GO-10 05: Cgo

Описание:

Go проектировался как простой язык программирования и по умолчанию содержит самый необходимый минимум функциональных возможностей. Однако предусмотрены инструменты, которые значительно его расширяют. Например, с помощью cgo в программах Go можно использовать код, написанный на C. Для работы с кодом C в системе должен быть установлен компилятор gcc.
Проверим наличие компилятора gcc в системе:

```
gcc --version
```

Если не удалось выполнить команду, то необходимо установить gcc:

```
sudo apt update
sudo apt install build-essential
```

1. Использование фрагмента кода, написанного на C, в Go

Чтобы использовать C-код, необходимо импортировать псевдопакет "C". Псевдопакет "C" отличается от обычных пакетов Go тем, что он целиком реализуется компилятором. Код на C пишется перед импортом в комментариях и его использует cgo. Обратите внимание, что между комментарием cgo и оператором импорта не должно быть пустых строк. Для начала мы хотим сложить 2 целых числа с использованием C-кода:

```
// // C - код
// #include <stdio.h>
// void add(int a, int b) {
//     printf("%d\n", a + b);
// }
import "C"

import "fmt"

// Add 2 numbers
func Add(a, b int) {
    fmt.Printf("%d + %d = ", a, b)
    C.add(C.int(a), C.int(b))
}
```

При использовании C в коде можно выделить 2 части: все, что идет вначале в комментариях, и строка импорта C, относится к cgo, а остальное - к Go. Обратите внимание, что перед вызовом C-функции в Go используется обращение к псевдопакету C, а переменные в виде аргументов также приводятся к формату C.

2. Использование библиотеки, написанной на C в Go

Немного усложним наш пример. Пусть теперь функция сложения у нас реализована в библиотеке C, и у нас есть сама библиотека и ее заголовок. Для начала давайте ее создадим.

```
// addition.h
void add(int a, int b);

// addition.c
#include <stdio.h>
#include "addition.h"

void add(int a, int b) {
    printf("%d\n", a + b);
}
```

Скомпилируем библиотеку:

```
gcc -c addition.c
gcc -shared -o libaddition.so addition.o
```

А теперь в файле addition.go изменим комментарий для cgo следующим образом:

```
// // C - код
// #cgo CFLAGS: -I.
// #cgo LDFLAGS: -L. -laddition
// #include "addition.h"
import "C"
```

И этого достаточно для использования библиотеки addition.so в Go.

3. Использование Go в программах, написанных на C

Для того чтобы экспортировать функцию, написанную на Go с C-код, необходимо ее пометить для cgo специальным комментарием export. Например, мы хотим, чтобы наша функцию складывала 2 числа и выводила результат в stdout.

```
//export Add2Numbers
func Add2Numbers(a C.int, b C.int) {
```

```
    fmt.Println(a + b)
}
```

При компиляции cgo с помощью этого комментария формирует Header-файл _cgo_export.h с описанием функций, помеченных директивой export.
Создадим C-файл со следующим содержанием - создаем поток, в котором будем складывать 2 числа цикла, сложение реализует функция, написанная на Go (callback-функция):

```
// addition.c
#include "_cgo_export.h"
#include <pthread.h>

void *myThreadFun(void *vargp){
    int a, b;
    for (size_t i = 0; i < 5; i++){
        Add2Numbers(a++, ++b);
    }
}

void adds() {
    pthread_t thread_id;
    printf("Before Thread\n");
    pthread_create(&thread_id, NULL, myThreadFun, NULL);
    pthread_join(thread_id, NULL);
    printf("After Thread\n");
    return;
}
```

Внесем правки в файл Go, теперь он у нас должен выглядеть следующим образом:

```
/*
// C - код
#cgo LDFLAGS: -lpthread
#include <stdio.h>
extern void adds();
*/
import "C"

// Go-код
import "fmt"
```

```
//export Add2Numbers
func Add2Numbers(a C.int, b C.int) {
    fmt.Println(a + b)
}

// Add 2 numbers
func Add(a, b int) {
    C.adds()
}
```

В Go-коде возможно использовать и многострочные комментарии для cgo с помощью /* ... */.
Основные недостатки использования cgo:

1. Поскольку мы используем компилятор gcc, соответственно, теряем кроссплатформенность сборки. Все проекты, которые импортируют псевдопакет "C", должны быть перекомпилированы в той среде, в которой используются.
2. Код будет перекомпилироваться каждый раз при сборке, поскольку компилятор не знает, какие файлы "C" были модифицированы.
3. При вызове C-функций происходит переключение runtime между go и cgo и, как следствие, будет снижение производительности.

Полезные ссылки:

- Command cgo
- cgo by Harrison Thorne
- C-вызовы в Go: принцип работы и производительность

Задание:

1. Создайте в группе с вашими репозиториями - golang_users_repos/<your_gitlab_id> проект module10_05 и ветку module10_05 в нем.
2. Создайте в пакете internal файл main.go.
3. Используя первый пример из описания, реализуйте функцию C void myprint (char * s) в Go-коде. Функция myprint выводит стандартный поток - сообщение "Hello from Go". Обратите внимание, что Go-строки и C-строки различаются. В Go необходимо сначала выделить память под C-строку, а потом ее освободить.
4. Зафиксируйте изменения в ветке и отправьте их в удаленный репозиторий проекта.
5. В качестве ответа пришлите ссылку на merge request в ветку master вашего проекта ветки module10_05.