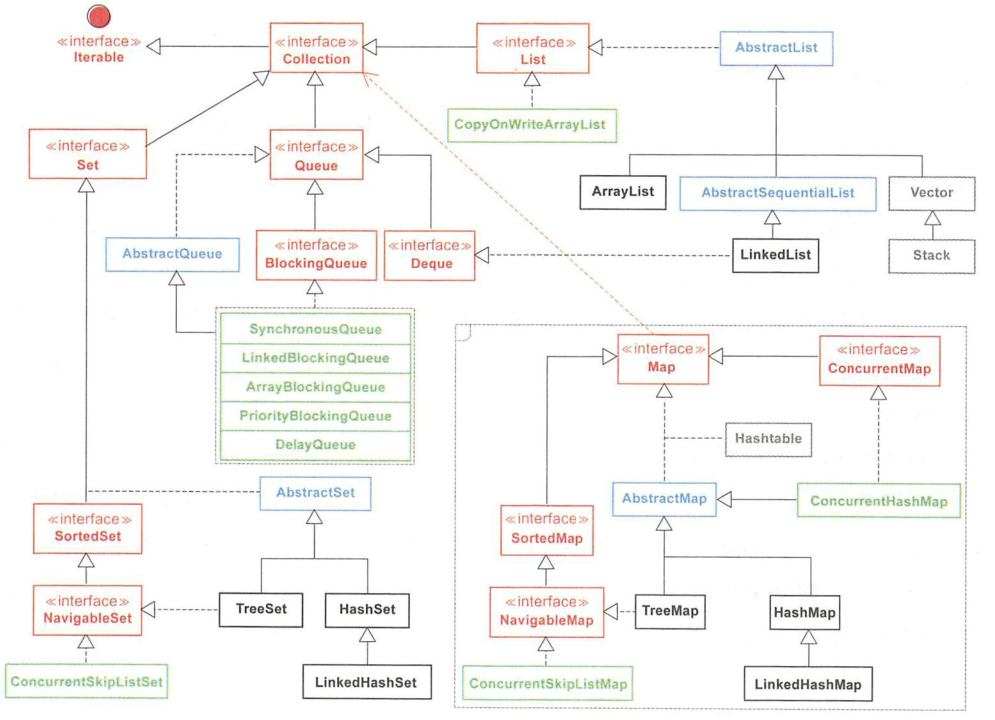
**集合**

集合就是一个放数据的容器，准确的说是放数据对象引用的容器。集合存放的都是对象的引用，而非对象本身。所以我们称集合中的对象就是集合中对象的引用。

**数组和集合的区别**

数组既可以存储基本数据类型,又可以存储引用数据类型,基本数据类型存储的是值,引用数据类型存储的是地址值, 长度是固定的,不能自动增长

集合只能存储引用数据类型(对象)，如果存储基本数据类型时，会自动装箱变成相应的包装类,集合的长度的是可变的,可以根据元素的增加而自动增长

****

**List集合**

List集合是线性数据结构的主要实现，集合元素通常存在明确的上一个和下一个元素，也存在明确的第一个元素和最后一个元素。List集合的遍历结果是稳定的。

**ArrayList集合**

ArrayList是容量可以改变的非线程安全集合。内部实现使用数组进行存储，集合扩容时会创建更大的数组结构，把原有数据复制到新数组中。ArrayList支持对元素的快速随机访问，但插入与删除时速度通常很慢，因为这个过程很有可能需要移动其他元素。

**LinkedList集合**

LinkedList的本质是双向链表。与ArrayList相比，LinkedList的插入与删除速度更快，随机访问速度很慢。除继承AbstractList抽象类外，LinkedList还实现了Deque。这个接口同时具有队列和栈的性质。LinkedList包含3个重要成员：size、first、last。size是双向链表中节点的个数。first和last分别指向第一个和最后一个节点的引用。LinkedList的优点在于可以将零散的内存单元通过附加引用的方式关联起来，形成按链路顺序查找的线性结构，内存利用率较高。

**Queue集合**

Queue是一种先进先出的数据结构，队列是一种特殊的线性表，它只允许在表的一端进行获取操作，在表的另一端进行插入操作。当队列中没有元素时，称为空队列。自从BlockingQueue（阻塞队列）问世以来，队列的地位得到极大的提升，在各种高并发编程场景中，由于其本身FIFO的特性和阻塞操作的特点，经常被作为Buffer（数据缓冲区）使用。

