个人资料

commandingofficer

微博

加好友 发纸条
写留言 加关注

博客地图 world map

博客等级：18
博客积分：648
博客访问：348,878
关注人气：95
获赠金笔：8
赠出金笔：0
荣誉徽章：

相关博文

- 田馥甄阳台自拍假装在度假红唇齿突出从为
- 现代社会竞争中的“孙子兵法”军顺之光
- 【云南瑞丽】—这里的市场人人都妖姬是朵花-
- 一组精美的世界珍奇鸟类摄影作品用户375996083
- [霞浦]海上迷宫遐想无边小花rainy
- 《战狼2》后吴京的星途陈悠然--北京
- 好莱坞3500万表白吴京，吴影视听工作室
- 刚刚，重要文件下发！楼市“多晓博说财经
- 天王黎明今后运势展望张鑫龙风水命理阁

正文 字体大小：大 中 小

Java多线程 阻塞队列和并发集合

(2011-05-24 22:05:01) 转载▼

标签： 线程安全集合 阻塞队列 it 分类： java高级编程与虚拟计算平台

本文由作者收集整理所得，作者不保证内容的正确行，转载请标明出处。

作者：关新全

Java多线程 阻塞队列和并发集合

本章主要探讨在多线程程序中与集合相关的内容。在多线程程序中，如果使用普通集合往往会造成数据错误，甚至造成程序崩溃。Java为多线程专门提供了特有的线程安全的集合类，通过下面的学习，您需要掌握这些集合的特点是什么，底层实现如何、在何时使用等问题。

3.1 BlockingQueue接口

java阻塞队列应用于生产者消费者模式、消息传递、并行任务执行和相关并发设计的大多数常见使用上下文。

BlockingQueue在Queue接口基础上提供了额外的两种类型的操作，分别是获取元素时等待队列变为非空和添加元素时等待空间变为可用。

BlockingQueue新增操作的四种形式：

	抛出异常	特殊值	阻塞	超时
插入	add(e)	offer(e)	put(e)	offer(e, time, unit)
移除	remove()	poll()	take()	poll(time, unit)
检查	element()	peek()	不可用	不可用

插入操作是指向队列中添加一个元素，至于元素存放的位置与具体队列的实现有关。移除操作将会移除队列的头部元素，并将这个移除的元素作为返回值反馈给调用者。检查操作是指返回队列的头元素给调用者，队列不对这个头元素进行删除处理。

抛出异常形式的操作，在队列已满的情况下，调用add方法将会抛出IllegalStateException异常。如果调用remove方法时，队列已经为空，则抛出一个NoSuchElementException异常。（实际上，remove方法还可以附带一个参数，用来删除队列中的指定元素，如果这个元素不存在，也会抛出NoSuchElementException异常）。如果调用element检查头元素，队列为空时，将会抛出NoSuchElementException异常。

特殊值操作与抛出异常不同，在出错的时候，返回一个空指针，而不会抛出异常。

阻塞形式的操作，调用put方法时，如果队列已满，则调用线程阻塞等待其它线程从队列中取出元素。调用take方法时，如果阻塞队列已经为空，则调用线程阻塞等待其它线程向队列添加新元素。

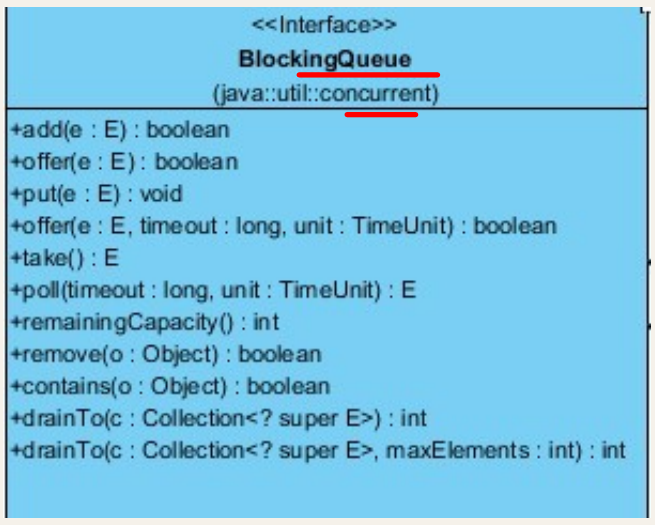
超时形式操作，在阻塞的基础上添加一个超时限制，如果等待时间超过指定值，抛出InterruptedException。

