# 一．断点调试

* 可以让程序员停留在指定的地方，然后去观察目前程序的数据，然后分析错误的原因及所在。

1. Step over ： 跳过本行代码（F6）
2. Step into ： 进入方法内部（F5）
3. Step return ： 结束方法，返回数据

# 二、批量修改变量名

* 选中变量名，右键-refactor-rename

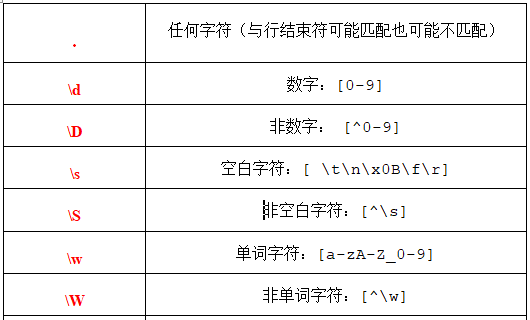
# 三、正则表达式

* 正则表达式其实就是一种规则，有自己特殊的应用，其作用就是针对字符串进行操作
* 正则：就是用于操作字符串的一种规则，其中这些规则使用了一些字符表示
* String类用于判断正则表达式是否匹配的方法：
  + **boolean matches(String regex)**:告诉这个字符串是否匹配指定的正则表达式(regular expression)

## 1. 正则表达式的规则

* **注意：任何预定义字符在没有加上数量词之前，都只能匹配一个字符而已。**

## 2. 预定义字符



## 3.正则表达式 预定义字符 案例：

|  |
| --- |
| // . 表示任意字符 System.**out**.println(**". 匹配任意字符？"** + (**"a"**.matches(**"."**)));  // \d 表示数字0-9其中一位 System.**out**.println(**"\\d 匹配任意数字：(匹配一位数字[0-9])?"** + (**"3"**.matches(**"\\d"**)));  // \D 表示非数字,即非\d [^0-9]，只要不是数字0-9即返回true System.**out**.println(**"\\D 匹配任意非数字字符：(匹配一位数字[0-9])?"** + (**"a"**.matches(**"\\D"**)));  // \s 表示空白字符 [ \t \n \x0B \f \r ] System.**out**.println(**"\\s 匹配任意空白字符?"** + (**"\t"**.matches(**"\\s"**)));  // \S 表示非空表字符 [^\s] System.**out**.println(**"\\S 匹配任意非空白字符?"** + (**"s"**.matches(**"\\S"**)));  // \w 表示单词字符：[a-z A-Z \_ 0-9]  System.**out**.println(**"\\w 匹配任意单词字符?"** + (**"="**.matches(**"\\w"**)));  // \W 表示非单词字符：[^\w] System.**out**.println(**"\\W 匹配任意非单词字符：?"** + (**"3"**.matches(**"\\W"**))); |
| **结果：**  . 匹配任意字符？true  \d 匹配任意数字：(匹配一位数字[0-9])?true  \D 匹配任意非数字字符：(匹配一位数字[0-9])?true  \s 匹配任意空白字符?true  \S 匹配任意非空白字符?true  \w 匹配任意单词字符?false  \W 匹配任意非单词字符：?false |

正则表达式：

1. 正则表达式其实就是一种规则，有自己特殊的应用，其作用就是针对字符串进行操作

 2. 正则：就是用于操作字符串的一种规则，其中这些规则使用了一些字符表示

 **注意：任何预定义字符在没有加上数量词之前，都只能匹配一个字符而已。**

预定义字符类:

. 任何字符（与行结束符可能匹配也可能不匹配）

\d 数字：[0-9]

\D 非数字： [^0-9]

\s 空白字符：[ \t \n \x0B \f \r]

\S 非空白字符：[^\s]

\w 单词字符：[a-z A-Z \_ 0-9]

\W 非单词字符：[^\w]

