07 | 链表（下）：如何轻松写出正确的链表代码?

2018-10-05 王争

07 | 链表（下）：如何轻松写出正确的链表代码?
朗读人：修阳　12′30″ | 5.73M

上一节我讲了链表相关的基础知识。学完之后，我看到有人留言说，基础知识我都掌握了，但是写链表代码还是很费劲。哈哈，的确是这样的！

想要写好链表代码并不是容易的事儿，尤其是那些复杂的链表操作，比如链表反转、有序链表合并等，写的时候非常容易出错。从我上百场面试的经验来看，能把“链表反转”这几行代码写对的人不足 10%。

为什么链表代码这么难写？究竟怎样才能比较轻松地写出正确的链表代码呢？

只要愿意投入时间，我觉得大多数人都是可以学会的。比如说，如果你真的能花上一个周末或者一整天的时间，就去写链表反转这一个代码，多写几遍，一直练到能毫不费力地写出 Bug free 的代码。这个坎还会很难跨吗？

当然，自己有决心并且付出精力是成功的先决条件，除此之外，我们还需要一些方法和技巧。我根据自己的学习经历和工作经验，总结了**几个写链表代码技巧**。如果你能熟练掌握这几个技巧，

加上你的主动和坚持，轻松拿下链表代码完全没有问题。

技巧一：理解指针或引用的含义

事实上，看懂链表的结构并不是很难，但是一旦把它和指针混在一起，就很容易让人摸不着头脑。所以，要想写对链表代码，首先就要理解好指针。

我们知道，有些语言有“指针”的概念，比如 C 语言；有些语言没有指针，取而代之的是“引用”，比如 Java、Python。不管是“指针”还是“引用”，实际上，它们的意思都是一样的，都是存储所指对象的内存地址。

接下来，我会拿 C 语言中的“指针”来讲解，如果你用的是 Java 或者其他没有指针的语言也没关系，你把它理解成“引用”就可以了。

实际上，对于指针的理解，你只需要记住下面这句话就可以了：

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。

这句话听起来还挺拗口的，你可以先记住。我们回到链表代码的编写过程中，我来慢慢给你解释。

在编写链表代码的时候，我们经常会有这样的代码：p->next=q。这行代码是说，p 结点中的 next 指针存储了 q 结点的内存地址。

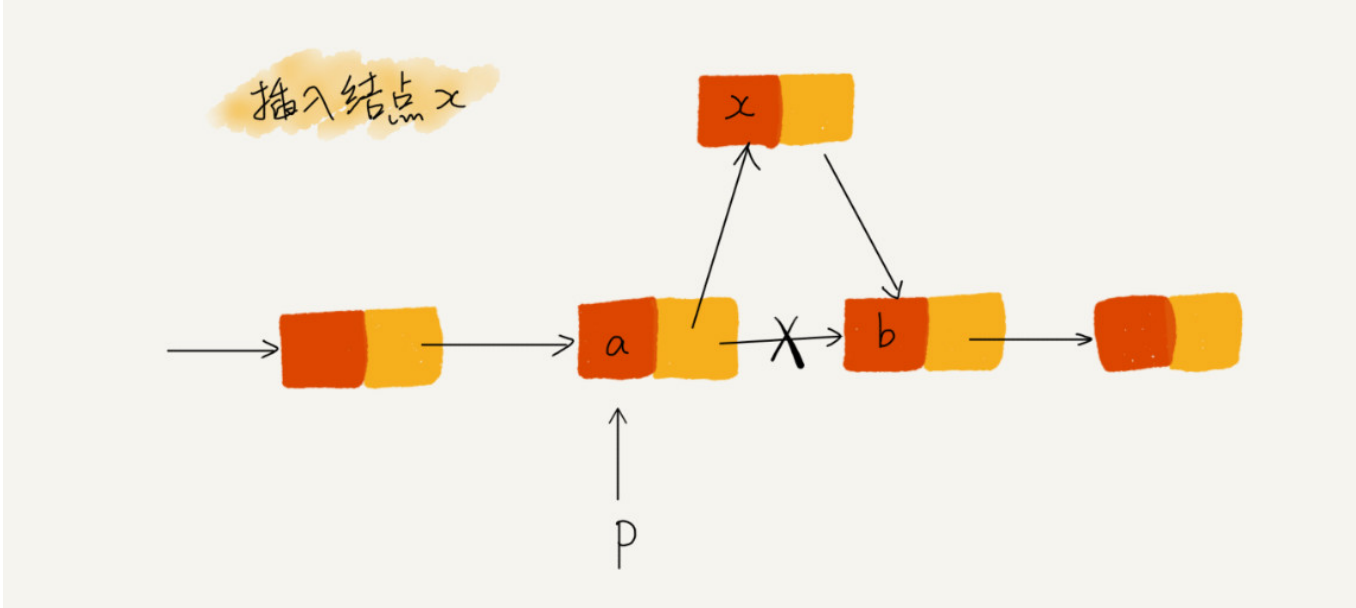
还有一个更复杂的，也是我们写链表代码经常会用到的：p->next=p->next->next。这行代码表示，p 结点的 next 指针存储了 p 结点的下下一个结点的内存地址。

