字符串，数组和文件的基本操作

傅宇飞
51215901011@stu.ecnu.edu.cn

实验内容

- 字符串
- 数组
- 文件I/O

字符串

- Java库中预定义了**字符串（String）类**。String类型不是基本类型，而是引用类型。
  - Java中一共定义了8中基本类型：字节型(byte)、短整型(short)、整型(int)、长整型(long)、字符型(char)、浮点型(float)、双精度型(double)、布尔型(boolean)。

字符串

- 声明：

```
String str;
```

- 创建：

```
String str = "Welcome to Java";
```

也可以通过构造函数的方式创建：

```
String str = new String("Welcome to Java");
```

字符串

- **直接赋值创建和构造函数创建的区别：**
  - 直接赋值创建的字符串存储在常量池中，而构造函数创建的字符串对象则存储在堆中。
  - 对于内容相同的字符串，在常量池中永远只有一份，在堆中有多份。

字符串

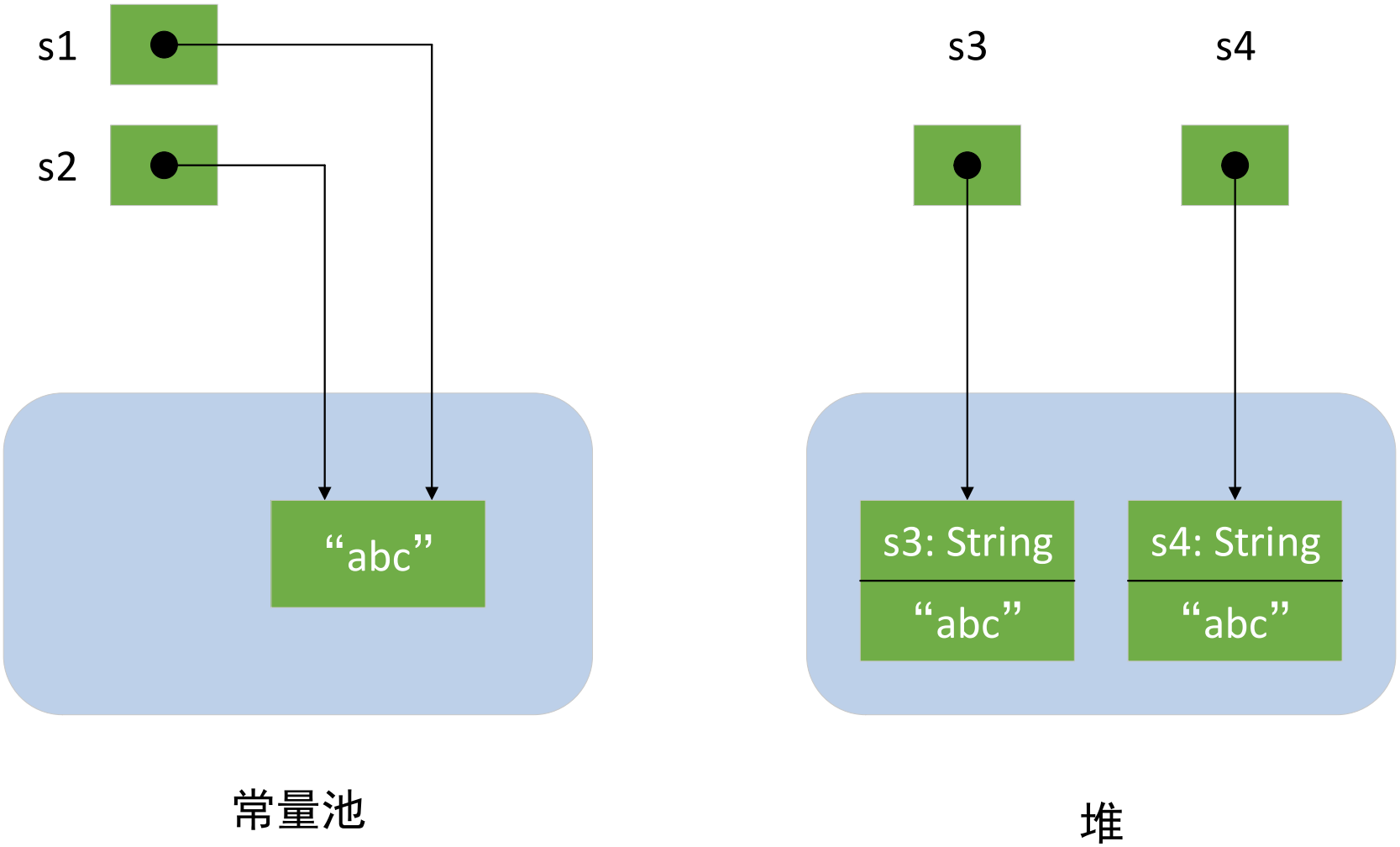
- **字符串的比较：**

  操作符==只能检测s1和s2是否指向同一个对象，不能检测其内容是否相同。

```
String s1 = "abc";
String s2 = "abc";
System.out.println(s1 == s2); // true

String s3 = new String("abc");
String s4 = new String("abc");
System.out.println(s3 == s4); // false
```

