# 1.评判一个算法的好坏？

## 大O表示法（Big O）

它表示的是数据规模为 n时所 对应的复杂度。

**原则：忽略常数、系数、阶数：**

|  |
| --- |
| 9 >> O(1)  2n + 3 >> O(n)  n2 + 2n + 6 >> O(n2)  4n3 + 3n2 + 22n + 100 >> O(n3)  对数阶一般省略底数，对数阶乘以一个常数，可以替换底数。  log29 ∗ log9n= log2n  所以 O(log2n) 、O(log9n) 统称为 O(logn) |

## 常见的复杂度





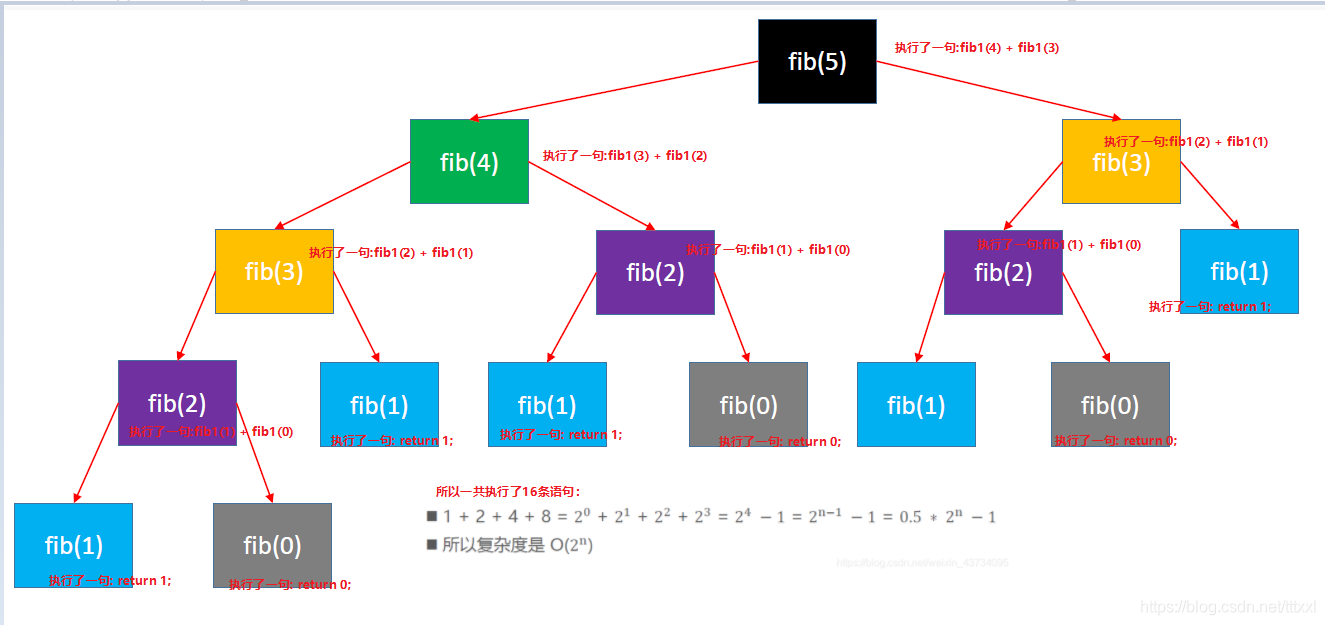
## 多个数据规模的情况

时间复杂度：O(n + k)

|  |
| --- |
| public static void test(int n, int k){  for(int i = 0; i < n; i++){  System.*out*.println("test");  }  for (int i = 0; i < k; i++){  System.*out*.println("test");  } } |

## 递归情况复杂度分析

|  |
| --- |
| public static int fib1(int n) {  if (n <= 1) return n;  return *fib1*(n - 1) + *fib1*(n - 2); } |

