Go 错误处理（附 GORM 示例）

1. 视频地址

https://www.bilibili.com/video/BV1Dm411S7Nn/?vd_source=fc6752da08268ab41ccb99bcd33482f4

Go 错误处理（包含 GORM 示例）_哔哩哔哩_bilibili

1. 目标 介绍 Go `errors` 包，以及通过实例说明，在使用 Gorm 时，如何进行错误处理。 ---# 2. errors 及 fmt 包 创建错误实例:* `errors.New` * 返回 `errorString` 实例* `fmt.Errorf` *

---

2. 创建错误实例

- `errors.New`：返回 `errorString` 实例
- `fmt.Errorf`：当格式化说明符中包含 `%w` 动词时，返回 `wrapError` 实例（包含一个 `%w`）或 `wrapErrors` 实例（包含多个 `%w`）；否则，使用 `errors.New` 创建错误实例

---

3. 比较错误以及类型断言

- 使用 `errors.Is(err, target)` 替代 `err == target`
- 使用 `errros.As(err, target)` 替代 `err.(*target)`

---

4. 获取封装的错误

`errors.Unwrap`

---

5. 组合错误

- uber/multierr

---

6. 示例代码

```
|  errors_test.go
|  go.mod
|  main.go
|  multierr_test.go
|
├─common
|      custom_error.go
|      custom_error_test.go
|
├─datasource
|      gorm_mysql.go
|
├─entity
|      user.go
|      user_mapper.go
|
├─model
|      user.go
|
├─repository
|      user.go
|
└─service
        common.go
        error.go
        user.go
```

errors_test.go：

```go
package main

import (
    "errors"
    "fmt"
    "testing"
)

func TestErrorsUnwrap(t *testing.T) {
    baseErr := errors.New("base error")
    wrapErr := fmt.Errorf("wrapped error: %w", baseErr)
    t.Log(wrapErr.Error())
    if !errors.Is(baseErr, errors.Unwrap(wrapErr)) {
       t.Fatal("should equal")
    }
}
```

go.mod：

```
module go-error-example

go 1.21

require (
    github.com/go-sql-driver/mysql v1.7.1
    github.com/jinzhu/copier v0.4.0
    go.uber.org/multierr v1.11.0
    golang.org/x/crypto v0.19.0
    gorm.io/driver/mysql v1.5.4
    gorm.io/gorm v1.25.7
)

require (
    github.com/jinzhu/inflection v1.0.0 // indirect
    github.com/jinzhu/now v1.1.5 // indirect
    golang.org/x/sys v0.17.0 // indirect
)
```

main.go：

```
package main

import (
    "context"
    "fmt"
    "go-error-example/datasource"
    "go-error-example/repository"
    "go-error-example/service"
    "log"
    "time"
)

func main() {
    dsn := "XXX:XXX@tcp(42.194.131.135:3306)/test?charset=utf8mb4&parseTime=True&loc=Local"
    db, err := datasource.NewGormMySQLDataSource(
       &datasource.GormMySQLConfig{DSN: dsn},
    )
    if err != nil {
       log.Panicln(err)
```

```
    }
    repo := repository.NewUserRepository(db)
    svc := service.NewUserService(repo)
    ctx := context.Background()

    for i := 1; ; i++ {
       id, err := svc.CreateUser(ctx, fmt.Sprintf("Tim-%d", i))
       if err != nil {
          log.Panic(err.Error())
       }
       log.Printf("id: %d", id)

       user, err := svc.GetUserByID(ctx, id)
       if err != nil {
          log.Panic(err.Error())
       }
       log.Printf("user.Name: %s", user.Name)

       time.Sleep(5 * time.Second)
    }
}
```

multierr_test.go：

```
package main

import (
    "errors"
    "go.uber.org/multierr"
    "testing"
)

func TestMultierr(t *testing.T) {
    var err error
    err = multierr.Append(err, errors.New("call 1 failed"))
    err = multierr.Append(err, nil)
    err = multierr.Append(err, errors.New("call 2 failed"))
    err = multierr.Append(err, nil)
    if err.Error() != "call 1 failed; call 2 failed" {
       t.Fatal("unexpected error string")
    }

    for _, e := range multierr.Errors(err) {
       t.Log(e.Error())
    }
```

```
}
```

