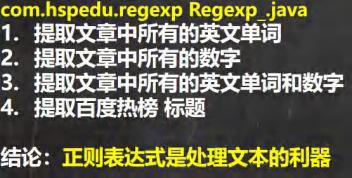
第 27 章正则表达式

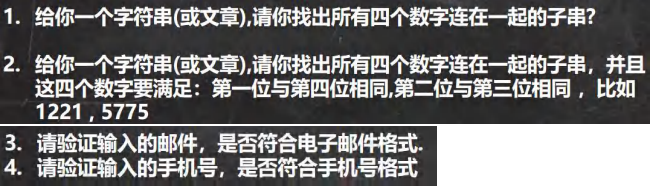
27. 1 为什么要学习正则表达式

27.1.1 极速体验正则表达式威力

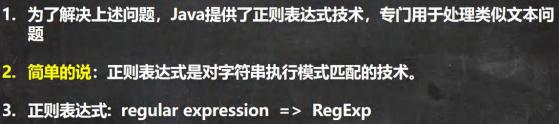


|  |
| --- |
| //1. 先创建一个 Pattern对象 ， 模式对象, 可以理解成就是一个正则表达式对象  //Pattern pattern = Pattern.compile("[a-zA-Z]+");  //Pattern pattern = Pattern.compile("[0-9]+");  //Pattern pattern = Pattern.compile("([0-9]+)|([a-zA-Z]+)");  //Pattern pattern = Pattern.compile("<a target=\"\_blank\" title=\"(\\S\*)\""); |

27.2 再提出几个问题?

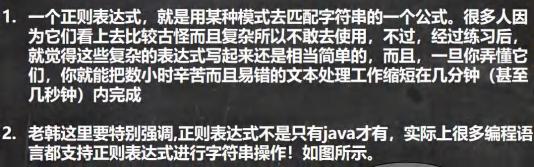


27.3 解决之道-正则表达式



27.4 正则表达式基本介绍

27.4.1 介绍



27.5 正则表达式底层实现(重要)

27.5.1 实例分析

为让大家对正则表达式底层实现有一个直观的映象，给大家举个实例

给你一段字符串(文本),请找出所有四个数字连在一起的子串， 比如:

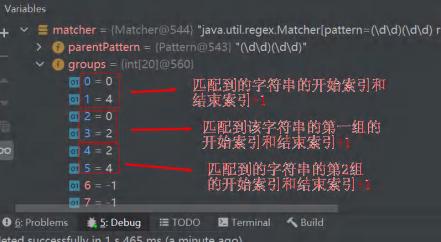
应该找到 1998 1999 3443 9889 ===> 分析底层实现 RegTheory.java

|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0 |

|  |
| --- |
| \* 分析java 的正则表达式的底层实现(重要.)  \*/  public class RegTheory {  public static void main(String[] args) {  String content = " 1998 年 12 月 8 日，第二代 Java 平台的企业版 J2EE 发布。1999 年 6 月，Sun 公司发布了"  +  "第二代 Java 平台（简称为 Java2）的 3 个版本：J2ME（Java2 Micro Edition，Java2 平台的微型" +  "版），应用于移动、无线及有限资源的环境；J2SE（Java 2 Standard Edition ，Java 2 平台的" +  "标准版），应用于桌面环境；J2EE（Java 2Enterprise Edition ，Java 2 平台的企业版），应" + "用 3443 于基于 Java 的应用服务器。Java 2 平台的发布，是 Java 发展过程中最重要的一个" +  "里程碑，标志着 Java 的应用开始普及 9889 ";  //目标：匹配所有四个数字  //说明  //1. \\d 表示一个任意的数字  String regStr = "(\\d\\d)(\\d\\d)";  //2. 创建模式对象[即正则表达式对象]  Pattern pattern = Pattern.compile(regStr);  //3. 创建匹配器  //说明：创建匹配器 matcher ， 按照 正则表达式的规则 去匹配 content 字符串  Matcher matcher = pattern.matcher(content);  //4.开始匹配  /\*\*  \*  \* matcher.find() 完成的任务 （考虑分组） |