掌握了指针或引用的概念，你应该可以很轻松地看懂链表代码。恭喜你，已经离写出链表代码近了一步！

技巧二：警惕指针丢失和内存泄漏

不知道你有没有这样的感觉，写链表代码的时候，指针指来指去，一会儿就不知道指到哪里了。所以，我们在写的时候，一定注意不要弄丢了指针。

指针往往都是怎么弄丢的呢？我拿单链表的插入操作为例来给你分析一下。



如图所示，我们希望在结点 a 和相邻的结点 b 之间插入结点 x，假设当前指针 p 指向结点 a。如果我们将代码实现变成下面这个样子，就会发生指针丢失和内存泄露。

复制代码

```
p->next = x;  // 将 p 的 next 指针指向 x 结点;
x->next = p->next;  // 将 x 的结点的 next 指针指向 b 结点;
```

初学者经常会在这儿犯错。p->next 指针在完成第一步操作之后，已经不再指向结点 b 了，而是指向结点 x。第 2 行代码相当于将 x 赋值给 x->next，自己指向自己。因此，整个链表也就断成了两半，从结点 b 往后的所有结点都无法访问到了。

对于有些语言来说，比如 C 语言，内存管理是由程序员负责的，如果没有手动释放结点对应的内存空间，就会产生内存泄露。所以，我们**插入结点时，一定要注意操作的顺序**，要先将结点 x 的 next 指针指向结点 b，再把结点 a 的 next 指针指向结点 x，这样才不会丢失指针，导致内存泄漏。所以，对于刚刚的插入代码，我们只需要把第 1 行和第 2 行代码的顺序颠倒一下就可以了。

同理，**删除链表结点时，也一定要记得手动释放内存空间**，否则，也会出现内存泄漏的问题。当然，对于像 Java 这种虚拟机自动管理内存的编程语言来说，就不需要考虑这么多了。

技巧三：利用哨兵简化实现难度

首先，我们先来回顾一下单链表的插入和删除操作。如果我们在结点 p 后面插入一个新的结点，只需要下面两行代码就可以搞定。

复制代码

```
new_node->next = p->next;
p->next = new_node;
```

但是，当我们要向一个空链表中插入第一个结点，刚刚的逻辑就不能用了。我们需要进行下面这样的特殊处理，其中 head 表示链表的头结点。所以，从这段代码，我们可以发现，对于单链表的插入操作，第一个结点和其他结点的插入逻辑是不一样的。

复制代码

```
if (head == null) {
  head = new_node;
}
```

我们再来看单链表结点删除操作。如果要删除结点 p 的后继结点，我们只需要一行代码就可以搞定。

复制代码

```
p->next = p->next->next;
```

但是，如果我们要删除链表中的最后一个结点，前面的删除代码就不 work 了。跟插入类似，我们也需要对于这种情况特殊处理。写成代码是这样子的：

复制代码

```
if (head->next == null) {
   head = null;
}
```

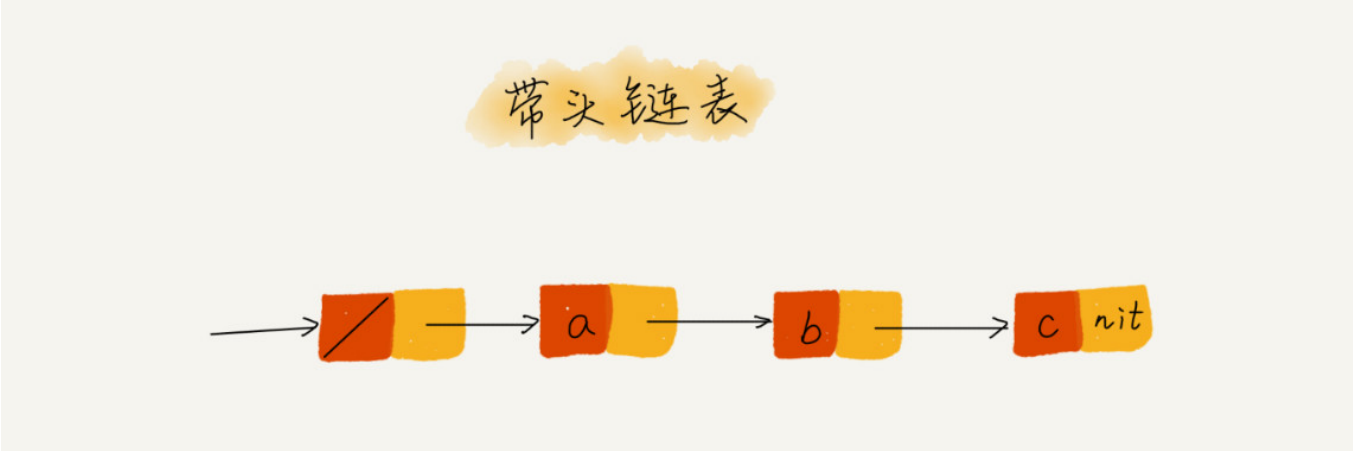
从前面的一步一步分析，我们可以看出，**针对链表的插入、删除操作，需要对插入第一个结点和删除最后一个结点的情况进行特殊处理**。这样代码实现起来就会很繁琐，不简洁，而且也容易因为考虑不全而出错。如何来解决这个问题呢？

