你是不是对切片有什么误解

一、本节内容

主要揭秘Go语言之中slice的详细用法，避免开发者在开发过程掉入一些不该有的陷阱，同时也会介绍一些slice的高级用法。

二、知识点概要

- 切片的越界问题
- 切片的扩容问题
- 切片的地址问题
- 切片的字符编码问题
- 切片与函数参数
- 切片取值时设置容量

1. 切片的越界问题

经过Go语言的入门，我们都已经知道切片是可以扩展的动态数组，但是切片在使用过程中，仍然要注意不可以越界，也就是说切片在访问时一定要确保该下标存在，下面就是一个失败访问的例子。

```
/*
  company: Pdj
  autor  : Yekai
  email  : yekai_23@sohu.com
  file   : 01-out-of-range.go
*/
package main

import "fmt"

func main() {

  // 创建5个元素的切片
  slice := make([]string, 5)
  slice[0] = "yekai"
  slice[1] = "fuhongxue"
  slice[2] = "luxiaojia"
  slice[3] = "lixunhuan"
  slice[4] = "jingwuming"
```

```
  // 下面这样就会越界
  slice[5] = "Runtime error"

  // Error: panic: runtime error: index out of range

  fmt.Println(slice)
}
```

执行上述代码，我们将会看到“panic: runtime error: index out of range“这样的错误。

2. 切片的增长问题

我们都知道，在使用append函数时，切片可以自动扩充容量，那么你知道切片是如何扩容的吗？来看看下面的代码：

```
/*
  company: Pdj
  autor  : Yekai
  email  : yekai_23@sohu.com
  file   : 02_cap_increment.go
*/
package main

import "fmt"

func main() {

  // 定义一个空切片
  var data []string

  // 计算切片的容量，大小是多少?
  lastCap := cap(data)

  // Append ~100k strings to the slice.
  for record := 1; record <= 1e5; record++ {

    // 使用append向切片追加字符串元素
    data = append(data, fmt.Sprintf("Rec: %d", record))

    // 当切片容量变化时打印一下
    if lastCap != cap(data) {

      // 计算容量扩充百分比
      capChg := float64(cap(data)-lastCap) / float64(lastCap) * 100

      // 保留最新容量大小
      lastCap = cap(data)

      fmt.Printf("Addr[%p]\tIndex[%d]\t\tCap[%d - %2.f%%]\n",
```

```
            data,
            record,
            cap(data),
            capChg)
        }
    }
}
```

执行一下看看，你就会发现，切片的增长与你想象的不太一样，你会发现另外一个事实就是切片是个地址，而且随着append的使用，这个地址还是变的！

3. 切片的引用问题

3.1 猜一猜

```
/*
    company : Pdj
    author  : Yekai
    email   : yekai_23@sohu.com
    file      : 03_share.go
*/
package main

import "fmt"

type user struct {
    age int
}

func main() {

    users := make([]user, 3)

    users[1].age = 30
    shareUser := &users[1]

    //年龄自增
    shareUser.age++

    //遍历打印全部切片
    for i := range users {
        fmt.Printf("User: %d Likes: %d\n", i, users[i].age)
    }

    // Add a new user.
    users = append(users, user{})

    //年龄再增加
    shareUser.age++
```

```
  // Display the number of age for all users.
  fmt.Println("-------------------------")
  for i := range users {
    fmt.Printf("User: %d Likes: %d\n", i, users[i].age)
  }

}
```

相信大多数对c语言比较熟悉的人都会猜错，真实的执行结果竟然是这样的：

```
localhost:slice yk$ go run 03_share.go
User: 0 Likes: 0
User: 1 Likes: 31
User: 2 Likes: 0
-------------------------
User: 0 Likes: 0
User: 1 Likes: 31
User: 2 Likes: 0
User: 3 Likes: 0
```

3.2 总结

切片是动态变化的，地址也并非一成不变的，当使用append的时候，原来的引用将会失效！

4. 切片与字符编码

我们都知道Go语言的字符编码是UTF-8，而我们也应该知道一个中文字符在UTF-8编码下占3个字节，本节我们就用一个例子来给大家演示一下一个中英文混杂的字符编码如何通过切片详细的展现出来。

```
/*
  company: Pdj
  autor  : Yekai
  email  : yekai_23@sohu.com
  file   : 04_utf8.go
*/
package main

import (
  "fmt"
  "unicode/utf8"
)

func main() {
```

```
  //定义一个中英文混杂的字符串
  s := "hello 中国"

  // UTFMax is 4 rune的长度是4个byte
  var buf [utf8.UTFMax]byte

  // 遍历字符串
  for i, r := range s {

    //计算每个字符串元素的rune长度,
    rl := utf8.RuneLen(r)

    // 根据i计算其在字符串的偏移位置
    si := i + rl

    // 将偏移内容考拷贝到buf
    copy(buf[:], s[i:si])

    // 打印内容
    fmt.Printf("%2d: %q; codepoint: %#6x; encoded bytes: %#v\n", i, r, r,
buf[:rl])
  }
}
```

