大厂面试课之Go语言

第8课：JSON

同学们好！在上一节课，我们曾提到，在HTTP客户机和HTTP服务器之间传递请求和响应的过程中，有时会通过一种名为JSON字符串的形式，承载请求和响应中的业务数据。这节课我们将详细地了解一下到底什么是JSON？以及如何在Go语言中使用JSON？

首先我们需要了解JSON的基本语法，然后看看在Go语言中如何通过代码对JSON做编解码处理。

JSON语法

JSON的全称是JavaScript Object Notation，即JavaScript对象表示法，是一种基于JavaScript语言的轻量级数据交换格式。它是ECMA，即欧洲计算机制造商协会，制定的JavaScript规范的一个子集，采用完全独立于编程语言的文本格式，存储和表示数据。简洁而清晰的层次结构，使得JSON成为理想的数据交换语言。相较于更为传统的XML语言，JSON不仅易于被人类读写，也更易于被机器解析和生成，能够显著提升网络传输的效率。

知识讲解

基本类型

- 字符串：用双引号引起来
- 数字：不用引号

列表类型

- [元素1, 元素2, 元素3, ...]
- 方括号括起0~N个元素
- 元素之间以逗号分隔，最后一个元素后无逗号
- 元素可以是字符串、数字、列表或字典

JSON中两种最常用的基本数据类型是字符串和数字，二者的区别在于前者需要被一对双引号引起，而后者不用。此外，JSON中还可以通过列表表示多个数据的集合。其语法为在一对方括号中列出若干被逗号分隔的数据元素。列表中也可以没有数据元素，即所谓空列表，或者只包含一个数据元素。任何情况下，列表中最后一个数据元素的后面都不要加逗号。列表中的数据元素可以是字符串或者数字，也可以又是一个列表甚至字典。

字典类型

知识讲解

- {键1:值1, 键2:值2, 键3:值3, ...}
- 花括号括起0~N个键值对
- 键值之间以冒号分隔，键值对之间以逗号分隔，最后一个键值对后无逗号
- 键必须是字符串且不可重复
- 值可以是字符串、数字、列表或字典

JSON还支持字典类型。其形式为被一对花括号括起来的0~N个键值对的集合。键值之间以冒号分隔，键值对之间以逗号分隔，最后一个键值对之后不要再加逗号。字典中每个键值对的键必须是字符串且必须唯一，即不能有两个或两个以上的键值对拥有完全相同的键。字典中每个键值对的值可以是字符串或者数字，也可以是列表甚至又是一个字典。

例如

知识讲解

```
{
    "name":"张飞",
    "gender":"男",
    "age":25,
    "courses":["语文","数学","英语","物理","化学"],
    "成绩":[80,85,70,90,75],
    "家电":{"洗衣机":"海尔","电冰箱":"西门子","电视":"三星"},
    "friends":[
        {"address":"北京","name":"Tina"},
        {"address":"香港","name":"Andy"},
        {"address":"台湾","name":"Abby"}
    ]
}
```

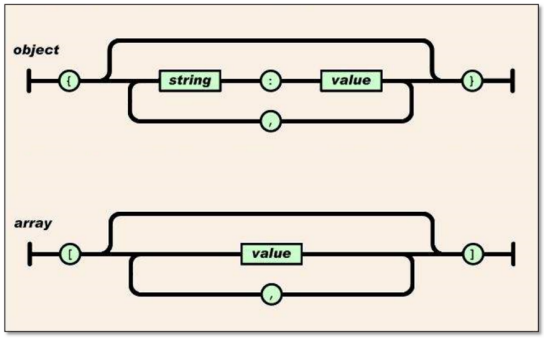
这里给出一个JSON字符串的示例，描述了一个学生对象的信息。我们看到整个JSON就是一个字典，包含七个键值对。第一个键值对的键为“name”表示学生的姓名，其值为字符串“张飞”。第二个键值对的键为“gender”表示学生的性别，其值为字符串“男”。第三个键值对的键为“age”表示学生的年龄，其值为数字25。第四个键值对的键为“courses”表示学生学习的课程，其值为一个列表，列表中包括“语文”、“数学”、“英语”、“物理”和“化学”五个字符串元素。第五个键值对的键为“成绩”，其值也是一个列表，包含80、85、70、90和75五个数字元素，分别与课程列表中的每门课程相对应。第六个键值对的键为“家电”表示学生寓所中的家用电器，其值为一个字典，列出了每台家电的名称和品牌，海尔牌的洗衣机、西门子的电冰箱和三星的电视。最后一个键值对的键为“friends”表示学生的好友，其值为一个列表，列表中的每个元素又是一个字典，包含每个好友的地址和姓名，住在北京的Tina、住在香港的Andy和住在台湾的Abby。

