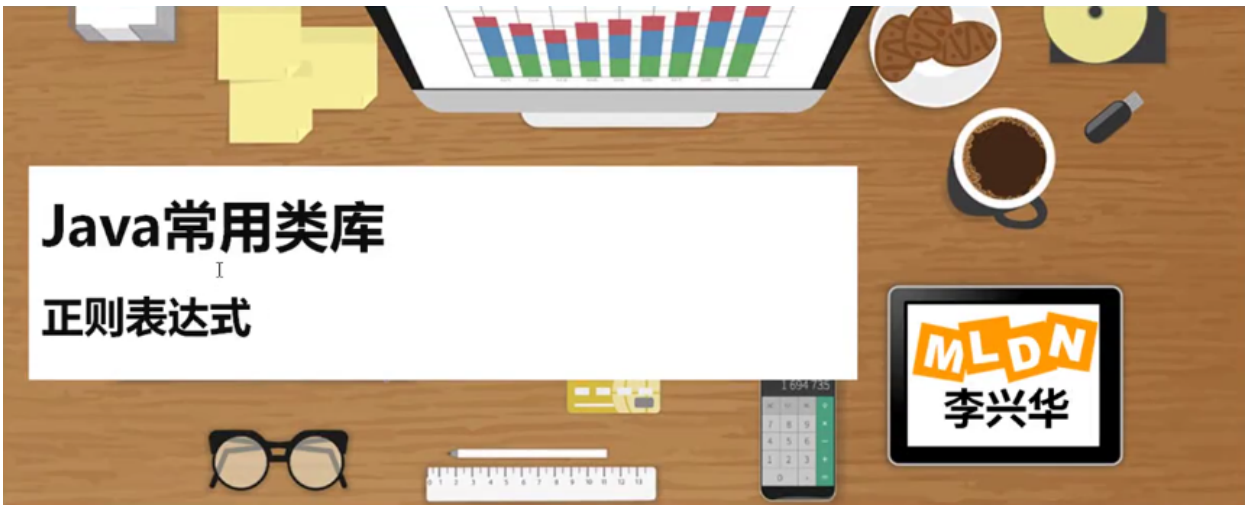


博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

通过之前一系列的分析可以发现，String是一个非常万能的类型，因为String不仅仅可以支持　有各种字符串的处理操作，也支持有向各个数据类型的转换功能，所以在项目的开发之中，只要是用户输入的信息基本上都用String表示。于是在向其它数据类型转换的时候，为了保证转换的正确性，往往需要对进行一些复杂的验证处理，那么这种情况下如果只是单纯的依靠String类中的方法是非常麻烦的。

■认识正则表达式

现在假设有一个字符串，要求判断字符串是否由数字所组成，如果由数字所组成则将其变为数字进行乘法计算。

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "123" ;
		if (isNumber(str)) {
			int num = Integer.parseInt(str) ;
			System.out.println(num * 2);
		}
	}
	public static boolean isNumber(String str) {
		char data [] = str.toCharArray() ;
		for (int x = 0 ; x < data.length ; x ++) {
			if (data[x] > '9' || data[x] < '0') {
				return false ;
			}
		}
		return true ;
	}
}
```

实际上这种验证的功能是非常简单的，但是这如此简单的功能却需要开发者编写大量的程序逻辑代码，那么如果是更加复杂的验证呢？那么在这样的情况下，对于验证来讲最好的做法就是利用正则表达式来完成。

范例：使用正则表达式实现同样的效果

```
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        String str = "123" ;
        if (str.matches("\\d+")) {
            int num = Integer.parseInt(str) ;
            System.out.println(num * 2);
        }
    }
}
```

正则表达式最早是从Perl语言里面发展而来的，而后在JDK1.4以前如果需要使用到正则表达式的相关定义则需要单独引入其它的*.jar文件，但是从JDK1.4之后，正则已经默认被JDK所支持，并且提供有java.util.regex开发包，同时针对于String类也进行了一些修改，使其可以有方法直接支持正则处理。

使用正则最大的特点在于方便进行验证处理，以及方便进行复杂字符串的修改处理。

■正则标记（背）

如果要想进行正则的处理操作，那么就首先需要对常用的正则标记有所掌握，从JDK1.4开始提供有java.util.regex开发包，这个包里面提供有一个Pattern程序类，在这个程序类里面定义有所有支持的正则标记。

1、【数量：单个】字符匹配：

·任意字符：表示由任意字符组成；

·\\：匹配"\"；

·\n：匹配换行；

·\t：匹配制表符；

```
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        String str = "a" ;      // 要判断的数据
        String regex = "a" ; // 正则表达式
        System.out.println(str.matches(regex));
    }
}
```

2、【数量：单个】字符集（可以从里面任选一个字符）：

·[abc]：表示可能是字母a、b、c中的任意一个；

·[^abc]：表示不是由字母a、b、c中的任意一个；

·[a-zA-Z]：表示由一个任意字母所组成，不区分大小写；

·[0-9]：表示由一位数字所组成；

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "c" ;	// 要判断的数据
		String regex = "[abc]" ;	// 正则表达式
		System.out.println(str.matches(regex));
	}
}
```

