# Java集合類別

2009年05月11日

[如果您喜歡這些文章，歡迎點擊此處訂閱本Blog](http://feed.feedsky.com/softwave)

## 集合類別說明及區別

Collection

├List

│├LinkedList

│├ArrayList

│└Vector

│└Stack

└Set

├HashSet

├TreeSet

Map

├Hashtable

├HashMap

└WeakHashMap

## Collection介面

單值保存的最大父介面，每次只能操作一個資料。

|  |  |
| --- | --- |
| 名稱 | 功能 |
| boolean [add](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#add(E))([E](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) e) | 增加一個物件 |
| boolean [addAll](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#addAll(java.util.Collection))([Collection](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<? extends [E](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)> c) | 增加傳入集合的所有物件 |
| void [clear](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#clear())() | 清除資料 |
| boolean [contains](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#contains(java.lang.Object))([Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) o) | 集合中是否包含特定物件 |
| boolean [containsAll](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#containsAll(java.util.Collection))([Collection](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<?> c) | 集合中是否包含傳入集合的所有物件 |
| boolean [equals](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#equals(java.lang.Object))([Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) o) | 比較 |
| int [hashCode](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#hashCode())() | 傳回集合的hash碼 |
| boolean [isEmpty](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#isEmpty())() | 判斷集合是否為空集合 |
| [Iterator](https://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html)<[E](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)> [iterator](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#iterator())() | 將所有集合轉成Iterator介面輸出 |
| boolean [remove](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#remove(java.lang.Object))([Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) o) | 移除一個物件 |
| boolean [removeAll](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#removeAll(java.util.Collection))([Collection](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<?> c) | 移除傳入集合的所有物件 |
| boolean [retainAll](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#retainAll(java.util.Collection))([Collection](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<?> c) | 只保留傳入集合所包含的物件 |
| int [size](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#size())() | 傳回集合的元件個數 |
| [Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)[] [toArray](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#toArray())() | 轉出到陣列 |
| <T> T[] [toArray](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#toArray(T[]))(T[] a) | 轉出到陣列 |

## List介面

允許保存重複的介面。

|  |  |
| --- | --- |
| 名稱 | 功能 |
| E get(int index) | 取出指定位址的資料 |
| List[Iterator](https://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html)<[E](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)> listI[iterator](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#iterator())() | 將ListIterator介面實體化 |

### LinkedList類別

### ArrayList類別

List介面中使用最多者。

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

List<String> all =new ArrayList<String>();

System.out.println("集合是否為空集合："+all.isEmpty());

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

all.add("World");

System.out.println("集合是否為空集合："+all.isEmpty());

System.out.println(all);

}

}

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

List<String> all =new ArrayList<String>();

System.out.println("集合是否為空集合："+all.isEmpty());

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

all.add("World");

System.out.println("集合是否為空集合："+all.isEmpty());

System.out.println(all);

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

System.out.println(all.get(i));

}

}

}

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

Collection<String> all =new ArrayList<String>();

//沒有get()方法

System.out.println("集合是否為空集合："+all.isEmpty());

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

all.add("World");

System.out.println("集合是否為空集合："+all.isEmpty());

System.out.println(all);

Object o[] = all.toArray();

for (int i=0; i<o.length; i++){

String s = (String) o[i];//向下強制轉換，有危險

System.out.println(s);

}

}

}

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

Collection<String> all =new ArrayList<String>();

//沒有get()方法

System.out.println("集合是否為空集合："+all.isEmpty());

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

all.add("World");

System.out.println("集合是否為空集合："+all.isEmpty());

System.out.println(all);

String s[] = all.toArray(new String[] {});

for (int i=0; i<o.length; i++){

String str = s[i];//不必向下強制轉換

System.out.println(str);

}

}

}

### Vector類別

List舊的實作類別。

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

List<String> all =new Vector<String>();

//沒有get()方法

System.out.println("集合是否為空集合："+all.isEmpty());

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

all.add("World");

System.out.println("集合是否為空集合："+all.isEmpty());

System.out.println(all);

}

}

Vector非常類似ArrayList，但是Vector是同步的。

|  |  |  |
| --- | --- | --- |
|  | ArrayList | Vector |
| 推出 | 晚 | 早 |
| 性能 | 異步處理，快 | 同步處理，慢 |
| 執行緒 | 非執行緒安全 | 執行緒安全 |
| 輸出 | Iterator, ListIterator, foreach | Iterator, ListIterator, foreach, Enumeration |

### Stack 類別

## Set介面

不允許保存重複的介面。

純實作Collection，沒有擴充。

### HashSet

不重複，無排序。

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

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

//沒有get()方法

System.out.println("集合是否為空集合："+all.isEmpty());

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

all.add("World");

System.out.println("集合是否為空集合："+all.isEmpty());

System.out.println(all);

}

}

### TreeSet

不重複，排序。

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

Set<String> all =new TreeSet<String>();

//沒有get()方法

System.out.println("集合是否為空集合："+all.isEmpty());

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

all.add("World");

System.out.println("集合是否為空集合："+all.isEmpty());

System.out.println(all);

}

}

自訂類別的排序須另訂。

## 集合輸出

### Iterator

由前向後檢視。

|  |  |
| --- | --- |
| 名稱 | 功能 |
| boolean hasNext() | 判斷是否還有資料 |
| E next() | 讀取資料 |

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

List<String> all =new ArrayList<String>();

all.add("Hello");

all.add("Hello");

all.add("World");

all.add("Hello");

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

While (iter.hasNext()){

String str = iter.next();

System.out.println(str);

}

}

}

### ListIterator

不能使用Collection介面，要用List介面。現在位置必須不在第一個元素。

package no.jinni.demo

import java.util.ArrayList;

import java.util.List;

public class CollectionDemo{

public static void main(String[] args){

List<String> all =new ArrayList<String>();

all.add("1");

all.add("2");

all.add("3");

all.add("4");

ListIterator<String> iter = all.listIterator();

While (iter.hasNext()){

String str = iter.next();

System.out.println(str);

}

While (iter.hasPrevious()){

String str = iter.previous();

System.out.println(str);

}

}

}

## Map介面

保存一對相關的物件，二個物件存在

key = value的型式。

先找key，再找到value。

|  |  |
| --- | --- |
| 名稱 | 功能 |
| V put(K key, V value) | 向集合中保存物件 |
| V get(Object key) | 依指定key取出對應的Value |
| Set<Map.Entry<K, V>> entrySet() | 將所有集合轉成Set集合 |
| Set<K> keySet() | 取得所有的key，按照Set返回 |
| int size() | 元素個數 |

### HashMap類別

package no.jinni.demo

import java.util.HashMap;

import java.util.Map;

public class CollectionDemo{

public static void main(String[] args){

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

map.put(1, "John");

map.put(null, null);

map.put(1, "Joe");

map.put(2, "Mary");

System.out.println(map);

System.out.println(map.get(2));

System.out.println(map.get(10));

}

}

package no.jinni.demo

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

import java.util.Iterator;

public class CollectionDemo{

public static void main(String[] args){

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

map.put(1, "John");

map.put(null, null);

map.put(1, "Joe");

map.put(2, "Mary");

Set<Integer> set = map.keySet();

Iterator<Integer> iter = set.iterator();

While (iter.hasNext){

Integer key = iter.next();

System.out.println(key);

System.out.println(map.get(key));

}

}

}

### Hashtable類別

舊的類別。

不能使用"null"。

## 集合的輸出

1. 使用Map介面的entrySet()將全部資料轉成Set集合。
2. 使用Set介面的iterator()取出Iterator介面的實體化物件，進行迭代輸出。
3. 透過Iterator介面的next()取出每一個Map.Entry物件，做為進行key和value分離之用。

package no.jinni.demo

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

import java.util.Iterator;

public class CollectionDemo{

public static void main(String[] args){

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

map.put(1, "John");

map.put(3, "Neo");

map.put(1, "Joe");

map.put(2, "Mary");

Set<Map.Entry<Integer, String>> set = map.entrySet();

Iterator< Map.Entry<Integer, String>> iter = set.iterator();

While (iter.hasNext){

Map.Entry<Integer, String> me = iter.next();

System.out.println(me.getKey()+"🡪"+me.getValue());

}

}

}

# [Java集合類別](http://www.cnblogs.com/LittleHann/p/3690187.html)

本文主要關注Java程式設計中涉及到的各種集合類別，以及它們的使用場景

## Java集合類別基本概念

在程式設計中，常常需要集中存放多個資料。從傳統意義上講，陣列是我們的一個很好的選擇，前提是我們事先已經明確知道我們將要保存的物件的數量。一旦在陣列初始化時指定了這個陣列長度，這個陣列長度就是不可變的，如果我們需要保存一個可以動態增長的資料(在編譯時無法確定具體的數量)，java的集合類別就是一個很好的設計方案了。

集合類別主要負責保存、盛裝其他資料，因此集合類別也被稱為容器類別。所以的集合類別都位於java.util包下，後來為了處理多執行緒環境下的併發安全問題，java5還在java.util.concurrent包下提供了一些多執行緒支援的集合類別。

在學習Java中的集合類別的API、程式設計原理的時候，我們一定要明白，"集合"是一個很古老的數學概念，它遠遠早於Java的出現。從數學概念的角度來理解集合能説明我們更好的理解程式設計中什麼時候該使用什麼型態的集合類別。

Java容器類別類別庫的用途是"保存物件"，並將其劃分為兩個不同的概念：

### Collection

一組"對立"的元素，通常這些元素都服從某種規則

1. List必須保持元素特定的順序
2. Set不能有重複元素
3. Queue保持一個佇列(先進先出)的順序

### Map

一組成對的"鍵值對"物件

Collection和Map的區別在於容器中每個位置保存的元素個數:

1. Collection 每個位置只能保存一個元素(物件)
2. Map保存的是"鍵值對"，就像一個小型資料庫。我們可以通過"鍵"找到該鍵對應的"值"

## Java集合類別架構層次關係

### Interface Iterable

反覆運算器介面，這是Collection類別的父介面。實現這個Iterable介面的物件允許使用foreach進行檢視，也就是說，所有的Collection集合物件都具有"foreach可檢視性"。這個Iterable介面只有一個方法: iterator()。它返回一個代表當前集合物件的泛型<T>反覆運算器，用於之後的檢視操作

### Collection

Collection是最基本的集合介面，一個Collection代表一組Object的集合，這些Object被稱作Collection的元素。Collection是一個介面，用以提供規範定義，不能被產生實體使用

#### Set

Set集合類似於一個罐子，"丟進"Set集合裡的多個物件之間沒有明顯的順序。Set繼承自Collection介面，不能包含有重複元素(記住，這是整個Set類別層次的共有屬性)。

Set判斷兩個物件相同不是使用"=="運算子，而是根據equals方法。也就是說，我們在加入一個新元素的時候，如果這個新元素物件和Set中已有物件進行注意equals比較都返回false，則Set就會接受這個新元素物件，否則拒絕。

因為Set的這個制約，在使用Set集合的時候，應該注意兩點：1) 為Set集合裡的元素的實現類別實現一個有效的equals(Object)方法、2) 對Set的構造函數，傳入的Collection參數不能包含重複的元素

##### HashSet

HashSet是Set介面的典型實現，HashSet使用HASH演算法來存儲集合中的元素，因此具有良好的存取和查找性能。當向HashSet集合中存入一個元素時，HashSet會呼叫該物件的hashCode()方法來得到該物件的hashCode值，然後根據該HashCode值決定該物件在HashSet中的存儲位置。

