Day13 Arrays -- 包装类的使用

一、Arrays工具类：针对数组进行操作

public static String toString(int[ ] a ) 把其他任意类型转换为字符串

publlic static void sort(int[] a) 对任意类型的数组进行排序

public static int binaySearch(int[] a , int key); 对任意类型的数组进行二分查找（数组必须是有序的）

开发原则：只要是对象就要判断是不是为空

二、基本类型包装类：为了对基本数据类型进行更多的操作

(基本类型) （引用类型）

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Char

boolean Boolean

三、Integer的构造方法：

public Integer(int value); 将一个基本的int类型的数据变成integer包装类型数据

public Interger(String s) ;;将一个string类型的数据变成integer

（条件是原始的string类型的数据必须只能是数字字符组成，不能是英文字符组成）

注意：如果要将int类型或者是string类型的数据转换成integer的话

必须用构造方法，创建对象进行转换

四、int 与 String 转换

\* int -----String

\* 最佳用：String.valueOf(int i)

\*

\* String-----int

\* Interger.parseInt(String s);

五、Integer进制的转换

十进制转换为其他进制： (只能是0 -----26)

public static String toString(int i,int radix); //radix表示想要转换成的进制数

其他进制到十进制

public static int parseInt(String s ,int radix)//将一个其他进制换成是十进制必须要是可以转变的

六、自动装箱和拆箱：

自动装箱：把基本类型转换为包装类类型

注意：对象不能为空，不然调用空指针报错，抛异常

Integer x = null 会抛异常，，每次判断对象，必须对对象进行是否为空判断

自动拆箱：把包装类类型转换为基本类型

七、面试题

针对-128 到 127 的数据，有一个数据缓冲区，如果数据是在此范围内，每次并不创建新的空间

所以对于== 判断或者equals判断都为true

例如：

Integer i =100;

Integer i2 = 100;

i = =12 ; 结果为真

i.equals(i2); 结果也为真

八、character

构造方法：public character(char value)；

成员方法：

public static boolean isUpperCase (char ch);//判断字符是不是大写

public static boolean isLowerCase (char ch );//判断字符是不是小写

public static boolean isDigit(char ch) ; //判断指定字符是不是数字

public static char toUpperCase(char ch);//把指定的字符转成大写

public static char toLowerCase(char ch);//把指定的字符转成小写

Day13 StringBuffer:

线程安全的可变字符串

一、StringBuffer 与String的区别？

a: 前者长度和内容可变，后者不可变

b:如果使用前者进行字符串的拼接，不会浪费太多的资源

二、StringBuffer的构造方法：

public StringBuffer(); 无参构造函数

public StringBuffer(int capacity) 指定容量的字符串缓冲区对象

public StringBuffer(String str)；指定字符串内容的字符串缓冲区对象

StringBuffer的方法：

public int capacity (); 返回当前容量 理论值 (默认初始值为16)

public int length() : 返回长度（字符数）， 实际值

StringBuffer的添加功能:

字符的设置和替换：

setCharAt(int index,char ch) ,将指定的字符放在index指出的位置

charAt(int index);获取指定位置的字符

字符的插入：（在指定的位置插入字符或者其他类型的数据）

insert(int offset,char ch)； 在offset位置插入字符ch

public StringBuffer insert(int offset ,String str)

字符的添加：（默认连接到末尾）

public StringBuffer append(String str);

StringBuffer的删除功能：

public StringBuffer deleteCharAt(int index) ; // 删除指定位置的字符

public StringBuffer delete(int start ,int end)；删除指定位置开始指定位置结束的内容，并且返回本身

StringBuffer的替换功能：

public StringBuffer replace(int start ,int end,String str)；//从start开始到end用str替换

StringBuffer的反转功能：

public StringBuffer reverse();

StringBuffer的截取功能：

(返回值类型不同，返回String类型的数据，将截取的数据放在String类型中，本身字符串不变)

public String substring(int start);

public String substring(int start,int end);

面试题：

StringBuffer与数组的区别：

1、两者都可以当成容器。但是StringBuffer能够放不同类型的数据，最终形成字符串

数组能放多个数据，但是只能放同种类型的数据

String与StringBuffer作为参数类型返回：

1、基本类型

2、引用类型：必须创建相应的对象

注意：String类型的参数转变和基本类型是一样的

常量池的内容不变

三、冒泡排序：

1、外层控制次数，内层控制排列

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

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

int temp = arr[y];

