阅读：68,947　　作者：解学武

链表（链式存储结构）及创建（C语言详解版）

上一节　　下一节

前面详细地介绍了顺序表，本节给大家介绍另外一种线性存储结构——链表。

链表，别名链式存储结构或单链表，用于存储逻辑关系为 "一对一" 的数据。与顺序表不同，链表不限制数据的物理存储状态，换句话说，使用链表存储的数据元素，其物理存储位置是随机的。

例如，使用链表存储 {1,2,3}，数据的物理存储状态如图 1 所示：

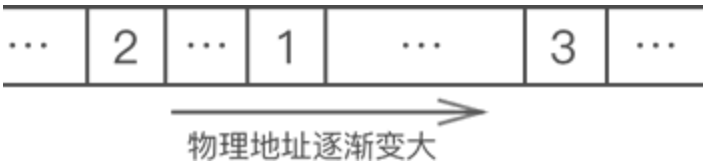

图 1 链表随机存储数据

我们看到，图 1 根本无法体现出各数据之间的逻辑关系。对此，链表的解决方案是，每个数据元素在存储时都配备一个指针，用于指向自己的直接后继元素。如图 2 所示：

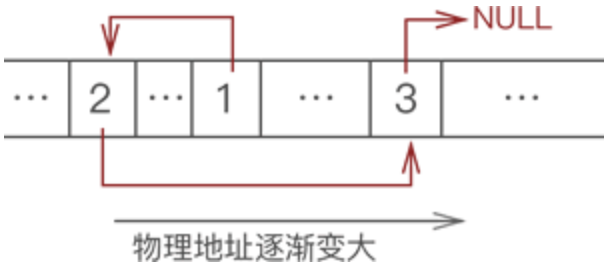

图 2 各数据元素配备指针

像图 2 这样，数据元素随机存储，并通过指针表示数据之间逻辑关系的存储结构就是链式存储结构。

链表的节点

从图 2 可以看到，链表中每个数据的存储都由以下两部分组成：

1. 数据元素本身，其所在的区域称为数据域；
2. 指向直接后继元素的指针，所在的区域称为指针域；

即链表中存储各数据元素的结构如图 3 所示：

图 3 节点结构

图 3 所示的结构在链表中称为节点。也就是说，链表实际存储的是一个一个的节点，真正的数据元素包含在这些节点中，如图 4 所示：

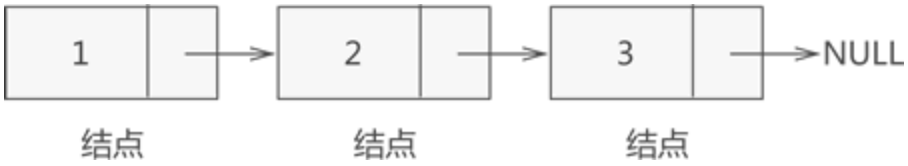

图 4 链表中的节点

因此，链表中每个节点的具体实现，需要使用 C 语言中的结构体，具体实现代码为：

```
01.  typedef struct Link{
02.      char elem; //代表数据域
03.      struct Link * next; //代表指针域，指向直接后继元素
04.  }link; //link为节点名，每个节点都是一个 link 结构体
```

提示，由于指针域中的指针要指向的也是一个节点，因此要声明为 Link 类型（这里要写成 `struct Link*` 的形式）。

头节点，头指针和首元节点

其实，图 4 所示的链表结构并不完整。一个完整的链表需要由以下几部分构成：

1. 头指针：一个普通的指针，它的特点是永远指向链表第一个节点的位置。很明显，头指针用于指明链表的位置，便于后期找到链表并使用表中的数据；
2. 节点：链表中的节点又细分为头节点、首元节点和其他节点：
   - 头节点：其实就是一个不存任何数据的空节点，通常作为链表的第一个节点。对于链表来说，头节点不是必须的，它的作用只是为了方便解决某些实际问题；
   - 首元节点：由于头节点（也就是空节点）的缘故，链表中称第一个存有数据的节点为首元节点。首元节点只是对链表中第一个存有数据节点的一个称谓，没有实际意义；
   - 其他节点：链表中其他的节点；

