# 基于SQL Server原生日志读取接口实现数据库事务日志解析工具详细设计

# 1 简介

本软件名为“SQL Reader”，是基于SQL Server原生日志读取接口实现的数据库事务日志解析工具。

开发者为来自电子科技大学的队伍“Sever三巨头”。

开发环境：SQL Server 2019

开发语言：Java

# 2 核心问题分析和解决方案

## 2.1 日志序列号(LSN)的递增规则

LSN通常分为三段（xxxxxxxx：xxxxxxxx：xxxx），全部为十六进制。第一段递增幅度为1；第二段最小为00000010，递增幅度为8的倍数；第三段最小为0001，递增幅度为1。通常同一个操作的事务日志具有相同的第二段LSN，第三段LSN从0001开始，每行日志加1。对此，我们设置了若干个方法，分别对各段LSN进行加减、置为初值的操作。

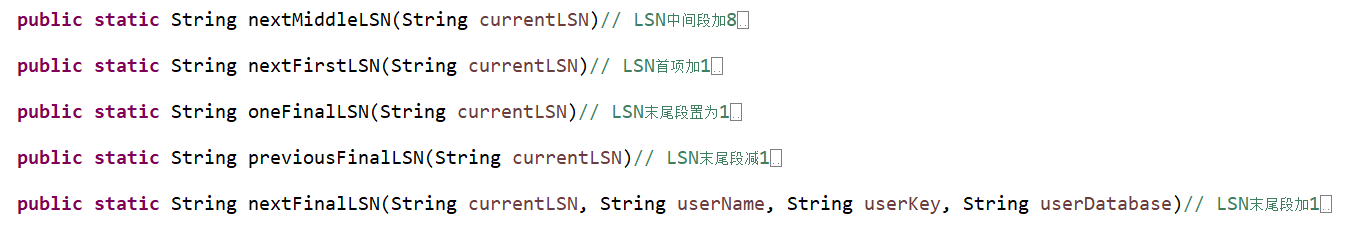


图 2-1

此外需要注意到，对于不同的数据库，第二段LSN具有不同的最大值（随该数据库最大存储空间而变），当达到最大值，第一段日志加1，第二段日志置为00000010，第三段日志置为0001，我们称之为“进位”。我们发现“进位”前的临界日志有如下特点：第二段递增任意值，第三段不变，对应的日志是存在的（但在查询出的日志不显示）；第二段不变，第三段递增，日志不存在。

利用此特点，我们解决了以下两种“进位”发生时，下一条日志的定位：1、当前事务已结束，“进位”发生在下一事务（应用场景：事务间跳转）：若LSN第二段加1后，其日志中的Current LSN和Previous LSN前两段不一致，说明发生了“进位”，于是我们返回“进位”后的LSN作为下一条日志的起点。2、当前事务未结束，“进位”发生在事务日志的中间（应用场景：事务内部的下一条日志的定位）：若LSN第三段加1后，查询日志报错，则说明发生了“进位”，于是我们返回“进位”后的LSN作为下一条日志的起点。

如图，事务间的“进位”和发生在事务内部的“进位”

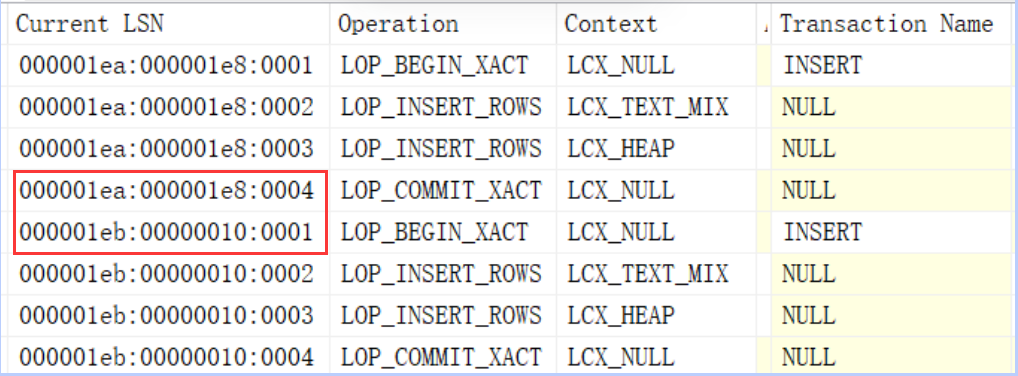


图 2-2

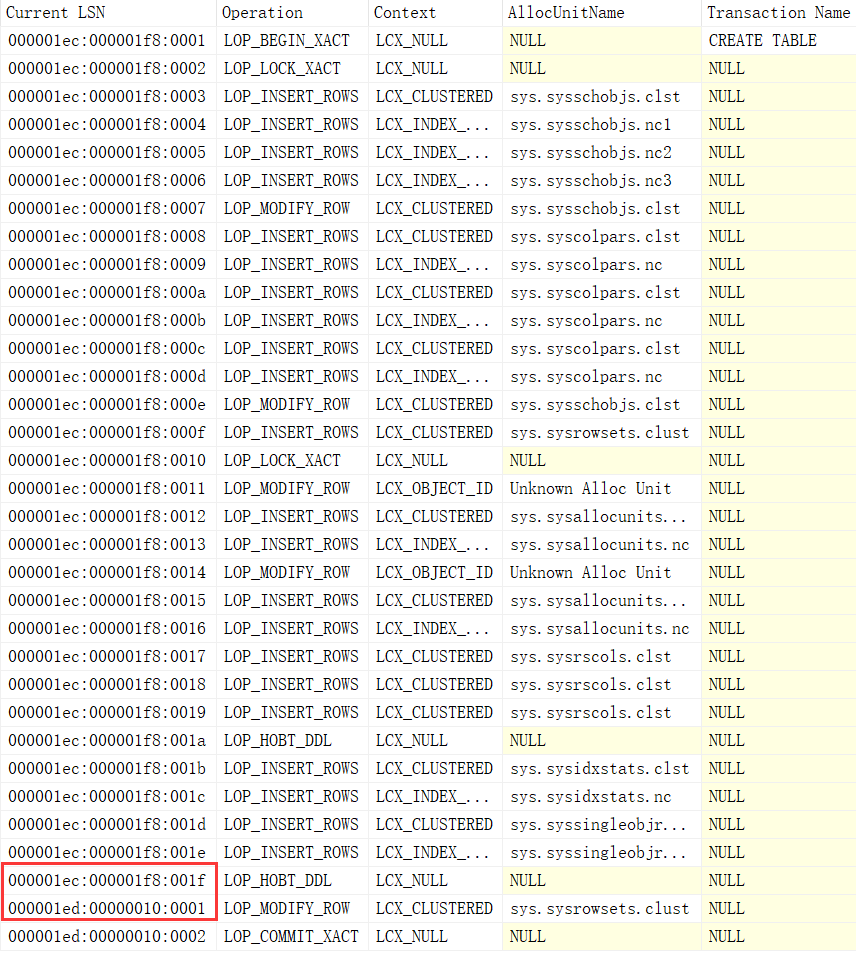


图 2-3

如图，用于定位下一事务的方法findNext，输入当前LSN，返回下一事务的起始LSN。 IMG_259

图 2-4

## 2.2 各类型DDL/DML操作的日志解析（包含大对象数据类型的处理）

操作类型的筛选：先定位事务日志的第一个BEGIN（Operation为“LOP\_BEGIN\_XACT”）所在行，再通过Transaton Name的内容判断操作类型。

