Урок Спринт 1.13

Пример: HTTP сервер с логгером

А теперь давайте напишем http-сервер и сделаем `middleware` для логирования запросов и ответов.

Установим зависимости:

```
go get github.com/gorilla/mux
go get go.uber.org/zap
```

```
package main

import (
    "fmt"
    "net/http"
    "time"

    "github.com/gorilla/mux"  // Маршрутизатор HTTP
    "go.uber.org/zap"         // Запись логов
    "go.uber.org/zap/zapcore" // Настройки логгера
)

func main() {
    // Создаём новый маршрутизатор Gorilla Mux
    r := mux.NewRouter()

    // Создаём логгер с настройками для production
    logger := setupLogger()

    // Добавляем middleware для логирования
    r.Use(loggingMiddleware(logger))

    // Определяем маршруты
    r.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Привет, мир!"))
    })

    // Используем маршрутизатор с логгером
    http.Handle("/", r)

    // Запускаем HTTP-сервер на порту 8080
    port := 8080
    logger.Info("Сервер запущен", zap.Int("port", port))
    http.ListenAndServe(fmt.Sprintf(":%d", port), r)
}

func setupLogger() *zap.Logger {
    // Настраиваем конфигурацию логгера
    config := zap.NewProductionConfig()

    // Уровень логирования
    config.Level = zap.NewAtomicLevelAt(zapcore.InfoLevel)

    // Настраиваем логгер с конфигом
    logger, err := config.Build()
    if err != nil {
        fmt.Printf("Ошибка настройки логгера: %v\n", err)
    }

    return logger
}
```

```
func loggingMiddleware(logger *zap.Logger) mux.MiddlewareFunc {
    // Middleware для логирования запросов и ответов
    return func(next http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
            start := time.Now()

            // Пропускаем запрос к следующему обработчику
            next.ServeHTTP(w, r)

            // Завершаем логирование после того, как запрос выполнен
            duration := time.Since(start)
            logger.Info("HTTP запрос",
                zap.String("method", r.Method),
                zap.String("path", r.URL.Path),
                zap.Duration("duration", duration),
            )
        })
    }
}
```

Здесь мы создали `middleware` `loggingMiddleware`, которое регистрирует информацию о каждом HTTP-запросе и ответе, в том числе метод, путь и длительность выполнения запроса. Middleware применяется ко всем маршрутам, которые добавлены к маршрутизатору, через `r.Use(loggingMiddleware(logger))`.

Итак, логирование жизненно необходимо для того, чтобы отслеживать и исправлять ошибки работы программы. Теперь вы знаете, как его настроить!