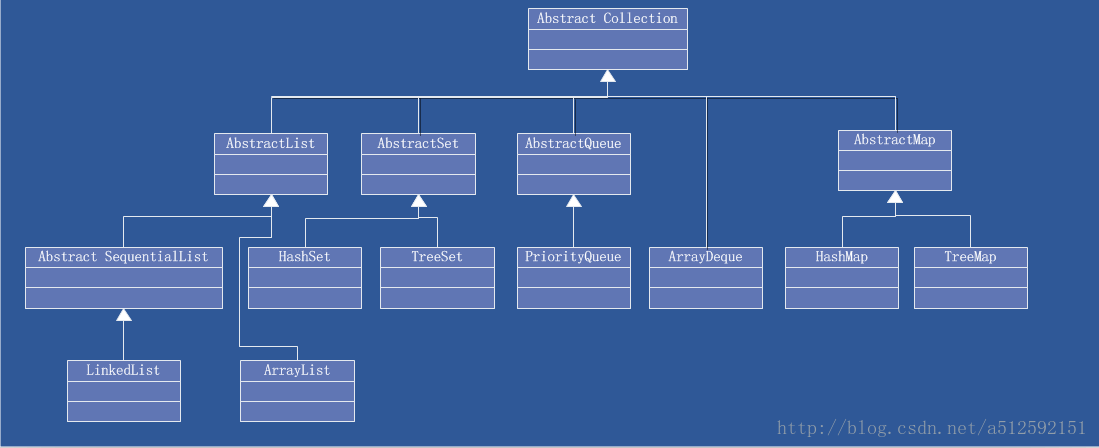
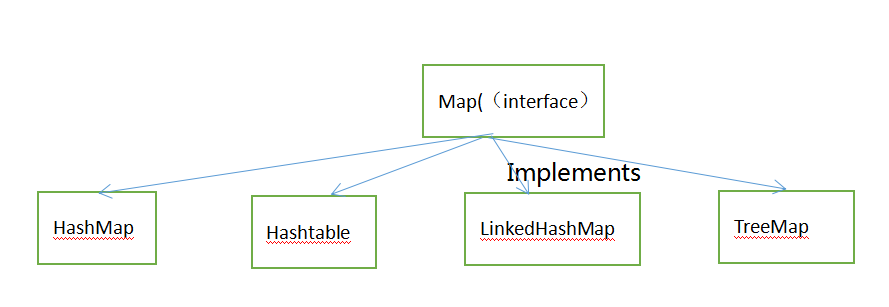
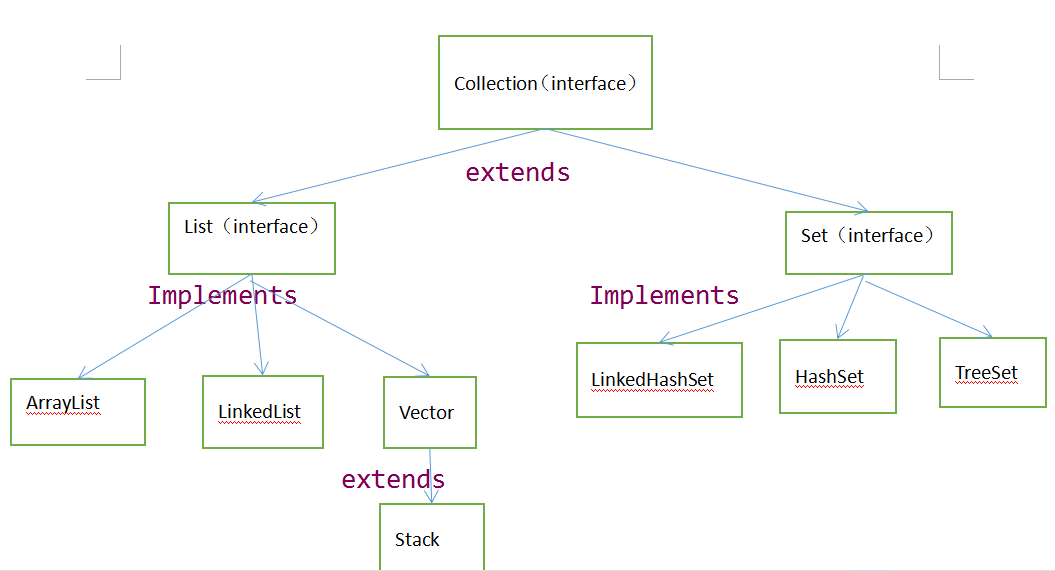
**前言：**数据结构对程序设计有着深远的影响，在面向过程的C语言中，数据库结构用struct来描述，而在面向对象的编程中，数据结构是用类来描述的，并且包含有对该数据结构操作的方法。

