1. Java集合框架
2. 队列的接口与实现
3. 接口与实现分离

Java集合类库将接口与实现分离。如队列接口指出可以在队列的队尾添加元素，在队列的头部删除元素，并且可以查找队列中的元素个数。

2， 实现方式

队列有两种实现方式，一种是使用循环数组，ArrayDeque类；另一种是使用链表，LinkedList类。数组是一个有界集合，链表是容量无上限，这是两者关键的差异。

1. Collection接口

Java类库中，集合类的基本接口是Collection接口，这个接口有两个基本方法：add(), iterator()。

1. add方法

add方法用于向集合中添加元素，如果添加元素改变了集合就返回true。

1. iterator方法

返回一个实现了iterator接口的对象，可以使用这个迭代器对象依次访问集合中的元素。

1. Iterator接口

Iterator接口中包含4个方法，如下所示：

1. next方法

逐个访问集合中的每个元素，如果到达了集合的末尾，next方法会抛出一个NoSuchElementException，因此每次调用next方法前都要使用hasNext方法检查是否到达了集合末尾。

1. hasNext方法

如果迭代器对象还有多个课可以访问的元素，则返回true。

1. remove方法

删除上一次调用next方法时返回的元素，注意，Java的迭代器可以理解为处于两个元素之间，当调用了next方法后，迭代器就会越过返回的元素，而remove方法此时删除的就是迭代器之前的那个元素。而且，每次调用remove方法前，都必须要调用next方法来让迭代器越过那个要删除的元素，否则就会抛出IllegalStateException异常。

Iterator<String> it = c.iterator();

it.next();

it.remove();

1. forEachRemaining方法

调用该方法并提供一个lambda表达式，将会对迭代器的每一个元素调用这个lambda表达式。

iterator.forEachRemaining(element -> do something with elemeng);

注意，此时的访问元素的顺序是取决于集合类型的，如果是一个ArrayList，则是顺序访问；若是一个HashSet，则是随机访问的。

for each 循环可以从处理任何实现了Iterator接口的对象。

Java 的迭代器只能根据next方法来向前移动，不像C++的迭代器那样根据数组索引建模的，可以随意向前向后移动。

1. 泛型使用方法

Collection和Iterator都是泛型接口，因此我们可以编写处理任何集合类型的实用方法，如检查任意集合中是否包含指定元素，这些方法对于用户使用集合是非常高效的。

然而，如果让Collection接口声明这些实用方法，再让所有实现Collection接口的实现类都实现这些方法，工作量很大。为了能够更容易地实现这些实用方法，Java类库提供了一个类AbstractCollection，实现了常用的实用方法。这样，具体的集合类就可以扩展AbstractCollection类，直接使用其中的实用方法。

现在更好的做法是将这些方法作为Collection接口的默认方法。

1. Java集合框架中的接口
2. 基本接口

集合中有两个基本接口，一个是Collection，另一个是Map,Map接口比较特殊，包含的是键值对，因此在元素的插入中使用put方法插入键值对，使用get方法获取键对应的值。

1. List接口

List是一个有序集合，元素会添加到容器中的特定位置，可以使用迭代器（顺序访问）或者整数索引（随机访问）的方式访问元素。有序集合的实现方式有两种，一种是数组，另一种是链表，链表的随机访问非常慢。我们可以使用一个Java1.4中引入的标记接口RandomAccess来测试一个特定的集合是否支持高效的随机访问：

if(c instanceof RandomAccess)

{

use random access algorithm;

}

else

{

use sequential access algorithm;

}

1. Set 接口

add方法不允许增加重复的元素；

equals方法不要求两个Set的元素排序相同，因为Set集合中元素的排序时任意的；

hashCode方法只要两个Set集合中元素相同，就会得到相同的散列码。

1. 具体集合

除了以Map结尾的类以外，其他类都实现了Collection接口，以Map结尾的类实现了Map接口。

1. 链表LinkedList
2. 双向链接

链表将每个对象存放在单独的链接中，每个链接中还存放着序列中下一个链接的引用，在Java中，所有链表都是双向链接的，还存放着前驱的引用。

1. 迭代器

链表是一个有序集合，add方法能够将对象添加到链表的尾部：

var staff = new LinkedList<String>();

staff.add(“amy”);

staff.add(“bob”);

当我们需要将对象添加到链表的任意位置时，需要迭代器的帮助，而Iterator接口中没有add方法，因为只有对自然有序的集合使用迭代器才是有意义的，而Set集合是无序的。

集合类库中提供了一个子接口ListIterator，其中包含add方法，该接口还可以反向遍历链表：

interface ListIterator<E> extends Iterator<E>

{

void add(E element);

