# Java

**1**. java 的强制转换：int a = (int) (c+b)

2. s=s+1 与s +=1不同。+=隐含了一个强制转换。

3. 赋值规则：

包：全部小写，多级包用 . 隔开，如com.china.gzy

类或接口：一个单词的，单词首字母大写，多个单词拼接的，每个单词首字母大写

方法或者变量：一个单词的，小写，多个单词拼接的，除第一个单词外，其他单词首字母大写，如getName

常量：一个单词的，全部大写，如PI，多个单词拼接的，全部大写，用 \_ 隔开，如MAX\_AGE

**4.** &&相比较&具有短路的效果，即左边false，右边不执行，因此效率高，开发中常用&&. || 也是同理。

**5.** ^ 在逻辑运算中表示相同则0，不同则1，类比于男女关系。

**6.** 键盘录入：共三步

import java.util.Scanner;

Scanner sc = new Scanner(System.in);

String x = sc.next();

**7.** 一般来说，有{}就没有; 有;就没有{}

**8.** for 循环中，break 表示退出当前循环，即直接退出本层循环。可以结合标签选择跳出的循环层，例如：

f1: for(;;){

f2: for(;;){

if(ture){

break f1;

}

}

}

continue 是退出本次循环，还是会继续下一次循环的。

return 是不仅退出循环，还退出了整个方法。

**9.** 方法：在java中称之为方法，在其他编程语言中称为函数。注意：方法不调用则不会执行；方法与方法是平级关系，不能嵌套定义，例如，不能在mian方法中定义方法；定义方法时，方法中的参数用逗号隔开；方法调用的时候不用再次传递参数类型；当参数不同时，方法名可以相同，称为“方法重载”

void 返回值方法只能单独调用，不可赋值给其他值。

格式：

修饰符 返回值类型 方法名(参数类型1 参数名1,…..,) {

方法体语句；

return 返回值；

}

修饰符: public static

返回值类型: 就是结果的数据类型

方法名：符合命名规则即可

参数：分为实际参数和形式参数

**10.** 数组

定义： int[] a; 或者 int a[]; 建议采用第一种

数组必须初始化后才可以使用，初始化方法：

i. 数据类型[] 数组名=new 数据类型[数组长度] //只指定长度

int[] arr = new int[3]; //数组默认值全是0

ii. 数据类型[] 数组名=new 数据类型[]{元素1, 元素2,….} //指定内容，不指定长度

iii. 数据类型[] 数组名={元素1, 元素2,….} 是第2种的简化方式

数组的常用操作：

arr.length //获取数组长度

二维数组：

int[][] arr = new int[m][n] //m表示有几个一维数组，n表示每个一维数组有n个元素。此时，arr.length = m

int[][] arr = new int[m][] //定义了二维数组，数组元素个数可变

数据类型[][] 变量名={{元素…}{元素….}{元素…}[{…}}

**11.** 面向对象的编程思想：把多个方法封装到一个类中。

**12.** 类的定义包括：变量和方法，方法包括成员方法和构造方法。

类名 对象名 = new 类名();

