java代码编译过程

java代码 源码（.java结束的文件，可以看）---（javac编译器）--

字节码文件（.class结尾的文件（看不了，乱码））--jvm加载--机器码

.java🡪.class：正常流程

.class🡪.java：反编译器（部分）

jdk:开发工具

jre:开发环境

jvm:虚拟机（翻译官）

java跨平台 一次编译，处处运行（到处改错）

jre+java系统的类库--运行最小环境

jdk+编译:开发一个程序的最小单元（环境）

实施，开发，测试，运维

1.基本类型：八种

2.引用类型：除了八种基本类型之外的都叫做引用类型

byte 1字节 范围（-128~127）255

short 2字节 范围（-32768~32767）65535

int 4字节 范围（-2147483648~2147483647）

long 8字节 范围（-9223372036854775808~9223372036854775807）

float 4字节 范围（1.40239846E-45~3.4028235E38）

double 8字节 范围（4.9406564584124654E-324~1.7976931348623157E308）

boolean 1字节

char 2字节 范围（ ~？）

自动类型转换

整数类型转成浮点类型会自动加上精度

整数类型字面量默认的是int

浮点数类型字面量默认的是double

基本的强制类型转换

自动类型转换 --------小类型到大类型

强制类型转换（引用类型转换）

int in = (int) 12.5;--------溢出的风险

小类型和大类型之间进行运算，会自动转向大类型

取整/除法

如果两个数都是整数类型则为取整

如果任意一个数为浮点数，则会将其他数转化成浮点数进行运算

++i….使用之前加/减

i++….使用之后加/减

关系运算符的结果是布尔类型（true，false）

&/&&…两边的表达式或者变量同时为真，整个运算才为真，有一边为假，才为假，整个逻辑运算表达式都为假

|/||…两边的表达式或者变量，有一方为真，整个逻辑运算结果都为真，两边同时为假，整个运算才为假

!...对结果进行取反

&&/||…短路运算符，一假则假，一真则真

条件运算符（三目运算符）

Boolean 表达式？ 表达式1： 表达式2;

注意点：在基本类型中，如果表达式1和表达式2的结果类型尽量一致，如果不一致，能自动转换的变量定义为大类型

如果是引用类型，且这两个类型没有关系，则会报错。要求两个表达式为统一类型

条件运算可以嵌套使用

分支结构：

总结：if语句：整个事件有多种情况，但是我们只选择需要的某种情况

If-else语句：将整个事件划分成两种对立的情况，两则必然会执行其中之一，然后结束整个语句

If-else if语句：将整个事件的多种情况都罗列出来，根据条件执行不同的语句，每种情况处于互斥的状态，满足其中之一。

Switch-case: 特殊的分支结构，根据switch中的值来与case句中的值，进行性比较，如果两个值相等就会执行该case语句中的逻辑代码一般是break 搭配使用，break的作用是跳出switch-case 执行该语句之后的有效代码

区别：都是分支结构，用于判断条件成立，执行某些逻辑代码

Switch-case 只能用于等值判断，不能判断范围

if语句都可以判断

While() or do {} while();

While直接进入判断，

Dowhile先执行循环体，再判断表达式的结果，若果表示为true，再执行循环体，然后再判断。

即while中的循环体可以一次都不执行，dowhile的循环体至少执行一次。

代码中的参数范围都是，左闭右开区间。

For语句：

执行流程，for(表达式1;表达式2; 表达式3;){}

表达式1：循环的初始条件

表达式2：boolean表达式，循环的判断条件（唯一）

表达式3：初始条件的变化量（自增或自减）

执行逻辑：

表达式1🡪表达式2（进行判断，true🡪循环体🡪表达式3 🡪表达式2（进行判断，如此重复），false🡪跳出循环）

数组：

数组的三种定义方式：Int[] arr1 = new int[10];

Int[] arr2 = {};

Int[] arr3 = new int[]{};