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

arr[y+1]= temp;

}

}

syso();

四、选择排序

————\*\*\*\*-————

第一层循环的数据和第二层循环的数据进行比较

————\*\*\*\*————

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

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

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

int temp = arr[x];

arr[x] = arr[y];

arr[y] = temp;

}

}

}

五、高级二分查找

基本查找：从头找到尾 （一般是无序的）

\*高级二分查找：对半找 （前提条件必须是有序的）

\*

\* 思路：

\* A:定义最小索引min，最大索引max

\* B:计算出中间索引mid

\* C:拿中间索引的值与要查找的元素进行比较

\* 相等 ，就直接返回当前的中间索引

\* 不相等

\* 大了，往左边找

\* 小了，往右边找

\* D:重新获取最大索引值以及最小索引值

\* 大了 ，往左边找

\* max = mid-1;

\* 小了，往右边找

\* min = mid+1;

\* E:重新回到B位置

Day14 校验，正则表达式，Random - Date - calendar

正则表达式：符合一定规则的字符串

正则表达式组成规则：

A：字符

x 字符 x , 举例 ： 'a' 表示字符a

\\ 反斜线字符

\n 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,c

[a-zA-Z] a 到z 或者A 到 Z 两头字母包括在内

[0-9] 0 到9的字符都包括

C：预定字符

. 任何字符 如果是 . 字符本身，表示为\.

\d 数字：[0-9]

\w 单词字符[a-zA-Z\_0-9]

表示在正则表达式中组成单词的东西必须有这些组成

D：边界匹配器

^ 行的开头

$ 行的结束

\b 单词边界

就是不是单词字符的地方

举例：hello world？haha;xixi

E: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

正则表达式的功能：

判断功能：

public boolean matches(String regex); 判断字符串是否匹配给定的正则表达式

分割功能：String类的public String[ ] spilt(String regex); 根据给定的正则表达式的匹配拆分此字符串

注意：1、对路径进行分割，应该将\改成\\ 对应正则表达式两个\\对应路径上一个\

2、. 分割 直接写.表示的是任意字符 如果要表示成.的话要写成\\.

替换功能：

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

注意： 使用给定的replacement 替换此字符串所有匹配正则表达式的子字符串

//定义一个字符串

String string ="caddo12345javaandweb";

//定义新的串接收替换后的字符串

String string2 = string.replaceAll("\\d+", "\*");

//注意\\d+ ， "\*" 表示出现数字一次或一次以上替换一个\*给原字符串

// \\d , "\*" 表示数字出现一次就替换一个\*给原字符串

获取功能：

Pattern 和Matcher类的使用

模式匹配器的基本使用顺序

1、把正则表达式编译成模式对象 public static Pattern compile(String) //将给定的正则表达式编译到模式中

Pattern p = Pattern.compile("a\*b");

2、通过模式对象得到匹配器对象

Matcher m = p.matcher("aaaaab");

3、调用匹配器对象的功能

boolean b = m.matches();

通过Matcher类的 public boolean find( );查找是否存在符合条件的子字符串

通过Matcher类的 public String group( ) ;返回由以前匹配操作所匹配的输入子序列

注意：调用group必须先调用find ，， 先找是否存在再来输出匹配的子字符串

Math类

成员变量 static final double E //自然底数

static final double PI //圆周率

成员方法：

public static int abs(int a) ;绝对值

public static double ceil(double b) ; 向上取整 返回最小的大于b的值

public static double floor(double b) ; 向下取整 返回最大的小于b的整数

public static double pow(int a,int b); a的b次幂

public static double random( ); 返回0.0-1.0的随机数

public static double sqrt(double b) ; 返回b的正平方根

public static int round(其他类型的值) ；返回最接近参数的int值

Random类：

构造方法：

public random() 没用给种子，默认是当前的毫秒值

public random(long seed)给出指定的种子

区别： 给定种子后，每次得到的随机数是相同的

成员方法：

public int nextInt();返回int类型的随机数

public int nextInt(int n)； 返回（0，n）范围内的随机数

System类：

方法：

public static void gc() 运行垃圾回收器 重写Object类的finalize() 方法，进行回收

public static void exit(int status) 终止当前正在运行的 Java 虚拟机。

public static long currentTimeMillis() 返回以毫秒为单位的当前时间

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

从指定源数组中复制一个数组，复制从指定的位置开始，到目标数组的指定位置结束。

src - 源数组。

srcPos - 源数组中的起始位置。

dest - 目标数组。

destPos - 目标数据中的起始位置。

length - 要复制的数组元素的数量。 最终用Arrays.toString( int[] arr);输出数组

BigInteger :可以让超过Integer范围内的数据进行运算

构造方法：BigInteger(String value )；

public BigInteger[] divideAndRemainder(String value)