### 2.2.1 create table操作

首先跳过事务内部包含的事务（若干个BEGIN-COMMIT块），定位第一条插入日志（Operation为"LOP\_INSERT\_ROWS"，Context为"LCX\_CLUSTERED"）所在行，获取该行的"RowLog Contents 0"字段（包含表名信息）。表名存储在"RowLog Contents 0"的第56个字节之后，为unicode编码的字符串，解析过后可得表名。

根据表名通过查询系统元数据库（information\_schema）获得该表的数据类型，数据长度，主键，列名，列数量。按标准SQL语句逐列拼装并打印结果。

### 2.2.2 insert操作

首先定位包含插入大对象类型text/ntext/image的日志（Operation为"LOP\_INSERT\_ROWS"，Context为"LCX\_TEXT\_MIX"），由于插入大对象的操作单独占用若干日志，且顺序在插入全部数据的日志之前，故首先获取这部分日志的"RowLog Contents 0"字段（包含大对象数据）并保存在专门分配的数组中，以便之后与其他类型的数据一并传参和解析。另外，大对象的插入分为短（0-64字节）、中（65-8040字节）、长（8040字节以上）三种，每种插入对应不同的日志结构。

短文本：日志结构如图，文本数据存储在Operation为"LOP\_INSERT\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 0"字段中，而存储全部数据的行日志（图中第二行）的"RowLog Contents 0"字段只保存该大对象数据的16字节指针。

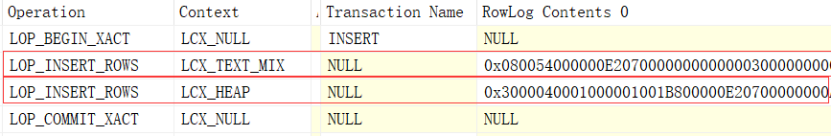


图 2-5

中文本：日志结构如图，文本数据存储在Operation为"LOP\_MODIFY\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 1"字段中，而存储全部数据的行日志（图中最后一行）的"RowLog Contents 0"字段只保存该大对象数据的16字节指针。

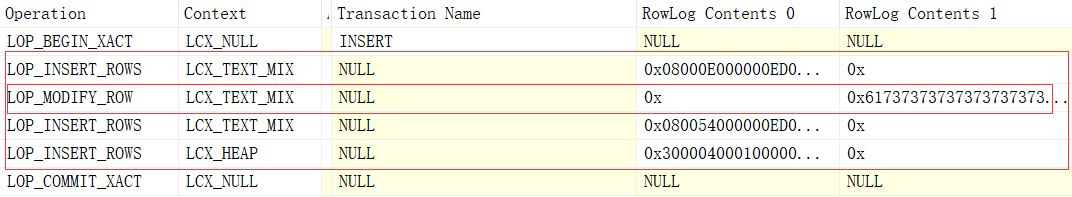


图 2-6

长文本：日志结构如图，文本数据存储在若干个Operation为"LOP\_MODIFY\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 1"字段中，，解析时需将多个16进制字符串拼接。而存储全部数据的行日志（图中最后一行）的"RowLog Contents 0"字段只保存该大对象数据的16字节指针。

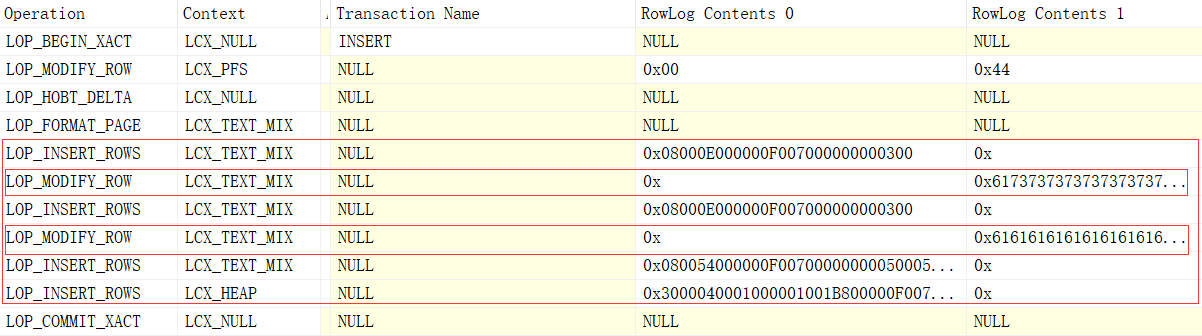


图 2-7

完成大对象的解析后，定位插入日志（Operation为"LOP\_INSERT\_ROWS"）所在行，获取该行的"AllocUnitName"（包含表名）和"RowLog Contents 0"字段（全部列的插入值）。根据表名通过查询系统元数据库（information\_schema）获得该表的数据类型，数据长度，主键，列数量等关键信息，并与"RowLog Contents 0"包含的16进制字符串一起传递给数据解析模块，返回解析后的各列插入值，根据表名，列数量以及解析后的插入值，按标准SQL语句逐列拼装并打印结果。

### 2.2.3 delete操作

首先定位第一条删除日志（Operation为"LOP\_DELETE\_ROWS"，Context为"LCX\_MARK\_AS\_GHOST"）所在行，获取该行的"AllocUnitName"（包含表名）和"RowLog Contents 0"字段（包含主键数据）。根据表名通过查询系统元数据库（information\_schema）获得该表的数据类型，数据长度，主键，列数量等关键信息，并与主键数据一起传递给数据解析模块，返回解析后的主键值。

