链滴

Java 集合体系总结 Set、List、Map、Queue

作者：someone10186

原文链接：https://ld246.com/article/1519976148595

来源网站：链滴

一、java集合类基本概念

有时我们需要集中存放多个数据，一般情况下，数组就是一个很好的选择，前提是我们事先已经明确道我们将要保存的对象的数量。一旦在数组初始化时指定了这个数组长度。这样数组长度就不可变了如果我们想要保存一个可以动态增长的数据，java集合类就是一个很好的设计方案。

集合类主要负责保存其他数据，所以集合类一般也被成为容器类。所以的集合类都位于java.util包下。

1. Collection

一组服从某种规则的元素 □□1.1) List必须保持元素特定的顺序 □□1.2) Set不能有重复元素 □□1.3) Queu保持一个队列(先进先出)的顺序 2) Map

一组成对的"键值对"对象

Collection 和Map的区别在于容器中每个位置保存元素的个数

(1)Collection 每个位置只能保存一个元素 (2)Map 保存的是“键值对”，就像一个小型数据库。我们以通过键找到对应的值

二、Java集合类架构层次关系

1.Iterface Iterable

迭代器接口，这是Collection类的父接口。实现这个Iterable接口的对象允许使用foreach进行遍历，就是说，所有的Collection集合对象都具有“foreach可遍历性”。这个Iterable接口只有一个方法：iterator（）。他返回一个代表当前集合对象的泛型迭代器，用于之后的遍历操作。

1.1 Collection

Collection 是最基本的集合接口，一个Collection代表一组Object的集合，这些Object被称作Collection的元素。Collection是一个接口，用以提供规范定义，不能被实例化使用

1）Set

Set集合类似于一个桶，放进Set集合里的多个对象之间没有明显的顺序。Set继承自Collection接口，能包含有重复元素。

Set判断两个对象相同不是使用”==“运算符，而是根据equals方法。也就是说，我们在加入一个新素时，如果这个新元素对象和Set中已有对你好进行注意equals比较都返回false，则Ser就会接受这个元素对象，否则拒绝。

因为Set的这个制约，在使用Set集合时，应该注意两点：1是Set集合里的元素的实现类实现一个有效equals（Object）方法 2对Set的构造函数，传入的Collection参数不能包含重复的元素

1.1）HashSet

HastSet是Set接口的典型实现，HashSet使用HASH算法来存储集合中的元素，因此具有良好的存取查找性能。当向HashSet集合中存入一个元素时，HashSet会调用该对象的hashCode（）方法来得该对象的hashCode值，然后根据该HashCode值决定该对象在HashSet的存储位置。

1.1.1）LinkedHashSet

LinkedHashSet集合也是根据元素的hashCode值来决定元素的存储位置，但和HashSet不同是，它时使用链表维护元素的次序，这样使得元素看起来是以插入的顺序保存。

当遍历LinkedHashSet集合里的元素是，LinkedHashSet将会按元素的添加顺序来访问集合里的元素。

LinkedHashSet需要维护元素的插入顺序，因此性能略低于HashSet的性能，但在迭代访问Set全部素时，将会有很好的性能。

1.2）SortedSet

此接口主要用于排序操作，即实现此接口的子类都属于排序的子类

1.2.1）TreeSet

TreeSet是Sorted接口的实现类，TreeSet可以确保集合元素属于排序状态

1.3）EnumSet

EnumSet是一个专门为美剧类设计的集合类，EnumSet中所有元素都必须是指定悲剧类型的的枚举，该枚举类型在创建Enumset时显示或隐式的指定。EnumSet的集合元素也是有序的。

2）List

List集合代表一个元素有序，可重复的集合，集合中每个元素都有其对应的顺序索引。List集合允许加重复元素，因为他可以通过索引来访问指位置的集合元素，list集合默认按元素的添加顺序设置元素索引

2.1）ArrayList

ArrayList是基于数组实现的List类，他封装了一个动态的增长的，允许再分配的Object[]数组。

2.2）Vector

Vector和ArrayList在用法上几乎完全相同，但由于Vector是一个古老的集合，所以Vector提供一些法名很长的方法，之后将Vector改为实现List接口，统一归入集合框架体系铜

