Golang Tutorial #4

Go practice

- go code error
- gin (RESTful) error handling
- go log
- go kafka

Go Error

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

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

HTTP Response Error


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

Gin middleware error

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

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

                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: topic, Value: line}:
                return nil
        case <-time.After(hook.enqueueTimeout):
                return errors.New("Enqueue Timeout Drop")
        }
}
```

EFK usecase

- elastic-search/fluen-bit/Kibana
- k8s fluentbit (forward log to ES)
- format json string for ELK
- ES automatic check value type and create index

Log Formater

```
// log formatter
func (f *M800JSONFormatter) Format(entry *logrus.Entry)
 ([]byte, error) {
        // ...
        data := make(logrus.Fields, len(entry.Data)+BuildInFieldNumber)
        data[goctx.LogKeyApp] = f.App

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

go kafka

- github.com/confluentinc/confluent-kafka-go (cgo library)
  - good performance
  - official support

go kafka consumer

- use consumer group to achieve one event only handled once by one group

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

go kafka producer

- example
- must create goroutine to handle ack queue to avoid local queue full

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

go kafka producer handle

```
// send event to kafka
err := p.Produce(&kafka.Message{
        //...
        Value:          []byte(word),
}, nil)
if err != nil {
        log.Println(err)
}
p.Flush(1 * 1000)
// read ack
go func(){
        for e := range p.Events() {
                ...
        }
}
```

go kafka serialization

- ```
  func (c *Consumer) ReadMessage(timeout time.Duration)
  (*Message, error)
  ```
- all we sent and got are []byte
- use header to filter event early
  ```