

# 目录

## 前言

前言	1.1
----	-----

## Golang版剑指Offer

剑指Offer	2.1
链表篇	3.1
剑指Offer06.从尾到头打印链表	3.1.1
剑指Offer24.反转链表	3.1.2
剑指Offer22.链表中倒数第k个节点	3.1.3
剑指Offer25.合并两个排序的链表	3.1.4
剑指Offer35.复杂链表的复制	3.1.5
栈与队列篇	4.1
剑指Offer09.用两个栈实现队列	4.1.1
剑指Offer30.包含min函数的栈	4.1.2

## Golang版LeetCode

Leetcode	5.1
数组篇	6.1
LeetCode136.只出现一次的数字	6.1.1
LeetCode169.多数元素	6.1.2
LeetCode15.三数之和	6.1.3

## 第二部分

开始	7.1
关于我	7.1.1
帅哥	7.1.1.1
照片测试	7.1.2

## 第三部分

例子	8.1
代码高亮	8.1.1
提示和插图风格	8.1.2
结构列表	8.1.3

## 第四部分

Introduction	9.1
基本安装	9.2
Node.js 安装	9.2.1
Gitbook 安装	9.2.2
Gitbook 命令行速览	9.2.3
图书项目结构	9.3
README.md 与 SUMMARY 编写	9.3.1
目录初始化	9.3.2
图书输出	9.4
输出为静态网站	9.4.1
输出PDF	9.4.2
发布	9.5
发布到Github Pages	9.5.1
结束	9.6

## 附录部分

附录	10.1
Reference	10.1.1

# Go语言——面向Offer学习编程

- 最新版本: v1.2
- 更新时间: 20211023

## 简介

从Offer角度零基础入门Golang，介绍Golang的基础语法，深入分析Golang结构、源码，介绍Golang著名框架，到实战项目。

## 源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

### Gitbook源码

- [crifan/gitbook\\_demo](#): Gitbook演示

如何使用此Gitbook源码去生成发布为电子书

详见: [crifan/gitbook\\_template: demo how to use crifan gitbook template and demo](#)

### 在线浏览

- Gitbook演示 [book.crifan.com](#)
- Gitbook演示 [crifan.github.io](#)

### 离线下载阅读

- Gitbook演示 PDF
- Gitbook演示 ePub
- Gitbook演示 Mobi

## 版权说明

此电子书教程的全部内容，如无特别说明，均为本人原创和整理。其中部分内容参考自网络，均已备注了出处。如有发现侵犯您的版权，请通过邮箱联系我 [sailaoda1@gmail.com](mailto:sailaoda1@gmail.com)，我会尽快删除。谢谢合作。

## 鸣谢

感谢包容理解和悉心照料，才使得我有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新： 2021-10-23 14:40:32

## Golang版剑指Offer

### 链表篇

剑指Offer06.[从尾到头打印链表](#)

剑指Offer24.[反转链表](#)

剑指Offer35.[复杂链表的复制](#)

剑指Offer22.[链表中倒数第k个节点](#)

剑指Offer25.[合并两个排序的链表](#)

### 栈与队列篇

剑指Offer09.[用两个栈实现队列](#)

剑指Offer30.[包含min函数的栈](#)

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 02:02:15

## 链表篇

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:35:00

## 剑指Offer06.从尾到头打印链表

### 题目描述：

输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。

### 示例：

输入：head = [1,3,2]  
输出：[2,3,1]

### 限制：

0 <= 链表长度 <= 10000

### 算法代码：

```
/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

//直接递归反转
func reversePrint(head *ListNode) []int {
    if head == nil {return nil}
    return append(reversePrint(head.Next) , head.Val)
}
```

```
func reversePrint(head *ListNode) []int {
    // 1.遍历链表时直接Val添加到数组头部:时间复杂度O(N) | 空间复杂度O(N)
    ans := []int{}
    for head != nil {
        ans = append([]int{head.Val}, ans...)
        head = head.Next
    }
    return ans
}
```

