阅读：17,455　　作者：解学武

双向链表及其创建（C语言）详解

上一节　　下一节

目前我们所学到的链表，无论是动态链表还是静态链表，表中各节点中都只包含一个指针（游标），且都统一指向直接后继节点，通常称这类链表为单向链表（或单链表）。

虽然使用单链表能 100% 解决逻辑关系为 "一对一" 数据的存储问题，但在解决某些特殊问题时，单链表并不是效率最优的存储结构。比如说，某场景中需要大量地查找某结点的前趋结点，这种情况下使用单链表无疑是灾难性的，因为单链表更适合 "从前往后" 找，"从后往前" 找并不是它的强项。

对于逆向查找（从后往前）相关的问题，使用本节讲解的双向链表，会更加事半功倍。

双向链表，简称双链表。从名字上理解双向链表，即链表是 "双向" 的，如图 1 所示：

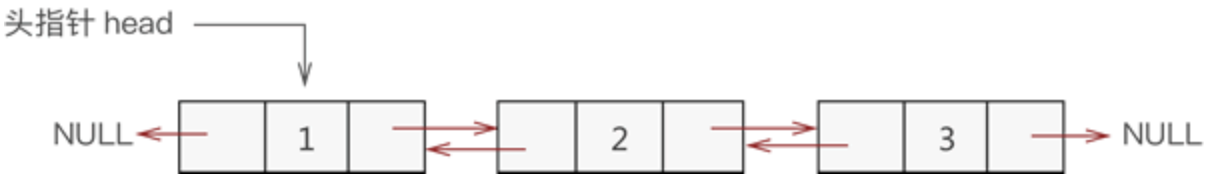

图 1 双向链表结构示意图

所谓双向，指的是各节点之间的逻辑关系是双向的，但通常头指针只设置一个，除非实际情况需要，可以为最后一个节点再设置一个“头指针”。

根据图 1 不难看出，双向链表中各节点包含以下 3 部分信息（如图 2 所示）：

1. 指针域：用于指向当前节点的直接前驱节点；
2. 数据域：用于存储数据元素；
3. 指针域：用于指向当前节点的直接后继节点。

图 2 双向链表的节点构成

因此，双链表的节点结构用 C 语言实现为：

```
01.  typedef struct line{
02.      struct line * prior; //指向直接前趋
03.      int data;
```

```
    struct line * next; //指向直接后继
}line;
```

> 读者可根据实际场景的需要，调整数据域 data 的数据类型。

双向链表的创建

同单链表相比，双链表仅是各节点多了一个用于指向直接前驱的指针域。因此，我们可以在单链表的基础轻松实现对双链表的创建。

和创建单链表不同的是，创建双向链表的过程中，每一个新节点都要和前驱节点之间建立两次链接，分别是：

- 将新节点的 prior 指针指向直接前驱节点；
- 将直接前驱节点的 next 指针指向新节点；

这里给出创建双向链表的 C 语言实现代码：

```
line* initLine(line * head) {
    int i = 0;
    line * list = NULL;
    //创建一个首元节点，链表的头指针为head
    head = (line*)malloc(sizeof(line));
    //对节点进行初始化
    head->prior = NULL;
    head->next = NULL;
    head->data = 1;
    //声明一个指向首元节点的指针，方便后期向链表中添加新创建的节点
    list = head;
    for (i = 2; i <= 5; i++) {
        //创建新的节点并初始化
        line * body = (line*)malloc(sizeof(line));
        body->prior = NULL;
        body->next = NULL;
        body->data = i;

        //新节点与链表最后一个节点建立关系
        list->next = body;
        body->prior = list;
        //list永远指向链表中最后一个节点
        list = list->next;
    }
    //返回新创建的链表
    return head;
}
```

我们可以尝试着在 main 函数中输出创建的双链表，C 语言代码如下：

```c
#include <stdio.h>
#include <stdlib.h>
//节点结构
typedef struct line {
    struct line * prior;
    int data;
    struct line * next;
}line;
//双链表的创建函数
line* initLine(line * head);
//输出双链表的函数
void display(line * head);

int main() {
    //创建一个头指针
    line * head = NULL;
    //调用链表创建函数
    head = initLine(head);
    //输出创建好的链表
    display(head);
    //显示双链表的优点
    printf("链表中第 4 个节点的直接前驱是：%d", head->next->next->next->prior->data);
    return 0;
}
line* initLine(line * head) {
    int i = 0;
    line * list = NULL;
    //创建一个首元节点，链表的头指针为head
    head = (line*)malloc(sizeof(line));
    //对节点进行初始化
    head->prior = NULL;
    head->next = NULL;
    head->data = 1;
    //声明一个指向首元节点的指针，方便后期向链表中添加新创建的节点
    list = head;
    for (i = 2; i <= 5; i++) {
        //创建新的节点并初始化
        line * body = (line*)malloc(sizeof(line));
        body->prior = NULL;
        body->next = NULL;
        body->data = i;

        //新节点与链表最后一个节点建立关系
        list->next = body;
```

```
45.          body->prior = list;
46.          //list永远指向链表中最后一个节点
47.          list = list->next;
48.      }
49.      //返回新创建的链表
50.      return head;
51.  }
52.  void display(line * head) {
53.      line * temp = head;
54.      while (temp) {
55.          //如果该节点无后继节点，说明此节点是链表的最后一个节点
56.          if (temp->next == NULL) {
57.              printf("%d\n", temp->data);
58.          }
59.          else {
60.              printf("%d <-> ", temp->data);
61.          }
62.          temp = temp->next;
63.      }
64.  }
```

程序运行结果：

```
1 <-> 2 <-> 3 <-> 4 <-> 5
链表中第 4 个节点的直接前驱是：3
```

上一节 下一节