

Go 每日一库之 logrus

Original dj GoUpUp 2020-02-08

收录于话题

#Go 每日一库

48个 >

简介

前一篇文章介绍了 Go 标准库中的日志库 `log`。最后我们也提到，`log` 库只提供了三组接口，功能过于简单了。今天，我们来介绍一个日志库中的“明星库”——`logrus`。本文编写之时（2020.02.07），`logrus` 在 GitHub 上 star 数已达到 13.8k。`logrus` 完全兼容标准的 `log` 库，还支持文本、JSON 两种日志输出格式。很多知名的开源项目都使用了这个库，如大名鼎鼎的 `docker`。

快速使用

第三方库需要先安装：

```
$ go get github.com/sirupsen/logrus
```

后使用：

```
package main
```

```
import (  
    "github.com/sirupsen/logrus"  
)  
  
func main() {  
    logrus.SetLevel(logrus.TraceLevel)  
  
    logrus.Trace("trace msg")  
    logrus.Debug("debug msg")  
    logrus.Info("info msg")  
    logrus.Warn("warn msg")  
    logrus.Error("error msg")  
    logrus.Fatal("fatal msg")  
    logrus.Panic("panic msg")  
}
```

`logrus` 的使用非常简单，与标准库 `log` 类似。`logrus` 支持更多的日志级别：

- `Panic`：记录日志，然后 `panic`。
- `Fatal`：致命错误，出现错误时程序无法正常运转。输出日志后，程序退出；
- `Error`：错误日志，需要查看原因；
- `Warn`：警告信息，提醒程序员注意；
- `Info`：关键操作，核心流程的日志；
- `Debug`：一般程序中输出的调试信息；
- `Trace`：很细粒度的信息，一般用不到；

日志级别从上向下依次增加，`Trace` 最大，`Panic` 最小。`logrus` 有一个日志级别，高于这个级别的日志不会输出。默认的级别为 `InfoLevel` 。所以为了能看到 `Trace` 和 `Debug` 日志，我们在 `main` 函数第一行设置日志级别为 `TraceLevel` 。

运行程序，输出：

```
$ go run main.go
time="2020-02-07T21:22:42+08:00" level=trace msg="trace msg"
time="2020-02-07T21:22:42+08:00" level=debug msg="debug msg"
time="2020-02-07T21:22:42+08:00" level=info msg="info msg"
time="2020-02-07T21:22:42+08:00" level=info msg="warn msg"
time="2020-02-07T21:22:42+08:00" level=error msg="error msg"
time="2020-02-07T21:22:42+08:00" level=fatal msg="fatal msg"
exit status 1
```

由于 `logrus.Fatal` 会导致程序退出，下面的 `logrus.Panic` 不会执行到。

另外，我们观察到输出中有三个关键信息，`time`、`level` 和 `msg`：

- `time`：输出日志的时间；
- `level`：日志级别；
- `msg`：日志信息。

定制

输出文件名

调用 `logrus.SetReportCaller(true)` 设置在输出日志中添加文件名和方法信息：

```
package main

import (
    "github.com/sirupsen/logrus"
)

func main() {
    logrus.SetReportCaller(true)

    logrus.Info("info msg")
}
```

输出多了两个字段 `file` 为调用 `logrus` 相关方法的文件名，`method` 为方法名：

```
$ go run main.go
time="2020-02-07T21:46:03+08:00" level=info msg="info msg" func=main.main file="D:/code/golang/src/github.com/darjun/go-daily-lib/logrus.go" method=main.main
```

添加字段

有时候需要在输出中添加一些字段，可以通过调用 `logrus.WithField` 和 `logrus.WithFields` 实现。`logrus.WithFields` 接受一个 `logrus.Fields` 类型的参数，其底层实际上为 `map[string]interface{}`：

```
// github.com/sirupsen/logrus/logrus.go type Fields map[string]interface{}
```