执行结果如下：

```
localhost:slice yekai$ go run 04_utf8.go
 0: 'h'; codepoint:   0x68; encoded bytes: []byte{0x68}
 1: 'e'; codepoint:   0x65; encoded bytes: []byte{0x65}
 2: 'l'; codepoint:   0x6c; encoded bytes: []byte{0x6c}
 3: 'l'; codepoint:   0x6c; encoded bytes: []byte{0x6c}
 4: 'o'; codepoint:   0x6f; encoded bytes: []byte{0x6f}
 5: ' '; codepoint:   0x20; encoded bytes: []byte{0x20}
 6: '中'; codepoint: 0x4e2d; encoded bytes: []byte{0xe4, 0xb8, 0xad}
 9: '国'; codepoint: 0x56fd; encoded bytes: []byte{0xe5, 0x9b, 0xbd}
```

5. 函数参数与切片

在Go语言的函数中，我们有时会看到"..."这样的参数，类似于C语言当中的变参函数，我们在Go语言当中也可以通过...让函数的参数更灵活。

比如我们有这样一个Person的结构体：

```
// 定义一个person结构体
type Person struct {
  id   int
  name string
}
```

再定义一个打印和修改的函数：

```
// 打印函数，注意参数设计
func display(persons ...Person) {
  fmt.Println("**************************")
  for _, u := range persons {
    fmt.Printf("%+v\n", u)
  }
}

// 修改函数，注意参数
func change(persons ...Person) {
  persons[1] = Person{99, "wuyazi"}
}
```

display的参数类型是 ...Person，这样就代表着传入的参数可以是多个Person类型对象的集合。由于切片本身也是数据的集合，所以切片也可以作为参数传入，只不过在切片传入时，后面需要携带"..."。

整体测试代码如下：

```
/*
  company: Pdj
  autor  : Yekai
  email  : yekai_23@sohu.com
  file   : 05_params.go
*/
package main

import "fmt"

// 定义一个person结构体
type Person struct {
  id   int
  name string
}

func main() {

  // Declare and initialize a value of type user.
  p1 := Person{
    id:   9527,
    name: "yekai",
```

```
  }

  // Declare and initialize a value of type user.
  p2 := Person{
    id:   9528,
    name: "lixunhuan",
  }

  // 显示2个人的信息
  display(p1, p2)

  // 创建一个切片
  ps := []Person{
    {1007, "qiaofeng"},
    {1008, "xuzhu"},
  }

  // 打印切片
  display(ps...)

  change(ps...)
  fmt.Println("----------------------------")
  for _, p := range ps {
    fmt.Printf("%+v\n", p)
  }
}

// 打印函数，注意参数设计
func display(persons ...Person) {
  fmt.Println("*************************")
  for _, u := range persons {
    fmt.Printf("%+v\n", u)
  }
}

// 修改函数，注意参数
func change(persons ...Person) {
  persons[1] = Person{99, "wuyazi"}
}
```

6. 切片取值的同时设置容量

我们学过切片都知道，切片可以用下面的方式切内容：

```
[start:end]
```

只不过这个end是开区间的，所以从某种意义来说，切片的取值方式更像是下面这样的描述：

```
[start:start+length]
```

当然，这也没什么，本节关键想说的是切片的双冒号取值，例如下面这种：

```
takeOneCapOne := slice[2:3:3]
```

我们都知道切片有len和cap两类表示，其实第二个冒号后的内容描述的就是cap的性质，不过cap是多少要这样去理解。

```
takeOneCapOne := slice[start:start+length:start+cap]
```

通过上述的公式，我们就可以知道slice[2:3:3]想表达的含义了，取下标为2的元素，并且构造成容量为1的切片。详细执行代码可以看看下面例子：

```
/*
  company: Pdj
  autor  : Yekai
  email  : yekai_23@sohu.com
  file   : 06_third_index.go
*/
package main

import "fmt"

func main() {

  //切5个水果
  slice := []string{"Apple", "Orange", "Banana", "Grape", "Plum"}
  printSlice(slice)

  // 把第3个元素请出来
  takeOne := slice[2:3]
  printSlice(takeOne)

  //动用第三个index位置，来设置cap为1
  takeOneCapOne := slice[2:3:3]
  printSlice(takeOneCapOne)

  // 添加一个新元素
  takeOneCapOne = append(takeOneCapOne, "Kiwi")
  printSlice(takeOneCapOne)
}

// 打印slice的详细信息，包括length，capacity，以及各个元素的情况
func printSlice(slice []string) {
  fmt.Printf("Length[%d] Capacity[%d]\n", len(slice), cap(slice))
  for i, s := range slice {
```

```
    fmt.Printf("[%d] %p %s\n",
      i,
      &slice[i],
      s)
  }
}
```

三、 总结

切片是Go语言之中使用非常灵活的容器，几乎无处不在，切片使用过程中的一些坑还是应尽量避免，比如切片本身就是一个地址，切片内容改变后，地址很可能会发生变化，原地址会失效。