成员变量：在类中方法外；随着对象存在或消失；有默认初始化值；

局部变量：在方法定义中或者方法声明上；随着方法存在或消失；没有默认初始化值，必须定义赋值后才能使用；

**13.** 匿名对象：没有名字的对象，如 new Student();

匿名对象可以调用方法，new Student().method(); 调用完被回收。匿名对象可以作为实际参数传递。

**14.** 封装：隐藏对象的属性和使用细节，仅对外提供公共访问方式。使用private修饰，private修饰的变量和方法只能在本类中进行访问。

**15.** this 代表所在类的对象引用，如this.age等价于Student.age。方法被哪个对象调用，this就代表哪个对象。

**16.** 构造方法：给对象的数据进行初始化。如：

class Student{

public Student(){}

}

如果没有定义构造方法，系统会自动提供一个无参构造方法。如果定义了构造方法，系统将不再提供默认的无参构造方法，必须自己给出才可调用。

构造方法可以重载。

**17.** static 修饰的变量为所有成员所共享，随着类的加载而加载，可以被类调用，也可以被方法调用；静态方法中没有关键字this，静态方法只能访问静态的成员变量和静态的成员方法。

**18.** 测试类的作用：创建其他类的对象，调用其他类的功能。

**19.** java.lang包下的类不需要导入，其他的全部需要导入。

**20.** java中，用{}括起来的代码称为代码块。

构造代码块：在类中的成员位置，每次调用构造方法执行前，都会先执行构造代码块。

静态代码块：在类中的成员位置，用static修饰，一般是对类进行初始化，只执行一次，意思是，多次实例化一个类也只执行一次静态代码块。

一个类中，最先执行静态代码块（甚至优先于main方法），然后构造代码块，最后是构造方法。

class Code{

static {// 这是静态代码块

int a =100;

}

{ //这是构造代码块

}

}

**21.** 继承：extends

class 子类名 extends 父类名{}

java中继承的特点：java只支持单继承，不支持多继承；java支持多层继承；子类只能继承父类所有非私有的成员（成员方法和成员变量）；子类不能继承父类的构造方法，可以通过super关键字来访问父类构造方法；子类的所有构造方法默认都会访问父类中空参数的构造方法；子类每一个构造方法的第一条语句默认都是super();

**22.** this和super的区别：

this 代表本类对应的引用，super代表父类存储空间的标识

**23.** 子类调用方法规则：先看子类中有没有，再看父类。

**24.** 方法重写：子类中出现了和父类中一模一样的方法声明。方法重写即沿袭了父类的功能，又定义了子类特有的功能。在重写的方法中，要想保留父类方法的功能，需要在子类重写的方法中添加super.xxx()

**25.** final 可以修饰类，方法，变量。如 final class Hello{} public final void show()

final 修饰的类无法被继承，final修饰的方法无法被重写，final修饰的变量无法被重新赋值，即只能被赋值一次如同常量，如 final in num; num = 100;

**26.** 多态：同一对象，在不同时刻体现的不同状态。多态的前提：要有继承关系；要有方法重写；要有父类引用指向子类对象，如 父类 f = new 子类，此时f只能访问父类的变量，因为存在方法重写，f访问的是子类的方法，

**27.**多态中父类不能使用子类的特有方法，解决方法：向下转型。向上转型：

父类 f = new 子类()，把一个子类类型安全地变为父类类型的赋值; 向下转型：父类 f = new 父类(); 子类 z =(子类) f，把一个父类类型强制转型为子类类型

**28.** 抽象类：如果父类的方法本身不需要实现任何功能(没有{})，仅仅是为了定义方法签名，目的是让子类去覆写它，那么，可以把父类的方法声明为抽象方法。一个类如果有抽象方法，这个类也必须是抽象类，用abstract修饰，如abstract class A{ public abstract void method();}

抽象类无法被实例化；抽象的子类要么也是抽象类，要么全部重写父类的全部抽象方法。

abstract不能与private, final, static 共存。

**29.** 接口：如果一个抽象类没有字段，所有方法全部都是抽象方法，这个类就可以定义为接口。方法：interface Person{}

当一个具体的class去实现一个interface时，需要使用implements关键字，如：

class Student implements Person {}

一个类可以实现多个interface, 例如class Student implements Person, Hello { // 实现了两个interface ... }

注意事项：接口仍是抽象的，不能实例化；接口内可以定义变量，但是全部为常量；接口可以继承接口，即可以单继承，也可以多继承；抽象类被继承，体现的是“is a”的关系，接口被继承，体现的是“like a”的关系。

