字符串、IO流和数组的基本操作

内容

- 字符串
- 数组
- 输入输出流

字符串

Java的字符串类型为String，它不属于基本数据类型。字符串在Java中被定义为对象，它的默认初始值为null。通过双引号来表示。

字符串 ≠ 字符

字符串

1. 声明字符串

```
String str;
```

2. 创建字符串

```
str = new String("Hello World");
```

另一种方式：

```
str = "Hello World";
```

3. 声明和创建可以一步完成

```
String str = "Hello World";
```

字符串

直接赋值创建和new String创建的区别:

String str = "hello world";（直接赋值）

如果hello world这个字符串在string池中已经存在，那么会直接把它的引用赋予str，不会再另外创建对象。

String str = new String("hello world");

无论hello world这个字符串是否已经存在于string池中，都会再创建一个对象。

一般建议使用直接赋值的方法来创建字符串。

字符串

例子:

String str1 = new String("abc");

String str2 = new String("abc");

str1 == str2的返回值为*false*

String str1 = "abc";

String str2 = "abc";

str1 == str2的返回值为*true*

字符串

方法	说明
int length()	返回字符串的长度
String toUpperCase()	将串中字符变成大写
String toLowerCase()	将串中字符变成小写
char charAt(int i)	返回位置i处的字符
String substring(int s,int e)	返回从位置s到e的字符子串
String substring(int s)	返回从位置s到末尾的字符子串
int indexOf(String s)	返回首次出现字符串s的位置
int indexOf(String s,int i)	返回在位置i之后首次出现s的位置
String trim()	返回一个新串，去除前后空白字符
String replace(String a,String b)	返回一个新串，将a替换为b

字符串

字符串的不可变性

Java中所有String都是final类型的，都是不可变的。

举例说明：

```
String str = "abc";

str = str + "d";

System.out.println(str);
```

输出的结果是abcd，那么str不是改变了吗？

其实不可变指的是内存不可变，字符串"abc"依然存在，
我们并没有对它进行修改，只是新建了一个更大的数组
进行了扩展，此时，字符串池中既有"abc"，又有"abcd"。

数组

数组：一组有序数据的集合。每个元素具有相同的数据类型，可以用一个统一的数组名和下标来唯一的确定数组中的元素。

数组类型：基本类型、引用类型。

数组可以按照使用分为一维数组、二维数组和多维数组。

数组

数组的定义形式：

数组类型 数组名[];

或是

数组类型[] 数组名;

如int i[]; double[] d; String s[];

数组的创建：

方法一：采用new 操作符

i = new int[10]; //int类型默认值为0

方法二:静态的初始化器

i = {1,2,3,4,5};

数组

可以在声明数组时直接初始化，如：

```
int[] i = new int[5];
```

或是

```
int[] i = {1,2,3,4,5};
```

一旦数组被创建了，那么数组对象的长度就不能再被改变。数组元素数组名和下标组成，下标的下界为0，上界为数组元素个数减一。

多维数组

二维数组的定义形式：

数组类型 多维数组名[][];

或是

数组类型[][] 多维数组名

例如：int[][] array; int array2[][];

二维数组的创建（分配内存）：

方法一：直接为每一维分配空间，如：

int a[][] = new int[2][3];

多维数组

方法二：从最高维开始，分别为每一维分配空间，如：

```
int a[][] = new int[2][];
a[0] = new int[3];
a[1] = new int[3];
```

多维数组初始化方法：

(1) 直接对每个元素进行赋值

(2) 在定义数组的同时进行初始化，如：

```
int a[][] = {{1,2},{3,4},{5,6}};
```

文件处理

文件是信息的一种组织形式，是存储在外部存储介质上具有标识名的一组相关信息集合。

文件可分为文本文件和二进制文件。

文件处理

File类

File类对象主要用来获取文件本身的一些信息，如文件所在的目录、文件长度、文件读写权限等。

File类的主要方法

方法定义	方法说明
public File(String pathname)	通过文件名创建一个file实例
public boolean canRead()	判断文件是否可读
public boolean canWrite()	判断文件是否可写
public boolean exits()	判断文件或目录是否存在

