

实验内容	第七周实验 字符串处理			成绩	
姓 名	王秋锋	学号	2015111948	班 级	计算机-03 班
专 业	计算机科学与技术			日 期	2017 年 10 月 18 日

【实验目的】--字符串处理

- ◆ 掌握 String、StringBuffer 字符串类
- ◆ 掌握常用的字符串处理类
- ◆ 掌握利用正规式检索字符串

【实验内容】

【实验目的 01】---字符串

一、编辑、编译、运行下面 java 程序

```

public class StringTest {
    public static void main(String[] args) {
        StringTest st = new StringTest();
        st.testString();
    }
    /*
    * 介绍String的常用方法
    */
    public void testString(){
        String str = "This is a test string!";
        System.out.println("\n用于被测试的字符串为: "+str);
        System.out.println("\n方法length()返回字符串的长度为: ");
        int length = str.length();
        System.out.println(String.valueOf(length));
        System.out.println("\n方法equals(Object o)判断字符串内容是否相同，与字符串\"test\"进行比较的结果为: ");
        boolean b = str.equals("test");
        System.out.println(b?"相等":"不相等");
        System.out.println("equalsIgnoreCase(Object o)比较\"test\"与\"Test\"比较的结果:");
        b = "test".equalsIgnoreCase("Test");
        System.out.println(b?"相等":"不相等");
        System.out.println("\ncharAt(int index)方法：某个位置的字符");
        char c = str.charAt(5);
        System.out.println("\n测试字符串第5个位置的字符为: "+String.valueOf(c)+"。注意索引从0开始");
        System.out.println("\n获取子串方法：subString(int fromindex)和substring(int fromindex,int endindex)");
        System.out.println("第一个方法从fromindex开始，第二个方法从fromindex开始到endindex结束的子串，索引从0开始,包含fromindex，不包含endindex。");
        String strSub = str.substring(3);
        System.out.println("str.substring(3)的结果为: "+strSub);
        strSub = str.substring(3,6);
    }
}

```

```
System.out.println("str.substring(3,6)的结果为: "+strSub);
System.out.println("\n去除字符串前面或者后面的空格, 可以使用trim()方法: ");
System.out.println("字符串\"100 01 \"原来的长度为: \""100 01 ".length()+"使用trim()方法之后的长度为: "+"100 01
.trim().length());
printContent("\n\n判断一个字符在字符串中的位置, 可以使用indexOf(int ch)和indexOf(int ch,int from)方法" +
", \n如果不存在返回0, 第一个方法从字符串开头查找, 第二个方法从from为置开始查找, 包含该位置");
int index = str.indexOf('s');
printMethod("str.indexOf('s')");
printResult(String.valueOf(index));
index = str.indexOf('s',6);
printMethod("\nstr.indexOf('s',6)");
printResult(String.valueOf(index));
printContent("\n\n判断一个字符在字符串中的位置, 可以使用indexOf(String str)和indexOf(String str,int from)方法" +
", \n如果不存在返回0, 第一个方法从字符串开头查找, 第二个方法从from为置开始查找, 包含该位置");
index = str.indexOf("is");
printMethod("str.indexOf(\"is\")");
printResult(String.valueOf(index));
index = str.indexOf("is",3);
printMethod("\nstr.indexOf(\"is\",3)");
printResult(String.valueOf(index));
index = str.indexOf("is",7);
printMethod("\nstr.indexOf(\"is\",7)");
printResult(String.valueOf(index));
printContent("\n\n与indexOf方法相似, 还有lastIndexOf方法, 用法基本相同, 不同的是开始查找的位置不同, 一个从前, 一个从后");
printContent("\n\n判断字符串是否以某个子串为后缀, 使用endsWith(String str)");
b = str.endsWith("test");
printMethod("str.endsWith(\"test\")");
printResult(String.valueOf(b));
b = str.endsWith("string!");
printMethod("\nstr.endsWith(\"string!\")");
printResult(String.valueOf(b));
printContent("\n\n与endsWith方法功能相似, startWith(String prefix)和" +
"\n\nstartWith(String prefix,int toffset)用于判断是否以某个子串为前缀");
printContent("\n\n替换字符串中的字符, 使用replace(char oldChar,char newChar)方法");
String str2 = str.replace('s','S');
printMethod("str.replace('s','S')");
printResult(str2);
printContent("\n\n替换字符串中的第一次出现的某个字串, 使用replaceFirst(String oldStr,String newStr)方法");
str2 = str.replaceFirst("is","IS");
printMethod("str.replaceFirst(\"is\",\"IS\")");
printResult(str2);
printContent("\n\n替换字符串中的所有的出现的某个字串, 使用replaceAll(String oldStr,String newStr)方法");
str2 = str.replaceAll("is","IS");
printMethod("str.replaceAll(\"is\",\"IS\")");
printResult(str2);
```

```
printContent("\n\n可以根据某个特定的格式把字符串分成多个子串，使用split方法," +
"\n使用的测试字符串为zhangsan-lisi-wangwu");

str2 = "zhangsan-lisi-wangwu";
String strSplit[] = str2.split("-");
printMethod("str.split(\"-\")");
for(int i=0;i<strSplit.length;i++)
    printResult(strSplit[i]+"");
}
/*
 * 显示注释的内容
 */

public void printContent(String str){
    System.out.println(str);
}
/*
 * 显示代码
 */

public void printMethod(String str){
    System.out.print(str);
    for(int i=0;i<30-str.length();i++)
        System.out.print(" ");
}
/*
 * 显示结果
 */

public void printResult(String str){
    System.out.print(str);
}
}
```

