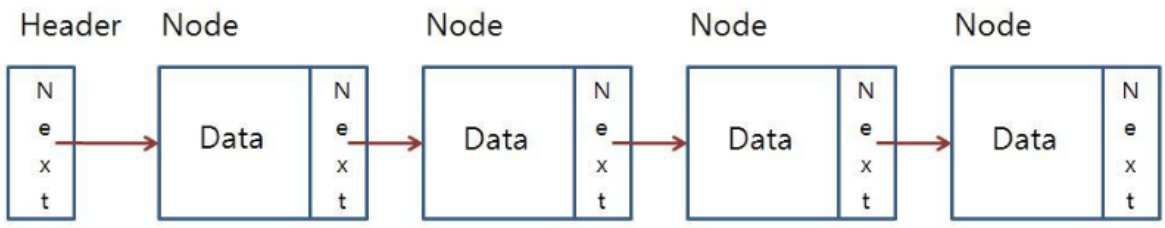
Linked List

LinkedList란?

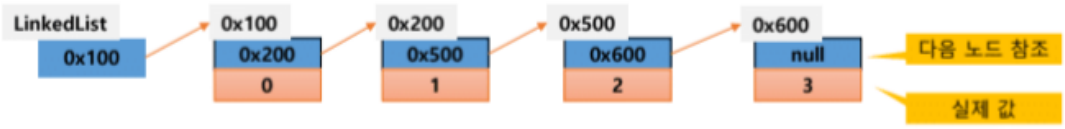
연결 리스트, 링크드 리스트(linked list)는 각 노드가 데이터와 포인터를 가지고 한 줄로 연결되어 있는 방식으로 데이터를 저장하는 자료 구조이다. 데이터를 담고 있는 노드들이 연결되어 있는데, 노드의 포인터가 다음이나 이전의 노드와의 연결을 담당하게 된다.

구조

일반적인 경우



자바의 경우



▪ 데이터 삭제 및 추가

번외

자바에 포인터가 없는 이유

포인터, 참조 모두 주소를 통해 원본 데이터에 접근하는 공통 기능을 가진다

하지만 포인터는 메모리를 직접 핸들링할 수 있지만, 참조는 메모리를 직접 핸들링할 수 없다는 차이점을 가진다

포인터는 직접적으로 주소 값을 변경할 수 있어 실수로 주소 값을 변경하고 문제가 발생할 가능성이 있다

참조는 간접적으로 주소 값에 접근할 수 있어 문제가 발생할 가능성이 낮다

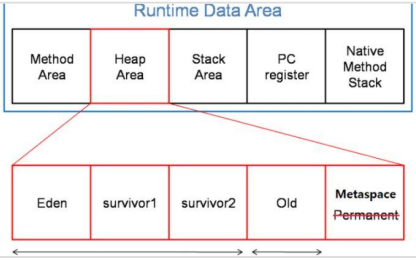
포인터를 사용하면 유연성과 성능을 향상시킬 수 있지만 안정성이 떨어지기 때문에 자바에서 포인터 개념을 제공하지 않는다

(~~그리고 가비지 컬렉터때문도 있다고 함~~)

[Java] 가비지 컬렉터(Garbage Collector)

C, C++은 개발자가 메모리를 직접 관리해야 하지만 Java에서는 개발자가 별도로 관리할 필요가 없이 JVM의 Garbage Collector에서 알아서 메모리를 관리해 준다. 가비지 컬렉터는 이름 그대로 쓰레기를 수

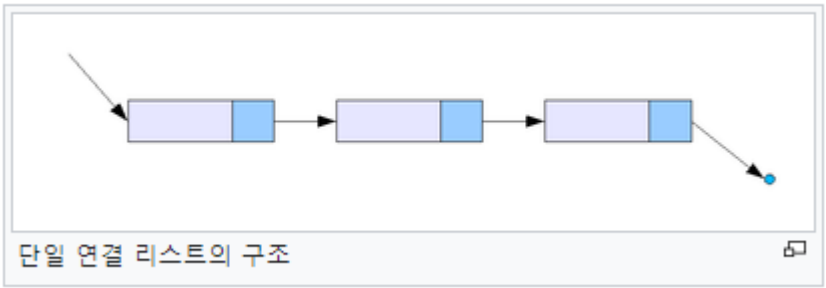
https://sorjfkrh5078.tistory.com/77?category=1007499



종류

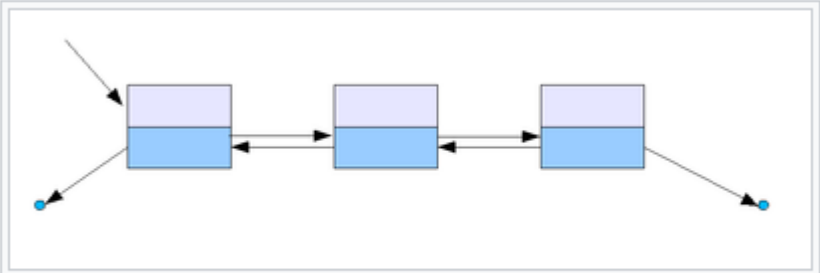
단일 연결 리스트

각 노드에 자료 공간과 한 개의 포인터 공간이 있고, 각 노드의 포인터는 다음 노드를 가리킨다. 반드시 시작 노드의 위치를 알고 있어야 하고, 이전 노드의 위치는 알 수 없다.