阻塞队列实现了Queue接口，而Queue接口实现了Collection接口，因此BlockingQueue也提供了remove(e)操作，即从队列中移除任意指定元素，但是这个操作往往不会按预期那样高效的执行，所以应当尽量少的使用这种操作。

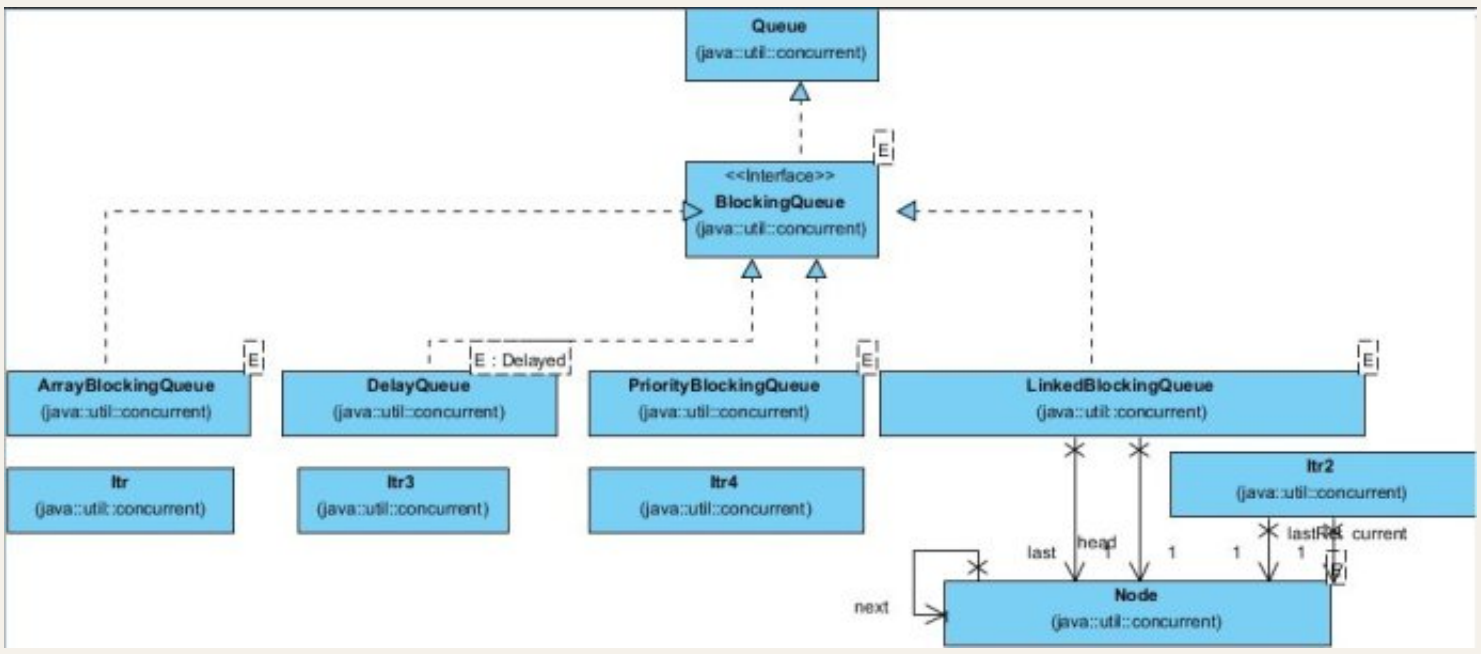
阻塞队列与并发队列（例如ConcurrentLinkQueue）都是线程安全的，但使用的场合不同。

Graphic3-1给出了阻塞队列的接口方法，Graphic3-2给出了阻塞队列的实现类结构。

Graphic 3-1 BlockingQueue接口



Graphic3-2阻塞队列的实现类



3.1.1 ArrayBlockingQueue类

列头部元素是队列中存在时间最长的元素，队列尾部是存在时间最短的元素，新元素将会被插入到队列尾部。队列从头部开始获取元素。

ArrayBlockingQueue是“有界缓存区”模型的一种实现，一旦创建了这样的缓存区，就不能再改变缓冲区的大小。ArrayBlockingQueue的一个特点是，必须在创建的时候指定队列的大小。当缓冲区已满，则需要阻塞新增的插入操作，同理，当缓冲区已空需要阻塞新增的提取操作。

