第28讲：单链表专题

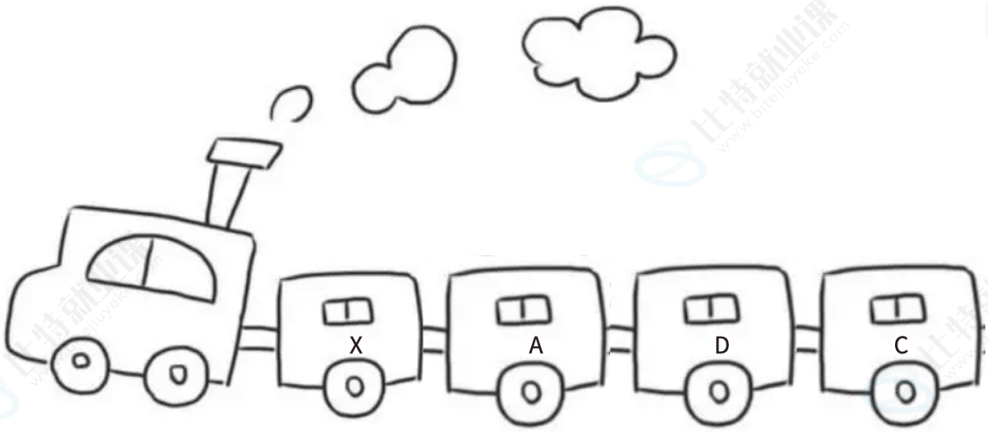
目录

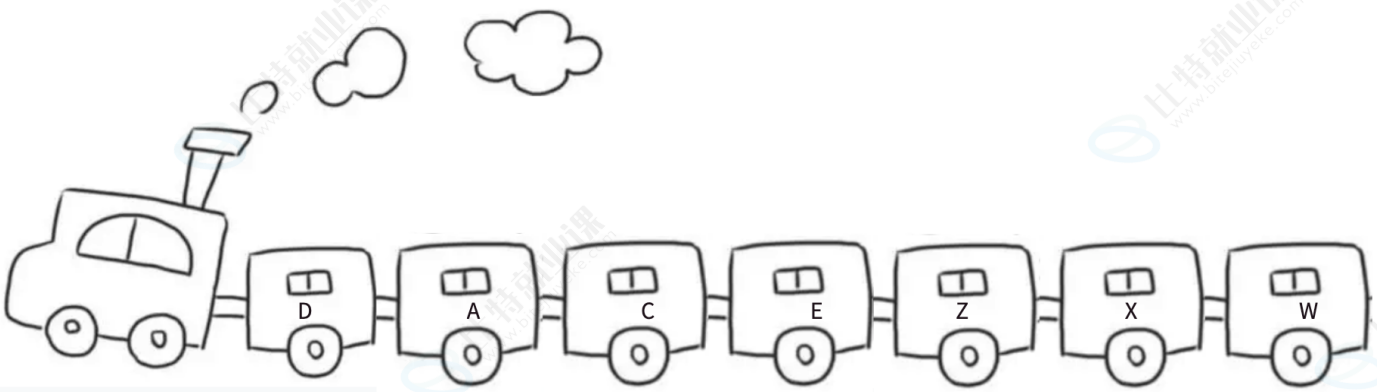
正文开始

单链表专题（1课时）

1. 链表的概念及结构

概念：链表是一种**物理存储结构上非连续**、非顺序的存储结构，数据元素的**逻辑顺序**是通过链表中的**指针链接**次序实现的 。



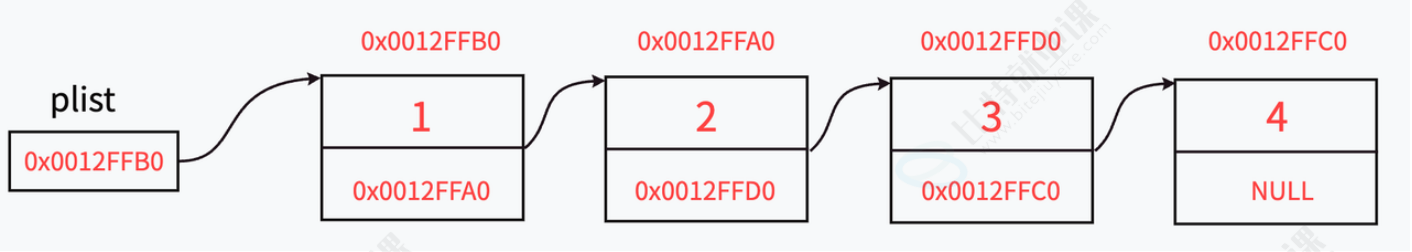


链表的结构跟火车车厢相似，淡季时车次的车厢会相应减少，旺季时车次的车厢会额外增加几节。只需要将火车里的某节车厢去掉/加上，不会影响其他车厢，每节车厢都是独立存在的。

车厢是独立存在的，且每节车厢都有车门。想象一下这样的场景，假设每节车厢的车门都是锁上的状态，需要不同的钥匙才能解锁，每次只能携带一把钥匙的情况下如何从车头走到车尾？

最简单的做法：每节车厢里都放一把下一节车厢的钥匙。

在链表里，每节“车厢”是什么样的呢？



