# 基本数据类型

## 整数

- 有符号
  - int8
  - int16
  - int32 — rune类型为int32类型
  - int64

  *int和int32是两种不同的类型，int会根据平台选择合适的大小*
  *若想要使用int32表示int，则必须使用显式转换*

- 无符号
  - uint8 — byte类型为uint8类型，强调原始数据
  - uint16
  - uint32
  - uint64

  *uint和uint32同样是不同的类型，使用时必须进行显式转换，即T(x)*

- 尽管Go提供了无符号整数，但是在绝大部分时刻，依然会选择使用有符号整数
  如代码所示，由于uint32恒大于0，两轮迭代后，i--将使得变成uint32的最大值，从而引发slice越界panic

```go
func main() {
    names := []string{"smart", "mario"}
    for i := uint32(len(names) - 1); i >= 0; i-- {
        fmt.Println(names[i])
    }
}
```

- **无符号整数通常用于解析二进制格式文件，散列或者加密，以及位运算等场景** — **有符号数的右移操作(>>)是按符号位的值填补空位，所以将整数位模式处理时，必须使用无符号整数**

- 在网络协议中，经常要做的事情就是将字节数组转换成整数

```go
func main() {
    // 3字节的字节数组,长度24bit,golang没有int24,所以需要转换
    s := []byte{1, 2, 3}
    // 假设大端直接,按位大端处理
    k := uint32(s[0]) << 16 | uint32(s[1]) << 8 | uint32(s[2])
    fmt.Println(k)
}
```

- 由于Golang提供了较为方便的byte数组，实现一个BitMap也较为简单

## 浮点数与布尔值

- float32
- float64
- true/false

## 指针

- 同C语言一样，Go存在指针类型，并使用指针来引用某一块内存区域 — Go指针要比C指针要简单许多，原因在于Go指针不能进行运算
- 指针操作符和C语言一样，使用`&`取变量地址，使用`*`取指针所指向的变量

```go
k := 10
ptr := &k       // 定义指针
value := *ptr   // 对指针解引用
```

- 指针的初始值为nil，表示不指向任何内存区域，当对这种指针进行操作时，会抛出空指针异常

```go
var ptr *int        // 此时ptr为nil
value := *ptr       // 异常
*ptr = 1            // 异常
```

- 指针的占用空间大小与机器相关，在32位机器上，指针占用4个字节; 在64位机器上，指针占用8个字节。与所指向的内容无关，毕竟指针保存的实际上是内存地址

## 字符串

- Go内置基本数据类型，为不可变字节序列，不可变性与Python相同
  - **内置len()方法将返回字符串字节数，而非字符数量**
  - **s[i]下标访问则取第i个字符，而非字节，下标起始值为0**
  - **s[i:j]将产生一个新的子串，单该子串与原有字符串共用同一段内存**
- Unicode — Unicode囊括了世界上所有文字体系的全部字符，总计12万余个
  - Go使用int32进行存储，并赋予别名: rune
  - **rune其实就是int32** — 即一个Unicode字符需要4个字节的存储空间
- UTF-8
  - Unicode可以使用int32进行存储，这种方式也称为UTF-32，每个Unicode字符的长度相同，都是32位。但是ASCII码只需要8字节就能存储，所以Unicode在网络传输上会有一定的浪费
  - UTF-8以字节为单位，对Unicode码作变长编码，每个字符采用1~4个字节表示，其中ASCII码仅占1个字节，其余常用文字字符占2或3个字节
  - UTF-8是自描述的编码方式
    - ASCII: 单字节，二进制编码最高位为0 — 0xxxxxxx
    - 2字节字符: 最高位为110，第二个字节以10开头 — 110xxxxx 10xxxxxx
    - 3字节字符: 最高位为1110，后续字节以10开头 — 1110xxxx 10xxxxxx 10xxxxxx
    - 4字节字符: 最高位为11110，后续字节以10开头 — 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

    *非常方便的进行解码，而且无需超前预读*
- unicode包 — 针对单个文字符号的函数，如大小写转换，区分数字和字母
  - ToUpper
  - ToLower
  - IsDigit

  *针对`rune`类型(单个unicode字符)*
- utf8包 — 提供了按utf-8编码和解码文字符号的函数
  - **RuneCountInString** — 统计字符串字符个数
  - DecodeRuneInString
- strings包 — 提供对字符串相关的函数，如搜索，替换，比较，修正，切分和连接等方法
  - HasPrefix/LastIndex/Contains/Index
  - Replace
  - Join/Split
- bytes包
  - 基本上实现了strings包所有的常用函数，只不过操作对象由string改为字节slice
  - bytes.Buffer提供各种write以及read方法，其用途极为广泛
    - WriteByte
    - WriteRune
    - WriteString
- strconv包 — 提供了字符串和数字之间的相互转换
  - int to string — **strconv.Itoa** — Integer to ASCII
  - string to int — **strconv.Atoi** — ASCII to integer
- 常量: Go枚举类型的实现
  - 单个常量定义 — const var = expression
  - 多个常量定义 —
    ```
    const (
        foo = expression
        bar = expression
    )
    ```
  - 常量生成器: iota
    - 初始值为0，逐项加1
    - 枚举值定义

```go
const (
    MAN uint8 = iota + 1  // 1
    WOMEN                 // 2
    UNKNOWN               // 3
)
```