TempDB 使用手册

目录

1、安装

1．1 环境需求

OS: windows 32 位操作系统（包含服务器版本）
JAVA 运行环境，JDK1.5 以上

MYSQL 5.0 或以上
支持 TCP/IP 协议

1.2 安装 TempDB

1.2.1 预备安装

- **安装 JDK**

从 http://java.sun.com/ 下载并安装 Java SDK 1.5 以上环境。并且配置好环境变量。

- **安装 Mysql**

从 http://www.mysql.com/downloads/mysql/ 下载并安装 Mysql。

- **设置 Mysql**

在安装的最后一步，安装程序会弹出对话框询问是否设置 MySql 数据库。此时可以对其进行相关设置。
使用 TCP/IP 网络连接，指定端口号（MySQL 默认为 3306 端口）。如图 1-1 所示。

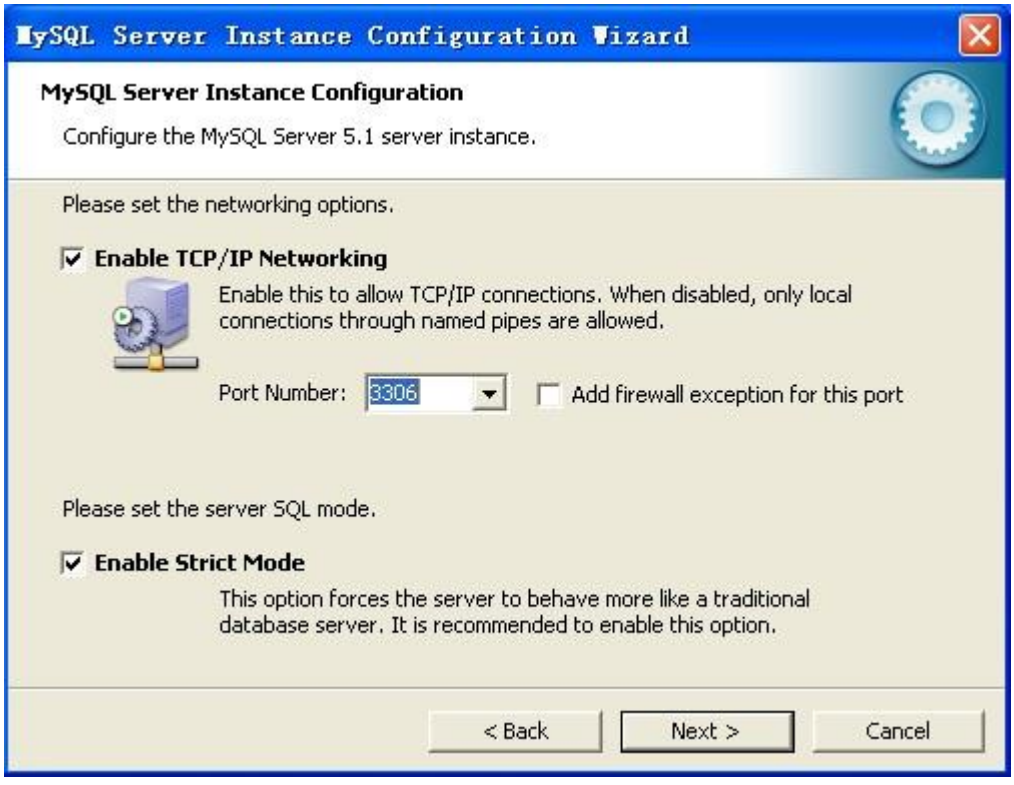

图 1.-1 Mysql 服务器的网络设置