定义基本类型数组的要点：

确切的数据类型

整体的数组名字

不能缺少的”[]”

数组的复制

System.arraycopy()

Public static void arraycopy(Object src, int srcPoc, Object dest, int destPos, int length);

Src -源数组

srcPos -源数组中的起始位置

dest -目标数组

destPos -目标数组中的起始位置

length -要复制的数组元素的数量

Arrays.copyOf

使用java.util.Arrays.copyOf(Object arr, int newLength);

特点：生成的新数组是原始数组的副本

newLength小于源数组，则进行截取

newLength大于源数组，则用0或null填充

数组的扩容

数组的长度在创建后是不可改变的，所谓扩容是指创建一个更大的新数组并将原有数组的内容复制到新数组中。

Int[] arr = {10,20,30,40,50};

Arr = Arrays.copyOf(arr,arr.length+1);

属于在内存中开辟空间，而非覆盖。

数组：相同数据类型的元素组成的集合

除了八种基本类型外，其他都是引用类型

New 强制型操作，内存中开辟空间

数组定义的默认值：

整数：0，浮点 0.0, 字符：/u0000,布尔：false;

相同功能的方法，参数多的要比参数少的更加灵活

一个长度为7的整数数组，再进行冒泡排序的时候，最多进行多少次交换，最少进行多少次交换： 最多21次 （6+5+4+3+2+1） 最少：0次

方法：

方法的概述：

方法是指面向对象语言中执行某一功能的代码块，部分语言称之为函数或过程；

封装方法可以方便的了解该区域代码执行后的结果类型以及获取调试信息

方法可用于重复执行，便于程序维护

方法用于封装一个特定的功能；

方法的定义：

方法的封装尽量做到功能单一；

方法是封装特定功能的代码块

方法的封装有助于提高代码的复用率和维护性

定义方法的五个要素：修饰词、返回类型、方法名、参数列表、方法体

使用void的方法也可以使用return，但只表示让方法结束

用static修饰的方法，可以直接使用类名进行调用

用static修饰的方法属于类，不用static修饰的方法属于对象

属于对象的方法，需要创建对象，使用对象来调用

面向对象：

This：指代调用该类的变量

类是抽象类型：类🡪创建对象：对象（属于类一个具体存在的个体）

1. 现实生活中由很多的对象组成
2. 现实世界中先有对象，后有类（根据对象属性—抽出类）
3. 代码中先有类，后有对象
4. 一个类可以创建无限个对象
   1. 同一类的多个对象，数据结构相同，数据不同
5. 类是一种数据结构
   1. 包含：
      1. 对象所共有的特性
         1. 属性—成员变量
      2. 对象所共有的功能
         1. 行为—方法
      3. 方法用与描述对象的行为，封装操作对象的功能（方法是操作数据的）
6. 如何通过一个类来创建对象
   1. 创建对象—实例化
      1. 类名 引用变量名 = new 类型名();
   2. 赋值:引用变量.成员变量名 = 值;
   3. 取值:引用变量.成员变量
   4. 调用方法(非static)：引用变量.方法名();

注意：

在使用类创建对象时，用对象调用成员变量和方法时，会有一个隐式的关键字，this –指代当前对象

方法的重载：同一个类中，方法名相同，参数列表不同，称之为方法的重载，

编辑器在进行方法的调用时，根据参数的不同，调用不同的方法

Java中允许方法名相同，参数列表不同的写法

方法的返回类型与方法的重载无关

构造方法：（构造器） 🡪 支持方法的重载

构造方法必须和类名相同

构造方法没有返回值，但也不能写void

构造方法的作用，一般用于初始化成员变量 🡪 创建对象使用

在一个类中，若果没有写任何的构造方法，编辑器会默认的追加一个无参的构造方法

若果有其它的构造方法，则编辑器不再追加无参构造方法，

This：

构造器的调用：可以通过this关键字，调用当前类中已经存在的对应的构造方法

一般即使我们不使用无参的构造方法时，也要将无参构造方法写出，防止编译报错

内存管理：

1. 对象生存环境，和空间 编译好的java程序运行在jvm虚拟机中。
2. Jvm分为“堆”，“栈”，“方法区”三个区域。
3. 方法区分为静态区和非静态区
4. 基本数据类型存储在栈中。引用数据类型的地址存储在栈中，然后在堆中开辟一个空间留着存储数据，然后栈中存储的地址，指向堆中的对象
5. 成员变量属于对象。所以存储在堆内存中

生命周期：

1.