JSON编解码

看到这里，同学们应该已经发现，实际上JSON是用来描述对象的，这和我们在编程语言中通过类描述对象有异曲同工之处。字典中的键，如“name”、“gender”、“age”等，就相当于类中的属性名，而每个键的值，如“张飞”、“男”、25等，就是类实例化为对象后，每个属性的值。把一个内存中对象变成JSON字符串的过程，被称为JSON编码，而把一个JSON字符串变成内存对象的过程，则被称为JSON解码。

在Go语言中编解码JSON字符串

知识讲解



- 编码：将结构体编码为JSON字符串
- 解码：将JSON字符串解码为结构体

具体到Go语言，如果我们已经有了一个结构体类型的变量，按照成员名对应键，成员值对应值的规则，即可得到一个字典形式的JSON字符串，这就是编码。反过来，从一个JSON字符串，将每个键的值，按照成员名对应地填写到一个结构体变量中，这就是解码。

JSON与Go的语法对应关系

知识讲解

- JSON第一级字典的键对应结构体成员的名
- JSON第一级字典的值对应结构体成员的值
  - 字符串——string
  - 数字——int/float32/float64...
  - 列表——切片
  - 字典——映射
- 只有首字母大写的结构体成员参与编解码
- 结构体成员可带有标签
  - `json:"-"`：不参与编解码
  - `json:"name,string,omitempty"`：指定JSON中的键和类型，忽略空值

至此，我们可以把JSON与Go的语法对应关系总结一下。JSON中一级字典的键对应结构体的成员名，值对应结构体的成员值。JSON中的字符串对应Go中的string类型，JSON中的数字对应Go中的int、float32或float64类型，JSON中的列表对应Go中的切片类型，JSON中的二级及以下各级字典对应Go中的映射类型。注意只有名字首字母大写的结构体成员参与JSON编解码，并与JSON中首字母小写的键相对应。为了使得这种对应更为灵活，可以为结构体成员设置被一对反引号引起的标签。比如`json:"-"`表示该成员不参与编解码，即便其名字的首字母为大写字符。又如`json:"name,string,omitempty"`指定与该成员对应的JSON键值对的键、值类型、忽略空值。

GOPATH/src/JSONCodec/main.go

代码实践

```
package main

import (
    "encoding/json"
    "fmt"
)

type Student struct {
    Name        string              `json:"name"`
    Gender      string              `json:"gender"`
    Age         int                 `json:"age"`
    Courses     []string            `json:"courses"`
    Scores      []float32           `json:"成绩"`
    Electricals map[string]string   `json:"家电"`
    Friends     []map[string]string `json:"friends"`
}
```


GOPATH/src/JSONCodec/main.go

代码实践

```
func main() {
    stu1 := Student{
        Name:        "张飞",
        Gender:      "男",
        Age:         25,
        Courses:     []string{"语文", "数学", "英语", "物理", "化学"},
        Scores:      []float32{80, 85, 70, 90, 75},
        Electricals: make(map[string]string, 3),
        Friends: []map[string]string{
            make(map[string]string, 2),
            make(map[string]string, 2),
            make(map[string]string, 2),
        },
    }
    stu1.Electricals["洗衣机"] = "海尔"
    stu1.Electricals["电视"] = "三星"
    stu1.Electricals["电冰箱"] = "西门子"
    stu1.Friends[0]["address"] = "北京"
```


