zap

Blazing fast, structured, leveled logging

최흥배
blog: https://jacking75.github.io/
Github: https://github.com/jacking75

짧은 시간이므로 잘 시간도 없습니다.
(강연 문서도 무미건조 합니다)

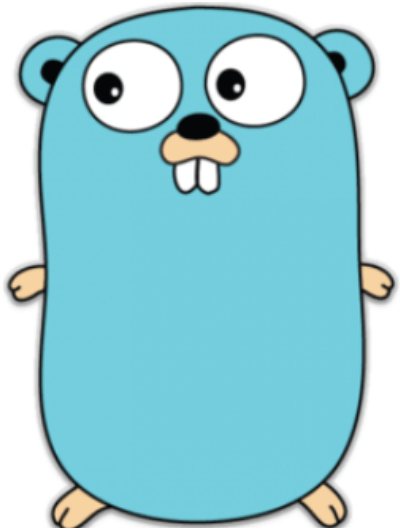
간단한 내용이므로 이 강의를 듣고 나면 바로 사용할 수 있습니다!!!

하는 일?

게임 회사 컴투스의 센트럴서버팀에서

기술 전파

게임 서버 개발 관련 R&D, 기술 지원 등의 일을 하고 있습니다



Go

체인 스트라이크 (Chain Strike)

에디터 추천

Com2uS 롤플레잉

★★★★☆ 21,469

12

광고 포함 · 인앱 구매 제공

사용 중인 모든 기기와 호환되는 앱입니다.

설치됨

Android, iOS

댄스빌 4+

소셜가무 게임!

Com2uS Corp.

음악 앱 5위

★★★★½ 4.7, 2.1천개의 평가

무료 · 앱 내 구입 제공

스크린샷 iPhone iPad

Android, iOS

Golang을 사용하는 새로운 프로젝트는 늘어나는 중...

게임 회사는 보통 게임 프로젝트 마다 사용되는 기술이 다른 경우가 많습니다.

zap

- Uber사에서 만든 오픈 소스 로그 라이브러리.
- https://github.com/uber-go/zap
- 로그 라이브러리 중 가장 빠름.
- 기능도 필요로 하는 것이 거의 다 있음(1, 2개는 없음).

zap - Blazing fast

Log a message and 10 fields:

Package	Time	Objects Allocated
⚡ zap	3131 ns/op	5 allocs/op
⚡ zap (sugared)	4173 ns/op	21 allocs/op
zerolog	16154 ns/op	90 allocs/op
lion	16341 ns/op	111 allocs/op
go-kit	17049 ns/op	126 allocs/op
logrus	23662 ns/op	142 allocs/op
log15	36351 ns/op	149 allocs/op
apex/log	42530 ns/op	126 allocs/op

https://github.com/uber-go/zap

zap - structured

```
logger, _ := zap.NewProduction()
defer logger.Sync() // flushes buffer, if any
sugar := logger.Sugar()
sugar.Infow("failed to fetch URL",
  // Structured context as loosely typed key-value pairs.
  "url", url,
  "attempt", 3,
  "backoff", time.Second,
)
sugar.Infof("Failed to fetch URL: %s", url)
```

```
logger.Info("Failed to fetch URL.",
    // Structured context as strongly typed fields.
    zap.String("url", url),
    zap.Int("attempt", 3),
    zap.Duration("backoff", time.Second),
)
```

Output:

```
{"level":"info","msg":"Failed to fetch URL.","url":"http://example.com","attempt":3,"backoff
{"level":"info","msg":"Failed to fetch URL: http://example.com"}
{"level":"info","msg":"Failed to fetch URL.","url":"http://example.com","attempt":3,"backoff
```

https://godoc.org/go.uber.org/zap

zap의 성능이 좋은 이유

- Reflection을 사용 하지 않는다.

```
logger.Info("failed to fetch URL", zap.String("url", url), zap.Int("attempt", 3))
```

- **할당**하지 않고 JSON 인코더를 사용한다.
- 가능한 직렬화 오버 헤드와 **할당을 피한다**.

