

优炫数据库JDBC驱动手册 2.1



UXSINO
优炫软件

优炫数据库JDBC驱动手册 2.1

版权 © 2016-2023 北京优炫软件股份有限公司

法律声明

优炫数据库管理系统(简称: UXDB) 是由北京优炫软件股份有限公司开发并发布的一款商业性数据库管理系统。

优炫数据库管理系统 (UXDB) 的一切知识产权以及与该软件产品相关的所有信息内容, 包括但不限于: 文字表述及其组合、图标、图饰、图表、色彩、界面设计、版面框架、有关数据、及电子文档等均属北京优炫软件股份有限公司所有。本软件及其文档的任何使用、复制、修改、出租、传播、销售及分发等行为均须经北京优炫软件股份有限公司书面许可。

凡侵犯北京优炫软件股份有限公司知识产权的行为, 北京优炫软件股份有限公司将依法追究其法律责任。

本声明的最终解释权归属于北京优炫软件股份有限公司。



和其他优炫公司商标均为北京优炫软件股份有限公司的商标。

本文档提及的其他所有商标或注册商标, 由各自的所有人拥有。

注意

由于产品版本安装或其他原因, 本文档内容会不定期进行更新。除非另有约定, 本文档仅作为使用指导, 本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

北京优炫软件股份有限公司 (总部)

- 地址: 北京市海淀区学院南路62号中关村资本大厦11层 (邮编: 100081)
 - 网址: <http://www.uxsino.com>
 - 邮箱: <uxdb_support@uxsino.com>
 - 电话: 010-82886998
 - 传真: 010-82886338
 - 服务热线: 400-650-7837
-

目录

前言	vii
1. 文档目的	vii
2. 文档对象	vii
3. 修改记录	vii
1. 系统概述	1
2. JDBC交互流程	3
3. 接口说明	7
3.1. 接口简述	7
3.1.1. 驱动连接相关类	7
3.1.2. 执行SQL相关类	7
3.1.3. SQL类型到JAVA类型映射相关类	7
3.1.4. 元数据相关类	7
3.2. 接口详解	8
3.2.1. Class CopyManager	8
3.2.2. Class Date	9
3.2.3. Class DriverAction	9
3.2.4. Class UxBlob	10
3.2.5. Class UXCalleableStatement	10
3.2.6. Class UXClob	11
3.2.7. Class UxConnection	12
3.2.8. Class UxConnectionPoolDataSource	25
3.2.9. Class UxDatabaseMetadata	25
3.2.10. Class UxParameterMetaData	28
3.2.11. Class UXPooledConnection	29
3.2.12. Class UXPreparedStatement	30
3.2.13. Class UxResultSet	36
3.2.14. Class UxResultSetMetaData	46
3.2.15. Class UXSQLException	47
3.2.16. Class UxStatement	47
3.2.17. Class UXTime	52
3.2.18. Class UXTimeStamp	53
3.2.19. Class Wrapper	53
4. 使用JDBC	55
4.1. 设置JDBC驱动	55
4.2. 初始化驱动	55
4.3. 发出查询和处理结果	57
4.3.1. 基于一个游标获取结果	57
4.3.2. 使用Statement或UXPreparedStatement接口	58
4.3.3. 使用ResultSet接口	58
4.3.4. 执行更新	58
4.4. 调用存储过程	59
4.5. 创建和更改数据库对象	60
4.6. 存储二进制数据	60
4.7. 多线程或服务小应用环境里使用驱动	62
5. JDBC API扩展	64
5.1. 访问扩展	64
5.1.1. 类org.UXDB.UXConnection	64
5.1.2. 类org.UXDB.Fastpath	66
5.1.3. 类org.UXDB.fastpath.FastpathArg	68
5.2. 几何数据类型	69
5.2.1. Class org.UXDB.geometric.UXbox	69

5.2.2. Class org.UXDB.geometric.UXcircle	71
5.2.3. Class org.UXDB.geometric.UXline	72
5.2.4. Class org.UXDB.geometric.UXlseg	73
5.2.5. Class org.UXDB.geometric.UXpath	75
5.2.6. Class org.UXDB.geometric.UXpoint	76
5.2.7. Class org.UXDB.geometric.UXpolygon	79
5.3. 大对象	80
5.3.1. 类org.UXDB.largeobject.LargeObject	81
5.3.2. 类org.UXDB.largeobject.LargeObjectManager	82
6. 连接池和数据源	84
6.1. 概述	84
6.2. ConnectionPoolDataSource	84
6.3. DataSource	85
6.4. 数据源和JNDI	86
7. JDBC Wrapper	88
7.1. 获取jdbc wrapper	88
7.2. 接口使用	88
7.2.1. 搭建mpp环境	88
7.2.2. 多CN轮询接口用例（mpp场景）	88
7.2.3. worker接口用例（mpp场景）	90
7.2.4. 普通jdbc连接池用例（非mpp场景）	91
7.3. 使用benchmark工具	93
7.3.1. 搭建mpp环境	93
7.3.2. 相关配置文件	93
7.3.3. 原有工具包替换	95
7.3.4. 查看结果	95
8. 高可用连接池	97
8.1. 概述	97
8.2. 用法示例	98
8.2.1. 连接池使用示例	98
8.2.2. 禁用连接池使用示例	98
9. 大对象适配	99
9.1. 概述	99
9.2. 用法示例	99
10. boolean值适配	102
10.1. 概述	102
10.2. 用法示例	102
11. jdbcurl数据库缺省值适配	104

表格清单

1. 文档更新记录	vii
3.1. 驱动连接相关类	7
3.2. 执行SQL相关类	7
3.3. 映射相关类	7
3.4. 元数据相关类	7
3.5. copyIn 参数说明	8
3.6. copyIn 参数说明	8
3.7. copyIn 参数说明	9
3.8. getBytes 参数说明	10
3.9. getString 参数说明	11
3.10. setBlob 参数说明	11
3.11. createStatement参数说明	13
3.12. createStatement-2参数说明	13
3.13. prepareStatement-1 参数说明	15
3.14. prepareStatement-2 参数说明	15
3.15. prepareStatement-3 参数说明	16
3.16. prepareStatement 参数说明	17
3.17. prepareStatement-4 参数说明	18
3.18. getConnection 参数说明	19
3.19. getConnection 参数说明	20
3.20. prepareCall 参数说明	23
3.21. prepareCall 参数说明	23
3.22. prepareCall 参数说明	24
3.23. isValid 参数说明	25
3.24. getPooledConnection 参数说明	25
3.25. getBestRowIdentifier 参数说明	27
3.26. getTables 参数说明	28
3.27. getParameterTypeName 参数说明	29
3.28. getParameterClassName 参数说明	29
3.29. setInt参数说明	32
3.30. setString 参数说明	32
3.31. setTimestamp 参数说明	34
3.32. setBlob 参数说明	34
3.33. setTime 参数说明	35
3.34. setTimestamp 参数说明	35
3.35. setDate 参数说明	35
3.36. getTimestamp 参数说明	39
3.37. getTimestamp参数说明	40
3.38. getBigDecimal 参数说明	41
3.39. getBoolean 参数说明	42
3.40. getDate 参数说明	42
3.41. getDouble 参数说明	43
3.42. getFloat 参数说明	43
3.43. getLong 参数说明	43
3.44. getShort 参数说明	44
3.45. updateNull 参数说明	45
3.46. updateString 参数说明	46
3.47. getColumnName 参数说明	46
3.48. getColumnType 参数说明	47
3.49. executeUpdate参数说明	50
3.50. executeUpdate 参数说明	51

3.51.	executeUpdate 参数说明	51
3.52.	addBatch 参数说明	52
3.53.	isWrapperFor 参数说明	54
4.1.	数据库连接参数	56
5.1.	addDataType参数说明	65
5.2.	fastpath-fnid参数说明	66
5.3.	fastpath-name参数说明	67
5.4.	getInteger参数说明	67
5.5.	getData参数说明	67
5.6.	FastpathArg参数说明	69
5.7.	UXbox-double参数说明	70
5.8.	UXbox-UXpoint参数说明	70
5.9.	UXcircle-double参数说明	71
5.10.	UXline-double参数说明	72
5.11.	UXline-UXpoint参数说明	72
5.12.	UXlseg-double参数说明	74
5.13.	UXlseg-UXpoint参数说明	74
5.14.	UXpath-UXpoint参数说明	75
5.15.	UXpoint-double参数说明	77
5.16.	translate-int参数说明	78
5.17.	translate-double参数说明	78
5.18.	move-int参数说明	78
5.19.	move-double参数说明	79
5.20.	setLocation-int参数说明	79
5.21.	read参数说明	81
5.22.	write参数说明	82
6.1.	ConnectionPoolDataSource实现	84
6.2.	ConnectionPoolDataSource 配置属性	84
6.3.	DataSource实现	85
6.4.	DataSource配置属性	85
6.5.	额外的连接池DataSource配置属性	85
7.1.	服务器及节点分布	93
8.1.	jdbc配置参数	97

前言

1. 文档目的

JDBC (Java Database Connectivity) (文档版本号: V1.1) 是一个通用的关系型数据库操作的接口规范, 它是 JAVA 访问数据库的一种标准。各数据库厂商根据自己的数据库特点提供对应的接口实现驱动程序。

2. 文档对象

- 技术支持工程师
- 维护工程师

3. 修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

表 1. 文档更新记录

工具版本	发布日期	修改说明
2.1.1.5C	2022-09-21	第一次正式发布。

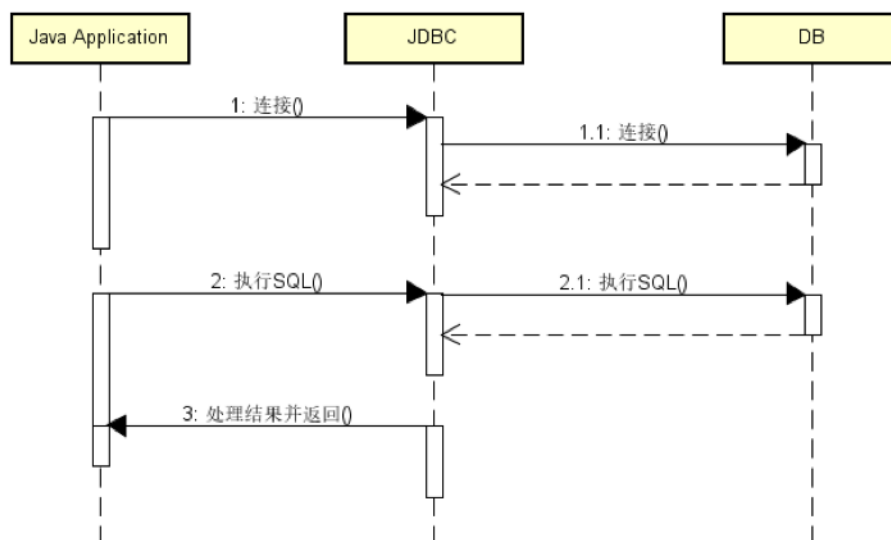
第 1 章 系统概述

JDBC用于JAVA应用程序与数据库的连接访问，是应用程序与数据库的中间层。

JDBC需要和数据库建立连接，接收JAVA应用程序的请求，并对请求进行加工处理，再转发请求操作给UXDB。同时，JDBC需要为JAVA应用程序服务，需要将对UXDB结果交付到应用程序中，所以JDBC还需要对返回的数据进行处理。

JDBC连接数据库进行查询大致包含三个步骤：

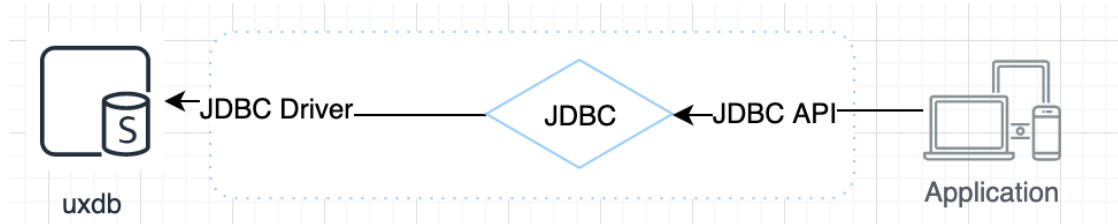
1. 连接数据库。
2. 执行SQL。
3. 处理返回结果。

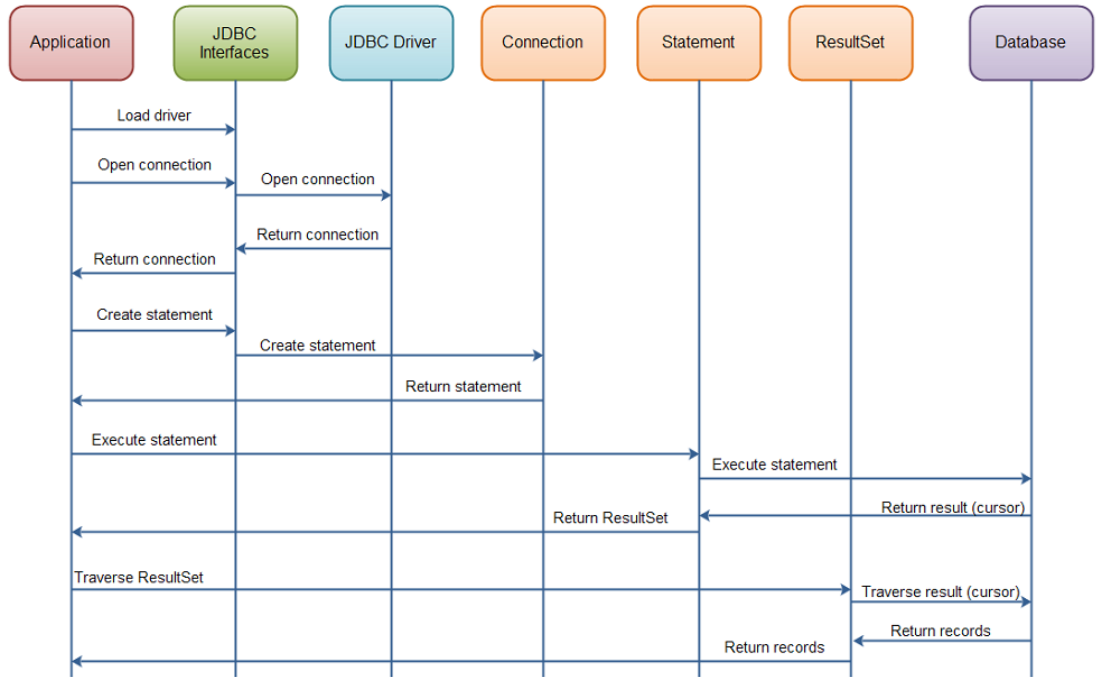


JDBC主要包括两类接口：

- 提供了JAVA API给应用程序开发者。
- 提供了JDBC driver API给数据库驱动开发者。

应用程序开发者借助于API用于开发可以访问数据库的程序。





DriverManager类，实际是在rt.jar包中，而其源码在openjdk中，对应DriverManager.java。

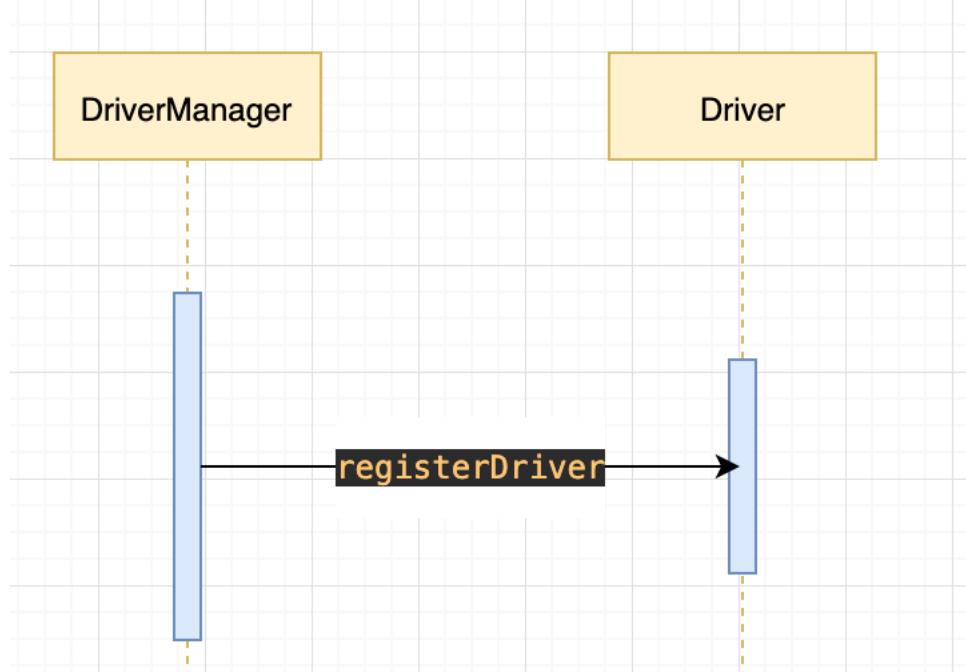
第 2 章 JDBC交互流程

1. 注册加载驱动器

应用程序加载驱动器，如下所示。

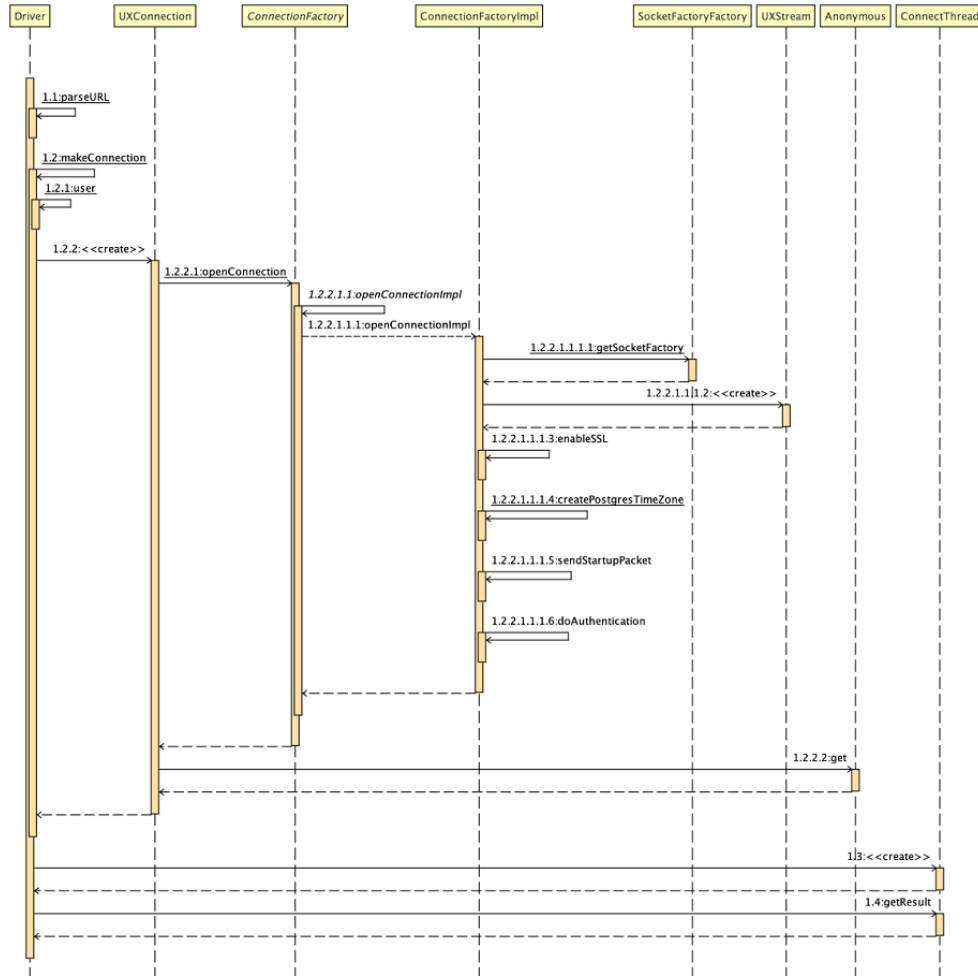
```
Class.forName("com.uxsino.uxdb.Driver")
```

加载驱动器的过程就是将UXDB的Driver注册进Driver管理模块，供后面的类使用。



2. 获取连接

获取连接的过程中最终会通过com.uxsino.uxdb.Driver的connect()为入口，沿着下图中链路创建connection，创建Connection的过程包含socket的连接创建和建立，最终创建出可用的连接。



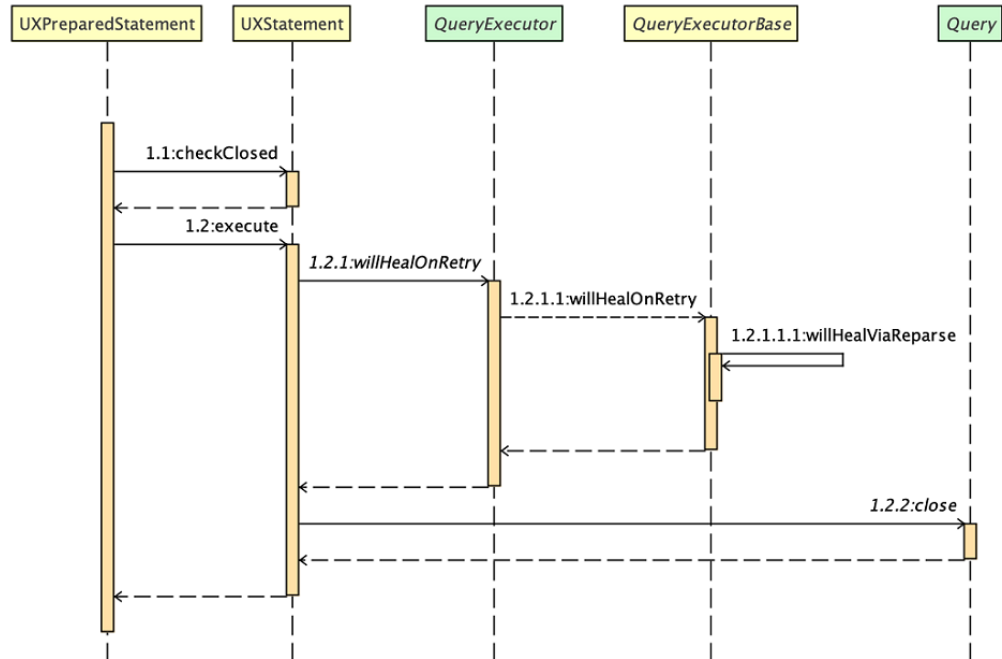
- 实际上获取连接有多个接口，其入参是不同的，通常会使用 `Drivermanager.getConnection()`。
- `connect` 函数先通过 `parseURL()` 解析 URL 中的信息，并保存在 `Properties` 中，该对象用于保存一些配置信息。
- `makeConnection` 创建连接对象。
- 打开 `Connection`，建立 `socket` 连接，发送 `startupPacket`。

