LinkedList

작성일시	@2023년 4월 15일 오후 3:18
강의 번호	자바의정석 Chapter11
유형	
자료	
복습	☐
Spring Framwork	

▼ 배열의 장점과 단점

▼ 장점

- 데이터를 읽어 오는데 걸리는 시간이 가장 빠르다.
- 구조가 간단하다.

▼ 단점

- 크기를 변경하려면 새로운 배열을 생성해서 데이터를 복사해야 한다.
- 실행속도를 향상시키기 위해서는 충분히 큰 크기의 배열을 생성해야 하므로 메모리가 낭비된다.
- 비순차적인 데이터의 추가 또는 삭제에 시간이 많이 걸린다.
- 차례대로 데이터를 추가하고 마지막에서부터 데이터를 삭제하는 것은 빠르다
  하지만, 중간에 데이터를 추가하려면, 빈자리를 만드릭 위해 다른 데이터를 복사해서 이동한다.

▼ 이러한 장단점의 보완 : Linked list

▼ 불연속적으로 존재하는 데이터를 서로 연결한 형태

- 이동방향이 단방햐잉므로 접근은 쉽지만 이전 요소에 접근이 어렵다.

▼ Linked list의 보완 : Double Linked List > doubly circular linked list

- 양방향으로 요소를 연결하고, 첫 번째 요소와 마지막 요소를 연결시킨 것
- 마치 Tv처럼 마지막채널이 끝나면 첫 번째 채널로 돌아온다.
- 실제로 LinkedList는 더블 링크드 리스트로 구현되어 있다.

```
LinkedList()  : LinkedList객체를 생성
LinkedList(Collection c) : 주어진 컬렉션을 포함하는 LinkedList객체를 생성
boolean add(Object o) : 지정된 객체를 LinkedList의 끝에 추가. 저장에 성공하면 boolean
void add(int index, Object element) : 지정된 위치에 객체 추가
boolean addALL(Collection c) : 주어진 컬렉션에 포함된 모든 요소를 LinkedList끝에 추가. 성공하면 true
boolean addAll(int index, Collection c) : 지정된 위치에 주어진 컬렉션을 추가. 성공하면 true
void clear() LinkedList의 모든 요소를 삭제
boolean contains(Object o) : 지정된 객체가 포함되어있는지 알려줌
boolean containsAll(Collection c) : 지정된 컬렉션의 모든 요소가 포함되어있는지 알려줌
Object get(int index) : 지정된 위치의 객체를 반환
int indexOf(Object o) : 지정된 객체가 저장된 위치를 반환
boolean isEmpty() : LinkedList가 비어있는지 알려준다. 비어있으면 true
int lastIndexOf(Object o) : 지정된 객체의 위치를 반환(끝부터 역순)
boolean remove(Ojbect o)
booelan removeAll(Collection c) : 컬렉션의 요소와 일치하는 모든 요소를 삭제
booelan retainAll(Collection c) : 지정된 컬렉션의 모든 요소가 포함되어 있는지 확인
Object set(int index, Object element) : 지정된 위치의 객체를 주어진 객체로 바꿈
int size()
List subList(int fromIndex, int toIndex)
Object[] toArray()
Object[] toArray(Object[] a)
Object element() : 첫번째 요소를 반환
boolean offer(Object o) : 지정된 객체를 LinkedList의 끝에 추가, 성공하면 true
Object peek() : 첫번째 요소를 반환
Object poll() : 첫번째 요소를 반환, 반환후에는 제거됨
Object remove() : 첫번째 요소를 제거
void addFirst(Object o): 맨 앞에 객체를 추가
void addLast(Object o)
Iterator descendingIterator() : 역순으로 조회하기위한 디센딩이터레이터를 반환
Object getFirst()
Object getLast()
boolean offerFirst(Object o) : 맨 앞에 객체 추가
boolean offerLast(Object o) : 맨 뒤에 객체 추가
Object peekFirst()
Object peekLast()
Object pollFirst()
ObjectLast()
Object pop() : removeFirst()와 동일
void push(Object o) : addFirst()와 동일
Object removeFirst()
Object removeLast()
boolean removeFirstOccurrence(Object o) : 첫번째로 일치하는 객체를 제거
booelan removeLastOccurrence(Object o) : 마지막으로 일치하는 객체를 제
```

