# day07

## 构造方法

### 构造方法的定义和使用

1. 构造方法 : 构造函数, 构造器, Constructor
2. Person p = new Person(); // ()就表示构造方法调用
3. 构造方法作用 : 可以给对象中的成员变量进行赋值, 当创建对象同时, JVM虚拟机会主动调用一次构造方法执行, 当对象创建完毕, 构造执行完毕, 对象中的成员变量就有值了
4. 构造方法定义 :

修饰符 构造方法名(参数列表){

方法体;// 主要就是给对象中的成员变量进行赋值操作

}

1. 说明:
2. 修饰符 : 通常使用public公共修饰即可
3. 构造方法没有返回值类型, 连void都没有
4. 构造方法名 : 必须与类名一致
5. 构造方法中没有return, 因为没有返回值类型,如果一定要写,写成 return;
6. 构造方法执行机制:

当创建对象同时, JVM虚拟机主动调用构造方法执行一次, 构造方法不能使用对象名再手动调用

代码

|  |
| --- |
| package com.ujiuye.constructor;  public class ConstructorClass {  private String name;  private int age;    public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  /\*  \* 修饰符 构造方法名(参数列表){  方法体;// 主要就是给对象中的成员变量进行赋值操作  }  \*/  public ConstructorClass() {  name = "李四";  age = 20;  System.out.println(name + "---" + age);  // return;  }    public static void main(String[] args) {  // 调用空参数构造方法  ConstructorClass cc = new ConstructorClass();  //The method ConstructorClass() is undefined for the type ConstructorClass  // cc.ConstructorClass();  System.out.println(cc.getName());//李四  System.out.println(cc.getAge());// 20    ConstructorClass cc1 = new ConstructorClass();  }  } |

### 构造方法使用的注意事项

1. 参数列表 :

a : 构造方法没有参数列表, 只能在构造方法中给成员变量进行默认的赋值; 创建对象时, 调用构造参数列表就是空的 Person p = new Person();

b : 构造方法有参数列表, 可以通过参数列表中的数据给对象中的成员变量进行赋值; 创建对象时, 调用有参数构造, 需要传递实际参数

Person p = new Person(“张三”,23);

Person p1 = new Person(“李四”,20);

1. 如果类型中没有手动定义出任何构造,那么系统会为类型自动添加1个空参数,空实现的构造方法, public 构造方法名(){}
2. 如果类型中手动定义出了构造, 不管空参还是有参, 系统不会再添加任何构造, 一律以自己定义出构造方法为调用标准
3. 构造方法可以重载(Overload):

方法重载 : 定义在同一个类中, 方法名相同, 参数列表不同, 与方法返回值类型无关

1. 如果在类型中定义出有参数构造, 请你将空参数构造添加上

注意: 如果类型中具有多个重载构造方法, 创建对象时, 挑选其中一个觉得合适的构造调用

代码

|  |
| --- |
| package com.ujiuye.constructor;  public class ConstructorParam {  private String name;  private int age;    public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  // 以下两个构造方法重载  // 定义出有参数构造方法  public ConstructorParam(String name, int age) {  this.name = name;  this.age = age;  }    // 定义出一个空参数构造  public ConstructorParam() {  name = "大美丽";  age = 18;  }    public static void main(String[] args) {  // 调用有参数构造, 传递实际参数  ConstructorParam cp = new ConstructorParam("张三", 20);  System.out.println(cp.getAge());// 20  System.out.println(cp.getName());// 张三    ConstructorParam cp1 = new ConstructorParam("李四", 19);  System.out.println(cp1.getName());// 李四  System.out.println(cp1.getAge());// 19    // 调用空参数构造  ConstructorParam cp2 = new ConstructorParam();  System.out.println(cp2.getName());// 大美丽  System.out.println(cp2.getAge());// 18  }  } |

### 构造方法赋值与set方法赋值的比较

结论 : 谁也无法取代set方法给私有成员变量赋值存在

1. 构造方法只能在创建对象同时调用一次, 不能手动调用构造修改对象中成员变量的值, 因此不能取代set方法赋值存在
2. 如果类型中定义出私有成员变量, 那么要求必须提供这个成员变量对应的set和get方法, 而set方法每次调用都可以修改对象中成员变量的值