同时，还要考虑连接的关闭，在 JDBC 程序结束之后，显式地需要关闭与数据库的所有连接以结束每个数据库会话。

如果在编写程序中忘记了关闭也没有关系，Java 的垃圾收集器在清除过时的对象时也会关闭这些连接。使用 JVM 的垃圾收集，特别是在数据库编程中使用，是一个非常差的编程实践。因此应该要使用与连接对象关联的 `close()` 方法关闭连接。要确保连接已关闭，可以将关闭连接的代码中编写在“finally”块中。不管是否发生异常，一个 finally 块总是会被执行。关闭 JDBC 连接，并释放连接对象。

3. 执行 SQL

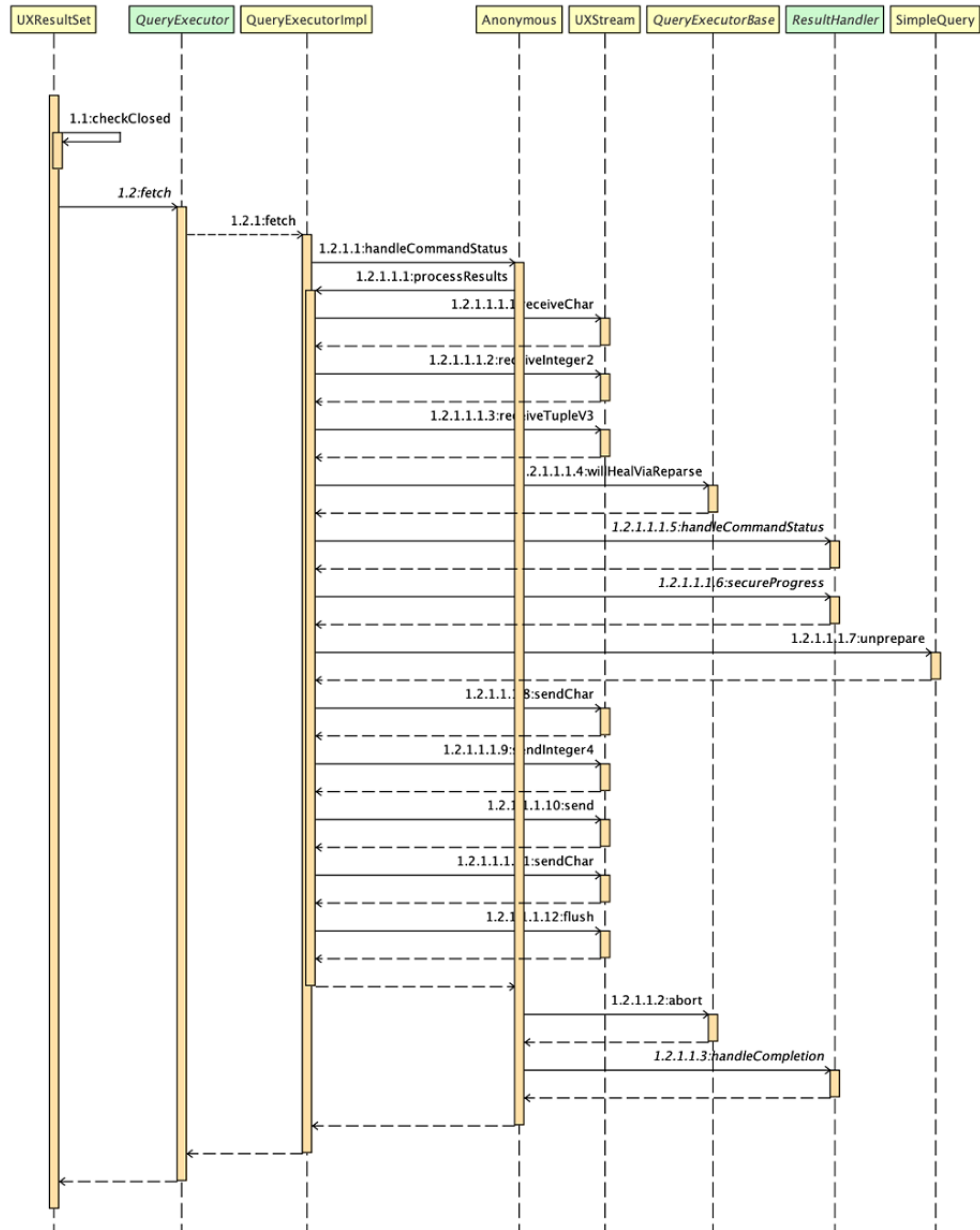
`Statement` 用于执行静态 SQL 语句并返回生成结果的对象。



整个执行sql执行过程如上图所示，围绕着Statement进行。

4. SQL执行结果

为了更好的获取执行结果，可以通过`UxResultSet`类的`next()`方法来获取，整个执行过程如下图所示：



第 3 章 接口说明

3.1. 接口简述

3.1.1. 驱动连接相关类

表 3.1. 驱动连接相关类

类名	类描述
DriverManager	与驱动程序建立连接
Driver	提供基于JDBC技术的驱动程序注册和连接API，通常只被DriverManager类使用
DriverPropertyInfo	为JDBC驱动程序提供属性；用户一般不使用

3.1.2. 执行SQL相关类

表 3.2. 执行SQL相关类

类名	类描述
UxStatement	执行对象，用于发送基本的SQL语句
UXPreparedStatement	用于发送准备好的语句或基本SQL语句
UXCallableStatement	用于调用数据库存储过程
UxConnection	提供用于创建语句和管理连接及其属性的方法
UXSQLSavepoint	在事务中提供Savepoint保存点
UxResultSet	返回结果

3.1.3. SQL类型到JAVA类型映射相关类

表 3.3. 映射相关类

类名	类描述
UxArray	Array类型
UXBlob	Blob类型
UXClob	Clob类型
UxSQLXML	SQL XML

3.1.4. 元数据相关类

表 3.4. 元数据相关类

类名	类描述
UxDatabaseMetaData	提供有关数据库的信息
UxResultSetMetaData	提供有关ResultSet对象的列的信息

类名	类描述
UxParameterMetaData	为UXPreparedStatement命令提供有关参数的信息

3.2. 接口详解

3.2.1. Class CopyManager

`copyIn(final String sql, Reader from) throws SQLException, IOException`

- 功能描述

使用COPY FROM STDIN可以非常快速地从Reader复制到数据库表中。

- 参数

表 3.5. copyIn 参数说明

名称	备注
sql	COPY FROM STDIN语句
from	一个CSV文件或类似文件

- 返回值

更新的行数。

- 异常

SQLException: 如果发生数据库访问错误, 则抛出该异常。

IOException: 在读取器或数据库连接失败时, 则抛出该异常。

`copyIn(final String sql, Reader from, int bufferSize) throws SQLException, IOException`

- 功能描述

使用COPY FROM STDIN可以非常快速地从Reader复制到数据库表中。

- 参数

表 3.6. copyIn 参数说明

名称	备注
sql	COPY FROM STDIN 语句
from	一个CSV文件或类似文件
bufferSize	要缓冲并一次性通过网络推送到服务器的字符数

- 返回值

更新的行数。

- 异常

SQLException: 如果发生数据库访问错误，则抛出该异常。

IOException: 在读取器或数据库连接失败时，则抛出该异常。

`copyIn(final String sql, BufferedWriter from)` throws `SQLException`, `IOException`

- 功能描述

使用COPY FROM STDIN可以非常快速地从Reader复制到数据库表中。

- 参数

表 3.7. copyIn 参数说明

名称	备注
sql	COPY FROM STDIN 语句
from	字节的来源，例如 ByteBufferByteStreamWriter

- 返回值

更新的行数。

- 异常

SQLException: 如果发生数据库访问错误，则抛出该异常。

IOException: 在读取器或数据库连接失败时，则抛出该异常。

3.2.2. Class Date

`toString()`

- 功能描述

以日期转义格式yyyy-mm-dd格式化日期。

- 返回值

一个yyyy-mm-dd格式的字符串。

3.2.3. Class DriverAction

`deregister()`

- 功能描述

方法调用DriverManager.deregisterDriver(Driver)通知JDBC驱动程序它已被注销。

注意

deregister方法仅供JDBC驱动程序使用，而不是由应用程序使用。推荐JDBC驱动程序在公共类中不实现DriverAction。如果在调用deregister方

法时存在与数据库的活动连接，那么具体是关于连接是关闭还是允许继续的实现。一旦这个方法被调用，它是实现特定于驱动程序是否可能限制创建到数据库的新连接的能力，调用其他Driver方法或者抛出SQLException。

3.2.4. Class UXBlob

length() throws SQLException

- 功能描述

返回由此Blob对象指定的Blob值中的字节数。

- 返回值

Blob的字节长度。

- 异常

SQLException: 如果访问Blob的长度时出现错误，则抛出该异常。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

getBytes(long pos, int length) throws SQLException

- 功能描述

以字节数组的形式检索此Blob对象表示的全部或部分Blob值。此字节数组包含从位置pos开始的最多个长度连续的字节。

- 参数

表 3.8. getBytes 参数说明

名称	备注
pos	要提取的Blob值中第一个字节的顺序位置；第一个字节在位置1
length	要复制的连续字节数；长度的值必须为0或更大

- 返回值

一个字节数组，包含从该Blob对象指定的Blob值开始的最长连续字节，从位置pos开始。

- 异常

SQLException: 如果访问Blob值时发生错误，如果pos小于1，则抛出该异常。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

3.2.5. Class UXCalleableStatement

getString(int parameterIndex) throws SQLException

- 功能描述

获取指定参数列的CHAR, VARCHAR或LONGVARCHAR值作为JAVA编程语言的String。

对于固定长度类型的JDBC CHAR, 返回的String对象与SQL CHAR值在数据库中具有完全相同的值, 包括数据库添加的任何填充。

- 参数

表 3.9. getString 参数说明

名称	备注
parameterIndex	第一个参数是1, 第二个是2, 以此类推

- 返回值

参数值。如果SQL中没有这个参数, 则返回NULL。

- 异常

SQLException: 如果发生数据库访问错误或者在UCallableStatement上调用此方法时parameterIndex无效, 则抛出该异常。

setBlob(String parameterName, Blob x) throws SQLException

- 功能描述

将指定的参数设置为给定的java.sql.Blob对象。当驱动程序将其发送到数据库时, 将其转换为SQL Blob值。

- 参数

表 3.10. setBlob 参数说明

名称	备注
parameterName	参数的名称
x	映射SQL Blob值的 Blob对象

- 异常

SQLException: 如果发生数据库访问错误或这在UCallableStatement上调用此方法时parameterName与命名参数不对应, 则抛出该异常。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

3.2.6. Class UXClob

length() throws SQLException

- 功能描述

检索由Clob对象指定的Clob值中的字符数。

- 返回值

Clob的字符长度。

- 异常

SQLException: 如果访问Clob值的长度时出现错误，则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法。

3.2.7. Class UxConnection

close() throws SQLException

- 功能描述

释放UxConnection对象的数据库和JDBC资源，而不是等待它们自动释放。

- 异常

SQLException: 如果发生数据库访问错误，则抛出SQLException。

注意

在已关闭的UxConnection对象上调用方法close是无操作的。

强烈建议在调用close方法之前，应用程序显式地提交或回滚活动事务。
如果调用了close方法，并且存在活动事务，则结果是实现定义的。

commit() throws SQLException

- 功能描述

使上次提交/回滚之后所做的所有更改都将永久性，并释放此UxConnection对象当前持有的任何数据库锁。

只有当自动提交模式被禁用时，才应该使用此方法。

- 异常

SQLException: 如果在关闭的连接上调用此方法，或者UxConnection对象处于自动提交模式，那么这个异常会被抛出；如果发生了数据库访问错误，那么此方法会在参与分布式事务时被调用。

createStatement() throws SQLException

- 功能描述

创建一个Statement对象，用于将SQL语句发送到数据库。没有参数的SQL语句通常使用Statement对象执行。如果相同的SQL语句执行了很多次，那么使用UXPreparedStatement对象可能会更有效。

使用返回的Statement对象创建的结果集将默认为TYPE_FORWARD_ONLY，并发级别为CONCUR_READ_ONLY。创建的结果集的可保持性可以通过调用getHoldability()来确定。

- 返回值

一个新的Statement对象。

- 异常

SQLException: 如果发生数据库访问错误或遇到关闭连接, 则抛出该异常。

createStatement(int resultSetType, int resultSetConcurrency) throws SQLException

- 功能描述

创建一个Statement对象, 该对象将生成具有给定类型和并发性的ResultSet对象。此方法与createStatement()相同, 但它允许覆盖默认的结果集类型和并发性。创建的结果集的可保持性可以通过调用getHoldability()来确定。

- 参数

表 3.11. createStatement参数说明

名称	描述	可选值
resultSetType	结果集类型	ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.TYPE_SCROLL_SENSITIVE
resultSetConcurrency	并发类型	ResultSet.CONCUR_READ_ONLY, ResultSet.CONCUR_UPDATABLE

- 返回值

将生成具有给定类型和并发的ResultSet对象的新Statement对象。

- 异常

SQLException: 如果发生数据库访问错误, 或在关闭连接上抛出该异常, 或者给定的参数不是结果集类型和并发性的ResultSet常量, 则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法, 或者该方法不支持指定的结果集类型和并发类型。

createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)

- 功能描述

创建一个Statement对象, 该对象将生成具有给定类型和并发性的ResultSet对象。此方法与createStatement()相同, 但它允许覆盖默认的结果集类型和并发性。

- 参数

表 3.12. createStatement-2参数说明

名称	可选值
resultSetType	ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.TYPE_SCROLL_SENSITIVE
resultSetConcurrency	ResultSet.CONCUR_READ_ONLY, ResultSet.CONCUR_UPDATABLE

名称	可选值
resultSetHoldability	ResultSet.HOLD_CURSORS_OVER _COMMIT, ResultSet.CLOSE_CURSORS_AT_COMM IT

- 返回值

一个新的Statement对象，将产生ResultSet对象具有给定类型，并发性和可保存。

- 异常

SQLException: 如果发生数据库访问错误，或者给定的参数不是指示类型和并发性的ResultSet常量，则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法，或者对于指定的结果集类型、结果集保持性和结果集并发性不支持此方法。

isClosed() throws SQLException

- 功能描述

检索此UxConnection对象是否已关闭。如果已经调用了方法close或者发生了某些致命错误，则连接被关闭。这种方法只有在方法UxConnection.close被调用之后被调用才能保证返回true。

通常无法调用此方法来确定与数据库的连接是有效还是无效。典型的客户端可以通过捕获在尝试操作时可能引发的任何异常来确定连接无效。

- 返回值

如果此Connection对象是关闭的，则返回true；如果它仍然处于打开状态，则返回false。

- 异常

SQLException: 如果发生数据库访问错误，则抛出该异常。

prepareStatement(String sql) throws SQLException

- 功能描述

创建一个UXPreparedStatement对象，用于将参数化的SQL语句发送到数据库。

带有IN参数或不带有IN参数的SQL语句都可以被预编译并存储在UXPreparedStatement对象中。可以使用该对象多次有效地执行此语句。

使用返回的UXPreparedStatement对象创建的结果集将默认为TYPE_FORWARD_ONLY类型，并发级别为CONCUR_READ_ONLY。创建的结果集的可保持性可以通过调用getHoldability()来确定。

注意

为了处理受益于预编译的带参数SQL语句，此方法进行了优化。

如果驱动程序支持预编译，则方法prepareStatement将将该语句发送到数据库进行预编译。

一些驱动程序可能不支持预编译。在这种情况下，执行UXPreparedStatement对象之前无法将语句发送给数据库。这对用户没有直接的影响；但它的确会影响抛出某些SQLException对象的方法。

使用返回的UXPreparedStatement对象创建的结果集在默认情况下类型为TYPE_FORWARD_ONLY，并带有CONCUR_READ_ONLY并发级别。

- 参数

表 3.13. preparedStatement-1 参数说明

名称	可选值
sql	可能包含一个或多个'?' IN参数占位符的SQL语句

- 返回值

一个新的默认的UXPreparedStatement对象包含预编译的SQL语句。

- 异常

SQLException: 如果发生数据库访问错误或者关闭连接，则抛出该异常。

preparedStatement(String sql, int autoGeneratedKeys) throws SQLException

- 功能描述

创建一个默认的UXPreparedStatement对象，该对象具有检索自动生成的键的能力。给定的常量告诉驱动程序是否应该使自动生成的密钥可用于检索。如果SQL语句不是INSERT语句，或者SQL语句能够返回自动生成的键，则忽略此参数。

注意

为了处理受益于预编译的带参数SQL语句，此方法进行了优化。

如果驱动程序支持预编译，方法preparedStatement将会将该语句发送到数据库进行预编译。

有些驱动程序可能不支持预编译。在这种情况下，执行UXPreparedStatement对象之前，可能不会将语句发送到数据库。这对用户没有直接影响；但它的确会影响抛出某些SQLException对象的方法。

使用返回的UXPreparedStatement对象创建的结果集将默认为TYPE_FORWARD_ONLY类型，并发级别为CONCUR_READ_ONLY。创建的结果集的可保持性可以通过调用getHoldability()来确定。

- 参数

表 3.14. preparedStatement-2 参数说明

名称	可选值
sql	可能包含一个或多个'?' IN参数占位符的SQL语句
autoGeneratedKeys	一个标志，指示是否应该返回自动生成的键；其中一个是 Statement.RETURN_GENERATED_KEYS或 Statement.NO_GENERATED_KEYS

- 返回值

一个新的UXPreparedStatement对象，其中包含预编译的SQL语句，该对象具有返回自动生成键的能力。

- 异常

SQLException: 如果发生数据库访问错误，则抛出该异常。或者给定参数不是Statement常量，提示是否应返回自动生成的密钥。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法与常量Statement.RETURN_GENERATED_KEYS。

prepareStatement(String sql, int[] columnIndexes) throws SQLException

- 功能描述

创建默认UXPreparedStatement对象，该对象能够返回由给定数组指定的自动生成的键。这个数组包含目标表中列的索引，这些列包含应该可用的自动生成的键。如果SQL语句不是INSERT语句，或者不是能够返回自动生成的键的SQL语句(这些语句的列表是特定于供应商的)，则驱动程序将忽略数组。

可以预编译包含或不包含IN参数的SQL语句，并将其存储在UXPreparedStatement对象中。然后可以使用此对象高效地多次执行此语句。

注意

为了处理受益于预编译的带参数SQL语句，此方法进行了优化。

如果驱动程序支持预编译，方法prepareStatement将会将该语句发送到数据库进行预编译。

有些驱动程序可能不支持预编译。在这种情况下，执行UXPreparedStatement对象之前，可能不会将语句发送到数据库。这对用户没有直接影响；但它的确会影响抛出某些SQLException对象的方法。

使用返回的UXPreparedStatement对象创建的结果集将默认为TYPE_FORWARD_ONLY类型，并发级别为CONCUR_READ_ONLY。创建的结果集的可保持性可以通过调用getHoldability()来确定。

- 参数

表 3.15. prepareStatement-3 参数说明

名称	可选值
sql	一个String对象，即要发送到数据库的SQL语句；可能包含一个或多个'?' IN参数占位符的SQL语句
columnIndexes	列索引，指示应从插入的行或行中返回的列

- 返回值

一个新的UXPreparedStatement对象，包含预编译语句，它能够返回由给定的列索引数组指定的自动生成的键。

- 异常

SQLException: 如果发生数据库访问错误或在已关闭的连接上调用此方法，则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法。

prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException

- 功能描述

创建一个UXPreparedStatement对象，将产生ResultSet对象具有给定类型，并发性和可保持性。

此方法与上述prepareStatement方法相同，但允许覆盖默认结果集类型，并发性和可保持性。

注意

为了处理受益于预编译的带参数SQL语句，此方法进行了优化。

如果驱动程序支持预编译，方法prepareStatement将会将该语句发送到数据库进行预编译。

有些驱动程序可能不支持预编译。在这种情况下，执行UXPreparedStatement对象之前，可能不会将语句发送到数据库。这对用户没有直接影响；但它的确会影响抛出某些SQLException对象的方法。

使用返回的UXPreparedStatement对象创建的结果集将默认为TYPE_FORWARD_ONLY类型，并发级别为CONCUR_READ_ONLY。创建的结果集的可保持性可以通过调用getHoldability()来确定。

- 参数

表 3.16. prepareStatement 参数说明

名称	可选值
sql	一个String对象，即要发送到数据库的SQL语句；可能包含一个或多个'?' IN参数占位符的SQL语句
resultSetType	ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.TYPE_SCROLL_SENSITIVE
resultSetConcurrency	ResultSet.CONCUR_READ_ONLY, ResultSet.CONCUR_UPDATABLE
resultSetHoldability	ResultSet.HOLD_CURSORS_OVER _COMMIT, ResultSet.CLOSE_CURSORS_AT_COMMIT

- 返回值

一个新的UXPreparedStatement对象，包含预编译的SQL语句，它将生成具有给定类型、并发性和可保持性的ResultSet对象。

- 异常

SQLException: 如果发生数据库访问错误,或在封闭连接上调用此方法,则抛出该异常。或者给定参数不是ResultSet常数指示类型,则也抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法,或者对于指定的结果集类型、结果集保持性和结果集并发性不支持此方法。

prepareStatement(String sql, String[] columnNames) throws SQLException

- 功能描述

创建一个默认的UXPreparedStatement对象,能够返回给定数组指定的自动生成的键。

该数组包含目标表中包含应该返回的自动生成的键的列的名称。如果SQL语句不是INSERT语句,或者SQL语句能够返回自动生成的键,驱动程序将忽略该数组。

具有或不具有IN参数的SQL语句可以预编译并存储在UXPreparedStatement对象中。然后可以使用该对象多次有效地执行此语句。

注意

为了处理受益于预编译的带参数SQL语句,此方法进行了优化。

如果驱动程序支持预编译,方法prepareStatement将会将该语句发送到数据库进行预编译。

有些驱动程序可能不支持预编译。在这种情况下,执行UXPreparedStatement对象之前,可能不会将语句发送到数据库。这对用户没有直接影响;但它的确会影响抛出某些SQLException对象的方法。

