今日内容

- 正则表达式

教学目标

- [ ] 能够理解正则表达式的作用
- [ ] 能够使用正则表达式的字符类
- [ ] 能够使用正则表达式的逻辑运算符
- [ ] 能够使用正则表达式的预定义字符类
- [ ] 能够使用正则表达式的限定符
- [ ] 能够使用正则表达式的分组
- [ ] 能够在String的split方法中使用正则表达式

正则表达式

1.1 正则表达式的概念及演示

- 在Java中，我们经常需要验证一些字符串，例如：年龄必须是2位的数字、用户名必须是8位长度而且只能包含大小写字母、数字等。正则表达式就是用来验证各种字符串的规则。它内部描述了一些规则，我们可以验证用户输入的字符串是否匹配这个规则。
- 先看一个不使用正则表达式验证的例子：下面的程序让用户输入一个QQ号码，我们要验证：
  - QQ号码必须是5--15位长度
  - 而且必须全部是数字
  - 而且首位不能为0

```
package com.itheima.a08regexdemo;

public class RegexDemo1 {
    public static void main(String[] args) {
        /* 假如现在要求校验一个qq号码是否正确。
            规则:6位及20位之内，日不能在开头，必须全部是数字。
            先使用目前所学知识完成校验需求然后体验一下正则表达式检验。
        */

        String qq ="1234567890";
        System.out.println(checkQQ(qq));

        System.out.println(qq.matches("[1-9]\\d{5,19}"));

    }

    public static boolean checkQQ(String qq) {
        //规则:6位及20位之内，日不能在开头，必须全部是数字 。
        //核心思想:
        //先把异常数据进行过滤
        //下面的就是满足要求的数据了。
```

```
        int len = qq.length();
        if (len < 6 || len > 20) {
            return false;
        }
        //0不能在开头
        if (qq.startsWith("0")) {
            return false;
        }
        //必须全部是数字
        for (int i = 0; i < qq.length(); i++) {
            char c = qq.charAt(i);
            if (c < '0' | c > '9') {
                return false;
            }
        }
        return true;
    }
}
```

- 使用正则表达式验证：

```
public class Demo {
    public static void main(String[] args) {
        String qq ="1234567890";
        System.out.println(qq.matches("[1-9]\\d{5,19}"));
    }
}
```

我们接下来就重点学习怎样写正则表达式

1.2 正则表达式-字符类

- 语法示例：

1. [abc]：代表a或者b，或者c字符中的一个。
2. [^abc]：代表除a,b,c以外的任何字符。
3. [a-z]：代表a-z的所有小写字符中的一个。
4. [A-Z]：代表A-Z的所有大写字符中的一个。
5. [0-9]：代表0-9之间的某一个数字字符。
6. [a-zA-Z0-9]：代表a-z或者A-Z或者0-9之间的任意一个字符。
7. [a-dm-p]：a 到 d 或 m 到 p之间的任意一个字符。