//以数组的形式将商和余数放在数组中

Date类：表示特定的时间，精确到毫秒值

构造方法：

Date( ) ; 根据当前默认的毫秒值创建日期对象

Date(long date) ;根据给定的毫秒值创建日期对象

成员方法：

public long getTime( );获取时间，以毫秒为单位

public void setTime(long time) :设置时间

注意：

从Date得到一个毫秒值：

创建Date对象，调用getTime( ）方法

把一个毫秒值转换为Date:

1、构造方法

2、setTime( )方法

DateFormat: (抽象类无实例对象)

进行日期和字符串格式化和解析：可以将Date和String类型进行转换：默认使用simpleDateFormat具体类实现

simpleDateFormat

构造方法：

simpleDateFormat( );默认模式

simpleDateFormat(String pattern ) ; 给定的模式 pattern代表输出的模式规则

规则表示：1、年y 2、月M 3、日d 4、 时H 5、 分m 6、 秒s

例如：yyyy-MM-dd HH:mm:ss 输出当前时间

成员方法：

public String format(Date d) ; 将Date类型转换为以字符串类型输出的模式

public Date parse(String s) ; 将字符串转换成一个Date类型的数据

注意一点，使用SimpleDateFormat需要捕获ParseException异常。

Calendar：类是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY\_OF\_MONTH、HOUR 等 日历字段之间的转换提供了一些方法，并为操作日历字段

//其日历字段由当前日期和时间进行初始化

Calendar rightNow = Calendar. getInstance(); //返回一个子类的对象

成员方法:

public int get(int filed); 返回给定字段的值，日历类中的每个日历字段都是静态的成员变量，且为int类型

注意：month 从0 开始显示

public void add(int filed, int amount)根据给定日历字段和对应的时间，来对当前的日历进行操作

public final void set(int year,int month,int date);设置当前日历的年月日

Day15 Collection 集合

Collection:是集合接口；

功能概述： (自动重写了toString方法)

1、添加功能：

boolean add(Object obj);添加一个元素

boolean addAll(Collection c) ;添加一个集合的元素(将另一个集合所有的元素)

2、删除功能：

void clear() 移除所有元素

boolean remove(Object obj)移除一个元素

Boolean removeAll(Collection c);移除一个集合的元素（移除一个就为true）

3、判断功能；

boolean contains(Object c) ;判断集合中是否包含指定的元素

boolean containsAll(Collection c) ; 判断集合中是否包含指定集合元素(只有包含所有的元素才是包含)

boolean isEmpty();判断集合是否为空

5、长度功能：

int size() ; 元素的个数

6、交集功能：

boolean retainAll(Collection c) :两个集合

（两个集合都有的元素，如果存在则为true，交集到的元素放在调用方法的集合中存储）

7、把集合转换为数组，实现数组遍历

Object[] toArray()

4、获取功能：（遍历） （迭代器依赖集合存在的）

Iterator<E> iterator()

Iterator ; 是一个接口，对象调用方法其实是一个子类的对象，采取的是多态的方式

Object next();获取元素，并移动到下一个位置

（注意，但是一旦超出范围就会抛异常）

所以每次获取时要判断是否存在元素可以取出，则用 hasNext()判断是否还有元素可以进行遍历

迭代器依赖集合对象去调用iterator方法进行迭代 ：Iterator 迭代对象 = 集合对象.iterator();

迭代对象.next(); 是一个object类对象 向下转型输出

二、集合的使用步骤

1、创建集合对象

2、创建元素对象

3、把元素添加到集合

4、遍历集合

a:通过集合对象获取迭代器对象

b:通过迭代器对象的hasNext() 方法判断是否含有元素

c:通过迭代器对象调用next() 方法获取元素并且移动到下一个位置

三、迭代器的原理及实现

原理：定义为一个接口，定义功能，但是未实现 ，由子类的内部类去实现方法

四、List类:

有序(存储和取出的元素顺序是一致的)的Collection序列 ,通过索引来操作。（与set不同其可以插入重复的）

List集合的特有功能： （索引值不能越界）

1、添加功能：

void add(int index, Object element) //在列表的指定位置插入指定元素 (能紧跟上一个元素索引，但是不能越界)

2、获取功能：

Object get(int index) //获取指定位置的元素

3、列表迭代器：

listIterator( ) ;//list集合特有迭代器

listIterator(int index) //迭代从指定位置的集合

4、删除元素

Object remove(int index) //根据索引删除元素，返回被删除的元素

5、修改元素：

Object set (int index, Object element) ;//根据索引修改元素，返回修改的元素

特有的遍历（ListIterator）：

1、get（）方法和size（）方法结合使用

2、特有遍历方法：(正向遍历)

a: hasNext() 和next()方法结合

3、Object prevoius( ) // 返回列表的前一个元素 （相当于倒着遍历）

boolean hasPrevious( ) //判断倒着遍历是否还要元素

注意：可以实现逆向遍历但是必须先正向遍历才可以逆向遍历，所以无啥意义，一般不适用

迭代器异常处理：出现不合法的并发修改时，抛异常

原因：因为迭代器是依赖于集合而存在的，当集合进行修改时迭代器不知道，所有抛异常

处理方法：

1、普通方法进行遍历get() 和size() 方法结合

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

String s = (String) list.get(i); //此时的S是一个字符串类型的对象，不单纯指代一个字符串

if("world".equals(s)) {

list.add("javaee");

}

}

System.out.println(list);

2、迭代器进行修改元素 ListIterator才有修改元素的方法

注意：添加对的元素在迭代元素的后面出现

//迭代器添加元素 // (此时添加的元素在迭代元素的后面)

ListIterator it = list.listIterator();

while(it.hasNext()) {

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

if("world".equals(s)) {

it.add("javaee");

}

}

System.out.println(list);

五、栈 ： 先进后出

队列： 先进先出

六、List的三个子类特点：

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

注意》不会分析用那种就用ArrayList

七、数组与集合的区别

A：长度区别

数组长度固定

集合长度可变

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

数组只能存储同种类型的数据

集合可以存储不同类型的数据

Day16 ArrayList ----- LinkedList - Vector --泛型

一、Vector特有功能：

1、添加功能：

public void addElement(Object obj); 将指定的组件添加到此向量的末尾，且向量的大小加 1。

2、获取功能：

public Object elementAt(int index) ;返回指定索引处的组件

public Enumeration elements( ) //返回向量组件的枚举

// Enumeration 是一个接口，因此与迭代器差不多，其实就是一个子类实现类对象

boolean hasMoreElements( ) //判断是否还有枚举元素

Object nextElements( ) //获取枚举元素

二、LinkedList特有的功能：

1、添加功能：

public Object addFirst( Object obj) //在集合第一个位置进行添加

public Object addLast(Object obj) //在集合最后一个位置进行添加 （其实与单独的add方法无区别，因此没啥意义）

2、获取功能：(对最终的集合元素无影响)

public Object getFirst( ) //获取集合中第一个元素

public Object getLast( ) //获取集合中的最后一个元素

3、移除功能：（对最终集合元素有影响）

public Object removeFirst( ) //移除集合中第一个元素

public Object removeLast( ) //移除集合中的最后一个元素

三、删除ArrayList中重复的元素

需求：删除集合中自定义对象的重复值（对象的成员变量值都相同）

为什么自定义对象删除时进行equals判断时容易出错

为什么呢？

判断功能contains( )是集合自动生成的，因此查看源码得知

contains ( )方法其实本质上比较的是equals方法

而学生类中没有equals（）方法，因此默认使用父类Object类的equals（）方法最终比较的地址值

因为每一个对象的地址值均不同，所以未被删除掉

所有按照自我的需求，比较成员变量的值，需要在自定义类中重写equals（）方法

四、用LinkedList实现默认自定义栈集合

五、泛型：是一种特殊的类型，把明确的工作推迟到创建对象或者调用方法的时候才明确的特殊的类型

格式：<数据类型> //这里的数据类型只能是引用类型

好处：

A: 将运行时会出现的问题提前到编译期间

B; 避免了强制转换

C: 优化了程序设计，解决黄色警告线

集合也会模仿数组的数据类型明确，因此这种技术也被叫做是泛型

出现原因：早期的Object可以存储任意类型，进行向上转型时不会出现错误，但是向下转型时容易抛安全问题

六、泛型类

《一》格式：类名<T> //T表明之后要用什么类，就定义成什么

注意：定义的泛型类必须是包装类

1、具体定义的类型（对类进行操作）具体进行初始化new对象 //泛型类

2、对方法进行操作（对方法进行定义），相应的对象不需要进行泛型定义 //泛型方法

3、接口定义成泛型类 //泛型接口

注意：接口定义为泛型类时实现类不需要定义，，当接口不明确类型时，实现类也需要定义为未知类型<T>

《二》泛型的高级通配符

1、泛型如果明确了类型时，前后应该一致

例：Collection<Object> c = new ArrayList<Object>( ) ;

