虚谷数据库

JDBC 标准接口 V12.3.2

开发指南

文档版本 01

发布日期 2025-01-10

虚谷数据库
XUGU DB

成都虚谷伟业科技有限公司

地址：四川省成都市锦江区锦盛路 138 号佳霖科创大厦 5 楼 3-14 号

邮编：610023

网址：www.xugudb.com

前言

概述

本文档主要介绍虚谷数据库 JDBC 驱动接口的主要功能和其简单的外围应用，旨在帮助使用虚谷数据库服务的应用开发人员快速开发有关于数据库交互的接口编程。

读者对象

- 数据库管理员
- 数据库用户

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
⚠ 注意	用于传递设备或环境安全警示信息，若不避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。
🕮 说明	对正文中重点信息的补充说明。“说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2025-01-10	第一次发布

目录

1 JDBC 概述

1.1 JDBC 介绍

JDBC（Java Database Connectivity）是 Java 语言中用来规范客户端程序如何访问数据库的应用程序接口，提供了诸如查询和更新数据库中数据的方法，可以为多种关系数据库提供统一访问。XuguJDBC 提供了 JAVA 的 JDBC 驱动程序, 它支持 SUN JDBC3.0 和部分 4.0 API 的标准。通过 JDBC 接口对象，数据库开发人员可构建更高级的工具和接口，也可通过此基准编写数据库应用程序完成与数据库的连接、执行 SQL 语句、从数据库中获取结果、状态及错误信息、终止事务和连接等操作。

1.2 系统架构

虚谷数据库的 JDBC 驱动程序（以下简称：虚谷 JDBC）实现了 Java 程序和虚谷数据库通信，支持 SQL 语句对数据库的访问，同时也是构造高级 API 和数据库开发工具的基础。

JDBC 结构如图1-1。

图 1-1 JDBC 结构

1.3 开发流程

使用虚谷 JDBC 实现应用程序与虚谷数据库之间的连接服务，应用程序所在计算机需已安装 JRE（Java Runtime Environment）。虚谷 JDBC 参照 JDBC API 实现，支持 JDBC 3.0 功能，Java 应用程序与虚谷数据库之间的所有操作均需通过虚谷 JDBC 来完成。

用户想要实现对数据库的访问，一般要经过如下的步骤：

1. 通过驱动器管理器获取连接接口。
2. 获得 Statement 或 Statement 子类。
3. 通过 Statement 执行 SQL 命令。

4. 处理执行结果。

5. 关闭 Statement。

6. 关闭连接。

2 虚谷 JDBC 接口和对象类

JDBC 3.0 规范由 JDBC 1.0、JDBC 2.0 衍生而来，主要包含 java.sql 包以及 javax.sql 包，数据库 JDBC 驱动实现 JDBC 3.0 规范所定义的特性接口，即可达到对标准 JDBC 3.0 规范的支持。

虚谷 JDBC 基于 JDBC 3.0 规范实现，实现接口类映射，如表2-1所示。

表 2-1 虚谷 JDBC 实现接口类

接口	虚谷 JDBC 实现类
java.sql.Driver	com.xugu.cloudjdbc.Driver
java.sql.Connection	com.xugu.cloudjdbc.Connection
java.sql.Statement	com.xugu.cloudjdbc.Statement
java.sql.PreparedStatement	com.xugu.cloudjdbc.PreparedStatement
java.sql.CallableStatement	com.xugu.cloudjdbc.CallableStatement
java.sql.ResultSet	com.xugu.cloudjdbc.ResultSet
java.sql.ResultSetMetaData	com.xugu.cloudjdbc.ResultSetMetaData
java.sql.DatabaseMetaData	com.xugu.cloudjdbc.DatabaseMetaData
java.sql.PrameterMetaData	com.xugu.cloudjdbc.PrameterMetaData
java.sql.Types	com.xugu.cloudjdbc.Types
java.sql.Blob	com.xugu.cloudjdbc.Blob
java.sql.Clob	com.xugu.cloudjdbc.Clob
java.sql.Savepoint	com.xugu.cloudjdbc.Savepoint
javax.sql.DataSource	com.xugu.pool.XgDataSource
javax.sql.ConnectionPoolDataSource	com.xugu.pool.XgConnectionPoolDataSource

接口	虚谷 JDBC 实现类
javx.sql.PooledConnection	com.xugu.pool.XgPooledConnection

除了以上标准接口外，虚谷 JDBC 还扩展实现了以下类：

- com.xugu.cloudjdbc.ClientInfo
- com.xugu.cloudjdbc.OutParameter
- com.xugu.cloudjdbc.Rowid
- com.xugu.cloudjdbc.SQLXml
- com.xugu.cloudjdbc.SSL
- com.xugu.pool.StatementWrapper
- com.xugu.pool.PreparedStatementWrapper

3 数据类型

3.1 基本数据类型

标准 SQL 数据类型、虚谷数据库数据类型、Java 数据类型三者映射关系，如表3-1所示。

表 3-1 数据类型映射关系

虚谷数据库数据类型	标准 SQL 数据类型	Java 数据类型
BOOLEAN	BOOLEAN	boolean
SMALLINT	SMALLINT	short
TINYINT	TINYINT	byte
INTEGER	INTEGER	int
BIGINT	BIGINT	long
FLOAT	FLOAT	float
DOUBLE	DOUBLE	double
NUMERIC	NUMERIC	BigDecimal
NUMERIC	DECIMAL	BigDecimal
CHAR	CHAR	String
NCHAR	NCHAR	String
VARCHAR	VARCHAR	String
VARCHAR2	VARCHAR	String
NVARCHAR2	VARCHAR	String
BLOB	BLOB	Blob
CLOB	CLOB	Clob

虚谷数据库数据类型	标准 SQL 数据类型	Java 数据类型
CLOB	NCLOB	Clob
BINARY	BINARY	byte[]
BINARY	VARBINARY	byte[]
BINARY	LONG VARBINARY	byte[]
DATE	DATE	Date
TIME	TIME	Time
TIMESTAMP	TIMESTAMP	Timestamp

3.2 扩展数据类型

虚谷 JDBC 除了实现标准 SQL 数据类型，还根据虚谷数据库自有类型扩展了部分数据类型，如表3-2所示，包括以下数据类型。

1 时间数据类型

- DATETIME：日期时间类型
- DATETIME WITH TIME ZONE：带时区的日期时间类型
- TIME WITH TIME ZONE：带时区的时间类型
- INTERVAL YEAR：年的时间间隔
- INTERVAL MONTH：月的时间间隔
- INTERVAL DAY：日的时间间隔
- INTERVAL HOUR：小时的时间间隔
- INTERVAL MINUTE：分钟的时间间隔
- INTERVAL SECOND：秒的时间间隔
- INTERVAL YEAR TO MONTH：年到月的时间间隔
- INTERVAL DAY TO HOUR：日到小时的时间间隔

- INTERVAL DAY TO MINUTE：日到分钟的时间间隔
- INTERVAL DAY TO SECOND：日到秒的时间间隔
- INTERVAL HOUR TO MINUTE：小时到分钟的时间间隔
- INTERVAL HOUR TO SECOND：小时到秒的时间间隔
- INTERVAL MINUTE TO SECOND：分钟到秒的时间间隔

2 游标数据类型（只在存储过程、函数、包中使用）

- CURSOR：游标数据类型
- SYS_REFCURSOR：返回游标数据类型

3 其他数据类型

- GUID：全局唯一标识符

表 3-2 扩展数据类型对应表

虚谷数据库数据类型	标准 SQL 数据类型	Java 数据类型
DATETIME	DATETIME	Timestamp
DATETIME WITH TIME ZONE	DATETIME_TZ	String
TIME WITH TIME ZONE	TIME_TZ	String
INTERVAL YEAR	INTERVAL_Y	String
INTERVAL MONTH	INTERVAL_M	String
INTERVAL DAY	INTERVAL_D	String
INTERVAL HOUR	INTERVAL_H	String
NTERVAL MINUTE	INTERVAL_MI	String
INTERVAL SECOND	INTERVAL_S	String
INTERVAL YEAR TO MONTH	INTERVAL_Y2M	String

