Урок Спринт 2.4

Передача значения в контексте

При обработке запросов (например, на сервере) обычно требуется передавать какие-либо данные, специфичные для этого запроса, ниже по стеку. Например:

```
func ProcessRequest(userID string) {
    // сохраним значение в контексте
    ctx := context.WithValue(context.Background(), "userID", userID)
    // функция обработки
    HandleResponse(ctx)
}
// здесь контекст уже содержит userID
func HandleResponse(ctx context.Context) {
    fmt.Printf("handling response for (%v)", ctx.Value("userID"))
}
```

Важно отметить, что значения, хранящиеся в определённом контексте, изменить нельзя (`immutable`). Когда мы вызываем `context.WithValue`, мы передаём родительский контекст и получаем контекст обратно. Мы получаем контекст обратно, потому что функция `context.WithValue` не изменила предоставленный нами контекст. Вместо этого она завернула родительский контекст в другой с новым значением. Рассмотрим пример:

```
// сохраним значение myValue по ключу myKey
ctx = context.WithValue(ctx, "myKey", "myValue")
fmt.Printf("my value is %s\n", ctx.Value("myKey")) // my value is myValue
// сохраним значение anotherValue по ключу myKey
anotherCtx := context.WithValue(ctx, "myKey", "anotherValue")
fmt.Printf("my value is %s\n", anotherCtx.Value("myKey")) // my value is anotherValue
```

Когда мы используем метод `Value`, он находит самое 'внешнее' значение (из последнего контекста) для данного ключа и возвращает это значение.

В простых программах допускается сохранять значения стандартных типов в контексте, однако это может привести к коллизиям. Давайте посмотрим на следующий код:

```
func HandleResponse(ctx context.Context) { // допустим переданный контекст содержит значение по
    newUserID := "22"
    ctx = context.WithValue(ctx, "userID", newUserID)  // запишем значение по ключу userID
    fmt.Printf("handling response for (%v)", ctx.Value("userID"))  // handling response for 22
}
```

В рамках одного пакета такую ошибку можно избежать, но что, если вы используете пакеты сторонних разработчиков? Чтобы избежать таких коллизий в реальных системах, рекомендуется создавать отдельный тип данных для сохранения в контексте:

```
// создадим тип данных для хранения id пользоваталя
type userID string
func ProcessRequest(id userID) {
    // сохраним значение в контексте
    ctx := context.WithValue(context.Background(), "userID", id)
}
```

При получении значения, достаточно проверить тип сохранённого объекта, чтобы убедиться что он сохранён в рамках вашего пакета:

```
id, ok := ctx.Value("userID").(userID)
if !ok{
    // другой тип объекта
}
```

При сохранении значения в контекст всегда помните, что это значение может быть использовано в различных горутинах.

Справка