Раздел 4. Коллекции

Коллекции в Go, как и в других языках программирования, - это структуры данных, которые обеспечивают эффективное хранение и обработку группы элементов. Другими словами, это своего рода контейнеры, которые позволяют собирать вместе различные элементы и работать с ними как с единой структурой.

В Go существуют разные типы коллекций, включая массивы, срезы (slices) и карты (maps). Массивы - это простейший тип коллекции, который хранит фиксированное количество элементов одного типа. Срезы похожи на массивы, но они более гибкие, так как могут изменять свой размер. Карты - это структуры данных, которые хранят пары ключ-значение и позволяют быстро искать значения по их ключам.

Коллекции предоставляют ряд полезных методов для работы с данными. Например, вы можете добавлять новые элементы в коллекцию, удалять существующие элементы или искать элементы по определенным критериям. Это делает коллекции важным инструментом для обработки данных в Go.

Представьте, что вы работаете с базой данных клиентов. Вместо того чтобы хранить каждого клиента как отдельную сущность, вы можете использовать коллекцию для хранения всех клиентов вместе. Это упрощает обработку данных, так как вы можете использовать методы коллекции для быстрого и эффективного поиска, добавления или удаления клиентов.

Массивы

Массив - это коллекция элементов фиксированной длины, которые могут быть одного типа данных. В Go, массивы могут быть объявлены с помощью ключевого слова `var` и указания типа данных элементов и количества элементов в массиве.

```
var numbers [5]int
```

В этом примере мы объявляем массив `numbers` типа `int` и со 5 элементами.

Мы можем обратиться к элементам массива, используя адрес, так называемый индекс, что вообще это такое? Представим, что у нас есть улица, на которой расположены дома с номерами от 0 до 4. Мы можем представить эту улицу как массив в Go. Каждый дом - это элемент массива, а номер дома - это индекс в массиве:

Обратите внимание! Самый первый элемент массива имеет **индекс 0!**

```
numbers[0] = 1
numbers[1] = 2
numbers[2] = 3
numbers[3] = 4
numbers[4] = 5
```

Мы подробнее рассмотрим циклы в следующих разделах, поскольку они являются важной концепцией в Go, которую нужно понимать, чтобы эффективно писать программы на этом языке.

Массивы - это просто коробочки, в которые можно положить разные вещи (например, числа). Но у этих коробочек есть некоторые ограничения:

1. Размер коробочки всегда постоянный. Это значит, что если коробочка может вместить только 5 вещей, мы не сможем добавить 6-ю вещь или убрать одну из них и сделать место для другой.
2. Массивы могут быть не самым лучшим решением для хранения очень много вещей, потому что мы должны заранее определить, сколько места нам понадобится в коробочке. Это может привести к тому, что место в коробочке будет использоваться неэффективно.

```
package main

import "fmt"

func main() {
    // объявление массива
    var names [5]string
```

```
    // присваивание значений элементам массива
    names[0] = "Alice"
    names[1] = "Bob"
    names[2] = "Charlie"
    names[3] = "Dave"
    names[4] = "Eve"

    // вывод элементов массива
    fmt.Println(names)
}
```

В этом примере мы создаем массив `names` типа `string` и со 5 элементами. Затем мы выводим в консоль информацию о нем.

Шпаргалка

1. Коллекция - общее обозначение структур данных одного типа.
2. Массив - это коллекция элементов неизменяемой длины, объявленная ключевым словом `var` и указанием длины в квадратных скобках (пример: var numbers [5]int)
3. Все элементы одного массива должны быть одинакового типа.
4. У каждого элемента массива есть свое место. Оно называется индексом элемента.
5. Первый элемент массива имеет индекс 0!

Домашнее задание

1. **Объявление массивов**: Создайте массив из пяти целых чисел и присвойте каждому элементу значение. Выведите каждый элемент на экран отдельно.
2. **Изменение элементов массива**: Создайте массив из трех строк. Измените значение второго элемента и выведите все элементы на экран.
3. **Длина массива**: Создайте массив с произвольным количеством элементов. Используйте функцию len для определения длины массива и выведите результат на экран.

