链滴

[gev] Go 语言优雅处理 TCP “粘包”

作者：Allenxuxu
原文链接：https://ld246.com/article/1572572128433
来源网站：链滴

https://github.com/Allenxuxu/gev

gev 是一个轻量、快速的基于 Reactor 模式的非阻塞 TCP 网络库，支持自定义协议，轻松快速搭建性能服务器。

TCP 为什么会"粘包"

TCP 本身就是面向流的协议，就是一串没有界限的数据。所以本质上来说 TCP 粘包是一个伪命题。

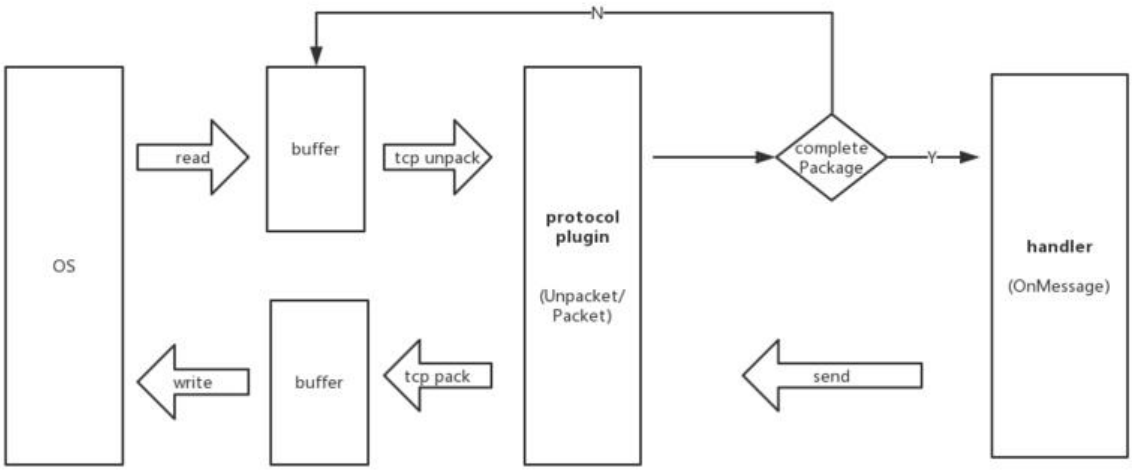
TCP 底层并不关心上层业务数据，会套接字缓冲区的实际情况进行包的划分，一个完整的业务数据可会被拆分成多次进行发送，也可能会将多个小的业务数据封装成一个大的数据包发送（Nagle算法）。

gev 如何优雅处理

gev 通过回调函数 OnMessage 通知用户数据到来，回调函数中会将用户数据缓冲区（ringbuffer）过参数传递过来。

用户通过对 ringbuffer 操作，来进行数据解包，获取到完整用户数据后再进行业务操作。这样又一明显的缺点，就是会让业务操作和自定义协议解析代码堆在一起。

所以，最近对 gev 进行了一次较大改动，主要是为了能够以插件的形式支持各种自定义的数据协议，使用者可以便捷处理 TCP 粘包问题，专注于业务逻辑。



做法如下，定义一个接口 Protocol

```
// Protocol 自定义协议编解码接口
type Protocol interface {
    UnPacket(c *Connection, buffer *ringbuffer.RingBuffer) (interface{}, []byte)
    Packet(c *Connection, data []byte) []byte
}
```

用户只需实现这个接口，并注册到 server 中，当客户端数据到来时，gev 会首先调用 UnPacket 方，如果缓冲区中的数据足够组成一帧，则将数据解包，并返回真正的用户数据，然后在回调 OnMess ge 函数并将数据通过参数传递。

