**四大接口：Collection、Map、Iterator、Enumeration**

**允许重复的接口 ArrayList：**

新的子类（ArrayList 重点 90%）

范例：测试ArrayList

**package** Set;

**import** java.util.List;

**class** Person{

**private** String name;

**private** **int** age;

**public** Person(String name,**int** age){

**this**.name = name;

**this**.age = age;

};

@Override

**public** **boolean** equals(Object obj) {

**if**(obj == **null**){

**return** **false**;

}

**if**(**this** == obj){

**return** **true**;

}

**if**(!(obj **instanceof** Person)){

**return** **false**;

}

Person person = (Person)obj;//向下转型以便操作子类的属性

**if**(person.name.equals(**this**.name) && person.age == **this**.age){

**return** **true**;

}

**return** **false**;

}

**public** String toString(){

**return** "姓名：" + **this**.name + "年龄：" + **this**.age;

}

}

**public** **class** ArrayList {

**public** **static** **void** main(String[] args) **throws** Exception{

java.util.List<Person> all = **new** java.util.ArrayList<Person>();//向集合中传入Person类，表示

//该集合是Person类的集合

all.add(**new** Person("张三", 21));//类集合

all.add(**new** Person("李四", 20));

all.add(**new** Person("王五", 25));

all.add(**new** Person("赵六", 15));

System.***out***.println(all.contains(**new** Person("李四", 20)));//检查是否存在对象

System.***out***.println(all.remove(**new** Person("王五", 25)));//删除王五

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

System.***out***.println(all.get(i));//得到集合中的对象元素

}

}

}

旧子类 Vextor(10%)

ArrayList与Vector的区别：（绝大部分都用ArrayList子类）

|  |  |  |  |
| --- | --- | --- | --- |
| No | 区别 | ArrayList | Vector |
| 1 | 推出时间 | JDK1.5时推出 | JDK1.0时推出 |
| 2 | 性能 | 采用异步处理方式，性能较高 | 采用同步处理方式，性能较差 |
| 3 | 安全性 | 数据安全性差 | 数据安全性高 |
| 4 | 输出 | Iterator、foreach、listItorator | Iterator、foreach、listItorator、enumeration |

**ArrayList为动态数组，linkedList为链表。此为二者区别。**

**不允许重复接口 Set:**

Set之中保存的元素是不能够有重复的，但是Set接口并不像List接口那样对Collection进行方法的扩充，而是继承了Collection接口所有方法，那么也就意味着在之前使用的get()方法不能使用了。

Set子接口之中重要使用的两个常用子类：HashSet、TreeSet

范例：HashSet

**public** **class** SetTest\_1 {

**public** **static** **void** main(String[] args) **throws** Exception {

Set<String> all = **new** HashSet<String>();

all.add("hello");//Set集合不允许重复数据

all.add("hello");//保存了重复的数据

all.add("文攀");

all.add("HELLO");

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

}

}

**关于数据排序说明：**

使用TreeSet可以实现集合中的数据排序的保存，那么在之前使用的是String类型数据，下面使用自定义类。但是如果想要正常的使用TreeSet保存，那么一定要在此类上实现java.lang.Comparable接口，以区分大小、和按指定的属性排序。

范例：使用自定义类

**package** set;

**import** java.util.List;

**import** java.util.Set;

**import** java.util.TreeSet;

**class** Person **implements** Comparable<Person>{

**private** String name;

**private** **int** age;

**public** Person(String name,**int** age){

**this**.name = name;

**this**.age = age;

};

**public** **int** compareTo(Person o){//增加实现的comparable接口的compareTo方法实现排序

**if**(**this**.age > o.age){

**return** -1;

}

**else** **if**(**this**.age < o.age){

**return** 1;

}

**return** **this**.name.compareTo(o.name);//年龄比较完后比较姓名

}

**public** String toString(){

**return** "姓名：" + **this**.name + "年龄：" + **this**.age;

}

}

**public** **class** ArrayList {

**public** **static** **void** main(String[] args) **throws** Exception{

// java.util.List<Person> all = new java.util.ArrayList<Person>();//向集合中传入Person类，表示

Set<Person> all = **new** TreeSet<Person>();//利用TreeSet集合实现对象的排序，但是要实现comparable接口

//该集合是Person类的集合

all.add(**new** Person("张三", 21));//类集合

all.add(**new** Person("李四", 21));

all.add(**new** Person("王五", 25));

all.add(**new** Person("王五", 25));

all.add(**new** Person("赵六", 15));