allocate 한 Buffer나 Encoder는 **sync.Pool로 재사용** 하고 있다. 이 Pool은 이미 allocate 된 아이템을 재 이용 하기 위한 것으로 **GC의 부담을 완화 시켜준다**. Pool의 아이템은 멋대로 삭제 되기도 하고, 만약 참조만 가지고 있다면 그대로 deallocate 된다.

https://github.com/uber-go/zap/blob/v1.4.0/buffer/pool.go#L34

```
func NewPool() Pool {
    return Pool{p: &sync.Pool{
        New: func() interface{} {
            return &Buffer{bs: make([]byte, 0, _size)}
        },
    }}
}
```

다양한 설정 변경과 독자적인 인코더 이용에 의한 유연성

- 스택 추적 표시/숨기기, caller 표시/숨기기, 출력 형식(Console 형식 / Json 형식)을 선택하고, Time과 Duration의 표시 형식 선택 등 기본적으로 제공되는 변경할 수 있는 설정도 다수 있다.

- 그래도 요건을 충족하지 않는다면 독자적인 엔코더를 작성하여 이용하는 것도 가능하다.

Quick Start

Installation

```
go get -u go.uber.org/zap
```

main.go

```go
package main

import "go.uber.org/zap"

func main() {
    logger, _ := zap.NewDevelopment()
    logger.Info("Hello zap", zap.String("key", "value"), zap.Time("now", time.Now()))
}
```

2017-03-23T17 : 52 : 59.005 + 0900 INFO zap-example / main.go : 7 Hello zap { "key" : "value" , "now" : "2017-03-23T17 : 52 : 59.005 + 0900" }

Logger 구조체

- ❏ 로깅을 실시하기 위한 구조체이다.
- ❏ SugaredLogger 보다 빠르고 낮은 할당으로 작동하지만 **구조적 스타일의 로깅 밖에 없다**(print와 printf 스타일의 로깅은 할 수 없다).

로깅 샘플 코드

```
logger.Debug("msg", zap.String("Key", "String"), zap.Ints("ints", []int{10, 20}))
```

output

2017-03-27T10 : 17 : 19.930 + 0900 DEBUG zap-example / main.go : 5 msg { "Key" : "String" , "int" : [10, 20] }

Method	Description
New	직접 만든 Core에서 만든다 가장 유연하고 가장 장황한 방법
Config # Build	직접 정의한 Config에서 만든다 가장 기본적인 방법 (자세한 내용은 Config편 참조)
NewDevelopment	미리 정의된 설정의 Logger를 작성 dev 용이라는 평가 답게 console 형식으로 표시
NewProduction	미리 정의된 설정의 Logger를 작성 prd 용이라는 평가 답게 json 형식으로 표시
Logger # With	현재 Logger를 clone, 인수에 지정된 필드를 유지한채 새로운 로거를 얻는다
Logger # WithOptions	현재 Logger를 clone, 인수에 지정한 옵션을 적용한 새로운 로거를 얻는다
Logger # Named	현재 Logger를 clone, 인수에 지정한 이름의 세그먼트를 추가한 새로운 로거를 취득 세그먼트는 Name 항목으로 표시
Logger # Sugar	SugaredLogger을 취득
Logger # { 로그 수준 }	지정한 로그 메시지와 필드로 로깅
Logger # Sync	버퍼링 된 로그 항목을 flush 한다

SugaredLogger 구조체

- ❏ Logger로부터 얻을 수 있는 간단한 Logger 이다.
- ❏ 구조화 로깅과 printf 스타일 로깅 모두 사용할 수 있지만, **Logger 보다 저속이고 많은 할당을 한다**(그래도 다른 로깅 라이브러리 보다 빠른 것 같다).

