序列化和反序列化

联系QQ：2816010068，加入会员群

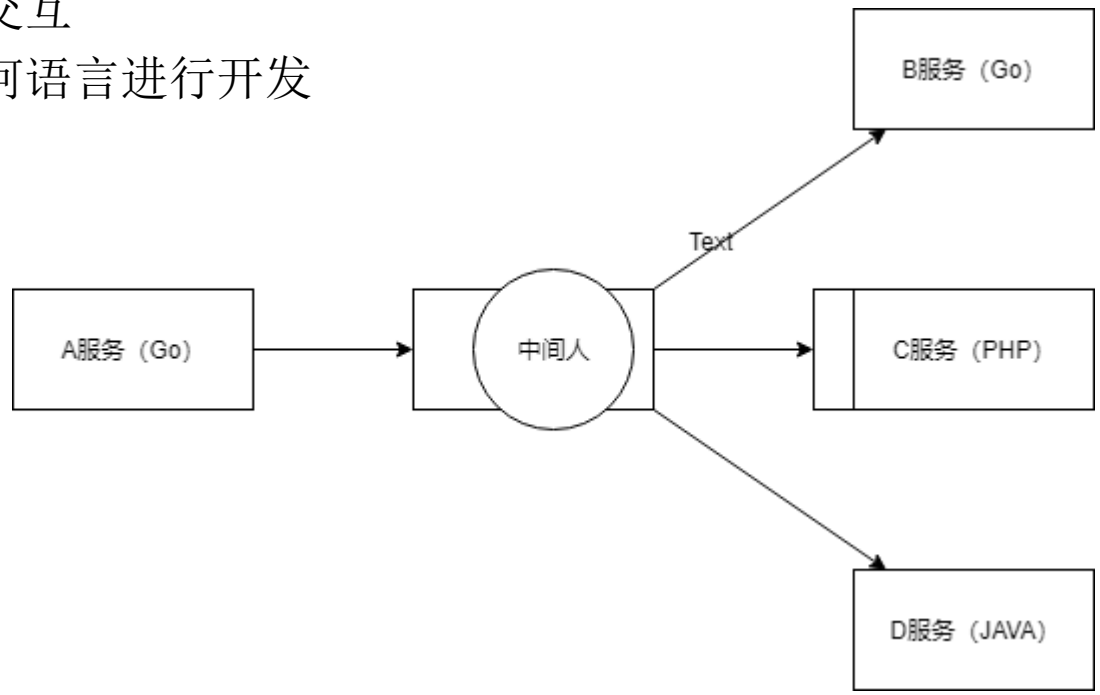
二叉树出品

目录

- 为什么要序列化和反序列化
- Json简介和使用
- Msgpack简介和使用
- Protobuf简介和使用

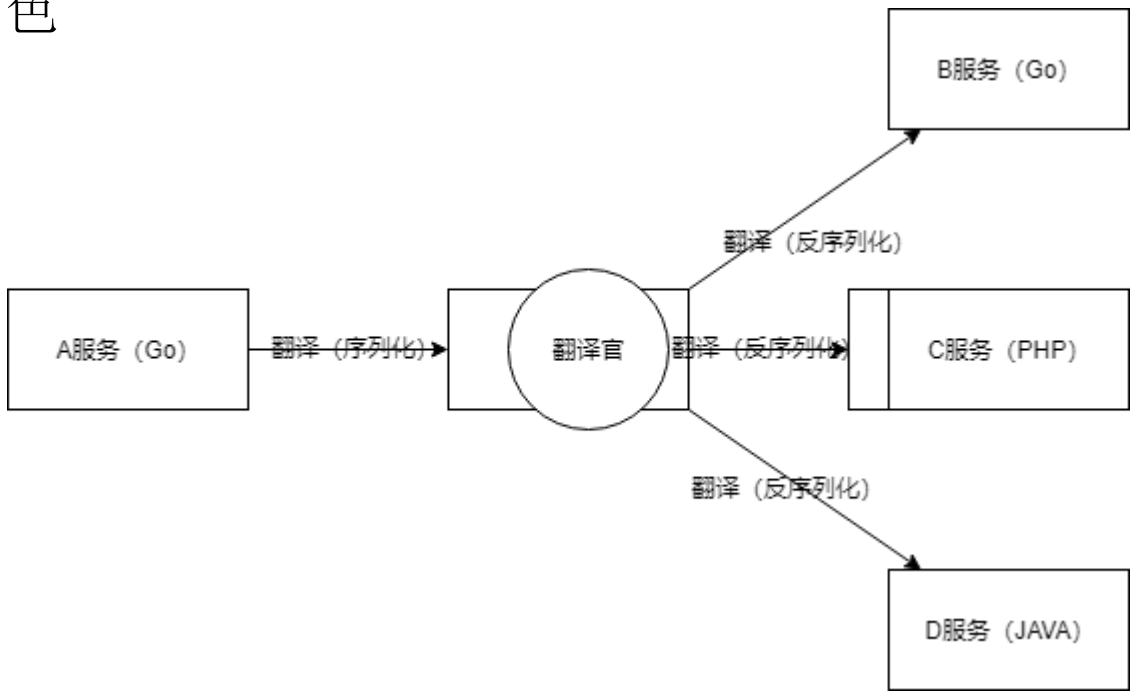
为什么需要序列化和反序列化?