**30.** 包的概念：包就是文件夹，对类就行分类管理，把相同的类名放到不同的包中，定义方法：package 包名; 多级包用 , 分开package放到程序的最上。

**31.** 不同包下的类之间的访问：使用import 包名.类名;

注意：虽然import 包名.\*; 可以导入所有的类，但是不推荐，应当用哪个类就导入哪个类。

**32.** java的权限修饰符：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **本类中** | **子类中** | **同包类中** | **其他类中** |
| **public** | 可以 | 可以 | 可以 | 可以 |
| **protected** | 可以 | 可以 | 可以 | 不可以 |
| **默认** | 可以 | 同包子类可以 | 可以 | 不可以 |
| **private** | 可以 | 不可以 | 不可以 | 不可以 |

类：只能public或者为默认，不能用protected和private修饰，内部类可以加private修饰。

构造方法：4中权限修饰符都可以修饰构造方法，且构造方法只能使用权限修饰符修饰，不能用static，final，abstract修饰。

**33.** 内部类：把一个类定义在其他类的内部，这个类就是内部类。内部类的访问特点：内部类可以访问外部类的所有成员，包括私有；内部类可以加private修饰；内部类和外部类不是继承关系。内部类可以分为成员内部类（定义在成员变量处）和局部内部类（定义在类的方法中）

**成员内部类的访问：**

class Person{

int num = 10;

class Student{

public void show() {

System.out.println(num);

}

}

}

要想访问内部类Student的show方法，代码如下：在main函数中，

// 外部类.内部类 对象名 = new 外部类().new 内部类();

Person.Student ps = new Person().new Student();

ps.show();

如果内部类用static作限定，访问方法为：

Person.Student ps = new Person().Student(); //访问静态方法

ps.show2();

**局部内部类的访问：**

局部内部类可以访问局部变量，这个局部变量必须用final修饰。

**34.** 匿名内部类：内部类的简化写法，前提是存在一个类或者接口。格式：

new 类或接口名(){ 重写方法(); }

**35.** Eclipse重要快捷键：Ctrl + Shift + F : 代码格式化

Ctrl + Shift + O : 自动导包

Ctrl + / : 选中行的添加、取消注释，生成的是//

Ctrl +Shift + / : 选中行的添加注释，生成的是/\* \*/

Ctrl +Shift + \ : 选中行的取消注释

F3 或者 按住Ctrl单机：查看选中类的源码

**36.** Eclipse 可以把一个项目导出为jar包。项目中添加jar包的方法：复制jar包到项目中，然后右键这个jar包，选择Build Path🡪Add….

**37.** Scanner中的常见方法：sc.hasNextXxx() 返回布尔类型，判断下面键盘输入的格式是否为Xxx。

Scanner sc = new Scanner(System.***in***);

if(sc.hasNextInt()) { //判断等下输入的是不是int类型

int x = sc.nextInt();

System.***out***.println(x);

}else {

System.***out***.println("格式不对");

}

**38.** 使用Scanner获取多个对象时，如果先获取int再获取String会出错，应当先把所有的对象设为String，然后再转换为相应的类型。

**39.** String 生成的变量一经赋值便无法修改

String s = "Hello";

s += "World";

这里，并非s的值发生了改变，而是生成了一个新地址指向s.

40. ==与equals的区别：

==比较的是地址值，equals比较的是内容。

String s1 = new String("abc");

String s2 = "abc";

System.***out***.println(s1==s2); //false

System.***out***.println(s1.equals(s2));//true

41. String 的几个功能：

.equals() 返回布尔值，判断是否相等

.equalsIgnoreCase 判断是否相等，忽略大小写

.contains() 是否包含

.isEmpty() 是否为空

42. String的获取功能：

substring(int start, int end) //截取子串，从指定位置开始，指定位置结束，包括start不包括end，而不是取几个。int end 可以省略，则意味着取到最后。

charAt(int index) 获取字符串指定索引的字符。

43. 字符串的遍历操作：

String s = "abcdefghijklmnopqrstuvwsyz";

for(int i=0;i<s.length();i++) {

System.***out***.println(s.charAt(i));