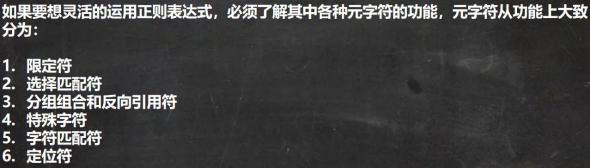
|  |
| --- |
| \* 什么是分组，比如 (\d\d)(\d\d) ,正则表达式中有() 表示分组,第 1 个()表示第 1 组,第 2 个()表示第 2 组...  \* 1. 根据指定的规则 ,定位满足规则的子字符串(比如(19)(98))  \* 2. 找到后，将 子字符串的开始的索引记录到 matcher 对象的属性 int[] groups;  \* 2. 1 groups[0] = 0 , 把该子字符串的结束的索引+1 的值记录到 groups[1] = [4](#bookmark1200)  \* 2.2 记录 1 组()匹配到的字符串 groups[2] = 0 groups[3] = [2](#bookmark1201)  \* 2.3 记录 2 组()匹配到的字符串 groups[4] = 2 groups[5] = [4](#bookmark1202)  \* 2.4.如果有更多的分组.....  \* 3. 同时记录 oldLast 的值为 子字符串的结束的 索引+1 的值即 35, 即下次执行 find 时，就从 35 开始匹  配  \*  \* matcher.group(0) 分析  \*  \* 源码:  \* public String group(int group) {  \* if (first < 0)  \* throw new IllegalStateException("No match found");  \* if (group < 0 || group > groupCount())  \* throw new IndexOutOfBoundsException("No group " + group);  \* if ((groups[group\*2] == - 1) || (groups[group\*2+1] == - 1))  \* return null;  \* return getSubSequence(groups[group \* 2], groups[group \* 2 + 1]).toString();  \* }  \* 1. 根据 groups[0]=31 和 groups[1]=35 的记录的位置，从 content 开始截取子字符串返回  \* 就是 [31,35) 包含 31 但是不包含索引为 35 的位置  \*  \* 如果再次指向 find 方法.仍然安上面分析来执行 |

|  |
| --- |
| \*/  while (matcher.find()) {  //小结  //1. 如果正则表达式有() 即分组  //2. 取出匹配的字符串规则如下  //3. group(0) 表示匹配到的子字符串  //4. group(1) 表示匹配到的子字符串的第一组字串  //5. group(2) 表示匹配到的子字符串的第 2 组字串  //6. ... 但是分组的数不能越界.  System.out.println("找到: " + matcher.group(0));  System.out.println("第 1 组()匹配到的值=" + matcher.group(1));  System.out.println("第 2 组()匹配到的值=" + matcher.group(2));  }  }  } |

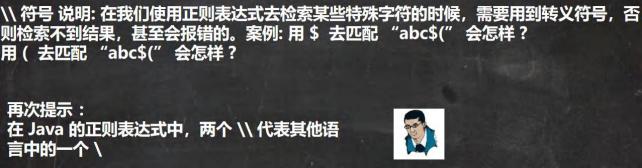


27.6 正则表达式语法

27.6.1 基本介绍



27.6.2 元字符(Metacharacter)-转义号 \\



|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示转义字符的使用  \*/ |

|  |
| --- |
| public class RegExp02 {  public static void main(String[] args) {  String content = "abc$(a.bc(123( )";  //匹配( => \\(  //匹配. => \\.  //String regStr = "\\.";  //String regStr = "\\d\\d\\d";  String regStr = "\\d{3}";  Pattern pattern = Pattern.compile(regStr);  Matcher matcher = pattern.matcher(content);  while (matcher.find()) {  System.out.println("找到 " + matcher.group(0));  }  }  } |



