day08【String类、static关键字、Arrays类、Math类】

今日内容

- String类
- static关键字
- Arrays类
- Math类

教学目标

- [ ] 能够使用String类的构造方法创建字符串对象
- [ ] 能够明确String类的构造方法创建对象,和直接赋值创建字符串对象的区别
- [ ] 能够使用文档查询String类的判断方法
- [ ] 能够使用文档查询String类的获取方法
- [ ] 能够使用文档查询String类的转换方法
- [ ] 能够理解static关键字
- [ ] 能够写出静态代码块的格式
- [ ] 能够使用Arrays类操作数组
- [ ] 能够使用Math类进行数学运算

第一章 String类

1.1 String类概述

目标:

String类的概述

步骤:

- 概述
- 特点

讲解:

概述

`java.lang.String` 类代表字符串。Java程序中所有的字符串文字（例如`"abc"` ）都可以被看作是实现此类的实例。

类 `String` 中包括用于检查各个字符串的方法，比如用于**比较**字符串，**搜索**字符串，**提取**子字符串以及创建具有翻译为**大写**或**小写**的所有字符的字符串的副本。

总结:

```
String类的概述和特点：
    1.java.lang.String 类代表字符串。Java程序中所有的字符串文字（例如"abc" ）都可以被看作是实现此类的实例。
    2.String类中有很多操作字符串的方法
```

特点

1. 字符串不变：字符串的值在创建后不能被更改。

```
String s1 = "abc";
s1 += "d";
System.out.println(s1); // "abcd"
// 内存中有"abc"，"abcd"两个对象，s1从指向"abc"，改变指向，指向了"abcd"。
```

2. String常量字符串，是可以被共享的。

```
String s1 = "abc";
String s2 = "abc";
// 内存中只有一个"abc"对象被创建，同时被s1和s2共享。
```

3. `"abc"` 等效于 `char[] data={ 'a' , 'b' , 'c' }`。

```
例如：
String str = "abc";

相当于：
char[] data = {'a', 'b', 'c'};
String str = new String(data);
// String底层是靠字符数组实现的。
```

小结:

- String类表示字符串,所以我们之前写的字符串常量都是String类的对象
- String类中有很多操作字符串的方法
- String类的对象是不可变的,字符串相加都会得到一个新的字符串对象
- String常量字符串，是可以被共享的

1.2 String类的构造方法

目标:

String类的使用步骤

步骤:

- 查看类
- 查看构造方法

讲解:

- 查看类
  - `java.lang.String` ：此类不需要导入。
- 查看构造方法
  - `public String()` ：初始化新创建的 String对象，以使其表示空字符序列。
  - `public String(char[] value)` ：通过当前参数中的字符数组来构造新的String。
  - `public String(byte[] bytes)` ：通过使用平台的默认字符集解码当前参数中的字节数组来构造新的String。
  - `public String(String original)` 初始化一个新创建的 `String` 对象，使其表示一个与参数相同的字符序列；换句话说，新创建的字符串是该参数字符串的副本。
  - 构造举例，代码如下：

```
// 无参构造
String str = new String ( ) ;

// 通过字符数组构造
char chars[] = {'a', 'b', 'c'};
String str2 = new String(chars);

// 通过字节数组构造
byte bytes[] = { 97, 98, 99 };
String str3 = new String(bytes);
```

小结:

略

1.3 能够明确String类的构造方法创建对象,和直接赋值创建字符串对象的区别

目标:

- 能够明确String类的构造方法创建对象,和直接赋值创建字符串对象的区别

步骤:

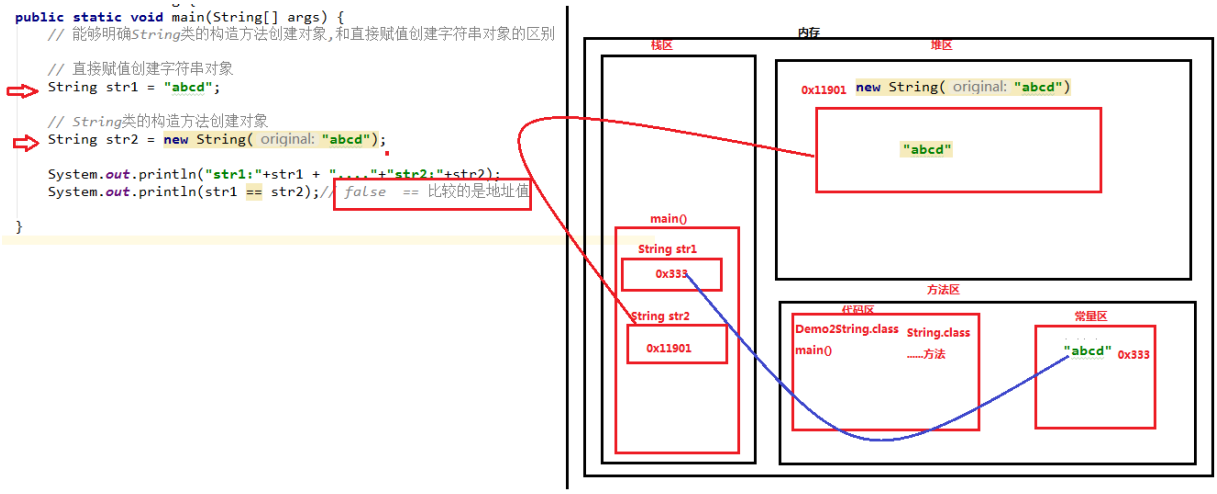
- 能够明确String类的构造方法创建对象,和直接赋值创建字符串对象的区别

讲解:

- 代码:

```
    public static void main(String[] args) {
        // 能够明确String类的构造方法创建对象,和直接赋值创建字符串对象的区别

        // 直接赋值创建字符串对象
        String str1 = "abcd";

        // String类的构造方法创建对象
        String str2 = new String("abcd");

        System.out.println("str1:"+str1 + "...."+"str2:"+str2);
        System.out.println(str1 == str2);// false  == 比较的是地址值

    }
```

- 分析:



小结:

- String类的构造方法创建对象在堆区
- 直接赋值创建字符串对象在常量区中

1.4 String类的常用方法

目标:

常用方法

步骤:

- 查看成员方法

讲解:

判断功能的方法

- `public boolean equals (Object anObject)` :将此字符串与指定对象进行比较。

- `public boolean equalsIgnoreCase (String anotherString)` ：将此字符串与指定对象进行比较，忽略大小写。

  方法演示，代码如下：

```
 public static void main(String[] args) {
        /*
           判断功能的方法
               - public boolean equals (Object anObject) ：将此字符串与指定对象进行比较。
               - public boolean equalsIgnoreCase (String anotherString) ：将此字符串与指定对象进行比较，忽略大小写。

           tips: Object 是” 对象”的意思，也是一种引用类型。作为参数类型，表示任意对象都可以传递到方法中
         */
        //  public boolean equalsIgnoreCase (String anotherString)
        // 1.创建一个字符串对象
        String str1 = "hello";

        // 2.创建一个字符串对象
        String str2 = "Hello";

        // 3.使用equalsIgnoreCase()方法比较str1和str2字符串是否相等
        boolean res1 = str1.equalsIgnoreCase(str2);
        System.out.println("res1的值是:"+res1);// true
    }

    /**
     * 演示 boolean equals (Object anObject)
     */
    private static void method01() {
        // - public boolean equals (Object anObject) ：将此字符串与指定对象进行比较。
        // 1.创建一个字符串对象
        String str1 = "hello";

        // 2.创建一个字符串对象
        String str2 = "Hello";

        // 3.使用equals方法比较str1和str2这2个字符串是否相等
        boolean res1 = str1.equals(str2);
        System.out.println("res1的值是:" + res1);// false


        // 扩展:
        // 创建一个字符串对象
        String str3 = new String("hello");

        // 使用equals方法比较str1与str3这2个字符串是否相等
        boolean res2 = str1.equals(str3);
        System.out.println("res2的值是:" + res2);// true equals()方法比较的是2个字符串的内容是否相等
        System.out.println(str1 == str3);// false  == 比较的是2个字符串对象的地址值是否相等

    }
```