要求:

- (1) 分析该程序，写出运行结果

【实验结果与分析】

初始字符串为"This is a test string!"

1. length()返回字符串的长度

用于被测试的字符串为: This is a test string!

方法length()返回字符串的长度为:
22

2. equals(Object o)判断字符串内容是否相同

显然"This is a test string!"与"test"不相等

equalsIgnoreCase()方法是忽略大小写的比较，所以"test"与"Test"相等

```
方法equals(Object o)判断字符串内容是否相同，与字符串"test"进行比较的结果为：
不相等
equalsIgnoreCase(Object o)比较"test"与"Test"比较的结果：
相等
```

3. 调用 charAt(int index)函数获得某个位置的字符

```
charAt(int index)方法：某个位置的字符
```

```
测试字符串第5个位置的字符为：i。注意索引从0开始
```

4. 调用 subString 函数获取子串

```
获取子串方法：subString(int fromindex)和substring(int fromindex,int endindex)
第一个方法从fromindex开始，第二个方法从fromindex开始到endindex结束的子串，索引从0开始，包含fromindex，不包含endindex
str.substring(3)的结果为：s is a test string!
str.substring(3,6)的结果为：s i
```

- 5.使用 trim()方法去除字符串前面或者后面的空格

```
去除字符串前面或者后面的空格，可以使用trim()方法：
字符串"100 01 "原来的长度为：7使用trim()方法之后的长度为：6
```

6. 使用 indexOf 方法判断一个字符在字符串中的位置

```
判断一个字符在字符串中的位置，可以使用indexOf(int ch)和indexOf(int ch,int from)方法，
如果不存在返回0，第一个方法从字符串开头查找，第二个方法从form为置开始查找，包含该位置
str.indexOf('s')3
str.indexOf('s',6)6
```

```
判断一个字符在字符串中的位置，可以使用indexOf(String str)和indexOf(String str,int from)方法，
如果不存在返回0，第一个方法从字符串开头查找，第二个方法从form为置开始查找，包含该位置
str.indexOf("is")2
str.indexOf("is",3)5
str.indexOf("is",7)-1
与indexOf方法相似，还有lastIndexOf方法，用法基本相同，不同的是开始查找的位置不同，一个从前，一个从后
```

7. 使用 endsWith 方法判断字符串是否以某个子串为后缀

```
判断字符串是否以某个子串为后缀，使用endsWith(String str)
str.endsWith("test")false
str.endsWith("string!")true
```

```
与endsWith方法功能相似，startsWith(String prefix)和
startsWith(String prefix,int toffset)用于判断是否以某个子串为前缀
```

8. 使用 replace 方法替换字符串中的字符

替换字符串中的字符，使用`replace(char oldChar,char newChar)`方法
`str.replace('s','S')`ThiS iS a teSt String!

替换字符串中的第一次出现的某个字串，使用`replaceFirst(String oldStr,String newStr)`方法
`str.replaceFirst("is","IS")`ThiS is a test string!

替换字符串中的所有的出现的某个字串，使用`replaceAll(String oldStr,String newStr)`方法
`str.replaceAll("is","IS")`ThiS IS a test string!