技巧三中提到的哨兵就要登场了。哨兵，解决的是国家之间的边界问题。同理，这里说的哨兵也是解决“边界问题”的，不直接参与业务逻辑。

还记得如何表示一个空链表吗？head=null 表示链表中没有结点了。其中 head 表示头结点指针，指向链表中的第一个结点。

如果我们引入哨兵结点，在任何时候，不管链表是不是空，head 指针都会一直指向这个哨兵结点。我们也把这种有哨兵结点的链表叫**带头链表**。相反，没有哨兵结点的链表就叫作**不带头链表**。

我画了一个带头链表，你可以发现，哨兵结点是不存储数据的。因为哨兵结点一直存在，所以插入第一个结点和插入其他结点，删除最后一个结点和删除其他结点，都可以统一为相同的代码实现逻辑了。



实际上，这种利用哨兵简化编程难度的技巧，在很多代码实现中都有用到，比如插入排序、归并排序、动态规划等。这些内容我们后面才会讲，现在为了让你感受更深，我再举一个非常简单的例子。代码我是用 C 语言实现的，不涉及语言方面的高级语法，很容易看懂，你可以类比到你熟悉的语言。

代码一：

复制代码

```
int find(char* a, int n, char key) {
  int i = 0;
  while (i < n) {
    if (a[i] == key) {
      return i;
    }
    ++i;
  }
  return -1;
}
```

代码二：

复制代码

```
inf find(char* a, int n, int key) {
  if (a[n-1] == key) {
    return n-1;
  }
  char tmp = a[n-1];
  a[n-1] = key;
  int i = 0;
  while (a[i] != key) {
```

```
    ++i;
  }
  a[n-1] = tmp;
  if (i == n-1) return -1;
  return i;
}
```

对比两段代码，在字符串 a 很长的时候，比如几万、几十万，你觉得哪段代码运行得更快点呢？答案是代码二，因为两段代码中执行次数最多就是 while 循环那一部分。第二段代码中，我们通过一个哨兵 a[n-1] = key，成功省掉了一个比较语句 i<n，不要小看这一条语句，当累积执行万次、几十万次时，累积的时间就很明显了。

当然，这只是为了举例说明哨兵的作用，你写代码的时候千万不要写第二段那样的代码，因为可读性太差了。大部分情况下，我们并不需要如此追求极致的性能。

技巧四：重点留意边界条件处理

软件开发中，代码在一些边界或者异常情况下，最容易产生 Bug。链表代码也不例外。要实现没有 Bug 的链表代码，一定要在编写的过程中以及编写完成之后，检查边界条件是否考虑全面，以及代码在边界条件下是否能正确运行。

我经常用来检查链表代码是否正确的边界条件有这样几个：

- 如果链表为空时，代码是否能正常工作？
- 如果链表只包含一个结点时，代码是否能正常工作？
- 如果链表只包含两个结点时，代码是否能正常工作？
- 代码逻辑在处理头结点和尾结点的时候，是否能正常工作？

当你写完链表代码之后，除了看下你写的代码在正常的情况下能否工作，还要看下在上面我列举的几个边界条件下，代码仍然能否正确工作。如果这些边界条件下都没有问题，那基本上可以认为没有问题了。

