Golang Tutorial #4

- go code error
- gin (RESTful) error handling
- go log
- go kafka
- go debug on vscode

Go Error

- native error is just a wrapper of string
- usually, we need more info in error for debugging
- gopkg

```
// CodeError - Code Error interface
type CodeError interface {
        ErrorCode() int
        Error() string
}
```

Gin Error

- https://issuetracking.maaii.com:9443/display/RnD/REST+API+JSON+Schema+for+Response+Body

```
{
    "code": 2010102,
    "message": "only owner can read/write plugin detail"
}
```

Gin middleware error

- use `c.Abort` and not go to `c.Next()`

```
func GinMiddleware()) gin.HandlerFunc {
        return func(c *gin.Context) {
                xxx, err := ...
                if err != nil {
                        common.GinError(c, err)
                        return
                }

                c.Next()
        }
}
```

Gin common Error handling

- unified handling for flexibility

```
func GinError(c *gin.Context, err gopkg.CodeError) {
        // ...
        response := Response{
                Code:    err.ErrorCode(),
                Message: err.Error(),
        }
        c.AbortWithStatusJSON(status, response)
}
```

go log

- `github.com/siruspen/logrus`
- `func (logger *Logger) WithField(key string, value interface{}) *Entry`
- `func (logger *Logger) WithFields(fields Fields) *Entry`
- `func (entry *Entry) WithField(key string, value interface{}) *Entry`
- `func (entry *Entry) WithFields(fields Fields) *Entry`
- logrus.Hook (flexible usage)
- easy to integrate common log framework

kafka hook example

```
func (hook *kafkaHook) Fire(entry *logrus.Entry) error {
// ...
        select {
        case hook.sendQueue <- &sp.ProducerRecord{Topic: to
                return nil
        case <-time.After(hook.enqueueTimeout):
                return errors.New("Enqueue Timeout Drop")
        }
}
```

current ELK usecase

- gitlab.com/general-backend/m800log
- k8s fluentbit (forward log to ES)
- format json string for ELK

```
func (f *M800JSONFormatter) Format(entry *logrus.Entry) ([]
        // ...
        data := make(logrus.Fields, len(entry.Data)+BuildIn
        data[goctx.LogKeyApp] = f.App

        serialized, err := json.Marshal(data)
        if err != nil {
                return nil, fmt.Errorf("failed to marshal f
        }
        return append(serialized, '\n'), nil
}
```

go kafka

- github.com/confluentinc/confluent-kafka-go (cgo library) more easy
- github.com/Shopify/sarama another common option

go kafka consumer

- use consumer group to achieve

  `maaii-pubsub Once/OncePerServer...`

```
func getConsumerConfig() *kafka.ConfigMap {
        config := &kafka.ConfigMap{
                "bootstrap.servers": viper.GetString("kafka
                "group.id":          "myTestGroup",
                "auto.offset.reset": "earliest",
        }
        return config
}
```

go kafka producer

```
func getProducerConfig() *kafka.ConfigMap {
        config := &kafka.ConfigMap{
                "bootstrap.servers": viper.GetString("kafka
        }
        return config
}
```

go debug on vscode

- go get -u github.com/go-delve/delve/cmd/dlv
- lazy: open main.go and launch debugger
- check your debugger launch point