2.2.1）Stack

Stack是Vector提供的一个子类，用于模拟栈这种数据结构

2.3)LinkedList

implement List,Deque。实现List接口，能对他进行队列操作，即可以根据索引来随机访问集合中元。同时他还实现Deque接口，即能将LinkedList当做双端队列使用。自然也可以被当做“栈来使用”。

1.2Map

Map用于保存具有“映射关系”数据，因此Map集合里保存着两组值，一组值用于保存Map里的key另外一组值用于保存Map里的value。key和value都可以是任何引用类型的数据。Map的key不允许复，即同一个Map对象的任何两个Key通过equals方法比较结果总是返回false；

Map的这些实现类和子接口中key集的存储形式和Set集合完全相同（即key不能重复）

Map的这些实现类和子接口中value集的存储形式和List非常类似（即value可以重复，根据索引来查）

1）HashMap

和HashSet不能保证元素的顺序一样，HashMap也不能保证key-value对的顺序。并且类似于HashSe

判断两个key是否相等的标准也是：两个key通过equals（）方法比较返回true。

同时两个key的hashCode值也必须相等。

1.1）LinkedHashMap

LinkedHashMap也使用双向链表来维护key-value对的顺序，与key-value对的插入顺序一致（注意TreeMap对所有的key-value进行排序进行区份）

2）HashTable

2.1）properties

3）sortedMap

正如Ser接口派生出SortedSet子接口，SortedSet接口有一个TreeSet实现类一样，Map接口也派生一个SortedMap实现类

3.1)TreeMap

TreeMap就是一个红黑树数据结构，每一个key-value对即作为红黑树一个节点。TreeMap存储key-value对（节点）时，需要根据key对及诶单进行排序。TreeMap可以包含保证所有的key-value对都处有序状态。同样，TreeMap也有两种排序方式：自然排序，定制排序

3.Java集合类的Demo

1.Set

HashSet

import java.util.*

//类A的equals()方法总是返回true，但没有重写其hashCode()方法。不能保证当前对象是HashSet唯一对象

class A { public bollean equals(Object obj) { return true;

} } //类B的hashCode()方法总是返回true，但没有重写其equals()方法。不能保证当前对象是HashSe的唯一对象 class B { public bollean hashCode(Object obj) { return 1;

} } //类C的hashCode()方法总是返回2,且有重写其equals()方法 class C { public int hashCode() { retrn 2; } public boolean equals(Object obj) { return true; } }

public class HashSetTest { public static void main(String[] args) { HashSet books=new HashSet);

```
    //分别向books集合中添加两个A对象，两个B对象，两个C对象

    books.add(new A());
    books.add(new A());

    books.add(new B());
    books.add(new B());

    books.add(new C());
```

```
    books.add(new C());
    System.out.println(books);


} }
```

结果

[B@1, B@1, C@2, A@3bc257, A@785d65]

可以看出，如果两个对象通过equals（）方法比较返回true，但这两个对象的hashCode方法返回不的hashCode值时，这将导致HashSet会把这两个对象保存在Hash表的不同位置，从而使对象可以添成功，这就与Set集合的规则有些出入了。所以，我们要明确的是：equals（）决定是否可以加入HasSet，而hashCode（）决定存放的位置，他们两者必须同时满足才能允许一个新元素加入HashSet。

但是要注意的是：如果两个对象的hashCode相同，但是他们的equals返回值不同，HashSet会在这位置用链式结构来保存多个对象。而HashSet访问集合元素时也是根据元素的HashCode的值来快速位的，这种链式结构会导致性能下降。

所以如果需要把某个类的对象保存到HashSet集合中，我们在重写这个类的equasl（）方法和hashCoe（）方法时，应该尽量保证两个对象通过equals（）方法毕节哦返回true时，他们hashCode（）方返回值也相等。

LinkedHashSet

import java.util.*; public class LinkedHashSetTest { public static void main(String[] args) { LinkdHashSet books=new LinkedHashSet(); books.add('Java1'); books.add('Java2');

System.out.println(books); //删除 Java1 books.remove("Java1"); //重新添加 Java1 books.add("Jva1"); System.out.println(books); } }

输出

[Java1, Java2] [Java1, Java2]

