■**文档属性**

|  |  |
| --- | --- |
| 标题 | Mssql指令解析 |
| 编号 |  |
| 编写日期 | 2018-2-28 |
| 修改日期 |  |
| 作者 |  |
| 产品名称 |  |
| 产品型号 |  |
| 产品版本号 |  |

文档修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 修订说明 | 修订人 |
| 2018-2-28 | V1.0 | 创建 | 周擎阳 |
|  |  |  |  |

**目 录**

[1. TDS 命令过程分析 3](#_Toc8163758)

[1. 格式分析 3](#_Toc273176801)

[1.1 发送命令阶段 3](#_Toc1860608236)

[1.1.2 回应阶段 4](#_Toc629085408)

[2. Wireshark 抓包 4](#_Toc2009774950)

[3. Dbshield 分析 5](#_Toc214927082)

# TDS 命令过程分析

## 格式分析

### 1.1 发送命令阶段

在微软给出的tds文档中，对于tds的格式做了详细的规定。由于sqlserver默认采取的是批处理形式，所以命令过程称为sql批处理（sql batch）语句。

根据相关文档，Tds格式如下

|  |  |  |
| --- | --- | --- |
| Package\_header | All\_headers | SQLtext |

Package\_header就是普通的tds数据包的表头，其中特殊的地方在于，他的type数值为0x01。