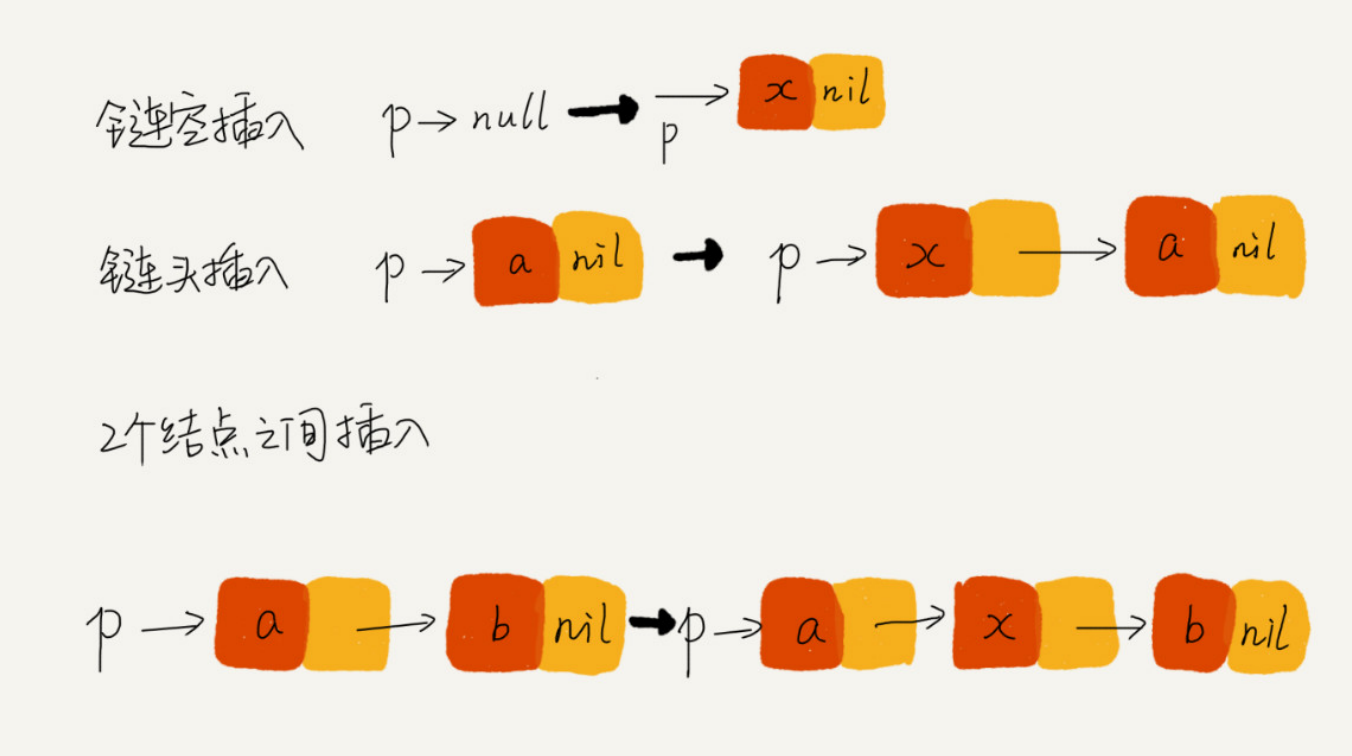
当然，边界条件不止我列举的那些。针对不同的场景，可能还有特定的边界条件，这个需要你自己去思考，不过套路都是一样的。

实际上，不光光是写链表代码，你在写任何代码时，也千万不要只是实现业务正常情况下的功能就好了，一定要多想想，你的代码在运行的时候，可能会遇到哪些边界情况或者异常情况。遇到了应该如何应对，这样写出来的代码才够健壮！

技巧五：举例画图，辅助思考

对于稍微复杂的链表操作，比如前面我们提到的单链表反转，指针一会儿指这，一会儿指那，一会儿就被绕晕了。总感觉脑容量不够，想不清楚。所以这个时候就要使用大招了，**举例法**和**画图法**。

你可以找一个具体的例子，把它画在纸上，释放一些脑容量，留更多的给逻辑思考，这样就会感觉到思路清晰很多。比如往单链表中插入一个数据这样一个操作，我一般都是把各种情况都举一个例子，画出插入前和插入后的链表变化，如图所示：



看图写代码，是不是就简单多啦？而且，当我们写完代码之后，也可以举几个例子，画在纸上，照着代码走一遍，很容易就能发现代码中的 Bug。

技巧六：多写多练，没有捷径

如果你已经理解并掌握了我前面所讲的方法，但是手写链表代码还是会出现各种各样的错误，也不要着急。因为我最开始学的时候，这种状况也持续了一段时间。

现在我写这些代码，简直就和“玩儿”一样，其实也没有什么技巧，就是把常见的链表操作都自己多写几遍，出问题就一点一点调试，熟能生巧！

所以，我精选了 5 个常见的链表操作。你只要把这几个操作都能写熟练，不熟就多写几遍，我保证你之后再也不会害怕写链表代码。

- 单链表反转
- 链表中环的检测

- 两个有序的链表合并
- 删除链表倒数第 n 个结点
- 求链表的中间结点

内容小结

这节我主要和你讲了写出正确链表代码的六个技巧。分别是理解指针或引用的含义、警惕指针丢失和内存泄漏、利用哨兵简化实现难度、重点留意边界条件处理，以及举例画图、辅助思考，还有多写多练。

