**WEB编程基础**

**正则表达式**

**尚学堂内部教材 禁止外传**

[正则表达式基础(Regular Expression) 2](#_Toc422857534)

[正则表达式简介 2](#_Toc422857535)

[工具软件RegexBuddy的使用 2](#_Toc422857536)

[正则表达式规则 3](#_Toc422857537)

[正则表达式的匹配模式 5](#_Toc422857538)

[开发中使用正则表达式的流程 5](#_Toc422857539)

[课堂练习 6](#_Toc422857540)

[开发软件中正则表达式的使用 6](#_Toc422857541)

[JAVA程序中使用正则表达式 7](#_Toc422857542)

[JAVASCRIPT中使用正则表达式 10](#_Toc422857543)

[针对表单域的验证，封装一个通用的函数 11](#_Toc422857544)

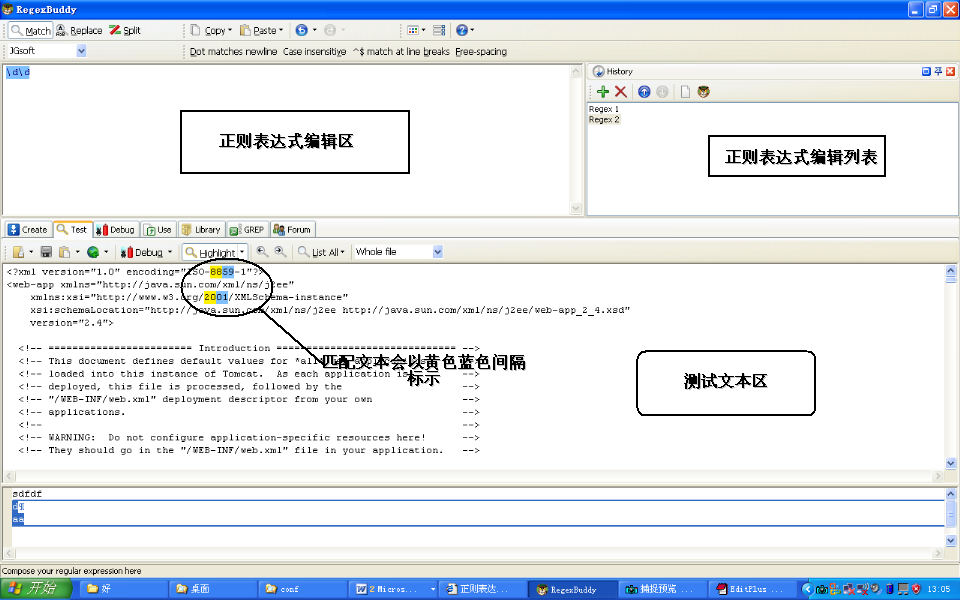
# 正则表达式基础(Regular Expression)

## 正则表达式简介

* 为什么需要正则表达式？
  + **文本的复杂处理。**
* 正则表达式的优势和用途？
  + 一种强大而灵活的文本处理工具；
  + 提供了一种紧凑的、动态的方式，能够以一种**完全通用的方式**来解决各种字符串处理（例如：**验证、查找、替换,分割**等）问题；
  + 大部分语言、数据库都支持正则表达式。
* 正则表达式定义：
  + 正如他的名字一样是描述了一个**匹配字符的规则，**通过这个规则可以匹配一类字符串。
* 正则表达式的用处：
  + **验证**给定字符串是否符合指定特征，比如验证是否是合法的邮件地址。
  + 用来**查找**字符串，从一个长的文本中查找符合指定特征的字符串。
  + 用来**替换**，比普通的替换更强大
  + **分割**

## 工具软件RegexBuddy的使用

* 为了提高开发效率，一般都先在工具软件中测试正则表达式，通过测试后，才在程序中使用。



## 正则表达式规则

* 普通字符
  + 字母、数字、汉字、下划线、以及没有特殊定义的标点符号，都是**“普通字符”**。表达式中的普通字符，在匹配一个字符串的时候，**匹配与之相同的一个字符**。
* 简单的转义字符

|  |  |
| --- | --- |
| \n | 代表换行符 |
| \t | 制表符 |
| \\ | 代表\本身 |
| \^ ，\$,\.，\(， \) ， \{， \} ， \? ， \+ ， \\* ， \| ，\[， \] | 匹配这些字符本身 |
| \u四位16进制整数 | 匹配unicode的字符 |