使用返回的UXPreparedStatement对象创建的结果集将默认为TYPE_FORWARD_ONLY类型,并发级别为CONCUR_READ_ONLY。创建的结果集的可保持性可以通过调用getHoldability()来确定。

- 参数

表 3.17. prepareStatement-4 参数说明

名称	备注
sql	一个String对象,即要发送到数据库的SQL语句;可能包含一个或多个'?' IN参数占位符的SQL语句
columnNames	列名称,表示应该从插入的行或行中返回的列

- 返回值

新UXPreparedStatement对象,包含预编译的SQL语句,能够返回由给定的列名数组指定的自动生成的键。

- 异常

SQLException: 如果发生数据库访问错误或在关闭的连接上调用此方法,则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法。

rollback() throws SQLException

- 功能描述

回滚在当前事务下做出的所有修改并且释放所有该连接上的数据库锁。此方法仅应在自动提交模式被禁用时使用。

- 异常

如果此方法在分布式数据库下调用，在一个关闭的连接下调用以及在自动提交模式下调用都会抛出SQLException异常。

setAutoCommit(boolean autoCommit) throws SQLException

- 功能描述

将此连接的自动提交模式设置为给定状态。如果连接处于自动提交模式下，则将执行其所有SQL语句，并将这些语句作为单独的事务提交。否则，其SQL语句将成组地进入通过调用commit方法或rollback中。

- 提交发生在语句完成时。语句完成的时间取决于SQL语句的类型：

1. 对于DML语句（如插入，更新或删除）和DDL语句，语句在执行完成后立即完成。
2. 对于Select语句，当关联的结果集关闭时，该语句将完成。
3. 对于CallableStatement对象或返回多个结果的语句，当所有关联的结果集都已关闭并且已检索到所有更新计数和输出参数时，该语句将完成。

- 参数

autoCommit: true表示启用自动提交模式；false表示禁用自动提交模式。

- 异常

SQLException: 如果发生数据库访问错误，或者在参与分布式事务时调用此方法，或者在关闭的连接上调用此方法，则抛出该异常。

注意

如果在事务中调用此方法并更改了自动提交模式，则事务将被提交。如果调用了setAutoCommit，并且自动提交模式没有改变，则调用是无操作的。

getConnection(String url, Properties info) throws SQLException

- 功能描述

尝试建立与给定数据库URL的连接。DriverManager尝试从一组已注册的JDBC驱动程序中选择适当的驱动程序。

- 参数

表 3.18. getConnection 参数说明

名称	备注
url	格式为jdbc:subprotocol:subname的数据库网址

名称	备注
info	作为连接参数的任意字符串标签/值对应的列表；通常至少应包含“用户”和“密码”属性

- 返回值

成功返回UxConnection客户端与数据库建立的连接；失败抛异常。

- 异常

SQLException: 如果发生数据库访问错误或url为null，则抛出该异常。

SQLException: 当驱动程序确定已超过setLoginTimeout方法指定的超时值，并且至少尝试取消当前数据库连接尝试。

注意

如果一个属性被指定为的一部分url和指定的Properties对象，它是实现定义哪个值将优先。为了最大可移植性，应用程序应仅指定一次属性。

getConnection(String url) throws SQLException

- 功能描述

尝试建立与给定数据库URL的连接。DriverManager尝试从一组已注册的JDBC驱动程序中选择适当的驱动程序。

- 参数

url: 格式为jdbc:subprotocol:subname的数据库网址。

- 返回值

成功返回UxConnection客户端与数据库建立的连接；失败抛异常。

- 异常

SQLException: 如果发生数据库访问错误或url是null。

SQLException: 当驱动程序确定已超过setLoginTimeout方法指定的超时值，并且至少尝试取消当前数据库连接尝试。

getConnection(String url, String user, String password) throws SQLException

- 功能描述

尝试建立与给定数据库URL的连接。DriverManager尝试从一组已注册的JDBC驱动程序中选择适当的驱动程序。

- 参数

表 3.19. getConnection 参数说明

名称	备注
url	格式为jdbc:subprotocol:subname的数据库网址

名称	备注
user	正在连接的数据库用户
password	用户密码

- 返回值

成功返回UxConnection客户端与数据库建立的连接；失败抛异常。

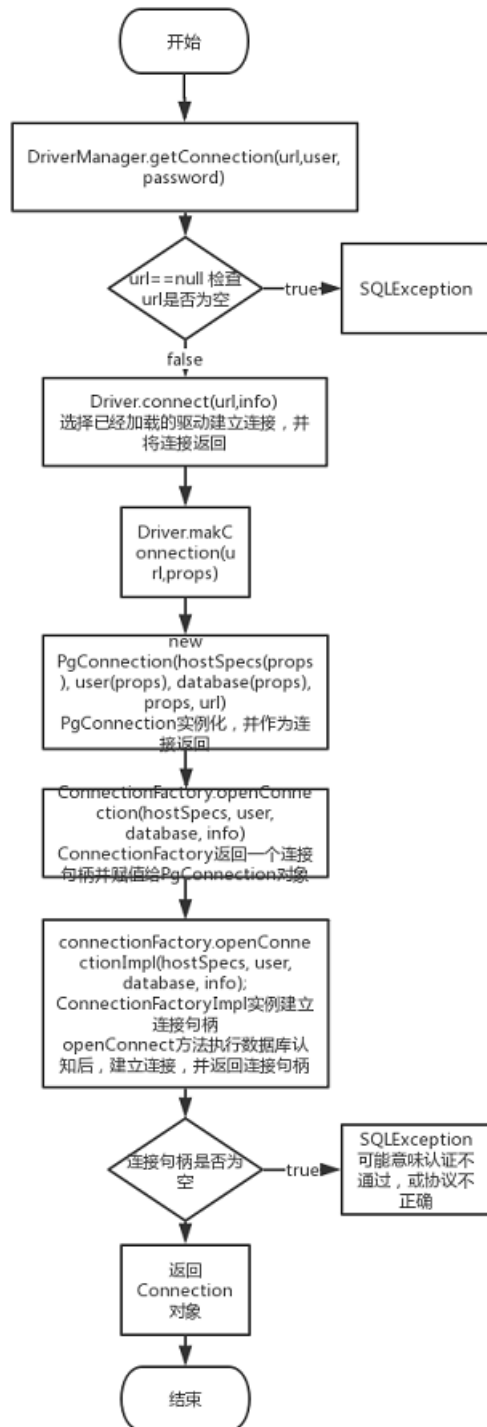
- 异常

SQLException: 如果发生数据库访问错误或url是null。

SQLException: 当驱动程序确定已超过setLoginTimeout方法指定的超时值，并且至少尝试取消当前数据库连接尝试。

注意

如果一个属性被指定为的一部分url和在还指定Properties对象，它是实现定义哪个值将优先。为了最大可移植性，应用程序应仅指定一次属性。



prepareCall(String#sql) throws SQLException

- 功能描述

创建一个CallableStatement对象来调用数据库存储过程。CallableStatement对象提供了设置其IN和OUT参数的方法，以及执行对存储过程的调用的方法。

注意

此方法针对处理存储过程调用语句进行了优化。当方法prepareCall完成时，一些驱动程序可能会将调用语句发送到数据库；其他人可能会等到执行CallableStatement对象。这对用户没有直接影响；但是，它确实会影响哪个方法抛出某些SQLExceptions。

- 参数

表 3.20. prepareCall 参数说明

名称	备注
sql	可能包含一个或多个“？”的SQL语句参数占位符通常使用JDBC调用转义语法指定此语句

- 返回值

一个新的默认的CallableStatement对象包含预编译的SQL语句。

- 异常

SQLException: 如果发生数据库访问错误或在闭合连接上调用此方法。

prepareCall(String sql, int resultSetType, int resultSetConcurrency) throws SQLException

- 功能描述

创建一个CallableStatement对象，该对象将生成具有给定类型和并发性的ResultSet对象。该方法与上面的prepareCall方法相同，但它允许覆盖默认的结果集类型和并发性。创建的结果集的可保持性可以通过调用getHoldability()来确定。

- 参数

表 3.21. prepareCall 参数说明

名称	备注
sql	一个String对象，即要发送到数据库的SQL语句；可能包含或多于‘?’参数
resultSetType	结果集类型，可选值如下所示。 <ul style="list-style-type: none"> • ResultSet.TYPE_FORWARD_ONLY • ResultSet.TYPE_SCROLL_INSENSITIVE • ResultSet.TYPE_SCROLL_SENSITIVE
resultSetConcurrency	并发类型

- 返回值

包含预编译的SQL语句的新的CallableStatement对象将生成具有给定类型和并发的ResultSet对象。

- 异常

SQLException: 如果发生数据库访问错误，或者在关闭的连接上调用该方法，或者给定参数不是ResultSet常数指示类型和并发，则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法，或者该方法不支持指定的结果集类型和结果集并发。

prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException

- 功能描述

创建一个CallableStatement对象，该对象将生成具有给定类型和并发性的ResultSet对象。此方法与上述prepareCall方法相同，但它允许默认结果集类型，结果集并发类型和可保持性被覆盖。

- 参数

表 3.22. prepareCall 参数说明

名称	备注
sql	一个String对象，即要发送到数据库的SQL语句；可能包含或多于'?'参数
resultSetType	可选值如下所示。 <ul style="list-style-type: none">• ResultSet.TYPE_FORWARD_ONLY• ResultSet.TYPE_SCROLL_INSENSITIVE• ResultSet.TYPE_SCROLL_SENSITIVE
resultSetConcurrency	<ul style="list-style-type: none">• ResultSet.CONCUR_READ_ONLY• ResultSet.CONCUR_UPDATABLE
resultSetHoldability	<ul style="list-style-type: none">• ResultSet.HOLD_CURSORS_OVER_COMMIT• ResultSet.CLOSE_CURSORS_AT_COMMIT

- 返回值

新CallableStatement对象，包含预编译的SQL语句，将产生ResultSet对象具有给定类型，并发性和可保存。

- 异常

SQLException: 如果发生数据库访问错误，或者在关闭的连接上调用该方法，或者给定参数不是ResultSet常数指定的类型，则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法，或者该方法不支持指定的结果集类型，则结果集可持续性和结果集并发。

isValid(int timeout) throws SQLException

- 功能描述

如果连接尚未关闭并且仍然有效，则返回true。驱动程序应该提交关于连接的查询或者使用一些其他机制，当调用此方法时，肯定验证连接仍然有效。驱动程序提交的验证连接的查询应在当前事务的上下文中执行。

- 参数

表 3.23. isValid 参数说明

名称	备注
timeout	等待数据库操作用于验证连接完成的时间（以秒为单位）。如果超时时间在操作完成之前到期，则此方法返回false。值为0表示超时未应用于数据库操作

- 返回值

如果连接有效，则为true，否则为false。

- 异常

SQLException: 如果为timeout提供的值小于0。

3.2.8. Class UXConnectionPoolDataSource

getPooledConnection(String user,String password) throws SQLException

- 功能描述

尝试建立可用作池连接的物理数据库连接。

- 参数

表 3.24. getPooledConnection 参数说明

名称	备注
user	正在连接的数据库用户
password	用户的密码

- 返回值

一个实现了PooledConnection接口的连接。

- 异常

SQLException: 发生数据库访问错误，则抛出该异常。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

3.2.9. Class UxDatabaseMetadata

getDriverName() throws SQLException

- 功能描述

检索此JDBC驱动程序的名称。

- 返回值

JDBC驱动程序名称。

- 异常

SQLException: 如果发生数据库访问错误, 则抛出该异常。

getUserName() throws SQLException

- 功能描述

检索此数据库已知的用户名。

- 返回值

数据库用户名。

- 异常

SQLException: 如果发生数据库访问错误, 则抛出该异常。

getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable) throws SQLException

- 功能描述

检索表格唯一标识行的最佳列集描述。它们由SCOPE命令。

每列描述都包含以下列, 如下所示。

1. SCOPE short =>实际结果范围。

- bestRowTemporary: 非常临时, 同时使用行。
- bestRowTransaction: 对当前事务的剩余部分有效。
- bestRowSession: 对当前会话剩余部分有效。

2. COLUMN_NAME String =>列名。

3. COLUMN_NAME int =>来自java.sql.Types的SQL数据类型。

4. TYPE_NAME String =>数据源相关类型名称, 对于UDT, 类型名称是完全限定的。

5. COLUMN_SIZE int =>精度。

6. BUFFER_LENGTH int =>未使用。

7. DECIMAL_DIGITS short => scale - 对于DECIMAL_DIGITS不适用的数据类型返回空值。

8. PSEUDO_COLUMN short =>是一个类似Oracle ROWID的伪列。

- bestRowUnknown: 可能是也可能不是伪列。

- bestRowNotPseudo: 不是伪列。
- bestRowPseudo: 是一个伪列。

COLUMN_SIZE列表示给定列的指定列大小。对于数值数据，这是最大精度。对于字符数据，这是字符长度。对于datetime数据类型，这是String表示形式的长度（假定分数秒分量的最大允许精度）。对于二进制数据，这是以字节为单位的长度。对于ROWID数据类型，这是以字节为单位的长度。对于列大小不适用的数据类型，返回空值。

- 参数

表 3.25. getBestRowIdentifier 参数说明

名称	备注
catalog	目录名称；必须匹配存储在数据库中的目录名称；“”检索没有目录的那些；null表示目录名称不应用于缩小搜索范围
schema	模式名称；必须匹配存储在数据库中的模式名称；“”检索没有模式的那些；null意味着不应该使用模式名称来缩小搜索范围
table	表名；必须匹配存储在数据库中的表名称
scope	感兴趣的范围；使用与SCOPE相同的值
nullable	包含可空的列

- 返回值

每行都是列描述。

- 异常

SQLException: 如果发生数据库访问错误，则抛出该异常。

`getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types) throws SQLException`

- 功能描述

检索给定目录中可用表的描述。仅返回与目录，模式，表名和类型条件匹配的表描述。他们根据TABLE_TYPE，TABLE_CAT，TABLE_SCHEM和TABLE_NAME 进行排序。

1. TABLE_CAT String => 表目录（可能为null）。
2. TABLE_SCHEM String => 表格式（可能为null）。
3. TABLE_NAME String => 表名。
4. TABLE_TYPE String => 表类型。典型的类型是“TABLE”，“VIEW”，“SYSTEM TABLE”，“GLOBAL TEMPORARY”，“LOCAL TEMPORARY”，“ALIAS”，“SYNONYM”。
5. REMARKS String => 对表的解释性评论。
6. TYPE_CAT String => 类型目录（可能为null）。
7. TYPE_SCHEM String => 类型模式（可能是null）。

8. TYPE_NAME String => 类型名称（可能为null）。
9. SELF_REFERENCING_COL_NAME String => 类型表的指定“标识符”列的名称（可能为null）。
10. REF_GENERATION String => 指定如何创建SELF_REFERENCING_COL_NAME中的值。值为“SYSTEM”，“USER”，“DERIVED”。（可能是null）

- 参数

表 3.26. getTables 参数说明

名称	备注
catalog	目录名称；必须匹配存储在数据库中的目录名称；“”检索没有目录的那些；null表示目录名称不应用于缩小搜索范围
schemaPattern	模式名称模式；必须匹配存储在数据库中的模式名称；“”检索没有模式的那些；null意味着不应该使用模式名称来缩小搜索范围
tableNamePattern	表名模式；必须匹配存储在数据库中的表名称
types	表格类型的列表，它们必须来自于从getTableTypes()返回的表格类型列表，以包括；null返回所有类型

- 返回值

每一行都是一个表的描述。

- 异常

SQLException: 如果发生数据库访问错误，则抛出该异常。

isReadOnly() throws SQLException

- 功能描述

检索该数据库是否处于只读模式。

- 返回值

如果是返回true，否则返回false。

- 异常

SQLException: 如果发生数据库访问错误，则抛出该异常。

3.2.10. Class UxParameterMetaData

getParameterTypeName(int param) throws SQLException

- 功能描述

检索指定参数的数据库特定类型名称。

- 参数

表 3.27. `getParameterTypeName` 参数说明

名称	备注
param	第一个参数是1，第二个是2，...

- 返回值

键入数据库使用的名称。如果参数类型是用户定义的类型，则返回完全限定类型的名称。

- 异常

`SQLException`: 如果发生数据库访问错误，则抛出该异常。

`getParameterClassName(int param) throws SQLException`

- 功能描述

检索其实例应传递给方法`UXPreparedStatement.setObject`的Java类的完全限定名称。

- 参数

表 3.28. `getParameterClassName` 参数说明

名称	备注
param	第一个参数是1，第二个是2，...

- 返回值

Java编程语言中的类的完全限定名称，方法`UXPreparedStatement.setObject`用于设置指定参数中的值。这是用于自定义映射的类名。

- 异常

`SQLException`: 如果发生数据库访问错误，则抛出该异常。

3.2.11. Class `UXPooledConnection`

`getConnection() throws SQLException`

- 功能描述

创建并返回一个`Connection`的代理连接`PooledConnection`对象。该方法应该由连接池调用而不是应用程序。

- 返回值

一个`Connection`对象，这是该`PooledConnection`对象的句柄。

- 异常

`SQLException`: 如果发生数据库访问错误，则抛出该异常。

`SQLFeatureNotSupportedException`: JDBC驱动程序不支持此方法。

3.2.12. Class UXPreparedStatement

`clearBatch()` throws `SQLException`

- 功能描述

清空这个Statement对象当前的SQL命令列表。

- 异常

`SQLException`: 如果发生数据库访问错误, 或者在关闭的连接中调用此方法, 或者驱动程序不支持批量更新, 则抛出该异常。

`close()` throws `SQLException`

- 功能描述

立即释放Statement对象的数据库和JDBC资源, 而不是等待它自动关闭时发生这种情况。通常最好的做法是在使用完资源后立即释放资源, 以避免占用数据库资源。针对已经关闭的Statement对象调用close方法没有任何效果。

- 异常

`SQLException`: 发生数据库访问错误, 则抛出该异常。

注意

当一个Statement对象关闭时, 其当前的ResultSet对象(如果存在)也被关闭。

`executeBatch()` throws `SQLException`

- 功能描述

向数据库提交一批命令以便执行, 如果所有命令都成功执行, 则返回一个更新计数数组。返回的数组的int 元素按顺序与批处理中的命令对应, 这些命令按照它们被添加到批处理中的顺序排序。方法executeBatch返回的数组中的元素可能是下列元素之一:

1. 大于或等于零的数字表示该命令已被成功处理, 并且是一个更新计数, 表示数据库中受命令执行影响的行数。
2. 值为SUCCESS_NO_INFO: 表示该命令已成功处理, 但受影响的行数未知。

如果批量更新中的其中一个命令无法正常执行, 则此方法将抛出一个 `BatchUpdateException`, 并且JDBC驱动程序可能会继续处理或不继续处理批处理中的剩余命令。但是, 驱动程序的行为必须与特定的DBMS保持一致, 或者始终继续处理命令或者从不继续处理命令。如果驱动程序在故障后继续处理, 则方法 `BatchUpdateException.getUpdateCounts` 返回的数组将包含与批处理中的命令一样多的元素, 并且至少有一个元素。

3. 值为EXECUTE_FAILED: 表示命令无法成功执行。

- 返回值

一个更新计数数组, 包含批处理中每个命令的一个元素。数组的元素根据命令添加到批处理中的顺序进行排序。

- 异常

SQLException: 如果发生数据库访问错误，或者在关闭的连接上调用此方法，Statement或者驱动程序不支持批处理语句。如果发送到数据库的命令之一无法正确执行或尝试返回结果集BatchUpdateException (SQLException 的子类)。

Sqltimeoutexception: 当驱动程序确定已超过setQueryTimeout方法指定的超时值并且尝试取消当前运行的Statement。

executeQuery () throws SQLException

- 功能描述

执行此UXPreparedStatement对象中的SQL查询，并返回查询UXPreparedStatement的ResultSet对象。

- 返回值

包含给定查询生成的数据的ResultSet对象；从不为null。

- 异常

SQLException: 如果发生数据库访问错误；或者这个方法在一个关闭的UXPreparedStatement上被调用，或者SQL语句不返回ResultSet对象，则抛出该异常。

SQLTimeoutException: 当驱动程序确定已经超过了setQueryTimeout方法指定的超时值，并且至少尝试取消当前运行的Statement。

executeUpdate () throws SQLException

- 功能描述

执行在该SQL语句UXPreparedStatement对象，它必须是一个SQL数据操纵语言（DML）语句，如INSERT，UPDATE或DELETE；或不返回任何内容的SQL语句，例如DDL语句。

- 返回值

1. SQL数据操作语言（DML）语句的行计数。
2. 0，针对不返回任何内容的SQL语句。

- 异常

SQLException: 如果发生数据库访问错误；或者这个方法在一个关闭的UXPreparedStatement上被调用，或者SQL语句不返回ResultSet对象，则抛出该异常。

SQLTimeoutException: 当驱动程序确定已经超过了setQueryTimeout方法指定的超时值，并且至少尝试取消当前运行的Statement。

execute () throws SQLException

- 功能描述

执行此UXPreparedStatement对象中的SQL语句，可能是任何类型的SQL语句。一些准备的语句返回多个结果；execute方法处理这些复杂语句以及由方法executeQuery和executeUpdate处理的更简单的语句形式。

execute方法返回一个boolean以指示第一个结果的形式。必须调用方法getResultSet或getUpdateCount来检索结果；并使用getMoreResults移动到任何后续结果。

- 返回值

如果第一个结果是ResultSet对象，则为true；如果第一个结果是更新计数或没有结果，则为false。

- 异常

SQLException：如果发生数据库访问错误；或者这个方法在一个关闭的UXPreparedStatement上被调用，或者给这个方法提供一个参数，则抛出该异常。

