go+区块链培训 讲师:张长志

Json与xml区别

Person

zhangsan

15

zhangsan

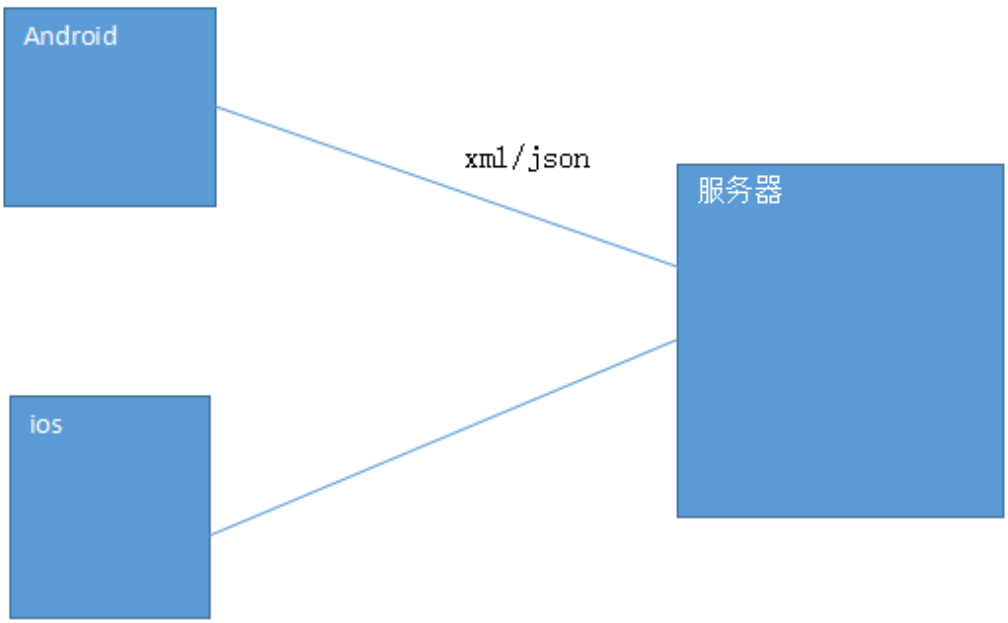
15

Json:

[{name:zhangsan,age:15},{name:lisi,age:15}]

网络传输：需要流量

Json（javaScript Object Notation）是一种比XML更轻量的数据交换格式，易于读和编写，也易于程序的解析和生成。Json表现形式一般表现为键/值对应的集合，这种json传输称为较为理想的跨平台 跨语言的数据交换语言



{

“company” : “zhczGO”,

"isok":true,

"price":99.00

"subjects":["go","docker","Test"]

}

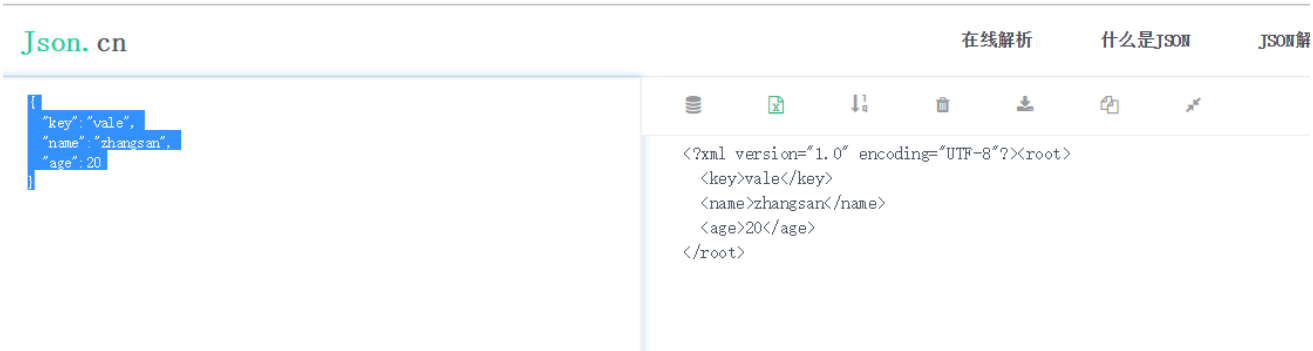
开发者可以json传输简单的字符串，数字，布尔值，也可以传输一个数组，或者一个更复杂的结构。在Web开发中，Json被广泛的应用于Web服务器和程序之间的数据通信。