Gready 数量词：

X **?** 一次或一次也没有（？表示 最多出现一次X字符）

X **\*** 零次或多次（\* 表示 任意次数的X字符）

X **+** 一次或多次（+ 表示 最少出现一次X字符）

X **{n}** 恰好n次（出现n次X字符）

X **{n,}** 至少n次（最多出现n次X字符）

X **{n,m}** 至少n次，但是不超过m次（出现n<=x<=m 次X字符）

范围表示：

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a 到 z 或 A 到 Z，两头的字母包括在内（范围）

[a-d[m-p]] a 到 d 或 m 到 p：[a-dm-p]（并集）

[a-z&&[def]] d、e 或 f（交集）

[a-z&&[^bc]] a 到 z，除了 b 和 c：[ad-z]（减去）

[a-z&&[^m-p]] a 到 z，而非 m 到 p：[a-lq-z]（减去）

*注意：范围词里面，不管内容有多长，没有数量词的配合，只能匹配其中的一个字符而已。*

## 4、正则 手机号与邮箱

1、查找需要使用的对象：

Pattern：正则对象

Matcher：匹配器

2、正则表达式的编译表示形式。

指定为字符串的正则表达式必须首先被编译为此类的实例。然后，可将得到的模式用于创建 Matcher 对象，依照正则表达式，该对象可以与任意字符序列匹配。执行匹配所涉及的所有状态都驻留在匹配器中，所以多个匹配器可以共享同一模式。

因此，典型的调用顺序是

**Pattern p = Pattern.compile("a\*b");**

**Matcher m = p.matcher("aaaaab");**

**boolean b = m.matches();**

3、匹配器要使用的方法

* find():通知匹配器去匹配字符串，查找符合规则的子串。如果能查找到符合规则的子字符串，返回true，否则返回false
* group():获取符合规则的子串
* 注意：使用group方法之前，一定要使用find方法去查找符合规则的字符串，否则报错。

4.1 案例

|  |
| --- |
| *// 需求：在字符串中获取由三个字母组成的单词* **public static void** getWord() {  String str = **"da jia hao , ming tian bu fang jia , xie xie ！"**;  String reg = **"\\b[a-zA-Z]{3}\\b"**;   *// 首先要把字符串的正则编译成Pattern正则对象，\b表示单词的边界* Pattern pattern = Pattern.*compile*(reg);   *// 使用正则对象匹配字符串，用于产生一个Matcher 匹配器对象* Matcher matcher = pattern.matcher(str);   *// matcher.find():判断字符串中是否存在符合正则表达式的内容  // matcher.group():获取符合正则规则的字符串内容* **while** (matcher.find()) {  System.***out***.println(matcher.group());  } |

## 5、正则对象

### 1、查找需要使用的对象：

Pattern：正则对象

Matcher：匹配器

### 2、正则表达式的编译表示形式。

指定为字符串的正则表达式必须首先被编译为此类的实例。然后，可将得到的模式用于创建 Matcher 对象，

依照正则表达式，该对象可以与任意字符序列匹配。执行匹配所涉及的所有状态都驻留在匹配器中，所以多个匹配器可以共享同一模式。

因此，典型的调用顺序是

Pattern p = Pattern.compile("a\*b");

Matcher m = p.matcher("aaaaab");

boolean b = m.matches();

### 3、匹配器要使用的方法

find():通知匹配器去匹配字符串，查找符合规则的子串。如果能查找到符合规则的子字符串，返回true，否则返回false

group():获取符合规则的子串

**注意：使用group方法之前，一定要使用find方法去查找符合规则的字符串，否则报错。**

|  |
| --- |
| *// 需求：在字符串中获取由三个字母组成的单词* **public static void** getWord() {  String str = **"da jia hao , ming tian bu fang jia , xie xie ！"**;  String reg = **"\\b[a-zA-Z]{3}\\b"**;   *// 首先要把字符串的正则编译成Pattern正则对象，\b表示单词的边界* Pattern pattern = Pattern.*compile*(reg);   *// 使用正则对象匹配字符串，用于产生一个Matcher 匹配器对象* Matcher matcher = pattern.matcher(str);   *// matcher.find():判断字符串中是否存在符合正则表达式的内容  // matcher.group():获取符合正则规则的字符串内容* **while** (matcher.find()) {  System.***out***.println(matcher.group());  } } |