GO

Logging in Go with slog

Author
Gurleen Sethi on 18 December 2022

What is slog?

slog is an experimental logging package from the Go team that provides the functionality of structured logging.

Note: At the time for writing this article, the package is still being developed separately from the Go core.

This article gives you an overview of logging functionality in this package.

Installation

Table of Contents

Create a new go project or use an existing and install `slog` .

```
go get golang.org/x/exp/slog
```

Using the logger

Import and start using the logger right away.

main.go

```
package main

import (
    "golang.org/x/exp/slog"
)

func main() {
    slog.Info("Go is best language!")
}
```

Output:

```
$ go run main.go
2022/12/15 01:31:23 INFO Go is best language!
```

By default the output includes time, log level and message.

The following log leves are available.

```
Debug
Info
Warn
Error
```

Structured loggir

≡ Table of Contents

`slog` is a structured logger that supports logging in two formats: **text** and **json**.

Let's take a look at text logger.

Text Handler

You start off by creating a text handler and a new logger.

main.go

```
package main

import (
    "os"

    "golang.org/x/exp/slog"
)

func main() {
    textHandler := slog.NewTextHandler(os.Stdout)
    logger := slog.New(textHandler)

    logger.Info("Go is the best language!")
}
```

Output:

```
$ go run main.go
time=2022-12-15T01:41:25.277-05:00 level=INFO msg="Go is the best lang
```

Pay close attention, you will see the output is formatted as **key=value** pairs. This is also commonly referred to as logfmt format.

Many modern systems can process logs in **logfmt** format. For example, DataDog, Splunk, Grafana Loki. Logfmt is human readable and fairly easy to parse.

JSON Handler

≡ Table of Contents

You can also output the logs in JSON format, all you have to do is switch out the handler.

main.go

```
package main

import (
    "os"

    "golang.org/x/exp/slog"
)

func main() {
    jsonHandler := slog.NewJSONHandler(os.Stdout)  // 👈
    logger := slog.New(jsonHandler)

    logger.Info("Go is the best language!")
}
```

Output:

```
$ go run main.go
{"time":"2022-12-17T18:05:48.479126-05:00","level":"INFO","msg":"Go is
```

Each log is logged as a json object with properties inside of it.

Attributes

slog being a structured logger, provides the ability to specify attributes.

main.go

```
package main

import (
    "os"

    "golang.org/x/exp/slog"
```

≡ Table of Contents

```
)
```



```
    textHandler := slog.NewTextHandler(os.Stdout)
    logger := slog.New(textHandler)

    logger.Info("Usage Statistics", slog.Int("current-memory", 50))
}
```

Output:

```
$ go run main.go
time=2022-12-17T18:28:38.246-05:00 level=INFO msg="Usage Statistics" c
```

In the above example, an integer attributes has been added using `slog.Int` .

Various types of attributes are available:

```
String
Int64
Int
Uint64
Float64
Bool
Time
Duration
```

You can add as many attributes as required.

main.go

```
package main

import (
    "os"

    "golang.org/x/exp/slog"
)

func main() {
    textHandler := slog.NewTextHandler(os.Stdout)
```

Table of Contents

```
    logger := slog.New(textHandler)

    logger.Info("Usage Statistics",
        slog.Int("current-memory", 50),
        slog.Int("min-memory", 20),
        slog.Int("max-memory", 80),
        slog.Int("cpu", 10),
        slog.String("app-version", "v0.0.1-beta"),
    )
}
```

Output:

```
$ go run main.go
time=2022-12-17T18:34:12.781-05:00 level=INFO msg="Usage Statistics" c
```

Grouping Attributes

You can group attributes under a single key. For example, all the memory attributes can be grouped under the `memory` key.

main.go

```
package main

import (
    "os"

    "golang.org/x/exp/slog"
)

func main() {
    textHandler := slog.NewTextHandler(os.Stdout)
    logger := slog.New(textHandler)

    logger.Info("Usage Statistics",
        slog.Group("memory",
            slog.Int("cu
            slog.Int("m
            slog.Int("max", 80)),
```

