# 第11章 持有对象

## 11.1 泛型和类型安全的容器

## 11.2 基本概念

1)Collection。一个独立元素的序列，这些元素都服从一条或多条规则。List必须按照插入的顺序保存元素，而Set不能有重复元素。Queue按照排队规则来确定对象产生的顺序(通常与他们被插入的顺序相同)。

2)Map。一组成对的”键值对”对象，允许你使用键来查找值。ArrayList允许你使用数字来查找值。

## 11.3 添加一组元素

在java.util包中的Arrays和Collections类中都有很多实用的方法，可以在一个Collection中添加一组元素。Arrays.asList()方法接受一个数组或是一个用逗号分隔的元素列表，并将其转换为一个List对象。Collections.addAll()方法接受一个Collection对象，以及一个数组或是一个用逗号分隔的列表，将元素添加到Collection中。

Collection<Integer> collection = new ArrayList<Integer>(Arrays.asList(1,2,3,4,5));  
Integer[] moreInts = {6,7,8,9,10};  
collection.addAll(Arrays.asList(moreInts));  
Collections.addAll(collection, 11,12,13,14,15);  
Collections.addAll(collection, moreInts);  
List<Integer> list = Arrays.asList(16,17,18,19,20);  
list.set(1, 99);

ArrayList和LinkedList都是List类型，从输出可以看出，它们都按照被插入的顺序保存元素。两者的不同之处不仅在于执行某些类型时的操作时的性能，而且LinkedList包含的操作也多于ArrayList。

HashSet、TreeSet和LinkedHashSet都是Set类型，输出显示在Set中。HashSet是使用的是相当复杂的方式来存储元素，这种技术是最快的获取元素方式，因此，存储的顺序看起来并无实际意义。如果存储顺序很重要，那么可以使用TreeSet，它按照比较结果的顺序保存对象；或者使用LinkedHashSet，它按照被添加的顺序保存对象。

不必指定Map的尺寸，因为它会自己自动的调整尺寸。Map还知道如何打印自己，它会显示相关联的键和值。键和值在Map中的保存顺序并不是他们的插入顺序，因为HashMap的实现使用的是一种非常快的算法来控制顺序。本例使用了三种基本风格的Map：HashMap、TreeMap和LinkedHashMap。与HashSet一样，HashMap也提供了最快的查找技术，也没有按照任何明显的顺序来保存其元素。TreeMap按照比较结果的升序保存键，而LinkedHashMap则按照插入顺序保存键，同时还保留了HashMap的查询速度。

## 11.5 List

List承诺可以将元素维持在特定的序列中。List接口在Collection的基础上添加了大量的方法，使得可以在List的中间插入和移除元素。

有两种类型的List：

* 基本的ArrayList，它长于随机访问元素，但是在List中间插入和移除元素较慢。
* LinkedList在随机访问方面较慢，但是它的特性集较ArrayList更大。

## 11.6 迭代器

对于List，add()可以插入元素，get()可以取出元素，但是只用这两个方法来遍历元素会发现有个缺点：要使用容器，必须对容器的确切类型编程。这好像没什么不好，但是如果原本是对着List编码的，但是后来发现如果能把相同的代码应用于Set，将会显得非常方便，那么如何才能不重写代码就能应用于不同类型的容器？

迭代器(也是一种设计模式)可以用于达成此目的。迭代器是一个对象，它的工作是遍历并选择序列中的对象，而客户端或者程序员不必知道或关心该底层的结构。

那么现在可以考虑创建一个display()方法，它不必知晓容器的确切类型：

public class Exercise {  
 public static void display (Iterator<String> it) {  
 while (it.hasNext()) {  
 Stirng str = it.next();  
 System.*out*.println(str);  
 }  
 }  
 public static void main (String[] arg) {  
 List<String> list = new ArrayList<String>();  
 Set<String> set = new HashSet<String>();  
 *display*(list.iterator());  
 *display*(set.iterator());  
 }  
}

### 11.6.1 ListIterator

ListIterator是一个更加强大的Iterator的子类型，它只能用于各种List类的访问。尽管Iterator只能向前移动。你可以通过调用listIterator()方法产生一个指向List开始处的ListIterator，并且还可以通过调用listIterator(n)方法创建一个一开始就指向列表索引为n的元素处的ListIterator。

## 11.7 LinkedList

LinkedList在执行插入和移除操作的时候比ArrayList更高效，但在随机访问操作方面却要逊色一些。

LinkedList还添加了可以使其用作栈、队列或双端队列的方法。

## 11.8 Stack

“栈”

public class Stack<T> {  
 private LinkedList<T> storage = new LinkedList<T>();  
 public void push(T v){storage.addFirst(v);}  
 public T peek(){return storage.getFirst();}  
 public T pop(){return storage.removeFirst();}  
 public boolean empty(){return storage.isEmpty();}  
 public String toString(){return storage.toString();}  
}

这里通过使用泛型，引入了在栈的类定义中最简单的可行示例。

## 11.9 Set

Set不保存重复元素(至于如何判断元素相同较为复杂，稍后会看到)。Set最常被使用的是测试某个对象是否在某个Set中，正因如此，查找成为了Set中最重要的操作，因此你会选择一个HashSet的实现，它专门对快速查找做了优化。

Set具有与Collection完全一样的接口，因此没有任何额外的功能。HashSet的输出顺序没有任何规律可循，这是因为处于速度的考虑，HashSet使用了散列。HashSet所维护的顺序与TreeSet和LinkedHashSet都不同，因为它们的实现具有不同的元素存储方式。TreeSet将元素存储在红-黑树数据结构中，而HashSet使用的是散列函数。LinkedHashList因为查询速度的原因也使用了散列，但是看起来它使用了链表来维护元素的插入顺序。

## 11.10 Map

## 11.11 Queue

# 第21章 并发

21.2.4 从任务中产生返回值

①如果希望任务在完成时返回一个值，那么可以实现Callable接口而不是Runnable接口。

②在Java SE5中引入的Callable是一个具有类型参数的泛型，它的类型参数表示的是从call()（而不是run()）中返回的值，并且必须使用ExecutorService.submit()方法调用它，代码如下：

ExecutorService exec=Executors.*newCachedThreadPool*();

ArrayList<Future<String>> results=**new** ArrayList<Future<String>>();

**for**(**int** i=0;i<10;i++){

results.add(exec.submit(**new** TaskWithResult(i)));

}

**for**(Future<String> fs:results){

**try** {

System.*out*.println(fs.get());

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

System.*out*.println(e);

e.printStackTrace();

} **catch** (ExecutionException e) {

// **TODO** Auto-generated catch block

System.*out*.println(e);

e.printStackTrace();

}**finally**{

exec.shutdown();

}

}

### 21.2.6 优先级

getPriority()获取当前线程的优先级。在任何时刻都可以通过setPriority()来修改优先级。

### 21.2.7 让步

yield()----暗示当前程序作出让步（只是一个暗示，没有任何机制保证它被采纳）。

### 21.2.8 后台线程

①所谓后台线程，是指在程序运行的时候在后台提供一种通用服务的线程，并且这种线程并不属于程序中不可或缺的部分。所以，非后台线程结束时，会杀死程序中的所有后台线程。

②必须在程序启动之前调用setDaemon()方法才能将其设置为后台线程。代码如下：

Thread daemon=new Thread(new SimpleDaemon());

daemon.setDaemon(true);

daemon.start();