-

Content Menu


```
// RuleFuncInput holds the input parameters that passed to custom rule
function RuleFunc.
type RuleFuncInput struct {
        // Rule specifies the validation rule string, like "required",
"between:1,100", etc.
        Rule string

        // Message specifies the custom error message or configured i18n
message for this rule.
        Message string

        // Value specifies the value for this rule to validate.
        Value *gvar.Var

        // Data specifies the `data` which is passed to the Validator. It
might be a type of map/struct or a nil value.
        // You can ignore the parameter `Data` if you do not really need
it in your custom validation rule.
        Data *gvar.Var
}

// RuleFunc is the custom function for data validation.
type RuleFunc func(ctx context.Context, in RuleFuncInput) error
```

1. ctx
2. RuleFuncInput
   - Rulerequired, between:1,100, length:6
   - Message
   - Value*gvar.Var
   - Datamapstructnil

i18n gf.gvalid.rule. i18ni18nMessage

```
// RegisterRule registers custom validation rule and function for package.
func RegisterRule(rule string, f RuleFunc) {
        customRuleFuncMap[rule] = f
}

// RegisterRuleByMap registers custom validation rules using map for
package.
func RegisterRuleByMap(m map[string]RuleFunc) {
        for k, v := range m {
                customRuleFuncMap[k] = v
        }
}
```

RuleFuncRegisterRulegvalidnilerror

⚠️ boot

1ID

IDorder-exist

```
package main

import (
        "context"
        "fmt"
        "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/gctx"
        "github.com/gogf/gf/v2/util/gvalid"
        "time"
)

type Request struct {
        OrderId     int64 `v:"required|order-exist"`
    ProductName string
        Amount      int64
        // ...
}

func init() {
        rule := "order-exist"
        gvalid.RegisterRule(rule, RuleOrderExist)
}

func RuleOrderExist(ctx context.Context, in gvalid.RuleFuncInput) error {
        // SELECT COUNT(*) FROM `order` WHERE `id` = xxx
        count, err := g.Model("order").
                Ctx(ctx).
                Cache(gdb.CacheOption{
                        Duration: time.Hour,
                        Name:     "",
                        Force:    false,
                }).
                WhereNot("id", in.Value.Int64()).
                Count()
        if err != nil {
                return err
        }
        if count == 0 {
                return gerror.Newf(`invalid order id "%d"`, in.Value.
Int64())
        }
        return nil
}

func main() {
        var (
                ctx = gctx.New()
                req = &Request{
                        OrderId:     65535,
                        ProductName: "HikingShoe",
                        Amount:      10000,
                }
        )
        err := g.Validator().CheckStruct(ctx, req)
        fmt.Println(err)
}
```

2

/unique-name

```
package main

import (
	"context"
	"fmt"
	"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/gctx"
	"github.com/gogf/gf/v2/util/gvalid"
	"time"
)

type User struct {
	Id   int
	Name string `v:"required|unique-name#|"`
	Pass string `v:"required|length:6,18"`
}

func init() {
	rule := "unique-name"
	gvalid.RegisterRule(rule, RuleUniqueName)
}

func RuleUniqueName(ctx context.Context, in gvalid.RuleFuncInput) error {
	var user *User
	if err := in.Data.Scan(&user); err != nil {
		return gerror.Wrap(err, `Scan data to user failed`)
	}
	// SELECT COUNT(*) FROM `user` WHERE `id` != xxx AND `name` != xxx
	count, err := g.Model("user").
		Ctx(ctx).
		Cache(gdb.CacheOption{
			Duration: time.Hour,
			Name:     "",
			Force:    false,
		}).
		WhereNot("id", user.Id).
		WhereNot("name", user.Name).
		Count()
	if err != nil {
		return err
	}
	if count > 0 {
		if in.Message != "" {
			return gerror.New(in.Message)
		}
		return gerror.Newf(`user name "%s" is already token by
others`, user.Name)
	}
	return nil
}

func main() {
	var (
		ctx  = gctx.New()
		user = &User{
			Id:   1,
			Name: "john",
			Pass: "123456",
		}
	)
	err := g.Validator().CheckStruct(ctx, user)
	fmt.Println(err)
}
```

```
// RuleFunc registers one custom rule function to current Validator.
func (v *Validator) RuleFunc(rule string, f RuleFunc) *Validator

// RuleFuncMap registers multiple custom rule functions to current
Validator.
func (v *Validator) RuleFuncMap(m map[string]RuleFunc) *Validator
```

- RuleFunc
- RuleFuncMap

```
package main

import (
        "context"
        "fmt"
        "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/gctx"
        "github.com/gogf/gf/v2/util/gvalid"
        "time"
)

type Request struct {
        OrderId     int64
        ProductName string
        Amount      int64
        // ...
}

func RuleOrderExist(ctx context.Context, in gvalid.RuleFuncInput) error {
        // SELECT COUNT(*) FROM `order` WHERE `id` = xxx
        count, err := g.Model("order").
                Ctx(ctx).
                Cache(gdb.CacheOption{
                        Duration: time.Hour,
                        Name:     "",
                        Force:    false,
                }).
                WhereNot("id", in.Value.Int64()).
                Count()
        if err != nil {
                return err
        }
        if count == 0 {
                return gerror.Newf(`invalid order id "%d"`, in.Value.
Int64())
        }
        return nil
}

func main() {
        var (
                ctx = gctx.New()
                req = &Request{
                        OrderId:     65535,
                        ProductName: "HikingShoe",
                        Amount:      10000,
                }
        )
        err := g.Validator().RuleFunc("order-exist", RuleOrderExist).Data
(req).Run(ctx)
        fmt.Println(err)
}
```