使用 UTF-8 作为默认的字符集，以便于存储和使用中文。如图 1-2 所示。

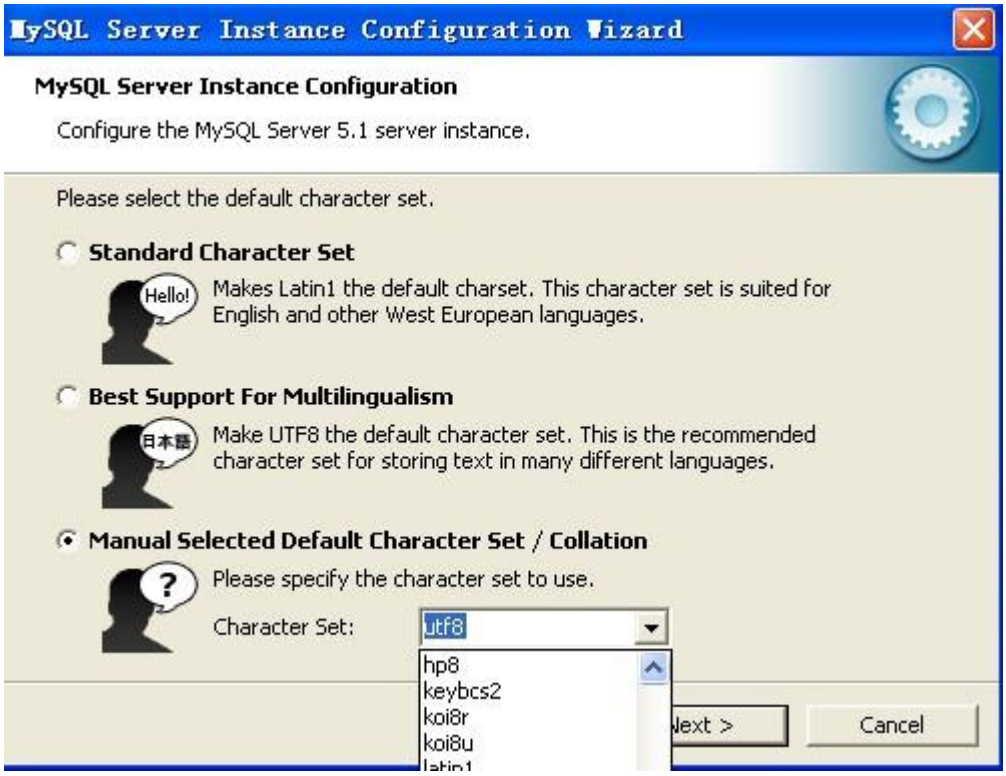

图 1-2 MySql 设置中选择字符集

需要为 root 用户指定密码（由数据库管理员完成）。

1.2.2 安装 TempDB

双击我们提供的安装程序，出现如下的安装界面，如图 2-3 所示：

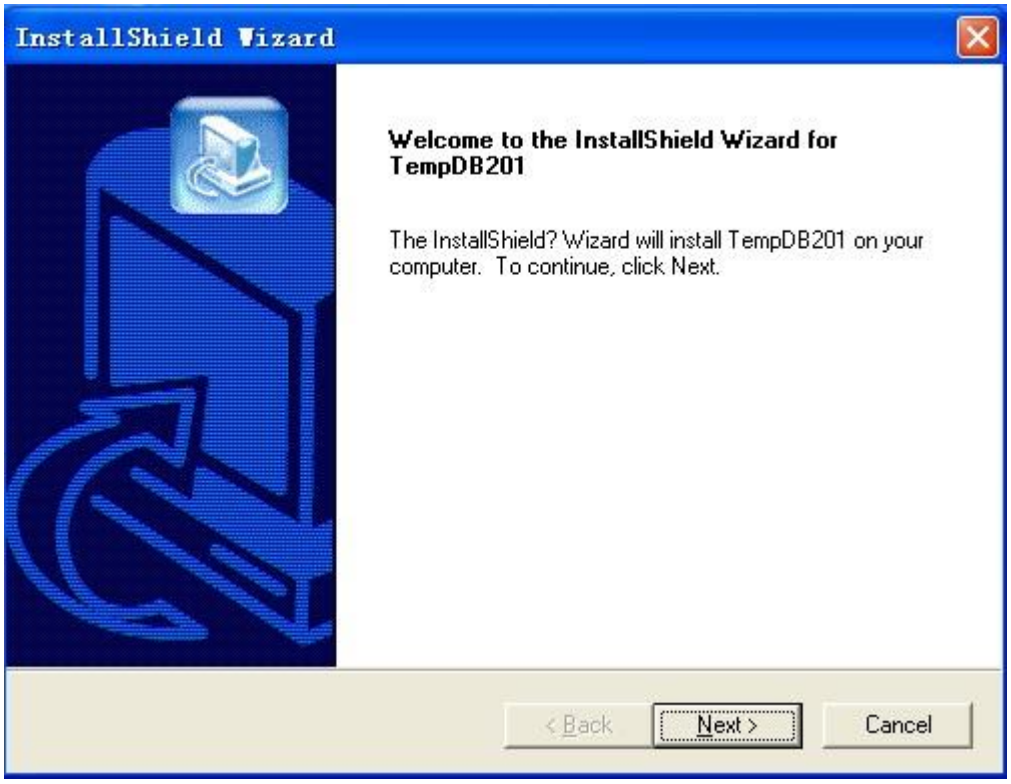

图 1-3 TempDB 安装界面

注意：当提示输入序列号的时候，只需要输入一连串的数字就可以了！如图 2-4 所示。

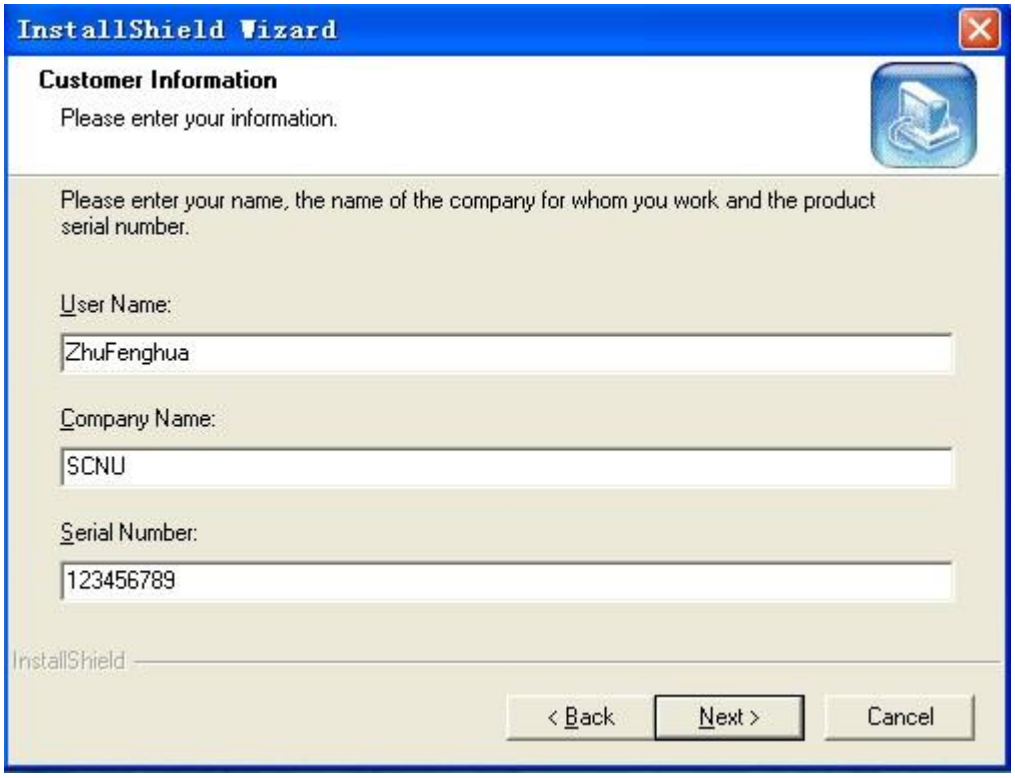

图 1-4 输入序列号

对于我们研究开发人员而言，我们需要采用 custom 的安装方法，以获取 TempDB 的源代码。如图 2-5 所示。接下来，我们在选择组件的界面中，选中“Source Code”，以获得源代码，如图 2-6 所示。