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "1" ;	// 要判断的数据
		String regex = "[a-zA-Z]" ;	// 正则表达式
		System.out.println(str.matches(regex));
	}
}
```

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "1" ;	// 要判断的数据
		String regex = "[0-9]" ;	// 正则表达式
		System.out.println(str.matches(regex));
	}
}
```

3、【数量：单个】简化字符集：

·.：表示任意的一个字符；

·\d：等价于"[0-9]"范围；

·\D：等价于“[^0-9]”范围；

·\s：匹配任意的一位空格，可能是空格、换行、制表符；

·\S：匹配任意的非空格数据；

·\w：匹配字符、数字、下划线，等价于“[a-zA-z_0-9]”；

·\W：匹配费字符、数字、下划线，等价于“[^a-zA-z_0-9]”；

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "#" ;	// 要判断的数据
		String regex = "." ;	// 正则表达式
		System.out.println(str.matches(regex));
	}
}
```

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "a\t" ;	// 要判断的数据
		String regex = "\\D\\s" ;	// 正则表达式
		System.out.println(str.matches(regex));
	}
}
```

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
```

```
        String str = "" ;       // 要判断的数据
        String regex = "\\w*" ;   // 正则表达式
        System.out.println(str.matches(regex));
    }
}
```

4、边界匹配：

·^：匹配边界开始；

·$：匹配边界开始；

5、数量表示，默认情况下只有添加上了数量单位才可以匹配多位字符：

·表达式?：该正则可以出现0次或1次；

·表达式*：该正则可以出现0次、1次或多次；

·表达式+：该正则可以出现1次或多次；

·表达式{n}：表达式的长度正好为n次；

·表达式{n,m}：表达式的长度为n次以上；

·表达式{n,m}：表达式的长度在m~m次；

```
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        String str = "ax" ;   // 要判断的数据
        String regex = "\\w{3,}" ;// 正则表达式
        System.out.println(str.matches(regex));
    }
}
```

6、逻辑表达式：可以连接多个正则：

·表达式X表达式Y：X表达式之后紧跟上Y表达式；

·表达式X|表达式Y：有一个表达式满足即可；

·（表达式）：为表达式设置一个整体描述，可以为整体描述设置数量单位；

■String类对正则的支持

在进行正则表达式大部分处理的情况下都会基于String类来完成，并且在String类里面提供有如下与正则有关的操作方法；

No	方法名称	类型	描述
01	public boolean matches(String regex)	普通	将指定字符串进行正则判断
02	public String replaceAll(String regex,String replacement)	普通	替换全部
03	public String replaceFirse(String regex,String replacement)	普通	替换首个
04	public String[] split(String regex)	普通	正则拆分

05	public String[] split(String regex,int limit)	普通	正则拆分

下面通过一些具体的范例来对正则的使用进行说明。

范例：实现字符串替换（删除掉非字母与数字）

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "JILO&*()@#$UISD&*(#$HUK34rwyhui*()@#$*()@#$" ;  // 要判断的数据
		String regex = "[^a-zA-Z0-9]+" ;  // 正则表达式
		System.out.println(str.replaceAll(regex, ""));
	}
}
```

范例：实现字符串的拆分

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "a1b22c333d4444e55555f666666g" ; // 要判断的数据
		String regex = "\\d+" ;  // 正则表达式
		String result [] = str.split(regex) ;
		for (int x = 0 ; x < result.length ; x ++) {
			System.out.print(result[x] + "、");
		}
	}
}
```

在正则处理的时候对于拆分与替换的操作相对容易一些，但是比较麻烦的是数据验证部分。

范例：判断一个数据是否为小数，如果是小数则将其变为double类型

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "100.1" ;	// 要判断的数据
		String regex = "\\d+(\\.\\d+)?" ;  // 正则表达式
		System.out.println(str.matches(regex));
	}
}
```

范例：现在判断一个字符串是否由日期所组成，如果是由日期所组成则将其转为Date类型

```
import java.text.SimpleDateFormat;

public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "1981-20-15" ;  // 要判断的数据
		String regex = "\\d{4}-\\d{2}-\\d{2}" ;  // 正则表达式
		if (str.matches(regex)) {
			System.out.println(new SimpleDateFormat("yyyy-MM-dd").parse(str));
		}
	}
}
```

需要注意的是，正则表达式无法对里面的内容进行判断，只能够对格式进行判断处理。

范例：判断给定的电话号码是否正确?

