Go底层探索(二):字符串

刘庆辉 猿码记 2023-01-28 03:30 Posted on 北京

收录于合集
#Go进阶 14 #Go 101

1. 介绍

> **@注: 以下内容来自《Go语言底层原理剖析》书中的摘要信息，本人使用版本(`Go1.18`) 与书中不一致，源码路径可能会有出入。**

字符串在编程语言中无处不在，程序的源文件本身就是由众多字符组成的，在程序开发中的存储、传输、日志打印等环节，都离不开字符串的显示、表达及处理。因此，字符与字符串是编程中最基础的学问。不同的语言对于字符串的结构、处理有所差异。

1.1 字符串长度

在编程语言中，字符串是一种重要的数据结构，通常由一系列字符组成。字符串一般有两种类型:

- 一种: 在编译时指定长度，不能修改。
- 另一种:具有动态的长度，可以修改。

> 在Go语言中，字符串不能被修改，只能被访问。
>
> **错误示例:**

```
var str = "hello word"
// 这里想把e改成o，不支持
str[1]='o'
```

1.2 字符串的终止方式

字符串的终止有两种方式:

- 一种是 C 语言中的隐式申明，以字符 “\0” 作为终止符。
- 一种是 Go 语言中的显式声明。

2. 结构&内存

2.1 数据结构

Go 语言运行时字符串 string 的表示结构如下：

```
type StringHeader struct {
 Data uintptr
 Len  int
}
```

- Data：指向底层的字符数组。
- Len：代表字符串的长度。

> 字符串在本质上是一串字符数组。

2.2 内存占用

Go 语言中所有的文件都采用 UTF-8 的编码方式，同时字符常量使用 UTF-8 的字符编码集。

> UFT-8 是一种长度可变的编码方式，可包含世界上大部分的字符。在 UTF-8 中，大部分字符都只占据 1 字节，但是特殊的字符（例如大部分中文）会占据3字节。

下面示例: 变量 str 看起来只有 4 个字符，但是 len (str) 获取的长度为 8 ，字符串 str 中每个中文都占据了 3 字节。

```
func TestRun(t *testing.T) {
 str := "Go语言"
 fmt.Println("str 长度:", len(str))
}
/**
str 长度: 8
*/
```

3. 字符串解析

3.1 解析源码

字符串常量在词法解析阶段最终会被标记成 StringLit 类型的 Token 并被传递到编译的下一个阶段。在语法分析阶段，采取递归下降的方式读取 Uft-8 字符，单撇号或双引号是字符串的标识。

go1.18/src/cmd/compile/internal/syntax/scanner.go:88

```
func (s *scanner) next() {
...
 switch s.ch {
 case -1:
 ...
 case '"':
  s.stdString()
 case '`':
    s.rawString()
```

```
    ...
}
```

根据上面解析源码可以得知:

- 如果在代码中识别到单撇号，则调用 `rawString` 函数;
- 如果识别到双引号，则调用 `stdString` 函数;

> `rawString` 函数和 `stdString` 函数对字符串的解析处理也略有不同;

3.2 解析函数: rawString

对于单撇号的处理比较简单：一直循环向后读取，直到寻找到配对的单撇号，源码如下:

`go1.18/src/cmd/compile/internal/syntax/scanner.go:706`

```
func (s *scanner) rawString() {
 ok := true
 s.nextch()
 for {
  if s.ch == '`' {
   s.nextch()
   break
  }
  if s.ch < 0 {
   s.errorAtf(0, "string not terminated")
   ok = false
   break
  }
  s.nextch()
 }
 s.setLit(StringLit, ok)
}
```

3.3 解析函数:stdString

双引号调用 `stdString` 函数，如果出现另一个双引号则直接退出；如果出现了 `\\` ，则对后面的字符进行转义。

go1.18/src/cmd/compile/internal/syntax/scanner.go:674

```
func (s *scanner) stdString() {
 ok := true
 s.nextch()

 for {
  if s.ch == '"' {
   s.nextch()
   break
  }
  if s.ch == '\\' {
   s.nextch()
   if !s.escape('"') {
    ok = false
   }
   continue
  }
    // 双引号中不能出现换行符
  if s.ch == '\n' {
   s.errorf("newline in string")
   ok = false
   break
  }
  if s.ch < 0 {
   s.errorAtf(0, "string not terminated")
   ok = false
   break
  }
  s.nextch()
 }
 s.setLit(StringLit, ok)
}
```

> @注:在双引号中不能出现换行符，以下代码在编译时会报错：newline in string。这是通过对每个字符判断r=='\n'实现的。

4. 字符串拼接

4.1非运行时拼接

在 Go 语言中，可以方便地通过加号操作符 (+) 对字符串进行拼接。如下代码:

```
func main(){
  str := "hello " + "world"
}
```

由于数字的加法操作也使用 + 操作符，因此需要编译时识别具体为何种操作:

- 在**抽象语法树阶段**: 当加号操作符两边是字符串时,具体操作的 Op 会被解析为 OADDSTR ；
- 在**语法分析阶段**: 调用 noder.sum 函数,将所有的字符串常量放到字符串数组中，然后调用 strings.Join 函数完成对字符串常量数组的拼接。