> Object 是" 对象"的意思，也是一种引用类型。作为参数类型，表示任意对象都可以传递到方法中。

- 如果想要比较2个字符串是否相等,那么就使用equals方法,千万别用 == 比较

获取功能的方法

- `public int length ()` ：返回此字符串的长度。
- `public String concat (String str)` ：将指定的字符串连接到该字符串的末尾。
- `public char charAt (int index)` ：返回指定索引处的 char值。
- `public int indexOf (String str)` ：返回指定子字符串第一次出现在该字符串内的索引。
- `public String substring (int beginIndex)` ：返回一个子字符串，从beginIndex开始截取字符串到字符串结尾。
- `public String substring (int beginIndex, int endIndex)` ：返回一个子字符串，从beginIndex到endIndex截取字符串。含beginIndex，不含endIndex。

方法演示，代码如下：

```
public class String_Demo02 {
     public static void main(String[] args) {
        /*
        获取功能的方法
            - public int length () ：返回此字符串的长度。
            - public String concat (String str) ：将指定的字符串连接到该字符串的末尾。
            - public char charAt (int index) ：返回指定索引处的 char值。
            - public int indexOf (String str) ：返回指定子字符串第一次出现在该字符串内的索引。
            - int lastIndexOf(String str)  返回指定子字符串在此字符串中最右边出现处的索引。
            - public String substring (int beginIndex) ：返回一个子字符串，从beginIndex开始截取字符串到字符串结尾。
            - public String substring (int beginIndex, int endIndex) ：返回一个子字符串，从beginIndex到endIndex截取字符串。含beginIndex，不含endIndex。
         */
        // 1.创建一个字符串对象
        String str1 = "hello-world!";

        // - public int length () ：返回此字符串的长度。
        int len = str1.length();
        System.out.println("str1的长度是:"+len);//str1的长度是:12

        // - public String concat (String str) ：将指定的字符串连接到该字符串的末尾。拼接字符串
        String newStr = str1.concat(" 世界,你好!");
        System.out.println("str1的值是:"+str1);// hello world!  字符串是不可变的
        System.out.println("newStr的值是:"+newStr);// hello world! 世界,你好!


        // - public char charAt (int index) ：返回指定索引处的 char值。
        // 获取str1字符串中1索引位置对于的元素
        char ch = str1.charAt(1);
        System.out.println("str1字符串中1索引位置对于的元素:"+ch);// e

        // 获取str1字符串的最后一个字符
        char ch2 = str1.charAt(len-1);
        System.out.println("str1字符串的最后一个字符:"+ch2);// !
```

```
        // public int indexOf (String str) ：返回指定子字符串第一次出现在该字符串内的索引
        // 获取"world"子字符串在str1字符串中第一次出现的索引
        int index = str1.indexOf("world");
        System.out.println("world子字符串在str1字符串中第一次出现的索引:"+index);// 6

        // 获取"l"子字符串在str1字符串中第一次出现的索引
        int index2 = str1.indexOf("l");
        System.out.println("l子字符串在str1字符串中第一次出现的索引:"+index2);// 2

        // 获取"l"子字符串在str1字符串中最后一次出现的索引
        int index3 = str1.lastIndexOf("l");
        System.out.println("l子字符串在str1字符串中最后一次出现的索引:"+index3);// 9

        //- public String substring (int beginIndex) ：返回一个子字符串，从beginIndex开始截取字符串到字符串结尾。
        // 截取str1字符串中的:world!
        String subStr1 = str1.substring(6);
        System.out.println("subStr1的值是:"+subStr1);// world!

        // - public String substring (int beginIndex, int endIndex) ：返回一个子字符串，从beginIndex到endIndex截取字符串。含beginIndex，不含endIndex。
        // 截取str1字符串中的:hello
        String subStr2 = str1.substring(0, 5);
        System.out.println("subStr2的值是:"+subStr2);

    }
}
```

转换功能的方法

- `public char[] toCharArray ()` ：将此字符串转换为新的字符数组。
- `public byte[] getBytes ()` ：使用平台的默认字符集将该 String编码转换为新的字节数组。
- `public String replace (CharSequence target, CharSequence replacement)` ：将与target匹配的字符串使用replacement字符串替换。

