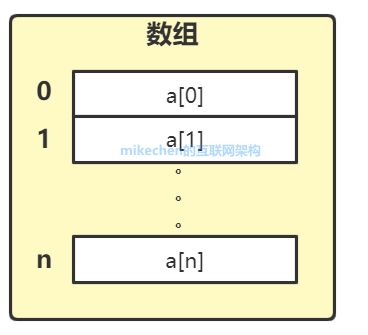
# 数组

## 基本概念

数组是一种最常见的数据结构，用来存储同一类型的集合，每个数组元素可以通过一个下标来访问它们，具体如下图所示



## 特点

1、在内存中申请一块连续的空间。

2、数组下标从0开始。

3、数组的类型只能是一个，且固定，在声明时确定。

4、数组按照索引查询元素，速度很快，时间复杂度为O(1)。

5、添加、删除元素的操作很耗时间，因为要移动其他元素，时间复杂度为O(n)。

# 链表

## 基本概念

一种物理存储结构上非连续**存储结构**，数据元素的逻辑顺序是通过链表中的引用链接次序实现的。链表由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域。相比于线性表顺序结构，操作复杂。由于不必须按顺序存储，链表在插入的时候可以达到O(1)的复杂度，比另一种线性表顺序表快得多，但是查找一个节点或者访问特定编号的节点则需要O(n)的时间，而线性表和顺序表相应的时间复杂度分别是O(logn)和O(1)。

## 分类

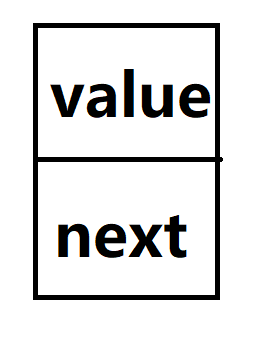
单向链表、双向链表

带头链表、不带头链表

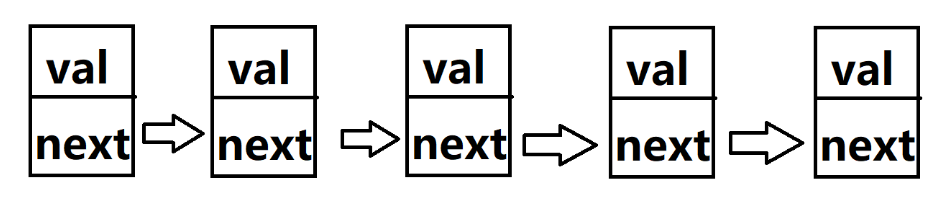
循环链表、非循环链表

## 结构

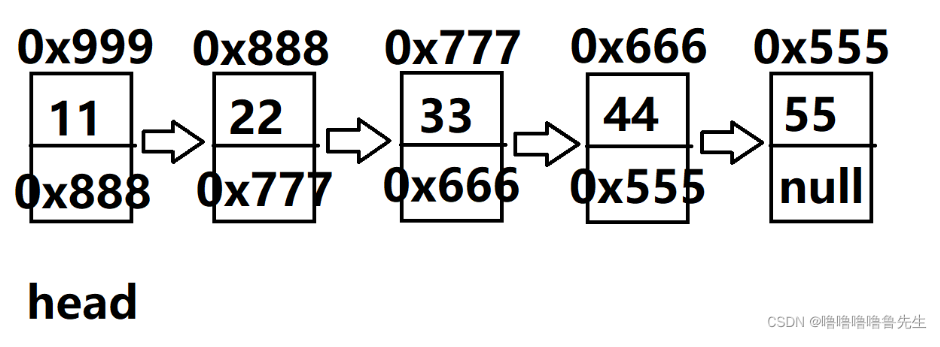
对于链表的结构，可以用如下这个图来模拟。



图中所示为链表的一个节点，value是这个节点所存储的数据值，next为下一节点的地址。下面是一个5个节点的链表。



我们来实现这样的链表的**增删查改**



第一个节点，地址假设是0x999，存储的数据是11，next存储的是下一个节点的地址(假设是0x888)

第二个节点，地址假设是0x888，存储的数据是22，next存储的是下一个节点的地址(假设是0x777)

第三个节点，地址假设是0x777，存储的数据是33，next存储的是下一个节点的地址(假设是0x666)

第四个节点，地址假设是0x666，存储的数据是44，next存储的是下一个节点的地址(假设是0x555)

