# MSSQL 协议分析

## Mssql基本介绍

Mssql的通信协议底层是TDS协议。目前TDS有许多版本。我们文档支持TDS7.1，TDS7.2，TDS7.3，TDS7.4的某些版本。对于TDS7.4，我们使用的版本会在登录时候进行确定。

下面规定两种格式：

B\_VARCHAR = 1字节长度+数据

US\_VARCHAR = 2字节长度+数据

## TDS 协议概述

TDS的基本格式如下：

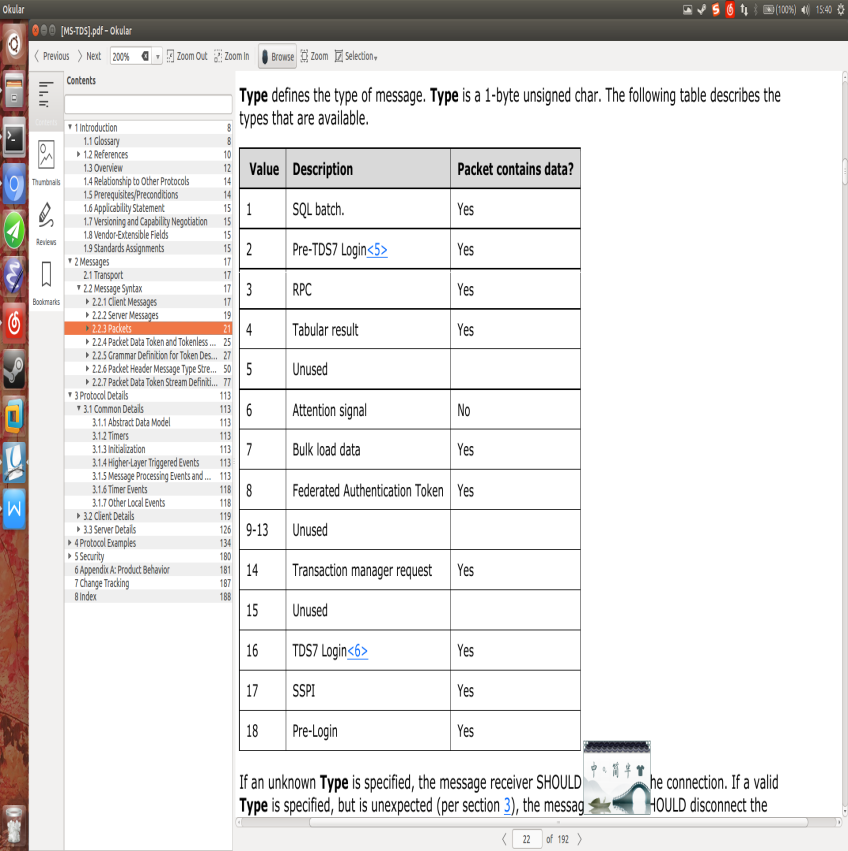
|  |  |
| --- | --- |
| head | data |

对于head部分，我们可以细分为：

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

Type：决定了这个包的作用。Dbshield中就是通过type类型判断这个包属于那种类型，进而推断这个属于那个过程。

所有的type如下：



注意群分2和18：type2表示当前TDS虽然也是在prelogin阶段，但是版本低于7.0. 18是7.0以上版本使用的prelogin头。

Status：决定当前包的状态。所有的status如下：

|  |  |
| --- | --- |
| 0x00 | 正常状态 |
| 0x01 | 消息结束（EOM） |
| 0x02 | 忽略此条消息 |
| 0x08 | 重置连接 |
| 0x10 | 重置连接，但是不改变事务状态 |

Length：包长。数据大端村存储。

Spid：服务器端进程号，多线程时有用。

Packid：包编号，每当收到一个包，编号加一。

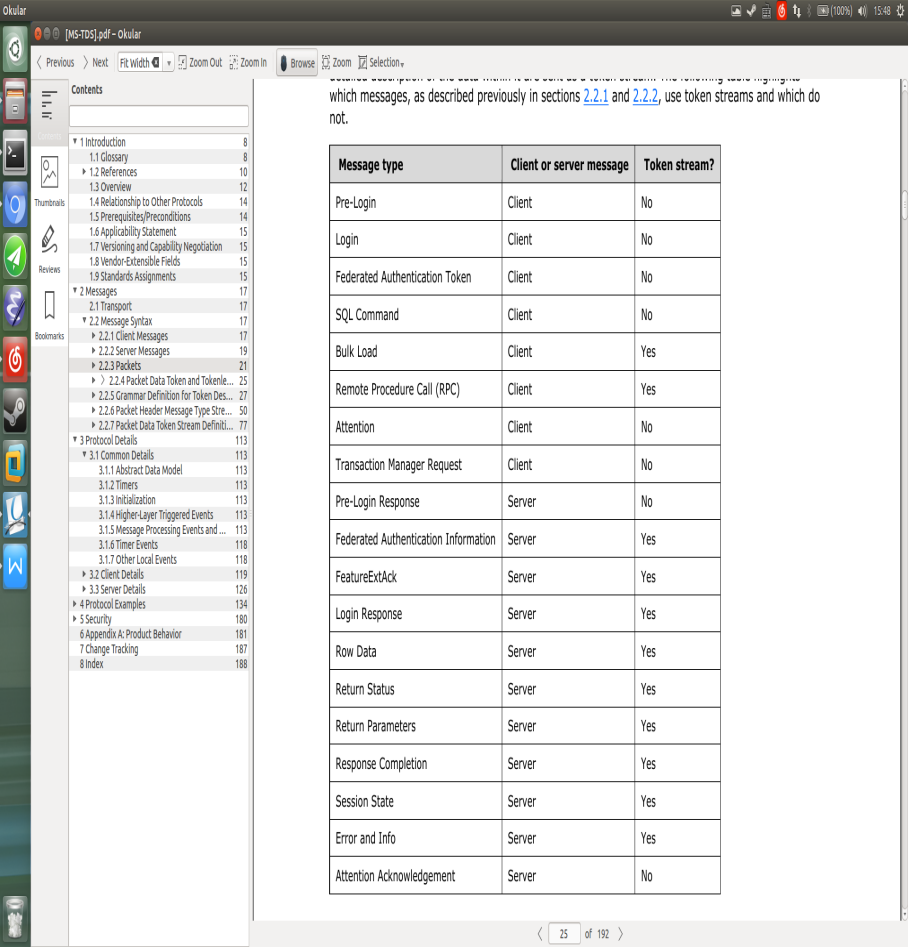
Window：保留位，恒为0x00。

对于data部分，一般有两种存储方式：

Token data： 数据存储在后面的data部分，前面token的类型，偏移值和数据长度等数据，用于定位。

Tokenless data：数据直接存储在当前位置。

每一种是token data还是tokenless data列表如下



## 从客户端请求报文中提取sql语句

请求报文有自己的规则：

|  |  |  |
| --- | --- | --- |
| 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。