### this关键字在构造方法中的使用(扩展,了解)

1. this : 关键字, 表示当前类型对象的引用

this关键字使用场景 :

a : 区分成员变量和局部变量重名问题, 带有this.关键字的变量使用表示成员变量使用

b : this关键字可以使用在同一个类中的构造方法之间, 作用就是可以进行同类构造方法彼此的调用, this(需要调用的构造方法具有的实际参数);

构造方法调用使用场景 :

如果A构造中的实现过程, 正好符合B构造逻辑的一部分, 那么可以直接在B构造方法第一行使用this(A构造方法实际参数), 让A构造帮助B构造实现一部分功能

注意 : this() 构造方法调用方式, 必须写在构造方法有效行第一行

代码

|  |
| --- |
| package com.ujiuye.constructor;  public class ConstructorParam {  private String name;  private int age;    public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  // 以下三个构造方法重载  // 定义出有参数构造方法  public ConstructorParam(String name, int age) {  // 构造方法的调用,必须写在构造中有效行第一行  //Constructor call must be the first statement in a constructor  this(name);  //this.name = name;  this.age = age;  }    public ConstructorParam(String name) {  this.name = name;  System.out.println("我是一个参数的构造方法, 我被调用了");  }    // 定义出一个空参数构造  public ConstructorParam() {  name = "大美丽";  age = 18;  }    public static void main(String[] args) {  // 调用有参数构造, 传递实际参数  ConstructorParam cp = new ConstructorParam("张三", 20);  System.out.println(cp.getAge());// 20  System.out.println(cp.getName());// 张三    ConstructorParam cp1 = new ConstructorParam("李四", 19);  System.out.println(cp1.getName());// 李四  System.out.println(cp1.getAge());// 19    // 调用空参数构造  ConstructorParam cp2 = new ConstructorParam();  System.out.println(cp2.getName());// 大美丽  System.out.println(cp2.getAge());// 18  }  } |

### 创建对象同时成员变量的赋值过程

1. JVM虚拟机在成员变量进入堆内存时, 默认赋初值
2. 定义一个成员变量同时, 直接赋值, 称为显示赋值
3. 创建对象同时, 构造方法也可以给成员变量赋值
4. set方法在创建对象完成之后, 可以修改成员变量的值

创建一个对象的同时, 对象中成员变量赋值过程(赋值先后顺序)

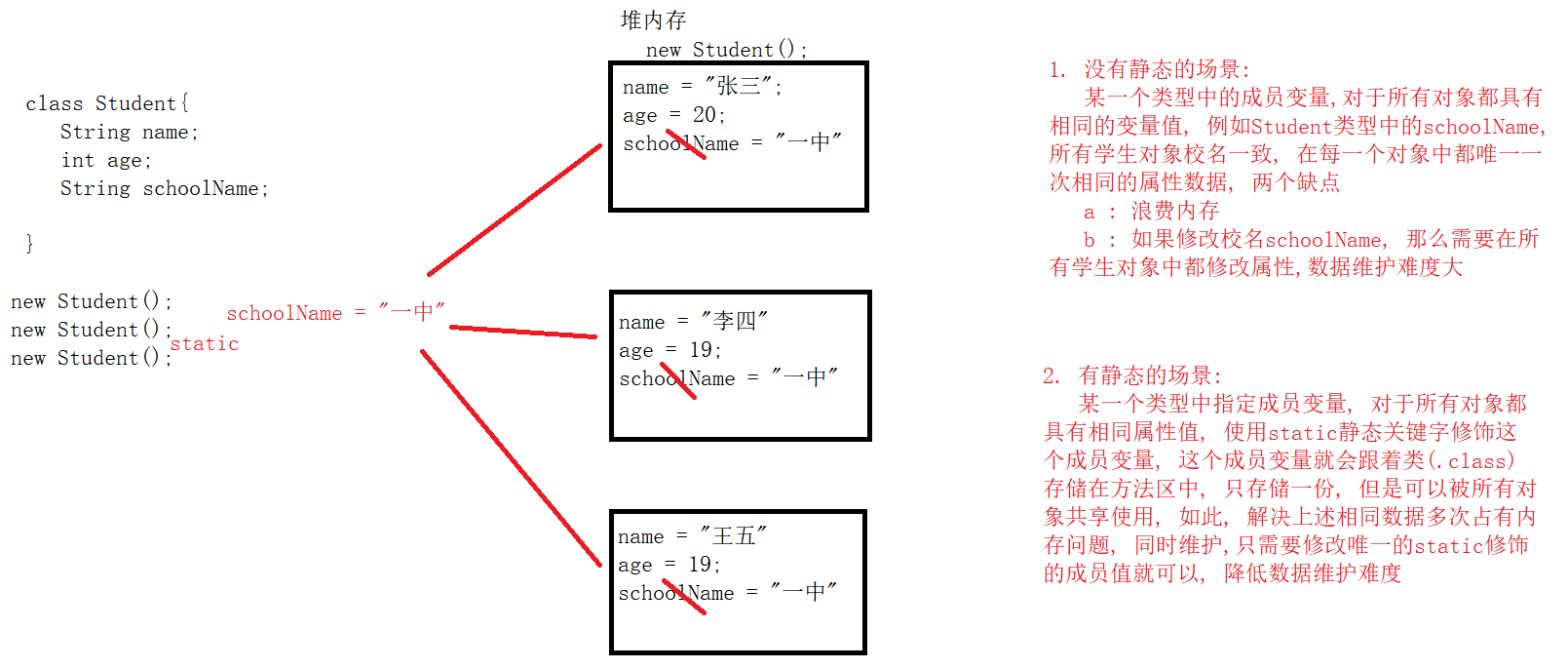
先JVM虚拟机赋值------>再变量显示赋值---->最后构造方法给成员变量赋值