- 微服务特点
  - 每个服务专注做好一件事
  - 微服务之间需要进行交互
  - 每个服务可以使用任何语言进行开发



为什么需要序列化和反序列化?

- 服务之间进行沟通（交互）的媒介
- 序列化和反序列化，充当翻译官的角色
- 翻译官的特点
  - 支持目前绝大多数的语言
  - 翻译的又快又准（性能好）
  - 空间开销小（节约带宽）



常见数据格式对比

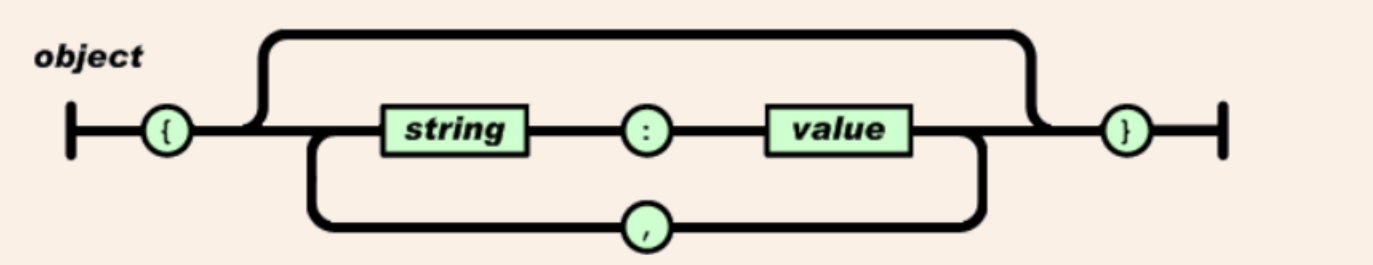
协议	实现	跨语言	性能	传输量	RPC
xml	广泛	几乎所有	低	很大	N（可实现）
json	广泛	大量	一般	一般	N（可实现）
php serialize	PHPRPC	大量	一般	一般	Y
hessian	hessian	大量	一般	小	Y
thrift	thrift	大量	高	小	Y
protobuf	protobuf	大量	高	小	N（可实现）
ice	ice	大量	高	小	Y
avro	Apache Avro	少量	高	小	Y
messagepack	messagepack	大量	高	小	Y

