Difference between ArrayList and LinkedList and Vector

#### https://www.programcreek.com/wp-content/uploads/2009/02/java-collection-hierarchy.jpeg

**ArrayList** is implemented as a resizable array. As more elements are added to ArrayList, its size is increased dynamically. It's elements can be accessed directly by using the get and set methods, since ArrayList is essentially an array.

**LinkedList** is implemented as a double linked list. Its performance on add and remove is better than Arraylist, but worse on get and set methods.

**Vector** is similar with ArrayList, but it is synchronized.

**ArrayList** is a better choice if your program is thread-safe. Vector and ArrayList require more space as more elements are added. Vector each time doubles its array size, while ArrayList grow 50% of its size each time. **LinkedList**, however, also implements Queue interface which adds more methods than ArrayList and Vector, such as offer(), peek(), poll(), etc.

Note: The default initial capacity of an ArrayList is pretty small. It is a good habit to construct the ArrayList with a higher initial capacity. This can avoid the resizing cost.

Performance **Example of ArrayList and LinkedList :**

**import** **java.util.ArrayList**;

**import** **java.util.LinkedList**;

**public** **class** **ArrayListLinkedListExample** {

**public** **static** **void** **main**(String[] args) {

 ArrayList<Integer> arrayList = new ArrayList<Integer>();

LinkedList<Integer> linkedList = new LinkedList<Integer>();

// ArrayList add

long startTime = System.nanoTime();

for (int i = 0; i < 100000; i++) {

arrayList.add(i);

}

long endTime = System.nanoTime();

long duration = endTime - startTime;

System.out.println("ArrayList add: " + duration);

// LinkedList add

startTime = System.nanoTime();

for (int i = 0; i < 100000; i++) {

linkedList.add(i);

}

endTime = System.nanoTime();

duration = endTime - startTime;

System.out.println("LinkedList add: " + duration);

// ArrayList get

startTime = System.nanoTime();

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

arrayList.get(i);

}

endTime = System.nanoTime();

duration = endTime - startTime;

System.out.println("ArrayList get: " + duration);

// LinkedList get

startTime = System.nanoTime();

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

linkedList.get(i);

}

endTime = System.nanoTime();

duration = endTime - startTime;

System.out.println("LinkedList get: " + duration);

// ArrayList remove

startTime = System.nanoTime();

for (int i = 9999; i >=0; i--) {

arrayList.remove(i);

}

endTime = System.nanoTime();

duration = endTime - startTime;

System.out.println("ArrayList remove: " + duration);

// LinkedList remove

startTime = System.nanoTime();

for (int i = 9999; i >=0; i--) {

linkedList.remove(i);

}

endTime = System.nanoTime();

duration = endTime - startTime;

System.out.println("LinkedList remove: " + duration); }

}

|  |
| --- |
|  |

And the output is:

ArrayList add: 13265642

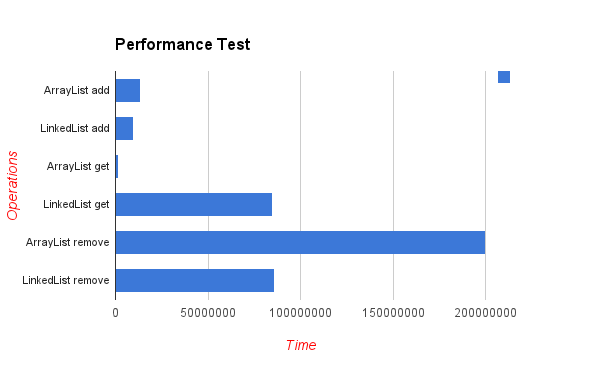
LinkedList add: 9550057

ArrayList get: 1543352

LinkedList get: 85085551

ArrayList remove: 199961301

LinkedList remove: 85768810



The difference of their performance is obvious. LinkedList is faster in add and remove, but slower in get. Based on the complexity table and testing results, we can figure out when to use ArrayList or LinkedList. In brief, LinkedList should be preferred if:

* there are no large number of random access of element
* there are a large number of add/remove operations

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are unsynchronized classes (not Thread safe).