文件处理

方法定义	方法说明
public long length()	获取文件长度
public String getName()	获取文件名字，不包含路径
public String getAbsolutePath()	获取文件的绝对路径
public boolean isFile()	判断是否是一个文件
public boolean isDirectory()	判断是否是一个目录
public Boolean mkdir()	创建一个目录
public boolean createNewFile()	创建新文件
public boolean delete()	删除文件或空目录
public boolean setReadOnly()	设置文件属性为只读

输入输出流

流：一组有顺序的、有起点和终点的字节集合，是对数据传输的总称。也就是说，数据在两个对象之间的传输称为流。

按照流的传输方法，可以分为输入流和输出流。

程序 -> 文件　输出

文件 -> 程序　输入

流的基本操作有读操作和写操作，从流中取得数据的操作称为读操作，向流中添加数据的操作称为写操作。对输入流只能进行读操作，对输出流只能进行写操作

从流的内容上划分，可以分为字节流和字符流。字节流是一个字节序列。字符流是一个字符序列。

输入输出流

字节流由InputStream和OutputStream处理，而字符流由Reader和Writer处理。

常用类(字节流)

输入流	输出流	说明
FileInputStream	FileOutputStream	读写文件
DataInputStream	DataoutputStream	可以读写具体类型的数据
BufferedInputStream	BufferedOutputStream	使用缓冲区，提高读写效率

输入输出流

常用类(字符流)

输入流	输出流	说明
InputStreamReader	OutputStreamWriter	读写数据，可以指定编码
FileReader	FileWriter	文件读写
BufferedReader	BufferedWriter	使用缓冲区，提高读写效率

以字节流为例的文件写操作：

```
public static void fileOutput() throws IOException{
        String str = "hello world!";
        File file = new File("d:\\test.txt");   //创建file对象
        if(!file.exists()){
            file.createNewFile();     //如果文件不存在，则进行创建
         }
        FileOutputStream fOutput = new FileOutputStream(file);
        BufferedOutputStream bOutput = new BufferedOutputStream(fOutput);
        byte[] buffer = str.getBytes(); //将字符串文本转换成字节数组
        bOutput.write(buffer);
        bOutput.close();
        fOutput.close();
}
```

以字节流为例的文件读操作：

```
public static void fileInput() throws IOException{
        File file = new File("d:\\test.txt");    //创建file对象
        FileInputStream fIutput = new FileInputStream(file);
        BufferedInputStream bIutput = new BufferedInputStream(fIutput);
        int temp = 0;
        while((temp = bIutput.read())!= -1){    //当temp为-1时，数据读取完毕
                System.out.print((char)temp);
        }
        bIutput.close();
        fIutput.close();
}
```

以字符流为例的文件写操作：

```
public static void fileWriter() throws IOException{
        String str = "hello world!";
        File file = new File("d:\\test.txt");
       if(!file.exists()){
            file.createNewFile();     //如果文件不存在，则进行创建
        }
        FileWriter fwWriter = new FileWriter(file);
        BufferedWriter bWriter = new BufferedWriter(fwWriter);
        bWriter.write(str);
        bWriter.close();
        fwWriter.close();
}
```

以字符流为例的文件读操作：

```
public static void fileReader() throws IOException{
        File file = new File("d:\\test.txt");
        FileReader fReader = new FileReader(file);
        BufferedReader buReader = new BufferedReader(fReader);
        String temp = null;
        while((temp = buReader.readLine()) != null){
                System.out.println(temp);
        }
        buReader.close();
        fReader.close();
}
```

输入输出流

上述两种方法都能实现将文本"hello world"写入到D盘test.txt文件中，并读取该文件文本内容并打印出来的功能。

注意：

(1)在做IO处理需要引入必要的IO包，在程序顶部加上

```
import java.io.*;
```

(2)在java中进行输入输出流操作时必须进异常处理，可以使用throws抛出IOException。

(3)对于流对象的操作，使用完毕后，一定要调用其close()方法将其释放掉。