## 算法思路:

1. 递归法
2. 辅助栈法
3. 直接顺序获取值放到数组，再反转结果

## 算法改进：（数组索引）

```
/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func reversePrint(head *ListNode) []int {
    if head == nil {
        return nil
    }

    res := []int{}
    for head != nil {
        res = append(res, head.Val)
        head = head.Next
    }

    for i, j := 0, len(res)-1; i < j; {
        res[i], res[j] = res[j], res[i]
        i++
        j--
    }

    return res
}
```



执行结果: **通过** [显示详情](#) [添加备注](#)

执行用时: **0 ms** , 在所有 Go 提交中击败了 **100.00%** 的用户

内存消耗: **3.1 MB** , 在所有提交中击败了 **90.17%** 的用户

通过测试用例: **24 / 24**

炫耀一下:



*Image 3.1.1.1 - 算法优化*

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:48:10

## 剑指Offer24.反转链表

### 题目描述：

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

### 示例：

输入：1->2->3->4->5->NULL  
输出：5->4->3->2->1->NULL

### 限制：

0 <= 节点个数 <= 5000

### 算法代码：

```
// 递归
func reverseList(head *ListNode) *ListNode {
    if head == nil || head.Next == nil {
        return head
    }
    var p = reverseList(head.Next)
    head.Next.Next = head
    head.Next = nil
    return p
}
```

### 算法思路：

常规的递归思路。

### 算法优化：

链表节点中有两个元素：

- 值
- 指针

```
type ListNode struct {
    Val int
    Next *ListNode
}
```

Next指向下一个节点

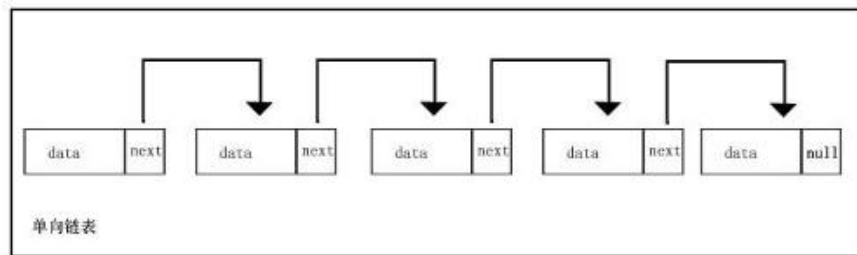


Image 3.1.2.1 - img

那么这道题其实就是把指针指向前一个节点

位置调换次数	pre	cur	whole
0	nil	1->2->3->4->5	1->2->3->4->5
1	1->nil	2->3->4->5	2->3->4->5->1->nil
2	2->1->nil	3->4->5	3->4->5->2->1->nil
3	3->2->1->nil	4->5	4->5->3->2->1->nil
4	4->3->2->1->nil	5	5->4->3->2->1->nil

可以看出来

- pre是cur的最前面那位 (pre = cur)
- cur就是当前位的后面链表元素 (cur = cur.Next)
- cur.Next肯定是接pre (cur.Next = pre)

优化代码:

```
//反转链表的实现
func reversrList(head *ListNode) *ListNode {
    cur := head
    var pre *ListNode = nil
    for cur != nil {
        pre, cur, cur.Next = cur, cur.Next, pre //这句话最重要
    }
    return pre
}
```

执行结果: **通过** [显示详情](#) [添加备注](#)

执行用时: **0 ms** , 在所有 Go 提交中击败了 **100.00%** 的用户

内存消耗: **2.5 MB** , 在所有 Go 提交中击败了 **99.94%** 的用户

通过测试用例: **27 / 27**

炫耀一下:



Image 3.1.2.2 - 算法优化

hackerwu.cn, 使用 [知识共享](#) 署名-非商业性使用-禁止演绎 4.0 国际 许可协议发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:50:38

