# Java中的集合类 (具有不同数据结构的容器)

Collection

|--List:有序可重复。因为该集合体系有索引。

|--ArrayList:底层是数组。特点：查询速度很快。但是增删稍慢。线程不同步。

|--LinkedList:底层是链表。特点：增删速度很快，查询稍慢。线程不同步。

|--Set：元素是无序，元素不可以重复。

|--HashSet:底层数据结构是哈希表。是线程不安全的。不同步。

通过调用hashCode和equals来保证元素的唯一性。

注意,对于判断元素是否存在，以及删除等操作，依赖的方法是元素的hashcode和equals方法。

|--TreeSet：可以对Set集合中的元素进行排序。底层数据结构是二叉树。

TreeSet ,保证元素唯一性的依据: compareTo方法return 0。

TreeSet排序的第一种方式：让元素自身具备比较性。

元素需要实现Comparable接口，覆盖compareTo方法。

这种方式也称为元素的自然顺序，或者叫做默认顺序。

TreeSet的第二种排序方式。

当元素自身不具备比较性，或者具备的比较性不是所需要的。

这时需要让容器自身具备比较性。

定义比较器class MyCompare implements Comparator，并覆盖compare方法，将比较器对象作为参数传递给TreeSet集合的构造函数。

当两种排序都存在时，以比较器为主。

迭代器 :

迭代器是取出方式，会直接访问集合中的元素,所以将迭代器通过内部类(所有容器的迭代器都实现了Iterrator接口)的形式来进行描述。

可以通过容器的iterator()方法获取该内部类的对象。

# 泛型：不确定的数据类型 - - Java1.5版本新特性

泛型出现之后的好处

1. 将运行时期出现问题ClassCastException，转移到了编译时期,将错误信息从运行时期转移到了编译时间,但编译完成泛型被擦除。定义反省,在.class文件中仍能添加其他类型,(Java中的反射)

方便于程序员解决问题。让运行时问题减少，安全。

1. 避免了强制转换麻烦。

泛型高级应用 :通过泛型编写一些通用的Java程序,结合反射!

泛型格式：通过<>来定义要操作的引用数据类型。

其实<> 就是用来接收类型的。当使用集合时，将集合中要存储的对象类型作为参数传递到<>中即可。

# 泛型类的定义

T template ,E elements ,K key ,V valus

List<E> E称为类型参数变量

ArrayList<String> String称为实际类型参数

什么时候定义泛型类？

当类中要操作的引用数据类型不确定的时候，早期定义Object来完成扩展。

现在定义泛型来完成扩展。

//定义工具类，获取不确定的引用对象。在类上定义泛型，在创建对象时传入泛型。则该泛型//在整个类有效。

class Util <T> {

private T t;

public void setT(T t){

this. t = t;

}

public T getT() {

return t;

}

}

class Person{

}

public class Test {

public static void main(String[] args) {

Util<Person> util = new Util<Person>();

util.setT(new Person());

Person person= util.getT();

}

}

# 泛型方法的定义

ArrayList<Person> mList = **new** ArrayList<Person>();

mList.add(**new** Person("121"));

//可以传入子类型

mList.add(**new** Student("2222"));

name(mList);

**public** **static** **void** name(List<? **extends** Person> list) {

Iterator<? **extends** Person> iterator =list.iterator();

**while** (iterator.hasNext()) {

Person person = iterator.next();

System.*out*.println(person.toString());

}

}

定义在方法返回值之前 .

练习: \* 编写一个通过数组交换元素方法

\* 编写一个通过数组倒序方法

**public** **static** <T>**void** reverse(T t[]){

**for** (**int** i = 0; i < t.length/2; i++) {

T tt;

tt=t[i];

t[i]=t[t.length-1-i];

t[t.length-1-i]=tt;

}

}

特殊之处：

静态方法不可以访问类上定义的泛型。

如果静态方法操作的应用数据类型不确定，可以将泛型定义在方法上。

**class** Demo<T> {

**public** **void** show(T t) {

System.***out***.println("show:"+t);

}

**public** <Q> **void** print(Q q ) { //调通该方法是传入什么类型，就是什么泛型

System.***out***.println("print:"+q);

}

**public** **static** <W> **void** method(W t) {

System.***out***.println("method:"+t);

}

}

# 泛型定义在接口上

**interface** Inter<T>{

**void** show(T t);

}

//实现该接口的子类要操作的数据类型也不确定。

**class** InterImpl<T> **implements** Inter<T>{

**public** **void** show(T t){

System.***out***.println("show :"+t);

}

}

# 泛型限定

? 通配符,代表任意类型。也可以理解为占位符。

使用通配符,不要使用泛型相关的方法,因为泛型不确定!!