**Map集合**

Map集合是以Key-Value键值对作为存储元素实现的哈希结构，Key按某种哈希函数计算后是唯一的，Value则是可以重复的。最早用于存储键值对的Hashtable因为性能瓶颈已经被淘汰，而如今广泛使用的HashMap，线程是不安全的。ConcurrentHashMap是线程安全的，在JDK8中进行了锁的大幅度优化，体现出不错的性能。在多线程并发环境中，优先推荐使用ConcurrentHashMap，而不是HashMap。TreeMap是Key有序的Map类集合。

**Set集合**

Set是不允许出现重复元素的集合类型。Set体系最常用的是HashSet、TreeSet和LinkedHashSet三个集合类。HashSet从源码分析是使用HashMap实现的，只是value固定为一个静态对象，使用Key保证集合元素的唯一性，但不保证集合元素的顺序。TreeSet也是如此，从源码分析是使用TreeMap来实现的，底层为树结构，在添加新元素到集合中时，按照某种比较规则将其插入合适的位置，保证插入后的集合仍然是有序的。LinkedHashSet继承自HashSet，具有HashSet的优点，内部使用链表维护了元素插入顺序。

**尽量显式的的设定集合容量的初始大小，集合的扩容会造成性能损耗**

**集合的容量不会在new的时候分配，而在第一次加入数据时分配**

**ArrayList默认大小为10，HashMap默认大小为16**

**并发集合与普通集合的区别**

**在Java中，有普通集合、同步的集合（即线程安全的集合）、并发集合。**

并发集合常见的有ConcurrentHashMap、ConcurrentLinkedQueue、ConcurrentLinkedDeque等。并发集合位于java.util.concurrent包下，是在JDK1.5之后才有的。

普通集合通常性能最高，但是不保证多线程的安全性和并发的可靠性；

线程安全集合仅仅是给集合添加了synchronized（同步的）同步锁，严重影响了性能，而且对并发的效率就更低了；

并发集合通过复杂的策略不仅保证了多线程的安全，又提高了并发时的效率。

**并发集合类是什么？**

Java1.5并发包（java.util.concurrent）包含线程安全集合类，允许在迭代时修改集合。迭代器被设计为fail-fast的，会抛出ConcurrentModificationException。

**Collection跟Collections的区别**

Collections是个java.util下的类，它包含有各种有关集合操作的静态方法。

Collection是个java.util下的接口，它是各种集合结构的父接口。

**Collection是所有集合框架的父接口么？**

不是，map和collection的数据结构不一样，collection定义了其实现类的基本操作，map也定义了其实现类的基本操作，操作不同，所以接口不同，而这正是因为接口是种规范。

**Collection接口**

**List：里面存放的数据是有顺序的，可以存放重复的数据。  
Set：里面存放的数据是没有顺序的，不能存放重复的数据。  
Queue：是一个队列，里面的数据是先进先出，可以存放重复的数据。**

**Map接口**

**Map：里面存放的数据是没有顺序的, 其键是不能重复的，它的值是可以有重复的**。

**List的三个子类的特点**

**ArrayList:**  
底层数据结构是数组，查询快，增删慢。  
线程不安全，效率高。

**Vector:**  
底层数据结构是数组，查询快，增删慢。  
线程安全，效率低。

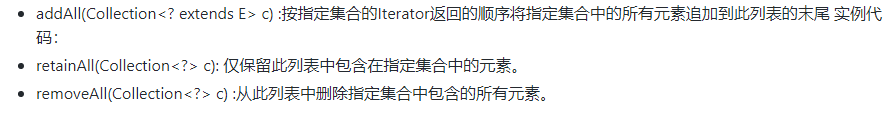
**LinkedList:**  
底层数据结构是链表，查询慢，增删快。  
线程不安全，效率高。

**这个三个子类有什么区别和共同点**

**Vector和ArrayList的区别**  
Vector是线程安全的,效率低  
ArrayList是线程不安全的,效率高  
共同点:都是数组实现的

**ArrayList和LinkedList的区别**  
ArrayList底层是数组结果,查询和修改快  
LinkedList底层是链表结构的,增和删比较快,查询和修改比较慢  
共同点:都是线程不安全的

**如何求ArrayList集合的交集 并集 差集 去重复并集**





**集合框架中的三种迭代方式删除数据**

**普通for循环：**可以删除,注意让索引做自减运算

**迭代器**：可以删除,但是必须使用迭代器自身的remove方法,否则会出现并发修改异常

**增强for循环：**不能删除

**Set的三个子类的特点**

**HashSet**

存储的元素不能重复，无序，HashSet中存放自定义类型对象时，一定要重写hashCode和equals方法

**LinkedHashSet**

底层使用了链表的数据结构，特点是读取元素的顺序跟存入元素的顺序是一致的，并且元素不能重复。

**TreeSet**

特点是可以对存放进去的元素进行排序，存储的元素不能重复。

**HashMap和Hashtable的区别**

Hashtable是JDK1.0版本出现的,是线程安全的,效率低，不可以存储null键和null值  
HashMap是JDK1.2版本出现的，可以存储null键和null值