## 剑指Offer22.链表中倒数第k个节点

### 题目描述：

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 6 个节点，从头节点开始，它们的值依次是 1、2、3、4、5、6。这个链表的倒数第 3 个节点是值为 4 的节点。

### 示例：

给定一个链表：1->2->3->4->5，和 k = 2。

返回链表 4->5。

### 算法思路：

#### 1. 快慢指针法：

快指针先走k，然后快慢指针一起走，快指针走到终点时，慢指针即为所求。

```

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func getKthFromEnd(head *ListNode, k int) *ListNode {
    var slow, fast *ListNode = head, head
    for i := 0; i < k; {
        fast = fast.Next
        i++
    }
    if fast == nil {
        return head
    }
    for fast != nil {
        slow = slow.Next
        fast = fast.Next
    }
    return slow
}

```

执行结果: 通过 [显示详情 >](#)

[添加评论](#)

执行用时: **0 ms** , 在所有 Go 提交中击败了 **100.00%** 的用户

内存消耗: **2.2 MB** , 在所有 Go 提交中击败了 **15.19%** 的用户

通过测试用例: **208 / 208**

炫耀一下:



Image 3.1.3.1 - image-20211018161140668

## 1. 数组索引法

```

func getKthFromEnd(head *ListNode, k int) *ListNode {
    var res []*ListNode
    for head != nil {
        res = append(res, head)
        head = head.Next
    }
    l := len(res)
    if l >= k {
        return res[l - k]
    }

    return nil
}

```

执行结果: 通过 [显示详情](#)

[添加备注](#)

执行用时: **0 ms** , 在所有 Go 提交中击败了 **100.00%** 的用户

内存消耗: **2.3 MB** , 在所有 Go 提交中击败了 **6.13%** 的用户

通过测试用例: **208 / 208**

炫耀一下:



Image 3.1.3.2 - image-20211018190424786

以空间换时间。空间都不大行。

## 1. 遍历递归

```

func getKthFromEnd(head *ListNode, k int) *ListNode {
    if head == nil {
        return nil
    }
    node, _ := getKthFromEndre(head, k)
    return node
}

func getKthFromEndre(head *ListNode, k int) (*ListNode, int) {
    if head == nil {
        return nil, 0 //遍历到最后返回0，开始往上+1判断是否等于k，等
    }
    node, res := getKthFromEndre(head.Next, k)
    if res == k {
        return node, res
    }
    res++
    return head, res
}

```

执行结果: 通过 [显示详情 >](#) ▶ 添加备注

执行用时: **0 ms** , 在所有 Go 提交中击败了 **100.00%** 的用户

内存消耗: **2.2 MB** , 在所有 Go 提交中击败了 **99.96%** 的用户

通过测试用例: **208 / 208**

炫耀一下:







Image 3.1.3.3 - image-20211018200136155

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:51:56



## 剑指Offer25.合并两个排序的链表

### 题目描述：

输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。

### 示例1：

```
输入：1->2->4, 1->3->4  
输出：1->1->2->3->4->4
```

### 限制：

```
0 <= 链表长度 <= 1000
```

### 解题思路：

中心思想：因为有序，则利用双指针分别指向两条链表的表头，然后通过比较大小改变这些节点的指向即可。

1. 利用一个头节点 `head` 简化合并过程。

### 算法代码：

```

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
    head := &ListNode {
        Val : 0 ,
        Next : nil ,
    }
    p := head
    for l1 != nil && l2 != nil {
        if l1.Val <= l2.Val {
            p.Next = l1
            l1 = l1.Next
        } else {
            p.Next = l2
            l2 = l2.Next
        }
        p = p.Next
    }

    if l1 != nil {
        p.Next = l1
    }
    if l2 != nil {
        p.Next = l2
    }

    return head.Next
}

```

执行结果: 通过 [显示详情](#)