我觉得，**写链表代码是最考验逻辑思维能力的**。因为，链表代码到处都是指针的操作、边界条件的处理，稍有不慎就容易产生 Bug。链表代码写得好坏，可以看出一个人写代码是否够细心，考虑问题是否全面，思维是否缜密。所以，这也是很多面试官喜欢让人手写链表代码的原因。所以，这一节讲到的东西，你一定要自己写代码实现一下，才有效果。

课后思考

今天我们讲到用哨兵来简化编码实现，你是否还能够想到其他场景，利用哨兵可以大大地简化编码难度？

欢迎留言和我分享，我会第一时间给你反馈。

戳此查看本节内容相关的详细代码

写留言

精选留言

姜威 10

总结：如何优雅的写出链表代码？6大学习技巧

一、理解指针或引用的含义

1.含义：将某个变量（对象）赋值给指针（引用），实际上就是就是将这个变量（对象）的地址赋值给指针（引用）。

2.示例：

p—>next = q; 表示p节点的后继指针存储了q节点的内存地址。

p—>next = p—>next—>next; 表示p节点的后继指针存储了p节点的下下个节点的内存地址。

二、警惕指针丢失和内存泄漏（单链表）

1.插入节点

在节点a和节点b之间插入节点x，b是a的下一节点，，p指针指向节点a，则造成指针丢失和内存泄漏的代码：p—>next = x;x—>next = p—>next; 显然这会导致x节点的后继指针指向自身。

正确的写法是2句代码交换顺序，即：x—>next = p—>next; p—>next = x;

2.删除节点

在节点a和节点b之间删除节点b，b是a的下一节点，p指针指向节点a：p—>next = p—>next—>next;

三、利用“哨兵”简化实现难度

1.什么是“哨兵”?

链表中的“哨兵”节点是解决边界问题的，不参与业务逻辑。如果我们引入“哨兵”节点，则不管链表是否为空，head指针都会指向这个“哨兵”节点。我们把这种有“哨兵”节点的链表称为带头链表，相反，没有“哨兵”节点的链表就称为不带头链表。

2.未引入“哨兵”的情况

如果在p节点后插入一个节点，只需2行代码即可搞定：

new_node—>next = p—>next;

p—>next = new_node;

但，若向空链表中插入一个节点，则代码如下：

if(head == null){

head = new_node;

}

如果要删除节点p的后继节点，只需1行代码即可搞定：

p—>next = p—>next—>next;

但，若是删除链表的最有一个节点（链表中只剩下这个节点），则代码如下：

if(head—>next == null){

head = null;

}

从上面的情况可以看出，针对链表的插入、删除操作，需要对插入第一个节点和删除最后一个

节点的情况进行特殊处理。这样代码就会显得很繁琐，所以引入“哨兵”节点来解决这个问题。

3.引入“哨兵”的情况

“哨兵”节点不存储数据，无论链表是否为空，head指针都会指向它，作为链表的头结点始终存在。这样，插入第一个节点和插入其他节点，删除最后一个节点和删除其他节点都可以统一为相同的代码实现逻辑了。

4.“哨兵”还有哪些应用场景?

这个知识有限，暂时想不出来呀！但总结起来，哨兵最大的作用就是简化边界条件的处理。

四、重点留意边界条件处理

经常用来检查链表是否正确的边界4个边界条件：

1.如果链表为空时，代码是否能正常工作?

2.如果链表只包含一个节点时，代码是否能正常工作?

3.如果链表只包含两个节点时，代码是否能正常工作?

4.代码逻辑在处理头尾节点时是否能正常工作?

五、举例画图，辅助思考

核心思想：释放脑容量，留更多的给逻辑思考，这样就会感觉到思路清晰很多。

六、多写多练，没有捷径

5个常见的链表操作：

1.单链表反转

2.链表中环的检测

3.两个有序链表合并

4.删除链表倒数第n个节点

5.求链表的中间节点

2018-10-05

zyzheng 8

一直对手写链表代码有恐惧心理，这次硬着头皮也要迈过这个坎

2018-10-05

Rain 7

谢谢老师，这节课又学到了，写完留言我要去思考那几个问题了，一个都不会。。

文中提到，

但是，如果我们要删除链表中的最后一个结点，前面的删除代码就不 work 了。跟插入类似，我们也需要对于这种情况特殊处理。写成代码是这样子的：

```
if (head->next == null) {
```

```
head = null;
}
```

感觉此处代码处理的是当链表中只有表头一个节点的删除情况，而不是"要删除链表中的最后一个结点"的情况。是不是head应该改成p?

