

# 工具类中引入静态

2018年6月20日 12:47

在同一文件夹下，类定义在两个文件中和定义在一个文件中其实是一样的。

工具类

```
class ArrayDemo{
    public static void main(String[] args){
        //定义数组
        int[] arr = {18,55,37,46,19};
        //遍历数组
        //printArray(arr);//静态只能访问静态
        //非静态方法
        //ArrayDemo ad = new ArrayDemo();
        //ad.printArray(arr);
        /*有了数组操作类的调用
        ArrayTool at = new ArrayTool();
        at.printArray(arr);
        */
        //静态直接用类名定义
        ArrayTool.printArray(arr);
        int max = ArrayTool.getMax(arr);
        System.out.println("max="+max);
        int index = ArrayTool.getIndex(arr,55);
        System.out.println("index="+index);
    }
}
```

```
class ArrayTool{
    //把构造方法私有，外界就不能创建对象了
    private ArrayTool(){

    }

    //只能通过类名访问静态方法
    public static void printArray(int[] arr){
        for(int x = 0;x<arr.length;x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+",");
            }
        }
    }

    public static int getMax(int[] arr){
        int max = arr[0];
        for(int x = 0;x<arr.length;x++){
            if(arr[x]>max){
                max = arr[x];
            }
        }
        return max;
    }

    public static int getIndex(int[] arr,int value){
        int index = -1;
        for(int x=0;x<arr.length;x++){
            if(arr[x]==value){
                index = x;
                break;
            }
        }
        return index;
    }
}
```

# API

2018年6月22日 15:06

API: Application Programming Interface  
应用程序编程接口(帮助文档)

# 说明书

2018年6月20日 13:43

@para参数

@return 返回值

制作说明书：


- 1.写一个工具类
- 2.对这个类加入文档注释
- 3.用工具解析

javadoc工具解析

4.格式

javadoc -d 目录 -author -version ArrayTool.java

目录：就可以写一个文件夹的路径

 index.html

class文件+说明书

出错：找不到可以文档化的公共或受保护的类  
权限不够 public

```
/**
 *这是针对数组进行操作的工具箱
 *@author 王若潇
 *@version V.10
 */
public class ArrayTool{
    //把构造方法私有，外界就不能创建对象了
    /**
     *这是私有构造
     */
    private ArrayTool(){

    }

    //只能通过类名访问静态方法
    /**
     *这是遍历数组的方法，遍历后的格式是：[元素1,元素2...]
     *@param arr 这是要遍历的数组
     */
    public static void printArray(int[] arr){
        for(int x = 0;x<arr.length;x++){
            if(x == arr.length-1){
                System.out.println(arr[x]);
            }else{
                System.out.print(arr[x]+"," );
            }
        }
    }

    /**
     *这是获取数组最大值的方法
     *@param arr 这是要获取最大值的数组
     *@return 返回数组中的最大值
     */
    public static int getMax(int[] arr){
```

```

        int max = arr[0];
        for(int x =0;x<arr.length;x++){
            if(arr[x]>max){
                max = arr[x];
            }
        }
        return max;
    }
}

```

```

/**

```

```

 *这是获取指定元素索引的方法，元素不存在返回-1

```

```

 *@param arr 被查找的数组

```

```

 *@param value 要查找的元素

```

```

 *@return 返回索引

```

```

 */

```

```

public static int getIndex(int[] arr,int value){

```

```

    int index = -1;

```

```

    for(int x=0;x<arr.length;x++){

```

```

        if(arr[x]==value){

```

```

            index = x;

```

```

            break;

```

```

        }

```

```

    }

```

```

    return index;

```

```

}

```

```

}

```

# 帮助文档

2018年6月20日 14:49

CHM文件

- 1.打开帮助文档
  - 2.点击显示，找到索引，看到输入框
  - 3.在输入框中输入Scanner，回车
  - 4.看包
- java.lang包下的类不需要导入，其它全部需要导入

java.util.Scanner

- 5.简单地看一下解释与说明，再看该类的版本
- 6.看类的结构 成员变量：字段摘要  
构造方法：构造方法摘要  
成员方法：方法摘要
- 7.学习构造方法
  - A.有构造方法：就创建对象
  - B.没有构造方法：成员都是静态的（不需要构造方法）
- 8.看成员方法：是否静态、返回值类型、看方法名称、看参数列表