GOPATH/src/JSONCodec/main.go

代码实践

```
    stu1.Friends[0]["name"] = "Tina"
    stu1.Friends[1]["address"] = "香港"
    stu1.Friends[1]["name"] = "Andy"
    stu1.Friends[2]["address"] = "台湾"
    stu1.Friends[2]["name"] = "Abby"
    jstr, err := json.Marshal(stu1)
    if err != nil {
        fmt.Println("json.Marshal错误:", err)
        return
    }
    fmt.Println(string(jstr))

    stu2 := Student{}
    if err = json.Unmarshal(jstr, &stu2); err != nil {
        fmt.Println("json.Unmarshal错误:", err)
        return
    }
    fmt.Printf("%+v\n", stu2)
}
```

输出

代码实践

```
{
    "name":"张飞",
    "gender":"男",
    "age":25,
    "courses":["语文","数学","英语","物理","化学"],
    "成绩":[80,85,70,90,75],
    "家电":{"洗衣机":"海尔","电冰箱":"西门子","电视":"三星"},
    "friends":[
        {"address":"北京","name":"Tina"},
        {"address":"香港","name":"Andy"},
        {"address":"台湾","name":"Abby"}
    ]
}
```

输出

代码实践

```
{
    Name:张飞
    Gender:男
    Age:25
    Courses:[语文 数学 英语 物理 化学]
    Scores:[80 85 70 90 75]
    Electricals:map[洗衣机:海尔 电冰箱:西门子 电视:三星]
    Friends:[
        map[address:北京 name:Tina]
        map[address:香港 name:Andy]
        map[address:台湾 name:Abby]
    ]
}
```

下面我们通过一个名为JSONCodec的工程，体验一下在Go语言中编解码JSON字符串的方法：

```go
package main

import (
    "encoding/json"
    "fmt"
)

type Student struct {
    Name        string              `json:"name"`
    Gender      string              `json:"gender"`
    Age         int                 `json:"age"`
    Courses     []string            `json:"courses"`
    Scores      []float32           `json:"成绩"`
    Electricals map[string]string   `json:"家电"`
    Friends     []map[string]string `json:"friends"`
}

```

```go
func main() {
    stu1 := Student{
        Name:        "张飞",
        Gender:      "男",
        Age:         25,
        Courses:     []string{"语文", "数学", "英语", "物理", "化学"},
        Scores:      []float32{80, 85, 70, 90, 75},
        Electricals: make(map[string]string, 3),
        Friends: []map[string]string{
            make(map[string]string, 2),
            make(map[string]string, 2),
            make(map[string]string, 2),
        },
    }
    stu1.Electricals["洗衣机"] = "海尔"
    stu1.Electricals["电视"] = "三星"
    stu1.Electricals["电冰箱"] = "西门子"
    stu1.Friends[0]["address"] = "北京"
    stu1.Friends[0]["name"] = "Tina"
    stu1.Friends[1]["address"] = "香港"
    stu1.Friends[1]["name"] = "Andy"
    stu1.Friends[2]["address"] = "台湾"
    stu1.Friends[2]["name"] = "Abby"
    jstr, err := json.Marshal(stu1)
    if err != nil {
        fmt.Println("json.Marshal错误:", err)
        return
    }
    fmt.Println(string(jstr))

    stu2 := Student{}
    if err = json.Unmarshal(jstr, &stu2); err != nil {
        fmt.Println("json.Unmarshal错误:", err)
        return
    }
    fmt.Printf("%+v\n", stu2)
}
```

在程序的开始部分，我们定义了一个名为Student的结构体类型，其中包括了表示学生对象各个属性的成员，并在反引号标签中注明了与每个成员相对应的JSON键。在main函数中，用Student结构体实例化了一个学生对象，为其中的各个成员赋予了相应的值。通过json包的Marshal函数将该学生对象编码为一个JSON字符串并打印。紧接着，又通过json包的Unmarshal函数将这个JSON字符串解码为另一个学生对象并打印。观察并比较两次打印输出的结果，验证JSON编解码的正确性。

更多精彩，敬请期待

谢谢大家，我们下节课再见！