GBASE®

GBase 8a 程序员手册 JDBC 篇

www.gbase.cn

GBase 8a 程序员手册 JDBC 篇，南大通用数据技术股份有限公司

通讯方式

南大通用数据技术股份有限公司

天津华苑产业区海泰发展六道 6 号海泰绿色产业基地 J 座(300384)

电话：400-013-9696　　邮箱：info@gbase.cn

目 录

前言

手册简介

GBase 8a 程序员手册从程序员进行数据库开发的角度对 GBase 8a 进行详细介绍。

本手册介绍供客户端连接 GBase 8a 服务器用的 GBase 8a JDBC 接口驱动程序。本部分内容通过大量示例为用户演示如何使用 GBase 8a JDBC 驱动，并回答了一些最常见的关于 JDBC 的问题。

公约

下面的文本约定用于本文档：

约 定	说 明
加粗字体	表示文档标题
大写英文（SELECT）	表示 GBase 8a 关键字
等宽字体	表示代码示例
…	表示被省略的内容。

1 概述

1.1GBase JDBC 介绍

GBase 数据库通过提供 JDBC 驱动为使用 JAVA 程序语言的应用程序提供访问 GBase 数据库的接口，叫做 GBase JDBC。

GBase JDBC 是一种兼容 JDBC-3.0、4.0 "类型 4"的驱动，这意味着它是符合 JDBC 3.0、4.0 版本规范的一种纯 Java 程序，并能使用 GBase 协议直接和 GBase 服务器通信。

这一章并不是完整的 JDBC 指南。如果需要了解 JDBC 的更多信息，请参考下面的在线教程，与本章提供的内容相比，它们介绍的更为详细也更具深度：

- JDBC Basics - Sun 提供的 JDBC 指南，涵盖了 JDBC 的基本主题。
- JDBC Short Course - Sun 和 JGuru 提供的更深入的指南。

1.2GBase JDBC 版本

GBase JDBC 8.3.81.51 提供了如下新特性：

1) 自动注册驱动（需要 jdk1.6 的）

2) National Character Set Conversion Support

GBase JDBC 8.3.81.53 提供了如下新特性：

1) 针对 GBase8a 集群提供高可用性（IP 自动路由）功能；

GBase JDBC 8.3.81.53_build51.1 提供了如下新特性：

1) 针对 GBase8a 集群提供高可用性负载均衡功能；

GBase JDBC 驱动与 Server 的对应关系如下表格所示：

GBase JDBC 版本	驱动类型	JDBC 版本	状态
8.3.81.53_build51.1	4	3.0、4.0	推荐（集群）
8.3.81.53	4	3.0、4.0	推荐（集群）
8.3.81.51	4	3.0、4.0	推荐
8.2.01	4	3.0	稳定

1.3GBase JDBC 与 jdk 的兼容性

GBase JDBC 与 jdk 的兼容性如下表格所示：

GBase JDBC 版本	jre 版本
8.3.81.53_build51.1	1.5 及以上版本
8.3.81.53	1.5 及以上版本
8.3.81.51	1.5 及以上版本
8.2.01	1.4 及以上版本

1.4从旧版本升级到 8.3.81.x 注意事项

GBase JDBC 8.3.81.x 版本之前使用使用 SELECT 别名的情况，ResultSetMetaData.getColumnName()返回的是列的别名。8.3.81.x 之后返回的是列名。

在 8.3.81.x 版本中可以调用 ResultSetMetaData.getColumnLabel()来获取列别名。

应用程序可以通过设置 useOldAliasMetadataBehavior 来确定 ResultSetMetaData.getColumnLabel()的返回值是别名还是列名。当 useOldAliasMetadataBehavior 等 true 时，ResultSetMetaData.

getColumnLabel()返回值为列别名，反之为列名。

GBase JDBC 8.3.81.x 之后 useOldAliasMetadataBehavior 的默认值为 false。

1.5安装文件

我们提供的 JDBC 接口的 jar 包文件格式如下:

gbase-connector-java-<product version>-<build version>-bin.jar。

例如: gbase-connector-java-8.3.81.53-build52.2-bin.jar。

2 GBase JDBC 基本概念

2.1GBase JDBC 主要类与接口

GBase JDBC 由具有建立和管理与 GBase 数据库连接功能、执行 SQL 语句功能和对结果集进行储存和管理功能的若干功能类组成。

下面是 Gbase JDBC 涉及的主要类:

- Driver 类 (com.gbase.jdbc.Driver)

当注册驱动的时候或配置软件以使用 GBase JDBC 的时候，应该使用这个类名。在使用 GBase JDBC 8.3.81.x 版本的驱动（jre1.6 及以上版本）时可以省去注册驱动，由 java 虚拟机自动完整驱动注册。

示例:

```
Class.forName("com.gbase.jdbc.Driver");
```

- DriverManager 类 (java.sql.DriverManager)

跟踪可用的驱动程序，在数据库与相应的驱动程序之间建立连接。

在应用服务器外使用 JDBC 时，DriverManager 类管理连接的确立。

需要告诉 DriverManager 应该与哪个 JDBC 驱动进行连接，最简单的方法就是使用实现了接口 java.sql.Driver 的类的 Class.forName()方法。在 GBase JDBC 中，这个类的名字叫做 com.gbase.jdbc.Driver。用这种方法，就可以在连接一个数据库时使用一个外部配置文件来给驱动提供类名和驱动参数。

- Connection 接口 (com.gbase.jdbc. Connection)

与特定数据库的连接（会话)。在连接上下文中执行 SQL 语句并返回结果。

示例:

```
Connection
con=DriverManager.getConnection("jdbc:gbase://host:port/dbname ",
```

"user", "password");

- Statement 接口 (com.gbase.jdbc.Statement)

用于执行 SQL 语句并返回结果。

- ResultSet 接口(java.sql.ResultSet)

通常由执行 SQL 语句来产生。

示例:

```
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(select c_custkey from customer);
```

2.2URL 语法、JDBC Connector 配置属性

对于 GBase JDBC ，JDBC URL 的格式如下，方括号([,])里的项是可选的:

```
jdbc:gbase://[host][:port]/[database][?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

如果没有指定数据库，这个连接将没有'当前'数据库。这种情况下，用户需要在连接实例上调用 setCatalog()方法，或者在 SQL 中使用数据库名完整指定表名(即 'SELECT dbname.tablename.colname FROM dbname.tablename...')。不指定连接使用的数据库，只用在建立的连接需要使用多个库的工具时才有用，如 GUI 数据库管理器。

示例:

```
Class.forName("com.gbase.jdbc.Driver");
java.sql.Connection con =
DriverManager.getConnection("jdbc:gbase:///", "user", "password");
Statement st = con.createStatement();
ResultSet rs = st.executQuery(select test.testTbale.* from
```

test.testTbale);

或者

Class.forName("com.gbase.jdbc.Driver");

java.sql.Connection con = DriverManager.getConnection("jdbc:gbase:///", "user", "password");

con.setCatalog("test");

Statement st = con.createStatement();

ResultSet rs = st.executQuery(select * from testTbale);

该示例查询了 test 数据库中的 testTbale 表的全部列数据。

属性配置定义了 GBase JDBC 将会如何连接到一个 GBase server 上。除非有其它说明，否则可以为 DataSource 对象和 Connection 对象设置属性。

配置属性可以通过下列任意一种方式来设置：

- 在 java.sql.DataSource 的 GBase 实例上使用 set*()方法：
  - ✧ com.gbase.jdbc.jdbc2.optional.GBaseDataSource
  - ✧ com.gbase.jdbc.jdbc2.optional.GBaseConnectionPoolDataSource
- 在 java.util.Properties 实例中，作为一个键/值对传给 DriverManager.getConnection() 或 Driver.connect()
- 在赋给 java.sql.DriverManager.getConnection()、java.sql.Driver.connect() 或 javax.sql.DataSource's setURL()方法的 GBase 实例的 URL 中，作为一个 JDBC URL 变量。

示例：

Class.forName("com.gbase.jdbc.Driver");

```
GBaseDataSource ds = new GBaseDataSource();
ds.setUrl("jdbc:gbase://localhost:5258/test");
ds.setUser("user");
ds.setPassword("password");
java.sql.Connection con = ds.getConnection();
Statement st = con.createStatement();
ResultSet rs = st.executQuery(select c_custkey from customer);
```

注释:

如果用户用来配置一个 JDBC URL 的方法是基于 XML 的，那么用户需要使用 XML 特征字符&amp 来分开配置参数，“&”是 XML 的保留字符。

使用 GBase JDBC 驱动创建连接时可以设定的属性如下表格所示:

表 2-1 创建连接时可以设定的属性

名称	定义	默认值
连接/验证		
user	连接时使用的用户	
password	连接时使用的密码	
socketFactory	驱动程序用于创建到服务器的 socket 连接的类名。这个类必须执行 com.gbase.jdbc.SocketFactory 接口且必须有公共无参数的构造函数。	com.gbase.jdbc.StandardSocketFactory

connectTimeout	socket 连接的超时（单位毫秒），如果是 0 表示没有超时。在 JDK-1.4 或更新版本下才能使用。默认为 0。	0（通过 socket 连接进行读写操作时这两个参数才起作用。如果不设置，网络突然断开时，读数据会处在等待状态。）
socketTimeout	网络 socket 操作的超时（默认为 0，意味着无超时）	0
useConfigs	在解析 URL 或这接受用户指定的属性之前，加载使用逗号分割的配置属性。这些配置在参数配置文档中有解释。	根据配置文件初始化参数
propertiesTransform	一个 com.gbase.jdbc.ConnectionPropertiesTransform的实施实例,在尝试连接前,驱动用来修改传递进来的 URL 属性。	parseURL 方法会建立该对象修改参数
useCompression	在和服务器通信时是否使用 zlib 压缩（true/false），默认为 false。	false
isCheckProperty	创建连接前是否验证 url 参数的合法性，如果参数名称不正确，将直接报错退出。	true

vcName	设置虚拟集群名称。如果集群支持虚拟集群，必须通过该参数指定默认的虚拟集群名称	
高可靠性和集群		
resourceId	全局独一无二的资源号，用来标识数据源或数据库连接可以连接到的资源机器。	
failoverEnable	8a 集群使用，创建连接时，如果集群当前集群节点不可用，是否自动路由到下一个可用的节点。默认为 false;	false
hostList	8a 集群使用,当 failoverEnable=true 的情况下生效。记录集群中节点的 IP 以逗号分隔。 例:集群三个节点 192.168.0.2; 192.168.0.3; 192.168.0.5; 在创建数据库连接时，如果当前节点为 192.168.0.2，不可用，希望自动路由到下一个可用连接的话，可以使用下面的连接数据串: jdbc:gbase://192.168.0.2:5258/?use	

	r=user&password=pwd&failoverEnable=true&hostList=192.168.0.3,192.168.0.5	
gclusterId	8a 集群使用，当 failoverEnable=true 且 hostList 参数不为空时，设置该参数，8a 集群接口的负载均衡开启（负载策略为轮询）。gclusterId 必须以 a-z 任意字符开头的字符串，gclusterId 组成可以包含 a-z、0-9 所有字符长度为最大为 20。在同一应用程序中，gclusterId 可有程序开发人员自行决定，全工程唯一即可。 注：当设置 gclusteId 时，自动路由功能自动转换为高可用负载均衡。	
安全		
allowMultiQueries	允许在多语句中使用“;”来分割查询（true/false，默认为 false）	false
useSSL	使用 SSL 连接数据库，默认值 false.	false
allowUrlInLocalInfile	驱动是否允许在“LOAD DATA LOCAL INFILE”语句中加入 URL	false

Paranoid	防止在错误中暴露敏感信息并在可能的时候清除数据结构中的敏感信息（默认为 false）	False
性能		
metadataCacheSize	如果 cacheResultSetMetaData 设置为 true，对 cacheResultSetMetaData 的查询次数（默认为 50）	50
prepStmtCacheSize	如果可以使用预处理语句的缓冲功能，应该缓冲的预处理语句条数	25
prepStmtCacheSqlLimit	如果可以使用预处理语句的缓冲功能，驱动可以缓冲并解析的最大 SQL	256
blobSendChunkSize	通过 ServerPreparedStatements 发送 BLOB/CLOBs 时的块大小。限制每次发送 blob 字段内容的大小。	1048576
cacheCallableStmts	驱动是否应该对 CallableStatements 的解析过程进行缓冲处理	false
cachePrepStmts	驱动程序是否应对客户端预处理语句的 PreparedStatements 的解析过程执行缓冲处理，是否应检查服务器端预处理语句的适用性以及服务器端预处理语句本	false

	身。	
cacheResultSetMetadata	驱动程序是否应对用于 Statements 和 PreparedStatements 的 ResultSetMetaData 执行缓冲处理， 要求 JDK-1.4+，真／假，默认为 false。	false
cacheServerConfiguration	驱动是否应该缓冲每个 URL 上 SHOW VARIABLES 和 SHOW COLLATION 的结果	False
defaultFetchSize	驱动程序将在所有新创建的 sql 语句中使用 setFetchSize(n)设置的值	0
dynamicCalendars	需要时，驱动程序是否应检索默认日历，或根据连接／会话对其进行缓冲处理	false
elideSetAutoCommits	如果在服务器状态不匹配 Connection.setAutoCommit(boolean)需要的状态时，驱动是否只执行“set autocommit=n”查询	false
holdResultsOpenOverStatementClose	驱动是否按照 JDBC 规范中要求关闭 Statement.close()上的全部结果集	false
locatorFetchBufferSize	如果 emulateLocators 设置为 true，在为 getBinaryInputStream 取数据时应	1048576

	该使用多大的缓冲	
rewriteBatchedStatements	当调用 executeBatch()时使用重写 sql 以达到批量提交操作，提高性能。	False（开启批量更新操作，大幅提高性能）
useFastIntParsing	使用内部 String 到 Integer 的转化来避免创建过多的对象	True
useJvmCharsetConverters	总是使用 JVM 内建的字符集编码规则，还是对单字节字符集使用检索表	True
useLocalSessionState	驱动是否应该引用 Connection.setAutoCommit() 和 Connection.setTransactionIsolation()设置的 autocommit 的中间值和事务独立等级，而不是查询数据库	false
useReadAheadInput	是否在从服务器读取时使用较新的，优化后的无阻塞缓冲输入流	True
调试		
logger	实现了 com.gbase.jdbc.log.Log 的类的名称，com.gbase.jdbc.log.Log 用于记录消息（默认为	com.gbase.jdbc.log.

	“com.gbase.jdbc.log.StandardLogger”，它会将日志记录到 STDERR)。	StandardLogger
profileSQL	跟踪查询以及它们对已配制记录器的执行/获取次数（true/false，默认为 false）(执行 sql 语句也打印出来)	false
reportMetricsIntervalMillis	如果允许 gatherPerfMetrics，记录它们的频率是多少（单位 ms）	30000
maxQuerySizeToLog	当调试或者跟踪时，控制记录的查询的最大长度	2048
packetDebugBufferSize	当 enablePacketDebug 为真时，需要保留的最大信息包数目	20
slowQueryThresholdMillis	如果开启 logSlowQueries，一个查询在记录为“慢”之前可以等待的时间(ms)	2000
useUsageAdvisor	驱动是否应该发出执行警告，和就有效的 JDBC 和 GBase JDBC 用法给出建议（true/false，默认为 false）	false
autoGenerateTestcaseScript	驱动程序是否应将正在执行的 SQL（包括服务器端预处理语句）转储到 STDERR	False

dumpMetadataOnColumnNotFound	当 ResultSet.findColumn()执行失败时，驱动程序是否将字段级的原数据添加到异常信息中	False
dumpQueriesOnException	驱动程序是否应将发送至服务器的查询内容转储到 SQLExceptions 中	false
enablePacketDebug	允许时,将保留 packetDebugBufferSize 信息包的环形缓冲区，并当在驱动程序代码的关键区域抛出异常时进行转储	false
explainSlowQueries	如果开启了 logSlowQueries，驱动是否自动在服务器上执行一个 EXPLAIN，并按照 WARN 等级发送配置日志	false
includeGsdbStatusInDeadlockExceptions	当出现死锁异常时，把“SHOW ENGINE GsDB STATUS”的输出结果添加到异常信息里。	false
logSlowQueries	是否记录时间长于 slowQueryThresholdMillis 的查询	false
resultSetSizeThreshold	当 useUsageAdvisor 设置为 true 的时候，并且结果集的条数大于当前设置的 resultSetSizeThreshold 的值时，发出警告信息并计入 log。	100

traceProtocol	是否记录跟踪网络协议	False
杂项		
useUnicode	处理字符串时，指定驱动是否使用 Unicode 编码。是否只在驱动无法决定字符集映射的时候使用，或者在不考虑 GBase 是否有本地化支持的情况下，尝试驱动使用该字符集（例如 UTF-8）默认为 false	false
characterEncoding	如果 useUnicode 设置为 true，驱动在处理字符串时应该使用什么样的编码（默认为 autodetect）	
characterSetResults	服务器返回结果使用的字符集	
connectionCollation	如果设置了，就是通知服务器通过设置 connection_collation 来校正	
sessionVariables	以逗号隔开的“名称／值”对列表，当驱动程序建立了连接后，以“SET SESSION ...”的方式将其发送给服务器	

allowNanAndInf	驱动是否允许 NaN 或 +/- INF 值用于 PreparedStatement.setDouble()中	False
autoClosePStmtStreams	当 streams/readers 通过 set*()方法作为参数使用完毕后，驱动程序是否应该自动调用它们的.close()方法	false
autoDeserialize	驱动是否自动探测别名和序列化存储在 BLOB 中的对象	false
capitalizeTypeNames	是否将 DatabaseMetaData 中的类型名转换为大写，通常仅在使用 WebObjects 时有用（true/false，默认为 true）	False
clobCharacterEncoding	当发送/取回 TEXT、MEDIUMTEXT、LONGTEXT 等类型数据时使用的字符编码，取代连接中设置的字符集编码	
clobberStreamingResults	这会使“流式”结果集被自动关闭，如果在所有数据尚未从服务器中读取完之前，执行了另一查询，正在从服务器流出的任何未完成数据均将丢失	false
continueBatchOnError	如果一个语句失败，驱动是否应该继续批处理命令。JDBC 指定了允许的方式(默认为 true)	true

createDatabaseIfNotExist	如果不存在，创建 URL 中给定的数据库。假定用户具有创建数据库的权限。	false
emptyStringsConvertToZero	驱动是否允许将空字符串转化为数字的 0	true
emulateUnsupportedPstmts	驱动是否探测服务器不支持的预处理语句，并使用客户端模拟版本替换它们	true
ignoreNonTxTables	是否忽略关于回退的非事务表 （默认为 false）	false
jdbcCompliantTruncation	连接到支持告警的服务器时，当按照 JDBC 的要求截短数据时，驱动程序是否应抛出 java.sql.DataTruncation 异常	true
maxRows	返回行的最大数量（0，默认意味着返回全部行）	-1
noAccessToProcedureBodies	何时决定 CallableStatements 调用的存储过程的参数类型	False
noDatetimeStringSync	不保证 ResultSet.getDatetimeType().toString().等价于(ResultSet.getString())	False

noTimezoneConversionForTimeType	即使 useTimezone=true 也不使用 Server 的时区对 TIME 数据类型进行转换	False
nullCatalogMeansCurrent	当 DatabaseMetadataMethods 请求目录参数时，值 Null 是否意味着使用当前目录， 它不兼容 JDBC，但符合驱动程序早期版本的传统行为	true
nullNamePatternMatchesAll	接受*pattern 参数的 DatabaseMetaData 方法是否应将 null 按对待“%”的相同方式处理（不兼容 JDBC，但驱动程序的早期版本能接受与规范的这类偏离）	True
pedantic	严格遵守 JDBC 规范	False
pinGlobalTxToPhysicalConnection	当使用分布式连接时，驱动程序是否应该保证一定给定 XID 上的操作总是路由到同一个物理连接	false
processEscapeCodesForPrepStmts	驱动是否做 sql 请求语句的转义处理	true
queryTimeoutKillsConnection	如果通过 Statement.setQueryTimeout() 设置超时间，当	false

	queryTimeoutKillsConnection = true 时，如果发生超时驱动会强行断开当前连接而不是尝试结束当前查询。	
relaxAutoCommit	如果 GBase 驱动的版本不支持事务，是否允许调用 commit()，rollback() 和 setAutoCommit()，(true/false，默认为 false)	false
rollbackOnPooledClose	驱动在连接池中的逻辑连接关闭时是否应该执行 rollback()	true
runningCTS13	允许在 Sun 与 JDBC 兼容的 testsuite 1.3 版中处理缺陷	false
serverTimezone	覆盖探测/映射时区。当服务器时区不能映射为 Java 时区时使用。	
strictFloatingPoint	只在旧版本中用于一致性检查	false
strictUpdates	驱动程序是否应对可更新结果集进行严格检查（选择所有的主键）（true/false，默认为 true）	true
tinyInt1isBit	驱动程序是否应将数据类型 TINYINT(1) 当作 BIT 类型对待，创建表时，服务器	true

	会执行 BIT -> TINYINT(1)操作	
transformedBitIsBoolean	如果驱动程序将TINYINT(1)转换为不同的类型，为了与 GBase 高版本兼容，驱动程序是否应使用 BOOLEAN 取代 BIT	false
treatUtilDateAsTimestamp	treatUtilDateAsTimestamp=true 时，调用 PreparedStatement.setObject()时，把 java.util.Date 映射为 TIMESTAMP 类型。	true
useGmtMillisForDatetimes	在创建 Date 和 Timestamp 实例之前将会话时区和 GMT 之间进行转换	false
useHostsInPrivileges	在 DatabaseMetaData.getColumn/TablePrivileges()中为用户添加“@hostname”。(true/false，默认为 true)	True
useJDBCCompliantTimezoneShift	使用 java.util.Calendar 作为 JDBC 参数时，驱动程序转换 TIME/TIMESTAMP/DATETIME 等数据类型的时区信息是否应该使用 JDBC 符合度规则	false
useOldAliasMetadataBehavior	当 useOldAliasMetadataBehavior =true 时，调用 ResultSetMetaData.getColumnName() 或	false

	ResultSetMetaData.getTableName()时返回值为别名。默认为 false（8.3.81.51 版本之前默认为 true）。	
useOnlyServerErrorMessages	对服务器返回的错误消息，不事先设定“标准的”SQLState 错误消息	true
useServerPrepStmts	如果服务器支持，那么使用服务器端预处理语句（默认为 true）	true
useSqlStateCodes	使用 SQL 标准的状态代码取代’legacy’X/Open/SQL 状态代码（true/false，默认为 true）	true
useStreamLengthsInPrepStmts	是否采用 PreparedStatement/ResultSet.setXXXStream()方法调用中的流长度参数（true/false,，默认为 true）	true
useTimezone	在服务器和客户端时区之间转换时间/日期类型(true/false，默认为 false)	false
useUnbufferedInput	不使用 BufferedInputStream 来从服务器读取数据	true
yearIsDateType	JDBC 对待 GBase 的 YEAR 类型，应该看作是 java.sql.Date，还是看作 SHORT	true

zeroDateTimeBehavior	当驱动程序遇到全由 0 组成的 DATETIME 值时，应出现什么（GBase 用于表示无效值），可选的有 exception、round 和 convertToNull。	exception
functionsNeverReturnBlobs	总是将函数返回类型为 blob 的返回为 strings。只影响返回类型为 blob 的函数。	False
caseSensitiveFlag	设置对通过 getColumnName 和 getColumnLabel 方法获取的列信息是否进行大小写转换。0：不转换，1：转小写，2：转大写	0

2.3GBase JDBC API 实现要点

GBase JDBC 通过了 Sun's JDBC 兼容性测试套件中的测试。然而，在许多地方，关于如何实现特定功能，JDBC 规范并没有给出明确的规定，因此存在一定程度的实现上的灵活性。

本节将在接口层面上讲述关于特定的实施方案中用户使用 GBase JDBC 的方式。

- Blob

可以在创建连接时通过设定参数 emulateLocators=true 来提高 Blob 类型的效率。通过设置这个参数，驱动会推迟加载真正的 Blob 数据，直到你调用（getInputStream()，getBytes()等方法）的时候数据才会真正加载到 blob 数据流中。

如果使用这个参数必须满足如下条件：

必须使用带有列值的列别名，在你编写的用于检索 Blob 的 SELECT 中，将列值设为 Blob 列的真实名称。SELECT 只能从单表检索，该表必须有 1 个主键，而且 SELECT 必须涵盖构成主键的所有列。随后，驱动程序将延期加载实际的 Blob 数据,直至检索了 Blob 并在其上调用了检索方法为止(getInputStream(), getBytes(),等)。

例如：

SELECT id, 'data' as blob_data from blobtable

Blob 的实例不允许“原地”调整（它们是“副本”，正如 DatabaseMetaData.locatorsUpdateCopies()方法所指明的那样)。因此，应使用对应的 PreparedStatement.setBlob()或 ResultSet.updateBlob() （对于可更新结果集）方法，将变化保存到数据库中。

- CallableStatement

当通过 CallableStatement 接口连接 GBase 时，支持存储过程，不支持函数，函数可以通过 PreparedStatemen 调用。

- Clob

Clob 实例不允许修改（它们是复本，就像 DatabaseMetaData.locatorsUpdateCopies()方法所指明的那样)。因此，用户可以使用对应的 PreparedStatement.setClob() 方法把变化保存回数据库。GBase JDBC API 没有提供 ResultSet.updateClob()支持。

- Connection

按照 JDBC 规范，如果在这个连接上调用了 closed()，那么它只返回 true。如果用户需要确定这个连接是否还有效，用户应该使用一个简单的查询，如 SELECT 1。如果连接已不再有效的话，驱动会抛出一个异常。

在 GBase 8a JDBC8.3.81.x 版本之后在 jdk1.6 及以上环境中运行时，可以调用 boolean isValid(int timeout)方法来检查一个连接是否是有效。

- DatabaseMetaData

DatabaseMetaData 提供数据库的结构信息。

外键信息(getImportedKeys、getExportedKeys()和 getCrossReference())仅仅对 GsDB 类型的表有效，但是驱动是通过 SHOW CREATE TABLE 来获取相关信息的，所以当其他存储引擎支持外键，驱动程序同样能支持他们。

- Driver

使用 Class.forName("com.gbase.jdbc.Driver")加载驱动，在使用 GBase 8a 8.3.81.x 版本的 JDBC 驱动（jre1.6 及以上版本）时可以省去注册驱动，由 java 虚拟机自动完整驱动注册。

- PreparedStatement

在 GBase 8a server 中不支持预处理功能，PreparedStatement 是由驱动来实现的。因此，驱动程序不支持 getParameterMetaData()或 getMetaData()等，因为其需要客户端上具有完整的 SQL 语法分析程序。

当服务器支持时，会使用服务端预处理语句和二进制编码的结果集。

当使用一个带有 large 参数的服务器端预处理命令时要小心，这些参数是通过 setBinaryStream()、setAsciiStream()、setUnicodeStream()、setBlob()或 setClob()设置的。如果用户将 large 参数修改为不带 large 的参数后重新执行语句，那么此用户必需先调用 clearParameters()并重置所有的参数。原因如下：

- ✧ 当设置参数时（在执行预处理语句前），驱动程序会将 large 数据 out-of-band 发送到服务器端预处理语句；
- ✧ 一旦完成了这些，这个在客户端用于读取数据的流就会被关闭（按照 JDBC 规范），并且不能再次读取流；
- ✧ 如果一个参数从 large 变为非 large，那么驱动必须重置服务器端的预处理语句状态，这样就可以允许被改变成的参数替代先前的 large 值了。这会删除掉所有已经发送给服务器的 large 数据，因此要求通过

setBinaryStream()、 setAsciiStream()、setUnicodeStream()、setBlob()或 setClob()方法重发数据。

因此，如果用户想把一个参数的 large 类型改成一个非 large 的类型，那么用户必须调用 clearParameters()并在重新执行前再次设置准备好语句的所有参数。

- ResultSet

缺省情况下，ResultSets 被完整地取回并存储在内存中。在大多数情况下，这是操作起来最有效的方式，而且由于 GBase 网络协议的设计这也更容易实现。如果用户使用着拥有大量行或大数据的 ResultSets，而且无法在 JVM 内为所需内存分配大量空间，可以通知驱动以“流”方式返回结果，一次一行。为了使这个功能可用，用户需要按如下方式创建一个语句实例：

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
java.sql.ResultSet.CONCUR_READ_ONLY);
stmt.setFetchSize(Integer.MIN_VALUE);
```

