Gin Bind Data and Validate

https://github.com/gin-gonic/gin

Model binding and validation

https://github.com/go-playground/validator

https://github.com/go-playground/validator/issues/649

- Type - Must bind
  - **Methods** - `Bind` , `BindJSON` , `BindXML` , `BindQuery` , `BindYAML` , `BindHeader`
  - **Behavior** - These methods use `MustBindWith` under the hood. If there is a binding error, the request is aborted with `c.AbortWithError(400, err).SetType(ErrorTypeBind)` . This sets the response status code to 400 and the `Content-Type` header is set to `text/plain; charset=utf-8` . Note that if you try to set the response code after this, it will result in a warning `[GIN-debug] [WARNING] Headers were already written. Wanted to override status code 400 with 422` . If you wish to have greater control over the behavior, consider using the `ShouldBind` equivalent method.
- Type - Should bind
  - **Methods** - `ShouldBind` , `ShouldBindJSON` , `ShouldBindXML` , `ShouldBindQuery` , `ShouldBindYAML` , `ShouldBindHeader`
  - **Behavior** - These methods use `ShouldBindWith` under the hood. If there is a binding error, the error is returned and it is the developer's responsibility to handle the request and error appropriately.

> When using the Bind-method, Gin tries to infer the binder depending on the Content-Type header. If you are sure what you are binding, you can use `MustBindWith` or `ShouldBindWith` .

Must bind vs Should bind

```
package main

import "github.com/gin-gonic/gin"

// 那我们写一个UUID把，这个参数必须填，并且他要符合UUID类型
type User struct {
    ID string `uri:"id" binding:"required,uuid"`
}

func main() {
    r := gin.Default()
```

```
    r.GET("/user/:id", func(c *gin.Context) {
        var user User
        // 然后把这里改成
        err := c.BindUri(&user)
        err := c.ShouldBindUri(&user)
        if err != nil {
            c.JSON(200, gin.H{
                "msg": "参数验证错误",
            })
            return
        }
        c.JSON(200, gin.H{
            "id": user.ID,
        })
    })
    r.Run()
}
```

Bind Json

```
package main

import "github.com/gin-gonic/gin"

type User struct {
    ID string `json:"id" binding:"required"`
}

func main() {
    r := gin.Default()
    r.POST("/user", func(c *gin.Context) {
        var user User
        if err := c.ShouldBindJSON(&user); err != nil {
            c.JSON(200, gin.H{
                "msg": "验证参数失败",
            })
            return
        }
        c.JSON(200, gin.H{
            "id": user.ID,
        })
    })
    r.Run()
}
```

Bind Form

```
package main

import "github.com/gin-gonic/gin"

type User struct {
```

```
    ID       string `form:"id" binding:"required"`
    UserName string `form:"user_name" binding:"required"`
    PassWord string `form:"pass_word" binding:"required"`
}

func main() {
    r := gin.Default()
    r.POST("/user", func(c *gin.Context) {
        var user User
        if err := c.ShouldBind(&user); err != nil {
            c.JSON(200, gin.H{
                "msg": "验证失败",
                "err": err.Error(),
            })
            return
        }
        c.JSON(200, gin.H{
            "id": user.ID,
        })
    })
    r.Run()
}
```

Only Bind Query String

```
package main

import "github.com/gin-gonic/gin"

type User struct {
    ID string `form:"id" binding:"required,uuid"`
}

func main() {
    r := gin.Default()
    r.GET("/user/", func(c *gin.Context) {

        var user User
        err := c.ShouldBindQuery(&user)
        if err != nil {
            c.JSON(200, gin.H{
                "msg":  "参数验证错误",
                //"type": c.GetHeader("Content-Type"),
            })
            return
        }
        c.JSON(200, gin.H{
            "id": user.ID,
        })
    })
    r.Run()
}
```

Bind Query String or Post Data

```
package main

import (
    "log"
    "time"

    "github.com/gin-gonic/gin"
)

type Person struct {
        Name       string    `form:"name"`
        Address    string    `form:"address"`
        Birthday   time.Time `form:"birthday" time_format:"2006-01-02" time_utc:"1"`
        CreateTime time.Time `form:"createTime" time_format:"unixNano"`
        UnixTime   time.Time `form:"unixTime" time_format:"unix"`
}

func main() {
    route := gin.Default()
    route.GET("/testing", startPage)
    route.Run(":8085")
}

func startPage(c *gin.Context) {
    var person Person
    // If `GET`, only `Form` binding engine (`query`) used.
    // If `POST`, first checks the `content-type` for `JSON` or `XML`, then uses
`Form` (`form-data`).
    // See more at https://github.com/gin-gonic/gin/blob/master/binding/binding.go#L48
        if c.ShouldBind(&person) == nil {
                log.Println(person.Name)
                log.Println(person.Address)
                log.Println(person.Birthday)
                log.Println(person.CreateTime)
                log.Println(person.UnixTime)
        }

    c.String(200, "Success")
}
```

Bind Uri

```
package main

import "github.com/gin-gonic/gin"

type User struct {
    ID string `uri:"id" binding:"required,ip"`
}

func main() {
    r := gin.Default()
```

```
    r.GET("/user/:id", func(c *gin.Context) {

        var user User
        err := c.ShouldBindUri(&user)
        if err != nil {
            c.JSON(200, gin.H{
                "msg": "参数验证错误",
            })
            return
        }
        c.JSON(200, gin.H{
            "id": user.ID,
        })
    })
    r.Run()
}
```

Bind Header

```
package main

import (
    "github.com/gin-gonic/gin"
)

type testHeader struct {
    Rate   int    `header:"Rate" binding:"required"`
    Domain string `header:"Domain"`
}

func main() {
    r := gin.Default()
    r.GET("/", func(c *gin.Context) {
        h := testHeader{}

        if err := c.ShouldBindHeader(&h); err != nil {
            c.JSON(200, gin.H{
                "msg": "验证失败",
            })
            return
        }
        c.JSON(200, gin.H{"Rate": h.Rate, "Domain": h.Domain})
    })

    r.Run()
}
```

Custom Validators

需求：id如果非1开头则验证失败，反之

https://github.com/gin-gonic/examples/tree/master/custom-validation/server.go

https://github.com/gin-gonic/gin

```go
package main

import (
    "strings"

    "github.com/gin-gonic/gin/binding"

    "github.com/gin-gonic/gin"
    "github.com/go-playground/validator/v10"
)

type User struct {
    ID string `form:"id" binding:"required,micheal"`
}

//
var customValidate validator.Func = func(fl validator.FieldLevel) bool {
    date := fl.Field().Interface().(string)
    if strings.HasPrefix(date, "1") {
        return true
    }
    return false
}

func main() {
    r := gin.Default()
    // 注册
    if v, ok := binding.Validator.Engine().(*validator.Validate); ok {
        _ = v.RegisterValidation("micheal", customValidate)
    }
    r.GET("/user", func(c *gin.Context) {
        var user User
        if err := c.ShouldBindQuery(&user); err != nil {
            c.JSON(200, gin.H{
                "msg": "验证失败",
            })
            return
        }
        c.JSON(200, gin.H{
            "msg": "success",
        })
    })
    _ = r.Run()
}
```