在Java语言中，Java语言的设计者对常用的数据结构和算法做了一些规范（接口）和实现（具体实现接口的类）。所有抽象出来的数据结构和操作（算法）统称为Java集合框架（Java Collection Framework）





容器不能持有基本类型，但是自动包装机制会仔细的执行基本类型到容器中所持有有的包装器类型之间的双向转换。

--ArrayList，底层采用的是数组存储元素，所以ArrayList对象适合作元素的查询操作，不适合作元素的增删。  
--LinkedList，底层采用的是链表存储元素，所以LinkedList对象适合作元素的增删，不适合作元素的查询。  
--Vector，底层采用的是数组存储元素。它还有一个特点，即线程安全的。所以Vector对象效率比较低，很少使用

* ArrayList 对于随机位置的add/remove，时间复杂度为 O(n),但是对于列表末尾的添加/删除操作,时间复杂度是 O(1).
* LinkedList对于随机位置的add/remove，时间复杂度为 O(n),但是对于列表 末尾/开头 的添加/删除操作,时间复杂度是 O(1).

**补充：**

**PriorityQueue：优先队列**

**当在PriorityQueue上调用offer()方法来插入一个对象时，这个对象会在队列中被排序。默认的排序将使用对象在队列中的自然排序，也可以通过提供自己的Comparator来修改这个顺序。**

实现Set接口的重要类有HashSet（无序不重复），LinkedHashSet（按放入顺序有序不重复），TreeSet（按红黑树方式有序不重复)

TreeSet采用红黑树的数据结构排序元素.   
HashSet采用散列函数，这是专门为快速查询而设计的.   
LinkedHashSet内部使用散列以加快查询速度，同时使用链表维护元素的次序.   
  
使用HashSet/TreeSet时，必须为类定义equals()；而HashCode()是针对HashSet，作为一种编程风格，当覆盖equals()的时候，就应该同时覆盖hashCode().

**1.Collection接口(不能实例化）**

Collection是最基本的*集合接口*，一个Collection代表一组Object，即Collection的元素（Elements）。一些 Collection允许相同的元素而另一些不行。一些能排序而另一些不行。JavaSDK不提供直接继承自Collection的类，JavaSDK提供的类都是继承自Collection的“子接口”如List和Set。

所有实现Collection接口的类都必须提供两个标准的构造函数：无参数的构造函数用于创建一个空的Collection，有一个 Collection参数的构造函数用于创建一个新的Collection，这个新的Collection与传入的Collection有相同的元素。后一个构造函数允许用户复制一个Collection。

如何遍历Collection中的每一个元素？不论Collection的实际类型如何，它都支持一个iterator()的方法，该方法返回一个迭代子，使用该迭代子即可逐一访问Collection中每一个元素。典型的用法如下：

Iterator it = collection.iterator();//获得一个迭代子

while(it.hasNext()){

Object obj = it.next();//得到下一个元素

}

由Collection接口派生的两个接口是List和Set。

List：

和数组类似，List可以动态增长，查找元素效率高，插入删除元素效率低，因为会引起其他元素位置改变。<相应类有 ArrayList,LinkedList，Vector>

List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组。

和下面要提到的Set不同，List允许有相同的元素。

除了具有Collection接口必备的iterator()方法外，List还提供一个listIterator()方法，返回一个 ListIterator接口，和标准的Iterator接口相比，ListIterator多了一些add()之类的方法，允许添加，删除，设定元素，还能向前或向后遍历。

实现List接口的常用类有LinkedList，ArrayList，Vector和Stack。

**LinkedList类**

LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在 LinkedList的首部或尾部。这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。

LinkedList 对顺序访问进行优化,向List 中间插入与移除的开销并不大，具有addFrist(),addLast(),getFirst,getLast,removeFirst和removeLast().这些方法使得LinkedList可当作堆栈／队列／双向队列.

注意LinkedList*没有同步方法。*如果多个线程同时访问一个List，则必须自己实现访问同步。一种解决方法是在创建List时构造一个同步的List：

List list = Collections.synchronizedList(new LinkedList(...));

**2.2.ArrayList类**

ArrayList实现了可变大小的数组。它允许所有元素，包括null。ArrayList没有同步。

size，isEmpty，get，set方法运行时间为常数。但是add方法开销为分摊的常数，添加n个元素需要O(n)的时间。其他的方法运行时间为线性。

每个*ArrayList实例*都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法并没有定义。当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率（自动增判断长度后增长也会浪费时间的呀！）。

和LinkedList一样，ArrayList也是非同步的（unsynchronized）。（扩展阅读：在java.util.concurrent包中定义的CopyOnWriteArrayList提供了线程安全的Arraylist，但是当进行add和set等变化操作时它是通过为底层数组创建新的副本实现的，所以比较耗费资源

