



## Урок Спринт 3.2

## Практические аспекты тестирования конкурентного кода

Давайте рассмотрим несколько практических аспектов тестирования конкурентного кода:

### Использование `t.Parallel()`

В Go, тесты могут быть запущены параллельно для проверки работы программы в параллельном режиме и для ускорения их выполнения, если тесты не связанные. Если ваша программа или часть программы, которую вы тестируете не зависит от незащищенных глобальных переменных или ресурсов, добавьте `t.Parallel()` в начало теста, чтобы разрешить его параллельное выполнение.

Вы можете встретиться с требованием использования `t.Parallel()` во всех тестах в некоторых компаниях.

Пример:

```
func TestIncrementParallel(t *testing.T) {
    t.Parallel()

    // тело теста...
}
```

### Использование `sync.WaitGroup` для ожидания завершения горутин

В тестах, где запускаются горутин, используйте `sync.WaitGroup`, чтобы дождаться их завершения перед завершением теста.

Пример:

```
func TestConcurrentProcessing(t *testing.T) {
    var wg sync.WaitGroup

    // ... инициализация ...

    // Запуск горутин
    for i := 0; i < numGoroutines; i++ {
        wg.Add(1)
        go func(i int) {
            defer wg.Done()
            // логика горутин...
        }(i)
    }

    // Ожидание завершения всех горутин
    wg.Wait()

    // ... проверки результатов ...
}
```

Ниже мы используем этот подход в примерах.

Эти практические аспекты помогут вам создавать надёжные и эффективные тесты для конкурентного кода в Go.

Справка

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».

[Пользовательское соглашение.](#)

© 2018 – 2024 ООО «Яндекс»