代码

|  |
| --- |
| package com.ujiuye.constructor;  public class VariableNew {  // name使用JVM虚拟机默认赋值  private String name;  // age显示赋值  private int age = 20;  private String sex = "男";    public VariableNew() {  sex = "女";  }    public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public String getSex() {  return sex;  }  public void setSex(String sex) {  this.sex = sex;  }  public static void main(String[] args) {  VariableNew vn = new VariableNew();  System.out.println(vn.getName());// null  System.out.println(vn.getAge());// 20  System.out.println(vn.sex);// 女  }  } |

## static静态

### 静态概述



代码

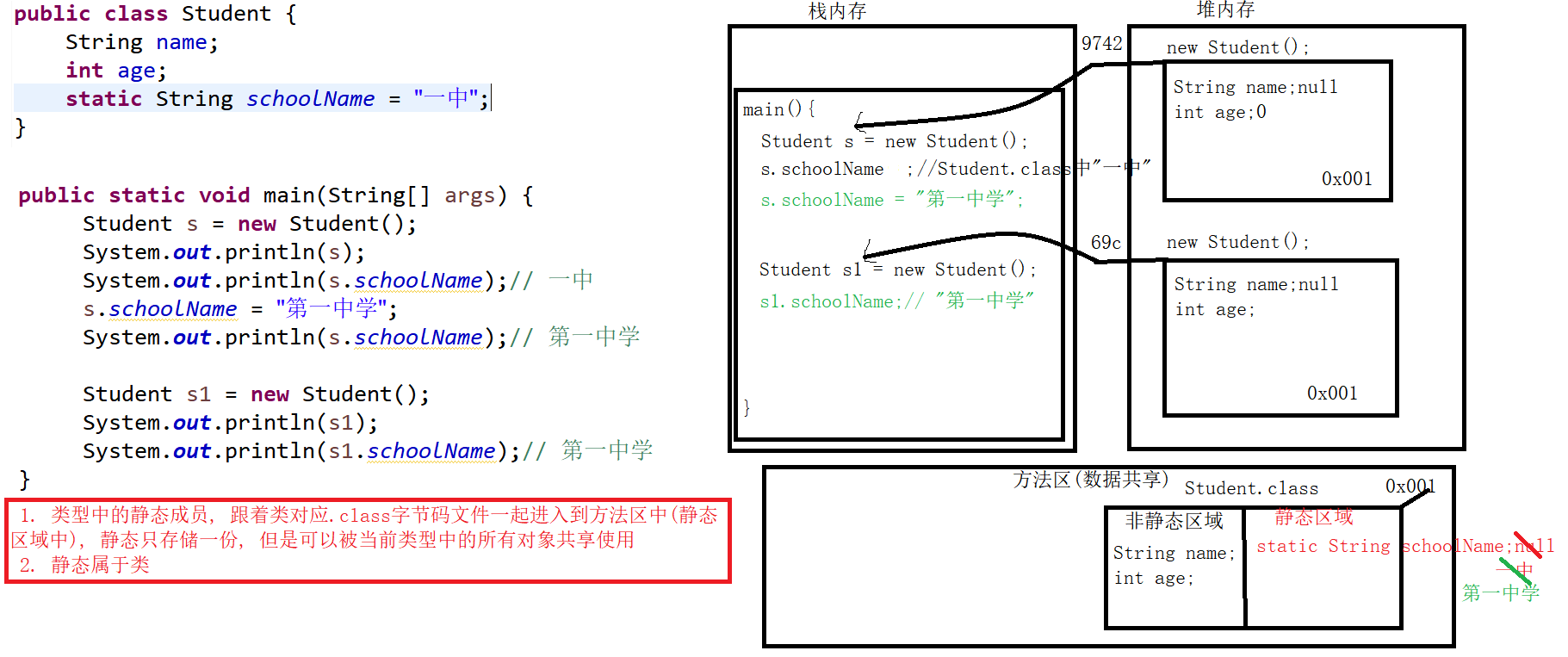
|  |
| --- |
| package com.ujiuye.staticdemo;  public class Student {  String name;  int age;  static String schoolName = "二中";  } |