元素顺序总是与添加顺序一致，同时要明白的是，LinkedHashSetTest是HashSet的子类，因为它不许集合元素重复

TreeSet

import java.util.*;

public class Test { public static void main(String[] args) { TreeSet nums = new TreeSet(); //向TreSet中添加四个Integer对象 nums.add(5); nums.add(2); nums.add(10); nums.add(-9);

```
    //输出集合元素，看到集合元素已经处于排序状态
    System.out.println(nums);
    [-9, 2, 5, 10]
    //输出集合里的第一个元素
    System.out.println(nums.first());
    -9
    //输出集合里的最后一个元素
    System.out.println(nums.last());
    10
    //返回小于4的子集，不包含4
    System.out.println(nums.headSet(4));
```

```
    [-9, 2]
    //返回大于5的子集，如果Set中包含5，子集中还包含5
    System.out.println(nums.tailSet(5));
    [5, 10]
    //返回大于等于-3，小于4的子集。
    System.out.println(nums.subSet(-3 , 4));
    [2]
}


}
```

与HashSet集合采用hash算法来决定元素的存储位置不同，TreeSet采用红黑树的数据结构来存储集元素。TreeSet支持两种排序方式：自然排序，定制排序

1.自然排序

TreeSet 会调用集合元素的compareTo（Object obj）方法来比较元素之间的太小关系，然后将集合素按升序排序，即自然排序，如果试图把一个对象添加到TreeSet时，则该对象的类必须实现Comparble接口，否则程序会抛出异常。

当把一个对象加入TreeSet集合中时，TreeSet会调用该对象的compareTo（Object obj）方法与容中的其他对象比较大小，然后根据红黑树结构找到他的存储位置。如果两个对象通过compareTo（Obect obj）方法比较相等，新对象将无法添加到TreeSet集合中（牢记Set不允许重复的概念）。

注意：当需要把一个对象放入TreeSet中，重写改对象对应类的equals（）方法是，应该保证该方法，应该保证该方法与compareTo（Object obj）方法有一致的结果，即如果有两个对象通过equals）方法比较返回true时，这两个对象通过compareTo（Object obj）方法比较结果应该也为0（即相）

对与Set来说，它定义了equals()为唯一性判断的标准，而对于到了具体的实现，HashSet、TreeSet说，它们又会有自己特有的唯一性判断标准，只有同时满足了才能判定为唯一性

2.定制排序

TreeSet的自然排序是根据集合元素的大小，TreeSet将它们以升序排序。如果我们需要实现定制排序则可以通过Comparator接口。该接口里包含一个int compare（to1，to2）方法，该方法用户比较小。

```
import java.util.*;

class M { int age; public M(int age) { this.age = age; } public String toString() { return "M[age:" + age + "]"; } }

public class Test { public static void main(String[] args) { TreeSet ts = new TreeSet(new Comparator() { //根据M对象的age属性来决定大小 public int compare(Object o1, Object o2) { M m1 = (M)o1; M m2 = (M)o2; return m1.age > m2.age ? -1 : m1.age < m2.age ? 1 : 0; } }); ts.add(new M()); ts.add(new M(-3)); ts.add(new M(9)); System.out.println(ts); } }
```

EnumSet

```
import java.util.*;

enum Season { SPRING,SUMMER,FALL,WINTER } public class EnumSetTest { public static void main(String[] args) { //创建一个EnumSet集合，集合元素就是Season枚举类的全部枚举值 EnumSet
```

```
es1 = EnumSet.allOf(Season.class); //输出[SPRING,SUMMER,FALL,WINTER] System.out.println(
s1);

    //创建一个EnumSet空集合，指定其集合元素是Season类的枚举值。
    EnumSet es2 = EnumSet.noneOf(Season.class);
    //输出[]
    System.out.println(es2);
    //手动添加两个元素
    es2.add(Season.WINTER);
    es2.add(Season.SPRING);
    //输出[SPRING,WINTER]
    System.out.println(es2);

    //以指定枚举值创建EnumSet集合
    EnumSet es3 = EnumSet.of(Season.SUMMER , Season.WINTER);
    //输出[SUMMER,WINTER]
    System.out.println(es3);

    EnumSet es4 = EnumSet.range(Season.SUMMER , Season.WINTER);
    //输出[SUMMER,FALL,WINTER]
    System.out.println(es4);

    //新创建的EnumSet集合的元素和es4集合的元素有相同类型，
    //es5的集合元素 + es4集合元素 = Season枚举类的全部枚举值
    EnumSet es5 = EnumSet.complementOf(es4);
    //输出[SPRING]
    System.out.println(es5);
}


}
```

输出

[SPRING, SUMMER, FALL, WINTER] [] [SPRING, WINTER] [SUMMER, WINTER] [SUMMER, FALL, WINTER] [SPRING]

以上是Set集合类的Demo，下面讲讲如何选择这些集合类呢?

