链滴

Go 对象关系映射框架 GORM 使用示例

作者：iTanken

原文链接：https://ld246.com/article/1612423384133

来源网站：链滴

前提条件

- 安装 PostgreSql，创建好数据库（GORM 自动迁移表结构）
- 安装 GoLand 和 Go SDK
- 获取 GORM： go get -u gorm.io/gorm
- 获取 Postgres 驱动： go get -u gorm.io/driver/postgres

使用说明

1. 复制本代码，打开 GoLand

2. 按 <kbd>Ctrl</kbd> + <kbd>Shift</kbd> + <kbd>Alt</kbd> + <kbd>Insert</kbd> 建草稿文件

3. 将代码粘贴到草稿文件中，修改数据库连接地址，按 <kbd>Ctrl</kbd> + <kbd>Shift</kbd> + <kbd>F10</kbd> 运行代码查看效果

```
package main

import (
    "database/sql"
    "encoding/json"
    "errors"
    "fmt"
    "gorm.io/driver/postgres"
    "gorm.io/gorm"
    "gorm.io/gorm/logger"
    "gorm.io/gorm/schema"
    "log"
    "os"
```

```
    "strconv"
    "time"
)

// 系统参数表结构体（实体类），GORM 约定参考： https://gorm.io/zh_CN/docs/conventions.html
type TSysParam struct {
    // gorm.Model
    ID                  string    `gorm:"type:varchar(32);not null;primaryKey;<-:create;comment:流号参数唯一 ID"`
    DataTableName       string    `gorm:"type:varchar(128);<-;comment:数据库表名，如“t_sys_ser”"`
    DataTableDescription string    `gorm:"type:varchar(255);<-;comment:数据库表名（中文）说信息"`
    SerialValue         int       `gorm:"size:32;<-;comment:流水号当前最大值"`
    SerialLength        int       `gorm:"size:32;<-;comment:流水号长度，不足前缀以“0”补齐"`
    Remark              string    `gorm:"type:varchar(1024);<-;comment:备注信息"`
    Enabled             bool      `gorm:"<-;default:true;comment:是否可用"`
    CreateTime          *time.Time `gorm:"type:timestamptz;<-:create;autoCreateTime:milli;comment:创建时间"`
    CreateBy            string    `gorm:"type:varchar(32);<-:create;comment:创建人 ID，t_sys_user.id"`
    LastUpdateTime      *time.Time `gorm:"type:timestamptz;<-;comment:最后修改时间"`
    LastUpdateBy        string    `gorm:"type:varchar(32);<-;comment:最后修改人 ID，t_sys_user.id"`
}

// 为 TSysParam 结构体实现获取表名方法，单独设置 TSysParam 结构体的表名为 `t_sys_param`，配置全局禁用复数表名时可使用此方法
//func (TSysParam) TableName() string {
//    return "t_sys_param"
//}

// GORM 参考文档： https://gorm.io/zh_CN/docs/
func main() {
    initDbConn()

    // CRUD 示例
    createExample()
    readExample()
    updateExample()
    deleteExample()

    printStats()
}

// GORM 数据库定义
var GormDB *gorm.DB

// 连接池数据库句柄
var SqlDB *sql.DB

// 错误信息
var err error
```

```
// 初始化数据库连接
func initDbConn() {
    GormDB, err = gorm.Open(postgres.New(postgres.Config{
        // 通过一个现有的数据库连接来初始化，无需使用 DSN
        // Conn: SqlDB,
        // 数据源名称
        DSN: "host=192.168.1.1 port=5432 user=test password=test dbname=db_test sslmode
disable TimeZone=Asia/Shanghai",
        // 禁用隐式预处理语句
        PreferSimpleProtocol: true,
    }), &gorm.Config{
        // 日志配置
        Logger: getLogger(),
        // 自定义命名策略
        NamingStrategy: schema.NamingStrategy{
            // 全局使用单数表，禁止自动转换为复数形式表名
            SingularTable: true,
        },
        // 插入数据默认批处理大小
        CreateBatchSize: 1000,
    })
    if err != nil {
        panic("数据库连接失败！")
    }

    // 数据库连接池
    SqlDB, err = GormDB.DB()
    if err != nil {
        panic("数据库连接池获取失败！")
    }
    // 设置空闲连接池中连接的最大数量
    SqlDB.SetMaxIdleConns(10)
    // 设置打开数据库连接的最大数量
    SqlDB.SetMaxOpenConns(1e3)
    // 设置连接可复用的最大时间
    SqlDB.SetConnMaxLifetime(time.Hour)
    printStats()

    // 自动迁移给定模型为数据库表结构，未创建表或需要修改表结构的情况下可以启用
    // _ = GormDB.AutoMigrate(&TSysParam{})
}

// 获取当前时间指针
func nowTime() *time.Time {
    now := time.Now()
    return &now
}

// 添加数据，参考 https://gorm.io/zh_CN/docs/create.html
func createExample() {
    // 添加单条数据
    sysParam := TSysParam{
        ID:                  "test_001",
```

```
        DataTableName:        "test_table",
        DataTableDescription: "测试表",
        SerialValue:          0,
        SerialLength:         10,
        Enabled:              true,
        CreateBy:             "00000",
        CreateTime:           nowTime(),
    }

    result := GormDB.Create(&sysParam)
    printData(&sysParam, result, "Create")