值得主要的是，HashSet集合判斷兩個元素相等的標準是兩個物件通過equals()方法比較相等，並且兩個物件的hashCode()方法的返回值相等

###### LinkedHashSet

LinkedHashSet集合也是根據元素的hashCode值來決定元素的存儲位置，但和HashSet不同的是，它同時使用鏈表維護元素的次序，這樣使得元素看起來是以插入的順序保存的。

當檢視LinkedHashSet集合裡的元素時，LinkedHashSet將會按元素的添加順序來存取集合裡的元素。

LinkedHashSet需要維護元素的插入順序，因此性能略低於HashSet的性能，但在反覆運算存取Set裡的全部元素時(檢視)將有很好的性能(鏈表很適合進行檢視)

##### SortedSet

此介面主要用於排序操作，即實現此介面的子類別都屬於排序的子類別

###### TreeSet

TreeSet是SortedSet介面的實現類別，TreeSet可以確保集合元素處於排序狀態

##### EnumSet

EnumSet是一個專門為枚舉類別設計的集合類別，EnumSet中所有元素都必須是指定枚舉型態的枚舉值，該枚舉型態在創建EnumSet時顯式、或隱式地指定。EnumSet的集合元素也是有序的，

它們以枚舉值在Enum類別內的定義順序來決定集合元素的順序

#### List

List集合代表一個元素有序、可重複的集合，集合中每個元素都有其對應的順序索引。List集合允許加入重複元素，因為它可以通過索引來存取指定位置的集合元素。List集合預設按元素

的添加順序設定元素的索引

##### ArrayList

ArrayList是基於陣列實現的List類別，它封裝了一個動態的增長的、允許再分配的Object[]陣列。

##### Vector

Vector和ArrayList在用法上幾乎完全相同，但由於Vector是一個古老的集合，所以Vector提供了一些方法名很長的方法，但隨著JDK1.2以後，java提供了系統的集合框架，就將

Vector改為實現List介面，統一歸入集合框架體系中

##### Stack

Stack是Vector提供的一個子類別，用於類比"堆疊"這種資料結構(LIFO後進先出)

##### LinkedList

implements List<E>, Deque<E>。實現List介面，能對它進行佇列操作，即可以根據索引來隨機存取集合中的元素。同時它還實現Deque介面，即能將LinkedList當作雙端佇列

使用。自然也可以被當作"堆疊來使用"

#### Queue

Queue用於類別比"佇列"這種資料結構(先進先出 FIFO)。佇列的頭部保存著佇列中存放時間最長的元素，佇列的尾部保存著佇列中存放時間最短的元素。新元素插入(offer)到佇列的尾部，

存取元素(poll)操作會返回佇列頭部的元素，佇列不允許隨機存取佇列中的元素。結合生活中常見的排隊就會很好理解這個概念

##### PriorityQueue

PriorityQueue並不是一個比較標準的佇列實現，PriorityQueue保存佇列元素的順序並不是按照加入佇列的順序，而是按照佇列元素的大小進行重新排序，這點從它的類別名也可以

看出來

##### Deque

Deque介面代表一個"雙端佇列"，雙端佇列可以同時從兩端來添加、刪除元素，因此Deque的實現類別既可以當成佇列使用、也可以當成堆疊使用

###### ArrayDeque

是一個基於陣列的雙端佇列，和ArrayList類似，它們的底層都採用一個動態的、可重分配的Object[]陣列來存儲集合元素，當集合元素超出該陣列的容量時，系統會在底層重

新分配一個Object[]陣列來存儲集合元素

###### LinkedList

#### Map

Map用於保存具有"映射關係"的資料，因此Map集合裡保存著兩組值，一組值用於保存Map裡的key，另外一組值用於保存Map裡的value。key和value都可以是任何參考型態的資料。Map的key不允

許重複，即同一個Map物件的任何兩個key通過equals方法比較結果總是返回false。

關於Map，我們要從程式碼複用的角度去理解，java是先實現了Map，然後通過包裝了一個所有value都為null的Map就實現了Set集合

Map的這些實現類別和子介面中key集的存儲形式和Set集合完全相同(即key不能重複)

Map的這些實現類別和子介面中value集的存儲形式和List非常類似(即value可以重複、根據索引來查找)

##### HashMap

和HashSet集合不能保證元素的順序一樣，HashMap也不能保證key-value對的順序。並且類似於HashSet判斷兩個key是否相等的標準也是: 兩個key通過equals()方法比較返回true、同時兩個key的hashCode值也必須相等

##### LinkedHashMap

LinkedHashMap也使用雙向鏈表來維護key-value對的次序，該鏈表負責維護Map的反覆運算順序，與key-value對的插入順序一致(注意和TreeMap對所有的key-value進行排序進行區分)

##### Hashtable

是一個古老的Map實現類別

###### Properties

Properties物件在處理屬性檔時特別方便(windows平臺上的.ini檔)，Properties類別可以把Map物件和屬性檔關聯起來，從而可以把Map物件中的key-value對寫入到屬性文

件中，也可以把屬性檔中的"屬性名-屬性值"載入到Map物件中

##### SortedMap

正如Set介面衍生出SortedSet子介面，SortedSet介面有一個TreeSet實現類別一樣，Map介面也衍生出一個SortedMap子介面，SortedMap介面也有一個TreeMap實現類別

##### TreeMap

TreeMap就是一個紅黑樹資料結構，每個key-value對即作為紅黑樹的一個節點。TreeMap存儲key-value對(節點)時，需要根據key對節點進行排序。TreeMap可以保證所有的

key-value對處於有序狀態。同樣，TreeMap也有兩種排序方式: 自然排序、定制排序

##### WeakHashMap

WeakHashMap與HashMap的用法基本相似。區別在於，HashMap的key保留了對實際物件的"強引用"，這意味著只要該HashMap物件不被銷毀，該HashMap所引用的物件就不會被垃圾回收。

但WeakHashMap的key只保留了對實際物件的弱引用，這意味著如果WeakHashMap物件的key所引用的物件沒有被其他強引用變數所引用，則這些key所引用的物件可能被垃圾回收，當垃

圾回收了該key所對應的實際物件之後，WeakHashMap也可能自動刪除這些key所對應的key-value對

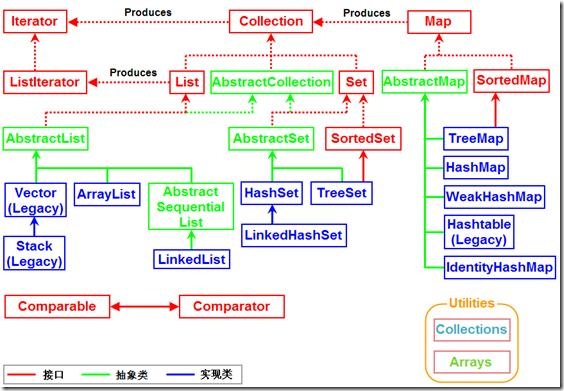
##### IdentityHashMap

IdentityHashMap的實現機制與HashMap基本相似，在IdentityHashMap中，當且僅當兩個key嚴格相等(key1 == key2)時，IdentityHashMap才認為兩個key相等

##### EnumMap

EnumMap是一個與枚舉類別一起使用的Map實現，EnumMap中的所有key都必須是單個枚舉類別的枚舉值。創建EnumMap時必須顯式或隱式指定它對應的枚舉類別。EnumMap根據key的自然順序

(即枚舉值在枚舉類別中的定義順序)



## Java集合類別的應用場景程式碼

學習了集合類別的基本架構框架之後，我們接著來學習它們各自的應用場景、以及細節處的注意事項

### 0x1: Set

#### HashSet

import java.util.\*;

//類別A的equals方法總是返回true,但沒有重寫其hashCode()方法。不能保證當前物件是HashSet中的唯一物件

class A {

public boolean equals(Object obj) {

return true;

}

}

// 類別B的hashCode()方法總是返回1,但沒有重寫其equals()方法。不能保證當前物件是HashSet中的唯一物件

class B {

public int hashCode() {

return 1;

}

}

// 類別C的hashCode()方法總是返回2,且有重寫其equals()方法

class C {

public int hashCode() {

return 2;

}

public boolean equals(Object obj) {

return true;

}

}

public class HashSetTest {

public static void main(String[] args) {

HashSet books = new HashSet();

// 分別向books集合中添加兩個A物件，兩個B物件，兩個C物件

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);

}

}

result:

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

可以看到，如果兩個物件通過equals()方法比較返回true，但這兩個物件的hashCode()方法返回不同的hashCode值時，這將導致HashSet會把這兩個物件保存在Hash表的不同位置，從而使物件可以添加成功，這就與Set集合的規則有些出入了。所以，我們要明確的是: equals()決定是否可以加入HashSet、而hashCode()決定存放的位置，它們兩者必須同時滿足才能允許一個新元素加入HashSet

但是要注意的是: 如果兩個物件的hashCode相同，但是它們的equlas返回值不同，HashSet會在這個位置用鏈式結構來保存多個物件。而HashSet存取集合元素時也是根據元素的HashCode值來快速定位的，這種鏈式結構會導致性能下降。

所以如果需要把某個類別的物件保存到HashSet集合中，我們在重寫這個類別的equlas()方法和hashCode()方法時，應該儘量保證兩個物件通過equals()方法比較返回true時，它們的hashCode()方法返回值也相等

#### LinkedHashSet

import java.util.\*;

public class LinkedHashSetTest {

public static void main(String[] args) {

LinkedHashSet books = new LinkedHashSet();

books.add("Java");

books.add("LittleHann");

System.out.println(books);

// 刪除 Java

books.remove("Java");

// 重新添加 Java

books.add("Java");

System.out.println(books);

}

}

元素的順序總是與添加順序一致，同時要明白的是，LinkedHashSetTest是HashSet的子類別，因此它不允許集合元素重複

#### TreeSet

import java.util.\*;

public class TreeSetTest {

public static void main(String[] args) {

TreeSet nums = new TreeSet();

// 向TreeSet中添加四個Integer物件

nums.add(5);

nums.add(2);

nums.add(10);

nums.add(-9);

// 輸出集合元素，看到集合元素已經處於排序狀態

System.out.println(nums);

// 輸出集合裡的第一個元素

System.out.println(nums.first());

// 輸出集合裡的最後一個元素

System.out.println(nums.last());

// 返回小於4的子集，不包含4

System.out.println(nums.headSet(4));

// 返回大於5的子集，如果Set中包含5，子集中還包含5

System.out.println(nums.tailSet(5));

// 返回大於等於-3，小於4的子集。

System.out.println(nums.subSet(-3, 4));

}

}

與HashSet集合採用hash演算法來決定元素的存儲位置不同，TreeSet採用紅黑樹的資料結構來存儲集合元素。TreeSet支援兩種排序方式: 自然排序、定制排序

### 自然排序:

TreeSet會呼叫集合元素的compareTo(Object obj)方法來比較元素之間的大小關係，然後將集合元素按昇冪排序，即自然排序。如果試圖把一個物件添加到TreeSet時，則該物件的類別必須實現Comparable介面，否則程式會拋出異常。

當把一個物件加入TreeSet集合中時，TreeSet會呼叫該物件的compareTo(Object obj)方法與容器中的其他物件比較大小，然後根據紅黑樹結構找到它的存儲位置。如果兩個物件通過compareTo(Object obj)方法比較相等，新物件將無法添加到TreeSet集合中(牢記Set是不允許重複的概念)。

注意: 當需要把一個物件放入TreeSet中，重寫該物件對應類別的equals()方法時，應該保證該方法與compareTo(Object obj)方法有一致的結果，即如果兩個物件通過equals()方法比較返回true時，這兩個物件通過compareTo(Object obj)方法比較結果應該也為0(即相等)