（1）HashSet的性能总是比TreeSet好（贴别是最常用的添加、查询元素等操作），因为TreeSet需额外的红黑树算法来维护集合元素的次序。只有当需要一个保持排序的Set时，才应该使用TreeSet，则都应该使用HashSet

（2）对于普通的插入，删除操作，LinkedHashSet比HashSet略慢一线，这是由于维护链表所带来开销造成的。不过，因为有了链接的存在，遍历LinkedHashSet会更快

（3）EnumSet是所有Set实现类中性能最好的，但它只能保存一个枚举类的枚举值作为集合元素。

（4）HashSet，TreeSet，EnumSet都是“线程不安全”的。

2.List

ArrayList

如果一开始就知道ArrayList集合需要保存多少元素，则可以在创建他们时就指定大小，这样可以减少

新分配的次数，提供性能，ArrayList还提供了如下方法来重新分配Object[]数组。

1. ensureCapacity(int minCapacity): 将ArrayList集合的Object[]数组长度增加minCapacity

2. trimToSize(): 调整ArrayList集合的Object[]数组长度为当前元素的个数。 程序可以通过此方法来少ArrayList集合对象占用的内存空间

import java.util.*;

public class Test { public static void main(String[] args) { List books = new ArrayList(); //向boos集合中添加三个元素 books.add(new String("轻量级Java EE企业应用实战")); books.add(new Strig("疯狂Java讲义")); books.add(new String("疯狂Android讲义")); System.out.println(books);

```
    //将新字符串对象插入在第二个位置
    books.add(1, new String("疯狂Ajax讲义"));
    for (int i = 0; i < books.size(); i++) {
        System.out.println(books.get(i));
    }

    //删除第三个元素
    books.remove(2);
    System.out.println(books);

    //判断指定元素在List集合中位置：输出1，表明位于第二位
    System.out.println(books.indexOf(new String("疯狂Ajax讲义")));  //①
    //将第二个元素替换成新的字符串对象
    books.set(1, new String("LittleHann"));
    System.out.println(books);

    //将books集合的第二个元素（包括）
    //到第三个元素（不包括）截取成子集合
    System.out.println(books.subList(1, 2));
}


}
```

输出

[轻量级Java EE企业应用实战, 疯狂Java讲义, 疯狂Android讲义] 轻量级Java EE企业应用实战 疯狂Aja讲义 疯狂Java讲义 疯狂Android讲义 [轻量级Java EE企业应用实战, 疯狂Ajax讲义, 疯狂Android讲义] 1 [轻量级Java EE企业应用实战, LittleHann, 疯狂Android讲义] [LittleHann]

Stack

注意Stack的后进先出的特点

import java.util.*;

public class Test { public static void main(String[] args) { Stack v = new Stack(); //依次将三个素push入"栈" v.push("疯狂Java讲义"); v.push("轻量级Java EE企业应用实战"); v.push("疯狂Androi讲义");

```
    //输出：[疯狂Java讲义, 轻量级Java EE企业应用实战 , 疯狂Android讲义]
    System.out.println(v);
```

```
    //访问第一个元素，但并不将其pop出"栈"，输出：疯狂Android讲义
    System.out.println(v.peek());

    //依然输出：[疯狂Java讲义, 轻量级Java EE企业应用实战 , 疯狂Android讲义]
    System.out.println(v);

    //pop出第一个元素，输出：疯狂Android讲义
    System.out.println(v.pop());

    //输出：[疯狂Java讲义, 轻量级Java EE企业应用实战]
    System.out.println(v);
}


}
```

输出

[疯狂Java讲义, 轻量级Java EE企业应用实战, 疯狂Android讲义] 疯狂Android讲义 [疯狂Java讲义, 量级Java EE企业应用实战, 疯狂Android讲义] 疯狂Android讲义 [疯狂Java讲义, 轻量级Java EE企业应用实战]