common/custom_error.go：

```go
package common

import "errors"

// RetryableError 表示知道是否进行重试的错误。
// 区分错误是否可以重试非常重要。
// 如果对不可重试错误进行重试，那么可能导致过多的无效重试，甚至可能带来其它意想不到的后果。
// 比如，如果发生 SQL 语法错误，那么无论重试多少次都将一直失败。
// 如果发生数据库读写超时，那么重试可以增加成功概率，使系统更加健壮
type RetryableError interface {
    error
    // CanRetry 返回 true 表示发生可重试错误，比如依赖的 Rest 服务返回 503 响应；
    // 返回 false 表示发生不可重试错误，比如参数校验失败
    CanRetry() bool
}

// IsRetryableError 用于判断 err 是否可以进行重试。
// 如果 err 或其封装的错误实现 RetryableError 接口，并且其 CanRetry 方法返回 true，那么 err 是可重试错误
func IsRetryableError(err error) bool {
    target := new(RetryableError)
    if errors.As(err, target) {
       return (*target).CanRetry()
    }
    return false
}

// CustomError 的所有字段都是可导出的，以便对其进行序列化
type CustomError struct {
    Message   string
    Retryable bool
}

func (c *CustomError) Error() string {
    return c.Message
}

func (c *CustomError) CanRetry() bool {
    return c.Retryable
}
```

```
40
41 // NewRetryableError 构造可重试错误实例
42 func NewRetryableError(message string) error {
43     return &CustomError{
44        Message:   message,
45        Retryable: true,
46     }
47 }
48
49 // NewNonRetryableError 构造不可重试错误实例
50 func NewNonRetryableError(message string) error {
51     return &CustomError{
52        Message:   message,
53        Retryable: false,
54     }
55 }
```

common/custom_error_test.go：

```
1  package common
2
3  import (
4      "errors"
5      "fmt"
6      "testing"
7  )
8
9  func TestIsRetryableError(t *testing.T) {
10     err := errors.New("test")
11     if IsRetryableError(err) {
12        t.Fatal("should not be retryable")
13     }
14
15     err = NewRetryableError("test")
16     if !IsRetryableError(err) {
17        t.Fatal("should be retryable")
18     }
19
20     // 封装可重试错误的错误
21     err = fmt.Errorf("%w", err)
22     if !IsRetryableError(err) {
23        t.Fatal("should be retryable")
24     }
25
26     err = NewNonRetryableError("test")
```

```
    if IsRetryableError(err) {
        t.Fatal("should not be retryable")
    }

    // 封装不可重试错误的错误
    err = fmt.Errorf("%w", err)
    if IsRetryableError(err) {
        t.Fatal("should not be retryable")
    }
}
```