- 代码示例：

```
package com.itheima.a08regexdemo;

public class RegexDemo2 {
```

```
    public static void main(String[] args) {
        //public boolean matches(String regex):判断是否与正则表达式匹配，匹配返回
true
        // 只能是a b c
        System.out.println("-----------1-------------");
        System.out.println("a".matches("[abc]")); // true
        System.out.println("z".matches("[abc]")); // false

        // 不能出现a b c
        System.out.println("-----------2-------------");
        System.out.println("a".matches("[^abc]")); // false
        System.out.println("z".matches("[^abc]")); // true
        System.out.println("zz".matches("[^abc]")); //false
        System.out.println("zz".matches("[^abc][^abc]")); //true

        // a到zA到Z(包括头尾的范围)
        System.out.println("-----------3-------------");
        System.out.println("a".matches("[a-zA-z]")); // true
        System.out.println("z".matches("[a-zA-z]")); // true
        System.out.println("aa".matches("[a-zA-z]"));//false
        System.out.println("zz".matches("[a-zA-Z]")); //false
        System.out.println("zz".matches("[a-zA-Z][a-zA-Z]")); //true
        System.out.println("0".matches("[a-zA-Z]"));//false
        System.out.println("0".matches("[a-zA-Z0-9]"));//true


        // [a-d[m-p]] a到d，或m到p
        System.out.println("-----------4-------------");
        System.out.println("a".matches("[a-d[m-p]]"));//true
        System.out.println("d".matches("[a-d[m-p]]")); //true
        System.out.println("m".matches("[a-d[m-p]]")); //true
        System.out.println("p".matches("[a-d[m-p]]")); //true
        System.out.println("e".matches("[a-d[m-p]]")); //false
        System.out.println("0".matches("[a-d[m-p]]")); //false

        // [a-z&&[def]] a-z和def的交集。为:d，e，f
        System.out.println("----------5------------");
        System.out.println("a".matches("[a-z&[def]]")); //false
        System.out.println("d".matches("[a-z&&[def]]")); //true
        System.out.println("0".matches("[a-z&&[def]]")); //false

        // [a-z&&[^bc]] a-z和非bc的交集。(等同于[ad-z])
        System.out.println("-----------6------------_");
        System.out.println("a".matches("[a-z&&[^bc]]"));//true
        System.out.println("b".matches("[a-z&&[^bc]]")); //false
        System.out.println("0".matches("[a-z&&[^bc]]")); //false

        // [a-z&&[^m-p]] a到z和除了m到p的交集。(等同于[a-1q-z])
        System.out.println("-----------7-------------");
        System.out.println("a".matches("[a-z&&[^m-p]]")); //true
        System.out.println("m".matches("[a-z&&[^m-p]]")); //false
        System.out.println("0".matches("[a-z&&[^m-p]]")); //false
```

```
    }
}
```

1.3 正则表达式-逻辑运算符

- 语法示例：
    1. &&：并且
    2. | ： 或者
    3. \ ： 转义字符
- 代码示例：

```
public class Demo {
    public static void main(String[] args) {
        String str = "had";

        //1.要求字符串是小写辅音字符开头，后跟ad
        String regex = "[a-z&&[^aeiou]]ad";
        System.out.println("1." + str.matches(regex));

        //2.要求字符串是aeiou中的某个字符开头，后跟ad
        regex = "[a|e|i|o|u]ad";//这种写法相当于：regex = "[aeiou]ad";
        System.out.println("2." + str.matches(regex));
    }
}
```

```
package com.itheima.a08regexdemo;

public class RegexDemo3 {
    public static void main(String[] args) {
        // \ 转义字符 改变后面那个字符原本的含义
        //练习:以字符串的形式打印一个双引号
        //"在Java中表示字符串的开头或者结尾

        //此时\表示转义字符，改变了后面那个双引号原本的含义
        //把他变成了一个普普通通的双引号而已。
        System.out.println("\"");

        // \表示转义字符
        //两个\的理解方式：前面的\是一个转义字符，改变了后面\原本的含义，把他变成一个普普通通的\而已。
        System.out.println("c:Users\\moon\\IdeaProjects\\basic-code\\myapi\\src\\com\\itheima\\a08regexdemo\\RegexDemo1.java");
```

```
    }
}
```

1.4 正则表达式-预定义字符

- 语法示例：
    1. "." ： 匹配任何字符。
    2. "\d"：任何数字[0-9]的简写；
    3. "\D"：任何非数字[^0-9]的简写；
    4. "\s"： 空白字符：[\t\n\x0B\f\r] 的简写
    5. "\S"： 非空白字符：[^\s] 的简写
    6. "\w"：单词字符：[a-zA-Z_0-9]的简写
    7. "\W"：非单词字符：[^\w]
- 代码示例：

```
public class Demo {
    public static void main(String[] args) {
        //.表示任意一个字符
        System.out.println("你".matches(".."));//false
        System.out.println("你".matches("."));//true
        System.out.println("你a".matches(".."));//true

        // \\d 表示任意的一个数字
        // \\d只能是任意的一位数字
        // 简单来记:两个\表示一个\
        System.out.println("a".matches("\\d"));// false
        System.out.println("3".matches("\\d"));// true
        System.out.println("333".matches("\\d"));// false

        //\\w只能是一位单词字符[a-zA-Z_0-9]
        System.out.println("z".matches("\\w"));// true
        System.out.println("2".matches("\\w"));// true
        System.out.println("21".matches("\\w"));// false
        System.out.println("你".matches("\\w"));//false

        // 非单词字符
        System.out.println("你".matches("\\W"));// true
        System.out.println("-----------------------------------------");
        // 以上正则匹配只能校验单个字符。


        // 必须是数字 字母 下划线 至少 6位
        System.out.println("2442fsfsf".matches("\\w{6,}"));//true
        System.out.println("244f".matches("\\w{6,}"));//false

        // 必须是数字和字符 必须是4位
        System.out.println("23dF".matches("[a-zA-Z0-9]{4}"));//true
```

```
        System.out.println("23 F".matches("[a-zA-Z0-9]{4}"));//false
        System.out.println("23dF".matches("[\\w&&[^_]]{4}"));//true
        System.out.println("23_F".matches("[\\w&&[^_]]{4}"));//false

    }
}
```