ArrayBlockingQueue是使用的是循环队列方法实现的，对ArrayBlockingQueue的相关操作的时间复杂度，可以参考循环队列进行分析。

3.1.2 LinkedBlockingQueue

一种通过链表实现的阻塞队列，支持先进先出。队列的头部是队列中保持时间最长的元素，队列的尾部是保持时间最短的元素。新元素插入队列的尾部。可选的容量设置可以有效防止队列过于扩张造成系统资源的过多消耗，如果不指定队列容量，队列默认使用Integer.MAX_VALUE。LinkedBlockingQueue的特定是，支持无限（理论上）容量。

3.1.3 PriorityBlockingQueue

PriorityBlockingQueue是一种基于优先级进行排队的无界队列。队列中的元素按照其自然顺序进行排列，或者根据提供的Comparator进行排序，这与构造队列时，提供的参数有关。

使用提取方法时，队列将返回头部，具有最高优先级（或最低优先级，这与排序规则有关）的元素。如果多个元素具有相同的优先级，则同等优先级间的元素获取次序无特殊说明。

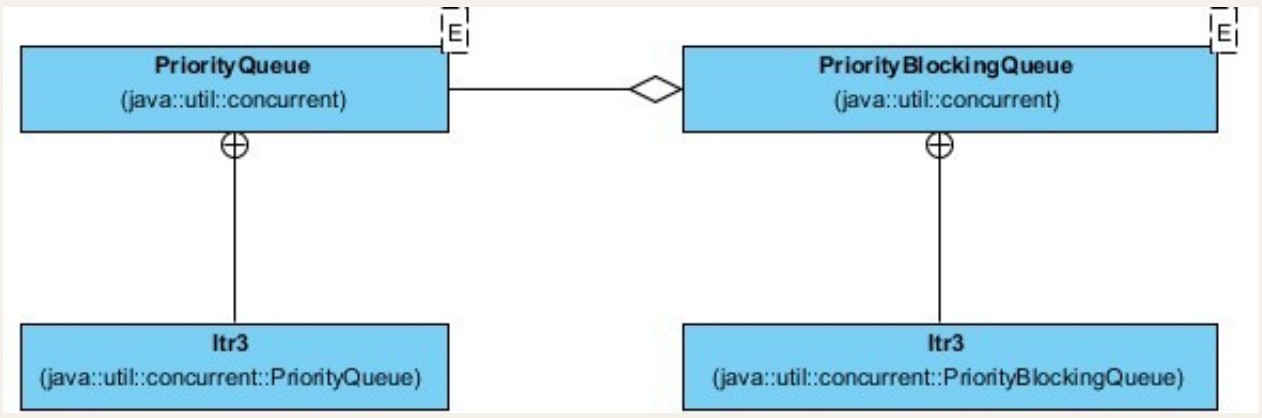
优先级队列使用的是一种可扩展的数组结构，一般可以认为这个队列是无界的。当需要新添加一个元素时，如果此时数组已经被填满，优先队列将会自动扩充当前数组（一般认为是，先分配一个原数组一定倍数空间的数组，之后将原数组中的元素拷贝到新分配的数组中，释放原数组的空间）。

如果使用优先级队列的iterator变量队列时，不保证遍历次序按照优先级大小进行。因为优先级队列使用的是堆结构。如果需要按照次序遍历需要使用Arrays.sort(pq.toArray())。关于堆结构的相关算法，请查考数据结构相关的书籍。

在PriorityBlockingQueue的实现过程中聚合了PriorityQueue的一个实例，并且优先队列的操作完全依赖与PriorityQueue的实现。在PriorityQueue中使用了一个一维数组来存储相