datasource/gorm_mysql.go：

```
package datasource

import (
    "context"
    "fmt"
    "github.com/go-sql-driver/mysql"
    "golang.org/x/crypto/ssh"
    gormMySQL "gorm.io/driver/mysql"
    "gorm.io/gorm"
    "net"
    "os"
    "time"
)

// SSHAgent 表示 SSH 代理。
// 可以通过 SSH 代理，连接部署在内网的 MySQL 数据库
type SSHAgent struct {
    Host string
    Port int
    User string
    // 当 Password 和 KeyFile 同时存在时，优先使用 KeyFile
    Password string
    KeyFile  string
}

// dialWithPassword 使用密码创建 SSH 客户端
func (s *SSHAgent) dialWithPassword() (*ssh.Client, error) {
    address := fmt.Sprintf("%s:%d", s.Host, s.Port)
    config := &ssh.ClientConfig{
        User: s.User,
        Auth: []ssh.AuthMethod{
            ssh.Password(s.Password),
```

```
        },
        HostKeyCallback: ssh.InsecureIgnoreHostKey(),
    }
    return ssh.Dial("tcp", address, config)
}

// dialWithKeyFile 使用密钥文件创建 SSH 客户端
func (s *SSHAgent) dialWithKeyFile() (*ssh.Client, error) {
    address := fmt.Sprintf("%s:%d", s.Host, s.Port)
    config := &ssh.ClientConfig{
        User:            s.User,
        HostKeyCallback: ssh.InsecureIgnoreHostKey(),
    }
    if k, err := os.ReadFile(s.KeyFile); err != nil {
        return nil, err
    } else {
        signer, err := ssh.ParsePrivateKey(k)
        if err != nil {
            return nil, err
        }
        config.Auth = []ssh.AuthMethod{
            ssh.PublicKeys(signer),
        }
    }
    return ssh.Dial("tcp", address, config)
}

// Dial 创建连接。可以用于 mysql.RegisterDialContext
func (s *SSHAgent) Dial(ctx context.Context, addr string) (net.Conn, error) {
    var sshClient *ssh.Client
    var err error
    // KeyFile 优先级比 Password 高
    if s.KeyFile != "" {
        sshClient, err = s.dialWithKeyFile()
    } else if s.Password != "" {
        sshClient, err = s.dialWithPassword()
    }
    if err != nil {
        return nil, err
    }
    return sshClient.DialContext(ctx, "tcp", addr)
}

type SSHConfig struct {
    Host     string
    Port     int
    User     string
```

```
80      Password string
81      KeyFile  string
82  }
83
84  type GormMySQLConfig struct {
85      ConnMaxLifetimeMS int64
86      ConnMaxIdleTimeMS int64
87      MaxIdleConns      int
88      MaxOpenConns      int
89      DSN               string
90      SSH               *SSHConfig
91  }
92
93  // NewGormMySQLDataSource 构造 Gorm MySQL 数据源
94  func NewGormMySQLDataSource(config *GormMySQLConfig) (*gorm.DB, error) {
95      connMaxLifetimeMs := config.ConnMaxLifetimeMS
96      if connMaxLifetimeMs <= 0 {
97          connMaxLifetimeMs = 10 * 60 * 1000
98      }
99      connMaxIdleTimeMS := config.ConnMaxIdleTimeMS
100     if connMaxIdleTimeMS <= 0 {
101         connMaxIdleTimeMS = 1 * 60 * 1000
102     }
103     maxIdleConns := config.MaxIdleConns
104     if maxIdleConns <= 0 {
105         maxIdleConns = 5
106     }
107     maxOpenConns := config.MaxOpenConns
108     if maxOpenConns <= 0 {
109         maxOpenConns = 50
110     }
111     dsn := config.DSN
112     if dsn == "" {
113         dsn = "root:@tcp(127.0.0.1:3306)/test?
    charset=utf8mb4&parseTime=True&loc=Local"
114     }
115     sshConfig := config.SSH
116     sshHost := ""
117     sshPort := 0
118     sshUser := ""
119     sshPassword := ""
120     sshKeyFile := ""
121     if sshConfig != nil {
122         sshHost = sshConfig.Host
123         if sshHost == "" {
124             sshHost = "localhost"
125         }
```

```
126         sshPort = sshConfig.Port
127         if sshPort <= 0 {
128             sshPort = 22
129         }
130         sshUser = sshConfig.User
131         if sshUser == "" {
132             sshHost = "root"
133         }
134         sshPassword = sshConfig.Password
135         sshKeyFile = sshConfig.KeyFile
136     }
137     if sshHost != "" && sshPort > 0 && sshPort <= 65535 && sshUser != "" &&
    (sshPassword != "" || sshKeyFile != "") {
138         agent := &SSHAgent{
139             Host:     sshHost,
140             Port:     sshPort,
141             User:     sshUser,
142             Password: sshPassword,
143             KeyFile:  sshKeyFile,
144         }
145         mysql.RegisterDialContext("mysql+ssh", agent.Dial)
146     }
147 
148     // 初始化 DB 会话
149     db, err := gorm.Open(
150         gormMySQL.New(
151             gormMySQL.Config{
152                 // 在 DSN 中设置 timeout（连接超时）、readTimeout（读超时）、
    writeTimeout（写超时）
153                 DSN:       dsn,
154                 DSNConfig: &mysql.Config{
155                     // 返回匹配的行数，而非改变的行数。
156                     // 在很多逻辑中，期望返回改变的行数，因此谨慎设置该选项
157                     // ClientFoundRows: false,
158                 },
159             },
160         ),
161         &gorm.Config{
162             // 预编译语句，提高性能
163             PrepareStmt: true,
164         },
165     )
166     if err != nil {
167         return nil, err
168     }
169 
170     // 配置连接池
```

```
171     sqlDB, err := db.DB()
172     if err != nil {
173        return nil, err
174     }
175     // 重用连接的最长时间
176     sqlDB.SetConnMaxLifetime(time.Duration(connMaxLifetimeMs) *
    time.Millisecond)
177     // 连接的最大空闲时间
178     sqlDB.SetConnMaxIdleTime(time.Duration(connMaxIdleTimeMS) *
    time.Millisecond)
179     // 空闲连接池中的最大连接数
180     sqlDB.SetMaxIdleConns(maxIdleConns)
181     // 打开的最大连接数
182     sqlDB.SetMaxOpenConns(maxOpenConns)
183
184     return db, nil
185 }
```