로깅 샘플 코드

```
suger.Info("one", "two", "three")
suger.Infof("one: %s, %d", "two", 10)
suger.Infow("msg", "key", "value", "intArray", []int{10, 100}, "duration", time.Second*200)
```

output

2017-03-27T10 : 17 : 19.930 + 0900 INFO zap-example / main.go : 5 onetwothree
2017-03-27T10 : 17 : 19.930 + 0900 INFO zap-example / main.go : 6 one : two 10
2017-03-27T10 : 17 : 19.930 + 0900 INFO zap-example / main.go : 7 msg { "key" : "value" , "intArray" : [10, 100], "duration" : "3m20s" }

Method	Description
SugaredLogger # With	현재 SugaredLogger (정확하게는 내부에 보유하고 있는 원래의 Logger)를 clone, 인수에 지정된 필드를 유지한채 새로운 로거를 얻는다
SugaredLogger # Named	현재 SugaredLogger (정확하게는 내부에 보유하고 있는 원래의 Logger)를 clone, 인수에 지정한 이름의 세그먼트를 추가한 새로운 로거를 취득 세그먼트 Name 항목으로 표시
SugaredLogger # Desugar	원래 Logger를 얻을
SugaredLogger # {로그 수준 }	fmt.Print 스타일로 로깅
SugaredLogger # {로그 수준 } f	fmt.Printf스타일로 로깅
SugaredLogger # {로그 수준 } w	구조적 스타일에서 로깅
SugaredLogger # Sync	버퍼링 된 로그 항목을 flush 하기

로그 필드 타입에 구조체 사용하기

func MarshalLogObject(ObjectEncoder) error 메소드를 구현해야 한다.

```
type user struct {
    Name      string
    Email     string
    CreatedAt time.Time
}

func (u user) MarshalLogObject(enc zapcore.ObjectEncoder) error {
    enc.AddString("name", u.Name)
    enc.AddString("email", u.Email)
    enc.AddInt64("created_at", u.CreatedAt.UnixNano())
    return nil
}

func main() {
    logger, _ := zap.NewDevelopment()
    user := &user{
        Name: "Zap",
        Email: "zap@sample.com",
        CreatedAt: time.Now(),
    }
    logger.Info("object sample", zap.Object("userObj", user))
}
```

ObjectMarshalerFunc을 사용한 클로져 패턴도 가능하다.

```
type user struct {
    Name      string
    Email     string
    CreatedAt time.Time
}

func main() {
    logger, _ := zap.NewDevelopment()
    user := &user{
        Name: "Zap",
        Email: "zap@sample.com",
        CreatedAt: time.Now(),
    }
    logger.Info("object sample", zap.Object("object", zapcore.ObjectMarshalerFunc(func(inner zapcore.ObjectEncoder) error {
        inner.AddString("name", user.Name)
        inner.AddString("email", user.Email)
        inner.AddInt64("created_at", user.CreatedAt.UnixNano())
        return nil
    })))
}
```

Array 사용하기

```
type user struct {
    Name string
}

type users []*user

func (us users) MarshalLogArray(enc zapcore.ArrayEncoder) error {
    for _, u := range us {
        enc.AppendString(u.Name)
    }
    return nil
}

func main() {
    logger, _ := zap.NewDevelopment()
    var users users = []*user{
        &user{Name: "Zap1"},
        &user{Name: "Zap2"},
        &user{Name: "Zap3"},
    }
    logger.Info("array sample", zap.Array("userArray", users))
}
```

array sample　{ "userArray" : ["Zap1" , "Zap2" , "Zap3"]}

```
type user struct {
    Name string
}

func main() {
    logger, _ := zap.NewDevelopment()
    users := []*user{
        &user{Name: "Zap1"},
        &user{Name: "Zap2"},
        &user{Name: "Zap3"},
    }
    logger.Info("array sample", zap.Array("userArray", zapcore.ArrayMarshalerFunc(func(inner zapcore.ArrayEncoder) error {
        for _, u := range users {
            inner.AppendString(u.Name)
        }
        return nil
    })))
}
```

