DbFacade

DbFacade `github.com/Cepave/open-falcon-backend/common/db/facade`

DbFacade 是把下列三種資料庫物件集合的 struct，由 `DbConfig` 初始化:

- Gorm - 包含原物件與 Wrapper 物件
- Sqlx - 包含原物件與 Wrapper 物件
- database/sql - 包含原物件與 Wrapper 物件

```
type DbFacade struct {
    SqlDb *sql.DB
    SqlDbCtrl *commonDb.DbController

    GormDb *gorm.DB

    SqlxDb *sqlx.DB
    SqlxDbCtrl
*commonSqlx.DbController
}
```

初始化 DbFacade

```
import (
  f "github.com/Cepave/open-falcon-backend/common/db/facade"
  commonDb "github.com/Cepave/open-falcon-backend/common/db"
)

var DbFacade = &f.DbFacade{}
err := DbFacade.Open(&commonDb.DbConfig{
  Dsn: "root:cepave@tcp(192.168.20.50:3306)/falcon_portal",
  MaxIdle: 32,
})

if err != nil {
  logger.Warnf("Open database error: %v", err)
}
```

關閉 DbFacade

```
DbFacade.Release()
```

使用 Error-Free Wrapper

對於資料庫的不可回覆錯誤(例:資料庫連線中斷)，用 panic 方式直接丟，
並以一致的方包裝錯誤(logging 或 error 物件)，最後由 representation layer 統一處理，
可減少大量的程式碼，並讓系統更可靠。

下列為提供的 wrapper

使用 Gorm

Package: `github.com/Cepave/open-falcon-backend/common/gorm`

Gorm 使用 Method chaining，
wrapper 提供方法將 gorm 物件轉為 error-free 的擴充物件。

```
import (
  gormExt "github.com/Cepave/open-falcon-backend/common/gorm"
)

toGormExt := gormExt.ToDefaultGormDbExt
toGormExt(DbFacade.GormDb.First(&user)).PanicIfError()
```

使用 Sqlx

Package: `github.com/Cepave/open-falcon-backend/common/db/sqlx`

Sqlx 採用 Decorator pattern，
wrapper 提供方法將 sqlx 不提供的 function 轉為 error-free

```
import (
  sqlxExt "github.com/Cepave/open-falcon-backend/common/db/sqlx"
)

DbFacade.NewSqlxDbCtrl().Select(&user, "SELECT * FROM tb_user WHERE id =
?", userId)
```

使用 GoLang `database/sql`

Package: `github.com/Cepave/open-falcon-backend/common/db`

wrapper 提供方法將 `database/sql` 不提供的 function 轉為 error-free

```
import (
  "database/sql"
)

r := DbFacade.NewDbCtrl().Exec("INSERT INTO tb_user VALUES(?, ?)", "Bob
Wang", 30)
```

Transaction

Package: `github.com/Cepave/open-falcon-backend/common/db`

在此 package 中定義了 transaction 的 interface:

```
type TxCallback interface {
    InTx(tx *sql.Tx) TxFinale
}
```

另外也有純 function 的 wrapper

```
type TxCallbackFunc func(*sql.Tx) TxFinale
func (callbackFunc TxCallbackFunc) InTx(tx *sql.Tx) TxFinale {
    return callbackFunc(tx)
}
```

每一個 transaction callback 必需回傳 `TxFinale`，代表 `commit` 或 `rollback`:

```
type TxFinale byte

const (
    TxCommit TxFinale = 1
    TxRollback TxFinale = 2
)
```

每個實作 transaction IoC 的 wrapper，一旦 panic，**一律 rollback 該 transaction**。

`database/sql`

`*DbController.InTx`

產生 transaction，並把 `*sql.Tx` 物件傳入到 callback function 裡

```
import (
  commonDb "github.com/Cepave/open-falcon-backend/common/db"
)

doThingInTx := func(tx *sql.Tx) TxFinal {
  /* Your code; it maybe returns commonDb.TxRollback */

  return commonDb.TxCommit
}

DbFacade.SqlDbCtrl.InTx(doThingInTx)
```

`*DbController.InTxForIf`

複合 transaction，常用於 update or insert 的語法

- `BootCallback(tx *sql.Tx) bool` - 當此 function 回傳 `true` 值時，執行下一個 function:
- `IfTrue(tx *sql.Tx)` - 當上一個 function 回傳 `true` 值時，本 function 會被執行

```
type ExecuteIfByTx interface {
  // First calling of database for boolean result
  BootCallback(tx *sql.Tx) bool
  // If the boot callback has true result, this callback would
get called
  IfTrue(tx *sql.Tx)
}
```

