**JDK, JRE, JVM的作用及关系**

(1) 作用

JVM (JVM Java Virtual Machine)：保证Java语言跨平台

JRE (Java Runtime Environment )：Java程序的运行环境

JDK (Java Development Kit )：Java程序的开发环境

(2) 关系

JDK：JRE+工具

JRE：JVM+类库

**Java程序的开发执行流程：**

A: 编写java源程序(.java)

B: 通过javac命令编译生成.class文件

C: 通过java命令运行.class文件

**标识符**

(1) 就是给类，接口，方法，变量等起名字的字符序列

(2) 组成规则：

A: 英文大小写字母

B: 数字

C: $和\_

(3) 注意事项：

A: 不能以数字开头

B: 不能是java中的关键字

C: 区分大小写

(4) 常见的命名规则(见名知意)

A: 包 全部小写

单级包：小写

举例：liuyi,com

多级包：小写，并用.隔开

举例：cn.itcast,com.baidu

B: 类或者接口

一个单词：首字母大写

举例：Student,Demo

多个单词：每个单词首字母大写

举例：HelloWorld,StudentName

C: 方法或者变量

一个单词：首字母小写

举例：name,main

多个单词：从第二个单词开始，每个单词首字母大写

举例：studentAge,showAllNames()

D: 常量

全部大写

一个单词：大写

举例：PI

多个单词：大写，并用\_隔开

举例：STUDENT\_MAX\_AGE

**注释**

(1) 分类：

A: 单行注释 //

B: 多行注释 /\*\*/

C: 文档注释 /\*\* \*/

(2) 注释的作用

A: 解释说明程序，提高了代码的阅读性。

B: 可以帮助我们调试程序。

**常量**

(2) 分类：

A: 字面值常量 B:自定义常量

(3) 字面值常量

A: 字符串常量 "hello"

B: 整数常量 12,23

C: 小数常量 12.345

D: 字符常量 'a','A','0'

E: 布尔常量 true,false

F: 空常量 null

(4) 在Java中针对整数常量提供了四种表现形式

A: 二进制 由0，1组成。以0b开头。

B: 八进制 由0，1，...7组成。以0开头。

C: 十进制 由0，1，...9组成。整数默认是十进制。

D: 十六进制 由0，1，...9,a,b,c,d,e,f(大小写均可)组成。以0x开头。

**进制转换**

(1)其他进制到十进制

每位上的数值\*（进制数^数字编号） 从右到左，从0开始编号

(2) 十进制到其他进制

除进制数取余，直到商为0，余数反转。

**变量**

变量的定义格式：

A: 数据类型 变量名 = 初始化值;

B: 数据类型 变量名;

变量名 = 初始化值 ;

**数据类型**

(1)Java是一种强类型语言，针对每种数据都提供了对应的数据类型。

(2)分类：

A: 基本数据类型：4类8种

B: 引用数据类型：类，接口，数组。

(3) 基本数据类型

A: 整数 占用字节数

byte 1

short 2

int 4

long 8

B: 浮点数

float 4

double 8

C: 字符

char 2

D: 布尔

boolean 1

注意： 整数默认是int类型，浮点数默认是double。

长整数要加L或者l。

单精度的浮点数要加F或者f。

**数据类型转换**

(1) 默认转换

A: 从小到大

B: byte, short, char -- int -- long -- float -- double

C: byte, short, char之间不相互转换，直接转成int类型参与运算。

(2) 强制转换

A: 从大到小

B: 可能会有精度的损失，一般不建议这样使用。

C: 格式：

目标数据类型 变量名 = (目标数据类型) (被转换的数据);

(3) 两个不同类型变量相加，相加前小变量类型先转换成大变量类型

(4) 思考题和面试题：

A: 下面两种方式有区别吗? 有

float f1 = 12.345f; f1是通过double类型转换过来的

float f2 = (float)12.345; f2本身就是float类型

B: 下面的程序有问题吗，如果有，在哪里呢?

byte b1 = 3;

byte b2 = 4;

byte b3 = b1 + b2; 有，两个byte变量相加前先转换成int，

结果可能损失精度。

byte b4 = 3 + 4; 没有，常量相加先把结果计算出来，

如果在byte范围内，就不会报错。

C: 下面的操作结果是什么呢?

byte b = (byte)130; 结果 -126

分析过程： 计算机中数据的运算都是补码进行的。

A: 获取130这个数据的二进制。

00000000 00000000 00000000 10000010

这是130的原码，也是反码，还是补码。

B: 做截取操作，截成byte类型的了。

10000010 这个结果是补码。

C: 已知补码求原码。

符号位 数值位

补码： 1 0000010

反码： 1 0000001

原码： 1 1111110 结果：-126

D: 字符参与运算

是查找ASCII里面的值

'a' 97

'A' 65

'0' 48

System.out.println('a'); 结果a

System.out.println('a' + 1); 结果98

E:字符串参与运算

System.out.println("hello"+'a'+1); => helloa1

System.out.println('a'+1+"hello"); => 98hello

System.out.println("5+5="+5+5); => 5+5=55

System.out.println(5+5+"=5+5"); => 10=5+5

**运算符**

(1) 赋值运算符

A: =,+=,-=,\*=,/=,%=等

B: =叫做赋值运算符，也是最基本的赋值运算符

int x = 10; 把10赋值给int类型的变量x。

C: 隐含了自动强制转换。

面试题： short s = 1;

s = s + 1; 有问题，s+1自动转成int类型（等号右边有变量就不行）

short s = 1;

s += 1; 没问题，隐含了强制转换。

(2) 逻辑运算符

&, |, ^, !, &&, ||

&:有false则false

|:有true则true

^:相同则false，不同则true。

!:非true则false，非false则true

&&: 结果和&是一样的，只不过有短路效果。左边是false，右边不执行。

||: 结果和|是一样的，只不过有短路效果。左边是true，右边不执行。

(3)位运算符 ： *都是对二进制数字的补码进行操作*

&, | , ^ , ~ , << , >> , >>>

<< : 右边补0

>>： 正数：左边补0 负数：左边补1

>>>：无论正负，左边都补0

A: ^的特殊用法

一个数据针对另一个数据位异或两次，该数不变

b: 请用最有效率的方式计算出2乘以8的结果 2<<3

(4)三元运算符

格式：比较表达式?表达式1:表达式2;