Config 구조체

- zap 설정을 정의하는 구조체이다.
- Config **코드** 기반으로 설정하는 방법과 **JSon** 또는 **Yaml** 파일로 설정하는 방법이 있다.
- 어떤 식 만든 Config의 Build 메소드로 Logger를 생성 할 수 있다.

```
level := zap.NewAtomicLevel()
level.SetLevel(zapcore.DebugLevel)

myConfig := zap.Config{
    Level: level,
    Encoding: "json",
    EncoderConfig: zapcore.EncoderConfig{
        TimeKey:        "Time",
        LevelKey:       "Level",
        NameKey:        "Name",
        CallerKey:      "Caller",
        MessageKey:     "Msg",
        StacktraceKey:  "St",
        EncodeLevel:    zapcore.CapitalLevelEncoder,
        EncodeTime:     zapcore.ISO8601TimeEncoder,
        EncodeDuration: zapcore.StringDurationEncoder,
        EncodeCaller:   zapcore.ShortCallerEncoder,
    },
    OutputPaths:      []string{"stdout"},
    ErrorOutputPaths: []string{"stderr"},
}
logger, _ := myConfig.Build()
```

```
{
  "level": "debug",
  "encoding": "json",
  "encoderConfig": {
    "messageKey": "Msg",
    "levelKey": "Level",
    "timeKey": "Time",
    "nameKey": "Name",
    "callerKey": "Caller",
    "stacktraceKey": "St",
    "levelEncoder": "capital",
    "timeEncoder": "iso8601",
    "durationEncoder": "string",
    "callerEncoder": "short"
  },
  "outputPaths": [
    "stdout"
  ],
  "errorOutputPaths": [
    "stderr"
  ]
}
```

```
configJson, err := ioutil.ReadFile("./config.json")
if err != nil {
    panic(err)
}
var myConfig zap.Config
if err := json.Unmarshal(configJson, &myConfig); err != nil {
    panic(err)
}
logger, _ := myConfig.Build()
```

```yaml
level: "debug"
encoding: "json"
encoderConfig:
  messageKey: "Msg"
  levelKey: "Level"
  timeKey: "Time"
  nameKey: "Name"
  callerKey: "Caller"
  stacktraceKey: "St"
  levelEncoder: "capital"
  timeEncoder: "iso8601"
  durationEncoder: "string"
  callerEncoder: "short"
outputPaths:
  - "stdout"
errorOutputPaths:
  - "stderr"
```

```go
configYaml, err := ioutil.ReadFile("./config.yaml")
if err != nil {
    panic(err)
}
var myConfig zap.Config
if err := yaml.Unmarshal(configYaml, &myConfig); err != nil {
    panic(err)
}
logger, _ := myConfig.Build()
```

항목 목록

Item	Type	Description	Key (for json / yaml)	Value (for json / yaml)
Level	AtomicLevel	유효한 로그 레벨 필수 동적으로 변경 가능하며, 변경하면 이 Config에서 생성된 모든 Logger에 반영	level	debug info warn error dpanic panic fatal
Development	bool	Dev 모드 / Prd 모드 true : Dev 모드, false : Prd 모드	development	true false
DisableCaller	bool	호출자 (로그 파일 이름과 줄 번호) 출력 여부 true : 출력하지 않는, false : 출력	disableCaller	true false
DisableStacktrace	bool	스택 추적 출력의 유무 true : 출력하지 않는 ,false : 출력	disableStacktrace	true false
Sampling	* SamplingConfig	샘플링 설정 자세한 내용은 SamplingConfig를 참조	sampling	SamplingConfig를 참조
Encoding	string	엔코더 필수 Console 형식 또는 Json 형식을 선택 가능	encoding	console json
EncoderConfig	EncoderConfig	표시에 관한 인코더 설정 자세한 내용은 EncoderConfig를 참조	encoderConfig	EncoderConfig를 참조
OutputPaths	[] string	로그 출력	outputPaths	모든 파일 경로 stdout stderr
ErrorOutputPaths	[] string	zap 내부 오류 출력	errorOutputPaths	모든 파일 경로 stdout stderr
InitialFields	map [string] interface {}	초기 필드 모든 Logger에 넣고 싶은 필드가 있는 경우 등에 이용 할 수있다	initialFields	모든 Key (string)과 Value

SamplingConfig

- 샘플링 설정을 정의하는 구조체이다.
- 이를 정의하면 같은 로그 수준, 같은 메시지 로그를 **1 초에 출력 할 수 있는 수를 제한 할 수** 있고, 이를 통해 CPU 부하 및 I / O 부하를 억제 할 수 있다.