1.5 正则表达式-数量词

- 语法示例：
    1. X? : 0次或1次
    2. X* : 0次到多次
    3. X+ : 1次或多次
    4. X{n} : 恰好n次
    5. X{n,} : 至少n次
    6. X{n,m}: n到m次(n和m都是包含的)
- 代码示例：

```
public class Demo {
    public static void main(String[] args) {
         // 必须是数字 字母 下划线 至少 6位
        System.out.println("2442fsfsf".matches("\\w{6,}"));//true
        System.out.println("244f".matches("\\w{6,}"));//false

        // 必须是数字和字符 必须是4位
        System.out.println("23dF".matches("[a-zA-Z0-9]{4}"));//true
        System.out.println("23 F".matches("[a-zA-Z0-9]{4}"));//false
        System.out.println("23dF".matches("[\\w&&[^_]]{4}"));//true
        System.out.println("23_F".matches("[\\w&&[^_]]{4}"));//false
    }
}
```

1.6 正则表达式练习1

需求：

请编写正则表达式验证用户输入的手机号码是否满足要求。

请编写正则表达式验证用户输入的邮箱号是否满足要求。

请编写正则表达式验证用户输入的电话号码是否满足要求。

验证手机号码 13112345678 13712345667 13945679027 139456790271

验证座机电话号码 020-2324242 02122442 027-42424 0712-3242434

验证邮箱号码 3232323@qq.com zhangsan@itcast.cnn dlei0009@163.com dlei0009@pci.com.cn

代码示例：

```
package com.itheima.a08regexdemo;

public class RegexDemo4 {
    public static void main(String[] args) {
        /*
            需求
            请编写正则表达式验证用户输入的手机号码是否满足要求。请编写正则表达式验证用户输入的邮箱号是否满足要求。请编写正则表达式验证用户输入的电话号码是否满足要求。
            验证手机号码 13112345678 13712345667 13945679027 139456790271
            验证座机电话号码 020-2324242 02122442 027-42424 0712-3242434
            验证邮箱号码 3232323@qq.com zhangsan@itcast.cnn dlei0009@163.com dlei0009@pci.com.cn
        */

        //心得:
        //拿着一个正确的数据，从左到右依次去写。
        //13112345678
        //分成三部分:
        //第一部分:1 表示手机号码只能以1开头
        //第二部分:[3-9] 表示手机号码第二位只能是3-9之间的
        //第三部分:\\d{9} 表示任意数字可以出现9次，也只能出现9次
        String regex1 = "1[3-9]\\d{9}";
        System.out.println("13112345678".matches(regex1));//true
        System.out.println("13712345667".matches(regex1));//true
        System.out.println("13945679027".matches(regex1));//true
        System.out.println("139456790271".matches(regex1));//false
        System.out.println("-----------------------------------");

        //座机电话号码
        //020-2324242 02122442 027-42424 0712-3242434
        //思路:
        //在书写座机号正则的时候需要把正确的数据分为三部分
        //一:区号@\\d{2,3}
        //      0:表示区号一定是以0开头的
        //      \\d{2,3}:表示区号从第二位开始可以是任意的数字，可以出现2到3次。
        //二:- ?表示次数，日次或一次
        //三:号码 号码的第一位也不能以日开头，从第二位开始可以是任意的数字，号码的总长度:5-10位
        String regex2 = "0\\d{2,3}-?[1-9]\\d{4,9}";
        System.out.println("020-2324242".matches(regex2));
        System.out.println("02122442".matches(regex2));
        System.out.println("027-42424".matches(regex2));
        System.out.println("0712-3242434".matches(regex2));

        //邮箱号码
        //3232323@qq.com zhangsan@itcast.cnn dlei0009@163.com dlei0009@pci.com.cn
        //思路:
        //在书写邮箱号码正则的时候需要把正确的数据分为三部分
        //第一部分:@的左边 \\w+
```

```
        //      任意的字母数字下划线，至少出现一次就可以了
        //第二部分:@ 只能出现一次
        //第三部分:
        //      3.1         .的左边[\\w&&[^_]]{2,6}
        //                  任意的字母加数字，总共出现2-6次(此时不能出现下划线)
        //      3.2         . \\.
        //      3.3         大写字母，小写字母都可以，只能出现2-3次[a-zA-Z]{2,3}
        //      我们可以把3.2和3.3看成一组，这一组可以出现1次或者两次
        String regex3 = "\\w+@[\\w&&[^_]]{2,6}(\\.[a-zA-Z]{2,3}){1,2}";
        System.out.println("3232323@qq.com".matches(regex3));
        System.out.println("zhangsan@itcast.cnn".matches(regex3));
        System.out.println("dlei0009@163.com".matches(regex3));
        System.out.println("dlei0009@pci.com.cn".matches(regex3));


        //24小时的正则表达式
        String regex4 = "([01]\\d|2[0-3]):[0-5]\\d:[0-5]\\d";
        System.out.println("23:11:11".matches(regex4));

        String regex5 = "([01]\\d 2[0-3])(:[0-5]\\d){2}";
        System.out.println("23:11:11".matches(regex5));
    }
}
```