SQLTimeoutException：当驱动程序确定已经超过了setQueryTimeout方法指定的超时值，并且至少尝试取消当前运行的Statement。

setInt(int parameterIndex, int x) throws SQLException

- 功能描述

将指定的参数设置为给定的Java int值。当驱动程序将其发送到数据库时，会转换为SQL INTEGER值。

- 参数

表 3.29. setInt参数说明

名称	备注
parameterIndex	第一个参数是1，第二个是2，...
x	参数值

- 异常

SQLException：如果parameterIndex不对应于SQL语句中的参数标记；或者发生数据库访问错误或在封闭的UXPreparedStatement上调用此方法，则抛出该异常。

setString(int parameterIndex, String x) throws SQLException

- 功能描述

将指定的参数设置为给定的Java String值。驱动程序将其转换为SQL VARCHAR或LONGVARCHAR值（取决于参数相对于VARCHAR值的驱动程序限制的大小），当它发送到数据库时。

- 参数

表 3.30. setString 参数说明

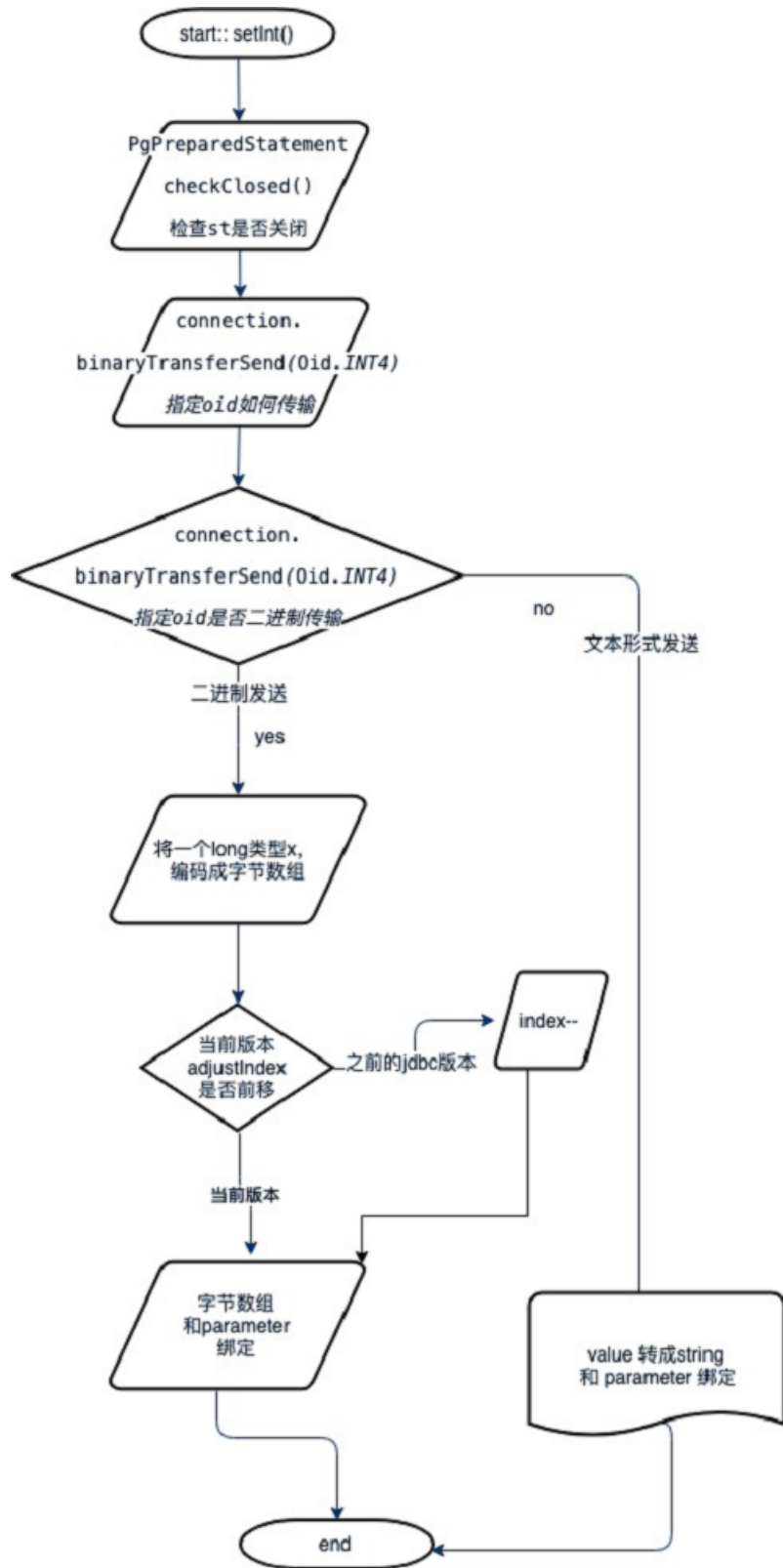
名称	备注
parameterIndex	第一个参数是1，第二个是2，...
x	参数值

- 返回值

void 返回为空。

- 异常

SQLException：如果parameterIndex不对应于SQL语句中的参数标记；或者发生数据库访问错误或在封闭的UXPreparedStatement上调用此方法，则抛出该异常。



`setTimestamp(int parameterIndex, Timestamp x, Calendar cal)` throws `SQLException`

- 功能描述

使用给定的Calendar对象将指定的Calendar设置为给定的java.sql.Timestamp值。驱动程序使用Calendar对象来构造一个SQL TIMESTAMP值，然后驱动程序将其发送到数据库。使用Calendar对象，驱动程序可以计算考虑到自定义时区的时间戳。如果没有Calendar对象，则驱动程序使用默认时区，即运行应用程序的虚拟机的时区。

- 参数

表 3.31. setTimestamp 参数说明

名称	备注
parameterIndex	第一个参数是1，第二个是2，...
x	参数值
cal	在Calendar对象的驱动程序将用来构造时间戳

- 返回值

void 返回为空。

- 异常

SQLException: 如果parameterIndex不对应于SQL语句中的参数标记；或者发生数据库访问错误或在封闭的UXPreparedStatement上调用此方法，则抛出该异常。

addBatch() throws SQLException

- 功能描述

向这个UXPreparedStatement对象的一批命令添加一组参数。

- 异常

SQLException: 如果发生数据库访问错误，或者在关闭的UXPreparedStatement调用此方法。

setBlob(int parameterIndex, Blob x) throws SQLException

- 功能描述

将指定的参数设置为给定的java.sql.Blob对象。当驱动程序将其发送到数据库时，将其转换为SQL Blob值。

- 参数

表 3.32. setBlob 参数说明

名称	备注
parameterIndex	第一个参数是1，第二个是2，...
x	映射SQL Blob值的 Blob对象

- 异常

SQLException: 如果parameterIndex不对应于SQL语句中的参数标记；如果发生数据库访问错误或在封闭的UXPreparedStatement上调用此方法。则抛出该异常。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

setTime(int parameterIndex, Time x) throws SQLException

- 功能描述

将指定的参数设置为给定的java.sql.Time值。当驱动程序将其发送到数据库时，将其转换为SQL TIME值。

- 参数

表 3.33. setTime 参数说明

名称	备注
parameterIndex	第一个参数是1，第二个是2，...
x	参数值

- 异常

SQLException: 如果parameterIndex不对应于SQL语句中的参数标记；如果发生数据库访问错误或在封闭的UXPreparedStatement上调用此方法。

setTimestamp(int parameterIndex, Timestamp x) throws SQLException

- 功能描述

将指定的参数设置为给定的java.sql.Timestamp值。当驱动程序将其发送到数据库时，将其转换为SQL TIMESTAMP值。

- 参数

表 3.34. setTimestamp 参数说明

名称	备注
parameterIndex	第一个参数是1，第二个是2，...
x	参数值

- 异常

SQLException: 如果parameterIndex不对应于SQL语句中的参数标记；如果发生数据库访问错误或在封闭的UXPreparedStatement上调用此方法。

setDate(int parameterIndex, Date x) throws SQLException

- 功能描述

使用运行应用程序的虚拟机的默认时区将指定的java.sql.Date设置为给定的java.sql.Date值。当驱动程序将其发送到数据库时，将其转换为SQL DATE值。

- 参数

表 3.35. setDate 参数说明

名称	备注
parameterIndex	第一个参数是1，第二个是2，...
x	参数值

- 异常

SQLException: 如果parameterIndex不对应于SQL语句中的参数标记; 如果发生数据库访问错误或在封闭的UXPreparedStatement上调用此方法。

3.2.13. Class UxResultSet

insertRow() throws SQLException

- 功能描述

将插入行的内容插入到此ResultSet对象中并进入数据库。当调用此方法时, 游标必须位于插入行上。

- 异常

SQLException: 如果发生数据库访问错误; 结果集并发性为CONCUR_READ_ONLY, 该方法在闭合结果集上调用, 如果当游标不在插入行上时调用此方法, 或者插入行中不是全部不可为空的列都被赋予非空值, 发生以上情况, 则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法, 则抛出该异常。

close() throws SQLException

- 功能描述

立即释放ResultSet对象的数据库和JDBC资源, 而不是等到它自动关闭时才释放。

ResultSet对象的关闭不会关闭ResultSet创建的Blob、Clob或NClob对象。Blob、Clob或NClob对象至少在创建它们的事务期间保持有效, 除非调用它们的free方法。

当ResultSet关闭时, 通过调用getMetaData方法创建的任何ResultSetMetaData实例仍然可访问。

注意

当Statement对象被关闭、重新执行或用于从多个结果序列中检索下一个结果时, ResultSet对象由生成它的Statement对象自动关闭。

调用已关闭的ResultSet对象上的方法close是无操作的。

- 异常

SQLException: 如果发生数据库访问错误。

getBlob(int columnIndex) throws SQLException

- 功能描述

将该ResultSet对象的当前行中指定列的值作为Java编程语言中的Blob对象检索。

- 参数

columnIndex: 第一列是1, 第二列是2, ...

- 返回值

一个Blob对象，表示指定列中的SQL Blob值。

- 异常

SQLException: 如果columnIndex无效；如果发生数据库访问错误或在关闭的结果集上调用此方法，发生以上情况，则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法。

getBlob(String columnLabel) throws SQLException

- 功能描述

将此ResultSet对象的当前行中指定列的值作为Java编程语言中的Blob对象检索。

- 参数

columnLabel: 使用SQL AS子句指定的列的标签。如果未指定SQL AS子句，则该标签是列的名称。

- 返回值

一个Blob对象，表示指定列中的SQL Blob值。

- 异常

SQLException: 如果columnLabel无效；如果发生数据库访问错误或在关闭的结果集上调用此方法，发生以上情况，则抛出该异常。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法。

getInt(int columnIndex) throws SQLException

- 功能描述

这个检索的是当前行中指定列的值。

- 参数

columnIndex: 第一列是1，第二列是2，...

- 返回值

列值；如果值为SQL NULL，则返回的值为0。

- 异常

SQLException: 如果columnIndex无效；如果发生数据库访问错误或在关闭的结果集上调用此方法。

getInt(String columnLabel) throws SQLException

- 功能描述

这个检索的是当前行中指定列的值。

- 参数

columnLabel: 使用SQL AS子句指定的列的标签。如果未指定SQL AS子句, 则该标签是列的名称。

- 返回值

列值; 如果值为SQL NULL, 返回值为0。

- 异常

SQLException: 如果columnLabel无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

getString(int columnIndex) throws SQLException

- 功能描述

这个检索的当前行中指定列的值ResultSet对象为String的Java编程语言。

- 参数

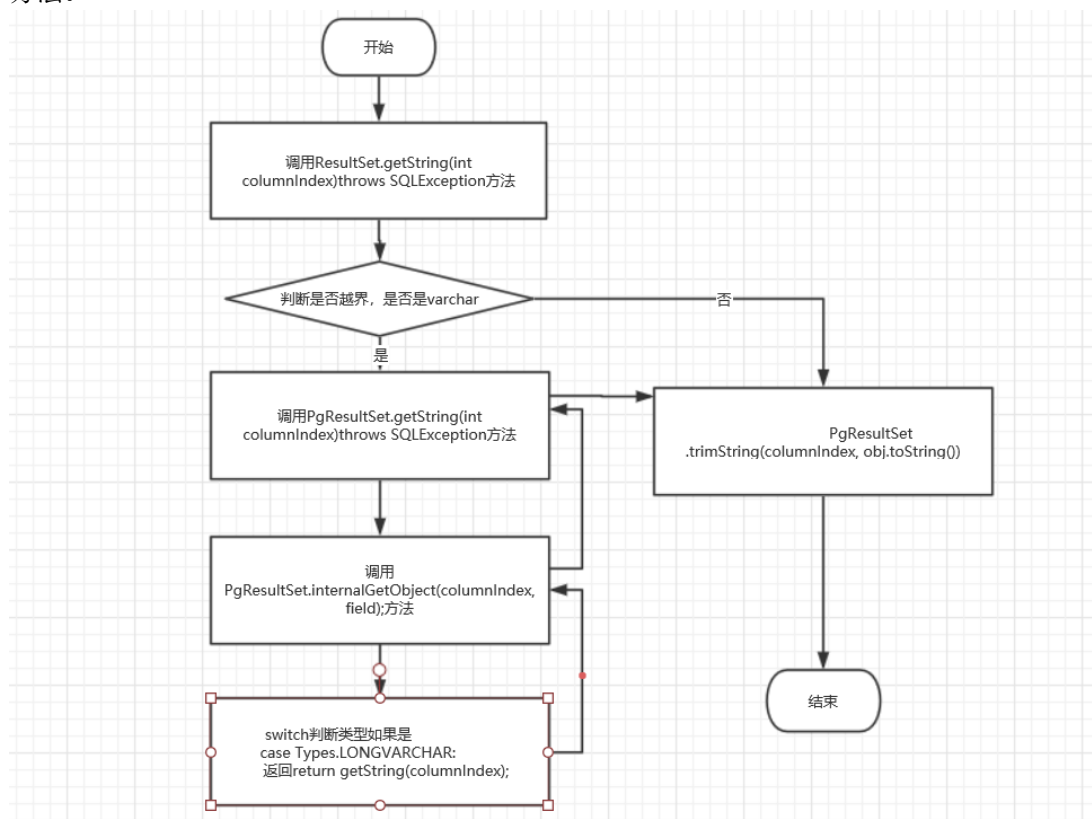
columnIndex: 第一列是1, 第二列是2, ...

- 返回值

列值; 如果值为SQL NULL, 返回值为null。

- 异常

SQLException: 如果columnIndex无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。



getString(String columnLabel) throws SQLException

• 功能描述

将该ResultSet对象的当前行中指定列的值作为Java编程语言中的String对象检索。

• 参数

columnLabel: 使用SQL AS子句指定的列的标签。如果未指定SQL AS子句, 则该标签是列的名称。

• 返回值

列值; 如果值为SQL NULL, 返回值为null。

• 异常

SQLException: 如果columnLabel无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

getTimestamp(int columnIndex) throws SQLException

• 功能描述

将该 ResultSet对象的当前行中指定列的值作为Java编程语言中的java.sql.Timestamp对象检索。

• 参数

columnIndex: 第一列是1, 第二列是2, ...

• 返回值

列值; 如果值为SQL NULL, 返回值为null。

• 异常

SQLException: 如果columnIndex无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

getTimestamp(int columnIndex, Calendar cal) throws SQLException

• 功能描述

将该ResultSet对象的当前行中指定列的值作为Java编程语言中的java.sql.Timestamp对象检索。如果基础数据库不存储时区信息, 则此方法使用给定的日历为时间戳创建适当的毫秒值。

• 参数

表 3.36. getTimestamp 参数说明

名称	备注
columnIndex	第一列是1, 第二列是2, ...
cal	用于java.util.Calendar对象

• 返回值

列值为java.sql.Timestamp对象; 如果值为SQL NULL, 则返回的值为null, 在Java编程语言。

- 异常

SQLException: 如果columnIndex无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

`getTimestamp(String columnLabel, Calendar cal) throws SQLException`

- 功能描述

将此ResultSet对象的当前行中指定列的值作为Java编程语言中的java.sql.Timestamp对象检索。如果基础数据库不存储时区信息, 则此方法使用给定的日历为时间戳创建适当的毫秒值。

- 参数

表 3.37. getTimestamp参数说明

名称	备注
columnLabel	使用SQL AS子句指定的列的标签。如果未指定SQL AS子句, 则该标签是列的名称
cal	用于java.util.Calendar日期的java.util.Calendar对象

- 返回值

列值为java.sql.Timestamp对象; 如果值为SQL NULL, 返回的值为null, 在Java编程语言。

- 异常

SQLException: 如果columnLabel无效或者发生数据库访问错误或者在关闭结果集上调用此方法。

`getTimestamp(String columnLabel) throws SQLException`

- 功能描述

将该 ResultSet对象的当前行中指定列的值作为Java编程语言中的 java.sql.Timestamp对象检索。

- 参数

columnLabel: 使用SQL AS子句指定的列的标签。如果未指定SQL AS子句, 则该标签是列的名称。

- 返回值

列值; 如果值为SQL NULL, 则返回值为null。

- 异常

SQLException: 如果columnLabel无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

`isClosed() throws SQLException`

- 功能描述

检索此ResultSet对象是否已关闭。如果对ResultSet调用了方法关闭, 或者该方法自动关闭, 则关闭ResultSet。

- 返回值

如果此ResultSet对象关闭，则为true；如果它仍然开放，则为false。

- 异常

SQLException: 如果发生数据库访问错误。

next() throws SQLException

- 功能描述

将光标从当前位置向前移动一行。ResultSet光标最初位于第一行之前；第一次调用方法next使第一行成为当前行；第二个调用使第二行成为当前行，以此类推。

当调用next方法返回false时，光标位于最后一行之后。任何调用需要当前行的ResultSet方法将导致抛出SQLException。如果结果集类型是TYPE_forward_only，则会抛出SQLException。

如果当前行的输入流已打开，则对方法next的调用将隐式关闭它。当读取新行时，ResultSet对象的警告链将被清除。

- 返回值

如果新的当前行有效，则为true；如果没有更多的行，则为false。

- 异常

SQLException: 如果发生数据库访问错误，或者在关闭的结果集上调用此方法。

deleteRow() throws SQLException

- 功能描述

从此ResultSet对象和底层数据库中删除当前行。当光标在插入行上时，无法调用此方法。

- 异常

SQLException: 如果发生数据库访问错误；结果集并发性为CONCUR_READ_ONLY；这个方法是在一个封闭的结果集上调用的，或者当光标在插入行上时调用这个方法。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

getBigDecimal(String columnLabel) throws SQLException

- 功能描述

获取结果集中字段名为columnLabel的值作为BigDecimal类型返回。

- 参数

表 3.38. getBigDecimal 参数说明

名称	备注
columnLabel	使用SQL AS子句指定的列的标签。如果未指定SQL AS子句，则该标签是列的名称

- 返回值

列值（全精度）；如果结果集中找不到列columnLabel，则返回null。

- 异常

SQLException: 如果columnLabel无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

getBoolean(String columnLabel) throws SQLException

- 功能描述

获取结果集中字段名为columnLabel的值作为boolean类型返回。

如果指定列的数据类型为CHAR或VARCHAR, 并且包含“0”或数据类型为BIT, TINYINT, SMALLINT, INTEGER或BIGINT, 并且包含0, 则返回值false。如果指定列的数据类型为CHAR或VARCHAR, 并且包含“1”或数据类型为BIT, TINYINT, SMALLINT, INTEGER或BIGINT, 并且包含1, 则返回值true。

- 参数

表 3.39. getBoolean 参数说明

名称	备注
columnLabel	使用SQL AS子句指定的列的标签。如果未指定SQL AS子句, 则该标签是列的名称

- 返回值

列值; 如果结果集中找不到列columnLabel, 则返回false。

- 异常

SQLException: 如果columnLabel无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

getDate(String columnLabel) throws SQLException

- 功能描述

获取结果集中字段名为columnLabel的值作为date类型返回。

- 参数

表 3.40. getDate 参数说明

名称	备注
columnLabel	使用SQL AS子句指定的列的标签。如果未指定SQL AS子句, 则该标签是列的名称

- 返回值

列值; 如果结果集中找不到列columnLabel, 则返回null。

- 异常

SQLException: 如果columnLabel无效; 如果发生数据库访问错误或在关闭的结果集上调用此方法。

getDouble(String columnLabel) throws SQLException

- 功能描述

获取结果集中字段名为columnLabel的值作为double类型返回。

- 参数

表 3.41. getDouble 参数说明

名称	备注
columnLabel	使用SQL AS子句指定的列的标签。如果未指定SQL AS子句，则该标签是列的名称结果

- 返回值

列值；如果结果集中找不到列columnLabel，则返回0。

- 异常

SQLException：如果columnLabel无效；如果发生数据库访问错误或在关闭的结果集上调用此方法。

getFloat(int columnIndex) throws SQLException

- 功能描述

获取结果集中字段名为columnLabel的值作为float类型返回。

- 参数

表 3.42. getFloat 参数说明

名称	备注
columnLabel	第一列是1，第二列是2，...

