

GoFrame



vvalidation

ci

- ci
- ci(Case Insensitive)same, different, in, not-in
-

```
func Example_Rule_CaseInsensitive() {  
    type BizReq struct {  
        Account    string `v:"required"`  
        Password    string `v:"required|ci|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(err)  
    }  
  
    // output:  
}
```

bail



required*bailrequired*bailrequired*

- bail
-
- HTTP ServerbailReqbail
-

Content Menu

- - ci
 - bail
 - foreach
- - required
 - required-if
 - required-unless
 - required-with
 - required-with-all
 - required-without
 - required-without-all
 - date
 - datetime
 - date-format
 - before
 - before-equal
 - after
 - after-equal
 - array
 - enums
 - email
 - phone
 - phone-loose
 - telephone
 - passport
 - password
 - password2
 - password3
 - postcode
 - resident-id
 - bank-card
 - qq
 - ip
 - ipv4
 - ipv6
 - mac
 - url
 - domain
 - size
 - length
 - min-length
 - max-length
 - between
 - min
 - max
 - json
 - integer
 - float
 - boolean
 - same
 - different
 - eq
 - not-eq
 - gt
 - gte
 - lt
 - lte
 - in
 - not-in
 - regex
 - not-regex

```

func Example_Rule_Bail() {
    type BizReq struct {
        Account string `v:"bail|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().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // output:
    // The Account value `gf` length must be between 6 and 16
}

```

foreach

- foreach
-
- >=v2.2.0
-

```

func Example_Rule_Foreach() {
    type BizReq struct {
        Value1 []int `v:"foreach|in:1,2,3"``
        Value2 []int `v:"foreach|in:1,2,3"``
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Value1: []int{1, 2, 3},
            Value2: []int{3, 4, 5},
        }
    )
    if err := g.Validator().Bail().Data(req).Run(ctx); err !=
nil {
        fmt.Println(err.String())
    }

    // Output:
    // The Value2 value `4` is not in acceptable range: 1,2,3
}

```

required

- :required
- Slice/Map
- Name

```

func Example_Rule_Required() {
    type BizReq struct {
        ID    uint   `v:"required"`
        Name  string `v:"required"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            ID: 1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The Name field is required
}

```

required-if

- :required-if:field,value,...
- (fieldvalue),
- Gender1WifeName Gender2HusbandName

```

func Example_Rule_RequiredIf() {
    type BizReq struct {
        ID    uint   `v:"required" dc:"Your ID"`
        Name  string `v:"required" dc:"Your name"`
        Gender uint   `v:"in:0,1,2" dc:"0:Secret;1:Male;2:Female"`
        WifeName string `v:"required-if:gender,1"`
        HusbandName string `v:"required-if:gender,2"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            ID:    1,
            Name: "test",
            Gender: 1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The WifeName field is required
}

```

required-unless

- :required-unless:field,value,...
- (fieldvalue),
- Gender0Gender2WifeNameId 0 Gender 2 HusbandName

```

func Example_Rule_RequiredUnless() {
    type BizReq struct {
        ID          uint   `v:"required" dc:"Your ID"`
        Name         string `v:"required" dc:"Your name"`
        Gender       uint   `v:"in:0,1,2" dc:"0:Secret;1:Male;
2:Female"`
        WifeName     string `v:"required-unless:gender,0,
gender,2"`
        HusbandName string `v:"required-unless:id,0,gender,2"
        `
    }

    var (
        ctx = context.Background()
        req = BizReq{
            ID:      1,
            Name:    "test",
            Gender:  1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The WifeName field is required; The HusbandName field is
    // required
}

```

required-with

- :required-with:field1,field2,...
- 0
- WifeNameHusbandName

```

func Example_Rule_RequiredWith() {
    type BizReq struct {
        ID          uint   `v:"required" dc:"Your ID"`
        Name         string `v:"required" dc:"Your name"`
        Gender       uint   `v:"in:0,1,2" dc:"0:Secret;1:Male;
2:Female"`
        WifeName     string
        HusbandName string `v:"required-with:WifeName"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            ID:      1,
            Name:    "test",
            Gender:  1,
            WifeName: "Ann",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The HusbandName field is required
}