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

下面是package\_header 的视图：

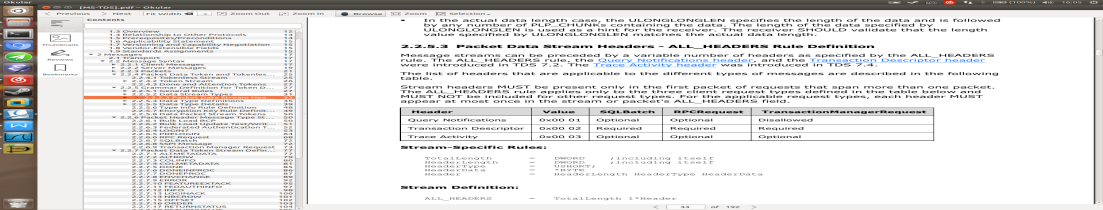
|  |  |  |  |
| --- | --- | --- | --- |
| TotalLength(4) | HeaderLength(4) | headerType(2) | HeaderData |

All\_headers 是tds自己规定的数据部分的开头，他的格式如下：

Totalength：all\_header总长度。可能含有多个header。小端存储。

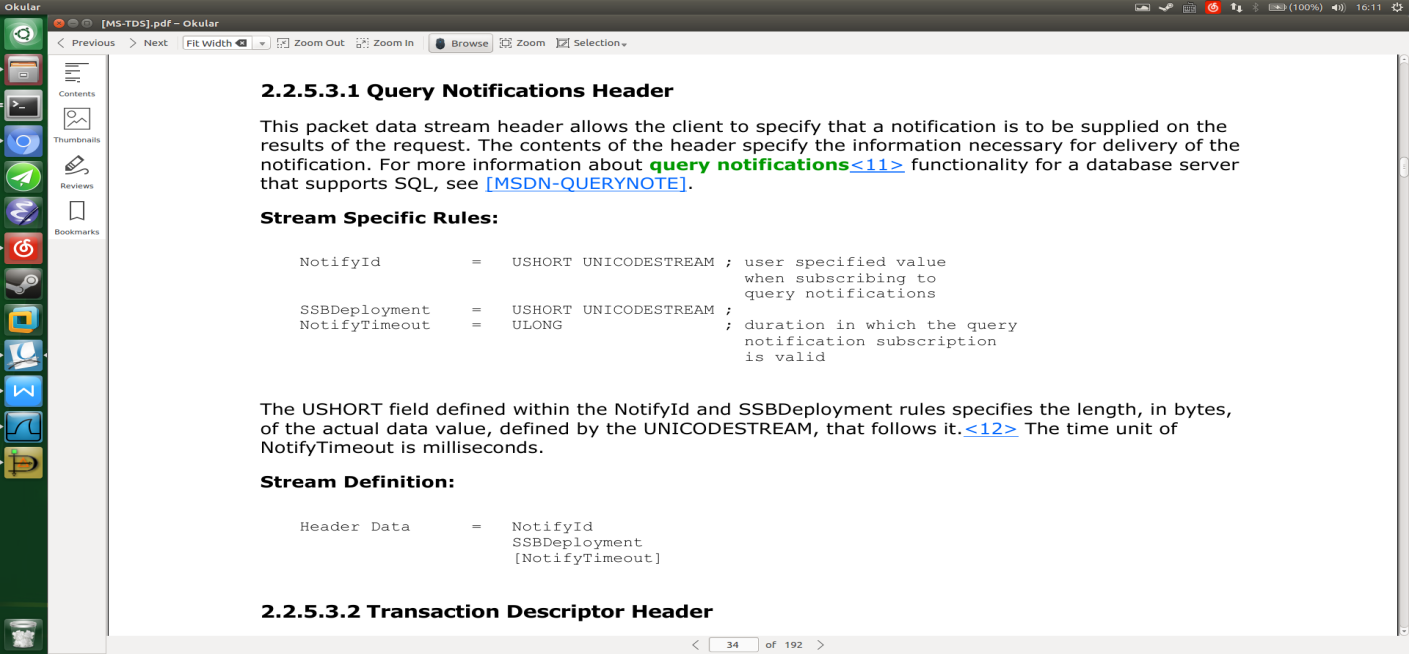
headerLength长度是每一个header长度。

headerType规定了当前head类型：

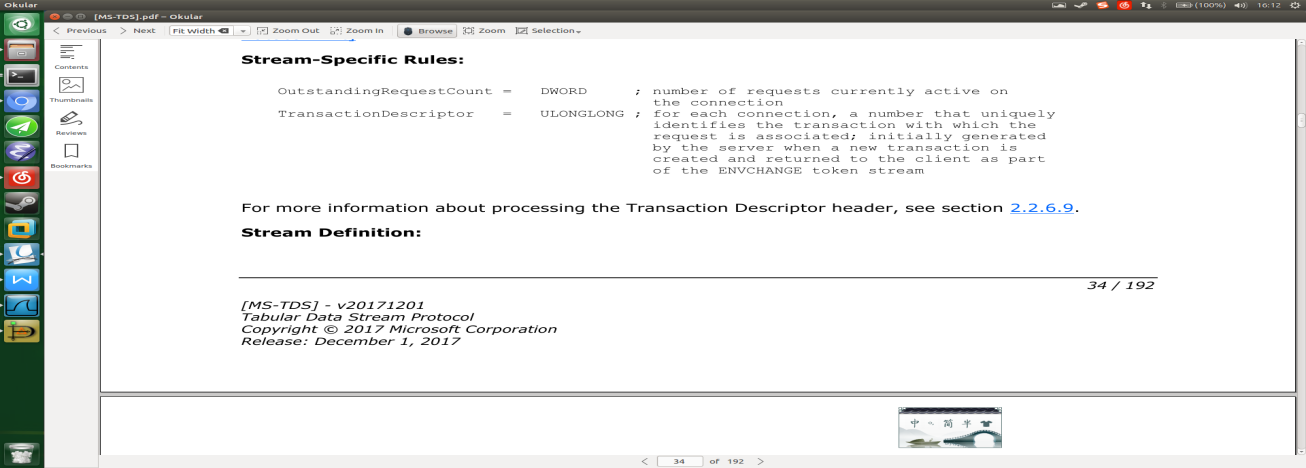


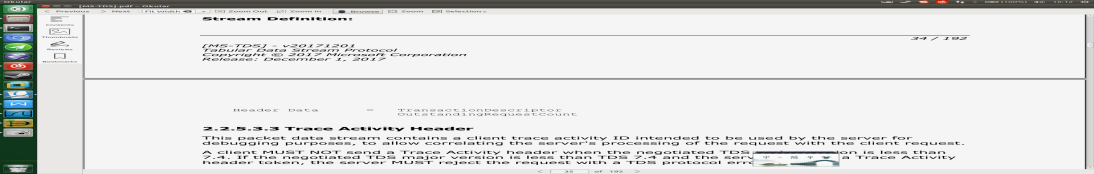
