Java中的排序(Comparable接口)

关键词： Comparable 要实现排序功能,一般有两种途径,这里对基本类型不适用,基本类型一般有Arrays中的静态方法.

1.对象本身实现Comparable接口,那么该类的实例就是可以排序的.

有关Comparable: http://blog.csdn.net/treeroot/archive/2004/09/09/99613.aspx

只要实现了Comparable接口,就可以调用Collections的sort方法对集合中的元素排序.

2.指定一个Comparator,也就是实现了Comparator的类的一个实例.

但是Java本身只提供了一个Comparator的实现,就是Collections.reverseOrder().

该方法返回的是一个已经实现了Comparable接口的反序.

看一下Comparator的全部内容:

public interface Comparator {

int compare(Object o1, Object o2);

boolean equals(Object obj);

}

定义了两个方法,其实我们一般都只需要实现compare方法就行了,因为类都是默认从Object继承

所以会使用Object的equals方法.

Comparator一般都作为一个匿名类出现,对于没有实现Comparable的对象的集合,排序的时候

需要指定一个Comparator.

这里举例说明

对于实现了Comparable的类我们就用最简单的Integer

List list=new ArrayList();

list.add(new Integer(3));

list.add(new Integer(53));

list.add(new Integer(34));

Collections.sort(list);

对于没有实现Comparable的,我们就用Object,按照hashCode大小来排序.

List list= new ArrayList();

list.add(new Object());

list.add(new Object());

list.add(new Object());

Collections.sort(list,new Comparator(){ public int compare(Object o1, Object o2){

return (o1.hashCode()-o2.hashCode());}});

对集合排序的例子

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

/\*

\* 当前文件： TestCollections.java

\* 创建日期： 2006-6-29

\* 版 本 号： 1.0

\* 作 者： Stevie Liu

\*

\*/

/\*\*

\* 亲爱的kang1，请您在此详细描述我的用途.

\*/

class BookType{

String idsn;

String name;

/\*\*

\* @param idsn

\* @param name

\*/

public BookType(String idsn, String name)

{

super();

this.idsn = idsn;

this.name = name;

}

/\*\*

\* @return Returns the idsn.

\*/

public String getIdsn()

{

return idsn;

}

/\*\*

\* @param idsn The idsn to set.

\*/

public void setIdsn(String idsn)

{

this.idsn = idsn;

}

/\*\*

\* @return Returns the name.

\*/

public String getName()

{

return name;

}

/\*\*

\* @param name The name to set.

\*/

public void setName(String name)

{

this.name = name;

}

}

public class TestCollections

{

public static void main(String[] args)

{

List list=new ArrayList();

list.add(new BookType("1","k"));

list.add(new BookType("5","z"));

list.add(new BookType("4","g"));

Comparator OrderIsdn = new Comparator(){

public int compare(Object o1, Object o2){

BookType b1=(BookType)o1;

BookType b2=(BookType)o2;

return (b1.getIdsn().hashCode()-b2.getIdsn().hashCode());

}

};

Comparator OrderName = new Comparator(){

public int compare(Object o1, Object o2){

BookType b1=(BookType)o1;

BookType b2=(BookType)o2;

return (b1.getName().hashCode()-b2.getName().hashCode());

}

};

Collections.sort(list,OrderName);

Collections.reverse(list);

for(int i=0;i

BookType ss=(BookType)list.get(i);

System.out.print(ss.getIdsn());

System.out.println(ss.getName());

}

}

}

[Comparable接口分析](http://blog.csdn.net/treeroot/article/details/99613)

现看一下该接口的定义:  
public interface Comparable{  
   public int compareTo(Object o);  
}  
该接口定义类的自然顺序,实现该接口的类就可以按这种方式排序.  
一般要求:  
e1.equals((Object)e2)和e1.compareTo((Object)e2)==0具有相同的值,  
这样的话我们就称自然顺序就和equals一致.  
这个接口有什么用呢?  
如果数据或者List中的元素实现了该接口的话,我们就可以调用Collections.sort或者Arrays方法给他们排序.

如果自然顺序和equals不一致的话,如果出现在Sorted Map和Set里面,  
就会出现预想不到的逻辑错误,可能你调用add的时候添加不了,而集合里面确没有这个元素.具体的讨论要接口哈希表的应用.