Item	Type	Description	Key (for json / yaml)	Value (for json / yaml)
Initial	int	첫 번째 샘플링 상한선	initial	1 이상의 정수
Thereafter	int	이후의 샘플링 상한선	thereafter	1 이상의 정수

예를 들어, first = 3, thereafter = 5로 SamplingConfig를 정의한 상태에서 logger.Info("sample")1 초에 10 회 호출했을 경우, 다음과 같이 출력된다.

```
[1 회] INFO sample <- 즉시 출력
[2 회] INFO sample <- 즉시 출력
[3 회] INFO sample <- 즉시 출력
[4 회] INFO sample <- 8 회째 같은 타이밍으로 출력
[5 회] INFO sample <- 8 회째 같은 타이밍으로 출력
[6 회] INFO sample <- 8 회째 같은 타이밍으로 출력
[7 회] INFO sample <- 8 회째 같은 타이밍으로 출력
[8 회] INFO sample <- 즉시 출력
[9 회] INFO sample <- 출력되지 않는다 (다음의 출력 타이밍 (13 번째)이 오지 않기 때문에)
[10 회] INFO sample <- 출력되지 않는다 (다음의 출력 타이밍 (13 번째)이 오지 않기 때문에)
```

설정 파일에 지정한 로그 출력 파일 이름 변경하기

- 설정 파일의 로그 파일 이름은 고정이라서, 실행 때 마다 이름을 다르게 하고 싶다.
- 로그 파일에 날짜시간을 주고 싶다면 설정 파일 로딩 후 OutputPaths 패쓰에 파일 이름이 있으면 이것을 다른 이름으로 교체한다.

```
{
  "level": "debug",
  "encoding": "json",
  "encoderConfig": {
    "messageKey": "Msg",
    "levelKey": "Level",
    "timeKey": "Time",
    "nameKey": "Name",
    "callerKey": "Caller",
    "levelEncoder": "capital",
    "timeEncoder": "iso8601",
    "durationEncoder": "string",
    "callerEncoder": "short"
  },
  "outputPaths": [
    "stdout", "server.log"
  ],
  "errorOutputPaths": [
    "stderr"
  ]
}
```

```go
var myConfig zap.Config

if err := json.Unmarshal(configJson, &myConfig); err != nil {
    panic(err)
}

for index := range myConfig.ErrorOutputPaths{
    if myConfig.OutputPaths[index] != "stdout"{
        myConfig.OutputPaths[index], _ = _createFileName(myConfig.OutputPaths[index])
    }
}

func _createFileName(outputName string) (string, error){
    currentTime := time.Now()
    formattedTime := currentTime.Format("20060102_150405")
    fileNameArr := strings.Split(outputName, ".")
    fileName := fileNameArr[0]
    fileExt := "." + fileNameArr[1]
    if len(fileNameArr) > 2{
        return "", errors.New("log ouput name Invalid")
    }
    return fileName + formattedTime + fileExt, nil
}
```

로그 파일 로테이션 하기

- zap은 로그 파일 로테이션 기능이 없다.
- 다른 외부 라이브러리와 결합해서 사용한다.
- sink를 이용한다.
- io.Writer를 wrap한 WriteSyncer에서 AddSync(w io.Writer)로 변환할 수 있다.
- 여기에 lumberjack을 넘겨주면 rotate 할 수 있다.

```
config := zap.NewProductionConfig()
enc := zapcore.NewJSONEncoder(config.EncoderConfig)
sink := zapcore.AddSync(
    &lumberjack.Logger{
        Filename:   "./aaaa.log",
        MaxSize:    500, // megabytes
        MaxBackups: 3,
        MaxAge:     28, //days
    },
)
logger := zap.New(
    zapcore.NewCore(enc, sink, config.Level),
)
defer logger.Sync()
logger.Error("aaa",
    zap.String("eeef", "eefe"),
)
```