其中headerData由headerType决定。

当headerType = Query Notification时：

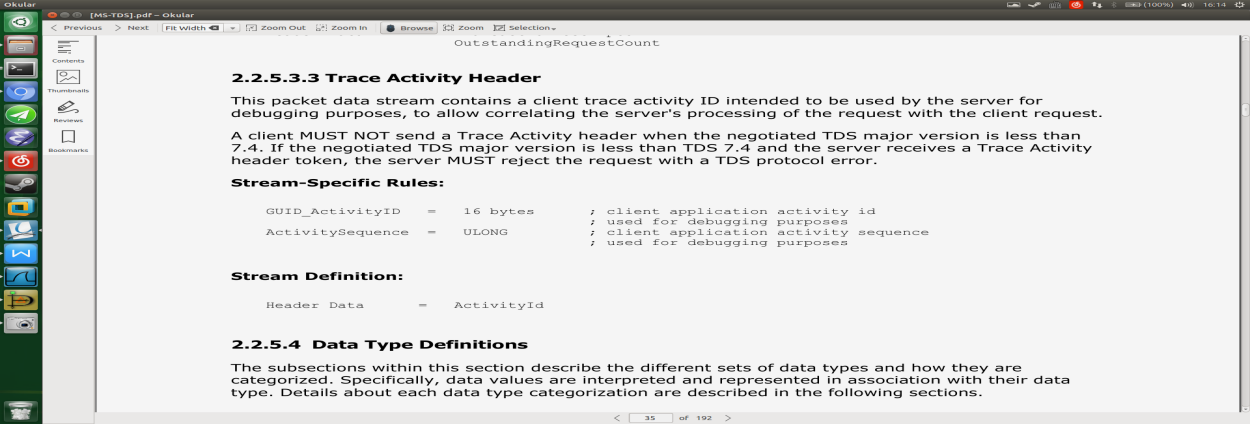


当headerType = Transaction Descriptor时：

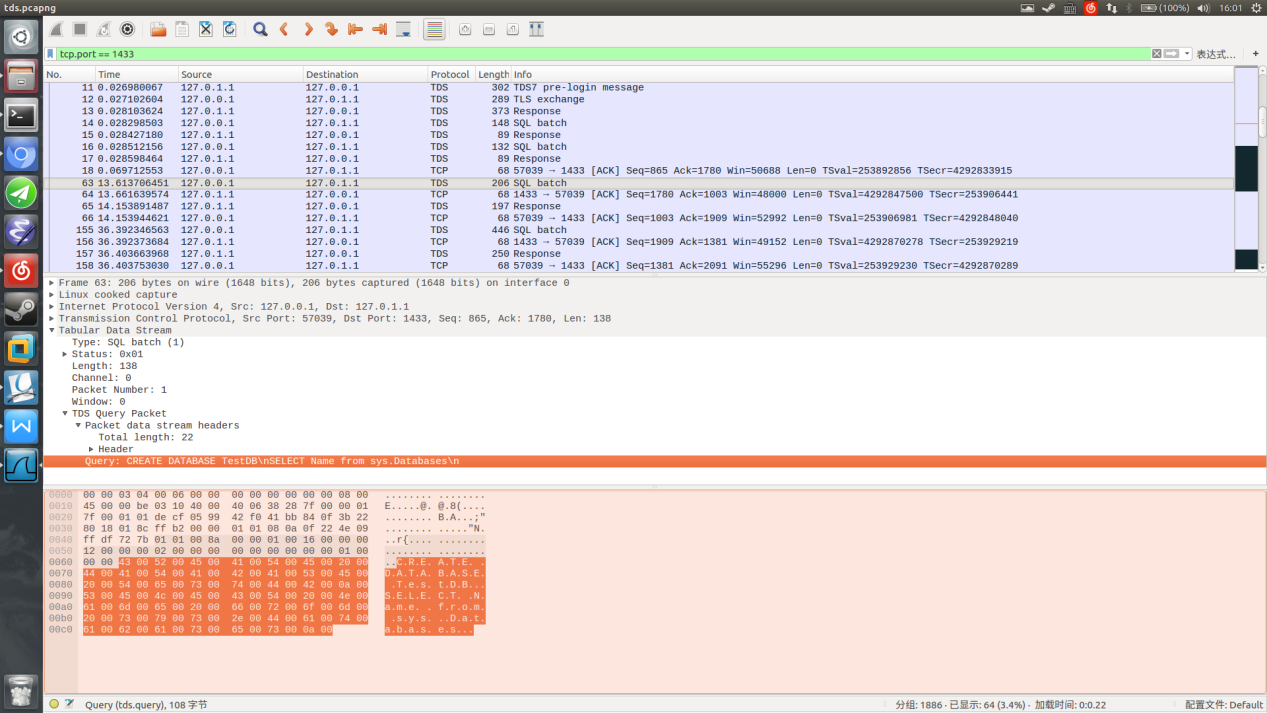


****

当headerType = trace activity时：

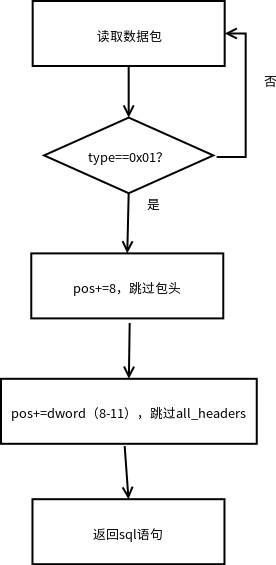


而后面的sqltext就是我们所要提取的sqltext。多条语句之间用/n分隔。



灰色部分是sqltext之外的部分，红色是我们的sql语句。

那么如何定位呢？我们看到all\_header下面一位就是长度，我们可以想到定位的方法：



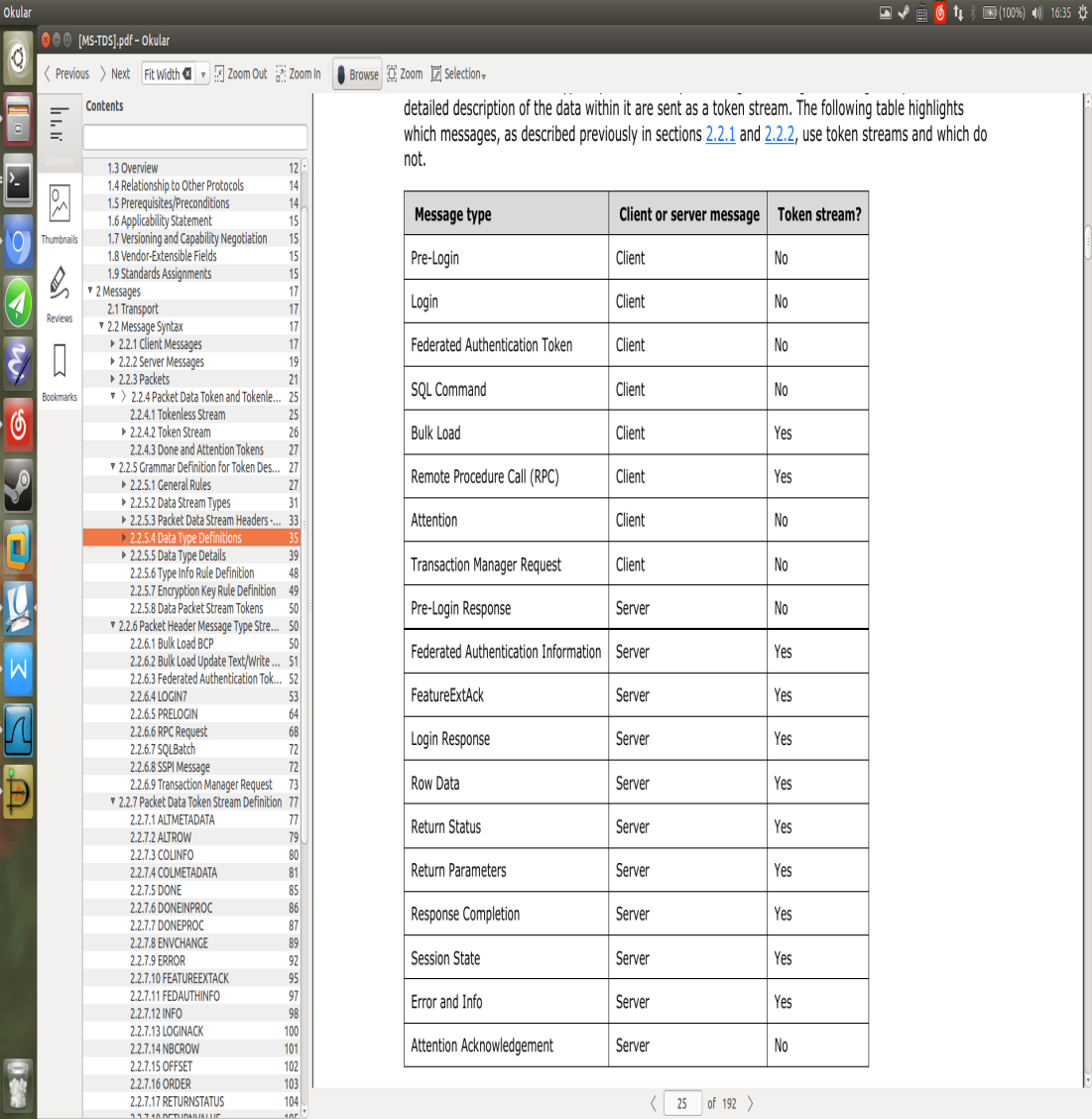
## 从response报文中提取数据

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 | Token 1 | ... | Token n |