变量：

根据有效域：局部变量，成员变量。

1. 局部变量：
   1. 定义在方法区；
   2. 没有默认值，使用之前必须初始化；
   3. 方法调用时，存储在栈帧中，方法调用完，栈帧清除相应的数据。
2. 成员变量：
   1. 成员变量定义在类中，方法外；
   2. 有默认初始值，可以不用初始化，
   3. 实例化对象后，存储在堆中，对象使用完后，被gc（）（垃圾回收器—线程）回收，默认调用（写出来理论上会快一点回收）。
   4. 根据类型：基本类型，引用类型变量。

面向对象的三大特性：封装，继承，多态。

一个文件中只能有一个public修饰的class，且public修饰的class必须是和文件名相同

1. 封装：方法的封装，类的封装（组件）。
   1. JavaBean概念：
      1. JavaBean是一种Java语言写成的可重用组件，为了写成JavaBean，类必须是具体的和公开的。
      2. JavaBean其实就是一个按照特定规范定义的类（一种封装方式），里面需要有无参的构造方法，成员变量私有，方法公开，类具体，公开。
   2. 封装的好处：
      1. 对外部提供可供调用的稳定的功能
      2. 降低代码的出错率，提高代码的复用性和可维护性
      3. 当封装的方法内部出现细节改变时，只要保证对外的功能不变，其他模块不会因此受到影响
2. 继承：让类与类之间产生关系，子父类关系（强耦合） --- 继承者是子类，被继承是父类（父类大于子类）
   1. 关键字 ： extends --- 让类与类之间产生子父类的关系
   2. 将公共属性或者方法抽出来放到一起形成一个新的事物 --- 父类
   3. 注意：子类可以使用继承自父类的成员变量和方法 ---- 非私有
   4. 优点：提高代码的复用性，可维护性
      1. 让类与类之间产生关系 ---- 是多态的前提
3. 开发遵循原则：
   1. 高内聚，低耦合
      1. 高内聚：一个类自己完成某件事的能力
      2. 低耦合：类与类之间的关系
   2. 弊端：类与类之间产生强耦合，不利于开发中的灵活使用
4. 继承的特点：
   1. 只能继承父类中非私有的成员变量和方法
   2. Java中继承，只支持单继承，不支持多继承
   3. 支持多层继承（继承体系） --- 继承具有传递性
5. 如果想看整个继承体系中共有属性和方法，用最顶层的类创建对象
6. 如果想看整个继承体系中所有属性和方法（非私有），用最底层的类创建对象
7. 注意事项：子类只能继承父类中非私有的成员变量和方法
   1. 子类不能继承父类的构造方法，但是可以通过super关键字调用（访问）
   2. 在子类的构造方法中 默认调用父类中无参的构造方法，有且只能子类构造方法的第一句， 🡪 super(); 调用父类中无参构造方法
   3. 也可以调用对应参数的构造方法，初始化对应成员变量super(name , sex);
   4. 谨慎使用继承 – 不要因为部分功能而强制使用继承
8. This和super的区别
   1. This：指代当前对象的引用
   2. Super：指代父类的引用
      1. 区别：
         1. 调用成员变量：
            1. This.成员变量：调用本类中的成员变量，也可以调用父类中的成员变量（非私有）
            2. Super.成员变量：只能调用父类中成员变量（非私有）
         2. 调用构造方法:
            1. This(…):调用本类中构造方法
            2. Super(…):调用父类中的构造方法
            3. 注意：在同一个构造方法中，super和this不能同时调用—防止出现重复调用父类中构造方法
         3. 调用成员方法
            1. This.方法名():调用本类或者父类中的成员方法
            2. super.方法名():只能调用父类中的成员方法
   3. 子类中每个构造方法第一句都是默认调用父类中的无参构造方法 – super();
      1. 为什么？ 因为子类会继承父类中的数据，可能还会使用父类中数据，所以子类在初始化之前，必须先完成父类数据的初始化
      2. Object是所有父类的顶级父类