이중 연결 리스트

구조는 비슷하지만, 포인터 공간이 두 개가 있고 각각의 포인터는 앞의 노드와 뒤의 노드를 가리킨다. 이전 노드를 가리키는 포인터도 가지고 있기 때문에 전방향, 후방향 어느쪽으로도 순환이 가능하다.

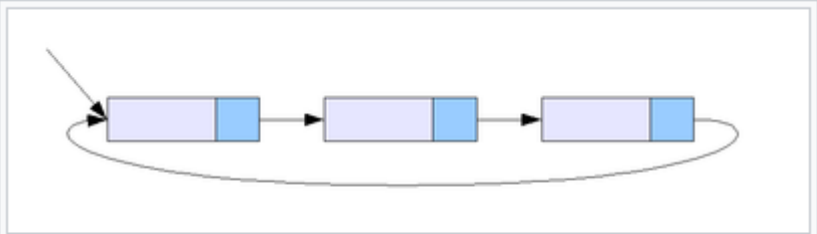


이중 연결 리스트의 구조

원형 연결 리스트

일반적인 연결 리스트에 마지막 노드와 처음 노드를 연결시켜 원형으로 만든 구조이다.

마지막 노드가 NULL이 아닌 처음 노드를 가리킨다.



단순 원형 연결 리스트의 구조

특징

1. 배열에 비해 탐색 속도가 떨어진다 → 배열의 경우 연속적인 공간에 위치함으로 특정 위치에 있는 데이터에 접근하고자 할 때 O(1)의 시간이 소모된다. 그러나 링크드리스트의 경우 처음 노드부터 순환해야하기 때문에 O(N)의 시간이 소모된다
    a. 인덱스가 없다! → get(index)를 쓸 때를 보면 인덱스가 있는게 아닌가?

       배열의 경우 3이라는 인덱스의 값을 탐색할때 arr[3]으로 바로 접근할 수 있다.

       하지만 링크드 리스트의 경우에는 3의 인덱스를 찾기 위해서 첫번째 노드로부터 다음 노드의 참조 값을 받아오고, 다음 노드로부터 3의 인덱스의 참조 노드를 받을 때까지 반복해야 한다.

index라는 같은 용어를 사용하고 있지만 앞에서부터 탐색했을 때 몇 번째에 있는지를 말한다고 보면 된다

2. 데이터의 추가, 삭제가 용이하다 → 배열은 중간에 있는 데이터를 삭제하거나 삽입할 때 매우 비효율적이지만 링크드 리스트는 매우 효율적이다
3. 사이즈가 가변적이다 → 배열의 경우 일단 만들어진 사이즈를 조절하기 힘들지만 링크드리스트는 자유롭다
4. 포인터형의 크기가 4byte로 데이터 수 x 4만큼의 부가적인 메모리가 더 필요하기 때문에 자료구조의 사이즈가 약간 커지게 된다

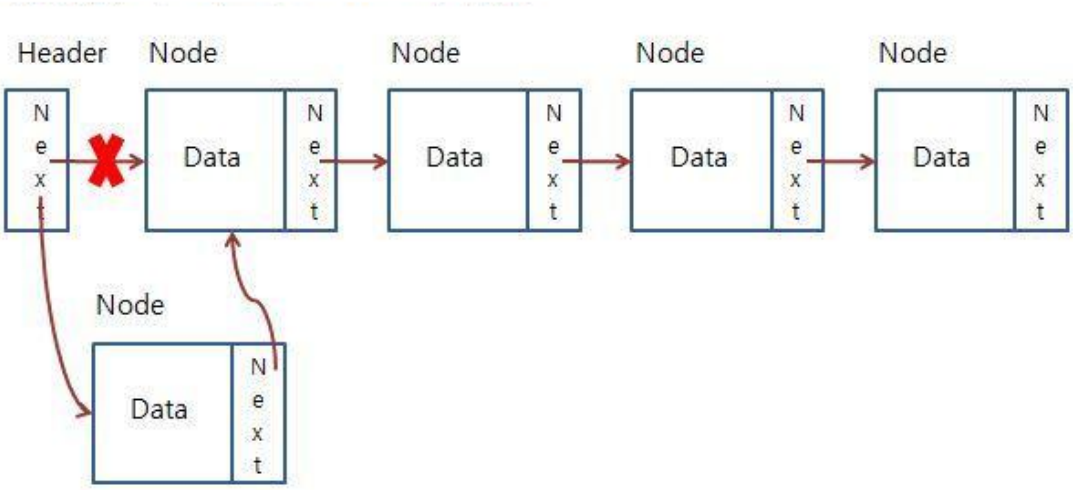
선언

```
LinkedList list = new LinkedList();//타입 미설정 Object로 선언된다.
LinkedList<Student> members = new LinkedList<Student>();//타입설정 Student객체만 사용가능
LinkedList<Integer> num = new LinkedList<Integer>();//타입설정 int타입만 사용가능
LinkedList<Integer> num2 = new LinkedList<>();//new에서 타입 파라미터 생략가능
LinkedList<Integer> list2 = new LinkedList<Integer>(Arrays.asList(1,2));//생성시 값추가
```