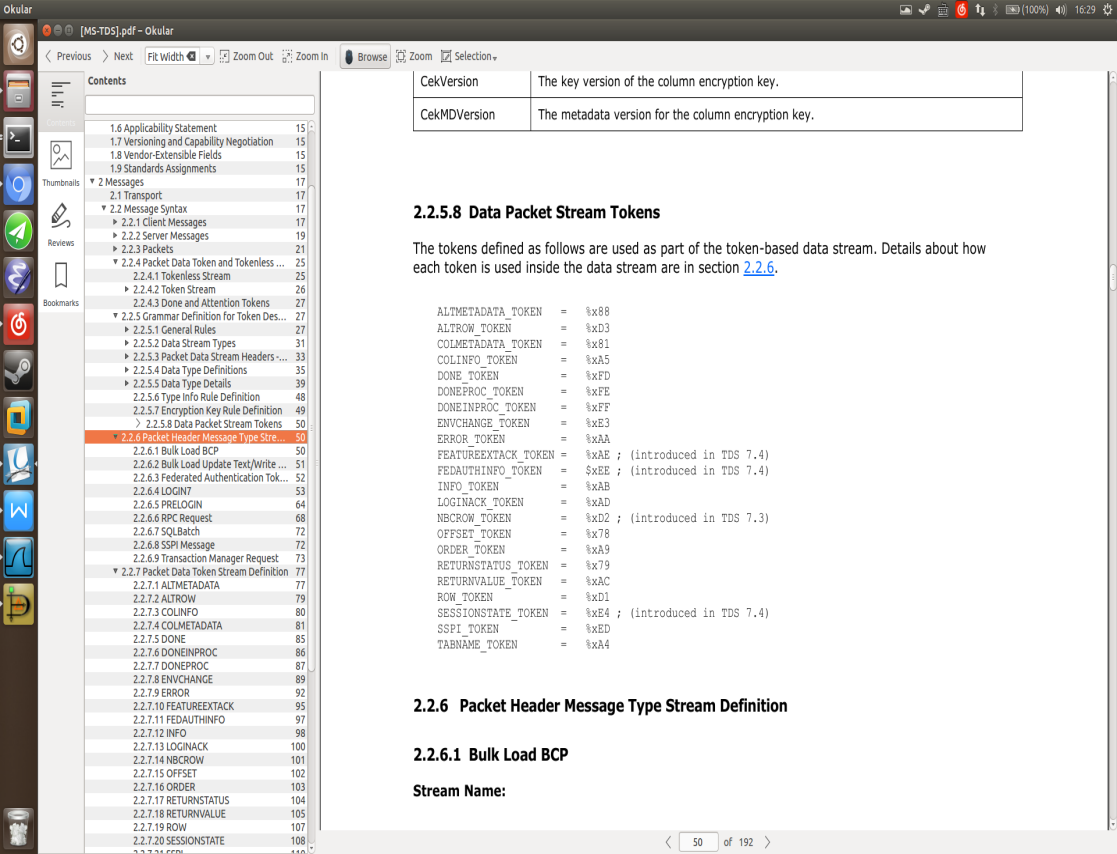
由于回应包是和发送包对应的，所以每一个token都是和相关的sql语句对应的。除了prelogin时返回的数据不是这个形式（因为prelogin阶段实际上在进行ssl握手，所以不符合这个形式），其他返回的都是由token包含的数据。具体来说：



Token格式常见为：

|  |  |
| --- | --- |
| Token type | data |

常见的token type如下：



Sqlserver总共有23种token，但是对于我们来说，常用的只有几种：

### Done token

Done 代表一个sql查询语句的结束，不同的版本done token长度不同但都是定长的。格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| TokenType = 0xfd | State （2） | curCmd（2） | doneRowCount（8） |