关的元素信息。一维数组使用最小堆算法进行元素添加。

Graphic3-3PriorityBlockingQueue的类关系



3.1.4 DelayQueue

一个无界阻塞队列，只有在延时期满时才能从中提取元素。如果没有元素到达延时期，则没有头元素。

3.2 并发集合

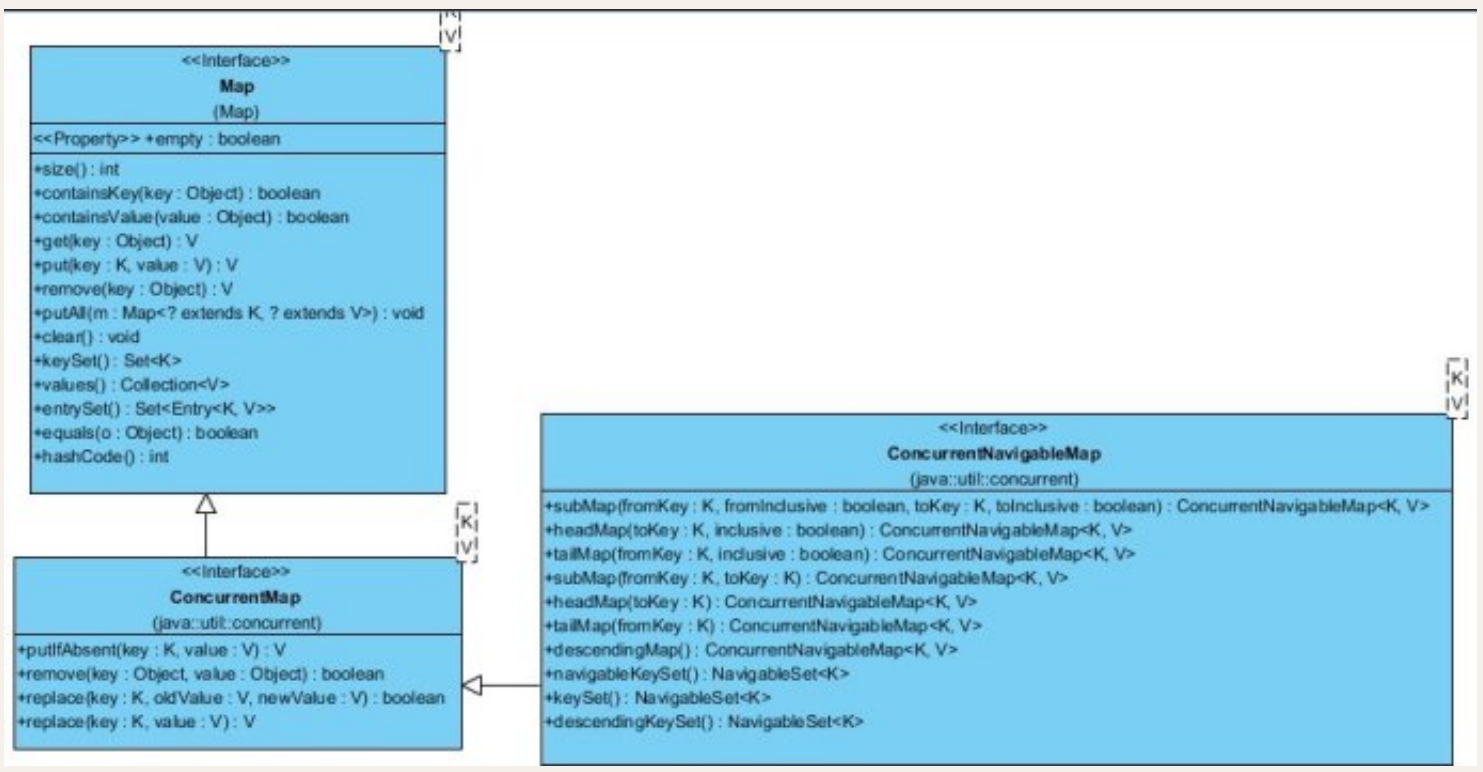
在多线程程序中使用的集合类，与普通程序中使用的集合类是不同的。因为有可能多个线程同时访问或修改同一集合，如果使用普通集合，很可能造成相应操作出现差错，甚至崩溃。Java提供了用于线程访问安全的集合。（前面讨论的BlockingQueue也是这里集合中的一种）。下面针对这些集合，以及集合中使用的相应算法进行探讨。在设计算法时，仅对相应算法进行简要说明，如果读者需要深入了解这些算法的原理，请参考其他的高级数据结构相关的书籍。