```

required-with-all

- :required-with-all:field1,field2,...
- 0
- IdNameGenderWifeNameHusbandName

```

func Example_Rule_RequiredWithAll() {
    type BizReq struct {
        ID      uint   `v:"required" dc:"Your ID"`
        Name     string `v:"required" dc:"Your name"`
        Gender   uint   `v:"in:0,1,2" dc:"0:Secret;1:Male;
2:Female"`
        WifeName string
        HusbandName string `v:"required-with-all:Id,Name,
Gender,WifeName"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            ID:      1,
            Name:     "test",
            Gender:   1,
            WifeName: "Ann",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The HusbandName field is required
}

```

required-without

- :required-without:field1,field2,...
- 0
- IdWifeNameHusbandName

```

func Example_Rule_RequiredWithout() {
    type BizReq struct {
        ID      uint   `v:"required" dc:"Your ID"`
        Name     string `v:"required" dc:"Your name"`
        Gender   uint   `v:"in:0,1,2" dc:"0:Secret;1:Male;
2:Female"`
        WifeName string
        HusbandName string `v:"required-without:Id,WifeName"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            ID:      1,
            Name:     "test",
            Gender:   1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The HusbandName field is required
}

```

required-without-all

- :required-without-all:field1,field2,...
- 0
- IdWifeNameHusbandName

```

func Example_Rule_RequiredWithoutAll() {
    type BizReq struct {
        ID          uint   `v:"required" dc:"Your ID"`
        Name         string `v:"required" dc:"Your name"`
        Gender       uint   `v:"in:0,1,2" dc:"0:Secret;1:Male;
2:Female"`
        WifeName     string
        HusbandName string `v:"required-without-all:Id,
WifeName"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Name: "test",
            Gender: 1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The HusbandName field is required
}

```

date

- : date
- -/.8 2006-01-02, 2006/01/02, 2006.01.02, 20060102
-

```

func Example_Rule_Date() {
    type BizReq struct {
        Date1 string `v:"date"`
        Date2 string `v:"date"`
        Date3 string `v:"date"`
        Date4 string `v:"date"`
        Date5 string `v:"date"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Date1: "2021-10-31",
            Date2: "2021.10.31",
            Date3: "2021-Oct-31",
            Date4: "2021 Octa 31",
            Date5: "2021/Oct/31",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Date3 value `2021-Oct-31` is not a valid date
    // The Date4 value `2021 Octa 31` is not a valid date
    // The Date5 value `2021/Oct/31` is not a valid date
}

```

datetime

- : datetime

- - 2006-01-02 12:00:00

•

```
func Example_Rule_Datetime() {
    type BizReq struct {
        Date1 string `v:"datetime"`
        Date2 string `v:"datetime"`
        Date3 string `v:"datetime"`
        Date4 string `v:"datetime"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Date1: "2021-11-01 23:00:00",
            Date2: "2021-11-01 23:00", // error
            Date3: "2021/11/01 23:00:00", // error
            Date4: "2021/Dec/01 23:00:00", // error
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Date2 value `2021-11-01 23:00` is not a valid datetime
    // The Date3 value `2021/11/01 23:00:00` is not a valid
datetime
    // The Date4 value `2021/Dec/01 23:00:00` is not a valid
datetime
}

```

date-format

- :date-format:format
- /formatgtime()[gtime](#)
- date-format:Y-m-d H:i:s

```
func Example_Rule_DateFormat() {
    type BizReq struct {
        Date1 string `v:"date-format:Y-m-d"`
        Date2 string `v:"date-format:Y-m-d"`
        Date3 string `v:"date-format:Y-m-d H:i:s"`
        Date4 string `v:"date-format:Y-m-d H:i:s"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Date1: "2021-11-01",
            Date2: "2021-11-01 23:00", // error
            Date3: "2021-11-01 23:00:00",
            Date4: "2021-11-01 23:00", // error
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Date2 value `2021-11-01 23:00` does not match the
format: Y-m-d
    // The Date4 value `2021-11-01 23:00` does not match the
format: Y-m-d H:i:s
}

