

18 | 重新认识数据结构（上）：初识链表结构

2020-02-27 胡光

人人都能学会的编程入门课

[进入课程 >](#)



讲述：胡光

时长 15:46 大小 12.65M



你好，我是胡光，欢迎来到“算法数据结构篇”的第一课。

在之前的学习中，我们对数据结构的认识，主要集中在它是用来做数据的表示，更具体地讲，就是数据结构所讨论的问题，是将现实中的数据如何合理地表示在程序中，以使程序完成既定的目标任务。

不知道你还记不记得，在上节课 Shift-And 算法的学习中，我们发现不同的数据，或信息表示方式，会给解决问题的效率带来很大的影响。因此，基本确定了数据的表示，在程序设计过程中发挥着非常重要的作用，也就意味着我们必须对数据结构的学习重视起来。



之前我们所讨论的数据结构呢，其实都只是停留在程序内部的数据结构，这是一种具体的，可见的数据结构，并对我们设计的程序产生重要影响。我们也认识到，这种具体的数据结构的重要作用，会对我们设计的程序产生重要的影响。今天呢，我将带你重新认识数据结构，发现它的另一面，那就是数据结构对我们思维方式的影响，这种影响更抽象，也更重要。

接下来的两次课程内容呢，我将通过链表结构的讲解，让你认识这种思维层面的数据结构。

必知必会，查缺补漏

今天我们将要学习的链表，是一种常见的基础数据结构，属于数据结构中线性结构的一种。在讲如何学习链表之前，我们先来看一看通常情况下，如何学习数据结构的相关知识。

1. 数据结构：结构定义 + 结构操作

你应该玩过拼装式的玩具吧，类似于高达机器人之类的。面对这样的玩具，我一般在拼装之前看看说明书，知道这个玩具包含哪几部分，然后对这些部分进行拼装，等把各部分拼好了后，再把它们组合起来，最终的成品就完成了。



图1：高达机器人

其实学习某样知识也是一样的，要先搞清楚这门知识的组成部分，从组成部分开始入手学习，最后把所有的知识碎片整合到一起，就是知识的全貌了。

回到如何理解数据结构这个问题，我先给你列出重要的两句话：

1. **数据结构 = 结构定义 + 结构操作**
2. **数据结构，就是定义一种性质，并且维护这种性质**

其实这两句话，说的是一回事儿。结构定义，指的是就是说明这种数据结构长成什么样子，具备什么性质。结构操作，就是确定这种数据结构都支持哪些操作，同时结构操作的结果，不能破坏这类结构原本的性质。这也就到了第二句话中说的内容，维护这种性质。

这就好像刚才我说到的高达机器人，结构定义类比高达机器人的样子，结构操作就是这个机器人都支持什么功能，比如抬手、伸腿之类的。但是无论是哪种结构操作，你都不能把机器人玩坏掉（也就是不能破坏结构定义），这就是我们所说的：操作归操作，但是你要维护这种性质。