下面程序在输出中添加两个字段 `name` 和 `age`：

```
package main

import (
    "github.com/sirupsen/logrus"
)

func main() {
    logrus.WithFields(logrus.Fields{
        "name": "dj",
        "age": 18,
    }).Info("info msg")
}
```

如果在一个函数中的所有日志都需要添加某些字段，可以使用 `WithFields` 的返回值。例如在 Web 请求的处理器中，日志都要加上 `user_id` 和 `ip` 字段：

```
package main

import (
    "github.com/sirupsen/logrus"
)

func main() {
```

```
requestLogger := logrus.WithFields(logrus.Fields{
    "user_id": 10010,
    "ip":      "192.168.32.15",
})

requestLogger.Info("info msg")
requestLogger.Error("error msg")
}
```

实际上，`WithFields` 返回一个 `logrus.Entry` 类型的值，它将 `logrus.Logger` 和设置的 `logrus.Fields` 保存下来。调用 `Entry` 相关方法输出日志时，保存下来的 `logrus.Fields` 也会随之输出。

重定向输出

默认情况下，日志输出到 `io.Stderr`。可以调用 `logrus.SetOutput` 传入一个 `io.Writer` 参数。后续调用相关方法日志将写到 `io.Writer` 中。现在，我们就能像上篇文章介绍log时一样，可以搞点事情了。传入一个 `io.MultiWriter`，同时将日志写到 `bytes.Buffer`、标准输出和文件中：

```
package main

import (
    "bytes"
    "io"
    "log"
    "os"
    "github.com/sirupsen/logrus"
)

func main() {
    writer1 := &bytes.Buffer{}
    writer2 := os.Stdout
    writer3, err := os.OpenFile("log.txt", os.O_WRONLY|os.O_CREATE, 0755)
```

```
if err != nil {
    log.Fatalf("create file log.txt failed: %v", err)
}

logrus.SetOutput(io.MultiWriter(writer1, writer2, writer3))
logrus.Info("info msg")
}
```

自定义

实际上，考虑到易用性，库一般会使用默认值创建一个对象，包最外层的方法一般都是操作这个默认对象。

我们之前好几篇文章都提到过这点：

- Go 每日一库之 flag：flag 标准库中的 `CommandLine` 对象；
- Go 每日一库之 log：log 标准库中的 `std` 对象。

这个技巧应用在很多库的开发中，logrus 也是如此：

```
// github.com/sirupsen/logrus/exported.govar (
    std = New()
)

func StandardLogger() *Logger {
    return std
}

func SetOutput(out io.Writer) {
```

```

std.SetOutput(out)
}

func SetFormatter(formatter Formatter) {
    std.SetFormatter(formatter)
}

func SetReportCaller(include bool) {
    std.SetReportCaller(include)
}

func SetLevel(level Level) {
    std.SetLevel(level)
}

```

首先，使用默认配置定义一个 **Logger** 对象 **std**，**SetOutput/SetFormatter/SetReportCaller/SetLevel** 这些方法都是调用 **std** 对象的对应方法！

我们当然也可以创建自己的 **Logger** 对象，使用方式与直接调用 **logrus** 的方法类似：

```

package main

import "github.com/sirupsen/logrus"

func main() {
    log := logrus.New()

    log.SetLevel(logrus.InfoLevel)
    log.SetFormatter(&logrus.JSONFormatter{})
}

```



```
log.Info("info msg")
}
```

日志格式

logrus 支持两种日志格式，文本和 JSON，默认为文本格式。可以通过 **logrus.SetFormatter** 设置日志格式：

```
package main

import (
    "github.com/sirupsen/logrus"
)

func main() {
    logrus.SetLevel(logrus.TraceLevel)
    logrus.SetFormatter(&logrus.JSONFormatter{})

    logrus.Trace("trace msg")
    logrus.Debug("debug msg")
    logrus.Info("info msg")
    logrus.Warn("warn msg")
    logrus.Error("error msg")
    logrus.Fatal("fatal msg")
    logrus.Panic("panic msg")
}
```