1.7 正则表达式练习2

需求
请编写正则表达式验证用户名是否满足要求。要求:大小写字母，数字，下划线一共4-16位
请编写正则表达式验证身份证号码是否满足要求。
简单要求:
18位，前17位任意数字，最后一位可以是数字可以是大写或小写的x
复杂要求:
按照身份证号码的格式严格要求。

身份证号码:
41080119930228457x
510801197609022309
15040119810705387X
130133197204039024
430102197606046442

代码示例:

```
public class RegexDemo5 {
    public static void main(String[] args) {
        /*
            正则表达式练习:
            需求
```

```
        请编写正则表达式验证用户名是否满足要求。要求:大小写字母，数字，下划线一共4-16位
        请编写正则表达式验证身份证号码是否满足要求。
        简单要求:
            18位，前17位任意数字，最后一位可以是数字可以是大写或小写的x
        复杂要求:
            按照身份证号码的格式严格要求。

        身份证号码:
        41080119930228457x
        510801197609022309
        15040119810705387X
        130133197204039024 I
        430102197606046442
    */

    //用户名要求:大小写字母，数字，下划线一共4-16位
    String regex1 = "\\w{4,16}";
    System.out.println("zhangsan".matches(regex1));
    System.out.println("lisi".matches(regex1));
    System.out.println("wangwu".matches(regex1));
    System.out.println("$123".matches(regex1));


    //身份证号码的简单校验:
    //18位，前17位任意数字，最后一位可以是数字可以是大写或小写的x
    String regex2 = "[1-9]\\d{16}(\\d|x|x)";
    String regex3 = "[1-9]\\d{16}[\\dXx]";
    String regex5 = "[1-9]\\d{16}(\\d(?i)x)";

    System.out.println("41080119930228457x".matches(regex3));
    System.out.println("510801197609022309".matches(regex3));
    System.out.println("15040119810705387X".matches(regex3));
    System.out.println("130133197204039024".matches(regex3));
    System.out.println("430102197606046442".matches(regex3));


    //忽略大小写的书写方式
    //在匹配的时候忽略abc的大小写
    String regex4 = "a((?i)b)c";
    System.out.println("------------------------------");
    System.out.println("abc".matches(regex4));//true
    System.out.println("ABC".matches(regex4));//false
    System.out.println("aBc".matches(regex4));//true


    //身份证号码的严格校验
    //编写正则的小心得:
    //第一步:按照正确的数据进行拆分
    //第二步:找每一部分的规律，并编写正则表达式
    //第三步:把每一部分的正则拼接在一起，就是最终的结果
    //书写的时候:从左到右去书写。
```

```
        //410801 1993 02 28 457x
        //前面6位:省份，市区，派出所等信息，第一位不能是0，后面5位是任意数字
[1-9]\\d{5}
        //年的前半段: 18 19 20
(18|19|20)
        //年的后半段: 任意数字出现两次
\\d{2}
        //月份: 01~ 09 10 11 12
(@[1-9]|1[0-2])
        //日期: 01~09 10~19 20~29 30 31
(0[1-9]|[12]\\d|3[01])
        //后面四位: 任意数字出现3次 最后一位可以是数字也可以是大写x或者小写x
\\d{3}[\\dXx]
        String regex6 = "[1-9]\\d{5}(18|19|20)\\d{2}(@[1-9]|1[0-2])(@[1-9]|
[12]\\d|3[01])\\d{3}[\\dxXx]";

        System.out.println("41080119930228457x".matches(regex6));
        System.out.println("510801197609022309".matches(regex6));
        System.out.println("15040119810705387X".matches(regex6));
        System.out.println("130133197204039024".matches(regex6));
        System.out.println("430102197606046442".matches(regex6));


    }
}
```