接下来呢，我将会通过这两句话，带你学习**链表**这种数据结构。

2. 链表的结构定义

链表的结构定义中，包含了两个信息，一个是数据信息，用来存储数据的，也叫做数据域；另外一个是指针信息，用来存储下一个节点地址的，也叫做指针域。

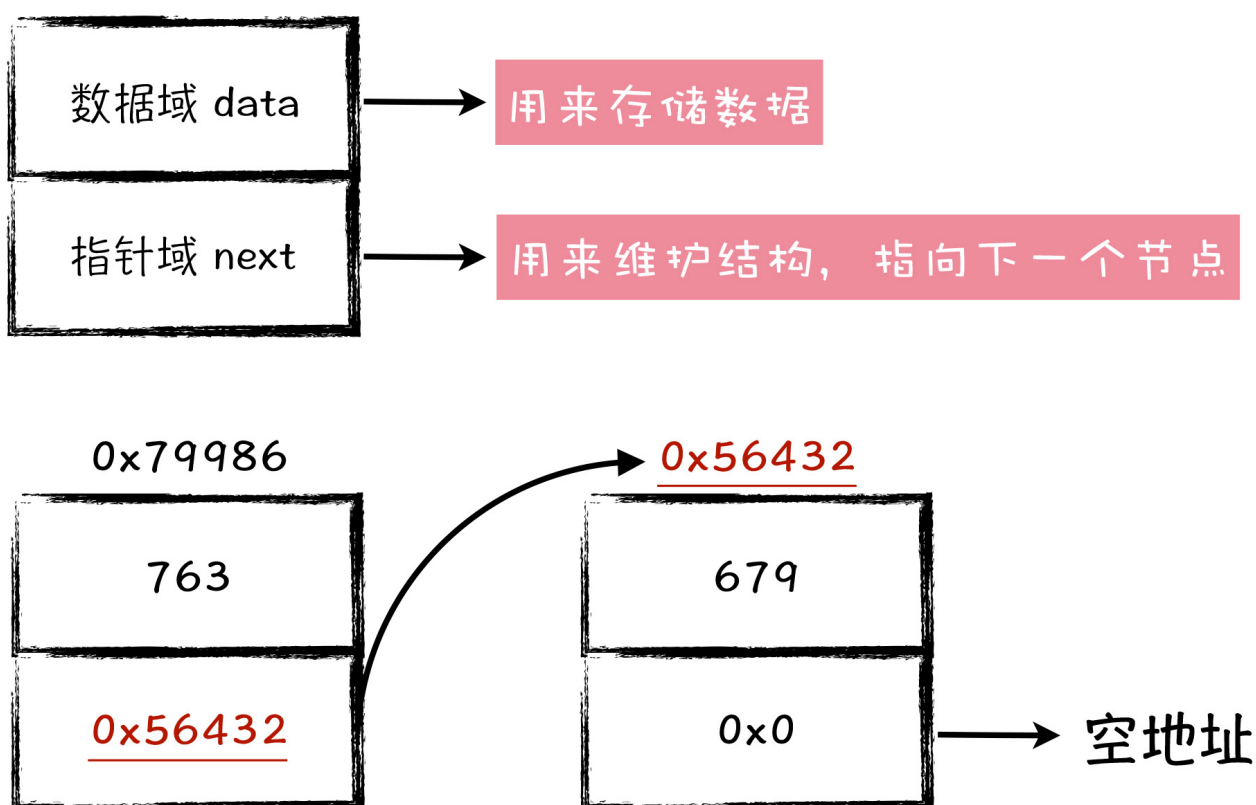


图2：链表结构定义


记住，链表结构是由一个一个链表节点组成的，在应用这种结构的时候，你无需对这种结构本身做改变，你只需要按照自己的需求，把自己想要的数

即可。比如说，整型是你想存储在链表中的数据，那么数据域的类型就是整型，如果字符串类型是你想存储的数据，那么数据域的类型就是字符串类型。

在示意图中可以看到，链表节点以整型作为数据域的类型，其中第一个链表节点，存储了 763 这个数据，指针域中呢，存储了一个 0x56432 地址，这个地址而 0x56432 正是第二个链表节点的地址。我们可以说，第一个节点指向第二个节点，因此这两个节点之间，在逻辑上构成了一个指向关系。

在第二个节点的指针域中呢，存储了一个地址，是 0x0，这个地址值所对应的整型值就是 0。这是一个特殊的地址，我们称它为空地址，在 C 语言中用 NULL 宏表示这个空地址，读作 nàio。我们让第二个链表节点指向空地址，就意味着它就是这个链表结构的最后一个节点。

看完了链表的结构示意图以后，就来让我们看一下在代码中，如何定义链表节点结构吧：

 复制代码

```
1 struct Node {  
2     int data;  
3     struct Node *next;  
4 };
```

正如这段代码所示，我们使用结构体，定义一种新的类型，叫做 struct Node 类型，来表示链表的节点结构。链表的每个节点内部，有一个数据域，一个指针域，对应到代码中，就是一个整型的 data 字段，和一个指向 struct Node 类型本身的指针字段 next。

值得注意的是，链表结构的指针域只有一个 next 变量，这就说明每一个链表节点，只能唯一地指向后续的一个节点，这是链表结构非常重要的一个性质，后续我们也会用到这个性质。

总地来说，链表结构中，数据域是留出来让我们实现自我需求的，就是想存整型，就改成整型，想存浮点型，就改成浮点型。而 next 指针域，是用来维护链表这个结构的，这里一般不需要你自由发挥，记住怎么回事儿，直接用就行了。**记住，要想修改内存中的链表结构，就一定要修改节点内部 next 指针域中存储的地址值。**

3. 链表的结构操作

接下来呢，我会给你介绍一种链表的基础操作，就是向链表中插入节点的操作。

在讲解链表的插入和删除方法之前呢，我们先来对齐一个概念，就是**链表节点的位置**。当你把链表结构画出来以后，你会发现链表结构和数组结构很类似，只不过数组结构在内存中存储是连续的，链表结构由于有指针域的存在，它的每一个节点在内存中存储的位置未必连续。