重写：**发生在两个类中，一般是子父类，子类中的方法名和参数列表相同，方法体不同，称之为方法的重写**。

1. 子类中出现和父类中一模一样的方法（和**返回值有关**）
2. 重写的应用：当子类需要父类的功能，而对于功能主体子类有自己特有的实现方式（方法体不同），可以重写父类中的方法
3. 如果子类中重写父类中的方法，在使用向上造型创建对象时，虽然是父类类型的引用，但是在用该引用变量调用的重写方法是子类中重写后的方法

注意事项：

1. 父类中的私有方法不能被重写（父类中私有方法不能被继承）
2. 子类重写父类中的方法时，访问权限要大于等于父类的权限修饰符（建议等于）
   1. Public > protected > default > private
3. 父类中的静态方法，子类也必须通过静态方法重写

重载和重写的区别：

1. overload：重载🡪一个类中，方法名相同，参数列表不同（类型，个数，位置），与返回值无关
2. override：重写🡪发生在两个类中（可以是子父类），方法名和参数列表都相同，与返回值有关

（父类默认大于子类）

泛化：由子类提取公共的属性和方法，形成一个新的类，这个过程称之为泛化

1. 向上造型 🡪 父类（类型）引用指向子类的对象
2. Animal animal = new Dog();
3. 注意：使用引用变量能点出什么，看类型（多态的体现）

特化：由一个公共的类型，衍生其诸多子类的过程，称之为特化

1. 向下造型 🡪 强制类型转换 instanceof(防止类型转换异常)
2. Java.lang.ClassCastException
3. Instanceof：用于判断某个引用变量是否属于某一类，返回值为boolean。
4. Dog dog = (Dog) animal;
5. Animal t = new Dog();
6. Animal c = new Cat();
7. If(c instanceof Dog){Dog dc = (Dog)c;}

Public : 都可以用（本类、同包的类、跨包子类、跨包类）

Protected ： 本类、同包的类、跨包子类（有限定，不能直接跨包的父类创建对象调用，只能使用跨包子类创建对象调用）🡪 通过子类类型对象访问  
默认（default）：本类，同包的类

Private： 本类

一般实际开发中，常用的为public和private

Static修饰词：

1. 随着类加载而加载，类的成员变量（类变量）
2. 优先于对象存在
3. 被类的所有对象共享（共享的用静态static，特性是非静态），无论创建多少个对象，存在于方法的只有一份只会加载一次

成员变量：

1. 实例变量：属于对象。
2. 静态变量（类变量）：属于类的，所有对象共享一份。

注意：

1. 静态方法没有This关键字
   1. This随着对象创建而存在，静态比对象优先存在
2. 静态方法只能访问静态变量和静态方法
3. 非静态方法既能调用静态方法也能调用非静态方法，能访问实例变量，也能访问类变量

静态变量和实例变量的区别

1. 所属不同
   1. 静态变量属于类 🡪 类变量
   2. 实例变量属于对象 🡪 对象的变量
2. 内存中存储的位置不同
   1. 静态变量存储在方法区的静态区
   2. 实例变量存储在堆中
3. 在内存中出现的时间不同
   1. 静态变量随着类加载而加载，随着类消失而消失
   2. 实例变量随着对象的创建而存在，随着对象的消失而消失
4. 调用不同
   1. 静态变量可以通过类名调用（非私有的），也可以通过对象调用（不推荐）
   2. 实例变量是只能通过对象调用