2、泛型如果是用 ？ 表示任意的类型都可以

例：Collection<?> c = new ArrayList<任意类型皆可>( ) ;

3、泛型用 ？ extends E 表示类型只能是 E类或者其子类 向下转型皆可

例：Collection<? extends E> c = new ArrayList<E或者子类>( ) ;

4、泛型如果用 ? super E 表示类型只能是 E或者父类 向上转型

例：Collection<? super E> c = new ArrayList<E或者E的父类>( ) ;

七、JDK新特性

1、自动装箱，自动拆箱

2、增强for : 是for循环的一种

格式：

for(元素数据类型 变量 ： 数组或者Collection集合 ) {

使用变量即可，该变量就是元素

}

优缺点： 好处：优化了数组以及集合的遍历

缺点：容易出现当对象为空时，抛异常

解决方案 ： 使用前先对对象是否为空进行判断

八、静态导入

格式：import 包名.类名.方法名

(一般不使用)

注意：

1、导入的方法必须是静态方法

2、当出现多个同名的方法时候，必须在使用导入方法前加上前缀（java.lang.math.ads(-10)）

九、可变参数

---在定义方法的时候不知道参数有多少个

格式

修饰符 返回值类型 方法名 （数据类型... 变量名）{ }

注意：

1、此处的变量名其实是一个数组

2、如果这个方法有多个参数，并且有可变参数，那么这个可变参数应该定义在最后

3、因为从本质上此处的可变参数是一个数组，数组的长度是固定的，因此当可变参数位于前面的时候表明

数组已经是固定的且不可改变的

十、Arrays工具类中的一个方法 ：将数组转换成集合

public static <T> List<T> asList(T... a）

//经过这个方法将数组转换为集合时，本质上是一个数组，长度是固定的，因此对集合不能进行长度修改有关的操作

//如add、remove

Day17 Set - HashSet ---- TreeSet

一、Set集合

要求：一个不包含重复元素的 collection(不允许重复出现相同的元素)

无序（元素进行存储和取出的顺序不一样）

元素唯一性

二、HashSet ：存储数据是无序并且唯一的

问题：为什么存储字符串内容时只存储了一个呢？

通过查看add方法的源码，我们知道这个方法实际上是利用了hashCode( ) 和 equals（）方法

步骤;

首先比较哈希值

如果相同，比较地址值或者用equals（）方法

如果不同，就添加到集合中

按照方法步骤来说：

A：先看hashCode( )值是否相同

相同; 用equals方法比较

返回true : 代表元素重复，不添加

返回false: 添加到集合

不同：直接添加到集合

/-------如果自定义的类没有重写这两个方法，默认使用Object类的方法（比较不出是否相同）

为什么String可以进行方法的使用？

因为String重写了方法

三、LinkedHashSet : 底层数据结构由哈希表和链表组成

哈希表保证元素的唯一性

链表保证元素有序（存储和取出顺序相同 ）

四、哈希表：一种有算法计算出来的哈希值 （相当于新华字典查找元素）

五、TreeSet : 能够对元素按照某种规则进行排序

特点： 元素排序且唯一

排序两种方式：

A：自然排序 （实现Comparable接口就是自然排序） 构造方法默认使用自然排序

// 通过源码可知，add( )方法里最终排序调用的是compareTo( )方法

//真正的比较依赖的于元素的compareTo（）方法，而这个方法是定义在Comparable接口里的

//所以如果要重写该方法，就必须实现Comparable接口，这个接口表示的就是自然排序

B：比较器排序

有参构造：

TreeSet(Comparator<? super E> c);