**HashMap、HashTable、HashSet三者的区别**

**实现接口的不同**

HashMap,HashTable是Map接口的实现类，而HashSet是Set接口的实现类，而Set接口是继承Collection接口。

**线程安全性，同步**

HashTable中的方法加了同步锁（synchronized），所以对象是线程安全，而HashMap是异步的，所以存放的对象并不是线程安全的，而HashSet的底层是用HashMap实现的，所以它也不是线程安全的。

**执行效率**

HashTable是同步的，而HashMap是异步的，所以HashMap的执行效率比HashTable要高，三则之间执行效率的排序是：HashMap>HashSet>HashTable

**key,value能否存放NULL**

HashMap的key,value是可以为null,而HashTable是不能存放NULL，HashSet它存放的不是键值对，而是对象，也是可以为NULL

**添加元素的方法不同**

HashMap是通过put(),来增加元素的，而HashSet是通过add()方法来增加元素的

**HashMap 和 ConcurrentHashMap 的区别**

ConcurrentHashMap对整个桶数组进行了分割分段(Segment)，然后在每一个分段上都用lock锁进行保护，相对于HashTable的synchronized锁的粒度更精细了一些，并发性能更好，而HashMap没有锁机制，不是线程安全的。（JDK1.8之后ConcurrentHashMap启用了一种全新的方式实现,利用CAS算法。）

HashMap的键值对允许有null，但是ConCurrentHashMap都不允许。

**HashSet 如何检查重复**

当你把对象加入HashSet 时，HashSet 会先计算对象的 hashcode 值来判断对象加入的位置，同时也会与其他已经加入的对象的 hashcode 值作比较，如果没有相符的hashcode，HashSet会假设对象没有重复出现。但是如果发现有相同 hashcode 值的对象，这时会调用 equals（）方法来检查 hashcode 相等的对象是否真的相同。如果两者相同，HashSet 就不会让其加入操作成功。如果不同的话，就会重新散列到其他位置。这样我们就大大减少了 equals 的次数，相应就大大提高了执行速度。

**两个对象值相同(x.equals(y) == true) ，但却可有不同的 hash code，这句话对不对**

对。

如果对象要保存在 HashSet 或 HashMap 中，它们的 equals 相等，那么，它们的 hashcode

值就必须相等。

如果不是要保存在HashSet或HashMap，则与hashcode没有什么关系了，这时候hashcode

不等是可以的，例如 arrayList 存储的对象就不用实现 hashcode，当然，我们没有理由不实

现，通常都会去实现的。

**Iterator是什么？**

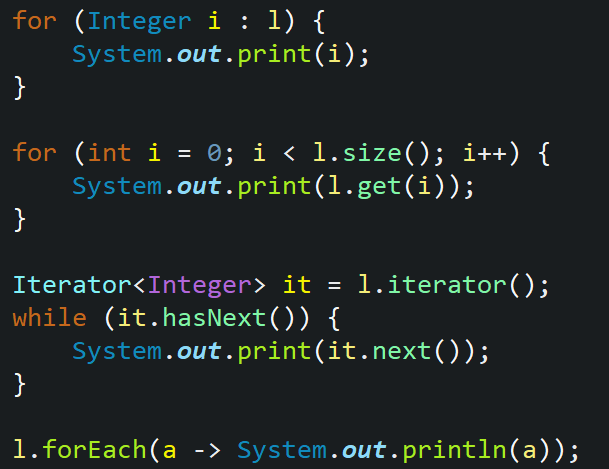
Iterator接口提供遍历任何Collection的接口。我们可以从一个Collection中使用迭代器方法来获取迭代器实例。迭代器取代了Java集合框架中的Enumeration。迭代器允许调用者在迭代过程中移除元素。它只有三个方法：hasNext()、next()、remove()

**Enumeration和Iterator接口的区别？**

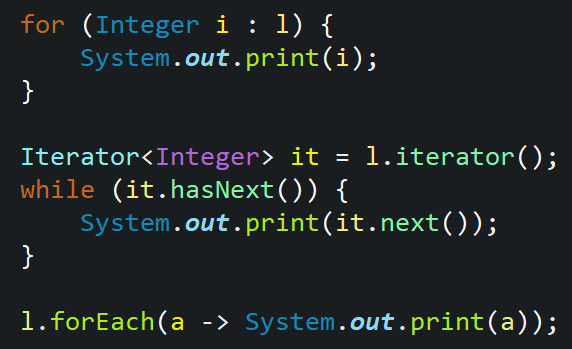
Enumeration的速度是Iterator的两倍，也使用更少的内存。Enumeration是非常基础的，也满足了基础的需要。但是，与Enumeration相比，Iterator更加安全，因为当一个集合正在被遍历的时候，它会阻止其它线程去修改集合。