Static代码块：

1. java中{}括起来的叫做代码块
2. 静态代码块：在类中，方法外，加上static修饰，用于给类进行初始化，在加载类的时候就执行，并且只执行一次一般用于加载驱动（JDBC）（声音，视频，音频，图片。。。）

Static优点：调用时简单，快捷，方便

Final：

1. 修饰成员变量：在声明时必须初始化，不可更改
   1. 声明的时候初始化
   2. 构造方法初始化
   3. 也可以修饰局部变量
2. 修饰方法：用final修饰的方法，该方法不能被重写（但是能被继承）
   1. 意义：防止子类在定义方法时造成“不经意的”的重写
3. 修饰类：用final修饰的不能被继承，但是可以继承别的类
   1. 意义：可以保护类不能被继承，可以控制滥用继承，对程序系统造成的伤害
4. Final和static搭配使用，用来修饰常量，常量名（大写）用以区别其他变量
   1. 例如：static final int NUM = 50;
   2. Static final 修饰常量，必须在声明同事初始化，不可更改，所有对象共有的
   3. Static final 在编译时会被直接替换具体的常量值
   4. 好处：效率高，方便维护

内部类：一个类可以定义在另外一个类中，外部的叫做外部类，内部的叫做内部类

内部定义在外部类中，通常只服务于外部类，对外部的其他类不可见，内部类可以直接调用外部的成员变量和方法（包括私有的）

匿名内部类---重点

何时用：

如果在一段程序中需要创建一类对象（一个对象），而且这个对象创建后，称之为匿名内部类

注意：创建对象的这个类一般需要继承某个类或者实现某个接口

Java中编辑器认为：父类大于子类，接口大于实现类强制类型转换的前提：存在继承或者实现关系

文件中有几个class编辑器就会生成几个class文件

抽象类：类中没有办法确定实际的逻辑业务，这个时候可以不写方法体，但是java中不建议空的方法体，因此引入一个关键字abstract来修饰方法，此时该方法不谢方法体，为不完整的方法即为抽象方法，既然方法不全，表示类也不全，所有用abstract修饰该类，整体为抽象类

1. 有抽象方法的类不一定是抽象类，抽象类中不一定有抽象方法
2. 抽象类不能创建对象（不能实例化）：抽象方法不全—抽象类不完整—不能创建对象（实例化）
3. 意义：可以用子类来继承这个类，去实现拓展里面的内容（需要重写）父类中的所有的抽象方法；可以提供一种公共的数据类型
4. 综合：
   1. 抽象方法：由abstract修饰，只有方法定义，没有方法体（abstract不能修饰变量）
   2. 抽象类： 由abstract修饰的类，可以有抽象方法，也可以没有抽象方法，同时可以包含普通方法
   3. 包含抽象方法的类，一定是抽象类，抽象类不一定有抽象方法
   4. 抽象类不能实例化（子类去实现方法体）
   5. 抽象类一般需要被继承（子类去实现方法体）
      1. 子类也可以定义为抽象类
      2. 子类（普通类）继承抽象类，需要重写抽象类中所有的抽象方法
   6. Abstract 和final不能同时修饰一个类，因为final不能被继承，而abstract需要被继承实现才有意义（功能有冲突），同理也不能同时修饰方法
   7. 一个继承抽象类之后，必须重写其全部的抽象方法，不同的子类可以有不同的实现
5. 抽象类的意义：封装子类的成员，为子类提供一种公共的数据类型
6. 定义抽象方法，由子类做不同的实现，但是入口（方法名）是一样的
7. 注意：和普通类的区别：
   1. 抽象类不能实例化
   2. 抽象类可以有抽象方法