看到這裡，我們應該明白：

1. 對與Set來說，它定義了equals()為唯一性判斷的標準，而對於到了具體的實現，HashSet、TreeSet來說，它們又會有自己特有的唯一性判斷標準，只有同時滿足了才能判定為唯一性
2. 我們在操作這些集合類別的時候，對和唯一性判斷有關的函數重寫要重點關注

### 定制排序

TreeSet的自然排序是根據集合元素的大小，TreeSet將它們以昇冪排序。如果我們需要實現定制排序，則可以通過Comparator介面的説明(類似PHP中的array\_map回檔處理函數的思想)。該介面裡包含一個int compare(T o1， T o2)方法，該方法用於比較大小

import java.util.\*;

class M {

int age;

public M(int age) {

this.age = age;

}

public String toString() {

return "M[age:" + age + "]";

}

}

public class TreeSetTest4 {

public static void main(String[] args) {

TreeSet ts = new TreeSet(new Comparator() {

// 根據M物件的age屬性來決定大小

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(5));

ts.add(new M(-3));

ts.add(new M(9));

System.out.println(ts);

}

}

看到這裡，我們需要梳理一下關於排序的概念

1. equals、compareTo決定的是怎麼比的問題，即用什麼field進行大小比較
2. 自然排序、定制排序、Comparator決定的是誰大的問題，即按什麼順序(昇冪、降冪)進行排序

它們的關注點是不同的，一定要注意區分

EnumSet

import java.util.\*;

enum Season{

SPRING,SUMMER,FALL,WINTER

}

public class EnumSetTest{

public static void main(String[] args){

//創建一個EnumSet集合，集合元素就是Season枚舉類別的全部枚舉值

EnumSet es1 = EnumSet.allOf(Season.class);

//輸出[SPRING,SUMMER,FALL,WINTER]

System.out.println(es1);

//創建一個EnumSet空集合，指定其集合元素是Season類別的枚舉值。

EnumSet es2 = EnumSet.noneOf(Season.class);

//輸出[]

System.out.println(es2);

//手動添加兩個元素

es2.add(Season.WINTER);

es2.add(Season.SPRING);

//輸出[SPRING,WINTER]

System.out.println(es2);

//以指定枚舉值創建EnumSet集合

EnumSet es3 = EnumSet.of(Season.SUMMER , Season.WINTER);

//輸出[SUMMER,WINTER]

System.out.println(es3);

EnumSet es4 = EnumSet.range(Season.SUMMER , Season.WINTER);

//輸出[SUMMER,FALL,WINTER]

System.out.println(es4);

//新創建的EnumSet集合的元素和es4集合的元素有相同型態，

//es5的集合元素 + es4集合元素 = Season枚舉類別的全部枚舉值

EnumSet es5 = EnumSet.complementOf(es4);

//輸出[SPRING]

System.out.println(es5);

}

}

以上就是Set集合類別的程式設計應用場景。那麼應該怎樣選擇何時使用這些集合類別呢？

1. HashSet的性能總是比TreeSet好(特別是最常用的添加、查詢元素等操作)，因為TreeSet需要額外的紅黑樹演算法來維護集合元素的次序。只有當需要一個保持排序的Set時，才應該使用TreeSet，否則都應該使用HashSet
2. 對於普通的插入、刪除操作，LinkedHashSet比HashSet要略慢一點，這是由維護鏈表所帶來的開銷造成的。不過，因為有了鏈表的存在，檢視LinkedHashSet會更快
3. EnumSet是所有Set實現類別中性能最好的，但它只能保存同一個枚舉類別的枚舉值作為集合元素
4. HashSet、TreeSet、EnumSet都是"執行緒不安全"的，通常可以通過Collections工具類別的synchronizedSortedSet方法來"包裝"該Set集合。

SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));

## 0x2: List

ArrayList

如果一開始就知道ArrayList集合需要保存多少元素，則可以在創建它們時就指定initialCapacity大小，這樣可以減少重新分配的次數，提供性能，ArrayList還提供了如下方法來重新分配Object[]陣列

ensureCapacity(int minCapacity): 將ArrayList集合的Object[]陣列長度增加minCapacity

trimToSize(): 調整ArrayList集合的Object[]陣列長度為當前元素的個數。程式可以通過此方法來減少ArrayList集合物件佔用的記憶體空間

import java.util.\*;

public class ListTest {

public static void main(String[] args){

List books = new ArrayList();

//向books集合中添加三個元素

books.add(new String("羽量級Java EE企業應用實戰"));

books.add(new String("瘋狂Java講義"));

books.add(new String("瘋狂Android講義"));

System.out.println(books);

//將新字串物件插入在第二個位置

books.add(1 , new String("瘋狂Ajax講義"));

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

System.out.println(books.get(i));

}

//刪除第三個元素

books.remove(2);

System.out.println(books);

//判斷指定元素在List集合中位置：輸出1，表明位於第二位

System.out.println(books.indexOf(new String("瘋狂Ajax講義")));//①

//將第二個元素替換成新的字串物件

books.set(1, new String("LittleHann"));

System.out.println(books);

//將books集合的第二個元素(包括)

//到第三個元素(不包括)截取成子集合

System.out.println(books.subList(1 , 2));

}

Stack

注意Stack的後進先出的特點

import java.util.\*;

public class VectorTest{

public static void main(String[] args){

Stack v = new Stack();

//依次將三個元素push入"堆疊"

v.push("瘋狂Java講義");

v.push("羽量級Java EE企業應用實戰");

v.push("瘋狂Android講義");

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(v);

//存取第一個元素，但並不將其pop出"堆疊"，輸出：瘋狂Android講義

System.out.println(v.peek());

//依然輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(v);

//pop出第一個元素，輸出：瘋狂Android講義

System.out.println(v.pop());

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰]

System.out.println(v);

}

}

LinkedList

import java.util.\*;

public class LinkedListTest{

public static void main(String[] args){

LinkedList books = new LinkedList();

//將字串元素加入佇列的尾部(雙端佇列)

books.offer("瘋狂Java講義");

//將一個字串元素加入堆疊的頂部(雙端佇列)

books.push("羽量級Java EE企業應用實戰");

//將字串元素添加到佇列的頭(相當於堆疊的頂部)

books.offerFirst("瘋狂Android講義");

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

System.out.println(books.get(i));

}

//存取、並不刪除堆疊頂的元素

System.out.println(books.peekFirst());

//存取、並不刪除佇列的最後一個元素

System.out.println(books.peekLast());

//將堆疊頂的元素彈出"堆疊"

System.out.println(books.pop());

//下面輸出將看到佇列中第一個元素被刪除

System.out.println(books);

//存取、並刪除佇列的最後一個元素

System.out.println(books.pollLast());

//下面輸出將看到佇列中只剩下中間一個元素：

//羽量級Java EE企業應用實戰

System.out.println(books);

}

}

從程式碼中我們可以看到，LinkedList同時表現出了雙端佇列、堆疊的用法。功能非常強大

## 0x3: Queue

PriorityQueue

import java.util.\*;

public class PriorityQueueTest{

public static void main(String[] args){

PriorityQueue pq = new PriorityQueue();

//下面程式碼依次向pq中加入四個元素

pq.offer(6);

pq.offer(-3);

pq.offer(9);

pq.offer(0);

//輸出pq佇列，並不是按元素的加入順序排列，

//而是按元素的大小順序排列，輸出[-3, 0, 9, 6]

System.out.println(pq);

//存取佇列第一個元素，其實就是佇列中最小的元素：-3

System.out.println(pq.poll());

}

}

PriorityQueue不允許插入null元素，它還需要對佇列元素進行排序，PriorityQueue的元素有兩種排序方式

1) 自然排序:

採用自然順序的PriorityQueue集合中的元素物件都必須實現了Comparable介面，而且應該是同一個類別的多個實例，否則可能導致ClassCastException異常

2) 定制排序

創建PriorityQueue佇列時，傳入一個Comparator物件，該物件負責對佇列中的所有元素進行排序

關於自然排序、定制排序的原理和之前說的TreeSet類似

 ArrayDeque

import java.util.\*;

public class ArrayDequeTest{

public static void main(String[] args){

ArrayDeque stack = new ArrayDeque();

//依次將三個元素push入"堆疊"

stack.push("瘋狂Java講義");

stack.push("羽量級Java EE企業應用實戰");

stack.push("瘋狂Android講義");

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(stack);

//存取第一個元素，但並不將其pop出"堆疊"，輸出：瘋狂Android講義

System.out.println(stack.peek());

//依然輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(stack);

//pop出第一個元素，輸出：瘋狂Android講義

System.out.println(stack.pop());

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰]

System.out.println(stack);

}

}

以上就是List集合類別的程式設計應用場景。我們來梳理一下思路

1. java提供的List就是一個"線性表介面"，ArrayList(基於陣列的線性表)、LinkedList(基於鏈的線性表)是線性表的兩種典型實現

2. Queue代表了佇列，Deque代表了雙端佇列(既可以作為佇列使用、也可以作為堆疊使用)

3. 因為陣列以一塊連續記憶體來保存所有的陣列元素，所以陣列在隨機存取時性能最好。所以的內部以陣列作為底層實現的集合在隨機存取時性能最好。

4. 內部以鏈表作為底層實現的集合在執行插入、刪除操作時有很好的性能

5. 進行反覆運算操作時，以鏈表作為底層實現的集合比以陣列作為底層實現的集合性能好

我們之前說過，Collection介面繼承了Iterable介面，也就是說，我們以上學習到的所有的Collection集合類別都具有"可檢視性"

Iterable介面也是java集合框架的成員，它隱藏了各種Collection實現類別的底層細節，向應用程式提供了檢視Collection集合元素的統一程式設計介面:

1) boolean hasNext(): 是否還有下一個未檢視過的元素

2) Object next(): 返回集合裡的下一個元素

3) void remove(): 刪除集合裡上一次next方法返回的元素

iterator實現檢視:

import java.util.\*;

public class IteratorTest{

public static void main(String[] args){

//創建一個集合

Collection books = new HashSet();

books.add("羽量級Java EE企業應用實戰");

books.add("瘋狂Java講義");

books.add("瘋狂Android講義");

//獲取books集合對應的反覆運算器

Iterator it = books.iterator();

while(it.hasNext()){

//it.next()方法返回的資料型態是Object型態，

//需要強制型態轉換

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

System.out.println(book);

if (book.equals("瘋狂Java講義")){

//從集合中刪除上一次next方法返回的元素

it.remove();

}

//對book變數賦值，不會改變集合元素本身

book = "測試字串";

}

System.out.println(books);

}

}

從程式碼可以看出，iterator必須依附於Collection物件，若有一個iterator物件，必然有一個與之關聯的Collection物件。

除了可以使用iterator介面反覆運算存取Collection集合裡的元素之外，使用java5提供的foreach迴圈反覆運算存取集合元素更加便捷

foreach實現檢視:

import java.util.\*;

public class ForeachTest{

public static void main(String[] args){

//創建一個集合

Collection books = new HashSet();

books.add(new String("羽量級Java EE企業應用實戰"));

books.add(new String("瘋狂Java講義"));

books.add(new String("瘋狂Android講義"));

for (Object obj : books){

//此處的book變數也不是集合元素本身

String book = (String)obj;

System.out.println(book);

if (book.equals("瘋狂Android講義")){

//下面程式碼會引發ConcurrentModificationException異常

//books.remove(book);

}

}

System.out.println(books);