2018-10-05

Miletos 4

C语言，二级指针可以绕过不带头结点链表删除操作的边界检查。

2018-10-05

来自地狱的勇士 3

问题一：文中提到，指针丢失会导致内存泄露，老师能解释下如何导致的内存泄露吗?
问题二：讲哨兵那块的内容时，说代码二比代码一成功省掉了一次比较i<n，这句不大理解，代码二中，while的条件a[i]!=key也是在比较吧?

2018-10-05

广兴 3

c语言不熟悉 看起来有点吃力

2018-10-05

Smallfly 2

如何写好链表代码?

1. 理解指针或引用的含义
什么是指针？指针是一个变量，该变量中存的是其它变量的地址。将普通变量赋值给指针变量，其实是把它的地址赋值给指针变量。

2. 警惕指针丢失和内存泄漏
在插入和删除结点时，要注意先持有后面的结点再操作，否者一旦后面结点的前继指针被断开，就无法再访问，导致内存泄漏。

3. 利用哨兵简化难度
链表的插入、删除操作，需要对插入第一个结点和删除最后一个节点做特殊处理。利用哨兵对象可以不用边界判断，链表的哨兵对象是只存指针不存数据的头结点。

4. 重点留意边界条件处理
操作链表时要考虑链表为空、一个结点、两个结点、头结点、尾结点的情况。学习数据结构和算法主要是掌握一系列思想，能在其它的编码中也养成考虑边界的习惯。

5. 举例画图，辅助思考

对于比较复杂的操作，可以用纸笔画一画，释放脑容量来做逻辑处理（时间换空间思想），也便于完成后的检查。

6. 多写多练，没有捷径
孰能生巧，不管是什么算法，只有经过反复的练习，才能信手拈来。

哨兵对象思想，在 iOS AutoreleasePool 中有用到，在 AutoreleasePoolPush 时添加一个哨兵对象，Pop 时将到哨兵对象之间的所有 Autorelease 对象发送 release 消息。

2018-10-05

hope 2

看完了，打卡，稍后手写作业，去GitHub上看了下 ，希望老师把c的代码也添加上，谢谢

2018-10-05

作者回复

要不你写下 提个pull request?

2018-10-05

Richard Zhong 2

深入浅出，太赞了

2018-10-05

失火的夏天 2

1.三个节点p.pre，p，p.next，将p的next指针指向p.pre，然后p.pre=p，p=p.next，p.next=p.next.next移动指针，就可以实现单链表反转。
2.最简单就是一个节点在头，一个节点一直遍历，地址相等就是环，不过好像还有一种简单的办法，快慢前进，一次就能搞定。这个老师能不能说下自己的思路，我有点想不明白。
3.建立第三个链表，每次比较a链表当前节点和b链表当前节点的大小。如果a比b小，则c的next指针指向a当前节点，c=c.next，然后a指针后移。接着继续比较a.b当前节点大小，反之则把a换成b就行了。
4.一个p节点，然后找到距离p有n个next节点的点，一起往后遍历，到pn.next为空的时候，p就是我们要求的那个地址。
5.快慢指针，一个每次前进2个节点一个每次前进1节点。前进两个节点到表尾的时候，前进一个的就是中间点。

2018-10-05

过些天再换个名字 现在想不出来... 1

代码二示例返回值int是不是写成inf了哈哈哈

算法设计思路应该是
// 用来找出给定key在数组中的下标，找不到则返回-1

a是被遍历的数组

n是数组长度
key是要寻找的值

1， 判断尾节点是不是要寻找的值，是的话返回n-1，因为数组下标从0开始所以要长度-1才是下标

2， 使用哨兵变量保存尾节点

3， 把key放到尾节点，让key成为数组中最后一个值，这样做是为了下一步的遍历

4， 开始从头开始遍历数组，i为数组下标，如果找到与key相等的元素则退出遍历，否则遍历整个数组

5， 如果i是尾节点下标，说明没有找到key，如果不是则i为寻找的节点下标，返回i

6， 把哨兵变量还原赋值到数组尾节点，也就是还原数组

也就是说平时用的临时变量就是哨兵变量

2018-10-06

起点·终站 1

那两段代码，把小于号换成不等号，有什么区别么？还是在比较N次啊😱

2018-10-06

蓝色的梦 1

老师，删除链表倒数第K个节点时候，判断prev==NULL是什么意思，这个什么时候可能为空？

2018-10-05

作者回复

一共有5个节点 删除倒数第5个。你试下这种情况

2018-10-05

广进 1

作为一个小白，每节课都有看不懂的，这次又来了，那个代码二，从while往下就不懂了，怎么感觉和一的功能不一样了。求指导。

还有您都觉得二可读性差了，加点注释照顾照顾我们这些小白呀。😭

2018-10-05

zeta 0

建议大家在实现之前的思考时间不要太长。一是先用自己能想到的暴力方法实现试试。另外就是在一定时间内(比如半个到一个小时)实在想不到就要在网上搜搜答案。有的算法，比如链表中环的检测，的最优解法还是挺巧妙的，一般来说不是生想就能想到的