Json数据格式介绍和使用

- 优点
  - 文本格式，易读性较好
  - 简单易用，开发成本低
- 缺点
  - 序列化后的体积比较大，影响并发
  - 序列化性能低

Json数据格式介绍和使用

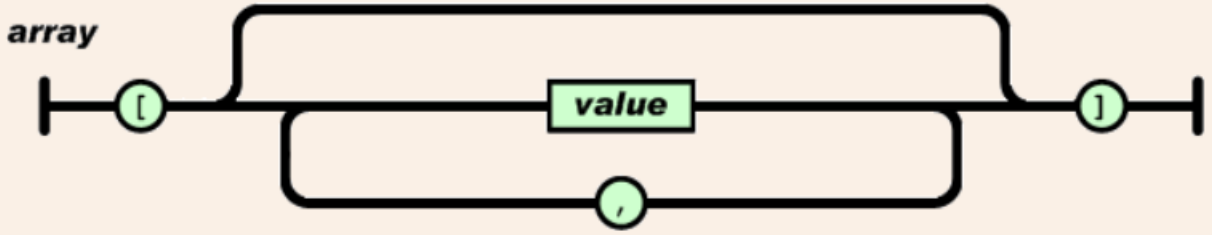
- 对象



Json数据格式介绍和使用

- 数组

数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）

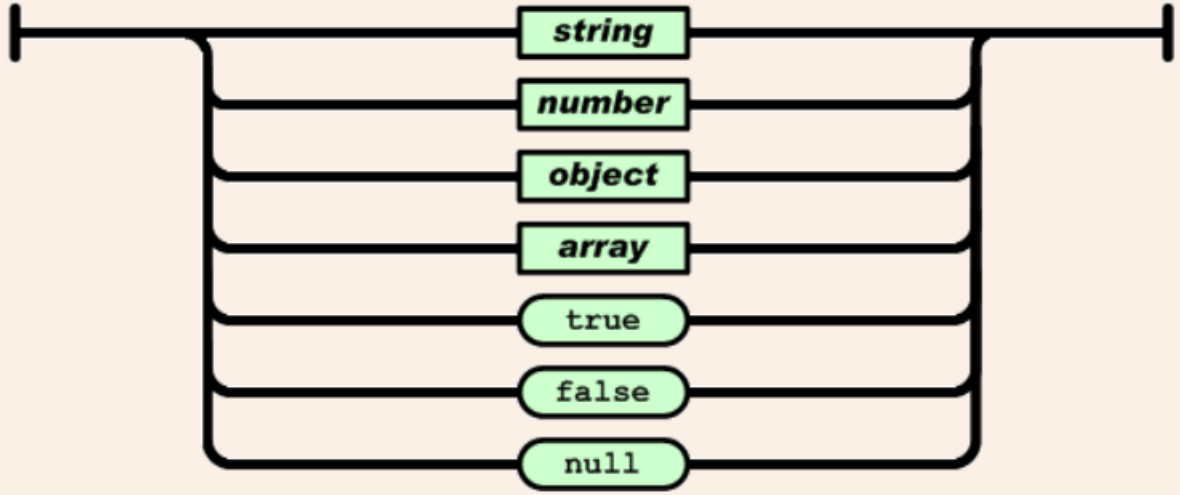


Json数据格式介绍和使用

- value

值（*value*）可以是双引号括起来的字符串（*string*）、数值(number)、**true**、**false**、 **null**、对象（object）或者数组（array）。这些结构可以嵌套。