与顺序表不同的是，链表里的每节"车厢"都是独立申请下来的空间，我们称之为“结点/节点”

节点的组成主要有两个部分：当前节点要保存的数据和保存下一个节点的地址（指针变量）。

图中指针变量 plist保存的是第一个节点的地址，我们称plist此时“指向”第一个节点，如果我们希望plist“指向”第二个节点时，只需要修改plist保存的内容为0x0012FFA0。

为什么还需要指针变量来保存下一个节点的位置？

链表中每个节点都是独立申请的（即需要插入数据时才去申请一块节点的空间），我们需要通过指针变量来保存下一个节点位置才能从当前节点找到下一个节点。

结合前面学到的结构体知识，我们可以给出每个节点对应的结构体代码：

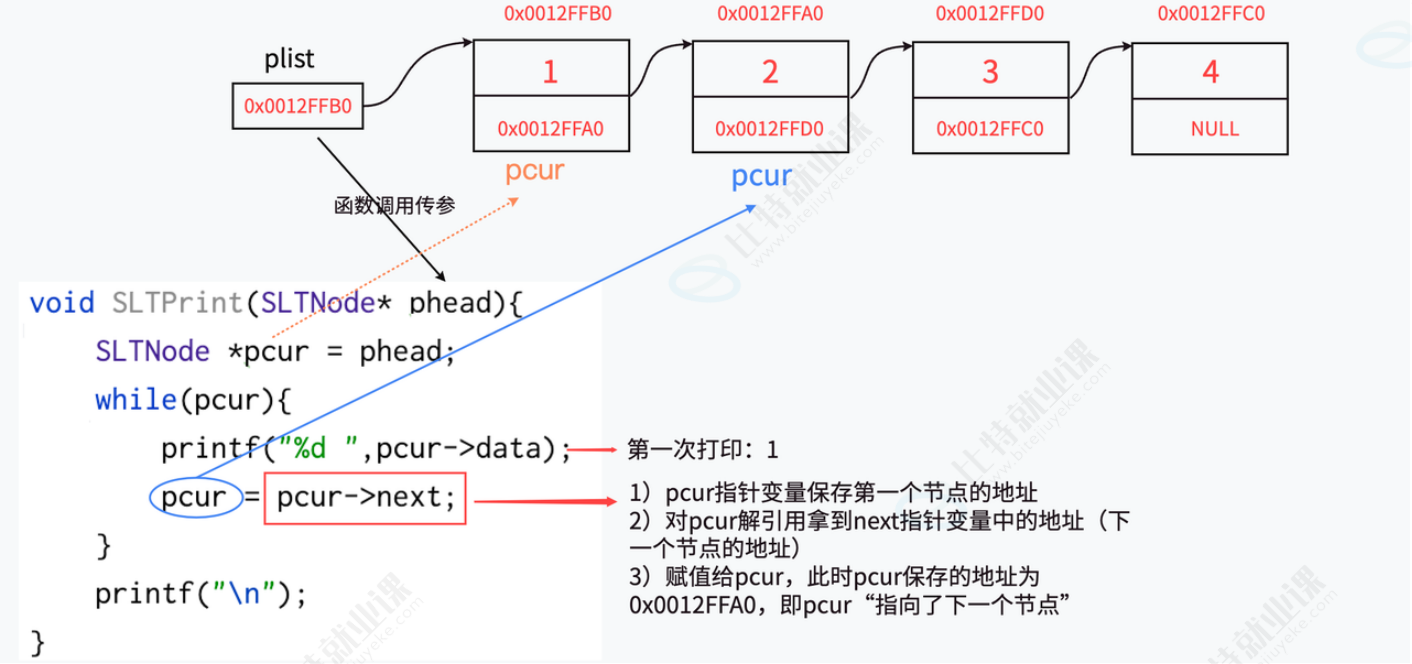
假设当前保存的节点为整型：

```
1   struct SListNode
2   {
3       int data;       //节点数据
4       struct SListNode* next; //指针变量用保存下一个节点的地址
5   };
```