9. 使用 `split` 方法根据某个特定的格式把字符串分成多个子串

可以根据某个特定的格式把字符串分成多个子串，使用`split`方法，
使用的测试字符串为`zhangsan-lisi-wangwu`
`str.split("-")`zhangsanlisiwangwu

二、程序填空

按模板要求，将【代码 1】～【代码 09】替换为 Java 程序代码。

StringExample.java

class StringExample

{

 public static void main(String args[])

 {

 String s1=new String("you are a student"),

 s2=new String("how are you");

 if (【代码 1】) // 判断 s1 与 s2 是否相同

 {

 System.out.println("s1 与 s2 相同");

 }

 else

 {

 System.out.println("s1 与 s2 不相同");

 }

 String s3=new String("51010019851022024");

 if (【代码 2】) // 判断 s3 的前缀是否是“510100”

 {

 System.out.println("四川省的身份证");

 }

 String s4=new String("你"),

 s5=new String("我");

 if(【代码 3】) // s4 大于 s5

 {

 System.out.println("s4 大于 s5");

 }

 else

 {

 System.out.println("s4 小于 s5");

```
    }
    int position=0;
    String path="c:\\java\\jsp\\A.java";
    position= 【代码 4】 // 获取 path 中最后出现目录分隔符号的位置
    System.out.println("c:\\java\\jsp\\A.java 中最后出现\\的位置:"+position);
    String fileName= 【代码 5】 // 获取 path 中“A.java”子字符串
    System.out.println("c:\\java\\jsp\\A.java 中含有的文件名:"+fileName);
    String s6=new String("100"),
        s7=new String("123.678");
    int n1= 【代码 6】 // 将 s6 转化成 int 型数据
    double n2= 【代码 7】 // 将 s7 转化成 double 型数据
    double n=n1+n2;
    System.out.println(n);
    String s8=new String("ABCDEF");
    char a[ ]= 【代码 8】 // 将 s8 存放到数组 a 中
    for(int i=a.length-1;i>=0;i--)
    {
        【代码 9】 // 打印 a[i]
    }
}
```

要求：（1）给出所缺的 Java 语句

（2）运行程序，给出正确的程序运行结果

【实验结果与分析】

完整代码：

```
public class StringExample
{
    public static void main(String args[ ])
    {
        String s1=new String("you are a student"),
            s2=new String("how are you");
        if (s1.equals(s2)) // 判断s1与s2是否相同
        {
            System.out.println("s1与s2相同");
        }
        else
        {
            System.out.println("s1与s2不相同");
        }
        String s3=new String("51010019851022024");
        if (s3.startsWith("510100")) // 判断s3的前缀是否是“510100”
```

```

{
    System.out.println("四川省的身份证");
}
String s4=new String("你"),
s5=new String("我");
if(s4.compareTo(s5) > 0)                // s4大于s5
{
    System.out.println("s4大于s5");
}
else
{
    System.out.println("s4小于s5");
}
int position=0;
String path="c:\\java\\jsp\\A.java";
position=path.lastIndexOf("\\");        // 获取path中最后出现目录
分隔符号的位置
System.out.println("c:\\java\\jsp\\A.java中最后出现\\的位
置:"+position);
String fileName=path.substring(position+1);    // 获取path中
“A.java”子字符串
System.out.println("c:\\java\\jsp\\A.java中含有的文件名:"+fileName);
String s6=new String("100"),
s7=new String("123.678");
int n1=Integer.parseInt(s6);                // 将s6转化成int型数据
double n2=Double.parseDouble(s7);           // 将s7转化成double型
数据
double n=n1+n2;
System.out.println(n);
String s8=new String("ABCDEF");
char a[] =s8.toCharArray();                // 将s8存放到数组a中
for(int i=a.length-1;i>=0;i--)
{
    System.out.print(a[i]);
}
}
}

```

运行结果:

```

s1与s2不相同
四川省的身份证
s4小于s5
c:\java\jsp\A.java中最后出现\的位置:11
c:\java\jsp\A.java中含有的文件名:A.java
223.678
FEDCBA