这是个向前只读结果集，它读取得容量以整数为单位。MIN_VALUE 指示驱动程序以逐行操作的方式形成“流”结果输出。

此后，任何用这个语句创建的结果集都按行检索。

使用这种方法有些要注意的地方，用户必须在当前连接上使用任何查询之前，读出结果集中所有的行（或者关闭它），否则就是会抛出异常。

这些语句持有的锁（在 GsDB 存储引擎中，无论它们是 GsSYS 表级锁还是行级锁），最早可以在语句完成的时候释放。

如果这个语句在一个事务的范围内，那么这些锁在事务结束时（这意味着那条语句需要先完成）释放。和大多数其它数据库一样，语句不会完成直到所有依赖于该语句的结果被读取或对于这个语句活跃的结果被关闭。

因此，当使用“流”结果时，如果用户想对那些被这个语句引用并产生结果集的表保持并发访问，应当尽快地处理它们。

- ResultSetMetaData

可用于获取关于 ResultSet 对象中列的类型和属性信息的对象。

- Statement

Statements 允许用户执行基本的 SQL 查询并且通过后面将介绍的类 ResultSet 获得结果。

调用在对象 Connection 上的 createStatement()方法，可以创建一个 Statement 实例，而这个 Connection 对象可以通过在前面介绍过的 DriverManager.getConnection()或 DataSource.getConnection()方法获得。

一旦拥有了一个 Statement 实例，就能调用 executeQuery(String)方法执行一个 SELECT 查询。

更新数据库中的数据可以使用 executeUpdate(String SQL)方法，该方法返回由更新语句影响到的行数。

如果事先不知道 SQL 语句将执行一个 SELECT 操作还是一个 UPDATE/INSERT 操作，那么就可以使用 execute(String SQL)方法。如果 SQL 查询是一个 SELECT，该方法返回 true，如果是一个 UPDATE/INSERT/DELETE 查询时返回 false。并且，如果是一个 SELECT 查询，可以调用 getResultSet()方法获得结果，如果是一个 UPDATE/INSERT/DELETE 查询，可以调用 Statement 实例上的 getUpdateCount()方法获得被影响到的行数。

2.4使用字符集和 Unicode

JDBC 驱动发送给服务器的所有字符串都自动地从本地的 Java Unicode 转换到客户端的字符编码，包括所有通过 Statement.execute()、Statement.executeUpdate()、Statement.executeQuery()发送的查询以及所有 PreparedStatement 和 CallableStatement 参数，但使用 setBytes()、setBinaryStream()、setAsiiStream()、setUnicodeStream()和 setBlob()的参数集不包含在内。

GBase 8a 使用 JDBC 的 ResultSets 在客户端与服务器之间传输数据支持单一编码形式以及任意数目的编码形式。

在连接时会自动地检测客户端与服务器之间的字符编码。通过配置变量，驱动使用的编码指定在服务器上，这个变量在 GBase 8a server 中是“character_set_server ”。如果不想客户端自动识别编码，可以在用来连接服务器的 URL 中使用 characterEncoding 属性，使用该属性的同时必须使用设置“useUnicode”的值为 true。

当在客户端指定字符编码时，应该使用 Java-style 名字。下表列出了对于 GBase 字符集的 Java-style 名字：

GBase Server 字符集名	Java 字符集名称
gbk	GBK
utf8	UTF-8

警告：在 GBase JDBC 中，不要执行“set names”来设置字符集，因为驱动不会发现字符集已经改变，而是会继续使用在初始化连接设置时使用的字符集。

为了允许从客户端发来的各种字符集，应当使用“UTF-8”编码，或者把“UTF-8”配置成缺省的服务器字符集，或者通过 characterEncoding 属性配置 JDBC 驱动使用“UTF-8”。

3 GBase JDBC 高可用特性

GBase JDBC 高可用特性是针对 GBase 数据库集群提供客户端高可用及负载均衡相关功能功能。

3.1GBase JDBC 集群高可用性

在通过 GBaseJDBC 访问 GBase 集群时，如果集群当前节点不可用，我希望直接连接到集群中一个可用的节点上时，我们可以使用 GBase JDBC 集群高可用性功能（该功能需要 GBase JDBC8.3.81.53 及以上版本)。

GBase JDBC 集群高可用性是接口针对 GBase 集群做的在接口层面的高可用性处理（IP 自动路由)。

高可用性适用于扁平结构的 GBase 集群，在创建 JDBC 连接时，如果当前 IP 的集群节点不可用，接口会根据相关参数信息把连接数据库请求自动路由到集群其他可用的节点。

假设有如下场景:

1、部署有一个 GBase 集群，三个节点 IP 如下:

192.168.1.56;

192.168.1.57;

192.168.1.58;

创建数据库连接时设置 hostList、hostList 参数。

```
String url =
"jdbc:gbase://192.168.1.56:5258/test?user=root&password=root
&failoverEnable=true&hostList=192.168.1.57,192.168.1.58";
……
DriverManager.getConnection(dbUrl);
```

……

即可实现接口层面的高可用性（IP 自动路由）功能，详细样例代码参照 2.8.13 章节。

3.2GBase JDBC 集群高可用负载均衡

如果我们希望把数据库连接请求平均分布到各个节点的上话，我们可以使用 GBase JDBC 集群高可用负载均衡功能（该功能需要 GBase JDBC 8.3.81.53_build51.1 及以上版本。）。

高可用负载均衡是 GBaseJDBC 针对集群开发的接口层面的客户端负载均衡功能。该功能能把客户发送来的数据库连接请求分发到各个节点，在 GBaseJDBC8.3.81.53_built51.1 版本中负载均衡策略为轮询。

假设有如下场景：

1、部署有一个 GBase 集群，三个节点 IP 如下：

192.168.1.56；

192.168.1.57；

192.168.1.58；

如果我们希望把数据库连接请求均摊到三个节点时，创建连接时设置 failoverEnable、hostList、gclusterId 三个参数即可。连接串写法如下：

String URL = "jdbc:gbase://192.168.1.56:5258/test?user=gbase&password=gbase20110531&failoverEnable=true&hostList=192.168.1.57,192.168.1.58&gclusterId=gcl1"

其中 gclusterId 参数取值范围:必须以 a-z 任意字符开头的可以包含 a-z、0-9 所有字符长度为最大为 20 的字符串。

……

```
for (int I = 0; i < 9; i++) {
DriverManager.getConnection(SampleGBaseJDBCLoadbalance.URL);
}
……
```

上述代码会创建 9 个连接，假设三个节点都是可用状态，执行结果是每个节点分配到三个连接；如果其中一个节点不可用状态，GBaseJDBC 驱动会把 9 个数据库连接请求分配到另外两个节点上。

进一步详细代码请参照 8.14 章节。

注:

高可用负载均衡功能集包含高可用性，即当开始高可用负载均衡时高可用性同时开启。

4 Java、JDBC 和 GBase 数据类型映射关系

4.1 Java 和 GBase 数据类型转换

由于 SQL 数据类型和 Java 数据类型是不同的，因此需要某种机制在使用 Java 类型的应用程序和使用 SQL 类型的数据库之间来读写数据。

为此，JDBC 提供了 getXXX 和 setXXX 方法集、方法 registerOutParameter 和类 Types。

GBase JDBC 在处理 GBase 数据类型与 Java 数据的类型的转换上很灵活。

一般地，任何 GBase 数据类型都可以转换成一个 java.lang.String，且任何数值类型都可以转换成 Java 的任意数值类型，虽然可能会发生近似，溢出或精度损失。

GBase JDBC 驱动会像 JDBC 规范所要求的那样使用警告或抛出 DataTruncation 意外,除非通过使用"jdbcCompliantTruncation"属性并把它设置为 false 来设置连接，让它不这么做。

在下表中列出了能可靠工作的转换:

表 4-1 GBase Server 数据类型与 Java 类型映射关系

GBase Server 数据类型	总是可以转化为的 Java 类型
CHAR, VARCHAR, BLOB, TEXT, LONGBLOB	java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Clob
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	java.math.BigDecimal 注意：与希望转换的 GBase 数据类型相比，如果选择了精度较低的 Java 数值类型，可能会出现舍入、溢出或精度损失
DATE, TIME, DATETIME,	java.lang.String, java.sql.Date,

GBase Server 数据类型	总是可以转化为的 Java 类型
TIMESTAMP	java.sql.Timestamp

在 GBase Server 类型和 Java 类型之间，ResultSet.getObject()方法采用了下述类型转换方式，在可能的情况下遵从 JDBC 规范：

表 4-2 GBase Server 数据类型、JDBC 数据类型、Java 类映射关系

GBase Server 数据类型名	JDBC 数据类型（GetColumnType 方法的返回值）	返回的 Java 类
TINYINT	TINYINT	java.lang.Boolean 如果配置属性 tinyInt1isBit 设置成 true（缺省），且存储大小设置为 1，如果没有上述设置为 java.lang.Integer
BOOL, BOOLEAN	BIT	java.lang.Boolean
SMALLINT[(M)]	SMALLINT	java.lang.Integer
MEDIUMINT[(M)]	INTEGER	java.lang.Integer
INT,INTEGER[(M)]	INTEGER	java.lang.Integer
BIGINT[(M)]	INTEGER	java.lang.Long
REAL[(M,D)]	DOUBLE	Java.lang.Double
FLOAT[(M,D)]	REAL	java.lang.Float
DOUBLE[(M,B)]	DOUBLE	java.lang.Double
DECIMAL[(M[,D])]	DECIMAL	java.math.BigDecimal

GBase Server 数据类型名	JDBC 数据类型（GetColumnType 方法的返回值）	返回的 Java 类
NUMERIC[(M,D)]	DECIMAL	java.math.BigDecimal
DATE	DATE	java.sql.Date
DATETIME	TIMESTAMP	java.sql.Timestamp
TIMESTAMP[(M)]	TIMESTAMP	java.sql.Timestamp
TIME	TIME	java.sql.Time
YEAR[(2|4)]	DATE	如果 yearIsDateType 配置属性设置为 false，则返回对象类型为 java.sql.Short. 如果设置成 true（缺省值）则返回 java.sql.Date
CHAR(M)	CHAR	java.lang.String（如果列字符集设置为 BINARY，则返回 byte[]）
VARCHAR(M) [BINARY]	VARCHAR	java.lang.String（如果列字符集设置为 BINARY，则返回 byte[]）
TINYBLOB	VARBINARY	byte[]
TINYTEXT	LONGVARCHAR	java.lang.String
TEXT	LONGVARCHAR	java.lang.String
MEDIUMBLOB	LONGVARBINARY	byte[]
MEDIUMTEXT	LONGVARCHAR	java.lang.String

GBase Server 数据类型名	JDBC 数据类型（GetColumnType 方法的返回值）	返回的 Java 类
LONGBLOB	LONGVARBINARY	byte[]
LONGTEXT	LONGVARCHAR	java.lang.String

4.2 Java 类型类介绍

4.2.1 com.gbase.jdbc.Clob 类

来自 com.gbase.jdbc.Clob，该类提供通用的方法来处理数据库中的 Clob 类型数据，类似的接口有 java.sql.Clob。GBase JDBC 通过以下几种数据类型来支持 Clob、TEXT、TINYTEXT、MEDIUMTEXT、LONGTEXT 。

所有已实现的接口 java.sql.Clob、
com.gbase.jdbc.OutputStreamWatcher、com.gbase.jdbc.WriterWatcher

属性如下：

- private String charData;

该属性为 Clob 对象包含字符数据。

- private ExceptionInterceptor exceptionInterceptor;

声明一个异常拦截器。ExceptionInterceptor 来自 com.gbase.jdbc.ExceptionInterceptor;