4.2 运行时拼接

运行时字符串的拼接原理如图5-1所示，其并不是简单地将一个字符串合并到另一个字符串中，而是找到一个更大的空间，并通过内存复制的形式将字符串复制到其中。

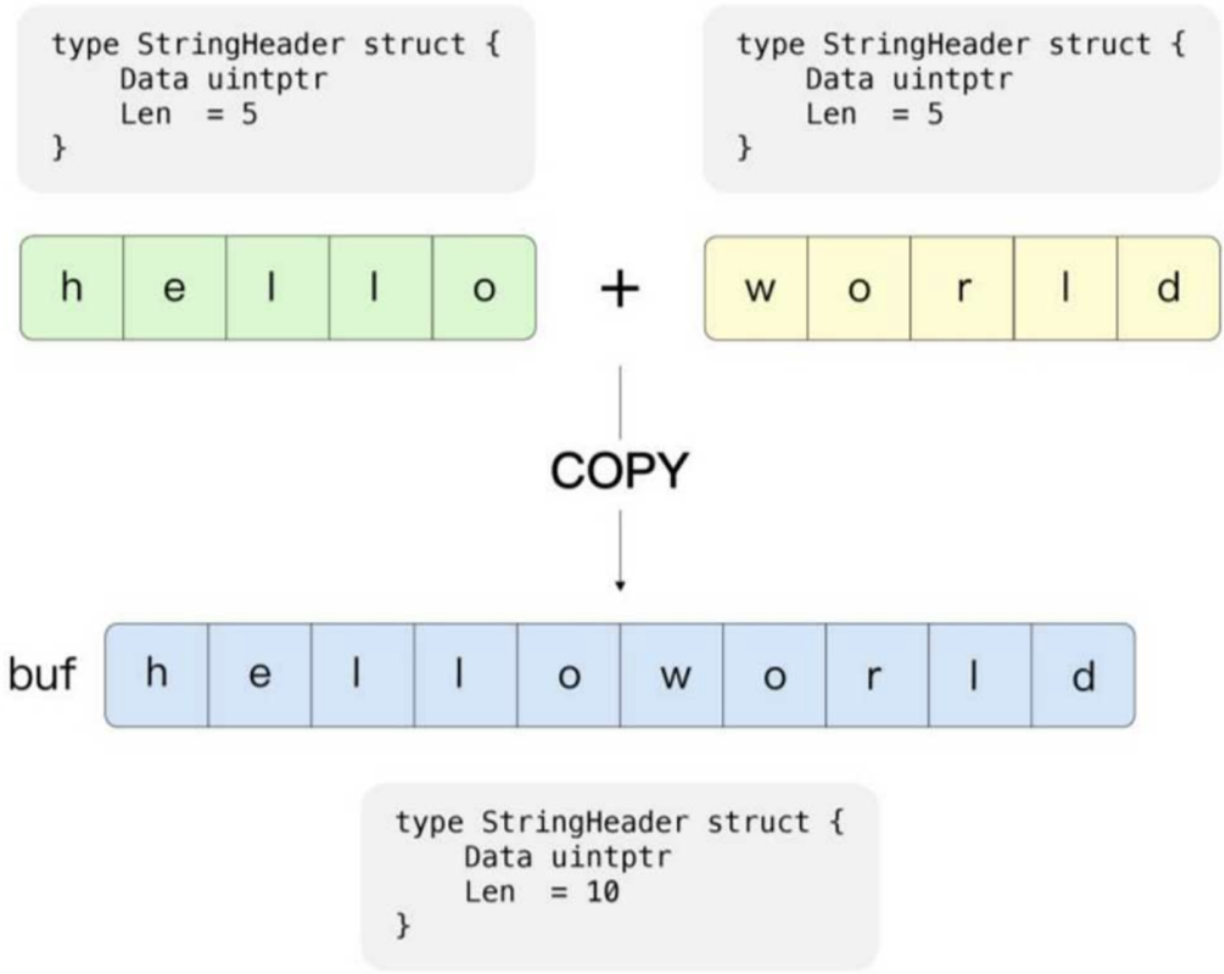

图5-1 字符串拼接原理

拼接后的字符串大于或小于32字节时的操作:

- 当拼接后的字符串小于32字节时，会有一个临时的缓存供其使用。
- 当拼接后的字符串大于32字节时，堆区会开辟一个足够大的内存空间，并将多个字符串存入其中，期间会涉及内存的复制.

5. 与字节数组转换

字节数组与字符串可以相互转换。如下所示，字符串 `a` 强制转换为字节数组 `b` ，字节数组 `b` 强制转换为字符串 `c` 。

```go
a := "hello go"
// a强制转换为字节数组b
b := []byte(a)
// 字节数组b强制转换为字符串c
c := string(b)
```

5.1 注意事项

字节数组与字符串的相互转换**并不是简单的指针引用，而是涉及了内存复制**；

- 当字符串小于 `32` 字节时：可以直接使用缓存 `buf` ；
- 当字符串大于 `32` 字节时：需要向堆区申请足够的内存空间。最后使用 `copy` 函数完成内存复制。

> @注：在涉及一些密集的转换场景时，需要评估这种转换带来的性能损耗。

戳"阅读原文"我们一起进步

收录于合集 #Go 101

上一篇

Go底层探索(一):编译器

下一篇

Go底层探索(三):切片