单链表有两个属性：

value，值

next，指向下一个节点的指针（引用）

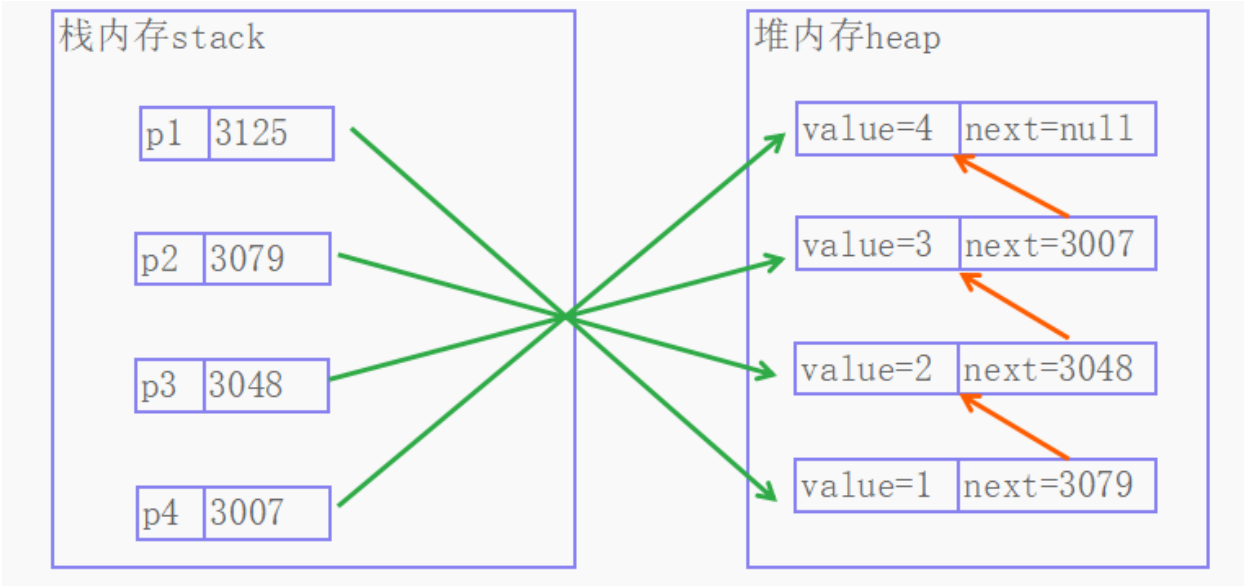
双链表附加一个属性：

pre，指向上一个节点的指针（引用）

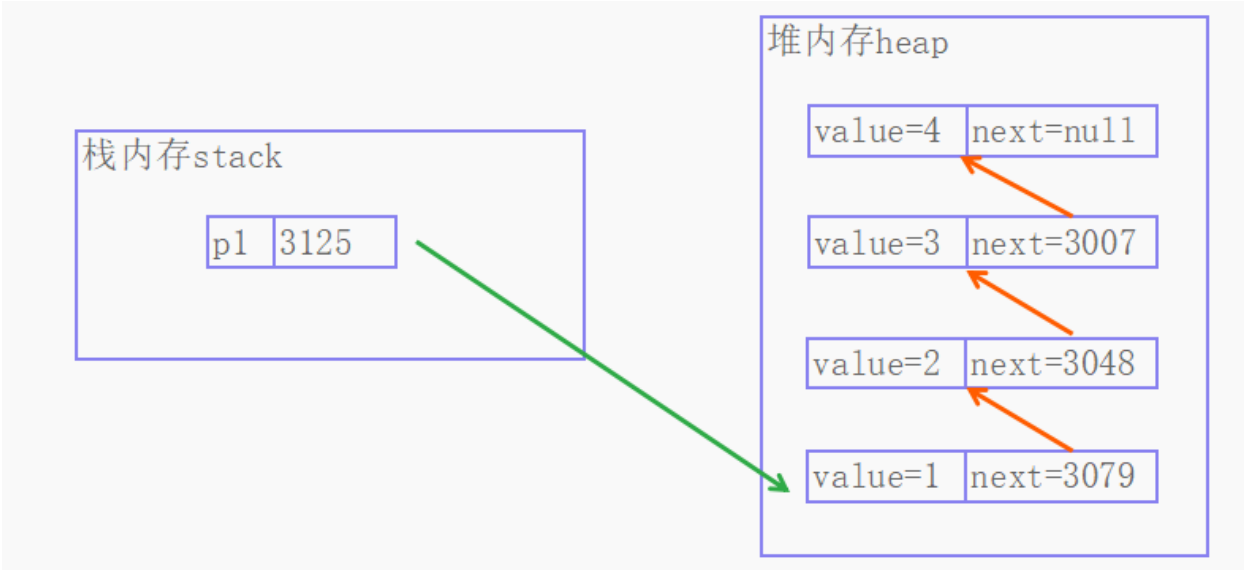
```
package com.jikexueyuan.common;

/**
 * 链表节点
 */
public class ListNode<T> {
/**
 * 值
 */
public T value;
/**
 * 指向下一个节点的指针(引用)
 */
public ListNode<T> next;
public ListNode(T value, ListNode<T> next) {
        super();
        this.value = value;
        this.next = next;
}
public ListNode() {
        super();
}
/**
 * 指向前趋节点的指针，用于双链表
 */
public ListNode<T> pre;
}
```

```
@SuppressWarnings("unused")
@Test
public void testNode01(){
        ListNode<Integer> p4=new ListNode<Integer>(4, null);
        ListNode<Integer> p3=new ListNode<Integer>(3, p4);