```

before

- :before:field
- //
- >=v2.2.0

```
func Example_Rule_Before() {
    type BizReq struct {
        Time1 string `v:"before:Time3"`
        Time2 string `v:"before:Time3"`
        Time3 string
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Time1: "2022-09-02",
            Time2: "2022-09-03",
            Time3: "2022-09-03",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err.String())
    }

    // Output:
    // The Time2 value `2022-09-03` must be before field Time3
    value `2022-09-03`
}
```

before-equal

- :before-equal:field
- ///
- >=v2.2.0

```
func Example_Rule_BeforeEqual() {
    type BizReq struct {
        Time1 string `v:"before-equal:Time3"`
        Time2 string `v:"before-equal:Time3"`
        Time3 string
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Time1: "2022-09-02",
            Time2: "2022-09-01",
            Time3: "2022-09-01",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Time1 value `2022-09-02` must be before or equal to
    field Time3
}
```

after

- :after:field
- //
- >=v2.2.0

```

func Example_Rule_After() {
    type BizReq struct {
        Time1 string
        Time2 string `v:"after:Time1"`
        Time3 string `v:"after:Time1"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Time1: "2022-09-01",
            Time2: "2022-09-01",
            Time3: "2022-09-02",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err.String())
    }

    // Output:
    // The Time2 value `2022-09-01` must be after field Time1
    value `2022-09-01`
}

```

after-equal

- `:after-equal:field`
- `///`
- `>=v2.2.0`

```

func Example_Rule_AfterEqual() {
    type BizReq struct {
        Time1 string
        Time2 string `v:"after-equal:Time1"`
        Time3 string `v:"after-equal:Time1"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Time1: "2022-09-02",
            Time2: "2022-09-01",
            Time3: "2022-09-02",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Time2 value `2022-09-01` must be after or equal to
    field Time1 value `2022-09-02`
}

```

array

- `:array`
- `JSON`
- `>=v2.2.0`

```

func Example_Rule_Array() {
    type BizReq struct {
        Value1 string `v:"array"`
        Value2 string `v:"array"`
        Value3 string `v:"array"`
        Value4 []string `v:"array"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Value1: "1,2,3",
            Value2: "[]",
            Value3: "[1,2,3]",
            Value4: []string{},
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Value1 value `1,2,3` is not of valid array type
}

```

enums

- enums
- gf gen enums [-gen enums](#)

```

func ExampleRule_Enums() {
    type Status string
    const (
        StatusRunning Status = "Running"
        StatusOffline Status = "Offline"
    )
    type BizReq struct {
        Id      int    `v:"required"`
        Name    string `v:"required"`
        Status  Status `v:"enums"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Id:      1,
            Name:    "john",
            Status:  Status("Pending"),
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // May Output:
    // The Status value `Pending` should be in enums of:
    ["Running", "Offline"]
}

```

email

- email
- EMAIL

```

func Example_Rule_Email() {
    type BizReq struct {
        MailAddr1 string `v:"email"`
        MailAddr2 string `v:"email"`
        MailAddr3 string `v:"email"`
        MailAddr4 string `v:"email"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            MailAddr1: "gf@goframe.org",
            MailAddr2: "gf@goframe", // error
            MailAddr3: "gf@goframe.org.cn",
            MailAddr4: "gf#goframe.org", // error
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The MailAddr2 value `gf@goframe` is not a valid email
address
    // The MailAddr4 value `gf#goframe.org` is not a valid email
address
}

```

phone

- phone

```

func Example_Rule_Phone() {
    type BizReq struct {
        PhoneNumber1 string `v:"phone"`
        PhoneNumber2 string `v:"phone"`
        PhoneNumber3 string `v:"phone"`
        PhoneNumber4 string `v:"phone"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            PhoneNumber1: "13578912345",
            PhoneNumber2: "11578912345", // error 11x
not exist
            PhoneNumber3: "17178912345", // error 171
not exit
            PhoneNumber4: "1357891234", // error len
must be 11
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The PhoneNumber2 value `11578912345` is not a valid phone
number
    // The PhoneNumber3 value `17178912345` is not a valid phone
number
    // The PhoneNumber4 value `1357891234` is not a valid phone
number
}

```

phone-loose

- : phone
- 1314151617181911

```
func Example_Rule_PhoneLoose() {
    type BizReq struct {
        PhoneNumber1 string `v:"phone-loose"`
        PhoneNumber2 string `v:"phone-loose"`
        PhoneNumber3 string `v:"phone-loose"`
        PhoneNumber4 string `v:"phone-loose"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            PhoneNumber1: "13578912345",
            PhoneNumber2: "11578912345", // error 11x
            PhoneNumber3: "17178912345",
            PhoneNumber4: "1357891234", // error len
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The PhoneNumber2 value `11578912345` is invalid
    // The PhoneNumber4 value `1357891234` is invalid
}
```