4. **Массивы с разными типами данных**: Создайте три массива: один с целыми числами, один с числами с плавающей точкой и один со строками. Выведите все значения всех массивов.
5. **Многомерные массивы**: Создайте двумерный массив (массив массивов) и присвойте каждому элементу значение. Выведите все значения на экран.
6. **Индексы массивов**: Создайте массив и присвойте каждому элементу значение. Выведите значение элемента по индексу 0 и последнему индексу в массиве.
7. **Сравнение массивов:** Создайте два массива типа int одинаковой длины и с одинаковыми элементами, выведите результат сравнения данных массивов.
   Создайте третий массив типа int с длиной на один элемент больше и начальными элементами как и у двух первых, новый элемент задайте произвольно, попробуйте сравнить его с любым из первых двух.
8. **Изменение нескольких элементов массивы:** Создайте массив типа int с элементами 1, 2, 3, 4, измените их порядок в массиве на 4, 3, 2, 1 с помощью изменения элементов по индексу.
9. **Сохранение значения переменной в массив:** Создайте 3 переменных с разными значениями типа string, создайте массив типа string с 3 элементами, положите в массив значения переменных под разные индексы.
10. **Сохранение элемента массивы в переменную:** Создайте массив типа int с 3 элементами, также создайте 3 переменные разных типов (int, string, float64). Сохраните элементы массива в переменные с помощью конвертации. Выведите результат.

Срезы

Срез - это своего рода "умный" массив, который может становиться больше или меньше, когда программа работает. Это удобно для хранения и работы с группами данных размер которых может меняться, так как он неизвестен.

Представь, что ты на кухне и у тебя есть несколько коробочек для хранения конфет (это как массивы в Go). Каждая коробочка имеет определенное

количество мест для конфет, и это количество не изменяется. То есть, если у тебя коробочка на 5 конфет, то ты не сможешь положить туда больше 5 конфет.

Теперь представь, что у тебя есть лента, на которую ты можешь нанизывать конфеты (это как срезы в Go). Лента может быть разной длины, и ты можешь добавлять или убирать конфеты с нее, изменяя ее длину. Таким образом, срезы более гибкие и могут изменять свой размер, в то время как массивы имеют фиксированный размер.

Вот пример создания среза:

```
var numbers []int
```

В этом примере мы объявляем срез `numbers` типа `int`. В отличие от массивов, срезы не имеют фиксированной длины, поэтому нам не нужно указывать количество элементов в квадратных скобках. Мы можем сразу добавлять новые элементы в срез или удалять существующие элементы из среза.

Мы можем использовать встроенную функцию `append` для добавления новых элементов в срез:

```
package main

import (
    "fmt"
)

func main() {
    numbers := []int{1, 2, 3, 4, 5}
    numbers = append(numbers, 6)
    fmt.Println(numbers)
    numbers = append(numbers, 7, 8, 9)
    fmt.Println(numbers)
}
```

Срезы более удобны для хранения и обработки больших объемов данных, поскольку они могут изменять свою длину во время выполнения программы.

Однако они могут потреблять больше памяти, чем массивы, поскольку они должны управлять своей длиной и емкостью.

Пример работы со срезами:

```
package main

import "fmt"

func main() {
    // Создаем срез с помощью литерала среза,
    // который выглядит как массив без длины.
    fruits := []string{"apple", "banana", "cherry", "dates",
"elderberry"}

    // Получаем подсрез, используя оператор ":",
    // который указывает на диапазон индексов среза.
    // Подсрез включает элементы, начиная с первого индекса
(1),
    // и до второго элемента (3), включая его.
    sub_fruits := fruits[1:3]
    fmt.Println(sub_fruits) // [banana cherry]

    // Мы можем изменить элемент среза, присвоив новое значен
ие по индексу.
    fruits[1] = "blueberry"
    fmt.Println(fruits) // [apple blueberry cherry dates elde
rberry]

    // Для добавления элемента в конец среза используем функц
ию append().
    // Она возвращает новый срез, в который добавлен новый эл
емент.
    fruits = append(fruits, "figs")
    fmt.Println(fruits) // [apple blueberry cherry dates elde
rberry figs]
```

```
    // Для удаления элемента из среза мы создаем новый срез,
    // пропуская элемент, который хотим удалить.
    // В данном случае мы собираем все элементы
    // до индекса 3 и после индекса 4, исключая элемент с инд
ексом 3.
    fruits = append(fruits[:3], fruits[4:]...)
    fmt.Println(fruits) // [apple blueberry cherry elderberry
figs]
}
```

Срезы предоставляют более гибкий способ хранения и обработки коллекций данных, чем массивы, поскольку они могут изменять свою длину во время выполнения программы.

В этом примере мы демонстрируем работу со срезами (slices) в языке Go.