# Math类

2018年6月20日 15:21

//Math类在lang包下, 因此不需要导包

//没有构造方法因为成员都是静态的

//获取随机数 public static double random()

```
class MathDemo{
    public static void main(String[] args){
        double d = Math.random();
        System.out.println(d);
        //需求: 获取1-100之间的随机数
        int num = (int)(Math.random()*100)+1;
        System.out.println(num);
    }
}
```

# 猜数小游戏

2018年6月20日 15:28

```
import java.util.Scanner;
class GuessNumber{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int num = (int)(Math.random()*100)+1;
        while(true){
            System.out.println("请输入100以内的数: ");
            int guessNumber = sc.nextInt();
            if(guessNumber>num){
                System.out.println("你猜的数字"+guessNumber+"大了");
            }else if(guessNumber<num){
                System.out.println("你猜的数字"+guessNumber+"小了");
            }else{
                System.out.println("猜对了");
                break;
            }
        }
    }
}
```

# 代码块

2018年6月20日 15:43

局部代码块 构造代码块 静态代码块

局部代码块：在局部位置，用于限定变量的生命周期

**构造代码块**：在类中的成员位置

用{}括起来，每次调用构造方法之前都要先执行构造代码块

作用：可以把多个构造方法中的共同代码放在一起

静态代码块：放在类中的成员位置，用{}括起来，用static修饰

一般是对类进行初始化，**只出现在第一次**

**静态代码块，构造代码块，构造方法的执行顺序**

静态代码块-->构造代码块-->构造方法

```
class Code{
    //静态代码块
    static{
        int a = 1000;
        System.out.println(a);
    }
    //构造代码块
    {
        int x = 100;
        System.out.println(x);
    }
    //构造方法
    public Code(){
    }
    //构造方法
    public Code(int a){
        System.out.println("code");
    }
    //构造代码块
    {
        int x = 200;
        System.out.println(x);
    }
    //静态代码块
    static{
        int b = 2000;
        System.out.println(b);
    }
}

class CodeDemo{
    public static void main(String[] args){
        {
            int x = 10;
            System.out.println(x);
        }
        //System.out.println(x);
        System.out.println("-----");
        Code c = new Code();
        Code c1 = new Code(2);
    }
}
/*
10
-----
1000
2000
*/
```



100  
200  
100  
200  
code  
\*/

# 读程序写结果

2018年6月20日 16:12

```
class Student {
    static {
        System.out.println("Student 静态代码块");
    }

    {
        System.out.println("Student 构造代码块");
    }

    public Student() {
        System.out.println("Student 构造方法");
    }
}

class StudentDemo {
    static {
        System.out.println("林青霞都60了，我很伤心");
    }

    public static void main(String[] args) {
        System.out.println("我是main方法");

        Student s1 = new Student();
        Student s2 = new Student();
    }
}
```

林青霞都60了，我很伤心  
我是main方法  
Student 静态代码块  
Student 构造代码块  
Student 构造方法  
Student 构造代码块  
Student 构造方法

# 继承

2018年6月20日 16:15

把相同的内容定义到一个类中

```
class Fu{
```

```
class Zi extends Fu{
```

**提高代码的复用性和维护性**

**让类与类之间产生了关系，是多态的前提**

```
class 子类名 extends 父类ming{
```

父类、基类、超类

子类、派生类

类与类产生关系也是一种弊端，类的耦合性增强了

**低耦合，高内聚**

耦合：类与类的关系

内聚：自身完成某件事物的能力

```
class Person{
    public void eat(){
        System.out.println("吃饭");
    }
    public void sleep(){
        System.out.println("睡觉");
    }
}
```

```
class Student extends Person{
```

```
class Teacher extends Person{
```

```
class ExtendsDemo{
    public static void main(String[] args){
        Student s = new Student();
        Teacher t = new Teacher();
        s.eat();
        s.sleep();
        t.eat();
        t.sleep();
    }
}
```