根据表名，主键以及解析后的主键值，按标准SQL语句逐列拼装并打印结果。

text/ntext等大对象数据类型无法作为主键，不需考虑解析。

### 2.2.4 update操作

首先定位包含更新大对象类型text/ntext/image的日志（Operation为"LOP\_MODIFY\_ROWS"或"LOP\_INSERT\_ROWS"，Context为"LCX\_TEXT\_MIX"），由于更新大对象的操作单独占用若干日志，且顺序在插入全部数据的日志之前，故首先处理这部分日志。大对象分为短（0-64字节）、长（64字节以上）两种，不同的更新情况共有6种，每种更新对应不同的日志结构。

null更新为短文本：日志结构如图，文本数据存储在Operation为"LOP\_INSERT\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 0"字段中，且该行的"RowLog Contents 0"长度不少于20字节，而记录所有更新的行日志（图中最后一行）的"RowLog Contents 1"字段保存修改后新增的16字节文本指针。

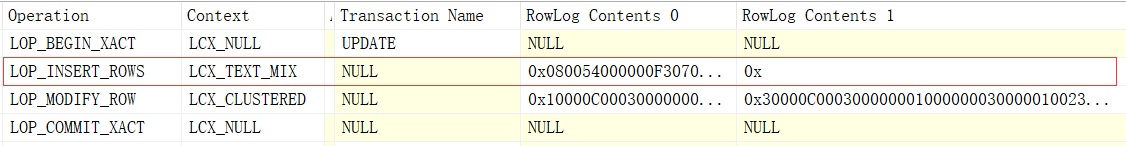


图 2-8

null更新为长文本：日志的结构如图，Operation为"LOP\_INSERT\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 0"字段长度不超过14字节，文本数据存储在Operation为"LOP\_MODIFY\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 1"字段中，且长度大于64字节，而记录所有更新的行日志（图中最后一行）的"RowLog Contents 1"字段保存修改后新增的16字节文本指针。

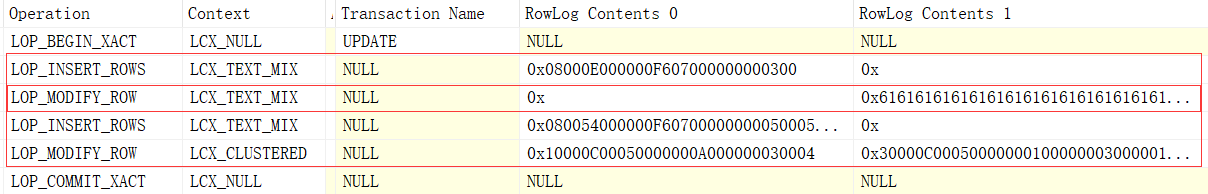


图 2-9

短文本更新为短文本： 短文本更新为null的情况与此情况一致。日志结构如图，文本数据存储在Operation为"LOP\_MODIFY\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 0"字段中，且"RowLog Contents 1"不是空白字符串。最后一行Operation为"LOP\_MODIFY\_ROWS"且Context不为"LCX\_TEXT\_MIX"的日志记录全部数据的更新情况。

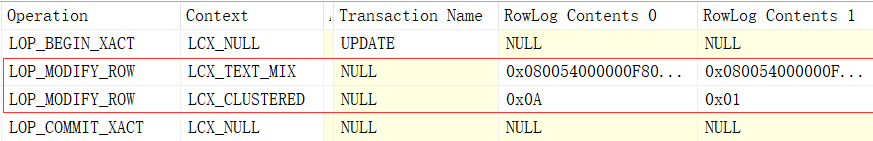


图 2-10

短文本更新为长文本：日志结构如图，Operation为"LOP\_INSERT\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 1"字段长度不超过14字节，下一行Operation为"LOP\_MODIFY\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 1"字段存储更新前的文本数据，且长度不超过64字节。最后一行Operation为"LOP\_MODIFY\_ROWS"且Context不为"LCX\_TEXT\_MIX"的日志记录全部数据的更新情况。

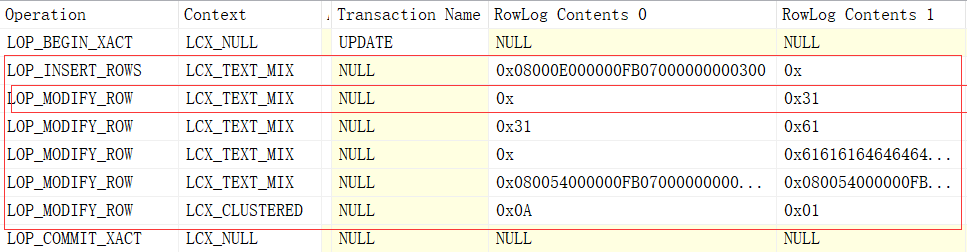


图 2-11

长文本更新为长文本：日志结构如图，Operation为"LOP\_MODIFY\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 1"字段为空白字符串，下一行Operation为"LOP\_MODIFY\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 0"字段存储更新前的文本数据。最后一行Operation为"LOP\_MODIFY\_ROWS"且Context不为"LCX\_TEXT\_MIX"的日志记录全部数据的更新情况。

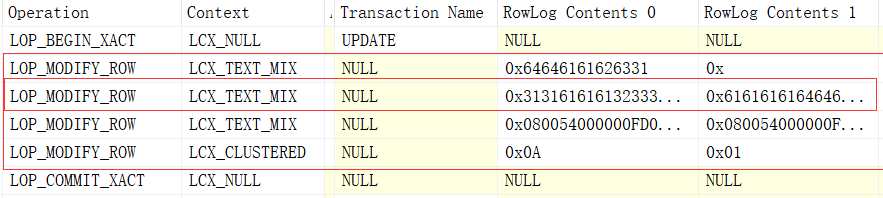


图 2-12

长文本更新为null：日志结构如图，Operation为"LOP\_DELETE\_ROWS"，Context为"LCX\_TEXT\_MIX"的行日志的"RowLog Contents 0"字段存储更新前的文本数据。最后一行Operation为"LOP\_MODIFY\_ROWS"且Context不为"LCX\_TEXT\_MIX"的日志记录全部数据的更新情况。