|  |
| --- |
| package com.ujiuye.staticdemo;  public class TestStudentStatic {  public static void main(String[] args) {  Student s = new Student();  System.out.println(s);  System.out.println(s.schoolName);// 一中  s.schoolName = "第一中学";  System.out.println(s.schoolName);// 第一中学    Student s1 = new Student();  System.out.println(s1);  System.out.println(s1.schoolName);// 第一中学  }  } |

### 静态在内存中的存储机制

1. 类型中的静态成员, 跟着类对应的 .class字节码文件一起进入到方法区中(静态区域中), 静态只存储一份, 但是可以被当前类型中的所有对象共享使用

2. 静态属于类



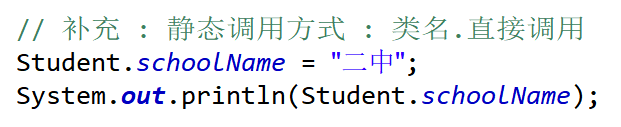
### 静态的特点

1. 静态使用static关键字修饰, static是修饰符, static静态的, 静止的
2. static修饰:
3. 成员变量 : 一个类型中的某属性, 对于所有对象有相同属性值, 使用static修饰这个成员变量
4. 方法 : 如果想直接在main方法中调用, 方法加static; 如果这个方法的使用很频繁, 不需要创建对象, 就可以调用这个方法, 那么可以使用static修饰方法
5. 代码块
6. 内部类
7. 静态加载机制 : 一个类型.class字节码文件进入内存, 静态成员就被加载进入静态区域; 静态优先于对象存在于内存中
8. 静态具有共享性, 虽然存储在方法区中, 但是可以被类型的所有对象共享使用
9. 静态属于类, 因此从代码实现的角度上考量 :

静态成员调用方式有两种:

a : 类名.直接调用; (最推荐静态使用方式)

b : 对象名.调用; (不推荐)



### 静态使用注意事项

1. 静态中不能直接使用非静态(因为静态优先于对象存在于内存中)