- 返回值

列值；如果结果集中找不到列columnLabel，则返回0。

- 异常

SQLException：如果columnLabel无效；如果发生数据库访问错误或在关闭的结果集上调用此方法。

getLong(String columnLabel) throws SQLException

- 功能描述

获取结果集中字段名为columnLabel的值作为long类型返回。

- 参数

表 3.43. getLong 参数说明

名称	备注
columnLabel	用SQL AS子句指定的列的标签。如果未指定SQL AS子句，则该标签是列的名称

- 返回值

列值；如果结果集中找不到列columnLabel，则返回0。

- 异常

SQLException：如果columnLabel无效；如果发生数据库访问错误或在关闭的结果集上调用此方法。

getRow() throws SQLException

- 功能描述

检索当前行号。第一行是第1行，第2行等等。

注意

对于结果集类型为TYPE_FORWARD_ONLY的ResultSets，是否支持getRow方法是可选的。

- 返回值

当前行号；如果没有当前行则返回0。

- 异常

SQLException：如果发生数据库访问错误或在关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：JDBC驱动程序不支持此方法。

getShort(String columnLabel) throws SQLException

- 功能描述

获取结果集中字段名为columnLabel的值作为short类型返回。

- 参数

表 3.44. getShort 参数说明

名称	备注
columnLabel	用SQL AS子句指定的列的标签。如果未指定SQL AS子句，则该标签是列的名称

- 返回值

列值，如果结果集中找不到列columnLabel，则返回0。

- 异常

SQLException：如果columnLabel无效；如果发生数据库访问错误或在关闭的结果集上调用此方法。

isFirst() throws SQLException

- 功能描述

检索光标是否在此ResultSet对象的第一行。

注意

对于结果集类型为TYPE_FORWARD_ONLY的ResultSet，是否支持isLast方法是可选的。

- 返回值

如果光标在第一行上则为true，否则为false。

- 异常

SQLException：如果发生数据库访问错误或在关闭结果集上调用此方法。

SQLFeatureNotSupportedException：JDBC驱动程序不支持此方法。

isLast() throws SQLException

- 功能描述

检索光标是否在此ResultSet对象的最后一行。注意：调用方法isLast可能是昂贵的，因为JDBC驱动程序可能需要提前一行才能确定当前行是否是结果集中的最后一行。

注意

对于结果集类型为TYPE_FORWARD_ONLY的ResultSet，是否支持isLast方法是可选的。

- 返回值

如果光标在最后一行则为true，否则为false。

- 异常

SQLException：如果发生数据库访问错误或在关闭结果集上调用此方法。

SQLFeatureNotSupportedException：JDBC驱动程序不支持此方法。

updateNull(int columnIndex) throws SQLException

- 功能描述

使用null值更新指定的列。更新器方法用于更新当前行或插入行中的列值。updater方法不更新底层数据库；而是updateRow或insertRow方法来更新数据库。

- 参数

表 3.45. updateNull 参数说明

名称	备注
columnIndex	第一列是1，第二列是2，...

- 异常

SQLException: 如果columnIndex无效; 如果发生数据库访问错误; 结果集并发性是CONCUR_READ_ONLY或者这个方法在一个封闭的结果集上被调用。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

updateRow() throws SQLException

- 功能描述

使用此ResultSet对象的当前行的新内容更新底层数据库。当光标在插入行上时, 无法调用此方法。

- 异常

SQLException: 如果发生数据库访问错误; 结果集并发性是CONCUR_READ_ONLY; 这个方法是在一个封闭的结果集上调用的, 或者当光标在插入行上时调用这个方法。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

updateString(int columnIndex, String x) throws SQLException

- 功能描述

使用String值更新指定的列。更新器方法用于更新当前行或插入行中的列值。updater方法不更新底层数据库; 而是updateRow或insertRow方法来更新数据库。

- 参数

表 3.46. updateString 参数说明

名称	备注
columnIndex	第一列是1, 第二列是2, ...
x	新的列值

- 异常

SQLException: 如果columnIndex无效; 如果发生数据库访问错误; 结果集并发性是CONCUR_READ_ONLY或者在一个封闭的结果集上调用这个方法。

SQLFeatureNotSupportedException: JDBC驱动程序不支持此方法。

3.2.14. Class UxResultSetMetaData

getColumnName(int column) throws SQLException

- 功能描述

获取指定列的名称。

- 参数

表 3.47. getColumnName 参数说明

名称	备注
column	第一列是1, 第二列是2, ...

- 返回值

列名。

- 异常

SQLException: 如果发生数据库访问错误, 则抛出该异常。

getColumnCount() throws SQLException

- 功能描述

返回此ResultSet对象中的列数。

- 返回值

列数。

- 异常

SQLException: 如果发生数据库访问错误, 则抛出该异常。

getColumnType(int column) throws SQLException

- 功能描述

检索指定列的SQL类型。

- 参数

表 3.48. getColumnType 参数说明

名称	备注
column	第一列是1, 第二列是2, ...

- 返回值

来自java.sql.Types的SQL类型。

- 异常

SQLException: 如果发生数据库访问错误, 则抛出该异常。

3.2.15. Class UXSQLException

getNextException()

- 功能描述

通过setNextException(SQLException ex) 检索链接到此 SQLException对象的异常。

- 返回值

链中的下一个SQLException对象; 如果没有就为null。

3.2.16. Class UxStatement

clearBatch() throws SQLException

- 功能描述

清空这个Statement对象当前的SQL命令列表。

- 异常

SQLException: 如果发生数据库访问错误，或在关闭的 Statement调用此方法或驱动程序不支持批量更新，则抛出该异常。

close() throws SQLException

- 功能描述

立即释放ResultSet对象的数据库和JDBC资源，而不是等到它自动关闭时才释放。一般来说，一旦完成调用，尽快释放资源，以避免捆绑数据库资源。

调用已关闭的Statement对象上的方法close不起作用。

注意

当一个Statement对象关闭时，其当前的ResultSet对象（如果存在）也被关闭。

- 异常

SQLException: 发生数据库访问错误，则抛出该异常。

executeBatch() throws SQLException

- 功能描述

向数据库提交一批命令以便执行，如果所有命令都成功执行，则返回一个更新计数数组。返回的数组的int 元素按顺序与批处理中的命令对应，这些命令按照它们被添加到批处理中的顺序排序。方法executeBatch返回的数组中的元素可能是下列元素之一：

1. 大于或等于零的数字表示该命令已被成功处理，并且是一个更新计数，表示数据库中受命令执行影响的行数。
2. 值为SUCCESS_NO_INFO，表示该命令已成功处理，但受影响的行数未知。

如果批量更新中的其中一个命令无法正常执行，则此方法将抛出一个BatchUpdateException，并且JDBC驱动程序可能会继续处理或不继续处理批处理中的剩余命令。但是，驱动程序的行为必须与特定的DBMS保持一致，或者始终继续处理命令或者从不继续处理命令。如果驱动程序在故障后继续处理，则方法BatchUpdateException.getUpdateCounts返回的数组将包含与批处理中的命令一样多的元素，并且至少有一个元素。

3. 值为EXECUTE_FAILED，表示命令无法成功执行，并且仅在发生失败后，驱动程序继续进行其他处理时才会发生。

- 返回值

包含批处理中每个命令一个元素的更新计数的数组。数组的元素根据命令添加到批处理中的顺序进行排序。

- 异常

SQLException: 如果发生数据库访问错误，或者在关闭的连接上调用此方法，Statement或者驱动程序不支持批处理语句。如果发送到数据库的命令之一无法正确执行或尝试返回结果集BatchUpdateException（SQLException的子类）。

SQLException: 当驱动程序确定已超过setQueryTimeout方法指定的超时值并且尝试取消当前运行的Statement。

executeQuery(String sql) throws SQLException

- 功能描述

执行给定的SQL语句，返回单个ResultSet对象。

注意

此方法无法在UXPreparedStatement或CallableStatement使用。

- 参数

sql: 要发送到数据库的SQL语句，通常为静态SQL SELECT语句。

- 返回值

一个ResultSet对象，其中包含给定查询产生的数据；从不为null。

- 异常

SQLException: 如果发生数据库访问错误，此方法在封闭的Statement上调用，给定的SQL语句生成除单个ResultSet对象之外的任何东西，该方法在UXPreparedStatement或CallableStatement不能使用。

SQLException: 当驱动程序确定已超过setQueryTimeout方法指定的超时值并且尝试取消当前运行的Statement。

execute(String sql) throws SQLException

- 功能描述

执行给定的SQL语句，这可能会返回多个结果。在某些情况下，单个SQL语句可能会返回多个结果集和/或更新计数。通常可以忽略这一点，除非执行知道可能返回多个结果的存储过程，或者正在动态地执行一个未知的SQL字符串。

注意

execute方法执行SQL语句并指示第一个结果的形式。必须使用方法getResultSet或getUpdateCount检索结果，然后getMoreResults移动到任何后续结果。

- 参数

sql: 任何SQL语句。

- 返回值

如果第一个结果是一个ResultSet对象，则返回true；如果是更新计数或没有结果，则返回false。

- 异常

SQLException: 如果发生数据库访问错误，此方法在封闭的 Statement上调用，该方法被 UXPreparedStatement或 CallableStatement调用，发生以上情况，则抛出该异常。

SQLTimeoutException: 当驱动程序确定已超过setQueryTimeout方法指定的超时值并且尝试取消当前运行的Statement。

executeUpdate(String sql,int[] columnIndexes) throws SQLException

- 功能描述

执行给定的SQL语句，并向驱动程序发出信号，指出给定数组中指示的自动生成的键应该可用于检索。该数组包含目标表中包含应该可用的自动生成的键的列的索引。如果SQL语句不是INSERT语句，或者SQL语句能够返回自动生成的键，驱动程序将忽略该数组。

注意

execute方法执行SQL语句并指示第一个结果的形式。必须使用方法 getResultSet或getUpdateCount检索结果，然后getMoreResults移动到任何后续结果。

- 参数

表 3.49. executeUpdate参数说明

名称	备注
sql	一个SQL数据操纵语言（DML）语句，如INSERT，UPDATE或DELETE；或不返回任何内容的SQL语句，例如DDL语句
columnIndexes	一列索引数组，指示应从插入的行返回的列

- 返回值

1. SQL数据操作语言（DML）语句的行计数。
2. 0；不返回的SQL语句。

- 异常

SQLException: 如果发生数据库访问错误，此方法在封闭的Statement上调用，SQL语句返回一个ResultSet对象，提供给此方法的第二个参数不是其元素为有效列索引的int数组，该方法被 UXPreparedStatement或CallableStatement调用。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法。

SQLTimeoutException: 当驱动程序确定已超过setQueryTimeout方法指定的超时值并且尝试取消当前运行的Statement。

executeUpdate(String sql, String[] columnNames) throws SQLException

- 功能描述

执行给定的SQL语句，并向驱动程序发出信号，指出给定数组中指示的自动生成的键应该可用于检索。该数组包含目标表中包含应该可用的自动生成的键的列的名称。如果SQL语句不是INSERT语句，或者SQL语句能够返回自动生成的键，驱动程序将忽略该数组。

注意

此方法无法在UXPreparedStatement或CallableStatement中使用。

- 参数

表 3.50. executeUpdate 参数说明

名称	备注
sql	一个SQL数据操纵语言（DML）语句，如INSERT，UPDATE或DELETE；或不返回任何内容的SQL语句，例如DDL语句
columnNames	应该从插入的行返回的列的名称数组

- 返回值

INSERT、UPDATE或DELETE语句的行计数，或不返回任何内容的SQL语句的行计数为0。

- 异常

SQLException: 如果发生数据库访问错误，此方法在关闭的Statement上调用，SQL语句返回一个ResultSet对象，提供给此方法的第二个参数不是其元素为有效列名的String数组，该方法被调用一个UXPreparedStatement或CallableStatement。

SQLFeatureNotSupportedException: 如果JDBC驱动程序不支持此方法。

SQLTimeoutException: 当驱动程序确定已超过setQueryTimeout方法指定的超时值并且尝试取消当前运行的Statement。

executeUpdate(String sql) throws SQLException

- 功能描述

执行给定的SQL语句，这可能是INSERT，UPDATE，或DELETE语句，或者不返回任何内容，如SQL DDL语句的SQL语句。

注意

此方法无法在UXPreparedStatement或CallableStatement中使用。

- 参数

表 3.51. executeUpdate 参数说明

名称	备注
sql	一个SQL数据操纵语言（DML）语句，如INSERT，UPDATE或DELETE；或不返回任何内容的SQL语句，例如DDL语句

- 返回值

INSERT、UPDATE或DELETE语句的行计数，或不返回任何内容的SQL语句的行计数为0。

- 异常

SQLException: 如果发生数据库访问错误，此方法在关闭的Statement上调用，SQL语句返回一个ResultSet对象，提供给此方法的第二个参数不是其元素为有效列名的String数组，该方法被调用一个UXPreparedStatement或CallableStatement。

SQLException: 当驱动程序确定已超过setQueryTimeout方法指定的超时值并且尝试取消当前运行的Statement。

getResultSet() throws SQLException

- 功能描述

检索当前结果为ResultSet对象。每个结果应该只调用一次这个方法。

- 返回值

当前结果为ResultSet对象，如果结果为更新计数或没有更多结果，则为null。

- 异常

SQLException: 如果发生数据库访问错误或此方法在关闭时调用Statement。

addBatch(String sql) throws SQLException

- 功能描述

将给定的SQL命令添加到此Statement对象的当前命令列表中。该列表中的命令可以通过调用方法executeBatch作为批处理执行。

注意

此方法无法在UXPreparedStatement或CallableStatement中使用。

- 参数

表 3.52. addBatch 参数说明

名称	备注
sql	通常这是一个SQL INSERT或UPDATE语句

- 异常

SQLException: 如果发生数据库访问错误，此方法在封闭的Statement上调用，驱动程序不支持批量更新，则在UXPreparedStatement或CallableStatement上调用此方法。

3.2.17. Class UXTime

toString()

- 功能描述

以JDBC时间戳格式格式化时间。

- 返回值

一个hh:mm:ss格式的字符串。

toLocalTime()

- 功能描述

将此Time对象转换为LocalTime。

转换创建LocalTime，表示相同的小时，分钟，和作为该第二时间值Time。

- 返回值

表示相同时间值的LocalTime对象。

3.2.18. Class UXTimeStamp

getTime()

- 功能描述

返回由此Timestamp对象表示的自1970年1月1日以来的毫秒数。

- 返回值

自1970年1月1日以来，以此日期为准的00:00:00GMT的毫秒数。

toString()

- 功能描述

以JDBC时间戳格式格式化时间戳。yyyy-mm-dd hh:mm:ss.fffffffff，其中fffffffff表示纳秒。

- 返回值

一个String对象在yyyy-mm-dd hh:mm:ss.fffffffff格式。

3.2.19. Class Wrapper

isWrapperFor(Class<?> iface) throws SQLException

- 功能描述

如果这实现了接口参数，或者直接或间接地为一个对象的包装器返回true。否则返回false。
如果这实现了接口，那么返回true，否则如果这是一个包装器，那么返回在包装对象上递归调用isWrapperFor的结果。如果这不实现接口并且不是包装器，则返回false。这个方法应该被实现为相比低成本运作unwrap使呼叫者可以用这种方法来避免昂贵unwrap调用可能失败。如果此方法返回true，则使用相同参数调用unwrap应该会成功。

- 参数

表 3.53. isWrapperFor 参数说明

名称	备注
iface	定义接口的类

- 返回值

如果这实现了接口，或者直接或间接地包装一个对象，则为true。

- 异常

SQLException: 在确定调用对象是不是给定的接口的包装器时发生错误的时候抛出。

第 4 章 使用JDBC

JDBC是Java 1.1及以后的核心API，为SQL兼容的数据库提供了一个标准的接口集合。

UXDB提供的JDBC驱动类型是本地协议驱动。本地协议驱动是用Java书写的，并且与数据库之间使用数据库自己的网络协议通讯。因此，驱动是平台无关的。一旦编译，该驱动可以用于任意平台。

4.1. 设置JDBC驱动

主要是针对用户开始书写或者运行使用JDBC接口的程序之前需要采取的步骤。

1. 设置类路径。

获取最新的uxdb jdbc驱动程序。

要使用驱动，它的JAR包必须包含在classpath里。可以选择把路径jar包的路径放到CLASSPATH环境变量里，或是在执行应用程序的命令行中加入对应的参数。

示例：有一个使用JDBC驱动访问数据库的应用，该应用安装在/usr/local/lib/myapp.jar。UXDB JDBC驱动安装在/usr/local/UXsql/share/java/UXDB.jar。要运行应用，有两种操作方式。

方法一：

```
export CLASSPATH=/usr/local/lib/myapp.jar:/usr/local/UXsql/share/java/UXDB.jar:java myApp
```

方法二：

```
export CLASSPATH=/usr/local/lib/myapp.jar
java -cp ./usr/local/UXsql/share/java/UXDB.jar myapp
```

2. 准备数据库服务器。

Java只使用TCP/IP 连接，因此UXDB服务器必须配置成接受TCP/IP连接，可以通过在uxsinodb.conf文件里设置tcpip_socket=true或者启动postmaster的时候带-i参数实现这个目的。

同样，在ux_hba.conf文件里的需要配置客户端认证设置。JDBC驱动支持trust, ident, password, md5和crypt认证方式。

4.2. 初始化驱动

主要描述如何在程序里装载和初始化JDBC驱动。

1. 导入JDBC

任何使用JDBC的源程序都需要输入java.sql包。

```
import java.sql.*;
```

2. 装载驱动

在于数据库连接之前，需要装载驱动。有两种方法，哪种更好取决于使用的代码。

- 代码用Class.forName()方法明确地装载驱动。对于UXDB需要使用：

```
Class.forName("com.uxsino.uxdb.Driver");
```

来装载驱动，并且在装载时，驱动将自动在JDBC注册。

注意

forName()方法可能抛出一个ClassNotFoundException，所以如果驱动不可获得时需要捕获它。

这是最常用的方法，但是会把代码限制于UXDB专用。如果代码以后还要访问其他数据库，并且不想使用任何UXDB相关的扩展，那么还有第二种方法可用。

- 把驱动做为参数在JVM启动时传递给它，使用-D参数。比如：

```
java -D jdbc.drivers= com.uxsino.uxdb.Driver example.ImageView
```

在这个例子里，JVM将试图把驱动作为它的初始化的一部分装载。一旦完成，启动ImageView：

现在这个方法更好一点，因为它允许代码用于其他数据库，而不用重新编译代码。唯一要修改的东西是URL。

当代码试图打开一个Connection，而收到一个抛出的No driver available SQLException，这可能是因为驱动不在类路径里，或者参数值不正确。

3. 数据库连接

在JDBC里，数据库是用URL（Uniform Resource Locator统一资源定位符）表示的。在UXDB中，可以由下面几种格式之一表示：

- jdbc:uxdb:database
- jdbc:uxdb://host/database
- jdbc:uxdb://host:port/database

表 4.1. 数据库连接参数

名称	含义
host	服务器的主机名。缺省是localhost。要想声明一个IPv6的地址，必须把host参数用方括弧包围起来，比如：jdbc:uxdb://[::1]:5740/accounting
port	服务器监听的端口号。缺省是UXDB标准的端口号（5432）
database	数据库名

要连接数据库，需要从JDBC获取一个Connection实例，会用到DriverManager.getConnection()方法。

```
Connection conn = DriverManager.getConnection(url, username, password);
```

4. 关闭连接

关闭数据库连接，只需要对Connection调用close()方法：

```
conn.close();
```

4.3. 发出查询和处理结果

在数据库运行一个SQL语句的时候，都需要一个Statement或PreparedStatement实例，就可以发出一个查询，会返回一个ResultSet实例，在其内部包含整个结果。

发出一个简单的查询然后用一个Statement打印出每行的第一个字段，示例如下。

```
Statement st = db.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM mytable WHERE columnfoo = 500");
while (rs.next()) {
    System.out.print("Column 1 returned ");
    System.out.println(rs.getString(1));
}
rs.close();
st.close();
```

使用PreparedStatement发出查询，并且在查询中绑定数值，示例如下。

```
int foovalue = 500;
PreparedStatement st = db.prepareStatement("SELECT * FROM mytable WHERE columnfoo = ?");
st.setInt(1, foovalue);
ResultSet rs = st.executeQuery();
while (rs.next()) {
    System.out.print("Column 1 returned ");
    System.out.println(rs.getString(1));
}
rs.close();
st.close();
```

4.3.1. 基于一个游标获取结果

缺省时，驱动程序一次从查询里获取所有的结果。这样可能对于大的数据集来说是不方便的，因此JDBC驱动提供了一个方法从一个数据库游标上抽取少数几行的ResultSet的方法。

在链接的客户端这边缓冲了一小部分数据行，并且在用完之后，通过重定位游标检索下一个数据行块。

把代码修改成游标模式，只是设置Statement的抓取大小到一个合适的程度。把抓取大小设置为0则是缓冲所有行的模式（缺省行为），示例如下。

```
Statement st = db.createStatement();
// 打开游标的使用。
st.setFetchSize(50);
ResultSet rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next()) {
    System.out.print("a row was returned.");
}
rs.close();
// 关闭游标的使用。
```



```
st.setFetchSize(0);
ResultSet rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next()) {
    System.out.print("many rows were returned.");
}
rs.close();
// 关闭语句
st.close();
```

4.3.2. 使用Statement或UXPreparedStatement接口

在使用Statement或UXPreparedStatement接口时必须考虑下面的问题：

- 可以将一个Statement或 UXPreparedStatement实例使用任意次。可以在打开一个连接后马上创建一个Statement 实例，并且在连接的生存期里使用。必须记住每个Statement或UXPreparedStatement只能存在一个ResultSet。
- 如果需要在处理一个ResultSet的时候执行一个查询，只需要创建并且使用另外一个Statement。
- 如果使用了线程，并且有几个线程使用数据库，针对每个线程都必须使用一个独立的Statement。
- 用完Statement或者UXPreparedStatement之后，应该关闭它。

4.3.3. 使用ResultSet接口

使用ResultSet接口时必须考虑下面的问题：

- 在读取任何数值的时候，必须调用next()。如果还有结果则返回真（true），但更重要的是，它为处理准备了数据行。
- 在JDBC规范里，对一个字段应该只访问一次。遵循这个规则是最安全的，不过目前UXDB驱动允许对一个字段访问任意次。
- 一旦结束对一个ResultSet的处理，必须调用对应的close()来关闭它。
- 一旦使用那个创建ResultSet的Statement做另一个查询请求，当前打开的ResultSet实例将自动关闭。

4.3.4. 执行更新

要改变数据（执行INSERT，UPDATE或者DELETE操作），需要使用executeUpdate()方法。

这个方法类似用于发出SELECT的executeQuery()，但是返回值不是ResultSet，返回的是INSERT，UPDATE或者DELETE语句影响的记录数。

发出一个简单的DELETE并打印出删除的行数，示例如下。

```
int foovalue = 500;
UXPreparedStatement st = db.prepareStatement("DELETE FROM mytable WHERE columnfoo = ?");
st.setInt(1, foovalue);
int rowsDeleted = st.executeUpdate();
System.out.println(rowsDeleted + " rows deleted");
st.close();
```

4. 4. 调用存储过程

UXDB' s JDBC驱动完全支持调用UXDB存储过程。

调用一个UXDB内置的函数upper，简单地把字符串参数转换成大写。示例如下。

```
// 关闭事务
con.setAutoCommit(false);
// 过程调用
CallableStatement upperProc = con.prepareCall("{ ? = call upper( ? ) }");
upperProc.registerOutParameter(1, Types.VARCHAR);
upperProc.setString(2, "lowercase to uppercase");
upperProc.execute();
String upperCased = upperProc.getString(1);
upperProc.close();
```

