Урок Спринт 2.4

Отмена контекста

Представим, что у нас есть функция, получающая данные из какого-либо источника:

```
func readSource(ctx context.Context) error {
    // имитируем долгую работу функции
    time.Sleep(3 * time.Second)
    // допустим возникла ошибка в процессе
    return fmt.Errorf("some error in readSource")
}
```

и другая функция, которая эти данные обрабатывает:

```go
func processSourceData(ctx context.Context) error {
    // получаем данные в цикле
    for {
        select {
        // раз в секунду получим новые данные
        case <-time.After(time.Second):
            // здесь может быть код получения очередной порции данных
            fmt.Println("process data bit by bit...")
        // проверим контекст на отмену
        case <-ctx.Done():
            fmt.Println("processSourceData was canceled")
            return nil
        }
    }
}
```

Попробуем всё собрать вместе:

```
func main(){
    ctx := context.Background()
    // для ожидания завершения горутин
    wg := sync.WaitGroup{}
    wg.Add(2)
    go func() {
```

```
        defer wg.Done()
        // запустим функцию обработки данных
        if err := processSourceData(ctx); err != nil {
            fmt.Printf("processSourceData(ctx): %s", err)
        }
    }()
    go func() {
        defer wg.Done()
        // запустим функцию чтения данных
        if err := readSource(ctx); err != nil {
            fmt.Printf("readSource(ctx): %s", err)
        }
    }()
    // ждем завершения
    wg.Wait()
}
```

Если вы запустите код, то у вас получится примерно следующий вывод:

```
process data bit by bit...
process data bit by bit...
process data bit by bit...
readSource(ctx): some error in readSource
process data bit by bit...
process data bit by bit...
...
```

То есть после возникновения ошибки в функции `readSource` функция `processSourceData` продолжит ожидать поступления новых данных и никогда не завершится. Давайте попробуем это исправить и создадим контекст, который можно отменить:

```
ctxWithCancel, cancelCtx := context.WithCancel(ctx)
```

Внесём изменения в функцию `main`:

```
    // контекст, который можно отменить
    ctx := context.Background()
    ctxWithCancel, cancelCtx := context.WithCancel(ctx)
    defer cancelCtx()
    wg := sync.WaitGroup{}
    wg.Add(2)
    go func() {
        defer wg.Done()
        if err := processSourceData(ctxWithCancel); err != nil {
            fmt.Printf("processSourceData(ctxWithCancel): %s", err)
        }
    }()
    go func() {
        defer wg.Done()
        if err := readSource(ctxWithCancel); err != nil {
            // при ошибке в функции чтения - подадим сигнал через контекст
            cancelCtx()
```

```
            fmt.Printf("readSource(ctxWithCancel): %s", err)
        }
    }()
    wg.Wait()
    return nil
```

Теперь после вызова `cancelCtx()` функция `processSourceData` завершит выполнение, и программа завершится.