[添加备注](#)

执行用时: **4 ms** , 在所有 Go 提交中击败了 **95.70%** 的用户

内存消耗: **4.1 MB** , 在所有 Go 提交中击败了 **100.00%** 的用户

通过测试用例: **218 / 218**

炫耀一下:



Image 3.1.4.1 - image-20211018223252610

hackerwu.cn, 使用 [知识共享](#) 署名-非商业性使用-禁止演绎 4.0 国际 许可协议发布 all right reserved, powered by Gitbook 最后更新: 2021-10-24 01:52:34

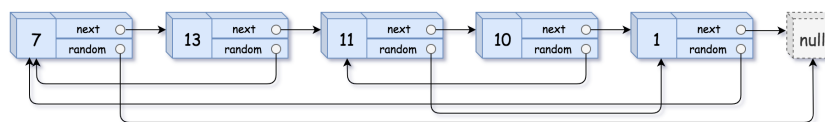
## 剑指Offer35.复杂链表的复制

### 题目描述：

请实现 `copyRandomList` 函数，复制一个复杂链表。在复杂链表中，每个节点除了有一个 `next` 指针指向下一个节点，还有一个 `random` 指针指向链表中的任意节点或者 `null`。

### 示例：

示例 1：



输入: `head = [[7,null],[13,0],[11,4],[10,2],[1,0]]`  
输出: `[[7,null],[13,0],[11,4],[10,2],[1,0]]`

示例 2：

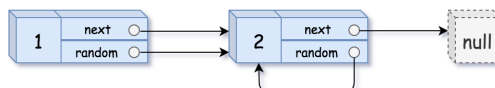


Image 3.1.5.1 - img

输入: `head = [[1,1],[2,1]]`  
输出: `[[1,1],[2,1]]`

示例 3：

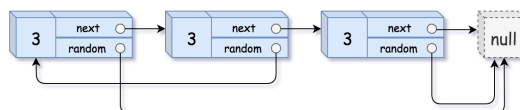


Image 3.1.5.2 - img

```
输入: head = [[3,null],[3,0],[3,null]]
输出: [[3,null],[3,0],[3,null]]
```

示例 4:

```
输入: head = []
输出: []
解释: 给定的链表为空（空指针），因此返回 null。
```

提示:

- $-10000 \leq \text{Node.val} \leq 10000$
- `Node.random` 为空（null）或指向链表中的节点。
- 节点数目不超过 1000 。

算法代码:

```
/**
 * Definition for a Node.
 * type Node struct {
 *     Val int
 *     Next *Node
 *     Random *Node
 * }
 */

func copyRandomList(head *Node) *Node {

}
```

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:53:11

## 栈与队列篇

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:44:50

## 剑指Offer09.用两个栈实现队列

### 题目描述：

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 `appendTail` 和 `deleteHead`，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，`deleteHead` 操作返回 -1 )

### 示例：

示例 1：

```
输入：
["CQueue", "appendTail", "deleteHead", "deleteHead"]
[[], [3], [], []]
输出：[null, null, 3, -1]
```

示例 2：

```
输入：
["CQueue", "deleteHead", "appendTail", "appendTail", "deleteHead", "deleteHead"]
[[], [], [5], [2], [], []]
输出：[null, -1, null, null, 5, 2]
```

### 提示：

- $1 \leq \text{values} \leq 10000$
- 最多会对 `appendTail`、`deleteHead` 进行 10000 次调用

