链滴

关于 golang 中 db 等事务处理时代码逻辑处理

作者：xhaoxiong
原文链接：https://ld246.com/article/1630496469000
来源网站：链滴

概述

在编码过程中，特别是在写dbo层的数据库操作时，我们经常会遇到写事务的地方，在很长一段时间都是使用的正常逻辑如下：

```
//在新建和更新的时候注意自动创建关联和更新
func (a *ArticleRepo) Create(article interface{}) (bool, error) {
    db := a.db.Begin()
    var tags []*models.Tag
    user := &models.User{}
    ac := article.(*models.Article)
    if err := db.Create(&ac).Error; err != nil {
        db.Rollback()
        return false, err
    }

    for _, t := range ac.Tags {
        tag := &models.Tag{}
        if err := db.Where("name = ? ", t.Name).First(&tag).Error; err != nil {
            tag.Name = t.Name
            tag.UserId = ac.AuthorID
        }
        tags = append(tags, tag)
    }

    if err := db.Where("id = ?", ac.AuthorID).First(&user).Error; err != nil {
        db.Rollback()
        return false, errors.New("请登陆")
    }

    if err := db.Model(&user).UpdateColumn("lottery_num",
```

```
        gorm.Expr("lottery_num + ?", 1)).Error; err != nil {
        db.Rollback()
        return false, err
    }

    if err := db.Model(&ac).Association("Tags").
        Append(tags).Error; err != nil {
        db.Rollback()
        return false, err
    }

    if err := db.Commit().Error; err != nil {
        db.Rollback()
        return false, err
    }
    return true, nil
}
```

我发现，在处理过程中会经常要写到db.rollback 于是在见识过比较多的常用写法后写法以“匿名函”的写法提取出主要流程完成主体业务省下大量无用代码，同时也避免忘记rollback和commit，目的写法以transction_test.go中为准：

首先创建事务接受上下文，db源，fn方法，然后将错误recover，进行wraper。

transction.go:

```
package dbx

import (
    "context"
    "fmt"
    "github.com/pkg/errors"
)

func NewTransaction(ctx context.Context, db Database, fn func(ctx context.Context, tx Transac
ion) error) (err error) {
    tx, err := db.Begin()
    if err != nil {
        return errors.Wrap(err, "begin")
    }
    // recover
    defer func() {
        if r := recover(); r != nil {
            var ok bool
            err, ok = r.(error)
            if !ok {
                err = fmt.Errorf("%v", r)
            }
            err = errors.WithMessage(err, "recover")
        }
        if err != nil {
            if e := tx.Rollback(); e != nil {
                err = errors.WithMessagef(err, "rollback %v", e)
            }
```

```
            return
        }
        err = errors.Wrap(tx.Commit(), "tx commit")
    }()

    return fn(ctx, tx)
}
```

sqlconn.go:

```
package dbx

import (
    "context"
    "database/sql"
    "github.com/jmoiron/sqlx"
)

type (
    Database interface {
        Begin() (Transaction, error)
        Ping() error
    }
    Transaction interface {
        Commit() error
        Rollback() error

        Get(dst interface{}, query string, args ...interface{}) error
        Select(dst interface{}, query string, args ...interface{}) error
        Exec(query string, args ...interface{}) (sql.Result, error)
        ExecContext(ctx context.Context, query string, args ...interface{}) (sql.Result, error)
        GetContext(ctx context.Context, dst interface{}, query string, args ...interface{}) error
        SelectContext(ctx context.Context, dst interface{}, query string, args ...interface{}) error
    }
)

type DatabaseSQLX struct {
    *sqlx.DB
}

func (db *DatabaseSQLX) Begin() (Transaction, error) {
    tx, err := db.Beginx()
    return &Tx{Tx: tx}, err
}

type Tx struct {
    *sqlx.Tx
}

func (t *Tx) Commit() error {
    return t.Tx.Commit()
}
```

```
func (t *Tx) Rollback() error {
    return t.Tx.Rollback()
}
```

mysql.go

```
package dbx

import (
    "fmt"
    "github.com/jmoiron/sqlx"
    "github.com/pkg/errors"
)

const DefaultMySQLDSN = "%s:%s@tcp(%s:%d)/%s?charset=utf8&parseTime=true"

func ConnectMySQL(format string, data DSN) (err error) {
    dsn := fmt.Sprintf(format, data.User, data.Password, data.Host, data.Port, data.DBName)
    db, err := sqlx.Connect("mysql", dsn)
    if err != nil {
        return errors.WithStack(err)
    }
    db.SetMaxIdleConns(5)
    db.SetMaxOpenConns(30)
    DB = &DatabaseSQLX{DB: db}
    return errors.WithStack(DB.Ping())
}
```

db.go

```
type DSN struct {
    Host     string
    Port     int
    User     string
    Password string
    DBName   string
}
```

trainsaction.go

```
func TestNewTransaction(t *testing.T) {
    ctx := context.Background()
    mockDB := &MockDB{}
    t.Run("success", func(t *testing.T) {
        defer resetTrans()
        assert.Equal(t, nil, NewTransaction(ctx, mockDB, func(ctx context.Context, tx Transaction)
error {
            return nil
        }))
        assert.Equal(t, false, rollback)
        assert.Equal(t, true, commit)
    })
    t.Run("return error", func(t *testing.T) {
        defer resetTrans()
        returnErr := errors.New("return error")
```

```
        err := NewTransaction(ctx, mockDB, func(ctx context.Context, tx Transaction) error {
            return errors.WithStack(returnErr)
        })
        assert.Equal(t, true, rollback)
        assert.Equal(t, false, commit)
        assert.Equal(t, returnErr, errors.Cause(err))
    })
    t.Run("panic error", func(t *testing.T) {
        defer resetTrans()
        panicErr := errors.New("panic error")
        err := NewTransaction(ctx, mockDB, func(ctx context.Context, tx Transaction) error {
            panic(panicErr)
        })
        assert.Equal(t, true, rollback)
        assert.Equal(t, false, commit)
        assert.Equal(t, panicErr, errors.Cause(err))
    })

}
```