go+区块链培训 讲师:张长志

字符串处理

字符串在开发中经常用到，包括用户的输入，数据库读取的数据等，我们经常需要对字符串进行分割、连接、转换等操作，我们可以通过Go标准库中的**strings**和**strconv**两个包中的函数进行相应的操作。

下面这些函数来自于strings包，这里介绍一些我平常经常用到的函数，更详细的请参考官方的文档。

http://docscn.studygolang.com/pkg/strings/

Contains

func Contains(s, substr string) bool 功能：字符串s中是否包含substr，返回bool值

```
//"hellogo" 中是否包含hello，包含返回true,不包含返回false
fmt.Println(strings.Contains("hellogo","hello"))
fmt.Println(strings.Contains("hellogo","abc"))
```

Joins组合

func Join(a []string, sep string) string 功能：字符串链接，把slice a通过sep链接起来

```
s :=[]string{"abc","hello","mike","go"}
buf := strings.Join(s,"_")
fmt.Println("buf=",buf)
```

Index 查找子串位置

func Index(s, sep string) int 功能：在字符串s中查找sep所在的位置，返回位置值，找不到返回-1

```
//Index 查找字段位置
fmt.Println(strings.Index("abcdhello","hello")) //4
fmt.Println(strings.Index("abcdhello","ddddd"))//-1
```

Repeat

func Repeat(s string, count int) string 功能：重复s字符串count次，最后返回重复的字符串

```
//Repeat
buf = strings.Repeat("go",5)
   fmt.Println("buf=",buf)
```

Split

func Split(s, sep string) []string 功能：把s字符串按照sep分割，返回slice

```
buf ="abc_hello_mike_go"
s2:=strings.Split(buf,"_")
fmt.Println("s2=",s2)
```

Trim去掉俩头的字符

```
//trim
buf = strings.Trim(" are you aoka    "," a") //去掉俩边头的空格指定的字符
fmt.Printf("buf=#%s#\n",buf)
```

Fields去掉空格，把元素放入切片中

```
  //Fields 去掉空格，把元素放入切片中
   s3 := strings.Fields(" are you ok?   ")
fmt.Println("s3=",s3)

for i,data := range s3{
   fmt.Println(i,data)
}
```

字符串转换

字符串转化的函数在strconv中，如下也只是列出一些常用的。Append 系列函数将整数等转换为字符串后，添加到现有的字节数组中。

一、转换成字符串添加到字节数组中

```
slice := make([]byte,0,1024)
slice =strconv.AppendBool(slice,true)
//第二个数要为追加是数，第三个指定10进制的方式追加
slice =strconv.AppendInt(slice,1234,16)
slice = strconv.AppendQuote(slice,"abcdefg")

fmt.Println("slice=",string(slice))
```

二、其他类型转换成字符串

```
//把其他类型转换成字符串
var str string
str = strconv.FormatBool(true)

str = strconv.FormatInt(10,8)
fmt.Println("str=",str)

str = strconv.FormatFloat(3.14,'f',-1,64)
fmt.Println("str=",str)
```

三、把字符串转换成其他类型

```
var flag bool
var err error
flag,err = strconv.ParseBool("true")
if err == nil{
   fmt.Println("flag=",flag)
}else{
   fmt.Println("err=",err)
}

//把字符串转换成整形
a,err:=strconv.Atoi("aaa")
if err == nil{
   fmt.Println("a=",a)
}else {
   fmt.Println("err=",err)
}
```

正则表达式

正则表达式是一种进行**模式匹配**和**文本操纵**的复杂而又强大的工具。虽然正则表达式比纯粹的文本匹配效率低，但是它却更灵活。按照它的语法规则，随需**构造出的匹配模式**就能够从原始文本中筛选出**几乎任何你想要得到的字符**组合。

Go语言通过regexp标准包为正则表达式提供了官方支持，如果你已经使用过其他编程语言提供的正则相关功能，那么你应该对Go语言版本的不会太陌生，但是它们之间也有一些小的差异，因为Go实现的是RE2标准，除了\C，详细的语法描述参考：http://code.google.com/p/re2/wiki/Syntax