### 算法代码：

```

type CQueue struct {
    // 栈 A, 用于添加元素
    stackA []int
    // 栈 B, 用于取出元素
    stackB []int
}

// CQueue 的构造函数
func Constructor() CQueue {
    // 返回一个新的 CQueue
    return CQueue{}
}

// 往队列插入整数
func (this *CQueue) AppendTail(value int) {
    // 在 stackA 的末尾追加 value
    this.stackA = append(this.stackA, value)
}

// 从队列取出整数并删除
func (this *CQueue) DeleteHead() int {
    // 如果 stackB 没有元素则从 stackA 中取出所有
    if len(this.stackB) == 0 {
        // 如果 stackA 里也没有元素, 则队列为空返回 -1
        if len(this.stackA) == 0 {
            return -1
        }
        // 将 stackA 的所有元素转移到 stackB
        for len(this.stackA) > 0 {
            // 获取 stackA 最末尾元素的下标
            index := len(this.stackA) - 1
            // 获取 stackA 最末尾元素的值 value
            value := this.stackA[index]
            // 向 stackB 的末尾追加 value
            this.stackB = append(this.stackB, value)
            // 从 stackA 中裁剪出末尾元素
            this.stackA = this.stackA[:index]
        }
        // 这时候表示 stackB 内已有元素
        // 获取 stackB 最末尾元素的下标
        index := len(this.stackB) - 1
        // 获取 stackB 最末尾元素的值 value
        value := this.stackB[index]
        // 从 stackB 中裁剪出末尾元素
        this.stackB = this.stackB[:index]
        // 返回 value
        return value
    }
}

```

```
/**
 * Your CQueue object will be instantiated and called as such:
 * obj := Constructor();
 * obj.AppendTail(value);
 * param_2 := obj.DeleteHead();
 */
```

## 算法思路:

用两个栈来模拟队列先进先出的特性，栈A用来添加元素，栈B用来取出元素，取出元素通过将A栈中元素取出倒序添加进入B栈中实现，去掉栈B的栈首元素即为去掉栈A的栈底元素。

以此来模拟队列中的先进先出特性。

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:56:57



## 剑指Offer30.包含min函数的栈

### 题目描述：

定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是  $O(1)$ 。

### 示例：

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.min();    --> 返回 -3.
minStack.pop();
minStack.top();     --> 返回 0.
minStack.min();     --> 返回 -2.
```

### 提示：

各函数的调用总次数不超过 20000 次

### 算法代码：

```

type MinStack struct {
    stack []int
    minstack []int
}

/** initialize your data structure here. */
func Constructor() MinStack {
    return MinStack{
        stack: []int{},
        minstack: []int{math.MaxInt64},
    }
}

func (this *MinStack) Push(x int) {
    this.stack = append(this.stack , x)
    this.minstack = append(this.minstack , min(this.minstack[len(this.minstack)-1], x))
}

func (this *MinStack) Pop() {
    this.minstack = this.minstack[:len(this.minstack) - 1]
    this.stack = this.stack[:len(this.stack) - 1]
}

func (this *MinStack) Top() int {
    return this.stack[len(this.stack) - 1]
}

func (this *MinStack) Min() int {
    return this.minstack[len(this.minstack) - 1]
}

func min(x int , y int ) int {
    if x >= y {
        return y
    }else {
        return x
    }
}

/**
 * Your MinStack object will be instantiated and called as such:
 * obj := Constructor();
 * obj.Push(x);

```

```
* obj.Pop();
* param_3 := obj.Top();
* param_4 := obj.Min();
*/
```

## 算法思路:

利用一个辅助栈来存放较小值，在辅助栈的栈顶总是为所有元素的最小值。

## 注意:

不能直接返回 `return MinStack{}`

```
func Constructor() MinStack {
    return MinStack{
        stack: []int{},
        minstack: []int{math.MaxInt64},
    }
}
```

当测试用例如下时:

```
["MinStack","push","push","push","top","pop","min","pop","min",""]
[[[],[2147483646],[2147483646],[2147483647],[],[],[],[],[],[21
```

直接返回 `return MinStack{}` 会发生错误。

hackerwu.cn, 使用 [知识共享](#) 署名-非商业性使用-禁止演绎 4.0 国际 许可协议发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:57:37

# Golang版Leetcode

## 数组篇

**LeetCode136.**[只出现一次的数字](#)

**LeetCode169.**[多数元素](#)

**LeetCode15.**[三数之和](#)

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 02:04:28

## 数组篇

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:44:50

## LeetCode136.只出现一次的数字

### 题目描述:

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

### 说明:

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

### 示例:

示例 1:

输入: [2,2,1]  
输出: 1

示例 2:

输入: [4,1,2,1,2]  
输出: 4

### 算法代码:

```
func singleNumber(nums []int) int {  
    single := 0  
    for _, num := range nums {  
        single ^= num  
    }  
    return single  
}
```

### 算法思路:

只有一个数字只出现一次，其他数字都出现两次，相同数字位运算结果为0，单独数字和0做位运算结果为他本身。

即只需要对整个数组做位运算，可以找出唯一单独的数字。

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:54:21

## LeetCode169.多数元素

### 题目描述:

给定一个大小为  $n$  的数组，找到其中的多数元素。多数元素是指在数组中出现次数 大于  $\lfloor n/2 \rfloor$  的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

### 示例:

示例 1:

输入: [3,2,3]  
输出: 3

示例 2:

输入: [2,2,1,1,1,2,2]  
输出: 2

### 算法代码:

```
func majorityElement(nums []int) int {  
    major := 0  
    count := 0  
    for _, num := range nums {  
        if count == 0 {  
            major = num  
        }  
        if major == num {  
            count ++  
        } else {  
            count --  
        }  
    }  
    return major  
}
```

### 算法思路:

摩尔投票法: 利用相互抵消的概念



排序法： 排序之后的中间值一定为众数

```
func majorityElement(nums []int) int {  
    sort.Ints(nums)  
    index := len(nums)  
    return nums[len/2]  
}
```

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:54:50

## LeetCode15.三数之和

### 题目描述:

给你一个包含  $n$  个整数的数组 `nums`，判断 `nums` 中是否存在三个元素  $a, b, c$ ，使得  $a + b + c = 0$ ？请你找出所有和为 0 且不重复的三元组。

### 注意:

答案中不可以包含重复的三元组。

### 示例:

示例 1:

```
输入: nums = [-1,0,1,2,-1,-4]
输出: [[-1,-1,2],[-1,0,1]]
```

示例 2:

```
输入: nums = []
输出: []
```

示例 3:

```
输入: nums = [0]
输出: []
```

### 提示:

```
0 <= nums.length <= 3000
-105 <= nums[i] <= 105
```

### 算法代码:

```

func threeSum(nums []int) [][]int {
    n := len(nums)
    sort.Ints(nums)
    //ans := make([][]int, 0)
    ans := [][]int{}

    // 枚举 a
    for first := 0; first < n; first++ {
        // 需要和上一次枚举的数不相同
        if first > 0 && nums[first] == nums[first - 1] {
            continue
        }
        // c 对应的指针初始指向数组的最右端
        third := n - 1
        target := -1 * nums[first]
        // 枚举 b
        for second := first + 1; second < n; second++ {
            // 需要和上一次枚举的数不相同
            if second > first + 1 && nums[second] == nums[second - 1] {
                continue
            }
            // 需要保证 b 的指针在 c 的指针的左侧
            for second < third && nums[second] + nums[third] > target {
                third--
            }
            // 如果指针重合，随着 b 后续的增加
            // 就不会有满足 a+b+c=0 并且 b<c 的 c 了，可以退出循环
            if second == third {
                break
            }
            if nums[second] + nums[third] == target {
                ans = append(ans, []int{nums[first], nums[second], nums[third]})
            }
        }
    }
    return ans
}

```

## 算法思路：

排序加双指针的思想

## 算法改进：