}

44. 字符串的转换：

返回值 方法

char[] toCharArray(); 把字符串转换为字符数组

String concat(String str); 字符串拼接

static String ValueOf(char[] chs); 把字符数组转换为字符串

String toLowerCase(); 把字符串转为小写；

String toUpperCase(); 把字符串转为大写；

String replace(old, new); 替换

String trim(); 只去除字符串两端的空格，中间的空格不去除

int compareTo(String str); 比较两个字符串是否相当，如果相等返回0

String[] split(String regex); 字符串分割，返回字符串数组，如String[] arr=str.split(“ ”); 按空格分隔

45. StringBuffer: 线程安全的可变字符串，与String相比，前者内容可变使用StringBuffer 的字符拼接功能节约空间，使用前需要生成对象：

StringBuffer sb = new StringBuffer(); //构建了一个缓存区

46. StringBuffer sb = new StringBuffer();

sb.append(“hello”);// 向后添加字符

sb.insert(int x," "); // 指定位置插入

sb.deleteCharAt(int index); //删除指定位置的字符

sb.delete(int start, int end); //删除起始位置的字符，不包括end，必须填入两个参数，不可省略

sb.toString(); //转成字符串

sb.replace(start, end, str); 把指定区域的字符进行替换

sb.reverse(); 反转

47. Arrays类的操作：

int[] arr = {……};

Arrays.sort(arr); //从小到大排序

Arrays.binarySearch(arr, key) //查找数组arr中数值key的索引

String s = Arrays.toString(arr); //数组转成字符串

48. int与String 的转换：

int num = 10;

String s = String.*valueOf*(num);// int\_to\_str

String str = "100";

int n = Integer.*parseInt*(str); //str\_to\_int

49. String 和StringBuffer之间的转换

String\_to\_StringBuffer: 使用StringBuffer的append方法

StringBuffer\_to\_String: 使用StringBuffer的toString方法

50. 正则表达式：

|  |  |
| --- | --- |
| **形式** | **含义** |
| [abc] | a,b,c任一个 |
| [^abc] | 除了a |
| [a-zA-Z] | 所有字母中的任一个 |
| \w | 任意字母，注意，要想使用，需要\\w, 其他的类似 |
| \W | 非字母 |
| [0-9] | 0到9 |
| \d | 任意一个数字 |
| \D | 非数字 |
| {n,m} | {}前面的重复n到m次 |
| . | 任意字符。想要表示.需要\\. |
| ^ | 行的开头 |
| $ | 行的结尾 |
| \b | 单词边界，即不是单词的地方 |
| ? | 0-1次 |
| \* | 0次或多次 |
| + | 一次以上 |
|  |  |

matches(String regex)，返回布尔类型, 判断字符串是否符合regex的规则

split(String regex) 返回类型为String[], 按照regex规则进行字符串拆分

replaceAll(String regex, String replacement) 替换功能

51. 正则表达式的获取功能：

import java.util.regex.Pattern;

Pattern p = Pattern.*compile*(String regex);

Matcher m = p.matcher(String str); //通过m调用相关的匹配函数

52. Random 类的常见方法

Random r = new Random(int n); //n是设置的随机数种子，可以省略

int num = r.nextInt(100); r.nextInt(100) 返回一个0-100之间的随机数

53. System的一些常用类

System.exit(0); //终止当前的jvm，下面的程序将不再执行，类似于break;

System.currentTimeMillis() //返回以毫秒为单位的当前时间，类型为long

54. Date类

Date d = new Date();

System.out.println(d); //按标准日期输出当前时间

long time = d.get.time(); //以毫秒为单位获取时间

SimpleDateFormat sdf = new SimpleDateFormat();

String s = sdf.format(d); //这一步将Date得到的时间按照默认的格式进行格式化

d.setTime(long time); //设置时间

//以下代码按照便于理解的格式输出当时日期

Date d = new Date();

SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");

String s = sdf.format(d);

System.***out***.println(s);