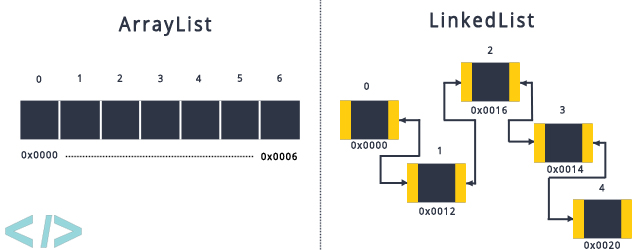
|  |  |  |
| --- | --- | --- |
|  | ArrayList | LinkedList |
| Implementation | ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| Manipulation | Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| Implements | ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| Sorting/Accessing | ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| ReverseIterator | No | Yes , descendingIterator() |
|  |  |  |
| Initial Capacity | 10 | Constructs empty list |
|  |  |  |
| get(int) operation | Fast | Slow in comparision |
|  |  |  |
| add(int) operation | Slow in comparision | Fast |
| Memory Overhead | No | Yes |

**1. Implementation :**ArrayList is the resizable array implementation of list interface , while LinkedList is the Doubly-linked list implementation of the list interface.  
  
  
**2. Performance  :**Performance of ArrayList and LinkedList depends on the type of operation  
  
a. get(int index) or search operation :  ArrayList get(int index) operation runs in constant time i.e O(1)  while LinkedList get(int index) operation run time is O(n) .  
  
The reason behind ArrayList being faster than LinkedList is that ArrayList uses index based system for its elements as it internally uses array data structure , on the other hand ,  
LinkedList does not provide index based access for its elements as it iterates either from the beginning or end (whichever is closer) to retrieve the node at the specified element index.  
  
b. insert() or add(Object) operation :  Insertions in LinkedList are generally fast as compare to ArrayList.  
  
In LinkedList adding or insertion is O(1) operation . While in ArrayList, if array is full i.e worst case,  there is extra cost of  resizing array and copying elements to the new array , which makes runtime of add operation in ArrayList O(n) , otherwise it is O(1) .  
  
c. remove(int) operation :  Remove operation in LinkedList is generally same as ArrayList i.e. O(n).  
In LinkedList , there are two overloaded remove methods. one is remove() without any parameter which removes the head of the list and runs in constant time O(1) .  
The other overloaded remove method in LinkedList is remove(int) or remove(Object) which removes the Object or int passed as parameter . This method traverses the LinkedList until it found the Object and unlink it from the original list . Hence this method run time is O(n).  
  
While in ArrayList remove(int) method involves copying elements from old array to new updated array , hence its run time is O(n).  
  
**3.  Reverse  Iterator :**LinkedList can be iterated in reverse direction using descendingIterator() while there is no descendingIterator() in ArrayList , so we need to write our own code to iterate over the ArrayList in reverse direction.  
  
**4. Initial Capacity :**If the constructor  is not overloaded , then ArrayList creates an empty list of initial capacity 10 , while LinkedList  only constructs the empty list without any initial capacity.  
  
**5. Memory Overhead :**Memory overhead in LinkedList is more as compared to ArrayList as node in LinkedList needs to maintain the addresses of next and previous node. While in ArrayList  each index only holds the actual object(data).

#### **When to Use ArrayList and LinkedList :** In real world applications , you will more frequently use ArrayList than LinkedList. But in a very specific situations LinkedList can be preferred. 1. ArrayList is preferred when there are more get(int) or search operations need to be performed as every search operation runtime is O(1). 2. If application requires more insert(int) , delete(int) operations then the get(int) operations then LinkedList is preferred as they do not need to maintain back and forth like arraylist  to preserve continues indices.