* **标准字符集合**：能够与 ‘多种字符’ 匹配的表达式
  + （**注意区分大小写，大写是相反的意思**）

|  |  |
| --- | --- |
| \d | 任意一个数字，0~9 中的**任意一个** |
| \w | 任意一个字母**或**数字**或下划线**，也就是 **A~Z,a~z,0~9,\_ 中任意一个** |
| \s | 包括空格、制表符、换行符等**空白字符**的其中任意一个 |
| **.** | 小数点可以匹配**除了换行符（\n）以外**的任意一个字符 |

* **自定义字符集合**：[ ]方括号匹配方式，能够匹配方括号中**任意一个**字符

|  |  |
| --- | --- |
| [ab5@] | 匹配 "a" 或 "b" 或 "5" 或 "@" |
| [^abc] | 匹配 "a","b","c" 之外的任意一个字符 |
| [f-k] | 匹配 "f"~"k" 之间的任意一个字母 |
| [^A-F0-3] | 匹配 "A"~"F","0"~"3" 之外的任意一个字符 |
| [\s\S][\d\D][\w\W] | 匹配任意字符 |
| [\u4E00-\u9FA5] | 匹配任意中文 |

**注意事项：**

1. 正则表达式中的特殊符号，如果被包含于中括号中，则失去特殊意义，**但 \ [ ] ^ - 除外**。

2. **标准字符集合，除小数点(.)外，**如果被包含于中括号中，自定义字符集合将包含该集合。  
比如：**[\d.\-+]**，将可以匹配**数字**，小数点和 + - 符号。**（小数点和 + 号失去特殊意义）**

* 修饰匹配次数的特殊符号

字符{n} 字符匹配 n次

|  |  |
| --- | --- |
| 字符{m,n} | 字符**至少**重复m次，**最多**重复n次 |
| 字符{m,} | 字符至少重复m次 |
| 字符? | 匹配字符0次或者1次，相当于 {0,1} |
| 字符+ | 字符至少出现1次，相当于 {1,} |
| 字符\* | 字符不出现或出现任意次，相当于 {0,} |

* 匹配次数中的**贪婪模式(匹配字符越多越好)**
  + “{m,n}”, “{m,}”, “?”, “\*”, “+”，具体匹配的次数随被匹配的字符串而定。这种重复匹配不定次数的表达式在匹配过程中，总是尽可能多的匹配。
* 匹配次数中的**非贪婪模式（匹配字符越少越好）**
  + **在修饰匹配次数的特殊符号后再加上一个 "?" 号**，则可以使匹配次数不定的表达式尽可能少的匹配，使可匹配可不匹配的表达式，尽可能的 "不匹配"。
* **字符边界**(本组标记匹配的不是字符而是位置，符合某种条件的位置)

|  |  |
| --- | --- |
| ^字符 | 以某一个字符开始 |
| 字符$ | 以某一个字符结束 |

* 选择符和分组

|  |  |
| --- | --- |
| 表达式 | 作用 |
| | | 左右两边表达式之间 "或" 关系，匹配左边或者右边 |
| ( ) | (1). 在被修饰匹配次数的时候，**括号中的表达式可以作为整体**被修饰  (2). 取匹配结果的时候，括号中的表达式匹配到的内容可以被单独得到  (3). 每一对括号会分配一个编号，使用 () 的捕获根据**左括号的顺序从 1 开始自动编号**。捕获元素编号为0的第一个捕获是由整个正则表达式模式匹配的文本 |

* **反向引用（\nnn）**
  + 每一对()会分配一个编号，使用 () 的捕获**根据左括号的顺序从 1 开始自动编号**。
  + 通过反向引用，可以对**分组已捕获的字符串**进行引用。
* 非捕获组(**?:**xxx)
  + 与捕获组的区别在于不捕获匹配的文本，仅仅作为分组。**其他地方一致。**
* **预搜索(零宽断言)** 
  + 判断当前位置的前后字符，是否符合指定的条件，但不匹配前后的字符。**是对位置的匹配。**

|  |  |
| --- | --- |
| (?=exp) | 断言自身出现的**位置的后面能**匹配表达式exp |
| (?<=exp) | 断言自身出现的**位置的前面能**匹配表达式exp |
| (?!exp) | 断言此**位置的后面不能**匹配表达式exp |
| (?<!exp) | 断言此**位置的前面不能**匹配表达式exp |