方法演示，代码如下：

```
  public static void main(String[] args) {
        /*
            转换功能的方法
                - public char[] toCharArray () ：将此字符串转换为新的字符数组。
                - public byte[] getBytes () ：使用平台的默认字符集将该 String编码转换为新的字节数组。
                - public String replace (CharSequence target, CharSequence replacement) ：将与target匹配的字符串使用replacement字符串替换。
                tips:CharSequence 是一个接口，也是一种引用类型。作为参数类型，可以把String对象传递到方法中。
                    简而言之:如果参数的类型是CharSequence,那么就可以传入字符串对象
         */
        // 创建一个字符串对象
        String str1 = "abcdefg";
```

```
        // 1. - public char[] toCharArray () ：将此字符串转换为新的字符数组。
        // 把str1字符串转换为字符数组
        char[] chs = str1.toCharArray();// {'a','b','c','d','e','f','g'}

        // 遍历chs字符数组
        for (int i = 0; i < chs.length; i++) {
            System.out.print(chs[i]+" ");
        }

        System.out.println();
        System.out.println("==========================");

        //  2.- public byte[] getBytes () ：使用平台的默认字符集将该 String编码转换为新的字节数组。
        // 把str1字符串转换为byte[]数组(字节数组)
        byte[] bys = str1.getBytes();

        // 遍历bys字节数组
        for (int i = 0; i < bys.length; i++) {
            System.out.print(bys[i]+" ");
        }

        System.out.println();
        System.out.println("==========================");

        // 3.public String replace (CharSequence target, CharSequence replacement) ：将与target匹配的字符串使用replacement字符串替换。
        // 使用nba替换str1中的abc
        String newStr = str1.replace("abc", "nba");
        System.out.println("str1的值是:"+str1);// abcdefg
        System.out.println("newStr的值是:"+newStr);// nbadefg


    }
```

> CharSequence 是一个接口，也是一种引用类型。作为参数类型，可以把String对象传递到方法中。

分割功能的方法

- `public String[] split(String regex)` ：将此字符串按照给定的regex（规则）拆分为字符串数组。

方法演示，代码如下：

```
public class String_Demo03 {
    public static void main(String[] args) {
        /*
            分割功能的方法
                - public String[] split(String regex) ：将此字符串按照给定的regex（规则）拆分为字符串数组。
         */
        // 1.特殊字符
        // 创建一个字符串
        String str = "aa|bb|cc";
```

```
        // - public String[] split(String regex) ：将此字符串按照给定的regex（规则）拆分为字符串数组。
        // 以|拆分str字符串----> 拆分后得到: aa bb cc
        String[] arr = str.split("\\|");

        System.out.println("arr数组的长度:"+arr.length);
        // 遍历split数组
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
        }

        System.out.println("================");
        // 2.没有特殊字符
        // 创建一个字符串
        String str2 = "aa,bb,cc";

        // - public String[] split(String regex) ：将此字符串按照给定的regex（规则）拆分为字符串数组。
        // 以,拆分str字符串----> 拆分后得到: aa bb cc
        String[] arr2 = str2.split(",");

        System.out.println("arr2数组的长度:"+arr2.length);
        // 遍历split数组
        for (int i = 0; i < arr2.length; i++) {
            System.out.println(arr2[i]);
        }
    }
}
```

小结:

略

1.5 String类的练习

目标:

String类的练习

步骤:

- 拼接字符串
- 统计字符个数

讲解:

拼接字符串

定义一个方法，把数组{1,2,3}按照指定个格式拼接成一个字符串。格式参照如下：[word1#word2#word3]。

```
public class StringTest1 {
  public static void main(String[] args) {
        /*
```

```
            拼接字符串
                定义一个方法，把数组{10,20,30}按照指定个格式拼接成一个字符串。格式参照如下：
[word1#word2#word3]。
         */
        int[] arr = {10,20,30};

        // 调用appendArray()方法
        String s = appendArray(arr);
        System.out.println("s:"+s);//[10#20#30]
    }

    public static String appendArray(int[] arr) {
        // 1. 定义一个字符串str,字符的内容是左中括号: [
        String str = "[";

        // 2. 遍历数组
        for (int i = 0; i < arr.length; i++) {
            // 3. 获取数组元素
            int e = arr[i];

            // 4. 判断元素是否是最后一个元素
            if (i == arr.length - 1) {
                // 5. 如果是最后一个元素,就使用str拼接上: 元素+]
                str += e + "]";
            } else {
                // 6. 如果不是最后一个元素,就使用str拼接上: 元素+#
                str += e + "#";
            }
        }
        // 7. 把拼接后的字符串返回
        return str;
    }
}
```

统计字符个数

键盘录入一个字符串，统计字符串中大小写字母及数字字符个数

```
public class StringTest2 {
  public static void main(String[] args) {
    //键盘录入一个字符串数据
    Scanner sc = new Scanner(System.in);
    System.out.println("请输入一个字符串数据：");
    String s = sc.nextLine();

    //定义三个统计变量，初始化值都是0
    int bigCount = 0;
    int smallCount = 0;
    int numberCount = 0;

    //遍历字符串，得到每一个字符
    for(int x=0; x<s.length(); x++) {
      char ch = s.charAt(x);
```

```
        //拿字符进行判断
        if(ch>='A'&&ch<='Z') {
          bigCount++;
        }else if(ch>='a'&&ch<='z') {
          smallCount++;
        }else if(ch>='0'&&ch<='9') {
          numberCount++;
        }else {
          System.out.println("该字符"+ch+"非法");
        }
      }

      //输出结果
      System.out.println("大写字符："+bigCount+"个");
      System.out.println("小写字符："+smallCount+"个");
      System.out.println("数字字符："+numberCount+"个");
    }
}
```

小结:

略

第二章 static关键字

目标:

static关键字

步骤:

- 概述
- static修饰成员变量
- static修饰成员方法

讲解:

2.1 概述

关于 `static` 关键字的使用，它可以用来修饰的成员变量和成员方法，被修饰的成员是**属于类**的，而不是单单是属于某个对象的。也就是说，既然属于类，就可以不靠创建对象来调用了。

特点:

1.static 是一个关键字

2.static 可以修饰成员变量和成员方法

3.被static修饰的成员变量和成员方法是属于类的,不单单属于某个对象的

4.被static修饰的成员变量和成员方法可以使用类名直接调用

2.2 static修饰成员变量------>类变量

- **类变量**：使用 static关键字修饰的成员变量。
- **定义格式**：

```
static 数据类型 变量名；
```

- **举例**：

```
public class Person {
    // 非静态变量
    String name;
    // 静态变量
    static String country;
}
```

- **访问静态变量:**
  - 类名.静态变量名
  - 对象名.静态变量名

```
// 测试类
public static void main(String[] args) {

    Person p1 = new Person();
    p1.name = "张无忌";
    p1.country = "中国";
    // 姓名:张无忌,国籍:中国
    System.out.println("姓名:"+p1.name+",国籍:"+p1.country);

    Person p2 = new Person();
    // 姓名:null,国籍:中国
    System.out.println("姓名:"+p2.name+",国籍:"+p2.country);

    // 类名.静态成员变量名
    System.out.println(Person.country);// 中国
}
```

- **特点:**被 `static` 修饰的成员变量会变成静态变量(也就是类变量)。该静态变量是属于类的,所以会被该类的每一个对象所共享,任何对象都可以更改该类变量的值，但也可以在不创建该类的对象的情况下对类变量进行操作。

- ## 小结:

```
- static修饰成员变量的格式:
    static 数据类型 变量名;
- 访问静态变量:
  - 对象名.静态变量名
  - 类名.静态变量名 ----->推荐
- 特点: 被 static 修饰的成员变量会变成静态变量,该静态变量会被该类的所有对象共享,也就是大家使用的是同一份数据
```

2.3 static修饰成员方法---->静态方法

- **类方法概述**：使用 static关键字修饰的成员方法，习惯称为**静态方法。**
- **定义格式**：

```
修饰符 static 返回值类型 方法名 (参数列表){
    // 执行语句
}
```