// System.out.println(all.contains(new Person("李四", 20)));//检查是否存在对象

// System.out.println(all.remove(new Person("王五", 25)));//删除王五

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

}

}

通过以上程序可以得出一个结论，TreeSet使用过程之中一定要是用到Comparable接口，而且TreeSet接口里面是依靠的comparable接口的CompareTo()方法来实现区分数据是否为重复数据

**关于重复数据的说明：**

Set子接口中不允许保留重复数据，之前所用的TreeSet子类虽然依靠了Comparable接口进行重复诗句的区分，但是此方式也只针对于TreeSet这种排序结构起作用，而真正意义上的排序不是依靠接口的，而是依靠Object类之中的

两个方法实现：

取得对象的hash码 ：public int hashCode();

对象比较： public boolean equals(Object obj);

比较是否重复的流程如下：

1. 首先使用要查询对象的hashcode与集合之中保存的对象的hashcode进行一一比较，
2. 如果hashcode相同，则再使用equals()方法进行内容的比较。

范例：真正意义上的Set集合重复对象判断

**package** set;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**import** java.util.TreeSet;

**class** Person{

**private** String name;

**private** **int** age;

**public** Person(String name,**int** age){

**this**.name = name;

**this**.age = age;

};

@Override

**public** **int** hashCode() {//使用eclipse的自动导入hashCode()方法和自动导入equals()方法实现Set集合的自动取出重复

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + age;

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (!(obj **instanceof** Person))

**return** **false**;

Person other = (Person) obj;

**if** (age != other.age)

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

**public** String toString(){

**return** "姓名：" + **this**.name + "年龄：" + **this**.age;

}

}

**public** **class** ArrayList {

**public** **static** **void** main(String[] args) **throws** Exception{

// java.util.List<Person> all = new java.util.ArrayList<Person>();//向集合中传入Person类，表示

Set<Person> all = **new** HashSet<Person>();

//该集合是Person类的集合

all.add(**new** Person("张三", 21));//类集合

all.add(**new** Person("李四", 21));

all.add(**new** Person("王五", 25));

all.add(**new** Person("王五", 25));

all.add(**new** Person("赵六", 15));

// System.out.println(all.contains(new Person("李四", 20)));//检查是否存在对象

// System.out.println(all.remove(new Person("王五", 25)));//删除王五

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

}

}

**集合的输出：**

**1.迭代输出：Iterator（核心，95%）**

Iterator接口中的几个方法：

Public boolean hasNext(); 判断是否有数据

Public E next(); 取出数据

Public void remove(); 删除数据

**但是Iterator本身就是接口，若想要为该接口实例化对象则必须要使用collection接口中的iterator()方法：public Iterator<E> iterator();取得Iterator接口对象；**

关于集合的输出都用Iterator输出。

范例：使用Iterator迭代输出

**public** **class** IteratorTest\_1 {

**public** **static** **void** main(String[] args) {

List<String> all = **new** ArrayList<String>();//集合对象的实例化

all.add("HELLO");//向集合中增加数据

all.add("hello");

all.add("word");

all.add("WORD");

//定义迭代器，利用Collection接口中的iterator方法为迭代器实例化

Iterator<String> iterator = all.iterator();

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

String data = iterator.next();

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

}

}

}

**2.古老的输出Enumeration:(核心 )**

Enumeration中的几个方法：

Public boolean hasMoreElements(); 是否有下一个元素

Public E nextElement(); 取得元素

**但是想要取得 Enumeration实例必须使用Vector类中的方法：public Enumeration <E> element();**

范例：使用

**public** **class** IteratorTest\_1 {

**public** **static** **void** main(String[] args) {

Vector<String> all = **new** Vector<String>();//集合对象的实例化

all.add("HELLO");//向集合中增加数据

all.add("hello");

all.add("word");

all.add("WORD");

Enumeration<String> enu = all.elements();

**while**(enu.hasMoreElements()){

String data = enu.nextElement();

System.out.print(data + "、");

}

}

}

**双值保存的最大父接口：Map(重点)**

Collection每次只能保存一个对象，而Map每次可以保存一对对象，按照(key,value)的形式存放，随后可以通过key找到对应的value。

**新的子类：HashMap(90%)**

Map集合中定义了如下常用方法：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Public V put(K key,V value) | 普通 | 向集合之中保存数据 |
| 2 | Public V get(Object key) | 普通 | 根据key取出对应的value |
| 3 | Public Set<K> KetSet() | 普通 | 取出全部的key，key不能重复 |
| 4 | Public Collection<V> values() | 普通 | 取得全部的value |
| 5 | Public Set<Map Entry<K,V>entrySet()> | 普通 | 将Map集合变为Set集合 |
|  |  | 普通 |  |