## 正则表达式的匹配模式

* IGNORECASE 忽略大小写模式
  + 匹配时忽略大小写。
  + 默认情况下，正则表达式是要区分大小写的。
* SINGLELINE 单行模式
  + 整个文本看作一个字符串，只有一个开头，一个结尾。
  + 使小数点 "." 可以匹配包含换行符（\n）在内的任意字符。
* MULTILINE 多行模式
  + 每行都是一个字符串，都有开头和结尾。
  + 在指定了 MULTILINE 之后，如果需要仅匹配字符串开始和结束位置，可以使用 \A 和 \Z

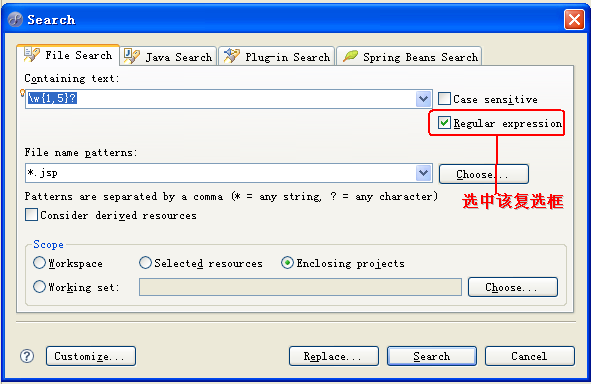
## **如果你在正则表达式内部插入修饰符(?ism)，则该修饰符只对其右边的正则表达式起作用。(?-i)是关闭大小写不敏感。你可以很快的进行测试。<<(?i)te(?-i)st>>应该匹配TEst，但是不能匹配teST或TEST.**

## 开发中使用正则表达式的流程

1. **分析所要匹配的数据，写出测试用的典型数据**
2. **在工具软件中进行匹配测试**
3. **在程序中调用通过测试的正则表达式**

## 开发软件中正则表达式的使用

* 文本编辑器editplus, ultraedit中使用正则表达式
  + **很多文本编辑器中并不一定支持所有正则表达式规则，使用前最好查看该软件相关文档**
* **Eclipse中使用正则表达式**



# JAVA程序中使用正则表达式

* 相关类位于：**java.util.regex**包下面
* **类 Pattern**：
  + 正则表达式的编译表示形式。
  + Pattern p = Pattern.*compile(r,int)*; //建立正则表达式，并启用相应模式
* **类 Matcher**：
  + 通过解释 [Pattern](mk:@MSITStore:F:\\306\\JDK_API6.0.CHM::/java/util/regex/Pattern.html" \t "_parent" \o "java.util.regex 中的类) 对 [character sequence](mk:@MSITStore:F:\\306\\JDK_API6.0.CHM::/java/lang/CharSequence.html" \t "_parent" \o "java.lang 中的接口) 执行匹配操作的引擎
  + Matcher m = p.matcher(str); //匹配str字符串
* 编程中使用正则表达式常见情况：
  + 验证表达式是否匹配整个字符串
  + 验证表达式是否可以匹配字符串的子字符串
  + 返回给定字符串中匹配给定正则表达式所有子字符串
  + 替换给定字符串中匹配正则表达式的子字符串
  + 根据正则表达式定义规则截取字符串