(4)三元运算符和if语句第二种格式的关系

所有的三元运算符能够实现的，if语句的第二种格式都能实现。

反之不成立。

如果if语句第二种格式控制的语句体是输出语句，就不可以。

因为三元运算符是一个运算符，必须要有一个结果返回，不能是一个输出语句。

**方法重载**

在同一个类中，方法名相同，参数列表不同。与返回值无关。

参数列表不同：

参数的个数不同。

参数的对应的数据类型不同。

**数组**

(1) 数组的初始化

A: 动态初始化

只给长度，系统给出默认值

举例：int[] arr = new int[3];

B: 静态初始化

给出值，系统决定长度

举例：int[] arr = {1,2,3};

**Java的内存分配**

A: 栈 存储局部变量

B: 堆 存储所有new出来的对象（包括成员变量）

C: 方法区(面向对象部分详细讲解)

D: 本地方法区(系统相关)

E: 寄存器(CPU使用)

注意：

a: 局部变量 ：在方法定义中或者方法声明上定义的变量。

b: 栈内存和堆内存的区别

栈：数据使用完毕，就消失。

堆：每一个new出来的东西都有地址

每一个变量都有默认值

byte,short,int,long 0

float,double 0.0

char '\u0000'

boolean false

引用类型 null

数据使用完毕后，在垃圾回收器空闲的时候回收。

**二维数组**

格式：

A: 数据类型[][] 数组名 = new 数据类型[m][n];

B: 数据类型[][] 数组名 = new 数据类型[m][];

C: 数据类型[][] 数组名 = new 数据类型[][]{{...},{...},{...}};

D: 数据类型[][] 数组名 = {{...},{...},{...}};

**Java中的参数传递**

Java中只有值传递。

基本类型：形式参数的改变不影响实际参数（传递的是数值）

引用类型：形式参数的改变直接影响实际参数（传递的是地址）

**面向对象**

面向对象的思想特点

A:是一种更符合我们思考习惯的思想

B:把复杂的事情简单化

C:让我们从执行者变成了指挥者

注意：如何让我们的操作更符合面向对象思想呢?

A:有哪些类

B:每个类有哪些成员

C:类与类的关系

类与对象

A:Java语言中最基本的单位是类。

成员变量 + 成员方法

B: 类：是一组相关的属性和行为的集合。是一个抽象的概念。

对象：是该类事物的具体存在，是一个具体的实例。

**成员变量和局部变量的区别**

1. 在类中的位置不同

成员变量：类中方法外

局部变量：方法定义中或者方法声明上

1. 在内存中的位置不同

成员变量：在堆中

局部变量：在栈中

1. 生命周期不同

成员变量：随着对象的创建而存在，随着对象的消失而消失

局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

1. 初始化值不同

成员变量：有默认值

局部变量：没有默认值，必须定义，赋值，然后才能使用

**匿名对象**

(1) 没有名字的对象

(2) 应用场景

A: 调用方法，仅仅只调用一次的时候。

b: 可以作为实际参数传递。

**封装**

(1) 隐藏实现细节，提供公共的访问方式

(2) 好处：

A:隐藏实现细节，提供公共的访问方式

B:提高代码的复用性

C:提高代码的安全性

(3) 设计原则

把不想让外界知道的实现细节给隐藏起来，提供公共的访问方式

**private关键字**

(1) 私有的意义，可以修饰成员变量和成员方法

(2) 特点：

被private修饰的后的成员只能在本类中被访问

(3) private的应用：

把所有的成员变量给private了

提供对应的getXxx( )/setXxx( )方法

**this关键字**

1. 代表当前类的引用对象

哪个对象调用方法，该方法内部的this就代表那个对象

1. this的应用场景：

解决了局部变量隐藏成员变量的问题

**构造方法**

(1) 作用：用于对对象的数据进行初始化

(2) 格式：

A: 方法名和类名相同

B: 没有返回值类型，连void都不能有

C: 没有返回值

(3) 构造方法的注意事项

A: 如果我们没写构造方法，系统将提供一个默认的无参构造方法

B: 如果我们给出了构造方法，系统将不再提供默认构造方法

如果这个时候，我们要使用无参构造方法，就必须自己给出。

推荐：永远手动自己给出无参构造方法。

**初始化对象做了什么：**

Student s = new Student(刘意,30);

(1)把Student.class文件加载到内存

(2)在栈内存为s开辟空间

(3)在堆内存为学生对象申请空间

(4)给学生的成员变量进行默认初始化。null,0

(5)给学生的成员变量进行显式初始化。林青霞,27

(6)通过构造方法给成员变量进行初始化。刘意,30

(7)对象构造完毕，把地址赋值给s变量

**static关键字**

(1) 静态的意思。可以修饰成员变量和成员方法。

(2) 静态的特点：

A: 随着类的加载而加载

B: 优先于对象存在

C: 被类的所有对象共享

D: 可以通过类名调用

既可以通过对象名调用，也可以通过类名调用，建议通过类名调用。

(3) 静态的内存图

静态的内容在方法区的静态区

(4) 静态的注意事项

A: 在静态方法中没有this对象

B: 静态只能访问静态

(5) 静态变量和成员变量的区别

A: 所属不同

静态变量：属于类，类变量

成员变量：属于对象，对象变量，实例变量

B: 内存位置不同

静态变量：方法区的静态区

成员变量：堆内存

C: 生命周期不同

静态变量：静态变量是随着类的加载而加载，随着类的消失而消失

成员变量：成员变量是随着对象的创建而存在，随着对象的消失而消失

D: 调用不同

静态变量：可以通过对象名调用，也可以通过类名调用

成员变量：只能通过对象名调用

(6) main方法是静态的

public:权限最大

static:不用创建对象调用

void:返回值给jvm没有意义

main:就是一个常见的名称。

String[] args:可以接收数据，提供程序的灵活性

格式： java MainDemo hello world java

java MainDemo 10 20 30

**如何制作帮助文档**

(1)写一个类

(2)加入文档注释 @author @version 等

(3)通过javadoc工具生成即可

javadoc -d 目录 -author -version ArrayTool.java

**代码块**

(1)用{}括起来的代码。

(2)分类：

A:局部代码块

用于限定变量的生命周期，及早释放，提高内存利用率。

B:构造代码块： 就一个 { }

把多个构造方法中相同的代码可以放到这里，每个构造方法执行前，首先执行构造代码块。

C:静态代码块

对类的数据进行初始化，仅仅只执行一次。

(3)静态代码块,构造代码块,构造方法的顺序问题

静态代码块 > 构造代码块 > 构造方法

父静态代码块 >子 静态代码块 >父构造代码块 >父构造方法>子构造代码块 >子构造方法

**继承**

(1)把多个类中相同的成员给提取出来定义到一个独立的类中。然后让这多个类和该独立的类产生一个关系，这个关系叫继承。

(2)Java中如何表示继承呢?格式是什么呢?