下面是package\_header 的视图：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type（1） = 0x01 | Status（1） | Length（2） | SPID（2） | PackID（1） | Window（1） |

All\_headers 是tds自己规定的数据部分的开头，他的格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| TotalLength(4) | HeaderLength(4) | headerType(2) | HeaderData |

其中headerData由headerType决定。对于我们sql batch来说，headertype = 0x02，也就是Transaction Descriptor Header，这时headerdata由TransactionDescriptor，OutstandingRequestCount 两部分组成。TransactionDescriptor唯一标示此次会话而outstandingrequestcount记录在这次活动中有多少活跃的访问。

### 1.1.2 回应阶段

回应阶段，包头和发送阶段相同，但是包头的type变成了0x04.

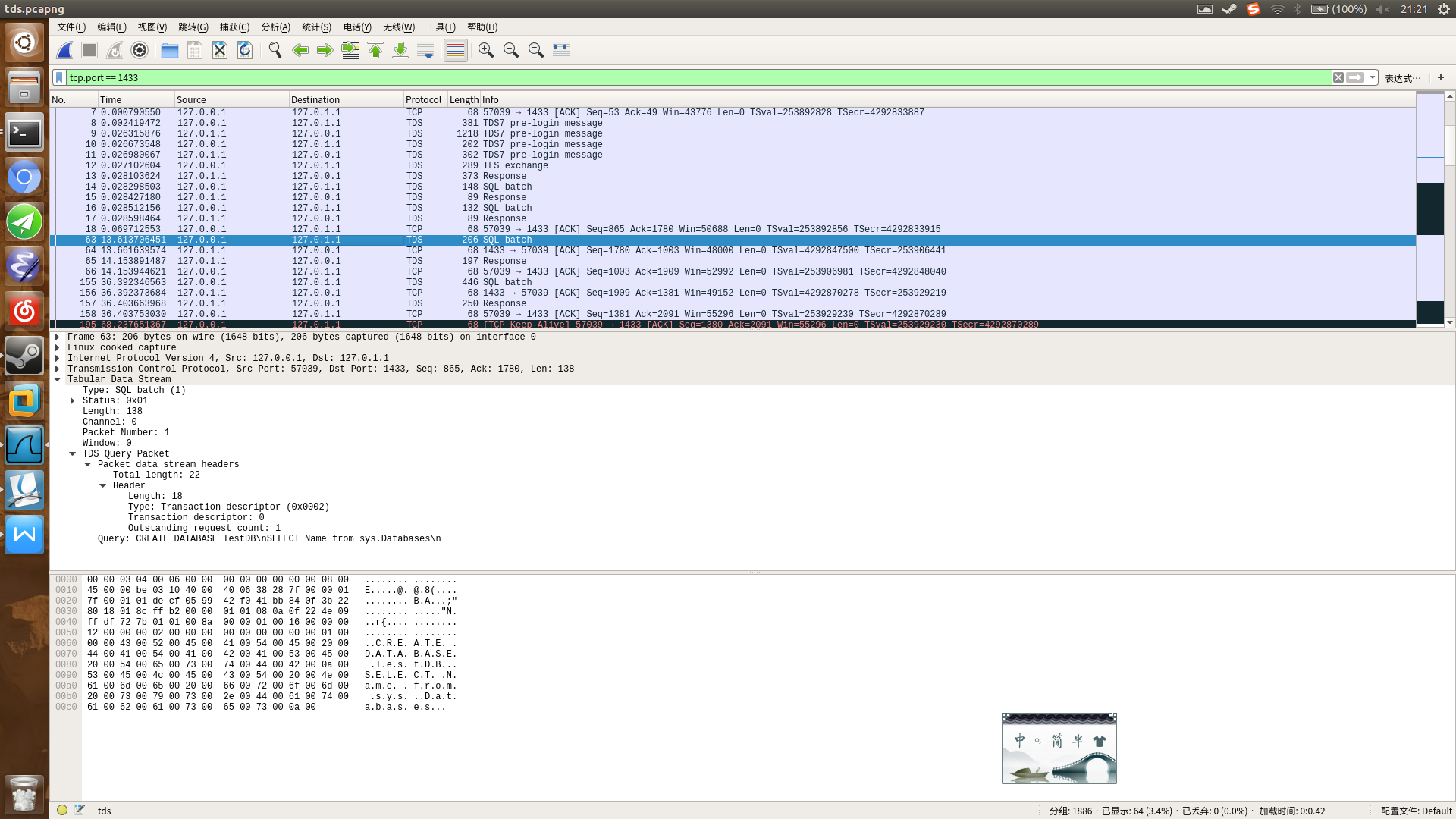
对于回应阶段，数据包类似这个形式：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据包头 | Token1 | ... | tokenn |