1.8 本地数据爬取

Pattern：表示正则表达式
Matcher：文本匹配器，作用按照正则表达式的规则去读取字符串，从头开始读取。
在大串中去找符合匹配规则的子串。

代码示例：

```
package com.itheima.a08regexdemo;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexDemo6 {
    public static void main(String[] args) {
        /* 有如下文本，请按照要求爬取数据。
                Java自从95年问世以来，经历了很多版本，目前企业中用的最多的是Java8和
Java11，
                因为这两个是长期支持版本，下一个长期支持版本是Java17，相信在未来不久
Java17也会逐渐登上历史舞台
                要求:找出里面所有的JavaXX
         */
```

```
        String str = "Java自从95年问世以来，经历了很多版本，目前企业中用的最多的是Java8和Java11，" +
                "因为这两个是长期支持版本，下一个长期支持版本是Java17，相信在未来不久Java17也会逐渐登上历史舞台";


        //1.获取正则表达式的对象
        Pattern p = Pattern.compile("Java\\d{0,2}");
        //2.获取文本匹配器的对象
        //拿着m去读取str，找符合p规则的子串
        Matcher m = p.matcher(str);

        //3.利用循环获取
        while (m.find()) {
            String s = m.group();
            System.out.println(s);
        }


    }

    private static void method1(String str) {
        //Pattern:表示正则表达式
        //Matcher: 文本匹配器，作用按照正则表达式的规则去读取字符串，从头开始读取。
        //          在大串中去找符合匹配规则的子串。

        //获取正则表达式的对象
        Pattern p = Pattern.compile("Java\\d{0,2}");
        //获取文本匹配器的对象
        //m:文本匹配器的对象
        //str:大串
        //p:规则
        //m要在str中找符合p规则的小串
        Matcher m = p.matcher(str);

        //拿着文本匹配器从头开始读取，寻找是否有满足规则的子串
        //如果没有，方法返回false
        //如果有，返回true。在底层记录子串的起始索引和结束索引+1
        // 0,4
        boolean b = m.find();

        //方法底层会根据find方法记录的索引进行字符串的截取
        // substring(起始索引，结束索引);包头不包尾
        // (0,4)但是不包含4索引
        // 会把截取的小串进行返回。
        String s1 = m.group();
        System.out.println(s1);


        //第二次在调用find的时候，会继续读取后面的内容
        //读取到第二个满足要求的子串，方法会继续返回true
        //并把第二个子串的起始索引和结束索引+1，进行记录
        b = m.find();
```

```
        //第二次调用group方法的时候，会根据find方法记录的索引再次截取子串
        String s2 = m.group();
        System.out.println(s2);
    }
}
```

1.9 网络数据爬取（了解）

需求：

把连接:https://m.sengzan.com/jiaoyu/29104.html?ivk sa=1025883i中所有的身份证号码都爬取出来。

代码示例：

```
public class RegexDemo7 {
    public static void main(String[] args) throws IOException {
        /* 扩展需求2:
            把连接:https://m.sengzan.com/jiaoyu/29104.html?ivk sa=1025883i
            中所有的身份证号码都爬取出来。
        */

        //创建一个URL对象
        URL url = new URL("https://m.sengzan.com/jiaoyu/29104.html?ivk
sa=1025883i");
        //连接上这个网址
        //细节:保证网络是畅通
        URLConnection conn = url.openConnection();//创建一个对象去读取网络中的数
据
        BufferedReader br = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
        String line;
        //获取正则表达式的对象pattern
        String regex = "[1-9]\\d{17}";
        Pattern pattern = Pattern.compile(regex);//在读取的时候每次读一整行
        while ((line = br.readLine()) != null) {
            //拿着文本匹配器的对象matcher按照pattern的规则去读取当前的这一行信息
            Matcher matcher = pattern.matcher(line);
            while (matcher.find()) {
                System.out.println(matcher.group());
            }
        }
        br.close();
    }
}
```