图 1-5 决定何种安装方式

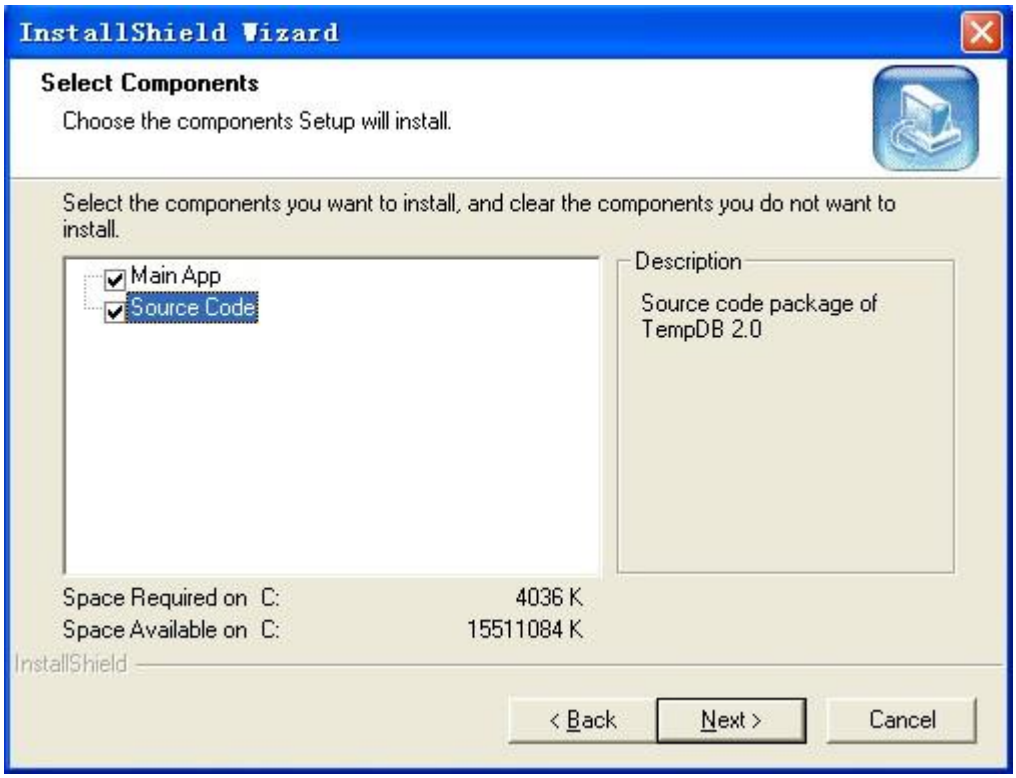

图 1-6 选择组件

1.2.3 卸载 TempDB

我们可从控制面板的“添加或删除”程序中移除本软件。如图 2-7 所示。选择“Remove”从系统中卸载本软件。

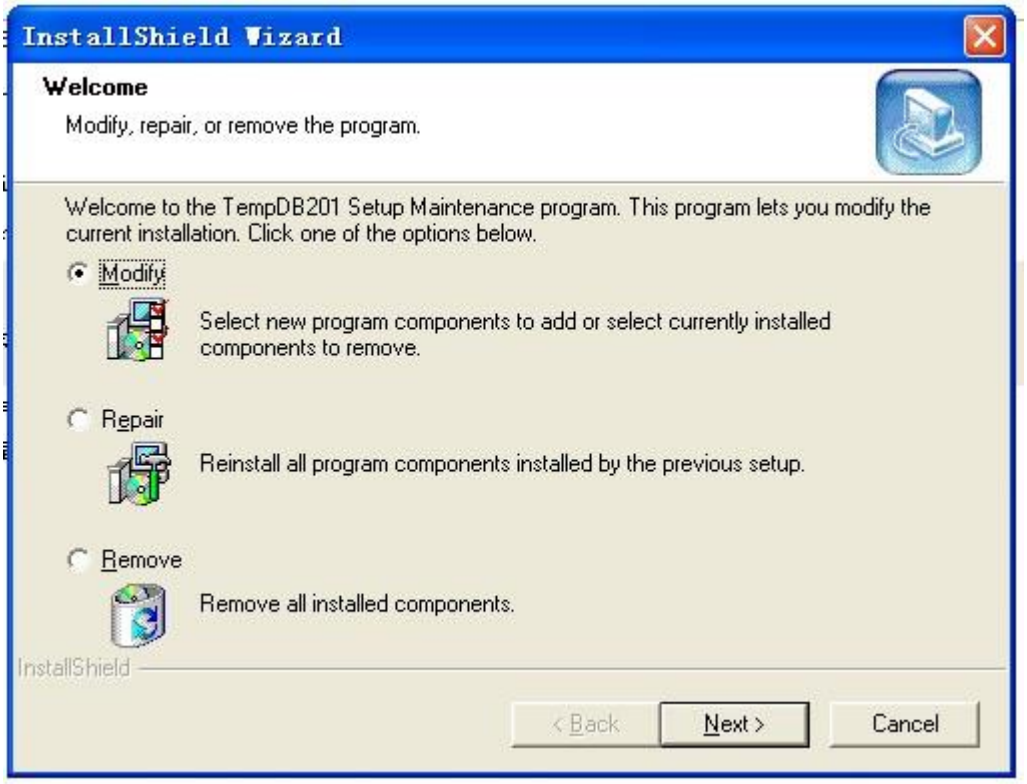

图 1-7 卸载 TempDB

注意：卸载程序不会删除时态数据处理构件运行过程中产生的日志文件（位于安装目录下）。用户需要手动删除这些文件和安装目录。

2. 如何使用 TempDB

2.1 启动和关闭

在正确安装 TempDB 之后，桌面上会出现名字为 “ TempDB2.0 Start”的图标，如图 2-1 所示。

图 2-1 TempDB 图标

我们只需双击该图标即可运行 TempDB。软件界面如图 2-2 所示。其中，后方控制台是用于监控软件运行的状态，请勿在使用的过程中关闭该控制台。而前方是软件的主界面，用于处

理时态数据和管理数据库的。

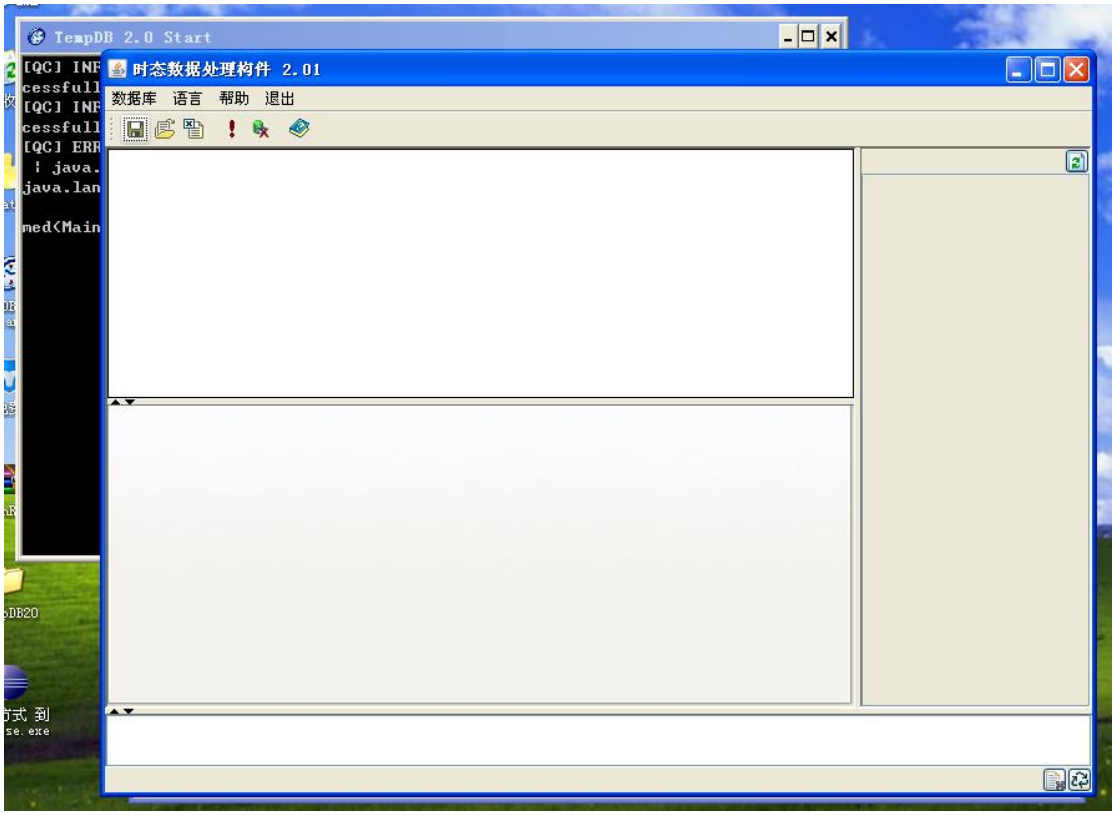

图 2-2 TempDB 的软件界面

注意：如果在启动软件的过程中出现错误的信息，应当遵循以下步骤进行解决：

1、 **请使用 MySql 的导入命令，将软件安装目录下名为：“TDBBatch_template.sql”的 sql 文件进行导入，以恢复我们要进行操作的数据库 tempdb20（由于技术等各方面问题，我们的 TempDB 未能实现“新建数据库”的功能，只能针对该数据库进行操作。）。**

2、 **请选择菜单“数据库”→“配置数据库”重新配置连接（主要针对数据库的 IP 地址，用户名和密码进行设置），设置界面如图**

图 2-3 TempDB 的配置连接界面

当我们使用要关闭 TempDB，只需像关闭我们常见的 windows 软件一样，单击其右上角的红

叉图标或者是单击菜单栏中的“退出”按钮。

2.2 软件的界面

TempDB 的软件界面如图 2-4 所示。用户界面从上到下，从左到右分别是标题栏、菜单栏、工具栏、ATSQL 命令编辑区、结果显示区、数据库列表显示区、系统反馈区。