`电话号码：51283346；

·电话号码：01051283346；

·电话号码：(010)-51283346；

```
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        String str = "51283346" ;       // 要判断的数据
        String regex = "\\d{7,8}" ;     // 正则表达式
        System.out.println(str.matches(regex));
    }
}
```

```
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        String str = "01051283346" ; // 要判断的数据
        String regex = "(\\d{3,4})?\\d{7,8}" ;      // 正则表达式
        System.out.println(str.matches(regex));
    }
}
```

```
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        String str = "(010)-51283346" ;     // 要判断的数据
        String regex = "((\\d{3,4})|(\\(\\d{3,4}\\)-))?\\d{7,8}" ;    // 正则表达式
        System.out.println(str.matches(regex));
    }
}
```

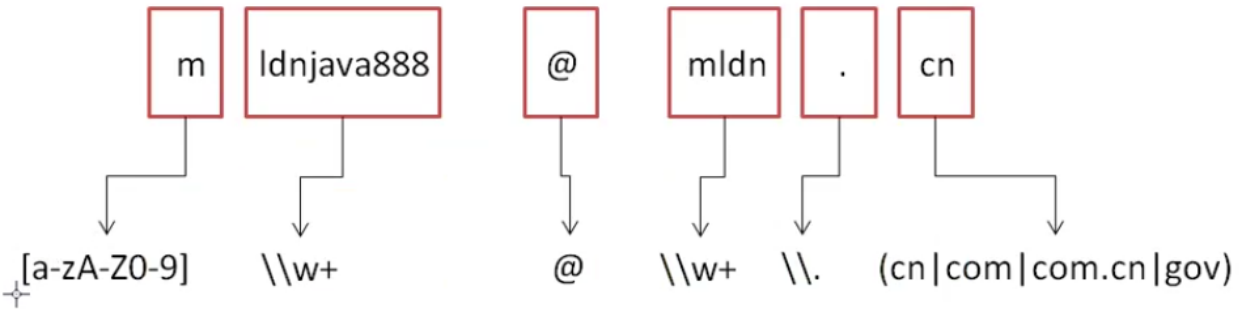
既然已经可以使用正则进行验证了，那么下面就可以利用其来实现一个email地址格式的验证。

范例：验证email格式

·email的用户名可以由字母、数字、_所组成（不应该使用“_”开头）；

·email的域名可以由字母、数字、_、-所组成；

·域名的后缀必须是：.cn、.com、.net、.com.cn、.gov；




```
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        String str = "mldnjava888@mldn.cn" ;  // 要判断的数据
        String regex = "[a-zA-Z0-9]\\w+@\\w+\\.(cn|com|com.cn|net|gov)" ; // 正则表达式
        System.out.println(str.matches(regex));
    }
}
```

现在这几种正则的匹配处理操作是最常用的几种处理形式。

■java.util.regex开发包

虽然在大部分的情况下都可以利用String类实现正则的操作，但是也有一些情况下需要使用到java.util.regex开发包中提供的正则处理类，在这个包里面一共定义有两个类：Pattern（正则表达式编译）、Matcher（匹配）。

1、Pattern类提供有正则表达式的编译处理支持：public static Pattern compile(String regex)；

同时也提供有字符串的拆分操作：public String[] split(CharSequence input)；

```
import java.util.regex.Pattern;

public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "JKL()UI$()QR@#JKLSD()QW#EIO$RJKLOSDF" ;
		String regex = "[^a-zA-Z]+" ;
		Pattern pat = Pattern.compile(regex) ; // 编译正则表达式
		String result [] = pat.split(str) ; // 拆分
		for (int x = 0 ; x < result.length ; x ++) {
			System.out.println(result[x]);
		}
	}
}
```

2、Matcher类，实现了正则匹配的处理类，这个类的对象实例化依靠Pattern类完成：

·Pattern类提供的方法：public Matcher matcher(CharSequence input)；

当获取了Matcher类的对象之后就可以利用该类中的方法进行如下操作：

·正则匹配：public boolean matches()；

·字符串替换：public String replaceAll(String replacement)；

范例：字符串匹配

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

public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "101" ;
		String regex = "\\d+" ;
		Pattern pat = Pattern.compile(regex) ; // 编译正则表达式
		Matcher mat = pat.matcher(str) ;
		System.out.println(mat.matches());
	}
}
```

范例：字符串替换

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

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		String str = "1KLKLKL()8908923892389123890JKLJKHJKL&*()&*()U" ;
		String regex = "\\D+" ;
		Pattern pat = Pattern.compile(regex) ; // 编译正则表达式
		Matcher mat = pat.matcher(str) ;
		System.out.println(mat.replaceAll(""));
	}
}
```

如果纯粹的是以拆分、替换、匹配三种操作为例根本用不到java.util.regex开发包，只依靠String类就都可以实现了。但是Matcher类里面提供有一种分组的功能，而这种分组的功能是String不具备的。

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

public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		// 要求取出“#{内容}”标记中的所有内容
		String str = "INSERT INTO dept(deptno,dname,loc) VALUES (#{deptno},#
{dname},#{loc})" ;
		String regex = "#\\{\\w+\\}" ;
		Pattern pat = Pattern.compile(regex) ; // 编译正则表达式
		Matcher mat = pat.matcher(str) ;
		while(mat.find()) {   // 是否有匹配成功的内容
			System.out.println(mat.group(0).replaceAll("#|\\{|\\}", ""));
		}
	}
}
```

java.util.regex开发包，如果不是进行一些更为复杂的正则处理是很难使用到的，而String类所提供的功能只适合于正则的基本操作。