27.6.3 元字符-字符匹配符





|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示字符匹配符 的使用  \*/  public class RegExp03 {  public static void main(String[] args) {  String content = "a11c8abc \_ABCy @";  //String regStr = "[a-z]";//匹配 a-z 之间任意一个字符  //String regStr = "[A-Z]";//匹配 A-Z 之间任意一个字符  //String regStr = "abc";//匹配 abc 字符串[默认区分大小写]  //String regStr = "(?i)abc";//匹配 abc 字符串[不区分大小写] |

|  |
| --- |
| //String regStr = "[0-9]";//匹配 0-9 之间任意一个字符  //String regStr = "[^a-z]";//匹配 不在 a-z 之间任意一个字符  //String regStr = "[^0-9]";//匹配 不在 0-9 之间任意一个字符  //String regStr = "[abcd]";//匹配 在 abcd 中任意一个字符  //String regStr = "\\D";//匹配 不在 0-9 的任意一个字符  //String regStr = "\\w";//匹配 大小写英文字母, 数字，下划线  //String regStr = "\\W";//匹配 等价于 [^a-zA-Z0-9\_]  //\\s 匹配任何空白字符(空格,制表符等)  //String regStr = "\\s";  //\\S 匹配任何非空白字符 ,和\\s 刚好相反  //String regStr = "\\S";  //. 匹配出 \n 之外的所有字符,如果要匹配.本身则需要使用 \\.  String regStr = ".";  //说明  //1. 当创建 Pattern 对象时，指定 Pattern.CASE\_INSENSITIVE, 表示匹配是不区分字母大小写.  Pattern pattern = Pattern.compile(regStr/\*, Pattern.CASE\_INSENSITIVE\*/);  Matcher matcher = pattern.matcher(content);  while (matcher.find()) {  System.out.println("找到 " + matcher.group(0));  }  }  } |

27.6.4 元字符-选择匹配符



|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 选择匹配符  \*/  public class RegExp04 {  public static void main(String[] args) {  String content = "hanshunping 韩 寒冷";  String regStr = "han|韩|寒";  Pattern pattern = Pattern.compile(regStr/\*, Pattern.CASE\_INSENSITIVE\*/);  Matcher matcher = pattern.matcher(content); |

|  |
| --- |
| while (matcher.find()) {  System.out.println("找到 " + matcher.group(0));  }  }  } |

27.6.5 元字符-限定符

用于指定其前面的字符和组合项连续出现多少次





应用案例:

|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\* |

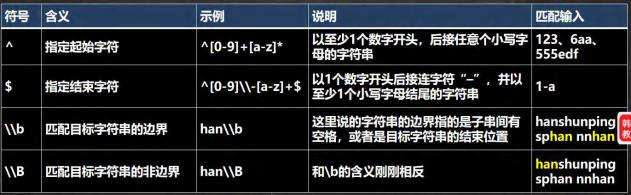
|  |
| --- |
| \* @author 韩顺平  \* @version 1.0  \* 演示限定符的使用  \*/  public class RegExp05 {  public static void main(String[] args) {  String content = "a211111aaaaaahello";  //a{3}, 1{4},\\d{2}  //String regStr = "a {3}";// 表示匹配 aaa  //String regStr = " 1{4}";// 表示匹配 1111  //String regStr = "\\d{2}";// 表示匹配 两位的任意数字字符  //a{3,4}, 1{4,5},\\d{2,5}  //细节：java 匹配默认贪婪匹配，即尽可能匹配多的  //String regStr = "a {3,4}"; //表示匹配 aaa 或者 aaaa  //String regStr = " 1{4,5}"; //表示匹配 1111 或者 11111  //String regStr = "\\d{2,5}"; //匹配 2 位数或者 3,4,5  //1+  //String regStr = " 1+"; //匹配一个 1 或者多个 1  //String regStr = "\\d+"; //匹配一个数字或者多个数字  //1\* |

|  |
| --- |
| //String regStr = " 1\*"; //匹配 0 个 1 或者多个 1  //演示?的使用, 遵守贪婪匹配  String regStr = "a1?"; //匹配 a 或者 a1  Pattern pattern = Pattern.compile(regStr/\*, Pattern.CASE\_INSENSITIVE\*/);  Matcher matcher = pattern.matcher(content);  while (matcher.find()) {  System.out.println("找到 " + matcher.group(0));  }  }  } |