55. Calendar 类

Calendar now = Calendar.*getInstance*();

int year = now.get(Calendar.***YEAR***);

int month = now.get(Calendar.***MONTH***); //获取月份，从0开始，加上1为正常的月份

int day = now.get(Calendar.***DATE***);

now.add(Calendar.***YEAR***,4); //设置时间为4年后

now.add(Calendar.***DATE***,-10); //设置时间为10天前

now.set(int year,int month,int dat) //设置日期为某年某月某日，注意，month是从0开始的，如果month为2，实际上是3月

56. 集合与数组的区别：数组的长度固定，集合的长度可变；数组的类型相同，集合的类型可以不同；集合只能存储引用类型

57. 集合的常用功能

// Collection c = new Collection(); //错误，接口不能实例化

Collection c = new ArrayList();

|  |  |
| --- | --- |
| **代码** | **作用** |
| c.add(String str) | 向集合c中添加元素，元素可以重复 |
| c.clear() | 清除集合c中的元素 |
| c.remove(String str) | 移除元素，如果c中不存在str则不移除，并返回false |
| c.contains(String str) | 判断是否包含 |
| c.isEmpty() | 判断是否为空 |
| c.size() | 求集合c的元素个数 |
| c.addAll(Collection c1) | 向集合c中添加另一个集合c1, 集合c中的元素可以重复 |
| c.containsAll(Collection c1) | 判断c是否包含c1, 只有c1是c的子集才会返回true |
|  |  |

58. 集合的遍历

//方式1

Object[] obj = c.toArray(); //把集合c转为Object数组

for(int i=0;i<obj.length;i++) {

String s = (String) obj[i]; //转为String，因为Object没有lenth()函数

System.***out***.println(s);

System.***out***.println(s.length());

}

//方式2 使用迭代器

Iterator it = c.iterator(); //生成迭代器

while(it.hasNext()) {

String s = (String) it.next(); //next()是获取一个元素，并指向下一个元素

System.***out***.println(s);

}

59. List: Lists 是有序的集合Collection, 特点是可重复。有序是指存储和取出的元素一致。

List list = new ArrayList();

list.add(String str); //添加元素

利用迭代器访问List的元素。

60. List的特有功能：

List list = new ArrayList();

|  |  |
| --- | --- |
| **代码** | **作用** |
| list.add(int index, Object element) | 在指定位置添加元素 |
| list.get(int index) | 获取指定位置的元素 |
| list.remove(int index) | 删除指定位置的元素 |
| list.set(int index, Object element) | 根据索引修改元素 |

遍历：

for(int i=0;i<list.size();i++) {

String s = (String) list.get(i);

System.***out***.println(s);

}

或者使用迭代器

61. 常见数据结构

栈：先进后出

队列：先进先出

链表：由一个链子把多个结点连起来组成的数据，结点包含数据和地址（数据域和指针域）。链表查找数据都是从头开始。

62. List的三个子类的特点ArrayList, Vector, LinkedList

ArrayList: 底层数据是数组，查询快，增删慢，线程不安全，效率高。

Vector: 底层数据是数组，查询快，增删慢，线程安全，效率低。

LinkedList: 底层数据是链表，查询慢，增删快，线程不安全，效率高。

63. 泛型：是一种把类型明确的工作推迟到创建对象或者调用方法的时候才去明确的特殊类型，也叫参数化类型。优点是把运行时期的问题提前到了编译时期，避免了强制类型转换。

ArrayList<String> array = new ArrayList<String>();

Iterator<String> it = array.iterator();

64. 泛型方法的使用：把泛型定义在方法上

public static class Student{

public <T> void show(T t) {

System.***out***.println(t);

}

}

然后在main方法中，

Student s = new Student();

s.show("Hello");

s.show(100);

65. 泛型的通配符：

? 表示任意的类型都可以

Collection <?> c = new ArrayList<Object>();

? extends E 向下限定，E及其子类

? super E 向上限定，E及其父类

66. 增强for

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

for(int s : arr) {

System.***out***.println(s);

