Урок Спринт 2.3

Дамп горутин

Аннотация

Приветствуем вас на уроке по тулингу в Go. Поговорим сегодня про то, как посмотреть, что у нас происходит с горутинами, где у нас происходит гонка и как построчно смотреть код.

Дамп горутин — это инструмент в Go, который позволяет увидеть текущее состояние всех горутин в программе и их стеки вызовов. Это полезно для отладки и выявления проблем с синхронизацией и параллелизмом.

Дамп горутин может быть вызван в случае, когда вы обнаруживаете странные или непредсказуемые события в вашей программе, которые могут быть связаны с горутинами. Он помогает вам понять, какие горутины активны в данный момент, и как они взаимодействуют друг с другом.

Вот пример кода, который можно использовать для получения дампа горутин в Go:

```
package main

import (
    "fmt"
    "os"
    "runtime/pprof"
    "time"
)

func main() {
    // Создание файла для записи дампа горутин
    f, err := os.Create("goroutine_dump.txt")
    if err != nil {
        fmt.Println("Не удалось создать файл:", err)
```

```
        return
    }
    defer f.Close()

    // Запись дампа горутин в файл каждую секунду
    go func() {
        for {
            // Получение дампа горутин
            pprof.Lookup("goroutine").WriteTo(f, 1)
            time.Sleep(time.Second)
        }
    }()

    // Ваш код, в котором создаются горутины
    // ...

    // Пример бесконечного цикла для демонстрации
    for {
        // Здесь может быть ваша основная логика программы
        time.Sleep(1 * time.Second)
    }
}
```

Этот код создаёт файл "goroutine_dump.txt" и записывает в него дамп горутин каждую секунду.
Вы можете добавить свою логику программы внутри этого примера. Чтобы получить дамп прервите программу и у вас появится файл goroutine_dump.txt.

Таким образом выглядит дамп

```
goroutine profile: total 2
1 @ 0x10023f3fc 0x10026f9e4 0x1002b6904 0x1002b6720 0x1002b38a8 0x1002c0314 0x1002749c4
#       0x10026f9e3     runtime/pprof.runtime_goroutineProfileWithLabels+0x23   /usr/local/go/sr
#       0x1002b6903     runtime/pprof.writeRuntimeProfile+0xb3                  /usr/local/go/sr
#       0x1002b671f     runtime/pprof.writeGoroutine+0x4f                       /usr/local/go/sr
#       0x1002b38a7     runtime/pprof.(*Profile).WriteTo+0x147                  /usr/local/go/sr
#       0x1002c0313     main.main.func1+0x43                                    /Users/yurasargs

1 @ 0x100248ec8 0x10027178c 0x1002c02a4 0x100248a6c 0x1002749c4
#       0x10027178b     time.Sleep+0x10b        /usr/local/go/src/runtime/time.go:195
#       0x1002c02a3     main.main+0x113         /Users/yurasargsyan/GolandProjects/awesomeProje
#       0x100248a6b     runtime.main+0x2bb      /usr/local/go/src/runtime/proc.go:267
```

1. **Сведения о горутинах:**

   - Общее количество горутин: 2.
   - Каждая горутина имеет свой уникальный идентификатор и стек вызовов.

2. **Анализ первой горутины:**

   - Горутина 1 запущена из `main.main.func1` в файле `/Users/yurasargsyan/GolandProjects/awesomeProject16/go-concurrency-exercises/main.go` на строке 23.

- Стек вызовов указывает на функции в стандартной библиотеке Go, такие как `runtime/pprof.writeGoroutine, runtime/pprof.writeRuntimeProfile` и т.д.

3. **Анализ второй горутины:**

- Горутина 2 находится во время ожидания с помощью функции `time.Sleep`. Это может быть ожидание в основной программе в `main.main` на строке 34.