A:用关键字extends表示

B:格式： class 子类名 extends 父类名 {}

(3)继承的好处：

A:提高了代码的复用性

B:提高了代码的维护性

C:让类与类产生了一个关系，是多态的前提

(4)继承的弊端：

A:让类的耦合性增强。这样某个类的改变，就会影响其他和该类相关的类。

原则：低耦合，高内聚。

耦合：类与类的关系

内聚：自己完成某件事情的能力

B:打破了封装性

(5)Java中继承的特点

A:Java中类只支持单继承

B:Java中可以多层(重)继承(继承体系)

(6)继承的注意事项：

A:子类不能继承父类的私有成员

B:子类不能继承父类的构造方法，但是可以通过super( )去访问

C:不要为了部分功能而去继承

(7)什么时候使用继承呢?

A:继承体现的是：is a的关系。

B:采用假设法

(8)Java继承中的成员关系

A:成员变量

a:子类的成员变量名称和父类中的成员变量名称不一样，这个太简单

b:子类的成员变量名称和父类中的成员变量名称一样

子类的方法访问变量的查找顺序：

在子类方法的局部范围找，有就使用。

在子类的成员范围找，有就使用。

在父类的成员范围找，有就使用。

B:构造方法

a:子类的构造方法默认会去访问父类的无参构造方法

是为了子类访问父类数据的初始化

b:父类中如果没有无参构造方法

子类构造方法通过super去明确调用父类带参构造

子类通过this调用本身的其他构造，但是一定会有一个去访问了父类的构造

C:成员方法

a:子类的成员方法和父类中的成员方法名称不一样，这个太简单

b:子类的成员方法和父类中的成员方法名称一样

通过子类对象访问一个方法的查找顺序：

在子类中找，有就使用

在父类中找，有就使用

**final关键字**

(1)特点：

A:它修饰的类， 不能被继承。

B:它修饰的方法，不能被重写。

C:它修饰的变量，是一个常量。

(2)面试相关：

A:局部变量

a:基本类型 值不能发生改变

b:引用类型 地址值不能发生改变，但是对象的内容是可以改变的

B:初始化时机

a:只能初始化一次。

b:常见的给值

定义的时候。(推荐)

构造方法中。

**多态**

(1)同一个对象在不同时刻体现出来的不同状态。

(2)多态的前提：

A:有继承或者实现关系。

B:有方法重写。

C:有父类或者父接口引用指向子类对象。

多态的分类：

a:具体类多态

class Fu {}

class Zi extends Fu {}

Fu f = new Zi();

b:抽象类多态

abstract class Fu {}

class Zi extends Fu {}

Fu f = new Zi();

c:接口多态

interface Fu {}

class Zi implements Fu {}

Fu f = new Zi();

(3)多态中的成员访问特点

A:成员变量

编译看左边，运行看左边

B:构造方法

子类的构造都会默认访问父类构造

因为成员方法有重写。

C:成员方法

编译看左边，运行看右边

D:静态方法

编译看左边，运行看左边

(4)多态的好处：

A:提高代码的维护性(继承体现)

B:提高代码的扩展性(多态体现)

(5)多态的弊端：

父不能使用子的特有功能。

(6)多态中的转型

A:向上转型 从子到父 Fu f = new Zi();

B:向下转型 从父到子 Zi z = (Zi) f ;

**抽象类**

(1)把多个共性的东西提取到一个类中，这是继承的做法。

但是在定义这些共性的方法的时候，就不能给出具体的方法体。

而一个没有具体的方法体的方法是抽象的方法。

在一个类中如果有抽象方法，该类必须定义为抽象类。

(2)抽象类的特点

A:抽象类和抽象方法必须用关键字abstract修饰

B:抽象类中不一定有抽象方法,但是有抽象方法的类一定是抽象类

C:抽象类不能实例化

D:抽象类的子类

a:是一个抽象类。

b:是一个具体类。这个类必须重写抽象类中的所有抽象方法。

(3)抽象类的成员特点：

A:成员变量：有变量，有常量

B:构造方法：有构造方法

C:成员方法：有抽象，有非抽象

(5)抽象类的几个小问题

A:抽象类有构造方法，不能实例化，用于子类访问父类数据的初始化

B:一个类如果没有抽象方法,却定义为了抽象类，为了不让创建对象

C:abstract不能和哪些关键字共存

a:final 冲突

b:private 冲突

c:static 无意义

**接口**

(1)为了给类添加额外的功能，java提供了接口表示。

(2)接口的特点：

A:接口用关键字interface修饰

interface 接口名 { }

B:类实现接口用implements修饰

class 类名 implements 接口名 { }

C:接口不能实例化

D:接口的实现类

a:是一个抽象类。

b:是一个具体类，这个类必须重写接口中的所有抽象方法。

(3)接口的成员特点：

A:成员变量：只能是常量

默认修饰符：public static final

B:构造方法：没有构造方法

C:成员方法：只能是抽象的

默认修饰符：public abstract

(4)类与类,类与接口,接口与接口

A:类与类： 继承关系，只能单继承，可以多层继承

B

C:接口与接口：继承关系，可以单继承，也可以多继承

(5)抽象类和接口的区别(自己补齐)?