虚谷数据库数据类型	标准 SQL 数据类型	Java 数据类型
INTERVAL DAY TO HOUR	INTERVAL_D2H	String
INTERVAL DAY TO MINUTE	INTERVAL_D2M	String
INTERVAL DAY TO SECOND	INTERVAL_D2S	String
INTERVAL HOUR TO MINUTE	INTERVAL_H2M	String
INTERVAL HOUR TO SECOND	INTERVAL_H2S	String
INTERVAL MINUTE TO SECOND	INTERVAL_M2S	String
CURSOR	CURSOR	无对应类型
SYS_REFCURSOR	REFCUR	无对应类型
GUID	GUID	Timestamp

说明

应用程序使用 ResultSet 对象来对应获取游标所返回数据

3.3 数据类型转换

在调用虚谷 JDBC 驱动中的某些方法时，需进行 Java 数据类型与虚谷数据库数据类型的转换，这些方法包括：

- ResultSet 类 get<Type> 方法
- ResultSet 类 update<Type> 方法
- CallableStatement 类 get<Type> 方法
- PreparedStatement 类 set<Type> 方法

> **⚠ 注意**
>
> 数据类型之间的转换，需遵从 Java 数据类型与虚谷数据库数据类型映射关系规则，否则可能转换失败。

- com.xugu.cloudjdbc.ResultSet 类中 get<Type> 方法的类型转换，如图3-1所示。

表中项目说明 X = 无丢失转换 Y = 直接转换 Z = 依赖转换 — = 不支持转换	boolean(BOOLEAN)	tinyint(TINYINT)	smallint(SMALLINT)	integer(INTEGER)	bigint(BIGINT)	numeric(NUMERIC)	float(FLOAT)	double(DOUBLE)	char(CHAR)	nchar(CHAR)	date(DATE)	time(TIME)	datetime(TIMESTAMP)	time_tz(VARCHAR)	datetime_tz(VARCHA	blob(BLOB)	clob(CLOB)	binary(BINARY)	guid(CHAR)	interval_X(CHAR)
标准 get<Type> 方法																				
getByte()	y	y	z	z	z	z	z	z	z	z	—	—	—	—	—	z	z	z	z	—
getBytes()	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y
getBoolean()	y	x	z	z	z	z	z	z	z	z	—	—	—	—	—	—	—	—	z	—
getshort()	z	y	z	z	z	z	z	z	z	z	—	—	—	—	—	—	—	—	z	—
getInt()	z	x	x	y	z	z	z	z	z	z	x	x	—	x	—	—	—	—	z	x/—
getLong()	z	x	x	x	y	z	z	z	z	z	x	x	x	x	x	—	—	—	z	x
getFloat()	z	x	x	x	x	z	y	z	z	z	—	—	—	—	—	—	—	—	z	—
getDouble()	z	x	x	x	x	z	x	y	z	z	—	—	—	—	—	—	—	—	z	—
getBigDecimal()	z	x	x	x	x	y	x	x	—	—	—	—	—	—	—	—	—	—	—	—
getString()	x	x	x	x	x	x	x	x	y	y	x	x	x	x	x	x	x	x	y	x
getTime()	—	—	—	—	—	—	—	—	—	—	z	y	z	z	—	—	—	—	—	—
getDate()	—	—	—	—	—	—	—	—	—	—	y	z	z	—	—	—	—	—	—	—
getTimestamp()	—	—	—	—	—	—	—	—	—	—	x	—	y	—	z	—	—	—	—	—
getAsciiStream()	—	—	—	—	—	—	—	—	z	z	—	—	—	—	—	z	z	z	—	—
getBinaryStream()	—	—	—	—	—	—	—	—	z	z	—	—	—	—	—	z	z	z	—	—
getCharacterStream()	—	—	—	—	—	—	—	—	z	z	—	—	—	—	—	—	z	—	—	—
getBlob()	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	y	z	x	—	—
getClob()	—	—	—	—	—	—	—	—	z	z	—	—	—	—	—	z	y	z	—	—
getObject()	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y
虚谷 JDBC 特有 get<Type> 方法																				
getInterval()	—	—	—	—	—	—	—	—	—	—	x	x	x	—	—	—	—	—	—	y
getGuid()	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	y	—

图 3-1 接口 get<type> 方法与虚谷数据库数据类型转换关系

说明

- X（无丢失转换）：指以长精度类型的 get<Type> 方法获取短精度数据类型列值无丢失。
  如：使用 getInt() 方法获取数据库数据类型为 smallint 的列数据，获取数据完整。

- Z（依赖转换）：指 get<Type> 方法中的 Type 类型与获取数据库中列的数据类型不一致，此时需要通过其他数据类型转换为 Type 数据类型，此转换过程值可能丢失部分数据。
  如：使用 getInt() 方法获取数据库数据类型为 float 的列数据，可获取 float 列数据的整数值，而小数精度值将丢失。

- Y（直接转换）：指 get<Type> 方法中的 Type 类型与获取数据库中列的数据类型一致。
  如：使用 getInt() 方法获取数据库数据类型为 integer 的列数据，直接获取完整数据。

- com.xugu.cloudjdbc.PreparedStatement 类中 set<Type> 方法的类型转换，如图3-2所示。

说明

- P（允许插入）：指 set<Type> 方法设置数据库中列数据时，数据无损失存入数据库中。
  如：使用 setInt() 方法设置列数据类型为 bigint 的数据时，数据可正确存入。

- Z（部分插入）：指 set<Type> 方法中的 Type 类型和数据库中列数据类型不一致，数据仅能部分正确存入。
  如：使用 setInt() 方法设置数据库列类型为 tinyint 的数据，因 tinyint 表示范围为-128～127 的整数值，故而 tinyint 表示范围内数据可正确存入，范围外数据将插入失败。

表中项目说明 P = 允许参数插入 Z = 部分允许插入 — = 不支持转换	boolean(BOOLEAN)	tinyint(TINYINT)	smallint(SMALLINT)	integer(INTEGER)	bigint(BIGINT)	numeric(NUMERIC)	float(FLOAT)	double(DOUBLE)	char(CHAR)	nchar(CHAR)	date(DATE)	time(TIME)	datetime(TIMESTAMP)	time_tz(VARCHAR)	datetime_tz(VARCHA	blob(BLOB)	clob(CLOB)	binary(BINARY)	guid(CHAR)	interval_X(CHAR)
标准 set<Type> 方法																				
setByte()	p	p	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
setBytes()	—	—	—	—	—	—	—	—	p	p	—	—	—	—	—	p	p	p	p	p
setBoolean()	p	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
setshort()	z	z	p	p	p	p	p	p	p	p	—	—	—	—	—	—	—	—	—	—
setInt()	z	z	z	p	p	p	p	p	p	p	z	—	—	—	—	—	—	—	z	z
getLong()	z	z	z	z	p	p	p	p	p	p	z	—	—	—	—	—	—	—	z	z
setFloat()	z	z	z	z	z	p	p	p	p	p	—	—	—	—	—	—	—	—	z	—
setDouble()	z	z	z	z	z	p	z	p	p	p	—	—	—	—	—	—	—	—	z	—
setBigDecimal()	z	z	z	z	p	p	z	z	p	p	—	—	—	—	—	—	—	—	z	z
setString()	z	z	z	z	z	z	z	z	p	p	z	z	z	z	z	—	z	—	z	z
setTime()	—	—	—	—	—	—	—	—	p	p	p	p	p	p	p	—	—	—	—	z
setDate()	—	—	—	—	—	—	—	—	p	p	p	p	p	p	p	—	—	—	—	z
setTimestamp()	—	—	—	—	—	—	—	—	p	p	p	p	p	p	p	—	—	—	—	—
setAsciiStream()	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	p	p	p	—	—
setBinaryStream()	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	p	p	p	—	—
setCharacterStream()	—	—	—	—	—	—	—	—	z	z	—	—	—	—	—	—	p	—	—	—
setBlob()	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	p	p	p	—	—
setClob()	—	—	—	—	—	—	—	—	z	z	—	—	—	—	—	z	p	z	—	—
setObject()	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p