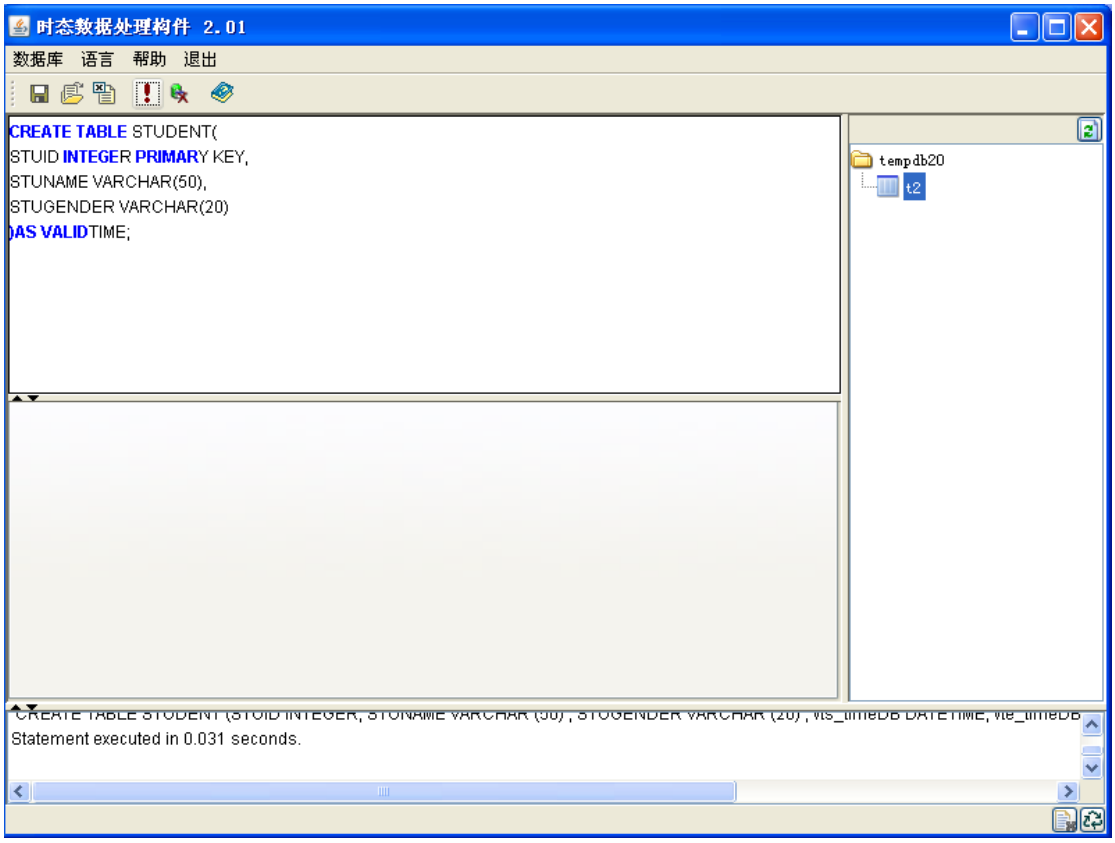

图 2-4 TempDB 软件界面

2.2.1 菜单栏

菜单栏包括“数据库”、“语言”、“帮助”和“退出”共四个菜单选项。

- “数据库”→“配置连接”按钮，弹出配置 DBMS 连接的对话框，如图 2-3 所示。
  - 服务器 IP 地址：DBMS 的安装地址，以及端口号。格式为：“服务器 IP：端口号”，目前只支持 MySql。
  - 数据库名：连接到 DBMS 中的特定数据库（目前只能是 tempdb20）。
  - 用户名：用于连接 DBMS 中的用户名。
  - 密码：上述中用户名所对应的用户密码。
- “数据库”→“创建数据库”按钮，弹出创建数据库对话框，如图 2-4 所示。（未能实现）
- “语言”→“中文（简体）”按钮和“English（US）”按钮，切换我们的 TempDB 的界面语言为简体中文或者是美式英语。
- “帮助”→“关于”按钮，弹出“关于”对话框。
- “退出”→“退出”按钮，退出 TempDB 软件。

2.2.2 工具栏

工具栏如图 2-5 所示。按钮从左到右分别是：

- 保存文件：把当前 ATSQL 命令区的内容保存到文件。
- 打开文件：打开文本文件并把内容加载到 ATSQL 命令编辑区。
- 关闭文件：关闭当前打开的文件。
- 执行语句：执行 ATSQL 命令编辑区的语句。（快捷键 Ctrl+E）
- 停止执行：停止当前正在执行的语句。
- 帮助：打开帮助手册。

图 2-5 工具栏

2.2.3 ATSQL 命令编辑区

ATSQL 命令编辑区用于编辑 ATSQL 语句。编辑区可以同时编辑多个语句，语句间用分号隔开。编辑区同时具有关键字高亮功能。编辑区如图 2-6 所示。

```
CREATE TABLE STUDENT(
STUID INTEGER PRIMARY KEY,
STUNAME VARCHAR(50),
STUGENDER VARCHAR(20)
)AS VALIDTIME;
```

图 2-6 ATSQL 命令编辑区

2.2.4 查询结果显示区

显示查询语句的执行结果。如果 ATSQL 命令编辑区有多个语句，则执行后查询结果显示区会产生多个页签，按顺序每个页签显示对应语句的查询结果。如图 2-7 所示。

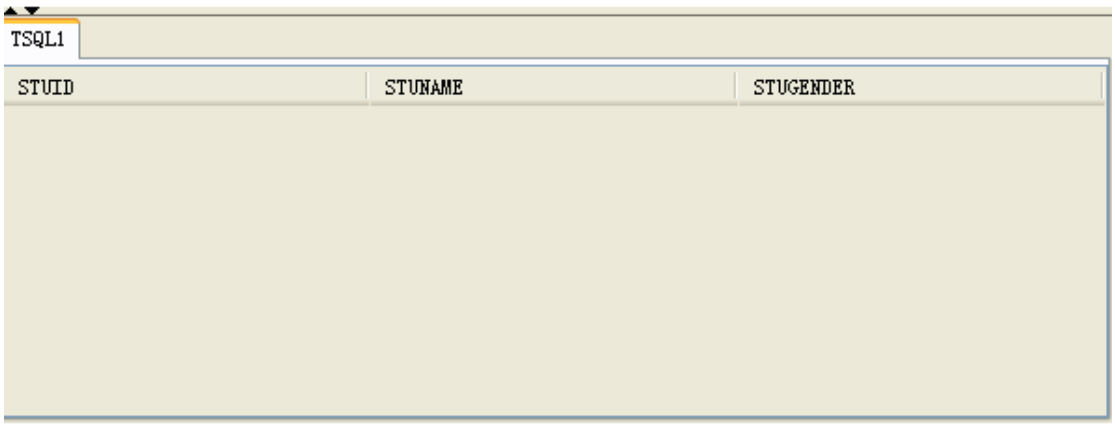

图 2-7 查询结果显示区

2.2.5 数据库表显示区

该区域显示 TempDB 当前连接的数据库名称，以及能够识别的数据库表的名称。如图 2-8 所

示。

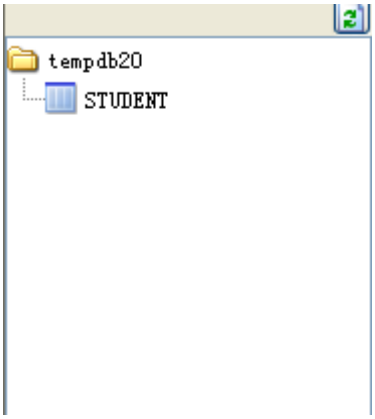

图 2-8 数据库表显示区

2.2.6 系统反馈显示区

该区域显示 ATSQL 语句执行的状态信息。包括，生成的标准 SQL 语句，内部执行时间，完成处理时间、执行结果记录数。同时该区域的右下方有两个按钮，分别用于清空该区域信息和强制垃圾回收。