- CallableStatement接口

所有适用于Statement和UXPreparedStatement的注意事项都适用于CallableStatement。

只能在一个事务里面调用一个存储过程。

- 从存储过程里获取ResultSet

UXDB的存储过程可以通过一个refcursor值返回结果集。

作为JDBC的扩展，UXDB JDBC驱动可以将refcursor值作为ResultSet值返回。

从一个函数里获取refcursor值，在调用一个返回refcursor的函数时，必须把返回类型getObject转换成ResultSet。示例如下。

```
// 关闭自动提交事务
con.setAutoCommit(false);
// 过程调用
CallableStatement proc = con.prepareCall("{ ? = call doquery ( ? ) }");
proc.registerOutParameter(1, Types.Other);
proc.setInt(2, -1);
proc.execute();
ResultSet results = (ResultSet) proc.getObject(1);
while (results.next()) {
    // 处理结果集
}
results.close();
proc.close();
```

把返回值refcursor看作一种特殊的类型。JDBC驱动提供了org. UXDB. UXRefCursorResultSet用于这个目的。示例如下。

```
con.setAutoCommit(false);
CallableStatement proc = con.prepareCall("{ ? = call doquery ( ? ) }");
proc.registerOutParameter(1, Types.Other);
proc.setInt(2, 0);
org. UXDB. UXRefCursorResultSet refcurs
    = (UXRefCursorResultSet) con.getObject(1);
String cursorName = refcurs.getRefCursor();
```

```
proc.close();
```

4.5. 创建和更改数据库对象

要创建，更改或者删除一个类似表或者视图这样的数据库对象，需要使用`execute()`方法。这个方法类似于`executeQuery()`，不过它不返回结果。

删除一个表，示例如下。

```
Statement st = db.createStatement();
st.execute("DROP TABLE mytable");
st.close();
```

4.6. 存储二进制数据

UXDB提供两种不同的方法存储二进制数据。二进制数据可以使用二进制数据类型`bytea`存储在表中，或者使用大对象特性，该特性以一种特殊的格式将二进制数据存储在一个独立的表中，然后通过表中保存一个指向该表的类型为`oid`的数值来引用它。

为了判断哪种方法比较合适，必须理解每种方法的局限。

`bytea`数据类型并不适合存储非常大数量的二进制数据。虽然类型为`bytea`的字段可以存储最多1G字节的二进制数据，但是这样它会要求数量巨大的内存来处理这样巨大的数值。存储二进制的大对象的方法更适合存储非常大的数值，但也有自己的局限。特别是删除一个包含大对象引用的行并未删除大对象。删除大对象是一个需要执行的独立的操作。大对象还有一些安全性的问题，因为任何连接到数据库的人都可以查看或者更改大对象，即使没有查看/更新包含大对象引用的行的权限也一样。

版本7.2是第一个支持`bytea`类型的JDBC驱动版本。在7.2中引入的这个功能同时也引入了一个和以往版本不同的行为。自7.2以来，方法`getBytes()`，`setBytes()`，`getBinaryStream()`和`setBinaryStream()`操作`bytea`类型。在7.1和更早的版本里这些方法操作和OID类型关联的大对象。可以通过在`Connection`上设置`compatible`属性为数值7.1来获取旧的7.1的行为。

要使用`bytea`数据类型只需要使用`getBytes()`，`setBytes()`，`getBinaryStream()`，或者`setBinaryStream()`方法。

要使用大对象的功能，可以使用UXDB JDBC驱动提供的`LargeObject`类，或者使用`getBLOB()`和`setBLOB()`方法。

必须在一次SQL事务内访问大对象。可以通过调用`setAutoCommit(false)`打开一个事务。

注意

在将来的JDBC驱动中，`getBLOB()`和`setBLOB()`方法可能不再操作大对象，而是将处理`bytea`数据类型。因此如果要用大对象，建议使用`LargeObject` API。

在JDBC里处理二进制数据，假设有一个表包含一幅图像和它的文件名，并且在`bytea`字段里存储图像，示例如下。

```
CREATE TABLE images (imgname text, img bytea);
```

插入一幅图像。

```

File file = new File("myimage.gif");
FileInputStream fis = new FileInputStream(file);
UXPreparedStatement ps = conn.prepareStatement("INSERT INTO images VALUES (?, ?)");
ps.setString(1, file.getName());
ps.setBinaryStream(2, fis, file.length());
ps.executeUpdate();
ps.close();
fis.close();

```

setBinaryStream() 把来自一个流的一些数目的字节转换成类型bytea的字段。如果图像的内容已经放在byte[]里面了，那么也可以使用setBytes()方法。

检索一幅图像使用的是UXPreparedStatement，也可以使用Statement。示例如下。

```

UXPreparedStatement ps = con.prepareStatement("SELECT img FROM images WHERE imgname = ?");
ps.setString(1, "myimage.gif");
ResultSet rs = ps.executeQuery();
if (rs != null) {
    while (rs.next()) {
        byte[] imgBytes = rs.getBytes(1);
        // 从这开始使用数据
    }
    rs.close();
}
ps.close();

```

这里的二进制数据是以byte[]形式检索的。也可以使用一个InputStream。另外可能会需要存储一个非常大的文件，因此希望使用LargeObject类存储该文件。

```
CREATE TABLE imageslo (imgname text, imgOID oid);
```

插入一个图像，示例如下。

```

// 所有大对象 API 调用都必须在一次事务中
conn.setAutoCommit(false);

// 获取大对象管理器以便进行操作
LargeObjectManager lobj = ((org.UXDB.UXConnection)conn).getLargeObjectAPI();

// 创建一个新的大对象
int oid = lobj.create(LargeObjectManager.READ | LargeObjectManager.WRITE);

// 打开一个大对象进行写
LargeObject obj = lobj.open(oid, LargeObjectManager.WRITE);

// 现在打开文件
File file = new File("myimage.gif");
FileInputStream fis = new FileInputStream(file);

// 从文件拷贝数据到大对象
byte buf[] = new byte[2048];
int s, tl = 0;
while ((s = fis.read(buf, 0, 2048)) > 0){
    obj.write(buf, 0, s);
}

```

```

        tl += s;
    }

    // 关闭大对象
    obj.close();

    //现在向 imgeslo 插入行
    UXPreparedStatement ps = conn.prepareStatement("INSERT INTO imageslo VALUES (?,?)");
    ps.setString(1, file.getName());
    ps.setInt(2, oid);
    ps.executeUpdate();
    ps.close();
    fis.close();

    从大对象中检索图像，示例如下。

    // 所有 LargeObject API 调用都必须在一个事务里
    conn.setAutoCommit(false);

    // 获取大对象管理器以便进行操作
    LargeObjectManager lobj = ((org.UXDB.UXConnection)conn).getLargeObjectAPI();

    UXPreparedStatement ps = con.prepareStatement("SELECT imgoid FROM imageslo WHERE
    imgname = ?");
    ps.setString(1, "myimage.gif");
    ResultSet rs = ps.executeQuery();
    if (rs != null) {
        while (rs.next()) {
            //打开大对象读
            int oid = rs.getInt(1);
            LargeObject obj = lobj.open(oid, LargeObjectManager.READ);

            //读取数据
            byte buf[] = new byte[obj.size()];
            obj.read(buf, 0, obj.size());
            //在这里对读取的数据做些处理

            // 关闭对象
            obj.close();
        }
        rs.close();
    }
    ps.close();

```

4. 7. 多线程或服务器小应用环境里使用驱动

许多JDBC驱动的一个共同问题是它们在任意时刻一个线程只能使用一个Connection，否则可能一个线程在发送一个查询而另一个线程正在接受结果，这样做可能导致服务器的混乱。

UXDB JDBC驱动都是线程安全的。如果应用要使用多线程，那么不必考虑任意时刻只允许一个线程使用数据库的复杂算法。

如果一个线程在其他线程正在使用数据库时试图访问数据库，那么它将等到另一个线程完成当前操作之后进行。如果这是一个普通的SQL语句，那么该操作就是发送该语句，并检索任何

ResultSet（完整的）。如果这是一个捷径调用（示例： 从一个大对象里读取一个数据块），那么这时就包含发送和接收该数据块。

这对客户端的大小应用都很好，但是可能造成服务器端小应用（servlets）的性能问题。如果有好几个线程执行查询，那么它们除一个之外其它都是暂停的。要解决这个问题，建议创建一个连接池(pool of Connections)。当一个线程需要使用数据库，它向管理类请求一个Connection。管理器赋予该线程一个空闲连接，然后把它标记为忙。如果没有空闲连接，管理器就打开一个。一旦线程完成数据库使用，该线程把连接返回给管理器，管理器既可以关闭该连接，也可以把它加到连接池里。管理器同样还检查连接是否仍然激活，如果连接死亡了就把它从连接池删除。缺点是这样做增加了服务器端的负荷，因为对每个Connection都要创建一个会话。选择何种方式是应用的需求决定的。

第 5 章 JDBC API扩展

UXDB是一种可扩展的数据库系统。可以向数据库后端里增加自己的函数，这些函数可以供查询调用，甚至可以增加自己的数据类型。因为这是UXDB特有的功能，因此我们在Java 里面支持它们，同时还带着一套扩展的API。在标准驱动的核心里实际上使用了这些扩展来实现大对象等等。

5.1. 访问扩展

要获得某些扩展，需要使用org.UXDB.UXConnection类里的一些额外的方法，需要转换Driver.getConnection()的返回值。

```
Connection db = Driver.getConnection(url, username, password);
// ...
Fastpath fp = ((org.UXDB.UXConnection)db).getFastpathAPI();
```

5.1.1. 类org.UXDB.UXConnection

主要介绍了适用于获取UXDB的扩展的额外方法。

```
public Fastpath getFastpathAPI() throws SQLException
```

- 功能

为当前连接返回Fastpath API，主要用于大对象API。

- 返回值

一个可以用来访问UXDB后端里面的函数的Fastpath对象。

- 异常

在第一次初始化的时候Fastpath抛出的SQLException。

- 示例

```
import org.UXDB.fastpath.*;
...
Fastpath fp = ((org.UXDB.UXConnection)myconn).getFastpathAPI();
```

这里的myconn是一个已经打开的与UXDB的Connection。

```
public LargeObjectManager getLargeObjectAPI() throws SQLException
```

- 功能

为当前连接返回一个大对象API。

- 返回值

实现大对象API的LargeObject对象。

- 异常

在第一次初始化的时候LargeObject抛出的SQLException。

- 示例

```
import org.UXDB.largeobject.*;
...
LargeObjectManager lo = ((org.UXDB.UXConnection)myconn).getLargeObjectAPI();
```

这里的myconn是一个已经打开的与UXDB的Connection。

```
public void addDataType(String type, String name)
```

- 功能

为当前连接返回一个大对象API。

- 返回值

实现大对象API的LargeObject对象。

- 异常

在第一次初始化的时候LargeObject抛出的SQLException。

- 示例

```
import org.UXDB.largeobject.*;
...
LargeObjectManager lo = ((org.UXDB.UXConnection)myconn).getLargeObjectAPI();
```

这里的myconn是一个已经打开的与UXDB的Connection。

```
public void addDataType(String type, String name)
```

- 功能

允许客户端代码为UXDB中比较独特的数据类型加一个句柄。

通常，驱动器不认识的数据类型是由ResultSet.getObject()以一个UXObject实例的形式返回的。这个方法允许写一个扩展UXObject的类，然后告诉驱动该类型名字，以及要使用的类名字。这个方法的缺点是，每次建立新连接的时候，都要调用这个方法。

- 参数

表 5.1. addDataType参数说明

参数	类型	说明
type	String	类型
name	String	类名称

实现大对象API的LargeObject对象。

- 示例

```
...
((org.UXDB.UXConnection)myconn).addDataType("mytype","my.class.name");
```


...

这里的myconn是一个已经打开的与UXDB的Connection。做控制的类必须扩展org.UXDB.util.UXObject。

5.1.2. 类org.UXDB.Fastpath

```
public class Fastpath extends Object
```

```
java.lang.Object
```

```
|
```

```
+----org.UXDB.fastpath.Fastpath
```

Fastpath是一套存在于libpqC接口里的API，并且这个接口允许客户机器执行后端数据库的函数。大多数客户端代码不需要使用这个方法，但是还是提供这个方法，因为大对象API使用它。

要使用这个扩展，需要输入UXDB.fastpath包，如下：

```
import org.UXDB.fastpath.*;
```

然后在代码里，需要获取一个FastPath对象：

```
Fastpath fp = ((org.UXDB.UXConnection)conn).getFastpathAPI();
```

这样将返回一个实例，该实例与发出命令的数据库连接相关联。必须把Connection转换成org.UXDB.UXConnection，因为getFastpathAPI()是我们自己的方法之一，而不是JDBC的。一旦拥有了Fastpath实例，就可以使用fastpath()方法执行一个后端函数。

```
public Object fastpath(int fnid,boolean resulttype,FastpathArg args[]) throws
SQLException
```

- 功能

向UXDB后端发送一个函数调用。

- 参数

表 5.2. fastpath-fnid参数说明

参数	类型	说明
fnid	int	函数id
resulttype	boolean	如果结果为整数则为true，如果为其它则为false
args	FastpathArg	传递给fastpath的FastpathArguments

- 返回值

如果没有数据返回空(null)，如果结果为整数返回一个Integer，否则返回byte[]。

```
public Object fastpath(String name,boolean resulttype,FastpathArg args[]) throws
SQLException
```

- 功能

通过名字向UXDB后端发送一个函数调用。

函数名到函数id的映射必须存在，通常先调用`addfunction()`。这是调用函数的优选方法，因为函数id在不同版本的后端里是可能会改变的。

- 参数

表 5.3. `fastpath-name`参数说明

参数	类型	说明
name	String	函数名称
resulttype	boolean	如果结果为整数则为true，如果为其它则为false
args	FastpathArg	传递给fastpath的FastpathArguments

- 返回值

如果没有数据返回空(`null`)，如果结果为整数返回一个Integer，否则返回`byte[]`。

```
public int getInteger(String name, FastpathArg args[]) throws SQLException
```

- 功能

假设返回值是一个Integer。

- 参数

表 5.4. `getInteger`参数说明

参数	类型	说明
name	String	函数名称
args	FastpathArg	函数参数

- 返回值

Integer。

- 异常

如果发生了数据库访问错误或者没有结果抛出SQLException。

```
public byte[] getData(String name, FastpathArg args[]) throws SQLException
```

- 功能

假设返回值是二进制数据。

- 参数

表 5.5. `getData`参数说明

参数	类型	说明
name	String	函数名称
args	FastpathArg	函数参数

- 返回值

包含结果的byte[]数组。

- 异常

如果发生了数据库访问错误或者没有结果抛出SQLException。

```
public void addFunction(String name, int fnid)
```

addFunction向函数检索表里增加一个函数。用户应该使用addFunctions方法，因为这个方法基于一个查询，而不是OID硬代码。不能保证一个函数的OID是静态的，甚至运行在不同服务器的同版本的数据库也不能保证是静态的。

```
public void addFunctions(ResultSet rs) throws SQLException
```

参数是包含两个字段的ResultSet，字段1包含函数名，字段2是OID。

UXDB在UX_proc表里存储函数id和它们对应的名称，在查找时不是从该表里查询每个所需函数的名称，而是使用了一个Hashtable（散列表）。同样，只有需要的函数的名称才放到这个表里，以保证连接速度。

org.UXDB.LargeObject 类在启动时执行一个查询，并且把返回的ResultSet传递给这里提到的addFunctions()方法。一旦这些工作完成，LargeObjectAPI就用名称引用函数。

目前可以手动的把它们转换成OID，但随着开发的进行这些可能被修改，所以这样做是防止未来可能出现的任何不受保障的手段。

调用完这个方法后记得用close()关闭ResultSet。

```
public int getID(String name) throws SQLException
```

getID返回与函数名关联的函数id，如果还没有对这个函数名调用addFunction()或addFunctions()，那么会抛出SQLException。

5.1.3. 类org.UXDB.fastpath.FastpathArg

```
public class FastpathArg extends Object
```

```
java.lang.Object
```

```
|
```

```
+----org.UXDB.fastpath.FastpathArg
```

每个fastpath调用都需要一个参数列表，其数目和类型取决于被调用的函数。这个类实现了提供这个功能所需要的方法。

```
public FastpathArg(int value)
```

- 功能

构造一个包含一个整数值的参数。

- 参数

value: 待设置的整数值。

```
public FastpathArg(byte bytes[])
```

- 功能

构造一个包含一个字节数组的参数。

- 参数

bytes: 要保存的数组。

```
public FastpathArg(byte buf[], int off, int len)
```

- 功能

构造一个包含一个数组的一部分的参数。

- 参数

表 5.6. FastpathArg参数说明

参数	类型	说明
buf	byte	原数组
off	int	数组内的偏移量
len	int	要包括的数据的长度

```
public FastpathArg(String s)
```

构造一个字符串组成的参数。

5.2. 几何数据类型

UXDB有一个往表里存储几何特性的数据类型集。范围包括点，线，和多边形。通过org. UXDB. geometric包在Java里支持这些类型。它包括扩展了org. UXDB. util. UXObject类的类。

5.2.1. Class org. UXDB. geometric. UXbox

```
java.lang.Object
|
+----org. UXDB. util. UXObject
|
+----org. UXDB. geometric. UXbox
```

```
public class UXbox extends UXObject implements Serializable, Cloneable
```

- 功能

在UXDB里表示盒子(box)数据类型。

- 变量

```
public UXpoint point[]
```

这些是盒子的两个对角点。

- 构造器

- `public UXbox(double x1, double y1, double x2, double y2)`

表 5.7. UXbox-double参数说明

参数	类型	说明
x1	double	第一个x坐标
y1	double	第一个y坐标
x2	double	第二个x坐标
y2	double	第二个y坐标

- **public UXbox(UXpoint p1, UXpoint p2)**

表 5.8. UXbox-UXpoint参数说明

参数	类型	说明
p1	UXpoint	第一个点
p2	UXpoint	第二个点

- **public UXbox(String s) throws SQLException**

参数: s表示UXDB语法里的盒子定义。

异常: 如果定义非法会抛出异常SQLException。

- **public UXbox()**

必须构造的方法。

public void setValue(String value) throws SQLException

- 功能

setValue设置这个对象的值。它应该被重载，但是仍然被子类调用，覆盖类UXObject里的setValue。

- 参数

value: 一个代表对象值的字符串。

- 异常

如果此数值对这个类型而言是非法的则会抛出异常SQLException。

public boolean equals(Object obj)

- 功能

比较两个盒子，覆盖类UXObject里的equals。

- 参数

obj: 要比较的对象。

- 返回

如果两个盒子相等返回true。

```
public Object clone()
```

必须覆盖这个方法以允许对象被克隆，覆盖类UXObject里的clone。

```
public String getValue()
```

覆盖类UXObject里的getValue，返回值是UXDB句法需要的UXbox。

5.2.2. Class org.UXDB.geometric.UXcircle

```
java.lang.Object
|
+----org.UXDB.util.UXObject
|
+----org.UXDB.geometric.UXcircle
```

```
public class UXbox extends UXObject implements Serializable, Cloneable
```

- 功能

这个类代表 UXDB 的圆数据类型，由一个点和一个半径组成。

- 变量

圆心: public UXpoint center; 半径: double radius

- 构造器

- public UXcircle(double x, double y, double r)

表 5.9. UXcircle-double参数说明

参数	类型	说明
x	double	圆心坐标
y	double	圆心坐标
r	double	圆半径

- public UXcircle(String s) throws SQLException

参数: s表示UXDB里语法定义的圆。

异常: 如果转换失败会抛出异常SQLException。

- public UXcircle()

这个构造方法被驱动器使用。

```
public void setValue(String s) throws SQLException
```

- 功能

覆盖类UXObject里的setValue。

- 参数

s: 用UXDB的语法定义的圆。

- 异常

如果转换失败会抛出异常SQLException。

```
public Object clone()
```

必须重载这个方法以便允许对象被克隆，覆盖类UXobject里的clone。

```
public String getValue()
```

覆盖UXobject里的getValue，返回值是UXDB语法里的UXcircle字符串。

5.2.3. Class org.UXDB.geometric.UXline

```
java.lang.Object
|
+----org.UXDB.util.UXobject
|
+----org.UXDB.geometric.UXline
```

```
public class UXline extends UXobject implements Serializable,Cloneable
```

- 功能

这个类实现由两个点组成的线。目前线还没有在后端实现，但这个类保证在后端实现后即可使用。

- 变量

```
public UXpoint point[]
```

这是两个点。

- 构造器

- `public UXline(double x1,double y1,double x2,double y2)`

表 5.10. UXline-double参数说明

参数	类型	说明
x1	double	第一个点的x坐标
y1	double	第一个点的y坐标
x2	double	第二个点的x坐标
y2	double	第二个点的y坐标

- `public UXline(UXpoint p1, UXpoint p2)`

表 5.11. UXline-UXpoint参数说明

参数	类型	说明
p1	double	第一个点
p2	double	第二个点

- `public UXline(String s) throws SQLException`

参数: s表示UXDB语法定义的点。

异常: 当发生转换错误会抛出异常SQLException。

- `public UXline()`

驱动需要。

`public void setValue(String s) throws SQLException`

- 功能

覆盖类UXObject里的setValue。

- 参数

s: UXDB里语法的线段的定义。

- 异常

如果转换失败会抛出异常SQLException。

`public boolean equals(Object obj)`

- 功能

覆盖类UXObject里的equals。

- 参数

obj: 要对比的对象。

- 返回值

如果两个圆相同返回true。

`public Object clone()`

必须重载这个方法以便允许对象被克隆, 覆盖类UXObject里的clone。

`public String getValue()`

覆盖UXObject里的getValue, 返回值是UXDB语法里的UXcircle字符串。

5.2.4. Class org.UXDB.geometric.UXlseg

java.lang.Object

|
+----org.UXDB.util.UXObject

|
+----org.UXDB.geometric.UXlseg

`public class UXlseg extends UXObject implements Serializable, Cloneable`

- 功能

实现了一条包含两个点的线段。

- 变量

```
public UXpoint point[]
```

这是两个点。

- 构造器

- `public UXlseg(double x1, double y1, double x2, double y2)`

表 5.12. UXlseg-double参数说明

参数	类型	说明
x1	double	第一个点的x坐标
y1	double	第一个点的y坐标
x2	double	第二个点的x坐标
y2	double	第二个点的y坐标