我们也可以参考数组下标的编号规则，给每个链表节点编一个号，从第一个开始依次是 0、1、2，具体如下图所示：

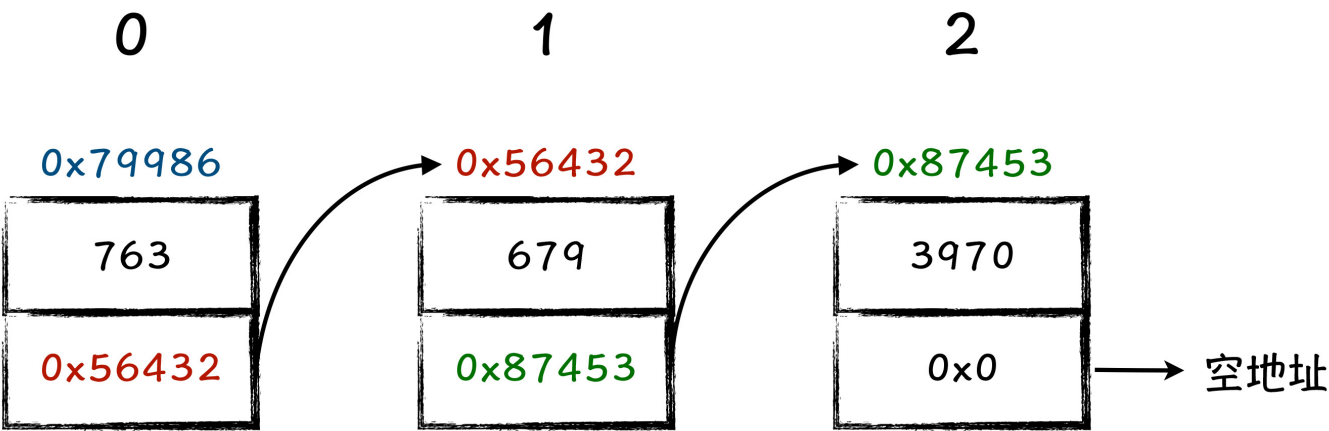


图3：链表节点位置定义

明白了什么是链表的节点位置以后呢，我们定义一个向链表中插入节点的函数方法：

复制代码

```
1 struct Node *insert(struct Node *head, int ind, struct Node *a);
```

这个插入方法呢，传入三个参数，第一个是待操作的链表的头结点地址，也就是链表中第一个节点的地址；第二个参数代表插入位置；第三个参数是一个指针变量，指向要插入的新节点。

简单来说，就是向 head 所指向的链表的 ind 位置插入一个由 a 所指向的节点，返回值代表插入新节点以后的链表头结点地址。为什么要返回插入以后的链表头结点地址呢？因为新节点有可能插入到链表的第 0 位，插入以后，链表的头结点地址就发生了改变，我们必须把这个信息返回。

由于插入操作，会改变链表结构，刚刚我们说了，只有修改链表节点中的 next 指针域的值，才算是修改了链表的结构。为了完成插入操作，我们都需要修改哪些节点的 next 指针域的值呢？

首先是让 ind - 1 位置的节点指向 a 节点，然后是 a 节点指向原 ind 位置的节点，也就是说，涉及到两个节点的 next 指针域的值修改，一个是 ind - 1 位置的节点，一个是 a 节点自身。我们就可以先找到 ind - 1 位置的节点，然后再进行相关操作即可。写成代码，如下所示：

 复制代码

```
1 struct Node *insert(struct Node *head, int ind, struct Node *a) {
2     struct Node ret, *p = &ret;
3     ret.next = head;
4     // 从【虚拟头节点】开始向后走 ind 步
5     while (ind--) p = p->next;
6     // 完成节点的插入操作
7     a->next = p->next;
8     p->next = a;
9     // 返回真正的链表头节点地址
10    return ret.next;
11 }
```

代码中，涉及到一个很重要的技巧，就是“虚拟头结点”这个链表操作技巧。所谓虚拟头结点，就是在原有的链表头结点前面，加上另外一个节点，这个额外增加的头结点，就是虚拟头结点。增加虚拟头结点的目的，是为了让我们操作链表更方便，实际上，如果在某个操作中，头结点地址有可能发生改变，我们也可以使用虚拟头结点这个技巧。

我们分析一下，对于插入操作，虚拟头结点有什么重要的作用。首先如果我们要在第 5 个位置插入新节点，那我们就要找到 4 号位的节点，也就是从头结点开始，向后走 4 步，确定了 4 号节点以后，再修改相关节点的 next 指针域即可。