表 4-3 Clob 类方法列表

类型	名称	说明
	Clob(ExceptionInterceptor exceptionInterceptor)	一个构造函数，用来创建一个新的 Clob 对象。
	Clob(String charDataInit, ExceptionInterceptor exceptionInterceptor)	一个构造函数，用来创建一个新的 Clob 对象。
	free()	该方法释放该 Clob 对象所占用

类型	名称	说明
		的资源。
InputStream	getAsciiStream()	以 ascii 流返回该 Clob 值。
Reader	getCharacterStream()	以 java.io.Reader 对象或以 characters 流的形式返回该 Clob 值。
Reader	getCharacterStream(long pos, long length)	返回从 pos 开始的，长度为 length 的部分 Clob 值。
String	getSubString(long startPos, int length)	返回该 Clob 对象指定位置开始一定长度的 String。
long	length()	返回该 Clob 对象的值长度。
long	position(java.sql.Clob arg0, long arg1)	返回指定的 Clob 值在该 Clob 对象值中指定位置后出现的位置。
long	position(String stringToFind, long startPos)	返回指定字符串在该 Clob 对象值中指定位置开始后出现的位置
OutputStream	setAsciiStream(long indexToWriteAt)	用于将 Ascii 字符写入此 Clob 对象表示的 Clob 值中的流(位置 indexToWriteAt 处)。
Writer	setCharacterStream(long indexToWriteAt)	用于将 Unicode 字符流写入此 Clob 对象表示的 CLOB 值中的流 (位置 indexToWriteAt 处)。
int	setString(long pos, String str)	在位置 pos 处将给定 Java String 写入此 Clob 对象指定的 CLOB 值中。
int	setString 诉讼(long pos, String str, int offset, int len)	将 str 的 len 个字符写入此 Clob 表示的 CLOB 值中,从字符 offset 开始。
	streamClosed(WatchableOutputStream out)	将该 Clob 对象值通过指定 stream 写完，并将 stream 关闭。
	truncate(long length)	截取该 Clob 对象值,使其长度为 length。

类型	名称	说明
	writerClosed(WatchableWriter out)	将指定的 char[]赋值给该 Clob 对象的 charData 属性。
	writerClosed(char[] charDataBeingWritten)	将该 Clob 对象值通过指定 Writer 写完，并将 Writer 关闭。

方法详细信息:

- Clob(ExceptionInterceptor exceptionInterceptor)

初始化构造一个 Clob 对象,该 Clob 对象初始化属性 charData 为空字符串，初始化属性 exceptionInterceptor 为参数 exceptionInterceptor。

注:ExceptionInterceptor 来自 com.gbase.jdbc. ExceptionInterceptor。

由于该构造函数是非 public 的，所以对于使用 JDBC 的开发人员来说，该构造函数是不可见的。

- Clob(String charDataInit, ExceptionInterceptor exceptionInterceptor)

初始化构造一个 Clob 对象，该 Clob 对象初始化属性 charData 为参数 charDataInit，初始化属性 exceptionInterceptor 为参数 exceptionInterceptor。

注:ExceptionInterceptor 来自 com.gbase.jdbc. ExceptionInterceptor。

由于该构造函数是非 public 的，所以对于使用 JDBC 的开发人员来说，该构造函数是不可见的。

- free()

public void free() throws SQLException

该方法释放该 Clob 对象所使用的资源，一旦使用该方法，该对象将变得不可用。在该方法被调用后如果还企图去调用该 Clob 对象 free()以外的方法，将会得到一个 SQLException 的异常。多次调用 free 方法，后来的 free 方法将被视为无操作。

抛出:

SQLException - 在释放 Clob 对象资源时发生错误。

SQLFeatureNotSupportedException - 如果 JDBC 版本不支持该方法。

- getAsciiStream()

public InputStream getAsciiStream() throws SQLException

以 ascii 流形式获取此 Clob 对象指定的 CLOB 值。

返回:

包含 CLOB 数据的 java.io.InputStream 对象。

抛出:

SQLException - 如果访问 CLOB 值时发生错误。

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

- getCharacterStream()

public Reader getCharacterStream() throws SQLException

以 java.io.Reader 对象形式（或字符流形式）获取此 Clob 对象指定的 CLOB 值。

返回:

包含 CLOB 数据的 java.io.Reader 对象

抛出:

SQLException - 如果访问 CLOB 值时发生错误

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

- getCharacterStream(long pos, long length)

public Reader getCharacterStream(long pos, long length)

throws SQLException

返回包含部分 Clob 值的 Reader 对象，该值从 pos 指定的字符开始，长度为 length 个字符。

参数:

pos - 将获取的部分值第一个字符的偏移量。Clob 中的第一个字符在位置 1 处。

length - 要获取的部分值的字符长度。

返回:

Reader，可以通过它来读取部分 Clob 值。

抛出:

SQLException - 如果 pos 小于 1，或者 pos 大于 Clob 中的字符数，或者 pos + length 大于 Clob 中的字符数

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

- getSubString(long startPos, int length)

public String getSubString(long startPos, int length)

throws SQLException

获取此 Clob 对象指定的 CLOB 值中指定子字符串的副本。子字符串开始于位置 startPos 处，有 length 个连续字符。

参数:

startPos - 要提取的子字符串的第一个字符。第一个字符位于位置 1 处。

length - 要复制的连续字符的数量；length 的值必须大于等于 0。

返回:

一个 String，它是由此 Clob 对象指定的 CLOB 值中的指定子字符串

抛出:

SQLException - 如果访问 CLOB 值时发生错误；如果 pos 小于 1 或者

length 小于 0。

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

- length()

public long length() throws SQLException

获取此 Clob 对象指定的 CLOB 值中的字符数。

返回:

CLOB 的字符长度。

抛出:

SQLException - 如果访问 CLOB 值的长度时发生错误。

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

- position(java.sql.Clob arg0, long arg1)

public long position(java.sql.Clob arg0, long arg1)

throws SQLException

获取此 Clob 对象中指定的 Clob 对象 arg0 出现的字符位置。从位置 arg1 开始搜索。

参数:

arg0- 要搜索的 Clob 对象。

arg1- 开始搜索的位置；第一个位置是 1。

返回:

Clob 对象出现的位置；如果没有出现，则返回 -1；第一个位置是 1。

抛出:

SQLException - 如果访问 CLOB 值时发生错误，或者 arg1 小于 1。

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

- position(String stringToFind, long startPos)

public long position(String stringToFind, long startPos)

throws SQLException

与 position(java.sql.Clob arg0, long arg1)相似，只不过查找的是 String 对象。

- setAsciiStream(long indexToWriteAt)

public OutputStream setAsciiStream(long indexToWriteAt)

throws SQLException

获取用于将 Ascii 字符写入此 Clob 对象表示的 Clob 值中的流，从位置 indexToWriteAt 处开始。写入流中的字符将从位置 indexToWriteAt 开始重写 Clob 对象中的现有字节。如果在将字符写入流中时到达 Clob 值的末尾，则将增加 Clob 值的长度，以容纳额外的字符。

注：如果为 indexToWriteAt 指定的值大于 CLOB 值的长度+1，则行为是不确定的。一些 JDBC 驱动程序可能抛出 SQLException，而另一些驱动程序可能支持此操作。

参数：

indexToWriteAt - 开始写入此 CLOB 对象中的位置；第一个位置是 1。

返回：

可以将 ASCII 编码字符写入其中的流。

抛出：

SQLException - 如果访问 CLOB 值时发生错误，或者 indexToWriteAt 小于 1。

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

- setCharacterStream(long indexToWriteAt)

public Writer setCharacterStream(long indexToWriteAt)

throws SQLException

获取用于将 Unicode 字符流写入此 Clob 对象表示的 CLOB 值中（位置 indexToWriteAt 处）的流。写入流中的字符将从位置 indexToWriteAt 开始重写 Clob 对象中的现有字节。如果在将字符写入流中时到达 Clob 值的末尾，则将增加 Clob 值的长度，以容纳额外的字符。

注：如果为 indexToWriteAt 指定的值大于 CLOB 值的长度+1，则行为是不确定的。一些 JDBC 驱动程序可能抛出 SQLException，而另一些驱动程序可能支持此操作。

参数：

indexToWriteAt - 开始写入 CLOB 值中的位置；第一个位置是 1。

返回：

可将 Unicode 编码字符写入其中的流。

抛出：

SQLException - 如果访问 CLOB 值时发生错误，或者 indexToWriteAt 小于 1。

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

- setString(long pos, String str)

public int setString(long pos, String str)

throws SQLException

在位置 pos 处将给定 Java String 写入此 Clob 对象指定的 CLOB 值中。该字符串将从位置 pos 开始重写 Clob 对象中的现有字节。如果在写入给定字符串时到达 Clob 值的末尾，则将增加 Clob 值的长度，以容纳额外的字符。

注：如果为 pos 指定的值大于 CLOB 值的长度+1，则行为是不确定的。一些 JDBC 驱动程序可能抛出 SQLException，而另一些驱动程序可能支持此操

作。

参数:

pos - 开始写入此 Clob 对象表示的 CLOB 值的位置; 第一个位置是 1。

str - 要写入此 Clob 指定的 CLOB 值中的字符串。

返回:

写入的字符数。

抛出:

SQLException - 如果访问 CLOB 值时发生错误, 或者 pos 小于 1。

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

- setString(long pos, String str, int offset, int len)

public int setString(long pos, String str, int offset, int len)

throws SQLException

将 str 的 len 个字符 (从字符 offset 开始) 写入此 Clob 表示的 CLOB 值中。该字符串将从位置 pos 开始重写 Clob 对象中的现有字节。如果在写入给定字符串时到达 Clob 值的末尾, 则将增加 Clob 值的长度, 以容纳额外的字符。

注: 如果为 pos 指定的值大于 CLOB 值的长度+1, 则行为是不确定的。一些 JDBC 驱动程序可能抛出 SQLException, 而另一些驱动程序可能支持此操作。

参数:

pos - 开始写入此 CLOB 对象的位置; 第一个位置是 1。

str - 要写入此 Clob 对象表示的 CLOB 值中的字符串。

offset - str 中开始读取要写入字符的偏移量。

len - 要写入的字符数。

返回:

写入的字符数。

抛出:

SQLException - 如果访问 CLOB 值时发生错误，或者 pos 小于 1。

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

- streamClosed(WatchableOutputStream out)

public void streamClosed(WatchableOutputStream out)

关闭所指定的 out 流。

- truncate(long length)

public void truncate(long length) throws SQLException

截取此 Clob 指定的 CLOB 值，使其长度为 length 个字符。

参数:

length- CLOB 值应被截取的字符长度。

抛出:

SQLException - 如果访问 CLOB 值时发生错误，或者 len 小于 0。

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

- writerClosed(WatchableWriter out)

public void writerClosed(WatchableWriter out)

关闭指定 out 流。

- writerClosed(char[] charDataBeingWritten)

public void writerClosed(char[] charDataBeingWritten)

将指定 char[]赋值给该 Clob 对象的 charData 属性。

com.gbase.jdbc.Clob 示例: 参考 2.5.3 小节示例

4.2.2 java.lang.String 类

请参考 Java 2 Platform SE 6 API java.lang.String 类，该类型映射 JDBC 数据类型：CHAR、VARCHAR、LONGVARCHAR。

4.2.3 java.sql.Date 类

请参考 Java 2 Platform SE 6 API java.sql.Date 类，该类型映射 JDBC 数据类型：DATE。

4.2.4 java.sql.Time 类

请参考 Java 2 Platform SE 6 API java.sql.Time 类，该类型映射 JDBC 数据类型：TIME。

4.2.5 java.sql.Timestamp 类

请参考 Java 2 Platform SE 6 API java.sql.Timestamp 类，该类型映射 JDBC 数据类型：TIMESTAMP。

4.2.6 java.lang.Integer 类

请参考 Java 2 Platform SE 6 API java.lang.Integer 类，该类型映射 JDBC 数据类型：TINYINT、SMALLINT、INTEGER。

4.2.7 java.lang.Long 类

请参考 Java 2 Platform SE 6 API java.lang.Long 类，该类型映射 JDBC 数据类型：BIGINT。

4.2.8 java.lang.Float 类

请参考 Java 2 Platform SE 6 API java.lang.Float 类，该类型映射 JDBC 数据类型：FLOAT、REAL。

4.2.9 java.lang.Double 类

请参考 Java 2 Platform SE 6 API java.lang.Double 类，该类型映射 JDBC 数据类型：DOUBLE、FLOAT。

4.2.10 java.math.BigDecimal 类

请参考 Java 2 Platform SE 6 API java.lang.BigDecimal 类，该类型映射 JDBC 数据类型：DECIMAL、NUMERIC。

4.3 JDBC 类型类介绍

java.sql.Types 类的成员变量如下表所示

表 4-4 JDBC 数据类型

GBase 8a 常用成员列表	名称	说明
static int	BIGINT	存储超大整数，映射到 GBase SQL 类型 BIGINT。
static int	BOOLEAN	布尔值，映射到 GBase SQL 类型 BOOLEAN。
static int	CHAR	固定长度字符串，映射到 GBase SQL 类型 CHAR。
static int	DATE	以 yyyy-mm-dd 格式的日期，映射到 GBase SQL 类型 DATE。
static int	DECIMAL	自定义精度的浮点型数据,映射到 GBase SQL 类型 DECIMAL。
static int	DOUBLE	双精度浮点型数据，映射到 GBase SQL

GBase 8a 常用成员列表	名称	说明
		类型 DOUBLE。
static int	FLOAT	单精度浮点型数据，映射到 GBase SQL 类型 FLOAT。
static int	INTEGER	32 位的有符号整数，映射到 GBase SQL 类型 INT、INTEGER。
static int	LONGVARCHAR	长度可变的大字符串，映射到 GBase SQL 类型 TEXT、LONGTEXT、TINYTEXT。
static int	LONGVARBINARY	可变数量数据的二进制大对象数据类型，映射到 GBase SQL 类型 TINYBLOB、MEDIUMBLOB、LONGBLOB。
static int	NUMERIC	表示固定精度的十进制值，映射到 GBase SQL 类型 NUMERIC。
static int	REAL	有 7 位尾数的“单精度”浮点数，映射到 GBase SQL 类型 REAL。
static int	SMALLINT	16 位的有符号整数，映射到 GBase SQL 类型 SMALLINT。
static int	TIME	小时、分钟和秒组成的时间，映射到 GBase SQL 类型 TIME。
static int	TIMESTAMP	DATE 加上 TIME，外加一个纳秒域，映射到 GBase SQL 类型 TIMESTAMP、DATETIME。
static int	TINYINT	8 位无符号整数，映射到 GBase SQL 类型 TINYINT、BOOL。
static int	VARCHAR	变长字符串，映射到 GBase SQL 类型 VARCHAR。

5 GBase 与 J2EE 应用服务器

5.1 一般 J2EE 连接池概念

连接池是创建和管理一个连接的缓冲池的技术，这些连接准备好被任何需要它们的线程使用。

这种把连接汇集起来的技术基于这样的一个事实：对于大多数应用程序，当它们正在处理通常需要数毫秒完成的事务时，仅需要能够访问 JDBC 连接的 1 个线程。当不处理事务时，这个连接就会闲置。相反，连接池允许闲置的连接被其它需要的线程使用。

事实上，当一个线程需要用 JDBC 对一个 GBase 或其它数据库操作时，它从连接池中请求一个连接。当这个线程使用完了这个连接，将它返回到连接池中，这样这就可以被其它想使用它的线程使用。

当连接从池中借出，它被请求它的线程专有地使用。从编程的角度来看，这和用户的线程每当需要一个 JDBC 连接的时候调用 DriverManager.getConnection()是一样的，采用连接池技术，可通过使用新的或已有的连接结束线程。

连接池可以极大的改善用户的 Java 应用程序的性能，同时减少全部资源的使用。连接池主要的优点有：

- 减少连接创建时间

虽然与其它数据库相比 GBase 提供了较为快速的连接功能，但是创建新的 JDBC 连接仍会招致网络和 JDBC 驱动的开销。如果这类连接是“循环”使用的，那么使用该方式造成的这些花销就可避免。

- 简化的编程模式

当使用连接池时，每一个单独的线程能够像创建了一个自己的 JDBC 连接一样操作，允许用户直接使用 JDBC 编程技术。

- 受控的资源使用

如果用户不使用连接池，而是每当线程需要时创建一个新的连接，那么用户的应用程序的资源使用会产生非常大的浪费并由可能会导致高负载下的异常发生。

注意：每个连到 GBase 的连接在客户端和服务器端都有花销（内存，CPU，上下文切换等等）。每个连接均会对应用程序和 GBase 服务器的可用资源带来一定的限制。不管这些连接是否在做有用的工作，仍将使用这些资源中的相当一部分。

连接池能够使性能最大化，同时还能将资源利用控制在一定的水平之下，如果超过该水平，应用程序将崩溃而不仅仅是变慢。

幸运的是，Sun 已经通过 JDBC-2.0 可选的接口，标准化了 JDBC 中的连接池的概念，并且所有的主流应用服务基本都实现了与 GBase JDBC 一起良好工作的这类 API。

通常，你可以在应用服务器的配置文件中配置连接池，并通过 Java 命名和目录接口（JNDI）访问它。在下面的代码中，介绍了在 J2EE 应用服务器上运行的应用程序中使用连接池的方法::

例：在 J2EE 应用服务器上使用连接池，部分代码如下：

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import javax.naming.InitialContext;
import javax.sql.DataSource;
public class GBaseServletJspOrEjb {
    public void doSomething() throws Exception {
        /*
        * Create a JNDI Initial context to be able to  lookup  the
        * DataSource In production-level code, this should be
*cached
        * as an instance or static variable, as it can be quite
```

```
            * expensive to create a JNDI context.
            * Note: This code only works when you are using servlets
            * or EJBs in a J2EE application server. If you are
            * using connection pooling in standalone Java code, you
            * will have to create/configure datasources using whatever
            * mechanisms your particular connection pooling library
            * provides.
            */
            InitialContext InitCtx = new InitialContext();
            /*
            * Lookup the DataSource, which will be backed by a pool
            * that the application server provides. DataSource instances
            * are also a good candidate for caching as an instance
            * variable, as JNDI lookups can be expensive as well.
            */
            DataSource ds = (DataSource)
InitCtx.lookup("java:comp/env/jdbc/GBaseDB");
            /*
            The following code is what would actually be in your
            Servlet, JSP or EJB 'service' method...where you need
            to work with a JDBC connection.
            */
            Connection conn = null;
            Statement stmt = null;
            try {
                conn = ds.getConnection();
                /*
                Now, use normal JDBC programming to work with
                GBase, making sure to close each resource when you're
                finished with it, which allows the connection pool
                resources to be recovered as quickly as possible
                */
                stmt = conn.createStatement();
                stmt.execute("DO SOME SQL QUERY");
                stmt.close();
                stmt = null;
                conn.close();
```