而在Map接口下又有两个常见的子类：HashMap,HashTable

范例： 利用hashmap验证操作

**public** **class** MapTest\_1 {

**public** **static** **void** main(String[] args) **throws** Exception{

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

map.put(1,"张三");

map.put(1,"李四");

map.put(2,"王五");

map.put(3,"null");

map.put(3,"赵六");

//在map集合的使用中如果出现了key重复的问题，那么会用新的value值替代原来的值

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

}

}

**在map集合的使用中如果出现了key重复的问题，那么会用新的value值替代原来的值。在collection中设置数据的目的是为了输出，而在Map中设置数据是为了查找。**

**旧的子类：Hashtable(10%)**

**范例：验证HashTable**

**public** **class** MapTest\_1 {

**public** **static** **void** main(String[] args) **throws** Exception{

Map<Integer, String> map = **new** Hashtable<Integer,String>();

map.put(1,"张三");

map.put(1,"李四");

map.put(2,"王五");

map.put(3,"null");

map.put(3,"赵六");

//在map集合的使用中如果出现了key重复的问题，那么会用新的value值替代原来的值

System.***out***.println(map.get(1));

// System.out.println(map.get(null));

System.***out***.println(map.get(200));

}

}

**HashMap与Hastable的区别：**在Hashtable中可以发现，里面是不能够存在null数据的。

**关于Map集合的输出问题：（核心）**

Map集合要实现输出，那么一定要用iterator。但是在Map集合中没有像collection集合中的iterator()方法，

可以发现在Map接口里面存在一个static定义的子接口：Map.Entry,而所有的Key和value都封装在接口对象之中，而在Map.Entry接口里面存在两个方法：

1. 取得包装的key:public K getKey();
2. 取得包装的value：public V getValue();

**关于Map的iterator输出方法：**

1. 首先使用Map集合中的entrySet()方法，将Map集合编程Set集合，而此时Set集合中的每一个元素类型都是Map.Entry接口实例。
2. 利用Set接口的iterator()方法，取得iterator接口对象，同时Iterator中操作的类型依然是Map.entry;
3. 利用Iterator进行迭代输出，取出每一个Map.Entry对象，并且利用getKey()和getValue()方法取出数据

**范例：利用iterator输出Map集合**

**public** **class** MapTest\_2 {//使用iterator输出map的重点方法

**public** **static** **void** main(String[] args) **throws** Exception{

Map<Integer,String> map = **new** HashMap<Integer,String>();//实例化Map集合

map.put(1,"张三");

map.put(2,"李四");

map.put(3,"王五");

map.put(4,"赵六");

Set<Map.Entry<Integer, String>> set = map.entrySet();//利用Map集合的entryset()方法将map集合转换为set集合

Iterator<Map.Entry<Integer, String>> iterator = set.iterator();//利用set集合的iterator()方法实例化迭代器

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

Map.Entry<Integer, String> me = iterator.next();//定义Map。Entry对象接收迭代器中的Map.Entry对象

System.***out***.println(me.getKey() + "-->" + me.getValue());//利用Map.Entry对象的getKey()和getValue()方法输出

}

}

}

Map集合之中Key的说明：

在之前的程序中都是利用的Integer作为key，但是也可以不用Integer作为key，可以用其他数据类型，比如String。也可以用自定义类型作为value和key.但是此时必须有一个要求：由于map集合根据key的查找属于对象匹配过程，所以自定义的key类型，一定要覆写Object类的hashcode（）和equals（） 方法。