- `public UXlseg(UXpoint p1, UXpoint p2)`

表 5.13. UXlseg-UXpoint参数说明

参数	类型	说明
p1	double	第一个点
p2	double	第二个点

- `public UXlseg(String s) throws SQLException`

参数: s表示UXDB里语法对线段定义的字串。

异常: 当发生转换错误会抛出异常SQLException。

- `public UXlseg()`

驱动需要。

```
public void setValue(String s) throws SQLException
```

- 功能

覆盖类UXObject里的setValue。

- 参数

s: UXDB里语法对线段定义的字串。

- 异常

在发生转换错误时会抛出异常SQLException。

```
public boolean equals(Object obj)
```

- 功能

覆盖类UXObject里的equals。

- 参数

obj: 要比较的对象。

- 返回值

如果两条线段相等true。

public Object clone()

必须覆盖这个方法以便允许这个对象被克隆，覆盖类UXObject里的clone。

public String getValue()

覆盖类UXObject里的getValue。

5.2.5. Class org.UXDB.geometric.UXpath

```
java.lang.Object
|
+----org.UXDB.util.UXObject
|
+----org.UXDB.geometric.UXpath
```

public class UXpath extends UXObject implements Serializable,Cloneable

- 功能

这是路径(可以为封闭的多线段图形)的实现。

- 变量

public boolean open

如果路径开放时为true，封闭时为false。

public UXpoint points[]

定义路径的点。

- 构造器

- **public UXpath(UXpoint points[],boolean open)**

表 5.14. UXpath-UXpoint参数说明

参数	类型	说明
points	UXpoint	定义路径的UXpoints
open	boolean	如果路径是开放的为true，封闭为false

- **public UXpath()**

驱动需要。

- **public UXpath(String s) throws SQLException**

参数: s表示UXDB的语法定义的路径。

异常: 当发生转换错误会抛出异常SQLException。

public void setValue(String s) throws SQLException

- 功能

覆盖类UXObject里的setValue。

- 参数

s: UXDB的语法定义的路径的字符串。

- 异常

在发生转换错误时会抛出异常SQLException。

public boolean equals(Object obj)

- 功能

覆盖类UXObject里的equals。

- 参数

obj: 要比较的对象。

- 返回值

如果两个路径相同返回true。

public Object clone()

必须覆盖这个方法以便允许这个对象被克隆, 覆盖类UXObject里的clone。

public String getValue()

覆盖类UXObject里的getValue。

public boolean isOpen()

返回值是如果路径是开放的这个方法返回true。

public boolean isClosed()

返回值是如果路径是封闭的这个方法返回true。

public void closePath()

标记路径为封闭。

public void openPath()

标记路径为开放。

5.2.6. Class org.UXDB.geometric.UXpoint

java.lang.Object

```

|
+----org.UXDB.util.UXObject
|
+----org.UXDB.geometric.UXpoint

```

```
public class UXpoint extends UXobject implements Serializable,Cloneable
```

- 功能

这个类实现了java.awt.Point的一个版本，但用double表示参数，对应于UXDB里的point数据类型。

- 构造器

- `public UXpoint(double x, double y)`

表 5.15. UXpoint-double参数说明

参数	类型	说明
x	double	点的x坐标
y	double	点的y坐标

- `public UXpoint(String value) throws SQLException`

这个方法主要从其他几何类型调用 -- 当一个点嵌入它们的定义中时。

参数：value表示UXDB语法定义的点。

- `public UXpoint()`

驱动需要。

```
public void setValue(String s) throws SQLException
```

- 功能

覆盖类UXObject里的setValue。

- 参数

s: UXDB语法定义的点。

- 异常

在转换失败时会抛出异常SQLException。

```
public boolean equals(Object obj)
```

- 功能

覆盖类UXObject里的equals。

- 参数

obj: 要比较的对象。

- 返回值

如果两个对象相同返回true。

```
public Object clone()
```

必须覆盖这个方法以便允许这个对象被克隆，覆盖类UXObject里的clone。

```
public String getValue()
```

覆盖类UXObject里的getValue，返回值是UXDB里语法UXpoint的表示。

```
public void translate(int x, int y)
```

- 功能

对点做指定数量的转换(位移)。

- 参数

表 5.16. translate-int参数说明

参数	类型	说明
x	int	向x轴增加的整型数量
y	int	向y轴增加的整型数量

```
public void translate(double x, double y)
```

- 功能

对点做指定数量的转换(位移)。

- 参数

表 5.17. translate-double参数说明

参数	类型	说明
x	double	向x轴增加的双精度型数量
y	double	向y轴增加的双精度型数量

```
public void move(int x, int y)
```

- 功能

把点移到指定坐标。

- 参数

表 5.18. move-int参数说明

参数	类型	说明
x	int	整数坐标
y	int	整数坐标

```
public void move(double x, double y)
```

- 功能

把点移到指定坐标。

- 参数

表 5.19. move-double参数说明

参数	类型	说明
x	double	双精度坐标
y	double	双精度坐标

```
public void setLocation(int x, int y)
```

- 功能

把点移到指定坐标。

- 参数

表 5.20. setLocation-int参数说明

参数	类型	说明
x	int	整数坐标
y	int	整数坐标

```
public void setLocation(Point p)
```

- 功能

把点移到指定坐标。

- 参数

p: 移动的目的点。

5.2.7. Class org.UXDB.geometric.UXpolygon

```
java.lang.Object
|
+----org.UXDB.util.UXObject
|
+----org.UXDB.geometric.UXpolygon
```

```
public class UXpolygon extends UXobject implements Serializable, Cloneable
```

- 功能

这个类在UXDB里实现了polygon（多边形）数据类型。

- 构造器

- `public UXpolygon(UXpoint points[])`

使用一个UXpoints数组创建一个多边形。

参数: point表示定义多边形polygon的点。

- `public UXpolygon(String s) throws SQLException`

参数: s表示用UXDB语法定义的多边形。

异常: 在转换失败时会抛出异常SQLException。

- `public UXpolygon()`

驱动需要。

`public void setValue(String s) throws SQLException`

- 功能

覆盖类UXObject里的setValue。

- 参数

s: 用UXDB语法定义的多边形。

- 异常

在转换失败时会抛出异常SQLException。

`public boolean equals(Object obj)`

- 功能

覆盖类UXObject里的equals。

- 参数

obj: 要比较的对象。

- 返回值

如果两个对象相同返回true。

`public Object clone()`

必须覆盖这个方法以便允许这个对象被克隆, 覆盖类UXObject里的clone。

`public String getValue()`

覆盖类UXObject里的getValue, 返回值是UXDB里语法表示的UXpolygon。

5.3. 大对象

标准的JDBC规范里也支持大对象。但是, 那个接口有一些限制, 而UXDB提供的API允许对对象内容的随机访问, 就像是一个本地文件。

org.UXDB.largeobject包为Java提供了libpq C接口的大对象API。它包含两个类: LargeObjectManager处理创建, 打开和删除大对象的任务; LargeObject处理独立的对象。

5.3.1. 类org.UXDB.largeobject.LargeObject

```
public class LargeObject extends Object
```

```
java.lang.Object
```

```
|
```

```
+----org.UXDB.largeobject.LargeObject
```

- 功能

这个类实现UXDB的大对象接口。

它提供运行接口的基本的方法，另外还有一对方法为此对象提供InputStream和OutputStream类。

通常，客户端代码将使用在BLOB里的方法访问大对象。但是，有时候需要低层次的大对象访问方法，那是JDBC规范还不支持的。

- 变量

```
public static final int SEEK_SET
```

标识从一个文件的开头进行一次搜索。

```
public static final int SEEK_CUR
```

标识从当前位置进行一次搜索。

```
public static final int SEEK_END
```

标识从一个文件的结尾进行一次搜索。

```
public int getOID()
```

返回这个LargeObject的OID。

```
public void close() throws SQLException
```

这个方法关闭对象，在调用这个方法后不能调用这个对象里的任何方法。

```
public byte[] read(int len) throws SQLException
```

从对象读取一些数据，并且做为byte[]数组返回。

```
public int read(byte buf[],int off,int len) throws SQLException
```

- 功能

从对象读取一些数据到现有数组。

- 参数

表 5.21. read参数说明

参数	类型	说明
buf	byte	目的数组

参数	类型	说明
off	int	数组内偏移量
len	int	读取的字节数

```
public void write(byte buf[]) throws SQLException
```

向对象里写入一个数组。

```
public int write(byte buf[],int off,int len) throws SQLException
```

- 功能

从数组里写一些数据到对象。

- 参数

表 5.22. write参数说明

参数	类型	说明
buf	byte	目的数组
off	int	数组内偏移量
len	int	写入字节数

5.3.2. 类org.UXDB.largeobject.LargeObjectManager

```
public class LargeObjectManager extends Object
```

```
java.lang.Object
```

```
|
```

```
+----org.UXDB.largeobject.LargeObjectManager
```

这个类型实现了UXDB的大对象接口。它提供了允许客户代码从数据库里创建，打开和删除大对象的方法。在打开一个对象时，返回一个UXDB.largeobject.LargeObject的实例，然后它的方法就可以访问该对象。

这个类只能由 org.UXDB.UXConnection 创建 要访问这个类，使用下面的代码片段：

```
import org.UXDB.largeobject.*;
Connection conn;
LargeObjectManager lobj;
// ... 打开一个连接的代码 ...
lobj = ((org.UXDB.UXConnection)myconn).getLargeObjectAPI();
```

通常，客户代码要使用BLOB方法访问大对象。但是，有时候需要低层次的大对象访问方法，那是JDBC规范还不支持的。

变量

- public static final int WRITE

这个模式表明要写入大对象。

- public static final int READ

这个模式表明要读取大对象。

- `public static final int READWRITE`

这个模式是缺省的，表明要对大对象进行读和写的操作。

```
public LargeObject open(int oid) throws SQLException
```

这打开一个现有的大对象，以其OID为基础。这个方法假设需要READ和WRITE访问模式（缺省模式）。

```
public LargeObject open(int oid,int mode) throws SQLException
```

这个方法以其OID为基础打开一个现有的大对象，并且允许设置访问模式。

```
public int create() throws SQLException
```

这个方法创建一个大对象，返回它的OID。把新创建的大对象模式设为缺省的READWRITE。

```
public int create(int mode) throws SQLException
```

这个方法创建一个大对象，返回它的OID。并设置访问模式。

```
public void delete(int oid) throws SQLException
```

这个方法删除一个大对象。

```
public void unlink(int oid) throws SQLException
```

这个方法删除一个大对象。这个方法等同于delete方法，并且作为类似使用“unlink”的C API出现。

第 6 章 连接池和数据源

6.1. 概述

JDBC API为连接池提供了一个客户端和一个服务器端的接口。客户端接口是 `javax.sql.DataSource`，通常就是应用代码用来请求一个缓冲了的数据库连接的东西。服务器接口是 `javax.sql.ConnectionPoolDataSource`，通常是大多数应用服务器和UXDB JDBC驱动之间使用的接口。

在应用服务器环境里，应用服务器配置通常将指向UXDB `ConnectionPoolDataSource`实现，而应用组件代码将通常要求一个由应用服务器实现的`DataSource`（不是UXDB）。

在一个没有应用服务器的环境里，UXDB提供了两种应用可以直接使用的`DataSource`的实现。一种实现执行连接池，另外一种只是简单的通过`DataSource`接口提供访问数据库的连接，而不使用任何缓冲池。同样，这些实现不应该在一个应用服务器环境里使用，除非应用服务器不支持`ConnectionPoolDataSource`接口。

6.2. ConnectionPoolDataSource

UXDB包含了一个用于JDBC 2的`ConnectionPoolDataSource`应用服务器的实现，以及一个用于JDBC 3的实现。

表 6.1. ConnectionPoolDataSource实现

JDBC	实现类
2	<code>org.UXDB.jdbc2.optional.ConnectionPool</code>
3	<code>org.UXDB.jdbc3.Jdbc3ConnectionPool</code>

两种实现都使用了同样的配置模式。JDBC要求一个`ConnectionPoolDataSource`通过JavaBean属性来实现，因此每个这样的属性都存在获取和设置方法。

表 6.2. ConnectionPoolDataSource 配置属性

属性	类型	描述
<code>serverName</code>	<code>String</code>	UXDB 数据库服务器主机名
<code>databaseName</code>	<code>String</code>	UXDB 数据库名
<code>portNumber</code>	<code>int</code>	UXDB 数据库服务器监听的TCP/IP 端口（为 0 则使用缺省端口）
<code>user</code>	<code>String</code>	用来进行数据库连接的用户
<code>password</code>	<code>String</code>	用来进行数据库连接的口令
<code>defaultAutoCommit</code>	<code>boolean</code>	提交给调用者的时候连接是应该打开 <code>autoCommit</code> 还是应该关闭。缺省是 <code>false</code> ，关闭 <code>autoCommit</code>

许多应用服务器使用属性风格的语法来配置这些属性，因此把属性当作一块文本输入应该并不罕见。如果应用服务器提供了单块的区域输入所有属性，如下所示。

```

serverName=localhost
databaseName=test
user=testuser
password=testpassword

```

或者使用分号而不是换行做分隔符，如下所示。

```
serverName=localhost;databaseName=test;user=testuser;password=testpassword
```

6.3. DataSource

UXDB包含两个用于JDBC 2的DataSource，还有两个用于JDBC 3的，如表 6.3 “DataSource实现”所示。连接池的实现在客户端调用close方法的时候实际上并不关闭连接，而是把连接返回到一个可用连接的连接池中给其它客户端使用。这样就避免了任何重复打开和关闭连接造成的开销，并且允许大量的客户端分享相对较少的数据库连接。

这里提供的连接池数据源实现并不是完全的。比如连接在池本身关闭之前绝对不会关闭，而且也没有办法缩小连接池。同样，非缺省配置的用户连接请求无法如池。许多应用服务器提供更加高级的连接池特性，并且使用的是ConnectionPoolDataSource实现。

表 6.3. DataSource实现

JDBC	连接池	实现类
2	否	org.UXDB.jdbc2.optional.SimpleDataSource
2	是	org.UXDB.jdbc2.optional.PoolingDataSource
3	否	org.UXDB.jdbc3.Jdbc3SimpleDataSource
3	是	org.UXDB.jdbc3.Jdbc3PoolingDataSource

所有实现使用同样的配置模式。JDBC要求DataSource通过JavaBean属性配置，如表 6.4 “DataSource配置属性”所示。因此针对这个属性都有获取和设置属性的方法。

表 6.4. DataSource配置属性

属性	类型	描述
serverName	String	UXDB数据库服务器主机名
databaseName	String	UXDB数据库名
portNumber	int	UXDB数据库服务器监听的TCP端口（或者0表示使用缺省端口）
user	String	用来进行数据库连接的用户
password	String	用来进行数据库连接的口令

连接池实现要求一些额外的配置属性，如下所示。

表 6.5. 额外的连接池DataSource配置属性

属性	类型	描述
dataSourceName	String	每一个连接池DataSource必须有一个唯一的名字
initialConnections	int	连接池初始化的时候要创建的数据库连接数目

属性	类型	描述
maxConnections	int	允许打开的最大数据库连接个数。如果请求了更多的连接，那么调用者将挂起，直到有一个连接返回给连接池

提供了一个使用连接池DataSource的典型应用的示例。

初始化连接池DataSource，如下所示。

```
Jdbc3PoolingDataSource source = new Jdbc3PoolingDataSource();
source.setDataSourceName("A Data Source");
source.setServerName("localhost");
source.setDatabaseName("test");
source.setUser("testuser");
source.setPassword("testpassword");
source.setMaxConnections(10);
```

然后使用来自连接池的代码，如下所示。请注意最终要关闭连接是非常关键的，否则这个池子就会“泄漏”连接，最后把所有客户端都锁在外面。

```
Connection con = null;
try {
    con = source.getConnection();
    // 使用连接
} catch (SQLException e) {
    // 记录错误
} finally {
    if (con != null) {
        try { con.close(); } catch (SQLException e) {}
    }
};
```

6.4. 数据源和JNDI

所有ConnectionPoolDataSource和DataSource实现都可以存储在JNDI里。在非连接池的实现中，每次从JNDI中检索对象都将创建一个新的实例，带有和存储的实例同样的设置。对于连接池实现而言，同一个实例是在可得的情况下检索出来的（也就是说，没有其它JVM从JNDI中检索连接池），否则就创建同样设置的新的实例。

在应用服务器环境，通常是应用服务器的DataSource实例将存储在JNDI中，而不是UXDB ConnectionPoolDataSource的实现。在应用服务器环境，应用可以在JNDI中存储DataSource，这样不用制作一个指向DataSource的引用提供给所有可能需要的应用组件使用它。

示例

初始化连接池DataSource并且加到JND，如下所示。

```
Jdbc3PoolingDataSource source = new Jdbc3PoolingDataSource();
source.setDataSourceName("A Data Source");
source.setServerName("localhost");
source.setDatabaseName("test");
source.setUser("testuser");
source.setPassword("testpassword");
```

```
source.setMaxConnections(10);  
new InitialContext().rebind("DataSource", source);
```

然后使用来自连接池的代码，如下所示。

```
Connection con = null;  
try {  
    DataSource source = (DataSource)new InitialContext().lookup("DataSource");  
    con = source.getConnection();  
    // 使用连接  
} catch (SQLException e) {  
    // 记录错误  
} catch (NamingException e) {  
    // 在 JNDI 里没有找到 DataSource  
} finally {  
    if (con != null) {  
        try { con.close(); } catch (SQLException e) {}  
    }  
}
```

第 7 章 JDBC Wrapper

主要功能：

- 支持多cn节点轮询和直连worker功能。
- 支持部分select, insert, update, delete操作路由到具体worker节点。
- 如果打开直连worker开关，直接路由到worker节点；如果关闭，轮询cn节点。
- 支持数据库集群连接池功能。
- 支持可用worker列表。
- 支持连接池懒加载，支持限制最大连接数。

7.1. 获取jdbc wrapper

请联系优炫技术人员获取最新版本的jdbc wrapper工具驱动包。

7.2. 接口使用

7.2.1. 搭建mpp环境

1. 节点分布

master: 192.168.120.100:5432

cn1: 192.168.120.100:5433

worker1: 192.168.120.100:1234

worker2: 192.168.120.100:1235

在master节点设置分片规则，分为两片，每个worker占用一片，并搭建mpp集群环境。

2. 创建表

在master节点上，创建一个测试表。

```
create table test(id int,name text); //创建普通表
#insert into test (id,name) select generate_series(1,10000),'test'; //添加10000条数据
select create_distributed_table('test','id');//以id字段分片
```

7.2.2. 多CN轮询接口用例（mpp场景）

1. 初始化集群连接。

```
//创建代理conn
//是否懒加载
private static boolean isLazyInitialize=false;
//是否轮询
```

```
private static boolean pollingMode=true;
//当前是否是mpp环境
private static boolean mppInstall=true;
ShardingConnection conn = new ShardingConnection(isLazyInitialize, pollingMode,mppInstall);

//添加master节点信息, 参数分别为驱动, 数据库url, 用户名, 密码
conn.setMasterCNDDataSource
("com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:5432/mydb", "uxdb", "123456");

//添加cn节点信息, 参数分别为驱动, 数据库url, 用户名, 密码
conn.putCNDatasources
("com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:5433/mydb", "uxdb", "123456");

//worker节点初始化名称; 取值必须包含 worker
String worker1Key="worker1";
String worker2Key="worker2";

//添加worker节点信息, 参数分别为驱动, 数据库url, 用户名, 密码
conn.putWorkerDataSources
(worker1Key, "com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:1234/mydb", "uxdb",
"123456");
conn.putWorkerDataSources
(worker2Key, "com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:1235/mydb", "uxdb",
"123456");
//连接池及mpp元数据初始化
conn.initConnection();
```

2. 关闭直连worker开关。

```
//关闭直连worker开关
conn.setDirectWorker(false);
```

3. 创建statement, 执行SQL。

```
ResultSet rs = null;
UXPreparedStatement statement = null;
UXPreparedStatement statement2 = null;
UXPreparedStatement statement3 = null;
UXPreparedStatement statement4 = null;

//注意insert语句格式, 必须包含表列名, 分片字段
statement2 = conn.prepareStatement("insert into test (ID,NAME) values (?,?)");
statement2.setInt(1, 1);
statement2.setString(2, "bar");
int count = statement2.executeUpdate();
System.out.println("添加数据" + count);

//注意update语句格式, 必须包含分片字段
statement4 = conn.prepareStatement("update test set name = ? where id=?");
statement4.setInt(2, 1);
statement4.setString(1, "xxxx");
int count2 = statement4.executeUpdate();
System.out.println("修改数据" + count2);

//注意select 语句格式, 必须包含分片字段
```



```
statement = conn.prepareStatement("select * from test where id = ?");
statement.setInt(1, 1);
rs = statement.executeQuery();
while (rs.next()) {
    System.out.print("查询数据---");
    System.out.print("id=" + rs.getString("id") + " ,");
    System.out.print("name=" + rs.getString("name") + "");
    System.out.println();
}

//注意delete语句格式，必须包含分片字段
statement3 = conn.prepareStatement("delete from test where id = ?");
statement3.setInt(1, 1);
int count1 = statement3.executeUpdate();
System.out.println("删除数据" + count1);
conn.commit();
```

7.2.3. worker接口用例（mpp场景）

1. 初始化集群连接。

```
//创建代理conn
//是否懒加载
private static boolean isLazyInitialize=false;

//是否轮询一直连worker,设置false
private static boolean pollingMode=false;

//当前是否是mpp环境
private static boolean mppInstall=true;
ShardingConnection conn = new ShardingConnection(isLazyInitialize, pollingMode,mppInstall);

//添加master节点信息,参数分别为驱动,数据库url,用户名,密码
conn.setMasterCNDataSource
("com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:5432/mydb", "uxdb", "123456");

//添加cn节点信息,参数分别为驱动,数据库url,用户名,密码
conn.putCNDataSources
("com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:5433/mydb", "uxdb", "123456");

//worker节点初始化名称;取值必须包含 worker
String worker1Key="worker1";
String worker2Key="worker2";

//添加worker节点信息,参数分别为驱动,数据库url,用户名,密码
conn.putWorkerDataSources
(worker1Key, "com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:1234/mydb", "uxdb",
"123456");
conn.putWorkerDataSources
(worker2Key, "com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:1235/mydb", "uxdb",
"123456");

//连接池及mpp元数据初始化
conn.initConnection();
```