```
                conn = null;
            } finally {
                /*
                * close any jdbc instances here that weren't
                * explicitly closed during normal code path, so
                * that we don't 'leak' resources...
                */
                if (stmt != null) {
                    try {
                        stmt.close();
                    } catch (sqlexception sqlex) {
                        //ignore -- as we can't do anything about it here
                    }
                    stmt = null;
                }
                if (conn != null) {
                    try {
                        conn.close();
                    } catch (sqlexception sqlex) {
                        //ignore -- as we can't do anything about it here
                    }
                    conn = null;
                }
            }
        }
    }
```

如上面的例子所示，在获得 JNDI InitialContext 并查找 DataSource 之后，剩下的代码与前面完成的 JDBC 编程看起来类似。

当使用连接池时，要记住的最重要的是确保无论在用户的代码中发生了什么（意外，控制流等），连接和任何由它们创建的东西（语句，结果集等）都是需要关闭的，以便于它们可以被再使用，否则的话它们会进退两难，这在最好的情况下意味着它们代表的 GBase 服务器资源（缓存，锁，套接字等）会在某时被占用，在最坏的情况下，会被永远占用。

连接池最佳大小是多少，与其它所有配置规则一样，答案是“取决于具体

情况”。最佳的大小依赖于预期负载和平均数据库事务时间，最适合的连接池大小比用户预期的要小。

如果用户以 Sun’s Java Petstore Blueprint 应用程序进行测试，测试使用 GBase 和 Tomcat。在 GBase 中设置 15-20 个连接的连接池为中等规模的负载（600 用户并发）服务，其响应时间在可接受的范围内。

要想确定用于应用程序的连接池大小，应使用诸如 Apache Jmeter 或 The Grinder 等工具创建负载测试脚本，并对应用程序进行负载测试。

一个决定在连接池中放置的合理连接数的最简单的办法是先将连接池的连接数设置到极大，然后运行负载测试对并发用户的最大值进行检测，这样用户就可以得出使用户指定程序得到最佳性能的连接数最大值和最小值。

5.2基于 Tomcat 使用 GBase JDBC

以下说明适合 Tomcat 5.5，对于新版本 GBase JDBC 8.3.81.51 需要使用 JDK1.6。需要将 GBase JDBC 驱动包 gbase*.jar 放入目录 $TOMCAT_HOME\common\lib 下来配置用户的服务器。在定义 Web 应用程序的场景内，通过在$TOMCAT_HOME\conf\Catalina\localhost 目录下增加声明资源文件，该文件以 Web 应用名称为名（例:GBaseapp.xml），配置 JNDI DataSource，内容部分如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Resource
      name="jdbc/GBaseDB"
      type="javax.sql.DataSource"
      password="somepassword"
      driverClassName="com.gbase.jdbc.Driver"
```

```
        maxIdle="2"
        maxWait="50"
        username="user"
        url="jdbc:gbase://localhost:5258/dbname"
        maxActive="4"/>
</Context>
```

需要在 Web 应用的 WEB-INF 目录内的 web.xml 文件中添加如下内容:

```
<resource-ref>
        <description>DB Connection</description>
        <res-ref-name>jdbc/GBaseDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
</resource-ref>
```

在程序中访问数据源的代码为:

```
Context initCtx = new InitialContext();
DataSource ds =
(DataSource)initCtx.lookup("java:comp/env/jdbc/GBaseDB");
```

一般地，用户应该遵照 Tomcat 自带的安装说明，因为用户在 Tomcat 中配置数据源的方法有时候是不同的，并且，如果在用户的 XML 文件里使用了错误的语法，那么很可能以一个如下的异常结束:

```
Error: java.sql.SQLException: Cannot load JDBC driver class 'null
' SQL
state: null
```

注: 关于 Tomcat 使用 Gbase JDBC 实例请查看 2.8.12 小节。

5.3基于 JBoss 使用 GBase JDBC

这些说明适用于 JBoss-4.x，对于新版本 GBase JDBC 8.3.81.51 可以使用 JDK1.5或以上版本。要想使应用服务器能够使用 JDBC 驱动类，需要把 GBase JDBC 带的 gbase*.jar 驱动包文件复制到用户的 lib 文件夹下来配置用户的服务器（通常叫做缺省值，位于 jboss_home\server\default）。然后，在相同的配置文件夹下，在名为 deploy 的子文件夹中，创建一个以“-ds.xml”结尾的数据源配置文件，它告诉 Jboss 把这个文件作为 JDBC 数据源。这个文件应该有如下内容：

```
<?xml version="1.0" encoding="UTF-8" ?>
<datasources>
    <local-tx-datasource>
        <!-- This connection pool will be bound into JNDI with the
name "java:/GBaseDB" -->
        <jndi-name>GBaseDB</jndi-name>
<connection-url>jdbc:gbase://localhost:5258/dbname</connection-url>
        <driver-class>com.gbase.jdbc.Driver</driver-class>
        <user-name>user</user-name>
        <password>pass</password>
        <min-pool-size>5</min-pool-size>
        <!-- Don't set this any higher than max_connections on your
         GBase server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->
        <max-pool-size>20</max-pool-size>
        <!-- Don't allow connections to hang out idle too long,
```

```
            never longer than what wait_timeout is set to on the
            server...A few minutes is usually okay here,
            it depends on your application
            and how much spikey load it will see -->
          <idle-timeout-minutes>5</idle-timeout-minutes>
          <!-- If you're using GBase JDBC 3.1.8 or newer, you can use
               our implementation of these to increase the robustness
               of the connection pool. -->
    <exception-sorter-class-name>com.gbase.jdbc.integration.jboss.ExtendedGBaseExceptionSorter</exception-sorter-class-name>
    <valid-connection-checker-class-name>com.gbase.jdbc.integration.jboss.GBaseValidConnectionChecker</valid-connection-checker-class-name>
        </local-tx-datasource>
    </datasources>
```

在 Web 项目中的 META-INF 文件夹下添加 jbosscmp-jdbc.xml 内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<jbosscmp-jdbc>
<defaults>
<datasource>java:/GBaseDB</datasource>
</defaults>
</jbosscmp-jdbc>
```

在程序中访问数据源的代码为：

```
Context initCtx = new InitialContext();