telephone

- : telephone
- "XXXX-XXXXXXX"XXXX-XXXXXXX"XXX-XXXXXXX"XXX-XXXXXXX"XXXXXXX"XXXXXXX"

```

func Example_Rule_Telephone() {
    type BizReq struct {
        Telephone1 string `v:"telephone"`
        Telephone2 string `v:"telephone"`
        Telephone3 string `v:"telephone"`
        Telephone4 string `v:"telephone"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Telephone1: "010-77542145",
            Telephone2: "0571-77542145",
            Telephone3: "20-77542145", // error
            Telephone4: "775421451",  // error len must
be 7 or 8
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Telephone3 value `20-77542145` is not a valid
telephone number
    // The Telephone4 value `775421451` is not a valid telephone
number
}

```

passport

- :passport
- 6-18

```

func Example_Rule_Passport() {
    type BizReq struct {
        Passport1 string `v:"passport"`
        Passport2 string `v:"passport"`
        Passport3 string `v:"passport"`
        Passport4 string `v:"passport"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Passport1: "goframe",
            Passport2: "1356666", // error starting
with letter
            Passport3: "goframe#", // error containing
only numbers or underscores
            Passport4: "gf",      // error length
between 6 and 18
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Passport2 value `1356666` is not a valid passport
format
    // The Passport3 value `goframe#` is not a valid passport
format
    // The Passport4 value `gf` is not a valid passport format
}

```

password

- :password
- 6~18

```
func Example_Rule_Password() {
    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
}
```

password2

- :password2

```
func Example_Rule_Password2() {
    type BizReq struct {
        Password1 string `v:"password2"`
        Password2 string `v:"password2"`
        Password3 string `v:"password2"`
        Password4 string `v:"password2"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Password1: "Goframe123",
            Password2: "gofra",      // error length
between 6 and 18
            Password3: "Goframe", // error must
contain lower and upper letters and numbers.
            Password4: "goframe123", // error must
contain lower and upper letters and numbers.
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Password2 value `gofra` is not a valid password format
    // The Password3 value `Goframe` is not a valid password
format
    // The Password4 value `goframe123` is not a valid password
format
}
```

password3

- : password3
-

```
func Example_Rule_Password3() {
    type BizReq struct {
        Password1 string `v:"password3"`
        Password2 string `v:"password3"`
        Password3 string `v:"password3"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Password1: "Goframe123#",
            Password2: "gofra",          // error length
            Password3: "Goframe123", // error must
                                // contain lower and upper letters, numbers and special chars.
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Password2 value `gofra` is not a valid password format
    // The Password3 value `Goframe123` is not a valid password
    format
}
```

postcode

- : postcode
-

```
func Example_Rule_Postcode() {
    type BizReq struct {
        Postcode1 string `v:"postcode"`
        Postcode2 string `v:"postcode"`
        Postcode3 string `v:"postcode"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Postcode1: "100000",
            Postcode2: "10000",          // error length must
            Postcode3: "1000000", // error length must
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Postcode2 value `10000` is not a valid postcode format
    // The Postcode3 value `1000000` is not a valid postcode
    format
}
```

resident-id

- : resident-id
-

```

func Example_Rule_ResidentId() {
    type BizReq struct {
        ResidentID1 string `v:"resident-id"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            ResidentID1: "320107199506285482",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The ResidentID1 value `320107199506285482` is not a valid
    resident id number
}

```

bank-card

- : bank-card
-

```

func Example_Rule_BankCard() {
    type BizReq struct {
        BankCard1 string `v:"bank-card"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            BankCard1: "6225760079930218",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The BankCard1 value `6225760079930218` is not a valid
    bank card number
}

```

qq

- : qq
- QQ

```

func Example_Rule_QQ() {
    type BizReq struct {
        QQ1 string `v:"qq"`
        QQ2 string `v:"qq"`
        QQ3 string `v:"qq"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            QQ1: "389961817",
            QQ2: "9999",        // error >= 10000
            QQ3: "514258412a", // error all number
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The QQ2 value `9999` is not a valid QQ number
    // The QQ3 value `514258412a` is not a valid QQ number
}