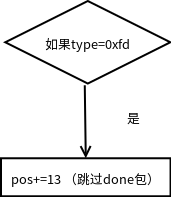
**State：**state表示当前done的状态，有一下几种状态：

|  |  |  |
| --- | --- | --- |
| 0x00 | DONE\_FINAL | 本次报文中最后一个done token |
| 0x1 | DONE\_MORE | 不是最后一个done token，后面还有数据 |
| 0x2 | DONE\_ERROR | 当前语句发生错误，前面必须有error token |
| 0x4 | DONE\_INXACT | 事务正在进行 |
| 0x10 | DONE\_COUNT | RowCount位有效，这是为了区分count为0的情况 |
| 0x20 | DONE\_ATTN | Attention消息的回应包 |
| 0x100 | DONE\_SERVERROR | 语句错误，所有的结果都被丢弃。 |

**CurCmd：**由应用层决定的数据，TDS不进行操作。

**doneRowCount：**多少行被当前语句影响了。

可以看出，done包对我们来说用处不大，分析时候直接跳过即可。



### INFO token

info token也是很重要的token，通过info token我们可以获得服务器名称等数据：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Token type = 0xab | Length（2） | Number（4） | State（2） | Class（2） | Msgtext（us\_varchar） | Server name(b\_varchar) | Proc name (b\_varchar) | Line number (4) |

**Length:**info token总长度。

**Number:** info数。

**State:** 错误状态，用来修改number。

**Class：**错误的严重性，小于10都作为通知信息。

**Msgtext:** 错误信息分为两部分：2字节长的长度+信息

**Server name:** 服务器名称，分为两部分：1字节长的长度+信息

**Proc name :** 运行存储好的程序时使用，记录存储过程的名称。分为两部分：1字节长的长度+信息.

**Line number :** 是第几条语句产生了错误。

可以看出，这个token我们能提取的有效数据是错误信息，过程名称与服务器名称。由于三个数据的位置关系，我们可以进行如此提取：

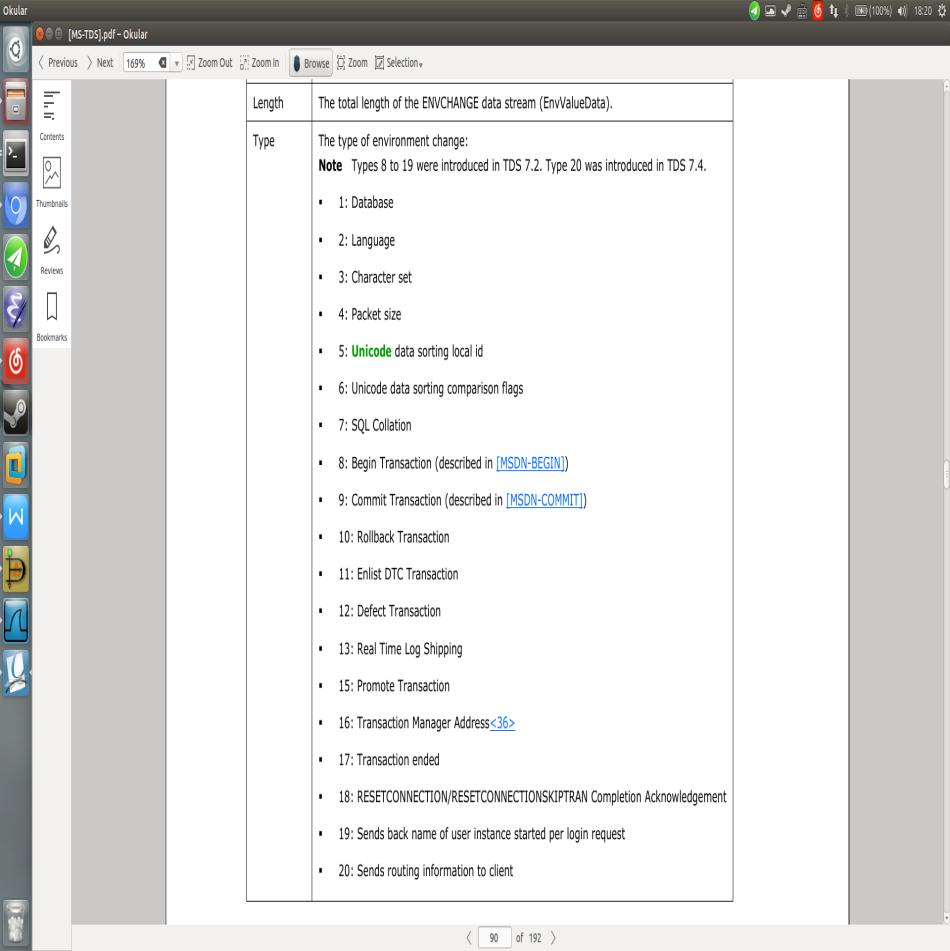


### env change token

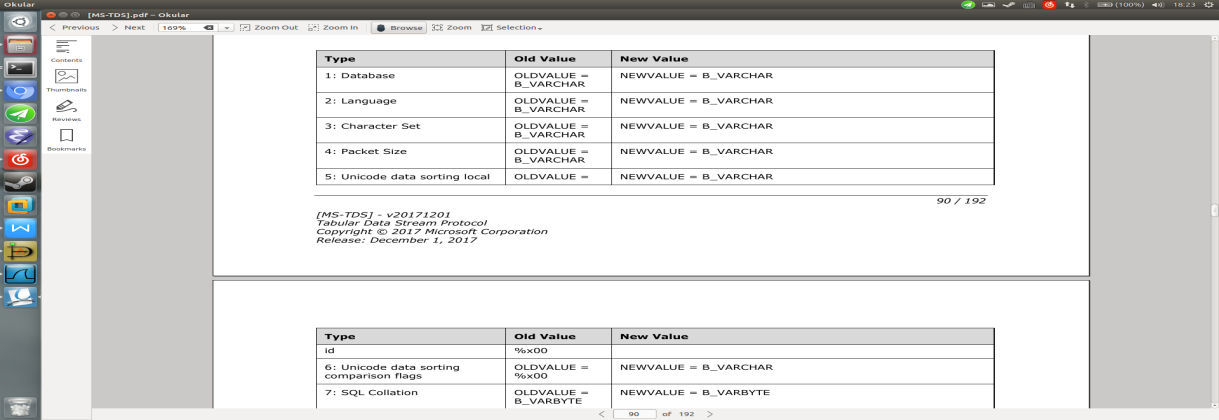
Env change token非常重要，因为他表示了环境的改变，一般都是代表了一些默认数据的改变，比如改变默认数据库等。因此env change token出现需要我们仔细分析。