DataSource ds =

(DataSource)initCtx.lookup("java:/GBaseDB");
```

注：关于 JBoss 使用 Gbase JDBC 实例请查看 2.8.11 小节。

5.4websphere6.0 配置 JDNI

本小节介绍 websphere6.0 如何通过 JNDI 配置连接池。以 websphere6.0 为例，由于 websphere 只能支持 jdk1.4 所以我们使用 GBaseJDBC8.2.01 作为 JDBC 驱动。

准备工作：把 gbasejdbc-8.2.01.jar 拷贝到 websphere 的安装目录 \IBM\WebSphere\AppServer\lib\下并定义 websphere 变量。

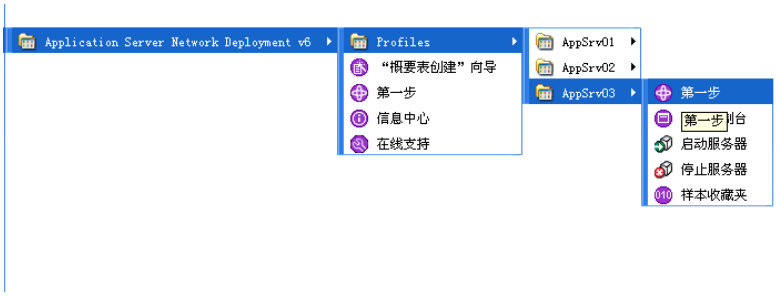
约定：

1、 方框表示需要注意的地方

2、 椭圆表示鼠标单击

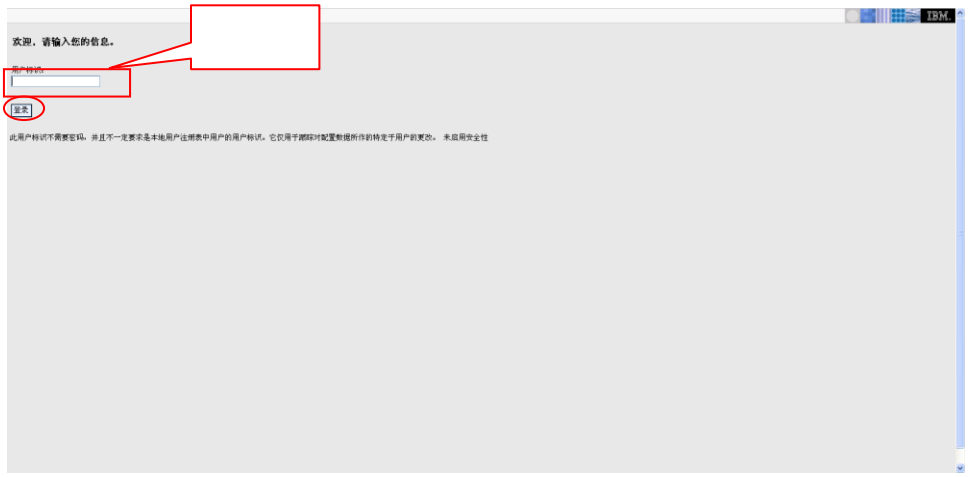
5.4.1配置方法

从开始菜单中找到 “IBM WebSphere”按照下午选择：



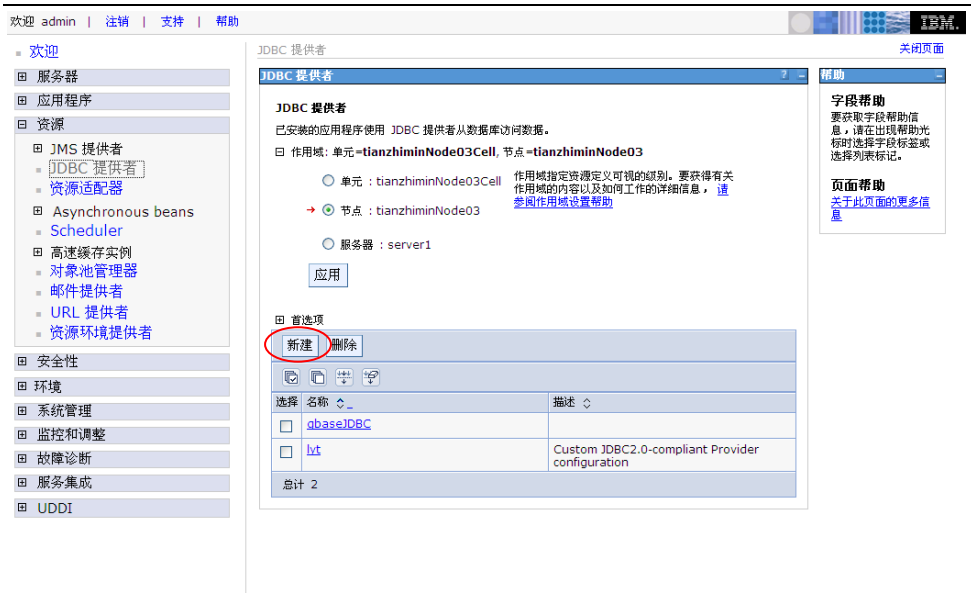

WebSphere Application Server — 第一步 - AppSrv03
WebSphere Application Server
第一步
安装验证
确认您的服务器是否已安装且是否能正确地启动。
启动服务器
启动服务器及其应用程序。
管理控制台
安装和管理应用程序。
"概要表创建"向导
创建概要表。
样本收藏夹
查看运行中的 WebSphere Application Server。
WebSphere Application Server 信息中心
了解更多有关 WebSphere Application Server 的信息。
"迁移"向导
将 WebSphere Application Server V4 或 V5 迁移到 V6.0。
退出

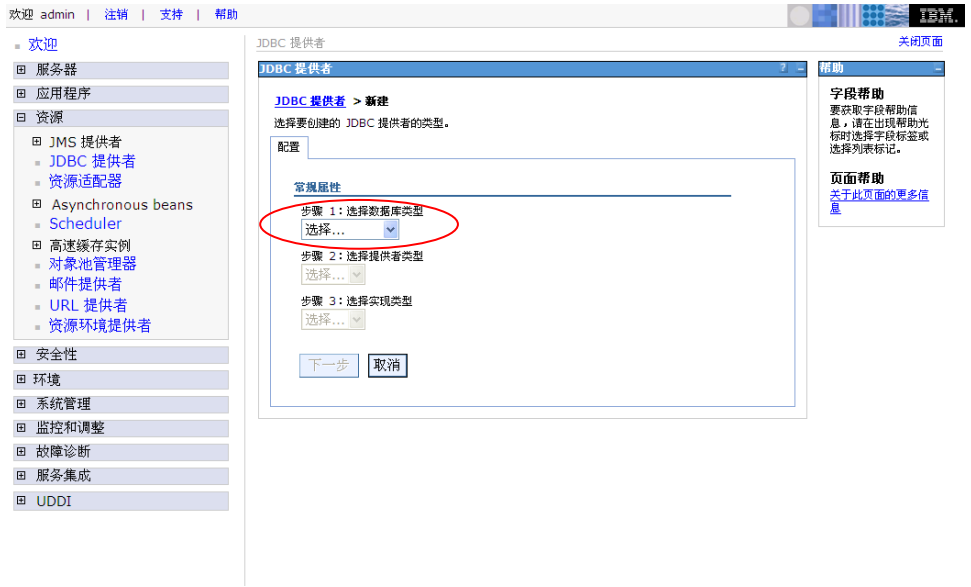

WebSphere Application Server — 第一步 — AppSrv03
WebSphere Application Server
IBM
第一步
安装验证
确认您的服务器是否已安装且是否能正确地启动。
停止服务器
停止服务器及其应用程序。
管理控制台
安装和管理应用程序。
"概要表创建"向导
创建概要表。
样本收藏夹
查看运行中的 WebSphere Application Server。
WebSphere Application Server 信息中心
了解更多有关 WebSphere Application Server 的信息。
"迁移"向导
将 WebSphere Application Server V4 或 V5 迁移到 V6.0。
退出


IBM
欢迎，请输入您的信息。
登录

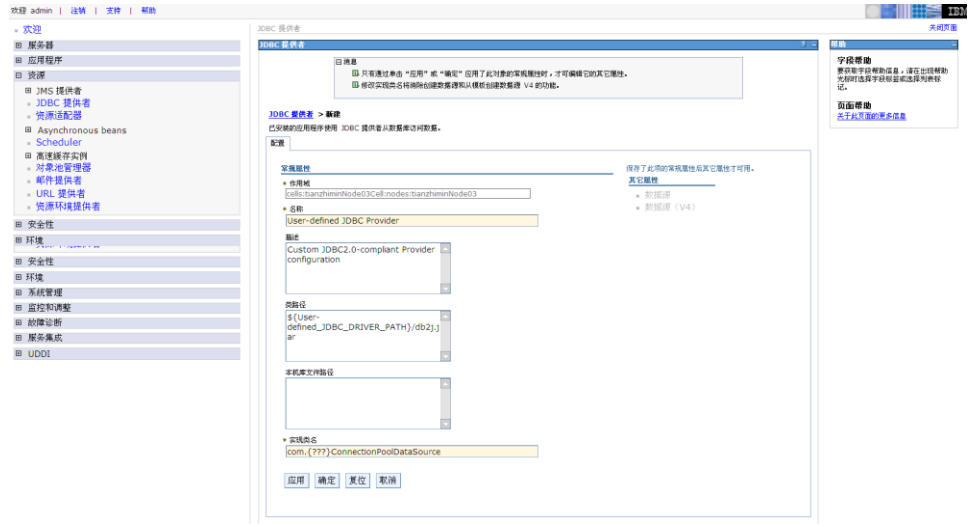

欢迎 admin | 注销 | 支持 | 帮助
IBM.
欢迎
服务器
应用程序
资源
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
普通管理任务
WebSphere Application Server 备忘单指导您完成普通管理任务。 当前没有安装备忘单。 请访问 WebSphere Application Server Support 以浏览并下载可用的备忘单。
关于 您的 WebSphere Application Server
IBM WebSphere Application Server - ND, 6.0.0.1
构建号： o0445.08
构建日期： 11/10/04
Licensed Material - Property of IBM
IBM.com 上的 WebSphere Application Server
在 IBM.com 上的产品信息中查找有关 WebSphere 软件系列的信息。OS/400 信息可以在 WebSphere Application Server for OS/400 产品 Web 站点上找到。
文档
要获取包括文章和 PDF 文件在内的文档，请访问在线信息中心。OS/400 用户可以在 WebSphere Application Server OS/400 Documentation Web 站点上找到此信息。
developerWorks WebSphere
可在 WebSphere Application Server Zone 上获取最新的技术文章、最佳实践、教程和更多信息。影响 WebSphere Application Server 的发展和请求新的产品功能部件。


欢迎 admin | 注销 | 支持 | 帮助
IBM.
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
普通管理任务
WebSphere Application Server 备忘单指导您完成普通管理任务。 当前没有安装备忘单。 请访问 WebSphere Application Server Support 以浏览并下载可用的备忘单。
关于 您的 WebSphere Application Server
IBM WebSphere Application Server - ND, 6.0.0.1
构建号： o0445.08
构建日期： 11/10/04
Licensed Material - Property of IBM
IBM.com 上的 WebSphere Application Server
在 IBM.com 上的产品信息中查找有关 WebSphere 软件系列的信息。OS/400 信息可以在 WebSphere Application Server for OS/400 产品 Web 站点上找到。
文档
要获取包括文章和 PDF 文件在内的文档，请访问在线信息中心。OS/400 用户可以在 WebSphere Application Server OS/400 Documentation Web 站点上找到此信息。
developerWorks WebSphere
可在 WebSphere Application Server Zone 上获取最新的技术文章、最佳实践、教程和更多信息。影响 WebSphere Application Server 的发展和请求新的产品功能部件。


欢迎 admin | 注销 | 支持 | 帮助
IBM.
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
已安装的应用程序使用 JDBC 提供者从数据库访问数据。
作用域：单元=tianzhiminNode03Cell, 节点=tianzhiminNode03
单元：tianzhiminNode03Cell
节点：tianzhiminNode03
服务器：server1
作用域指定资源定义可视的级别。要获得有关作用域的内容以及如何工作的详细信息，请参阅作用域设置帮助
应用
首选项
新建
删除
选择
名称
描述
qbaseJDBC
lvt
Custom JDBC2.0-compliant Provider configuration
总计 2
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
IBM.
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者 > 新建
选择要创建的 JDBC 提供者的类型。
配置
常规属性
步骤 1：选择数据库类型
选择...
步骤 2：选择提供者类型
选择...
步骤 3：选择实现类型
选择...
下一步
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息

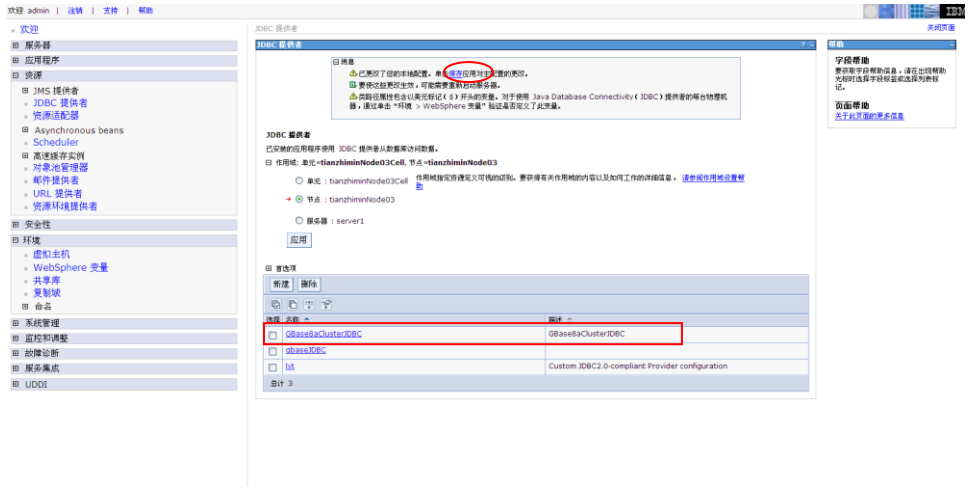

欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
JDBC 提供者 > 新建
选择要创建的 JDBC 提供者的类型。
配置
常规属性
步骤 1：选择数据库类型
用户定义的
步骤 2：选择提供者类型
User-defined JDBC Provider
步骤 3：选择实现类型
用户定义的
下一步
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


JDBC 提供者 > 新建
配置
常规属性
User-defined JDBC Provider
Custom JDBC2.0-compliant Provider configuration
com.{???}ConnectionPoolDataSource
应用
确定
复位
取消

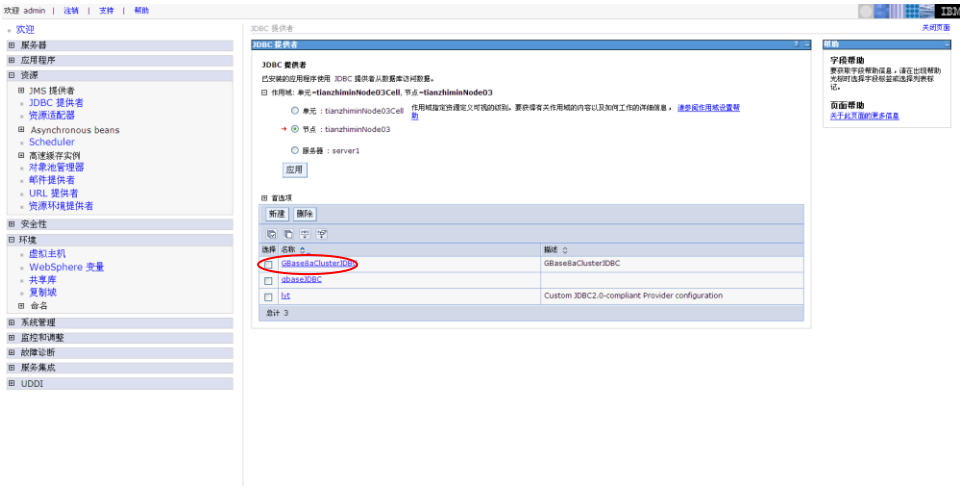
实现类名：

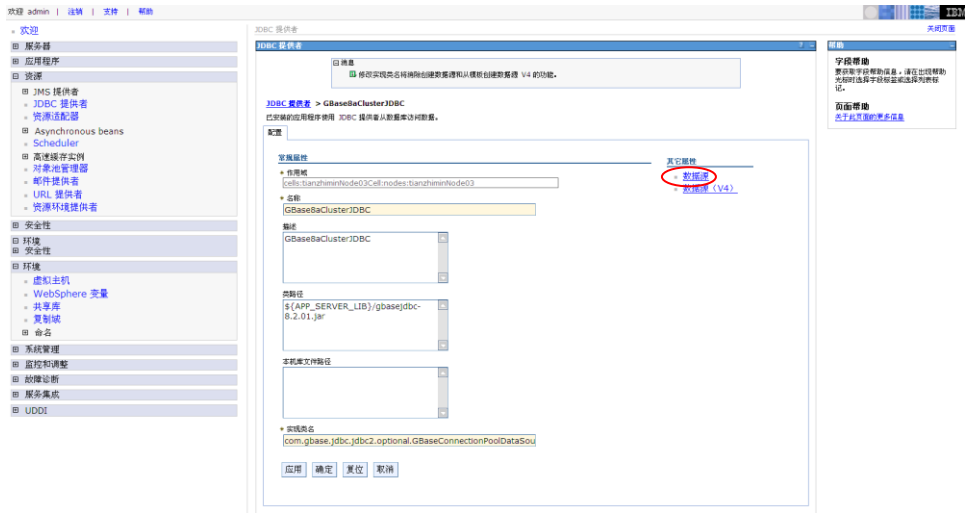
com.gbase.jdbc.jdbc2.optional.GBaseConnectionPoolDataSource

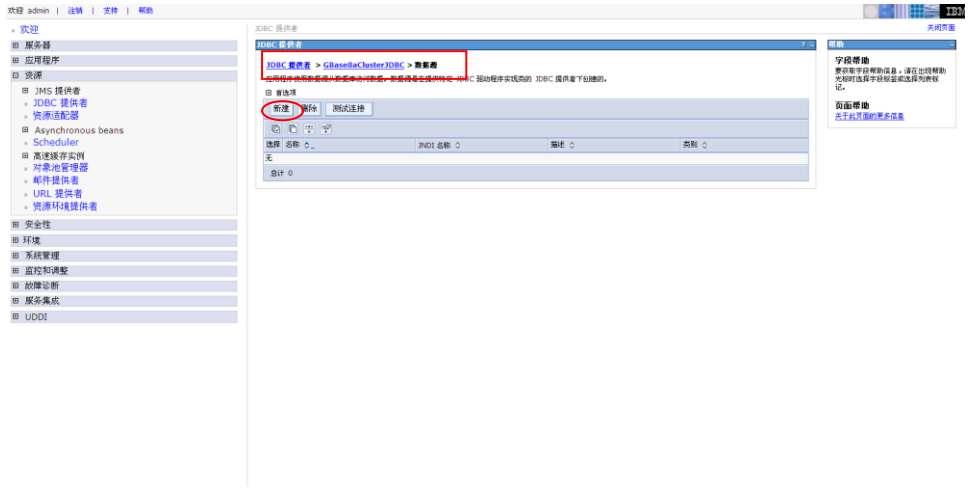
实现类路径：${APP_SERVER_LIB}/gbasejdbc-8.2.01.jar

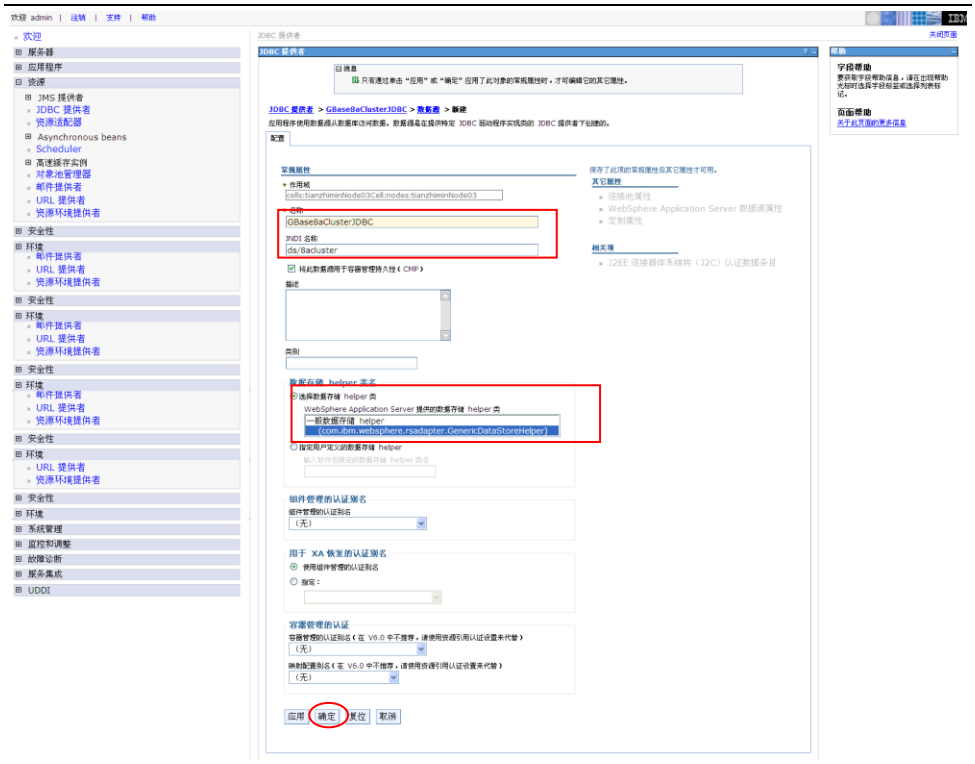



欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
虚拟主机
WebSphere 变量
共享库
复制域
命名
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者 > 保存
将工作空间更改保存到主配置
单击“保存”，以用您的更改更新主资源库。单击“放弃”放弃您的更改，并使用主资源库配置再次开始工作。单击“取消”，以继续使用您的更改。
总计更改的文档： 1
保存
放弃
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
虚拟主机
WebSphere 变量
共享库
复制域
命名
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
已安装的应用程序使用 JDBC 提供者从数据库访问数据。
作用域: 单元=tianzhiminNode03Cell, 节点=tianzhiminNode03
单元：tianzhiminNode03Cell
节点：tianzhiminNode03
服务器：server1
应用
首选项
新建
删除
选择
名称
描述
GBase8aClusterJDBC
GBase8aClusterJDBC
GBase8aJDBC
lut
Custom JDBC2.0-compliant Provider configuration
总计 3
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
虚拟主机
WebSphere 变量
共享库
复制域
命名
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
消息
修改实现类名将移除创建数据源和从模板创建数据源 V4 的功能。
JDBC 提供者 > GBaseBaClusterJDBC
已安装的应用程序使用 JDBC 提供者从数据库访问数据。
配置
常规属性
其它属性
数据源
数据源（V4）
作用域
cells:tianzhiminNode03Cell:nodes:tianzhiminNode03
名称
GBaseBaClusterJDBC
描述
GBaseBaClusterJDBC
类路径
${APP_SERVER_LIB}/gbasejdbc-8.2.01.jar
本机库文件路径
实现类名
com.gbase.jdbc.jdbc2.optional.GBaseConnectionPoolDataSou
应用
确定
复位
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息

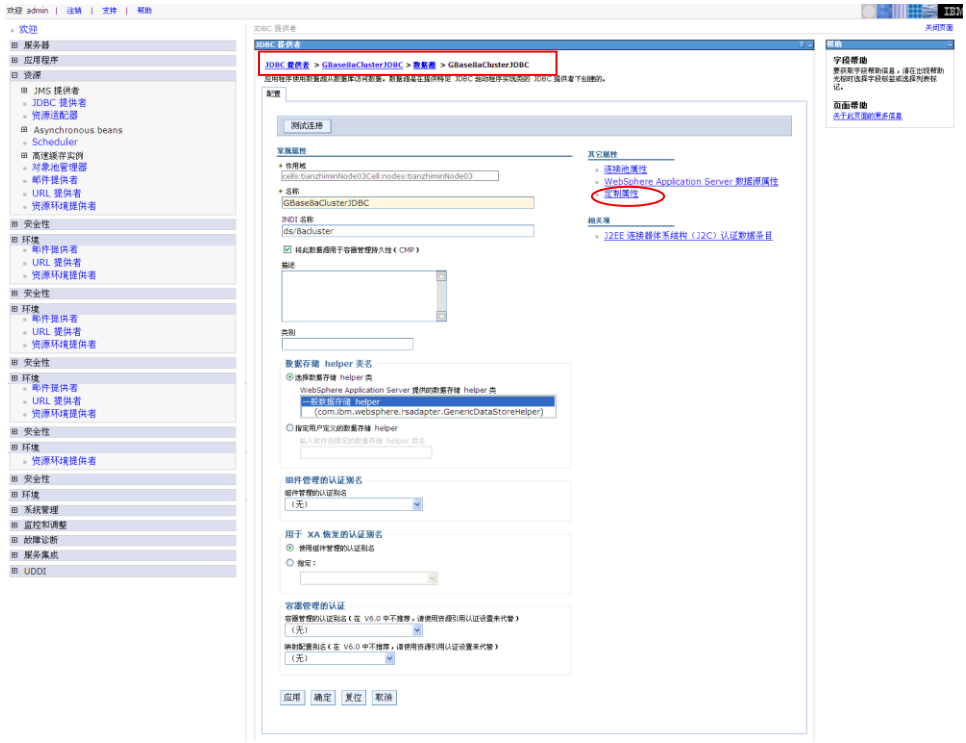

欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者 > GBaseBaClusterJDBC > 数据源
首选项
新建
删除
测试连接
选择
名称
JNDI 名称
描述
类别
无
总计 0
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息

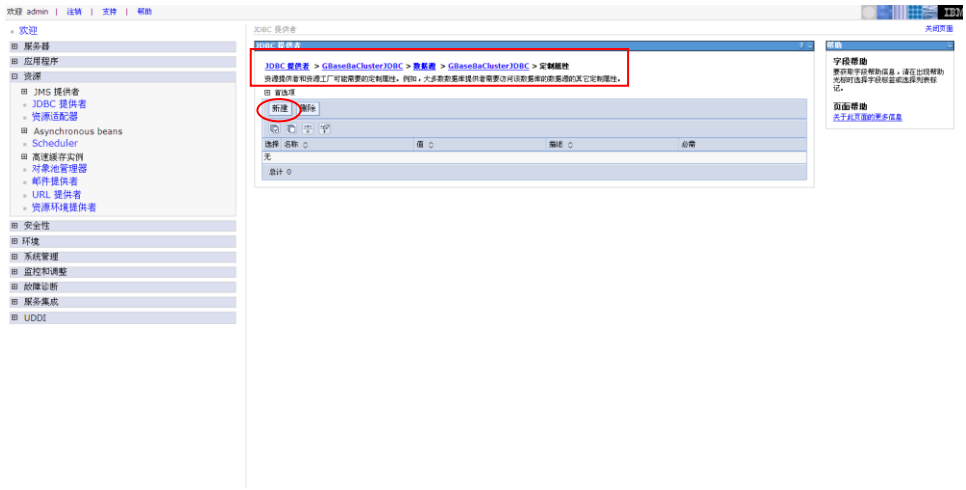

欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
消息
只有通过单击 "应用" 或 "确定" 应用了此对象的常规属性时，才可编辑它的其它属性。
JDBC 提供者 > GBaseBaClusterJDBC > 数据源 > 新建
应用程序使用数据源从数据库访问数据。数据源是在提供特定 JDBC 驱动程序实现类的 JDBC 提供者下创建的。
配置
常规属性
* 作用域
cells:tianzhiminNode03Cell:nodes:tianzhiminNode03
* 名称
GBaseBaClusterJDBC
JNDI 名称
ds/Baduster
将此数据源用于容器管理持久性 (CMP)
描述
类别
数据存储 helper 类名
选择数据存储 helper 类
WebSphere Application Server 提供的数据存储 helper 类
一般数据存储 helper
(com.ibm.websphere.rsadapter.GenericDataStoreHelper)
指定用户定义的数据存储 helper
输入软件包限定的数据存储 helper 类名
组件管理的认证别名
组件管理的认证别名
(无)
用于 XA 恢复的认证别名
使用组件管理的认证别名
指定：
容器管理的认证
容器管理的认证别名 (在 V6.0 中不推荐，请使用资源引用认证设置来代替)
(无)
映射配置别名 (在 V6.0 中不推荐，请使用资源引用认证设置来代替)
(无)
应用
确定
复位
取消
常规属性
其它属性
连接池属性
WebSphere Application Server 数据源属性
定制属性
相关项
J2EE 连接器体系结构 (J2C) 认证数据条目
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息

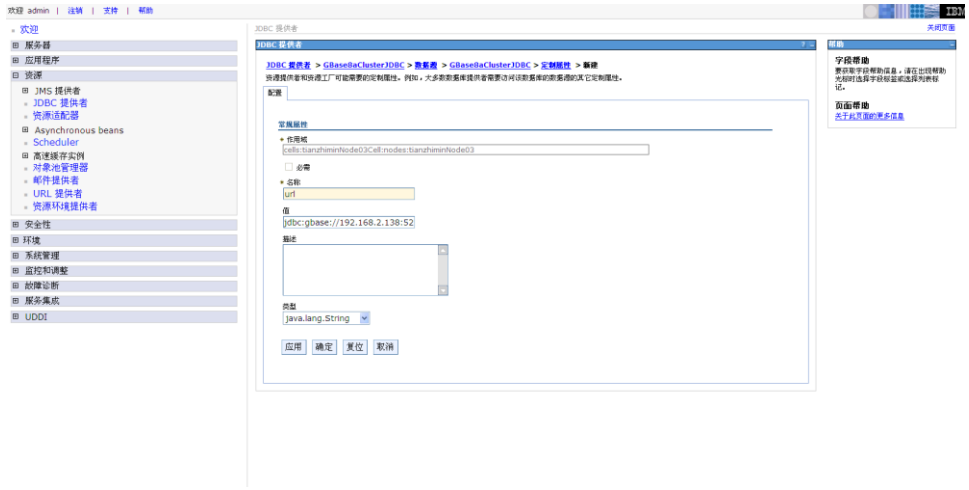

欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
消息
已更改了您的本地配置。单击保存应用对配置的更改。
要使这些更改生效，可能需要重新启动服务器。
GenericDataStoreHelper 类不包含任何错误异常映射，所以它可能不像应用程序所期望的那样执行。
JDBC 提供者 > GBaseBaClusterJDBC > 数据源
应用程序使用数据源从数据库访问数据。数据源是在提供特定 JDBC 驱动程序实现类的 JDBC 提供者下创建的。
首选项
新建
删除
测试连接
选择 名称
JNDI 名称
描述
类别
GBaseBaClusterJDBC
ds/Baduster
总计 1
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息

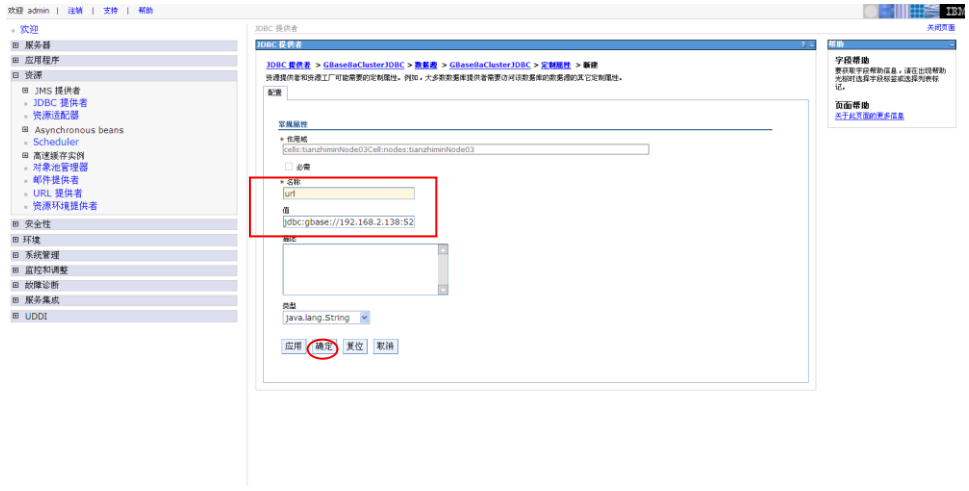

欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
JDBC 提供者 > GBase8aClusterJDBC > 数据源 > 保存
将工作空间更改保存到主配置
单击“保存”，以将您的更改更新主资源库。单击“放弃”放弃您的更改，并使用主资源库配置再次开始工作。单击“取消”，以继续使用您的更改。
总计更改的文档：1
保存
放弃
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
JDBC 提供者 > GBase8aClusterJDBC > 数据源
应用程序使用数据源从数据库访问数据，数据源是在提供特定 JDBC 驱动程序实现类的 JDBC 提供者下创建的。
首选项
新建
删除
测试连接
选择
名称
JNDI 名称
描述
类别
GBase8aClusterJDBC
ds/8aduster
总计 1
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者 > GBaseBaClusterJDBC > 数据源 > GBaseBaClusterJDBC
配置
测试连接
常规属性
作用域
cells:tianzhiminNode03Cell:nodes:tianzhiminNode03
名称
GBaseBaClusterJDBC
JNDI 名称
ds/Bacluster
将此数据源用于容器管理持久性 (CMP)
描述
类别
数据存储 helper 类名
选择数据存储 helper 类
WebSphere Application Server 提供的数据存储 helper 类
一般数据存储 helper
(com.ibm.websphere.rsadapter.GenericDataStoreHelper)
指定用户定义的数据存储 helper
组件管理的认证别名
(无)
用于 XA 恢复的认证别名
使用组件管理的认证别名
指定:
容器管理的认证
容器管理的认证别名 (在 V6.0 中不推荐，请使用资源引用认证首选来代替)
(无)
映射配置别名 (在 V6.0 中不推荐，请使用资源引用认证首选来代替)
(无)
其它属性
连接池属性
WebSphere Application Server 数据源属性
定制属性
相关项
J2EE 连接器体系结构（J2C）认证数据条目
应用
确定
复位
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者 > GBaseBaClusterJDBC > 数据源 > GBaseBaClusterJDBC > 定制属性
指定提供者和资源工厂可能需要的定制属性。例如，大多数数据库提供者需要访问该数据库的数据源的其它定制属性。
首选项
新建
删除
选择
名称
值
描述
必需
无
总计 0
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息





名称：url

值：

jdbc:gbase://192.168.2.138:5258/gbase?user=gbase&password=gbase20110531

如果需要设置 GBase 8a JDBC 其他属性只需新建一个定制属性，分别把属性名称和值写入到“名称”和“值”中即可。

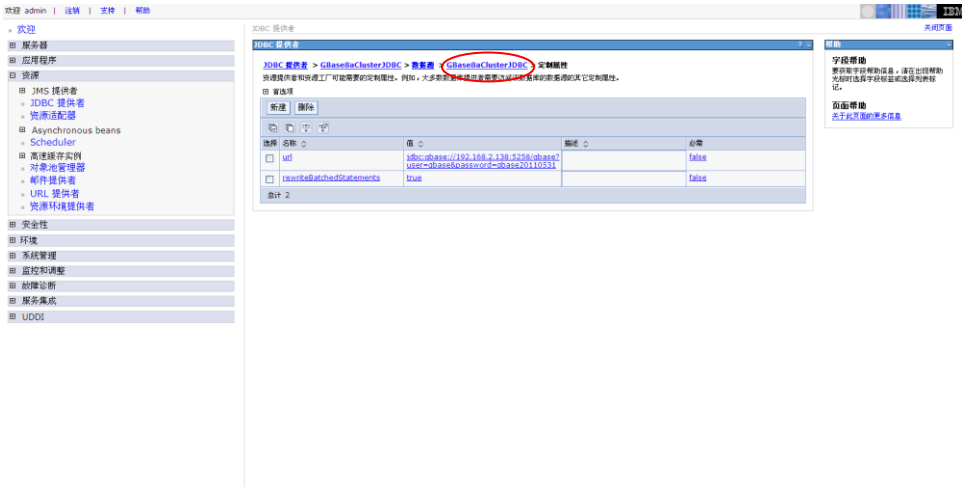
比如下面我们要设置 rewriteBatchedStatements=true 这个属性可以按照下面的步骤：

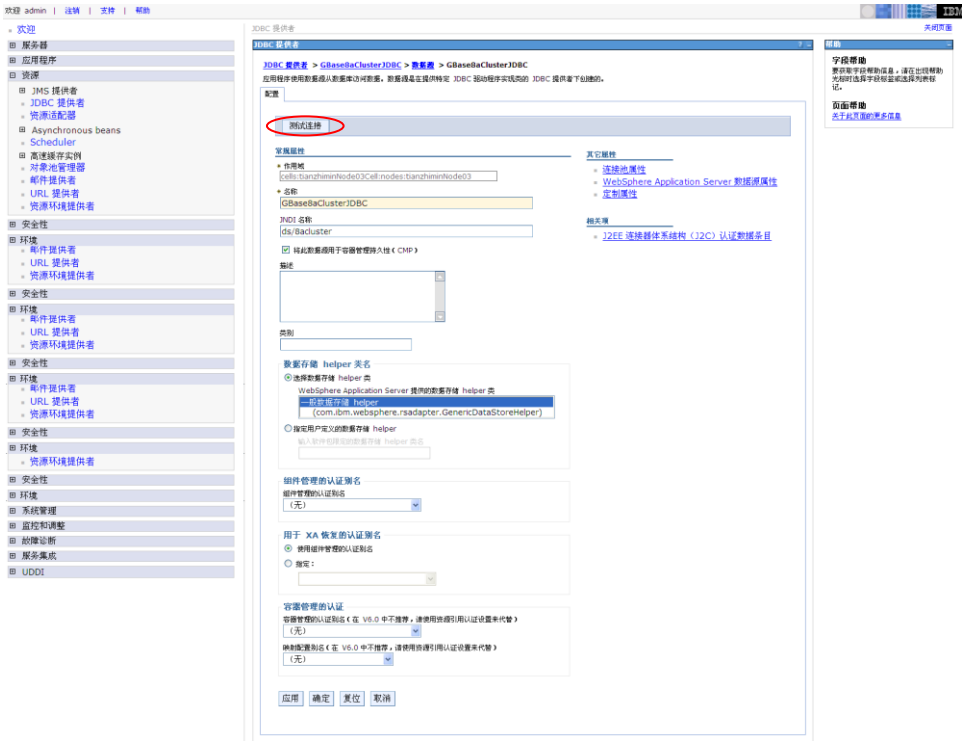

欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
消息
已更改了您的本地配置。单击保存应用对主配置的更改。
要使这些更改生效，可能需要重新启动服务器。
JDBC 提供者 > GBaseBaClusterJDBC > 数据源 > GBaseBaClusterJDBC > 定制属性 > 新建
资源提供者和资源工厂可能需要的定制属性。例如，大多数数据库提供者需要访问该数据库的数据源的其它定制属性。
配置
常规属性
作用域
cells:tianzhiminNode03Cell:nodes:tianzhiminNode03
必需
名称
值
描述
类型
java.lang.String
应用
确定
复位
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息

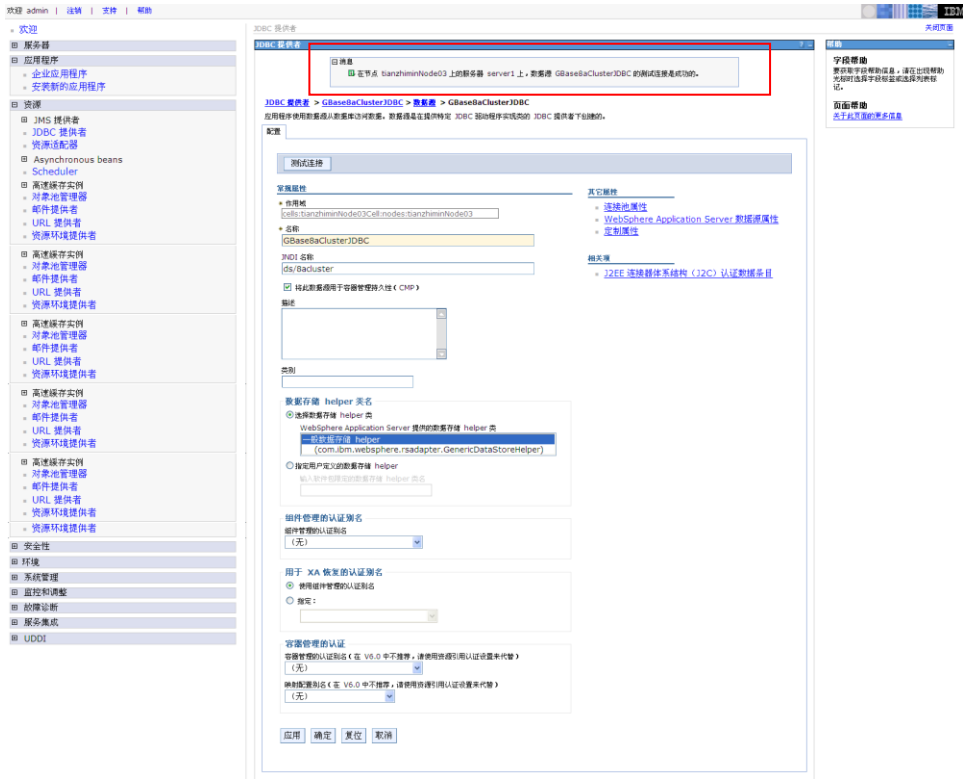

欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
消息
已更改了您的本地配置。单击保存应用对主配置的更改。
要使这些更改生效，可能需要重新启动服务器。
JDBC 提供者 > GBaseBaClusterJDBC > 数据源 > GBaseBaClusterJDBC > 定制属性 > 新建
资源提供者和资源工厂可能需要的定制属性。例如，大多数数据库提供者需要访问该数据库的数据源的其它定制属性。
配置
常规属性
作用域
cells:tianzhiminNode03Cell:nodes:tianzhiminNode03
必需
名称
rewriteBatchedStatements
值
true
描述
类型
java.lang.String
应用
确定
复位
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者
消息
已更改了您的本地配置。单击重新应用对主配置的更改。
要使这些更改生效，可能需要重新启动服务器。
JDBC 提供者 > GBase8aClusterJDBC > 数据源 > GBase8aClusterJDBC > 定制属性
资源提供者和资源工厂可能需要的定制属性。例如，大多数数据库提供者需要访问该数据库的数据源的其它定制属性。
首选项
新建
删除
选择 名称 值 描述 必需
url jdbc:gbase://192.168.2.138:5258/gbase?user=gbase&password=gbase20110531 false
rewriteBatchedStatements true false
总计 2
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
关闭页面
JDBC 提供者 > GBase8aClusterJDBC > 数据源 > GBase8aClusterJDBC > 定制属性 > 保存
将工作空间更改保存到主配置
单击“保存”，以用您的更改更新主资源库。单击“放弃”放弃您的更改，并使用主资源库配置再次开始工作。单击“取消”，以继续使用您的更改。
总计更改的文档： 1
保存
放弃
取消
帮助
字段帮助
要获取字段帮助信息，请在出现帮助光标时选择字段标签或选择列表标记。
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
欢迎
服务器
应用程序
资源
JMS 提供者
JDBC 提供者
资源适配器
Asynchronous beans
Scheduler
高速缓存实例
对象池管理器
邮件提供者
URL 提供者
资源环境提供者
安全性
环境
系统管理
监控和调整
故障诊断
服务集成
UDDI
JDBC 提供者
JDBC 提供者 > GBaseBaCluster3JDBC > 数据源 > GBaseBaCluster3JDBC 定制属性
首选项
新建
删除
选择 名称 值 描述 必需
url jdbc:gbase://192.168.2.138:5258/gbase?user=gbase&password=gbase20110531 false
rewriteBatchedStatements true false
总计 2
帮助
字段帮助
页面帮助
关于此页面的更多信息


欢迎 admin | 注销 | 支持 | 帮助
JDBC 提供者
JDBC 提供者 > GBaseBaCluster3JDBC > 数据源 > GBaseBaCluster3JDBC
配置
测试连接
常规属性
作用域
cells:tianzhiminNode03Cell:nodes:tianzhiminNode03
名称
GBaseBaCluster3JDBC
JNDI 名称
ds/Bacluster
将此数据源用于容器管理持久性 (CMP)
描述
类别
数据存储 helper 类名
选择数据存储 helper 类
WebSphere Application Server 提供的数据存储 helper 类
一般数据存储 helper
(com.ibm.websphere.rsadapter.GenericDataStoreHelper)
指定用户定义的数据存储 helper
组件管理的认证别名
组件管理的认证别名
(无)
用于 XA 恢复的认证别名
使用组件管理的认证别名
指定:
容器管理的认证
容器管理的认证别名 (在 V6.0 中不推荐，请使用资源引用认证设置来代替)
(无)
映射配置别名 (在 V6.0 中不推荐，请使用资源引用认证设置来代替)
(无)
应用
确定
复位
取消
其它属性
连接池属性
WebSphere Application Server 数据源属性
定制属性
相关项
J2EE 连接器体系结构 (J2C) 认证数据条目
帮助
字段帮助
页面帮助
关于此页面的更多信息



5.4.2 程序验证

编写如下 Serverlet，到处 war 包，部署到 websphere 上。