1.10 爬取数据练习

需求：

把下面文本中的座机电话，邮箱，手机号，热线都爬取出来。

来黑马程序员学习Java，手机号:18512516758，18512508907或者联系邮箱:boniu@itcast.cn，座机电话:01036517895，010-98951256邮箱:bozai@itcast.cn，热线电话:400-618-9090 ，400-618-4000，4006184000，4006189090手机号的正则表达式:1[3-9]\d{9}

代码示例：

```
package com.itheima.a08regexdemo;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexDemo8 {
    public static void main(String[] args) {
        /*
            需求:把下面文本中的座机电话，邮箱，手机号，热线都爬取出来。
            来黑马程序员学习Java，
            手机号:18512516758，18512508907或者联系邮箱:boniu@itcast.cn，
            座机电话:01036517895，010-98951256邮箱:bozai@itcast.cn，
            热线电话:400-618-9090 ，400-618-4000，4006184000，4006189090

            手机号的正则表达式:1[3-9]\d{9}
            邮箱的正则表达式:\w+@[\w&&[^_]]{2,6}(\.[a-zA-Z]{2,3}){1,2}座机电话的
正则表达式:0\d{2,3}-?[1-9]\d{4,9}
            热线电话的正则表达式:400-?[1-9]\\d{2}-?[1-9]\\d{3}

        */

        String s = "来黑马程序员学习Java，" +
                "电话:18512516758，18512508907" + "或者联系邮
箱:boniu@itcast.cn，" +
                "座机电话:01036517895，010-98951256" + "邮箱:bozai@itcast.cn，"
+
                "热线电话:400-618-9090 ，400-618-4000，4006184000，4006189090";

        System.out.println("400-618-9090");

        String regex = "(1[3-9]\\d{9})|(\\w+@[\\w&&[^_]]{2,6}(\\.[a-zA-Z]{2,3})
{1,2})" +
                "|(0\\d{2,3}-?[1-9]\\d{4,9})" +
                "(400-?[1-9]\\d{2}-?[1-9]\\d{3})";

        //1.获取正则表达式的对象
        Pattern p = Pattern.compile(regex);

        //2.获取文本匹配器的对象
        //利用m去读取s，会按照p的规则找里面的小串
        Matcher m = p.matcher(s);
        //3.利用循环获取每一个数据 while(m.find()){
```

```
            String str = m.group();
            System.out.println(str);

        }
    }
```

1.11 按要求爬取

需求:

有如下文本，按要求爬取数据。

Java自从95年问世以来，经历了很多版本，目前企业中用的最多的是Java8和Java11，因为这两个是长期支持版本，下一个长期支持版本是Java17，相信在未来不久Java17也会逐渐登上历史舞台。

需求1:

爬取版本号为8，11.17的Java文本，但是只要Java，不显示版本号。

需求2:

爬取版本号为8，11，17的Java文本。正确爬取结果为：Java8 Java11 Java17 Java17

需求3:

