Структуры шпаргалка

Структуры в Go - это типы данных, которые могут содержать поля с различными типами данных. Они позволяют определить пользовательский тип данных, который может использоваться в программе.

Объявление структуры

Структура объявляется с помощью ключевого слова `type`, за которым следует имя структуры и список ее полей в фигурных скобках. Каждое поле состоит из имени и типа данных.

```
type Person struct {
    Name string
    Age int
}
```

Создание экземпляра структуры

Экземпляр структуры создается с помощью ключевого слова `var`, за которым следует имя переменной и вызов конструктора структуры. Конструктор структуры возвращает указатель на созданный экземпляр.

```
var john Person
john.Name = "John"
john.Age = 30
```

Так же можно создать и заполнить в формате быстрого объявления:

```
john := Person{Name:"John",Age:30}
```

Вложенные структуры

Структуры могут содержать в себе другие структуры в качестве полей. Для доступа к полям вложенной структуры используется имя родительской структуры и имя поля вложенной структуры через точку.

```
type Address struct {
    Street string
```

```
    City string
}

type Person struct {
    Name string
    Age int
    Address Address
}

var john Person
john.Name = "John"
john.Age = 30
john.Address.Street = "123 Main St"
john.Address.City = "New York"
```

Анонимные структуры

Анонимные структуры - это структуры без имени (не используется объявление type), которые могут использоваться для временного хранения данных в рамках функции.

```
var person struct {
    Name string
    Age int
}

person.Name = "John"
person.Age = 30
```

Методы структур в Go

Методы в Go - это функции, которые определены внутри структуры и могут использоваться для манипулирования полями структуры. Методы могут быть определены для любого типа данных, включая пользовательские структуры.

Объявление методов

Методы объявляются с помощью ключевого слова `func`, за которым следует имя метода, список параметров, тип значения, который возвращает метод, а также блок кода, который выполняется при вызове метода.

```
func (p Person) GetAge() int {
    return p.Age
}
```

В данном примере метод `GetAge` определен для структуры `Person`. Он возвращает возраст человека, который определяется полем `Age` структуры `Person`.

Вызов методов

Методы вызываются на экземпляре структуры с помощью оператора точки `.` и имени метода.

```
var john Person
john.Age = 30
age := john.GetAge()
```

В данном примере метод `GetAge` вызывается на экземпляре структуры `john`, который является экземпляром структуры `Person`.

Встроенные методы

В Go есть несколько встроенных методов, которые определены для структур, таких как методы `len` и `cap` для срезов и методы `append` и `copy` для срезов и массивов. Эти методы могут использоваться без явного вызова.

Интерфейсы

Интерфейсы в Go позволяют определить набор методов, которые должны быть реализованы для конкретного типа данных.

Объявление интерфейсов

Интерфейсы объявляются с помощью ключевого слова `type`, за которым следует имя интерфейса и список его методов. Каждый метод определяется с помощью имени, списка параметров и типа возвращаемого значения.

```
type Writer interface {
    Write(p []byte) (n int, err error)
}
```

В данном примере определяется интерфейс `Writer`, который содержит метод `Write`. Метод `Write` принимает срез байтов и возвращает количество записанных байтов и ошибку.

Реализация интерфейсов

Для того чтобы тип данных реализовывал интерфейс, ему необходимо определить все методы, которые определены в интерфейсе. Для этого тип данных должен определить функцию с тем же именем, списком параметров и типом возвращаемого значения, что и методы в интерфейсе.

```
type MyWriter struct{}

func (w MyWriter) Write(p []byte) (n int, err error) {
    // implementation here
}
```

Использование интерфейсов

Интерфейсы могут использоваться для создания абстрактных типов данных, которые могут быть использованы в качестве общих интерфейсов для различных реализаций. Например, интерфейс `Writer` может использоваться для записи данных в различные источники, такие как файлы, сокеты или буферы.

```
func writeTo(writer Writer, data []byte) error {
    _, err := writer.Write(data)
    return err
}
```

В данном примере определяется функция `writeTo`, которая принимает интерфейс `Writer` и срез байтов. Функция вызывает метод `Write` на интерфейсе `Writer`, что позволяет записать данные в любой тип данных, который реализует этот метод.

nil интерфейсы

`nil` интерфейс - это интерфейс без конкретного значения. Он может быть использован для представления любого типа данных, но не содержит конкретного значения.

```
var w interface{}
```

В данном примере объявляется переменная типа **interface{}**, но ей не присваивается конкретное значение. Таким образом, `w` является `nil` интерфейсом.

Type assertion

Type assertion используется для проверки типа значения, которое хранится в интерфейсе, и получения его конкретного значения.

```
value, ok := x.(T) // T - какой-то тип, в данном контексте не важно какой

// Пример
s, ok := x.(string)
if ok {
    fmt.Println(s)
} else {
    fmt.Println("not a string")
}
```

В данном примере `i` - это интерфейс, содержащий значение типа `T`. `value` - это конкретное значение типа `T`, полученное из интерфейса. `ok` - это логическое значение, которое указывает, было ли получено значение типа `T` из интерфейса.

Интерфейсы и структуры

Интерфейс	Структура
Определяет набор методов, которые должны быть реализованы	Определяет поля и методы, которые могут быть использованы для работы с данными
Используется для создания абстрактных типов данных	Используется для определения типов данных с конкретными значениями
Может иметь несколько реализаций	Имеет только одну конкретную реализацию
Используется для работы с различными типами данных через общий интерфейс	Используется для работы с конкретным типом данных