Урок Спринт 2.6

WaitGroup

Настала пора раскрыть карты. Всё, что мы написали руками, уже есть в стандартной библиотеке. Этот функционал нам предоставляет объект **WaitGroup**. **WaitGroup** представляет собой счётчик горутин, выполнения которых нужно дождаться.

Для того, чтобы воспользоваться этим типом, давайте сначала создадим его экземпляр.

```
// создаем экземпляр WaitGroup
wg := &sync.WaitGroup{}
```

`WaitGroup` предоставляет 3 проассациированные функции:

- **Wait** — программа блокируется до того момента, пока значение счётчика положительное
- **Add** — увеличивает количество счётчика на n (если передать отрицательное число — то счётчик уменьшается)
- **Done** — уменьшает счётчик на 1 (тождественно вызову `Add(-1)` — так эта функция и имплементирована в стандартной библиотеке)

Давайте перед вызовом каждой нашей горутины увеличим счётчик на 1 (хотим дождаться выполнения еще одной горутины)

```
for i := 0; i < size; i++ {
    // Добавляем в группу 1 элемент
    wg.Add(1)
    go func() {
        ...
}
```

Обратите внимание — мы знаем, что хотим дождаться выполнения `size` горутин. Поэтому можем заменить это вызовом `wg.Add(size)`

```
wg.Add(size)
for i := 0; i < size; i++ {
    // Добавляем в группу 1 элемент
```

```
    go func() {
        ...
}
```

Добавим ожидание выполнения всех горутин группы.

```
wg.Add(size)
for i := 0; i < size; i++ {
    go func() {
        ...
}
// Ждем выполнения всех горутин группы (пока счетчик не станет равным 0)
wg.Wait()
```

После выполнения горутины нужно не забыть декрементировать счётчик.

```
go func() {
    // Удаляем один элемент из группы
    defer wg.Done()
    safeSlice.Append(random())
}()
```

Обратите внимание, что к моменту, когда мы вызываем `wg.Wait()`, счётчик должен быть инкрементирован. Частая ошибка:

```
for i := 0; i < size; i++ {
    go func() {
        // мы не можем быть уверены, что эта функция вызовется до wg.Wait()
        wg.Add(size)
        ...
}
wg.Wait()
```

В итоге получим следующий код:

```
package main

import (
    "fmt"
    "math/rand"
    "sync"
)

type SafeSlice struct {
    results []int
    mx      *sync.Mutex
}

func New() *SafeSlice {
    return &SafeSlice{
        mx:      &sync.Mutex{},
```

```
        results: []int{},
    }
}

// Добавляем к слайсу элемент item
func (s *SafeSlice) Append(item int) {
    // Вызван Lock, поэтому только одна горутина за раз может получить доступ к слайсу
    s.mx.Lock()
    defer s.mx.Unlock()
    s.results = append(s.results, random())
}

// Получаем элемент слайсу по индексу
func (s *SafeSlice) Get(index int) int {
    // Вызван Lock, поэтому только одна горутина за раз может получить доступ к слайсу
    s.mx.Lock()
    defer s.mx.Unlock()
    return s.results[index]
}

// Функция генерирует случайное число в интервале [0, 100)
func random() int {
    const max int = 100
    return rand.Intn(max)
}

func main() {
    safeSlice := New()
    const size int = 10
    // создаем экземпляр WaitGroup
    wg := &sync.WaitGroup{}

    // Заполняем слайс случайными числами
    for i := 0; i < size; i++ {
        // Добавляем в группу 1 элемент
        wg.Add(1)
        go func() {
            // Удаляем один элемент из группы
            defer wg.Done()
            safeSlice.Append(random())
        }()
    }
    // Ждем выполнения всех горутин группы
    wg.Wait()

    // Поэлементно печатаем слайс на экран
    for i := 0; i < size; i++ {
        fmt.Println(safeSlice.Get(i))
    }
}
```

Это простой поддерживаемый код, который реализован с помощью стандартного инструмента.

При работе с `WaitGroup` используется следующий алгоритм:

- создать экземпляр `var wg WaitGroup`
- добавить количество горутин, которые будут выполняться `wg.Add(<количество горутин>)`
- запустить горутины, сопровождая их `wg.Done()`
- дождаться завершения `wg.Wait()`

Резюмируя работу с `WaitGroup`, нужно еще раз обратить внимание на:

- `WaitGroup` нельзя копировать после первого использования
- `Add` добавляет `delta`, которая может быть отрицательной, к счётчику `WaitGroup`.
- Если счётчик обнуляется, все горутины, заблокированные `Wait`, освобождаются.
- Если счётчик становится отрицательным, `Add` вызывает `panic`.
- `Done` уменьшает счётчик `WaitGroup` на один.
- Все вызовы `Add`, которые происходят при счётчике равным 0 должны происходить до `Wait`.