3.2.1 ConcurrentMap接口

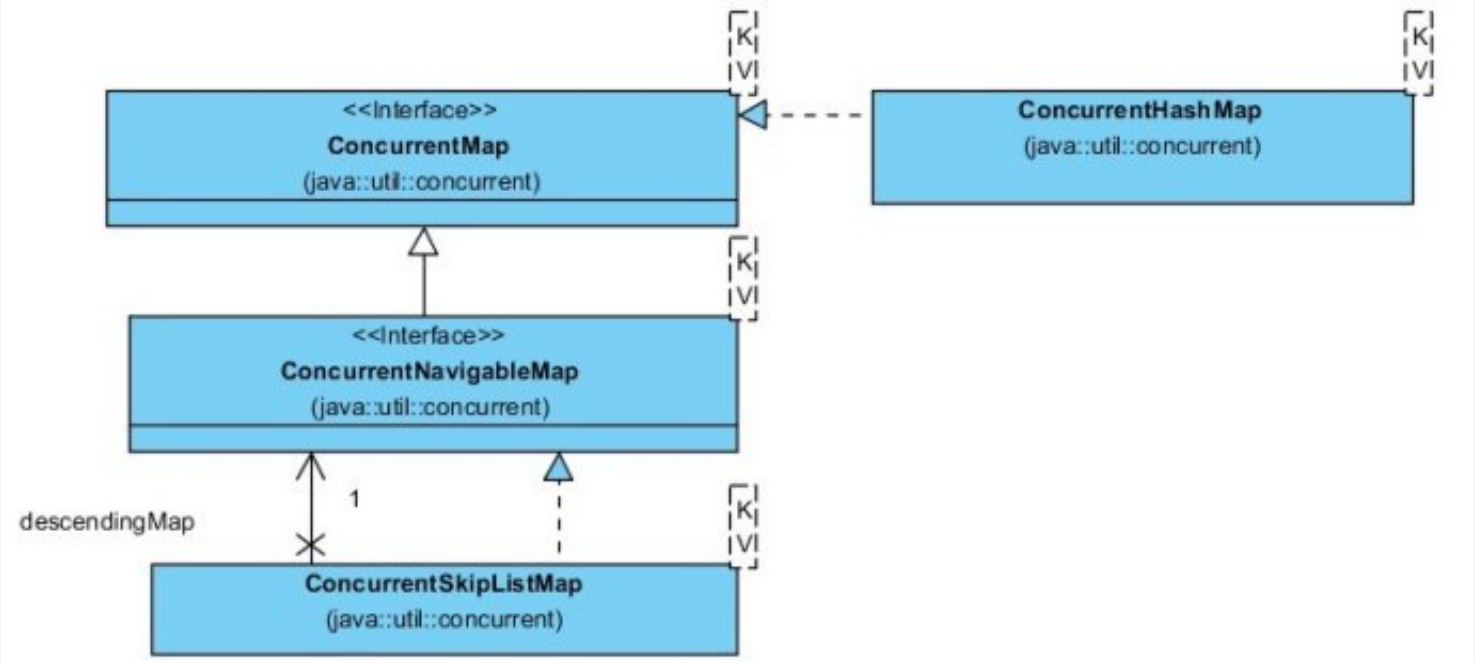
ConcurrentMap接口在Map接口的基础上提供了一种线程安全的方法访问机制。ConcurrentMap接口额外提供了多线程使用的四个方法，这四个方法实际是对Map已有方法的一个组合，并对这种组合提供一种原子操作。Graphic3-4给出了ConcurrentMap相关的操作。Graphic3-5给出了ConcurrentMap的实现类关系图。

从Graphic3-5中可以看出ConcurrentNavigableMap继承自ConcurrentMap，ConcurrentNavigableMap是一种SortedMap，就是说，映射中的元素会根据键值进行排序的。在java.util类库中，有两个类实现了SortedMap接口，分别是TreeMap和ConcurrentSkipListMap。TreeMap使用的是红黑树结构。而ConcurrentSkipListMap使用作为底层实现的SkipList（翻译为跳表）数据结构。此外ConcurrentHashMap实现了ConcurrentMap接口，使用的是HashMap方法。