图 3-2 接口 get<type> 方法与虚谷数据库数据类型转换关系

4 虚谷数据库连接

4.1 获取 JDBC 驱动包

虚谷数据库 JDBC 驱动程序可通过以下两种方式获取：

1. 通过虚谷伟业科技有限公司官方网站获取**https://www.xugudb.com/**。
2. 通过虚谷数据库产品包接口文件获取。

虚谷数据库 JDBC 驱动程序运行在 JDK 1.8 及以上平台。

4.2 部署 JDBC 驱动程序

通过虚谷 JDBC 驱动访问虚谷数据库，首先应保证应用程序正确引用虚谷 JDBC 驱动包（cloudjdbc-*.jar），即将驱动包拷贝至应用程序并添加引用。

4.3 加载 JDBC 驱动程序

在应用程序中进行数据库连接，调用虚谷 JDBC 驱动程序，首先要将虚谷 JDBC 驱动加载到系统内存中，然后供系统使用。虚谷 JDBC 的驱动实现类为 com.xugu.cloudjdbc.Driver，以下是将虚谷 JDBC 驱动加载到内存中的代码：

```
//显示注册
DriverManager.registerDriver(new com.xugu.cloudjdbc.Driver());
//隐式注册
Class.forName("com.xugu.cloudjdbc.Driver");
```

说明

隐式注册即通过 Class.forName() 将对应的驱动类加载到内存中，将执行内存中的 static 静态代码段，此代码段会创建一个虚谷 JDBC 驱动的实例，并放入 DriverManager 中，供 DriverManager 使用。

4.4 连接字串及参数设置

虚谷 JDBC 驱动连接 URL 串格式。

```
jdbc:xugu://serverIP:portNumber/databaseName[?property=value[&
   property=value]]
```

说明

- jdbc:xugu:// ：（必需）虚谷数据库特定协议。
- serverIP ：（必需）虚谷数据库所在服务器的 IP 地址。
- portNumber：（必需）是虚谷数据库服务 TCP 访问端口，默认 5138。
- databaseName：（必需）虚谷数据库数据库名称。
- property：（可选）虚谷数据库的 URL 连接属性。多个连接属性之间使用“&”分隔，不可重复。

property 设置虚谷数据库连接属性，常用连接属性见**表 4-1**。

表 4-1 虚谷数据库连接属性

会话参数名称	连接参数名称	初始值	说明
database	database		数据库名（必填）。
user	user		用户名称（必填）。
password	password		用户密码（必填）。
version	version	301	服务器版本。
encryptor	encryptor	301	用于启动加密数据库，若库未加密，则可不要，但带上此参数不会影响连接。
char_set	charset	GBK	连接字符集。’GB2312’、’GBK’、’UTF8’ 等。

会话参数名称	连接参数名称	初始值	说明
lob_ret	lob_ret		大对象（Blob、Clob）数据返回方式。属性为 descriptor 时，以描述符方式返回；其他以数据流方式返回。默认：以数据流方式返回。
time_zone	timeZone		客户端时区。默认：虚谷数据库配置参数 def_timezone 所设置时区（只对系统时间函数生效）。
iso_level	isoLevel	READ COMMITED	事务隔离级别。'READ COMMITED'：读已提交；'REPEATIABLE READ'：可重复读；'SERIALIZABLE'：序列化。
lock_timeout	lockTimeout	0	表示连接中的事务锁征用时，最大等候时间，单位：毫秒。若在设置时间内加锁失败，回滚事务并报错。
recv_mode	recv_mode	0	接收模式选项。属性值为 0 时，本地全接收模式；属性值为 1 时，网络接收模式；属性值为 2 时，服务器端游标接收模式。
auto_commit	autoCommit	true	事务是否自动提交。属性为 TRUE 时，连接自动提交；属性为 FALSE 时，连接非自动提交。

会话参数名称	连接参数名称	初始值	说明
return_rowid	return_rowid	false	是否返回记录 rowid 值。属性为 TRUE 时，返回 rowid；其他返回 FALSE。
is_debug	Command.debug	false	是否开启调试。
recv_type	discadeRow	false	是否丢弃已经接收到记录。
batch_mode	useBatch	false	PreparedStatement 是否使用批量处理。属性为 TRUE 时，executeBatch 采用批量模型；FALSE 则采用单条处理模型。
uselike	useLike	false	是否使用 like 在 getTables 和 getColumns 中模糊查询。
iscompatibleqx	compatibleQX	false	是否兼容气象局。
islogversion	logVersionDate		是否在 JDBC 日志中输出版本打包日期信息。
isprintversion	printVersionTimes	0	控制打印 JDBC 驱动打包时间信息的开关，只打印一次。
compatibleoracle	isora	false	是否兼容 ORACLE。
emptystringasnull	emptyStringAsNull	false	是否把空串当空处理。
rebuilconnection	rebuildConnection	false	是否在连接被卡断后重建连接。
curfetchsize	cursorFetchSize	1000	recv_mode=2 时，每次从服务器读取记录条数。
xgversion	version	301	客户端协议版本，支持 301 协议，同时兼容 201 协议。

会话参数名称	连接参数名称	初始值	说明
ddlprepare	ddlPrepare	false	是否支持 DDL 语句的 Prepared-Statement。
validatefrequency	validateFrequency	1	活性探测的时间间隔（分钟）。
validateconalive	validateConAlive	true	是否开启死链接探活功能。
validatetimes	validatePerTimes	500	一次 task 任务执行死链接探活的次数。
slowsqltime	slowSQLRecordTime	500	设置慢 SQL 执行时间。SQL 语句执行时间超过该数值，日志则会输出执行 SQL。
schemaon	return_schema	on	查询 SQL 是否返回模式信息。
closecurrresult	closeCurrResult	false	结果集数据全部读取完后，是否自动关闭。
resultkeepsize	resultkeepsize	false	每个 Statement 实例对象保留 ResultSetList 对象的队列大小。
benchmarkmode	benchmarkMode	false	是否开启 benchmark 优化模式。
benchmarkmodewid	benchmarkModeWid	false	是否开启 benchmark 分域优化模式（只有在 benchmark-Mode=true 时，该参数才生效）。
cursorfirstsize	cursorFirstSize	10	recv_mode=2 时，firstResult 从服务器首次读取记录条数。

会话参数名称	连接参数名称	初始值	说明
insertbatchmode	batchOrMoreResultMode	1	控制 INSERT 操作的结果接收方式。1：多括号批量，将多个插入操作合并到一个批量操作中，使用单个 INSERT 语句完成；0：多结果集，每次插入操作会返回一个独立的结果集。
identity_mode	identity_mode		数据库服务器端自增长使用模式。DEFAULT：default 自增；NULL_AS_AUTO_INCREMENT：NULL 自增；ZERO_AS_AUTO_INCREMENT：0 和 NULL 自增。
KEYWORD_FILTER	KEYWORD_FILTER		该参数需要大写，数据库连接配置连接上需要开放的关键字串，用逗号分隔，例如 TABLE,FUNCTION,CONSTANT。
disable_binlog	disable_binlog		不记载 binlog 日志。
current_schema	current_schema		指定连接的模式名。
connect_timeout	sockeTimeOut	3600000	socket 超时时间。
compatiblemode	compatiblemode	NONE	适配其他数据库（MYSQL/ORACLE）。
iskernel	iskernel	false	控制数据库元信息返回。
isSto	isSto	3600000	是否开启分分域功能。

说明

连接参数使用 SSL 属性时，依赖程序包中提供的动态链接库，使用前需将动态链接库复制到操作系统指定位置方可正常使用 SSL 加密功能。

- Windows 32 位操作系统，使用 SSL 属性前，需将程序包 ssl/win32 下 xgssl.dll 动态链接库复制拷贝到系统目录 *C:/windows/system32* 目录位置。
- Windows 64 位操作系统，使用 SSL 属性前，需将程序包 ssl/win64 下 xgssl.dll 动态链接库复制拷贝到系统目录 *C:/windows/system32* 目录下。
- Linux X64 位操作系统，使用 SSL 属性前，需将程序包 ssl/linux64 下 libxgssl.so 动态链接库复制拷贝到系统目录*/usr/lib* 或者*/usr/lib4* 目录下。

虚谷 JDBC 设置连接属性示例如下（版本号、大对象描述符、时区、关闭连接自动提交）：

```
String url=jdbc:xugu://localhost:5138/System?user=GUEST&password=
   GUEST&version=110&auto_commit=false&lob_ret=descriptor&time_zone=
   GMT-8:00;
```

4.5 建立数据库连接

在应用程序将虚谷 JDBC 驱动加载到系统内存中后，可通过 DriverManager 类建立数据库连接，方法如下：

```
DriverManager.getConnection(String url);
DriverManager.getConnection(String url,Properties);
DriverManager.getConnection(String url,String user,String password)
   ;
```

