GO-02 03: Основы языка. Слайсы

Описание:

Наряду с массивами в Go есть еще один тип данных, который, в отличие от массивов, вы будете очень часто использовать в своей практике — это слайсы или срезы.
Глядя на объявление слайса, может показаться, что он является тем же массивом, хотя это не так.

```
var s []int
s = []int{4, 5, 6, 7}
```

Чтобы было нагляднее, лучше заглянуть в исходный код и найти описание структуры слайса.
Файл: src/runtime/slice.go

```
type slice struct {
  array unsafe.Pointer
  len   int
  cap   int
}
```

Как видно по коду, слайс представляет собой структуру. В его основе лежит массив, указатель на который хранится в слайсе. Кроме этого, есть два атрибута: len - длина и cap - вместимость.
Длина показывает, какое количество элементов в данный момент находится в слайсе. В отличие от массива, слайс может изменять свой размер в процессе выполнения программы. Размер слайса можно узнать при помощи уже известной встроенной функции len().
На свойстве вместимости следует остановиться подробнее. Вместимость среза показывает, сколько еще элементов можно добавить до того, как Go осуществит новую аллокацию памяти.
Когда объявляется срез, Go производит выделение памяти, которой будет достаточно для хранения исходного массива указанной размерности. Если известна длина, то, как и в случае с массивом, Go заполняет массив нужным количеством элементов со значением по умолчанию для типа, который был указан в выражении инициализации слайса. При этом значение атрибута len равно количеству проинициализированных элементов, а cap — значению len. Узнать, какая вместимость в данный момент у слайса, можно с помощью функции cap().

```
var s []int

fmt.Println(len(s))       // 0
fmt.Println(cap(s))       // 0
```

Доступ к элементам слайса можно получить по индексу.

```
s := []string{"e", "l", "e", "m", "e", "n", "t"}
s[0] = "E"
fmt.Println(s)          // [E l e m e n t]
```

При попытке указать индекс за границей длины слайса компилятор Go покажет ошибку, сообщающую о выходе за пределы среза.

```
s[10] = "E"  // panic: runtime error: index out of range [10] with
length 7
fmt.Println(s)
```

Так как слайс является ссылочным типом, то, как упоминалось ранее, значением по умолчанию для слайса является nil. При этом длина и вместимость будут равны 0.
Объявление слайса и функция make()
В примере выше мы объявили слайс через ключевое слово var и видим, что длина и вместимость равны нулю.
Объявлять слайс можно через литерал или при помощи встроенной функции make(). При объявлении через литерал длина слайса будет равна количеству указанных значений, а вместимость — длине.

```
s := []string{"e", "l", "e", "m", "e", "n", "t"}
fmt.Println(len(s), cap(s))             // len = 7, cap = 7
```

В случае использования функции make() помимо типа элементов слайса можно явно указать начальную длину или вместимость. Функция принимает на вход тип элементов слайса, длину и необязательный аргумент вместимость. В результате вызова функции получим слайс с заданной длиной и емкостью.

```
s := make([]int, 4, 6)
fmt.Println(s, len(s), cap(s))
```

Стоит отметить, что, так как в основе слайса лежит указатель на массив, на один и тот же базовый массив могут ссылаться разные слайсы, причем слайсы могут пересекаться и ссылаться на одни и те же элементы.

```
s := []string{"e", "l", "e", "m", "e", "n", "t"}
s1 := s[:4]
s2 := s[1:6]
fmt.Println(s1, s2)             // [e l e m] [l e m e n]
s1[1] = "L"
fmt.Println(s, s2)             // [e L e m e n t] [L e m e n]
```

На примере выше видно, что мы объявили слайс s, в основе которого лежит массив с элементами. Далее из этого слайса сформировали два новых слайса: s1, в который

вошли элементы с индексами от 0 включительно до 4 исключительно, и s2, в который вошли элементы с индексами от 1 включительно до 6 исключительно. Эти операции мы осуществили при помощи оператора : (слайс). Далее в слайсе s1 мы изменили элемент с индексом 1, и в результате все слайсы, которые ссылались на один и тот же базовый массив, изменились.
Но что будет, если добавить в один из слайсов какое-то количество элементов? Рассмотрим ответ на этот важный вопрос далее.