构造一个新的空 set，该 set 根据指定的比较器进行排序。

1、@@ (Comparator<? super E> c代表的是接口实现类的父类或者本身

其实就是定义了一个实现Comparable接口的类，对这个类进行创建对象

2、格式为

TreeSet<Student> set = new TreeSet<Student>(new Comparable接口的实现类)；

此时应该在接口实现类中重写接口里的compare（）方法

@@@@@@ 注意：如果一个方法的参数是接口，其实要的是接口的实现类对象

匿名内部类可以实现该操作

六、TreeSet ：底层是二叉树结构（红黑树是一种自平衡的二叉树）

元素如何存储进去？

1、第一个元素作为根节点进行存储

2、从第二个元素开始，每个元素都和根节点进行比较

大 就作为右孩子

小 就作为左孩子

相等 就不添加

元素如何取出？（前序遍历，中序遍历，后序遍历）

前序遍历：

1.访问根节点

2.前序遍历左子树

3.前序遍历右子树

中序遍历：

1.中序遍历左子树

2.访问根节点

3.中序遍历右子树

后序遍历：

1.后序遍历左子树

2.后序遍历右子树

3.访问根节点

七、TreeSet集合保证元素排序唯一的原理

唯一性：是根据比较返回的0来决定的

排序：

A：自然排序;(元素具备比较性)

让元素所属的类去实现自然排序接口Comparable

B：比较器排序;(集合具备比较性)

让集合构造方法接收一个比较器接口的子类对象Comparator

七、针对Collection集合我们到底如何使用呢？

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,使用 HashSet

否：List

要安全吗？

是：Vector

否; ArrayList 或LinkedList

查询多：ArrayList

增删多：LinkedList

如果知道要用List , 但是不知道用哪个，就用ArrayList

如果知道要用集合，不知道用什么就用ArrayList

Day18 Map集合

一、Map<K , value> 接口

特点：存储键值，获取元素, 无序

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一个值。

注意：键值对应是一个Set集合(元素唯一性)，value对应的是一个List集合（元素可以重复）

二、 Map和Collection 集合的区别？

1、Map集合存储元素是成对存在的，键值不能重复，只能对应一个元素，元素可以重复

Collection集合：存储元素是单独出现的，元素可以重复（List）也可以不重复(Set )

2、Map集合数据结构是针对键值，跟值无关

Collection集合数据结构是针对元素有效

三、Map的功能概述

1、添加功能：

V put(K key, V value) ; 添加元素

注意事项：当键是第一次存储就返回null ， 当键不是第一次存储就把第一次的元素进行替换(如果输出时是返回第一次的元素)

2、删除功能

void clear( ); 移除所有的键值对元素

V remove(Object key) ; 根据键值移除掉键值元素对，并返回

3、获取功能：(V代表泛型)

Set<Map.Entry<K,V>> entrySet() ; 获取所有的键值和元素

V get(Object key) ; 根据键值获取元素

Set<K> keySet() ; 获取集合元素中所有的键值 ,并且存储到集合中

Collection<V> values() : 获取集合中的所有元素

4、判断功能：（包含返回true）

boolean containsKey(Object key) ; 判断集合是否包含特定的键值

boolean containsValue(Object value) ; 判断集合是否包含特定的元素值

boolean isEmpty() ; 判断集合是否为空

5、长度功能

int size( ) ; 返回集合中的键值对长度

四、Map集合的遍历：

方式1：根据键找值

1、获取所有键的集合

2、根据键的集合获取每一个键值

3、根据键值输出元素值

方式2：根据键值对对象找值

1、获取所有键值对象的集合

2、遍历键值对对象的集合，获取每一个键值对对象

3、根据键值对象找键和值

五、HashMap: 是基于哈希表的Map接口实现

特性：哈希表用来保证元素的唯一性

注意事项：HashMap对键进行的操作，具有唯一性，如果要进行键的添加put( ) 方法操作时则要依靠hashCode( ) 和equals( ) 方法

而如果键的数据类型是字符串类型，或其他包装类，可以直接操作，因为自动重写了那两个方法

但是如果是自定义类型的话，需要重写hashCode( ) 和equals( )方法

六、LinkedHashMap : 是Map接口的哈希表和链表实现

哈希表：保证元素是唯一的

链表：保证键值是有序的(有序不代表排序);

八、TreeMap: 键是红黑树，保证键是唯一的;

1、自然排序：无参构造实现自然排序,键值所在类实现comparable接口 ,并且在类中重写compareTo() 方法

2、比较器排序：创建集合有参构造默认实现比较器排序,需要创建接口的实现类对象,,接口实现类重写compare()方法，

因为实际上需要的实现类对象，而操作方法的参数是接口的方法，

可以通过创建匿名对象类创建对象，并重写方法

九、集合嵌套集合

1、HashMap嵌套HashMap集合

2、HashMap嵌套ArrayList集合

3、HashMap 嵌套 HashMap 再嵌套ArrayList集合

十、面试题

1、HashMap 和Hashtable 的区别：

HashMap ：线程不安全，效率高，允许null键和null值

Hashtable ：线性安全，效率低，不允许null键和null值

2、List , Set , Map 等接口是否都继承Map接口?