下面，我们实现一个简单的自定义协议插件，来启动一个 Server：

```
| 数据长度 n |  payload |
|  4字节    |  n 字节   |

// protocol.go
package main

import (
    "encoding/binary"
    "github.com/Allenxuxu/gev/connection"
    "github.com/Allenxuxu/ringbuffer"
    "github.com/gobwas/pool/pbytes"
)

const exampleHeaderLen = 4

type ExampleProtocol struct{}

func (d *ExampleProtocol) UnPacket(c *connection.Connection, buffer *ringbuffer.RingBuffer)
interface{}, []byte) {
    if buffer.VirtualLength() > exampleHeaderLen {
        buf := pbytes.GetLen(exampleHeaderLen)
        defer pbytes.Put(buf)
        _, _ = buffer.VirtualRead(buf)
        dataLen := binary.BigEndian.Uint32(buf)

        if buffer.VirtualLength() >= int(dataLen) {
            ret := make([]byte, dataLen)
            _, _ = buffer.VirtualRead(ret)

            buffer.VirtualFlush()
            return nil, ret
        } else {
            buffer.VirtualRevert()
        }
    }
    return nil, nil
}

func (d *ExampleProtocol) Packet(c *connection.Connection, data []byte) []byte {
    dataLen := len(data)
    ret := make([]byte, exampleHeaderLen+dataLen)
    binary.BigEndian.PutUint32(ret, uint32(dataLen))
    copy(ret[4:], data)
    return ret
}

// server.go
package main

import (
    "flag"
    "log"
```

```
    "strconv"

    "github.com/Allenxuxu/gev"
    "github.com/Allenxuxu/gev/connection"
)

type example struct{}

func (s *example) OnConnect(c *connection.Connection) {
    log.Println(" OnConnect :  ", c.PeerAddr())
}
func (s *example) OnMessage(c *connection.Connection, ctx interface{}, data []byte) (out []byt
) {
    log.Println("OnMessage:  ", data)
    out = data
    return
}

func (s *example) OnClose(c *connection.Connection) {
    log.Println("OnClose")
}

func main() {
    handler := new(example)
    var port int
    var loops int

    flag.IntVar(&port, "port", 1833, "server port")
    flag.IntVar(&loops, "loops", -1, "num loops")
    flag.Parse()

    s, err := gev.NewServer(handler,
        gev.Address(":"+strconv.Itoa(port)),
        gev.NumLoops(loops),
        gev.Protocol(&ExampleProtocol{}))
    if err != nil {
        panic(err)
    }

    log.Println("server start")
    s.Start()
}
```

完整代码地址

当回调 OnMessage 函数的时候，会通过参数传递已经拆好包的用户数据。

当我们需要使用其他协议时，仅仅需要实现一个 Protocol 插件，然后只要 gev.NewServer 时指定即：

```
gev.NewServer(handler, gev.NumLoops(2), gev.Protocol(&XXXProtocol{}))
```

基于 Protocol Plugins 模式为 gev 实现 WebSocket 插件

得益于 Protocol Plugins 模式的引进，我可以将 WebSocket 的实现做成一个插件（WebSocket 协构建在 TCP 之上），独立于 gev 之外。

```
package websocket

import (
    "log"

    "github.com/Allenxuxu/gev/connection"
    "github.com/Allenxuxu/gev/plugins/websocket/ws"
    "github.com/Allenxuxu/ringbuffer"
)

// Protocol websocket
type Protocol struct {
    upgrade *ws.Upgrader
}

// New 创建 websocket Protocol
func New(u *ws.Upgrader) *Protocol {
    return &Protocol{upgrade: u}
}

// UnPacket 解析 websocket 协议，返回 header， payload
func (p *Protocol) UnPacket(c *connection.Connection, buffer *ringbuffer.RingBuffer) (ctx inter
ace{}, out []byte) {
    upgraded := c.Context()
    if upgraded == nil {
        var err error
        out, _, err = p.upgrade.Upgrade(buffer)
        if err != nil {
            log.Println("Websocket Upgrade :", err)
            return
        }
        c.SetContext(true)
    } else {
        header, err := ws.VirtualReadHeader(buffer)
        if err != nil {
            log.Println(err)
            return
        }
        if buffer.VirtualLength() >= int(header.Length) {
            buffer.VirtualFlush()

            payload := make([]byte, int(header.Length))
            _, _ = buffer.Read(payload)

            if header.Masked {
                ws.Cipher(payload, header.Mask, 0)
            }

            ctx = &header
            out = payload
        } else {
```

```
            buffer.VirtualRevert()
        }
    }
    return
}

// Packet 直接返回
func (p *Protocol) Packet(c *connection.Connection, data []byte) []byte {
    return data
}
```

具体的实现，可以到仓库的 plugins/websocket 查看。

相关文章

- 开源 gev: Go 实现基于 Reactor 模式的非阻塞 TCP 网络库
- Go 网络库并发吞吐量测试

项目地址

https://github.com/Allenxuxu/gev