Graphic3-4 ConcurrentMap



Graphic3-5 实现ConcurrentMap接口。



3.2.1.1 TreeMap

尽管TreeMap不是线程安全的，但是基于其数据结构的复杂性和方便对比说明，还是在这里简单提一下。TreeMap实现了SortedMap接口。TreeMap使用的是红黑树（这是高等数据结构中的一种），在红黑树算法中，当添加或删除节点时，需要进行旋转调整树的高度。使用红黑树算法具有较好的操作特性，插入、删除、查找都能在O(log(n))时间内完成。红黑树理论和实现是很复杂的，但可以带来较高的效率，因此在许多场合也得到了广泛使用。红黑树的一个缺陷在于，可变操作很可能影响到整棵树的结构，针对修改的局部效果不好。相关算法请参考http://blog.sina.com.cn/s/blog_616e189f0100qgcm.html。

TreeMap不是线程安全的，如果同时有多个线程访问同一个Map，并且其中至少有一个线程从结构上修改了该映射，则必须使用外部同步。可以使用Collections.synchronizedSortedMap方法来包装该映射。（注意使用包装器包装的SortMap是线程安全的，但不是并发的，效率上很可能远远不及ConcurrentSkipListMap，因此使用包装器的方法并不十分推荐，有人认为那是一种过时的做法。包装器使用了锁机制控制对Map的并发访问，但是这种加锁的粒度可能过大，很可能影响并发度）。

3.2.1.2 ConcurrentSkipListMap

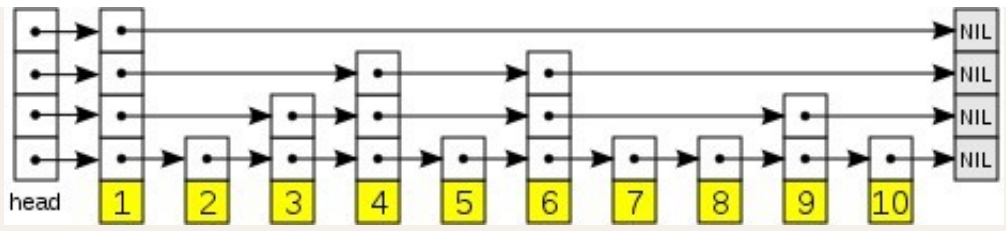
另外一种实现了SortedMap接口的映射表是ConcurrentSkipListMap。ConcurrentSkipListMap提供了一种线程安全的并发访问的排序映射表。SkipList（跳表）结构，在理论上能够在O(log(n))时间内完成查找、插入、删除操作。SkipList是一种红黑树的替代方案，由于SkipList与红黑树相比无论从理论和实现都简单许多，所以得到了很好的推广。SkipList是基于一种统计学原理实现的，有可能出现最坏情况，即查找和更新操作都是O(n)时间复杂度，但从统计学角度分析这种概率极小。Graphic3-6给出了SkipList的数据表示示例。有关skipList更多的说明可以参考：http://blog.csdn.net/caoeryingzi/archive/2010/11/18/6018070.aspx 和 http://en.wikipedia.org/wiki/Skip_list 这里不在累述。希望读者自行学习。

使用SkipList类型的数据结构更容易控制多线程对集合访问的处理，因为链表的局部处理性比较好，当多个线程对SkipList进行更新操作（指插入和删除）时，SkipList具有较好的局部性，每个单独的操作，对整体数据结构影响较小。而如果使用红黑树，很可能一个更新操作，将会波及整个树的结构，其局部性较差。因此使用SkipList更适合实现多个线程的并发处理。在非多线程的情况下，应当尽量使用TreeMap，因为似乎红黑树结构要比SkipList结构执行效率略优（无论是时间复杂度还是空间复杂度，作者没有做够测试，只是直觉）。此外对于并发性相对较低的并行程序可以使用Collections.synchronizedSortedMap将TreeMap进行包装，也可以提供较好的效率。对于高并发程序，应当使用ConcurrentSkipListMap，能够提供更高的并发度。

所以在多线程程序中，如果需要对Map的键值进行排序时，请尽量使用ConcurrentSkipListMap，可能得到更好的并发度。

注意，调用ConcurrentSkipListMap的size时，由于多个线程可以同时对映射表进行操作，所以映射表需要遍历整个链表才能返回元素个数，这个操作是个O(log(n))的操作。