}

}

除了Collection固有的iterator()方法，List還額外提供了一個listIterator()方法，該方法返回一個ListIterator物件，ListIterator介面繼承了Iterator介面，提供了專門操作List的方法。ListIterator介面在Iterator介面的繼承上增加了如下方法:

1) boolean hasPrevious(): 返回該反覆運算器關聯的集合是否還有上一個元素

2) Object previous(): 返回該反覆運算器的上一個元素(向前反覆運算)

3) void add(): 在指定位置插入一個元素

ListIterator實現檢視:

import java.util.\*;

public class ListIteratorTest{

public static void main(String[] args){

String[] books ={"瘋狂Java講義","羽量級Java EE企業應用實戰"};

List bookList = new ArrayList();

for (int i = 0; i < books.length; i++){

bookList.add(books[i]);

}

ListIterator lit = bookList.listIterator();

while (lit.hasNext()){

System.out.println(lit.next());

lit.add("-------分隔符號-------");

}

System.out.println("=======下面開始反向反覆運算=======");

while(lit.hasPrevious()){

System.out.println(lit.previous());

}

}

}

0x4: Map

HashMap、Hashtable

import java.util.\*;

class A{

int count;

public A(int count){

this.count = count;

}

//根據count的值來判斷兩個物件是否相等。

public boolean equals(Object obj){

if (obj == this) return true;

if (obj!=null && obj.getClass()==A.class){

A a = (A)obj;

return this.count == a.count;

}

return false;

}

//根據count來計算hashCode值。

public int hashCode(){

return this.count;

}

}

class B{

//重寫equals()方法，B物件與任何物件通過equals()方法比較都相等

public boolean equals(Object obj){

return true;

}

}

public class HashtableTest{

public static void main(String[] args){

Hashtable ht = new Hashtable();

ht.put(new A(60000) , "瘋狂Java講義");

ht.put(new A(87563) , "羽量級Java EE企業應用實戰");

ht.put(new A(1232) , new B());

System.out.println(ht);

//只要兩個物件通過equals比較返回true，

//Hashtable就認為它們是相等的value。

//由於Hashtable中有一個B物件，

//它與任何物件通過equals比較都相等，所以下面輸出true。

System.out.println(ht.containsValue("測試字串"));//①

//只要兩個A物件的count相等，它們通過equals比較返回true，且hashCode相等

//Hashtable即認為它們是相同的key，所以下面輸出true。

System.out.println(ht.containsKey(new A(87563)));//②

//下面語句可以刪除最後一個key-value對

ht.remove(new A(1232));//③

//通過返回Hashtable的所有key組成的Set集合，

//從而檢視Hashtable每個key-value對

for (Object key : ht.keySet()){

System.out.print(key + "---->");

System.out.print(ht.get(key) + "\n");

}

}

}

當使用自訂類別作為HashMap、Hashtable的key時，如果重寫該類別的equals(Object obj)和hashCode()方法，則應該保證兩個方法的判斷標準一致--當兩個key通過equals()方法比較返回true時，兩個key的hashCode()的返回值也應該相同

LinkedHashMap

import java.util.\*;

public class LinkedHashMapTest{

public static void main(String[] args){

LinkedHashMap scores = new LinkedHashMap();

scores.put("語文" , 80);

scores.put("英文" , 82);

scores.put("數學" , 76);

//檢視scores裡的所有的key-value對

for (Object key : scores.keySet()){

System.out.println(key + "------>" + scores.get(key));

}

}

}

Properties

import java.util.\*;

import java.io.\*;

public class PropertiesTest{

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

Properties props = new Properties();

//向Properties中增加屬性

props.setProperty("username" , "yeeku");

props.setProperty("password" , "123456");

//將Properties中的key-value對保存到a.ini文件中

props.store(new FileOutputStream("a.ini"), "comment line");//①

//新建一個Properties物件

Properties props2 = new Properties();

//向Properties中增加屬性

props2.setProperty("gender" , "male");

//將a.ini文件中的key-value對追加到props2中

props2.load(new FileInputStream("a.ini"));//②

System.out.println(props2);

}

}

Properties還可以把key-value對以XML檔的形式保存起來，也可以從XML檔中載入key-value對

TreeMap

import java.util.\*;

class R implements Comparable{

int count;

public R(int count){

this.count = count;

}

public String toString(){

return "R[count:" + count + "]";

}

//根據count來判斷兩個物件是否相等。

public boolean equals(Object obj){

if (this == obj)

return true;

if (obj!=null && obj.getClass()==R.class){

R r = (R)obj;

return r.count == this.count;

}

return false;

}

//根據count屬性值來判斷兩個物件的大小。

public int compareTo(Object obj){

R r = (R)obj;

return count > r.count ? 1 :

count < r.count ? -1 : 0;

}

}

public class TreeMapTest{

public static void main(String[] args){

TreeMap tm = new TreeMap();

tm.put(new R(3) , "羽量級Java EE企業應用實戰");

tm.put(new R(-5) , "瘋狂Java講義");

tm.put(new R(9) , "瘋狂Android講義");

System.out.println(tm);

//返回該TreeMap的第一個Entry物件

System.out.println(tm.firstEntry());

//返回該TreeMap的最後一個key值

System.out.println(tm.lastKey());

//返回該TreeMap的比new R(2)大的最小key值。

System.out.println(tm.higherKey(new R(2)));

//返回該TreeMap的比new R(2)小的最大的key-value對。

System.out.println(tm.lowerEntry(new R(2)));

//返回該TreeMap的子TreeMap

System.out.println(tm.subMap(new R(-1) , new R(4)));

}

}

從程式碼中可以看出，類似於TreeSet中判斷兩個元素是否相等的標準，TreeMap中判斷兩個key相等的標準是:

1) 兩個key通過compareTo()方法返回0

2) equals()放回true

我們在重寫這兩個方法的時候一定要保證它們的邏輯關係一致。

再次強調一下:

Set和Map的關係十分密切，java原始碼就是先實現了HashMap、TreeMap等集合，然後通過包裝一個所有的value都為null的Map集合實現了Set集合類別

WeakHashMap

import java.util.\*;

public class WeakHashMapTest{

public static void main(String[] args){

WeakHashMap whm = new WeakHashMap();

//將WeakHashMap中添加三個key-value對，

//三個key都是匿名字串物件(沒有其他引用)

whm.put(new String("語文") , new String("良好"));

whm.put(new String("數學") , new String("及格"));

whm.put(new String("英文") , new String("中等"));

//將WeakHashMap中添加一個key-value對，

//該key是一個系統緩存的字串物件。"java"是一個常量字串強引用

whm.put("java" , new String("中等"));

//輸出whm物件，將看到4個key-value對。

System.out.println(whm);

//通知系統立即進行垃圾回收

System.gc();

System.runFinalization();

//通常情況下，將只看到一個key-value對。

System.out.println(whm);

}

}

如果需要使用WeakHashMap的key來保留物件的弱引用，則不要讓key所引用的物件具有任何強引用，否則將失去使用WeakHashMap的意義

IdentityHashMap

import java.util.\*;

public class IdentityHashMapTest{

public static void main(String[] args){

IdentityHashMap ihm = new IdentityHashMap();

//下面兩行程式碼將會向IdentityHashMap物件中添加兩個key-value對

ihm.put(new String("語文") , 89);

ihm.put(new String("語文") , 78);

//下面兩行程式碼只會向IdentityHashMap物件中添加一個key-value對

ihm.put("java" , 93);

ihm.put("java" , 98);

System.out.println(ihm);

}

}

EnumMap

import java.util.\*;

enum Season{

SPRING,SUMMER,FALL,WINTER

}

public class EnumMapTest{

public static void main(String[] args){

//創建一個EnumMap物件，該EnumMap的所有key

//必須是Season枚舉類別的枚舉值

EnumMap enumMap = new EnumMap(Season.class);

enumMap.put(Season.SUMMER , "夏日炎炎");

enumMap.put(Season.SPRING , "春暖花開");

System.out.println(enumMap);

}

}

與創建普通Map有所區別的是，創建EnumMap是必須指定一個枚舉類別，從而將該EnumMap和指定枚舉類別關聯起來

以上就是Map集合類別的程式設計應用場景。我們來梳理一下思路

1) HashMap和Hashtable的效率大致相同，因為它們的實現機制幾乎完全一樣。但HashMap通常比Hashtable要快一點，因為Hashtable需要額外的執行緒同步控制

2) TreeMap通常比HashMap、Hashtable要慢(尤其是在插入、刪除key-value對時更慢)，因為TreeMap底層採用紅黑樹來管理key-value對

3) 使用TreeMap的一個好處就是： TreeMap中的key-value對總是處於有序狀態，無須專門進行排序操作

# Java集合類別

2009年05月11日

[如果您喜歡這些文章，歡迎點擊此處訂閱本Blog](http://feed.feedsky.com/softwave)

## 集合類別說明及區別

Collection

├List

│├LinkedList

│├ArrayList

│└Vector

│└Stack

└Set

├HashSet

├TreeSet

Map

├Hashtable

├HashMap

└WeakHashMap

## Collection介面

Collection是最基本的集合介面，一個Collection代表一組Object，即Collection的元素(Elements)。一些Collection允許相同的元素而一些不行。一些能排序而一些不行。Java SDK不提供直接繼承自Collection的類別，Java SDK提供的類別都是繼承自Collection的"子介面"如List和Set。

所有實現Collection介面的類別都必須提供兩個標準的構造函數：無參數的構造函數用於創建一個空的Collection，有一個 Collection參數的構造函數用於創建一個新的Collection，這個新的Collection與傳入的Collection有相同的元素。後 一個構造函數允許使用者複製一個Collection。

如何檢視Collection中的每一個元素？不論Collection的實際型態如何，它都支援一個iterator()的方法，該方法返回一個反覆運運算元，使用該反覆運運算元即可逐一存取Collection中每一個元素。典型的用法如下：

Iterator it = collection.iterator();//獲得一個反覆運算元

while(it.hasNext()){

Object obj = it.next();//得到下一個元素

}

由Collection介面衍生的兩個介面是List和Set。

## List介面

List是有序的Collection，使用此介面能夠精確的控制每個元素插入的位置。使用者能夠使用索引(元素在List中的位置，類似於陣列下標)來存取List中的元素，這類似於Java的陣列。

和下面要提到的Set不同，List允許有相同的元素。

除了具有Collection介面必備的iterator()方法外，List還提供一個listIterator()方法，返回一個 ListIterator介面，和標準的Iterator介面相比，ListIterator多了一些add()之類的方法，允許添加，刪除，設定元素， 還能向前或向後掃視。

實現List介面的常用類別有LinkedList, ArrayList, Vector和Stack。

### LinkedList類別

LinkedList實現List介面，允許null元素。此外LinkedList提供額外的get，remove，insert方法在 LinkedList的首部或尾部。這些操作使LinkedList可被用作堆疊(stack)，佇列(queue)或雙向佇列(deque)。

注意LinkedList沒有同步方法。如果多個執行緒同時存取一個List，則必須自己實現存取同步。一種解決方法是在創建List時構造一個同步的List：

List list = Collections.synchronizedList(new LinkedList(...));

### ArrayList類別

ArrayList實現了可變大小的陣列。它允許所有元素，包括null。ArrayList沒有同步。

size，isEmpty，get，set方法執行時間為常數。但是add方法開銷為分攤的常數，添加n個元素需要O(n)的時間。其他的方法執行時間為線性。

每個ArrayList實例都有一個容量(Capacity)，即用於存儲元素的陣列的大小。這個容量可隨著不斷添加新元素而自動增加，但是增長演算法 並沒有定義。當需要插入大量元素時，在插入前可以呼叫ensureCapacity方法來增加ArrayList的容量以提高插入效率。