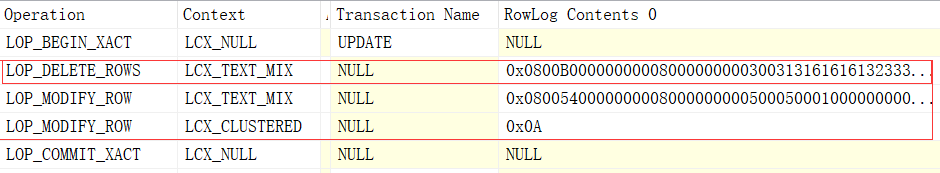


图 2-13

完成大对象的解析后，定位更新日志（Operation为"LOP\_MODIFY\_ROWS"或"LOP\_MODIFY\_COLUMNS"）所在行，获取该行的"AllocUnitName"（包含表名）、“Offset”（16进制日志记录发生修改的起始位置）和"RowLog Contents 0"（包含修改前数据）、"RowLog Contents 1"（包含修改后数据）、"RowLog Contents 2"（包含主键数据）字段。根据表名通过查询系统元数据库（information\_schema）获得该表的数据类型，数据长度，主键，列数量等关键信息。

update的日志只保存变化的部分，所以Rowlog Content中的内容会被截断，以至于不足以解析出被修改的数据。我们采用还原16进制字符串的方法，根据表名和主键查询修改后的数据，传递给数据解析模块进行拼装，生成完整的16进制Rowlog Content（类似INSERT操作中的16进制字符串），并根据"RowLog Contents 0"和"RowLog Contents 1"以及“Offset”进行字符串替换，生成存储修改前数据的16进制字符串，传递给数据解析模块进行解析，返回修改前的所有数据。

比较修改前、修改后的全部数据，只保留不同的数据（进行了update）。根据表名，主键，列名以及解析后的值，按标准SQL语句逐列拼装并打印结果。

## 2.3 各类型数据的解析

整型

Tinyint 定长 （ 0 到 255 ）1 byte

Smallint 定长（-32,768 到 32,767）2 byte，小端，10进制转换成2进制后补0，正数直接从2进制转换为16进制再转换字节序，负数则需将2进制字符串按位变反再加1后转换为16进制，接着再转换字节序

int 定长（ -2,147,483,648 到 2,147,483,647）4 byte，小端，10进制转换成2进制后补0，正数直接从2进制转换为16进制再转换字节序，负数则需将2进制字符串按位变反再加1后转换为16进制，接着再转换字节序

Bigint 定长（-9,223,372,036,854,775,808 与 9,223,372,036,854,775,807）8 byte，小端，10进制转换成2进制后补0，正数直接从2进制转换为16进制再转换字节序，负数则需将2进制字符串按位变反再加1后转换为16进制，接着再转换字节序

字符串型

Char(n) 定长 n byte，大端，一个字节（4个16进制数）存一个字符，英文字符等为1个字符存1个字节，汉字为1个字符存两个字节，在数据后面补'20'（ASCII码对应字符'0'）至 n byte

Varchar(n) 变长 大端，一个字节存一个字符，长度为内容长度

Varchar(max) 变长 大端，一个字节存一个字符，长度为内容长度

text 需要另起一行日志，前一行从第20字节后开始存数据，一个字节存一个字符，后一行在变长列部分存指向存储位置的16位指针

字符型

nchar（n） 定长 2n byte，大端，两个字节（4个16进制数）存一个字符，补'2000'（字符'0'）至 2n byte

nvarchar（n） 变长 2n byte，大端，两个字节存一个字符，长度为内容长度

nvarchar（max） 变长 2n byte，大端，两个字节存一个字符，长度为内容长度

ntext 需要另起一行日志，前一行从第20字节后开始存数据，两个字节存一个字符

时间型

smalldatetime 定长 4 byte，小端，精确到分钟，基准为1900-01-01 00:00，前两个字节存放总分钟数，分解为时分；后两个字节的值是总天数，从 0000 开始递增，分解为年月日

timestamp 定长 8 byte，默认输入default，默认列名TIMESTAMP

datetime 定长 8 byte，小端，精度为 3.33 毫秒，基准为 1753-01-01 0:0:0.000，前四个字节存放总秒数，转换为16进制后转换字节序，分解为时分秒微秒，后四个字节存放总天数，从 462EFFFF 开始递增，转换字节序，分解为年月日

datetime2 定长 8 byte，小端，精度为 100纳秒，基准为 1753-01-01 0:0:0.0000000，前五个字节分解为时分秒以及小数点右边的值（1秒为十进制10000000），后三个字节存放总天数，从 A1C309 开始递增，分解为年月日

date 定长 3 byte，小端，仅存储日期，基准为 0001-01-01 ，存放当前日期较基准时间的总天数，该10进制数会转换为16进制数，并按小端存储，分解为年月日

time 定长 5 btye，小端，仅存储时间，基准为00:00:00，存放总秒数且为989680（十进制10000000）的倍数，分解为时分秒

datetimeoffset 定长 10 btye，格式为yyyy-MM-dd HH:mm:ss

uniqueidentifier 定长，16byte，数据插入形式是：‘XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX’的分成5段的16进制字符串。在RowLog Contents 0中的存储是分段存储，前3段按小端存储，后2段按大端存储，段与段之间的存储顺序不变

浮点型

numeric(p,s) 定长，5~17byte，1<=p<=38,1<=s<=p,默认情况下p=18，p代表数据总位数（按10进制算的位数），s代表小数的位数。

decimal(p,s) 定长，5~7byte，1<=p<=38,1<=s<=p,默认情况下p=18，p代表数据总位数（按10进制算的位数），s代表小数的位数。

smallmoney 定长，4byte，数据在小数点后第5位四舍五入，注意smallmoney型数据在Rowlog Contents 0字段中是把数据看作10进制字符串（即去掉小数点）然后转换为2进制字符串，先在字符串前补0然后，正数字节转为16进制字符串后转换字节序，负数则按位变反再加1后转位16进制字符串后转换字节序，采用小端存储（比如插入1.5807，已知15807=3DBF，显示为BF3D0000）

money 定长，8byte，数据在小数点后第5位四舍五入，与smallmoney型数据相同的存储方式

float(n)，定长，8byte，输入数据不超过38位，从 -1.79E + 308 到 1.79E + 308 的浮动精度数字数据

real，定长，4byte，从 -3.40E + 38 到 3.40E + 38 的浮动精度数字数据

二进制型

bit 定长 1字节，不能定义为null值，输入值为0或者1（1字符对应1字节）

binary（n） 定长，n byte ，1<=n<=8000,输入数据前加0x，可以直接用16进制数值赋值（比如0xabcd12），也可以用10进制数赋值（int可以隐式转换为binary），但如果要用字符串类型赋值需要用convert函数强制转换。按大端的方式存储数据，n值指示字节数，并会在数据后补0至相应长度

varbinary（n），变长，1<=n<=8000，n byte，存储的结束位置由那两个字节显示，大端方式存储

varbinary（max），变长，和varbinary（n）相同的存储方式

image，需要另起一行日志，前一行从第20字节后开始存数据，一个字节存两个字符，后一行在变长列部分存指向存储位置的16位指针

## 2.4 checkpoint问题的解决

数据修改操作都是在内存中的数据页进行修改，每次修改后并没有立即把这些页面写入磁盘，而是等到一定时期，数据库引擎对数据库发起检查点命令。

条件：

1.在数据库备份之前，数据库引擎会自动执行检查点

2.日志的 70% 已满

这时该命令就会创建一个检查点，把当前所有在内存中已修改的页面（脏页）即事务日志信息从内存中写入到磁盘，并且记录下有关事务日志的信息。由于我们不使用备份功能，所以通过日志空间的使用情况手动执行checkpoint，从而避免系统执行checkpoint对日志的影响。