2018-10-06

嘿嘿啊 0

提到哨兵，很明显的就是快排啊。

2018-10-06

code047 0

一直觉得链表有两种写法，一种头节点不存数据，只存链表头节点地址，原件这就是哨兵，一种头节点包含数据和下一个节点地址，每次联系都把两种写了，强迫症

2018-10-06

张飞online 0

学校学习时，我自己写了这些东西，那时才感觉到指针的强大，到了工作中，发现这些倒不是难度，难度是多线程操作，线程安全的数据结构，而且是性能可观，如果仅仅加了把大锁，性能就下去了，我去看了lockfree，但是感觉没有掌握，不太敢用。

2018-10-06

wistbean 0

------总结一下-------

写链表代码的技巧

1.理解指针或者引用的含义
指针存储了变量的内存地址，指向了这个变量，通过指针就能找到这个变量。
p->next->q：p节点的next指针存储了q的内存地址；
p->next = p->next->next:p节点的next指针存储了p的下下一个节点的内存地址。

2.警惕指针丢失和内存泄漏
注意不要弄丢指针。
插入节点要注意顺序，删除节点要注意释放。

3.利用哨兵简化实现难度
在带头链表中，哨兵节点不存储数据，可以统一为相同的代码实现逻辑。

4.重点留意边界条件处理
考虑：
a.链表为空时，代码是否能正常工作?
b.链表只含一个节点，代码是否能正常工作?
c.链表只含两个节点，代码是否能正常工作?
d.代码逻辑处理头尾节点时，代码是否能正常工作?

5.举例画图，辅助思考
画出来，会比较清晰。

6.多写多练

多练以下常见的操作
a.单链表反转
b.两个有序的链表合并
c.删除链表倒数第 n 个结点
d.求链表的中间结点

2018-10-05

梦其不可梦 0

你好，老师，我想提个问题，创建链表时，为什么要用
link L=(link)malloc(sizeof(node));
而不能使用
node t;
link L=&L;
呢?

2018-10-05

乘坐Tornado的线程魔法师 0

在网上看到了链表反转的C语言实现。其中有这样先后两个语句：pNode->Next = pPrev；pPrev = pNode; 我可否将这两个语句合并成一个语句：pNode-> Next = pNode。这样做是否对等?

2018-10-05

liangjf 0

很多人问文中的内存泄露。在这里主要是c c++等没有内存回收机制的语言。在链表中的内存泄露实质有“两个“，一是在链表操作时，结点的next指针丢弃，造成对新链表某些结点被“与世隔绝”了，这些结点也就无法free掉，可以想象成僵尸进程。二是链表操作时，next指针指向ok，但忘了free掉对应的结点。
总的来说，链表的内存泄露就是没有free掉对应的结点。

同时对代码二有疑问❓ while(a[i] != key)不是也相当于代码一的循环?

2018-10-05

wean 0

这节课主要讲了如何轻松正确地写出正确地链表代码

技巧一：理解指针或引用的含义
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，指向这个变量，通过指针就能找到这个变量。

技巧二：警惕指针丢失和内存泄漏
写链表时，一定要注意指针指向哪了，对于脑子转不过来的情况，可以在纸上画图辅助思考。对于自己管理内存的语言，如 C 语言，如果没有手动释放节点对应的内存空间，就会产生内存泄漏。不过，对于 Java 这种虚拟机自动管理内存的编程语言来说，就不需要考虑那么多

了。

技巧三：利用哨兵简化实现难度
有时代码写不出来，也是因为代码的小逻辑多而乱，如果能够实现分析代码，简化逻辑，那么写起代码来就会更容易轻松了。
比如，当在单链表插入一个新节点时，需要两个小逻辑：1. *链表是空的的情况* 2. *链表不是空的情况* 写起来的代码就会像这样复杂
而如果我们有一个哨兵节点，那么就只需“无脑”往这个节点后面插入新节点而不用进行一次判空特殊处理了。

拓展：在很多算法中都有用到哨兵做简化，比如插入排序、归并排序、动态规划等。
下面这个例子就是用了哨兵提升性能：
```
inf find(char* a, int n, int key) {
if (a[n-1] == key) {
return n-1;
}
char tmp = a[n-1];
a[n-1] = key;
int i = 0;
while (a[i] != key) {
++i;
}
a[n-1] = tmp;
if (i == n-1) return -1;
return i;
}
```