# 继承的特点

2018年6月20日 19:39

```
/*
```

java中继承的特点:

1.Java中只支持单继承, 不支持多继承(有些语言是支持多继承的)

2.java支持多层继承 (继承体系)

```
*/
```

```
/*
```

```
class Father{
```

```
class Mother{
```

```
class Son extends Father,Mother{//错误
```

```
*/
```

```
class GrandFather{
```

```
    public void show(){
```

```
        System.out.println("grandfather");
```

```
    }
```

```
}
```

```
class Father extends GrandFather{
```

```
    public void method(){
```

```
        System.out.println("father");
```

```
    }
```

```
}
```

```
class Son extends Father{
```

```
class ExtendsDemo2{
```

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

```
        Son s = new Son();
```

```
        s.show();
```

```
        s.method();
```

```
    }
```

```
}
```

# 继承的注意事项

2018年6月20日 19:55

/\*

继承的注意事项

- 1.子类只能继承父类所有的非私有成员
- 2.子类不能继承父类的构造方法
- 3.不要为了部分功能而使用继承

什么时候考虑继承：

继承其实实现的是一种关系："is a"

采用假设法

\*/

```
class Father{
    private int num = 10;
    public int num2 = 20;
    private void method(){
        System.out.println(num);
        System.out.println(num2);
    }
    public void show(){
        System.out.println(num);
        System.out.println(num2);
    }
}

class Son extends Father{
    public void function(){
        //System.out.println(num); 子类不能继承父类的私有成员
        System.out.println(num2);
    }
}

class ExtendsDemo3{
    public static void main(String[] args){
        Son s = new Son();
        s.show();
        //s.method() 子类不能继承父类的私有方法
        s.function();
    }
}
```

# 继承中的成员变量关系

2018年6月20日 22:27

- 1.子类中的成员变量和父类中的成员变量名称不一样
- 2.子类中的成员变量和父类中的成员变量名称不一样：在子类方法中访问一个变量的查找顺序：
  - a.在子类方法的局部范围找，有就使用
  - b.在父类的成员范围找，有就使用
  - c.在父类的成员范围找，有就使用
  - d.再找不到，报错

# supper

2018年6月21日 13:56

this代表本类对应的引用

super代表父类储存空间的标识（可理解为父类引用）

用法

1.访问成员变量

2.调用构造方法

this(...) 调用本类的构造方法

super(...)调用父类的构造方法

3.调用成员方法

```
/*
不仅要输出局部范围的num，还要输出成员范围的num
还要输出父类成员范围的num：super
*/
class Father{
    public int num = 10;
}
class Son extends Father{
    private int num = 20;
    public void show(){
        int num = 30;
        System.out.println(num);
        System.out.println(this.num);
        System.out.println(super.num);
    }
}
class ExtendsDemo4{
    public static void main(String[] args){
        Son s = new Son();
        s.show();
    }
}
```

# super调用构造方法

2018年6月21日 14:00

```
/*
1.子类的所有构造方法都会默认访问父类的无参构造方法
子类初始化之前，一定要完成父类的初始化
注意：子类每一个构造方法的第一条默认语句都是super()
*/
class Father{
    public Father(){

        System.out.println("father的无参构造方法");
    }
    public Father(String name){
        System.out.println("father的带参构造方法");
    }
}
class Son extends Father{
    public Son(){
        super();//默认有
        System.out.println("son的无参构造方法");
    }
    public Son(String name){
        System.out.println("son的带参构造方法");
    }
}
class ExtendsDemo5{
    public static void main(String[] args){
        Son s1 = new Son();
        System.out.println("-----");
        Son s2 = new Son("王若潇");
    }
}
/*
father的无参构造方法
son的无参构造方法
-----
father的无参构造方法
son的带参构造方法
*/
```