# 3 软件总体结构

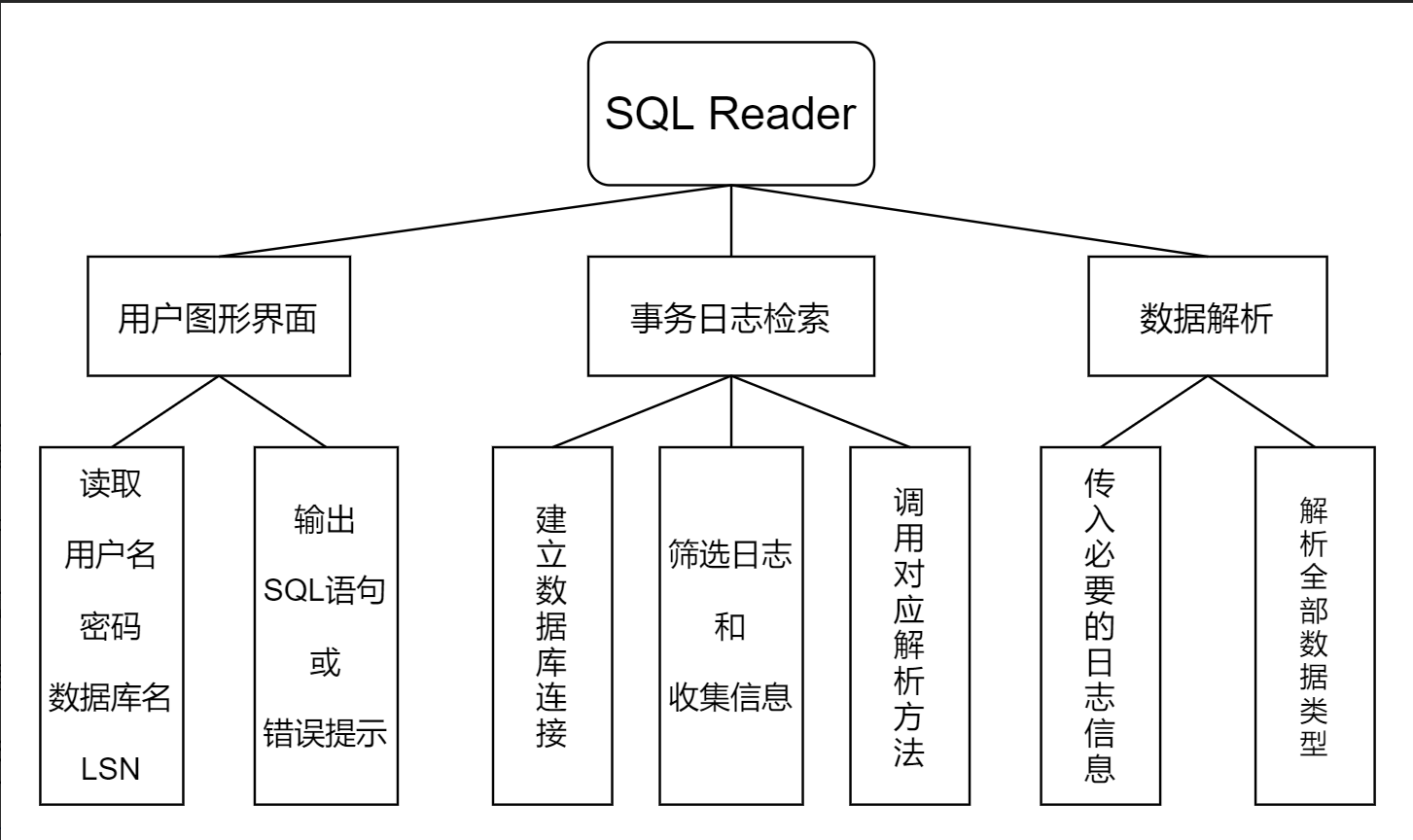


图3.1 软件总体结构

# 4 系统功能模块详细设计

## 4.1 用户图形界面模块详细设计

1. TestGUI类：继承JFrame，创建窗体和输入输出文本框。OutputSQL类：调用事务日志检索模块的方法，解析并打印标准SQL语句，并对错误信息进行弹窗提醒。TestGUI和OutputSQL的方法相互调用从而实现多线程实时打印。
2. GUIPrintStream类：继承PrintStream，重写write()方法实现输出重定向，将输出到控制台的信息重定向到用户图形界面
3. SimpleListener类：实现ActionListener接口，利用actionPerformed(ActionEvent e)方法监听登录按钮，捕捉点击按钮的动作，执行事务日志检索模块。
4. 排版上，使用BorderLayout和GridLayout，界面简洁清晰，用户易懂
5. updataGuiDemo()方法：将光标保持在最新输出的位置

## 4.2 事务日志检索模块详细设计

### 4.2.1 JDBC连接

利用静态代码块效率高的特点加载数据库驱动，根据用户输入的用户名、密码、数据库名获得和数据库的连接。

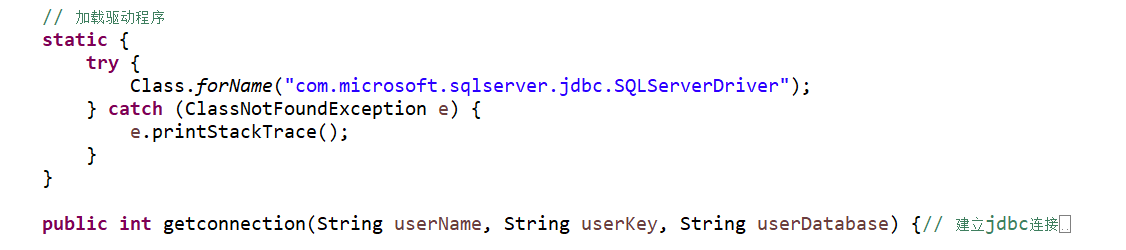


图 4-1

### 4.2.2 下一事务日志起点的定位

每当一条DML/DDL语句解析结束后，调用定位下一事务的方法获取下一事务日志的起始位置。由于事务间日志序列号的间隔存在浮动，故该方法应该对当前LSN进行若干次递增。若全部LSN对应的日志均不存在，说明该LSN处于所有日志的末尾。此时重置LSN为输入的值，重新开始一轮尝试，若数据库端有新增的操作，则能够即时捕捉其日志，从而实现实时解析。

