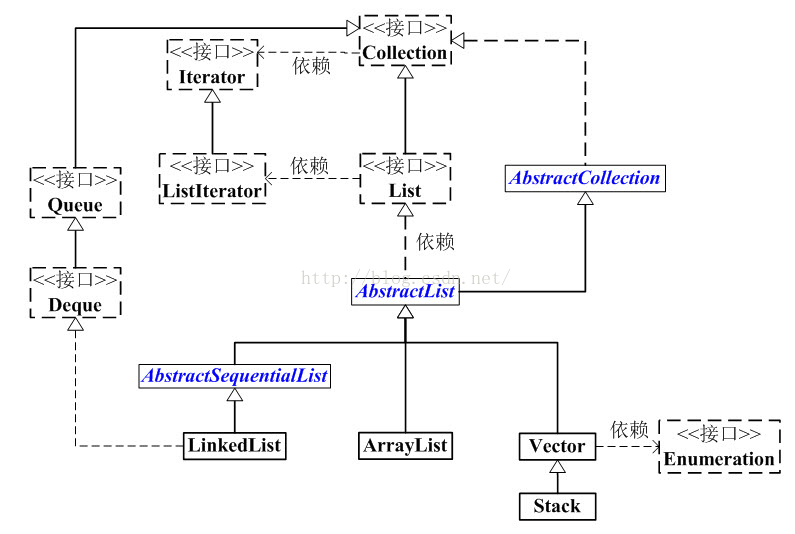
# ****一、List概括****

先来回顾一下List在Collection中的的框架图：



从图中我们可以看出：

1. List是一个接口，它继承与Collection接口，代表有序的队列。
2. AbstractList是一个抽象类，它继承与AbstractCollection。AbstractList实现了List接口中除了size()、get(int location)之外的方法。
3. AbstractSequentialList是一个抽象类，它继承与AbstrctList。AbstractSequentialList实现了“链表中，根据index索引值操作链表的全部方法”。
4. ArrayList、LinkedList、Vector和Stack是List的四个实现类，其中Vector是基于JDK1.0，虽然实现了同步，但是效率低，已经不用了，Stack继承与Vector，所以不再赘述。
5. LinkedList是个双向链表，它同样可以被当作栈、队列或双端队列来使用。

**二、ArrayList和LinkedList的区别**

我们知道，通常情况下，ArrayList和LinkedList的区别有以下几点：

1. ArrayList是实现了基于动态数组的数据结构，而LinkedList是基于链表的数据结构；
2. 对于随机访问get和set，ArrayList要优于LinkedList，因为LinkedList要移动指针；
3. 对于添加和删除操作add和remove，一般大家都会说LinkedList要比ArrayList快，因为ArrayList要移动数据。但是实际情况并非这样，对于添加或删除，LinkedList和ArrayList并不能明确说明谁快谁慢，下面会详细分析。

我们结合之前分析的源码，来看看为什么是这样的：

ArrayList中的随机访问、添加和删除部分源码如下：

//获取index位置的元素值

public E get(int index){

rangeCheck(index);//首先判断index的范围是否合法

return elementData(index);

}

//将index位置的值设为element，并返回原来的值

public E set(int index,E element){

rangeCheck(index);

E oldValue=elementData(index);

elementData[index]=element;

return oldValue;

}

//将element添加到ArrayList的指定位置

public void add(int index,E element){

rangeCheckForAdd(index);

ensureCapacityInternal(size+1);//Increments modCount!!

//将index以及index之后的数据复制到index+1的位置往后，即从index开始向后挪了一位

System.arraycopy(elementData,index,elementData,

index+1,size-index);

elementData[index]=element;//然后在index处插入element

size++;

}

//删除ArrayList指定位置的元素

public E remove(int index){

rangeCheck(index);

modCount++;

E oldValue=elementData(index);

int numMoved=size-index-1;

if(numMoved>0)

//向左挪一位，index位置原来的数据已经被覆盖了

System.arraycopy(elementData,index+1,elementData,

index,numMoved);

//多出来的最后一位删掉

elementData[--size]=null;//clear to let GC do its work

return oldValue;

}

LinkedList中的随机访问、添加和删除部分源码如下：

//获得第index个节点的值

public E get(int index){

checkElementIndex(index);

return node(index).item;

}

//设置第index元素的值

public E set(int index,E element){

checkElementIndex(index);

Node<E>x=node(index);

E oldVal=x.item;

x.item=element;

return oldVal;