说明

DriverManager 类（即所谓的 JDBC 管理器）将尝试找到可与 URL 所代表的那个数据库进行连接的驱动程序。DriverManager 类存有已注册的 Driver 类清单，当调用方法 getConnection 时，它将检查清单中的每个驱动程序，直到找到可与 URL 中指定的数据库进行连接的驱动程序为止。

5 虚谷数据库数据源和连接池

5.1 使用虚谷数据库数据源创建连接

XgDataSource 提供 set 方法设置数据源连接属性。

示例：

通过 XgDataSource 与虚谷数据库建立连接。

```
com.xugu.pool.XgDataSource ds = new com.xugu.pool.XgDataSource();
ds.setUrl("jdbc:xugu://127.0.0.1:5138/db");
ds.setUser("SYSDBA");
ds.setPassword("SYSDBA");
Properties pro = new Properties();
pro.setProperty("return\_rowid", "true");
pro.setProperty("auto\_commit", "off");
ds.setPro(pro);
//指定数据库登录串。url为数据库登录串
setUrl(String url);
//指定数据库服务器。host为服务器地址
setHostName(String host);
//指定数据库名。dbName为数据库名
setDatabaseName(String dbName);
//指定数据库端口号。port为数据库端口号
setPort(int port);
//指定数据库用户。userName为数据库登录用户名
setUser(String userName);
//指定数据库登录密码。passWord为数据库登录密码
setPassword(Stirng passWord);
//指定数据库登录参数。ds\_pro为数据库连接参数
setPro(Properties ds\_pro);
```

```
//  通过数据源获取数据库连接
Connection conn = ds.getConnection();
```

在 com.xugu.pool.XgDataSource 类中通过 setXX() 方法设置数据源的连接属性，若未指定则使用以下默认属性值：

- 数据库服务器地址：localhost
- 连接数据库：SYSTEM
- 访问端口号：5138
- 访问用户名：GUEST
- 用户密码：GUEST

- URL 默认连接串：jdbc:xugu://127.0.0.1:5138/SYSTEM?user=GUEST&password=GUEST&version=110。

数据源建立后，可通过调用 XgDataSource 类中的 getConnection() 和 (String username,String password) 方法获取数据库连接。

5.2 虚谷数据库连接池

连接池技术的核心思想是：连接复用，通过建立一个数据库连接池以及一套连接使用、分配、管理策略，该连接池中的连接可以得到高效、安全的复用，避免了数据库连接频繁建立、关闭的开销。

虚谷 JDBC 连接池实现类对虚谷数据库原始连接进行了封装，从而方便了数据库应用对于连接的使用（特别是对于事务处理），提高了获取数据库连接效率，也正是因为这个封装层的存在，隔离了应用的本身的处理逻辑和具体数据库访问逻辑，使应用本身的复用成为可能。连接池主要由三部分组成：连接池的建立、连接池中连接的使用管理、连接池的关闭。

虚谷 JDBC 连接池的基本思想：数据库连接建立一个“缓冲池”。预先在“缓冲池”中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。通过设定连接池最大连接数来防止系统无尽的与数据库连接。亦可通过连接池的管理机制监视数据库的连接数量、使用情况，为系统开发、测试及性能调整提供依据。

> **说明**
>
> 当应用程序使用完数据库连接调用连接关闭方法时，该连接并未销毁，而是被归还给连接池等待下次继续调用。

虚谷 JDBC 实现连接池服务所提供的接口类如下：

- XgDataSource 实现 javax.sql.DataSource 接口。
- XgPooledConnection 实现 javax.sql.PooledConnection 接口。
- XgConnectionPoolDataSource 实现 javax.sql.ConnectionPoolDataSource 接口。
- XgConnectionEventListener 实现 javax.sql.ConnectionEventListener 接口。
- XgConnectionEvent 继承自 javax.sql.ConnectionEvent 类。

虚谷数据库连接池中的连接均是通过数据源建立的，故而 xgConnectionPoolDataSource 继承

了 com.xugu.pool.XgDataSource 类。用户亦可通过 XgConnectionPoolDataSource 对象设置数据源属性和连接池属性：

```
//最小连接数
setMinIdle(int min)
//最大连接数
setMaxActive(int max)
//登录空闲时间
setLoginTimeout(int time)
//最大等待连接超时
setMaxWaitTime(long waitTime)
```

> 说明
>
> setLoginTimeout() 和 setMaxWaitTime() 方法中参数单位为毫秒，其他方法中参数单位为秒。

示例：

创建一个虚谷 JDBC 连接池，并通过连接池获取虚谷数据库连接。

```
XgConnectionPoolDataSource CPDSource = new
   XgConnectionPoolDataSource();
// 创建连接池数据源对象
CPDSource.setHostName(“localhost”);
CPDSource.setPort(5138);
CPDSource.setDatabaseName(“xuguNew”);
// 设置要连接的数据库实例名称
CPDSource.setUser(“SYSDBA”);
CPDSource.setPassword(“SYSDBA”);
// 设置登录数据库所需的用户名和密码
CPDSource.setUrl("jdbc:xugu://localhost:5138/xuguNew);
// 设置数据库连接的 URL
CPDSource.setMaxActive(50);
// 配置连接池的最大活动连接数50
CPDSource.setMinIdle(5);
// 指定连接池中最小空闲连接数 5
CPDSource.setLoginTimeout(3000);
// 设置登录数据库的超时时间为 3000 毫秒（3 秒）
CPDSource.setMaxWaitTime(3000);
// 确定了从连接池中获取连接时的最大等待时间为 3000 毫秒（3 秒）
XgPooledConnection Pconn = (XgPooledConnection)CPDSource.
   getPooledConnection();
Connection conn = Pconn.getConnection();
// 获取池化连接
```

5.3 虚谷数据库数据源和连接池

通过 DriverManager 建立数据库连接以外，还可以使用数据源或连接池方式建立数据库连接。

通过数据源方式或连接池方式建立数据库连接有以下优点：

- 资源重用

  由于数据库连接得到重用，避免了频繁创建、释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增进了系统运行环境的平稳性（减少内存碎片以及数据库临时进程/线程的数量）。

- 更快的系统响应

  数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了数据库连接初始化和释放过程的时间开销，从而缩减了系统整体响应时间。

- 新的资源分配手段

  对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接的配置，限制单一应用申请数据库连接数，从而避免单一应用独占所有数据库资源。

数据源实现类为 com.xugu.pool.XgDataSource。

连接池实现类为 com.xugu.pool 包中的 XgDataSource、XgPooledConnection、XgConnectionPoolDataSource、XgConnectionEventListener、XgConnectionEvent。

6 虚谷数据库的 JDBC 交互流程

虚谷 JDBC 驱动实现与虚谷数据库交互的一般使用步骤：

1. 导入虚谷 JDBC 驱动包
2. 注册驱动（仅做一次）
3. 建立连接（Connection）
4. 创建执行 SQL 语句对象（Statement）
5. 执行 SQL 语句
6. 处理执行结果（ResultSet）
7. 释放资源

6.1 导入虚谷 JDBC 驱动包

下载虚谷数据库专用虚谷 JDBC 驱动包（cloudjdbc-*.*.*.jar），并导入到应用程序中。

6.2 加载虚谷 JDBC 驱动

虚谷 JDBC 驱动程序中 Driver 接口实现类为 com.xugu.cloudjdbc.Driver。

加载驱动方式为 DriverManager.registerDriver(new com.xugu.cloudjdbc.Driver()) 或 Class.forName(“com.xugu.cloudjdbc.Driver”)。

6.3 建立数据库连接

6.3.1 打开数据库连接

打开数据库连接即获取数据库连接对象 Connection 类。

如：DriverManager 通过 getConnection() 方法获取 Connection 类对象。

DriverManager 的 getConnection() 方法创建数据库连接的方式亦有多种。

6.3.1.1 getConnection(String url)

参数说明:

url：访问数据库的 URL 路径。

示例

下面的代码利用 getConnection 方法创建与虚谷数据库的连接，并返回连接对象。

```
public Connection getConnection()
{
Connection con=null;
try{
//注册数据库驱动
Class.forName("com.xugu.cloudjdbc.Driver");
//定义连接数据库的url
String url = "jdbc:xugu://localhost:5138/db?user=user1&password=
   pwd";
//获取数据库连接
con = DriverManager.getConnection(url);
System.out.println("数据库连接成功！");
}catch(Exception e){
e.printStackTrace();
}
//返回一个连接
return con;
}
```