对于每一个sql语句，在返回相应的数据之后，我们都会返回一个token为done 的包，证明数据返回完毕。

不同的数据返回了不同的token，

## Wireshark 抓包



和我们的分析一致，证明文档正确。

## Dbshield 分析

下面是go语言对这串代码的分析：

for {

var buf []byte

buf, err = m.reader(m.client)

if err != nil || len(buf) < 8 {

return err

}

switch buf[0] {

case 0x01: //SQL batch

query := buf[8:]

context := sql.QueryContext{

Query: query,

Database: m.currentDB,

User: m.username,

Client: remoteAddrToIP(m.client.RemoteAddr()),

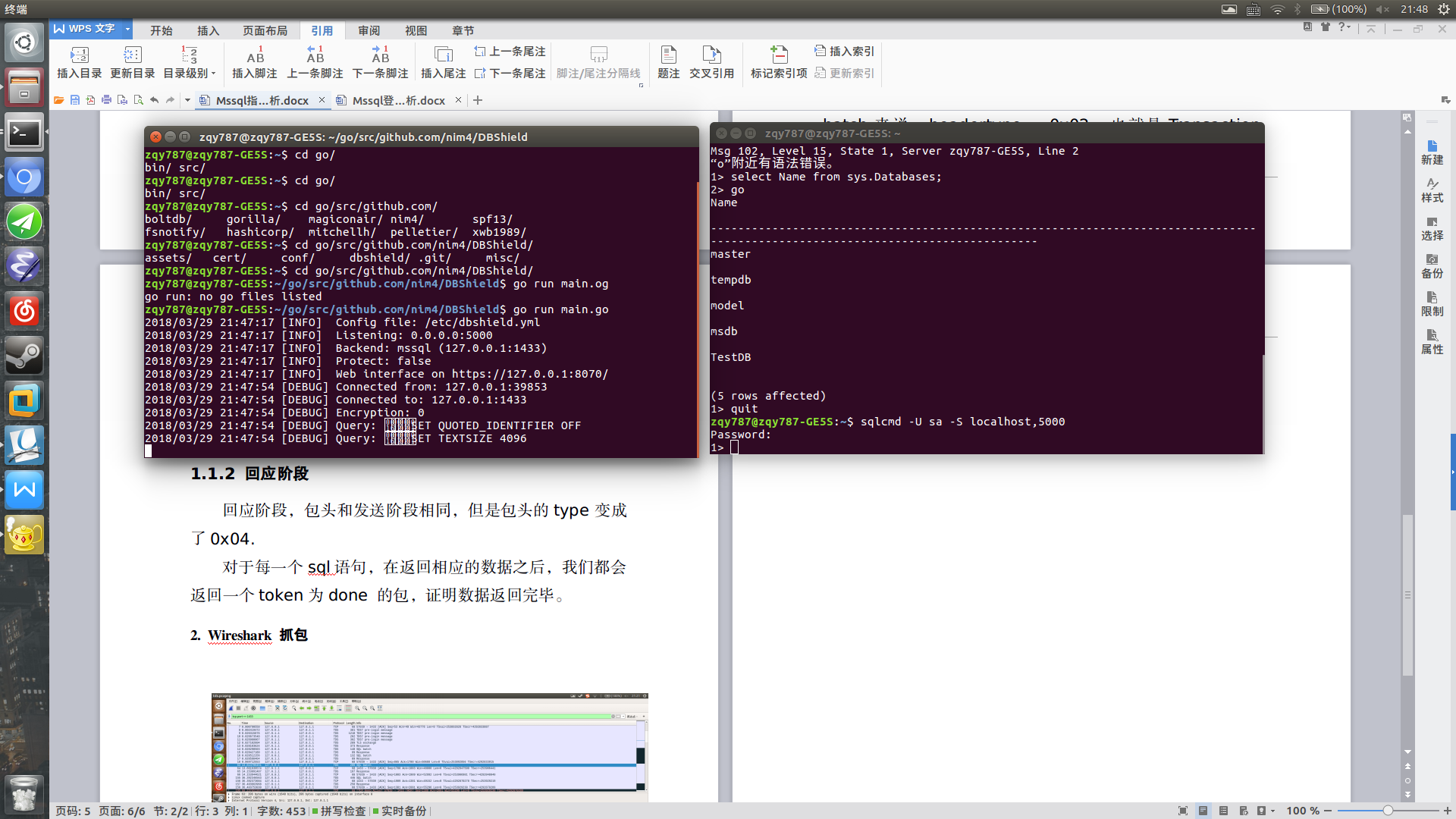
Time: time.Now(),

}

processContext(context)

}

遗憾的是，这串代码也是错的，我们可以由下图看到：



之所以出现乱码的原因，是因为go语言没有把all header部分剔除出去，导致把all header部分也当成了query text导致的。我们进行修改后，可以看到:

for {

var buf []byte

buf, err = m.reader(m.client)

if err != nil || len(buf) < 8 {

return err

}

switch buf[0] {

case 0x01: //SQL batc

curr := 8 + int(buf[8]) + int(buf[9]) \* 16 + int(buf[10]) \* 16 \* 16 + int(buf[11]) \* 16 \* 16 \* 16;

query := buf[curr:]