因此，一个存储 `{1,2,3}` 的完整链表结构如图 5 所示：

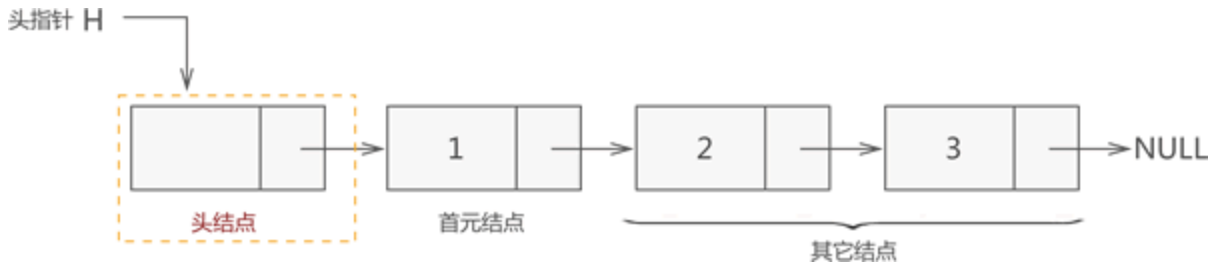

图 5 完整的链表示意图

注意：链表中有头节点时，头指针指向头节点；反之，若链表中没有头节点，则头指针指向首元节点。

明白了链表的基本结构，下面我们来学习如何创建一个链表。

链表的创建（初始化）

创建一个链表需要做如下工作：

1. 声明一个头指针（如果有必要，可以声明一个头节点）；
2. 创建多个存储数据的节点，在创建的过程中，要随时与其前驱节点建立逻辑关系；

例如，创建一个存储 {1,2,3,4} 且无头节点的链表，C 语言实现代码如下：

```
01.  link * initLink() {
02.      int i;
03.      link * p = NULL;//创建头指针
04.      link * temp = (link*)malloc(sizeof(link));//创建首元节点
05.      //首元节点先初始化
06.      temp->elem = 1;
07.      temp->next = NULL;
08.      p = temp;//头指针指向首元节点
09.      //从第二个节点开始创建
10.      for (i = 2; i < 5; i++) {
11.          //创建一个新节点并初始化
12.          link *a = (link*)malloc(sizeof(link));
13.          a->elem = i;
14.          a->next = NULL;
15.          //将temp节点与新建立的a节点建立逻辑关系
16.          temp->next = a;
17.          //指针temp每次都指向新链表的最后一个节点，其实就是 a节点，这里写temp=a也对
18.          temp = temp->next;
19.      }
20.      //返回建立的节点，只返回头指针 p即可，通过头指针即可找到整个链表
21.      return p;
22.  }
```

如果想创建一个存储 {1,2,3,4} 且含头节点的链表，则 C 语言实现代码为：

```
01.  link * initLink(){
```

```
    int i;
    link * p=(link*)malloc(sizeof(link));//创建一个头结点
    link * temp=p;//声明一个指针指向头结点，
    //生成链表
    for (i=1; i<5; i++) {
        link *a=(link*)malloc(sizeof(link));
        a->elem=i;
        a->next=NULL;
        temp->next=a;
        temp=temp->next;
    }
    return p;
}
```

我们只需在主函数中调用 initLink 函数，即可轻松创建一个存储 {1,2,3,4} 的链表，C 语言完整代码如下：

```
#include <stdio.h>
#include <stdlib.h>
//链表中节点的结构
typedef struct Link {
    int  elem;
    struct Link *next;
}link;
//初始化链表的函数
link * initLink();
//用于输出链表的函数
void display(link *p);

int main() {
    link*p = NULL;
    //初始化链表 (1, 2, 3, 4)
    printf("初始化链表为: \n");
    p = initLink();
    display(p);
    return 0;
}

link * initLink() {
    int i;
    link * p = NULL;//创建头指针
    link * temp = (link*)malloc(sizeof(link));//创建首元节点
    //首元节点先初始化
    temp->elem = 1;
    temp->next = NULL;
    p = temp;//头指针指向首元节点
    for (i = 2; i < 5; i++) {
```

```
31.          link *a = (link*)malloc(sizeof(link));
32.          a->elem = i;
33.          a->next = NULL;
34.          temp->next = a;
35.          temp = temp->next;
36.      }
37.      return p;
38.  }
39.  void display(link *p) {
40.      link* temp = p;//将temp指针重新指向头结点
41.      //只要temp指针指向的结点的next不是Null，就执行输出语句。
42.      while (temp) {
43.          printf("%d ", temp->elem);
44.          temp = temp->next;
45.      }
46.      printf("\n");
47.  }
```

程序运行结果为：

```
初始化链表为:
1 2 3 4
```

注意，如果使用带有头节点创建链表的方式，则输出链表的 display 函数需要做适当地修改：

```
01.  void display(link *p){
02.      link* temp=p;//将temp指针重新指向头结点
03.      //只要temp指针指向的结点的next不是Null，就执行输出语句。
04.      while (temp->next) {
05.          temp=temp->next;
06.          printf("%d",temp->elem);
07.      }
08.      printf("\n");
09.  }
```

上一节 下一节