Java.util包简单总结

大树先生 2017-02-28 00:30:13 35351 收藏 107

分类专栏： java 文章标签： java-util

版权

之前听一位老师讲过，学Java的程序员，lang包和util包至少是要过一遍的。

很惭愧的是，从入门到现在，我还没完整的探究过这两个基础包。

今天借着跟公司小伙伴分享的机会，把util包简单的梳理一遍。由于最近加班很多，此篇先做粗略总结，日后有时间再完善。

1. util包的框架

常用的集合类主要实现两个“super接口”而来：Collection和Map。

1.1 Collection有两个子接口：List和Set

List特点是元素有序，且可重复。实现的常用集合类有ArrayList、LinkedList，和Vector（线程安全）。

Set特点是元素无序，不可重复。实现的常用集合类有HashSet，LinkedHashSet，TreeSet（可排序）

1.2 Map是key、value键值对的集合

特点是key值无序不可重复，value值可重复（这样表述其实不太准确，因为实际上key和value是绑定在一起的）。常用的有HashMap，HashTable（线程安全），TreeMap（可排序）。

1.3 其余重要接口和类

上面是util包中的集合框架，一般Java教材里面都会讲到。但我们深入研究一下，会发现还有其余几个重要的内容：

Iterator：迭代接口

集合类实现该接口后便具有了迭代功能。最简单的迭代实现是ArrayList，迭代过程其实就是数组的迭代。LinkedList、LinkedHashSet和LinkedHashMap迭代过程就是链表的迭代。这两者的迭代效率都很高，迭代时间与容器里的元素数目成正比。但HashSet、HashMap迭代效率就略低了，因为采用了哈希表，所以元素是散列在数组中的，迭代时必须读完整个数组，迭代时间与容器的容量成正比。

Comparator：比较接口

实现该接口后，集合内元素便可比较通过compare()方法实现元素排序

AbstractXXX：骨架类

所谓骨架类，其实就是不同集合的核心代码实现，让继承这个抽象类的子类少干点活。例如AbstarctList代表“随机访问”集合（底层数组实现）的骨干代码实现。AbstractSequentialList代表“连续访问”（底层链表实现）集合的骨干代码实现。

Collections、Arrays

集合工具类和数组工具类。Java中的工具类好像都喜欢在对应的接口或类名称后，加S来表示其工具类。

接下来给一张比较完整的util包框架图：

2. 常用集合类原理

2.1 ArrayList

ArrayList的实现最简单，采用的顺序表，底层就是一个Object数组，初始容量为10，每当元素要超过容量时，重新创建一个更大的数组，并把原数据拷到新数组中来。

2.2 LinkedList

LinkedList采用双向链表。集合中的每一个元素都会有两个成员变量prev和next，分别指向它的前一元素和后一元素。

ArrayList和LinkedList的区别这里就不详细讨论了，其实就是顺序表和链表两种数据结构的区别。之前写的博文中已经提到（包括ArrayList和LinkedList的详细实现）：

数据结构基础（一）线性表

2.3 Vector

Vector底层实现和ArrayList类似，区别在于在许多方法上加了synchronized关键字，来实现了多线程安全。但代价是性能的降低。由于加锁的是整个集合，所以并发情况下进行迭代会锁住很长时间。

2.4 HashMap

HashMap采用的是哈希表结构，用链表法来解决hash冲突。这里不详细讨论，之前的文章写过：

HashMap原理解析

2.5 HashTable

HashTable的底层实现和HashMap类似，区别也是在许多方法上加了synchronized关键字，来实现了多线程安全。

2.6 LinkedHashMap

在HashMap的基础上加了双链表，该集合中的每个元素也都保留了前一个元素和后一个元素的“指针”。这样便可以按照插入顺序来读取集合元素。也可设置为按照访问顺序来读取集合元素。

由于要维护额外的双链表，LinkedHashMap增删操作会比HashMap慢，但迭代时会比HashMap快。

2.7 TreeMap

采用了红黑树数据结构，从而实现了有序集合。这个比较复杂，以后单独开出一篇来讨论，此处略。

2.8 HashSet、LinkedHashSet、TreeSet

Set和Map有千丝万缕的联系呀。例如HashSet底层实现其实就是一个固定value的HashMap。LinkedHashSet就是一个value固定的LinkedHashMap，TreeSet就是一个value固定的TreeMap。

3. 集合的并发

3.1 并发类的选择

讲到并发的集合，一般都想到util包中的两个类：HashTable和Vector。然而实际使用情况中，并不推荐使用这两个类。

首先，HashTable和Vector是从JDK1.0便存在的“古老”类，当时Collection、Map接口都还没。这样导致的问题是，当后来HashTable和Vector实现Map，Collection接口时，出现了许多无用而重复的方法。例如Vector原本有一个addElement()的方法，当实现了Collection接口后，又出现了一个add()方法。而实际上，这两个方法一模一样。

替代的这两个并发类的常见方法是Collections.synchronizedXXX(…)，这个方法可以把ArrayList，HashMap等集合变为线程安全的集合类。

那么，Vector和Collections.synchronizedXXX(…)的底层实现有什么区别呢？

我们来看看两者的add()方法实现：

//Vector

public synchronized boolean add(E e) {

modCount++;

ensureCapacityHelper(elementCount + 1);

elementData[elementCount++] = e;

return true;

}

//Collections.SynchronizedList

public void add(int index, E element) {

synchronized (mutex) {list.add(index, element);}

}

1

2

3

4

5

6

7

8

9

10

11

12

可以看出， 两者实现多线程的方式都是对集合的方法加锁，区别在于，Vector是对方法加锁，锁的是本对象，而Collections.synchronizedXXX(…)是对一个变量加锁。区别并不大。

那么，既然Collections.synchronizedXXX(…)比较好，用它创建出线程安全的集合类是不是就一劳永逸的满足我们所有的需求了呢？很不幸，不完全是。

Collections.synchronizedXXX(…)和HashTable、Vector在高并发时都有着很大的性能缺陷。因为它们的增、删、取都会锁住整个集合。想一想，一个线程在迭代十万个元素的Vector，其余线程对集合的操作时不时就阻塞了，受到了多大的影响啊。

为了解决这两种方法在高并发下的性能的低下。我们查找一下Java的API，发现在java.util.concurrent里面有许多针对高并发设计的类，例如：CopyOnWriteArrayList和ConcurrentHashMap。

ConcurrentHashMap的优化原理在于，采用了Segment的机制：

可以看成，ConcurrentHashMap底层每一个Segment都是一个HashMap,这样增删取时只需要锁住一段的Segment，而不是整个集合。从而优化了高并发下的性能。

CopyOnWriteArrayList主要是对高并发下的读、迭代做优化。实现原理在于每次add,remove操作都是重新创建一个新的数组，等操作结束再把引用指向新的数组。add,remove都是加了锁的，而get方法没有加锁，因为每次迭代时都是在旧的数组上迭代。所以CopyOnWriteArrayList适用于读多写少的并发场景。

3.2 迭代fail-fast机制

之前写的博文：Java迭代foreach原理解析（java.util.ConcurrentModificationException的原因）

————————————————

版权声明：本文为CSDN博主「大树先生」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/z55887/article/details/57587632