**（源码在此：**

**public boolean** add(E e) {

**final** ReentrantLock  lock =**this**.lock;

lock.lock();

**try** {

Object[] elements = getArray();

**int** len = elements.length;

Object[] newElements = Arrays.copyOf(elements,len + 1);

newElements[len] = e;

setArray(newElements);

**return true**;

} **finally** {

lock.unlock();

}

}**）**，

但是如果存在频繁遍历，遍历操作比变化（写入和修改）操作多的时候这种遍历就相对于自己进行的同步遍历效果要好，而且它也允许存在null元素）

**Vector类**

Vector非常类似ArrayList，但是Vector是同步的。

由Vector创建的Iterator，虽然和ArrayList创建的 Iterator是同一接口，但是，因为Vector是同步的，当一个Iterator被创建而且正在被使用，另一个线程改变了Vector的状态（例如，添加或删除了一些元素），这时调用Iterator的方法时将抛出ConcurrentModificationException，因此必须捕获该异常。

**通过使用capacity和ensurecapacity操作以及capacityIncrement域可以优化存储操作，这个前面讲过**，

（**Vector的Iterator和listIterator方法翻译的迭代器支持fail-fast机制，因此如果在使用迭代器的过程中有其他线程修改了map，那么将抛出ConcurrentModificationException，这就是所谓fail-fast策略。**

**官方对此的说明是** java.util 包中的集合类都返回 fail-fast迭代器，这意味着它们假设线程在集合内容中进行迭代时，集合不会更改它的内容。如果 fail-fast迭代器检测到在迭代过程中进行了更改操作，那么它会抛出 ConcurrentModificationException，这是不可控异常。）

**2.4.Stack类**

Stack继承自Vector，实现一个后进先出的堆栈。Stack提供5个额外的方法使得Vector得以被当作堆栈使用。基本的push和pop方法，还有peek方法得到栈顶的元素，empty方法测试堆栈是否为空，search方法检测一个元素在堆栈中的位置。Stack刚创建后是空栈。

stack 有几个比较实用的方法

|  |  |
| --- | --- |
| boolean | **empty**()  测试堆栈是否为空。 |
| E | **peek**()  查看堆栈顶部的对象，但不从堆栈中移除它。 |
| E | **pop**()  移除堆栈顶部的对象，并作为此函数的值返回该对象。 |
| E | **push**([E](http://doc.okbase.net/admin/blogs/2105360/" \o "Stack 中的类型参数" \t "http://doc.okbase.net/DavidIsOK/archive/_blank)item) 把项压入堆栈顶部。 |
| int | **search**([Object](http://doc.okbase.net/admin/blogs/2105360/" \o "java.lang 中的类" \t "http://doc.okbase.net/DavidIsOK/archive/_blank) o) 返回对象在堆栈中的位置，以 1 为基数。 |

**Set接口：**

检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。<对应类有 HashSet,TreeSet>.

HashSet：以哈希表的形式存放元素，插入删除速度很快。

Set具有与Collection完全一样的接口，因此没有任何额外的功能，不像前面有两个不同的List。实际上Set就是Collection,只是行为不同。(这是继承与多态思想的典型应用：表现不同的行为)

Set不保存重复的元素

Set : 存入Set的每个元素都必须是唯一的，因为Set不保存重复元素。加入Set的元素必须定义equals()方法以确保对象的唯一性。Set与Collection有完全一样的接口。Set接口不保证维护元素的次序。

HashSet : 它不允许出现重复元素；不保证集合中元素的顺序，可以自己做个例子可以看出加入的字段顺序跟遍历出的不一样，允许包含值为null的元素，但最多只能有一个null元素（不允许重复嘛！）。

TreeSet : 可以实现排序等功能的集合，它在讲对象元素添加到集合中时会自动按照某种比较规则将其插入到有序的对象序列中，并保证该集合元素组成按照“升序”排列。

a)（在对大量信息进行检索的时候，TreeSet比AraayList更有效率，能保证在log(n)的时间内完成）。

b)TreeSet是实用树形结构来存储信息的，每个节点都会保存一下指针对象，分别指向父节点，左分支，右分支，相比较而言，ArrayList就是一个含有元素的简单数组了，正因为如此，它占的内存也要比ArrayList多一些。

c)想TreeSet插入元素也比ArrayList要快一些，因为当元素插入到ArrayList的任意位置时*，平均每次要移动一半的列表*，需要O(n)的时间， 而TreeSet深度遍历查询花费的实施只需要O(log(n))**（普遍的都是，set查询慢，插入快，list查询快，插入慢）**

LinkedHashSet : 具有HashSet的查询速度，且内部使用链表维护元素的顺序(插入的次序)。于是在使用迭代器遍历Set时，结果会按元素插入的次序显示。