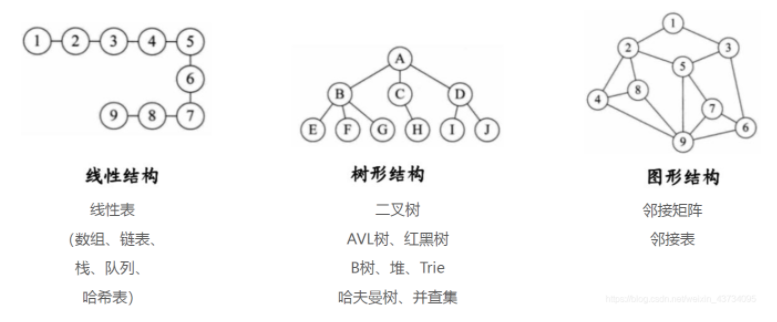


## 算法的优化方向

* 用尽量少的存储空间
* 用尽量少的执行步骤（执行时间）
* 根据情况，可以
  + 空间换时间
  + 时间换空间

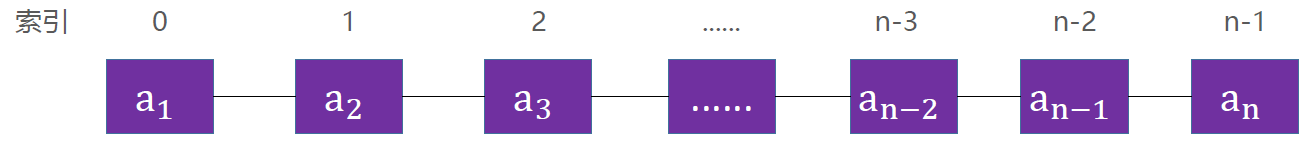
# 2.什么是数据结构？

数据结构是计算机存储、组织数据的方式；



在实际应用中，根据使用场景来选择最合适的数据结构；

## 1.线性表

* 线性表是具有 n 个相同类型元素的有限序列（ n ≥ 0 ）；
* **在逻辑上具有索引和数据两个部分**：是否能通过索引能直接访问到数据部分，要看具体的线性表类型（典型代表：数组和链表）。  
  

a1 是首节点（首元素）， an 是尾结点（尾元素）

a1 是 a2 的前驱， a2 是 a1 的后继

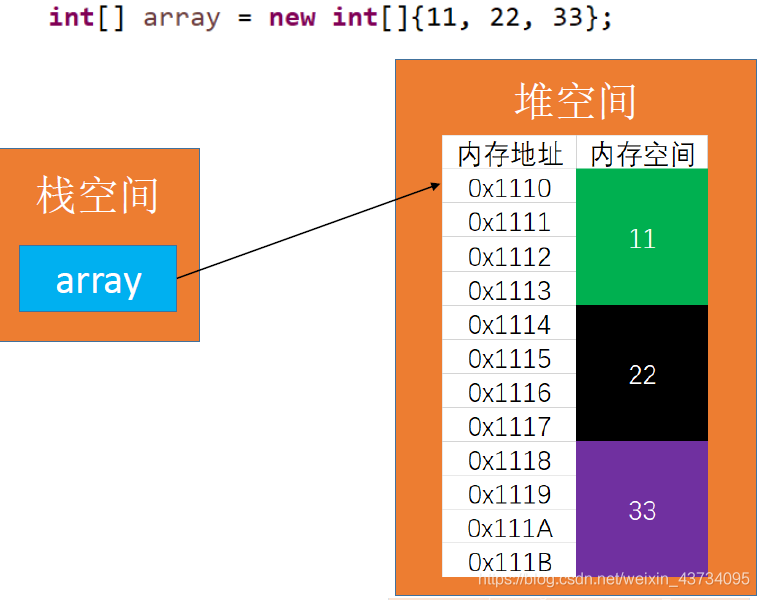
常见的线性表有：

* 数组
* 链表
* 栈
* 队列
* 哈希表（散列表）

# 线性表

## 数组

数组是一种顺序存储的线性表，所有元素的****内存地址是连续****的；



数组都有个致命的缺点 ：

数组array初始化后，其容量大小就是固定好了，以后无法动态修改数组容量。数组元素存满后，就无法再存放数据了，因为不能扩大数组的大小了。

## 动态数组（Dynamic Array）

实际开发中，我们更希望数组的容量是可以动态改变的，所以我们会自己实现一个可以动态扩容的数组。那么我们需要考虑一个问题：我们自己实现的可扩容的数组需要提供哪些API接口给别人调用？

### 2.1动态数组接口设计

|  |
| --- |
| int size(); *// 元素的数量* boolean isEmpty(); *// 是否为空* boolean contains (E element); *// 是否包含某个元素* void add (E element); *// 添加元素到最后面* E get (int index); *// 返回index位置对应的元素* E set (int index, E element); *// 设置index位置的元素* void add (int index, E element); *// 往index位置添加元素* E remove (int index); *// 删除index位置对应的元素* int indexOf (E element); *// 查看元素的位置* void clear (); *// 清除所有元素* |

## 链表

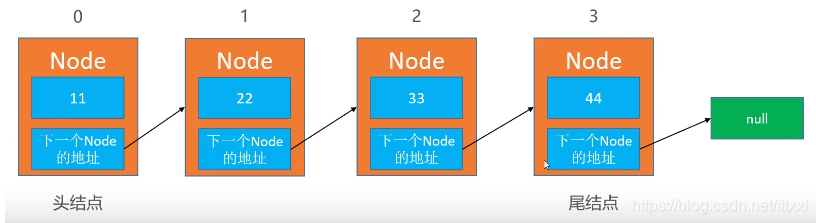
### 3.1.动态数组的缺点

1.底层是数组，容量未满造成空间浪费

2.扩容机制繁琐 O(n)