entity/user.go：

```
1  package entity
2
3  type UserEntity struct {
4      ID   int64  `gorm:"column:id;primaryKey;autoIncrement"`
5      Name string `gorm:"column:name;size:128;unique"`
6  }
7
8  func (u *UserEntity) TableName() string {
9      return "user"
10 }
```

entity/user_mapper.go：

```
1  package entity
2
3  import (
4      "github.com/jinzhu/copier"
5      "go-error-example/model"
6  )
7
8  func UserModelToEntity(user *model.User) (entity *UserEntity, err error) {
9      entity = new(UserEntity)
10     err = copier.CopyWithOption(entity, user, copier.Option{IgnoreEmpty: true})
```

```
    return
}

func UserEntityToModel(entity *UserEntity) (user *model.User, err error) {
    user = new(model.User)
    err = copier.CopyWithOption(user, entity, copier.Option{IgnoreEmpty:
false})
    return
}
```

model/user.go：

```
package model

type User struct {
    ID   int64  `json:"id"`
    Name string `json:"name"`
}
```

repository/user.go：

```
package repository

import (
    "context"
    "go-error-example/entity"
    "gorm.io/gorm"
)

type UserRepository struct {
    db *gorm.DB
}

func (u *UserRepository) CreateUser(ctx context.Context, user
*entity.UserEntity) (int64, error) {
    db := u.db.WithContext(ctx).Omit("ID").Create(user)
    if err := db.Error; err != nil {
       return 0, err
    }
    return user.ID, nil
}

```

```go
func (u *UserRepository) GetUserByID(ctx context.Context, id int64)
(*entity.UserEntity, error) {
    user := new(entity.UserEntity)
    db := u.db.WithContext(ctx).Model(user).Where("`id` = ?",
id).Limit(1).First(user)
    if err := db.Error; err != nil {
        return nil, err
    }
    return user, nil
}

func NewUserRepository(db *gorm.DB) *UserRepository {
    return &UserRepository{db: db}
}
```

service/common.go：

```go
package service

import (
    "errors"
    "github.com/go-sql-driver/mysql"
    "net"
)

func MapError(err error) error {
    if err == nil {
        return nil
    }

    var opError *net.OpError
    if errors.As(err, &opError) {
        return ErrRetryableRepositoryFailure
    }

    if errors.Is(err, mysql.ErrInvalidConn) {
        return ErrRetryableRepositoryFailure
    }

    return err
}
```

service/error.go：

```go
package service

import "go-error-example/common"

var (
    ErrRetryableRepositoryFailure = common.NewRetryableError("retryable repository failure")
    ErrUserNotFound               = common.NewNonRetryableError("user not found")
    ErrUserAlreadyExists          = common.NewNonRetryableError("user already exists")
)
```

service/user.go：

```go
package service

import (
    "context"
    "errors"
    "github.com/go-sql-driver/mysql"
    "go-error-example/entity"
    "go-error-example/model"
    "go-error-example/repository"
    "gorm.io/gorm"
)

type UserService struct {
    repo *repository.UserRepository
}

func (u *UserService) mapError(err error) error {
    if errors.Is(err, gorm.ErrRecordNotFound) {
       return ErrUserNotFound
    }
    var mysqlError *mysql.MySQLError
    if errors.As(err, &mysqlError) {
       switch mysqlError.Number {
       case 1062:
          return ErrUserAlreadyExists
       default:
       }
    }
    return MapError(err)
}
```

```go

func (u *UserService) CreateUser(ctx context.Context, name string) (int64, error) {
    user := &model.User{Name: name}
    userEntity, err := entity.UserModelToEntity(user)
    if err != nil {
        return 0, err
    }
    id, err := u.repo.CreateUser(ctx, userEntity)
    if err != nil {
        // 转换错误
        return 0, u.mapError(err)
    }
    return id, nil
}

func (u *UserService) GetUserByID(ctx context.Context, id int64) (*model.User, error) {
    userEntity, err := u.repo.GetUserByID(ctx, id)
    if err != nil {
        // 转换错误
        return nil, u.mapError(err)
    }
    return entity.UserEntityToModel(userEntity)
}

func NewUserService(repo *repository.UserRepository) *UserService {
    return &UserService{repo: repo}
}
```