当我们想要保存一个整型数据时，实际是向操作系统申请了一块内存，这个内存不仅要保存整型数据，也需要保存下一个节点的地址（当下一个节点为空时保存的地址为空）。

当我们想要从第一个节点走到最后一个节点时，只需要在前一个节点拿上下一个节点的地址（下一个节点的钥匙）就可以了。

给定的链表结构中，如何实现节点从头到尾的打印？



思考：当我们想保存的数据类型为字符型、浮点型或者其他自定义的类型时，该如何修改？

补充说明：

1、链式结构在逻辑上是连续的，在物理结构上不一定连续

2、节点一般是从堆上申请的

3、从堆上申请来的空间，是按照一定策略分配出来的，每次申请的空间可能连续，可能不连续

2. 单链表的实现

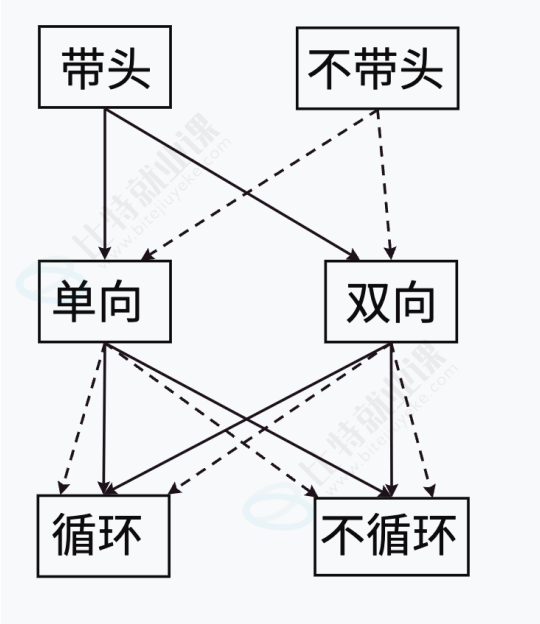
```c
typedef int SLTDataType;
typedef struct SListNode
{
    SLTDataType data;       //节点数据
    struct SListNode* next; //指针保存下一个节点的地址
}SLTNode;


void SLTPrint(SLTNode* phead);

//头部插入删除/尾部插入删除
void SLTPushBack(SLTNode** pphead, SLTDataType x);
void SLTPushFront(SLTNode** pphead, SLTDataType x);
void SLTPopBack(SLTNode** pphead);
void SLTPopFront(SLTNode** pphead);

//查找
SLTNode* SLTFind(SLTNode* phead, SLTDataType x);
//在指定位置之前插入数据
void SLTInsert(SLTNode** pphead, SLTNode* pos, SLTDataType x);
```

```
21  //删除pos节点
22  void SLTErase(SLTNode** pphead, SLTNode* pos);
23  //在指定位置之后插入数据
24  void SLTInsertAfter(SLTNode* pos, SLTDataType x);
25  //删除pos之后的节点
26  void SLTEraseAfter(SLTNode* pos);
27  //销毁链表
28  void SListDesTroy(SLTNode** pphead);
```