27.6.6 元字符-定位符

定位符, 规定要匹配的字符串出现的位置，比如在字符串的开始还是在结束的位置，这个也是相当有用的，必须掌

握 RegExp06.java

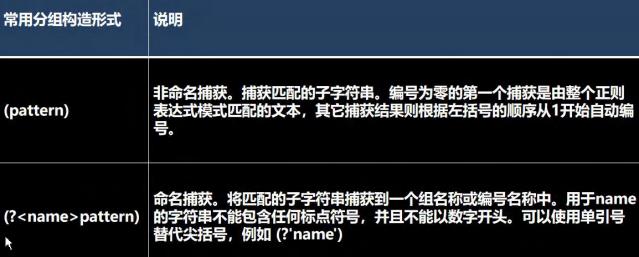


|  |
| --- |
| package com.hspedu.regexp; |

|  |
| --- |
| import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示定位符的使用  \*/  public class RegExp06 {  public static void main(String[] args) {  String content = "hanshunping sphan nnhan";  //String content = " 123-abc";  //以至少 1 个数字开头，后接任意个小写字母的字符串  //String regStr = "^[0-9]+[a-z]\*";  //以至少 1 个数字开头, 必须以至少一个小写字母结束  //String regStr = "^[0-9]+\\-[a-z]+$";  //表示匹配边界的 han[这里的边界是指：被匹配的字符串最后,  // 也可以是空格的子字符串的后面]  //String regStr = "han\\b";  //和\\b 的含义刚刚相反  String regStr = "han\\B"; |

|  |
| --- |
| Pattern pattern = Pattern.compile(regStr);  Matcher matcher = pattern.matcher(content);  while (matcher.find()) {  System.out.println("找到=" + matcher.group(0));  }  }  } |

27.6.7 分组





|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 分组:  \*/  public class RegExp07 {  public static void main(String[] args) {  String content = "hanshunping s7789 nn1189han";  //下面就是非命名分组  //说明 |

|  |
| --- |
| // 1. matcher.group(0) 得到匹配到的字符串  // 2. matcher.group(1) 得到匹配到的字符串的第 1 个分组内容  // 3. matcher.group(2) 得到匹配到的字符串的第 2 个分组内容  //String regStr = "(\\d\\d)(\\d\\d)";//匹配 4 个数字的字符串  //命名分组： 即可以给分组取名  String regStr = "(?<g1>\\d\\d)(?<g2>\\d\\d)";//匹配 4 个数字的字符串  Pattern pattern = Pattern.compile(regStr);  Matcher matcher = pattern.matcher(content);  while (matcher.find()) {  System.out.println("找到=" + matcher.group(0));  System.out.println("第 1 个分组内容=" + matcher.group(1));  System.out.println("第 1 个分组内容[通过组名]=" + matcher.group("g1"));  System.out.println("第 2 个分组内容=" + matcher.group(2));  System.out.println("第 2 个分组内容[通过组名]=" + matcher.group("g2"));  }  }  } |
| package com.hspedu.regexp;  import java.util.regex.Matcher; |

|  |
| --- |
| import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示非捕获分组, 语法比较奇怪  \*/  public class RegExp08 {  public static void main(String[] args) {  String content = "hello 韩顺平教育 jack 韩顺平老师 韩顺平同学 hello 韩顺平学生";  // 找到 韩顺平教育 、韩顺平老师、韩顺平同学 子字符串  //String regStr = "韩顺平教育|韩顺平老师|韩顺平同学";  //上面的写法可以等价非捕获分组, 注意：不能 matcher.group(1)  //String regStr = "韩顺平(?:教育|老师|同学)";  //找到 韩顺平 这个关键字,但是要求只是查找韩顺平教育和 韩顺平老师 中包含有的韩顺平  //下面也是非捕获分组，不能使用 matcher.group(1)  //String regStr = "韩顺平(?=教育|老师)";  //找到 韩顺平 这个关键字,但是要求只是查找 不是 (韩顺平教育 和 韩顺平老师) 中包含有的韩顺平  //下面也是非捕获分组，不能使用 matcher.group(1)  String regStr = "韩顺平(?!教育|老师)";  Pattern pattern = Pattern.compile(regStr); |

|  |
| --- |
| Matcher matcher = pattern.matcher(content);  while (matcher.find()) {  System.out.println("找到: " + matcher.group(0));  }  }  } |