```

三、编辑并运行下面程序，理解正规式的使用

```
/*身份号码验证*/

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

public class RegTest
{
    static void test()
    {
        Pattern p = null; //正则表达式
        Matcher m = null; //操作的字符串
        boolean b = false;
        //正则表达式表示 15 位或者 18 位数字的一串数字
        p = Pattern.compile("\\d{15}\\d{18}");
        m = p.matcher("120101198506020080");
        b = m.matches();
        System.out.println("身份号码正确: "+b); //输出: true
        p = Pattern.compile("\\d{15}\\d{18}");
        m = p.matcher("020101198506020080"); //错误 首位为 0
        b = m.matches();
        System.out.println("身份号码错误: "+b); //输出: false
    }
    public static void main(String argus[])
    {
        test();
    }
}
```

要求：运行程序，给出正确的程序运行结果，理解正规式的使用。

【实验结果与分析】

修改后代码：

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegTest{
    static void test(){
        Pattern p = null; //正则表达式
        Matcher m = null; //操作的字符串
        boolean b = false;
        //正则表达式表示15位或者18位数字的一串数字
        p = Pattern.compile("[^0](\\d{14}|\\d{17})");
        m = p.matcher("120101198506020080");
    }
}
```



```

    b = m.matches();
    p = Pattern.compile("[^0](\\d{14}|\\d{17})");
    System.out.println("身份证号码正确: "+b); //输出: true
    m = p.matcher("020101198506020080");//错误 首位为0
    b = m.matches();
    System.out.println("身份证号码错误: "+b); //输出: false
}
public static void main(String argus[]){
    test();
}
}

```

这个`[^0](\\d{14}|\\d{17})`正则表达式可以处理首字不以 0 开头的身份证号
运行结果:

```

身份证号码正确: true
身份证号码错误: false

```

四、编写程序

(1) 程序 1: 从一字符串中 (测试字符串中可包含多个 email 地址), 利用正规式处理功能, 检索出所有的 EMAIL 地址。

源程序代码

```

import java.util.regex.*;
public class Tester {
    public static void main (String[] args) {
        String str = "cyberdebut@gmail.com15528330350@qq.com";
        String reg = "([a-z0-9_\\.-]+)@([\\da-z\\.\\.-]+)\\.([a-z\\.]{2,6})";
        Pattern pattern = Pattern.compile (reg);
        Matcher matcher = pattern.matcher (str);
        System.out.println ("输出所有的email地址: ");
        while (matcher.find()) { System.out.println (matcher.group()); }
    }
}

```

【实验结果与分析】

```

输出所有的email地址:
cyberdebut@gmail.com
15528330350@qq.com

```

四、编写程序

(2) 程序 2: 编写程序, 利用正规式处理功能, 给定一字符串, 判定其是否为 11 位的手机号

码（包括符合目前中国电信、移动与联通的所有号码都要给出判定）

源程序代码

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

public class Mobile{
    public static int matchesPhoneNumber(String phone_number) {
        String cm =
"^(13[4-9])|(147)|(15[0-2,7-9])|(17[8])|(18[2-4,7-8])\\d{8}|(170[5])\\d{7}$";
        String cu =
"^(13[0-2])|(145)|(15[5-6])|(17[156])|(18[5,6])\\d{8}|(170[4,7-9])\\d{7}$";
        String ct =
"^(133)|(149)|(153)|(17[3,7])|(18[0,1,9])\\d{8}|(170[0-2])\\d{7}$";
        int flag = 0;
        if (phone_number.matches(cm)) {
            flag = 1;
        } else if (phone_number.matches(cu)) {
            flag = 2;
        } else if (phone_number.matches(ct)) {
            flag = 3;
        } else {
            flag = 4;
        }
        return flag;
    }

    public static void whichOperator(int x){
        switch(x){
            case 1 :
                System.out.println("移动号码");
                break;
            case 2:
                System.out.println("联通号码 ");
                break;
            case 3 :
                System.out.println("电信号码");
                break;
            case 4:
                System.out.println("输入有误");
                break;
            default: System.out.println("输入有误");
        }
    }
}
```

```

    }

    public static void main(String[] args) {
        System.out.println("请输入电话号码: ");
        Scanner sc = new Scanner(System.in);
        String e = sc.next();
        System.out.println("匹配结果: ");
        whichOperator(matchesPhoneNumber(e));
    }
}

```

【实验结果与分析】

```

请输入电话号码:
15528330350
匹配结果:
联通号码

```

四、编写程序

(3) 程序 3: 从键盘输入多行文本并作处理:

- A、显示各元音字母出现次数;
- B、输出各个单词以及单词的长度。

源程序代码

```

import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Calc {
    public static void main(String[] args) {
        System.out.println("请输入你要处理的文本: ");
        String regex = "(a|e|i|o|u)";
        String word = "[a-zA-z]{1,}";
        Scanner sc = new Scanner(System.in);
        String chap = sc.nextLine();
        Pattern p1 = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
        Pattern p2 = Pattern.compile(word, Pattern.CASE_INSENSITIVE);
        Matcher m = p1.matcher(chap);
        Matcher n = p2.matcher(chap);
        int end = 0;
        int[] a = {0,0,0,0,0};
        int i = 0;
        while(m.find(end)) {
            if(m.group().equals("a")||m.group().equals("A"))a[0]++;
            else if(m.group().equals("e")||m.group().equals("E"))a[1]++;
            else if(m.group().equals("i")||m.group().equals("I"))a[2]++;
            else if(m.group().equals("o")||m.group().equals("O"))a[3]++;

```

```
        else if(m.group().equals("u")||m.group().equals("U"))a[4]++;
        end = m.end();
    }
    end = 0;
    System.out.println("\n输出单词表: ");
    while(n.find(end)) {
        System.out.println("单词:\t"+n.group()+"\t长度:"+n.group().length());
        end = n.end();
    }
    System.out.println("\n元音字母数目表: ");
    System.out.println("a\\A:\t"+a[0]);
    System.out.println("e\\E:\t"+a[1]);
    System.out.println("i\\I:\t"+a[2]);
    System.out.println("o\\O:\t"+a[3]);
    System.out.println("u\\U:\t"+a[4]);
}
}
```

【实验结果与分析】

使用正则匹配所有的元音字母，并且处理出所有的单词

实验结果：



```
请输入你要处理的文本：
hello java 2017, next time

输出单词表：
单词：    hello    长度：5
单词：    java     长度：4
单词：    next     长度：4
单词：    time     长度：4

元音字母数目表：
a\\A:    2
e\\E:    3
i\\I:    1
o\\O:    1
u\\U:    0
```