// 也可以进行其他操作

}

可以用于集合的遍历

67. 可变参数：定义的时候未指定

参数的个数，格式为：

修饰符 返回值类型 方法名(类型.. 变量名){}

public static int sum(int... a){} //这里的a 实质是一个数组

68. Set: 一个不包含重复元素的Collection，特点：无序，存储和取出顺序不一致，有两个子类:HashSet 和 TreeSet

Set<String> set = new HashSet<String>();

69. TreeSet 可以对元素按照某种规则进行排序，底层是二叉树结构。排序方法：自然排序，比较器排序。

自然排序：

TreeSet<Integer> ts = new TreeSet<Integer>();

比较器排序：重写comepareTo

public int compareTo(Student s) {

int num = this.name.length()-s.name.length();

***return*** name;

}

70. Map集合的特点：可以存储键值对的元素；键是唯一的，值可重复；键的存储是无序的；

Map<String,String> map = new HashMap<String,String>();

//此map的k,v均为String

由于map有一个HashMap子类，所以也可以写成：

HashMap<String,String> map = new HashMap<String,String>();

|  |  |
| --- | --- |
| **代码** | **功能** |
| map.put(String key, String value) | 添加元素。如果键是第一次添加，就直接存储并返回null；如果键已存在，则把value覆盖，并返回以前的value。 |
| map.clear() | 清除所有元素 |
| map.remove(Object key) | 按照key的值进行删除 |
| map.containsKey(Object key) | 判断是否存在这个vkey |
| map.isEmpty() | 判断是否为空 |
| map.size() | 返回map的长度 |
| map.get(Object key) | 返回该key所对应的value |
|  |  |
|  |  |

//一下子获取所有的key

Set<String> set = map.keySet();

for(String s:set) {

System.***out***.println(s);

}

//一下子获取所有的value

Collection<String> con = map.values();

for(String values:con) {

System.***out***.println(values);

}

//同步获取key-value

Set<String> set = map.keySet();

for(String s:set) {

System.***out***.println(s+"--"+map.get(s));

}

71. TreeMap: 基于红黑树，可以保证排序和唯一性。

TreeMap<String,String> tm = new TreeMap<String,String>();

Set<String> set1 = tm.keySet();

for(String s:set1) {

String ss = tm.get(s);

System.***out***.println(s+"----"+ss);

}

72. Collections 常用工具类

List<Integer> list = new ArrayList<Integer>();

list.add(xxx) //省略多个添加语句

|  |  |
| --- | --- |
| **代码** | **功能** |
| Collections.sort(list) | 对list的内容进行排序 |
| Collections.binarySearch(list, key) | 查找list中key值的索引，没有找到返回-1 |
| Collections.max(list) | 返回list中的最大值 |
| Collections.shuffle(list); | 把list中元素进行随机打乱 |
|  |  |

73. 异常处理方案：

A: try{

可能出现问题的代码;

}catch(异常名1 | 异常名2| … 变量){

针对问题的处理;

}catch(Exception e){} Exception可以匹配所有的异常，要放到最后一个catch

finally{

// finally中的语句一定会执行，不管是否遇见错误，除非在catch等语句中退出了jvm，如System.exit(0);

}

注意：一旦try里面的内容出现错误，就会结束try中的内容去匹配catch，不会再返回try

B: throws

74. 异常中的几个方法：

catch(Exception e){}

|  |  |
| --- | --- |
| **代码** | **功能** |
| e,getMessage() | 返回异常信息，返回值为字符串 |
| e.toString() | 返回异常信息，比较详细，包括getMessage |
| e.printStackTrace(); | 把错误信息直接输出在控制台 |
|  |  |

75. throws: 在定义的方法上使用

public static void method() throws 异常名1,异常名2..{}

76. throw: 如果出现了异常，可以抛出异常，抛出的是一个对象

public static void method() throws Exception {

int a = 10;

int b = 0;

if(b==0) {

throw new Exception();

}else {

System.***out***.println(a/b);