其实字符串处理我们可以使用strings包来进行搜索(Contains、Index)、替换(Replace)和解析(Split、Join)等操作，但是这些都是简单的字符串操作，他们的搜索都是大小写敏感，而且固定的字符串，如果我们需要匹配可变的那种就没办法实现了，当然如果strings包能解决你的问题，那么就尽量使用它来解决。因为他们足够简单、而且性能和可读性都会比正则好

. 匹配除换行符以外的任意字符 \w 匹配字母或数字或下划线或汉字 \s 匹配任意的空白符 **\d 匹配数字** \b 匹配单词的开始或结束 **^ 匹配字符串的开始 $ 匹配字符串的结束**

***重复零次或更多次**

+**重复一次或更多次 ? 重复零次或一次 {n} 重复n次 {n,} 重复n次或更多次 {n,m} 重复n到m次**

捕获 (exp) 匹配exp,并捕获文本到自动命名的组里

官方提供方法

func Match(pattern string, b []byte) (matched bool, err error)

func MatchString(pattern string, s string) (matched bool, err error)

func MustCompile(str string) *Regexp

func (re *Regexp) FindAllString(s string, n int) []string

```
func main() {
    //reg := regexp.MustCompile("\\w+") 正则表达式中的\需要转义
    reg := regexp.MustCompile(`^z.*l$`)

    result := reg.FindAllString("zhangsanl", -1)
    fmt.Printf("%v\n", result)

    reg1 := regexp.MustCompile(`^z(.*)l$`)

    result1 := reg1.FindAllString("zhangsand", -1)
    fmt.Printf("%v\n", result1)
}
```

使用正则表达式规范

1 ）我们写一个规则进行编译

```
reg := regexp.MustCompile(`^z.*l$`)
```

2) 拿规则reg 去配置你的字符串

```
result := reg.FindAllString("zhangsanl zhangsanl", -1)
```

3) FindAllStringSubmatch 在次分组，在得到配置结构 在次根据内部条件获取 返回二维数组

```
 reslut := reg1.FindAllStringSubmatch(buf,-1)
fmt.Println("result=",reslut)
```

案例一

```
package main

import (
   "regexp"
   "fmt"
)

func main(){
    buf := "abc azc a7c aac 888 a9c ac tac" //axxxc

    //1,写一个解析规则
    //reg1 := regexp.MustCompile(`a(.)c`) //(.) 把满足条件放在一组
     //配置数字
     //reg1 := regexp.MustCompile(`a([0-9]+)c`) //a1c a2c a3c
   // reg1 := regexp.MustCompile(`a(\d+)c`) //a1c a2c a3c [0-9]=\d
    //配置字符串 [a-zA-Z0-9_]=\w
    //reg1 := regexp.MustCompile("a[a-zA-Z]c")
   reg1 := regexp.MustCompile(`a\wc`)
   //2.拿规则去配置字符串
   //reslut := reg1.FindAllString(buf,-1)
    reslut := reg1.FindAllStringSubmatch(buf,-1)
   fmt.Println("result=",reslut)
}
```

案例二 、 .的使用

```
package main

import (
   "regexp"
```

```
    "fmt"
)

func main(){

    buf := "43.14 567 agsdg 1.23 7. 8.9 dddddss 6.66 7.8"
    //正则表达式
    reg := regexp.MustCompile(`\d+\.\d+`) //4.5  \.
    //result := reg.FindAllString(buf,-1)
    result := reg.FindAllStringSubmatch(buf,-1)
    fmt.Println("result=",result)

}
```

案例三 、通过正则表达式查找邮箱

```
package main

import (
    "regexp"
    "fmt"
)

//const test  = "my email is 530979104@qq.com"
const text  =`
my email is 530979104@qq.com
email is adbd@qq.com
email is ccc@qqq.org.com
`
func main()  {

  // res := regexp.MustCompile("530979104@qq.com") //生成的匹配规则去往原始文件里面匹配
    //res := regexp.MustCompile("[A-Za-z0-9]+@[A-Za-z0-9]+\\.[A-Za-z0-9]+")
    //
    res := regexp.MustCompile(`([A-Za-z0-9]+)@([A-Za-z0-9]+)\.([A-Za-z0-9.]+)+`)
     match := res.FindAllStringSubmatch(text,-1) //匹配成功进行截取


    fmt.Println(match)

    for _,m:=range match{
         fmt.Println(m[1])
    }


}
```