1. Создается срез `fruits` с пятью различными фруктами.
2. Получаем подсрез `sub_fruits`, который включает только "banana" и "cherry" из основного среза `fruits`.
3. Меняем второй элемент (с индексом 1) среза `fruits` с "banana" на "blueberry".
4. Добавляем "figs" в конец среза `fruits`, используя функцию `append()`.
5. Удаляем "dates" (элемент с индексом 3) из среза `fruits`, создавая новый срез без этого элемента.

Каждый этап манипуляций с срезом выводится на экран с помощью функции `fmt.Println()`.

Шпаргалка

1. Срезы - это массивы, которые могут изменять свою длину во время работы программы.
2. Как и у массива, первый элемент среза имеет индекс 0.
3. В срезе точно так же можно обратиться к любому элементу по его индексу.

4. Из-за того, что длина изменяемая - мы не указываем никакого значения в квадратных скобках при объявлении или инициализации среза. Например, объявление среза ages будет выглядеть таким образом : var ages **[]**int
5. Срезы удобнее в работе из-за своей возможности изменять свою длину, но из-за этого они могут потреблять больше памяти! Используйте срезы с осторожностью!

Домашнее задание

1. **Создание слайсов**: Объявите слайс целых чисел, добавьте в него несколько значений и выведите эти значения на экран.
2. **Изменение элементов слайса**: Создайте слайс с несколькими строками, измените одно из значений и выведите все элементы на экран.
3. **Длина и вместимость слайса**: Создайте слайс и добавьте в него несколько элементов. Используйте функции len и cap для определения длины и вместимости слайса. Выведите эти значения на экран.
4. **Создание слайса с помощью make**: Создайте слайс с помощью функции make, добавьте в него несколько элементов и выведите эти элементы на экран.
5. **Добавление элементов в слайс**: Создайте слайс, добавьте в него несколько элементов с помощью функции append и выведите все элементы на экран.
6. **Срезы слайса**: Создайте слайс с несколькими элементами, затем создайте новый слайс как срез первого слайса и выведите его на экран.
7. **Копирование слайсов**: Создайте два слайса, скопируйте элементы из одного слайса в другой с помощью функции copy и выведите оба слайса на экран.
8. **Удаление элемента из слайса**: Создайте произвольного типа слайс с 5 элементами , с помощью функции append() удалите элемент под индексом 2. Выведите результат на экран.
9. **Изменение вместимости слайса**: Создайте слайс произвольного типа с 6 элементами, узнайте его вместимость с помощью функции cap, опытным

путём определите в каком случае вместимость слайса будет 12, 24 и более.

10. **Минимальная длина и вместимость слайса:** Создайте слайс с минимально возможной длиной и вместимостью, выведите слайс, его длину и вместимость.
11. **Взаимосвязь слайсов и массивов (⭐ сложное):** Создайте массив типа int с 5 элементами 1, 2 3, 4, 5, после чего создайте слайс на основе первых трёх элементов созданного массива с помощью оператора слайса, добавьте к нему элемент 9 с помощью функции append(), после чего создайте новый слайс на основе первого и добавьте к нему два элемента 8 и 7, выведите массив и оба слайса на экран. Попробуйте объяснить полученный результат.

Мапы

Мапы - это коллекции данных, которые используются для хранения пар "ключ-значение".

Представь, что у тебя есть большой ящик с множеством отделений, и на каждом отделении есть ярлык с именем. Этот ящик используется для хранения разных вещей, и чтобы найти нужную вещь, ты просто ищешь ярлык с именем этой вещи. Мапы - это как такой ящик. Они позволяют хранить информацию (значения) и связывать их с ярлыками (ключами). В Go, мапы создаются с помощью ключевого слова `map`, и они могут хранить разные типы значений, например числа, строки и другие структуры данных.

Все ключи в одной мапе должны иметь один общий тип, как и значения.

А вот между собой ключи и значения могут отличаться по типу.

Ключи и значения могут быть любого типа данных, но обычно они являются строками или числами.

```
graph TD

subgraph map
    key1 --> value1
```

```
    key2 --> value2
    key3 --> value3
    key4 --> value4
end
```

В Go, мапы могут быть объявлены с помощью ключевого слова `map` и указания типа данных ключей и значений.

```
ages := make(map[string]int)
```

В этом примере мы объявляем мапу `ages`, которая будет хранить пары "имя-возраст". Ключи мапы - это строки, а значения - это целые числа.

Мы можем добавлять новые элементы в мапу, используя ключ:

```
ages["Alice"] = 25
ages["Bob"] = 30
ages["Charlie"] = 35
```

Мапы очень удобны для хранения и обработки больших объемов данных. Однако они могут быть менее эффективными, чем массивы или срезы, если используются неправильно или при работе с большими объемами данных.