接口：

1. java是面向对象的编程语言
2. 面向接口编程：是一面向对对象为基础的，面向接口编程
3. 接口就是一种标准，一个规范（目的是制定你要遵循的规范）关键字：interface表示这是一个接口
4. 接口中只能包含常量（较少） 和抽象方法（居多）
5. 接口不能实例化，实现接口的类，叫做实现类，引入一个关键字implements实现接口，同时接口支持多实现,多个接口之间用逗号隔开，如果一个实现类即继承又实现，则遵循先继承再实现（接口与接口之间支持多继承）
6. 实现类实现接口，必须实现接口中所有的抽象方法
7. 接口的实现类的权限修饰符等于接口中的权限修饰符

抽象类和接口

1. 成员变量的区别：
   1. 抽象类：
      1. 可以是变量也可以常量
      2. 有构造方法
      3. 成员方法可以是抽象方法也可是非抽象方法
   2. 接口
      1. 成员变量只能是常量
      2. 没有构造方法
      3. 成员方法只能是抽象方法
2. 关系区别：类与类，类与接口，接口与接口
3. 相同点：都不能被实例化，需要多态时实现实例化

Javame：微型的javase，在java的基础上

Javase：基础java🡪 一些基本的类库，常见的一些接口，抽象类，方法等

Javaee： 只有文档注释才会生成对应的文档，其余注释皆不显示

@see 参考文献

@version 封装这个类时，所使用的jdk版本 ---jdk 1.6版本

@since 从哪个jdk版本开始

/\*\* 文档注释

\*/

生成doc文档： export 🡪 java 🡪 javadoc

生成jar 包： export 🡪 java 🡪 jarfile

导入jar 包 : create lib files 🡪 build path 🡪 add build path

Web ： export 🡪 java 🡪 wa file

导出后，直接放入服务器上，运行

字符串的底层就是字符数组

String：引用数据类型

1. 字符串字面量“abc”，可以看成一个字符串对象
2. 字符串是常量，一旦定义不可更改
3. 正常情况下，引用类型创建对象一般是new关键字创建对象，但是字符串使用频率过高，java对字符串做了一定的妥协可以不使用new创建对象

== 与 equals的区别：

== ：基本类型：比较的是值

引用类型：比较的是地址值

Equals：一般用于引用类型类型的比较

1. 这个类中没有重写equals，比较地址值
2. 这个类中已经重写equals，这时是根据业务要求来写equals中的逻辑比较
3. String：已经重写equals，比较的是字符串的内容，如果内容相同返回true

字符串使用字面量创建对象时，会在堆内存中有一个常量池，来存储字符串，在创建对象时，会首先在常量池中寻找是否有与该对象“一样”的对象，如果一样就将该常量池中的地址值赋值给该变量，如果不一样则会在常量池中重新创建该字符串字面量对象

字面量和new的关系

New表示创建对象，强制性操作，只要出现new都会在内存中开辟空间存储数据

New创建对象在常量中分：

1. 常量池中没有相同的对象，会在常量池和常量池之外的内存中，都创建对象
2. 如果常量池中有相同内容的对象，则在常量池中不再创建新对象，但是在常量池之外的内存中，依然会创建新对象

编辑器特性：当一个表达式运算两边都是字面量时，那么编辑器运算时就会先计算，将结构生成在表达式的位置，减少运算开销

属于字符串常用的方法

1. length()返回字符串的长度
2. 字符串的小标是从0开始到length -1 结束
3. 检索指定字符的位置，返回值为下标
   1. 先整体匹配，然后返回第一个字符的小标
   2. 如果没有匹配，返回-1

StringBuilder

StringBuffer

它们的用法写法一致，包含的方法也一致

StringBuilder: 线程不安全，并发处理，性能较快

StringBuffer： 线程安全，同步处理，性能稍慢

这些方法返回值为当前类型的对象的好处：

1. 因为字符串是需要频繁的修改，使用创建如上创建引用对象的方法修改字符串比较繁琐，此时使用该方式封装方法，可以对字符串进行连续修改，

正则表达式：

本身不属于java，一般用于字符串的格式验证

1. matches 方法： 将一个字符串与正则表达式进行全匹配，返回boolean
2. split方法：进行对字符串以正则表达式分割。返回数组
3. replaceAll方法：将字符串中匹配正则表达式reg的字符串替换成rep