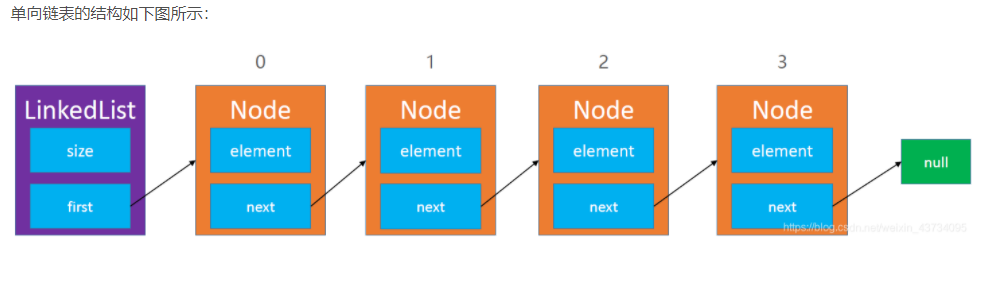
3.add(index,element)的时候，index<size的时候，要移动元素

### 3.2链表的结构



### 3.3单向链表设计

线性表的特征为元素序列，存储数据的节点之间有前后顺序；对比动态数组，ArrayList底层的数据存储结构是一个Object数组；链表的设计也需要有先后顺序，那么链表的底层存储数据的结构可以设计为一个Node类，最少需要有两个属性，一个存储数据，一个存储下一个节点的地址。那么LinkedList就是封装了操作Node的各种方法的一个类。



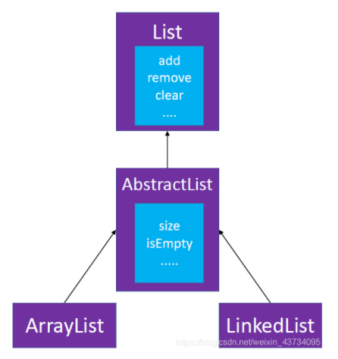
* 需要一个size属性，表明链表中有多少节点；
* 需要一个first属性，指向第一个结点；
* 结点Node只有LinkedList类用到，所以定义成LinkedList的内部类；
* LinkedList和Node也应该是泛型的，而且两者泛型应该一致；
* Node类中有element属性，表示存储的数据部分；
* Node类中有next属性，指向下一个Node节点；
* **单向链表类原型：**

|  |
| --- |
| package \_01\_线性表;  public class LinkedList<E> extends AbstractList<E>{  //1.指向第一个节点的 底层结构  private Node<E> first;   //2.一个内部类 存储数据和指针的节点  private class Node<E>{  E element;  Node<E> next;   public Node(E element){  this.element = element;  this.next = null;  }   public Node(E element,Node<E> next){  this.element = element;  this.next = next;  }  } } |

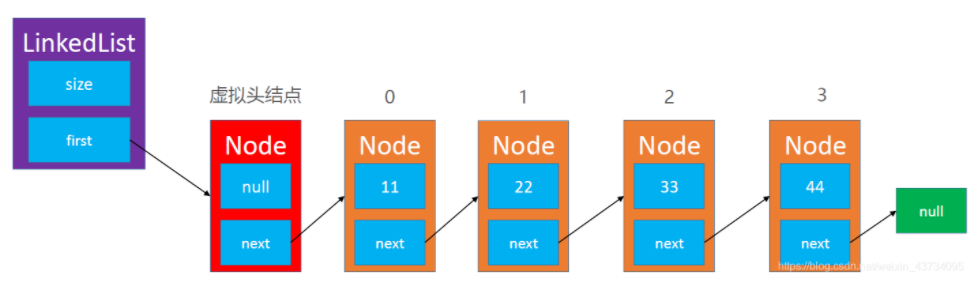
|  |
| --- |
| 关于index对应的node的寻找经验：  有时一直纠结遍历到哪才是第index位置的node  如下代码： current初始位置为first，对应的index=0  从初始状态来看，current=first，此时i=0,是对应的上的  For循环结束的状态，i是等于index的，此时current对应的就是index的节点。 private Node<E> node(int index){  rangeCheck(index);  Node<E> current = first;  for(int i = 0;i < index;i++){  current = current.next;  }  return current; } |

### 3.4链表的接口设计

链表也是线性表，因此实现了List接口。



### 3.5 带虚拟头结点的单向链表



### 3.6动态数组、链表复杂度分析

数组的****随机访问速度****非常快：elements[n] 的效率与 n 是多少无关；  