`*DbController.ExecQueriesInTx`

把多個 SQL statement 在一個 transaction 裡執行(產生測試資料常用)

```
inTx := DbFacade.SqlDbCtrl.ExecQueriesInTx

inTx(
  "INSERT INTO tab_1 VALUES(...)",
  "INSERT INTO tab_2 VALUES(...)",
)
```

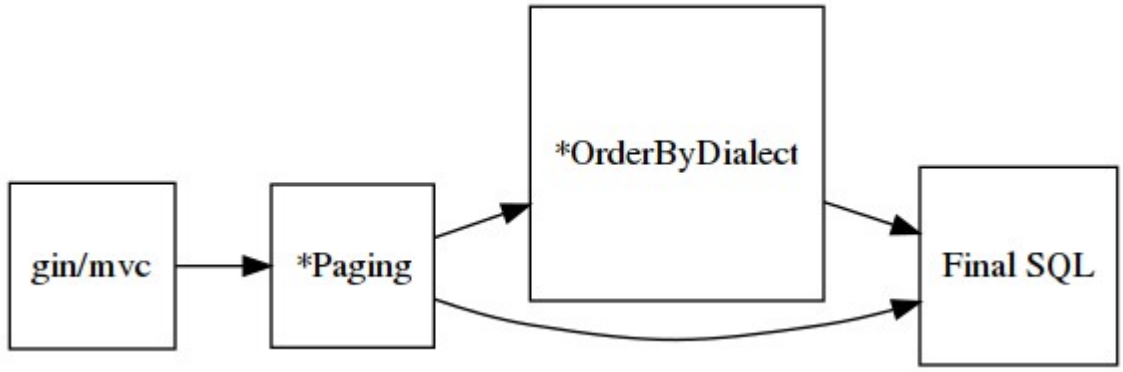
`Sqlx`

見 `*DbController.InTx` function

`Gorm`

見 `*GormDbExt.InTx` function

分頁



可由 `github.com/Cepave/open-falcon-backend/common/gin` 提供的 function 來產生 paging 物件:

package: `github.com/Cepave/open-falcon-backend/common/model`

`Paging` 物件定義幾件事:

1. 每頁幾筆(輸入/輸出)
2. 第幾頁(輸入/輸出)
3. 總筆數(輸出)
4. 是否有下一頁(輸出)
5. 排序方式(輸入/輸出)

```
// The paging object used to hold information
type Paging struct {
    Size int32
    Position int32
    TotalCount int32
    PageMore bool
    OrderBy []*OrderByEntity
}
```

排序方式，兩個參數，屬性名稱與方向，由開發者定義

```
type OrderByEntity struct {
    // Could name of column, property or any user-defined text
    Expr string
    // See Asc/Desc constant
    Direction byte
}
```

排序方向，定義在 `github.com/Cepave/open-falcon-backend/common/utils` 內

```
const (
    DefaultDirection byte = 0
    // Sorting by ascending
    Ascending byte = 1
    // Sorting by descending
    Descending byte = 2
)
```

取分頁值

`*Paging.GetOffset() int32`
取得可用在 `MySql` 語法的值(`[LIMIT {[offset,] row_count | row_count OFFSET offset}]`)

`*Paging.SetTotalCount(int32)`
設定總筆數，會自動設定 `PageMore` 的值

`model.ExtractPage(interface{}, *Paging) interface{}`
: 從一個 array/slice，取得分頁

分頁轉為 SQL 語法

在 `db/model` 中的 `OrderByDialect` 提供了將 `*Paging` 物件轉為 SQL 語法的引擎:

```
// 以 Map 對應屬性與資料庫欄位
var orderByDialectForAgents = commonModel.NewSqlOrderByDialect(
    map[string]string{
        "id":                 "ag_id",
        "status":             "ag_status",
        "name":               "ag_name",
        "connection_id":      "ag_connection_id",
        "comment":            "ag_comment",
        "province":           "pv_name",
        "city":               "ct_name",
        "last_heartbeat_time": "ag_last_heartbeat",
        "name_tag":           "nt_value",
        "applied":            "applying_ping_task",
    },
)

// 產生 a_column DESC, b_column ASC 語法
sortingSyntax, err :=
orderByDialectForAgents.ToQuerySyntax(paging.OrderBy)
```

若需要複雜語法，可覆寫 `OrderByDialect.FuncEntityToSyntax` 來自定所需的語法

Sql Data serialization/de-serialization

一個 type 可實作下列 interface，用來支援自訂的 Go type to SQL type:

- Scanner - 把 SQL 資料轉為你的型別
- Valuer - 把你的型別轉為 SQL 儲存的型別

Last modified on 2017-04-26T10:43:06+08:00