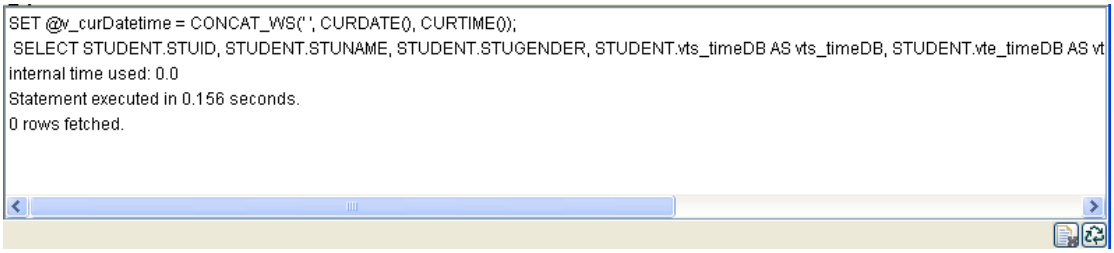

图 2-9 系统反馈显示区

3. TempDB 的 atsql 命令举例说明

本示例是基于一个有四个数据表格的示例数据库。数据库的 ER 图如图 3.1 所示。

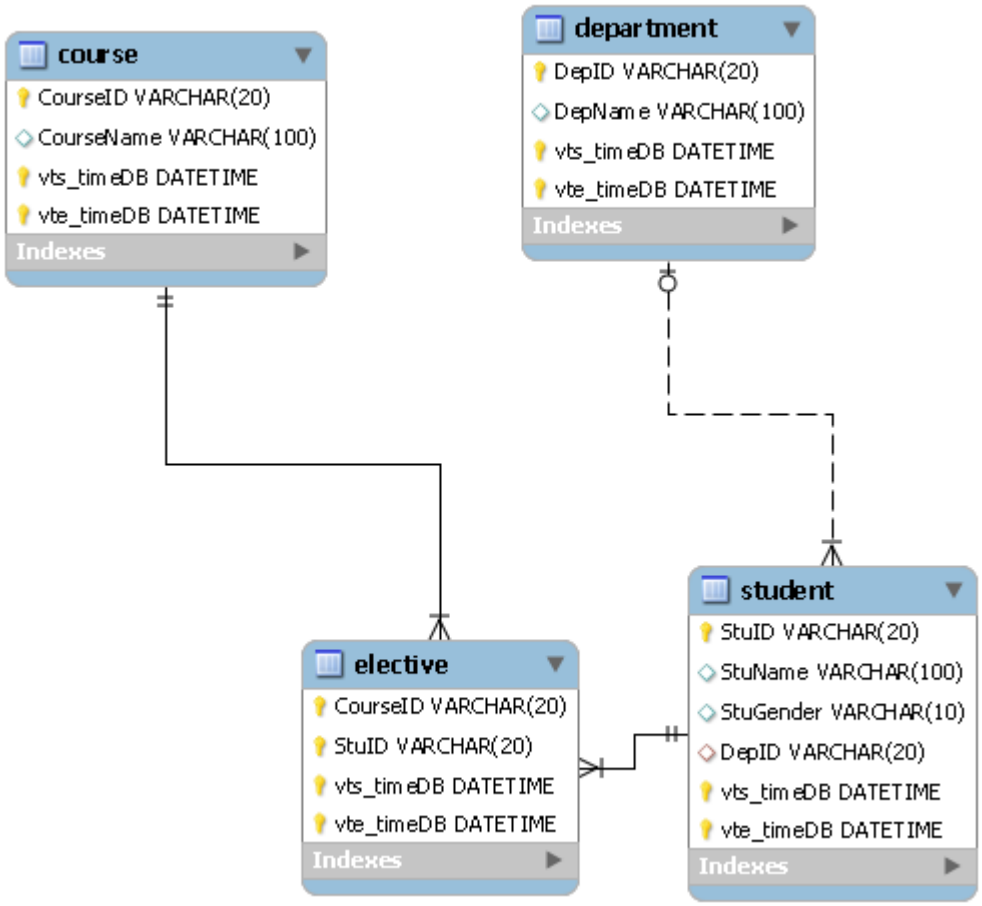

图 3.1 示例数据库 ER 图

值得注意的是，由于目前没有一个通用的结构来表示我们的时态数据库中的表格。因此我们现在示例的 ER 图（图 4.1）是由后台关系型数据库中的表格的角度来说明的。需要说明的是，这个三个时态表格中的主键是为用户指定主键联合两个有效时间列构成的联合主键。

3.1 创建时态表格

由于四个时态表格的创建是具有相似性，因此，本节仅给出 Student 表的创建语句并进行分析。

A、ATSQL 语句

```
create table Student(
StuID varchar(20) primary key,
StuName varchar(100),
StuGender varchar(10),
DepID varchar(20),
constraint FK_Student_Department foreign key(DepID) references Department(DepID)
)as validtime;
```

在此语句中，我们指定了 Student 表格的表结构，并指定创建的是一张时态表。

B、转换后的标准 SQL 语句。

```
CREATE TABLE Student (
StuID VARCHAR (20) ,
 StuName VARCHAR (100) ,
 StuGender VARCHAR (10) ,
DepID VARCHAR (20) ,
CONSTRAINT FK_Student_Department FOREIGN KEY (DepID) REFERENCES Department (DepID) ,
vts_timeDB DATETIME,
vte_timeDB DATETIME,
PRIMARY KEY (StuID, vts_timeDB, vte_timeDB) ) ;
```

从在转换之后的标准 sql 语句可以看到，这个时态表格的创建反映在后台关系数据库中会增加两个时间列用以存储有效时间，并且讲用户指定的逻辑主键和两个有效时间列一同设置为联合主键。

3.2 删除时态表

删除某一特定的时态表和我们的 SQL 操作是一致的。只需要使用“drop table”关键字进行删除即可。

3.3 插入记录

3.3.1 不引起归并操作的时态数据插入

A 时态数据插入语句

```
validtime period[date "1985-01-01"-date “1999-01-01”)
insert into Department values ('0007','School of Public Administration');
```

B 转换后的标准 SQL 语句

```
insert into Department values ('0007','School of Public Administration',
’1985-01-01 00:00:00’,’1999-01-01 00:00:00’);
```