A:成员区别

抽象类：有变量，有常量

接口：只能是常量

B:关系区别:

类与类： 继承，单继承

类与接口： 实现，可单/多实现

接口与接口： 继承，可单/多继承

C:设计理念不同

抽象类中定义的是共性功能。

接口中定义的是扩展功能。

**形式参数和返回值的问题**

(1)形式参数：

类名：需要该类的对象

抽象类名：需要该类的子类对象

接口名：需要该接口的实现类对象

(2)返回值类型：

类名：返回的是该类的对象

抽象类名：返回的是该类的子类对象

接口名：返回的是该接口的实现类的对象

(3)链式编程

对象.方法1( ).方法2( ).......方法n( );

这种用法：其实在方法1()调用完毕后，应该一个对象；

方法2()调用完毕后，应该返回一个对象。

方法n()调用完毕后，可能是对象，也可以不是对象。

**包**

(1)其实就是文件夹

(2)作用：

A:区分同名的类

B:对类进行分类管理

a:按照功能分

b:按照模块分

(3)包的定义

package 包名;

多级包用用 . 分开。

(4)注意事项：

A:package语句必须在文件中的第一条有效语句

B:在一个java文件中，只能有一个package

C:如果没有package，默认就是无包名

(5)带包的编译和运行

A:手动式：手动创建文件夹

B:自动式 ：例： javac -d . HelloWorld.java

**导包**

(1)多次使用一个带包的类，Java就提供了一个关键字import。

(2)格式：

import 包名...类名;

另一种： import 包名...\*;(不建议)

(3) package, import, class的先后顺序

package > import > class

**权限修饰符**

(1)权限修饰符

本类 同一个包下 不同包下的子类 不同包下的无关类

private Y

默认 Y Y

protected Y Y Y

public Y Y Y Y

(2) 这四种权限修饰符在任意时刻只能出现一种。

**常见的修饰符**

(1)分类：

权限修饰符：private, 默认, protected, public

状态修饰符：static, final

抽象修饰符：abstract

(2)常见的类及其组成的修饰

类： 默认, public, final, abstract

成员变量： private, 默认, protected, public, static, final

构造方法： private, 默认, protected, public

成员方法： private, 默认, protected, public, static, final, abstract

**内部类**

(1)把类定义在另一个类的内部，该类就被称为内部类。

(2)内部类的访问规则

A:可以直接访问外部类的成员，包括私有

B:外部类要想访问内部类成员，必须创建对象

(3)内部类的分类

A:成员内部类

B:局部内部类（在外部类的方法中定义的类）

(4)成员内部类

A:private 为了数据的安全性

B:static 为了访问的方便性

成员内部类不是静态的：

外部类名.内部类名 对象名 = new 外部类名.new 内部类名();

成员内部类是静态的：

外部类名.内部类名 对象名 = new 外部类名.内部类名();

(5)成员内部类的面试题

外部类成员变量 num

内部类成员变量 this.num

内部类局部变量 Outer.this.num

(6)局部内部类

A:局部内部类访问的局部变量必须加final修饰。

因为堆内存还是用该变量，而改变量已经没有了。

为了让该值还存在，就加final修饰。

(7)匿名内部类

A:是局部内部类的简化形式

B:前提

存在一个可以被继承的类或者接口

C:格式:

new 类名或者接口名() {

重写方法;

}

D:本质： 其实是继承该类或者实现接口的子类匿名对象

(8)匿名内部类在开发中的使用

我们在开发的时候，会看到抽象类，或者接口作为参数。

而这个时候，我们知道实际需要的是一个子类对象。

如果该方法仅仅调用一次，我们就可以使用匿名内部类的格式简化。

interface Person { public abstract void study(); }

class PersonDemo {

public void method(Person p) {

p.study();

}

}

class PersonTest {

public static void main(String[] args) {

PersonDemo pd = new PersonDemo();

pd.method(new Person() {

public void study() {

System.out.println("好好学习，天天向上");

}

});

}

}

**Eclipse**

Eclipse是一种可扩展的开放源代码的IDE(Integrated Development Environment)。

MyEclipse ：在Eclipse基础上追加的功能性插件，对插件收费

在WEB开发中提供强大的系统架构平台

常用快捷键：格式化 ctrl+shift+f

注释 单行 ctrl+/ 双行ctrl+shift+/, ctrl+shift+\

代码上下移动 选中代码alt+上/下箭头

查看源码 选中类名(F3或者Ctrl+鼠标点击)

提高开发效率：右键自动生成构造方法，get/set方法。

右键自动生成继承抽象类，或者实现接口的方法

制作帮助文档（API）：针对源程序添加文档注释：如@author PJYAN

项目--右键--Export--Java--Javadoc—Finish

打jar包并使用jar包： 可以让别的包是用这个包的类和接口。

打jar包“选中项目--右键--Export--Java--Jar--自己指定路径和名称--Finish

使用jar包：复制到项目路径下并添加至构建路径。

**API**(Application Programming Interface) ：程序编程接口。

就是JDK提供给我们的一些提高编程效率的java类。

**Object类**

(1)Object是类层次结构的根类，所有的类都直接或者间接的继承自Object类。

(2)Object类的构造方法有一个，并且是无参构造

(3)要掌握的方法：

toString()：返回对象的字符串表示

默认是由类的全路径+'@'+哈希值的十六进制表示。

这个表示其实是没有意义的，一般子类都会重写该方法。

但是最终还是右键自动生成。

equals()：比较两个对象是否相同。默认情况下，比较的是地址值是否相同。

而比较地址值是没有意义的，所以，一般子类也会重写该方法。

但是最终还是有右键自动生成。

String类重写equals( )方法，比较内容是否相同

(4)要了解的方法：

hashCode() 返回对象的哈希值。不是实际地址值，可以理解为地址值。

getClass() 返回对象的字节码文件对象，

finalize() 用于垃圾回收，在不确定的时间

clone() 实现对象的克隆，包括成员变量的数据复制，但是它和两个引用指 向同一个对象是有区别的。

(5)两个注意问题；

A:直接输出一个对象名称，其实默认调用了该对象的toString()方法。

B: ==和equals()的区别?

== 基本类型：比较的是值是否相同

引用类型：比较的是地址值是否相同

equals()：只能比较引用类型。默认情况，比较的是地址值是否相同。

但是，我们可以根据自己的需要重写该方法。

**Scanner类**

(1)构造方法： Scanner sc = new Scanner(System.in);

(2)基本方法格式：

A:hasNextXxx() 判断是否是某种类型的

B:nextXxx() 返回某种类型的元素

(3)要掌握的两个方法

A:public int nextInt()

B:public String nextLine()

(4)需要注意的小问题

A:同一个Scanner对象，先获取数值，再获取字符串会出现一个小问题。

B:解决方案：

a:重新定义一个Scanner对象

b:把所有的数据都用字符串获取，然后再进行相应的转换

**String类**

(1)构造方法：

A:public String()

B:public String(byte[] bytes)

C:public String(byte[] bytes,int offset,int length)

D:public String(char[] value)

E:public String(char[] value,int offset,int count)

F:public String(String original)

下面的这一个虽然不是构造方法，但是结果也是一个字符串对象

G:String s = "hello";

(2)字符串的特点

A:字符串一旦被赋值，就不能改变。

注意：这里指的是字符串的内容不能改变，而不是引用不能改变。

B:字面值作为字符串对象和通过构造方法创建对象的不同

String s = new String("hello"); 和String s = "hello"的区别?