**PS:set有几个比较好的方法:**

**removeAll(Collection<?> c)**

移除 set 中那些包含在指定 collection 中的元素（可选操作）。

**boolean retainAll(Collection<?> c)**

仅保留 set 中那些包含在指定 collection 中的元素（可选操作）。

**containsAll(Collection<?> c)**

如果此 set 包含指定 collection 的所有元素，则返回 true。

Queue数据结构

**这方面知识涉及到线程比较多，有线程基础的口语参考这篇文章**

**http://blog.csdn.net/a512592151/article/details/38454745**

**Map的功能方法**

java*为 数据结构中的映射* 定义了一个接口java.util.Map;它有四个实现类,分别是**HashMap Hashtable LinkedHashMap** 和**TreeMap**

Map主要用于存储健值对，根据键得到值，因此不允许键重复,但允许值重复。

HashMap是一个最常用的Map,它根据键的HashCode 值存储数据,根据键可以直接获取它的值，**具有很快的访问速度。**

HashMap最多只允许一条记录的键为Null;允许多条记录的值为 Null;。

在Map 中插入、删除和定位元素，HashMap 是最好的选择。

HashMap不支持线程的同步，即任一时刻可以有多个线程同时写HashMap;可能会导致数据的不一致。如果需要同步，可以用 Collections的synchronizedMap方法使HashMap具有同步的能力.HashMap继承自AbstractMap类。

Hashtable 与HashMap类似,不同的是:它不允许记录的键**或者值为空;**

它支持线程的同步，用于多线程。即任一时刻只有一个线程能写Hashtable,因此也导致了**Hashtale在写入时会比较慢。**,它继承自Dictionary类。

LinkedHashMap保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的.

在遍历 的时候会比HashMap慢。LinkedHashMap是HashMap的一个子类，如果需要输出的顺序和输入的相同,那么用LinkedHashMap可以实现。

TreeMap 能够把它保存的记录根据键**排序**,默认是按升序排序，也可以指定排序的比较器，当用Iterator 遍历TreeMap时，得到的记录是排过序的。键不能为空，值可以为空。

3、其他特征

\***List，Set，Map将持有对象一律视为Object型别。**

\*Collection、List、Set、Map都是接口，不能实例化。

继承自它们的 ArrayList, Vector, HashTable, HashMap是具象class，这些才可被实例化。

\*Vector容器确切知道它所持有的对象隶属什么型别。Vector不进行边界检查。

三、Collections

Collections是针对集合类的一个帮助类。提供了一系列静态方法实现对各种集合的搜索、排序、线程完全化等操作。

相当于对Array（数组）进行类似操作的类——Arrays。

如，Collections.max(Collection coll); 取coll中最大的元素。

Collections.sort(List list); 对list中元素排序

四、如何选择？

1、容器类和Array的区别、择取

\* 容器类仅能持有对象引用（指向对象的指针），而不是将对象信息copy一份至数列某位置。

\* 一旦将对象置入容器内，便损失了该对象的型别信息。

2、

\* 在各种Lists中，最好的做法是以ArrayList作为缺省选择。**当插入、删除频繁时，使用LinkedList()；**

**Vector总是比ArrayList慢，所以要尽量避免使用。**

\* 在各种Sets中，HashSet通常优于TreeSet（插入、查找）。只有当需要产生一个经过排序的序列，才用TreeSet。

TreeSet存在的唯一理由：能够维护其内元素的排序状态。

\* 在各种Maps中

HashMap用于快速查找。

\* 当元素个数固定，用Array，因为Array效率是最高的。

结论：最常用的是ArrayList，HashSet，HashMap，Array。而且，我们也会发现一个规律，**用TreeXXX都是排序的。**

注意：

1、Collection没有get()方法来取得某个元素。只能通过iterator()遍历元素。

2、Set和Collection拥有一模一样的接口。

3、List，可以通过get()方法来一次取出一个元素。使用数字来选择一堆对象中的一个，get(0)...。(add/get)

4、一般使用ArrayList。**用LinkedList构造堆栈stack、队列queue。**

5、Map用 put(k,v) / get(k)，还可以使用containsKey()/containsValue()来检查其中是否含有某个key/value。

HashMap会利用对象的hashCode来快速找到key。

\* hashing

哈希码就是将对象的信息经过一些转变形成一个独一无二的int值，这个值存储在一个array中。

我们都知道所有存储结构中，array查找速度是最快的。所以，可以加速查找。

发生碰撞时，让array指向多个values。即，数组每个位置上又生成一个梿表。（处理冲突）

6、Map中元素，可以将key序列、value序列单独抽取出来。

使用keySet()抽取key序列，**将map中的所有keys生成一个Set。**

使用values()抽取value序列，**将map中的所有values生成一个Collection。**

为什么一个生成Set，一个生成Collection？那是因为，key总是独一无二的，value允许重复。