C 执行该插入语句前后数据库表格的相关部分如图 3.2 所示。
时态数据插入前

DEPID	DEPNAME	vts_timeDB	vte_timeDB
0001	school of computer sc...	1985-01-01 00:00:00.0	Now
0002	school of school of m...	1979-01-01 00:00:00.0	Now
0003	school of law	1982-01-01 00:00:00.0	Now
0004	school of geography	1985-01-01 00:00:00.0	Now
0005	school of music	1988-01-01 00:00:00.0	Now
0006	school of chemistry a...	1986-01-01 00:00:00.0	Now

时态数据插入后

DEPID	DEPNAME	vts_timeDB	vte_timeDB
0001	school of computer sc...	1985-01-01 00:00:00.0	Now
0002	school of school of m...	1979-01-01 00:00:00.0	Now
0003	school of law	1982-01-01 00:00:00.0	Now
0004	school of geography	1985-01-01 00:00:00.0	Now
0005	school of music	1988-01-01 00:00:00.0	Now
0006	school of chemistry a...	1986-01-01 00:00:00.0	Now
0007	school of public admi...	1985-01-01 00:00:00.0	1999-01-01 00:00:00.0

图 3.2 不引起归并操作的时态数据插入的执行结果

D 用例解释

此用例展示的是最为普通的时态插入数据，即插入的元组与其他已存在的元组不构成特殊的时态关系。运行该语句后，后台数据库中生成一条对应插入元组的记录。

3.3.2 引起归并操作的时态数据插入

A 时态数据插入语句

```
validtime period[date "1999-01-01"-now)
insert into Department values ('0007','School of Public Administration');
```

此时态插入语句要求插入的元组除了有效时间信息之外，其他信息与上上一条插入的信息都相同，于是会引起 TempDB 做时态归并操作。

B 转换后的标准 SQL 语句

```
Delete from Department
Where DepID='0007' and DepName=' School of Public Administration' and
vts_timeDB>='1985-01-01 00:00:00' and vte_timeDB<='1999-01-01 00:00:00';
Insert into Department
Values('0007','School of Public Administration','1985-01-01 00:00:00','9999-01-01 00:00:00')
```

可以看到，上述的时态插入语句结果 TempDB 的处理之后，生成相应的删除和插入语句，形成相应的时态归并效果。

C 执行该语句前后数据库表格的相关部分情况如图 3.3 所示。

时态数据插入前

DEPID	DEPNAME	vts_timeDB	vte_timeDB
0001	school of computer sc...	1985-01-01 00:00:00.0	Now
0002	school of school of m...	1979-01-01 00:00:00.0	Now
0003	school of law	1982-01-01 00:00:00.0	Now
0004	school of geography	1985-01-01 00:00:00.0	Now
0005	school of music	1988-01-01 00:00:00.0	Now
0006	school of chemistry a...	1986-01-01 00:00:00.0	Now
0007	school of public admi...	1985-01-01 00:00:00.0	1999-01-01 00:00:00.0

时态数据插入后

DEPID	DEPNAME	vts_timeDB	vte_timeDB
0001	school of computer sc...	1985-01-01 00:00:00.0	Now
0002	school of school of m...	1979-01-01 00:00:00.0	Now
0003	school of law	1982-01-01 00:00:00.0	Now
0004	school of geography	1985-01-01 00:00:00.0	Now
0005	school of music	1988-01-01 00:00:00.0	Now
0006	school of chemistry a...	1986-01-01 00:00:00.0	Now
0007	school of public admi...	1985-01-01 00:00:00.0	Now

图 3.3 引起归并操作的时态数据插入的执行效果

请注意红圈中的信息变化。

D 用例解释

此用例展示的是会造成时态归并的时态插入情形，即插入的元组与其他已存在的元组构成了特殊的时态关系。在本例中，TempDB 通过删除已有数据记录和新建数据记录来间接修改了原先的相应元组中的有效时间。

3.4 删除记录

3.4.1 不引起分裂操作的时态数据删除

A. 时态数据删除语句

```
validtime period [date "1987-01-01"-date"2001-01-01")
delete from department;
```

B．转换之后的标准 SQL 语句

```
Delete from department
Where T_GE(department.vts_timeDB, ‘1987-01-01 00:00:00’) and
T_LE(department.vte_timeDB, ’2001-01-01 00:00:00’)
```

C．执行该删除语句前后数据库表格的相关部分情况如图 3.4 所示

时态数据删除前

DEPID	DEPNAME	vts_timeDB	vte_timeDB
0001	school of computer sc...	1985-01-01 00:00:00.0	Now
0002	school of school of m...	1979-01-01 00:00:00.0	Now
0003	school of law	1982-01-01 00:00:00.0	Now
0004	school of geography	1985-01-01 00:00:00.0	Now
0005	school of music	1988-01-01 00:00:00.0	Now
0006	school of chemistry a...	1986-01-01 00:00:00.0	Now
0007	school of public admi...	1985-01-01 00:00:00.0	Now
0008	school of physical	1988-01-01 00:00:00.0	2000-01-01 00:00:00.0

时态数据删除后

DEPID	DEPNAME	vts_timeDB	vte_timeDB
0001	school of computer sc...	1985-01-01 00:00:00.0	Now
0002	school of school of m...	1979-01-01 00:00:00.0	Now
0003	school of law	1982-01-01 00:00:00.0	Now
0004	school of geography	1985-01-01 00:00:00.0	Now
0005	school of music	1988-01-01 00:00:00.0	Now
0006	school of chemistry a...	1986-01-01 00:00:00.0	Now
0007	school of public admi...	1985-01-01 00:00:00.0	Now

图 3.4 不引起分裂操作的时态数据删除的执行结果

D.用例解释

此用例没有显示给出删除条件，而是间接地给出了要删除元组的有效时间范围。转换后的标准 SQL 语句把这个有效时间范围转换成一种显式的区间条件，并按照时态删除的语义把包含于指定有效时间内的所有元组删除

3.5 查询记录

3.5.1 顺序查询

A 查询语句

```
validtime
select * from Department;
```

B 转后的标准 SQL 语句

```
SELECT Department.DEPID, Department.DEPNAME,
Department.vts_timeDB AS vts_timeDB, Department.vte_timeDB AS vte_timeDB
FROM Department WHERE T_LT(Department.vts_timeDB,Department.vte_timeDB);
```

C 执行该查询语句后的结果集如图 3.5 所示。

DEPID	DEPNAME	vts_timeDB	vte_timeDB
0001	school of computer sc...	1985-01-01 00:00:00.0	Now
0002	school of school of m...	1979-01-01 00:00:00.0	Now
0003	school of law	1982-01-01 00:00:00.0	Now
0004	school of geography	1985-01-01 00:00:00.0	Now
0005	school of music	1988-01-01 00:00:00.0	Now
0006	school of chemistry a...	1986-01-01 00:00:00.0	Now
0007	school of public admi...	1985-01-01 00:00:00.0	Now

图 3.5 单表顺序查询的执行结果

D 用例解析

只是一个时态查询的例子。在默认的情况下，时态查询将执行顺序查询的语义，也就是将两个时间列整合成一个有效时间来对待的时态查询。

值得注意的是，在后台数据库中，时态变量 NOW 被记录为“9999-01-01 00:00:00”，如图 3.5 所示，但是界面上显示的是 NOW。这是 TempDB 的结果包装模块在起作用，对转换后的标准 SQL 语句的结果集进行了包装，然后再呈现给用户。

