-

https://pkg.go.dev/github.com/gogf/gf/v2/text/gregex

> *All , slice*
>
> *String , string []byte*



# IsMatch/IsMatchString

- IsMatch()truefalse
-

```
IsMatch(pattern string, src []byte) bool
IsMatchString(pattern string, src string) bool
```

- Tip: regexpRegex
-

```
func ExampleIsMatch() {
        patternStr := `\d+`
        g.Dump(gregex.IsMatch(patternStr, []byte("hello 2022! hello
GoFrame!")))
        g.Dump(gregex.IsMatch(patternStr, nil))
        g.Dump(gregex.IsMatch(patternStr, []byte("hello GoFrame!")))

        // Output:
        // true
        // false
        // false
}
```

# Match/MatchString

- Matchgregex FindSubmatch slice
-

```
Match(pattern string, src []byte) ([][]byte, error)
MatchString(pattern string, src string) ([]string, error)
```

- url

```
func ExampleMatch() {
        patternStr := `(\w+)=(\w+)`
        matchStr := "https://goframe.org/pages/viewpage.action?
pageId=1114219&searchId=8QC5D1D2E!"
        // This method looks for the first match index
        result, err := gregex.Match(patternStr, []byte(matchStr))
        g.Dump(result)
        g.Dump(err)

        // Output:
        // [
        //     "pageId=1114219",
        //     "pageId",
        //     "1114219",
        // ]
        // <nil>
}
```

# MatchAll/MatchAllString

- `MatchAll`gregex `MatchAll` `FindAllSubmatch` `slice`
-

```
MatchAllString(pattern string, src string) ([][]string, error)
MatchAll(pattern string, src []byte) ([][][]byte, error)
```

- `url`

```
func ExampleMatchAll() {
        patternStr := `(\w+)=(\w+)`
        matchStr := "https://goframe.org/pages/viewpage.action?
pageId=1114219&searchId=8QC5D1D2E!"
        result, err := gregex.MatchAll(patternStr, []byte(matchStr))
        g.Dump(result)
        g.Dump(err)

        // Output:
        //  [
        //      [
        //          "pageId=1114219",
        //          "pageId",
        //          "1114219",
        //      ],
        //      [
        //          "searchId=8QC5D1D2E",
        //          "searchId",
        //          "8QC5D1D2E",
        //      ],
        // ]
        // <nil>
}

func ExampleMatchAllString() {
        patternStr := `(\w+)=(\w+)`
        matchStr := "https://goframe.org/pages/viewpage.action?
pageId=1114219&searchId=8QC5D1D2E!"
        result, err := gregex.MatchAllString(patternStr, matchStr)
        g.Dump(result)
        g.Dump(err)

        // Output:
        // [
        //      [
        //          "pageId=1114219",
        //          "pageId",
        //          "1114219",
        //      ],
        //      [
        //          "searchId=8QC5D1D2E",
        //          "searchId",
        //          "8QC5D1D2E",
        //      ],
        // ]
        // <nil>
}
```

# Quote

- :
-

```
Quote(s string) string
```

-

```
func ExampleQuote() {
        result := gregex.Quote(`[1-9]\d+`)
        g.Dump(result)

        // Output:
        // "\[1-9\]\\d\+"
}
```

## Replace/ReplaceString

- 
- :

```
Replace(pattern string, replace, src []byte) ([]byte, error)
ReplaceString(pattern, replace, src string) (string, error)
```

- 

```
func ExampleReplace() {
        var (
                patternStr  = `\d+`
                str         = "hello GoFrame 2020!"
                repStr      = "2021"
                result, err = gregex.Replace(patternStr, []byte
(repStr), []byte(str))
        )
        g.Dump(err)
        g.Dump(result)

        // Output:
        // <nil>
        // "hello GoFrame 2021!"
}
```

## ReplaceFunc/ReplaceStringFunc

- Replace
- 

```
ReplaceFunc(pattern string, src []byte, replaceFunc func(b []byte) []
byte) ([]byte, error)
ReplaceStringFunc(pattern string, src string, replaceFunc func(s
string) string) (string, error)
```

- 

```
func ExampleReplaceFunc() {
        var (
                patternStr = `(\d+)~(\d+)`
                str        = "hello GoFrame 2018~2020!"
        )
        // In contrast to [ExampleReplace]
        result, err := gregex.ReplaceFunc(patternStr, []byte(str),
func(match []byte) []byte {
                g.Dump(match)
                return bytes.Replace(match, []byte("2020"), []byte
("2023"), -1)
        })
        g.Dump(result)
        g.Dump(err)

        // ReplaceStringFunc
        resultStr, _ := gregex.ReplaceStringFunc(patternStr, str,
func(match string) string {
                g.Dump(match)
                return strings.Replace(match, "2020", "2023", -1)
        })
        g.Dump(resultStr)
        g.Dump(err)

           // Output:
        // "2018~2020"
        // "hello gf 2018~2023!" // ReplaceFunc result
        // <nil>
        // "2018~2020"
        // "hello gf 2018-2023!" // ReplaceStringFunc result
        // <nil>
}
```

## ReplaceFuncMatch/ReplaceStringFuncMatch

> *ReplaceFuncMatchsrcregexp*

- MatchString
-

```
ReplaceFuncMatch(pattern string, src []byte, replaceFunc func(match
[][]byte) []byte) ([]byte, error)
ReplaceStringFuncMatch(pattern string, src string, replaceFunc func
(match []string) string) (string, error)
```

-

```
func ExampleReplaceStringFuncMatch() {
         var (
                patternStr = `([A-Z])\w+`
                str        = "hello Golang 2018~2021!"
        )
        // In contrast to [ExampleReplaceFunc]
        // the result contains the `pattern' of all subpatterns that
use the matching function
        result, err := gregex.ReplaceFuncMatch(patternStr, []byte
(str), func(match [][]byte) []byte {
                g.Dump(match)
                return []byte("GoFrame")
        })
        g.Dump(result)
        g.Dump(err)

        // ReplaceStringFuncMatch
        resultStr, err := gregex.ReplaceStringFuncMatch(patternStr,
str, func(match []string) string {
                g.Dump(match)
                match[0] = "Gf"
                return match[0]
        })
        g.Dump(resultStr)
        g.Dump(err)

        // Output:
        // [
        //         "Golang",
        //         "G",
        // ]
        // "hello GoFrame 2018~2021!" // ReplaceFuncMatch result
        // <nil>
        // [
        //         "Golang",
        //         "G",
        // ]
        // "hello Gf 2018~2021!" // ReplaceStringFuncMatch result
        // <nil>
}
```

# Split

- strings.SplitN
-

```
Split(pattern string, src string) []string
```

-

```
func ExampleSplit() {
        patternStr := `\d+`
        str := "hello2020GoFrame"
        result := gregex.Split(patternStr, str)
        g.Dump(result)

        // Output:
        // [
        //     "hello",
        //     "GoFrame",
        // ]
}
```

# Validate

- compile
-

```
Validate(pattern string) error
```

-

```
func ExampleValidate() {
        // Valid match statement
        g.Dump(gregex.Validate(`\d+`))
        // Mismatched statement
        g.Dump(gregex.Validate(`[a-9]\d+`))

        // Output:
        // <nil>
        // {
        //     Code: "invalid character class range",
        //     Expr: "a-9",
        // }
}
```