创建新对象和常量||只创建常量，如果常亮存在则不再创建

(3)字符串的面试题

A:==和equals()

String s1 = new String("hello");

String s2 = new String("hello");

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

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

String s3 = new String("hello");

String s4 = "hello";

System.out.println(s3 == s4);// false

System.out.println(s3.equals(s4));// true

String s5 = "hello";

String s6 = "hello";

System.out.println(s5 == s6);// true

System.out.println(s5.equals(s6));// true

B:字符串的拼接

String s1 = "hello";

String s2 = "world";

String s3 = "helloworld";

System.out.println(s3 == s1 + s2);// false

System.out.println(s3.equals((s1 + s2)));// true

System.out.println(s3 == "hello" + "world");// true，因为是常量相加

System.out.println(s3.equals("hello" + "world"));// true

(4)字符串的功能

A:判断功能

boolean equals(Object obj)

boolean equalsIgnoreCase(String str)

boolean contains(String str)

boolean startsWith(String str)

boolean endsWith(String str)

boolean isEmpty()

B:获取功能

int length()

char charAt(int index)

int indexOf(int ch)

int indexOf(String str)

int indexOf(int ch,int fromIndex)

int indexOf(String str,int fromIndex)

String substring(int start)

String substring(int start,int end)

C:转换功能

byte[] getBytes()

char[] toCharArray()

static String valueOf(char[] chs)

static String valueOf(int i)

String toLowerCase()

String toUpperCase()

String concat(String str) //字符串拼接

D:其他功能

a:替换功能

String replace(char old,char new) //所有都替换

String replace(String old,String new) //所有都替换

b:去空格功能

String trim()

c:按字典比较功能

int compareTo(String str)

int compareToIgnoreCase(String str)

**StringBuffer**

(1)字符串做拼接，比较耗时并且也耗内存， Java就提供了 一个字符串缓冲区类。

(2)StringBuffer的构造方法

A:StringBuffer()

B:StringBuffer(int size) //初始容量

C:StringBuffer(String str)

(3)StringBuffer的常见功能(自己补齐方法的声明和方法的解释)

A:添加功能 append() insert()

B:删除功能 delete()

C:替换功能 replace()

D:反转功能 reverse()

E:截取功能(注意这个返回值)

(4)String和StringBuffer相互转换

String – StringBuffer sb的构造方法

StringBuffer – String sb.toString()方法

(5)小细节：

StringBuffer：同步的，数据安全，效率低。

StringBuilder：不同步的，数据不安全，效率高。

(6)注意的问题：

String作为形式参数跟普通数据类型一样。形参改变不影响实参。

**数组高级以及Arrays**

(1)排序

A:冒泡排序

相邻元素两两比较，大的往后放，第一次完毕，最大值出现在了最大索引处。同理，其他的元素就可以排好。

for(int x=0; x<arr.length-1; x++) {

for(int y=0; y<arr.length-1-x; y++) {

if(arr[y] > arr[y+1]) {

int temp = arr[y];

arr[y] = arr[y+1];

arr[y+1] = temp;

}

}

}

B:选择排序

把0索引的元素，和索引1以后的元素都进行比较，第一次完毕，最小值出现在了0索引。同理，其他的元素就可以排好。

for(int x=0; x<arr.length-1; x++) {

for(int y=x+1; y<arr.length; y++) {

if(arr[y] < arr[x]) {

int temp = arr[x];

arr[x] = arr[y];

arr[y] = temp;

}

}

}

(2)查找

A:基本查找

针对数组无序的情况，按顺序查找。

B:二分查找(折半查找)

针对数组有序的情况(千万不要先排序，在查找)

int min = 0;

int max = arr.length-1;

int mid = (min+max)/2;

while(arr[mid] != value) {

if(arr[mid] > value) {

max = mid - 1;

}else if(arr[mid] < value) {

min = mid + 1;

}

if(min > max) { return -1; }

mid = (min+max)/2;

}

(3)Arrays工具类

A:是针对数组进行操作的工具类。包括排序和查找等功能。

B:要掌握的方法(自己补齐方法)

把数组转成字符串： toString()

排序： sort()

二分查找： binarySearch() 如果不存在返回(-(insertion point) - 1)

**Integer类**

(1)为了让基本类型的数据进行更多的操作，Java就为每种基本类型提供了对应的包装类类型

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

(2)Integer的构造方法

A:Integer i = new Integer(100);

B:Integer i = new Integer("100");

注意：这里的字符串必须是由数字字符组成

(3)String和int的相互转换

A:String -- int

Integer.parseInt("100");

B:int -- String

String.valueOf(100);

(4) JDK5的新特性

自动装箱 基本类型--引用类型

自动拆箱 引用类型--基本类型

把下面的这个代码理解即可：

Integer i = 100; Integer.valueOf(100)自动装箱

i += 200; Integer.valueOf(i.intValue() + 200) 自动拆箱后自动装箱

**Character类**

(1)Character构造方法

Character ch = new Character('a');

(2)要掌握的方法：

A:判断给定的字符是否是大写 isUpperCase()

B:判断给定的字符是否是小写 isLowerCase()

C:判断给定的字符是否是数字字符 isDigit()

D:把给定的字符转成大写 toUpperCase()

E:把给定的字符转成小写 toLowerCase()

**正则表达式**

(1)就是符合一定规则的字符串

(2)常见规则

A:字符

x 字符 x。举例：'a'表示字符a

\\ 反斜线字符。

\n 新行（换行）符 ('\u000A')

\r 回车符 ('\u000D')

B:字符类

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a到 z 或 A到 Z，两头的字母包括在内（范围）

[0-9] 0到9的字符都包括

C:预定义字符类

. 任何字符。 使用 \. 表示 . 本身

\d 数字：[0-9]

\w 单词字符：[a-zA-Z\_0-9]

在正则表达式里面组成单词的东西必须有这些东西组成

D:边界匹配器

^ 行的开头

$ 行的结尾

\b 单词边界

就是不是单词字符的地方。

举例：hello world?haha;xixi

E:Greedy 数量词

X? X，一次或一次也没有

X\* X，零次或多次

X+ X，一次或多次

X{n} X，恰好 n 次

X{n,} X，至少 n 次

X{n,m} X，至少 n 次，但是不超过 m 次

(3)常见功能：(分别用的是谁呢?)