6.3.1.2 getConnection(String url,String user,String password)

参数说明：

- url：访问数据库的 URL 路径。
- user：数据库连接用户名。
- password：数据库连接用户密码。

示例：下面的代码利用 getConnection 方法创建与虚谷数据库的连接，并返回连接对象。

```
private Connection con;
private String user = "user1";  //定义连接数据库的用户名
private String password = "pwd";  //定义连接数据库的密码
private String className = "com.xugu.cloudjdbc.Driver";//定义虚谷
   JDBC驱动
private String url = "jdbc:xugu://localhost:5138/db";  /\\*\*创建数
   据库连接\*/
public Connection getConnection(){
try{
Class.forName(className);//加载数据库驱动
System.out.println("数据库驱动加载成功！");
con = DriverManager.getConnection(url,user,password);  //连接数据库
System.out.println("成功地获取数据库连接！");
}catch(Exception e){
System.out.println("创建数据库连接失败！");
con = null;
e.printStackTrace();
} finally {
If(conn != null){
```

```
//释放连接资源
conn.close();
}
}
return con;
}
```

6.3.1.3 getConnection(String url,Properties info)

参数说明：

- url：访问数据库的 URL 路径。
- info：一个持久的属性集对象，包括 user 和 password 等属性。

示例：下面的代码利用 getConnection 方法创建与虚谷数据库的连接，并返回连接对象。

```
public Connection getConnection(){
Connection con = null;  //定义数据库连接对象
Properties info = new Properties();  //定义Properties对象
info.setProperty("user","user1");  //设置Properties对象属性
info.setProperty("password","pwd");
try{
//注册数据库驱动
Class.forName("com.xugu.cloudjdbc.Driver");
//test为数据库名称
String url = "jdbc:xugu://localhost:5138/db";
//获取连接数据库的Connection对象
con = DriverManager.getConnection(url,info);
System.out.println("数据库连接成功！");
}catch(Exception e){
e.printStackTrace();
}
//返回一个连接
return con;
}
```

6.3.2 连接事务管理

数据库事务（transaction）是访问并可能操作各种数据项的一个数据库操作序列，这些操作要么全部执行, 要么全部不执行，是一个不可分割的工作单位。事务由事务开始到事务结束之间执行的全部数据库操作组成。

自动提交模式，即当一条 SQL 语句执行完成后，数据库系统将自动提交该语句所在事务。在虚谷 JDBC 驱动中控制事务自动提交模式有两种方式：

- 通过数据库连接 URL 串的 auto_commit 属性，控制事务的开启与关闭，默认情况下 auto_commit 属性为 on，即自动提交模式。

- 通过 Connection 类的 setAutoCommit(boolean commit) 方法控制事务的开启与关闭，默认情况下 commit 属性为 TRUE，即自动提交模式。

在禁用自动提交模式下，使用 Connection 类的 commit() 方法提交 SQL 语句对数据库做的更改，而 Connection 类的 rollback() 方法将回滚 SQL 语句对数据库的更改，上述两种方法执行完成后均将释放事务持有的全部锁资源。

> ⚠ 注意
>
> 事务仅对数据库 DML 操作有效。

以下为使用 setAutoCommit() 方法控制事务提交的示例代码：

```
//设置事务为非自动提交模式
conn.setAutoCommit(false);
//创建数据库Statement对象
Statement stm = conn.createStatement();
//数据库事务执行SQL语句
String sql = "insert into test (comp_id,comp_name) values(1,'comp')
   ";
//执行静态SQL语句
stm.execute(sql);
//数据库预处理SQL语句
PreparedStatement pstm = conn. prepareStatement( “update test set
   id=? where comp_name=?” )
//预处理SQL语句第一个参数赋值
pstm.setInt(1,15);
//预处理SQL语句第二个参数赋值
pstm.setString(2,'DBMS');
//执行预处理SQL语句
pstm.execute();
//提交事务
conn.commit();
//删除表中指定行数据
stm.execute( “delete from test where id=15” );
//事务回滚方法
conn.rollback();
```

上述代码中执行提交事务操作，执行静态 SQL 语句与执行预处理 SQL 语句对数据库的更改将永久生效；而后执行的删除表中指定行数据操作，因其后调用的 rollback() 方法，delete 命令对数据库的数据更改将被回滚掉，故而对数据库的删除操作不会生效。

6.3.3 连接事务控制

DBC 规范 3.0 中为了增加对事务的控制，新增 Savepoint 接口，其代表事务中的一个逻辑事务点。在非自动提交模式下，一个事务中可以设置多个 Savepoint，在代码中进行事务回滚操作

时，可以指定事务回滚到指定 Savepoint 位置，指定回滚位置前的事务操作仍然保留。此接口大大提高了事务处理的粒度，方便应用程序控制事务处理逻辑。

示例：连接事务控制接口 Savepoint 使用

```
//设置非自动提交
conn.setAutoCommit(false);
//创建Statement对象
Statement stm = conn.createStatement();
//执行SQL命令
stm.execute(sql1);
stm.execute(sql2);
//设置事务回滚点save1
java.sql.Savepoint save1 =conn.setSavepoint("save\_1");
stm.execute(sql3);
//设置事务回滚点save2
java.sql.Savepoint save2 =conn.setSavepoint("save\_2");
stm.execute(sql4);
//设置事务回滚点save3
java.sql.Savepoint save3 =conn.setSavepoint("save\_3");
stm.execute(sql5);
//回滚事务到回滚点save3
conn.rollback(save3);
conn.commit();
```

上述代码段中，事务执行语句 sql5 将被回滚掉。

6.4 创建执行 SQL 语句对象

6.4.1 执行存储过程对象 CallableStatement

CallableStatement 对象实现了从 Java 应用程序中调用数据库存储过程。但此调用过程并不包含存储过程本身，因存储过程存储在数据库中。

CallableStatement 对象为所有的数据库提供了一种以标准形式调用存储过程的方法，通过 Connection 类的 prepareCall() 方法创建。

示例代码：

```
//创建执行存储过程对象CallableStatement
CallableStatement cstmt= conn.prepareCall(“{call 存储过程名(参数表
    列)}”);
```

6.4.2 执行 SQL 对象 PreparedStatement

PreparedStatement 对象使用时，数据库系统对执行 SQL 语句进行预编译处理，预编译 SQL 语句将在未来的使用中被重用。与 Statement 对象相比，PreparedStatement 对象操作多条 SQL 语句的效率更高。

经过预编译并存储在 PreparedStatement 对象中的 SQL 语句，通过 Connection 类的 prepareStatement() 方法创建。

示例代码：

```
//创建执行预编译SQL语句对象PreparedStatement
PreparedStatement preStatement = conn.prepareStatement(“预编译SQL
    语句”);
```

预处理 PreparedStatement 对象相对于静态 Statement 对象的优势：

- PreparedStatement 可以动态传入参数值
- PreparedStatement 比 Statement 更快

  PreparedStatement 会将 SQL 语句预编译在数据库系统中，执行计划同样会被缓存起来，它允许数据库做参数化查询。使用预处理语句比普通的 Statement 查询更快，因做的工作更少（数据库对 SQL 语句的分析、编译、优化已经在第一次查询前完成了）
- PreparedStatement 可以防止 SQL 注入式攻击 PreparedStatement 在使用参数化查询的情况下，数据库系统不会将参数的内容视为 SQL 指令的一部分来处理，而是在数据库完成 SQL 指令的编译后，才套用参数运行，因此就算参数中含有破坏性的指令，也不会被数据库所运行。

6.4.3 执行 SQL 对象 Statement

Statement 对象用于执行静态 SQL 语句和获得 SQL 产生的结果。在虚谷 JDBC 驱动中，虚谷数据库 DDL 操作均通过 Statement 对象来执行。