```

ip

- :ip
- IPv4/IPv6

```

func Example_Rule_IP() {
    type BizReq struct {
        IP1 string `v:"ip"`
        IP2 string `v:"ip"`
        IP3 string `v:"ip"`
        IP4 string `v:"ip"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            IP1: "127.0.0.1",
            IP2: "fe80::812b:1158:1f43:f0d1",
            IP3: "520.255.255.255", // error >= 10000
            IP4: "ze80::812b:1158:1f43:f0d1",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The IP3 value `520.255.255.255` is not a valid IP address
    // The IP4 value `ze80::812b:1158:1f43:f0d1` is not a valid
    IP address
}

```

ipv4

- :ipv4
- IPv4

```

func Example_Rule_IPV4() {
    type BizReq struct {
        IP1 string `v:"ipv4"`
        IP2 string `v:"ipv4"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            IP1: "127.0.0.1",
            IP2: "520.255.255.255",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The IP2 value `520.255.255.255` is not a valid IPv4
    address
}

```

ipv6

- :ipv6
- IPv6

```

func Example_Rule_IPV6() {
    type BizReq struct {
        IP1 string `v:"ipv6"`
        IP2 string `v:"ipv6"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            IP1: "fe80::812b:1158:1f43:f0d1",
            IP2: "ze80::812b:1158:1f43:f0d1",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The IP2 value `ze80::812b:1158:1f43:f0d1` is not a valid
    IPv6 address
}

```

mac

- :mac
- MAC

```

func Example_Rule_Mac() {
    type BizReq struct {
        Mac1 string `v:"mac"`
        Mac2 string `v:"mac"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Mac1: "4C-CC-6A-D6-B1-1A",
            Mac2: "Z0-CC-6A-D6-B1-1A",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The Mac2 value `Z0-CC-6A-D6-B1-1A` is not a valid MAC
    address
}

```

url

- :url
- URL
- httphttpsftpfile

```

func Example_Rule_Url() {
    type BizReq struct {
        URL1 string `v:"url"`
        URL2 string `v:"url"`
        URL3 string `v:"url"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            URL1: "http://goframe.org",
            URL2: "ftp://goframe.org",
            URL3: "ws://goframe.org",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The URL3 value `ws://goframe.org` is not a valid URL
    address
}

```

domain

- :domain
-
- xxx.yyy

```

func Example_Rule_Domain() {
    type BizReq struct {
        Domain1 string `v:"domain"`
        Domain2 string `v:"domain"`
        Domain3 string `v:"domain"`
        Domain4 string `v:"domain"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Domain1: "goframe.org",
            Domain2: "a.b",
            Domain3: "goframe#org",
            Domain4: "1a.2b",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Domain3 value `goframe#org` is not a valid domain
format // The Domain4 value `1a.2b` is not a valid domain format
}

```

size

- :size:size
- **size** ()Unicode1

```

func Example_Rule_Size() {
    type BizReq struct {
        Size1 string `v:"size:10"`
        Size2 string `v:"size:5"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Size1: "goframe",
            Size2: "goframe",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The Size2 value `goframe` length must be 5
}

```

length

- :length:min,max
- minmax()Unicode1

```

func Example_Rule_Length() {
    type BizReq struct {
        Length1 string `v:"length:5,10"`
        Length2 string `v:"length:10,15"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Length1: "goframe",
            Length2: "goframe",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The Length2 value `goframe` length must be between 10 and
15
}

```

min-length

- :min-length:min
- min()Unicode1

```

func Example_Rule_MinLength() {
    type BizReq struct {
        MinLength1 string `v:"min-length:10"`
        MinLength2 string `v:"min-length:8"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            MinLength1: "goframe",
            MinLength2: "goframe",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The MinLength2 value `goframe` length must be equal or
greater than 8
}

```

max-length

- :max-length:max
- max()Unicode1

```

func Example_Rule_MaxLength() {
    type BizReq struct {
        MaxLength1 string `v:"max-length:10"`
        MaxLength2 string `v:"max-length:5"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            MaxLength1: "goframe",
            MaxLength2: "goframe",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The MaxLength2 value `goframe` length must be equal or
    // lesser than 5
}

```

between

- : between:min,max
- minmax()

```

func Example_Rule_Between() {
    type BizReq struct {
        Age1    int    `v:"between:1,100"`
        Age2    int    `v:"between:1,100"`
        Score1 float32 `v:"between:0,10"`
        Score2 float32 `v:"between:0,10"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Age1:    50,
            Age2:    101,
            Score1:  9.8,
            Score2: -0.5,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Age2 value `101` must be between 1 and 100
    // The Score2 value `-0.5` must be between 0 and 10
}