≡ Table of Contents

```
            slog.Int("cpu", 10),
            slog.String("app-version", "v0.0.1-beta"),
        )
    }
```

Output:

```
$ go run main.go
time=2022-12-17T18:36:46.660-05:00 level=INFO msg="Usage Statistics" m
```

Using a `JsonHandler` the output in json would be as follow.

```
$ go run main.go | jq
{
  "time": "2022-12-17T18:38:04.74786-05:00",
  "level": "INFO",
  "msg": "Usage Statistics",
  "memory": {
    "current": 50,
    "min": 20,
    "max": 80
  },
  "cpu": 10,
  "app-version": "v0.0.1-beta"
}
```

Common Attributes

Let's say you want to have an attribute that should be included in all the logs being generated, examples of such an attribute would include **name of the service**, **application version**.

You can attach attributes to the handler that will be included in each log statement.

main.go

```
package main
```

≡ Table of Contents

```
import (
    "context"
    "os"

    "golang.org/x/exp/slog"
)

func main() {
    textHandler := slog.NewTextHandler(os.Stdout).
        WithAttrs([]slog.Attr{slog.String("app-version", "v0.0.1-beta"
    logger := slog.New(textHandler)

    logger.Info("Generating statistics")
    logger.Info("Usage Statistics",
        slog.Group("memory",
            slog.Int("current", 50),
            slog.Int("min", 20),
            slog.Int("max", 80)),
        slog.Int("cpu", 10),
    )
}
```

Output:

```
$ go run main.go
time=2022-12-17T20:21:27.664-05:00 level=INFO msg="Generating statisti
time=2022-12-17T20:21:27.664-05:00 level=INFO msg="Usage Statistics" a
```

You can see the `app-version` attribute being included in both the logs. Attributes specified using `WithAttrs` function on the handler will be included in all the logs.

Passing logger in context

You would ideally want to create a single logger with certain configurations, attributes and use it throughout the application.

≡ Table of Contents

slog has inbuilt functions tha[illegible]e logger inside a `context`.

main.go

```
package main

import (
    "context"
    "os"

    "golang.org/x/exp/slog"
)

func main() {
    textHandler := slog.NewTextHandler(os.Stdout).
        WithAttrs([]slog.Attr{slog.String("app-version", "v0.0.1-beta"
    logger := slog.New(textHandler)

    ctx := slog.NewContext(context.Background(), logger) // 👈 context
    sendUsageStatus(ctx)
}

func sendUsageStatus(ctx context.Context) {
    logger := slog.FromContext(ctx) // 👈 grab logger from context

    logger.Info("Generating statistics")

    logger.Info("Usage Statistics",
        slog.Group("memory",
            slog.Int("current", 50),
            slog.Int("min", 20),
            slog.Int("max", 80)),
        slog.Int("cpu", 10),
    )
}
```

Output:

```
$ go run main.go
time=2022-12-17T20:27:58.797-05:00 level=INFO msg="Generating statisti
time=2022-12-17T20:27:58                    ) msg="Usage Statistics" a
```

≡ Table of Contents

`NewContext` creates a new context containing the logger.

`FromContext` grabs the logger from a context. In case the context doesn't contain a logger, it returns the default logger.

Log Level Logging

If you are using the default logger, it doesn't log debug logs because the default log level is `Info`.

You can create a new logger with the default log level set to `Debug` to show debug logs.

main.go

```
package main

import (
    "os"

    "golang.org/x/exp/slog"
)

func main() {
    opts := slog.HandlerOptions{
        Level: slog.LevelDebug,
    }

    textHandler := opts.NewTextHandler(os.Stdout)
    logger := slog.New(textHandler)

    logger.Debug("Debug")
    logger.Info("Info")
    logger.Warn("Warn")
}
```

Output:

≡ Table of Contents

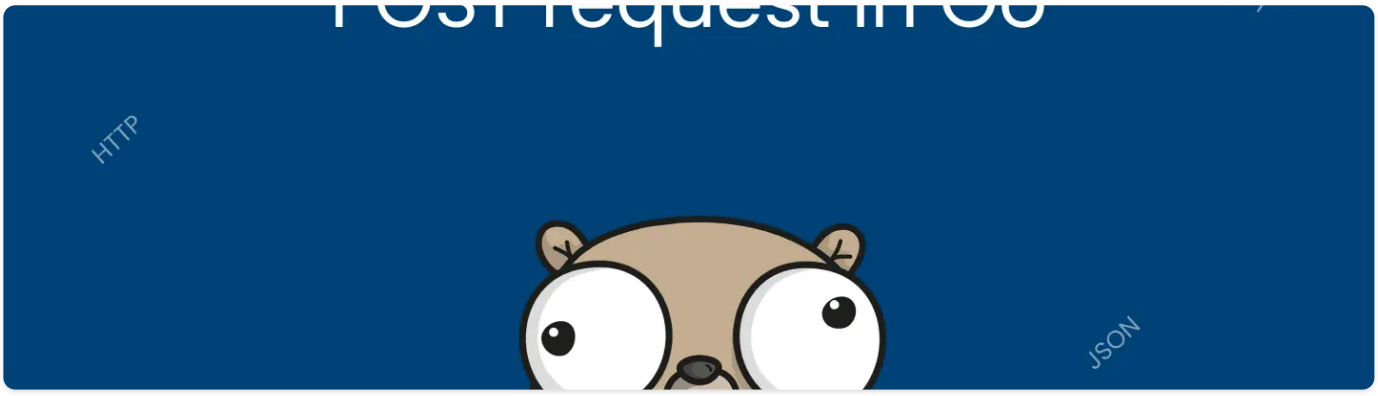
```
$ go run main.go
time=2022-12-17T23:28:29.130-05:00 level=DEBUG msg=Debug
```

```
time=2022-12-17T23:28:29.130-05:00 level=INFO msg=Info
time=2022-12-17T23:28:29.130-05:00 level=WARN msg=Warn
```

It is amazing to see these types of packages that could potentially make there way to golang core, makes the language so much more powerful and appealing.

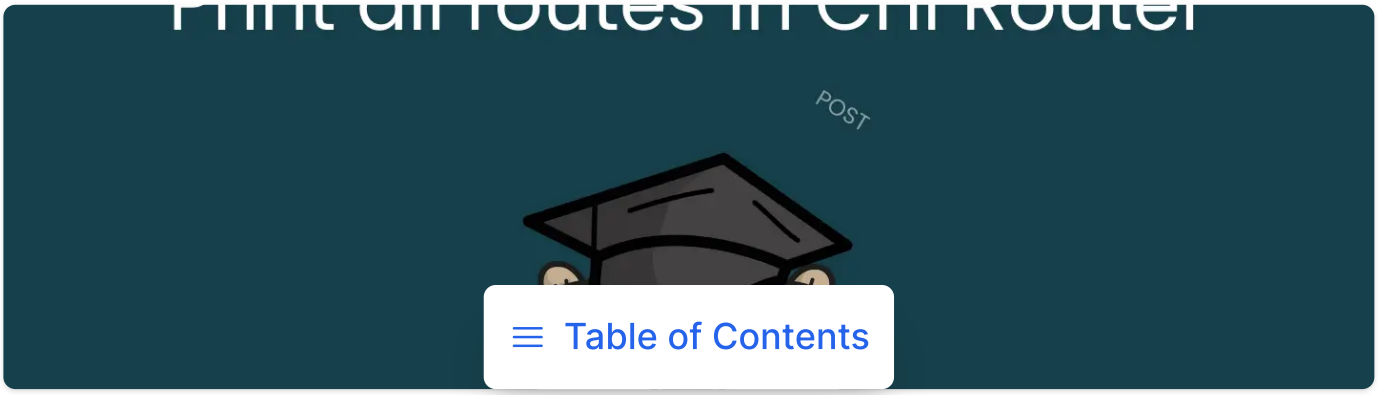
Thank you for reading the article, if you liked it please consider joining our email newsletter.

Related Articles



GO

Make POST request in Go using net/http



GO

Print all routes in Chi router

GO

Connect to Postgres in Go (Golang)

TheDeveloperCafe

Table of Contents