```
package com.gbase.test;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map;
```

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class WebshpereJNDITest extends HttpServlet {

    /**
    *
    */
    private static final long serialVersionUID =
8570136076952105578L;

    /**
    * Constructor of the object.
    */
    public WebshpereJNDITest() {
        super();
    }

    /**
    * Destruction of the servlet. <br>
    */
    public void destroy() {
        super.destroy(); // Just puts "destroy" string in log
        // Put your code here
    }

    /**
    * The doGet method of the servlet. <br>
    *
```

```
        * This method is called when a form has its tag value method equals
to get.
        *
        * @param request the request send by the client to the server
        * @param response the response send by the server to the client
        * @throws ServletException if an error occurred
        * @throws IOException if an error occurred
        */
        public void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            Connection conn = null;
            Statement stmt = null;
            ResultSet rs = null;
            try {
                System.out.println("come");
                Context initCtx = new InitialContext();
                // 获得连接池
                DataSource ds = (DataSource) initCtx
                                    .lookup("ds/8acluster");
                Map cacheMap = new HashMap();
                // 创建连接
                conn = ds.getConnection();

                System.out.println(conn.getMetaData().getURL());
                if (conn != null) {
                    out.println("The Gbase connection is ok!!");
                    System.out.println("ok");
                } else {
                    out.println("The Gbase connection occur error");
                    System.out.println("error");
                }
                // 使用连接创建 Statement 对象执行 SQL 语句
```

```
                Statement st = conn.createStatement();
                ResultSet rs1 = st
    .executeQuery("select * from information_schema.TABLES");
                ResultSetMetaData rsm = rs1.getMetaData();
                // 获得执行结果，输出结果
                while (rs1.next()) {
                    for (int i = 0; i < rsm.getColumnCount(); i++) {
                        out.println(rsm.getColumnName(i + 1)+ "= "+
rs1.getString(i + 1));
                        out.println("<br/>");
                    }
                    out.println("++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++<br/>");
                }
            } catch (Exception e) {
                e.printStackTrace();
                out.println(e);
            } finally {
                out.flush();
                out.close();
                if (conn == null) {
                    try {
                        conn.close();
                    } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }
        }

        public String getCurrentIp(String url) {
            return  url.substring(url.indexOf("currentIp"),
url.length());
        }
        /**
        * The doPost method of the servlet. <br>
```

```
        *
        * This method is called when a form has its tag value method equals
to post.
        *
        * @param request the request send by the client to the server
        * @param response the response send by the server to the client
        * @throws ServletException if an error occurred
        * @throws IOException if an error occurred
        */
        public void doPost(HttpServletRequest request,
HttpServletResponse response)
                throws ServletException, IOException {

            doGet(request, response);
        }

        /**
        * Initialization of the servlet. <br>
        *
        * @throws ServletException if an error occurs
        */
        public void init() throws ServletException {
            // Put your code here
        }
    }
```

web.xml 配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
```

```
    <servlet-name>websphereJNDITest</servlet-name>
<servlet-class>com.gbase.test.WebshpereJNDITest</servlet-cl
ass>
  </servlet>