(<> << 제네릭스는 선언할 수 있는 타입이 객체 타입이기 때문에 기본 자료 타입들을 객체로 다루기 위한 클래스인 래퍼 클래스(wrapper class)를 사용합니다)

값 추가

```
LinkedList<Integer> list = new LinkedList<Integer>();
list.addFirst(1);//가장 앞에 데이터 추가
list.addLast(2);//가장 뒤에 데이터 추가
list.add(3);//데이터 추가
list.add(1, 10);//index 1에 데이터 10 추가
```



먼저 인자로 받은 값으로 Node를 생성하여 생성한 노드는 이전 노드가 가리키게 하고, 그 다음 노드를 가리키도록 지정합니다

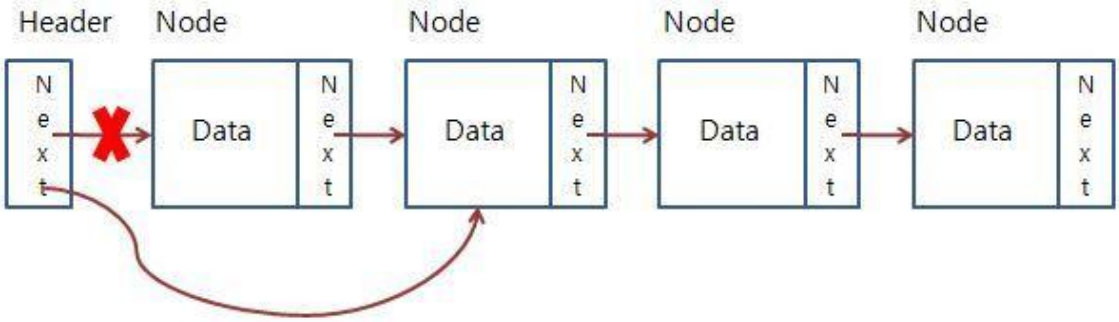
값 변경

```
LinkedList<String> ll = new LinkedList<String>();

    ll.add("Hello");
    ll.add("Hello");
    ll.add(1, "World");

    System.out.println(ll);

    ll.set(1, "Hello");

    System.out.println(ll);
```

값을 바꾸려면 index를 알아야 합니다

값 삭제

```
LinkedList<Integer> list = new LinkedList<Integer>(Arrays.asList(1,2,3,4,5));
list.removeFirst(); //가장 앞의 데이터 제거
list.removeLast(); //가장 뒤의 데이터 제거
list.remove(); //생략시 0번째 index제거
list.remove(1); //index 1 제거
list.clear(); //모든 값 제거
```



삭제 대상 노드의 이전 노드가 삭제 대상 노드의 다음 노드를 가리키게 하고 삭제할 노드를 삭제합니다

크기 구하기

```
System.out.println(list.size()); //list 크기 : 3
```

값 출력

```
LinkedList<Integer> list = new LinkedList<Integer>(Arrays.asList(1,2,3));

System.out.println(list.get(0));//0번째 index 출력

for(Integer i : list) { //for문을 통한 전체출력
    System.out.println(i);
}

Iterator<Integer> iter = list.iterator(); //Iterator 선언
while(iter.hasNext()){//다음값이 있는지 체크
    System.out.println(iter.next()); //값 출력
}
```

링크드리스트는 get(index)를 사용할 수 있지만, 순차 탐색으로 이루어져 있어 ArrayList보다 속도가 느립니다

값 검색

```
LinkedList<Integer> list = new LinkedList<Integer>(Arrays.asList(1,2,3));
System.out.println(list.contains(1)); //list에 1이 있는지 검색 : true
System.out.println(list.indexOf(1)); //1이 있는 index반환 없으면 -1
```