Пример работы с мапами:

```
package main

import "fmt"

func main() {

    // Инициализация map под названием ages,
    // который будет хранить имена (строки) и возраста (целые
числа).
    // Это делается с использованием литерала map,
    // который позволяет нам определить и инициализировать ma
p одновременно.
```

```
    ages := map[string]int{
        "Alice":   25,
        "Bob":     30,
        "Charlie": 35,
        "Dave":    40,
        "Eve":     45,
    }

    // Добавление нового значения в map.
    // В данном случае мы добавляем "Frank" с возрастом 50.
    // Это достигается путем присвоения значения ключу "Fran
k" в map ages.
    ages["Frank"] = 50

    // Печать возраста Alice.
    // Мы используем функцию Printf из пакета fmt,
    // чтобы напечатать форматированную строку.
    // В этой строке %s будет заменено на имя
    // (Alice), а %d - на ее возраст (25).
    // Подробнее про эту конструкцию будет ниже
    fmt.Printf("%s is %d years old\n", ages["Alice"])

    // Аналогично, печать возраста Bob.
    fmt.Printf("Bob is %d years old\n", ages["Bob"])

        // Удаляем значение из map по ключу "Alice"
    delete(ages, "Alice")
}
```

Мапы предоставляют удобный способ хранения и обработки пар "ключ-значение" в программе, и они могут использоваться для хранения и обработки различных типов данных. Однако они должны использоваться осторожно, чтобы избежать проблем с производительностью и памятью.

Подробнее про плейсхолдеры

Плейсхолдеры в Go используются в функциях форматирования строк, таких как `Printf`, `Sprintf`, `Fprintf` и других, представленных в пакете `fmt`. Плейсхолдеры представляют собой специальные символы, которые заменяются на соответствующие значения при выводе.

Например, в функции `Printf`:

```
fmt.Printf("У %s есть %d яблок.\\n", "Алисы", 7)
```

В этой строке `%s` и `%d` являются плейсхолдерами. `%s` заменяется на строку "Алисы", а `%d` - на число 7. Результатом будет строка "У Алисы есть 7 яблок."

Существует много различных плейсхолдеров для разных типов данных и форматов вывода. Например:

- `%d` для целых чисел
- `%f` для чисел с плавающей точкой
- `%s` для строк
- `%v` для любого значения
- `%%` для вывода символа процента

Шпаргалка

1. Мапы - коллекции, которые хранят данные по образцу “ключ-значение”.
2. Все ключи одной мапы должны иметь один тип данные, как и значения. Но сам тип ключей и значений может быть разным.
3. В мапе можно напрямую обратить к любому значению, использовав его ключ.

Домашнее задание

1. **Создание map**: Создайте map, где ключом является строка, а значением - целое число. Добавьте в нее несколько пар ключ-значение и выведите их на экран.

2. **Изменение значений в map**: Создайте map с несколькими парами ключ-значение. Измените одно из значений и выведите всю map на экран.
3. **Получение значения по ключу из map**: Создайте map, добавьте в нее несколько пар ключ-значение. Затем получите и выведите на экран значение по одному из ключей.
4. **Удаление значения из map**: Создайте map, добавьте в нее несколько пар ключ-значение. Затем удалите одну из пар с помощью функции delete и выведите всю map на экран.
5. **Проверка существования ключа в map**: Создайте map, добавьте в нее несколько пар ключ-значение. Затем проверьте существование одного из ключей с помощью двухзначной формы индексирования и выведите результат на экран.
6. **Проход по всем значениям map**: Создайте map и добавьте в нее несколько пар ключ-значение. Затем напечатайте все пары ключ-значение. Для этого вам потребуется использовать цикл range (я знаю, что вы еще не изучали циклы, но это хорошая возможность начать).
7. **Массив как ключ**: Создайте два массива одного типа с одинаковой длиной. Создайте map, в которой ключом являются созданные массивы, тип значений задайте произвольно, попробуйте наполнить созданную map разными данными.
   Попробуйте использовать слайс как ключ в мапе, объясните итог.
8. **Слайс как значение:** Создайте map, где значением будут слайс типа int. Создайте несколько пар ключ-значение в данной map. Выведите результат.
9. **Map как значение map:** Создайте map, в которой ключом будет string, а значением map. Map-значение в себе имеет ключ int, значение string. В первой map (с ключом string) ключом будет название улицы, во второй map (которая является значением первой) - ключом будет номер дома, а значением фамилия семьи живущей в нём. Создайте несколько пар ключ-значение. Выведите результат.