泛型的限定 限定泛型可接受的范围

定义在方法的参数上

? extends E: 可以接收E类型或者E的子类型。上限。

? super E: 可以接收E类型或者E的父类型。下限

# 上下边界应用：

范例一

Set 中 addAll(Collection<? extends E> c) ----- 将目标集合c的内容添加到当前set , ? extends E目标集合是E的子类型

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

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

set.addAll(list); // list 中 Integer 自动转换为 Number

范例二

TreeSet(Comparator<? super E> comparator)构造函数 ,类泛型Set<E>

Set<Apple> set = new TreeSet<Apple>(); //按照Apple的比较器排序

class FruitComparator implements Comparator<Fruit> {}//定义水果比较器

Set<Apple> set = new TreeSet<Apple>(new FruitComparator()); //传入 Fruit比较器

Map集合

该集合存储键值对。一对一对往里存。而且要保证键的唯一性。

Map

|--HashMap：底层是哈希表，允许使用 null 值和 null 键，该集合是不同步的。

|--TreeMap：底层是二叉树。线程不同步。可以用于给map集合中的键进行排序。

和Set很像,Set底层就是使用了Map集合。

1. 添加。

put(K key, V value)

putAll(Map<? extends K,? extends V> m)

1. 删除。

clear()

remove(Object key)

1. 判断。

containsValue(Object value)

containsKey(Object key)

isEmpty()

1. 获取。

get(Object key)

size()

values()

1. 遍历

keySet() 将map中键转换成Set集合，然后通过迭代器取出键，取出value。

Iterator<String> iterator = map.keySet().iterator();

entrySet()

将map集合中的映射关系存入到了set集合中，而这个关系的数据类型就是：Map.Entry。Entry是Map的一个静态内部接口。定义在内部直接访问map集合数据。

Iterator<Map.Entry<String, String>> it=

map.entrySet().iterator();

**while** (it.hasNext()) {

Map.Entry<String, String> entry = it.next();

System.***out***.println(entry.getKey());

System.***out***.println(entry.getValue());

# 集合框架工具类

# Collections:集合框架的工具类。里面定义的都是静态方法。

Collections和Collection有什么区别？

Collection是集合框架中的一个顶层接口，它里面定义了单列集合的共性方法。

它有两个常用的子接口，

List：对元素都有定义索引。有序的。可以重复元素。

Set：不可以重复元素。无序。

Collections是集合框架中的一个工具类。该类中的方法都是静态的

提供的方法中有可以对list集合进行排序，二分查找等方法。

通常常用的集合都是线程不安全的。因为要提高效率。

如果多线程操作这些集合时，可以通过该工具类中的同步方法，将线程不安全的集合，转换成安全的。

# Arrays:用于操作数组的工具类。里面都是静态方法。

asList:将数组变成list集合

List<String> list = Arrays.asList(arr);

把数组变成list集合有什么好处？

可以使用集合的思想和方法来操作数组中的元素。

注意：将数组变成集合，不可以使用集合的增删方法。因为数组的长度是固定。会发生UnsupportedOperationException异常。

contains。

get

indexOf()

subList();

如果数组中的元素都是对象。那么变成集合时，数组中的元素就直接转成集合中的元素。

如果数组中的元素都是基本数据类型，那么会将该数组作为集合中的元素存在。

//int[] nums = {2,4,5}; //该数据转变成集合，nums会作为元素存在。int不是对象

Integer[] nums = {2,4,5};

List<Integer> li = Arrays.asList(nums);

# 集合变数组。

Collection接口中的toArray方法。

String[] arr = al.toArray(new String[al.size()]); //String [] arr = new String[al.size()]

System.out.println(Arrays.toString(arr));

# Java可变数组

可变参数。其实就是上一种数组参数的简写形式。不用每一次都手动的建立数组对象。只要将要操作的元素作为参数传递即可。隐式将这些参数封装成了数组。

在使用时注意：可变参数一定要定义在参数列表最后面

show("haha",2,3,4,5,6);

show("haha"); //默认传入null。打印结果为0；

public static void show(String str,int... arr){

System.out.println(arr.length);

}

# Java中的其他对象

1. System

System.out 标准输出，控制台。

System.in 标准输入，键盘。返回inputStream。

获取系统属性信息：Properties getProperties();

1. Runtime

该类并没有提供构造函数。说明不可以new对象。

该类肯定会提供了方法获取本类对象。而且该方法是静态的，并返回值类型是本类类型。

由这个特点可以看出该类使用了单例设计模式完成。

通过static Runtime getRuntime()获取Runtime对象;

Runtime r = Runtime.*getRuntime*();

Process p = r.exec("notepad.exe SystemDemo.java"); //执行文件

//Thread.sleep(4000);

//p.destroy(); //杀掉子进程，只能杀掉Java虚拟机自己启动的进程

1. Date

Date d = **new** Date(); //创建时间对象

System.***out***.println(d);//打印的时间格式不好辨认

//将模式封装到SimpleDateformat对象中。

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

//调用format方法让模式格式化指定Date对象。

String time = sdf.format(d);

System.***out***.println("time="+time);

1. Calender

Calendar c = Calendar.*getInstance*();

String[] mons = {"一月","二月","三月","四月"

,"五月","六月","七月","八月"

,"九月","十月","十一月","十二月"};

String[] weeks = {"","星期日","星期一","星期二","星期三",

"星期四","星期五","星期六", };

//返回月份 0~11

**int** index = c.get(Calendar.***MONTH***);

//返回星期 1~7 代表周日~周六

**int** index1 = c.get(Calendar.***DAY\_OF\_WEEK***);

System.***out***.println(weeks[index1]); //输出星期，其他相同。

1. Math – Random

random() 返回带正号的 double 值，该值大于等于 0.0 且小于 1.0。

(int)(Math.random()\*10+1) //生成1~10随机数。

//生成随机数 ， 通过Random对象。

Random random = **new** Random();

**while**(**true**){

System.***out***.println(random.nextInt(10)); //0~9

}