|  |
| --- |
| package test;  import java.util.ArrayList;  import java.util.List;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  public class TestReg {    /\*\*  \* 验证传入的字符串是否整个匹配正表达式  \* @param regex: 正则表达式  \* @param decStr：要匹配的字符串  \* @return ：若匹配，则返回true;否则，返回false;  \*/  public static boolean validate(String regex, String decStr) {  // 表达式对象  Pattern p = Pattern.compile(regex,Pattern.CASE\_INSENSITIVE);  // 创建 Matcher 对象  Matcher m = p.matcher(decStr);  // 是否完全匹配  boolean yesorno = m.matches(); //该方法尝试将整个输入序列与该模式匹配  return yesorno;  }    /\*\*  \* 验证传入的字符串是否有子字符串匹配正表达式  \* @param regex: 正则表达式  \* @param decStr：要匹配的字符串  \* @return ：若匹配，则返回true;否则，返回false;  \*/  public static boolean validate2(String regex, String decStr) {  // 表达式对象  Pattern p = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);  // 创建 Matcher 对象  Matcher m = p.matcher(decStr);  // 是否完全匹配  boolean yesorno = m.find(); //该方法扫描输入序列以查找与该模式匹配的下一个子序列。  return yesorno;  }    /\*\*  \* 给定字符串中是否有符合给定正则表达式的子字符串，返回匹配的第一个子字符串  \* @param regex：正则表达式  \* @param decStr：要匹配的字符串  \* @return :返回匹配的第一个字符串，若不匹配则null  \*/  public static String search(String regex, String decStr) {  Pattern p = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);  Matcher m = p.matcher(decStr);  // 是否找到匹配  boolean found = m.find(); //该方法扫描输入序列以查找与该模式匹配的下一个子序列。  String foundstring="";  if(found){  foundstring = m.group();  String foundstring0 = m.group(0); //group(),group(0)返回符合整个表达式的子字符串  String foundstring1 = m.group(1); //group(1)返回符合整个表达式的子字符串中匹配第一个表达式的子字符串  String foundstring2 = m.group(2); //group(2)返回符合整个表达式的子字符串中匹配第一个表达式的子字符串  // String foundstring3 = m.group(3);  System.out.println("foundstring:"+foundstring);  System.out.println("foundstring0:"+foundstring0);  System.out.println("foundstring1:"+foundstring1);  System.out.println("foundstring2:"+foundstring2);  // System.out.println("foundstring3:"+foundstring3);  }  return foundstring;  }    /\*\*  \* 返回给定字符串中匹配给定正则表达式所有子字符串  \* @param regex  \* @param decStr  \* @return List：返回所有匹配正则表达式的子字符串  \*/  public static List searchSubStr(String regex,String decStr) {  Pattern p = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);  Matcher m = p.matcher(decStr);  List<String> list = new ArrayList<String>();  while(m.find()){ //12a345b666cwer  list.add(m.group());  }  for (String string : list) {  System.out.println(string);  }  return list;  }    /\*\*  \* 替换给定字符串中匹配正则表达式的子字符串  \* @param regex：正则表达式  \* @param decStr：所要匹配的字符串  \* @param replaceStr：将符合正则表达式的子串替换为该字符串  \* @return：返回替换以后新的字符串  \*/  public static String replace(String regex,String decStr,String replaceStr) {  Pattern p = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);  Matcher m = p.matcher(decStr);  // 替换  String newstring = m.replaceAll(replaceStr);  System.out.println(newstring);  return newstring;  }    public static void testSplit() {  String str = "abc5Adefghi7Ajklmn";  // 分割  String [] strs = str.split("(\\d)A");  for(int i=0;i<strs.length;i++) {  System.out.println(strs[i]);  }  }  public static void main(String[] args) {  replace("\\d", "dsfd;sa;ksd12a34b567c890d888e999f","\*");  // System.out.println(validate("\\d+", "334455aaa"));  // System.out.println(validate2("\\d+", "334455aaa"));  // System.out.println(searchSubStr("(\\d+)([a-z]+)", "334455aaa--3232423aaa-32324bbb"));  testSplit();  }  } |

# JAVASCRIPT中使用正则表达式

* Javascript中**正则表达式对象**的创建
  + **re = /pattern/[“flags”] 🡪 var** re1 = /\d{3}/ig;
  + **re = new RegExp("pattern",["flags"]) 🡪 var** re2 = **new** RegExp("\\d{3}","ig");
  + Flags可选项常见值：
    - **g** **（全文查找出现的所有pattern）**
    - **i （忽略大小写）**
* **RegExp对象常用方法：**
  + exec()**:** 返回的是一个数组。该数组包含了匹配该模式的第一个子字符串以及该子字符串中匹配相关分组的字符串。比如：

