAVA**课堂
www.mldnjava.cn

我们的课程·一切为了就业

JAVA SE基础课程

正则表达式

北京**MLDN**软件教学研发中心

李兴华

培训咨询热线：010-51283346 院校合作：010-62350411
官方JAVA学习社区：bbs.mldn.cn

2、知识点

2.1、上次课程的主要知识点

- 1、 日期操作：Date、DateFormat、SimpleDateFormat、Calendar
- 2、 StringBuffer 类的使用，String 和 StringBuffer 的区别
- 3、 对象克隆：使用 Object 类中的 clone()方法，但是对象所在的类必须实现 Cloneable 接口
- 4、 大数操作：BigInteger、BigDecimal
- 5、 比较器，两种比较器（Comparator、Comparable），及基本排序原理
- 6、 国际化程序的实现思路：通过资源文件设置要显示的内容，并且通过 Locale 找到指定的资源文件
- 7、 Runtime 和 System 类以及垃圾收集操作，Object 类中的 finalize()方法的使用。
- 8、 Math 和 Random 类

2.2、本次预计讲解的知识点

- 1、 正则表达式的作用
- 2、 Pattern 和 Matcher 类
- 3、 String 对正则的支持

3、具体内容

正则表达式：是在 JDK 1.4 之后加入到 Java 的开发环境之中的，在 JDK 1.4 之前如果要想使用正则表达式进行开发，则必须从网上单独下载 Apache 的一个正则的表达式开发包，正则最早是从 PHP 中开始兴起的，主要的作用可以非常方便完成一些复杂的严整功能等基本实现。

3.1、认识正则（理解）

下面通过一个程序来简单了解一下正则有哪些用处。

例如：现在有如下的一个要求：要求判断一个字符串是否由数字组成。

实现一：不使用正则

- 将字符串变为字符数组，之后将数组中的每个内容取出进行验证。

```
package org.lxx.regexdemo;

public class RegexDemo01 {

    public static void main(String[] args) {

        String str = "12345678"; // 此字符串现在是由数字组成
        char c[] = str.toCharArray(); // 将字符串变为字符数组
        boolean flag = true;
        for (int x = 0; x < c.length; x++) { // 进行循环验证
            if (!(c[x] >= '0' && c[x] <= '9')) {
                flag = false;
                break; // 退出循环
            }
        }
        if (flag) {
            System.out.println("字符串由数字组成!");
        } else {
            System.out.println("字符串由非数字组成!");
        }
    }
}
```

以上是实现了基本的操作，但是现在只是一个小小的验证，已经编写了很多的行，如果更加复杂的严整呢？

实现二：使用正则实现

```
package org.lxx.regexdemo;

public class RegexDemo02 {
```

```
public static void main(String[] args) {
    String str = "12345as678";// 此字符串现在是由数字组成
    if (str.matches("\\d+")) {
        System.out.println("字符串由数字组成!");
    } else {
        System.out.println("字符串由非数字组成!");
    }
}
```

以上的操作明显比第一种实现更加容易，而且代码较少，那么在操作中使用的“\d+”实际上就属于正则表达式。

3.2、正则表达式（重点）

如果要知道有多少种正则表达式，则可以观察 java.util.regex 包中的 Pattern 类，里面列出全部的正则表达式内容。

字符匹配

No.	表达式	描述
1	[abc]	表示取值可能是 a，可能是 b，可能是 c
2	[^abc]	表示取值不是 a、b、c 的任意一个内容
3	[a-zA-Z]	表示全部的字母，大写和小写。[a-z]表示小写字母，[A-Z]表示大写字母

简短表达式

No.	表达式	描述
1	\d	表示由数字组成
2	\D	表示由非数字组成
3	\s	表示有空格组成，空格包含了“\n”、“\t”之类的
4	\S	表示由非空格组成
5	\w	表示由字母、数字、下划线组成
6	\W	表示由非字母、数字、下划线组成

列出出现的次数（以 X 表示一个完整的正则）

No.	表达式	描述
1	X?	表示正则表达式出现 0 次或 1 次
2	X*	表示正则表达式出现 0 次、1 次或多次
3	X+	表示正则表达式出现 1 次或多次
4	X{n}	表示出现的长度正好是 n 次
5	X{n,}	表示出现的长度大于 n 次
6	X{n,m}	表示出现的长度正好是 n 到 m 次

关系运算

No.	表达式	描述
1	X Y	要么是 X 的正则，要么是 Y 的正则
2	(X)	表示一组规范

但是，如果要想使用以上的正则表达式，则需要 Pattern 类和 Matcher 类的支持。

在 Pattern 类中需要指定要操作的正则表达式规范，而在 Matcher 类进行验证。

3.3、Pattern 类（理解）

java.util.regex.Pattern 类是正则操作的最重要的一个类，所有的正则规范需要在 Pattern 类中进行指定，此类中的操作方法如下：

No.	方法名称	类型	描述
1	public static Pattern compile(String regex)	普通	通过此方法取得 Pattern 实例，并设置正则
2	public Matcher matcher(CharSequence input)	普通	为 Matcher 类实例化
3	public String[] split(CharSequence input)	普通	字符串拆分
4	public String pattern()	普通	返回使用的正则表达式

在此类中构造方法本隐藏起来，所以需要通过 compile 进行对象的实例化操作，下面通过 Pattern 类完成一个拆分操作。例如：现在有如下字符串 “a1b22C333D4444E55555”，要求按照数字拆分。

