1. 两数之和

```
func twoSum(nums []int, target int) []int {
    //查找组合（A,B）满足 A+B=target
    for i := 0; i < len(nums); i++ { //先确定 A
        //B 的下标总是比 A 的下标大，避免类似（1，5）（5，1）这样的重复组合
        for j := i+1; j < len(nums); j++ {
            if nums[i] + nums[j] == target { //再确定 B
                return []int{i, j} //只有一个答案，找到就可以返回了
            }
        }
    }
    return nil
}
```

1108. ip 地址无效化

```
func defangIPaddr(address string) string {
    return strings.Replace(address, ".", "[.]", -1)
}
```

```
func defangIPaddr(address string) string {
  sb := strings.Builder{}
  for i := 0; i < len(address); i++ {
      c := address[i]
      if c != '.' {
          sb.WriteByte(c)
      } else {
          sb.Write([]byte("[.]"))
      }
  }
  return sb.String()
}
```

```
//用[]byte 实现 : byte 是 uint8 的别名，占 8 位，所以处理 ASCII 没问题
func defangIPaddr(address string) string {
```

```
    origin := []byte(address)
    n := len(origin)
    newN := n+2*3
    newString := make([]byte, newN)
    k := 0
    for i:= 0; i < n; i++ {
        if origin[i] != '.' {
            newString[k] = origin[i]
            k++
        } else {
            newString[k] = '[' //go 中 newString[k++] 编译不通过
            k++
            newString[k] = '.'
            k++
            newString[k] = ']'
            k++
        }
    }
    return string(newString)
}

//用[]rune：rune 是 int32 的别名，占 32 位，4 个字节，如果处理中文字符更适合用 rune
func defangIPaddr(address string) string {
    origin := []rune(address)
    n := len(origin)
    newN := n+2*3
    newString := make([]rune, newN)
    k := 0
    for i:= 0; i < n; i++ {
        if origin[i] != '.' {
            newString[k] = origin[i]
            k++
        } else {
            newString[k] = '['
            k++
            newString[k] = '.'
            k++
```

```
            newString[k] = ']'
            k++
        }
    }
    return string(newString)
}
```

344. 反转字符串

```
func reverseString(s []byte)  {
    n := len(s)
    for i := 0; i< n/2; i++ {
        s[i],s[n-i-1] = s[n-i-1], s[i]
    }
}

func reverseString(s []byte)  {
    n := len(s)
    i := 0
    j := n-1
    for i <= j {
        s[i],s[j] = s[j], s[i]
        i++
        j--
    }
}
```

剑指 Offer 58 - I. 翻转单词顺序

```
func reverseWords(s string) string {
    str := []byte(s)

    n := trim(str)
    if n == 0 {return ""}
    reverse(str, 0, n-1)
```

```
    for p := 0; p < n; {
        r := p
        for r < n && str[r] != ' ' {
            r++
        }
        reverse(str, p, r-1)
        p = r+1
    }
    //这里只是为了配合输出
    newStr := make([]byte, n)
    for i := 0; i < n; i++ {
        newStr[i] = str[i]
    }
    return string(newStr)

}

//原地删除前置空格和后置红歌，以及内部多余的空格，返回新字符串长度, 单词之间只留一个空格
func trim (str []byte) int {
    i := 0
    n := len(str)
    k := 0 //记录删除多余空格之后的数组长度
    for i < n && str[i] == ' ' {
        i++
    }
    for i < n {
        if str[i] == ' ' { //删除内部多余的空格和末尾空格
            if i+1 < n && str[i+1] != ' ' {
                str[k] = ' '
                k++
            }
        } else {
            str[k] = str[i]
            k++
        }
```

```
        i++
    }
    return k
}

//返回[p,r]之间的字符串， 注意这里是闭区间
//当然，前开后闭区间也可以，但代码中 i<=mid 应该改为 i<mid
func reverse(str []byte, p, r int) {
    mid := (p+r)/2
    for i := p; i <= mid; i++ {
        str[i], str[r-(i-p)] = str[r-(i-p)], str[i]
    }
}
```

125. 验证回文串

```
func isPalindrome(s string) bool {
    i := 0
    j := len(s) - 1
    for i < j {
        //不是数字或字母的话，i 就一直++
        if !isAlpha(s[i]) {
            i++
            continue
        }
        //不是数字或字母的话，j 就一直--
        if !isAlpha(s[j]) {
            j--
            continue
        }
        //走到这里的话， i, j 都指向数字或字母， 看看两个字符是否相等
        if toLower(s[i]) != toLower(s[j]) {
            return false
        } else {
            // i 和 j 往中间挪一位
            i++
```

```
            j--
        }
    }
    return true
}

//大写转小写
func toLower(c byte) byte{
    if c >= 'a' && c <= 'z' {return c}
    if c >= '0' && c <= '9' {return c}
    //ASCII 码：大写 A~Z 65～90，小写 a~z 97~122
    return c+32
}

//判断是不是数字或字母
func isAlpha(c byte) bool{
    if c >= 'a' && c <= 'z' {return true}
    if c >= 'A' && c <= 'Z' {return true}
    if c >= '0' && c <= '9' {return true}
    return false
}
```