**var** re = **new** RegExp("(\\d+)([a-z]+)","ig");

**var** result = re.exec("33ff55tt77uu88yy");

返回的数组为：**[33ff,33,ff]**

* + **test():** **返回一个 Boolean 值，它指出在被查找的字符串中是否存在模式匹配的子字符串**
* **字符串中常用的使用正则表达式的方式：**
  + **match():使用正则表达式模式对字符串执行查找，并将符合该模式的所有子字符串包含到数组中返回。**

**var** re = **new** RegExp("(\\d+)([a-z]+)","i**g**");

**var** t = "33ff55tt77uu88yy".match(re);

**结果为数组：[33ff,55tt,77uu,88yy]**

* + search(): **返回与正则表达式查找内容匹配的第一个子字符串的位置**
  + **split(regex)：**按照指定的模式切割字符串，返回一个数组。

**var** t = "asdfd33ff55tt77uu88yy".split(/\d+/);

* + **replace()**：

**var** t = "asdfd33ff55tt77uu88yy".replace(/\d+/g,"\*\*");

# 针对表单域的验证，封装一个通用的函数

1. 分析用户名和邮箱验证的共同点：
   1. 错误提示文字都会放到某一个元素中。
   2. 都不能为空
2. 分析用户名和邮箱验证的不同点：
   1. 不为空时，还需要有其他的格式验证。（用户名：长度不能超过30. 邮箱：符合邮箱规格）

**封装成的函数：**

|  |
| --- |
| <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=GB18030" />  <title>Untitled Document</title>  <script>  **function** check(frm){  /\*  var uname = frm.userName.value.replace(/(^\s+)|(\s+$)/g,"");  var flag = true;  document.getElementById("unameMsg").innerHTML="";  document.getElementById("pwdMsg").innerHTML="";  document.getElementById("ageMsg").innerHTML="";  if(uname.length==0){  document.getElementById("unameMsg").innerHTML="用户名不能为空！";  flag = false;  }else{  if(!(/^\w+$/.test(uname))){  document.getElementById("unameMsg").innerHTML="用户名必须是：字母、数字、下划线！";  flag = false;  }  }    var pwd = frm.pwd.value.replace(/(^\s+)|(\s+$)/g,"");  if(pwd.length==0){  document.getElementById("pwdMsg").innerHTML="pwd不能为空！";  flag = false;  }else{  if(!(/^\w+$/.test(pwd))){  document.getElementById("pwdMsg").innerHTML="pwd必须是：字母、数字、下划线！";  flag = false;  }  }    var age = frm.age.value.replace(/(^\s+)|(\s+$)/g,"");  if(age.length==0){  document.getElementById("ageMsg").innerHTML="age不能为空！";  flag = false;  }else{  if(!(/^\d{3}$/.test(age))){  document.getElementById("ageMsg").innerHTML="年龄不能超过3位数或者不是数字";  flag = false;  }  }    return flag;  \*/  **var** unameFlag = checkField(frm.userName,$("unameMsg"),/^\w+$/,"用户名不能为空","用户名必须为字母、数字、下划线组成！");  **var** pwdFlag = checkField(frm.pwd,$("pwdMsg"),/^\w{6,16}$/,"pwd不能为空","用户名必须为字母、数字、下划线组成，并且长度在6-16位之间！");  **var** ageFlag = checkField(frm.age,$("ageMsg"),/^((\d{1,2})|(1[0123]\d))$/,"age不能为空","年龄不能高于139！");  **return** unameFlag&&pwdFlag&&ageFlag;  }    **function** $(a){  **return** document.getElementById(a);  }      **function** checkField(fieldObj,msgObj,re,nullMsg,errorMsg){  msgObj.innerHTML = "";  **var** v = fieldObj.value.replace(/(^\s+)|(\s+$)/g,"");  **var** flag = true;  **if**(v.length==0){  msgObj.innerHTML=nullMsg;  flag = false;  }**else**{  **if**(!(re.test(v))){  msgObj.innerHTML=errorMsg;  flag = false;  }  }    **return** flag;  }    </script>  </head>  <body>  <form action="formValidate.htm" name=testFrm onsubmit="**return** check(**this**);" >  用户名：<input type=text name=userName /> <span id=unameMsg ></span> <br>  密码：<input type=password name=pwd /> <span id=pwdMsg></span> <br>  年龄：<input type=text name=age /> <span id=ageMsg></span> <br>  <input type=submit value=注册 />  </form>  </body>  </html> |