A:判断功能

String类的public boolean matches(String regex)

B:分割功能

String类的public String[] split(String regex)

C:替换功能

String类的public String replaceAll(String regex,String replacement)

D:获取功能

Pattern和Matcher

Pattern p = Pattern.compile("a\*b"); 参数：正则表达式

Matcher m = p.matcher("aaaaab"); 参数：被查找字符串

先m.find():查找存不存在

再m.group():获取刚才查找过的数据

**Math类**

(1)针对数学运算进行操作的类

(2)常见方法(自己补齐)

A:绝对值 abs()

B:向上取整 ceil()

C:向下取整 floor()

D:两个数据中的大值 max()

E:a的b次幂 pow(a,b)

F:随机数 random()

G:四舍五入 round()

H:正平方根 sqrt()

**Random类**

(1)用于产生随机数的类

(2)构造方法:

A:Random() 默认种子，每次产生的随机数不同

B:Random(long seed) 指定种子，每次种子相同，随机数就相同

(3)成员方法:

A:int nextInt() 返回int类型范围内的随机数

B:int nextInt(int n) 返回[0,n)范围内的随机数

**System类**

(1)系统类,提供了一些有用的字段和方法

(2)成员方法

A:运行垃圾回收器 gc()

B:退出jvm exit(int a) 0为正常退出 其他数值为异常退出

C:获取当前时间的毫秒值 [currentTimeMillis](http://www.runoob.com/manual/jdk1.6/java/lang/System.html#currentTimeMillis())()

D:数组复制 （src的部分内容复制到dest中）

[arraycopy](http://www.runoob.com/manual/jdk1.6/java/lang/System.html#arraycopy(java.lang.Object, int, java.lang.Object, int, int))([Object](http://www.runoob.com/manual/jdk1.6/java/lang/Object.html) src, int srcPos, [Object](http://www.runoob.com/manual/jdk1.6/java/lang/Object.html) dest, int destPos, int length)

**BigInteger类**

(1)针对大整数的运算

(2)构造方法

A:BigInteger(String s)

(3)成员方法

A:加 add(BigInteger val)

B:减 subtract(BigInteger val)

C:乘 multiply(BigInteger val)

D:除 divide(BigInteger val)

E:商和余数 public BigInteger[] divideAndRemainder(BigInteger val)

6:BigDecimal(理解)

1. 浮点数据做运算，会丢失精度。

所以，针对浮点数据的操作建议采用BigDecimal。

(2)构造方法

A:BigDecimal(String s)

(3)成员方法：

A:加 add(BigDecimal augend)

B:减 subtract(BigDecimal subtrahend)

C:乘 multiply(BigDecimal multiplicand)

D:除 divide(BigDecimal divisor)

E:自己保留小数几位 BigDecimal divide(BigDecimal divisor,int sc

**Date/DateFormat类**

(1)Date是日期类，可以精确到毫秒。

A:构造方法

Date() 当前时间

Date(long time) 1970年后的毫秒值

B:成员方法

getTime() 返回毫秒值

setTime(long time)

(2)DateFormat针对日期进行格式化和针对字符串进行解析的类，但是是抽象类，所以使用其子类SimpleDateFormat

A:创建对象并给定格式

例：yyyy-MM-dd HH:mm:ss

B:日期和字符串的转换

a:Date -- String

SimpleDateFormat(String pattern) 传入String所需格式

format(Data)返回String

b:String – Date

SimpleDateFormat(String pattern) 传入String当前格式

parse(String)返回Date

**Calendar类**

(1)日历类，封装了所有的日历字段值，通过统一的方法根据传入不同的日历字段可以获取值。

(2)创建方法

Calendar rightNow = Calendar.getInstance();

本质返回Calendar的子类对象

(3)成员方法

A:根据日历字段得到对应的值 get(int field)

B:根据日历字段和一个正负数确定是添加还是减去对应日历字段的值

add(int field,int amount)

C:设置日历对象的年月日 set(int year,int month,int date)

**Collection接口**

(1)集合的由来?

Java -- 面向对象 -- 操作很多对象 -- 存储 -- 容器(数组和StringBuffer) -- 数组

而数组的长度固定，不适合做变化的需求，Java提供了集合供我们使用。

(2)集合和数组的区别

A:长度区别

数组固定

集合可变

B:内容区别

数组可以是基本类型，也可以是引用类型

集合只能是引用类型

C:元素内容

数组只能存储同一种类型

集合可以存储不同类型(其实集合一般存储的也是同一种类型)

(3)集合的继承体系结构

它们都是要提供存储和遍历功能的，

Collection

|--List

|--ArrayList

|--Vector

|--LinkedList

|--Set

|--HashSet

|--TreeSet

(4)Collection集合的遍历

A:把集合转数组 toArray()方法返回数组

B:迭代器 iterator

(5)迭代器iterator

A:是集合的获取元素的方式。

B:是依赖于集合而存在的。

C:迭代方法

Iterator it = c.iterator();

while(it.hasNext()) {

String s =(String) it.next();

}

**List类**

(1)List是Collection的子接口

特点：有序(存储顺序和取出顺序一致)，可重复。

(2)List集合的特有遍历功能

由size()和get()结合。 size定范围，get提取

(3)列表迭代器的特有功能（ListInterator）

可以逆向遍历，但是要先正向遍历，所以无意义，基本不使用。

(5)并发修改异常

A:出现的现象

迭代器遍历集合期间，集合元素被修改

B:原因

迭代器是依赖于集合的，而集合的改变迭代器并不知道。

C:解决方案

a:迭代器遍历，迭代器修改(ListIterator)

ListIterator的add（）方法将元素添加在刚才迭代的位置

b:集合遍历，集合修改(size()和get())