Statement 对象通过 Connection 类的 createStatement() 方法创建。

示例代码：

```
//创建执行SQL语句对象Statement
Statement stm = conn.createStatement();
```

6.4.4 执行 SQL 语句对象属性扩充

在创建执行 SQL 语句对象时，可通过构造方法的不同参数设置 ResultSet 对象属性，属性包括：

- 结果类型 ResultSetType ResultSetType 默认为 ResultSet.TYPE_SCROLL_INSENSITIVE：

  支持结果集 backforward、last、first 等操作，对数据库中其它 session 进行的数据更改不敏感。

- 结果集并发类型 ResultSetConcurrency ResultSetConcurrency 默认为 ResultSet.CONCUR_READ_ONLY: 不可更新结果集；该属性亦可更改为 ResultSet.CONCUR_UPDATABLE，支持在 ResultSet 中对记录进行修改，修改后记录更新到数据库，修改操作可包含插入、删除或更新。
- 结果集持久类型 ResultSetHoldability。ResultSetHoldability 默认为 ResultSet.HOLD_CURSORS_OVER_COMMIT: 在事务 commit 或 rollback 后，ResultSet 仍然可用。

Connection 类创建 Statement 对象构造方法：

```
//Statement对象扩展结果类型、结果集并发类型构造方法
createStatement(int resultSetType,int resultSetConcurrency);
//Statement对象扩展结果类型、结果集并发类型、结果集持久类型构造方法
createStatement(int resultSetType,int resultSetConcurrency, int
   resultSetHoldability);
```

Connection 类创建 PreparedStatement 对象构造方法：

```
//PreparedStatement对象扩展结果类型、结果集并发类型构造方法
prepareStatement(String sql, int resultSetType,int
   resultSetConcurrency)
//PrepareStatment对象扩展结果类型、结果集并发类型、结果集持久类型构
   造方法
prepareStatement(String sql, int resultSetType,int
   resultSetConcurrency, int resultSetHoldability)
```

Connection 类创建 CallableStatement 对象构造方法：

```
//CallableStatement对象扩展结果类型、结果集并发类型构造方法
prepareCall(String sql, int resultSetType,int resultSetConcurrency)
//CallableStatement对象扩展结果类型、结果集并发类型、结果集持久类型
   构造方法
prepareCall(String sql, int resultSetType,int resultSetConcurrency
   , int resultSetHoldability)
```

6.5 执行 SQL 语句

6.5.1 通过 Statement 进行批量数据操作

⚠ 注意

- 执行批处理操作前，自动提交模式设置为 FALSE
- 执行批处理操作，并进行手动提交（commit）
- 执行批处理操作后，自动提交模式恢复为 TRUE
- 执行批处理操作异常发生时，进行 rollback(回滚)

示例：通过 Statement 批量操作数据

```
try {
//创建Statement对象
stm = con.createStatement();
//设置事务非自动提交
con.setAutoCommit(false);
//  添加多条SQL语句到批处理中
stm.addBatch("insert into student values(23,'tangbao','高数',100)")
   ;
stm.addBatch("insert into student values(24,'王定','c#',98)");
stm.addBatch("insert into student values(25,'王国云','java',90)");
//执行批处理操作
stm.executeBatch();
System.out.println("插入成功!");
//执行成功后进行数据提交
con.commit();
System.out.println("提交成功!");
//关闭非自动提交
con.setAutoCommit(true);
} catch (SQLException e) {
if (!con.isClosed()) {
con.rollback(); //出现异常，回滚事务
System.out.println("提交失败，回滚！");
con.setAutoCommit(true); //关闭非自动提交
}
} finally {
if (stmt != null) {
//关闭操作对象
stmt.close();
}
}
```

6.5.2 通过 PreparedStatement 进行批量数据操作

⚠ 注意

- 执行批处理操作前，自动提交模式设置为 FALSE
- 执行批处理操作，并进行手动提交（commit），数据量大时，设置手动提交间隔
- 执行批处理操作后，自动提交模式恢复为 TRUE
- 执行批处理操作异常发生时，进行 rollback（回滚）
- PreparedStatement 执行语句不能为 DDL 语句

示例：将 10000 条记录插入数据库中，每 100 条执行一次提交操作

```
try{
conn.setAutoCommit(false); //设置非自动提交
//创建PreparedStatement对象
ps = conn.prepareStatement("insert into testtab values (?, ?, ?)");
//循环添加批处理数据
for(int i = 0;i<list.size();i++){
HashMap map = list.get(i);//list为数据封装，此处略
ps.setString(1, map.get("id").toString());
ps.setString(2, map.get("name").toString());
ps.setString(3, map.get("age").toString());
ps.addBatch();
if(mod(i,100)==0){//每100条执行一次执行操作
ps.executeBatch();//执行批量插入
conn.commit();//执行事务提交
}
}
ps.executeBatch();//执行批量插入
conn.commit(); //提交事务
}catch(Exception e){
if (!conn.isClosed()) {
conn.rollback(); //出现异常，回滚事务
System.out.println("提交失败，回滚！");
conn.setAutoCommit(true); //关闭非自动提交
}
} finally {
if (ps!= null) {
ps.close();//关闭操作对象
}
}
```

6.5.3 CallableStatement 绑定参数

CallableStatement 存在三种类型的参数 IN、OUT 和 INOUT。

- IN：创建 SQL 语句时其参数值未知。使用 setXXX() 方法将值绑定到 IN 参数。

- OUT：由 SQL 语句返回的参数值。可以使用 getXXX() 方法从 OUT 参数中检索值。
- INOUT：提供输入和输出值的参数。使用 setXXX() 方法绑定变量并使用 getXXX() 方法检索值。

示例 1：通过参数下标绑定参数

```
//存储过程调用语句
String sql ="{call pro_2(?,?,?)}";
//创建CallableStatement对象
CallableStatement cstm =  conn.prepareCall(sql);
//通过参数下标设置参数(IN参数)
cstm.setInt(1, 20490);
//注册OUT参数数据类型
cstm.registerOutParameter(2, com.xugu.cloudjdbc.Types.VARCHAR);
cstm.registerOutParameter(3, com.xugu.cloudjdbc.Types.VARCHAR);
//执行存储过程调用
cstm.execute();
//获取存储过程输出参数
String name = (String)cstm.getObject(2);
String address = (String)cstm.getObject(3);
```

示例 2：通过参数名绑定参数

```
//存储过程调用语句
CallableStatement pstm = conn.prepareCall("insert into tt1 values(:
   id,:name)");
//通过参数名设置参数(IN参数)
pstm.setInt("id", 1);
pstm.setString("name", "cloud");
//执行存储过程调用
pstm.execute();
```

6.6 处理执行结果

6.6.1 更新 ResultSet 结果集

创建 Statement 对象时，设置 Statement 属性 RresultSetConcurrency 为 ResultSet.CONCUR_UPDATABLE，并且数据库 URL 连接串属性 return_rowid 设置为 TRUE 时，Statement 对象执行 SQL 语句返回的 ResultSet 结果集才能被更新。当 ResultSet 对象为可更新结果集时，将光标移至插入位置，可调用 ResultSet 类对应的 update 方法修改结果集中行数据，并通过 updateRow 方法将更改数据更新到数据库中。

示例：通过修改结果集中数据值更改数据库中对应记录

```
//创建Statement对象并设置结果集属性为可更新
Statement stm = conn.createStatement(ResultSet.TYPE\_SCROLL\
   _INSENSITIVE, ResultSet.CONCUR\_UPDATABLE);
//执行SQL语句
```

```
ResultSet rs = stm.executeQuery(sql);
//结果集遍历
while(rs.next())
{
//更新结果集中指定列数据
rs.updateLong(1, 5);
//更新结果集中指定列数据
rs.updateFloat(3, 12.3f);
//结果集变更回写入数据库
rs.updateRow();
}
```

通过调用 ResultSet 类的 next()、previous()、absolute(int) 方法移动光标位置，将光标位置移动到指定数据行，通过 ResultSet 类对应的 get 方法获取行数据。

> 📖 **说明**
>
> 如使用结果集更新数据库数据，需在建立数据库连接时，设置连接属性 return_rowid 为 TRUE，且在生成执行 SQL 语句对象时，并发属性指定为 ResultSet.CONCUR_UPDATABLE。

6.6.2 通过移动光标获取结果集条数

示例：统计结果集总条数

```
//创建Statement对象
Statement stm= conn.createStatement();
stm.execute("create table test(id int identity(1,1));insert into
   test values(default)");
//执行结果集查询
ResultSet rs = stm.executeQuery("select * from test")
//移动光标至结果集末尾
rs.last();
//获取结果集行数
int rsCounts = rs.getRow();
```