【实验目的 02】---包装类

- ◆ 掌握日期相关类；

◆ 掌握 Math 相关类等

【实验内容】

1、运行下面程序，给出程序运行结果

```
public class IntegerTest {
    public static void main(String[] args) {
        System.out.println("Integer中的常量*****");
        System.out.println("Integer的最大取值: \t"+Integer.MAX_VALUE);
        System.out.println("Integer的最小取值: \t"+Integer.MIN_VALUE);
        System.out.println("Integer的最大位数: (以二进制补码形式表示 int 值的位数。)\t"+Integer.SIZE);
        System.out.println("Integer的类型的: \t"+Integer.TYPE);

        System.out.println();
        System.out.println("*****Integer中方法的使用*****");
        int i=12345;
        System.out.println("12345的二进制表示: \t"+Integer.toBinaryString(i));
        System.out.println("12345的二进制串中“1”的总数量: \t"+Integer.bitCount(i));
        /**
         * numberOfLeadingZeros计算方法为: 32(Integer.SIZE)-Integer.toBinaryString(12345).length()
         */
        System.out.println("12345的二进制串中从最左边算起连续的“0”的总数量: \t"+Integer.numberOfLeadingZeros(i));
        System.out.println("12345的二进制串中从最右边算起连续的“0”的总数量: \t"+Integer.numberOfTrailingZeros(i));
        /**
         * Integer decode(String nm)
         * 给定一个10进制, 8进制, 16进制中任何一种进制的字符串,
         * 该方法可以将传入的字符串转化为10进制数字的Integer类型并返回。
         */
        System.out.println("8的八进制为010, 转换为10进制: \t"+Integer.decode("010"));
        System.out.println("10的十进制为10, 转换为10进制: \t"+Integer.decode("10"));
        System.out.println("16的十六进制 为0X10, 转换为10进制: \t"+Integer.decode("0X10"));
        System.out.println("1000反转整数二进制补码的位顺序: \t"+Integer.reverse(i));
        System.out.println("1000反转整数字节的顺序: \t"+Integer.reverseBytes(i));
        /**
         * 获取整数符号, 为负返回-1, 正返回1, 零返回0
         */
        System.out.println("12345获取整数符号为: \t"+Integer.signum(i));
        System.out.println("创建12345的Integer对象: \t"+Integer.valueOf(i));
        System.out.println("Integer.valueOf对象的使用(12345的radix进制数): \t"+Integer.valueOf("12345", 10));

        System.out.println();
        System.out.println("*****Integer对象的方法使用*****");
        Integer obj=new Integer(1000);
        System.out.println("1000转换为byte类型的数为: \t"+obj.byteValue());
        System.out.println("Integer1000和Integer2000大小比较: \t"+obj.compareTo(new Integer(2000)));
        System.out.println("Integer2000和Integer1000大小比较: \t"+new Integer(2000).compareTo(obj));
    }
}
```

```
System.out.println("Integer1000转换为double类型的数为: \t"+obj.doubleValue());
System.out.println("Integer1000和Integer2000大小比较: \t"+obj.equals(new Integer(2000)));
System.out.println("Integer2000和Integer1000大小比较: \t"+new Integer(2000).equals(obj));
System.out.println("Integer2000和Integer1000大小比较: \t"+new Integer(2000).equals(new Integer(2000)));
System.out.println("Integer1000的哈希码: \t"+obj.hashCode());
System.out.println("Integer1000的int值: \t"+obj.intValue());
System.out.println("Integer1000的long值: \t"+obj.longValue());
System.out.println("Integer1000的short值: \t"+obj.shortValue());
System.out.println("将字符串1000解析为int类型的数: \t"+Integer.parseInt("1000"));
/**
 * Integer.parseInt("1000", 2)
 * 返回第一个参数的(字符串)的2进制(参数2为转换的进制)
 */
System.out.println("返回1000的2进制"+Integer.parseInt("1000", 2));
/**
 * 进制转换
 */
System.out.println("1000十进制转成二进制"+Integer.toBinaryString(i));
System.out.println("1000十进制转八进制: \t"+Integer.toOctalString(i));
System.out.println("1000十进制转十六进制: \t"+Integer.toHexString(i));
System.out.println("十六进制转成十进制:\t"+Integer.valueOf("FFFF",16).toString());
System.out.println("十六进制转成二进制:\t"+Integer.toBinaryString(Integer.valueOf("FFFF",16)));
System.out.println("十六进制转成八进制:\t"+Integer.toOctalString(Integer.valueOf("FFFF",16)));

System.out.println("八进制转成十进制:\t"+Integer.valueOf("576",8).toString());
System.out.println("八进制转成二进制:\t"+Integer.toBinaryString(Integer.valueOf("23",8)));
System.out.println("八进制转成十六进制:\t"+Integer.toHexString(Integer.valueOf("23",8)));

System.out.println("二进制转十进制:\t"+Integer.valueOf("0101",2).toString());
System.out.println("二进制转八进制:\t"+Integer.toOctalString(Integer.parseInt("0101", 2)));
System.out.println("二进制转十六进制:\t"+Integer.toHexString(Integer.parseInt("0101", 2)));

System.out.println();
System.out.println("1000的二进制形式最左边的最高一位且高位后面全部补零, 最后返回int型的结果
"+Integer.highestOneBit(i));

}

}
```

要求: (1) 运行程序, 给出正确的程序运行结果
(2) 理解掌握包括 Integer 类的包装类。

【实验结果与分析】

Java 中的 Integer 是 int 的包装类型

运行结果:

```
Integer中的常量*****
Integer的最大取值: 2147483647
Integer的最小取值: -2147483648
Integer的最大位数: (以二进制补码形式表示 int 值的位数。) 32
Integer的类型的: int

*****Integer中方法的使用*****
12345的二进制表示: 11000000111001
12345的二进制串中“1”的总数里: 6
12345的二进制串中从最左边算起连续的“0”的总数里: 18
12345的二进制串中从最右边算起连续的“0”的总数里: 0
8的八进制为010, 转换为10进制: 8
10的十进制为10, 转换为10进制: 10
16的十六进制为0X10, 转换为10进制: 16
1000反转整数二进制补码的位顺序: -1676935168
1000反转整数字节的顺序: 959447040
12345获取整数符号为: 1
创建12345的Integer对象: 12345
Integer.valueOf对象的使用(12345的radix进制数): 12345
```

```

*****Integer对象的方法使用*****
1000转换为byte类型的数为:    -24
Integer1000和Integer2000大小比较:    -1
Integer2000和Integer1000大小比较:    1
Integer1000转换为double类型的数为:    1000.0
Integer1000和Integer2000大小比较:    false
Integer2000和Integer1000大小比较:    false
Integer2000和Integer1000大小比较:    true
Integer1000的哈希码:    1000
Integer1000的int值:    1000
Integer1000的long值:    1000
Integer1000的short值:    1000
将字符串1000解析为int类型的数:    1000
返回1000的2进制8
1000十进制转成二进制11000000111001
1000十进制转八进制:    30071
1000十进制转十六进制:    3039
十六进制转成十进制:    65535
十六进制转成二进制:    111111111111111
十六进制转成八进制:    177777
八进制转成十进制:    382
八进制转成二进制:    10011
八进制转成十六进制:    13
二进制转十进制:    5
二进制转八进制:    5
二进制转十六进制:    5

1000的二进制形式最左边的最高一位且高位后面全部补零，最后返回int型的结果8192