迭代器取代了Java集合框架中的Enumeration。迭代器允许调用者从集合中移除元素，而Enumeration不能做到。为了使它的功能更加清晰，迭代器方法名已经经过改善。

**List遍历**

****

**Set遍历**

****

**Map遍历**

****

**四种迭代方式对比**

**普通for循环**  
此种方式在遍历ArrayList时效率会高一些，因为ArrayList底层使用的是数组实现的，所以可以认为ArrayList中的元素都是有下标的，而此种普通for循环中的变量i可以快速的定位到ArrayList中的元素。

**增强for循环和迭代器**  
可以认为增强for循环是迭代器的一种简便的写法，而迭代器比较适合遍历LinkedList，因为它底层使用的是链表的数据结构。

**使用forEach方法+lambda表达式**  
如果你使用的是jdk8以上的版本，那么建议使用此种方式，该方式内部默认的使用增强for循环去遍历集合，不过在ArrayList类中重写了forEach方法，里面使用了普通的for循环去遍历。不管你使用哪一种，这种方式底层会选择最优的遍历方式

**在迭代一个集合的时候，如何避免ConcurrentModificationException？**

在遍历一个集合的时候，我们可以使用并发集合类来避免ConcurrentModificationException，比如使用CopyOnWriteArrayList，而不是ArrayList。

**快速失败(fail-fast)和安全失败(fail-safe)**

**快速失败（fail—fast）**

**在用迭代器遍历一个集合对象时，如果遍历过程中对集合对象的内容进行了修改（增加、删除、修改），则会抛出Concurrent Modification Exception。**

原理：迭代器在遍历时直接访问集合中的内容，并且在遍历过程中使用一个 modCount 变量。集合在被遍历期间如果内容发生变化，就会改变modCount的值。每当迭代器使用hashNext()/next()遍历下一个元素之前，都会检测modCount变量是否为expectedmodCount值，是的话就返回遍历；否则抛出异常，终止遍历。

注意：这里异常的抛出条件是检测到 modCount！=expectedmodCount 这个条件。如果集合发生变化时修改modCount值刚好又设置为了expectedmodCount值，则异常不会抛出。因此，不能依赖于这个异常是否抛出而进行并发操作的编程，这个异常只建议用于检测并发修改的bug。

**java.util包下的集合类都是快速失败的，不能在多线程下发生并发修改（迭代过程中被修改）。**

**安全失败（fail—safe）**

**采用安全失败机制的集合容器，在遍历时不是直接在集合内容上访问的，而是先复制原有集合内容，在拷贝的集合上进行遍历。**

原理：由于迭代时是对原集合的拷贝进行遍历，所以在遍历过程中对原集合所作的修改并不能被迭代器检测到，所以不会触发Concurrent Modification Exception。

缺点：基于拷贝内容的优点是避免了Concurrent Modification Exception，但同样地，迭代器并不能访问到修改后的内容，即：迭代器遍历的是开始遍历那一刻拿到的集合拷贝，在遍历期间原集合发生的修改迭代器是不知道的。

**java.util.concurrent包下的容器都是安全失败，可以在多线程下并发使用，并发修改。**

**compareTo方法的返回值**

TreeSet使用了二叉树的数据结构，负数放到左边，正数放到右边。

当compareTo方法返回0的时候，系统会认为两者一致，所以不会向集合中添加元素

当compareTo方法返回正数的时候，系统将元素存储到右边，所以集合存取顺序一致,compare返回正数

当compareTo方法返回负数的时候，系统将元素存储到左边，所以集合会倒序存储

**Collections常用方法**

Collections.max() 最大值

Collections.reverse() 反转集合

Collections.shuffle(); 随机打乱

Collections.sort();排序

**数组与List的相互转换**



**集合框架中的泛型有什么优点？**

Java1.5引入了泛型，所有的集合接口和实现都大量地使用它。泛型允许我们为集合提供一个可以容纳的对象类型，因此，如果你添加其它类型的任何元素，它会在编译时报错。这避免了在运行时出现ClassCastException，因为你将会在编译时得到报错信息。泛型也使得代码整洁，我们不需要使用显式转换和instanceOf操作符。它也给运行时带来好处，因为不会产生类型检查的字节码指令。