6.6.3 多结果集处理

虚谷 JDBC 支持多结果集处理，允许 Statement 接口支持多重打开的 ResultSet 对象。执行语句可以是多条 SQL 语句的集合，即一次性向虚谷服务器发送多条 SQL 语句，虚谷服务器执行完 SQL 语句后返回多个执行结果；亦或存储过程定义有多个返回结果集。虚谷 JDBC 的 Statement 重载了 Statement 接口中的 getMoreResults() 方法。

示例：多结果处理

```
//SQL语句包含多条查询命令
```

```
String sql=" SELECT * FROM A1 WHERE D2=(SELECT AVG(D2) FROM A1
   GROUP BY D1);SELECT * FROM A1 WHERE D2>(SELECT MIN(D2) FROM A1
   GROUP BY D1)";
//创建Statement对象
Statement stm = conn.createStatement();
//执行SQL语句
boolean f = stm.execute(sql);
//判断是否有ResultSet对象产生
if(f == true){
//获取结果集对象
ResultSet rs1 = stm.getResultSet();
... ... ... ...
}
while(true)
{
//判断Statement是否返回了多个ResultSet对象
if(!stm.getMoreResults(Statement.CLOSE\_CURRENT\_RESULT)){
break;
} else  if((rs2 = cstm.getResultSet())!=null) {
... ... ... ...//数据处理部分省略
}
}
```

6.6.4 服务器端游标使用

虚谷 JDBC 中服务器端游标结果集生成需满足两个条件：URL 连接串属性 recv_mode 值为 2，Statement 对象执行 SQL 语句为单条 SELECT 语句。

应用程序端通过 Statement 类的 setFetchSize(int) 方法或 ResultSet 类的 setFetchSize(int) 方法设置每次从服务器端获取多少条数据。

> **说明**
>
> 使用服务器端游标的 ResultSet 对象，光标只能单向向前正向移动，不能向后移动或来回滚动。

示例：虚谷数据库服务器端游标使用

```
//创建虚谷JDBC特有Statement对象
Statement stm = (com.xugu.cloudjdbc.Statement)conn.createStatement
   ();
//开启服务器端游标
stm.setServerCursor(true);
//设置服务器端游标单次返回数据条数
stm.setFetchSize(1000);
//执行SQL语句
ResultSet rs = stm.executeQuery("select * from t1");
//遍历数据
while（rs.next()）
{
```

```
... ... ... ...
}
//关闭服务器端游标
stm.setServerCursor(false);
```

6.6.5 检索自动产生的关键字 GeneratedKeys

JDBC 3.0 规范中对于自动产生的关键字数值，定义了接口规范，即：getGeneratedKeys() 方法。

为了解决对获取自动产生的或自动增加的关键字的值的需求，虚谷 JDBC 根据 JDBC3.0 规范提供了获得关键字值的方法。获取关键字的值，需要在执行方法中指定自增关键字标记。SQL 语句执行完后，调用 Statement 对象的 getGeneratedKeys() 方法，获得包含自增长值的 ResultSet 对象。

示例：获取数据库自增长关键字的值

```
//创建Statement对象
Statement stmt = conn.createStatement();
//指定自增长关键字的标识
String[] colN = {"id"};
//在执行方法中引入自增长关键字标识
stmt.executeUpdate("INSERT INTO authors(first\_name, last\_name)
   VALUES ( 'Ghq' , 'Wxl' )",colN);
//获取结果集
ResultSet rs = stmt.getGeneratedKeys();
if ( rs.next() ) {
//获取结果集中的自增长值
int key = rs.getInt();
}
```

6.7 释放资源

在应用程序中因处理需要所建立的 ResultSet、Statement、PreparedStatement 等数据库资源对象，在使用完后必须关闭。关闭数据库资源对象，直接调用对应的数据库资源对象 close() 方法即可。

当不再需要和虚谷数据库进行交互时，需断开数据库连接。调用 Connection 类的 close() 方法即会关闭数据库连接，同时释放连接上所持有的所有数据库资源。

⚠ 注意

数据库连接为非自动提交时，数据库资源对象在使用完后，若未及时关闭，此时进行 DDL 操作将会导致虚谷数据库加锁超时或失败。

7 大对象类型处理

7.1 向表中插入大对象

LOB 对象的插入必须通过 PreparedStatement、CallableStatement 类的 setXXX() 方法方式实现。虚谷 JDBC 中 PreparedStatement 提供多种方法设置大对象参数值，如下列方法：

```
setBinaryStream(int parameterIndex,InputStream x)
//用于在 SQL 预处理语句（如PreparedStatement）中设置二进制流参数
setBlob(int parameterIndex, Blob x)
// 用于将一个Blob对象设置为 SQL 预处理语句中的参数
setBlob(int parameterIndex, InputStream x)
// 将一个InputStream中的二进制数据设置为 SQL 预处理语句中的参数
setBytes(int parameterIndex,byte[] x)
// 用于将一个字节数组作为参数设置到 SQL 预处理语句中
setCharacterStream(int parameterIndex, Reader reader)
//用于将一个字符流（Reader）设置为 SQL 预处理语句中的参数
setClob(int parameterIndex, Clob x)
//用于将一个Clob对象设置为 SQL 预处理语句中的参数
setClob(int parameterIndex, Reader reader)
// 用于将一个字符流（Reader）设置为 SQL 预处理语句中的参数，但它会
   在内部将字符流的数据转换为Clob对象进行处理
setString(int parameterIndex, String x)
// 用于将一个字符串设置为 SQL 预处理语句中的参数
```

示例：应用程序插入大对象数据，其中表 b1 字段 photo 为 blob 类型，字段 text 为 clob 类型

```
PreparedStatement ps = conn.prepareStatement("INSERT INTO b1 VALUES
    (?, ?)");
// blobBytes is a byte[] type
ps.setBlob(1, new com.xugu.cloudjdbc.Blob(blobBytes));
// clobString is a String type
ps.setClob(2, new com.xugu.cloudjdbc.Clob(clobString));
ps.execute();
```

7.2 查询大对象数据

通过虚谷 JDBC 获取大对象值，数据库 CLOB 对象可通过字符串形式或流式数据形式获取，数据库 BLOB 对象只通过流式数据形式获取。

示例 1：流式获取 CLOB 对象

```
Statement stm = xgConn.createStatement();
// 创建 Statement 对象（Statement stm = xgConn.createStatement();）
String sql=”select remark  from  t_clob  where id=101”;
// 定义 SQL 查询语句（String sql=”select remark from t_clob where
   id=101”;）
```

```
ResultSet rs = stm.executeQuery(sql);
//执行查询并获取结果集（ResultSet rs = stm.executeQuery(sql);）
while(rs.next)
{
Clob data = rs.getClob("remark ");
//以字符流方式获取数据
Reader rd = data.getCharacterStream();
//获取该 Clob 对象所代表的字符大对象数据的字符流表示形式，返回一个
   Reader 类型的对象
char[] sc = new char[1000];
StringBuffer sb = new StringBuffer();
int len = 0;
while((len=rd.read(sc)) !=-1)
{
sb.append(new String(sc,0,len));
}
System.out.println(sb.toString());
}
//输出展示数据
```

示例 2：虚谷 JDBC 允许以字符串的形式读取 CLOB 对象

```
String sql="select text from t_clob";
//定义 SQL 查询语句（String sql="select text from t_clob";）
Statement stm=conn.createStatement();
//创建 Statement 对象（Statement stm=conn.createStatement();）
ResultSet rs=stm.executeQuery(sql);
// 执行查询并获取结果集（ResultSet rs=stm.executeQuery(sql);）
while(rs.next())
{
String clobStr =rs.getString("text");
}
// 遍历结果集获取数据
```

7.3 虚谷 JDBC 接口和对象类

JDBC 3.0 规范由 JDBC 1.0、JDBC 2.0 衍生而来，主要包含 java.sql 包以及 javax.sql 包，数据库 JDBC 驱动实现 JDBC 3.0 规范所定义的特性接口，即可达到对标准 JDBC 3.0 规范的支持。