和LinkedList一樣，ArrayList也是非同步的(unsynchronized)。

### Vector類別

Vector非常類似ArrayList，但是Vector是同步的。由Vector創建的Iterator，雖然和 ArrayList創建的Iterator是同一介面，但是，因為Vector是同步的，當一個Iterator被創建而且正在被使用，另一個執行緒改變了 Vector的狀態(例如，添加或刪除了一些元素)，這時呼叫Iterator的方法時將拋出 ConcurrentModificationException，因此必須捕獲該異常。

### Stack 類別

Stack繼承自Vector，實現一個後進先出的堆疊。Stack提供5個額外的方法使得Vector得以被當作堆疊使用。基本的push和pop 方法，還有peek方法得到堆疊頂的元素，empty方法測試堆疊是否為空，search方法檢測一個元素在堆疊中的位置。Stack剛創建後是空堆疊。

## Set介面

Set是一種不包含重複的元素的Collection，即任意的兩個元素e1和e2都有e1.equals(e2)=false，Set最多有一個null元素。

很明顯，Set的構造函數有一個約束條件，傳入的Collection參數不能包含重複的元素。

請注意：必須小心操作可變物件(Mutable Object)。如果一個Set中的可變元素改變了自身狀態導致Object.equals(Object)=true將導致一些問題。

## Map介面

請注意，Map沒有繼承Collection介面，Map提供key到value的映射。一個Map中不能包含相同的key，每個key只能映射一個 value。Map介面提供3種集合的視圖，Map的內容可以被當作一組key集合，一組value集合，或者一組key-value映射。

### Hashtable類別

Hashtable繼承Map介面，實現一個key-value映射的雜湊表。任何非空(non-null)的物件都可作為key或者value。

添加資料使用put(key, value)，取出資料使用get(key)，這兩個基本操作的時間開銷為常數。

Hashtable通過initial capacity和load factor兩個參數調整性能。通常缺省的load factor 0.75較好地實現了時間和空間的均衡。增大load factor可以節省空間但相應的查找時間將增大，這會影響像get和put這樣的操作。

使用Hashtable的簡單範例如下，將1，2，3放到Hashtable中，他們的key分別是"one"，"two"，"three"：

Hashtable numbers = new Hashtable();

numbers.put("one", new Integer(1));

numbers.put("two", new Integer(2));

numbers.put("three", new Integer(3));

要取出一個數，比如2，用相應的key：

Integer n = (Integer)numbers.get("two");

System.out.println("two = " + n);

由於作為key的物件將通過計算其散列函數來確定與之對應的value的位置，因此任何作為key的物件都必須實現hashCode和equals方 法。hashCode和equals方法繼承自根類別Object，如果你用自訂的類別當作key的話，要相當小心，按照散列函數的定義，如果兩個物件相 同，即obj1.equals(obj2)=true，則它們的hashCode必須相同，但如果兩個物件不同，則它們的hashCode不一定不同，如 果兩個不同物件的hashCode相同，這種現象稱為衝突，衝突會導致操作雜湊表的時間開銷增大，所以儘量定義好的hashCode()方法，能加快雜湊 表的操作。

如果相同的物件有不同的hashCode，對雜湊表的操作會出現意想不到的結果(期待的get方法返回null)，要避免這種問題，只需要牢記一條：要同時複寫equals方法和hashCode方法，而不要只寫其中一個。

Hashtable是同步的。

### HashMap類別

HashMap和Hashtable類似，不同之處在於HashMap是非同步的，並且允許null，即null value和null key。，但是將HashMap視為Collection時(values()方法可返回Collection)，其反覆運運算元操作時間開銷和HashMap 的容量成比例。因此，如果反覆運算操作的性能相當重要的話，不要將HashMap的初始化容量設得過高，或者load factor過低。

### WeakHashMap類別

WeakHashMap是一種改進的HashMap，它對key實行"弱引用"，如果一個key不再被外部所引用，那麼該key可以被GC回收。

## 總結

如果涉及到堆疊，佇列等操作，應該考慮用List，對於需要快速插入，刪除元素，應該使用LinkedList，如果需要快速隨機存取元素，應該使用ArrayList。

如果程式在單執行緒環境中，或者存取僅僅在一個執行緒中進行，考慮非同步的類別，其效率較高，如果多個執行緒可能同時操作一個類別，應該使用同步的類別。

要特別注意對雜湊表的操作，作為key的物件要正確複寫equals和hashCode方法。

儘量返回介面而非實際的型態，如返回List而非ArrayList，這樣如果以後需要將ArrayList換成LinkedList時，用戶端程式碼不用改變。這就是針對抽象程式設計。

### 同步性

Vector是同步的。這個類別中的一些方法保證了Vector中的物件是執行緒安全的。而ArrayList則是非同步的，因此ArrayList中的物件並 不是執行緒安全的。因為同步的要求會影響執行的效率，所以如果你不需要執行緒安全的集合那麼使用ArrayList是一個很好的選擇，這樣可以避免由於同步帶 來的不必要的性能開銷。

### 資料增長

從內部實現機制來講ArrayList和Vector都是使用陣列(Array)來控制集合中的物件。當你向這兩種類別型中增加元素的時候，如果元素的數目 超出了內部陣列目前的長度它們都需要擴展內部陣列的長度，Vector缺省情況下自動增長原來一倍的陣列長度，ArrayList是原來的50%,所以最 後你獲得的這個集合所占的空間總是比你實際需要的要大。所以如果你要在集合中保存大量的資料那麼使用Vector有一些優勢，因為你可以通過設定集合的初 始化大小來避免不必要的資源開銷。

### 使用模式

在ArrayList和Vector中，從一個指定的位置(通過索引)查找資料或是在集合的末尾增加、移除一個元素所花費的時間是一樣的，這個時間我們用 O(1)表示。但是，如果在集合的其他位置增加或移除元素那麼花費的時間會呈線形增長：O(n-i)，其中n代表集合中元素的個數，i代表元素增加或移除 元素的索引位置。為什麼會這樣呢？以為在進行上述操作的時候集合中第i和第i個元素之後的所有元素都要執行位移的操作。這一切意味著什麼呢？

這意味著，你只是查找特定位置的元素或只在集合的末端增加、移除元素，那麼使用Vector或ArrayList都可以。如果是其他操作，你最好選擇其他 的集合操作類別。比如，LinkList集合類別在增加或移除集合中任何位置的元素所花費的時間都是一樣的?O(1)，但它在索引一個元素的使用缺比較慢 －O(i),其中i是索引的位置.使用ArrayList也很容易，因為你可以簡單的使用索引來代替創建iterator物件的操作。LinkList也 會為每個插入的元素創建物件，所有你要明白它也會帶來額外的開銷。

最後，在《Practical Java》一書中Peter Haggar建議使用一個簡單的陣列(Array)來代替Vector或ArrayList。尤其是對於執行效率要求高的程式更應如此。因為使用陣列 (Array)避免了同步、額外的方法呼叫和不必要的重新分配空間的操作。

### 相互區別

#### Vector和ArrayList

1. vector是執行緒同步的，所以它也是執行緒安全的，而arraylist是執行緒非同步的，是不安全的。如果不考慮到執行緒的安全因素，一般用arraylist效率比較高。
2. 如果集合中的元素的數目大於目前集合陣列的長度時，vector增長率為目前陣列長度的100%,而arraylist增長率為目前陣列長度的50%.如過在集合中使用資料量比較大的資料，用vector有一定的優勢。
3. 如果查找一個指定位置的資料，vector和arraylist使用的時間是相同的，都是0(1),這個時候使用vector和arraylist都可以。而如果移動一個指定位置的資料花費的時間為0(n-i)n為總長度，這個時候就應該考慮到使用linklist,因為它移動一個指定位置的資料所花費的時間為0(1),而查詢一個指定位置的資料時花費的時間為0(i)。

ArrayList 和Vector是採用陣列方式存儲資料，此陣列元素數大於實際存儲的資料以便增加和插入元素，都允許直接序號索引元素，但是插入資料要設計到陣列元素移動 等記憶體操作，所以索引資料快插入資料慢，Vector由於使用了synchronized方法(執行緒安全)所以性能上比ArrayList要 差，LinkedList使用雙向鏈表實現存儲，按序號索引資料需要進行向前或向後檢視，但是插入資料時只需要記錄本項的前後項即可，所以插入數度較快！

#### arraylist和linkedlist

1. ArrayList是實現了基於動態陣列的資料結構，LinkedList基於鏈表的資料結構。
2. 對於隨機存取get和set，ArrayList覺得優於LinkedList，因為LinkedList要移動指標。
3. 對於新增和刪除操作add和remove，LinedList比較佔優勢，因為ArrayList要移動資料。

這一點要看實際情況的。若只對單條資料插入或刪除，ArrayList的速度反而優於LinkedList。但若是批量隨機的插入刪除數 據，LinkedList的速度大大優於ArrayList. 因為ArrayList每插入一條資料，要移動插入點及之後的所有資料。

#### HashMap與TreeMap

(注)文章出處：http://www.diybl.com/course/3\_program/java/javaxl/200875/130233.html

1. HashMap通過hashcode對其內容進行快速查找，而TreeMap中所有的元素都保持著某種固定的順序，如果你需要得到一個有序的結果你就應該使用TreeMap(HashMap中元素的排列順序是不固定的)。HashMap中元素的排列順序是不固定的)。
2. HashMap通過hashcode對其內容進行快速查找，而TreeMap中所有的元素都保持著某種固定的順序，如果你需要得到一個有序的結果你就應該 使用TreeMap(HashMap中元素的排列順序是不固定的)。集合框架"提供兩種普通的Map實現：HashMap和TreeMap (TreeMap實現SortedMap介面)。
3. 在Map 中插入、刪除和定位元素，HashMap 是最好的選擇。但如果您要按自然順序或自訂順序檢視鍵，那麼TreeMap會更好。使用HashMap要求添加的鍵類別明確定義了hashCode()和 equals()的實現。這個TreeMap沒有調優選項，因為該樹總處於平衡狀態。

結過研究，在原作者的基礎上我還發現了一點，二樹map一樣，但順序不一樣，導致hashCode()不一樣。

同樣做測試：

在hashMap中，同樣的值的map,順序不同，equals時，false;

而在treeMap中，同樣的值的map,順序不同,equals時，true，說明，treeMap在equals()時是整理了順序了的。

#### hashtable與hashmap

1. 歷史原因:Hashtable是基於陳舊的Dictionary類別的，HashMap是Java 1.2引進的Map介面的一個實現。
2. 同步性:Hashtable是執行緒安全的，也就是說是同步的，而HashMap是線程式不安全的，不是同步的。
3. 值：只有HashMap可以讓你將空值作為一個表的條目的key或value。

# [Java集合類別: Set、List、Map、Queue使用場景梳理](http://www.cnblogs.com/LittleHann/p/3690187.html)

本文主要關注Java程式設計中涉及到的各種集合類別，以及它們的使用場景

## Java集合類別基本概念

在程式設計中，常常需要集中存放多個資料。從傳統意義上講，陣列是我們的一個很好的選擇，前提是我們事先已經明確知道我們將要保存的物件的數量。一旦在陣列初始化時指定了這個陣列長度，這個陣列長度就是不可變的，如果我們需要保存一個可以動態增長的資料(在編譯時無法確定具體的數量)，java的集合類別就是一個很好的設計方案了。