    // 向指定(Select)字段中保存数据，忽略未指定的字段(NULL)
    sysParam = TSysParam{
        ID:                   "test_002",
        DataTableName:        "test_table_002",
        DataTableDescription: "测试表",
        SerialValue:          0,
        SerialLength:         10,
        Enabled:              true,
        CreateBy:             "00000",
    }
    result = GormDB.
        Select("ID", "DataTableName", "DataTableDescription", "SerialValue", "SerialLength").
        Create(&sysParam)
    printData(&sysParam, result, "Create")

    // 添加多条数据
    dataSize := 3
    sysParams := make([]TSysParam, dataSize)
    for i := 0; i < dataSize; i++ {
        index := strconv.Itoa(i)
        sysParams[i] = TSysParam{
            ID:                   "test_list_" + index,
            DataTableName:        "test_table_" + index,
            DataTableDescription: "测试表_" + index,
            SerialValue:          0,
            SerialLength:         10,
            Enabled:              true,
            CreateBy:             "00000",
        }
    }
    // 未配置全局 CreateBatchSize 参数的情况下，一次性批量保存全部数据
    result = GormDB.Create(&sysParams)
    // 指定单次批量保存的条数分批保存，每循环到 batchSize 条保存一次直至全部完成，保存大量
据可用此方法分批保存
    // result = GormDB.CreateInBatches(&sysParams, dataSize)
    printData(&sysParams, result, "Create", "(batch)")
}

// 查询数据，参考 https://gorm.io/zh_CN/docs/query.html  https://gorm.io/zh_CN/docs/advan
ed_query.html
func readExample() {
    var sysParam *TSysParam
```

```
    // 根据主键查询单条数据，默认根据主键正序排序
    result := GormDB.First(&sysParam, "00001")
    printData(sysParam, result, "First")
    sysParam = nil

    // 根据自定义条件查询最后一条数据，默认根据主键倒序排序
    result = GormDB.Last(&sysParam, "enabled = ?", false)
    printData(sysParam, result, "Last")
    sysParam = nil

    // 获取一条数据，未指定排序字段
    result = GormDB.Take(&sysParam, "create_by = ?", "00000")
    printData(sysParam, result, "Take")
    sysParam = nil

    // 不使用结构体查询，直接使用表名
    result = GormDB.Table("t_sys_param").First(&sysParam)
    printData(sysParam, result, "Table", "First")
    sysParam = nil

    // 查询全部
    sysParams := new([]TSysParam)
    result = GormDB.Find(&sysParams, "enabled = ?", true)
    // 查询可用数据
    // result = GormDB.Find(&sysParams, "enabled = ?", true)
    printData(&sysParams, result, "Find", "(all)")

    // 按 AND 条件查询多条
    sysParams = new([]TSysParam)
    result = GormDB.Find(&sysParams, TSysParam{Enabled: true, CreateBy: "00000"})
    // 不使用结构体，直接使用字段 Map
    // result = GormDB.Find(&sysParams, map[string]interface{}{"enabled": true, "create_by": "
0000"})
    printData(&sysParams, result, "Find", "(AND)")

    // 按 OR 条件查询多条
    sysParams = new([]TSysParam)
    result = GormDB.Where("enabled", true).Or("create_by", "00000").Find(&sysParams)
    printData(&sysParams, result, "Find", "(OR)")

    // 按 IN 条件查询多条
    sysParams = new([]TSysParam)
    result = GormDB.Find(&sysParams, "id IN ?", []string{"00001", "00002"})
    printData(&sysParams, result, "Find", "(IN)")

    // 按 NOT 条件查询多条
    sysParams = new([]TSysParam)
    result = GormDB.Not("serial_value", 0).Find(&sysParams)
    printData(&sysParams, result, "Find", "(NOT)")

    // 分页排序查询指定字段
    sysParams = new([]TSysParam)
    allSysParams := new([]TSysParam)
    result = GormDB.
```

```
        // 每页 5 条，第二页
        Offset(5).Limit(5).
        Order("serial_value DESC, id").
        Select("data_table_name", "serial_value", "serial_length").
        Find(&sysParams)
    printData(&sysParams, result, "Find", "(select-order-paging)")

    // 消除分页，获取全部
    unpageResult := result.Offset(-1).Limit(-1).Find(&allSysParams)
    // 总条数 pagingResult.RowsAffected
    printData(&allSysParams, unpageResult, "Find", "(select-order-paging-all)")
}

// 修改数据，参考 https://gorm.io/zh_CN/docs/update.html
func updateExample() {
    sysParam := TSysParam{