程序输出 JSON 格式的日志：

```
$ go run main.go
{"level":"trace","msg":"trace msg","time":"2020-02-07T21:40:04+08:00"}
{"level":"debug","msg":"debug msg","time":"2020-02-07T21:40:04+08:00"}
{"level":"info","msg":"info msg","time":"2020-02-07T21:40:04+08:00"}
{"level":"info","msg":"warn msg","time":"2020-02-07T21:40:04+08:00"}
{"level":"error","msg":"error msg","time":"2020-02-07T21:40:04+08:00"}
{"level":"fatal","msg":"fatal msg","time":"2020-02-07T21:40:04+08:00"}
exit status 1
```

第三方格式

除了内置的 `TextFormatter` 和 `JSONFormatter`，还有不少第三方格式支持。我们这里介绍一个 `nested-logrus-formatter`。

先安装：

```
$ go get github.com/antonfisher/nested-logrus-formatter
```

后使用：

```
package main

import (
    nested "github.com/antonfisher/nested-logrus-formatter"
    "github.com/sirupsen/logrus"
)

func main() {
    logrus.SetFormatter(&nested.Formatter{
```

```

    HideKeys:    true,
    FieldsOrder: []string{"component", "category"},
})

logrus.Info("info msg")
}

```

程序输出：

```
Feb  8 15:22:59.077 [INFO] info msg
```

nested 格式提供了多个字段用来定制行为：

```

// github.com/antonfisher/nested-logrus-formatter/formatter.go type Formatter struct {
    FieldsOrder    []string
    TimestampFormat string
    HideKeys       bool
    NoColors       bool
    NoFieldsColors bool
    ShowFullLevel  bool
    TrimMessages   bool
}

```

- 默认，**logrus** 输出日志中字段是 **key=value** 这样的形式。使用 **nested** 格式，我们可以通过设置 **HideKeys** 为 **true** 隐藏键，只输出值；
- 默认，**logrus** 是按键的字母序输出字段，可以设置 **FieldsOrder** 定义输出字段顺序；

- 通过设置 `TimestampFormat` 设置日期格式。

```
package main

import (
    "time"

    nested "github.com/antonfisher/nested-logrus-formatter"
)

func main() {
    logrus.SetFormatter(&nested.Formatter{
        // HideKeys:      true,
        TimestampFormat: time.RFC3339,
        FieldsOrder:     []string{"name", "age"},
    })

    logrus.WithFields(logrus.Fields{
        "name": "dj",
        "age":  18,
    }).Info("info msg")
}
```

如果不隐藏键，程序输出：

```
$ 2020-02-08T15:40:07+08:00 [INFO] [name:dj] [age:18] info msg
```

隐藏键，程序输出：

```
$ 2020-02-08T15:41:58+08:00 [INFO] [dj] [18] info msg
```

注意到，我们将时间格式设置成 `time.RFC3339`，即 `2006-01-02T15:04:05Z07:00` 这种形式。

通过实现接口 `logrus.Formatter` 可以实现自己的格式。

```
// github.com/sirupsen/logrus/formatter.gotype Formatter interface {  
    Format(*Entry) ([]byte, error)  
}
```

设置钩子

还可以为 `logrus` 设置钩子，每条日志输出前都会执行钩子的特定方法。所以，我们可以添加输出字段、根据级别将日志输出到不同的目的地。`logrus` 也内置了一个 `syslog` 的钩子，将日志输出到 `syslog` 中。这里我们实现一个钩子，在输出的日志中增加一个 `app=awesome-web` 字段。

钩子需要实现 `logrus.Hook` 接口：

```
// github.com/sirupsen/logrus/hooks.gotype Hook interface {  
    Levels() []Level  
    Fire(*Entry) error  
}
```