E previous();

Boolean hasPrevious();

}

1. set方法

set方法用一个新元素替换调用next或者previous方法返回的上一个元素。如下所示，刚替换了刚刚返回的next()方法的值：

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

String oldValue = iter.next();

iter.set(newValue);

1. 迭代器关联规则

可以为一个集合关联多个迭代器，前提是这些迭代器只能读取集合，或者再关联一个可同时读写的迭代器。集合可以跟踪更改操作，每个迭代器都会为它负责的更改操作维护一个单独的更改操作数，如果该迭代器的更改操作数与集合的更改操作数不等，则抛出Concurrent ModificationException异常。

链表只跟踪对于列表的结构性修改，如添加或者修改链接，set方法不被视为结构性修改，因此迭代器使用set方法来修改现有链接的内容。

1. get方法

访问某个特定元素，效率很低，因为链表中每次都要从列表的头部开始搜索，根本没有缓存位置信息。

而如果通过迭代器就可以获取特定位置的迭代器，如list.listIterator(n)就可以返回一个迭代器，该迭代器指向索引为n的元素前面的位置，只要调用next方法就可以得到和llist.get(n)相同的元素，只是获取迭代器的效率比较低。

1. 数组列表ArrayList

List接口用于描述一个有序集合，ArrayList类同样实现了List接口，通过动态再分配的对象数组的方式实现有序集合。但是，对于动态数组还有一种方式就是vector，这是一个线程安全的动态数组，ArrayList方法不是同步的，但是正常情况下都是通过一个线程来访问一个数组，因此ArrayList是更常用的。

1. 散列集HashSet

散列表能够快速地查找对象，为每个对象计算一个整数，称为散列码，不同的对象产生不同的散列码。在Java中，散列表用链表数组实现，每个列表称为桶，一个桶就是一个链表数组。首先计算每个对象的散列码，然后与桶的数量取余，得到的余数就是这个对象要保存的桶的位置。若桶已经被填充了，即出现了散列冲突，就需要将该对象与桶中的所有对象进行比较，查看该对象是否已经存在。

桶数是桶的数目，如果大致知道最终会有多少个元素要插入到散列表中，可以设置桶数为预计元素个数的75% - 150%。当散列表过于满时，需要再散列，创建一个桶数更多的表，将所有元素插入到这个新表中，然后丢弃原有的表。装填因子用于确定何时对应散列表进行再散列，即散列表的饱和度。

HashSet类实现了基于散列表的集，可以使用add方法添加元素，使用contains方法查看某个元素是否已经在集中，且只需要通过计算散列码，查看一个桶中的元素即可，不需要查看整个集合中的所有元素。iterator方法返回迭代器，依次访问所有的桶。

1. 树集TreeSet

树集是一个有序集合，可以以任意顺序将元素插入到集合中，在对集合遍历时，值将自动地按照排序后的顺序呈现，排序是使用红黑树的结构实现的，每次插入一个新的元素，都会放置在正确的排序位置上。如果数据不需要排序，使用HashSet会更快一些，且要使用TreeSet的话，存储的对象就必须进行排序比较，实现Comparble接口。

从Java 6开始，TreeSet实现了NavigableSet接口，这个接口增加了几个查找元素的方法以及反向遍历的方法，如查找最大最小元素，删除最大最小元素，细节请查询源码。

var sorter = new TreeSet<String>();

sorter.add(“amy”);

sorter.add(“bob”);

1. 队列Queue与双端队列Deque

队列能够在尾部添加元素，头部删除元素；双端队列可在尾部和头部添加，删除元素。Java6中引入了Deque接口，ArrayDeque和LinkedList类实现了这个接口，都能够提供双端队列，大小可以根据需要扩展。

队列Queue：

add和offer方法能够添加元素到队列中，offer更加安全，当队列满了时，返回null不会出现异常；

remove和poll方法删除队列的队头的元素并返回，同样的，poll方法更加安全，当队列为空时，返回null不会出现异常，

element和peek方法返回队列队头元素，但不删除，同理，peek更安全。

双端队列Deque的API与队列Queue类似，只不过多了队头和队尾两种方式，不过多介绍。

1. 优先队列PriorityQueue

优先队列中元素可以按照任意顺序插入，按照有序的顺序检索，注意，并不是所有的元素都是有序的，而只是能够获取最小的元素。无论何时调用remove方法，总会获得当前优先队列中最小的元素。

不过，优先队列并没有对所有元素排序，而是使用堆这个数据结构，可以通过添加和删除操作让最小的元素移动到根节点，堆的这个特点可以应用于任务调度中，每次删除优先级最高的任务。