如图，输入当前日志序列号、用户名、密码、数据库名称，返回下一事务日志的起点。

IMG_270

图 4-2

### 4.2.3 checkpoint的处理

确认当前所有日志已经处理完毕后，LSN到达日志的末尾。此时通过控制台命令“DBCC SQLPERF (LOGSPACE)”查询日志空间的使用情况。若该数据库日志已用空间达到60%（系统checkpoint触发值为70%），则手动执行checkpoint语句，将日志空间使用率重置至20%~30%，并打印此次操作，以通知用户后台已经主动执行了checkpoint语句。

此外，还需要跳过checkpoint在日志中留下的记录，为下一事务的解析做准备。我们将checkpoint的处理置于定位下一事务的方法findNext中。

如图为checkpoint日志：

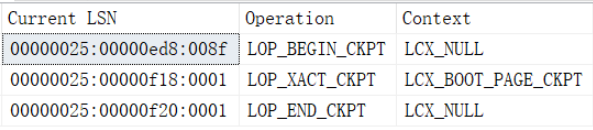


图 4-3

### 4.2.4 日志解析

接收参数：用户名、密码、数据库名，以及日志序列号LSN作为解析的起点（之后自动持续运行）。判断LSN是否有效，无效则抛出异常。

以该LSN为起点对日志开始递增查询，若未定位到BEGIN（Operation为“LOP\_BEGIN\_XACT”），则说明该事务日志不完整或该事务日志不需要解析（例如数据库维护的日志），转向下一事务。

定位到BEGIN后，通过该行的Transation Name的内容判断操作类型。本软件目前只处理create table/insert/update/delete操作。以事务为单位循环遍历日志，根据日志结构的特点对各操作进行解析，并打印标准SQL语句。若数据库端有新增的SQL语句，则通过事务定位算法进行增量事务的解析，从而实现实时性。

如图，日志解析核心方法：

IMG_272

图 4-4

### 4.2.5 工具类

对日志序列号LSN进行操作：

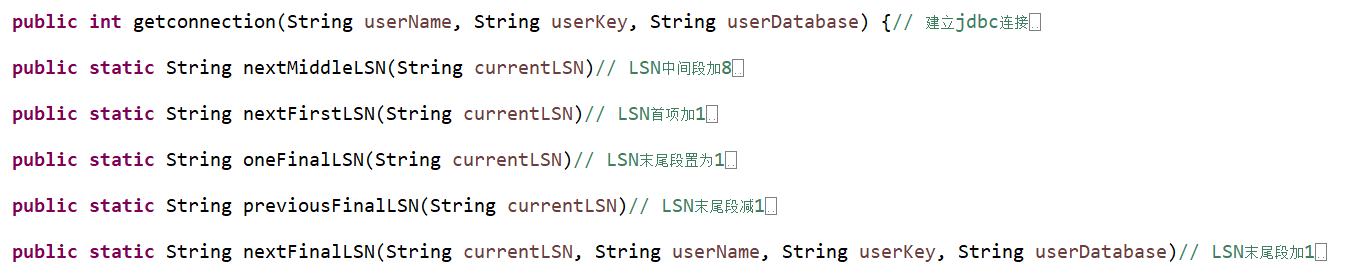


图 4-5

其他操作：

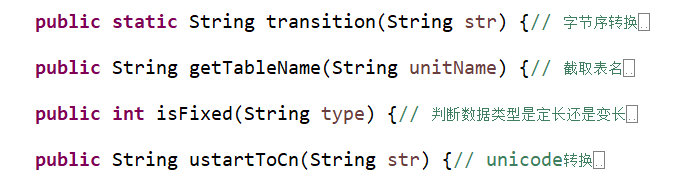


图 4-6

## 4.3 数据解析模块详细设计//可加入工具类的描述

1.首先是分为定长数据类型的处理和变长数据类型的处理

Analyse类的anaFixedType方法：

传入的参数有：定长类型字符串数组（String[] fixedType），定长数据类型的数量(int fixedNum)，定长数据长度数组（int[] fixedLength），字符串rowlog（String rowlog）,指示是否为null的int型数组（int[] isFixedNull）。

首先处理字符串rowlog：切割前4个字节，即切割掉状态位A（1b字节），状态位B（1字节），定长数据大小（2字节）。

然后通过定长数据类型的数量设置循环（循环次数为fixedNum，即定长数据的数量），在每次循环中通过定长数据类型的名称进行分别调用相应的数据解析方法，而每次循环中相应的数据解析方法的返回值类型为字符串数组（String[ ]），需要存放并返回：解析出的数据和切割剩余的rowlog字符串。

解析出的数据覆盖到定长类型字符串数组中，切割剩余的rowlog字符串则传递到下一个循环。循环结束后，切割剩余的rowlog字符串会被放到定长类型字符串数组的最后一个元素中，将作为解析变长数据类型的依据。

Analyse类的anaVaribaleType方法

传入的参数有变长类型字符串数组（String[] variableType），变长数据类型的数量（int variableNum），数据类型的总数量（int totalNum），rowlog字符串（String rowlog），定长数据解析切割剩下的rowlog字符串（String row\_log），指示是否为null的int型数组（int[] isVariableNull）。

