-

https://pkg.go.dev/github.com/gogf/gf/v2/util/gvalid

Content Menu

- New
- Run
- Clone
- I18n
- Bail
- Ci
- Data
- Assoc
- Rules
- Message
- RuleFunc
- RuleFuncMap

New

- NewValidator
-

```
New() *Validator
```

-

```
func ExampleNew() {
        validator := gvalid.New()

        if err := validator.Data(16).Rules("min:18").Run(context.
Background()); err != nil {
                fmt.Print(err)
        }

        // Output:
        // The value `16` must be equal or greater than 18
}
```

Run

- Run
-

```
Run(ctx context.Context) Error
```

-

```
func ExampleValidator_Run() {
	// check value mode
	if err := g.Validator().Data(16).Rules("min:18").Run(context.
Background()); err != nil {
		fmt.Println("check value err:", err)
	}
	// check map mode
	data := map[string]interface{}{
		"passport":  "",
		"password":  "123456",
		"password2": "1234567",
	}
	rules := map[string]string{
		"passport":  "required|length:6,16",
		"password":  "required|length:6,16|same:password2",
		"password2": "required|length:6,16",
	}
	if err := g.Validator().Data(data).Rules(rules).Run(context.
Background()); err != nil {
		fmt.Println("check map err:", err)
	}
	// check struct mode
	type Params struct {
		Page      int    `v:"required|min:1"`
		Size      int    `v:"required|between:1,100"`
		ProjectId string `v:"between:1,10000"`
	}
	rules = map[string]string{
		"Page":      "required|min:1",
		"Size":      "required|between:1,100",
		"ProjectId": "between:1,10000",
	}
	obj := &Params{
		Page: 0,
		Size: 101,
	}
	if err := g.Validator().Data(obj).Run(context.Background());
err != nil {
		fmt.Println("check struct err:", err)
	}

	// May Output:
	// check value err: The value `16` must be equal or greater
than 18
	// check map err: The passport field is required; The
passport value `` length must be between 6 and 16; The password
value `123456` must be the same as field password2
	// check struct err: The Page value `0` must be equal or
greater than 1; The Size value `101` must be between 1 and 100
}
```

Clone

- CloneValidator
-

```
(v *Validator) Clone() *Validator
```

-

```
func ExampleValidator_Clone() {
        if err := g.Validator().Data(16).Rules("min:18").Run(context.
Background()); err != nil {
                fmt.Println(err)
        }

        if err := g.Validator().Clone().Data(20).Run(context.
Background()); err != nil {
                fmt.Println(err)
        } else {
                fmt.Println("Check Success!")
        }

        // Output:
        // The value `16` must be equal or greater than 18
        // Check Success!
}
```

I18n

- I18nI18Ni18n
-

```
I18n(i18nManager *gi18n.Manager) *Validator
```

-

```
func ExampleValidator_I18n() {
        var (
                i18nManager = gi18n.New()
                ctxCn       = gi18n.WithLanguage(context.
Background(), "cn")
                validator   = gvalid.New()
        )

        validator = validator.Data(16).Rules("min:18")

        if err := validator.Run(context.Background()); err != nil {
                fmt.Println(err)
        }

        if err := validator.I18n(i18nManager).Run(ctxCn); err != nil
{
                fmt.Println(err)
        }

        // Output:
        // The value `16` must be equal or greater than 18
        // `16`18
}
```

Bail

- Bail
-

```
Bail() *Validator
```

-

```
func ExampleValidator_Bail() {
        type BizReq struct {
                Account   string `v:"required|length:6,16|same:QQ"`
                QQ        string
                Password  string `v:"required|same:Password2"`
                Password2 string `v:"required"`
        }
        var (
                ctx = context.Background()
                req = BizReq{
                        Account:   "gf",
                        QQ:        "123456",
                        Password:  "goframe.org",
                        Password2: "goframe.org",
                }
        )

        if err := g.Validator().Bail().Data(req).Run(ctx); err !=
nil {
                fmt.Println("Use Bail Error:", err)
        }

        if err := g.Validator().Data(req).Run(ctx); err != nil {
                fmt.Println("Not Use Bail Error:", err)
        }

        // output:
        // Use Bail Error: The Account value `gf` length must be
between 6 and 16
        // Not Use Bail Error: The Account value `gf` length must be
between 6 and 16; The Account value `gf` must be the same as field QQ
}
```

Ci

- Ci
-

```
Ci() *Validator
```

-

```
func ExampleValidator_Ci() {

        type BizReq struct {
                Account   string `v:"required"`
                Password  string `v:"required|same:Password2"`
                Password2 string `v:"required"`
        }

        var (
                ctx = context.Background()
                req = BizReq{
                        Account:   "gf",
                        Password:  "Goframe.org", // Diff from
Password2, but because of "ci", rule check passed
                        Password2: "goframe.org",
                }
        )

        if err := g.Validator().Data(req).Run(ctx); err != nil {
                fmt.Println("Not Use CI Error:", err)
        }

        if err := g.Validator().Ci().Data(req).Run(ctx); err == nil {
                fmt.Println("Use CI Passed!")
        }

        // output:
        // Not Use CI Error: The Password value `Goframe.org` must
be the same as field Password2
        // Use CI Passed!
}
```