}

}

77. throws和throw的区别：

throws: 用在方法声明后面，后面跟的是异常类名；可以跟多个异常，用逗号隔开；表示抛出异常，由调用该方法的调用者来解决；表示出现异常的一种可能性，不一定会抛出异常；

throw: 用在方法体内，后面跟的是异常对象名；只能抛出一个异常对象名；表示抛出异常，由方法体内的语句处理；表示一定出现了异常；

78. 自定义一个异常类，必须继承Exception或者RuntimeException，自己构建的异常类需要提供一个无参构造方法和一个有参构造方法。

79. File类：文件名和路径的抽象表示形式

File file1 = new File("E:\\java\\demo"); //必须指定一个文件夹目录或一个文件

File file2 = new File("E:\\java\\demo\\a.txt");

File file3 = new File("E:\\java\\demo\\b.txt");

File file4 = new File("E:\\");

|  |  |
| --- | --- |
| **代码** | **功能** |
| file1.mkdirs() | 根据指定的目录"E:\\java\\demo"创建一个文件夹，返回布尔值，创建成功为true。如果该文件已经存在，则创建不成功，返回false；这个命令只能创建文件夹 |
| file1.mkdir() | 只能创建一级的文件夹 |
| file2.createNewFile(); | 创建指定文件，该目录必须提前创建 |
| file2.delete() | 删除指定文件或文件夹，返回布尔值。如果删除的是文件夹，这个文件夹下不能有其他文件或文件夹 |
| file2.renameTo(file3); | 修改名称 |
| file2.isDirectory() | 判断是否为一个目录 |
| file2.isFile() | 判断是否为一个文件 |
| file2.getAbsolutePath() | 获取绝对文件地址 |
| file2.getPath() | 获取相对文件地址 |
| file4.list() | 返回一个字符串数组，获取指定路径下的所有文件或文件夹名称  String[] list=file4.list(); |
| file4.listFiles() | 返回一个File数组，获取指定路径下的所有文件  File[] file = file4.listFiles(); |
|  |  |

80. IO流的分类：

按流向分：输入流 读取数据；输出流 写出数据

按数据类型分：字节流（InputStream, OutputStream）；字符流（Reader, Writer）

81. 创建字节流输出对象：

FileOutputStream fos = new FileOutputStream("E:\\java\\a.txt",true);

//fos 将指向a.txt. a.txt可以不存在，会自动创建，但是这个文件夹目录必须存在。true表示追加写入，省略ture表示每次都重新开始

String s1 = "春江潮水连海平，海上明月共潮生。";

fos.write(s1.getBytes()); //写入字符串。getBytes()是转成字节型，是必须的。

fos.close(); //关闭文件，释放空间，此后无法再写入。

fos.write("\r\n".getBytes()); //换行，相当于回车键

82. 创建字节流输入对象：

FileInputStream fis = new FileInputStream("……");

fis.read(); //读取一个字节，并返回下一个字节。当到末尾时。返回-1

//使用循环输出文件方法1

int by = fis.read();

while(by !=- 1) {

System.***out***.print((char)by); //这里字节流读取中文文件会乱码。

by = fis.read();

}

fis.close();

//使用循环输出文件方法2

int by = 0;

while((by = fis.read()) !=-1) {

System.***out***.print((char)by);

}

fis.close();

//使用循环输出文件方法3

byte[] arrb = new byte[1024]; //1024个字节代表一次读取lKb

int len = 0;

while((len = fis.read(arrb)) !=-1) { //len是实际读取的长度

System.***out***.println(new String(arrb,0,len)); //此方法读中文不是乱码

}

83. 转换流：把字节流转换成字符流

//写入数据

OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("E:\\java\\test.txt"),"UTF-8");

osw.write("中国");

osw.flush(); //刷新缓冲区

osw.close();

//读取数据方式

InputStreamReader isr = new InputStreamReader(new FileInputStream("E:\\java\\test.txt"),"UTF-8");

int ch = 0;

while((ch=isr.read()) !=-1) {

System.***out***.print((char)ch);

}

isr.close();