context := sql.QueryContext{

Query: query,

Database: m.currentDB,

User: m.username,

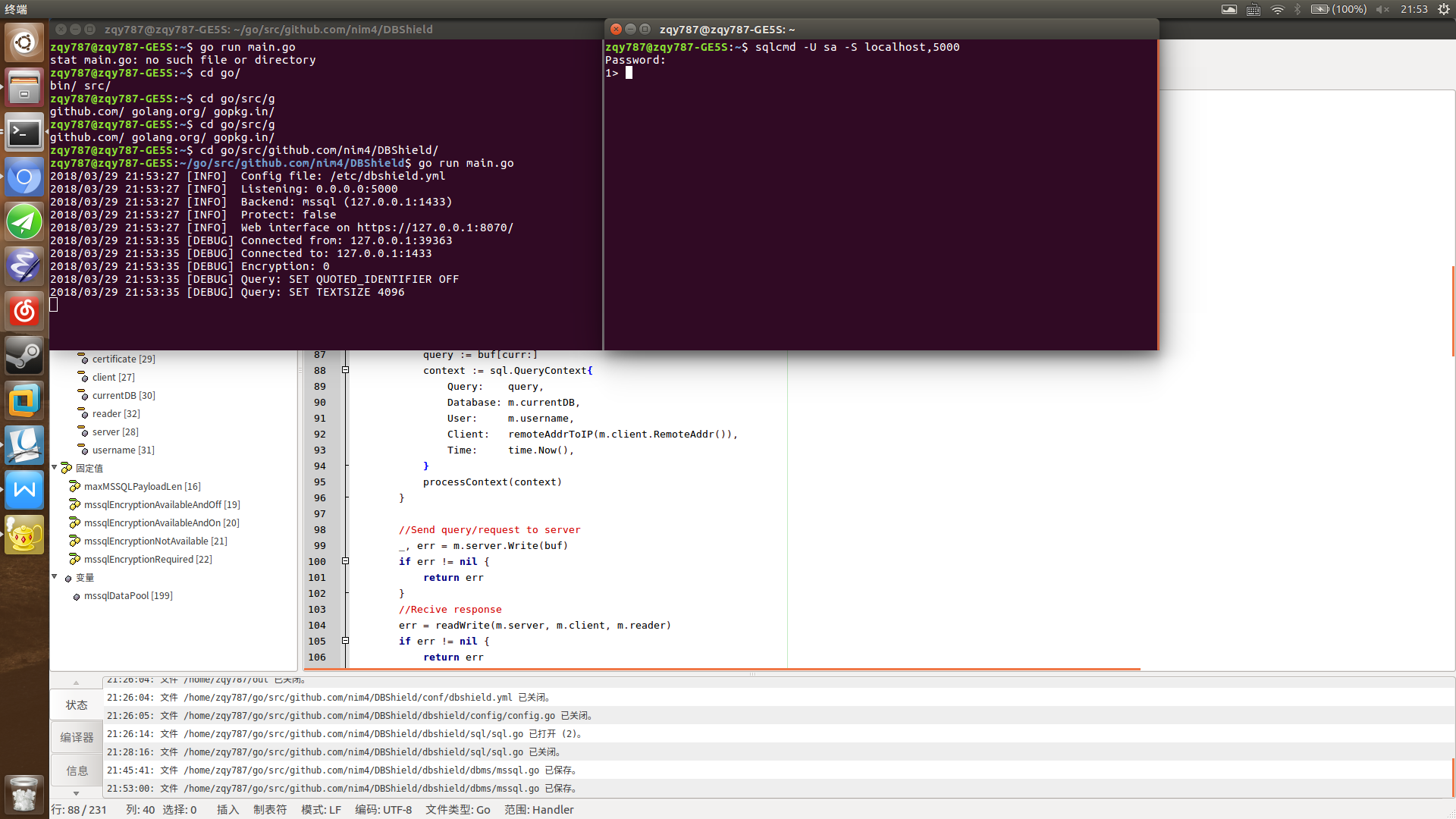
Client: remoteAddrToIP(m.client.RemoteAddr()),

Time: time.Now(),

}

processContext(context)

}



访问正确。

## 操作方法

### 2.1 发送包

|  |  |  |  |
| --- | --- | --- | --- |
| 操作类型 | 报文类型（包第一字节） | 如何定位sql语句 | 备注 |
| 普通sql语句 | 0x01 | 8 + 从第九位开始的dword数据（all header长度） | 连续的sql语句用/n隔开。Go不计入sql语句中 |
| 批处理sql语句 | 0x07 | Row token | 以一个colmetadata开始，后面跟许多row token。最后用done token结尾。 |

### 2.2 接受包

|  |  |  |  |
| --- | --- | --- | --- |
| token | 特征值 | Token意义 | 备注 |
| row | 0xD1 | 描述了一组数据及其格式。 | 必须有一个colmetdata token开头 |
| done | 0xFD | 证明命令被正确执行 | 除了变量定义，所有指令都必须返回一个done token |
| Env change | 0xE3 | 告知环境的变化 |  |
| colmetadata | 0x81 | 描述接下来的row token | 描述了row token的个数和格式。 |
| info | 0xAB | 返回有关信息 | 经常用来返回错误状态。 |