元素添加在集合的末尾

**常见数据结构**

A:栈 先进后出

B:队列 先进先出

C:数组 查询快，增删慢

D:链表 查询慢，增删快

**List的子类**

(1)List的子类特点

ArrayList:

底层数据结构是数组，查询快，增删慢

线程不安全，效率高

Vector:

底层数据结构是数组，查询快，增删慢

线程安全，效率低

LinkedList:

底层数据结构是链表，查询慢，增删快

线程不安全，效率高

(2)ArrayList

A:没有特有功能需要学习

(3)Vector

A:有特有功能（都可被后边方法替代）

a:添加

public void addElement(E obj) -- add()

b:获取

public E elementAt(int index) -- get()

public Enumeration<E> elements() -- iterator()

(4)LinkedList

A:有特有功能

a:添加

addFirst()

addLast()

b:删除

removeFirst()

removeLast()

c:获取

getFirst()

getLast()

**泛型**

(1)泛型概述

是一种把明确类型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的类型。

(2)格式：

<数据类型>

注意：该数据类型只能是引用类型。

(3)好处：

A:把运行时期的问题提前到了编译期间

B:避免了强制类型转换

C:优化了程序设计，解决了黄色警告线问题，让程序更安全

转型问题

B:泛型类 类名<T t>{ … }

C:泛型方法 public <T t > void 方法名(T t){ }

D:泛型接口 接口名<T t>

E:泛型高级通配符

? 对象可以是任何引用对象

? extends E 对象只可以是E或E的子类对象

? super E 对象只能是E或E的父类对象

**增强for循环**

(1)是for循环的一种

(2)格式：

for(元素的数据类型 变量名 : 数组或者Collection集合的对象) {

使用该变量即可，该变量其实就是数组或者集合中的元素。

}

(3)好处：

简化了数组和集合的遍历

(4)弊端

增强for循环的目标不能为null。建议在使用前，先判断是否为null。

**静态导入**

(1)可以导入到方法级别的导入

(2)格式：import static 包名....类名.方法名;

(3)注意事项：

A:方法必须是静态的

B:如果多个类下有同名的方法，就不好区分了，还得加上前缀。

所以一般我们并不使用静态导入，但是一定要能够看懂。

**可变参数**

(1)如果我们在写方法的时候，参数个数不明确，就应该定义可变参数。

(2)格式：

修饰符 返回值类型 方法名(数据类型... 变量) { }

注意：

A:该变量其实是一个数组名(可用for循环遍历)

B:如果一个方法有多个参数，并且有可变参数，可变参数必须在最后

(3)Arrays工具类的一个有可变参数的方法方法

static <T>  [List](http://docs.oracle.com/javase/8/docs/api/java/util/List.html)<T> asList(T…a)把数组转成集合。

注意：返回的集合的长度不能改变。

**Set集合**

(1)Set集合的特点

无序,唯一

**HashSet类**

A:底层数据结构是哈希表(是一个元素为链表的数组)

B:哈希表底层依赖两个方法：hashCode()和equals()

执行顺序：

首先比较哈希值是否相同

相同：继续执行equals()方法

返回true：元素重复了，不添加

返回false：直接把元素添加到集合

不同：就直接把元素添加到集合

C:如何保证元素唯一性的呢?

由元素的hashCode()和equals()方法保证的（String等已经重写）

D:开发的时候，为了重写hashCode()和equals()，右键自动生成即可。

**TreeSet类**

A:底层数据结构是红黑树(是一个自平衡的二叉树)

B:保证元素的排序方式

a:自然排序(元素自身具备比较性)

让元素所属的类实现Comparable接口

b:比较器排序(集合具备比较性)

让集合构造方法接收Comparator的实现类对象

一般是传入一个Comparator的匿名内部类

**Collection集合总结**

Collection

|--List 有序,可重复

|--ArrayList

底层数据结构是数组，查询快，增删慢。

线程不安全，效率高

|--Vector

底层数据结构是数组，查询快，增删慢。

线程安全，效率低

|--LinkedList

底层数据结构是链表，查询慢，增删快。

线程不安全，效率高

|--Set 无序,唯一

|--HashSet

底层数据结构是哈希表。

如何保证元素唯一性的呢?

依赖两个方法：hashCode()和equals()

开发中自动生成这两个方法即可

|--LinkedHashSet

底层数据结构是链表和哈希表

由链表保证元素有序（符合输入顺序）

由哈希表保证元素唯一

|--TreeSet

底层数据结构是红黑树。

如何保证元素排序的呢?

自然排序

比较器排序

如何保证元素唯一性的呢?

根据比较的返回值是否是0来决定

**针对Collection集合我们到底使用谁呢**

唯一吗?

是：Set

排序吗?

是：TreeSet

否：HashSet

如果你知道是Set，但是不知道是哪个Set，就用HashSet。

否：List

要安全吗?

是：Vector

否：ArrayList或者LinkedList

查询多：ArrayList

增删多：LinkedList

如果你知道是List，但是不知道是哪个List，就用ArrayList。

如果你知道是Collection集合，但是不知道使用谁，就用ArrayList。

**在集合中常见的数据结构**

ArrayXxx:底层数据结构是数组，查询快，增删慢

LinkedXxx:底层数据结构是链表，查询慢，增删快

HashXxx:底层数据结构是哈希表。依赖两个方法：hashCode()和equals()

TreeXxx:底层数据结构是二叉树。两种方式排序：自然排序和比较器排序

**Map类**

(1)将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。

(2)Map和Collection的区别

A:Map 存储的是键值对形式的元素，键唯一，值可以重复。夫妻对

B:Collection 存储的是单独出现的元素，子接口Set元素唯一，子接口List元素可重复。光棍

(4)Map集合的遍历

A:键找值

a:获取所有键的集合

b:遍历键的集合,得到每一个键

c:根据键到集合中去找值

Set<String> set = hm.keySet(); //hm是个map对象

for(String key : set) {

String value = hm.get(key);

System.out.println(key+"---"+value);

}

B:键值对对象找键和值

a:获取所有的键值对对象的集合

b:遍历键值对对象的集合，获取每一个键值对对象

c:根据键值对对象去获取键和值

Set<Map.Entry<String,String>> set2 = hm.entrySet();

for(Map.Entry<String,String> me : set2) {

String key = me.getKey();

String value = me.getValue();

System.out.println(key+"---"+value);

