1. 正则表达式(RegularExpressions)—**字符串处理利器**
2. 用途：处理字符串字符串
   1. 字符串匹配(字符匹配)："afe".matches("[a-z]{3}");//等价于：

Pattern pattern = Pattern.compile("[a-z]{3}");//编译后效率高

Matcher matcher = pattern.matcher("afe");

* 1. 字符串查找(字符查找) "a8845a".replaceAll("\\d", "-");
  2. 字符串替换(字符替换)：

1. 例如：IP地址是否正确、从网页中揪出email地址、从网页中揪出链接等
2. Java中正则表达式的相关类：java.lang.String、java.util.regx.Pattern、java.util.regex.Matcher
3. 正则表达式语法：(API中的java.util.regex类中有说明)
   1. metaCharacters:
      1. **.** ：任意字符(Any character (may or may not match line terminators))
      2. **\*** ：零个或多个(zero or more times)
      3. **+** ：一个或多个(one or more times)
      4. **?** ： 零个或一个(once or not at all)
      5. **\d** ：一位数字(A digit: [0-9])
      6. **\D** ：非一位数字(A non-digit: [^0-9])
      7. **\s** ：一个空白字符(A whitespace character: [ \t\n\x0B\f\r])
      8. **\S** ：一个非空白字符(A non-whitespace character: [^\s])
      9. **\w** ：构成单词的字符:字母数字下划线(A word character: [a-zA-Z\_0-9])
      10. **\W** ：不是构成单词的字符(A non-word character: [^\w])
      11. **{n}** ：正好n次(exactly n times)
      12. **{n,}** ： 至少n次(at least n times)
      13. **{n,m}** ：至少n次，不超过m次(at least n but not more than m times)
   2. 范围：
      1. **[]** ：匹配其中的一个字符
      2. **^** ：在中括号里面第一个字符表示取反；在中括号外面代表输入的开头(The beginning of a line)
      3. [abc] ：a, b, or c (simple class)
      4. [^abc] ：Any character except a, b, or c (negation)
      5. [a-zA-Z] ：a through z or A through Z, inclusive (range)
      6. [a-d[m-p]] ：a through d, or m through p: [a-dm-p] (union)
      7. [a-z&&[def]] ：d, e, or f (intersection)
      8. [a-z&&[^bc]] ：a through z, except for b and c: [ad-z] (subtraction)
      9. [a-z&&[^m-p]] ：a through z, and not m through p: [a-lq-z](subtraction)
   3. 边界处理
      1. **^** ：在中括号里面第一个字符表示取反；在中括号外面代表输入的开头(The beginning of a line)
      2. $ ：The end of a line
      3. \b ：一个单词的边界(A word boundary)
      4. \B ：A non-word boundary
      5. \A ：The beginning of the input
      6. \G ：The end of the previous match
      7. \Z ：The end of the input but for the final terminator, if any
      8. \z ：The end of the input
4. Note：java中的字符串中的\是转义字符，正则表达式中的\是语法，若想在正则表达式中匹配一个\，则需要在字符串中写四个\。例："\\".matches("\\\\");
5. matches find lookingAt

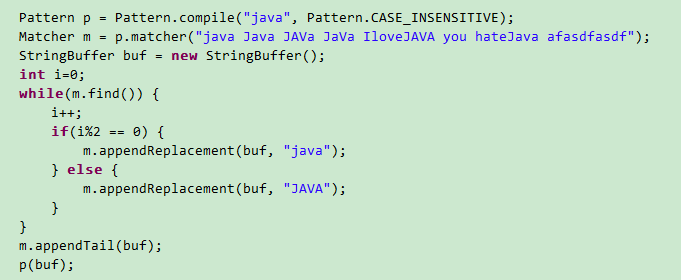
Pattern p = Pattern.compile("\\d{3,5}");

String s = "123-34345-234-00";

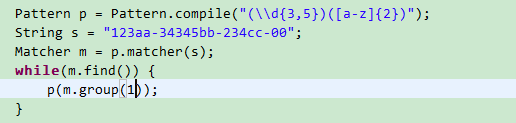
Matcher m = p.matcher(s);

* 1. matches：匹配整个字符串：m.matches()//false
  2. find：寻找子字符串(Attempts to find the next subsequence of the input sequence that matches the pattern.)：m.find();//true。Find、matches后Matcherm会移动到匹配结束的地方，可以使用m.reset()方法重置(即从开头匹配)
  3. lookingAt：每次都从开头寻找子串(Attempts to match the input sequence, starting at the beginning of the region, against the pattern.)：m.lookingAt();//true

1. start end(必须能够找到子串才能调用)
   1. m.start()//匹配到子串的开始位置
   2. m.end()//匹配到子串的结束位置(匹配到字符位置的下一个位置)
2. 例子：

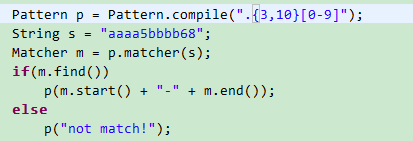


1. 分组：()表达式

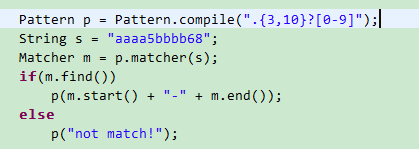


组号**从1开始**，以**左小括号**为准

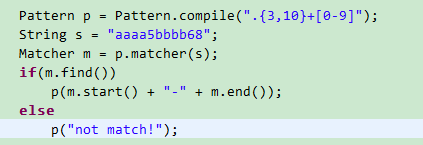
1. Reluctant Quantifiers(不情愿的，后面多加一个？)
2. Possessive Quantifiers(独占性的，后面多加一个+)
3. Greedy Quantifiers(**默认都是贪婪的**)：尽量多匹配
4. 例：



贪婪的(默认)：先“吞进去”10个，发现不匹配，然后吐出来一个，发现匹配了，结束，打印：0-10



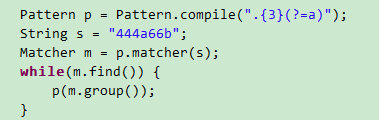
不情愿的(多加一个?)：先“吞进去”3个，发现不匹配，再吞进去1个，发现匹配，结束，打印：0-5



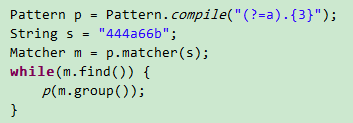
独占性的(多加一个+)：与贪婪一样，但是不往外吐，所以匹配不上，打印：not match

1. Special constructs (named-capturing and **non-capturing**)：()里面以?打头的话，不是用来匹配字符串的(非捕获组)。

查阅api文档：(?=X) X, via zero-width positive lookahead

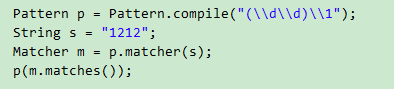


打印：444

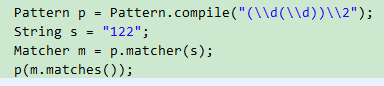


打印：a66

1. Back references
   1. \n Whatever the nth capturing group matched
   2. \k<name> Whatever the named-capturing group "name" matched



\1代表第一个组捕捉之后的那个字符串，打印：true



\2代表第二个组捕捉之后的那个字符串，打印：true

1. flags的简写

Pattern p = Pattern.compile("java", Pattern.CASE\_INSENSITIVE);

等价于：

p("Java".matches("(?i)(java)"));