```

func threeSum(nums []int) [][]int{
    n := len(nums)
    sort.Ints(nums)
    ans := [][]int{}
    for first := 0; first < n - 2; first ++ {
        n1 := nums[first]
        if n1 > 0 {
            break
        }
        if first > 0 && nums[first] == nums[first - 1] {
            continue
        }
        second, third := first + 1, n - 1
        for second < third {
            n2, n3 := nums[second], nums[third]
            if n1 + n2 + n3 == 0 {
                ans = append(ans, []int {n1, n2, n3})
                for second < third && nums[second] == n2 {
                    second ++
                }
                for second < third && nums[third] == n3 {
                    third --
                }
            } else if n1 + n2 + n3 < 0 {
                second ++
            } else {
                third --
            }
        }
    }
    return ans
}

```

hackerwu.cn, 使用 知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议发布 all right reserved, powered by Gitbook最后更新: 2021-10-24 01:55:32

# 开始

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08

## 关于我

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08

# 帅哥

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08

## 照片测试



*Image 7.1.2.1 - 高一时候的瓢，枪个憋气*

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:45:33



## 演示各种功能和效果

下面各个章节用来演示 `gitbook` 中的各种功能和效果。

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新： 2021-10-23 15:31:21

# 代码高亮显示

此处 `gitbook` 中代码高亮的插件用的是 `prism`

官网是: <http://prismjs.com>

下面列出常见的几种代码的代码高亮效果。

## markdown

### ⚡ HTTP简介

``HTTP` = `Hyper Text Transfer Protocol` = 超文本传输协议``

HTTP的总体思路是:

- \* 客户端: 向服务器发送一个`请求` = ``Request``, 请求头包含请求的方法、URI
- \* `服务器`: 以一个状态行作出`响应` = ``Response``, 相应的内容包括消息协议

简而言之就是一个你问我答的协议:

- \* 客户端Client 问 = ``请求` = `Request``
- \* 服务器Server 答 = ``响应` = `Response``

关于HTTP的相关的详细知识, 或许很多人不是很熟悉。

但是:

- 作为普通电脑用户的你, 经常用`浏览器` ( ``IE` / `Chrome` / `Firefox`` 等 )
  - 作为程序开发, 用网络库去调用服务器后台提供的接口时
- 其实内部都用到了HTTP的技术。

## python

```

def saveJsonToFile(jsonDict, fullFilename, indent=2, fileEncoding="utf-8"):
    """
    save dict json into file
    for non-ascii string, output encoded string, without \uxxxx
    """
    with codecs.open(fullFilename, 'w', encoding="utf-8") as outputFp:
        json.dump(jsonDict, outputFp, indent=indent, ensure_ascii=False)

#####
# Main Part
#####

templateJson = {}
currentJson = {}

# run python in : /Users/crifan/GitBook/Library/Import/youdao_note
currPath = os.getcwd()
curBasename = os.path.basename(currPath)
CurrentGitbookName = curBasename
print("CurrentGitbookName=%s" % CurrentGitbookName) #youdao_note

bookJsonTemplateFullPath = os.path.join(BookRootPath, BookJsonTemplateFullPath)
# print("bookJsonTemplateFullPath=%s" % bookJsonTemplateFullPath)
# /Users/crifan/GitBook/Library/Import/book_common.json
with open(bookJsonTemplateFullPath) as templateJsonFp:
    templateJson = json.load(templateJsonFp, encoding="utf-8")

```

javascript

```
// extract single sub string from full string
// eg: extract '012345678912345' from 'www.ucows.cn/qr?id=012345'
export function extractSingleStr(curStr, pattern, flags='i') {
  let extractedStr = null;

  let rePattern = new RegExp(pattern, flags);
  console.log(rePattern);

  let matches = rePattern.exec(curStr);
  console.log(matches);

  if (matches) {
    extractedStr = matches[0];
    console.log(extractedStr);
  }

  if (extractedStr === null) {
    extractedStr = "";
  }

  console.log(`curStr=${curStr}, pattern=${pattern}, flags=${flags}`);

  return extractedStr;
}
```

## html

```
<!DOCTYPE HTML>
<html lang="zh-hans" >
  <head>
    <meta charset="UTF-8">
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <title>HTTP相关 · HTTP知识总结</title>
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="description" content="">
    <meta name="generator" content="GitBook 3.2.3">
    <meta name="author" content="Crifan Li <admin@crifan.com>">