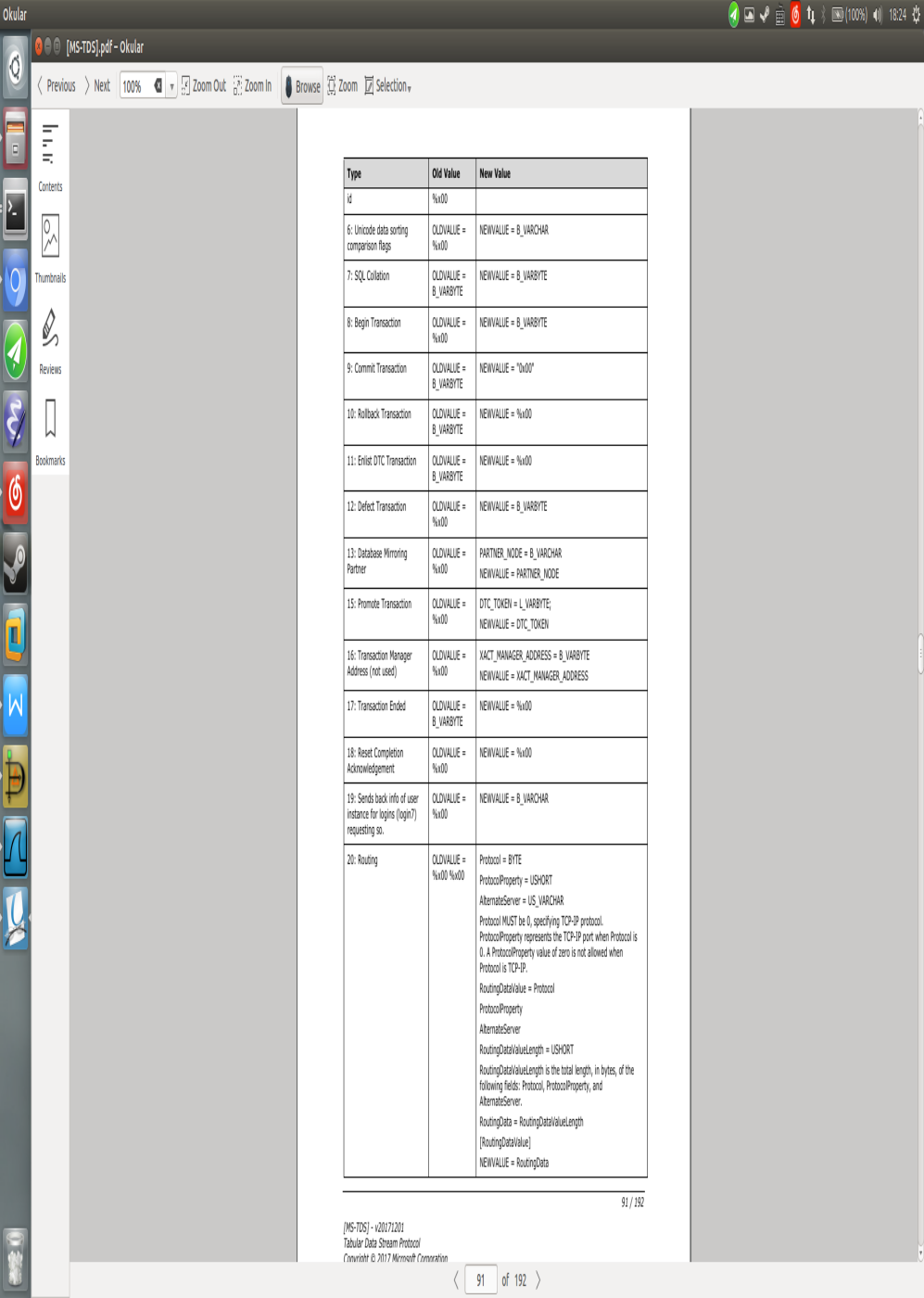
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| tokenType = 0xe3 | Length（2） | Type（1） | New value | [old value] |

其中type，表示当前是什么类型的环境改变



对于每一种type，我们都有如下new value 的规定：





### COLMETADATA token

COLMETADATA token是用来描述接下来一组row token的。他规定了row token的格式。Colmetada 格式非常复杂，我们要分层来说明：

|  |  |  |  |
| --- | --- | --- | --- |
| tokenType = 0x81 | Count（2） | cektable | Nometadata/count×columndata |

Count：记录有多少列数据。

Cektable：用来加密的秘钥，暂时可忽略。没有加密时此项不存在。

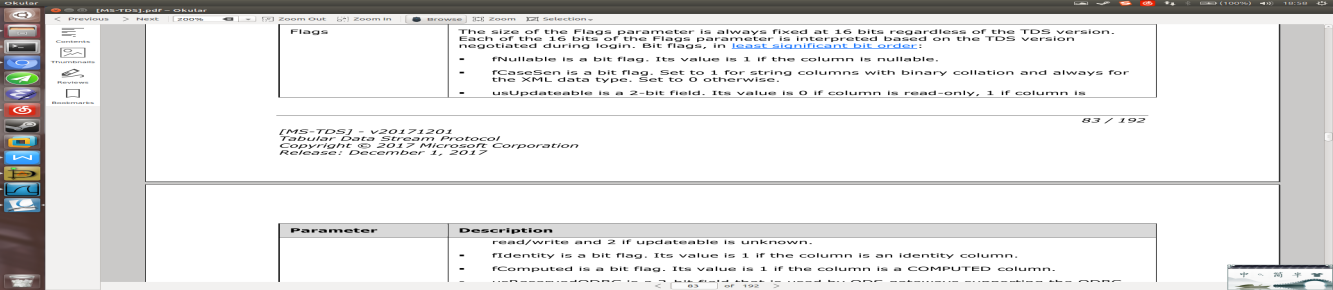
Nometadata：当此项存在，说明metadata已经发送过了，接下来没有metadata了。