a : 静态方法中不能直接使用非静态成员变量, 但是创建对象之后, 可以使用对象名.调用非静态成员变量

b : 静态方法中不能直接调用非静态方法, 但是创建对象之后,可以使用对象名.调用非静态方法功能

c : 静态方法中不能使用this关键字

代码

|  |
| --- |
| package com.ujiuye.staticdemo;  public class Demo01\_静态使用注意事项 {  // 定义出一个成员变量  static int i = 10;  int j = 20;    public static void main(String[] args) {  // 1. Cannot make a static reference to the non-static field i  // 静态方法中不能直接使用非静态成员变量  /\*  \* 分析 :  \* 当类加载机进入到内存中时, 静态成员变量有值可以使用; 静态方法也可以直接调用堆内存中, 没有任何对象出生; 非静态成员变量跟着对象创建才出现在内存中有值  \*/  System.out.println(i);// 10    // 2. 如果使用非静态成员变量, 也可以先创建对象, 使用对象名.调用  System.out.println(new Demo01\_静态使用注意事项().j);    // 3. 静态方法中可以直接调用其他静态方法  eat();    // 4. 静态方法中不能直接调用非静态方法, 创建对象之后,可以调用  //Cannot make a static reference to the non-static method sleep()  // from the type Demo01\_静态使用注意事项  new Demo01\_静态使用注意事项().sleep();    // 5. 因为this表示对象引用, 静态优先于对象存在于内存中, 直接使用this  // Cannot use this in a static context  // System.out.println(this.j);  }    public static void eat() {  System.out.println("吃饭");  }    public void sleep() {  System.out.println("睡觉");  }  } |

### 静态成员变量和非静态成员变量的比较

1. 所属不同:

a : 静态成员变量 : 属于类

b : 非静态成员变量 : 属于对象

1. 内存中存储区域不同:

a : 静态成员变量 : 跟着类存储在方法区静态区域中

b : 非静态成员变量 : 跟着对象存储在堆内存中

1. 生命周期不同:

a : 静态成员变量 : 与类的生命周期一致, 当类加载进内存, 出生; 当类中的所有内容都执行完毕, 类才会出内存, 消亡;

b : 非静态成员变量 : 与对象的生命周期一致; 当创建对象时, 出生; 当对象再也没有使用的地方, 变成垃圾等待回收, 成员跟着消亡

1. 访问方式不同:

a : 静态成员变量, 2种访问方式

1. 类名.直接调用;
2. 对象名.调用;

b : 非静态成员变量 : 创建对象, 使用对象名.调用

### 工具类和生成帮助文档(了解)

1. 工具类 : 也是一个类, 类的功能就是为了解决实际开发中常见问题, 而工具类中一般都是方法功能, 设计属性比较少; 很多工具类中方法功能是static静态修饰
2. 文档注释 : (掌握)

在代码中也是起到解释说明代码的目的, 但是文档注释使用在类或者方法声明之上, 文档注释主要就是说明类或者方法主要功能, 文档注释存在必要性有2点:

1. 文档注释使用注解形式存在的, 整体注释上非常工整, 具有很好阅读性
2. 文档注释中的内容, 会被同步生成到代码的帮助文档中

1. 如何书写文档注释:

/\*\*

@author 作者lyl

@version 版本, 初始1.0, 小变动 1.1 1.2.. 大变动, 2.0

@param 参数, 使用在方法之上, 用于说明方法的参数个数和参数存在意义, 一个param只能形容一个参数

@return 返回值, 使用在方法之上, 用于描述方法的返回

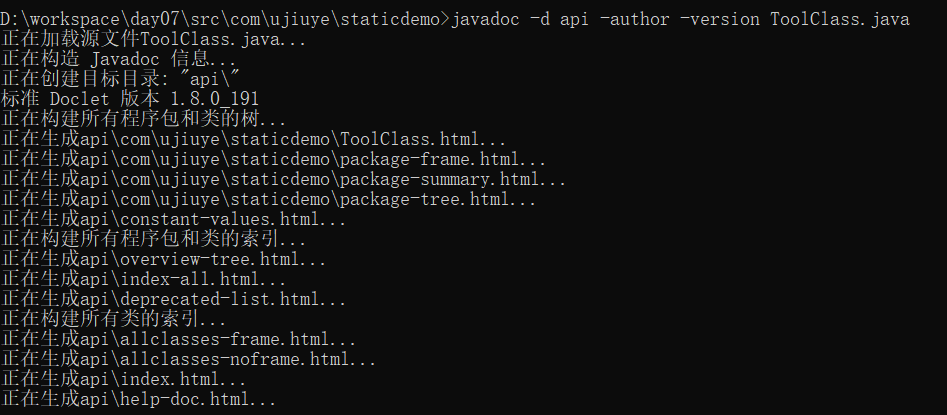
\*/

1. 实际开发中的代码需要交付给客户, 交付的都是.class字节码文件, 而字节码文件客户一般读不懂, 需要给客户提供代码对应的帮助文档(操作手册), 代码中文档注释中内容就会同步到帮助文档中(实际开发中, 不需要开发人员生成帮助文档)