27.7 应用实例

27.7.1 对字符串进行如下验证



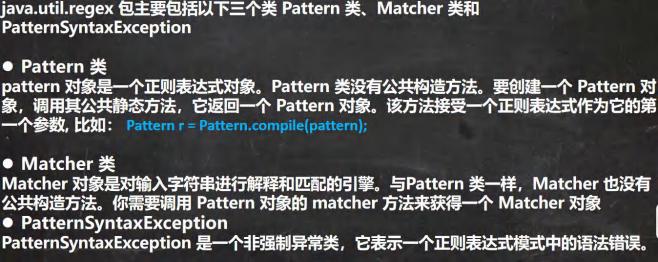
|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 正则表达式的应用实例 |

|  |
| --- |
| \*/  public class RegExp10 {  public static void main(String[] args) {  String content = " 13588889999";  // 汉字  //String regStr = "^[\u0391-\uffe5]+$";  // 邮政编码  // 要求：1.是 1-9 开头的一个六位数. 比如：123890  // 2.  // 3.  //String regStr = "^[1-9]\\d{5}$";  // QQ 号码  // 要求: 是 1-9 开头的一个(5 位数- 10 位数) 比如: 12389 , 1345687 , 187698765  //String regStr = "^[1-9]\\d{4,9}$";  // 手机号码  // 要求: 必须以 13, 14, 15, 18 开头的 11 位数 , 比如 13588889999  String regStr = "^ 1[3|4|5|8]\\d{9}$";  Pattern pattern = Pattern.compile(regStr);  Matcher matcher = pattern.matcher(content);  if(matcher.find()) {  System.out.println("满足格式");  } else {  System.out.println("不满足格式");  } |

|  |
| --- |
| }  } |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示正则表达式的使用  \*/  public class RegExp11 {  public static void main(String[] args) {  //String content = "https://www.bilibili.com/video/BV1fh411y7R8?from=search&seid= 1831060912083761326"; String content =  "<http://edu.3dsmax.tech/yg/bilibili/my6652/pc/qg/05-51/index.html#201211-1?track_id=jMc0jn-hm-yHrNfVad37YdhOUh41XY> mjlss9zocM26gspY5ArwWuxb4wYWpmh2Q7GzR7doU0wLkViEhUlO1qNtukyAgake2jG1bTd23lR57XzV83E9bAXWkStcAh  4j9Dz7a87ThGlqgdCZ2zpQy33a0SVNMfmJLSNnDzJ71TU68Rc-3PKE7VA3kYzjk4RrKU";  /\*\*  \* 思路  \* 1. 先确定 url 的开始部分 https:// | http:// |

|  |
| --- |
| \* 2.然后通过 ([\w-]+\.)+[\w-]+ 匹配 www.bilibili.com  \* 3. /video/BV1fh411y7R8?from=sear 匹配(\/[\w-?=&/%.#]\*)?  \*/  //多写多练，多总结  String regStr = "**^((http|https)://)?([\\w-]+\\.)+[\\w-]+(\\/[\\w-?=&/%.#]\*)?$**";//注意：[. ? \*]表示匹配就是.本身  Pattern pattern = Pattern.compile(regStr);  Matcher matcher = pattern.matcher(content);  if(matcher.find()) {  System.out.println("满足格式");  } else {  System.out.println("不满足格式");  }  //这里如果使用 Pattern 的 matches 整体匹配 比较简洁  System.out.println(Pattern.matches(regStr, content));//  }  } |