基于数据结构-单链表 实现通讯录项目

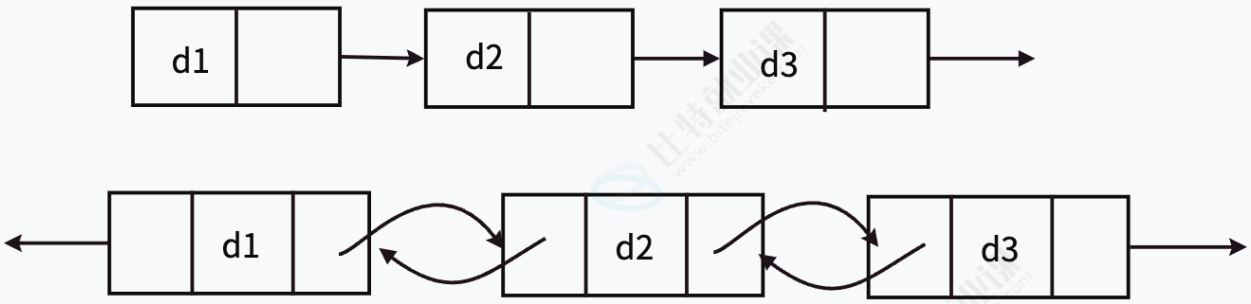
3. 链表的分类

链表的结构非常多样，以下情况组合起来就有8种（2 x 2 x 2）链表结构：

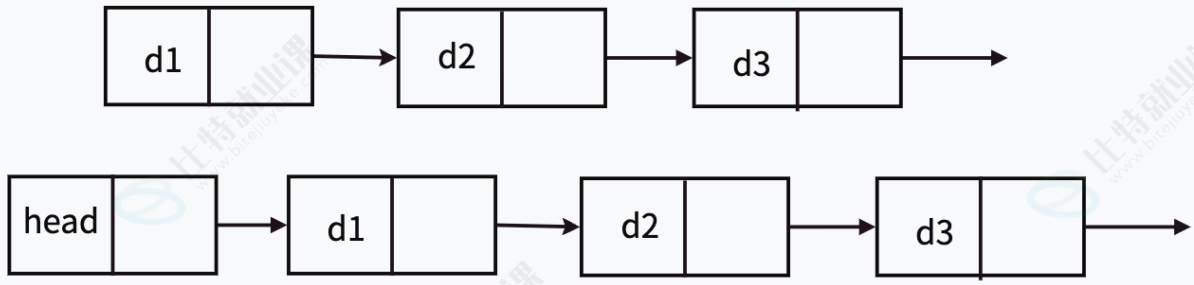


> 链表说明：

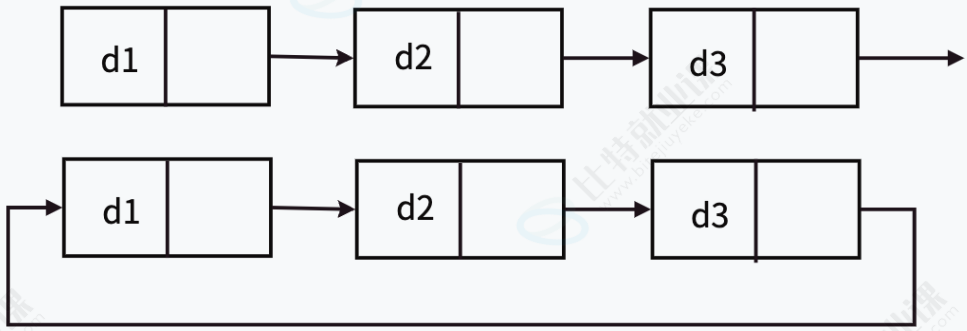
1.单向或者双向



2.带头或者不带头



3.循环或者不循环



虽然有这么多的链表的结构，但是我们实际中最常用还是两种结构：单链表和双向带头循环链表

1. 无头单向非循环链表：**结构简单**，一般不会单独用来存数据。实际中更多是作为**其他数据结构的子结构**，如哈希桶、图的邻接表等等。另外这种结构在**笔试面试**中出现很多。

2. 带头双向循环链表：**结构最复杂**，一般用在单独存储数据。实际中使用的链表数据结构，都是带头双向循环链表。另外这个结构虽然结构复杂，但是使用代码实现以后会发现结构会带来很多优势，实现反而简单了，后面我们代码实现了就知道了。

完