Добавление элементов в слайс

Мы вплотную подошли к вопросу добавления элементов в слайс и тем особенностям, которые при этом происходят.
Для добавления новых элементов в конец слайса реализована функция append(), которая принимает первым аргументом исходный слайс, а далее — один или несколько элементов, которые необходимо добавить. Если понадобится вставить элементы одного слайса в другой, то можно применить оператор … , который может упаковать или распаковать слайс и передать его элементы в качестве аргументов в функцию. В качестве результата работы функция append() возвращает новый слайс, который содержит новые элементы. Поэтому, как правило, при выполнении добавления элементов в слайс, его перезаписывают новым.

```
s := []int{1, 2, 3}
p := []int{4, 5, 6}

s = append(s, p...)
fmt.Print(s)         //[1 2 3 4 5 6]
```

Или вариант без использования оператора …

```
s := []int{1, 2, 3}

s = append(s, 22, 33, 44)
fmt.Print(s)         //[1 2 3 22 33 44]
```

Вернемся к нашему предыдущему примеру с несколькими слайсами и узнаем, что произойдет со слайсами, которые были образованы от одного базового массива.

```
s := []string{"e", "l", "e", "m", "e", "n", "t"}
s1 := s[:4]
s2 := s[1:6]
fmt.Println(s1, s2)              // [e l e m] [e m e n]
s1[1] = "L"
fmt.Println(s, s2)           // [e L e m e n t] [L e m e n]

fmt.Println(cap(s1))          // cap = 7

s1 = append(s1, "one", "two", "three", "four")
```

```
fmt.Println(s, s1, s2)       // [e L e m e n t] [e L e m one two three
four] [L e m e n]

fmt.Println(cap(s1))         // cap = 14
```

Когда мы запускали этот пример в первый раз, то при изменении элемента слайса он менялся во всех пересекающихся слайсах. Но сейчас мы добавили несколько новых элементов в один из слайсов, тем самым превысив вместимость слайса. Go выполнил процедуру аллокации памяти и перенес слайс в другое место, дополнив его новыми значениями, после чего поместил в исходный слайс указатель на новый базовый массив. Как вы понимаете, перемещение слайса - это дорогая операция и, чтобы избежать слишком частых перемещений, Go увеличил вместимость не просто на len(s1) + 1, а в два раза.

Копирование слайсов

В Go реализована функция copy(), которая позволяет скопировать поэлементно один слайс в другой. В качестве первого аргумента функция принимает слайс, в который будут добавлены новые элементы. Вторым аргументом передается слайс, из которого будут скопированы элементы. Возвращаемый функцией результат представляет собой количество скопированных элементов — минимальное из значений len(src) или len(dst).

```
src := []int{1, 2, 3, 4, 5}
dst := make([]int, 3, 10)

qnt := copy(dst, src)
fmt.Print(qnt, dst)          // 3 [1 2 3]
```

Как видно на примере, слайс dst содержит три элемента, скопированных из слайса src. Даже несмотря на то, что вместимость слайса dst превышает длину слайса src, функция копирует только то количество элементов, которое не превышает текущее значение длины слайса dst.

Полезные ссылки:

- Go Slices: usage and internals
- The anatomy of Slices in Go
- Go: Slice and Memory Management

Задание:

1. Создайте в проекте module02 новую ветку 03_task.
2. Создайте новую директорию. В ней создайте файл main.go. Напишите код, в котором:

- из слайса с днями недели надо скопировать в новый слайс рабочие дни, а из исходного слайса удалить скопированные, чтобы остались только выходные дни.
- Выведите на экран слайсы с выходными и рабочими днями недели.

3. Создайте новую директорию с файлом main.go. Напишите код, в котором:

- нужно объединить слайс с выходными днями и слайс с рабочими в один слайс. Выведите на экран итоговый слайс с днями.

4. Создайте новый коммит и отправьте его в удаленный репозиторий проекта.
5. В ответе пришлите ссылку на merge request в ветку master своего проекта ветки 03_task.