2. 开启直连worker开关，设置worker列表。

```
//开启直连worker开关
conn.setDirectWorker(true);
List > String > directWorkerList = new ArrayList<>();
//规定那些worker可以路由，取值范围在worker节点初始化时已经说明
directWorkerList.add("worker1");
directWorkerList.add("worker2");
conn.setDirectWorkerList(directWorkerList);
```

3. 创建statement，执行SQL。

```
ResultSet rs = null;
UXPreparedStatement statement = null;
UXPreparedStatement statement2 = null;
UXPreparedStatement statement3 = null;
UXPreparedStatement statement4 = null;

//注意insert语句格式，必须包含表列名，分片字段
statement2 = conn.prepareStatement("insert into test (ID,NAME) values (?,?)");
statement2.setInt(1, 1);
statement2.setString(2, "bar");
int count = statement2.executeUpdate();
System.out.println("添加数据" + count);

//注意update语句格式，必须包含分片字段
statement4 = conn.prepareStatement("update test set name = ? where id=?");
statement4.setInt(2, 1);
statement4.setString(1, "xxxx");
int count2 = statement4.executeUpdate();
System.out.println("修改数据" + count2);

//注意select 语句格式，必须包含分片字段
statement = conn.prepareStatement("select * from test where id = ?");
statement.setInt(1, 1);
rs = statement.executeQuery();
while (rs.next()) {
    System.out.print("查询数据---");
    System.out.print("id=" + rs.getString("id") + " ,");
    System.out.print("name=" + rs.getString("name") + "");
    System.out.println();
}

//注意delete语句格式，必须包含分片字段
statement3 = conn.prepareStatement("delete from test where id = ?");
statement3.setInt(1, 1);
int count1 = statement3.executeUpdate();
System.out.println("删除数据" + count1);
conn.commit();
```

7.2.4. 普通jdbc连接池用例（非mpp场景）

1. 初始化集群连接。

```
//建立数据库对象
//初始化连接数
private static int initialSize=2;

//最大连接数
private static int maxActive=5;

//是否是公平锁
private static boolean lockFair=false;

//是否懒加载
private static boolean isLazyInitialize=true;

//是否轮询一直连worker,设置false
private static boolean pollingMode=false;

//当前是否是mpp环境
private static boolean mppInstall=false;
ShardingDataSource shardingDataSource = new
    ShardingDataSource(initialSize, maxActive, lockFair, true, false,false);

shardingDataSource.setMasterCNDDataSource
    ("com.uxsino.uxdb.Driver", "jdbc:uxdb://192.168.120.100:5432/mydb", "uxdb", "123456");
//建立操作对象
Connection conn = shardingDataSource.getConnection();
```

2. 创建statement，执行SQL。

```
ResultSet rs = null;
UXPreparedStatement statement = null;
UXPreparedStatement statement2 = null;
UXPreparedStatement statement3 = null;
UXPreparedStatement statement4 = null;

//注意insert语句格式，必须包含表列名，分片字段
statement2 = conn.prepareStatement("insert into test (ID,NAME) values (?,?)");
statement2.setInt(1, 1);
statement2.setString(2, "bar");
int count = statement2.executeUpdate();
System.out.println("添加数据" + count);

//注意update语句格式，必须包含分片字段
statement4 = conn.prepareStatement("update test set name = ? where id=?");
statement4.setInt(2, 1);
statement4.setString(1, "xxxx");
int count2 = statement4.executeUpdate();
System.out.println("修改数据" + count2);

//注意select 语句格式，必须包含分片字段
statement = conn.prepareStatement("select * from test where id = ?");
statement.setInt(1, 1);
rs = statement.executeQuery();
```

```

while (rs.next()) {
    System.out.print("查询数据---");
    System.out.print("id=" + rs.getString("id") + " ,");
    System.out.print("name=" + rs.getString("name") + "");
    System.out.println();
}

//注意delete语句格式，必须包含分片字段
statement3 = conn.prepareStatement("delete from test where id = ?");
statement3.setInt(1, 1);
int count1 = statement3.executeUpdate();
System.out.println("删除数据" + count1);
conn.commit();

```

7.3. 使用benchmark工具

请联系优炫技术人员获取jdbc wrapper版本的benchmark工具。

7.3.1. 搭建mpp环境

以1个master节点，3个cn节点和3个worker节点为例。

表 7.1. 服务器及节点分布

搭建环境	IP端口	节点功能	是否需要同步元数据，使其成为cn
master	20.0.0.1:10001	master，存储完整数据及元数据	
worker1	20.0.0.2:10001	worker，存储参照表和每个分片表其中一个分片	
worker2	20.0.0.3:10001	worker，存储参照表和每个分片表其中一个分片	
worker3	20.0.0.4:10001	worker，存储参照表和每个分片表其中一个分片	
cn1	20.0.0.2:10002		是
cn2	20.0.0.3:10002		是
cn3	20.0.0.4:10002		是
Benchmark工具	20.0.0.1	跑benchmark工具	

数据分布：1000仓库100GB数据，3个分片，每个worker节点1个分片。

7.3.2. 相关配置文件

master节点信息及tpcc相关参数配置文件：uxdb.pg

db=postgres //数据库类型，不需要更改
 driver=com.uxsino.uxdb.Driver //驱动，不需要更改

```

conn=jdbc:uxdb://20.0.0.1:10001/uxdb //数据库连接字符串
user=uxdb //数据库用户名
password=123456 //如上用户密码

warehouses=1000 //仓库数量，每个warehouse大小大概是100MB,建议将数据库的大小设置为服务器物理内存的2-5倍
loadWorkers=100 //用于在数据库中初始化数据的加载进程数量
terminals=15 //终端数，即并发客户端数量，通常设置为CPU线程总数的2~6倍
runTxnsPerTerminal=0//每个终端（terminal）运行的固定事务数量，该参数配置为非0值时，下面的runMins参数必须设置为0
runMins=1 //要测试的整体时间，单位为分钟该值设置为非0值时，runTxnsPerTerminal 参数必须设置为0。
limitTxnsPerMin=0 // 每分钟事务总数限制

//终端和仓库的绑定模式，设置为true时可以运行4.x兼容模式，意思为每个终端都有一个固定的仓库。设置为false时可以均匀的使用数据库整体配置。TPCC规定每个终端都必须有一个绑定的仓库，所以一般使用默认值true。
terminalWarehouseFixed=true

//下面五个值的总和必须等于100，默认值为：45, 43, 4, 4 & 4，与TPC-C测试定义的比例一致，实际操作过程中，可以调整比重来适应各种场景。
newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4

resultDirectory=my_result_%Y-%tm-%td_%tH%M%S //测试数据生成目录，默认无需修改，默认生成在run目录下面，名字形如my_result_xxxx的文件夹。
osCollectorScript=./misc/os_collector_linux.py //操作系统性能收集脚本，默认无需修改，需要操作系统具备有python环境
osCollectorInterval=1 //操作系统收集操作间隔，默认为1秒

//操作系统收集所对应的主机，如果对本机数据库进行测试，该参数保持注销即可，如果要对远程服务器进行测试，请填写用户名和主机名。
//osCollectorSSHAddr=user@dbhost

//osCollectorDevices=net_eth0 blk_sda //操作系统中被收集服务器的网卡名称和磁盘名称

cn/worker节点配置文件：dbfarm.pg

mastercn= jdbc:uxdb://20.0.0.1:10001/uxdb
#cn1=jdbc:uxdb://20.0.0.2:10002/uxdb
#cn2=jdbc:uxdb://20.0.0.3:10002/uxdb
#cn3=jdbc:uxdb://20.0.0.4:10002/uxdb
isLazyInitialize=true //是否开启懒加载：是
pollingMode=false //是否轮询：否
useDataSource=false //是否使用连接池：否
isdirectWorker=true //是否直连worker：是
availableWorkerList=worker1,worker2,worker3 //可用worker列表
worker1=jdbc:uxdb://20.0.0.2:10001/uxdb
worker2=jdbc:uxdb://20.0.0.3:10001/uxdb
worker3=jdbc:uxdb://20.0.0.4:10001/uxdb

```

提示

cn节点名称必须以cn开头，worker节点必须以worker开头。

isdirectWorker=false时需要配置cn节点，isdirectWorker=true时需要配置worker节点信息。

availableWorkerList=worker1,worker2,worker3，可用worker列表：分割worker信息的key值（isdirectWorker=true时需要配置）。

7.3.3. 原有工具包替换

1. jdbc包替换，在原有工具lib目录下加入jdbc-wrapper.jar包。

示例：cd /home/uxdb/software/benchmarksql-5.0-mpp/lib

```
[uxdb@localhost lib]$ ls
apache-log4j-extras-1.1.jar  log4j-1.2.17.jar  postgres
firebird                    ojdbc7.jar        uxdbjdbc-4.2.jar.bak
jdbc-wrapper.jar            oracle            uxdbjdbc-4.2.jar-old
```

2. 获取jdbc wrapper版本benchmark工具，确保benchmark工具dist目录下的benchmarksql-5.0.jar是jdbc wrapper版本的工具包。

示例：cd /home/uxdb/software/benchmarksql-5.0-mpp/dist

```
[uxdb@localhost dist]$ ls
benchmarksql-5.0.jar  BenchmarkSQL-5.0.jar.bak  BenchmarkSQL-5.0.jar.bak35
[uxdb@localhost dist]$ pwd
/home/uxdb/software/benchmarksql-5.0-mpp/dist
```

7.3.4. 查看结果

进入benchmarksql工具下的run目录。

示例：cd /home/uxdb/software/benchmarksql-5.0-mpp/run

查看runBenchmark.sh启动类是否是TPCC，以及查看dbfarm.pg配置文件是否配置正确。

```
[uxdb@localhost run]$ cat runBenchmark.sh
#!/usr/bin/env bash

if [ $# -ne 1 ] ; then
    echo "usage: ${basename $0} PROPS_FILE" >&2
    exit 2
fi

SEQ_FILE="./jTPCC_run_seq.dat"
if [ ! -f "${SEQ_FILE}" ] ; then
    echo "0" > "${SEQ_FILE}"
fi
SEQ=$(expr $(cat "${SEQ_FILE}") + 1) || exit 1
echo "${SEQ}" > "${SEQ_FILE}"

source funcs.sh $1

setCP || exit 1

dbfarm=dbfarm.pg

master=master.pg
myOPTS="-Dprop=$1 -DrunID=${SEQ} -Ddbfarm=${dbfarm} -Dmaster=${master#}"

#myOPTS="-Dprop=$1 -DrunID=${SEQ} -Ddbfarm=${dbfarm}"
#java -cp "$myCP" $myOPTS client.jTPCCByRouteFilter
java -cp "$myCP" $myOPTS client.jTPCC
[uxdb@localhost run]$
```

配置正确，执行./runBenchmark.sh uxdb.pg，结果如下。

```
18:00:10,886 [Thread-10] INFO jTPCC : Term-00,
18:00:10,886 [Thread-10] INFO jTPCC : Term-00,
18:00:10,886 [Thread-10] INFO jTPCC : Term-00, Measured tpmC (NewOrders) = 1873.65
18:00:10,887 [Thread-10] INFO jTPCC : Term-00, Measured tpmTOTAL = 4146.6
18:00:10,887 [Thread-10] INFO jTPCC : Term-00, Session Start = 2020-02-04 17:59:10
18:00:10,887 [Thread-10] INFO jTPCC : Term-00, Session End = 2020-02-04 18:00:10
18:00:10,887 [Thread-10] INFO jTPCC : Term-00, Transaction Count = 4153
[uxdb@localhost run]$
```

第 8 章 高可用连接池

8.1. 概述

uxdbjdbc高可用连接池实现了事务级别连接复用。传统连接池中的连接都是相互独立的，为各自的线程提供独立的服务。高可用连接池改变了常规的限制，实现客户端无感知的为多个数据端提供服务的功能。

jdbc配置参数如下所示。

表 8.1. jdbc配置参数

参数	数据类型	默认值	描述	备注
maxActive	整数	5	设置连接池中最大db连接数	参数enableEnhance配置为false时不起作用
initialSize	整数	2	设置连接池中初始db连接数量	参数enableEnhance配置为false时不起作用
terminalStep	整数	10	当客户端数量超出10倍的maxActive后，超出的部分的步长。每超出terminalStep个客户端连接(即一个步长)会新建一定数量(由physicsStep控制)的db连接	参数enableEnhance配置为false时不起作用
physicsStep	整数	1	当客户端数量超出10倍的maxActive后，每超出一个步长内创建的db连接数量。每超出若干个客户端连接会新建physicsStep个db连接	参数enableEnhance配置为false时不起作用
enableEnhance	枚举: true \false	false	设置是否启用com.uxsino.uxdb.EnhanceDriver驱动(是否启用连接池)	
hold	枚举: true \false	false	设置是否启用常驻连接，true常驻连接不爭抢，false常驻连接爭抢	tpcc-benchmarksql测试专用
initInstanceSize	整数	2	设置rac集群的实例个数	tpcc-benchmarksql测试专用
paramVal	整数	null	用于计算targetInstance，依据业务定制分发参数	tpcc-benchmarksql测试专用
enableHA	枚举: true \false	false	是否启用haproxy	tpcc-benchmarksql测试专用

8.2. 用法示例

8.2.1. 连接池使用示例

- 方法一：在url中添加参数。

```
jdbc:uxdb://192.71.1.35:5432/uxdb?  
maxActive=100&initialSize=10&terminalStep=20&physicsStep=1&enableEnhance=true
```

表示开启连接池，设置连接池最大db连接数为100，初始db连接为10，当创建的客户端连接超过100*10=1000的时候，每超出20个客户端连接创建一个db连接。

- 方法二：配置文件添加参数。

创建配置文件，文件名任意，添加如下内容。

```
maxActive=100  
initialSize=10  
terminalStep=20  
physicsStep=1  
enableEnhance=true
```

在代码中加载此配置文件，使用DriverManager.getConnection(url, properties)调用。

表示开启连接池，设置连接池最大db连接数为100，初始db连接为10，当创建的客户端连接超过100*10=1000的时候，每超出20个客户端连接创建一个db连接。

8.2.2. 禁用连接池使用示例

- 方法一：在url中添加参数。

url配置为jdbc:uxdb://192.71.1.35:5432/uxdb?enableEnhance=false表示显式地禁用连接池，作为原生jdbc使用。

url配置也可以配置为jdbc:uxdb://192.71.1.35:5432/uxdb，表示禁用连接池，enableEnhance默认值为false。

- 方法二：配置文件添加参数。

创建配置文件，文件名任意，添加如下内容。

```
enableEnhance=false
```

在代码中加载此配置文件，使用DriverManager.getConnection(url, properties)调用。

表示显式地禁用连接池，作为原生jdbc使用。

第 9 章 大对象适配

9.1. 概述

由于大对象在oracle和在uxdb存储的方式不同，在oracle大对象迁移的过程中，Blob被迁移成bytea，Clob被迁移成text.，而且与原生的大对象存储方式不同，使用的是单表存储。因此，开启本功能可以正常使用jdbc的blob和clob的set和get方法，达到兼容oracle应用代码的目的。

使用场景

oracle数据库完成数据迁移后，涉及对迁移后的大对象表的增删改查的情况下，需要开启和使用本功能。

使用方法

url中添加参数控制本功能，参数名为transfer，可选值：true/false。true表示开启本功能，false表示关闭本功能，缺省值为false，如下所示。

```
jdbc:uxdb://192.71.1.35:5432/db1? transfer=true
```

插入Clob/Blob的基本步骤，如下所示。

1. 通过DriverManager创建连接，并在url后设置参数transfer=true。

```
Connection conn = DriverManager.getConnection("jdbc:uxdb://192.71.1.35:5432/db1?
transfer=true", "uxdb", "1");
```

2. 设置手动提交。

```
conn.setAutoCommit(false);
```

3. 创建UXPreparedStatement对象，设置插入、查询语句。

4. 调用setClob\setBlob方法，指定参数的索引，传入输入流对象。

```
insert_lo_to_test.setClob(1, fileReader);
```

5. 执行更新并提交。

```
int i = insert_lo_to_test.executeUpdate();
conn.commit();
```

9.2. 用法示例

以jdbc插入和查询Clob为例。

在运行本用例之前需要在数据库db1中创建表test (lo text)，并准备一个文本大对象2.txt；然后编写下列代码直接运行，可以看到表test插入了一条数据，数据为2.txt中的全部内容，同时控制台输出了2.txt中的全部内容。Blob操作方法与Clob类似，将字节流对象设置到setBlob中，执行更新即可。

```
package com.uxsino.uxdb;
```

```
import java.io.BufferedReader;
```

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;
import java.io.Writer;
import java.sql.Clob;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.UXPpreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class ClobTest {

    public static void main(String[] args) {
        Connection conn = null;
        FileReader fileReader = null;
        try {
            // 1. 加载驱动
            Class.forName("com.uxsino.uxdb.Driver");
            // 2. 获取连接对象, url后在?之后添加参数transfer=true,如果有其他参数用&分隔
            conn = DriverManager.getConnection("jdbc:uxdb://192.71.1.35:5432/db1? transfer=true",
"uxdb", "1");
            // 3. 设置为手动提交, 也可以不设置, 使用自动提交
            conn.setAutoCommit(false);
            // 4. 初始化UXPreparedStatement对象
            UXPpreparedStatement insert_lo_to_test = conn.prepareStatement("insert into test values (?)");
            UXPpreparedStatement select_lo_from_test = conn.prepareStatement("select lo from test");

            // 5. "2.txt"是待插入到表中的文本大对象
            fileReader = new FileReader("D:" + File.separator + "2.txt");

            // 6. 将reader直接设置到test的lo字段中
            insert_lo_to_test.setClob(1, fileReader);

            int i = insert_lo_to_test.executeUpdate();
            conn.commit();

            ResultSet resultSet = select_lo_from_test.executeQuery();
            while (resultSet.next()) {
                Clob clob = resultSet.getClob(1);
                // 处理查到的clob, 例如打印输出
                Reader reader = clob.getCharacterStream();
                char[] buf = new char[1024];
                int length;
                while ((length = reader.read(buf)) > 0) {
                    System.out.println(length);
                    System.out.print(new String(buf, 0, length));
                }
                System.out.println();
            }
        } catch (ClassNotFoundException | SQLException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (fileReader != null) {
            try {
                fileReader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

}
```

第 10 章 boolean值适配

10.1. 概述

oracle\mysql数据库没有Boolean类型，而uxdb有。oracle和mysql的jdbc在插入Boolean值的时候，会将数据映射成1和0插入数据库，而uxdb的jdbc插入Boolean值时则是直接插入Boolean类型的值。数据迁移的过程中，不会将mysql、oracle的Boolean值1和0迁移成Boolean类型，直接使用uxdbjdbc操作迁移后的数据会类型不匹配异常。因此，uxdb在这方面做了适配。

对于数据库中以char\varchar\smallint和numeric存储的0和1数据使用场景如下所示。

oracle、mysql数据库完成数据迁移后，涉及boolean类型的表的，在应用代码中涉及到setBoolean和getBoolean方法的调用，需要开启和使用本功能。

使用方法

url中添加参数控制本功能，参数名为transfer，可选值：true/false。true表示开启本功能，false表示关闭本功能，缺省值为false，如下所示。

```
jdbc:uxdb://192.71.1.35:5432/db1? transfer=true
```

Boolean值插入和查询的步骤。如下所示。

1. 通过DriverManager创建连接，并在url后设置参数transfer=true。

```
Connection conn = DriverManager.getConnection("jdbc:uxdb://192.71.1.35:5432/db1?
transfer=true", "uxdb", "1");
```

2. 创建UXPreparedStatement对象，设置插入语句。

```
UXPreparedStatement insert_boolean_to_test = conn.prepareStatement("insert into test values (?");
```

3. 调用setBoolean方法，指定参数的索引，传入Boolean类型数据true或false。uxdbjdbc会将true转为1，false转为0存到数据库。

```
insert_boolean_to_test.setBoolean(1, true);
```

4. 执行更新。

```
int i = insert_boolean_to_test.executeUpdate();
```

如果是查询，执行executeQuery()获取到结果集resultSet。调用resultSet.getBoolean方法查询0和1代表Boolean值，uxdbjdbc会将1和0转成boolean类型的true、false返回。

5. 关闭连接释放资源。

```
conn.close();
```

10.2. 用法示例

以jdbc插入和查询boolean值使用为例。

在数据库db1中创建表 test (bo smallint)，用来模拟迁移后的boolean值表，也可以创建varchar、char和numeric字段的表。编写以下测试代码，运行后可以在表中能看到true和false两条数据，并在控制台中看到true被查到并打印。

```
public static void main(String[] args) {
    Connection conn = null;
    try {
        // 1. 加载驱动
        Class.forName("com.uxsino.uxdb.Driver");
        // 2. 获取连接对象, url后在?之后添加参数transfer=true,如果有其他参数用&分隔
        conn = DriverManager.getConnection("jdbc:uxdb://192.71.1.35:5432/db1?transfer=true",
            "uxdb", "1");
        // 3. 初始化UXPreparedStatement对象
        UXPreparedStatement insert_bo_to_test = conn.prepareStatement("insert into test values (?)");
        UXPreparedStatement select_bo_from_test = conn.prepareStatement("select bo from test where
            bo=?");
        insert_bo_to_test.setBoolean(1, true);
        insert_bo_to_test.executeUpdate();
        insert_bo_to_test.setBoolean(1, false);
        insert_bo_to_test.executeUpdate();
        select_bo_from_test.setBoolean(1, true);
        ResultSet rs = select_bo_from_test.executeQuery();
        while (rs.next()) {
            boolean aBoolean = rs.getBoolean(1);
            System.out.println(aBoolean);
        }

    } catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

第 11 章 jdbcurl数据库缺省值适配

使用jdbc创建连接的时候，针对连接缺省的数据库，uxdbjdbc支持了在url中不写数据库名默认连接缺省数据库的功能。

如果在url中不写数据库名，uxdbjdbc将会连接“uxdb”数据库。

url中“jdbc:uxdb://192.71.1.35:5432/”和“jdbc:uxdb://192.71.1.35:5432/uxdb”效果一样。