字符串



字符串

- **字符串的比较：**

  应该使用equals方法比较两个字符串是否相同

```
String s1 = "abc";
String s2 = "abc";
System.out.println(s1.equals(s2)); // true

String s3 = new String("abc");
String s4 = new String("abc");
System.out.println(s3.equals(s4)); // true
```

字符串

- 字符串对象的简单方法

方法	描述
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 index)	返回位置index处的字符
String substring(int s, int e)	返回从位置s到e的字符子串[s,e)
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

- 求字符串长度：

```
String str = "Welcome to Java";
System.out.println(str.length());  // 15
```

字符串

- 字符串的转换：

```
String str = "Welcome";
System.out.println(str.toUpperCase());  // WELCOME

String str = "Welcome";
System.out.println(str.toLowerCase());  // welcome
```

字符串

- 字符串的转换：

```
String str = "\t  Good    Night   \n";
System.out.println(str.trim());  // 输出字符串Good Night
```

数组

- **数组（Array）**是存储数据的集合。Java 语言中提供的数组用来存储固定大小的同类型元素。
- 为了在程序中使用数组，必须声明一个引用数组的变量，并指明数组的元素类型：

```
type[] array;
```

或者

```
type array[];
```

数组

- **数组的初始化：**

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

或者

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

- 数组一旦被创建，其长度就不能改变了。
- 数组元素由数组名和下标组成，下标的下界为0，上界为数组元素个数减一。

多维数组

- 二维数组的声明：

```
type[][] array;
```

或者

```
type array[][];
```

多维数组

- 二维数组的初始化：

```
int[][] array = new int[2][3];
```

或者

```
int[][] array = new int[2][];
// 从最高维开始，为每一维分配空间
array[0] = new int[3];
array[1] = new int[2];
```

多维数组

- **二维数组的赋值：**
  1. 直接对每个元素进行赋值，例如

     ```
     array[2][3] = 7;  // 将7赋值给行下标为2、列下标为1的特定元素
     ```

  2. 创建数组的同时进行初始化，例如

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

文件

- **流（Stream）：**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

文件

- **文件（File）：**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**
- Java库中定义了**File类**来获取文件本身的一些信息，如文件名、文件所在的目录、文件大小、文件读写权限等。
- 文件对象由以下方式创建：

```
File file = new File("C://java/hello.txt");
```

- 使用前需要在程序顶部`import java.io.*;`

文件

- File类的主要方法：

方法	描述
public File(String pathname)	通过文件名创建一个File实例
public boolean exits()	判断文件或目录是否存在
public boolean canRead()	判断文件是否可读
public boolean canWrite()	判断文件是否可写
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 isAbsolute()	判断抽象路径名是否是绝对路径
public boolean isHidden()	判断是否是隐藏文件

文件

- File类的主要方法：

方法	描述
public String getName()	获取文件名字，不包含路径
public String getParent()	获取文件的父目录
public String getPath()	获取文件（对象构造时）的路径
public String getAbsolutePath()	获取文件的绝对路径
public long length()	获取文件大小
public long lastModified()	获取文件最后修改时间
public File[] listFile()	获取目录下的文件列表

文件

- File类的主要方法：

方法	描述
public boolean mkdir()	创建一个目录
public boolean mkdirs()	与mkdir相同，除开父目录不存在的情况下，将和父目录一同创建
public boolean createNewFile()	创建新文件
public boolean renameTo(File dest)	重命名文件或目录
public boolean delete()	删除文件或空目录
public boolean setReadOnly()	设置文件属性为只读

文件

- **输入输出流：**

  字节流由InputStream和OutputStream处理

输入流	输出流	说明
FileInputStream	FileOutputStream	读写文件
BufferedInputStream	BufferedOutputStream	使用缓冲区，提高读写效率

文件

- **输入输出流：**

  字符流由Reader和Writer处理

输入流	输出流	说明
FileReader	FileWriter	文件读写
BufferedReader	BufferedWriter	使用缓冲区，提高读写效率

文件

- 以字节流为例的写操作：

```
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();
}
```

文件

- 以字节流为例的读操作：

```
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();
}
```

文件

- 以字符流为例的写操作：

```
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();
}
```

文件

- 以字符流为例的读操作：

```
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();
}
```

文件

- 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I/O处理需要引入必要的I/O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。

阅读资料

- 字符串（https://www.runoob.com/java/java-string.html）
- 数组（https://www.runoob.com/java/java-array.html）
- 文件IO（https://www.runoob.com/java/java-files-io.html）