Data

- Data
-

```
Data(data interface{}) *Validator
```

-

```
func ExampleValidator_Data() {
        type BizReq struct {
                Password1 string `v:"password"`
                Password2 string `v:"password"`
        }

        var (
                ctx = context.Background()
                req = BizReq{
                        Password1: "goframe",
                        Password2: "gofra", // error length between
6 and 18
                }
        )
        if err := g.Validator().Data(req).Run(ctx); err != nil {
                fmt.Print(err)
        }

        // Output:
        // The Password2 value `gofra` is not a valid password format
}
```

Assoc

- AssocValidatorassocmapunion validatormap
- nilassocuseDataInsteadOfObjectAttributestrue
-

```
Assoc(assoc interface{}) *Validator
```

-

```
func ExampleValidator_Assoc() {

        type User struct {
                Name string `v:"required"`
                Type int    `v:"required"`
        }

        data := g.Map{
                "name": "john",
        }

        user := User{}

        if err := gconv.Scan(data, &user); err != nil {
                panic(err)
        }

        if err := g.Validator().Data(user).Assoc(data).Run(context.
Background()); err != nil {
                fmt.Print(err)
        }

        // Output:
        // The Type field is required
}
```

Rules

- Rules
-

```
Rules(rules interface{}) *Validator
```

-

```
func ExampleValidator_Rules() {

        if err := g.Validator().Data(16).Rules("min:18").Run(context.
Background()); err != nil {
                fmt.Println(err)
        }

        // Output:
        // The value `16` must be equal or greater than 18
}
```

Message

- Messages
-

```
Messages(messages interface{}) *Validator
```

-

```
func ExampleValidator_Messages() {
        if err := g.Validator().Data(16).Rules("min:18").Messages
("Can not regist, Age is less then 18!").Run(context.Background());
err != nil {
                fmt.Println(err)
        }

        // Output:
        // Can not regist, Age is less then 18!
}
```

RuleFunc

- RuleFuncValidator
-

```
RuleFunc(rule string, f RuleFunc) *Validator
```

-

```
func ExampleValidator_RuleFunc() {
        var (
                ctx             = context.Background()
                lenErrRuleName  = "LenErr"
                passErrRuleName = "PassErr"
                lenErrRuleFunc  = func(ctx context.Context, in
gvalid.RuleFuncInput) error {
                        pass := in.Value.String()
                        if len(pass) != 6 {
                                return errors.New(in.Message)
                        }
                        return nil
                }
                passErrRuleFunc = func(ctx context.Context, in
gvalid.RuleFuncInput) error {
                        pass := in.Value.String()
                        if m := in.Data.Map(); m["data"] != pass {
                                return errors.New(in.Message)
                        }
                        return nil
                }
        )

        type LenErrStruct struct {
                Value string `v:"uid@LenErr#Value Length Error!"`
                Data  string `p:"data"`
        }

        st := &LenErrStruct{
                Value: "123",
                Data:  "123456",
        }
        // single error sample
        if err := g.Validator().RuleFunc(lenErrRuleName,
lenErrRuleFunc).Data(st).Run(ctx); err != nil {
                fmt.Println(err)
        }

        type MultiErrorStruct struct {
                Value string `v:"uid@LenErr|PassErr#Value Length
Error!|Pass is not Same!"`
                Data  string `p:"data"`
        }

        multi := &MultiErrorStruct{
                Value: "123",
                Data:  "123456",
        }
        // multi error sample
        if err := g.Validator().RuleFunc(lenErrRuleName,
lenErrRuleFunc).RuleFunc(passErrRuleName, passErrRuleFunc).Data
(multi).Run(ctx); err != nil {
                fmt.Println(err)
        }

        // Output:
        // Value Length Error!
        // Value Length Error!; Pass is not Same!
}
```

RuleFuncMap

- RuleFuncMapValidator
-

```
RuleFuncMap(m map[string]RuleFunc) *Validator
```

-

```
func ExampleValidator_RuleFuncMap() {
        var (
                ctx             = context.Background()
                lenErrRuleName  = "LenErr"
                passErrRuleName = "PassErr"
                lenErrRuleFunc  = func(ctx context.Context, in
gvalid.RuleFuncInput) error {
                        pass := in.Value.String()
                        if len(pass) != 6 {
                                return errors.New(in.Message)
                        }
                        return nil
                }
                passErrRuleFunc = func(ctx context.Context, in
gvalid.RuleFuncInput) error {
                        pass := in.Value.String()
                        if m := in.Data.Map(); m["data"] != pass {
                                return errors.New(in.Message)
                        }
                        return nil
                }
                ruleMap = map[string]gvalid.RuleFunc{
                        lenErrRuleName:  lenErrRuleFunc,
                        passErrRuleName: passErrRuleFunc,
                }
        )

        type MultiErrorStruct struct {
                Value string `v:"uid@LenErr|PassErr#Value Length
Error!|Pass is not Same!"`
                Data  string `p:"data"`
        }

        multi := &MultiErrorStruct{
                Value: "123",
                Data:  "123456",
        }

        if err := g.Validator().RuleFuncMap(ruleMap).Data(multi).Run
(ctx); err != nil {
                fmt.Println(err)
        }

        // Output:
        // Value Length Error!; Pass is not Same!
}
```