NATS重复ID的处理

1:让每一个客户端上传自己的DevId给服务器端,服务器端接收到并存在map中,map的结构如下：clients map[uint64]\*client

在客户端的nats.go文件的以下部分增加DevId字段：

const (

    Version = "1.2.0"

    DefaultURL = "nats://localhost:4222"

    DefaultPort = 4222

    DefaultMaxReconnect = 60

    DefaultReconnectWait = 2 \* time.Second

    DefaultTimeout = 2 \* time.Second

    DefaultPingInterval = 2 \* time.Minute

    DefaultMaxPingOut = 2

    DefaultMaxChanLen = 8192 // 8k

    DefaultReconnectBufSize = 8 \* 1024 \* 1024 // 8MB

    RequestChanLen = 8

    LangString = "go"

**DevId = "devidtp"**

)

type connectInfo struct {

    Verbose bool `json:"verbose"`

    Pedantic bool `json:"pedantic"`

    User string `json:"user,omitempty"`

    Pass string `json:"pass,omitempty"`

    Token string `json:"auth\_token,omitempty"`

    TLS bool `json:"tls\_required"`

    Name string `json:"name"`

    Lang string `json:"lang"`

    Version string `json:"version"`

**DevId string `json:"devId"`**

}

func (nc \*Conn) connectProto() (string, error) {

    o := nc.Opts

    var user, pass, token string

    u := nc.url.User

    if u != nil {

        // if no password, assume username is authToken

        if \_, ok := u.Password(); !ok {

            token = u.Username()

        } else {

            user = u.Username()

            pass, \_ = u.Password()

        }

    }

    cinfo := connectInfo{o.Verbose, o.Pedantic,

        user, pass, token,

        o.Secure, o.Name, LangString, Version, **DevId**}增加**DevId**到cinfo

    b, err := json.Marshal(cinfo)

    if err != nil {

        return \_EMPTY\_, ErrJsonParse

    }

    return fmt.Sprintf(conProto, b), nil

}

发送cinfo到服务器端

func (nc \*Conn) sendConnect() error {

    // Construct the CONNECT protocol string

    cProto, err := nc.connectProto()

    if err != nil {

        return err

    }

    // Write the protocol into the buffer

**\_, err = nc.bw.WriteString(cProto)**

    fmt.Println("C send INFO:", cProto)

    if err != nil {

        return err

    }

    // Add to the buffer the PING protocol

**\_, err = nc.bw.WriteString(pingProto)**

    if err != nil {

        return err

    }

为了在服务器端接收到DevId,在client.go的以下部分增加DevId字段:

type clientOpts struct {

    Verbose bool `json:"verbose"`

    Pedantic bool `json:"pedantic"`

    SslRequired bool `json:"ssl\_required"`

    Authorization string `json:"auth\_token"`

    Username string `json:"user"`

    Password string `json:"pass"`

    Name string `json:"name"`

    Lang string `json:"lang"`

    Version string `json:"version"`

    DevId string `json:"devId"`

}

在服务器端的read函数接收cinfo中的字段：

func (c \*client) readLoop() {

    // Grab the connection off the client, it will be cleared on a close.

    // We check for that after the loop, but want to avoid a nil dereference

    c.mu.Lock()

    nc := c.nc

    s := c.srv

    defer s.grWG.Done()

    c.mu.Unlock()

    if nc == nil {

        return

    }

    // Start read buffer.

    b := make([]byte, startBufSize)

    for {

        fmt.Println("S C bf read:", string(b))

        fmt.Println("S C cid:", c.cid)

**n, err := nc.Read(b)**

        fmt.Println("S C read n:", n)

        fmt.Println("S C cid:", c.cid)

2:在client.go文件中增加服务器端接收到重复客户端id的逻辑处理代码：

func (c \*client) readLoop() {

    // Grab the connection off the client, it will be cleared on a close.

    // We check for that after the loop, but want to avoid a nil dereference

    c.mu.Lock()

    nc := c.nc

    s := c.srv

    defer s.grWG.Done()

    c.mu.Unlock()

    if nc == nil {

        return

    }

    // Start read buffer.

    b := make([]byte, startBufSize)

    for {

        fmt.Println("S C bf read:", string(b))

        fmt.Println("S C cid:", c.cid)

        n, err := nc.Read(b)

        fmt.Println("S C read n:", n)

        fmt.Println("S C cid:", c.cid)

        //maybe can check b[] with "CONNECT"

        fmt.Println("S C af read:", string(b))

        if err != nil {

            c.closeConnection()

            return

        }

        // Grab for updates for last activity.

        last := time.Now()

        // Clear inbound stats cache

        c.cache.inMsgs = 0

        c.cache.inBytes = 0

        c.cache.subs = 0

        if err := c.parse(b[:n]); err != nil {

            // handled inline

            if err != ErrMaxPayload && err != ErrAuthorization {

                c.Errorf("Error reading from client: %s", err.Error())

                c.sendErr("Parser Error")

                c.closeConnection()