#### **1. Memory Allocation**

ArrayList uses Array as underline datastructure to store the elements, and we know Array stores elements in consecutive manner. In LinkedList, elements can be stored at any available memory location as address of node is stored in previous node.

Memory Allocation

#### **2. Element Retrieval**

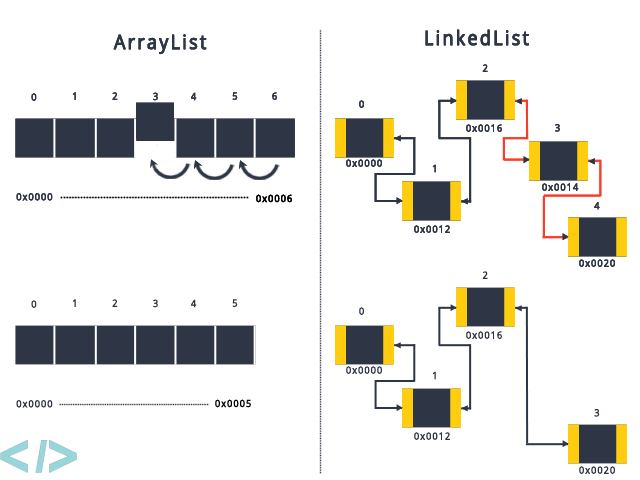
Element retrieval in ArrayList is much faster then LinkedList, ArrayList take O(1) of time to retrieve any location where LinkedList take O(n) time to retrieve the element.

#### **3. Element Manipulation**

Element manipulation with ArrayList is slow because it uses internally Array, For storing or removing of any new element needs to shift other elements in Array.

###### **Example :**

If we want to delete element at index 3.

Remove Element at index 3

Element manipulation with LinkedList is fast compare to ArrayList, For storing or removing of new element needs to update the node address only.

#### **4. Role as a Queue**

ArrayList can be act as List only as it implements List interface only, Where LinkedList can be act as List and Queue also as it implements List and Deque interface.

## **Differences Between ArrayList And LinkedList In Java:**

|  |  |  |
| --- | --- | --- |
|  | **ArrayList** | **LinkedList** |
| Structure | ArrayList is an index based data structure where each element is associated with an index. | Elements in the LinkedList are called as nodes, where each node consists of three things – Reference to previous element, Actual value of the element and Reference to next element. |
| Insertion And Removal | Insertions and Removals in the middle of the ArrayList are very slow. Because after each insertion and removal, elements need to be shifted. | Insertions and Removals from any position in the LinkedList are faster than the ArrayList. Because there is no need to shift the elements after every insertion and removal. Only references of previous and next elements are to be changed. |
| Insertion and removal operations in ArrayList are of order O(n). | Insertion and removal in LinkedList are of order O(1). |
| Retrieval(Searching or getting an element) | Retrieval of elements in the ArrayList is faster than the LinkedList . Because all elements in ArrayList are index based. | Retrieval of elements in LinkedList is very slow compared to ArrayList. Because to retrieve an element, you have to traverse from beginning or end (Whichever is closer to that element) to reach that element. |
| Retrieval operation in ArrayList is of order of O(1). | Retrieval operation in LinkedList is of order of O(n). |
| Random Access | ArrayList is of type Random Access. i.e elements can be accessed randomly. | LinkedList is not of type Random Access. i.e elements can not be accessed randomly. you have to traverse from beginning or end to reach a particular element. |
| Usage | ArrayList can not be used as a Stack or Queue. | LinkedList, once defined, can be used as ArrayList, Stack, Queue, Singly Linked List and Doubly Linked List. |
| Memory Occupation | ArrayList requires less memory compared to LinkedList. Because ArrayList holds only actual data and it’s index. | LinkedList requires more memory compared to ArrayList. Because, each node in LinkedList holds data and reference to next and previous elements. |
| When To Use | If your application does more retrieval than the insertions and deletions, then use ArrayList. | If your application does more insertions and deletions than the retrieval, then use LinkedList. |