JDK的bin文件夹路径下, 有应用程序命令 javadoc.exe

到源文件所在路径下, 打开dos窗口, 执行以下命令

命令 : javadoc -d api -author -version 源文件



到api文件夹下,找到index.html网页文件, 双击打开文件就可以查看帮助文件中的内容

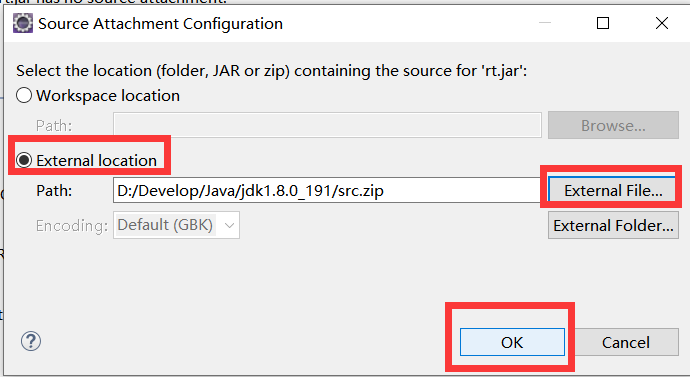
代码

|  |
| --- |
| package com.ujiuye.staticdemo;  /\*\*  \* @author lyl  \* @version 1.0  \*  \*/  public class ToolClass {  /\*\*  \* 方法求任意两个整数的和  \* @param first, 第一个int类型参数  \* @param second, 第二个int类型参数  \* @return first + second,得到两数的和  \*/  public int getSum(int first, int second) {  return first + second;  }  } |

### Arrays工具类

1. Arrays : JDK给开发人员提供的类, 工具类, 类中封装了很多对于数组操作常见的方法功能, 来自于java.util包, 使用需导包
2. 查看某个变量, 某个方法, 某个类源代码, ctrl + 变量名/方法名/类名, 进入到源代码查看

导入JDK的源代码到项目工程中: 找到JDK安装路径, 将src.zip源文件导入进去



1. Arrays工具类中JDK的API帮助文档中,没有给出任何构造, 证明Arrays工具类没有可用的构造, Arrays工具类不能new对象因为其构造方法使用private私有修饰, 于是Arrays工具类中的所有方法全部static静态修饰
2. sort(int[] arr): 将参数数组arr中的元素, 按照默认的升序进行排序, 从小到大排列, 方法没有返回值类型
3. toString(int[] arr) : 将参数数组arr中的每一个元素获取到, 以字符串形式返回, 方法返回值类型String
4. binarySearch(int[] arr, int key): 折半或者二分查找法, 查找key值在数组arr中出现的索引位置; 需要参数数组arr升序排列, 如果没有找到指定元素的索引, 返回一个负数

代码

|  |
| --- |
| package com.ujiuye.staticdemo;  import java.util.Arrays;  public class ArraysDemo {  public static void main(String[] args) {  // 1) sort(int[] arr): 将参数数组arr中的元素, 按照默认的升序进行排序, 从小到大排列, 方法没有返回值类型  int[] arr = {1,24,5,88,-8,9};  System.out.println(Arrays.toString(arr));  Arrays.sort(arr);    // 2)toString(int[] arr) : 将参数数组arr中的每一个元素获取到, 以字符串形式返回, 方法返回值类型String  String s = Arrays.toString(arr);  System.out.println(s);    // 3) binarySearch(int[] arr, int key): 折半或者二分查找法, 查找key值在数组arr中出现索引位置;  // 需要参数数组arr升序排列, 如果没有找到指定元素的索引, 返回一个负数  int index = Arrays.binarySearch(arr, 99);  //[-8, 1, 5, 9, 24, 88, 99]  System.out.println(index);// 3  }  } |