首先是计算位图所占字节数。

然后切割掉row\_log字符串的位图字段，变长数据列数量字段以及总的列数量字段，得到以偏移阵列字段开头的row\_log。

然后通过变长数据类型的数量设置为循环（循环次数为variableNum），通过数据类型名称调用相应的处理方法。

每次调用处理方法是，返回类型设置为字符串数组，需要存放并返回：下一个数据的起始位置，解析出的数据。注意在每个类型的解析方法中必须优先考虑数据为null 的情况。

2.一些工具的描述

首先是字节序的转换，因为SQL server在rowlog中的16进制字符串一般会采用小端的存储方式，我们处理时需要转换字节序为大端（即顺序）

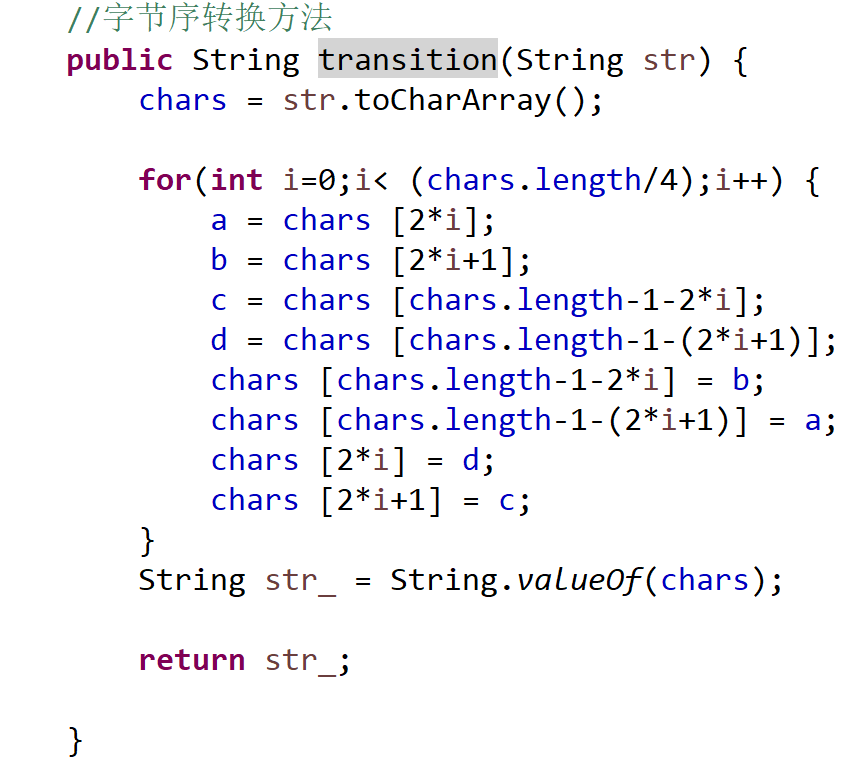


图 4-7

2进制字符串转16进制字符串的方法，在浮点型数据和int型数据都会用到