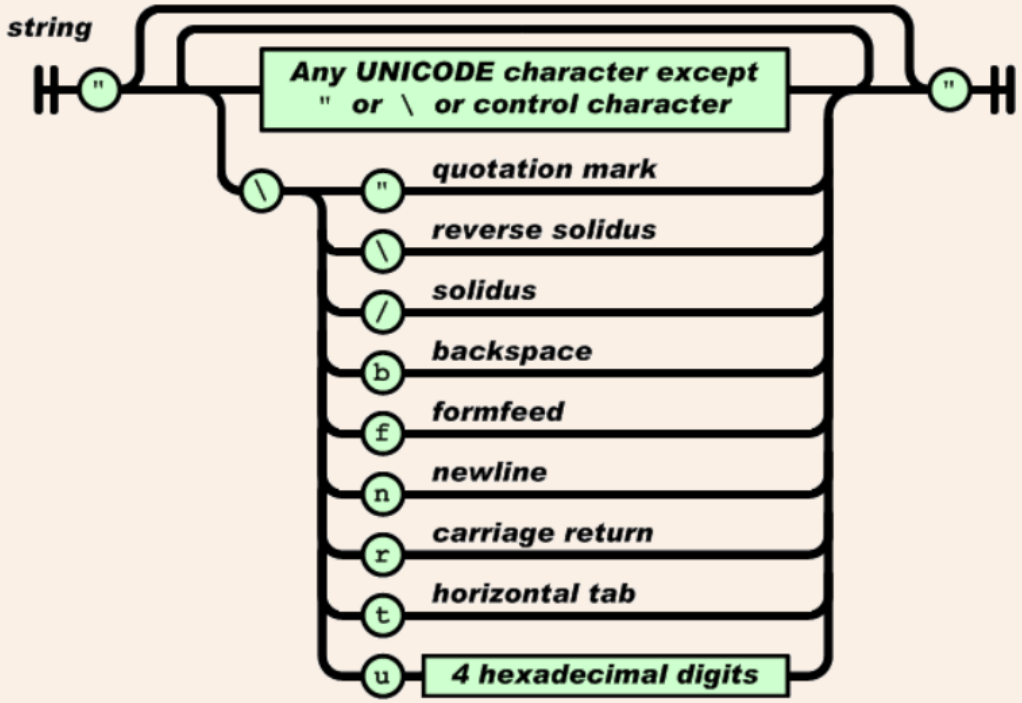
value



Json数据格式介绍和使用

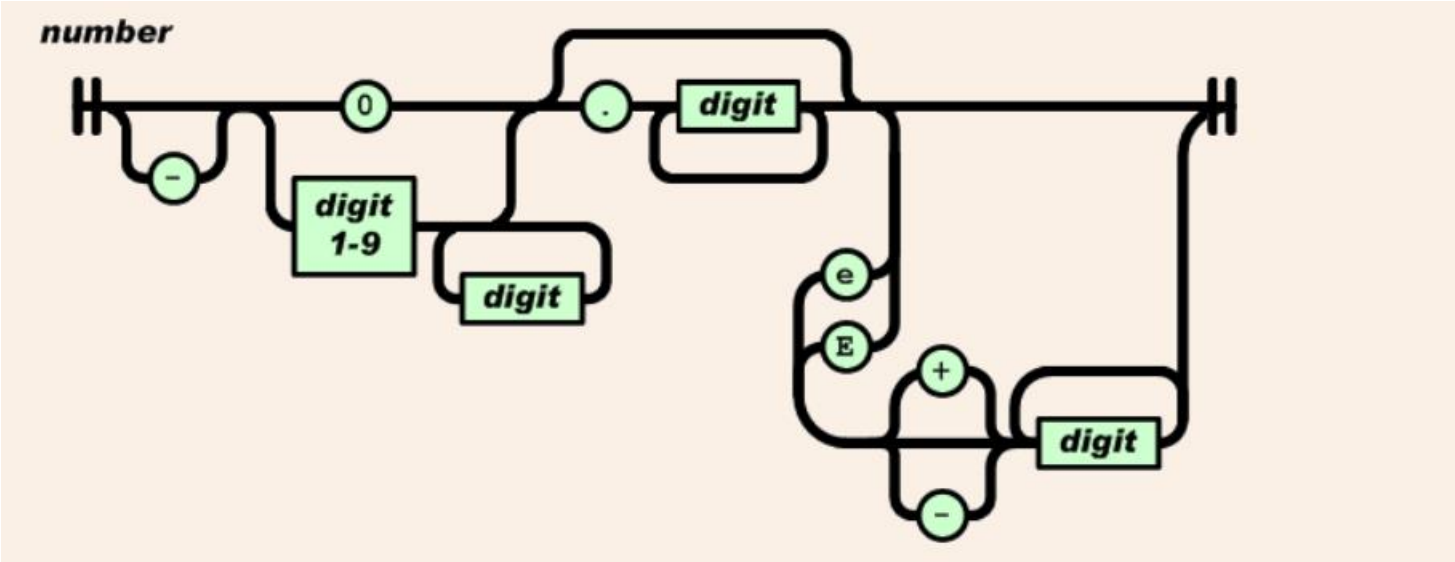
- 字符串

字符串（*string*）与C或者Java的字符串非常相似。



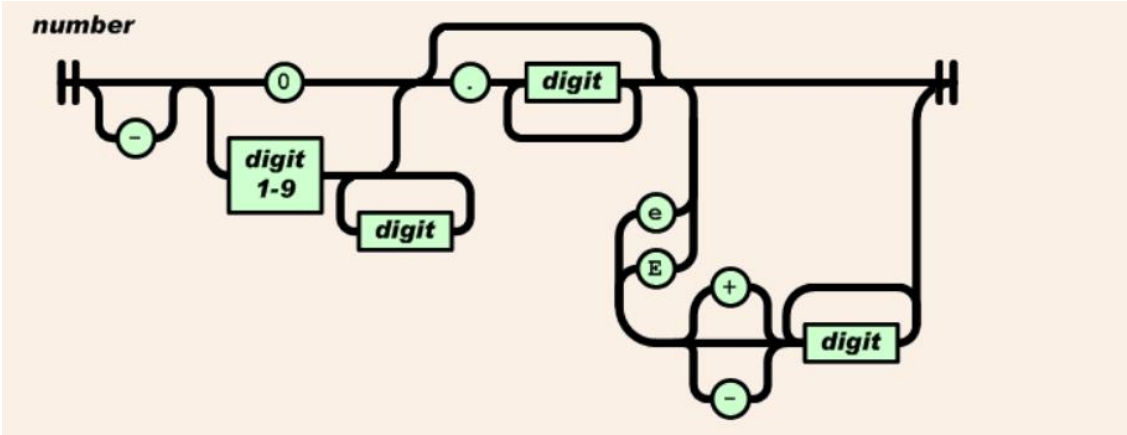
Json数据格式介绍和使用

- 数字



Json数据格式介绍和使用

- 数字



- 坑爹的地方
  - Json中数字是使用double进行存储，最大值只支持，2~53-1。
  - 如果程序使用int64或uint64，可能存在溢出的风险。
  - 这种情况下，只用string来传递

Json数据格式介绍和使用

- 简单使用：
  - 导入 encoding/json
  - json.Marshal
  - json.Unmarshal

Json数据格式介绍和使用

- tag使用
  - 定制json中的key名字