    <link rel="stylesheet" href="../gitbook/style.css">
    <link rel="stylesheet" href="../gitbook/gitbook-plugin-sphinxsearch/sphinxsearch.css">
  </head>
</html>
```

## CSS

```
a#weixin {  
  position: relative;  
}  
  
#weixin img {  
  visibility: hidden;  
  opacity: 0;  
  transform: translate(0, 10px);  
  transition: all 0.3s ease-in-out;  
  position: absolute;  
  right: -30px;  
  bottom: 40px;  
  width: 150px;  
  height: 150px;  
}  
  
#weixin:hover img {  
  visibility: visible;  
  transform: translate(0, 0px);  
  opacity: 1;  
}
```

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-22 17:05:13

## 提示 `hint` 和插图编号 `callout`

`hint` == `callout` == 提示/警告/错误

演示用插件 `callouts` 实现的 `callout` == `hint`，即各种类型的提示/提醒的效果

语法：

```
> ##### {type}:: Your Title
>
> Content
```

其中 `{type}` 是下面中的任意一种：

- `primary`
- `success`
- `danger`
- `warning`
- `info`

如果你用过 `Bootstrap` 就能看出来，其实这几种标题类型就是 `Bootstrap` 中的标题的类型

效果如下的显示：

提示= `tip` = `note` = `info`


### 提醒类信息的标题

提醒类信息中的 内容

成功= `success` = 

### 成功的标题

成功中的 内容

警告= `warning` = 

### 警告的标题

警告中的 内容

错误= error = danger = ✖

## 错误的标题

错误中的 内容

## 其他几种(不常见的)类型

## Tag的标题

Tag中的 内容

---

## Note的标题

Note中的 内容

---

## Comment的标题

Comment中的 内容

---

## Hint的标题

Hint中的 内容

---

## Caution的标题

Caution中的 内容

---

## Quote的标题

Caution中的 内容

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-22 17:05:13

## 当前 **Gitbook** 源码目录结构

```
|— Makefile
|— README.md
|— book.json
|— book_current.json
|— debug/ # 用于保存debug时的文件
|— node_modules/ # 用于保存Gitbook安装后的各种plugin
└─ src/
    |— README.md
    |— SUMMARY.md
    |— appendix/
    |   |— README.md
    |   └─ reference.md
    |— assets/
    |   |— favicon.ico
    |   └─ img/
    |       └─ crifan_github_home.png
    |— chapter_1/
    |   └─ README.md
    |— chapter_2/
    |   |— README.md
    |   |— ch2_section_1.md
    |   └─ ch2_section_2.md
    └─ chapter_3/
        |— README.md
        |— ch3_code_highlight.md
        |— ch3_hint_callout.md
        └─ ch3_structure_list.md
```

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-22 17:05:13



## 基本安装

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

# Node.js 安装

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

## Gitbook安装

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

## Gitbook命令行速览

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

## 图书项目结构

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

## README.md 与 SUMMARY编写

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

# 目录初始化

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08

## 图书输出

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08



## 输出为静态网站

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

## 输出PDF

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08

# 发布

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

## 发布到Github Pages

hackerwu.cn，使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved，powered by Gitbook最后更新： 2021-10-23 19:36:08

# 结束

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08

## 附录

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08

## Reference

hackerwu.cn, 使用 [知识共享 署名-非商业性使用-禁止演绎 4.0 国际 许可协议](#)发布 all right reserved, powered by Gitbook最后更新: 2021-10-23 19:36:08