```

补充内容

1. 编写输出回文的程序，即字符串颠倒顺序后再输出。

提示：运行结果参考如下。



图 运行界面

部分程序参考如下（也可不用 StringBuffer 类，即都用 String 类实现，但代价高）。

```

public class Palindrome {
    public static void main(String[] args) {
        String str1 = "僧游云隐寺"; //字符串常量类对象
        String str2 = reverse(str1); //调用串倒转方法
        System.out.println(str1 + "," + str2);
        str1 = "人过大佛寺";
    }
}

```



```
...
}

public static String reverse(String s){ //串倒转方法
    int len = s.length(); //串常量长度
    StringBuffer sb = new StringBuffer(); //字符串变量类对象
    for (int i=...; ...; ... ) {
        sb.append(...); //串变量对象追加串常量的第 i 个字符 s.charAt(i)
    }
    return sb.toString();
}
}
```

要求：把上面的程序补充完整并给出程序运行结果

【实验结果与分析】

完整代码：

```
public class Palindrome {
    public static void main(String[] args) {
        String str1 = "僧游云隐寺"; //字符串常量类对象
        String str2 = reverse(str1); //调用串倒转方法
        System.out.println(str1 + "," + str2);
        str1 = "人过大佛寺";
        str2 = reverse(str1);
        System.out.println(str1 + "," + str2);
    }
    public static String reverse(String s){ //串倒转方法
        int len = s.length(); //串常量长度
        StringBuffer sb = new StringBuffer(); //字符串变量类对象
        for (int i =len-1; i>=0; i-- ) {
            sb.append(s.charAt(i)); //串变量对象追加串常量的第i个字符s.charAt(i)
        }
        return sb.toString();
    }
}
```

运行结果：

```
僧游云隐寺,寺隐云游僧
人过大佛寺,寺佛大过人
```

2. 编写程序，调用上题的回文方法，把数字字符串"123456789"变成"987654321"，并输出这两个字符串连接后的结果，以及其对应整数相加的结果。

提示：运行结果参考如下。

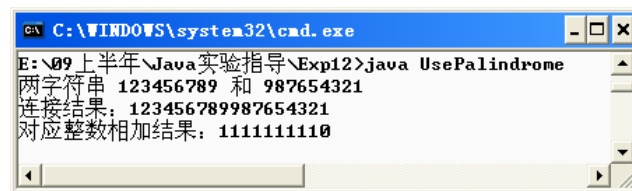


图 运行界面

部分程序参考如下。

```
public class UsePalindrome {  
    public static void main(String[] args) {  
        String str1 = "123456789";  
        String str2 = Palindrome.reverse(str1);  
        int i1 = Integer.parseInt(str1);  
        int i2 = ...  
        ...  
    }  
}
```

要求：把上面的程序补充完整并给出程序运行结果

【实验结果与分析】

完整代码：

```
public class UsePalindrome {  
    public static void main(String[] args) {  
        String str1 = "123456789";  
        String str2 = Palindrome.reverse(str1);  
        int i1 = Integer.parseInt(str1);  
        int i2 = Integer.parseInt(str2);  
        System.out.println("两字符串 " + str1 + " 和 " + str2);  
        System.out.println("连接结果: " + str1 + str2);  
        int i3=i1+i2;  
        System.out.println("对应整数相加结果: "+i3);  
    }  
}
```

运行结果：

```
两字符串 123456789 和 987654321  
连接结果: 123456789987654321  
对应整数相加结果: 111111110
```