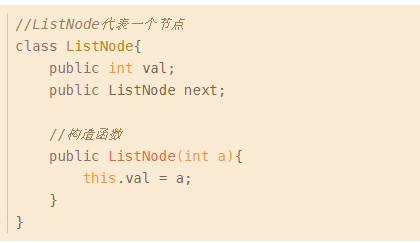
第五个节点，地址假设是0x555，存储的数据是55，由于没有后续节点，next存储的是空指针null

定义一个head，存储头节点（第一个节点）的地址（假设为0x999）。

## 代码实现链表

### 1、创建节点类

节点由val域(**数据域**)，以及next域(**指针域**)组成，对于next域，其是引用类型，存放下一个节点的地址，故用public ListNode next来创建next。同时设置构造函数，方便对val进行初始化。



### 2、创建链表

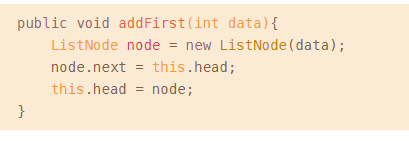
#### 方法一：枚举法



直接进行val的赋值以及对next的初始化。不用对最后一个节点的next进行赋值，因为next是引用类型，不赋值则默认为null。

#### 方法二：头插法

头插法是指在链表的头节点的位置插入一个新节点，定义一个node表示该节点，然后就是对node的next进行赋值，用node.next = this.head即可完成（注意：head应指向新节点）



# 栈

# 队列

# 双端队列

# 二叉树

图示

描述已自动生成

上图数字 1、3、7是叶子节点；（因为他们下面没有分叉出子节点，所以称为：叶子节点）【度为0】

数字2、8是子节点； （除了根节点、叶子节点之外的，都称为：子节点）【度为1】

数字5是根节点；（因为他是最顶部，所以称为：根节点）【度为2】

## 根节点

根节点：树的最顶端的节点。（根节点只有一个）

## 子节点

子节点：除根节点之外，并且本身下面还连接有节点的节点。

## 叶子节点

叶子结点：自己下面不再连接有节点的节点（即末端），称为叶子节点（又称为终端结点）。度为0

# 平衡二叉树（AVL树）

# B树（B-树）

# B+树

# 红黑树

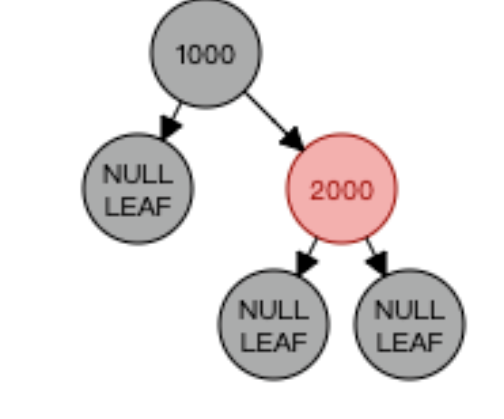
## 基本概念

红黑树，Red-Black Tree 「RBT」是一个自平衡(不是绝对的平衡)的二叉查找树(BST)。红黑树是一种特化的AVL树（[平衡二叉树](https://links.jianshu.com/go?to=https%3A%2F%2Fbaike.baidu.com%2Fitem%2F%25E5%25B9%25B3%25E8%25A1%25A1%25E4%25BA%258C%25E5%258F%2589%25E6%25A0%2591%2F10421057)），都是在进行插入和删除操作时通过特定操作保持二叉查找树的平衡，从而获得较高的查找性能。它虽然是复杂的，但它的最坏情况运行时间也是非常良好的，并且在实践中是高效的： 它可以在O(log n)时间内做查找，插入和删除，这里的n 是树中元素的数目。

## 性质

红黑树是一种含有红黑结点并能自平衡的二叉查找树。它必须满足下面性质：

1、每个节点要么是黑色，要么是红色。



2、根节点是黑色。

3、每个叶子节点（NIL）是黑色。

4、每个红色结点的两个子结点一定都是黑色。

5、任意一结点到每个叶子结点的路径都包含数量相同的黑结点。（保证这棵树尽量是平衡的。）

## 优点

1、红黑树是效率相对较高的，当我们插入和删除数据相对频繁的时候。

2、红黑树是自我平衡的，所有操作的复杂度最多是O(logn)。

3、不管怎么变化，只有红黑两个常数。