集合類別主要負責保存、盛裝其他資料，因此集合類別也被稱為容器類別。所以的集合類別都位於java.util包下，後來為了處理多執行緒環境下的併發安全問題，java5還在java.util.concurrent包下提供了一些多執行緒支援的集合類別。

在學習Java中的集合類別的API、程式設計原理的時候，我們一定要明白，"集合"是一個很古老的數學概念，它遠遠早於Java的出現。從數學概念的角度來理解集合能説明我們更好的理解程式設計中什麼時候該使用什麼型態的集合類別。

Java容器類別類別庫的用途是"保存物件"，並將其劃分為兩個不同的概念：

### Collection

一組"對立"的元素，通常這些元素都服從某種規則

1. List必須保持元素特定的順序
2. Set不能有重複元素
3. Queue保持一個佇列(先進先出)的順序

### Map

一組成對的"鍵值對"物件

Collection和Map的區別在於容器中每個位置保存的元素個數:

1. Collection 每個位置只能保存一個元素(物件)
2. Map保存的是"鍵值對"，就像一個小型資料庫。我們可以通過"鍵"找到該鍵對應的"值"

## Java集合類別架構層次關係

### Interface Iterable

反覆運算器介面，這是Collection類別的父介面。實現這個Iterable介面的物件允許使用foreach進行檢視，也就是說，所有的Collection集合物件都具有"foreach可檢視性"。這個Iterable介面只有一個方法: iterator()。它返回一個代表當前集合物件的泛型<T>反覆運算器，用於之後的檢視操作

#### Collection

Collection是最基本的集合介面，一個Collection代表一組Object的集合，這些Object被稱作Collection的元素。Collection是一個介面，用以提供規範定義，不能被產生實體使用

##### Set

Set集合類似於一個罐子，"丟進"Set集合裡的多個物件之間沒有明顯的順序。Set繼承自Collection介面，不能包含有重複元素(記住，這是整個Set類別層次的共有屬性)。

Set判斷兩個物件相同不是使用"=="運算子，而是根據equals方法。也就是說，我們在加入一個新元素的時候，如果這個新元素物件和Set中已有物件進行注意equals比較都返回false，則Set就會接受這個新元素物件，否則拒絕。

因為Set的這個制約，在使用Set集合的時候，應該注意兩點：1) 為Set集合裡的元素的實現類別實現一個有效的equals(Object)方法、2) 對Set的構造函數，傳入的Collection參數不能包含重複的元素

###### HashSet

HashSet是Set介面的典型實現，HashSet使用HASH演算法來存儲集合中的元素，因此具有良好的存取和查找性能。當向HashSet集合中存入一個元素時，HashSet會呼叫該物件的hashCode()方法來得到該物件的hashCode值，然後根據該HashCode值決定該物件在HashSet中的存儲位置。

值得主要的是，HashSet集合判斷兩個元素相等的標準是兩個物件通過equals()方法比較相等，並且兩個物件的hashCode()方法的返回值相等

###### LinkedHashSet

LinkedHashSet集合也是根據元素的hashCode值來決定元素的存儲位置，但和HashSet不同的是，它同時使用鏈表維護元素的次序，這樣使得元素看起來是以插入的順序保存的。

當檢視LinkedHashSet集合裡的元素時，LinkedHashSet將會按元素的添加順序來存取集合裡的元素。

LinkedHashSet需要維護元素的插入順序，因此性能略低於HashSet的性能，但在反覆運算存取Set裡的全部元素時(檢視)將有很好的性能(鏈表很適合進行檢視)

##### SortedSet

此介面主要用於排序操作，即實現此介面的子類別都屬於排序的子類別

###### TreeSet

TreeSet是SortedSet介面的實現類別，TreeSet可以確保集合元素處於排序狀態

##### EnumSet

EnumSet是一個專門為枚舉類別設計的集合類別，EnumSet中所有元素都必須是指定枚舉型態的枚舉值，該枚舉型態在創建EnumSet時顯式、或隱式地指定。EnumSet的集合元素也是有序的，

它們以枚舉值在Enum類別內的定義順序來決定集合元素的順序

#### List

List集合代表一個元素有序、可重複的集合，集合中每個元素都有其對應的順序索引。List集合允許加入重複元素，因為它可以通過索引來存取指定位置的集合元素。List集合預設按元素

的添加順序設定元素的索引

##### ArrayList

ArrayList是基於陣列實現的List類別，它封裝了一個動態的增長的、允許再分配的Object[]陣列。

##### Vector

Vector和ArrayList在用法上幾乎完全相同，但由於Vector是一個古老的集合，所以Vector提供了一些方法名很長的方法，但隨著JDK1.2以後，java提供了系統的集合框架，就將

Vector改為實現List介面，統一歸入集合框架體系中

###### Stack

Stack是Vector提供的一個子類別，用於類比"堆疊"這種資料結構(LIFO後進先出)

##### LinkedList

implements List<E>, Deque<E>。實現List介面，能對它進行佇列操作，即可以根據索引來隨機存取集合中的元素。同時它還實現Deque介面，即能將LinkedList當作雙端佇列

使用。自然也可以被當作"堆疊來使用"

#### Queue

Queue用於類別比"佇列"這種資料結構(先進先出 FIFO)。佇列的頭部保存著佇列中存放時間最長的元素，佇列的尾部保存著佇列中存放時間最短的元素。新元素插入(offer)到佇列的尾部，

存取元素(poll)操作會返回佇列頭部的元素，佇列不允許隨機存取佇列中的元素。結合生活中常見的排隊就會很好理解這個概念

##### PriorityQueue

PriorityQueue並不是一個比較標準的佇列實現，PriorityQueue保存佇列元素的順序並不是按照加入佇列的順序，而是按照佇列元素的大小進行重新排序，這點從它的類別名也可以

看出來

##### Deque

Deque介面代表一個"雙端佇列"，雙端佇列可以同時從兩端來添加、刪除元素，因此Deque的實現類別既可以當成佇列使用、也可以當成堆疊使用

###### ArrayDeque

是一個基於陣列的雙端佇列，和ArrayList類似，它們的底層都採用一個動態的、可重分配的Object[]陣列來存儲集合元素，當集合元素超出該陣列的容量時，系統會在底層重

新分配一個Object[]陣列來存儲集合元素

###### LinkedList

### Map

Map用於保存具有"映射關係"的資料，因此Map集合裡保存著兩組值，一組值用於保存Map裡的key，另外一組值用於保存Map裡的value。key和value都可以是任何參考型態的資料。Map的key不允

許重複，即同一個Map物件的任何兩個key通過equals方法比較結果總是返回false。

關於Map，我們要從程式碼複用的角度去理解，java是先實現了Map，然後通過包裝了一個所有value都為null的Map就實現了Set集合

Map的這些實現類別和子介面中key集的存儲形式和Set集合完全相同(即key不能重複)

Map的這些實現類別和子介面中value集的存儲形式和List非常類似(即value可以重複、根據索引來查找)

#### HashMap

和HashSet集合不能保證元素的順序一樣，HashMap也不能保證key-value對的順序。並且類似於HashSet判斷兩個key是否相等的標準也是: 兩個key通過equals()方法比較返回true、同時兩個key的hashCode值也必須相等

##### LinkedHashMap

LinkedHashMap也使用雙向鏈表來維護key-value對的次序，該鏈表負責維護Map的反覆運算順序，與key-value對的插入順序一致(注意和TreeMap對所有的key-value進行排序進行區分)

#### Hashtable

是一個古老的Map實現類別

###### Properties

Properties物件在處理屬性檔時特別方便(windows平臺上的.ini檔)，Properties類別可以把Map物件和屬性檔關聯起來，從而可以把Map物件中的key-value對寫入到屬性文

件中，也可以把屬性檔中的"屬性名-屬性值"載入到Map物件中

### SortedMap

正如Set介面衍生出SortedSet子介面，SortedSet介面有一個TreeSet實現類別一樣，Map介面也衍生出一個SortedMap子介面，SortedMap介面也有一個TreeMap實現類別

#### TreeMap

TreeMap就是一個紅黑樹資料結構，每個key-value對即作為紅黑樹的一個節點。TreeMap存儲key-value對(節點)時，需要根據key對節點進行排序。TreeMap可以保證所有的

key-value對處於有序狀態。同樣，TreeMap也有兩種排序方式: 自然排序、定制排序

### WeakHashMap

WeakHashMap與HashMap的用法基本相似。區別在於，HashMap的key保留了對實際物件的"強引用"，這意味著只要該HashMap物件不被銷毀，該HashMap所引用的物件就不會被垃圾回收。

但WeakHashMap的key只保留了對實際物件的弱引用，這意味著如果WeakHashMap物件的key所引用的物件沒有被其他強引用變數所引用，則這些key所引用的物件可能被垃圾回收，當垃

圾回收了該key所對應的實際物件之後，WeakHashMap也可能自動刪除這些key所對應的key-value對

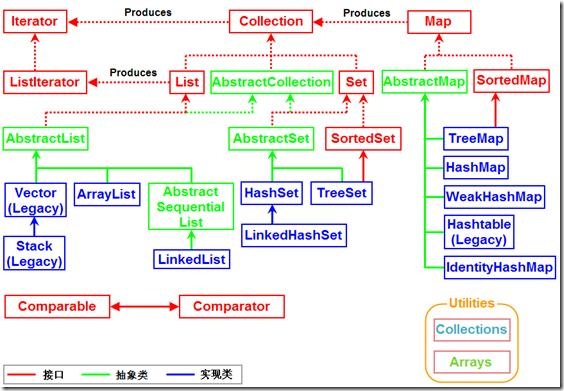
### IdentityHashMap

IdentityHashMap的實現機制與HashMap基本相似，在IdentityHashMap中，當且僅當兩個key嚴格相等(key1 == key2)時，IdentityHashMap才認為兩個key相等

### EnumMap

EnumMap是一個與枚舉類別一起使用的Map實現，EnumMap中的所有key都必須是單個枚舉類別的枚舉值。創建EnumMap時必須顯式或隱式指定它對應的枚舉類別。EnumMap根據key的自然順序

(即枚舉值在枚舉類別中的定義順序)



## Java集合類別的應用場景程式碼

學習了集合類別的基本架構框架之後，我們接著來學習它們各自的應用場景、以及細節處的注意事項

0x1: Set

HashSet

import java.util.\*;

//類別A的equals方法總是返回true,但沒有重寫其hashCode()方法。不能保證當前物件是HashSet中的唯一物件

class A {

public boolean equals(Object obj) {

return true;

}

}

// 類別B的hashCode()方法總是返回1,但沒有重寫其equals()方法。不能保證當前物件是HashSet中的唯一物件

class B {

public int hashCode() {

return 1;

}

}

// 類別C的hashCode()方法總是返回2,且有重寫其equals()方法

class C {

public int hashCode() {

return 2;

}

public boolean equals(Object obj) {

return true;

}

}

public class HashSetTest {

public static void main(String[] args) {

HashSet books = new HashSet();

// 分別向books集合中添加兩個A物件，兩個B物件，兩個C物件

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);

}

}

result:

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

可以看到，如果兩個物件通過equals()方法比較返回true，但這兩個物件的hashCode()方法返回不同的hashCode值時，這將導致HashSet會把這兩個物件保存在Hash表的不同位置，從而使物件可以添加成功，這就與Set集合的規則有些出入了。所以，我們要明確的是: equals()決定是否可以加入HashSet、而hashCode()決定存放的位置，它們兩者必須同時滿足才能允許一個新元素加入HashSet