List , Set : 继承的是Collection 接口

Map : 其本身就是一个顶层接口

十一、Collections； 是针对集合进行操作的工具类，都是静态方法

十一、Collection和Collections 的区别？

Collection : 是单列的顶层集合接口，有子接口List和Set

Collections : 是对集合进行操作的工具类，有对集合进行排序和二分查找的方法

十一、Collections

public static <T> void sort(List<T> list) ; 排序默认使用自然排序

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

public static <T> T max (Collection<?>coll) ; 最大值

public static void reverse(List<?> list) ;反转

public static void shuffle(List<?> list) ; 随机置换

十二、ArrayList存储自定义对象并且进行排序

注意：ArrayListu元素是有序，指代的并不是元素的大小或其他格式的排序，而是存储和取出的顺序是相同的

因此称为是有序的，，要对其进行排序可以使用Collections集合操作类对集合进行排序，使用其静态方法sort( )

但是此操作类对集合进行排序操作时，是依靠Comparable接口的，所以自定义的类实现接口，还要重写接口的方法

重写compareTo( ) 方法

十三、集合的特点，数据结构的总结

1、集合分为两个体系：Collection （单列集合） 和Map（双列集合)

Collection : 单列集合

List : 有序，可以重复

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 ： 底层数据结构是数据，查询快，增删慢

线程安全，效率低

LinkedList ： 底层数据结构，查询慢，增删快

线程不安全， 效率高

Set : 无序，元素唯一

HashSet : 底层数据结构是哈希表

比较元素是否重复，保证元素唯一性

哈希表依赖于两个方法进行判断hashCode( ) 和equals ( ) 方法

1、先比较哈希值hashCode( )

· 相同，再接着比较地址值或equals( ) 方法

返回true； 代表元素重复，不添加

返回false : 代表元素可以进行添加操作

不同 ： 直接添加元素到集合中

注意：如果是自定义的类，需要重写以上两个方法，可以直接系统生成

TreeSet : 底层数据结构是一种二叉树（红黑树是一种自平衡的二叉树）

二叉树，进行排序操作 根据返回值是否为0 ，进行排序

自然排序：(对元素所在的类进行实现接口Comparable)

依赖于无参构造，实现接口Comparable，重写compareTo（）方法

比较器排序 (集合具备比较性，让集合接收一个comparator实现类对象)

依赖于有参构造，产生接口的实现类，重写接口的compare( ) 方法

注意，最重要的是接口的实现类对象，所以直接使用匿名内部类进行操作，重写compare( ) 方法

linkedHashSet : 底层数据结构是链表和哈希表

特点： 由链表确定元素的有序性，有哈希表确认元素的唯一性

Map： 双列集合 是对键值进行的操作，产生的是键值对 , 数据结构仅仅针对键有效，值无效

一种键和值进行映射的关系，键值不允许重复，元素值可以是重复的，每个键值最多对应一个元素值

HashMap : 线程不安全，效率高，允许键为null，值为null

键的数据结构是一种哈希表，保证键值唯一

对键进行添加操作是，使用put(T key , T value) ；方法中，依赖的是两个方法hashCode( ) 和equals（)

最终系统进行自动生成即可

linkedHashMap : 线程安全，效率低，不允许键值和元素为null

键数据结构是一种，链表和哈希表

链表确认键值存取和取出顺序相同，哈希表确认键值唯一性

和上述HashMap : 差不多用法

TreeMap : 键的数据结构是红黑树

自然排序：(对元素所在的类进行实现接口Comparable)

依赖于无参构造，实现接口Comparable，重写compareTo（）方法

比较器排序 (集合具备比较性，让集合接收一个comparator实现类对象)

依赖于有参构造，产生接口的实现类，重写接口的compare( ) 方法

注意，最重要的是接口的实现类对象，所以直接使用匿名内部类进行操作，重写compare( ) 方法