技巧四：重点留意边界条件处理
软件开发中，我们往往是从“通常情况入手，设计代码”，这种情况下，如果不注意特殊情况（边界情况）时，就容易产生 BUG。一定要在写完代码后，检查边界条件是否考虑齐全。
常用来检查链表代码是否正确的边界条件有这样几个：
- 如果链表为空时，代码能否正常工作?
- 如果链表只包含一个节点时，代码能否正常工作?
- 如果链表只包含两个节点时，代码能否正常工作?
- 代码逻辑在处理头节点和尾节点时，能否正常工作?

技巧五：画图举例，辅助思考
感觉这也是软件设计图的细节版。写出来，整理一下，就明白了。

技巧六：多写多练，没有接近
水滴石穿，时刻铭记。

2018-10-05

刘远通 0

链表的null指的是空地址

所以既可以是一个节点名字 也可以是地址

2018-10-05

Never too late 0

没有反转链表等其他代码吗?

2018-10-05

作者回复

有啊java实现的 github上有

2018-10-05

小老鼠 0

可否举个用Java或python 写链表的例子，在Java或python如何获得变量地址。

2018-10-05

fiseasky 0

学习了好几节数据结构和算法了，我是也CRUD业务代码的，感觉还是用不着啊?

2018-10-05

作者回复

1. 建议再看下“为什么要学习数据结构和算法”那节课，包括里面的留言，有很多留言都写的很好，很多人都对这门课有比较清晰深刻的认识。

2. 你的疑问应该是：局限于你现在的工作，你觉得用不上对吧。这个是很有可能的。如果你做的项目都是很小的项目，也没有什么性能压力，平时自己也不去思考非功能性的需求，只是完成业务代码就ok了，那确实感觉用不到。但这是你个人的原因，并不代表就真用不到呢，兄弟!

3. 专栏里有很多贴近开发的内容，比如链表这一节，我就讲了LRU算法。数组这一节，我讲了容器和数组的选择。复杂度这一节，我讲了如何预判代码的性能。这些都是很贴合开发的。

4. 我尽量将内容贴近实际的开发，但并不代表一定贴近你的CRUD开发。知识如何用到你的项目中，需要你自己根据我的文章举一反三的思考。

2018-10-05

涛 0

问题:

1、内存泄露具体是什么意思?

2、哨兵的作用就是减少删除头节点和尾节点的特殊情况处理嘛?

3、建议每一节讲完知识点，顺便说一下leetcode 涉及此知识点的题，大家好去刷一下

2018-10-05

作者回复

1. 自己百度一下，如果还不懂再来问吧

2. 在本例中是的，当然文章还写了不仅限于此。

2018-10-05

陈鹏 0

1. 单链表反转

带头链表实现：注意空链表（无数据结点）的情况，其他用3个指针p, q, r循环遍历操作即可： r = q.next; q.next = p; p = q; q = r. 注意将第一个结点的next置为空指针。

2. 链表中环的检测

快慢指针：p, p+2

3. 两个有序的链表合并

两个指针循环遍历比较即可

4. 删除链表倒数第 n 个结点

快慢指针: p–1, p, p+n

5. 求链表的中间结点

遍历两遍链表可得。

2018-10-05

阳仔 0

学习反馈：

本节内容主要是编码实践，光看内容学习效果不是特别好，因此要自己动手写代码，有几个技巧可以学习：

1、理解指针或引用的含义：指针存储的是变量的地址；

2、操作指针或引用时要注意指针丢失和内存泄露的问题；

3、利用哨兵简化实现难度；

4、注意边界处理；

5、通过举例画图来学习；

6、多写多练、多写多练、多写多练；

2018-10-05

！ null 0

好像没觉得链表复杂过，不过一直都写的是单链表和循环链表，没写过双向链表。

2018-10-05

往事随风，顺其自然 0

单链表如果有数据，指针是指向第一个节点，没有指向不是第一个节点？还是说指向头节点？

2018-10-05

HhZzz 0

老师，github上C的07链表代码是空的，请问是需要我们自己查找资料吗？

2018-10-05

作者回复

是的 自己写吧 我实现了java的 你可以参考下

2018-10-05

卡罗 0

不会早上4点钟起来备课吧

2018-10-05

shallwetalk 0

老师你好，我想问个比较幼稚的问题，就是list.get(i)这个的代码或是公式是什么样的，如果是通过下标查找的话时间复杂度不也是O(1)吗，但是不连续的公式是与数组的完全不一样的啊。求解

2018-10-05

作者回复

同学你好，这个list.get(i)中的list是啥?

2018-10-05