}

//在index个节点之前添加新的节点

public void add(int index,E element){

checkPositionIndex(index);

if(index==size)

linkLast(element);

else

linkBefore(element,node(index));

}

//删除第index个节点

public E remove(int index){

checkElementIndex(index);

return unlink(node(index));

}

//定位index处的节点

Node<E>node(int index){

//assert isElementIndex(index);

//index<size/2时，从头开始找

if(index<(size>>1)){

Node<E>x=first;

for(int i=0;i<index;i++)

x=x.next;

return x;

}else{//index>=size/2时，从尾开始找

Node<E>x=last;

for(int i=size-1;i>index;i--)

x=x.prev;

return x;

}

}

从源码可以看出，ArrayList想要get(int index)元素时，直接返回index位置上的元素，而LinkedList需要通过for循环进行查找，虽然LinkedList已经在查找方法上做了优化，比如index<size/2，则从左边开始查找，反之从右边开始查找，但是还是比ArrayList要慢。这点是毋庸置疑的。

ArrayList想要在指定位置插入或删除元素时，主要耗时的是System.arraycopy动作，会移动index后面所有的元素；LinkedList主耗时的是要先通过for循环找到index，然后直接插入或删除。这就导致了两者并非一定谁快谁慢，下面通过一个测试程序来测试一下两者插入的速度：

import java.util.ArrayList;

import java.util.Collections;

import java.util.LinkedList;

import java.util.List;

/\*

\*@description测试ArrayList和LinkedList插入的效率

\*@eson\_15

\*/

public class ArrayOrLinked{

static List<Integer>array=new ArrayList<Integer>();

static List<Integer>linked=new LinkedList<Integer>();

public static void main(String[]args){

//首先分别给两者插入10000条数据

for(int i=0;i<10000;i++){

array.add(i);

linked.add(i);

}

//获得两者随机访问的时间

System.out.println("array time:"+getTime(array));

System.out.println("linked time:"+getTime(linked));

//获得两者插入数据的时间

System.out.println("array insert time:"+insertTime(array));

System.out.println("linked insert time:"+insertTime(linked));

}

public static long getTime(List<Integer>list){

long time=System.currentTimeMillis();

for(int i=0;i<10000;i++){

int index=Collections.binarySearch(list,list.get(i));

if(index!=i){

System.out.println("ERROR!");

}

}

return System.currentTimeMillis()-time;

}

//插入数据

public static long insertTime(List<Integer>list){

/\*

\*插入的数据量和插入的位置是决定两者性能的主要方面，

\*我们可以通过修改这两个数据，来测试两者的性能

\*/

long num=10000;//表示要插入的数据量

int index=1000;//表示从哪个位置插入

long time=System.currentTimeMillis();

for(int i=1;i<num;i++){

list.add(index,i);

}

return System.currentTimeMillis()-time;

}

}

主要有两个因素决定他们的效率，插入的数据量和插入的位置。我们可以在程序里改变这两个因素来测试它们的效率。

当数据量较小时，测试程序中，大约小于30的时候，两者效率差不多，没有显著区别；当数据量较大时，大约在容量的1/10处开始，LinkedList的效率就开始没有ArrayList效率高了，特别到一半以及后半的位置插入时，LinkedList效率明显要低于ArrayList，而且数据量越大，越明显。比如我测试了一种情况，在index=1000的位置(容量的1/10)插入10000条数据和在index=5000的位置以及在index=9000的位置插入10000条数据的运行时间如下：

在index=1000出插入结果：

array time:4

linked time:240

array insert time:20

linked insert time:18

在index=5000处插入结果：

array time:4

linked time:229

array insert time:13

linked insert time:90

在index=9000处插入结果：

array time:4

linked time:237

array insert time:7

linked insert time:92

从运行结果看，LinkedList的效率是越来越差。

所以当插入的数据量很小时，两者区别不太大，当插入的数据量大时，大约在容量的1/10之前，LinkedList会优于ArrayList，在其后就劣与ArrayList，且越靠近后面越差。所以个人觉得，**一般首选用ArrayList**，由于LinkedList可以实现栈、队列以及双端队列等数据结构，所以当特定需要时候，使用LinkedList，当然咯，数据量小的时候，两者差不多，视具体情况去选择使用；当数据量大的时候，如果只需要在靠前的部分插入或删除数据，那也可以选用LinkedList，反之选择ArrayList反而效率更高。