也就是说，如果我们想插入到 ind 位，就需要从头结点向后走 ind - 1 步，定位到 ind - 1 号节点。如果插入的位置为 0 呢？我们总不能走 -1 步吧？这个时候，在程序中我们就只能对 ind 等于 0 的情况进行特殊判断了。这确实是一种可行的实现方法，可不够优美，因为这种做法没有统一 ind 在等于 0 和 不等于 0 时候的处理情况。

可是当我们在原链表前面，加入了一个虚拟头结点以后，这一切的操作就变得自然了许多！一开始 p 指向虚拟头结点，由于链表头部增加了一个节点，原先我们要定位链表 ind - 1 位置，要走 ind - 1 步，现在就是走 ind 步。

也就是说，在有虚拟头结点的情况下，如果我们插入到 5 号位，就从虚拟头结点向后走 5 步就行，同样的，想要插入到 0 号位呢，就向后走 0 步即可，即 p 指针指向虚拟头结点不动，直接将新的节点，插入到虚拟头结点后面即可。

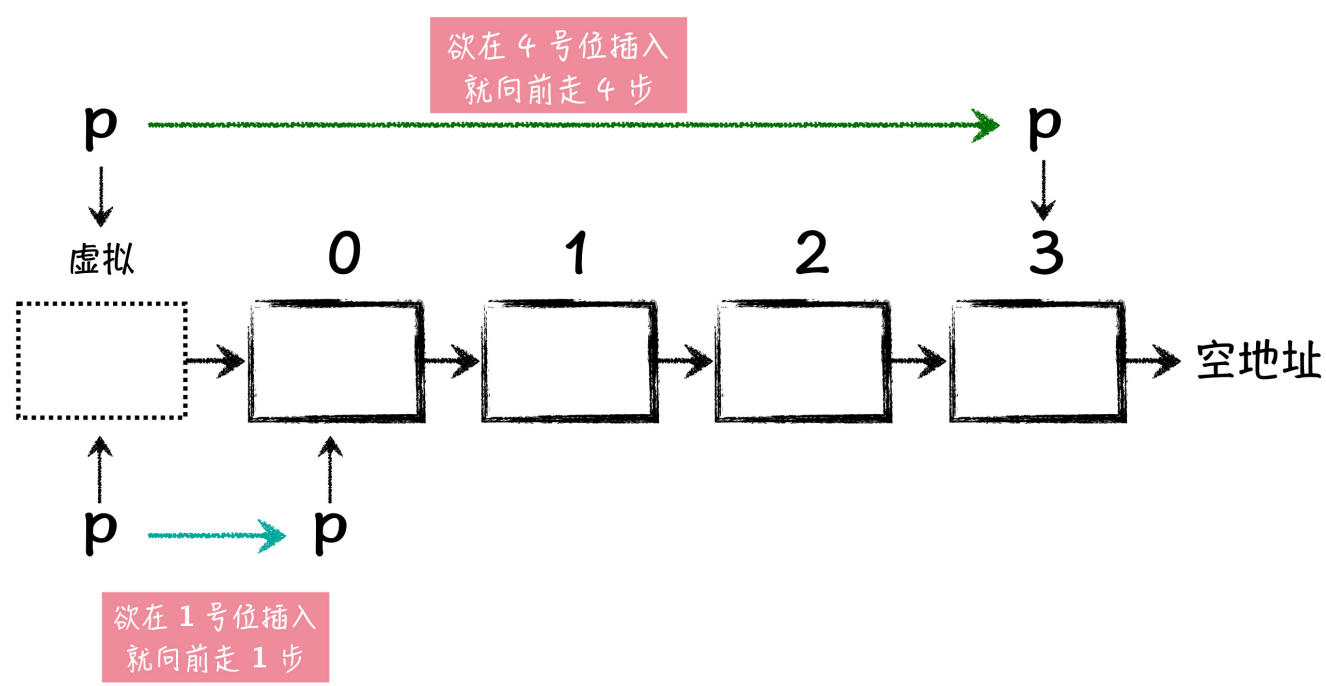


图4：虚拟节点示意图

其实，对于链表的相关操作，无论是插入还是删除，只要是有可能改变原有链表头结点的操作，增加虚拟头结点都是一个很实用的处理技巧。

一起动手，搞事情

今天给你留的作业呢，与链表的操作有关系，请看如下函数接口定义：

```
1 struct Node *erase(struct Node *head, int ind);
```

复制代码

请你参照文中的链表插入操作，实现一个链表节点删除的操作，删除函数传入两个参数，分别代表指向链表头结点的指针变量 head，以及要删除的节点位置 ind，返回值代表删除节点以后的链表头结点地址。