LinkedList

```
import java.util.*;

public class Test { public static void main(String[] args) { LinkedList books = new LinkedList();

    //将字符串元素加入队列的尾部(双端队列)
    books.offer("疯狂Java讲义");

    //将一个字符串元素加入栈的顶部(双端队列)
    books.push("轻量级Java EE企业应用实战");

    //将字符串元素添加到队列的头(相当于栈的顶部)
    books.offerFirst("疯狂Android讲义");

    for (int i = 0; i < books.size() ; i++ )
    {
        System.out.println(books.get(i));
    }

    //访问、并不删除栈顶的元素
    System.out.println(books.peekFirst());
    //访问、并不删除队列的最后一个元素
    System.out.println(books.peekLast());
    //将栈顶的元素弹出"栈"
    System.out.println(books.pop());
    //下面输出将看到队列中第一个元素被删除
    System.out.println(books);
    //访问、并删除队列的最后一个元素
    System.out.println(books.pollLast());
    //下面输出将看到队列中只剩下中间一个元素：
    //轻量级Java EE企业应用实战
```

```
    System.out.println(books);
}


}
```

输出

疯狂Android讲义 轻量级Java EE企业应用实战 疯狂Java讲义 疯狂Android讲义 疯狂Java讲义 疯狂Android讲义 [轻量级Java EE企业应用实战, 疯狂Java讲义] 疯狂Java讲义 [轻量级Java EE企业应用实战]

Queue

```
import java.util.*;

public class PriorityQueueTest { public static void main(String[] args) { PriorityQueue pq = new PriorityQueue(); //下面代码依次向pq中加入四个元素 pq.offer(6); pq.offer(-3); pq.offer(9); pq.offe(0);

    //输出pq队列，并不是按元素的加入顺序排列，
    //而是按元素的大小顺序排列，输出[-3, 0, 9, 6]
    System.out.println(pq);
    //访问队列第一个元素，其实就是队列中最小的元素：-3
    System.out.println(pq.poll());
}


}
```

PriorityQueue不允许插入null元素，它还需要对队列元素进行排序，PriorityQueue的元素有两种排方式

1. 自然排序: 采用自然顺序的PriorityQueue集合中的元素对象都必须实现了Comparable接口，而且该是同一个类的多个实例，否则可能导致ClassCastException异常

2. 定制排序 创建PriorityQueue队列时，传入一个Comparator对象，该对象负责对队列中的所有元进行排序 关于自然排序、定制排序的原理和之前说的TreeSet类似

ArrayDeque

```
import java.util.*;

public class Test { public static void main(String[] args) { ArrayDeque stack = new ArrayDeque(; //依次将三个元素push入"栈" stack.push("疯狂Java讲义"); stack.push("轻量级Java EE企业应用实"); stack.push("疯狂Android讲义");

    //输出：[疯狂Java讲义, 轻量级Java EE企业应用实战 , 疯狂Android讲义]
    System.out.println(stack);

    //访问第一个元素，但并不将其pop出"栈"，输出：疯狂Android讲义
    System.out.println(stack.peek());

    //依然输出：[疯狂Java讲义, 轻量级Java EE企业应用实战 , 疯狂Android讲义]
    System.out.println(stack);
```

```
    //pop出第一个元素，输出：疯狂Android讲义
    System.out.println(stack.pop());

    //输出：[疯狂Java讲义, 轻量级Java EE企业应用实战]
    System.out.println(stack);
}


}
```

[疯狂Android讲义, 轻量级Java EE企业应用实战, 疯狂Java讲义] 疯狂Android讲义 [疯狂Android讲义, 轻量级Java EE企业应用实战, 疯狂Java讲义] 疯狂Android讲义 [轻量级Java EE企业应用实战, 疯狂Jav 讲义]

以上就是List集合类的编程应用场景。我们来梳理一下思路

java提供的List就是一个“线性表接口”，ArrayList（基于数组的线性表），LinkedList（基于链的性表）是线性表的两种典型实现

Queue代表了队列，Deque代表了双端队列（即可以作为队列使用，也可以作为栈使用）

因为数组以一块连续内存来保存所有的数组元素，所以数组在随机访问时性能最好。

内部以链表作为底层实现的集合在执行插入，删除操作时有很好的的性能

遍历

我们之前说过，Collection接口继承了Iterable接口，也就是说，我们以上学习到的所有的Collection合类都具有"可遍历性"