27.8 正则表达式三个常用类



|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示 matches 方法，用于整体匹配, 在验证输入的字符串是否满足条件使用  \*/  public class PatternMethod {  public static void main(String[] args) {  String content = "hello abc hello, 韩顺平教育";  //String regStr = "hello";  String regStr = "hello.\*";  boolean matches = Pattern.matches(regStr, content); |

|  |
| --- |
| System.out.println("整体匹配= " + matches);  }  } |
| package com.hspedu.regexp;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* Matcher 类的常用方法  \*/  public class MatcherMethod {  public static void main(String[] args) {  String content = "hello edu jack hspedutom hello smith hello hspedu hspedu";  String regStr = "hello";  Pattern pattern = Pattern.compile(regStr);  Matcher matcher = pattern.matcher(content);  while (matcher.find()) {  System.out.println("=================");  System.out.println(matcher.start());  System.out.println(matcher.end());  System.out.println("找到: " + content.substring(matcher.start(), matcher.end())); |

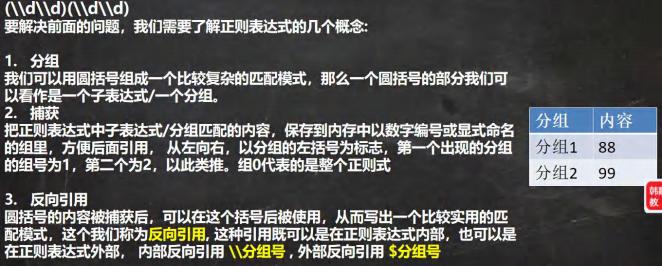
|  |
| --- |
| }  //整体匹配方法，常用于，去校验某个字符串是否满足某个规则  System.out.println("整体匹配=" + matcher.matches());  //完成如果 content 有 hspedu 替换成 韩顺平教育  regStr = "hspedu";  pattern = Pattern.compile(regStr);  matcher = pattern.matcher(content);  //注意：返回的字符串才是替换后的字符串 原来的 content 不变化  String newContent = matcher.replaceAll("韩顺平教育");  System.out.println("newContent=" + newContent);  System.out.println("content=" + content);  }  } |

27.9 分组、捕获、反向引用

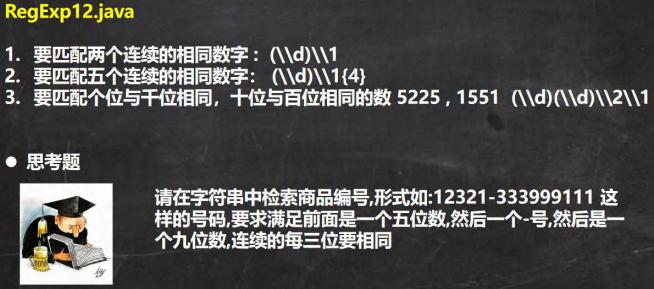
27.9.1 提出需求



27.9.2 介绍



27.9.3 看几个小案例



27.9.4 经典的结巴程序

把 类似 : "我....我要....学学学学....编程java!";

通过正则表达式 修改成 "我要学编程java" RegExp13.java

|  |
| --- |
| package com.hspedu.regexp;  import java.util.regex.Matcher; |

|  |
| --- |
| import java.util.regex.Pattern;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class RegExp13 {  public static void main(String[] args) {  String content = "我....我要....学学学学....编程java!";  //1. 去掉所有的.  Pattern pattern = Pattern.compile("\\.");  Matcher matcher = pattern.matcher(content);  content = matcher.replaceAll("");  // System.out.println("content=" + content);  //2. 去掉重复的字 我我要学学学学编程java!  // 思路  //(1) 使用 (.)\\1+  //(2) 使用 反向引用$1 来替换匹配到的内容  // 注意：因为正则表达式变化，所以需要重置 matcher  // pattern = Pattern.compile("(.)\\1+");//分组的捕获内容记录到$1  // matcher = pattern.matcher(content);  // while (matcher.find()) { |

|  |
| --- |
| // System.out.println("找到=" + matcher.group(0));  // }  //  // //使用 反向引用$1 来替换匹配到的内容  // content = matcher.replaceAll("$1");  // System.out.println("content=" + content);  //3. 使用一条语句 去掉重复的字 我我要学学学学编程java!  content = Pattern.compile("(.)\\1+").matcher(content).replaceAll("$1");  System.out.println("content=" + content);  }  } |

