**第二章 链表**

# 1.1链表的介绍

链表是物理存储单元上非连续、非顺序的存储结构，链表的结点间通过指针相连，每个结点只对应有一个前驱结点和一个后继结点，其中，首结点没有前驱结点，尾节点没有后继结点。

链表主要有：单链表、单项循环链表、双向链表、双向循环链表。

链表有哪些特征：

① 链表是以节点的方式存储的，是链式存储；

② 每个节点包含data域和next域，next域指向下一个节点；

③ 链表的各节点不一定是连续存储的；

④ 链表分为带头节点的和没有头结点的链表；

# 1.2链表的定义

链表由头结点+有效节点构成，其中有效节点由分为首结点、普通节点、尾节点。

头结点：链表中有效结点前面的结点，头结点并不存放实际数据，仅仅是为了方便对链表的操作。

头指针：头结点并没有什么实际的含义，仅仅是为了方便对链表的操作，而头指针，顾名思义，就是指向头结点的指针。我们通过头指针指向的头结点的指针域对链表进行操作。

首结点：链表的第一个有效的结点。

尾结点：链表中最后一个有效的结点，尾结点的指针域为空。

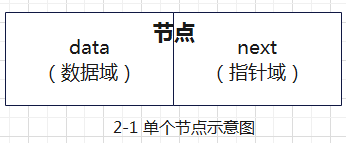
Java实现链表主要依靠引用传递，引用可以理解为地址；

如下为单向链表：

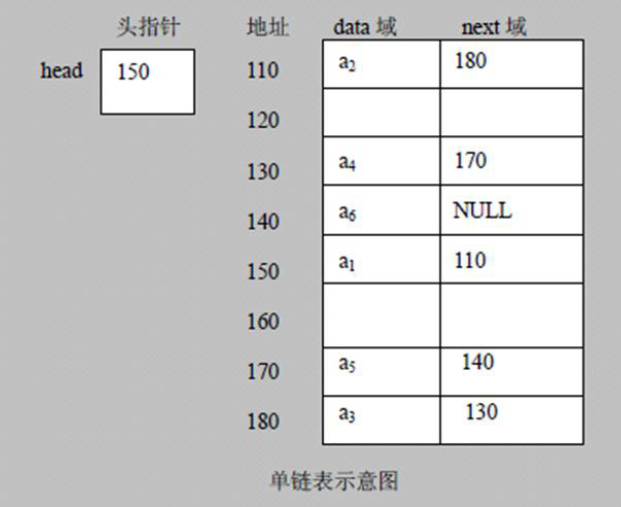
**public class** Node {  
 **private** String **data**;  
 **private** Node **next**;  
}

# 1.3链表的图释

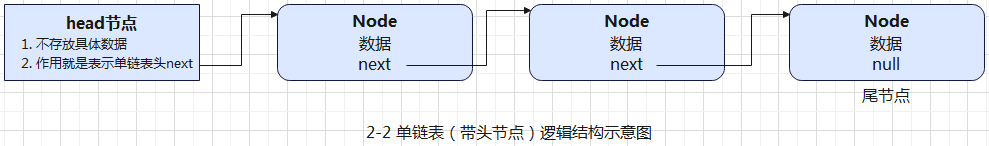
# 单个节点示意图：



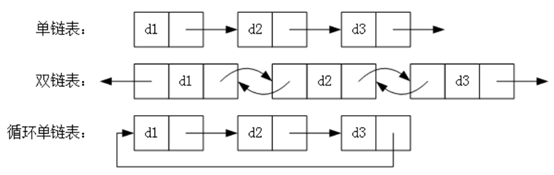
链表的存储示意图：



链表的逻辑示意图：



单链表、双链表、环形链表示意图：



# 1.4常用方法介绍

**1.4.1常用方法 – 单链表**

1) 新增：不仅可以在链表最后插入新的节点，还可以在指定位置插入新节点；

2) 修改：先找到该节点，然后修改节点数据即可；

3) 删除：先找到需要删除的前一个节点temp，然后让temp.next = temp.next.next,最后被删除的节点将不会被其他引用指向，最终被垃圾回收机制回收；

4) 遍历：通过一个辅助节点来帮助遍历整个链表；

**1.4.2代码示例**

|  |
| --- |
| **class** SingleLinkedList {  **private** Node **head** = **new** Node(0,**""**);   */\*\*  \* 添加节点到单链表  \** ***@param node*** *待添加节点  \*/* **public void** add(Node node) {  Node temp = **head**;  **while** (temp.**next** != **null**){  temp = temp.**next**;  }  temp.**next** = node;  }   */\*\*  \* 遍历链表  \*/* **public void** list() {  *// 判断是否为空链表* **if** (**head**.**next** == **null**) {  System.***out***.println(**"链表为空！"**);  **return**;  }   Node temp = **head**.**next**;  **while** (temp.**next** != **null**) {  System.***out***.println(temp); *// 打印节点* temp = temp.**next**;  }  }   */\*\*  \* 修改：修改指定id的节点数据  \** ***@param node*** *\*/* **public void** update(Node node) {  **if** (**head**.**next** == **null**) {  System.***out***.println(**"链表为空！"**);  **return**;  }   Node temp = **head**.**next**;  **boolean** flag = **false**;*// 表示是否找到待修改的节点* **while** (temp.**next** != **null**) {  **if** (temp.**id** == node.**id**) {  flag = **true**;*// 找到待修改节点* **break**;  }  temp = temp.**next**;  }   **if** (flag) {  temp.**data** = node.**data**;  } **else** {  System.***out***.println(**"没有指定的节点！"**);  }  }   */\*\*  \* 删除节点：删除指定id的节点  \** ***@param id*** *待删除数据的标志  \*/* **public void** delete(**int** id) {  Node temp = **head**;  **boolean** flag = **false**;*// 标识是否找到带删除节点* **while**(temp.**next** != **null**) {  **if** (temp.**next**.**id** == id) {  flag = **true**;  **break**;  }  temp = temp.**next**;  }   **if** (flag) {  temp.**next** = temp.**next**.**next**;*// 删除操作* } **else** {  System.***out***.println(**"没有找到指定节点！"**);  }  } }  */\*\*  \* 节点实体  \*/*  **public class** Node {  **public** Integer **id**;  **public** String **data**;  **public** Node **next**;   **public** Node(**int** id, String data) {  **this**.**id** = id;  **this**.**data** = data;  } } |

# 1.5常见误区洞察

1) 指针/引用，实际上都是存储所指对象内存。将某个变量赋值给指针，实际上就是将这个变量的地址赋值给指针，或者反过来说，指针中存储了这个变量的内存地址，指向了这个变量，通过指针就能找到这个变量。

2) 插入节点时，注意操作顺序；删除节点时，注意释放空间。

# 1.6优缺点漫谈

1)优点

* 插入和删除速度快，保留原有的物理顺序，在插入或者删除一个元素的时候，只需要改变指针指向即可；
* 没有空间限制,存储元素无上限,只与内存空间大小有关；
* 动态分配内存空间，不用事先开辟内存；
* 内存的利用率变高。

2)缺点

* 占用额外的空间以存储指针，比较浪费空间，不连续存储，malloc函数开辟空间碎片比较多)；
* 查找速度比较慢，因为在查找时，需要循环链表；

查找操作为O(n) ,插入和删除操作为O(1)；

# 1.7解题技巧/源码分析/使用场景