  - 指定数据类型， string,number,boolean
  - 忽略空值，omitempty
  - 忽略字段，-

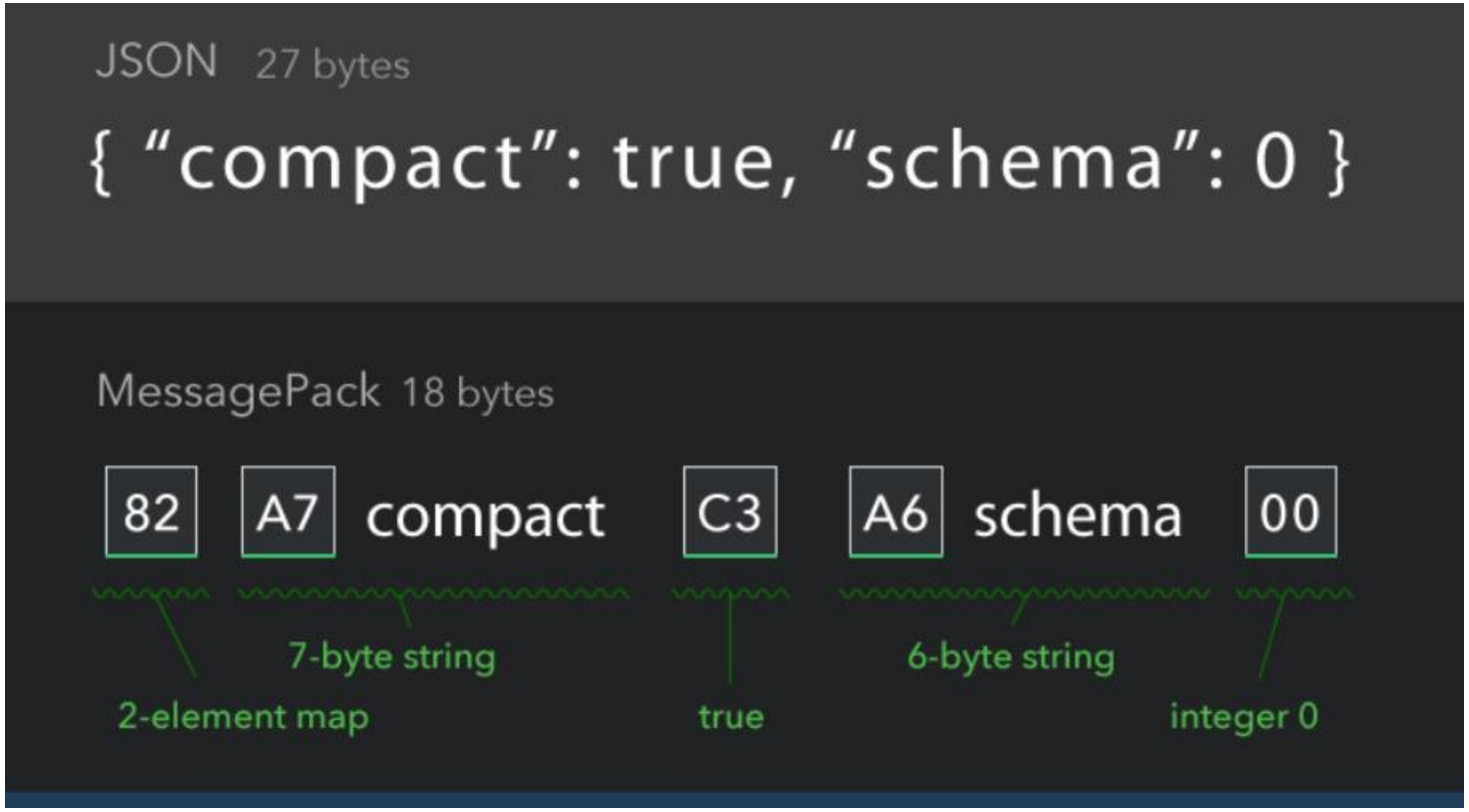
```go
package main

import (
        "encoding/json"
        "fmt"
)

type A struct {
        Id int64 `json:"id,string,omitempty"`
        Name string `json:"-"`
}

func main() {

        var a = &A{Id:38822}
        data, _ := json.Marshal(a)
        fmt.Printf("%s\n", string(data))

        var b  A
        json.Unmarshal(data, &b)
        fmt.Printf("b=%#v\n", b)
}
```