9. 回文数

```
func isPalindrome(x int) bool {
    //-2147483648 ~ 2147483647
    digits := make([]int, 10)
    if x < 0 {return false}
    k := 0
    //将 x 转化成字符串数组
    for x != 0 {
        digits[k] = x % 10
        x = x / 10
        k++
    }
```

```
    //判断回文串
    for i := 0; i < k/2; i++ { //举例验证
        if digits[i] != digits[k-i-1] { //举例验证
            return false
        }
    }
    return true
}


func isPalindrome(x int) bool {
    //-2147483648 ~ 2147483647
    if x < 0 {return false}
    backupX := x
    y := 0 //y 为 x 反转之后的值
    for x != 0 { //将 x 转化成字符串数组的过程计算 y
        y = y*10 + x % 10
        x = x / 10
    }
    return backupX == y
}
```

58. 最后一个单词的长度

```
func lengthOfLastWord(s string) int {
    n := len(s)
    i := n-1
    for i >= 0 && s[i] == ' ' {
        i--
    }
    if i < 0 {return 0}
    len := 0
    for i >= 0 && s[i] != ' ' {
        len++
        i--
    }
```

```
    return len
}
```

剑指 Offer 05. 替换空格

```
//跟“IP 地址无效化”相同
func replaceSpace(address string) string {
    sb := strings.Builder{}
    for i := 0; i < len(address); i++ {
        c := address[i]
        if c != ' ' {
            sb.WriteByte(c)
        } else {
            sb.Write([]byte("%20"))
        }
    }
    return sb.String()
}
```

剑指 Offer 58 - II. 左旋转字符串

```
//解法 1 往左移动 n 位
func reverseLeftWords(s string, n int) string {
    str := []byte(s)
    for i := 0; i < n; i++ { //移动 n 次，每次左移 1 位
        tmp := str[0]
        for j := 1; j < len(str); j++{
            str[j-1] = str[j]
        }
        str[len(str)-1] = tmp
    }
    return string(str)
}

//解法 2 往左移动 n 位
func reverseLeftWords(s string, n int) string {
```

```
    tmp := make([]byte, len(s))
    //数组分为[0~n~len],先把 0～n-1 放在 tmp 后面
    for i := 0; i < n; i++ {
        tmp[i+(len(s)-n)] = s[i]
    }
    //再把 n～len-1 放到 tmp 的前面
    for i := n; i < len(s); i++ {
        tmp[i-n] = s[i]
    }
    return string(tmp)
}

//解法 3 用 go slice 实现
func reverseLeftWords(s string, n int) string {
    str := []byte(s)
    tmp := s[:n] //[0~n)， 下标 0~n-1, 不含 n
    //str[n:]表示 [n~len),含 n
    str = append(str[n:], tmp...) //[0~n) 拼接到[n~len)后面
    return string(str)
}
```

26. 删除排序数组中的重复项

```
func removeDuplicates(nums []int) int {
    if len(nums) == 0 {return 0}
    n := len(nums)
    k := 0 //[0,k]栈中元素
    for i := 1; i < n; i++ {
        if nums[i] != nums[k] { //放入栈
            k++
            nums[k] = nums[i]
        }
    }
    return k+1
}
```

剑指 Offer 67.把字符串转换成整数

```go
func strToInt(str string) int {
    chars := []byte(str)
    n := len(chars)
    //处理空
    if n == 0 {return 0}
    //处理前置空格
    i := 0
    for i < n && chars[i] == ' ' {
        i++
    }
    //全为空格
    if i == n {return 0}
    //处理符号
    sign := 1
    c := chars[i]
    if c == '-' {
        sign = -1
        i++
    } else if c == '+' {
        sign = 1
        i++
    }
    //真正处理数字
    //整数范围 -2147483648 ~ 2147483647
    intAbsHigh := 214748364
    result := 0
    for i < n && chars[i] >= '0' && chars[i] <= '9' {
        d := chars[i] - '0'
        //判断再乘以 10， 加 d 之后，是否越界
        if result > intAbsHigh {
            if sign == 1 {
                return math.MaxInt32    //2147483645d
            } else {
                return math.MinInt32    //-2147483645d
            }
```

```
// "   -42" 前导空格， 符号位
// "4193abc" 后置非数字字符
// "abc978" 返回 0
// "-9182838383838" 超过整数范围 返回 math.MinInt32
// "9182838383838" 超过整数范围 返回 math.MaxInt32
// "" 空字符串 返回 0
// "   " 全文空格 返回 0
```

```
        }
        if result == intAbsHigh {
            if (sign == 1) && (d > 7) {
                return math.MaxInt32    //2147483648
            }
            if (sign == -1) && (d > 8) {
                return math.MinInt32    //-2147483649
            }
        }
        //正常逻辑
        result = result * 10 + int(d)
        i++
    }
    return sign * result
}
```