        ListNode<Integer> p2=new ListNode<Integer>(2, p3);
        ListNode<Integer> p1=new ListNode<Integer>(1, p2);
}
```



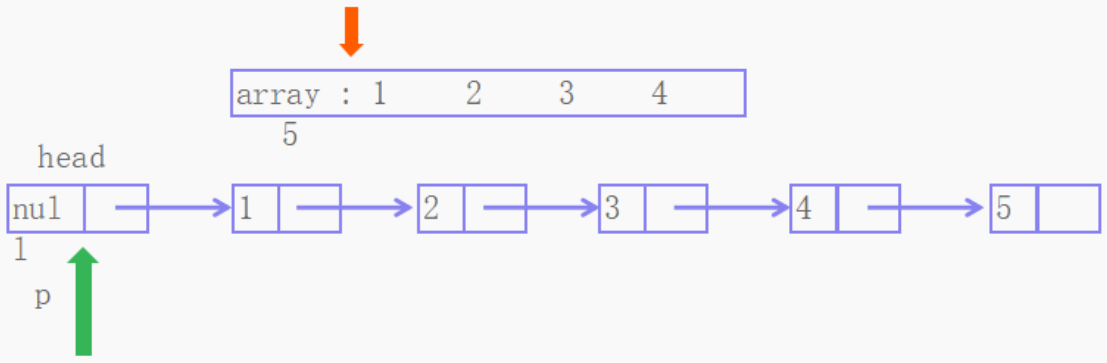

```
@SuppressWarnings("unused")
@Test
public void testNode02(){
        ListNode<Integer> p1=new ListNode<Integer>(1, new ListNode<Integer>(2,
                        new ListNode<Integer>(3, new ListNode<Integer>(4,
null))));
}
```



单链表API：MiniList.java

方法名/属性名	作用
head	头结点，固定
void arrayToList(T[] array)	根据数组array创建链表
void printList()	打印链表
void insert(int index, T value)	在第index个节点后面插入value
T remove(int index)	删除第index个节点，并返回节点的值
T get(int index)	返回第index个节点的值
void set(int index,T value)	将第index个节点的值设置为value

