/*
正则表达式:符合一定规则的表达式。
作用：用于专门操作字符串。
特点：用于一些特定的符号来表示一些代码操作。这样就简化书写。
所以学习正则表达式，就是在学习一些特殊符号的使用。

好处：可以简化对字符串的复杂操作。
弊端：符号定义越多，正则越长，阅读性越差。

具体操作功能：

1，匹配：String matches 方法。用规则匹配整个字符串，只要有一处不符合规则，就匹配结束，返回 false。
\\d

\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]
X?	*X*, once or not at all
*X**	*X*, zero or more times
X+	*X*, one or more times
X{*n*}	*X*, exactly *n* times
X{*n*,}	*X*, at least *n* times
X{*n*,*m*}	*X*, at least *n* but not more than *m* times

2，切割：String split();
用.进行切割 \\.
表示叠词 String regex = " (.)\\1" "(.)\\1+"
splitDemo("erkktyqqquizzzzzo","(.)\\1+");//按照叠词完成切割。为了可以让规则的结果被重用
//可以将规则封装成一个组。用()完成。组的出现都有编号。
//从 1 开始。 想要使用已有的组可以通过 \n(n 就是组的编号)的形式来获取。

3，替换：String replaceAll(regex,str);如果 regex 中有定义组，可以在第二参数中通过$符号获取正则表达式中的已有的组。
*/

String str1 = "erkktyqqquizzzzzo";//将叠词替换成$. //将重叠的字符替换成单个字母。zzzz->z
replaceAllDemo(str1,"(.)\\1+","$1");

/*
正则表达式的第四个功能。
4，获取:将字符串中的符合规则的子串取出。

操作步骤：
1，将正则表达式封装成对象。
2，让正则对象和要操作的字符串相关联。
3，关联后，获取正则匹配引擎。
4，通过引擎对符合规则的子串进行操作，比如取出。
*/

```
String str = "ming tian jiu yao fang jia le ,da jia。";
System.out.println(str);
String reg = "\\b[a-z]{4}\\b";

//将规则封装成对象。
Pattern p = Pattern.compile(reg);

//让正则对象和要作用的字符串相关联。获取匹配器对象。
Matcher m   = p.matcher(str);

//System.out.println(m.matches());//其实 String 类中的 matches 方法。用的就是 Pattern 和 Matcher 对象来完成的。
                                  //只不过被 String 的方法封装后，用起来较为简单。但是功能却单一。


while(m.find())
    {
        System.out.println(m.group());
        System.out.println(m.start()+"...."+m.end());
    }
```

```
/*
需求：
将下列字符串转成：我要学编程.

到底用四种功能中的哪一个呢？或者哪几个呢？
思路方式：
1，如果只想知道该字符是否对是错，使用匹配。
2，想要将已有的字符串变成另一个字符串，替换。
3，想要按照自定的方式将字符串变成多个字符串。切割。获取规则以外的子串。
4，想要拿到符合需求的字符串子串，获取。获取符合规则的子串。
*/


/*
192.68.1.254 102.49.23.013 10.10.10.10 2.2.2.2 8.109.90.30
将 ip 地址进行地址段顺序的排序。

还按照字符串自然顺序，只要让它们每一段都是 3 位即可。
1，按照每一段需要的最多的 0 进行补齐，那么每一段就会至少保证有 3 位。
2，将每一段只保留 3 位。这样，所有的 ip 地址都是每一段 3 位。
*/
public static void ipSort()
{
	String ip = "192.68.1.254 102.49.23.013 10.10.10.10 2.2.2.2 8.109.90.30";

	ip = ip.replaceAll("(\\d+)","00$1");
	System.out.println(ip);

	ip = ip.replaceAll("0*(\\d{3})","$1");
	System.out.println(ip);

	String[] arr = ip.split(" ");

	TreeSet<String> ts = new TreeSet<String>();

	for(String s : arr)
	{
		ts.add(s); }

	for(String s : ts)
	{
		System.out.println(s.replaceAll("0*(\\d+)","$1"));
	}
}
```