包装类：

针对基本类型—基本类型的包装类

Java是面向对象的编程语言，操作的数据都是以对象为目标。

基本类型🡪引用类型（对象）

包装类：是将基本类型包装成引用类型，可以通过new关键字创建对象，符合就java面向对象编程的核心思想，封装成对象，方便后期对数据的统一操作

1. 自动装箱
2. 自动拆箱
3. intValue()：将基本类型的对应的包装拆箱成基本类型
4. int I = new integer(23)自动拆箱

Number是byte、double、float、integer、long、short的父类

Object是char、boolean的父类

Object：是所有引用类型的顶级父类

1. 没有成员变量，有很少的方法
2. –toString()：描述一个类（对象）的信息的方法，后期只是作为调试代码的一种工具方法而已
3. Equals()：比较各个类型对象，不同类型的对象比较是没有意义的，但是可以进行比较的操作
4. 类当中重写toString方法之后，不管写与不写，都会自动调用toString方法
5. Equals是用于比较两个不同对象是否相似，可以根据业务要求，如果某些属性相同可以认为是“同一对象” – 重写

Date：

1. getTime():获取当前时间距离1970年1月1日 00：00:00
2. 返回值long毫秒数

日历类在：java.util.Calendar

因为Date类对时间分量的操作不是很方便，所以引入Calendar

该类是一个抽象类，其方法可以对时间分量进行逐一设置

Add(int fiel, int num); 直接在时间分量上加减

加：正数

减：负数

使用add增减天数，（会自动根据日历计算）

集合：

是新的数据结构，可以保存不同类型的元素，

而且可以动态拓展

集合由来：

数组定义，数组的长度就固定，不可更改，当添加元素超过了数组的长度，就需要重新定义数组，只能存相同类型的元素

Java内部给我们提供了集合类，该类的实例可以存任意对象，长度是可以动态改变的，随着元素的增加或者减少而改变空间大小

集合和数组的区别：

1. 数组既可以存基本类型的数据，也可以存引用类型数据，基本类型存的是值，引用类型存的是地址值，
2. 集合只能存储引用类型数据，存储的是地址值，也能存储基本类型的数据，但是在存储过程中，会自动装箱成引用类型存入集合
3. 数组长度是固定的，不能自动增减，集合长度是不定的，可以随着元素的增减而动态改变
   1. 如果元素个数是固定的，则推荐使用数组
   2. 如果元素的个数是不定的，则推荐使用集合

集合的继承体系：

1. 父接口：Collection—(Collections集合的工具类)
   1. 子接口：继承Collection接口
      1. List：有序，可以重复的集合 -- 元素下标
         1. 存取是有顺序的
         2. 实现类：ArrayList：数组式（数组实现）
            1. LinkedList：链表式（链表实现）
            2. Vector：数组式
      2. Set：无序，不可以重复的集合（部分有序）--- 元素值
         1. 存和取是一样的
         2. 实现类：
            1. HashSet：哈希算法
            2. TreeSet：二叉树算法

Collection.add(null) 此时Collection不是空

集合的遍历：两种

1. 迭代器遍历
   1. 迭代器遍历集合：Iterator接口
      1. Java希望我们使用统一的方式来遍历集合，定义了遍历集合的接口，同时定义了如何遍历集合—使用迭代器
      2. 迭代器遍历的步骤：
         1. 获取迭代器
         2. 问：hasNext()：询问是否有下一个元素，给予指针的操作，每次调用该方法，指针会下移，返回的是当前指针指向的元素
         3. 取
         4. 删除：在用迭代器遍历的时候，不能使用集合的删除方法，否则会报错（Java.util.ConcurrentModificationException）
2. 增强型for循环