go语言对json的支持

内存了标准库 encoding/json ，开发者可以轻松使用go语言程序生成json格式的数据

JSON官方网站 ： http://www.json.org/

在线格式化：https://www.json.cn/



代码方式 结构体 和map 转换成json数据

1、把结构体转换成json案例

```
package main

import (
    "encoding/json"
    "fmt"
)

/**
```

```
{

  “company”：“zhczGO”,

 "isok":true,

 "price":99.00

"subjects":["go","docker","Test"]

}
 */
 type IT struct {
   Company string
   Subjects []string
   IsOk   bool
   Price  float64
 }

func main(){

   s:= IT{"zhczGO",[]string{"go","docker","fabric","Test"},true,999}

   //内置方法
   buf,err:=json.Marshal(s)
   if err !=nil{
      fmt.Println("err=",err)
      return
   }
   fmt.Println("buf=",string(buf))
}
```

结构展示：

json的结构体二次编码

```
type IT struct {
   Company string `json:"company"` //二次编码
   Subjects []string `json:"subjects"`
   IsOk   bool `json:"isok",string`
   Price  float64 `json:"price,string"`
}
```

json的结构体隐藏部分字段

```
type IT struct {
   Company string `json:"-"` //-此字段不会输入到屏幕上面
   Subjects []string `json:"subjects"`
   IsOk   bool `json:"isok",string`
   Price  float64 `json:"price,string"`
}
```

json的格式编码

```
func main(){

   s:= IT{"zhczGO",[]string{"go","docker","fabric","Test"},true,999}

   //内置方法
   //buf,err:=json.Marshal(s)
   buf,err:=json.MarshalIndent(s,""," ") //格式化代码
   if err !=nil{
      fmt.Println("err=",err)
      return
   }
   fmt.Println("buf=",string(buf))
}
```

```
buf= {
     "subjects": [
          "go",
          "docker",
          "fabric",
          "Test"
     ],
     "isok": true,
     "price": "999"
}
```

map生成json

```
package main

import (
   "encoding/json"
   "fmt"
)

func main(){
   m := make(map[string]interface{},4)
   m["company"] = "zhczGO"
   m["subjects"]=[]string{"go","fabric","python","Test"}
   m["isok"] = true
   m["price"] = 99

   // result,err :=json.Marshal(m)
   result,err := json.MarshalIndent(m,"","    ")
    if err !=nil{
      fmt.Println("err=",err)
      return
    }
    fmt.Println("result=",string(result))
}
```

json解析到结构体

```
package main

import (
   "encoding/json"
   "fmt"
)

type IT1 struct {
   Company string `json:"company"`
   Subjects []string `json:"subjects"`
   IsOk bool       `json:"isok"`
   Price float64 `json:"price"`
}

func main(){
   jsonbuff :=`{
   "company": "zhczGO",
   "isok": true,
   "price": 99,
   "subjects": [
```

```
        "go",
        "fabric",
        "python",
        "Test"
    ]
}`

var temp IT1
err :=json.Unmarshal([]byte(jsonbuff),&temp) //要传入地址，定义一个结构体变量
if err != nil{
    fmt.Println("err=",err)
    return
}
//fmt.Println("temp=",temp)
fmt.Printf("tmp=%+v\n",temp)
}
```

```
package main

import (
    "encoding/json"
    "fmt"
)

type IT1 struct {
    Company string `json:"company"`
    Subjects []string `json:"subjects"`
    IsOk bool       `json:"isok"`
    Price float64 `json:"price"`
}

type IT2 struct {
    Company string `json:"company"`

}

func main(){
    jsonbuff :=`{
    "company": "zhczGO",
    "isok": true,
    "price": 99,
    "subjects": [
        "go",
        "fabric",
        "python",
        "Test"
    ]
}`

var temp IT1
err :=json.Unmarshal([]byte(jsonbuff),&temp)
```

```
if err != nil{
   fmt.Println("err=",err)
   return
}
//fmt.Println("temp=",temp)
fmt.Printf("tmp=%+v\n",temp)

var temp2 IT2
err = json.Unmarshal([]byte(jsonbuff),&temp2)
if err != nil{
   fmt.Println("err=",err)
   return
}
fmt.Printf("tmp2=%+v\n",temp2)

}
```

