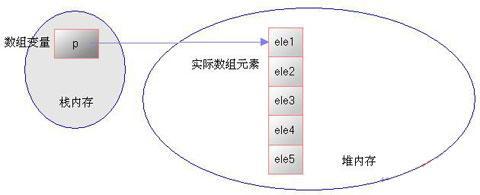
# 链表

数组和链表都是线性存储结构的基础，栈和队列都是线性存储结构的应用。

## 数组

数组是一种**连续存储**线性结构，元素类型相同，大小相等。



数组的优点：存取速度快

数组的缺点：1.事先必须知道数组的长度、2.**插入删除元素很慢**、空间通常是有限制的、需要大块连续的内存块、插入删除元素的效率很低。如果向数组中添加更多的元素，那么必须创建一个新的数组，然后把原数组中的元素复制到新数组中去，这将花费大量的时间；

1.默认的构造函数：

我们需要创建一些方法来初始化我们的数组，那肯定是需要传一个capacity来表示数组的容量嘛：

// 构造函数，传入数组的容量capacity构造Array

public Array(int capacity) {

data = (E[]) new Object[capacity];

size = 0;

}

// 无参数的构造函数，默认数组的容量capacity=10。

2. 涉及到挪动数组元素

## 链表

**链表是离散存储线性结构。**

链表优点：空间没有限制、插入删除元素很快

链表缺点：存取速度很慢。链表中的额外指针引用需要浪费内存

**确定一个链表我们只需要头指针**，通过头指针就可以把整个链表都能推导出来了。

**链表虚拟头结点的作用**

* 为了屏蔽掉链表头结点的特殊性：因为头结点是没有前序结点的，所以我们不管是删除还是增加操作都要对头结点进行单独的判断，为了我们编写逻辑的方便，引入了一个虚拟头结点的概念；

**链表又分了好几类**：

单向链表：一个节点指向下一个节点

双向链表：一个节点有两个指针域

循环链表：能通过任何一个节点找到其他所有的节点，将两种(双向/单向)链表的最后一个结点指向第一个结点从而实现循环

操作一个链表只需要**知道它的头指针就可以做任何操作了**

* 添加数据到链表中

遍历找到尾节点，插入即可(只要while(temp.next != null)，退出循环就会找到尾节点)

* 遍历链表

从首节点(有效节点)开始，只要不为null，就输出

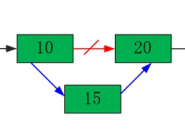
* 给定位置插入节点到链表中

首先判断该位置是否有效(在链表长度的范围内)

**找到想要插入位置的上一个节点**

将原本由上一个节点的指向交由插入的节点来指向

上一个节点指针域指向想要插入的节点



* 获取链表的长度

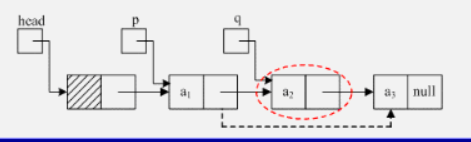
每访问一次节点，变量++操作即可

* 删除给定位置的节点

首先判断该位置是否有效(在链表长度的范围内)

**找到想要插入位置的上一个节点**

**将原本由上一个节点的指向后面一个节点**



* 对链表进行排序

使用冒泡算法对其进行排序

找到链表中倒数第k个节点

设置两个指针p1、p2，让p2比p1**快k个**节点，同时向后遍历，**当p2为空，则p1为倒数第k个节点**

* 删除链表重复数据

操作跟冒泡排序差不多，**只要它相等，就能删除了**～

* 查询链表的中间节点

这个算法也挺有趣的：一个每次走1步，一个每次走两步，走两步的遍历完，然后走一步的指针，那就是**中间**节点

* 递归从尾到头输出单链表

只要下面还有数据，那就往下找，**递归是从最后往前翻**。

* 反转链表

有递归和非递归两种方式，我觉得是挺难的。可以到我给出的链接上查阅～

参考：

http://www.cnblogs.com/whgk/p/6589920.html