27. 10String 类中使用正则表达式

27.10. 1 替换功能

StringReg.java

String 类 public String replaceAll(String regex,String replacement)

27.10.2 判断功能

String 类 public boolean matches(String regex){} **//使用** **Pattern 和** **Matcher 类**

27.10.3 分割功能

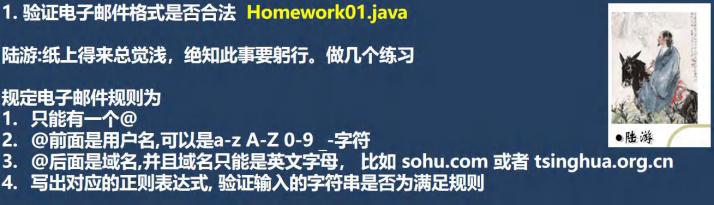
String 类 public String[] split(String regex)

|  |
| --- |
| package com.hspedu.regexp; |

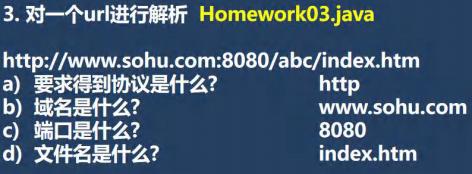
|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class StringReg {  public static void main(String[] args) {  String content = "2000 年 5 月，JDK1.3 、JDK1.4 和 J2SE1.3 相继发布，几周后其" +  "获得了 Apple 公司 Mac OS X 的工业标准的支持。2001 年 9 月 24 日，J2EE1.3 发" +  "布。" +  "2002 年 2 月 26 日，J2SE1.4 发布。 自此 Java 的计算能力有了大幅提升";  //使用正则表达式方式，将 JDK1.3 和 JDK1.4 替换成 JDK  content = content.replaceAll("JDK1\\.3|JDK1\\.4", "JDK");  System.out.println(content);  //要求 验证一个 手机号， 要求必须是以 138 139 开头的  content = " 13888889999";  if (content.matches(" 1(38|39)\\d{8}")) {  System.out.println("验证成功");  } else {  System.out.println("验证失败");  } |

|  |
| --- |
| //要求按照 # 或者 - 或者 ~ 或者 数字 来分割  System.out.println("===================");  content = "hello#abc-jack12smith~北京";  String[] split = content.split("#|-|~|\\d+");  for (String s : split) {  System.out.println(s);  }  }  } |

27. 11本章作业

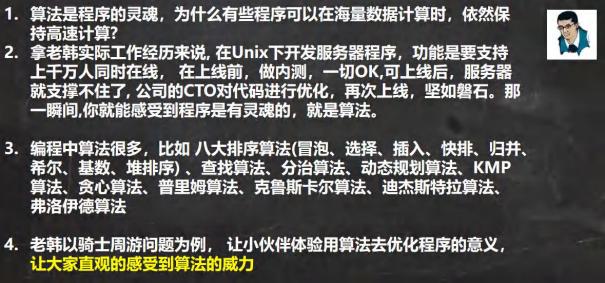






第 28 章算法优化体验课-骑士周游问题

28. 1 算法优化意义

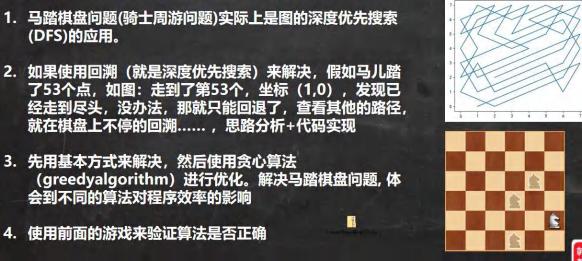


28.2 经典算法面试题-骑士周游问题

28.2.1 马踏棋盘算法介绍和游戏演示



28.2.2 马踏棋盘算法介绍和游戏演示



28.3 第三个阶段(结束) ，大家好好总结