由于删除操作，有可能改变链表的头结点，所以你可以尝试使用前面我们讲到的虚拟头结点的编码技巧。仔细分析，你可以的！

课程小结

我们今天介绍的链表呢，其实真实姓名叫做“单向链表”，这是一种很有代表性的链表结构。实际上，你学会了单向链表，也就很容易掌握其他形式的链表了。比如说：单向循环链表、双向链表、双向循环链表、块状链表、跳跃表。

尤其是块状链表和跳跃表，在工程中应用最广泛，C++ STL 中的 vector 底层用的就是块状链表。关于这些概念，如果你感兴趣，可以自行上网搜索相关资料。篇幅有限，我们就不一个个展开介绍了。

最后，我们来做一下今天课程的总结，今天我希望你记住如下几点：

1. 数据结构 = 结构定义 + 结构操作，这个等式说明了我们学习数据结构的方法顺序。
2. 单向链表节点中，存在数据域和指针域，指针域控制了链表的结构，一般不会根据应用场景的变化而变化，而数据域是根据应用场景的需求而设计的。

下节课呢，我将给你讲几种更有意思的链表操作。好了，今天就到这里了，我是胡光，咱们下期见。

关注极客时间服务号
每日学习签到



月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 [做好闭环（三）：编码能力训练篇的思考题答案都在这里啦！](#)

下一篇 19 | 重新认识数据结构（下）：有趣的“链表”思维

精选留言 (8)

写留言



HappyJoo

2020-02-29

不知道有没有用，“画”了一个图给对于插入不太懂的同学看看哈哈哈哈哈：

```

----p----p.next----| ----> |----p p.next----
                    | ----> | \ /
                    | ----> | \ /...

```

展开 ∨

作者回复: d(^ ^o)给个封号: 最强助教!



徐洲更

2020-02-27



👍 1

python里的list可以存放不同数据类型，也是用链表实现的嘛，C语言如何写出这种数据结构呢

作者回复: python中的list不是用链表实现的，那样的话，查找效率太差了，C中的话，可以用指针数组实现，指针类型是void *即可指向任意一种类型数据。



1



胖胖胖

2020-02-29

作业打卡：

```
#include<stdio.h>
struct Node {
    int data;
    struct Node *next;...
```

展开 ∨

作者回复: 对，实际上是需要手动释放的，如果是 C 的话，你可以使用 free 来进行释放。没有强调释放的原因是，咱们这个不是一份完整的链表代码，没有从创建开始讲，所以突然提到释放，就会很突兀。你可以想到这个问题，是很棒的！



HappyJoo

2020-02-29

作业: https://github.com/HappyJoo/CLearningScript/blob/master/Linked-list/2_Erase_Node.cpp

但是暂时还不清楚如何使用这个函数，等会了再来update哈~老师辛苦啦~

作者回复: 代码中 ret 的类型，不是一个指针类型哦，而应该是一个实实在在的节点类型，虚拟节点，本身就是一个节点，把前面的 * 去掉就好了。



HappyJoo

2020-02-29

老师您好，请问一下，这个虚拟头结点是如何不影响链表的呢？我的理解是，每次执行插入函数（这个*insert是函数吧。C语言的函数就是方法？）时，都创建一个虚拟结点（计算机自动创造吗？），让它指向链表的真正头结点，执行函数。函数执行结束后，将虚拟

结点与真正头结点断开，这样子就可以做到“虚拟”了。请问老师我的理解有哪些不对的地方吗？谢谢老师~~~

展开 ∨

作者回复: 当然不是计算机自动创造的，而是我们通过代码逻辑加上去的。所谓虚拟，就是只有在操作过程中，存在的一个节点，这个节点完全是为了操作简便，额外加上去的，不是链表的实际节点。



doge

2020-02-28

```
struct Node* erase(struct Node *head, int ind){
    struct Node ret, *p = &ret, *q = NULL;
    if (ind < 1){
        err("invalid position %d\n", ind);
        return head;...
```

展开 ∨

作者回复: 非常棒！还实现了非法情况判断！d(^_^o)



一步

2020-02-27

// 删除指定索引位置的节点

```
struct Node *erase(struct Node *head, int ind) {
    struct Node preNode, *p = &preNode;
    preNode.next = head;
    while (ind--) {...
```

展开 ∨

作者回复: 完美！d(^_^o)



宋jia wen

2020-02-27

```
struct Node *insert(strcut Node *head, int ind, struct Node *a)
```


老师这是 struct Node 型指针 还是 一个函数

展开 ▾

作者回复: 这是一个函数, 返回值是struct Node 类型的地址。