爬取除了版本号为8，11，17的Java文本。
代码示例:

```
public class RegexDemo9 {
    public static void main(String[] args) {
        /*
            有如下文本，按要求爬取数据。
                Java自从95年问世以来，经历了很多版本，目前企业中用的最多的是Java8和
Java11，
                因为这两个是长期支持版本，下一个长期支持版本是Java17，相信在未来不久
Java17也会逐渐登上历史舞台


            需求1:爬取版本号为8，11.17的Java文本，但是只要Java，不显示版本号。
            需求2:爬取版本号为8，11，17的Java文本。正确爬取结果为:Java8 Java11
Java17 Java17
            需求3:爬取除了版本号为8，11.17的Java文本，
        */
        String s = "Java自从95年问世以来，经历了很多版本，目前企业中用的最多的是
Java8和Java11，" +
            "因为这两个是长期支持版本，下一个长期支持版本是Java17，相信在未来不久
Java17也会逐渐登上历史舞台";

        //1.定义正则表达式
        //?理解为前面的数据Java
        //=表示在Java后面要跟随的数据
```

```
        //但是在获取的时候，只获取前半部分
        //需求1:
        String regex1 = "((?i)Java)(?=8|11|17)";
        //需求2:
        String regex2 = "((?i)Java)(8|11|17)";
        String regex3 = "((?i)Java)(?:8|11|17)";
        //需求3:
        String regex4 = "((?i)Java)(?!8|11|17)";

        Pattern p = Pattern.compile(regex4);
        Matcher m = p.matcher(s);
        while (m.find()) {
            System.out.println(m.group());
        }
    }
}
```

1.12 贪婪爬取和非贪婪爬取

只写+和表示贪婪匹配，如果在+和后面加问号表示非贪婪爬取
+? 非贪婪匹配
*? 非贪婪匹配
贪婪爬取:在爬取数据的时候尽可能的多获取数据
非贪婪爬取:在爬取数据的时候尽可能的少获取数据

举例:
如果获取数据: ab+
贪婪爬取获取结果:abbbbbbbbbbbb
非贪婪爬取获取结果:ab

代码示例:

```
public class RegexDemo10 {
    public static void main(String[] args) {
        /*
            只写+和*表示贪婪匹配

            +? 非贪婪匹配
            *? 非贪婪匹配

            贪婪爬取:在爬取数据的时候尽可能的多获取数据
            非贪婪爬取:在爬取数据的时候尽可能的少获取数据

            ab+:
            贪婪爬取:abbbbbbbbbbbb
            非贪婪爬取:ab
        */
```

```
        String s = "Java自从95年问世以来，abbbbbbbbbbbaaaaaaaaaaaaaaaa" +
                "经历了很多版木，目前企业中用的最多的是]ava8和]ava11，因为这两个是长期支持版木。" +
                "下一个长期支持版本是Java17，相信在未来不久Java17也会逐渐登上历史舞台";

        String regex = "ab+";
        Pattern p = Pattern.compile(regex);
        Matcher m = p.matcher(s);

        while (m.find()) {
            System.out.println(m.group());
        }


    }
}
```

1.13 String的split方法中使用正则表达式

- String类的split()方法原型：

```
public String[] split(String regex)
//参数regex表示正则表达式。可以将当前字符串中匹配regex正则表达式的符号作为"分隔符"来切割字符串。
```

- 代码示例：

```
/*
            有一段字符串:小诗诗dqwefqwfqwfwq12312小丹丹dqwefqwfqwfwq12312小惠惠
            要求1:把字符串中三个姓名之间的字母替换为vs
            要求2:把字符串中的三个姓名切割出来*/

String s = "小诗诗dqwefqwfqwfwq12312小丹丹dqwefqwfqwfwq12312小惠惠";
//细节:
//方法在底层跟之前一样也会创建文本解析器的对象
//然后从头开始去读取字符串中的内容，只要有满足的，那么就切割。
String[] arr = s.split("[\\w&&[^_]]+");
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}
```

1.14 String类的replaceAll方法中使用正则表达式

- String类的replaceAll()方法原型：

```
public String replaceAll(String regex,String newStr)
//参数regex表示一个正则表达式。可以将当前字符串中匹配regex正则表达式的字符串替换为newStr。
```

- 代码示例：

```
/*
            有一段字符串:小诗诗dqwefqwfqwfwq12312小丹丹dqwefqwfqwfwq12312小惠惠
            要求1:把字符串中三个姓名之间的字母替换为vs
            要求2:把字符串中的三个姓名切割出来*/

String s = "小诗诗dqwefqwfqwfwq12312小丹丹dqwefqwfqwfwq12312小惠惠";
//细节:
//方法在底层跟之前一样也会创建文本解析器的对象
//然后从头开始去读取字符串中的内容，只要有满足的，那么就用第一个参数去替换。
String result1 = s.replaceAll("[\\w&&[^_]]+", "vs");
System.out.println(result1);
```

1.15 正则表达式-分组括号( )

细节：如何识别组号?