        ID:                  "test_001", // 根据主键修改指定数据
        DataTableName:       "test_table_001",
        DataTableDescription: "测试表-001",
        SerialValue:         1,
        SerialLength:        10,
        Enabled:             true,
        CreateTime:          nowTime(), // CreateTime 和 CreateBy 已在标签中配置为可读、可创建不可修改
        CreateBy:            "00001",
        LastUpdateTime:      nowTime(),
        LastUpdateBy:        "00000",
    }
    // 更新所有字段，包含零值
    result := GormDB.Save(&sysParam).
        // 查询修改结果重新赋值给 sysParam
        Find(&sysParam)
    printData(&sysParam, result, "UPDATE", "Save")

    // 更新单个字段
    result = GormDB.Model(&sysParam).
        Where("enabled", true).
        Update("serial_value", sysParam.SerialValue+1).
        Find(&sysParam)
    printData(&sysParam, result, "UPDATE", "Model", "WhereUpdate")

    // 更新多个字段，使用结构体只会更新非零值字段，要更新零值字段需要使用 Select 指定要修改字段，或者直接使用 Select("*") 更新全部字段
    sysParam = TSysParam{
        ID:             "test_001", // 根据主键修改指定数据
        SerialValue:    sysParam.SerialValue + 1,
        SerialLength:   10,
        Enabled:        false, // false 为 GORM Model 结构体零值，不会修改此字段
        LastUpdateTime: nowTime(),
        LastUpdateBy:   "00000",
    }
    result = GormDB.Model(&sysParam).
        Where("enabled", true).
        Updates(&sysParam).
```

```
        Find(&sysParam)
    printData(&sysParam, result, "UPDATE", "Model", "Updates")

    // 使用 map 可修改零值字段
    result = GormDB.Model(&sysParam).
        Updates(map[string]interface{}{"serial_value": sysParam.SerialValue + 1, "enabled": false})

        Find(&sysParam)
    printData(&sysParam, result, "UPDATE", "Model", "UpdatesMap")

    // 使用 Omit 忽略更新指定字段
    result = GormDB.Model(&sysParam).
        Omit("serial_value").
        Updates(map[string]interface{}{"serial_value": sysParam.SerialValue + 1, "enabled": true}).
        Find(&sysParam)
    printData(&sysParam, result, "UPDATE", "Model", "UpdatesOmit")
}

// 删除数据，参考 https://gorm.io/zh_CN/docs/delete.html
func deleteExample() {
    // !!! 注意，删除数据时如果未指定主键或其他条件，将会触发无条件的批量删除
    sysParam := TSysParam{
        ID: "test_002", // 主键
    }

    // 根据主键删除指定数据
    result := GormDB.Delete(&sysParam)
    sysParam = TSysParam{}
    result.Find(&sysParam)
    printData(&sysParam, result, "Delete")

    // 通过复合条件删除数据
    result = GormDB.Where("id LIKE ? AND data_table_name LIKE ?", "test_%", "test_table_%").
        Delete(&sysParam).
        Find(&sysParam)
    printData(&sysParam, result, "DeleteWhere")
}

// 获取 GORM 日志接口
func getLogger() logger.Interface {
    gormLogger := logger.New(
        // io writer
        log.New(os.Stdout, "\r\n", log.LstdFlags),
        logger.Config{
            // 慢 SQL 阈值
            SlowThreshold: 3 * time.Second,
            // 日志级别
            LogLevel: logger.Info,
            // 是否启用彩色打印
            Colorful: true,
        },
    )

    return gormLogger
```

```
}

// 打印数据
func printData(sysParam interface{}, result *gorm.DB, morInfo ...interface{}) {
    jsonByte, _ := json.Marshal(&sysParam)

    result.Logger.Info(nil, string(jsonByte))
    fmt.Println("条数：", result.RowsAffected,
        "\t错误信息：[", result.Error,
        "]\t是否为无记录错误：", errors.Is(result.Error, gorm.ErrRecordNotFound))

    if len(morInfo) > 0 {
        fmt.Println(morInfo)
    }
}

// 打印数据库统计信息
func printStats() {
    dbStats := SqlDB.Stats()
    jsonByte, _ := json.Marshal(dbStats)
    fmt.Println(string(jsonByte))

    /*fmt.Printf(`
    空闲连接数：  %d
    使用中的连接数：  %d
    由于达到设置的空闲连接池的最大数量而关闭的连接数：  %d
    由于达到设置的连接可空闲的最长时间而关闭的连接数：  %d
    由于达到设置的可重用连接的最长时间而关闭的连接数：  %d
    数据库的最大打开连接数：  %d
    等待的连接总数：  %d
    等待新连接被阻止的总时间：  %d`, dbStats.Idle, dbStats.InUse, dbStats.MaxIdleClosed, dbStats
MaxIdleTimeClosed,
    dbStats.MaxLifetimeClosed, dbStats.MaxOpenConnections, dbStats.WaitCount, dbStats.Wai
Duration)*/
}
```