虚谷 JDBC 基于 JDBC 3.0 规范实现，实现接口类映射，如表 7-1 所示。

表 7-1 虚谷 JDBC 实现接口类

接口	虚谷 JDBC 实现类
java.sql.Driver	com.xugu.cloudjdbc.Driver

接口	虚谷 JDBC 实现类
java.sql.Connection	com.xugu.cloudjdbc.Connection
java.sql.Statement	com.xugu.cloudjdbc.Statement
java.sql.PreparedStatement	com.xugu.cloudjdbc.PreparedStatement
java.sql.CallableStatement	com.xugu.cloudjdbc.CallableStatement
java.sql.ResultSet	com.xugu.cloudjdbc.ResultSet
java.sql.ResultSetMetaData	com.xugu.cloudjdbc.ResultSetMetaData
java.sql.DatabaseMetaData	com.xugu.cloudjdbc.DatabaseMetaData
java.sql.PrameterMetaData	com.xugu.cloudjdbc.PrameterMetaData
java.sql.Types	com.xugu.cloudjdbc.Types
java.sql.Blob	com.xugu.cloudjdbc.Blob
java.sql.Clob	com.xugu.cloudjdbc.Clob
java.sql.Savepoint	com.xugu.cloudjdbc.Savepoint
javax.sql.DataSource	com.xugu.pool.XgDataSource
javax.sql.ConnectionPoolDataSource	com.xugu.pool.XgConnectionPoolDataSource
javx.sql.PooledConnection	com.xugu.pool.XgPooledConnection

除了以上标准类接口外，虚谷 JDBC 还扩展实现了以下类：

- com.xugu.cloudjdbc.ClientInfo
- com.xugu.cloudjdbc.OutParameter
- com.xugu.cloudjdbc.Rowid
- com.xugu.cloudjdbc.SQLXml
- com.xugu.cloudjdbc.SSL

- com.xugu.pool.StatementWrapper
- com.xugu.pool.PreparedStatementWrapper

8 虚谷数据库负载均衡

虚谷数据库是一款具备高安全、高性能、高扩展等特性的、基于分布式架构的分布式事务型数据库。部署数据库集群时，多台服务器作为一个整体对外提供数据服务。在多机环境下，为使数据库连接均匀分布于各台服务，虚谷 JDBC 在接口层实现了基于软负载均衡技术的数据库连接建立方式。

虚谷数据库负载均衡分配数据库连接有两种方式：顺序分配、随机分配。通过在数据库 URL 连接串上设置参数 conn_type(conn_type:1, 随机分配；2, 顺序分配) 类型，实现数据库连接在多集群环境下的负载均衡。默认负载均衡分配方式：顺序分配。

虚谷 JDBC 提供三种配置负载均衡的方式：

- 数据库 URL 连接串配置

```
Class.forName("com.xugu.cloudjdbc.Driver");
//URL连接串中使用ips属性配置多IP负载均衡
Connection conn = DriverManager.getConnection("jdbc:xugu
   ://192.168.0.201:5138/SYSTEM?user=SYSDBA\&password=SYSDBA\&
   conn\_type=1\&ips=192.168.0.205,192.168.0.204,192.168.1.206
   ")
```

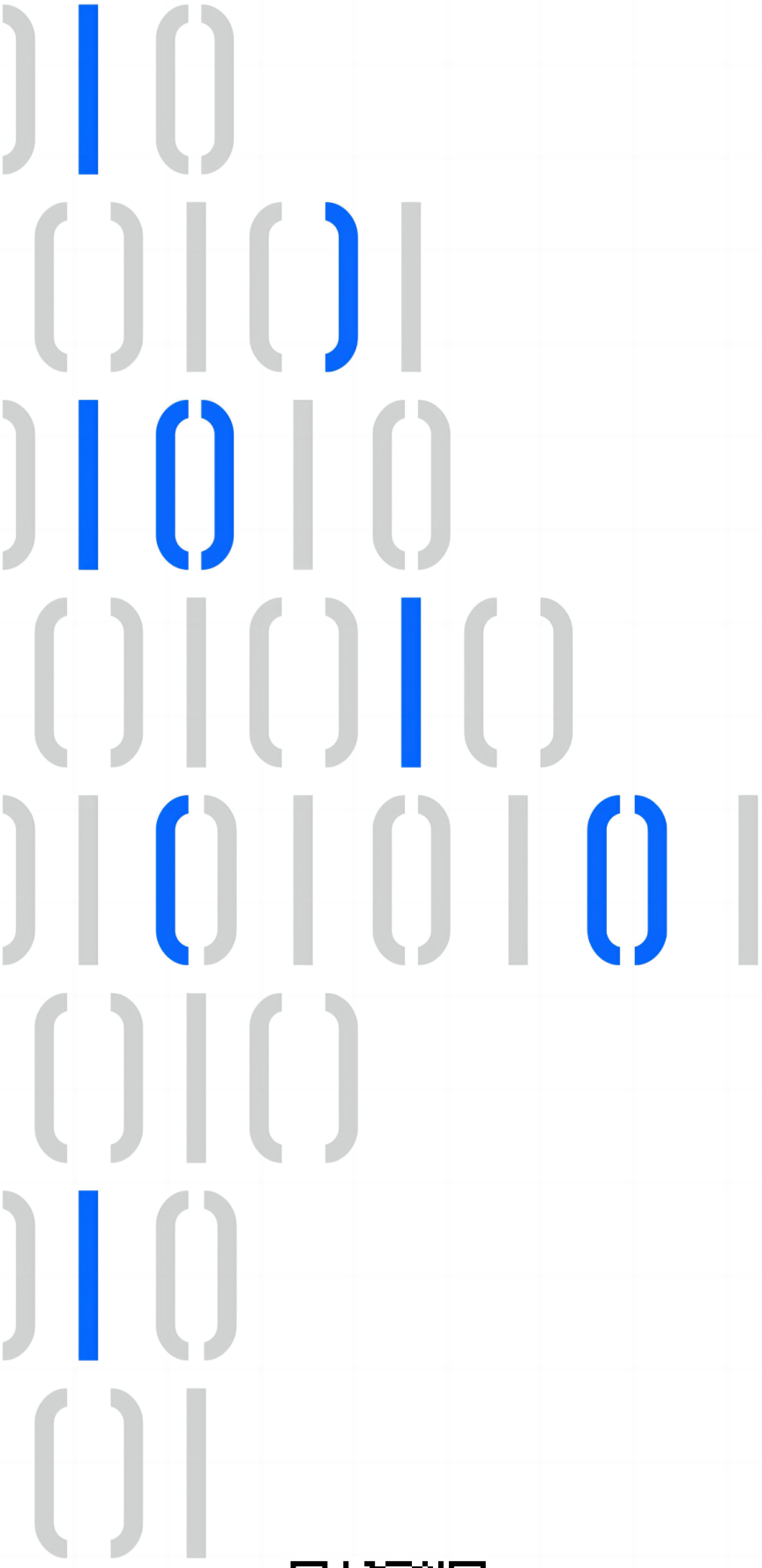
- 数据库连接属性文件 Properties 配置

```
Properties info=new Properties();
Class.forName("com.xugu.cloudjdbc.Driver");
//Vector结构存放负载均衡多IP地址方式
Vector<String> ipsVector=new Vector<String>();
ipsVector.add("192.168.1.201");
ipsVector.add("192.168.1.204");
ipsVector.add("192.168.1.205");
ipsVector.add("192.168.1.206");
info.put("ips", ipsVector);
//数组存放负载均衡多IP地址方式
String[] ips={"192.168.1.205","192.168.1.204","192.168.1.206"
   };
info.put("ips", ips);
Connection conn = DriverManager.getConnection("jdbc:xugu
   ://192.168.1.201:5138/SYSTEM?user=SYSDBA\&password=SYSDBA\&
   conn\_type=2",info);
```

- 数据库连接 XML 文件配置

```
 <?xml version="1.0" encoding="utf-8"?>
 <listeners>
 <listener>
 <ip>192.168.1.206</ip>
 </listener>
```

```
      <listener>
      <ip>192.168.1.204</ip>
      </listener>
      <listener>
      <ip>192.168.1.205</ip>
      </listener>
      </listeners>
// XML（可扩展标记语言）代码，以<listeners>作为根元素，表明这是一
   个关于 “监听器” 相关信息的集合
      Class.forName("com.xugu.cloudjdbc.Driver");
      Connection conn = DriverManager.getConnection("jdbc:xugu:file
        ://xuguClouldListener.xml:5138/SYSTEM?user=SYSDBA\&password
        =SYSDBA\&conn\_type=1");
//  数据库连接获取语句
```



成都虚谷伟业科技有限公司

联系电话：400-8886236

官方网站：www.xugudb.com