```

min

- : min:min
- min()

```

func Example_Rule_Min() {
    type BizReq struct {
        Age1    int    `v:"min:100"`
        Age2    int    `v:"min:100"`
        Score1 float32 `v:"min:10"`
        Score2 float32 `v:"min:10"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Age1:    50,
            Age2:    101,
            Score1: 9.8,
            Score2: 10.1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Age1 value `50` must be equal or greater than 100
    // The Score1 value `9.8` must be equal or greater than 10
}

```

max

- :max:max
- max()

```

func Example_Rule_Max() {
    type BizReq struct {
        Age1    int    `v:"max:100"`
        Age2    int    `v:"max:100"`
        Score1 float32 `v:"max:10"`
        Score2 float32 `v:"max:10"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Age1:    99,
            Age2:    101,
            Score1: 9.9,
            Score2: 10.1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Age2 value `101` must be equal or lesser than 100
    // The Score2 value `10.1` must be equal or lesser than 10
}

```

json

- :json
- JSON

```

func Example_Rule_Json() {
    type BizReq struct {
        JSON1 string `v:"json"`
        JSON2 string `v:"json"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            JSON1: "{\"name\":\"goframe\",\"author\":\""
            JSON2: "{\"name\":\"goframe\",\"author\":\""
            "\"}",
            "\",\"test\"}",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The JSON2 value `{"name":"goframe","author":"","test"}`
    is not a valid JSON string
}

```

integer

- :integer

```

func Example_Rule_Integer() {
    type BizReq struct {
        Integer string `v:"integer"`
        Float    string `v:"integer"`
        Str      string `v:"integer"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Integer: "100",
            Float:   "10.0",
            Str:     "goframe",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Float value `10.0` is not an integer
    // The Str value `goframe` is not an integer
}

```

float

- :float

```

func Example_Rule_Float() {
    type BizReq struct {
        Integer string `v:"float"`
        Float   string `v:"float"`
        Str      string `v:"float"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Integer: "100",
            Float:   "10.0",
            Str:     "goframe",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(err)
    }

    // Output:
    // The Str value `goframe` is invalid
}

```

boolean

- : boolean
- (1,true,on,yestruet | 0,false,off,no,""false)

```

func Example_Rule_Boolean() {
    type BizReq struct {
        Boolean bool    `v:"boolean"`
        Integer int     `v:"boolean"`
        Float   float32 `v:"boolean"`
        Str1    string   `v:"boolean"`
        Str2    string   `v:"boolean"`
        Str3    string   `v:"boolean"`
    }

    var (
        ctx = context.Background()
        req = BizReq{
            Boolean: true,
            Integer: 1,
            Float:   10.0,
            Str1:    "on",
            Str2:    "",
            Str3:    "goframe",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Float value `10` field must be true or false
    // The Str3 value `goframe` field must be true or false
}

```

same

- : same:field
- field
- PasswordPassword2

```

func Example_Rule_Same() {
    type BizReq struct {
        Name      string `v:"required"`
        Password   string `v:"required|same:Password2"`
        Password2  string `v:"required"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Name:      "gf",
            Password:   "goframe.org",
            Password2:  "goframe.net",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The Password value `goframe.org` must be the same as
    field Password2
}

```

different

- :different:field
- field
- OtherMailAddrMailAddr

```

func Example_Rule_Different() {
    type BizReq struct {
        Name      string `v:"required"`
        MailAddr   string `v:"required"`
        ConfirmMailAddr string `v:"required|different:MailAddr"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Name:      "gf",
            MailAddr:   "gf@goframe.org",
            ConfirmMailAddr: "gf@goframe.org",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The ConfirmMailAddr value `gf@goframe.org` must be
    different from field MailAddr
}

```

eq

- :eq:field
- fieldsamesame
- >=v2.2.0

```

func Example_Rule_EQ() {
    type BizReq struct {
        Name      string `v:"required"`
        Password   string `v:"required|eq:Password2"`
        Password2  string `v:"required"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Name:      "gf",
            Password:   "goframe.org",
            Password2:  "goframe.net",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The Password value `goframe.org` must be equal to field
    Password2 value `goframe.net`
}

