Русский

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 паралельном 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 горутины, 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 горутины...
}(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