            }

            return

        }

        if n == 129 {

            fmt.Println("S C \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            for Tcid, Tclient := range s.clients {

                fmt.Println("S C cid:", Tcid)

                fmt.Println("client:", Tclient.opts.DevId)

                if Tcid == c.cid {

                    continue

                }

                if c.opts.DevId == Tclient.opts.DevId {

                    fmt.Println("S C bf process: send PING")

                    Tclient.mu.Lock()

                    Tclient.bw.WriteString("PING\r\n")

                    err := Tclient.bw.Flush()

                    if err != nil {

                        fmt.Println("S C T will clear")

                        //                        Tclient.clearConnection()

                        //                        Tclient.mu.Unlock()

//链接1连接上来之后断掉了，但服务器端没有释放服务器还不知

，当链接2连接上来之后，DevId与链接1重复，服务器就会给1

发ping,出现error之后，服务器就会认为这个是重新连接

相同DevId的连接，有且只保留一个

                        continue

                    } else {

                        fmt.Println("S C devId real exists & will close this connection")

                        c.closeConnection()

                    }

                    Tclient.mu.Unlock()

                    break

                } else {

                    fmt.Println("S C devId not exist")

                    continue

                }

            }

            fmt.Println("S C ~~~~~~~~~~~~~~~~~~~~~")

        }

改完n=129之后的client.go的代码：

func (c \*client) readLoop() {

    // Grab the connection off the client, it will be cleared on a close.

    // We check for that after the loop, but want to avoid a nil dereference

    c.mu.Lock()

    nc := c.nc

    s := c.srv

    defer s.grWG.Done()

    c.mu.Unlock()

    if nc == nil {

        return

    }

    // Start read buffer.

    b := make([]byte, startBufSize)

    for {

        fmt.Println("S C bf read:", string(b))

        fmt.Println("S C cid:", c.cid)

        n, err := nc.Read(b)

        fmt.Println("S C read n:", n)

        fmt.Println("S C cid:", c.cid)

        //maybe can check b[] with "CONNECT"

        var flag bool

        flag = false

        fmt.Println("len(b):", len(b))

        fmt.Println("string(b[:6]):", string(b[:7]))

        if len(b) > 7 {

            fmt.Println("b[0]:", string(b[:7]))

            s := strings.EqualFold(string(b[:7]), "connect")

            fmt.Println(s)

            if strings.EqualFold(string(b[:7]), "connect") {

                flag = true

            }

        }

        fmt.Println("S C af read:", string(b))

        if err != nil {

            c.closeConnection()

            return

        }

        // Grab for updates for last activity.

        last := time.Now()

        // Clear inbound stats cache

        c.cache.inMsgs = 0

        c.cache.inBytes = 0

        c.cache.subs = 0

        if err := c.parse(b[:n]); err != nil {

            // handled inline

            if err != ErrMaxPayload && err != ErrAuthorization {

                c.Errorf("Error reading from client: %s", err.Error())

                c.sendErr("Parser Error")

                c.closeConnection()

            }

            return

        }

        if flag == true {

            flag = false

            fmt.Println("S C \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            for Tcid, Tclient := range s.clients {

                fmt.Println("S C cid:", Tcid)

                fmt.Println("client:", Tclient.opts.DevId)

                if Tcid == c.cid {

                    continue

                }

                if c.opts.DevId == Tclient.opts.DevId {

                    fmt.Println("S C bf process: send PING")

                    Tclient.mu.Lock()

                    Tclient.bw.WriteString("PING\r\n")

                    err := Tclient.bw.Flush()

                    if err != nil {

                        fmt.Println("S C T will clear")

                        //                        Tclient.clearConnection()

                        //                        Tclient.mu.Unlock()

                        continue

                    } else {

                        fmt.Println("S C devId real exists & will close this connection")

                        c.closeConnection()

                    }

                    Tclient.mu.Unlock()

                    break

                } else {

                    fmt.Println("S C devId not exist")

                    continue

                }

            }

            fmt.Println("S C ~~~~~~~~~~~~~~~~~~~~~")

        }

3: 服务器检测到重复id关掉相应连接之后，连接进行重连问题的解决修改客户端nats.go文件：

func (nc \*Conn) readLoop() {

    // Release the wait group on exit

    defer nc.wg.Done()

    // Create a parseState if needed.

    nc.mu.Lock()

    if nc.ps == nil {

        nc.ps = &parseState{}

    }

    nc.mu.Unlock()

    // Stack based buffer.

    b := make([]byte, defaultBufSize)

    for {

        // FIXME(dlc): RWLock here?

        nc.mu.Lock()

        sb := nc.isClosed() || nc.isReconnecting()

        if sb {

            nc.ps = &parseState{}

        }

        conn := nc.conn

        nc.mu.Unlock()

        if sb || conn == nil {

            break

        }

        fmt.Println("bf read:conn", conn)

        n, err := conn.Read(b)

        fmt.Println("af read", string(b))

        if err != nil {

            fmt.Println("nc.processOpErr(err):", err)

            //nc.processOpErr(err)

            break

        }

Github:git@github.com:hyhlinux/tpnats.git上的测试用例:

1,2,3客户端连接服务器,4连接服务器的时候上传的devid与1的相同,因为1存在,所以杀掉4的连接;如果1不存在,那么就让4连接进来.