//新建一个文件new.txt，把文件test.txt的内容复制进去

InputStreamReader isr = new InputStreamReader(new FileInputStream("E:\\java\\test.txt"),"UTF-8");

OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("E:\\java\\new.txt"),"UTF-8");

char[] ch = new char[1024];

int len =0;

while((len = isr.read(ch)) !=-1) {

osw.write(ch, 0, len);

}

osw.close();

isr.close();

84. 读写文件的简化操作：

FileReader fr = new FileReader("E:\\java\\test.txt");

FileWriter fw = new FileWriter("E:\\java\\test.txt");

85. 字符缓冲流

BufferedReader br = new BufferedReader(new FileReader("E:\\java\\test.txt"));

BufferedWriter bw = new BufferedWriter(new FileWriter("E:\\java\\test.txt"));

bw.write("One");

bw.newLine();

bw.flush();

//按行读取

String line = null;

while((line=br.readLine())!=null) {

System.***out***.println(line);

}

br.close();

bw.close();

86. 读取文件并获取行号

LineNumberReader lnr = new LineNumberReader(new FileReader("E:\\student.txt"));

String line = null;

while((line=lnr.readLine())!=null) {

System.***out***.println(lnr.getLineNumber()+":" +line);

}

lnr.close();

87. 打印流：PrintWriter，可以写入任意类型的数据，但只能写，不能读

PrintWriter pw = new PrintWriter(new FileWriter("pw.txt"),true);

pw.println(100);

pw.println("Hello"); //会自动换行，且不需要close,不需要flush

88. 多线程概述：

1) 进程：正在运行的程序，是系统进行资源分配和调用的独立单位，每一个进程都有它自己的内存空间和独立资源。单进程的计算机只能做一件事，多进程可以是计算机在一个时间段执行多个程序，提高CPU的使用率。

2）线程：同一个进程内可以执行多个任务，每一个任务可以看成一个线程。线程是程序的执行单元，执行路径，是程序使用CPU的最基本单位。多线程提高了应用程序的使用率。线程的执行有随机性。

89. java多线程的实现方法：

1）方法1：创建一个类继承Thread；重写run()方法；创建对象，启动线程调用start()方法。

例如：

public class MyThread extends Thread {

@Override

public void run() {

for(int i = 0;i<200;i++) {

System.***out***.println(i);

}

}

}

在main()方法中，实例化MyThread对象，注意，要想实现多线程，应当实例化多个对象，然后每个对象都调用start方法。

MyThread mt1 = new MyThread();

MyThread mt2 = new MyThread();

MyThread mt3 = new MyThread();

mt1.setName(“线程1”); //更改线程名字

mt1.start();

//mt1.join()：此方法保证mt1执行完后，才会执行其他线程

mt2.start();

mt3.start();

mt1.interrupt();//中断线程，并抛出一个异常

从结果可以看出，三个start并非按顺序执行，而是随机的，如同在交替运行一样。

在public void run(){}方法中，常用函数：

getName(); // 获取线程的名字

Thread.sleep(long xxx); //使进程进行休眠，毫秒为单位

90. 线程的优先级

System.***out***.println(mt1.getPriority());

System.***out***.println(mt2.getPriority());

System.***out***.println(mt3.getPriority());//获取优先级，默认优先级为5

mt1.setPriority(int newPriority); //设置优先级，范围为1-10，数字越大优先级越高

91. java多线程的实现方法：

1）方法2：实现Runnable接口；重写run() 方法

public class MyRunnable implements Runnable {

@Override

public void run() {

for(int i=1;i<=10;i++) {

System.***out***.println(Thread.*currentThread*().getName()+":"+i);

}

}

}

在main()方法中：

MyRunnable mr = new MyRunnable();

Thread t1 = new Thread(mr);

t1.start();

92. 多线程中容易出现错误，使用synchronized (obj) 把run()中的程序包起来会避免错误。或者使用lock() 和 unlock()把可能出现的问题就行锁住。

# End