Json数据格式介绍和使用

- 自定义Marshal和Unmarshal
  - 灵活方便
- 自定义时间类型

MsgPack介绍和使用

- Json的二进制版本
  - 空间占用小
  - 序列化和反序列化性能高



MsgPack介绍和使用

- 导入："github.com/vmihailenco/msgpack“
- 调用msgpack.Marshal进行序列化
- 调用msgpack.Unmarshal进行反序列化

```
type Person struct {
    Name string
    Age  int
    Sex  string
}

func writeJson(filename string) (err error) {
    var persons []*Person
    for i := 0; i < 10; i++ {
        p := &Person{
            Name: fmt.Sprintf("name%d", i),
            Age:  rand.Intn(100),
            Sex:  "Man",
        }

        persons = append(persons, p)
    }

    data, err := msgpack.Marshal(persons)
    if err != nil {
        fmt.Printf("=marshal failed, err:%v\n", err)
        return
    }

    err = ioutil.WriteFile(filename, data, 0755)
    if err != nil {
        fmt.Printf("write file failed, err:%v\n", err)
        return
    }

    return
}

func readJson(filename string) (err error) {
    var persons []*Person
    data, err := ioutil.ReadFile(filename)
    if err != nil {
        return
    }

    err = msgpack.Unmarshal(data, &persons)
```

MsgPack介绍和使用

- 使用代码自动生成
  - go get github.com/tinylib/msgp
  - 通过 go:generate msgp指定生成msgpack序列化的代码

```go
package main

import (
        "encoding/json"
        "fmt"
)

//go:generate msgp
type A struct {
        Id int64 `json:"id,string,omitempty" msg:"id"`
        Name string `json:"-" msg:"name"`
}

func main() {

        var a = &A{Id:38822}
        data, _ := json.Marshal(a)
        fmt.Printf("%s\n", string(data))

        var b  A
        json.Unmarshal(data, &b)
        fmt.Printf("b=%#v\n", b)
```

Protobuf介绍和使用

- Google推出的序列化协议
  - 灵活的、高效的、自动化的用于对结构化数据进行序列化的协议
  - 二进制
  - 基于IDL（接口描述语言）的自动化代码生成

Protobuf介绍和使用

- protobuf开发流程
  - IDL编写
  - 生成指定语言的代码
  - 序列化和反序列化

Protobuf介绍和使用

- IDL编写

.proto Type	Notes	C++ Type	Java Type	Python Type[2]	Go Type
double		double	double	float	float64
float		float	float	float	float32
int32	使用变长编码，对于负值的效率很低，如果你的域有可能有负值，请使用sint64替代	int32	int	int	int32
uint32	使用变长编码	uint32	int	int/long	uint32
uint64	使用变长编码	uint64	long	int/long	uint64
sint32	使用变长编码，这些编码在负值时比int32高效的多	int32	int	int	int32
sint64	使用变长编码，有符号的整型值。编码时比通常的int64高效。	int64	long	int/long	int64
fixed32	总是4个字节，如果数值总是比总是比228大的话，这个类型会比uint32高效。	uint32	int	int	uint32
fixed64	总是8个字节，如果数值总是比总是比256大的话，这个类型会比uint64高效。	uint64	long	int/long	uint64
sfixed32	总是4个字节	int32	int	int	int32
sfixed64	总是8个字节	int64	long	int/long	int64
bool		bool	boolean	bool	bool
string	一个字符串必须是UTF-8编码或者7-bit ASCII编码的文本。	string	String	str/unicode	string
bytes	可能包含任意顺序的字节数据。	string	ByteString	str	[]byte