  <servlet-mapping>
    <servlet-name>websphereJNDITest</servlet-name>
    <url-pattern>/websphereJNDITest</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
    </web-app>
```

在部署测试 war 包是 “上下文根”使用 “/JNDITest”

部署成功后使用如下 URL 访问:

http://localhost:9082/JDNITest/websphereJNDITest

该测试用例会打印出目标数据库中 information_schema.TABLES 的内容。

5.5weblogic 12c 配置 JNDI

本小节介绍 oracle weblogic 12c 如何通过 JNDI 配置连接池。该版本支持 jdk1.6，所以我们把 gbase-connector-java-8.3.81.51-build1.0-bin.jar 作为 jdbc 驱动。

准备工作：把 gbase-connector-java-8.3.81.51-build1.0-bin.jar 拷贝到 weblogic 12c 的安装目录，如 F:\Oracle\Middleware\user_projects\domains\base_domain\lib 下。

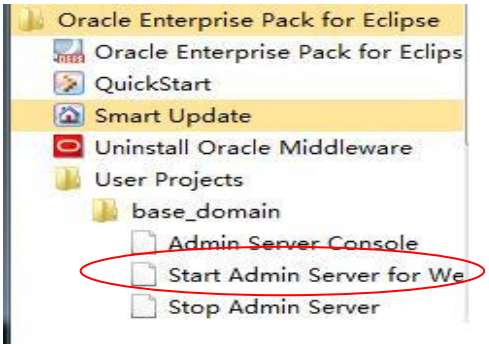
约定:

1、 方框表示需要注意的地方

2、 椭圆表示鼠标单击

5.5.1配制方法

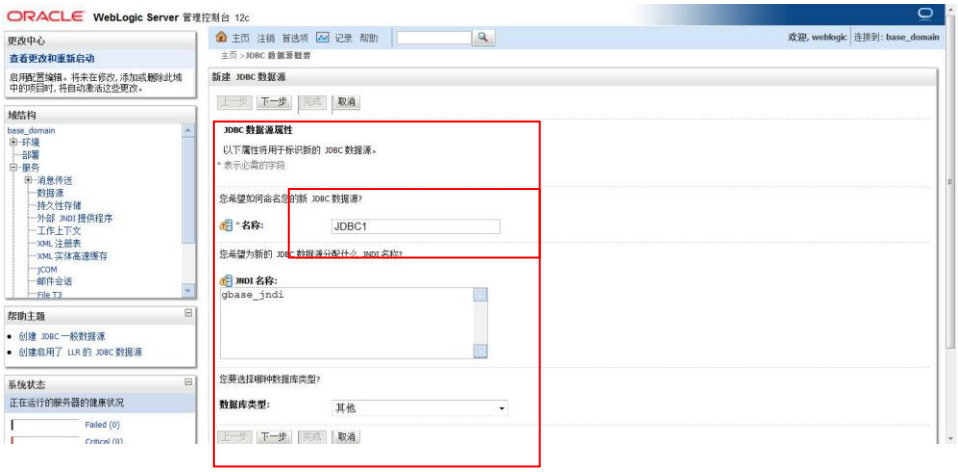
从开始菜单中找到“Oracle Enterprise Pack for Eclipse”，按照下面进行选择：




ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
base_domain
帮助主题
系统状态
主页
信息和资源
域配置

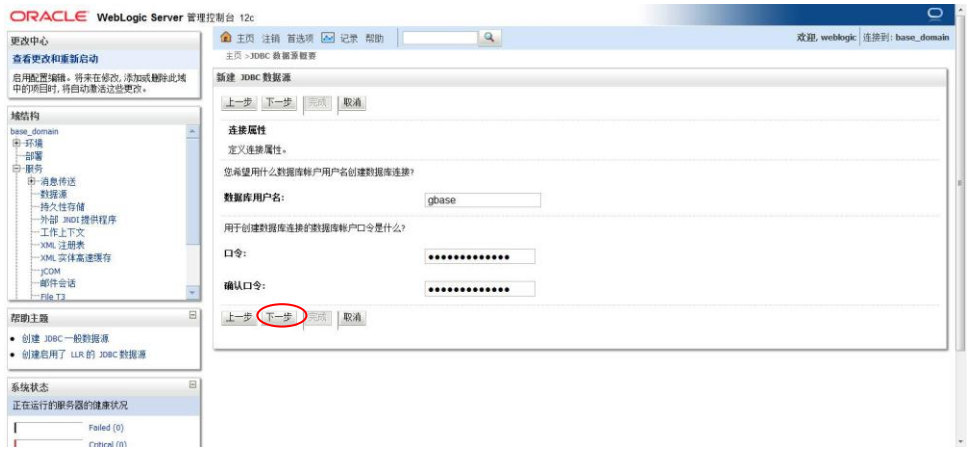

ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
帮助主题
系统状态
JDBC 数据源概要
配置
监视
定制此表
数据源（已筛选 - 更多列存在）
名称
类型
JNDI 名称
目标
JDBC0
一般
gbase
AdminServer
Mysql
一般
mysql
AdminServer

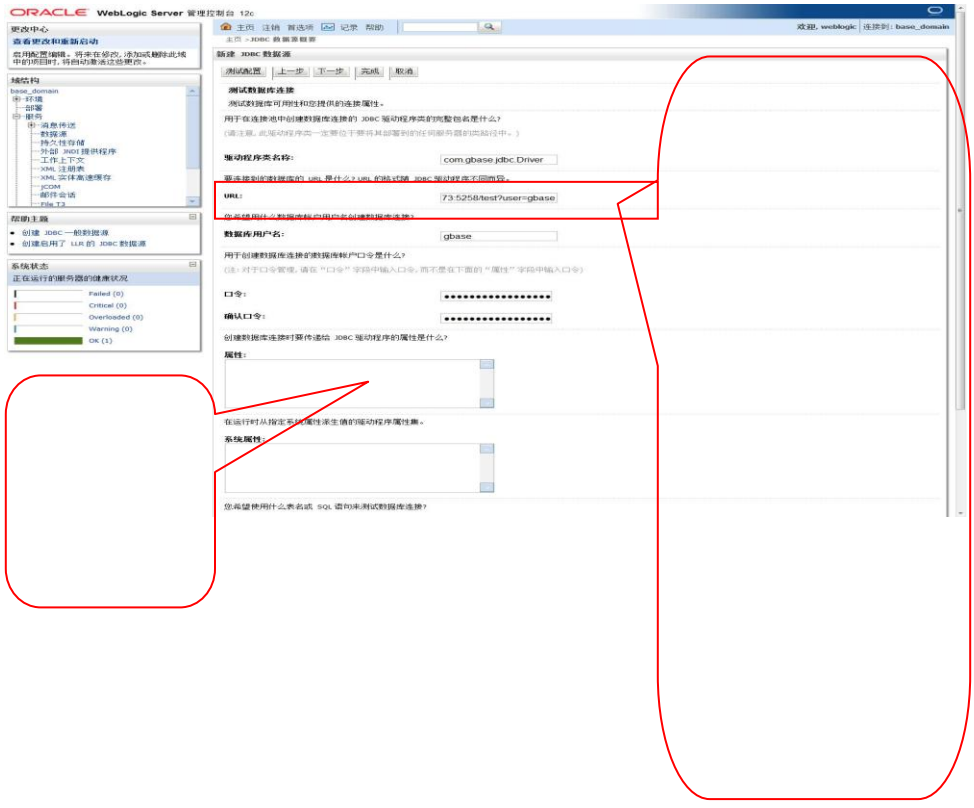

ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
base_domain
环境
部署
服务
消息传送
数据源
持久性存储
外部 JNDI 提供程序
工作上下文
XML 注册表
XML 实体高速缓存
jCOM
邮件会话
File T3
帮助主题
创建 JDBC 一般数据源
创建 JDBC GridLink 数据源
创建 JDBC 多数据源
删除 JDBC 数据源
删除 JDBC 多数据源
系统状态
正在运行的服务器的健康状况
Failed (0)
主页 注销 首选项 记录 帮助
欢迎, weblogic 连接到: base_domain
主页 >JDBC 数据源摘要
JDBC 数据源摘要
配置 监视
JDBC 数据源是指与 JNDI 树 (通过一组 JDBC 连接提供数据库连接) 绑定的对象。应用程序可在 JNDI 树上查找数据源, 然后借用一个与数据源的数据库连接。
此页概括了已在该域中创建的 JDBC 数据源对象。
定制此表
数据源 (已筛选 - 更多列存在)
新建
删除
显示 1 到 2 个, 共 2 个 上一个 | 下一个
一般数据源
GridLink 数据源
多数据源
名称
类型
JNDI 名称
目标
gbase
一般
gbase
AdminServer
Mysql
一般
mysql
AdminServer
新建
删除
显示 1 到 2 个, 共 2 个 上一个 | 下一个

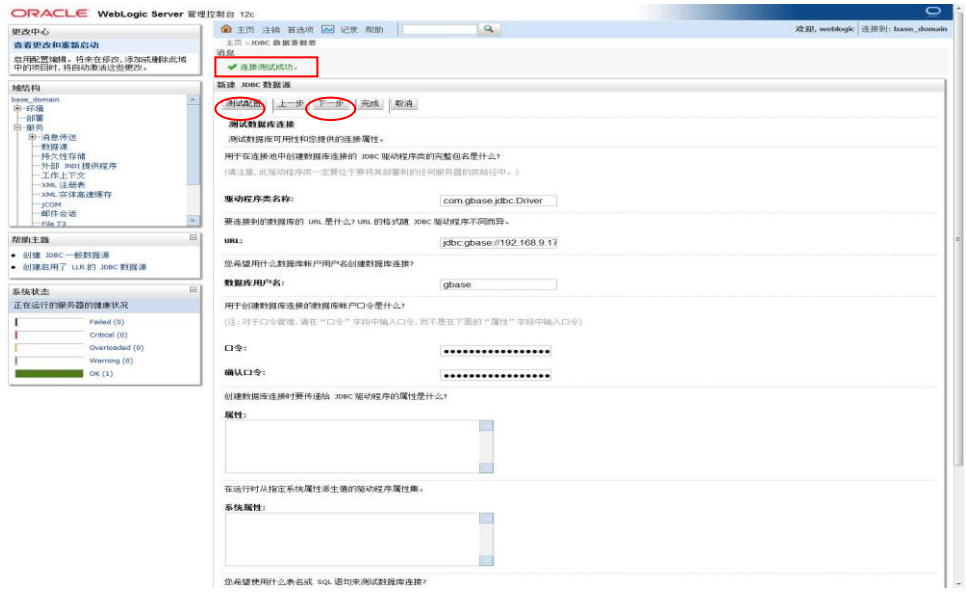

ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
base_domain
环境
部署
服务
消息传送
数据源
持久性存储
外部 JNDI 提供程序
工作上下文
XML 注册表
XML 实体高速缓存
jCOM
邮件会话
File T3
帮助主题
创建 JDBC 一般数据源
创建启用了 LLR 的 JDBC 数据源
系统状态
正在运行的服务器的健康状况
Failed (0)
Critical (0)
主页 注销 首选项 记录 帮助
欢迎, weblogic 连接到: base_domain
主页 >JDBC 数据源摘要
新建 JDBC 数据源
上一步 下一步 完成 取消
JDBC 数据源属性
以下属性将用于标识新的 JDBC 数据源。
* 表示必需的字段
您希望如何命名您的新 JDBC 数据源?
* 名称:
JDBC1
您希望为新的 JDBC 数据源分配什么 JNDI 名称?
JNDI 名称:
gbase_jndi
您要选择哪种数据库类型?
数据库类型:
其他
上一步 下一步 完成 取消

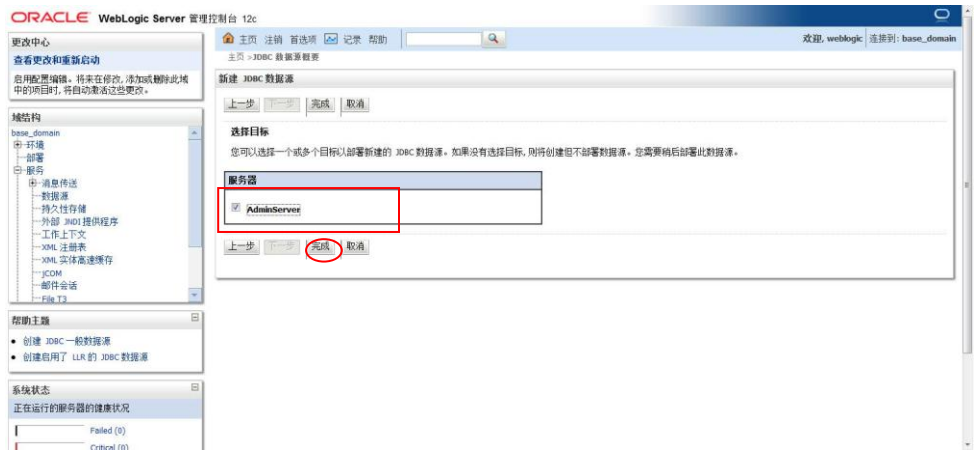

ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
帮助主题
系统状态
新建 JDBC 数据源
上一步
下一步
完成
取消
JDBC 数据源属性
以下属性将用于标识新的 JDBC 数据源。
数据库类型:
Other
数据库驱动程序:
其他


ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
帮助主题
系统状态
新建 JDBC 数据源
上一步
下一步
完成
取消
事务处理选项
您已选择了非 XA JDBC 驱动程序，以在新数据源中创建数据库连接。
此数据源是否支持全局事务处理？如果支持，请为此数据源选择事务处理协议。
支持全局事务处理
记录上一个资源
仿真两阶段提交
一阶段提交


ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
帮助主题
系统状态
新建 JDBC 数据源
上一步
下一步
完成
取消
连接属性
定义连接属性。
您希望用什么数据库帐户用户名创建数据库连接?
数据库用户名:
gbase
用于创建数据库连接的数据库帐户口令是什么?
口令:
确认口令:

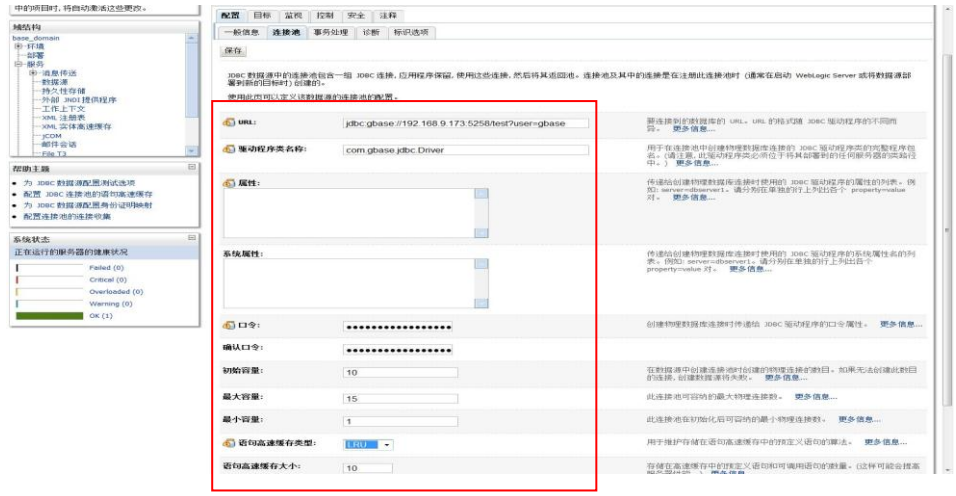

ORACLE WebLogic Server 管理控制台 12c
新建 JDBC 数据源
测试配置
测试数据库连接
驱动程序类名称:
com.gbase.jdbc.Driver
URL:
73:5258/test?user=gbase
数据库用户名:
gbase
口令:
确认口令:
属性:
系统属性:


ORACLE WebLogic Server 管理控制台 12c
连接测试成功。
新建 JDBC 数据源
测试连接
上一步
下一步
完成
取消
测试数据库连接
驱动程序类名称:
com.gbase.jdbc.Driver
URL:
jdbc:gbase://192.168.9.17
数据库用户名:
gbase
口令:
确认口令:
属性:
系统属性:


ORACLE WebLogic Server 管理控制台 12c
新建 JDBC 数据源
上一步
下一步
完成
取消
选择目标
您可以选择一个或多个目标以部署新建的 JDBC 数据源。如果没有选择目标，则将创建但不部署数据源。您需要稍后部署此数据源。
服务器
AdminServer


ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
base_domain
帮助主题
系统状态
主页 注销 首选项 记录 帮助
欢迎, weblogic 连接到: base_domain
JDBC 数据源概要
配置 监视
定制此表
数据源 (已筛选 - 更多列存在)
新建 删除
名称 类型 JNDI 名称 目标
JDBC0 一般 gbase AdminServer
JDBC1 一般 gbase_jndi AdminServer
Mysql 一般 mysql AdminServer


ORACLE WebLogic Server 管理控制台 12c
更改中心
查看更改和重新启动
域结构
base_domain
帮助主题
系统状态
主页 注销 首选项 记录 帮助
欢迎, weblogic 连接到: base_domain
JDBC1的设置
配置 目标 监视 控制 安全 注释
一般信息 连接池 事务处理 诊断 标识选项
保存
名称: JDBC1
JNDI 名称:
gbase_jndi
启用行预取
行预取大小: 48
流块大小: 256



当完成上述配置时，会在 F:\Oracle\Middleware\user_projects\domains\base_domain\config\jdbc 目录中自动产生数据源的配置文件 JDBC1-1894-jdbc.xml，其中的内容为(以上述配置为例)：

```
<?xml version='1.0' encoding='UTF-8'?>
<jdbc-data-source
xmlns="http://xmlns.oracle.com/weblogic/jdbc-data-source"
xmlns:sec="http://xmlns.oracle.com/weblogic/security"
xmlns:wls="http://xmlns.oracle.com/weblogic/security/wls"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/weblogic/jdbc-data-sourc
e
http://xmlns.oracle.com/weblogic/jdbc-data-source/1.2/jdbc-data-sour
ce.xsd">
    <name>JDBC1</name>
    <jdbc-driver-params>
      <url>jdbc:gbase://192.168.9.173:5258/test?user=gbase</url>
      <driver-name>com.gbase.jdbc.Driver</driver-name>
```

<password-encrypted>{AES}+NOjloveIb5I4AFsav1QJ/DIFMuXeCy/FwWrgovEnbQ=</password-encrypted>

</jdbc-driver-params>

<jdbc-connection-pool-params>

<initial-capacity>10</initial-capacity>

<test-table-name></test-table-name>

</jdbc-connection-pool-params>

<jdbc-data-source-params>

<jndi-name>gbase_jndi</jndi-name>

<global-transactions-protocol>OnePhaseCommit</global-transactions-protocol>

</jdbc-data-source-params>

</jdbc-data-source>

同时，会在 F:\Oracle\Middleware\user_projects\domains\base_domain\config\config.xml 中自动添加内容为:

<jdbc-system-resource>

<name>JDBC1</name>

<target>AdminServer</target>

<descriptor-file-name>jdbc/JDBC1-1894-jdbc.xml</descriptor-file-name>

</jdbc-system-resource>

5.5.2程序验证

编写如下 servlet，然后把项目部署到 weblogic 上。

```
package com.gbase.jndi;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
public class TestJNDIWithGBase extends HttpServlet {

    /**
    *
    */
    private static final long serialVersionUID = 1L;

    /**
    * The doGet method of the servlet. <br>
    *
    * This method is called when a form has its tag value method equals
to get.
    *
    * @param request the request send by the client to the server
    * @param response the response send by the server to the client
    * @throws ServletException if an error occurred
    * @throws IOException if an error occurred
```

```
        */
        public void doGet(HttpServletRequest request,
HttpServletResponse response)
                    throws ServletException, IOException {

            Connection conn = null;
            ResultSet rs = null;
            Statement st = null;
            try {
                Context ctx = new InitialContext();
                DataSource ds = (DataSource) ctx.lookup("gbase_jndi");
                conn = ds.getConnection();
                st = conn.createStatement();
                rs = st.executeQuery("select id,name,age,department
from emp");
                while(rs.next()){
                    System.out.println("员工 id:"+rs.getInt(1)+" 员工
姓名: "+rs.getString(2)+" 员工年龄: "+rs.getInt(3)+" 员工部门:
"+rs.getString(4));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }finally{
                if(rs != null){
                    try {
                        rs.close();
                        rs = null;
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    if(st != null){
                        try {
                            st.close();
                            st = null;
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
```

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

        /**
        * The doPost method of the servlet. <br>
        *
        * This method is called when a form has its tag value method equals
to post.
        *
        * @param request the request send by the client to the server
        * @param response the response send by the server to the client
        * @throws ServletException if an error occurred
        * @throws IOException if an error occurred
        */
        public void doPost(HttpServletRequest request,
HttpServletResponse response)
                    throws ServletException, IOException {

            this.doGet(request, response);
        }
    }
```

web.xml 配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <description>This is the description of my J2EE
component</description>
    <display-name>This is the display name of my J2EE
component</display-name>
    <servlet-name>TestJNDIWithGBase</servlet-name>