Graphic3-6 SkipList示例



3.2.1.3 HashMap类

对Map类的另外一个实现是HashMap。HashMap使用Hash表数据结构。HashMap假定哈希函数能够将元素适当的分布在各桶之间，提供一种接近O(1)的查询和更新操作。但是如果需要对集合进行迭代，则与HashMap的容量和桶的大小有关，因此HashMap的迭代效率不会很高（尤其是你为HashMap设置了较大的容量时）。

与HashMap性能有影响的两个参数是，初始容量和加载因子。容量是哈希表中桶的数量，初始容量是哈希表在创建时的容量。加载因子是哈希表在容器容量被自动扩充之前，HashMap能够达到多满的一种程度。当hash表中的条目数超出了加载因子与当前容量的乘积时，Hash表需要进行rehash操作，此时Hash表将会扩充为以前两倍的桶数，这个扩充过程需要进行完全的拷贝工作，效率并不高，因此应当尽量避免。合理的设置Hash表的初始容量和加载因子会提高Hash表的性能。HashMap自身不是线程安全的，可以通过Collections的synchronizedMap方法对HashMap进行包装。

3.2.1.4 ConcurrentHashMap类

ConcurrentHashMap类实现了ConcurrentMap接口，并提供了与HashMap相同的规范和功能。实际上Hash表具有很好的局部可操作性，因为对Hash表的更新操作仅会影响到具体的某个桶（假设更新操作没有引发rehash），对全局并没有显著影响。因此ConcurrentHashMap可以提供很好的并发处理能力。可以通过concurrencyLevel的设置，来控制并发工作线程的数目（默认为16），合理的设置这个值，有时很重要，如果这个值设置的过高，那么很有可能浪费空间和时间，使用的值过低，又会导致线程的争用，对数量估计的过高或过低往往会带来明显的性能影响。最好在创建ConcurrentHashMap时提供一个合理的初始容量，毕竟rehash操作具有较高的代价。

3.2.2 ConcurrentSkipListSet类

实际上Set和Map从结构来说是很像的，从底层的算法原理分析，Set和Map应当属于同源的结构。所以Java也提供了TreeSet和ConcurrentSkipListSet两种SortedSet，分别适合于非多线程（或低并发多线程）和多线程程序使用。具体的算法请参考前述的Map相关介绍，这里不在累述。

3.2.3 CopyOnWriteArrayList类

CopyOnWriteArrayList是ArrayList的一个线程安全的变体，其中对于所有的可变操作都是通过对底层数组进行一次新的复制来实现的。

由于可变操作需要对底层的数据进行一次完全拷贝，因此开销一般较大，但是当遍历操作远远多于可变操作时，此方法将会更有效，这是一种被称为“快照”的模式，数组在迭代器生存期内不会发生更改，因此不会产生冲突。创建迭代器后，迭代器不会反映列表的添加、移除或者更改。不支持在迭代器上进行remove、set和add操作。CopyOnWriteArraySet与CopyOnWriteArrayList相似，只不过是Set类的一个变体。

3.2.3 Collections提供的线程安全的封装

Collections中提供了synchronizedCollection、synchronizedList、synchronizedMap、synchronizedSet、synchronize等方法可以完成多种集合的线程安全的包装，如果在并发度不高的情况下，可以考虑使用这些包装方法，不过由于Concurrent相关的类的出现，已经不这么提倡使用这些封装了，这些方法有些人称他们为过时的线程安全机制。

3.2.4 简单总结

提供线程安全的集合简单概括分为三类，首先，对于并发性要求很高的需求可以选择以Concurrent开头的相应的集合类，这些类主要包括：ConcurrentHashMap、ConcurrentLinkedQueue、ConcurrentSkipListMap、ConcurrentSkipSet。其次对于可变操作次数远远小于遍历的情况，可以使用CopyOnWriteArrayList和CopyOnWriteArraySet类。最后，对于并发规模比较小的并行需求可以选择Collections类中的相应方法对已有集合进行封装。

此外，本章还对一些集合类的底层实现进行简单探讨，对底层实现的了解有利于对何时使用何种方式作出正确判断。希望大家能够将涉及到原理（主要有循环队列、堆、HashMap、红黑树、SkipList）进行仔细研究，这样才能更深入了解Java为什么这样设计类库，在什么情况使用，应当如何使用。