- 举例：在Student类中定义静态方法

```
public class Student {
    // 静态方法
    public static void method1(){
        System.out.println("静态方法method1执行了...");
    }

    public static void method2(){
        System.out.println("静态方法method2执行了...");
    }

}
```

- **访问方式**:
  - 类名.静态方法名(实参);
  - 对象名.静态方法名(实参);

```
    public static void main(String[] args) {

        // 调用静态方法:
        // 类名.静态方法名(实参);
        Student.method1();

        // 对象名.静态方法名(实参);
        Student stu = new Student();
        stu.method2();
    }
```

- **特点:**当`static`修饰成员方法时，该方法称为**类方法** 。静态方法在声明中有`static` ，建议使用类名来调用，而不需要创建类的对象。调用方式非常简单。

小结:

```
static修饰成员方法:
    1.格式:
        权限修饰符 static 返回值类型 方法名(参数列表){
            代码...
        }
    2.使用:
        对象名.方法名(实参);
        类名.方法名(实参);    --->推荐

    3.特点:
        被static修饰的成员方法,会变成静态方法(类方法),那么就不单单属于某个对象,而是
        属于类的,所有可以直接使用类名直接调用
```

2.4 静态方法调用的注意事项：

- 静态方法可以直接访问静态变量和静态方法。
- 静态方法**不能直接访问**普通成员变量或成员方法。反之，非静态成员方法可以直接访问类变量或静态方法。
- 静态方法中，不能使用**this**关键字。

> 小贴士：静态方法只能访问静态成员。

- 被static修饰的成员可以并且建议通过**类名直接访问**。虽然也可以通过对象名访问静态成员，原因即多个对象均属于一个类，共享使用同一个静态成员，但是不建议，会出现警告信息。(**推荐使用类名直接访问静态成员变量和静态成员方法)**
- 演示效果图:

```
// 静态方法
public static void method1(){
    // 静态方法可以直接访问静态变量和静态方法
    System.out.println("访问静态变量:"+country);
    method2();// 访问静态方法

    // 静态方法不能直接访问普通成员变量或成员方法
    System.out.println("访问非静态变量:"+name); // 报错
    method3();//访问非静态方法    报错

    // 静态方法中，不能使用this关键字。
    System.out.println(this.country);// 报错
    System.out.println(this.name);// 报错

    System.out.println("静态方法method1执行了...");
}
```

- 源码:

```
public class Student {
    String name;// 非静态变量
    static String country;// 静态变量

    // 静态方法
    public static void method1(){
        // 静态方法可以直接访问静态变量和静态方法
        System.out.println("访问静态变量:"+country);
        method2();// 访问静态方法

        // 静态方法不能直接访问普通成员变量或成员方法
        System.out.println("访问非静态变量:"+name); // 报错
        method3();//访问非静态方法   报错

        // 静态方法中，不能使用this关键字。
        System.out.println(this.country);// 报错
        System.out.println(this.name);// 报错

        System.out.println("静态方法method1执行了...");
    }

    public static void method2(){
        System.out.println("静态方法method2执行了...");
    }

    // 非静态方法
     // 非静态方法
    public  void method3(){
        // 非静态成员方法可以直接访问类变量或静态方法。
        System.out.println("访问非静态成员变量:"+name);
        System.out.println("访问静态成员变量:"+country);

        // 访问非静态方法
        method4();
        // 访问静态方法
        method2();

        System.out.println("method3方法执行了...");
    }

    public  void method4(){
        System.out.println("非静态方法method4执行了...");
    }

}
```

小结:

- 静态方法中只能直接访问静态成员,不能访问非静态成员和this。
- 非静态方法中可以访问一切的成员(成员变量和成员方法)

- 推荐大家使用类名直接访问静态成员

2 静态原理图解

目标:

静态原理图解

步骤:

- 静态原理图解

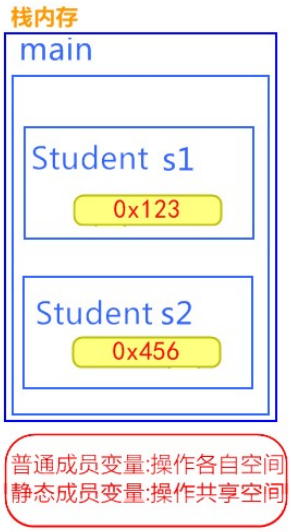
讲解:

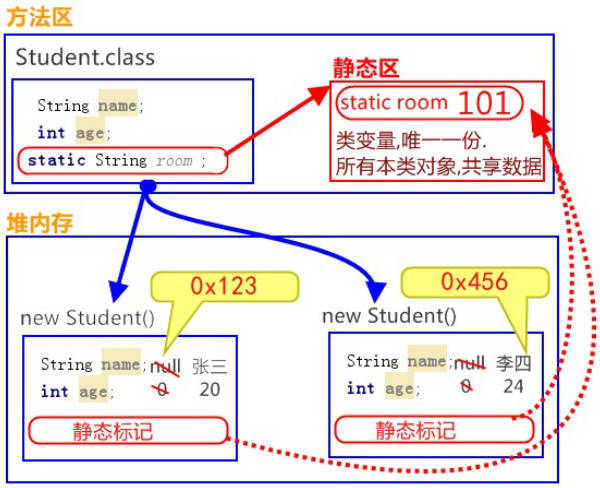
`static` 修饰的内容：

- 是随着类的加载而加载的，且只加载一次。
- 存储于一块固定的内存区域（静态区），所以，可以直接被类名调用。
- 它优先于对象存在，所以，可以被所有对象共享。

```
public class Student {
    private String name;
    private int age;
    public static String room ;

    public Student() {
    }
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Student s1  = new Student();
        s1.name = "张三";
        s1.age = 20;
        s1.room = "101";

        Student s2  = new Student();
        s2.name = "李四";
        s2.age = 24;
        System.out.println(s2.room);
    }
}
```





小结:

- 静态成员是随着类的加载而加载,并且只会加载一次

3 静态代码块

目标:

静态代码块

步骤:

- 静态代码块格式
- 静态代码块执行特点

讲解:

- **静态代码块**：定义在成员位置，使用static修饰的代码块{ }。
  - 位置：类中方法外。
  - 执行：随着类的加载而执行且执行一次，优先于main方法和构造方法的执行。

格式：

```
public class ClassName{
  static {
    // 执行语句
  }
}
```

演示:

```
public class Person {
    // 静态代码块
    static {
        System.out.println("Person类中的静态代码块");
    }

    public Person(){
        System.out.println("Person类中的构造方法");
    }

}

//测试类:
public class Demo6 {

    static {
        System.out.println("Demo6中的静态代码块");
    }

    public static void main(String[] args) {

        System.out.println("这是main方法中的代码");
        new Person();// 匿名对象
    }
}
```

作用：给类变量进行初始化赋值。用法演示，代码如下：

```
public class Game {
  public static int number;
  public static ArrayList<String> list;

  static {
    // 给类变量赋值
    number = 2;
    list = new ArrayList<String>();
    // 添加元素到集合中
    list.add("张三");
    list.add("李四");
  }
}
```

> 小贴士：
>
> static 关键字，可以修饰变量、方法和代码块。在使用的过程中，其主要目的还是想在不创建对象的情况下，去调用方法。下面将介绍两个工具类，来体现static 方法的便利。

小结:

```
静态代码块：定义在成员位置，使用static修饰的代码块{ }。
    - 格式:
        static{
            代码
        }
    - 位置：类中方法外。
    - 执行：随着类的加载而执行且执行一次，优先于main方法和构造方法的执行。
    - 作用：给类变量进行初始化赋值
```

第三章 Arrays类

目标:

Arrays类

步骤:

- Arrays类的概述
- 操作数组的方法
- 练习

讲解:

3.1 概述

`java.util.Arrays` 此类包含用来操作数组的各种方法，比如排序和搜索等。其所有方法均为静态方法，调用起来非常简单。

特点:

1.Arrays类在java.util包,需要导包

2.Arrays类中的方法都是静态方法,所以可以直接使用类名调用

3.Arrays类主要用来操作数组的

3.2 操作数组的方法

- `public static String toString(int[] a)` ：返回指定数组内容的字符串表示形式。

```
public static void main(String[] args) {
  // 定义int 数组
  int[] arr  =  {2,34,35,4,657,8,69,9};
  // 打印数组,输出地址值
  System.out.println(arr); // [I@2ac1fdc4

  // 数组内容转为字符串
  String s = Arrays.toString(arr);
  // 打印字符串,输出内容
  System.out.println(s); // [2, 34, 35, 4, 657, 8, 69, 9]
}
```

- `public static void sort(int[] a)` ：对指定的 int 型数组按数字升序进行排序。

```
public static void main(String[] args) {
  // 定义int 数组
  int[] arr  =  {24, 7, 5, 48, 4, 46, 35, 11, 6, 2};
  System.out.println("排序前:"+ Arrays.toString(arr)); // 排序前:[24, 7, 5, 48, 4, 46, 35, 11, 6,
2]
  // 升序排序
  Arrays.sort(arr);
  System.out.println("排序后:"+ Arrays.toString(arr));// 排序后:[2, 4, 5, 6, 7, 11, 24, 35, 46,
48]
}
```

3.3 练习

请使用`Arrays`相关的API，将一个随机字符串中的所有字符升序排列，并倒序打印。

```
public class ArraysTest {
  public static void main(String[] args) {
    // 定义随机的字符串
    String line = "ysKUreaytWTRHsgFdSAoidq";
    // 转换为字符数组
    char[] chars = line.toCharArray();

    // 升序排序
    Arrays.sort(chars);
    // 反向遍历打印
    for (int i =  chars.length-1; i >= 0 ; i--) {
      System.out.print(chars[i]+" "); // y y t s s r q o i g e d d a W U T S R K H F A

    }
```

```
    }
}
```

小结:

- toString
- sort

第四章 Math类

目标:

Math类

步骤:

- Math类的概述
- 基本运算的方法
- 练习

讲解:

4.1 概述

`java.lang.Math` 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。类似这样的工具类，其所有方法均为静态方法，并且不会创建对象，调用起来非常简单。

- 特点:
  - Math类在java.lang包,不需要导包
  - Math类主要提供执行基本数学运算的方法
  - Math.类中的方法也都是静态方法
  - 可以直接使用类名调用

4.2 基本运算的方法

- `public static double abs(double a)` ：返回 double 值的绝对值。

```
double d1 = Math.abs(-5); //d1的值为5
double d2 = Math.abs(5); //d2的值为5
```

- `public static double ceil(double a)` ：返回大于等于参数的最小的整数。

```
double d1 = Math.ceil(3.3); //d1的值为 4.0
double d2 = Math.ceil(-3.3); //d2的值为 -3.0
double d3 = Math.ceil(5.1); //d3的值为 6.0
```

- `public static double floor(double a)` ：返回小于等于参数最大的整数。

```
double d1 = Math.floor(3.3); //d1的值为3.0
double d2 = Math.floor(-3.3); //d2的值为-4.0
double d3 = Math.floor(5.1); //d3的值为 5.0
```

- `public static long round(double a)` ：返回最接近参数的 long。(相当于四舍五入方法)

```
long d1 = Math.round(5.5); //d1的值为6.0
long d2 = Math.round(5.4); //d2的值为5.0
```

4.3 练习

请使用`Math`相关的API，计算在`-10.8`到`5.9`之间，绝对值大于`6`或者小于`2.1`的整数有多少个？

```
public class MathTest {
  public static void main(String[] args) {
    // 定义最小值
    double min = -10.8;
    // 定义最大值
    double max = 5.9;
    // 定义变量计数
    int count = 0;
    // 范围内循环
    for (double i = Math.ceil(min); i <= max; i++) {
      // 获取绝对值并判断
      if (Math.abs(i) > 6 || Math.abs(i) < 2.1) {
        // 计数
        count++;
      }
    }
    System.out.println("个数为: " + count + " 个");
  }
}
```

小结:

略