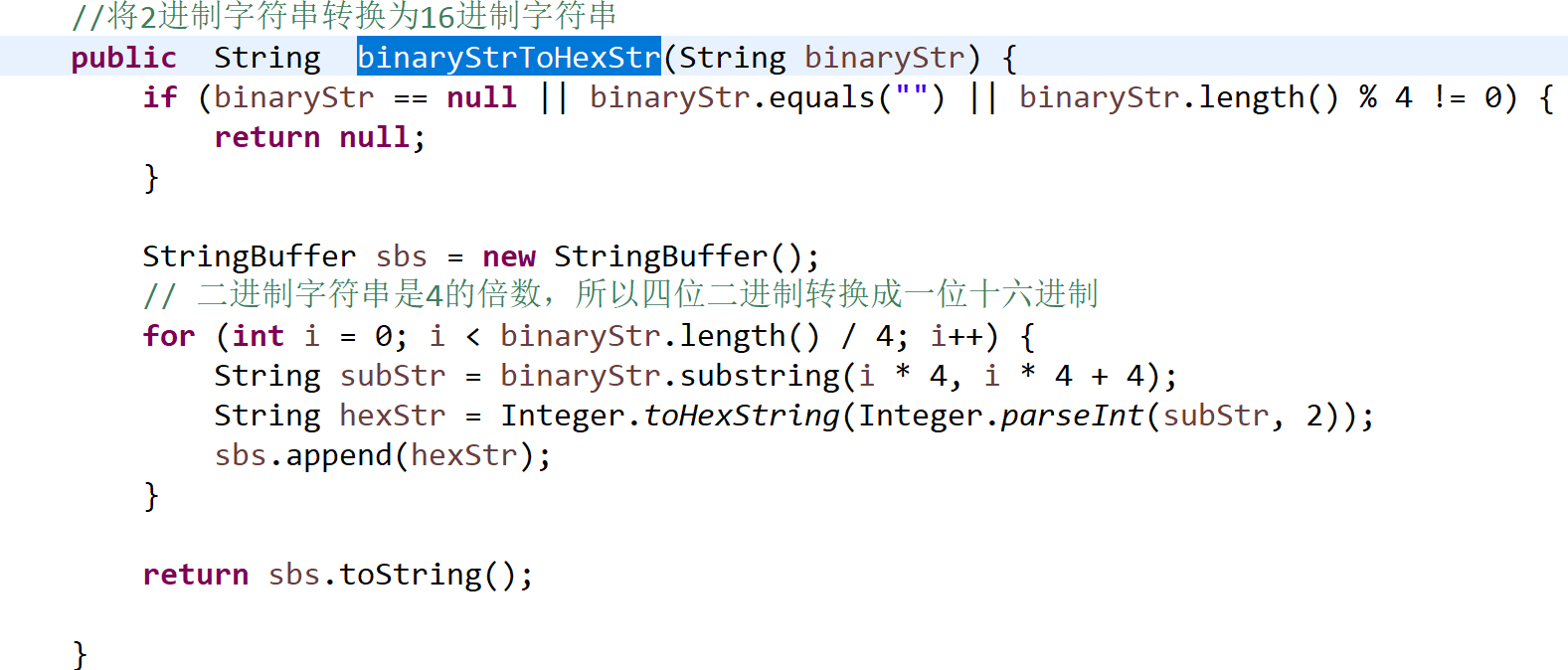


图 4-8

16进制转2进制字符串的方法



图 4-9

2进制字符串去反码的方法，在解析负数的时候需要用到

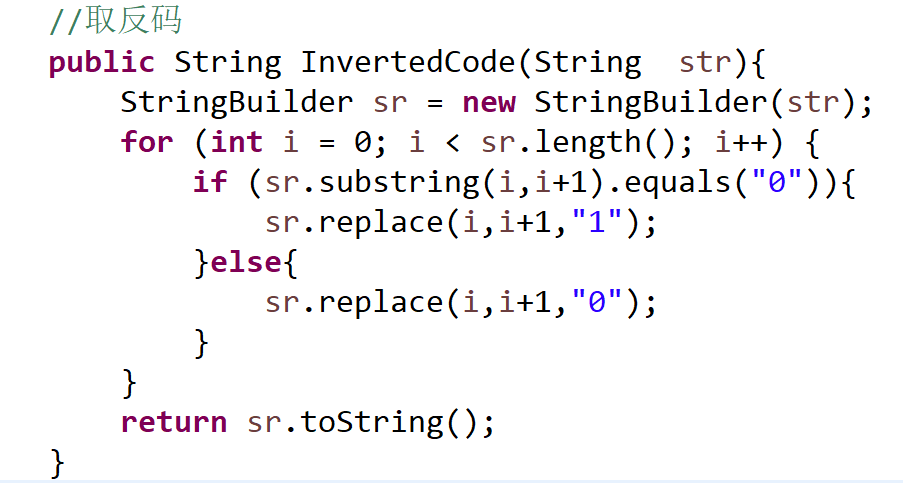


图 4-10

2进制字符串取了反码后的加1操作

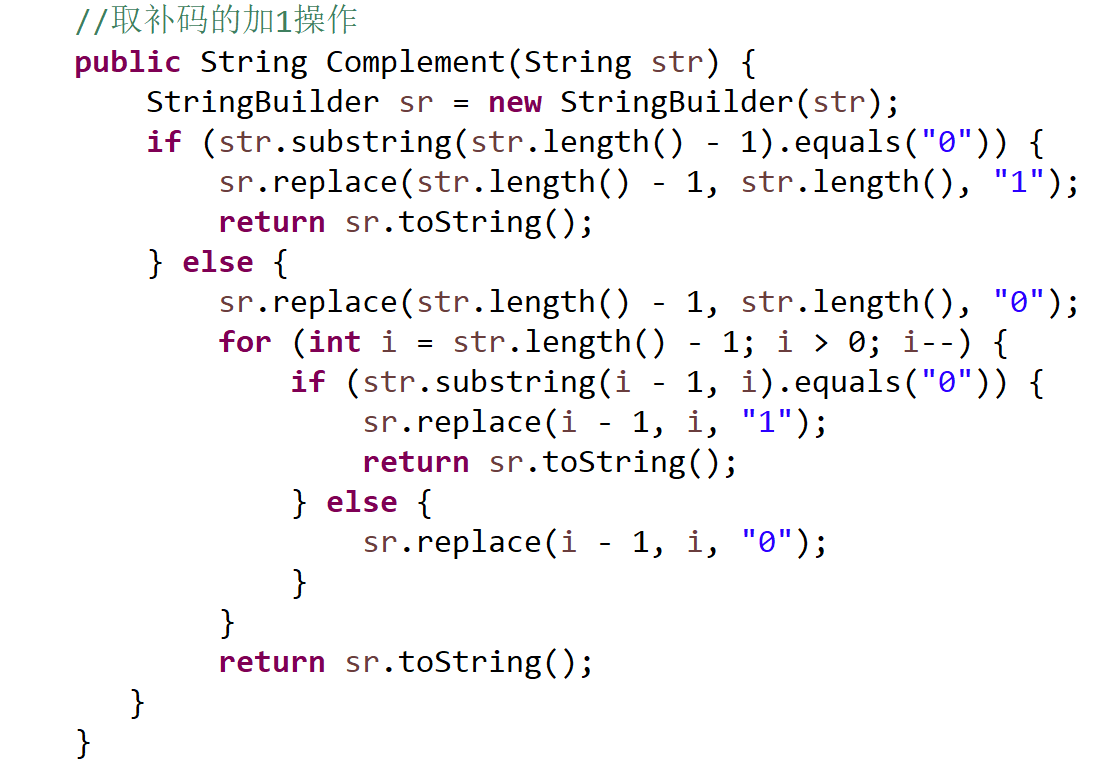


图 4-11

GBK汉字编码 转汉字，char，varchar，varcharmax等类型解析时需要用到

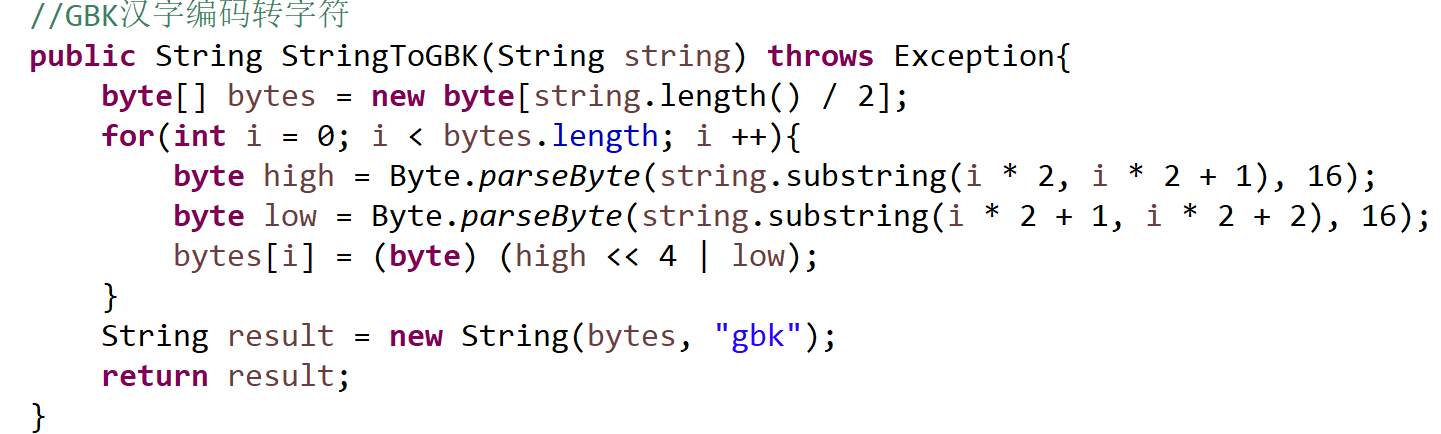


图 4-12

Unicode汉字编码 转汉字 nchar，nvarchar，nvarcharmax等类型解析时需用到

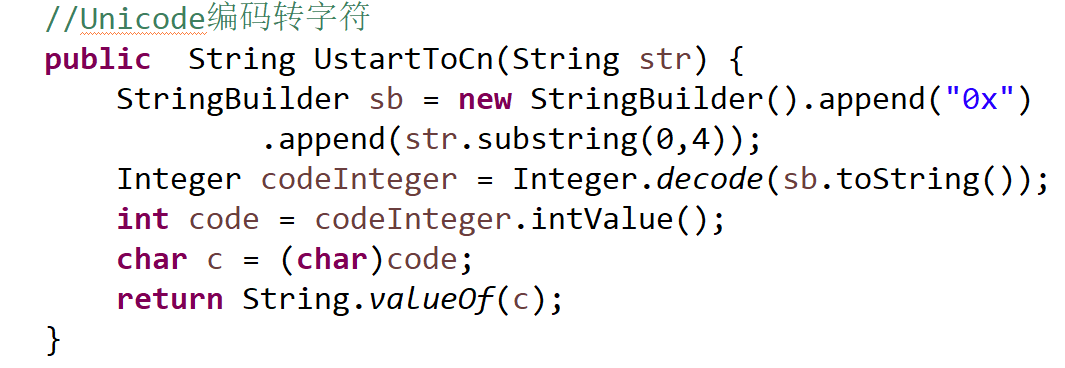


图 4-13