增强型，for循环

使用增强型for循环遍历集合 – Java5.0出现的新特性，该循环不通用于传统for循环工作，其知识用于集合或者数组的遍历

语法：for（元素类型 元素名：集合或者数组名）{

}

增强型for循环不是新语种，知识在编译器在编译生成.class文件时，将新循环转成迭代器，所以在遍历集合时，不能通过集合的方式来改变集合

增强型for循环的本质为迭代器。

注意点：不允许在遍历的过程中修改集合元素，因为本质是迭代器

泛型：

1. 主要作用是限制集合中所存放的元素的类型，方便后期对集合数据的操作，所有集合都支持泛型，迭代器也支持泛型
2. Jdk1.5之后引入的概念

<Integer> 🡪泛型

List 集合：

1. 是可以通过元素下标来操作元素（下标是从0开始）
2. ArrayList：数组结构—检索数组比较快
3. LinkedList：链表结构—插入和删除速度快
4. 在处理数据量比较大的时候，体现的较为明显
5. subList(int index, int endIndex)如果子集发生改变，对应的原集合也会发生改变
6. 集合与数组之间的转换
   1. toArray()：自动转成Object类型的数组，比较麻烦
   2. toArray(new 引用类型[数组长度])：这种转换方式会直接转成我们想要的类型的数组
      1. 数组长度：定义的数组长度，大于集合元素的个数，多余的部分，补充null，
      2. 定义的数组长度，小于集合元素的个数，系统会给我们定义一个默认为集合个数的数组长度，不再使用我们自己定义的数组长度
7. 数组转集合：一般是转成List集合（set中不允许重复）
   1. asList，转成对应泛型的List集合
   2. 从数组转成集合后，不能向集合中添加元素，否则会报异常，需要重新建立一个集合，将该转成后得集合复制过去，就可以想新集合中添加元素
   3. 使用addAll()使用创建集合的有参数的构造方法

Collections：是关于一个Collection接口集合的一个类（工具类），里面封装着一些对集合操作的特定的方法

Sort(List<T> list)：按照默认的排序顺序，给给定的集合进行排序

1. 比较器
   1. 排序接口：comparable，相当“内部比较器”，若一个类实现该接口，意味着“该类支持排序“，该类的对象拥有着比较的资格，但是需要重新定义比较规则，即重写compareTo()方法
      1. 该方法用于使当前对象与给定的对象进行比较，返回的值为int类型的值，与值得大小无关，与值得正负有关
      2. 若当前对象大于给定得对象，返回值大于0 降序
      3. 若当前对象小于给定得对象，返回值小于0 升序
      4. 若当前对象等于给定得对象，返回值为0，顺序不变
   2. 接口comparator比较器，相当于“外部比较器“，不用修改源代码，而是实现另外一个比较器；当对象需要比较时，将对象集合和比较器一起传过去即可
   3. 匿名内部类
      1. Comparator<String> c = new Comparator<String>{};
   4. 什么时候使用外部比较器：
      1. 一般情况下，是当集合中的元素，已经实现了comparator接口，重写了比较规则，但是该比较规则满足不了业务需求，此时我们需要定义一个外部比较器，按照业务需求定义比较规则，如果只是想使用一次，则最好定义成匿名内部类
2. Collection的子接口：
   1. Set接口：无序不重复（添加重复元素会失败）
   2. Set<引用类型> set = new hashSet<引用类型>

队列：是一种特殊的线性表，它限制了对线性表的访问从一端添入，另一端取出

Poll 取出元素，队列中也出队的操作

Peek只能取队首的引用，队列中依旧存在的，不能进行出队操作

队列只能进行一次操作，操作之后，队列中就不存在元素了Queue

栈：其实也叫双端队列，用来记录一组可追溯的数据，计算机当中的栈的原理就是使用队列做的

原则：先进后出，每次操作只能栈顶元素 Deque

入栈push，，，出栈只能从栈顶pop，，，

栈的遍历也是一次性的，只能遍历一次，