}

**Collections类**

(1)是针对集合进行操作的工具类

(2)Collection和Collections的区别

A:Collection 是单列集合的顶层接口，有两个子接口List和Set

B:Collections 是针对集合进行操作的工具类，可以对集合进行排序和查找等

(3)常见的几个小方法：

A:public static <T> void sort(List<T> list) 排序

B:public static <T> int binarySearch(List<?> list,T key) 二分查找

C:public static <T> T max(Collection<?> coll) 找最大值

D:public static void reverse(List<?> list) 反转集合

E:public static void shuffle(List<?> list) 随机排序集合

**异常**

(1)程序出现的不正常的情况。

(2)异常的体系

Throwable

|--Error 严重问题，我们不处理。

|--Exception

|--RuntimeException 运行期异常，我们需要修正程序代码

|--非RuntimeException 编译期异常，必须处理的，否则程序编译不通过

(3)异常的处理：

A:JVM的默认处理

把异常的名称,原因,位置等信息输出在控制台，但是程序不能继续执行了。

B:自己处理

a:try...catch...finally

自己编写处理代码,后面的程序可以继续执行

b:throws

把自己处理不了，在方法上声明，告诉调用者，这里有问题

(4)面试题

A:编译期异常和运行期异常的区别?

编译期异常 必须要处理的，否则编译不通过

运行期异常 可以不处理，也可以处理

B:throw和throws是的区别

throw:

在方法体中,后面跟的是异常对象名,并且只能是一个

throw抛出的是一个异常对象，说明这里肯定有一个异常产生了

throws:

在方法声明上,后面跟的是异常的类名,可以是多个

throws是声明方法有异常，是一种可能性，这个异常并不一定会产生

(5)finally关键字

A:finally用于释放资源，它的代码永远会执行。

特殊情况：在执行到finally之前jvm退出了（ System.exit(0) ）

B:面试题

a:final,finally,finalize的区别? 标识符，异常处理，垃圾回收

b:如果在catch里面有return,请问finally还执行吗?

如果执行,在return前还是后?

会，前。实际上在中间。

(6)自定义异常

继承自Exception或者RuntimeException,只需要提供无参构造和一个带参构造即可。带参构造用于抛出提示。

(7)异常的注意实现

A:父的方法有异常抛出,子的重写方法在抛出异常的时候必须要小于等于父的异常

B:父的方法没有异常抛出,子的重写方法不能有异常抛出

C:父的方法抛出多个异常,子的重写方法必须比父少或者小

**File类**

（1）IO流操作中大部分都是对文件的操作，Java就提供了File类供我们来操作文件

File类是 文件和目录路径名的抽象表示形式。

(2)构造方法

A:File file = new File("e:\\demo\\a.txt");

B:File file = new File("e:\\demo","a.txt");

C:File file = new File("e:\\demo");

File file2 = new File(file,"a.txt");

(3)File类的功能

A:创建功能

B:删除功能

C:重命名功能

D:判断功能

E:获取功能

F:高级获取功能

G:过滤器功能 FileNameFilter

**递归**

(1)方法定义中调用方法本身的现象

(2)递归的注意事项；

A:要有出口，否则就是死递归

B:次数不能过多，否则内存溢出

C:构造方法不能递归使用

**IO流**

(1)IO用于在设备间进行数据传输的操作

(2)分类：

字节流

字节输入流 读取数据（从JAVA程序的角度）

字节输出流 写出数据

字符流

字符输入流 读取数据

字符输出流 写出数据

注意：

a:如果我们没有明确说明按照什么分，默认按照数据类型分。

b:除非文件用windows自带的记事本打开我们能够读懂，才采用字符流，否则建议使用字节流。

**FileOutputStream**写出数据

A:操作步骤

a:创建字节输出流对象 FileOutputStream fos = new FileOutputStream("fos.txt");

b:调用write()方法 fos.write("hello".getBytes());

c:释放资源 fos.close();

**FileInputStream**读取数据

A:操作步骤

a:创建字节输入流对象

b:调用read()方法

c:释放资源

B:代码体现：

FileInputStream fis = new FileInputStream("fos.txt");

//方式1

int by = 0;

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

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

}

//方式2

byte[] bys = new byte[1024];

int len = 0;

while((len=fis.read(bys))!=-1) {

System.out.print(new String(bys,0,len));

}

fis.close();

C.字节缓冲区流

A:BufferedOutputStream 构造方法BufferedOutputStream(OuputStream)

B:BufferedInputStream 构造方法BufferedInputStream(InputStream)

**字符流**

(1)字节流操作中文数据不是特别的方便，所以就出现了转换流。

转换流的作用就是把字节流转换字符流来使用。

(2)转换流其实是一个字符流

字符流 = 字节流 + 编码表

(3)编码表

A:就是由字符和对应的数值组成的一张表

B:常见的编码表

ASCII

ISO-8859-1

GB2312

GBK

GB18030

UTF-8

C:字符串中的编码问题

编码

String 🡪 byte[]

解码

byte[] 🡪 String

(4)IO流中的编码问题

A:OutputStreamWriter

OutputStreamWriter(OutputStream os):默认编码，GBK

OutputStreamWriter(OutputStream os,String charsetName):指定编码。

B:InputStreamReader

InputStreamReader(InputStream is):默认编码，GBK

InputStreamReader(InputStream is,String charsetName):指定编码

(5)字符流

Reader

|--InputStreamReader

|--FileReader

|--BufferedReader

Writer

|--OutputStreamWriter

|--FileWriter

|--BufferedWriter

**IO流小结**

IO流

|--字节流

|--字节输入流

InputStream

int read():一次读取一个字节

int read(byte[] bys):一次读取一个字节数组

|--FileInputStream

|--BufferedInputStream

|--字节输出流

OutputStream

void write(int by):一次写一个字节

void write(byte[] bys,int index,int len):一次写一个字节数组的一部分

|--FileOutputStream

|--BufferedOutputStream

|--字符流

|--字符输入流

Reader

int read():一次读取一个字符

int read(char[] chs):一次读取一个字符数组

|--InputStreamReader

|--FileReader

|--BufferedReader

String readLine():一次读取一个字符串

|--字符输出流

Writer

void write(int ch):一次写一个字符

void write(char[] chs,int index,int len):一次写一个字符数组的一部分

|--OutputStreamWriter

|--FileWriter

|--BufferedWriter

void newLine():写一个换行符

void write(String line):一次写一个字符串