只看左括号，不看有括号，按照左括号的顺序，从左往右，依次为第一组，第二组，第三组等等

```
//需求1:判断一个字符串的开始字符和结束字符是否一致?只考虑一个字符
//举例: a123a b456b 17891 &abc& a123b(false)
// \\组号:表示把第X组的内容再出来用一次
String regex1 = "(.).+\\1";
System.out.println("a123a".matches(regex1));
System.out.println("b456b".matches(regex1));
System.out.println("17891".matches(regex1));
System.out.println("&abc&".matches(regex1));
System.out.println("a123b".matches(regex1));
System.out.println("--------------------------");


//需求2:判断一个字符串的开始部分和结束部分是否一致?可以有多个字符
//举例: abc123abc b456b 123789123 &!@abc&!@ abc123abd(false)
String regex2 = "(.+).+\\1";
System.out.println("abc123abc".matches(regex2));
System.out.println("b456b".matches(regex2));
System.out.println("123789123".matches(regex2));
System.out.println("&!@abc&!@".matches(regex2));
System.out.println("abc123abd".matches(regex2));
System.out.println("----------------------");

//需求3:判断一个字符串的开始部分和结束部分是否一致?开始部分内部每个字符也需要一致
```

```
//举例: aaa123aaa bbb456bbb 111789111 &&abc&&
//(.):把首字母看做一组
// \\2:把首字母拿出来再次使用
// *:作用于\\2,表示后面重复的内容出现日次或多次
String regex3 = "((.)\\2*).+\\1";
System.out.println("aaa123aaa".matches(regex3));
System.out.println("bbb456bbb".matches(regex3));
System.out.println("111789111".matches(regex3));
System.out.println("&&abc&&".matches(regex3));
System.out.println("aaa123aab".matches(regex3));
```

1.16 分组练习

需求:

将字符串：我要学学编编编编程程程程程程。

替换为：我要学编程

```
String str = "我要学学编编编编程程程程程程";

//需求:把重复的内容 替换为 单个的
//学学                  学
//编编编编              编
//程程程程程程          程
//  (.)表示把重复内容的第一个字符看做一组
//  \\1表示第一字符再次出现
//  + 至少一次
//  $1 表示把正则表达式中第一组的内容，再拿出来用
String result = str.replaceAll("(.)\\1+", "$1");
System.out.println(result);
```

1.17 忽略大小写的写法

```
//(?i) : 表示忽略后面数据的大小写
//忽略abc的大小写
String regex = "(?i)abc";
//a需要一模一样，忽略bc的大小写
String regex = "a(?i)bc";
//ac需要一模一样，忽略b的大小写
String regex = "a((?i)b)c";
```

1.18 非捕获分组

非捕获分组：分组之后不需要再用本组数据，仅仅是把数据括起来。

```
//身份证号码的简易正则表达式
//非捕获分组:仅仅是把数据括起来
//特点:不占用组号
//这里\\1报错原因:(?:)就是非捕获分组，此时是不占用组号的。


//(?:) (?=) (?!)都是非捕获分组//更多的使用第一个
//String regex1 ="[1-9]\\d{16}(?:\\d|x|x)\\1";
String regex2 ="[1-9]\\d{16}(\\d Xx)\\1";
//^([01]\d|2[0-3]):[0-5]\d:[@-5]\d$

System.out.println("41080119930228457x".matches(regex2));
```

1.19 正则表达式练习

```
手机号码:1[3-9]\\d{9}
座机号码: 0\\d{2,3}-?[1-9]\\d{4,9}
邮箱号码: \\w+@[\\w&&[^_]]{2,6}(\\.[a-zA-Z]{2,3}){1,2}
24小时: ([01]\\d|2[0-3]):[0-5]\\d:[0-5]\\d
       ([01]\\d|2[0-3])(:[0-5]\\d){2}
用户名:  \\w{4,16}
身份证号码，简单校验:
        [1-9]\\d{16}(\\d|X|x)
        [1-9]\\d{16}[\\dXx]
        [1-9]\\d{16}(\\d(?i)X)
身份证号码，严格校验:
        [1-9]\\d{5}(18|19|20)\\d{2}(0[1-9]|1[0-2])(0[1-9|[12])\\d|3[01])\\d{3}
[\\dXx]
```