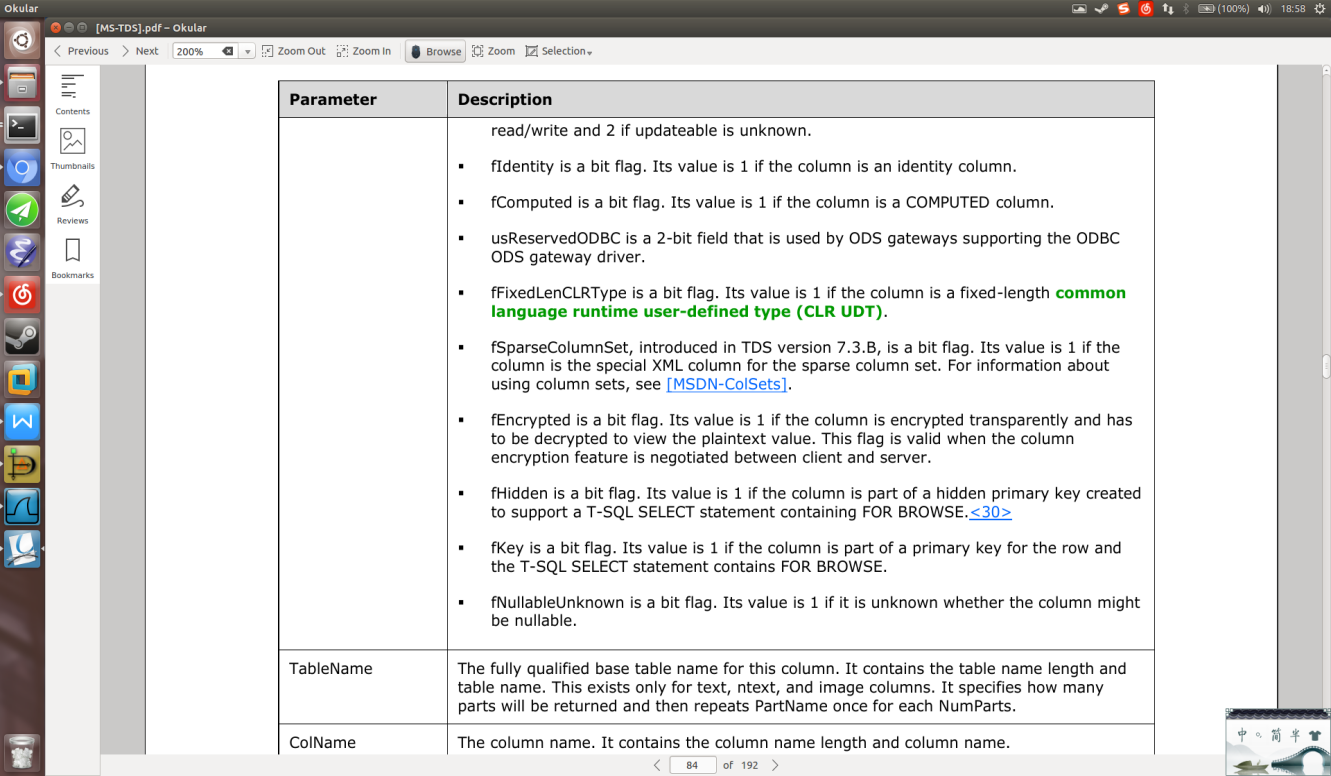
Columnadata：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| userType(4) | Flags(2) | TYPE\_INFO | [TABLE\_NAME] | [CryptoMetaData] | ColName(B\_varchar) |