json解析到map

json转换成map，非常简单，取出对应的值要借助断言

```
package main

import (
   "encoding/json"
   "fmt"
)

func main()  {
   jsonbuff :=`{
   "company": "zhczGO",
   "isok": true,
   "price": 99,
   "subjects": [
      "go",
      "fabric",
      "python",
      "Test"
   ]
}`

  //创建一个map
  m := make(map[string]interface{},4)
  err := json.Unmarshal([]byte(jsonbuff),&m) //一定要是地址

  if err != nil{
     fmt.Println("err=",err)
     return
  }
```

```
    fmt.Printf("m=%+v/n",m)

    //var str string
    //str = m["company"]//err 无法转换

    //类型断言
     for key,value := range m  {
        switch data := value.(type) {
        case string:
            //str = data
            fmt.Printf("map[%s]的值类型为string，value=%s\n",key,data)
        case bool:
            fmt.Printf("map[%s]的值类型为bool，value=%v\n",key,data)
        case float64:
            fmt.Printf("map[%s]的值类型为float64，value=%f\n",key,data)
        case []string:
            fmt.Printf("map[%s]的值类型为[]string，value=%v\n",key,data)
        case []interface{}:
            fmt.Printf("map[%s]的值类型为interface{}，value=%v\n",key,data)
        }
     }


}
```

文件设备使用

```
package main

import (
    "fmt"
    //"os"
    "os"
)

func main()  {
    //os.Stdout.Close()  //关闭后，无法输出  os.Stdout标准设备文件 默认是打开的，用户之间使用
    fmt.Println("are you ok?") //往标准输入设备（屏幕）写内容
    //os.Stdout.
    os.Stdout.WriteString("are you ok?\n")

    //os.Stdin
    var a int
    fmt.Println("请输入一个a")
    fmt.Scan(&a)
    fmt.Println("a=",a)
```

```
}
```

往文件写数据

```
package main

import (
   "os"
   "fmt"
   "io"
   "bufio"
)
//文件的写入
func WriteFile(path string)  {
   //打开文件，新建文件
   f,err :=os.Create(path)
   if err != nil{
      fmt.Println("err =",err)
      return
   }
   //使用完毕，关闭文件
   defer f.Close()

   var buf string
   for i:=0;i<10;i++{
      //"i=%d\n" 这个字符串存储到buf里面
      buf = fmt.Sprintf("i=%d\n",i)
      n,err:=f.WriteString(buf)
      if err !=nil{
         fmt.Println("err=",err)
         return
      }
      fmt.Println("n=",n)
   }

}

//文件读取
func ReadFile(path string){
    f,err := os.Open(path)
    if err != nil{
      fmt.Println("err=",err)
      return
    }
    defer  f.Close()

    buf := make([]byte,1024*2) //2k大小
    n,err := f.Read(buf)
```

```
    if err!=nil && err != io.EOF{ //文件出错同时文件没有到结尾
       fmt.Println("err=",err)
       return
    }
    fmt.Println("buf=",string(buf[:n]))
}

//每次读取一行
func ReadFileLine(path string){
   f,err := os.Open(path)
   if err != nil{
      fmt.Println("err=",err)
      return
   }
   //关闭文件
   defer  f.Close()
   //新建一个缓冲区。把内容先放着缓冲里面
   r := bufio.NewReader(f)
   for{
       //遇到\n结束读取，但是\n也进入缓冲区
        buf,err := r.ReadBytes('\n')
        if err != nil{
         if err == io.EOF{ //文件已经结束
              break;
         }
         fmt.Println("err=",err)
        }
        fmt.Printf("buf=#%s#\n",string(buf))

   }
}



func main(){
   path :="./demo.txt"
   //WriteFile(path)
   //ReadFileLine(path)
   ReadFile(path)

}
```

作业

利用文件读写实现文件拷贝