Iterable接口也是java集合框架的成员，它隐藏了各种Collection实现类的底层细节，向应用程序提供遍历Collection集合元素的统一编程接口:

1. boolean hasNext(): 是否还有下一个未遍历过的元素
2. Object next(): 返回集合里的下一个元素
3. void remove(): 删除集合里上一次next方法返回的元素 iteration

import java.util.*;

public class Test { public static void main(String[] args) { //创建一个集合 Collection books=new HashSet(); books.add("1"); books.add("2"); books.add("3"); //获取books集合对应的迭代器 Iterat r it=books.iterator(); while(it.hasNext()) { String book=(String)it.next(); System.out.println(book ; if (book.equals("2")) { //从集合中删除上一次next方法返回的元素 it.remove(); } //对book变量赋 ，不会改变集合元素本身 book = "测试字符串"; } System.out.println(books);

```
}


}
```

输出

3 2 1 [3, 1]

从代码可以看出，iterator必须依附于Collection对象，若有一个iterator对象，必然有一个与之关联Collection对象。

除了可以使用iterator接口迭代访问Collection集合里的元素之外，使用java5提供的foreach循环迭访问集合元素更加便捷

foreach 实现遍历

import java.util.*;

public class Test { public static void main(String[] args) { //创建一个集合 Collection books = ne HashSet(); books.add(new String("1")); books.add(new String("2")); books.add(new String("3")) ;

```
    for (Object obj : books)
    {
        //此处的book变量也不是集合元素本身
        String book = (String)obj;
        System.out.println(book);
        if (book.equals("2"))
        {
            //下面代码会引发ConcurrentModificationException异常
            //books.remove(book);
        }
    }
    System.out.println(books);
}


}
```

输出

3 2 1 [3, 2, 1]

Map

HashMap，HashTable

import java.util.*;

class A { int count; public A(int count) { this.count = count; } //根据count的值来判断两个对象是相等。 public boolean equals(Object obj) { if (obj == this) return true; if (obj!=null && obj.get lass()==A.class) { A a = (A)obj; return this.count == a.count; } return false; } //根据count来计算 ashCode值。 public int hashCode() { return this.count; } } class B { //重写equals()方法，B对象任何对象通过equals()方法比较都相等 public boolean equals(Object obj) { return true; } } public c ass Test { public static void main(String[] args) { Hashtable ht = new Hashtable(); ht.put(new A 60000) , "疯狂Java讲义"); ht.put(new A(87563) , "轻量级Java EE企业应用实战"); ht.put(new A(12 2) , new B()); System.out.println(ht);

```
    //只要两个对象通过equals比较返回true，
    //Hashtable就认为它们是相等的value。
    //由于Hashtable中有一个B对象，
    //它与任何对象通过equals比较都相等，所以下面输出true。
    System.out.println(ht.containsValue("测试字符串"));  //①
```

```
    //只要两个A对象的count相等，它们通过equals比较返回true，且hashCode相等
    //Hashtable即认为它们是相同的key，所以下面输出true。
    System.out.println(ht.containsKey(new A(87563)));   //②

    //下面语句可以删除最后一个key-value对
    ht.remove(new A(1232));    //③

    //通过返回Hashtable的所有key组成的Set集合，
    //从而遍历Hashtable每个key-value对
    for (Object key : ht.keySet())
    {
        System.out.print(key + "---->");
        System.out.print(ht.get(key) + "\n");
    }
}


}
```

输出

```
{A@ea60=疯狂Java讲义, A@1560b=轻量级Java EE企业应用实战, A@4d0=B@547c9586} true tr
e A@ea60---->疯狂Java讲义 A@1560b---->轻量级Java EE企业应用实战
```

当使用自定义类作为HashMap，Hashtable的key时，如果重写该类的equals（Object obj）和hash ode（）方法，则应该保证两个方法的判断标准一致-当两个key通过equals（）方法比较返回true时两个key的hashCode（）的返回值也应该相同。