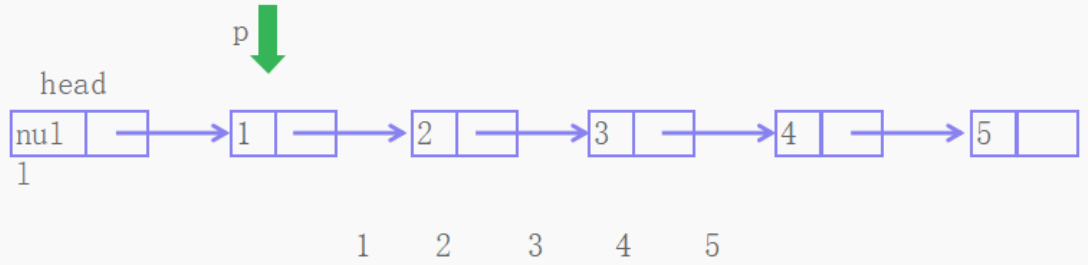
根据数组array创建链表：



依次打印链表，其实就是遍历

遍历的方法：

p=p.next




```
package com.jikexueyuan.common;

import java.util.Comparator;
import java.util.Stack;

/**
泛型版本的单链表
 */
public class MiniList<T> {
/**
 * 默认的头结点
 */
private ListNode<T> head=new ListNode<T>(null, null);
/**
 * 比较器
 */
public Comparator<T> comp;
/**
 * 比较a和b的大小
 * 如果a==b,返回0
 * 如果a<b,返回负数
 * 如果a>b,返回正数
```

```
 */
@SuppressWarnings("unchecked")
public int compare(T a,T b){
        if(comp!=null){
                return comp.compare(a, b);
        }else{
                Comparable<T> c=(Comparable<T>) a;
                return c.compareTo(b);
        }
}
/**
 * 取得链表的最大值
 */
public T getMax(){
        if(head.next==null){
                return null;
        }
        ListNode<T> p=head.next;
        T max=p.value;
        p=p.next;
        while(p!=null){
                if(compare(p.value, max)>0){
                        max=p.value;
                }
                p=p.next;
        }
        return max;
}
/**
 * 数组转换成链表
 */
public void arrayToList(T[] array){
        ListNode<T> p=head;
        for(T t:array){
                ListNode<T> node=new ListNode<T>(t, null);
```

```
                p.next=node;
                p=node;
        }
}
/**
 * 打印链表
 */
public void printList(){
        ListNode<T> p=head.next;
        while(p!=null){
                System.out.print(p.value+" ");
                p=p.next;
        }
        System.out.println();
}
/**
 *插入
 */
public void insert(int index,T value){
        ListNode<T> p=head;
        for(int i=0;i<=index;i++){
                p=p.next;
        }
        ListNode<T> node=new ListNode<T>(value,null);
        node.next=p.next;
        p.next=node;
}
/**
 *删除
 */
public T remove(int index){
        ListNode<T> pre=head;
        for(int i=0;i<index;i++){
                pre=pre.next;
        }
```

```
        ListNode<T> p=pre.next;
        pre.next=p.next;
        return p.value;
}
/**
 * 查询
 */
public T get(int index){
        ListNode<T> p=head;
        for(int i=0;i<=index;i++){
                p=p.next;
        }
        return p.value;
}
/**
 * 修改
 */
public void set(int index,T value){
        ListNode<T> p=head;
        for(int i=0;i<=index;i++){
                p=p.next;
        }
        p.value=value;
}
/**
 * 逆序打印链表，非递归算法
 */
public void printInverse(){
        if(head.next==null){
                return;
        }
        Stack<T> stack=new Stack<T>();
        ListNode<T> p=head.next;
        while(p!=null){
                stack.push(p.value);
```

```
                p=p.next;
        }
        while(!stack.isEmpty()){
                System.out.print(stack.pop()+" ");
        }
        System.out.println();
}
/**
 * 逆序打印链表，递归算法
 */
public void printInverseRecursive(){
        if(head.next==null){
                return;
        }
        recursive(head.next);
        System.out.println();
}
private void recursive(ListNode<T> p){
        if(p!=null){
                recursive(p.next);
                System.out.print(p.value+" ");
        }
}
}



/**
 * 测试：单链表的功能
 */
@Test
public void testMiniList(){
        MiniList<Integer> list=new MiniList<Integer>();
        Integer[] array={0,1,2,3,4};
        list.arrayToList(array);
```

```
        list.printList();
        list.insert(2, 10);
        list.printList();
        list.remove(4);
        list.printList();
        list.set(3, 13);
        list.printList();
        System.out.println(list.get(4));
}
```