```

not-eq

- `:not-eq:field`
- `fielddifferentdifferent`
- `>=v2.2.0`

```

func Example_Rule_NotEQ() {
    type BizReq struct {
        Name      string `v:"required"`
        MailAddr   string `v:"required"`
        OtherMailAddr string `v:"required|not-eq:MailAddr"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Name:      "gf",
            MailAddr:   "gf@goframe.org",
            OtherMailAddr: "gf@goframe.org",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The OtherMailAddr value `gf@goframe.org` must not be
    equal to field MailAddr value `gf@goframe.org`
}

```

gt

- `:gt:field`
- `>=v2.2.0`

```

func Example_Rule_GT() {
    type BizReq struct {
        Value1 int
        Value2 int `v:"gt:Value1"`
        Value3 int `v:"gt:Value1"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Value1: 1,
            Value2: 1,
            Value3: 2,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err.String())
    }

    // Output:
    // The Value2 value `1` must be greater than field Value1
    value `1`
}

```

gte

- :gte:field
-
- >=v2.2.0

```

func Example_Rule_GTE() {
    type BizReq struct {
        Value1 int
        Value2 int `v:"gte:Value1"`
        Value3 int `v:"gte:Value1"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Value1: 2,
            Value2: 1,
            Value3: 2,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err.String())
    }

    // Output:
    // The Value2 value `1` must be greater than or equal to
    field Value1 value `2`
}

```

lt

- :lt:field
-
- >=v2.2.0

```

func Example_Rule_LT() {
    type BizReq struct {
        Value1 int
        Value2 int `v:"lt:Value1"`
        Value3 int `v:"lt:Value1"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Value1: 2,
            Value2: 1,
            Value3: 2,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err.String())
    }

    // Output:
    // The Value3 value `2` must be lesser than field Value1
    value `2`
}

```

lte

- :lte:field
-
- >=v2.2.0

```

func Example_Rule_LTE() {
    type BizReq struct {
        Value1 int
        Value2 int `v:"lte:Value1"`
        Value3 int `v:"lte:Value1"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Value1: 1,
            Value2: 1,
            Value3: 2,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err.String())
    }

    // Output:
    // The Value3 value `2` must be lesser than or equal to
    field Value1 value `1`
}

```

in

- :in:value1,value2,...
- value1,value2,...
- Gender0/1/2

```

func Example_Rule_In() {
    type BizReq struct {
        ID      uint   `v:"required" dc:"Your Id"`
        Name     string  `v:"required" dc:"Your name"`
        Gender   uint   `v:"in:0,1,2" dc:"0:Secret;1:Male;2:
Female"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            ID:      1,
            Name:     "test",
            Gender: 3,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The Gender value `3` is not in acceptable range: 0,1,2
}

```

not-in

- `:not-in:value1,value2,...`
- `value1,value2,...`
- `InvalidIndex-1/0/1`

```

func Example_Rule_NotIn() {
    type BizReq struct {
        ID      uint   `v:"required" dc:"Your Id"`
        Name     string  `v:"required" dc:"Your name"`
        InvalidIndex uint `v:"not-in:-1,0,1"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            ID:      1,
            Name:     "test",
            InvalidIndex: 1,
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Println(err)
    }

    // Output:
    // The InvalidIndex value `1` must not be in range: -1,0,1
}

```

regex

- `:regex:pattern`
- `pattern`

```

func Example_Rule_Regex() {
    type BizReq struct {
        Regex1 string `v:"regex:[1-9][0-9]{4,14}"`
        Regex2 string `v:"regex:[1-9][0-9]{4,14}"`
        Regex3 string `v:"regex:[1-9][0-9]{4,14}"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Regex1: "1234",
            Regex2: "01234",
            Regex3: "10000",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Regex1 value `1234` must be in regex of: [1-9][0-9]
{4,14}
    // The Regex2 value `01234` must be in regex of: [1-9][0-9]
{4,14}
}

```

not-regex

- :not-regex:pattern
- pattern
- >=v2.2.0

```

func Example_Rule_NotRegex() {
    type BizReq struct {
        Regex1 string `v:"regex:\\d{4}"`
        Regex2 string `v:"not-regex:\\d{4}"`
    }
    var (
        ctx = context.Background()
        req = BizReq{
            Regex1: "1234",
            Regex2: "1234",
        }
    )
    if err := g.Validator().Data(req).Run(ctx); err != nil {
        fmt.Print(gstr.Join(err.Strings(), "\n"))
    }

    // Output:
    // The Regex2 value `1234` should not be in regex of: \d{4}
}

```