Урок Спринт 1.5

Context

`context.Context` в Go управляет жизненным циклом запроса и хранит информацию о нём — таймаут, отмену и т. д. Пока запрос обрабатывается, мы можем создать новый контекст и передать его в функции, которые подключатся к обработке.

С помощью контекста мы можем отменять чересчур времязатратные запросы. Это сэкономит ресурсы и повысит производительность приложения.

Контекст позволяет нам передавать значения между функциями без глобальных переменных и длинных списков параметров функции. Это делает код чище и понятнее.

Context API overview

В API пакета context в Go есть функции и типы данных, которые позволяют управлять контекстом запроса. Вот они:

1. `context.Background()` — создаёт пустой базовый контекст для всех запросов.
2. `context.WithTimeout(parent, timeout)` — создаёт новый (дочерний) контекст на основе родительского и времени ожидания. Если запрос не завершится в указанное время (например 2s, 100ms, 1h), контекст будет отменён.
3. `context.WithValue(parent, key, value)` — создаёт новый контекст на основе родительского и добавляет значение, которое связано с указанным ключом.
4. `context.Value(key)` — возвращает значение, которое связано с указанным ключом.

Запросы с контекстом

Чтобы отправить HTTP-запрос с таймаутом в Go, можно использовать пакет `net/http` и контексты. Они передают значения и устанавливают сроки ожидания для операций.

Вот пример кода:

```
package main
```

```go
import (
    "context"
    "fmt"
    "net/http"
    "time"
)

func main() {
    // создаём контекст с таймаутом в 5 секунд
    ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
    defer cancel()

    // создаём клиент
    client := &http.Client{}

    // создаём запрос
    req, err := http.NewRequest("GET", "http://ya.ru", nil)
    if err != nil {
        fmt.Println(err)
        return
    }

    // отправляем запрос с контекстом
    resp, err := client.Do(req.WithContext(ctx))
    if err != nil {
        fmt.Println(err)
        return
    }
    // в этом примере мы не собираемся читать тело ответа
    defer resp.Body.Close()

    // обрабатываем ответ
    fmt.Println(resp.StatusCode)
}
```

Здесь мы создаём контекст с таймаутом в 5 секунд и передаем его в метод `WithContext` запроса. Если запрос не будет выполнен за это время, контекст отменится и возвратится ошибка `context.DeadlineExceeded`.

Мы можем немедленно отменить запрос, если контекст будет отменён. Это особенно полезно, если отправки запросов проходят долго.

Далее

Справка