LinkedHashMap

```
import java.util.*;

public class Test { public static void main(String[] args) { LinkedHashMap scores = new Linke
HashMap(); scores.put("语文" , 80); scores.put("英文" , 82); scores.put("数学" , 76); //遍历score
里的所有的key-value对 for (Object key : scores.keySet()) { System.out.println(key + "------>" +
cores.get(key)); } } }
```

输出

```
语文------>80 英文------>82 数学------>76
```

properties

```
import java.util.; import java.io.;

public class Test { public static void main(String[] args) throws Exception { Properties props =
ew Properties(); //向Properties中增加属性 props.setProperty("username" , "yeeku"); props.setP
operty("password" , "123456");

    //将Properties中的key-value对保存到a.ini文件中
    props.store(new FileOutputStream("a.ini"), "comment line");   //①

    //新建一个Properties对象
    Properties props2 = new Properties();
```

```
    //向Properties中增加属性
    props2.setProperty("gender" , "male");

    //将a.ini文件中的key-value对追加到props2中
    props2.load(new FileInputStream("a.ini") );    //②
    System.out.println(props2);
}


}
```

输出

```
{password=123456, gender=male, username=yeeku}
```

TreeMap

```
import java.util.*;

class R implements Comparable { int count; public R(int count) { this.count = count; } public St
ing toString() { return "R[count:" + count + "]"; } //根据count来判断两个对象是否相等。 public
oolean equals(Object obj) { if (this == obj) return true; if (obj!=null && obj.getClass()==R.clas
) { R r = (R)obj; return r.count == this.count; } return false; } //根据count属性值来判断两个对象
大小。 public int compareTo(Object obj) { R r = (R)obj; return count > r.count ? 1 : count < r.c
unt ? -1 : 0; } } public class TreeMapTest { public static void main(String[] args) { TreeMap tm =
new TreeMap(); tm.put(new R(3) , "轻量级Java EE企业应用实战"); tm.put(new R(-5) , "疯狂Java
义"); tm.put(new R(9) , "疯狂Android讲义");

    System.out.println(tm);

    //返回该TreeMap的第一个Entry对象
    System.out.println(tm.firstEntry());

    //返回该TreeMap的最后一个key值
    System.out.println(tm.lastKey());

    //返回该TreeMap的比new R(2)大的最小key值。
    System.out.println(tm.higherKey(new R(2)));

    //返回该TreeMap的比new R(2)小的最大的key-value对。
    System.out.println(tm.lowerEntry(new R(2)));

    //返回该TreeMap的子TreeMap
    System.out.println(tm.subMap(new R(-1) , new R(4)));
}


}
```

输出

```
{R[count:-5]=疯狂Java讲义, R[count:3]=轻量级Java EE企业应用实战, R[count:9]=疯狂Android讲义
 R[count:-5]=疯狂Java讲义 R[count:9] R[count:3] R[count:-5]=疯狂Java讲义 {R[count:3]=轻量级J
va EE企业应用实战}
```

从代码中可以看出，类似于TreeSet中判断两个元素是否相等的标准，TreeMap中判断两个key相等标准是

1. 两个key通过compareTo()方法返回0

2. equals()放回true

EnumMap

import java.util.*;

enum Season { SPRING,SUMMER,FALL,WINTER } public class Test { public static void main(Strig[] args) { //创建一个EnumMap对象，该EnumMap的所有key //必须是Season枚举类的枚举值 EnmMap enumMap = new EnumMap(Season.class); enumMap.put(Season.SUMMER , "夏日炎炎"; enumMap.put(Season.SPRING , "春暖花开"); System.out.println(enumMap); } }

输出

{SPRING=春暖花开, SUMMER=夏日炎炎}

与创建普通Map有所区别的是，创建EnumMap是必须指定一个枚举类，从而将该EnumMap和指定举类关联起来

以上就是Map集合类的编程小demo。我们来梳理一下思路

（1）HashMap和Hashtable的效率大致相同，因为它们的实现机制几乎完全一样。但HashMap通比Hashtable要快一点，因为Hashtable需要二外的线程同步控制

（2）TreeMap通常比HashMap，Hashtable要慢（尤其是在插入，删除key-value对要慢），因为TreeMap底层采用的

作者：Fysddsw_lc

链接：https://juejin.im/post/5a98b571518825555c1d0f8f

来源：掘金