如果父类没有无参构造方法，那么子类的构造方法会出现什么现象  
**会报错**

如何解决：

- 1.在父类中加一个无参方法
- 2.通过使用super关键字去显示的调用父类的带参构造方法
- 3.子类通过this去调用本类的其它构造方法

子类中一定要有一个去访问了父类的构造方法，否则父类数据就没有初始化

注意：this () 和super()必须出现在第一条语句上，否则会对父类进行次初始化



# 练习1

2018年6月21日 15:08

//看程序写结果

```
class Fu{
    static{
        System.out.println("静态代码块Fu");
    }
    {
        System.out.println("构造代码块Fu");
    }
    public Fu(){
        System.out.println("构造方法Fu");
    }
}
```

```
class Zi extends Fu{
    static{
        System.out.println("静态代码块Zi");
    }
    {
        System.out.println("构造代码块Zi");
    }
    public Zi(){
        System.out.println("构造方法Zi");
    }
}
```

```
class ExtendsTest2{
    public static void main(String[] args){
        Zi z = new Zi();
    }
}
```

**结果:**

静态代码块Fu  
静态代码块Zi  
构造代码块Fu  
构造方法Fu  
构造代码块Zi  
构造方法Zi

## 练习2

2018年6月21日 20:54

```
class X{
    Y b = new Y();
    X(){
        System.out.println("X");
    }
}
```

```
class Y{
    Y(){
        System.out.println("Y");
    }
}
```

```
class Z extends X{
    Y y = new Y();
    Z(){
        System.out.println("Z");
    }
    public static void main(String[] args){
        new Z();
    }
}
```

Y  
X  
Y  
Z

成员变量初始化：默认初始化-->显示初始化-->构造方法初始化  
先初始化父类数据、再初始化子类数据

# 继承中成员方法关系

2018年6月21日 21:09

- 1.子类中的方法和父类中的方法声明不一样，简单
- 2.子类中的方法和父类中的方法声明一样，  
通过子类调用方法：a.先找子类中有没有这个方法，有就使用  
b.再在父类中找此方法，没有就报错

# 方法的重写

2018年6月21日 21:16

方法重写：子类中出现了和父类中方法声明一模一样的方法

方法重载：方法名一样，参数列表不同的方法，与返回值无关



方法覆盖、方法复写

当子类需要父类的功能，而功能主体子类有自己特有的内容时，可以复写父类中的方法，这样沿袭了父类的功能，又定义了子类特有的功能。

```
class Phone{
    public void call(String name){
        System.out.println("给" + name + "打电话");
    }
}

class NewPhone extends Phone{
    public void call(String name){
        //System.out.println("给" + name + "打电话");
        super.call(name);
        System.out.println("可以听前期预报了");
    }
}

class ExtendsDemo5{
    public static void main(String[] args){
        NewPhone np = new NewPhone();
        np.call("王若潇");
    }
}
```

# 方法重写注意事项

2018年6月21日 22:18

- 1.父类中私有方法不能被重写：因为父类私有方法子类根本访问不到
- 2.子类重写父类方法时，访问权限不能更低  
如子类中默认 而父类中public
- 3.父类静态方法，子类也必须通过静态方法重写

**子类重写父类方法的时候最好声明一模一样**

# 学生类例子

2018年6月22日 14:27

无论父类中的成员变量是`private`、`public`还是其它类型的，子类都会**拥有（继承）**父类中的这些成员变量。但是父类中的私有成员变量，无法在子类中直接访问，可以通过从父类中继承得到的`protected`、`public`方法（如`getter`、`setter`方法）来访问。

```
class Person{
    private String name;
    private int age;
    public Person(){
    }
    public Person(String name,int age){
        this.name = name;
        this.age = 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;
    }
}
```

```
class Student extends Person{
    public Student(){
    }
    public Student(String name,int age){
        super(name,age);//初始化父类
    }
}
```

```
class ExtendsTest4{
    public static void main(String[] args){
        Student s1 = new Student();
        s1.setName("王若潇");
    }
}
```

```
s1.setAge(22);
System.out.println(s1.getName()+"---"+s1.getAge());
Student s2 = new Student("王若潇",22);
System.out.println(s2.getName()+"---"+s2.getAge());
    }
}
```