但是要注意的是: 如果兩個物件的hashCode相同，但是它們的equlas返回值不同，HashSet會在這個位置用鏈式結構來保存多個物件。而HashSet存取集合元素時也是根據元素的HashCode值來快速定位的，這種鏈式結構會導致性能下降。

所以如果需要把某個類別的物件保存到HashSet集合中，我們在重寫這個類別的equlas()方法和hashCode()方法時，應該儘量保證兩個物件通過equals()方法比較返回true時，它們的hashCode()方法返回值也相等

LinkedHashSet

import java.util.\*;

public class LinkedHashSetTest {

public static void main(String[] args) {

LinkedHashSet books = new LinkedHashSet();

books.add("Java");

books.add("LittleHann");

System.out.println(books);

// 刪除 Java

books.remove("Java");

// 重新添加 Java

books.add("Java");

System.out.println(books);

}

}

元素的順序總是與添加順序一致，同時要明白的是，LinkedHashSetTest是HashSet的子類別，因此它不允許集合元素重複

TreeSet

import java.util.\*;

public class TreeSetTest {

public static void main(String[] args) {

TreeSet nums = new TreeSet();

// 向TreeSet中添加四個Integer物件

nums.add(5);

nums.add(2);

nums.add(10);

nums.add(-9);

// 輸出集合元素，看到集合元素已經處於排序狀態

System.out.println(nums);

// 輸出集合裡的第一個元素

System.out.println(nums.first());

// 輸出集合裡的最後一個元素

System.out.println(nums.last());

// 返回小於4的子集，不包含4

System.out.println(nums.headSet(4));

// 返回大於5的子集，如果Set中包含5，子集中還包含5

System.out.println(nums.tailSet(5));

// 返回大於等於-3，小於4的子集。

System.out.println(nums.subSet(-3, 4));

}

}

與HashSet集合採用hash演算法來決定元素的存儲位置不同，TreeSet採用紅黑樹的資料結構來存儲集合元素。TreeSet支援兩種排序方式: 自然排序、定制排序

### 自然排序:

TreeSet會呼叫集合元素的compareTo(Object obj)方法來比較元素之間的大小關係，然後將集合元素按昇冪排序，即自然排序。如果試圖把一個物件添加到TreeSet時，則該物件的類別必須實現Comparable介面，否則程式會拋出異常。

當把一個物件加入TreeSet集合中時，TreeSet會呼叫該物件的compareTo(Object obj)方法與容器中的其他物件比較大小，然後根據紅黑樹結構找到它的存儲位置。如果兩個物件通過compareTo(Object obj)方法比較相等，新物件將無法添加到TreeSet集合中(牢記Set是不允許重複的概念)。

注意: 當需要把一個物件放入TreeSet中，重寫該物件對應類別的equals()方法時，應該保證該方法與compareTo(Object obj)方法有一致的結果，即如果兩個物件通過equals()方法比較返回true時，這兩個物件通過compareTo(Object obj)方法比較結果應該也為0(即相等)

看到這裡，我們應該明白：

1. 對與Set來說，它定義了equals()為唯一性判斷的標準，而對於到了具體的實現，HashSet、TreeSet來說，它們又會有自己特有的唯一性判斷標準，只有同時滿足了才能判定為唯一性
2. 我們在操作這些集合類別的時候，對和唯一性判斷有關的函數重寫要重點關注

### 定制排序

TreeSet的自然排序是根據集合元素的大小，TreeSet將它們以昇冪排序。如果我們需要實現定制排序，則可以通過Comparator介面的説明(類似PHP中的array\_map回檔處理函數的思想)。該介面裡包含一個int compare(T o1， T o2)方法，該方法用於比較大小

import java.util.\*;

class M {

int age;

public M(int age) {

this.age = age;

}

public String toString() {

return "M[age:" + age + "]";

}

}

public class TreeSetTest4 {

public static void main(String[] args) {

TreeSet ts = new TreeSet(new Comparator() {

// 根據M物件的age屬性來決定大小

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(5));

ts.add(new M(-3));

ts.add(new M(9));

System.out.println(ts);

}

}

看到這裡，我們需要梳理一下關於排序的概念

1. equals、compareTo決定的是怎麼比的問題，即用什麼field進行大小比較
2. 自然排序、定制排序、Comparator決定的是誰大的問題，即按什麼順序(昇冪、降冪)進行排序

它們的關注點是不同的，一定要注意區分

EnumSet

import java.util.\*;

enum Season{

SPRING,SUMMER,FALL,WINTER

}

public class EnumSetTest{

public static void main(String[] args){

//創建一個EnumSet集合，集合元素就是Season枚舉類別的全部枚舉值

EnumSet es1 = EnumSet.allOf(Season.class);

//輸出[SPRING,SUMMER,FALL,WINTER]

System.out.println(es1);

//創建一個EnumSet空集合，指定其集合元素是Season類別的枚舉值。

EnumSet es2 = EnumSet.noneOf(Season.class);

//輸出[]

System.out.println(es2);

//手動添加兩個元素

es2.add(Season.WINTER);

es2.add(Season.SPRING);

//輸出[SPRING,WINTER]

System.out.println(es2);

//以指定枚舉值創建EnumSet集合

EnumSet es3 = EnumSet.of(Season.SUMMER , Season.WINTER);

//輸出[SUMMER,WINTER]

System.out.println(es3);

EnumSet es4 = EnumSet.range(Season.SUMMER , Season.WINTER);

//輸出[SUMMER,FALL,WINTER]

System.out.println(es4);

//新創建的EnumSet集合的元素和es4集合的元素有相同型態，

//es5的集合元素 + es4集合元素 = Season枚舉類別的全部枚舉值

EnumSet es5 = EnumSet.complementOf(es4);

//輸出[SPRING]

System.out.println(es5);

}

}

以上就是Set集合類別的程式設計應用場景。那麼應該怎樣選擇何時使用這些集合類別呢？

1. HashSet的性能總是比TreeSet好(特別是最常用的添加、查詢元素等操作)，因為TreeSet需要額外的紅黑樹演算法來維護集合元素的次序。只有當需要一個保持排序的Set時，才應該使用TreeSet，否則都應該使用HashSet
2. 對於普通的插入、刪除操作，LinkedHashSet比HashSet要略慢一點，這是由維護鏈表所帶來的開銷造成的。不過，因為有了鏈表的存在，檢視LinkedHashSet會更快
3. EnumSet是所有Set實現類別中性能最好的，但它只能保存同一個枚舉類別的枚舉值作為集合元素
4. HashSet、TreeSet、EnumSet都是"執行緒不安全"的，通常可以通過Collections工具類別的synchronizedSortedSet方法來"包裝"該Set集合。

SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));

## 0x2: List

ArrayList

如果一開始就知道ArrayList集合需要保存多少元素，則可以在創建它們時就指定initialCapacity大小，這樣可以減少重新分配的次數，提供性能，ArrayList還提供了如下方法來重新分配Object[]陣列

ensureCapacity(int minCapacity): 將ArrayList集合的Object[]陣列長度增加minCapacity

trimToSize(): 調整ArrayList集合的Object[]陣列長度為當前元素的個數。程式可以通過此方法來減少ArrayList集合物件佔用的記憶體空間

import java.util.\*;

public class ListTest {

public static void main(String[] args){

List books = new ArrayList();

//向books集合中添加三個元素

books.add(new String("羽量級Java EE企業應用實戰"));

books.add(new String("瘋狂Java講義"));

books.add(new String("瘋狂Android講義"));

System.out.println(books);

//將新字串物件插入在第二個位置

books.add(1 , new String("瘋狂Ajax講義"));

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

System.out.println(books.get(i));

}

//刪除第三個元素

books.remove(2);

System.out.println(books);

//判斷指定元素在List集合中位置：輸出1，表明位於第二位

System.out.println(books.indexOf(new String("瘋狂Ajax講義")));//①

//將第二個元素替換成新的字串物件

books.set(1, new String("LittleHann"));

System.out.println(books);

//將books集合的第二個元素(包括)

//到第三個元素(不包括)截取成子集合

System.out.println(books.subList(1 , 2));

}

Stack

注意Stack的後進先出的特點

import java.util.\*;

public class VectorTest{

public static void main(String[] args){

Stack v = new Stack();

//依次將三個元素push入"堆疊"

v.push("瘋狂Java講義");

v.push("羽量級Java EE企業應用實戰");

v.push("瘋狂Android講義");

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(v);

//存取第一個元素，但並不將其pop出"堆疊"，輸出：瘋狂Android講義

System.out.println(v.peek());

//依然輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(v);

//pop出第一個元素，輸出：瘋狂Android講義

System.out.println(v.pop());

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰]

System.out.println(v);

}

}

LinkedList

import java.util.\*;

public class LinkedListTest{

public static void main(String[] args){

LinkedList books = new LinkedList();

//將字串元素加入佇列的尾部(雙端佇列)

books.offer("瘋狂Java講義");

//將一個字串元素加入堆疊的頂部(雙端佇列)

books.push("羽量級Java EE企業應用實戰");

//將字串元素添加到佇列的頭(相當於堆疊的頂部)

books.offerFirst("瘋狂Android講義");

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

System.out.println(books.get(i));

}

//存取、並不刪除堆疊頂的元素

System.out.println(books.peekFirst());

//存取、並不刪除佇列的最後一個元素

System.out.println(books.peekLast());

//將堆疊頂的元素彈出"堆疊"

System.out.println(books.pop());

//下面輸出將看到佇列中第一個元素被刪除

System.out.println(books);

//存取、並刪除佇列的最後一個元素

System.out.println(books.pollLast());

//下面輸出將看到佇列中只剩下中間一個元素：

//羽量級Java EE企業應用實戰

System.out.println(books);

}

}

從程式碼中我們可以看到，LinkedList同時表現出了雙端佇列、堆疊的用法。功能非常強大

## 0x3: Queue

PriorityQueue

import java.util.\*;

public class PriorityQueueTest{

public static void main(String[] args){

PriorityQueue pq = new PriorityQueue();

//下面程式碼依次向pq中加入四個元素

pq.offer(6);

pq.offer(-3);

pq.offer(9);

pq.offer(0);

//輸出pq佇列，並不是按元素的加入順序排列，

//而是按元素的大小順序排列，輸出[-3, 0, 9, 6]

System.out.println(pq);

//存取佇列第一個元素，其實就是佇列中最小的元素：-3

System.out.println(pq.poll());

}

}

PriorityQueue不允許插入null元素，它還需要對佇列元素進行排序，PriorityQueue的元素有兩種排序方式

1) 自然排序:

採用自然順序的PriorityQueue集合中的元素物件都必須實現了Comparable介面，而且應該是同一個類別的多個實例，否則可能導致ClassCastException異常

2) 定制排序

創建PriorityQueue佇列時，傳入一個Comparator物件，該物件負責對佇列中的所有元素進行排序

關於自然排序、定制排序的原理和之前說的TreeSet類似

 ArrayDeque

import java.util.\*;

public class ArrayDequeTest{

public static void main(String[] args){

ArrayDeque stack = new ArrayDeque();

//依次將三個元素push入"堆疊"

stack.push("瘋狂Java講義");

stack.push("羽量級Java EE企業應用實戰");

stack.push("瘋狂Android講義");

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(stack);

//存取第一個元素，但並不將其pop出"堆疊"，輸出：瘋狂Android講義

System.out.println(stack.peek());

//依然輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰 , 瘋狂Android講義]

System.out.println(stack);

//pop出第一個元素，輸出：瘋狂Android講義

System.out.println(stack.pop());