那么这种时候使用正则拆分是最方便的，因为如果按照之前的方法拆分，太麻烦了。

此时应该使用 “\d” 进行数字的匹配，但是每一个 “\d” 只能表示一位数字，在此情况下需要表示多次，所以要使用 “\d+”，表示数字可能出现 1 次或多次。

```
package org.lxh.regexdemo;
import java.util.regex.Pattern;
public class PatternDemo {
    public static void main(String[] args) {
        Pattern pat = Pattern.compile("\\d+"); // 实例化Pattern类对象
        String str = "a1b22C333D4444E55555";
        String s[] = pat.split(str); // 进行字符串的拆分
        for (int x = 0; x < s.length; x++) {
            System.out.print(s[x] + "、");
        }
    }
}
```

Pattern 类主要是定义正则规范，并进行字符串拆分操作的。

3.4、Matcher 类（理解）

Matcher 类的主要功能是用于进行正则的匹配，通过 Pattern 类中定义完的正则，再使用 Matcher 类进行验证或者替换。

No.	方法名称	类型	描述
1	public boolean matches()	普通	进行正则的匹配
2	public String replaceAll(String replacement)	普通	替换全部
3	public String replaceFirst(String replacement)	普通	替换第一个

下面使用操作验证以上的方法。

范例：进行字符串的验证

如果要想进行字符串的验证操作，则需要 Pattern 类指定严整的正则，最终使用 Matcher 类进行匹配，例如，现在有如下车牌号码的规定：“XXX – XX - XXX”，其中 “X” 都表示一个个的数字，现在给出一个车牌号码，验证此号码是否正确。

此时的正则表达式：“\d{3}-\d{2}-\d{3}”。

之后使用 Pattern 类中的 matcher()方法，设置要验证的字符串进行最终的验证。

```
package org.lxx.regexdemo;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class MatcherDemo01 {
    public static void main(String[] args) {
        String str = "111-22-333"; // 定义字符串
        String pat = "\\d{3}-\\d{2}-\\d{3}"; // 定义正则
        Pattern p = Pattern.compile(pat);
        Matcher m = p.matcher(str);
        if (m.matches()) {
            System.out.println("验证成功!");
        } else {
            System.out.println("验证失败!");
        }
    }
}
```

使用 `Matcher` 类还可以完成替换的功能。

范例：完成替换功能的实现

```
package org.lxx.regexdemo;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class MatcherDemo02 {
    public static void main(String[] args) {
        String str = "alb22C333D4444E55555"; // 定义字符串，将数字全部替换成“☆”
        String pat = "\\d+"; // 定义正则
        Pattern p = Pattern.compile(pat);
        Matcher m = p.matcher(str);
        System.out.println(m.replaceAll("☆")); // 替换
    }
}
```

以上的这些操作好想在 `String` 中都存在。实际上在 JDK 1.4 之后 `String` 类就被修改了。

3.5、String 类对正则的支持（重点）

在 JDK 1.4 之后加入了正则，随后又更新了 `String` 的操作类，因为在使用正则中，所有的内容通过字符串表示的比较多。在 `String` 类中有以下的方法可以完成对正则的支持：

No.	方法名称	类型	描述
1	public boolean matches(String regex)	普通	指定正则规则，并进行验证
2	public String replaceAll(String regex,String replacement)	普通	字符串替换，支持正则
3	public String replaceFirst(String regex,String replacement)	普通	替换第一个，支持正则
4	public String[] split(String regex)	普通	字符串拆分，支持正则

在开发中如果要想使用正则，基本上都是直接应用 `String` 类，很少去直接使用 `Pattern` 类或 `Matcher` 类，因为 `String` 的功能已经足够强大了。

范例：使用 `split` 完成拆分操作

- 给定一个 IP 地址，要求 “.” 拆分

```
package org.lxx.regexdemo;

public class StringSplit {

    public static void main(String[] args) {
        String ip = "192.168.1.3"; // 定义IP地址
        String str[] = ip.split("\\.");
        for (int x = 0; x < str.length; x++) {
            System.out.print(str[x] + "、");
        }
    }
}
```

只要拆不开，就加入转义操作。

范例：验证操作

- 要求验证一个 email 地址是否合法
- 正则：“`\\w+@\\w+\\.\\w+`”

```
package org.lxx.regexdemo;

public class StringMatches {

    public static void main(String[] args) {
        String ip = "aa@aa.com"; // 定义email地址
        System.out.println(ip.matches("\\w+@\\w+\\.\\w+"));
    }
}
```

以上的验证非常方便的完成了，但是这样的正则是否真的合理呢？

对于 email 验证来说，域名的后缀只有有限的几个，所以，此时以上的验证并不符合于真实的要求。

取值范围 “com、com.cn、net、cn、org、edu、gov”。

```
package org.lxx.regexdemo;

public class StringMatches {

    public static void main(String[] args) {
        String ip = "aa@aa.com.cn"; // 定义email地址
        System.out.println(ip
            .matches("\\w+@\\w+.(com|com.cn|cn|net.cn|org|edu|gov)"));
    }
}
```

4、总结

- 1、正则可以方便的完成验证的操作，正则表达式是一组标准性的规范，在各个语言都可以使。
- 2、`String` 类对正则有所支持，这一点在以后的开发中将经常使用。