DepID	DepName	vts_timeDB	vte_timeDB
0001	school of computer science	1985-01-01 00:00:00	9999-01-01 00:00:00
0002	school of school of mathematics	1979-01-01 00:00:00	9999-01-01 00:00:00
0003	school of law	1982-01-01 00:00:00	9999-01-01 00:00:00
0004	school of geography	1985-01-01 00:00:00	9999-01-01 00:00:00
0005	school of music	1988-01-01 00:00:00	9999-01-01 00:00:00
0006	school of chemistry and environmen	1986-01-01 00:00:00	9999-01-01 00:00:00
0007	school of public administration	1985-01-01 00:00:00	9999-01-01 00:00:00

图 3.6 在后台单表顺序查询的执行结果

3.5.2 指定字段的顺序查询

A 查询语句

```
validtime
select DE.DEPID, DE.DEPNAME, BEGIN(VALIDTIME(DE))+INTERVAL 3 YEAR AS THREEYEARSLATER
FROM DEPARTMENT AS DE;
```

B 转换后的标准 SQL 语句

```
SELECT DE.DEPID, DE.DEPNAME,
T_ADDTIME( DE.vts_timeDB , '3-0-0 0:0:0') AS THREEYEARSLATER, DE.vts_timeDB AS vts_timeDB,
DE.vte_timeDB AS vte_timeDB FROM DEPARTMENT AS DE
WHERE T_LT(DE.vts_timeDB,DE.vte_timeDB);
```

C 执行语句之后的查询结果集如图 3.7 所示。

DEPID	DEPNAME	THREEYEARSLATER	vts_timeDB	vte_timeDB
0001	school of compute...	1988-01-01 00:00:00.0	1985-01-01 00:00:...	Now
0002	school of school ...	1982-01-01 00:00:00.0	1979-01-01 00:00:...	Now
0003	school of law	1985-01-01 00:00:00.0	1982-01-01 00:00:...	Now
0004	school of geography	1988-01-01 00:00:00.0	1985-01-01 00:00:...	Now
0005	school of music	1991-01-01 00:00:00.0	1988-01-01 00:00:...	Now
0006	school of chemist...	1989-01-01 00:00:00.0	1986-01-01 00:00:...	Now
0007	school of public ...	1988-01-01 00:00:00.0	1985-01-01 00:00:...	Now

图 3.7 指定字段的顺序查询的执行结果

D 用例解析

此例子与上一个例子在形式上的区别仅仅是限定了输出结果集的列。结果列中用用户自定义列“BEGIN(VALIDTIME(DE))+INTERVAL 3 YEAR AS THREEYEARSLATER”，表达的语义是这个自定义列使用 Department 表格的有效时间起点加上 3 年的时间跨度偏移量构成，并将这个列命名为“THREEYEARSLATER”。

3.5.3 快照查询

A 查询语句

```
select *
from Department;
```

B 转换后的标准 SQL 语句

```
SELECT Department.DEPID, Department.DEPNAME
FROM Department WHERE (T_LE(Department.vts_timeDB,'9999-01-01 00:00:00'))
AND T_LE('9999-01-01 00:00:00',Department.vte_timeDB);
```

C 执行该查询语句后的结果集如图 3.8 所示

DEPID	DEPNAME
0001	school of computer science
0002	school of school of mathematics
0003	school of law
0004	school of geography
0005	school of music
0006	school of chemistry and environment
0007	school of public administration

图 3.8 单表快照查询的执行结果

D 用例解析

从转换后的 SQL 语句中可以看出，TempDB 对待在时态表格上的标准 SQL 查询，会默认地将当前有效的元组组织成为结果集输出。所以，在转换后的 SQL 语句中，TempDB 增加了判断元组的有效时间区间是否包含 NOW 的条件。从图 3.8 的结果集中可以看出，该查询语句输出的结果集只包含了当前有效的元组。

3.5.4 带条件的快照查询

A 查询语句

```
select *
from Department de
where de.depid='0001';
```

B 转换之后的 SQL 语句

```
SELECT de.DEPID, de.DEPNAME
FROM Department de
WHERE ( (de.depid='0001') AND T_LE(de.vts_timeDB,'9999-01-01 00:00:00'))
AND T_LE('9999-01-01 00:00:00',de.vte_timeDB);
```

C 执行该查询语句后的结果集如图 3.9 所示

DEPID	DEPNAME
0001	school of computer science

图 3.9 单表带条件的快照查询

D 用例解析

从转换后的 SQL 语句看出，单表带条件的快照查询与单表不带条件的快照查询相差不大，只是原始 ATSQL 语句所带的条件融合有效时间区间约束一起构成了新的显示查询条件。

4. TempDB 编程接口设计

通过这些编程接口，使用者可以通过程序交互的方式使用 TempDB 提供的时态处理能力。

4.1 时态数据构件 TempDB 的编程接口说明

TempDB 中实现了各种时态操作功能。为了方便使用者利用程序与 TempDB 进行交互，TempDB 的接口设计成一个门户类。这个门户类类名为 TDBStatement，处于 TempDB 部署包的 cn.edu.sysu.cosoft.tempdb.service 命名空间下，满足线程安全的要求。

表 4-1 给出了 TDBStatement 类中公开的方法。

表 4-1 TDBStatement 的公共方法

initContext()	**根据默认设置初始化环境**
initContext(String)	**根据参数中指定的配置文件路径初始化环境**
close()	**关闭数据库连接**

createTable(String)	ATSQL2 操作接口，新建表格
createView(String)	ATSQL2 操作接口，新建视图
dropTable(String)	ATSQL2 操作接口，撤销表格
dropView(String)	ATSQL2 操作接口，撤销视图
executeDelete(String)	ATSQL2 操作接口，执行删除
executeInsert(String)	ATSQL2 操作接口，执行插入
executeSelect(String)	ATSQL2 操作接口，执行查询
getDao()	获得数据访问对象

可以看到，通过 TDBStatement 这个门户类，应用程序可以方便地获得 TempDB 的时态处理能力。

使用 TempDB 的时候，使用者可以通过配置文件定制 TempDB 的某些行为。TempDB 的配置文件名称默认为 App.config.xml，默认路径为 TempDB 部署包下面的 conf/App.config.xml。配置文件的格式示例如图 4-1 所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Configuration>
  - <DBProperty>
      <property name="DbType" value="MySql" />
      <property name="databaseName" value="jdbc:mysql://localhost:3306/tempdb20" />
      <property name="username" value="root" />
      <property name="password" value="root" />
      <property name="connection.pool.size" value="sa" />
    </DBProperty>
  </Configuration>
```

图 4-1　配置文档格式示例

通过配置文件，使用者可以配置时态数据软件构件的运行环境，包括：使用的底层数据库类型、并发数目、数据库用户的账号和密码等信息。

4.2　TempDB 编程接口的使用示例

本节给出利用程序与 TempDB 的门户类 TDBStatement 进行交互以完成时态操作的使用示例。

1、initContext()

