

-UnmarshalValue

structgconvstructstructUnmarshalValuegconvstructstructgconvUnmarshalValue



UnmarshalText(text []byte) errorUnmarshalValue

```
// apiUnmarshalValue is the interface for custom defined types customizing
value assignment.
// Note that only pointer can implement interface apiUnmarshalValue.
type apiUnmarshalValue interface {
    UnmarshalValue(interface{}) error
}
```

UnmarshalValueinterface{}



UnmarshalValue(Receiver)

```
func (c *Receiver) UnmarshalValue(interface{}) error
```

```
func (c Receiver) UnmarshalValue(interface{}) error
```

Content Menu

•

- [1struct](#)
- [2TCP](#)

1struct

```
CREATE TABLE `user` (
  id bigint unsigned NOT NULL AUTO_INCREMENT,
  passport varchar(45),
  password char(32) NOT NULL,
  nickname varchar(45) NOT NULL,
  create_time timestamp NOT NULL,
  PRIMARY KEY (id)
) ;
```

```

package main

import (
    "fmt"
    "github.com/gogf/gf/v2/container/garray"
    "github.com/gogf/gf/v2/database/gdb"
    "github.com/gogf/gf/v2/errors/gerror"
    "github.com/gogf/gf/v2/frame/g"
    "github.com/gogf/gf/v2/os/gtime"
    "reflect"
)

type User struct {
    Id          int
    Passport    string
    Password    string
    Nickname    string
    CreateTime  *gtime.Time
}

// UnmarshalValue
func (user *User) UnmarshalValue(value interface{}) error {
    if record, ok := value.(gdb.Record); ok {
        *user = User{
            Id:          record["id"].Int(),
            Passport:    record["passport"].String(),
            Password:    "",
            Nickname:    record["nickname"].String(),
            CreateTime:  record["create_time"].GTime(),
        }
        return nil
    }
    return gerror.Newf(`unsupported value type for UnmarshalValue: %
v`, reflect.TypeOf(value))
}

func main() {
    var (
        err  error
        users []*User
    )
    array := garray.New(true)
    for i := 1; i <= 10; i++ {
        array.Append(g.Map{
            "id":          i,
            "passport":    fmt.Sprintf(`user_%d`, i),
            "password":    fmt.Sprintf(`pass_%d`, i),
            "nickname":    fmt.Sprintf(`name_%d`, i),
            "create_time": gtime.NewFromStr("2018-10-24 10:00:
00").String(),
        })
    }
    //
    _, err = g.Model("user").Data(array).Insert()
    if err != nil {
        panic(err)
    }
    //
    err = g.Model("user").Order("id asc").Scan(&users)
    if err != nil {
        panic(err)
    }
    g.Dump(users)
}

```

```
{
  Id:      1,
  Passport: "user_1",
  Password: "",
  Nickname: "name_1",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      2,
  Passport: "user_2",
  Password: "",
  Nickname: "name_2",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      3,
  Passport: "user_3",
  Password: "",
  Nickname: "name_3",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      4,
  Passport: "user_4",
  Password: "",
  Nickname: "name_4",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      5,
  Passport: "user_5",
  Password: "",
  Nickname: "name_5",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      6,
  Passport: "user_6",
  Password: "",
  Nickname: "name_6",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      7,
  Passport: "user_7",
  Password: "",
  Nickname: "name_7",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      8,
  Passport: "user_8",
  Password: "",
  Nickname: "name_8",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      9,
  Passport: "user_9",
  Password: "",
  Nickname: "name_9",
  CreateTime: "2018-10-24 10:00:00",
},
{
  Id:      10,
  Passport: "user_10",
  Password: "",
  Nickname: "name_10",
  CreateTime: "2018-10-24 10:00:00",
},
},
```



UnmarshalValue100WUnmarshalValue

2TCP

TCP

```

package main

import (
    "errors"
    "fmt"
    "github.com/gogf/gf/v2/crypto/gcrc32"
    "github.com/gogf/gf/v2/encoding/gbinary"
    "github.com/gogf/gf/v2/util/gconv"
)

type Pkg struct {
    Length uint16 // Total length.
    Crc32  uint32 // CRC32.
    Data   []byte
}

// NewPkg creates and returns a package with given data.
func NewPkg(data []byte) *Pkg {
    return &Pkg{
        Length: uint16(len(data) + 6),
        Crc32:  gcrc32.Encrypt(data),
        Data:   data,
    }
}

// Marshal encodes the protocol struct to bytes.
func (p *Pkg) Marshal() []byte {
    b := make([]byte, 6+len(p.Data))
    copy(b, gbinary.EncodeUint16(p.Length))
    copy(b[2:], gbinary.EncodeUint32(p.Crc32))
    copy(b[6:], p.Data)
    return b
}

// UnmarshalValue decodes bytes to protocol struct.
func (p *Pkg) UnmarshalValue(v interface{}) error {
    b := gconv.Bytes(v)
    if len(b) < 6 {
        return errors.New("invalid package length")
    }
    p.Length = gbinary.DecodeToUint16(b[:2])
    if len(b) < int(p.Length) {
        return errors.New("invalid data length")
    }
    p.Crc32 = gbinary.DecodeToUint32(b[2:6])
    p.Data = b[6:]
    if gcrc32.Encrypt(p.Data) != p.Crc32 {
        return errors.New("crc32 validation failed")
    }
    return nil
}

func main() {
    var p1, p2 *Pkg

    // Create a demo pkg as p1.
    p1 = NewPkg([]byte("123"))
    fmt.Println(p1)

    // Convert bytes from p1 to p2 using gconv.Struct.
    err := gconv.Struct(p1.Marshal(), &p2)
    if err != nil {
        panic(err)
    }
    fmt.Println(p2)
}

```

&{9 2286445522 [49 50 51]}

&{9 2286445522 [49 50 51]}