范例：

**class Person{**

**private String name;**

**private int age;**

**public Person(){};**

**public Person(String name,int age){**

**this.name = name;**

**this.age = age;**

**};**

**@Override**

**public int hashCode() {**

**final int prime = 31;**

**int result = 1;**

**result = prime \* result + age;**

**result = prime \* result + ((name == null) ? 0 : name.hashCode());**

**return result;**

**}**

**@Override**

**public boolean equals(Object obj) {**

**if (this == obj)**

**return true;**

**if (obj == null)**

**return false;**

**if (!(obj instanceof Person))**

**return false;**

**Person other = (Person) obj;**

**if (age != other.age)**

**return false;**

**if (name == null) {**

**if (other.name != null)**

**return false;**

**} else if (!name.equals(other.name))**

**return false;**

**return true;**

**}**

**public String toString(){**

**return "姓名： " + this.name + "年龄： " + this.age;**

**}**

**}**

**public class MapTest\_2 {**

**public static void main(String[] args) throws Exception{**

**Map<String, Person> map = new HashMap<String,Person>();//value是自定义对象**

**map.put(new String("zs"), new Person("张三",21));**

**map.put(new String("ls"), new Person("李四",22));**

**// System.out.println(map.get("zs"));//根据key查找value 可以查找到**

**System.out.println(map.get(new Person("张三",21)));//根据value查找key 不能查找到 如果想要实现这种查找**

**//一定要在类中覆写Object类中的hashcode()和equals()方法。**

**}**

**}**

面试题：请解释collection和collections的区别：

Collection是集合操作的父接口。

CollectionS是集合操作的工具类，提供各种集合操作的支持。

**栈操作：**

Stack属于vector子类：需要用到进栈和出栈两个方法：

Public E push():进栈

Public E pop():出栈

**public** **class** StackTest\_1 {

**public** **static** **void** main(String[] args) {

Stack<String> all = **new** Stack<String>();

all.push("a");

all.push("b");

all.push("c");

all.push("d");

System.***out***.println(all.pop());

}

}

**属性操作类：properties(重点)**

Properties 是HashTable的子类，在HashTable操作过程之中，key和value都可以是任何数据类型，但是在Properties之中key和value只能是String类型数据，而且操作的方法如下：

1. 设置属性：public Object setProperty(String key,String value)
2. 取得属性：public String getProperty(String key), 没有属性返回null
3. 取得属性：public String getProperty(String key,String defaultValue);没有属性返回默认值

**范例：验证Properties使用**

**public** **class** StackTest\_1 {

**public** **static** **void** main(String[] args) {

Properties pro = **new** Properties();//建立属性类对象

pro.setProperty("SC", "四川");

pro.setProperty("YN", "云南");

System.***out***.println(pro.getProperty("SC"));

System.***out***.println(pro.getProperty("BJ"));//若没有此属性返回null

System.***out***.println(pro.getProperty("BJ","没有此属性！"));//若没有此属性则返回默认值

}

}

通过properties类设置的属性也是可以通过IO操作进行输出的，定义了如下输出方法：

保存属性：public void store(outputstream out，String comment) throws Exception,,,,,

范例：将属性保存到属性

**public** **class** StackTest\_1 {

**public** **static** **void** main(String[] args) **throws** Exception{

Properties pro = **new** Properties();//建立属性类对象

pro.setProperty("SC", "四川");

pro.setProperty("YN", "云南");

pro.store(**new** FileOutputStream(**new** File("D:" + //输出流

File.***separator*** + "wenpan" + File.***separator*** + "area.properties")),"area");

}

}

属性文件的后缀一般都是.properties

**通过输入流读取文件信息：**

**public** **class** PropertiesTest\_1 {

**public** **static** **void** main(String[] args) **throws** Exception{

Properties pro = **new** Properties();

pro.load(**new** FileInputStream(**new** File("D:" + File.***separator*** + "wenpan" + File.***separator*** + "area.properties")));

System.***out***.println(pro.getProperty("bj"));

//中文在属性文件中是以转码的形式存放的

//直接在输入文件中添加中文值 用输入流读入后显示乱码，需要再JDK安装目录下的bin文件夹下找到native2ascii.exe

//打开然后输入你要在输入文件中输入的字符，然后得到该文字的编码，然后将编码写到输入文件中，即可正常读入

System.***out***.println(pro.getProperty("SC"));

}

}

**总结：**

1. **类集就是一个动态对象数组，还不需要用户自己编写链表，十分方便**
2. **集合的核心操作：**
   * 1. **单值保存：Collection(public void add(),public Iterator iterator());**
        1. **允许重复：List接口（public E get(int index),使用ArrayList子类，优先考虑此类）**
        2. **不允许重复：Set接口，使用HashSet子类**

**偶对象保存：Map(put(),get(),entrySet()),主要是用HashSet子类，作为key的类型一般都是String**

**输出操作：**

1. **Iterator(hasNext(),next())只要是集合就是用Iterator输出；**
2. **Enumeration(hasMoreElements(),nextElement());只要老的输出使用**

**操作属性文件：使用Properties**