```
public static void main(String[] args){
TDBStatement statement = new TDBStatement();
statement.initContext();
}
```

执行此段代码，TempDB 便根据默认设置初始化运行环境。

2、initContext(String path)

```
public static void main(String[] args){
TDBStatement statement = new TDBStatement();
statement.initContext("F://config.xml");
```

```
}
```
执行此段代码，TempDB 便根据参数中指定的配置文件“F://config.xml”初始化运行环境。

3、executeSelect(String atsql2)

```
public static void main(String[] args){
TDBStatement statement = new TDBStatement();
statement.initContext();
Object result = statement.executeSelect(args[0]);
}
```
执行此段代码，TempDB 便执行参数 args[0]指定的一条查询语句，并返回结果集存于变量 result 中。

4、executeInsert(String atsql2)

```
public static void main(String[] args){
TDBStatement statement = new TDBStatement();
statement.initContext();
statement.executeInsert(args[0]);
}
```
执行此段代码，TempDB 便执行参数 args[0]指定的一条插入语句。

5、executeDelete(String atsql2)

```
public static void main(String[] args){
TDBStatement statement = new TDBStatement();
statement.initContext();
statement.executeDelete(args[0]);
}
```
执行此段代码，TempDB 便执行参数 args[0]指定的一条删除语句。

附录一　ATSQL2 的 BNF 定义

ATSQL2 为本文采用的时态查询原型，以下为 TempDB 中实现的 ATSQL2 变型语言的 BNF 语法定义：

0. 语句组成

Statement　: = (query | ddl | dml | control) ';'

1. 查询语句

```
timeFlag        ::= [ 'nonsequenced' ] 'validtime' [ identifier | interval ]
coal            ::= '(' 'period' ')'
query               ::= [ timeFlag ] queryExp
queryExp        ::= queryTerm { ('union' | 'except') queryTerm }
queryTerm           ::= queryFactor { 'intersect' queryFactor }
queryFactor     ::= '(' query ')' [ coal ] | sfw
sfw             ::= 'select' selectItemList
                    'from' tableRefList
                    [ 'where' condExp ]
                    [ 'group' 'by' groupByList ]
                    [ 'having' condExp ]
selectItemList  ::= '*' | selectItem { ',' selectItem }
selectItem          ::= scalarExp [ alias ]
tableRefList    ::= tableRef { ',' tableRef }
tableRef        ::= '(' query ')' [ coal ] alias [ '(' colList ')' ] |
                    identifier [ coal ] [ alias ]
alias           ::= ['as'] identifier
condExp         ::= condTerm { 'or' condTerm }
condTerm            ::= condFactor { 'and' condFactor }
condFactor          ::= [ 'not' ] simpleCondFactor
simpleCondFactor::= '(' condExp ')'                                          |
                    'exists' '(' query ')'
|
                    constScalarExp commonOp constScalarExp                   |
                    constScalarExp commonOp ('all' | 'any' | 'some') '(' query ')'
|
                    constScalarExp [ 'not' ] 'between' constScalarExp 'and' constScalarExp   |
                    scalarExp [ 'not' ] 'in' '(' query ')'                   |
                    tempScalarExp timeOp tempScalarExp                       |
                    tempScalarExp timeOp ('all' | 'any' | 'some') '(' query ')'
|
                    eventTerm [ 'not' ] 'between' eventTerm 'and'    eventTerm
```

|

condOp ::= commonOp | timeOp
commonOp ::= '<' | '>' | '<=' | '>=' | '<>' | '='
timeOp ::= 'before' | 'contains' | 'overlaps' | 'meets' | ‘starts’ | ‘finishes’ | 'equals'

groupByList ::= colRef { ',' colRef }
scalarExp ::= constScalarExp | tempScalarExp
constScalarExp ::= term { ('+' | '-') term }
term ::= factor { ('*' | '/') factor }
factor ::= [('+' | '-')] simpleFactor
simpleFactor ::= colRef | const | '(' constScalarExp ')' | 'abs' '(' constScalarExp ')'
colRef ::= identifier ['.' identifier]
const ::= integer | float | ''' string '''

tempScalarExp ::= interval | eventTerm | span { ('+' | '-') span } |
colRef '-' event |
event '-' event

eventTerm ::= event { ('+' | '-') span } |
colRef {('+' | '-') span}

interval ::= 'validtime' '(' identifier ')' |
'period' intervalExp |
'period' '(' eventTerm ',' eventTerm ')'
intervalExp ::= '[' time '-' time ')'
time ::= timeDBDate | eventExp
event ::= ('begin' | 'end') '(' interval ')' |
('first' | 'last') '(' eventTerm ',' eventTerm ')' |
eventExp
eventExp ::= 'now' |
'beginning' |
'forever' |
'date' dateString |
'date' timeDBDate |
'timestamp' timestampString
dateString ::= ''' YYYY '-' MM '-' DD '''
timestampString ::= ''' YYYY '-' MM '-' DD ' ' HH ':' MM ':' SS '''
timeDBDate ::= ''' YYYY ['/' MM ['/' DD ['~' HH [':' MM [':' SS]]]]] '''
span ::= 'interval' spanExp
spanExp ::= integer qualifier { integer qualifier }
qualifier ::= 'year' | 'month' | 'day' | 'hour' | 'minute' | 'second'

2．数据定义语句

```
ddl             ::= ddlTable | ddlView | dropTable | dropView
ddlTable        ::= 'create' 'table' identifier ( tableDef | ddlQuery )
ddlView         ::= 'create' 'view' identifier ddlQuery
tableDef        ::= '(' colDefList ')' [ 'as' 'validtime' ]
ddlQuery        ::= [ '(' colList ')' ] 'as' query
colDefList      ::= colDef { ',' (colDef | tableConstraint) }
colDef          ::= identifier dataType [ columnConstraint ]
columnConstraint::= primKeyCol | refIntegrity | checkConstraint
tableConstraint     ::= ['constraint' identifier] (primKeyTab | foreignKey | checkConstraint)
primKeyCol      ::= 'primary' 'key'
primKeyTab      ::= 'primary' 'key' '(' colList ')'
refIntegrity        ::= 'references' identifier '(' identifier ')'
foreignKey          ::= 'foreign' 'key' '(' colList ')' 'references' identifier '(' colList ')'
checkConstraint     ::= 'check' '(' condExp ')'
colList         ::= col { ',' col }
col             ::= identifier
dataType        ::= 'integer'                       |
                    'float'                         |
                    'char' typeLength               |
                    'varchar' typeLength            |
                    'interval'                      |
                    'datetime'
typeLength          ::= '(' integer ')'
dropTable           ::= 'drop' 'table' identifier
dropView        ::= 'drop' 'view' identifier
```

3. 数据表操作语句

```
dml             ::= insert | delete
insert              ::=insertByValues | insertByQuery
insertByValues      ::= ['validtime' 'period' intervalExp] 'insert' 'into' identifier 'values' '(' valList ')'
insertByQuery       ::=[ timeFlag] 'insert' 'into' identifier queryExp
delete          ::= [ 'validtime' 'period' intervalExp ]    'delete' 'from' identifier [ 'where' condExp ]
valList         ::= val { ',' val }
val             ::= integer | float | ''' string ''' | intervalExp | dateString | timestampString |
timeDBDate
```

4. 控制语句

```
control         ::= 'commit' | 'rollback'
```