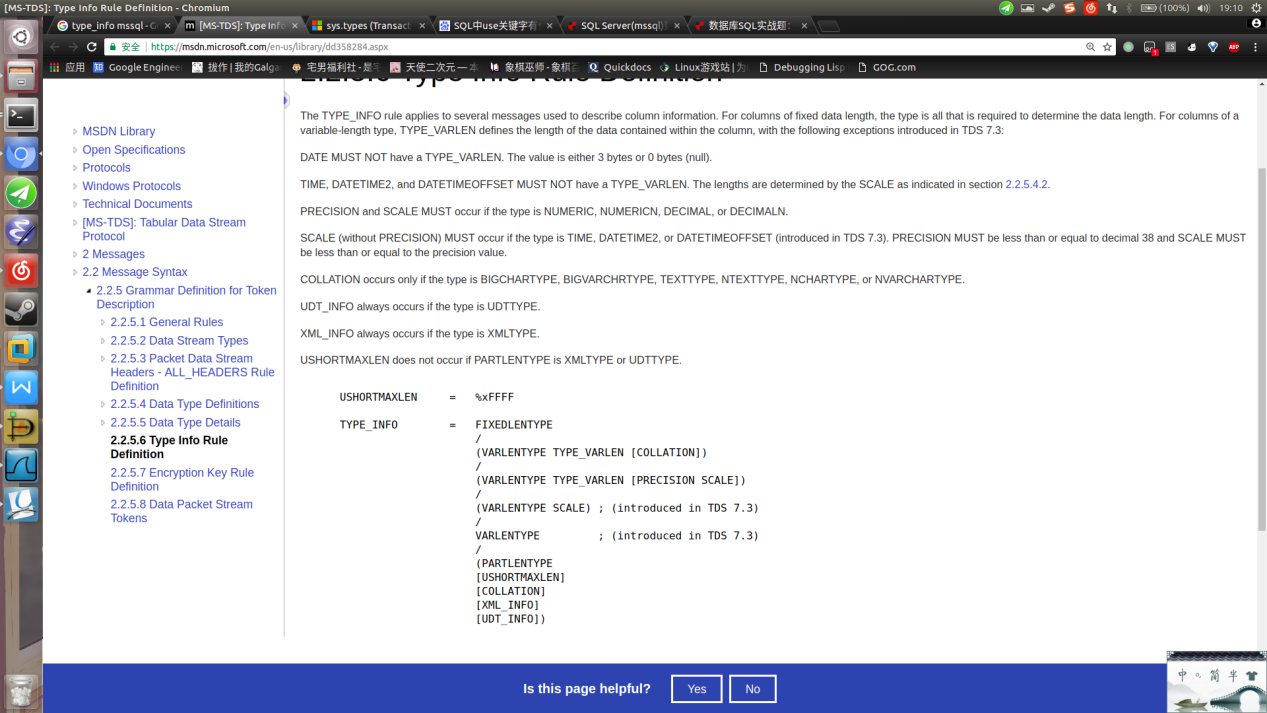
Usertype: 数据类型。和自定义数据类型有关。

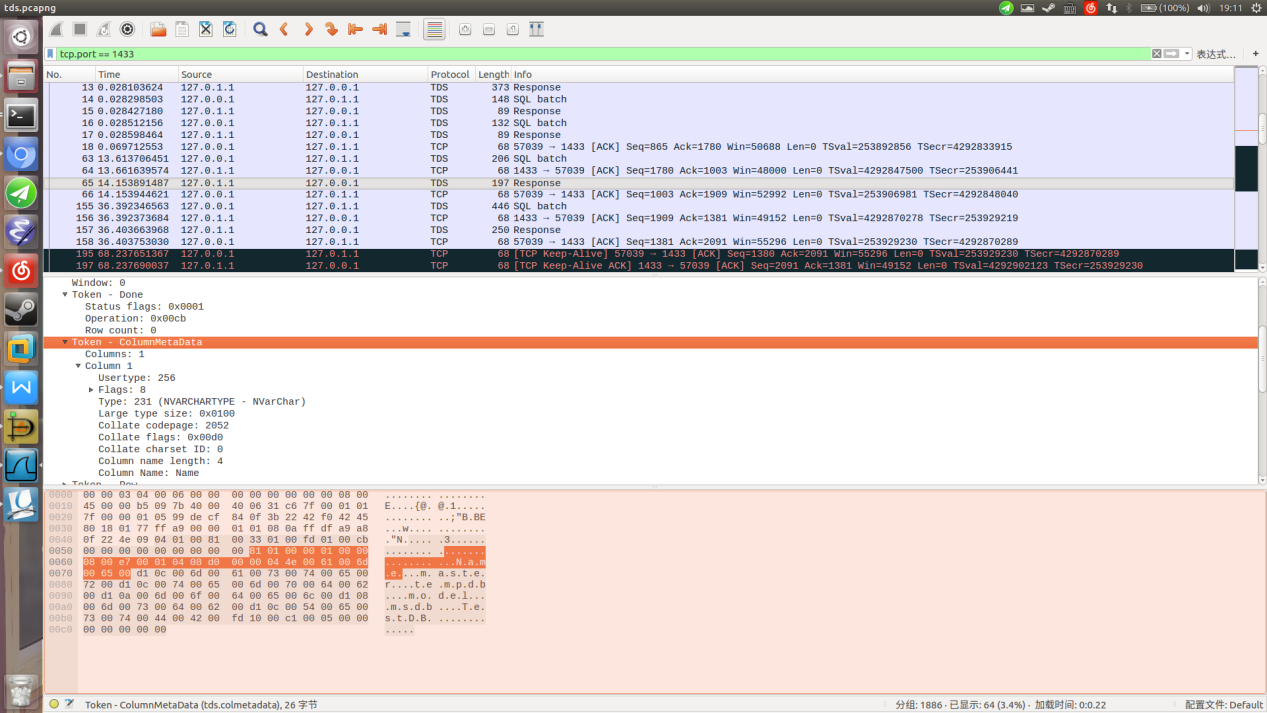
Flags： 标志，具体如下所述：



****

type\_info : 规定了当前类的类型。 此处文档和wireshark不同，暂存疑。





Colname：存储了列的名字。

### Row token

如果说这是最重要的token也不为过。所有返回的数据都在row token 中。由于有colmetadata token的协助，row token结构非常简单。

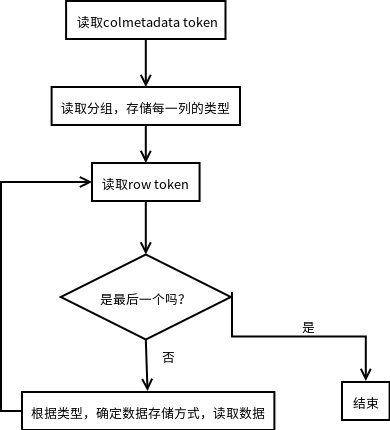
|  |  |  |  |
| --- | --- | --- | --- |
| typeToken = 0xD1 | [Text\_pointer] (b\_varbyte) | [Timestamp](2) | Data(type\_varbyte) |

Text pointer:数据指针的长度和本身

Timestamp：时间戳

Data：数据本身，和数据类型有关。

一般来说，text pointer timestamp 不经常使用。我们使用这token时候，要配合COLMETADATA使用。



## 从bulk insert语句中提取sql语句

Bulk insert/load 语句，是特殊的sql查询语句。他就是我们常说的批处理。对于这种形式，sql server给出了不一样的查询方式：

包头的type数值为0x07。

|  |  |  |  |
| --- | --- | --- | --- |
| Colmetadata token | Row token 1 | ... Row token n | Done token |

可以看出，bulk insert语句， 就是简单的利用上面所说的token机制，把每一个sql语句封装在每一个 row token中。利用上文对token的分析，就可以提取出批处理每一条语句了。但是要注意，bulk过程传输的只是简单的二进制数据，因此我们要对提取到的二进制数据做出更加精细的分析。

## Sql server登录验证

在sql server 中，sys.server\_principals表和sys.sql\_logins

中记录了所有有关sql server登录的信息。虽然没有具体文档，但是我们可以发现只有这个数据库存储了用户名，登录密码以及登陆时所用的证书。因此我们认为验证过程和这两个表有关。

sys.server\_principals表内容如下：

<https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-server-principals-transact-sql?view=sql-server-2017>

sys.sql\_logins内容如下：  
 <https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-sql-logins-transact-sql?view=sql-server-2017>

根据文档，我们不但能找到login信息，还能找到对应证书信息，对于下一步提取有所帮助。

## 从登录报文中提取用户名，密码信息

Sql server有三种加密方式：

1. 明文传输
2. 仅加密登录过程
3. 加密所有过程

默认情况下，sql server会自动采取2方案，就是仅加密登录过程。但是明文是所有提取的基础，我们会先从明文进行分析。

**Login提取：**

Login报文包头为0x10

 LOGIN7           =

Length（4）

                      TDSVersion （4）

                      PacketSize （4）

                      ClientProgVer （4）

                      ClientPID （4）

                      ConnectionID （4）

                      OptionFlags1 （1）

                      OptionFlags2 （1）

                      TypeFlags （1）

                      (FRESERVEDBYTE / OptionFlags3) （1）

                      ClientTimZone （4）

                      ClientLCID （4）

                      OffsetLength

                      Data

                      [FeatureExt]

我们主要关注offsetLength 与data。Offsetlength文档里面有所说明：

The variable portion of this message. A stream of bytes in the order shown, indicates the offset (from the start of the message) and length of various parameters:

ibHostname & cchHostName: The client machine name.

ibUserName & cchUserName: The client user ID.

ibPassword & cchPassword: The password supplied by the client.

ibAppName & cchAppName: The client application name.

ibServerName & cchServerName: The server name.

ibUnused & cbUnused: These parameters were reserved until TDS 7.4.

ibExtension & cbExtension: This points to an extension block. Introduced in TDS 7.4 when fExtension is 1. The content pointed by ibExtension is defined as follows:

ibFeatureExtLong    =  DWORD

Extension           =  ibFeatureExtLong

ibFeatureExtLong provides the offset (from the start of the message) of FeatureExt block. ibFeatureExtLong MUST be 0 if FeatureExt block does not exist.

Extension block can be extended in future. The client MUST NOT send more data than needed. The server SHOULD ignore any appended data that is unknown to the server.

ibCltIntName & cchCltIntName: The interface library name (ODBC or OLEDB).

ibLanguage & cchLanguage: The initial language (overrides the user ID's default language).

ibDatabase & cchDatabase: The initial database (overrides the user ID's default database).

ClientID: The unique client ID (created by using the NIC address). ClientID is the MAC address of the physical network layer. It is used to identify the client that is connecting to the server. This value is mainly informational, and no processing steps on the server side use it.

ibSSPI & cbSSPI: SSPI data.

If cbSSPI < USHRT\_MAX, then this length MUST be used for SSPI and cbSSPILong MUST be ignored.

If cbSSPI == USHRT\_MAX, then cbSSPILong MUST be checked.

If cbSSPILong > 0, then that value MUST be used. If cbSSPILong ==0, then cbSSPI (USHRT\_MAX) MUST be used.

ibAtchDBFile & cchAtchDBFile: The file name for a database that is to be attached during the connection process.

ibChangePassword & cchChangePassword: New password for the specified login. Introduced in TDS 7.2.

cbSSPILong: Used for large SSPI data when cbSSPI==USHRT\_MAX.   
Introduced in TDS 7.2.

所以我们要根据offsetlength，提取出用户名称，然后在data数据里面进行提取数据。

当加密时，如果用的是自己证书（3情况），则要根据自己的证书解密数据。 用的是官方证书（2情况），要根据sys.sql\_logins表里面的证书进行提取。