//輸出：[瘋狂Java講義, 羽量級Java EE企業應用實戰]

System.out.println(stack);

}

}

以上就是List集合類別的程式設計應用場景。我們來梳理一下思路

1. java提供的List就是一個"線性表介面"，ArrayList(基於陣列的線性表)、LinkedList(基於鏈的線性表)是線性表的兩種典型實現

2. Queue代表了佇列，Deque代表了雙端佇列(既可以作為佇列使用、也可以作為堆疊使用)

3. 因為陣列以一塊連續記憶體來保存所有的陣列元素，所以陣列在隨機存取時性能最好。所以的內部以陣列作為底層實現的集合在隨機存取時性能最好。

4. 內部以鏈表作為底層實現的集合在執行插入、刪除操作時有很好的性能

5. 進行反覆運算操作時，以鏈表作為底層實現的集合比以陣列作為底層實現的集合性能好

我們之前說過，Collection介面繼承了Iterable介面，也就是說，我們以上學習到的所有的Collection集合類別都具有"可檢視性"

Iterable介面也是java集合框架的成員，它隱藏了各種Collection實現類別的底層細節，向應用程式提供了檢視Collection集合元素的統一程式設計介面:

1) boolean hasNext(): 是否還有下一個未檢視過的元素

2) Object next(): 返回集合裡的下一個元素

3) void remove(): 刪除集合裡上一次next方法返回的元素

iterator實現檢視:

import java.util.\*;

public class IteratorTest{

public static void main(String[] args){

//創建一個集合

Collection books = new HashSet();

books.add("羽量級Java EE企業應用實戰");

books.add("瘋狂Java講義");

books.add("瘋狂Android講義");

//獲取books集合對應的反覆運算器

Iterator it = books.iterator();

while(it.hasNext()){

//it.next()方法返回的資料型態是Object型態，

//需要強制型態轉換

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

System.out.println(book);

if (book.equals("瘋狂Java講義")){

//從集合中刪除上一次next方法返回的元素

it.remove();

}

//對book變數賦值，不會改變集合元素本身

book = "測試字串";

}

System.out.println(books);

}

}

從程式碼可以看出，iterator必須依附於Collection物件，若有一個iterator物件，必然有一個與之關聯的Collection物件。

除了可以使用iterator介面反覆運算存取Collection集合裡的元素之外，使用java5提供的foreach迴圈反覆運算存取集合元素更加便捷

foreach實現檢視:

import java.util.\*;

public class ForeachTest{

public static void main(String[] args){

//創建一個集合

Collection books = new HashSet();

books.add(new String("羽量級Java EE企業應用實戰"));

books.add(new String("瘋狂Java講義"));

books.add(new String("瘋狂Android講義"));

for (Object obj : books){

//此處的book變數也不是集合元素本身

String book = (String)obj;

System.out.println(book);

if (book.equals("瘋狂Android講義")){

//下面程式碼會引發ConcurrentModificationException異常

//books.remove(book);

}

}

System.out.println(books);

}

}

除了Collection固有的iterator()方法，List還額外提供了一個listIterator()方法，該方法返回一個ListIterator物件，ListIterator介面繼承了Iterator介面，提供了專門操作List的方法。ListIterator介面在Iterator介面的繼承上增加了如下方法:

1) boolean hasPrevious(): 返回該反覆運算器關聯的集合是否還有上一個元素

2) Object previous(): 返回該反覆運算器的上一個元素(向前反覆運算)

3) void add(): 在指定位置插入一個元素

ListIterator實現檢視:

import java.util.\*;

public class ListIteratorTest{

public static void main(String[] args){

String[] books ={"瘋狂Java講義","羽量級Java EE企業應用實戰"};

List bookList = new ArrayList();

for (int i = 0; i < books.length; i++){

bookList.add(books[i]);

}

ListIterator lit = bookList.listIterator();

while (lit.hasNext()){

System.out.println(lit.next());

lit.add("-------分隔符號-------");

}

System.out.println("=======下面開始反向反覆運算=======");

while(lit.hasPrevious()){

System.out.println(lit.previous());

}

}

}

0x4: Map

HashMap、Hashtable

import java.util.\*;

class A{

int count;

public A(int count){

this.count = count;

}

//根據count的值來判斷兩個物件是否相等。

public boolean equals(Object obj){

if (obj == this) return true;

if (obj!=null && obj.getClass()==A.class){

A a = (A)obj;

return this.count == a.count;

}

return false;

}

//根據count來計算hashCode值。

public int hashCode(){

return this.count;

}

}

class B{

//重寫equals()方法，B物件與任何物件通過equals()方法比較都相等

public boolean equals(Object obj){

return true;

}

}

public class HashtableTest{

public static void main(String[] args){

Hashtable ht = new Hashtable();

ht.put(new A(60000) , "瘋狂Java講義");

ht.put(new A(87563) , "羽量級Java EE企業應用實戰");

ht.put(new A(1232) , new B());

System.out.println(ht);

//只要兩個物件通過equals比較返回true，

//Hashtable就認為它們是相等的value。

//由於Hashtable中有一個B物件，

//它與任何物件通過equals比較都相等，所以下面輸出true。

System.out.println(ht.containsValue("測試字串"));//①

//只要兩個A物件的count相等，它們通過equals比較返回true，且hashCode相等

//Hashtable即認為它們是相同的key，所以下面輸出true。

System.out.println(ht.containsKey(new A(87563)));//②

//下面語句可以刪除最後一個key-value對

ht.remove(new A(1232));//③

//通過返回Hashtable的所有key組成的Set集合，

//從而檢視Hashtable每個key-value對

for (Object key : ht.keySet()){

System.out.print(key + "---->");

System.out.print(ht.get(key) + "\n");

}

}

}

當使用自訂類別作為HashMap、Hashtable的key時，如果重寫該類別的equals(Object obj)和hashCode()方法，則應該保證兩個方法的判斷標準一致--當兩個key通過equals()方法比較返回true時，兩個key的hashCode()的返回值也應該相同

LinkedHashMap

import java.util.\*;

public class LinkedHashMapTest{

public static void main(String[] args){

LinkedHashMap scores = new LinkedHashMap();

scores.put("語文" , 80);

scores.put("英文" , 82);

scores.put("數學" , 76);

//檢視scores裡的所有的key-value對

for (Object key : scores.keySet()){

System.out.println(key + "------>" + scores.get(key));

}

}

}

Properties

import java.util.\*;

import java.io.\*;

public class PropertiesTest{

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

Properties props = new Properties();

//向Properties中增加屬性

props.setProperty("username" , "yeeku");

props.setProperty("password" , "123456");

//將Properties中的key-value對保存到a.ini文件中

props.store(new FileOutputStream("a.ini"), "comment line");//①

//新建一個Properties物件

Properties props2 = new Properties();

//向Properties中增加屬性

props2.setProperty("gender" , "male");

//將a.ini文件中的key-value對追加到props2中

props2.load(new FileInputStream("a.ini"));//②

System.out.println(props2);

}

}

Properties還可以把key-value對以XML檔的形式保存起來，也可以從XML檔中載入key-value對

TreeMap

import java.util.\*;

class R implements Comparable{

int count;

public R(int count){

this.count = count;

}

public String toString(){

return "R[count:" + count + "]";

}

//根據count來判斷兩個物件是否相等。

public boolean equals(Object obj){

if (this == obj)

return true;

if (obj!=null && obj.getClass()==R.class){

R r = (R)obj;

return r.count == this.count;

}

return false;

}

//根據count屬性值來判斷兩個物件的大小。

public int compareTo(Object obj){

R r = (R)obj;

return count > r.count ? 1 :

count < r.count ? -1 : 0;

}

}

public class TreeMapTest{

public static void main(String[] args){

TreeMap tm = new TreeMap();

tm.put(new R(3) , "羽量級Java EE企業應用實戰");

tm.put(new R(-5) , "瘋狂Java講義");

tm.put(new R(9) , "瘋狂Android講義");

System.out.println(tm);

//返回該TreeMap的第一個Entry物件

System.out.println(tm.firstEntry());

//返回該TreeMap的最後一個key值

System.out.println(tm.lastKey());

//返回該TreeMap的比new R(2)大的最小key值。

System.out.println(tm.higherKey(new R(2)));

//返回該TreeMap的比new R(2)小的最大的key-value對。

System.out.println(tm.lowerEntry(new R(2)));

//返回該TreeMap的子TreeMap

System.out.println(tm.subMap(new R(-1) , new R(4)));

}

}

從程式碼中可以看出，類似於TreeSet中判斷兩個元素是否相等的標準，TreeMap中判斷兩個key相等的標準是:

1) 兩個key通過compareTo()方法返回0

2) equals()放回true

我們在重寫這兩個方法的時候一定要保證它們的邏輯關係一致。

再次強調一下:

Set和Map的關係十分密切，java原始碼就是先實現了HashMap、TreeMap等集合，然後通過包裝一個所有的value都為null的Map集合實現了Set集合類別

WeakHashMap

import java.util.\*;

public class WeakHashMapTest{

public static void main(String[] args){

WeakHashMap whm = new WeakHashMap();

//將WeakHashMap中添加三個key-value對，

//三個key都是匿名字串物件(沒有其他引用)

whm.put(new String("語文") , new String("良好"));

whm.put(new String("數學") , new String("及格"));

whm.put(new String("英文") , new String("中等"));

//將WeakHashMap中添加一個key-value對，

//該key是一個系統緩存的字串物件。"java"是一個常量字串強引用

whm.put("java" , new String("中等"));

//輸出whm物件，將看到4個key-value對。

System.out.println(whm);

//通知系統立即進行垃圾回收

System.gc();

System.runFinalization();

//通常情況下，將只看到一個key-value對。

System.out.println(whm);

}

}

如果需要使用WeakHashMap的key來保留物件的弱引用，則不要讓key所引用的物件具有任何強引用，否則將失去使用WeakHashMap的意義

IdentityHashMap

import java.util.\*;

public class IdentityHashMapTest{

public static void main(String[] args){

IdentityHashMap ihm = new IdentityHashMap();

//下面兩行程式碼將會向IdentityHashMap物件中添加兩個key-value對

ihm.put(new String("語文") , 89);

ihm.put(new String("語文") , 78);

//下面兩行程式碼只會向IdentityHashMap物件中添加一個key-value對

ihm.put("java" , 93);

ihm.put("java" , 98);

System.out.println(ihm);

}

}

EnumMap

import java.util.\*;

enum Season{

SPRING,SUMMER,FALL,WINTER

}

public class EnumMapTest{

public static void main(String[] args){

//創建一個EnumMap物件，該EnumMap的所有key

//必須是Season枚舉類別的枚舉值

EnumMap enumMap = new EnumMap(Season.class);

enumMap.put(Season.SUMMER , "夏日炎炎");

enumMap.put(Season.SPRING , "春暖花開");

System.out.println(enumMap);

}

}

與創建普通Map有所區別的是，創建EnumMap是必須指定一個枚舉類別，從而將該EnumMap和指定枚舉類別關聯起來

以上就是Map集合類別的程式設計應用場景。我們來梳理一下思路

1) HashMap和Hashtable的效率大致相同，因為它們的實現機制幾乎完全一樣。但HashMap通常比Hashtable要快一點，因為Hashtable需要額外的執行緒同步控制

2) TreeMap通常比HashMap、Hashtable要慢(尤其是在插入、刪除key-value對時更慢)，因為TreeMap底層採用紅黑樹來管理key-value對

3) 使用TreeMap的一個好處就是： TreeMap中的key-value對總是處於有序狀態，無須專門進行排序操作