3. 编写程序，将一串以逗号分隔的数字字符串分割并转换成若干个单独的数值，再进行累加运算，并输出结果。

提示：运行结果参考如下。

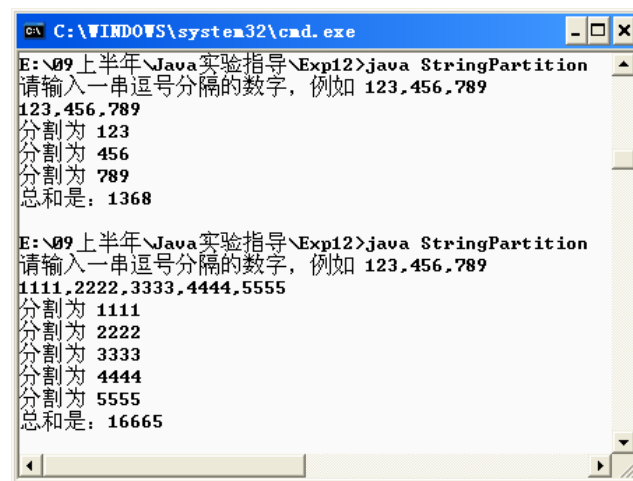


图 运行界面

提示：部分程序参考如下（也可用 `String` 类结合 `StringBuffer` 类完成本题）。

```
import java.io.*;
public class StringPartition {
    public static void main(String[] args) {
        try {
            BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
            System.out.println("请输入一串逗号分隔的数字，例如 123,456,789");
            String str = in.readLine();
            String s = "";
            int sum = 0;
            for (int i = 0; i<...; ...) {
                if (str.charAt(i) != ',') {
                    s += ...;    //字符串 s 连接字符串 str 的第 i 个字符
                }
                else {
                    System.out.println("分割为 " + s);
                    sum += Integer.parseInt(s);
                    s = "";
                }
            }
            ...
            System.out.println("总和是: " + sum);
        }
        catch(Exception e){
            System.out.println("输入异常: " + e);
        }
    }
}
```

要求：（1）把上面的程序补充完整并给出程序运行结果

（2）使用 `String` 类的 `split` 方法：`String[] split(String regex)`
根据给定的正则表达式 `regex` 的匹配来拆分此字符串，编写程序完成

【实验结果与分析】

（1）完整程序：

```
import java.io.*;
public class StringPartition {
```

```

public static void main(String[] args) {
    try {
        BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
        System.out.println("请输入一串逗号分隔的数字，例如 123,456,789");
        String str = in.readLine();
        String s = ""; int sum = 0;
        for (int i = 0; i<str.length(); i++) {
            if (str.charAt(i) != ',') {
                s += str.charAt(i);    //字符串s连接字符串str的第i个字符
            }
            else {
                System.out.println("分割为 " + s);
                sum += Integer.parseInt(s);
                s = "";
            }
        }
        sum += Integer.parseInt(s);
        System.out.println("分割为 " + s);
        System.out.println("总和是: " + sum);
    }
    catch(Exception e){
        System.out.println("输入异常: " + e);
    }
}
}

```

运行结果:

```

请输入一串逗号分隔的数字，例如 123,456,789
123,456,789
分割为 123
分割为 456
分割为 789
总和是: 1368

```

(2) 完整程序:

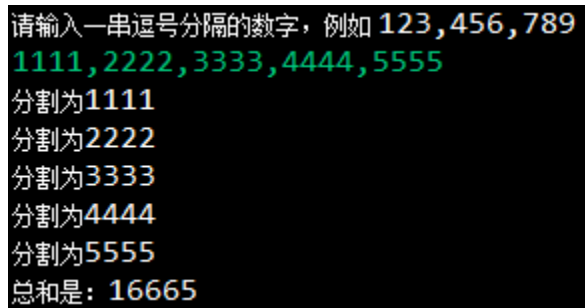
```

import java.io.*;
public class StringPartition {
    public static void main(String[] args) {
        try {
            BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("请输入一串逗号分隔的数字，例如 123,456,789");
            String str = in.readLine();
            String strSplit[] = str.split(",");
            int sum = 0;
            for(int i=0;i<strSplit.length;i++){
                System.out.println("分割为"+strSplit[i]);
            }
        }
    }
}

```

```
        sum += Integer.parseInt(strSplit[i]);
    }
    System.out.println("总和是: " + sum);
}
catch(Exception e){
    System.out.println("输入异常: " + e);
}
}
```

运行结果:



```
请输入一串逗号分隔的数字，例如 123,456,789
1111,2222,3333,4444,5555
分割为1111
分割为2222
分割为3333
分割为4444
分割为5555
总和是: 16665
```