Protobuf介绍和使用

- 枚举定义

```
enum EnumAllowingAlias {
     UNKNOWN = 0;
     STARTED = 1;
     RUNNING = 2;
}
```

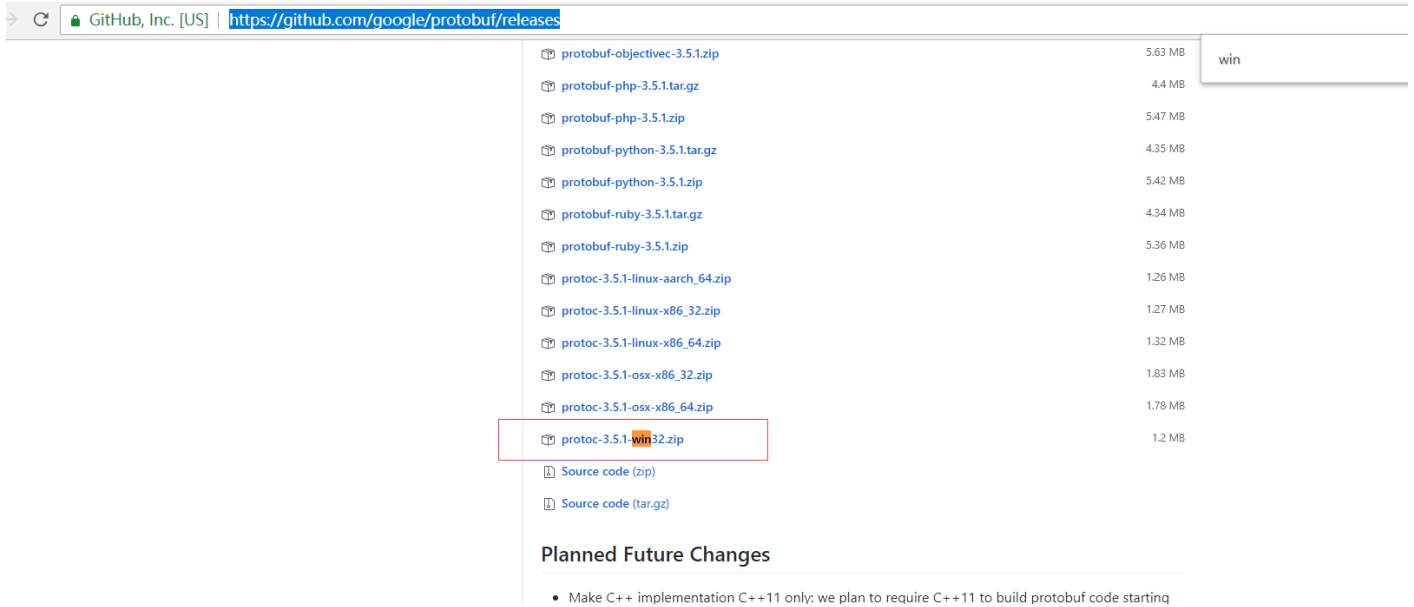
Protobuf介绍和使用

- 结构体定义

```
message Person {
    //后面的数字表示标识号
    int32 id = 1;
    string name = 2;
    //repeated表示可重复
    //可以有多个手机
    repeated Phone phones = 3;
}
```

Protobuf介绍和使用

- 工具安装
  - 安装protoc编译器，解压后拷贝到GOPATH/bin目录下
  - https://github.com/google/protobuf/releases
  - 安装golang代码插件，go get -u github.com/golang/protobuf/protoc-gen-go



Protobuf介绍和使用

- 代码生成
  - 生成代码, protoc --go_out=. xxx.proto
- 使用
  - 导入 github.com/golang/protobuf/proto
  - proto.Marshal序列化
  - proto.Unmarshal反序列化