`Levels()` 方法返回感兴趣的日志级别，输出其他日志时不会触发钩子。`Fire` 是日志输出前调用的钩子方法。

```
package main

import (
    "github.com/sirupsen/logrus"
)

type AppHook struct {
    AppName string
}

func (h *AppHook) Levels() []logrus.Level {
    return logrus.AllLevels
}

func (h *AppHook) Fire(entry *logrus.Entry) error {
    entry.Data["app"] = h.AppName
    return nil
}

func main() {
    h := &AppHook{AppName: "awesome-web"}
    logrus.AddHook(h)

    logrus.Info("info msg")
}
```

只需要在 `Fire` 方法实现中，为 `entry.Data` 添加字段就会输出到日志中。

程序输出：

```
$ time="2020-02-08T15:51:52+08:00" level=info msg="info msg" app=awesome-web
```

`logrus` 的第三方 Hook 很多，我们可以使用一些 Hook 将日志发送到 redis/mongodb 等存储中：

- `mgorus`：将日志发送到 mongodb；
- `logrus-redis-hook`：将日志发送到 redis；
- `logrus-amqp`：将日志发送到 ActiveMQ。

这里我们演示一个 redis，感兴趣自行验证其他的。先安装 `logrus-redis-hook`：

```
$ go get github.com/rogierlommers/logrus-redis-hook
```

然后编写程序：

```
package main

import (
    "io/ioutil"

    logredis "github.com/rogierlommers/logrus-redis-hook"
    "github.com/sirupsen/logrus"
)
```

```
)

func init() {
    hookConfig := logredis.HookConfig{
        Host:      "localhost",
        Key:        "mykey",
        Format:     "v0",
        App:        "awesome",
        Hostname:   "localhost",
        TTL:        3600,
    }

    hook, err := logredis.NewHook(hookConfig)
    if err == nil {
        logrus.AddHook(hook)
    } else {
        logrus.Errorf("logredis error: %q", err)
    }
}

func main() {
    logrus.Info("just some info logging...")

    logrus.WithFields(logrus.Fields{
        "animal": "walrus",
        "foo":    "bar",
        "this":   "that",
    }).Info("additional fields are being logged as well")
}
```



```
logrus.SetOutput(ioutil.Discard)
logrus.Info("This will only be sent to Redis")
}
```

为了程序能正常工作，我们还需要安装 `redis` 。

windows 上直接使用choco安装 redis：

```
PS C:\Users\Administrator> choco install redis-64
Chocolatey v0.10.15
Installing the following packages:
redis-64
By installing you accept licenses for the packages.
Progress: Downloading redis-64 3.0.503... 100%

redis-64 v3.0.503 [Approved]
redis-64 package files install completed. Performing other installation steps.
ShimGen has successfully created a shim for redis-benchmark.exe
ShimGen has successfully created a shim for redis-check-aof.exe
ShimGen has successfully created a shim for redis-check-dump.exe
ShimGen has successfully created a shim for redis-cli.exe
ShimGen has successfully created a shim for redis-server.exe
The install of redis-64 was successful.
Software install location not explicitly set, could be in package or
default install location if installer.

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
```

直接输入 `redis-server`，启动服务器：



运行程序后，我们使用 `redis-cli` 查看：



我们看到 `mykey` 是一个 `list`，每过来一条日志，就在 `list` 后新增一项。

总结

本文介绍了 `logrus` 的基本用法。`logrus` 的可扩展性非常棒，可以引入第三方格式和 Hook 增强功能。在社区也比较受欢迎。

参考

1. [logrus GitHub 仓库](#)

2. Hooks

我

我的博客

欢迎关注我的微信公众号【GoUpUp】，共同学习，一起进步~



People who liked this content also liked

Go 每日一库之 reflect

GoUpUp



权威解读：为啥要实行财产和行为税合并申报？

国家税务总局



彩妆届的天花板！时装周御用的彩妆真的那么好吗？

是皮哥Piggg

