链滴

从源码对比 ArrayList 和 LinkedList

作者：18380422102

原文链接：https://ld246.com/article/1534833863344

来源网站：链滴

1. 存储结构

ArrayList:数组

```
transient Object[] elementData;
```

LinkedList:链表

```
transient Node first;

transient Node last;
```

2.扩容

ArrayList:1.5倍

```
private void grow(int minCapacity) {
  int oldCapacity = elementData.length;
//新容量为旧容量的1.5倍
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    if (newCapacity - minCapacity < 0)
        newCapacity = minCapacity;
    if (newCapacity - MAX_ARRAY_SIZE > 0)
        newCapacity = hugeCapacity(minCapacity);
    // 以新容量为长度创建一个新的数组，并把旧数组拷贝进去
  elementData = Arrays.copyOf(elementData, newCapacity);
}
```

LinkedList:不需要扩容

3.Get方法

ArrayList: 直接根据下标从数组中获取元素

```
public E get(int index) {
    rangeCheck(index);

    return elementData(index);
}
@SuppressWarnings("unchecked")
E elementData(int index) {
    return (E) elementData[index];
}
```

LinkedList: 遍历一半的链表

```
public E get(int index) {
    checkElementIndex(index);
    return node(index).item;
}
Node node(int index) {
    // assert isElementIndex(index);

  if (index < (size >> 1)) {
        Node x = first;
        for (int i = 0; i < index; i++)
            x = x.next;
        return x;
    } else {
        Node x = last;
        for (int i = size - 1; i > index; i--)
            x = x.prev;
        return x;
    }
}
```

4.add方法

a) 尾部添加新元素

ArrayList: 如果容量足够的话，直接在数组尾部添加元素，否则先扩容再添加元素

```
public boolean add(E e) {
//判断是否需要扩容
    ensureCapacityInternal(size + 1);
  elementData[size++] = e;
    return true;
}
```

LinkedList:若链表为空，则头尾为同一个新节点，否则在链表尾部添加新节点

```
public boolean add(E e) {
    linkLast(e);
```

```
    return true;
}
void linkLast(E e) {
    final Node l = last;
    final Node newNode = new Node<>(l, e, null);
    last = newNode;
    if (l == null)
        first = newNode;
    else
        l.next = newNode;
    size++;
    modCount++;
}
```

b) 中间添加新元素

ArrayList: 将要插入的位置之后的元素向后拷贝，然后在该位置插入新元素

```
public void add(int index, E element) {
    rangeCheckForAdd(index);

    ensureCapacityInternal(size + 1);
//通过拷贝，将index位置之后的所有元素都后移一位
    System.arraycopy(elementData, index, elementData, index + 1,
                     size - index);
    elementData[index] = element;
    size++;
}
```

LinkedList: 在指定节点的前面创建一个新节点，并改变相应的指向即可。

```
public void add(int index, E element) {
    checkPositionIndex(index);

    if (index == size)
//在最后插入元素
        linkLast(element);
    else
        linkBefore(element, node(index));
}
void linkBefore(E e, Node succ) {
    final Node pred = succ.prev;
//创建一个新节点，前节点为指定节点的前节点，后节点为指定节点
    final Node newNode = new Node<>(pred, e, succ);
//使指定节点的前节点指向新节点
    succ.prev = newNode;
    if (pred == null)
//新节点为首节点
        first = newNode;
    else
//指定节点的前节点指向新节点
        pred.next = newNode;
    size++;
    modCount++;
```

```
}
```

5.删除元素

ArrayList:将要删除的下标之后的所有元素向前拷贝一位，并将最后一位置为null

```
public E remove(int index) {
    rangeCheck(index);

    modCount++;
    E oldValue = elementData(index);

    int numMoved = size - index - 1;
    if (numMoved > 0)
//将index+1之后的元素都向前移动一位
        System.arraycopy(elementData, index+1, elementData, index,
                         numMoved);
    elementData[--size] = null; // clear to let GC do its work

  return oldValue;
}
```

LinkedList: 先遍历找到节点，然后删除该节点即可

```
public E remove(int index) {
    checkElementIndex(index);
//通过遍历找到要删除的节点
    return unlink(node(index));
}
E unlink(Node x) {
    // assert x != null;
    final E element = x.item;
    final Node next = x.next;
    final Node prev = x.prev;

    if (prev == null) {
//要删除的节点是首节点，就讲first节点指向下一个节点
        first = next;
    } else {
//不是首节点，就讲前节点的后节点指向为目标节点的后节点
        prev.next = next;
        x.prev = null;
    }

    if (next == null) {
//要删除的节点是最后一个节点，就讲last节点指向目标节点的前节点
        last = prev;
    } else {
//不是最后一个节点，就讲后节点的前节点指向为目标节点的前节点
        next.prev = prev;
        x.next = null;
    }

    x.item = null;
```

```
    size--;
    modCount++;
    return element;
}
```

6.总结

ArrayList使用数组实现的，所以在查找元素方面比较有优势，可是涉及扩容，向中间插入节点时需要贝部分数组，比较消耗资源。

LinkedList是使用链表实现的，所以在插入元素比较有优势，但是也需要遍历链表去查找节点。不过频繁插入时仍然比数组有优势。但是在查找元素时，需要遍历一半的链表，不如数组方便。