결론

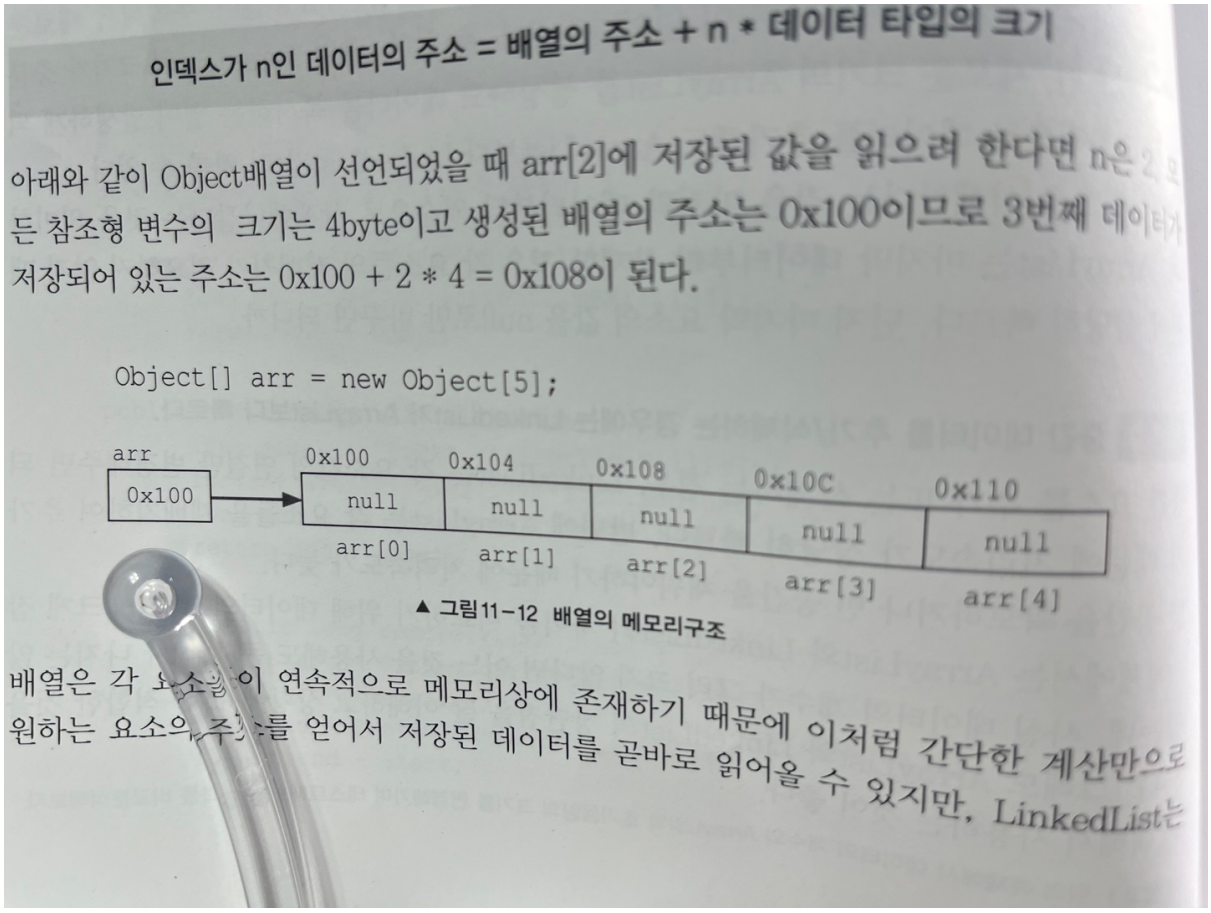
- 순차적으로 추가/삭제하는 경우 ArrayList가 보다 빠르다.
- 중간 데이터를 추가/삭제하는 경우에는 LinkedLIst가 빠르다.

```
package ch11;

import java.util.*;

public class ArrayListLinkedListTest2 {
public static void main(String[] args) {
    ArrayList al = new ArrayList(1000000);
    LinkedList ll = new LinkedList();
    add(al);
    add(ll);

    System.out.println("== 접근 시간 테스트 ==");
    System.out.println("ArrayList :" + access(al));
    System.out.println("LinkedList :" + access(ll));
  }
public static void add(List list) {
  for(int i =0; i<100000; i++) list.add(i+"");
}
public static long access(List list) {
  long start = System.currentTimeMillis();
  for(int i=0; i<10000;i++) list.get(i);
  long end = System.currentTimeMillis();
  return end-start;
}
}
```

▼ 코드 결과

== 접근 시간 테스트 ==
ArrayList :2
LinkedList :113

index가 n인 데이터의 주소 = 배열의주소 +n*데이터 타입의 크기



인덱스가 n인 데이터의 주소 = 배열의 주소 + n * 데이터 타입의 크기

아래와 같이 Object배열이 선언되었을 때 arr[2]에 저장된 값을 읽으려 한다면 n은 2로
든 참조형 변수의 크기는 4byte이고 생성된 배열의 주소는 0x100이므로 3번째 데이터가
저장되어 있는 주소는 0x100 + 2 * 4 = 0x108이 된다.

```
Object[] arr = new Object[5];
```


▲ 그림11-12 배열의 메모리구조

배열은 각 요소들이 연속적으로 메모리상에 존재하기 때문에 이처럼 간단한 계산만으로
원하는 요소의 주소를 얻어서 저장된 데이터를 곧바로 읽어올 수 있지만, LinkedList는

- 배열은 각 요소들이 연속적으로 메모리상에 존재하기 때문에 간단한 계산만으로 원하는 요소의 주소를 얻어서 저장된 데이터를 곧바로 읽어올 수 있지만, LinkedList는 불연속적이다.
- 그래서 LinkedList는 저장해야하는 데이터의 개수가 많아질수록 데이터를 읽어 오는 시간, 즉 접근시간이 길어진다는 단점이 있다.

컬렉션	접근시간	추가/삭제	비고
ArrayList	빠르다	느리다	순차적인 추가삭제는 더 빠르다. 비효율적인 메모리사용
LinkedList	느리다	빠르다	데이터가 많을수록 접근성이 떨어짐

▼ 가장 효율적인 방법

- 두 클래스의 장점을 이용해서 조합하여 사용한다.
- 처음 작업하기 전 데이터 저장시 : ArrayList

- 작업시 LinkedList로 데이터를 옮겨서 작업

```
ArrayList al = new ArrayList(1000000);
for(int i=0; i<1000000;i++) al.add(i+"");

LinkedList ll = new LinkedList(al);
for(int i=0; i<1000;i++) ll.add(500,"X");
```