<servlet-class>com.gbase.jndi.TestJNDIWithGBase</servlet-cl
ass>
  </servlet>

  <servlet-mapping>
    <servlet-name>TestJNDIWithGBase</servlet-name>
    <url-pattern>/TestJNDIWithGBase</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
    </web-app>
```

当访问如下 URL 时:

http://localhost:7001/TestMysqlJNDIWithWeblogic/TestJNDIWithGBas
e

该测试用例会打印出数据库为 test 表为 emp 中的内容。

5.6GlassFish 配置 JNDI

本小节介绍 GlassFish v2.1.1 如何通过 JNDI 配置连接池。

准备工作：首先把驱动程序 gbase-connector-java-8.3.81.51-build1.0-bin.jar 拷贝到 glassfish 的安装目录/domains/domain1/lib/ext 下。

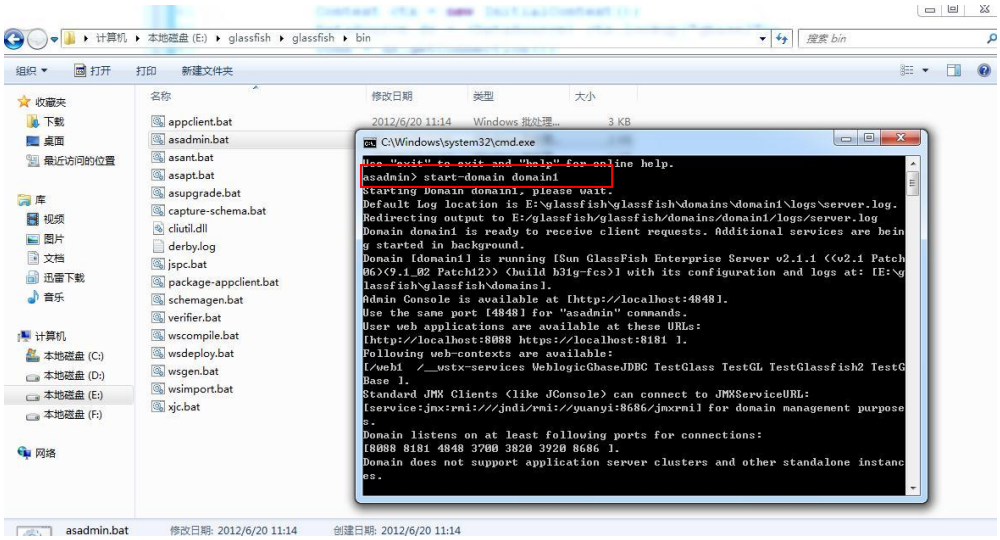
约定：

1、 方框表示需要注意的地方

2、 椭圆表示鼠标单击

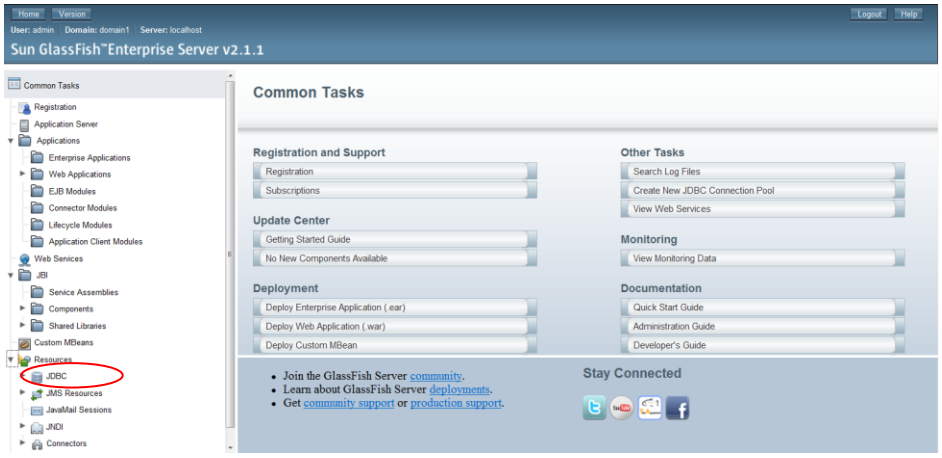
5.6.1 配制方法

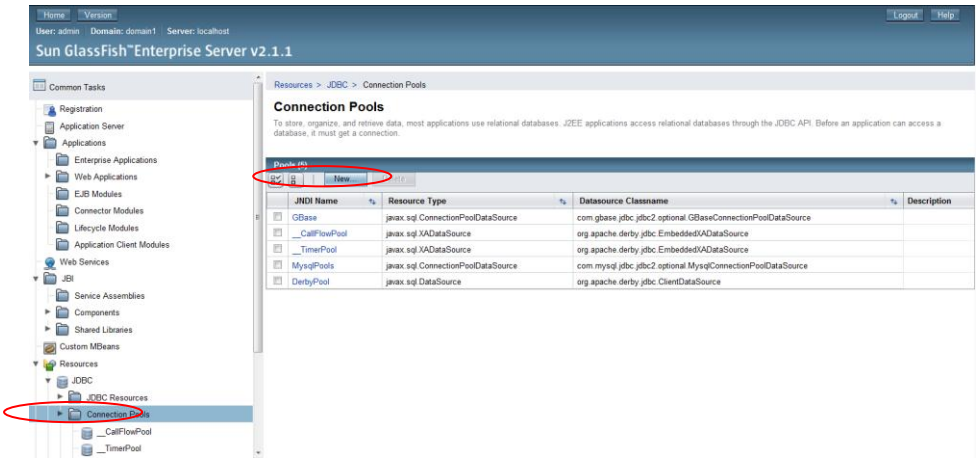
从 glassfish 的安装目录中找到 bin/asadmin.bat，如下图所示：



然后地址栏输入：http://localhost:4848 进入管理界面，4848 端口是在安装时设置的，为默认的端口号，其中的用户名和密码分别为默认的 admin 和 adminadmin，如下图所示：

分两个步骤：首先建立 Connection Pools，然后再建立 JDBC Resources。详见如下步骤：

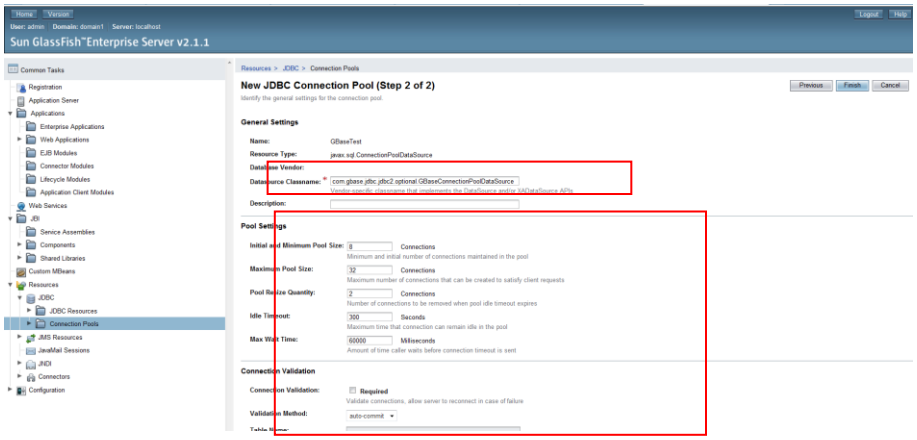


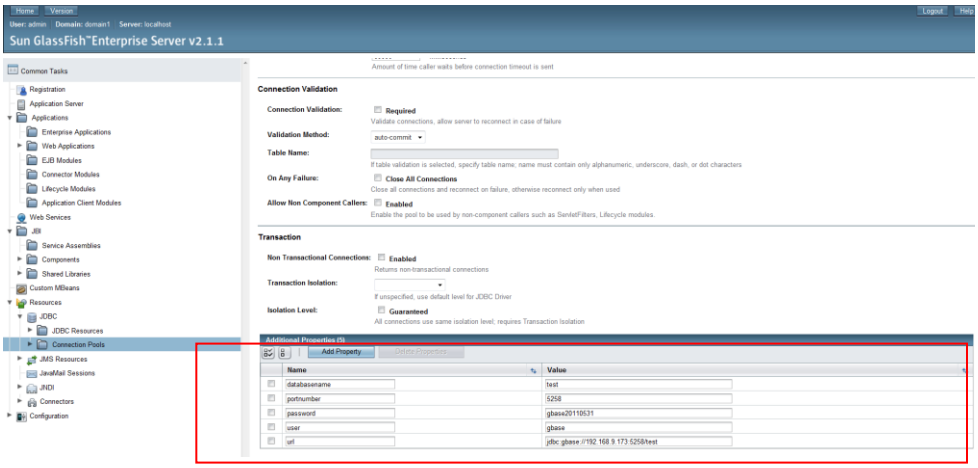


下一步，由于 Database Vendor(数据库供应商)没有 GBase，因此，选择空，如下所示：

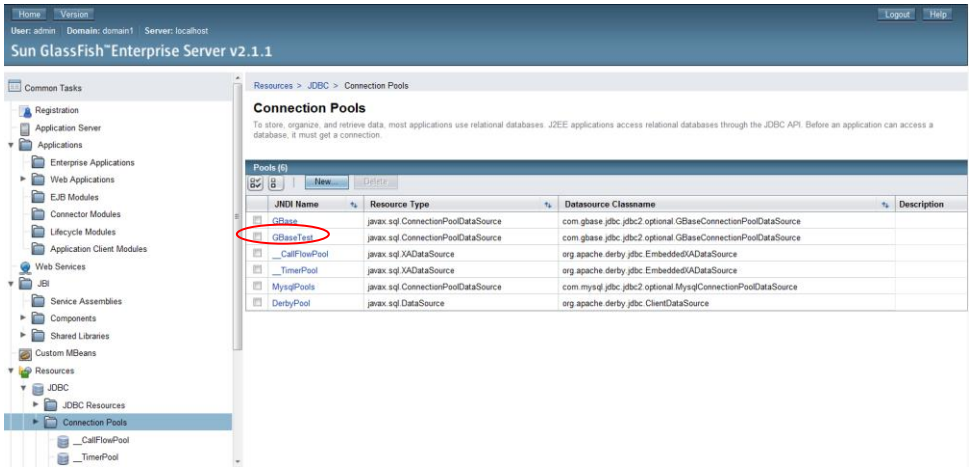
注意：在下一步的 Datasource Classname 输入：

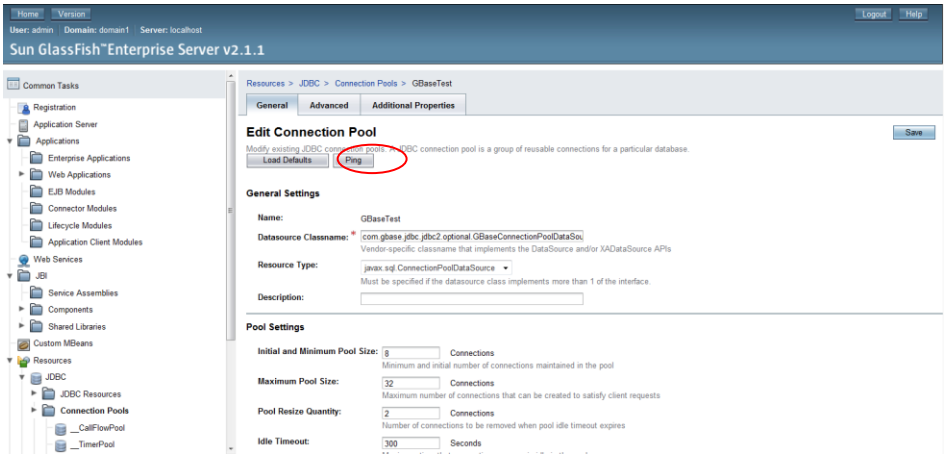
com.gbase.jdbc.jdbc2.optional.GBaseConnectionPoolDataSource



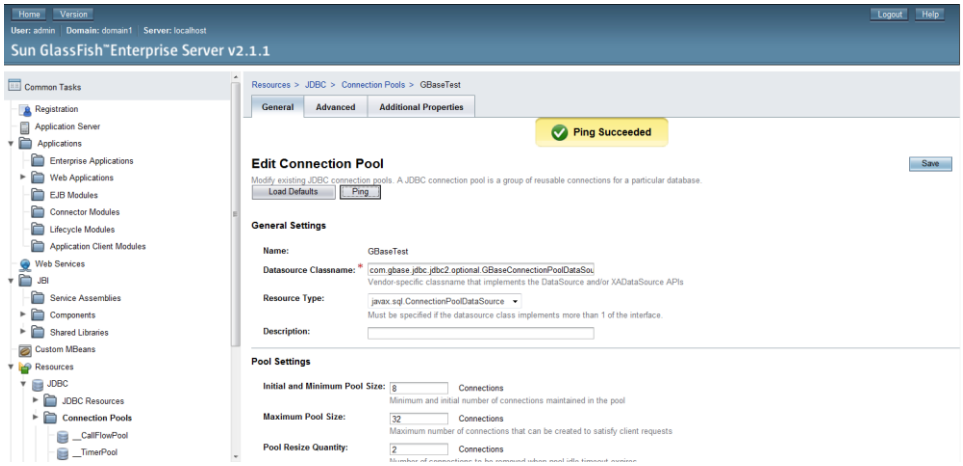


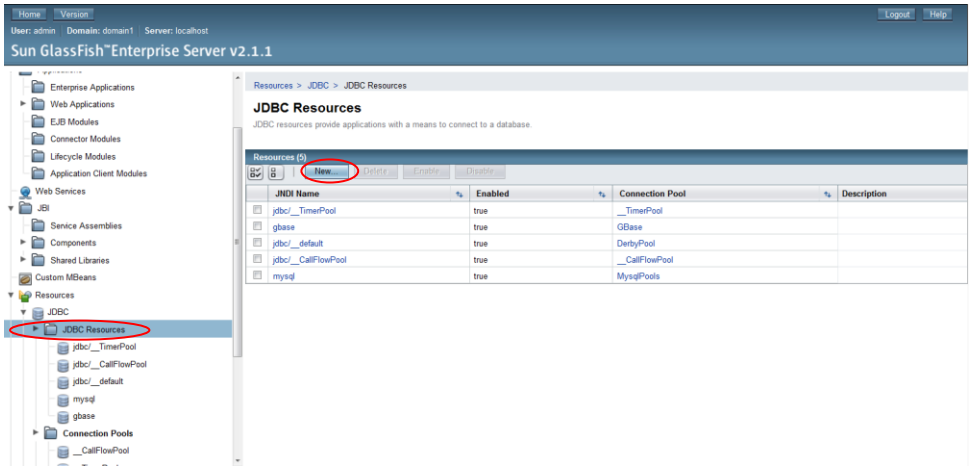
点击本页的“finish”按钮进入如下页面：

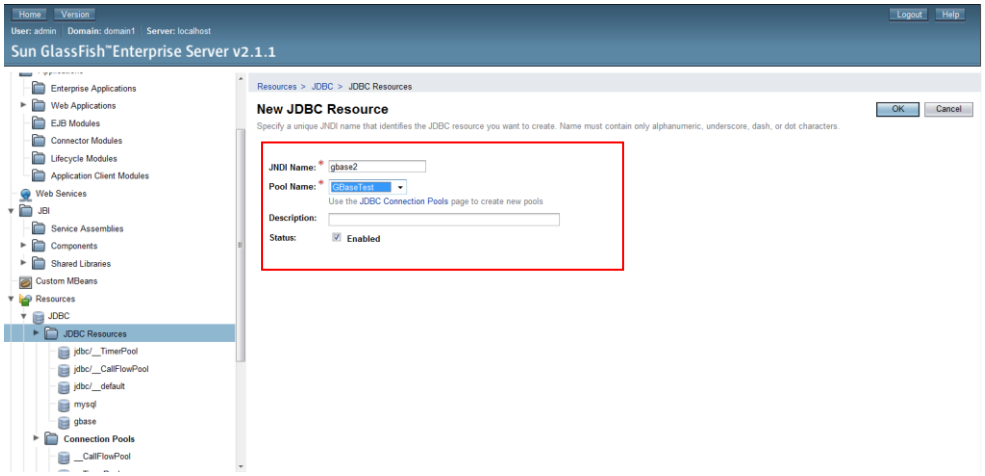




点击上图中的“Ping”按钮，测试是否设置正确，如下所示：







5.6.2 程序验证

编写如下 servlet，然后把项目部署到 GlassFish 上。

```
package com.gbase.glassfish;

import java.io.IOException;
```

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class TestServlet extends HttpServlet {

    /**
    *
    */
    private static final long serialVersionUID = 1L;

    /**
    * The doGet method of the servlet. <br>
    *
    * This method is called when a form has its tag value method equals
to get.
    *
    * @param request the request send by the client to the server
    * @param response the response send by the server to the client
    * @throws ServletException if an error occurred
    * @throws IOException if an error occurred
    */
    public void doGet(HttpServletRequest request,
HttpServletResponse response)
                throws ServletException, IOException {
```

```
            Connection conn = null;
            ResultSet rs = null;
            Statement st = null;
            try {
                Context ctx = new InitialContext();
                DataSource ds = (DataSource) ctx.lookup("gbase2");
                conn = ds.getConnection();
                st = conn.createStatement();
                rs = st.executeQuery("select id,name,age,department
from emp");
                while(rs.next()){
                    System.out.println("员工 id:"+rs.getInt(1)+" 员工
姓名: "+rs.getString(2)+" 员工年龄: "+rs.getInt(3)+" 员工部门:
"+rs.getString(4));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }finally{
                if(rs != null){
                    try {
                        rs.close();
                        rs = null;
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    if(st != null){
                        try {
                            st.close();
                            st = null;
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                        if(conn != null){
                            try {
                                conn.close();
                                conn = null;
                            } catch (SQLException e) {
```

```
                                e.printStackTrace();
                            }
                        }
                    }
                }
            }
        }

        /**
        * The doPost method of the servlet. <br>
        *
        * This method is called when a form has its tag value method equals
to post.
        *
        * @param request the request send by the client to the server
        * @param response the response send by the server to the client
        * @throws ServletException if an error occurred
        * @throws IOException if an error occurred
        */
        public void doPost(HttpServletRequest request,
HttpServletResponse response)
                    throws ServletException, IOException {

            this.doGet(request, response);
        }
    }
```

web.xml 文件如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <description>This is the description of my J2EE
```

```
component</description>
    <display-name>This is the display name of my J2EE
component</display-name>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>com.gbase.glassfish.TestServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <url-pattern>/TestServlet</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
    </web-app>
```

部署成功后，启动 glassfish 服务器，访问如下 url：

http://localhost:8088/TestGBase/TestServlet

该测试用例会打印出数据库为 test 表为 emp 中的内容。

6 第三方持久层使用

6.10penjpa 结合 jdbc 的使用

6.1.1openjpa 介绍

Openjpa 是 apache 组织的开源项目，实现了 ejb3.0 中的 jpa 标准，为开发者提供功能强大，使用简单的持久化管理框架。Openjpa 封装了和关系型数据库交互的操作。Openjpa 可以单独作为持久层框架发挥作用，也可以轻松与其他 j2ee 容器或者符合 ejb3.0 标准的容器集成。

6.1.2openjpa 使用

1. 根据 openjpa 版本，获取正确的 Gbase 字典包。
2. 引入 openjpa 包， gbase 数据库字典包和 jdbc 驱动。
3. 设置 persistence.xml 文件，参考黑色背景设置。

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
version="1.0">

    <persistence-unit name="JPAJC1"
transaction-type="RESOURCE_LOCAL">
        <provider>
```

```
    org.apache.openjpa.persistence.PersistenceProviderImpl
            </provider>
            <class>com.demo1.Bjcy</class>
            <class>com.demo1.BaseEntity</class>
             <properties>
                <property name="openjpa.jdbc.DBDictionary" value="
org.apache.openjpa.jdbc.sql.GBaseDictionary "/>
                <property name="openjpa.ConnectionDriverName"
value="com.gbase.jdbc.Driver" />
                <property name="openjpa.ConnectionURL"
value="jdbc:gbase://192.168.7.235:5258/openjpatest" />
                <property name="openjpa.ConnectionUserName"
value="gbase" />
                <property name="openjpa.ConnectionPassword"
value="gbase20110531" />
                <property name="openjpa.Log" value="SQL=TRACE"/>
            </properties>
        </persistence-unit>
    </persistence>
```

4. 程序验证

```
    package com.test;

    import javax.persistence.EntityManager;
    import javax.persistence.EntityManagerFactory;
```

```
import javax.persistence.Persistence;

import com.demo1.Bjcy;

public class TestJpa {
    private static  EntityManagerFactory emf;
    private static  ThreadLocal<EntityManager> threadLocal;

    public static void main(String[] args) {
        emf = Persistence.createEntityManagerFactory("JPAJC1");
        threadLocal = new ThreadLocal<EntityManager>();
        Bjcy entity = new Bjcy();
        getEntityManager().persist(entity);
    }
    public static EntityManager getEntityManager() {
        EntityManager manager = threadLocal.get();
        if (manager == null || !manager.isOpen()) {
            manager = emf.createEntityManager();
            threadLocal.set(manager);
        }
        return manager;
    }
}
```

6.2Hibernate 结合 jdbc 的使用

6.2.1hibernate 介绍

hibernate 是一个对象关系映射框架,它对 jdbc 进行了非常轻量级的封装,使得开发人员能够以面向对象的方式操作数据库。它屏蔽了不同数据库间的差异，使得开发人员以统一的方式操作数据库。

6.2.2hiberante 使用

1. 根据 hibernate 版本获取对应的 dialect 包。

2. 引入 hibernate，dialect 包和 jdbc 驱动包

3. 配置 hibernate.cfg.xml，设置 gbase 使用的方言类,参考黑色背景设置。

```
<session-factory>
        <property
name="dialect">org.hibernate.dialect.GBaseDialect
</property>
<property
name="connection.driver_class">com.gbase.jdbc.Driver</property>
<property
name="connection.url">jdbc:gbase://192.168.5.66:5258/bhtjdbctest?pro
fileSql=true
</property>
<property name="connection.username">root</property>
<property name="connection.password">1</property>
</session-factory>
```

4. 程序验证

```
package gbase.hibernate.clienttest;

import java.util.List;
import junit.framework.TestCase;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
```

```
    import org.hibernate.Transaction;

    public class clientTest extends TestCase{
        private SessionFactory sf;
        Session session = null;
        PoliceAudit adt = new PoliceAudit();
        Transaction tx = null;

        @Override
        public void setUp(){
            sf = new
Configuration().configure().buildSessionFactory();
        }
        @Override
        public void tearDown(){
            if(sf != null){
                sf.close();
            }
        }
        public void testInsertOfCurd() {
            try{
                for (int i=1;i<=5;i++){
                    session = sf.openSession();
                    Transaction tx = session.beginTransaction();
                    adt.setCreateDate(java.sql.Date.valueOf("2011-05
-01"));
                    adt.setAuditType(i+1);
                    adt.setContentId("test client project");
                    adt.setCount(i);
                    adt.setStatus(i);
                    session.save(adt);
                    session.flush();
                    tx.commit();
                }
                session.close();
            }catch(Exception ex){
                ex.printStackTrace();
```

```
            }
        }
        public void testSelectOfCurd(){
            try{
                session = sf.openSession();
                String hql = "from t_police_auditing in class
gbase.hibernate.clienttest.PoliceAudit";

                Query q = session.createQuery(hql);
                List<PoliceAudit>  ls = q.list();
                for (int j=0;j<ls.size();j++){
                    tx = session.beginTransaction();
                    adt = (PoliceAudit)ls.get(j);
                }
                session.close();
            }catch(Exception ex){
                ex.printStackTrace();
            }
        }
    }
```

7 GBase JDBC 使用示例

本章节共包括 15 部分，分别覆盖了以下功能点的样例：

1） 使用 JDBC 创建连接；

2） 自动装载 JDBC 驱动；（JDBC4.0 新特性需要 jre1.6 及以上版本支持）

3） 通过 JDBC 执行查询 SQL 语句，并从 ResultSet 中读取结果；

4） 通过 JDBC 执行 DDL 和 DML 语句；

5） 通过 JDBC 调用存储过程；

6） NATIONAL Character 相关操作； （JDBC4.0 新特性需要 jre1.6 及以上版本支持）；

7） 大对象类型使用；

8） 使用 Statement.getGeneratedKeys()获取 AUTO_INCREMENT 列的值；

9） 使用 SELECT LAST_INSERT_ID()获取 AUTO_INCREMENT 列的值；

10）使用可更新结果集获取 AUTO_INCREMENT 列的值；

11）GBase JDBC 在 Jboss 应用中使用示例；

12）GBase JDBC 在 Tomcat 应用中使用示例.

13）GBase JDBC 集群高可用性（IP 自动路由，需要 GBaseJDBC8.3.81.53 及以上版本）

14）GBase JDBC 集群高可用负载均衡（GBaseJDBC8.3.81.5x 及以上版本）

15)GBase JDBC 流式读取使用示例

7.1使用 JDBC 创建连接

本示例实现了通过 JDBC 建立数据库链接的功能。

示例如下:

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionSimple {

    public static void main(String[] args) {
            ConnectionSimple connectionSimple = new
ConnectionSimple();
    connectionSimple.userDriverManagerGetConnection();
    }
    /**
    * 使用 DriverManager 获取连接.
    */
    public void userDriverManagerGetConnection() {
        Connection conn = null;
        try {
            Class.forName("com.gbase.jdbc.Driver");
            conn =
DriverManager.getConnection("jdbc:gbase://192.168.5.210:5258/test?us
er=root&password=");
        } catch (SQLException ex) {
            // 处理错误
            System.out.println("SQLException: " +
ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " +
ex.getErrorCode());
```

```
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } finally {
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
    }
```

DriverManager 有很多重构函数，获取连接也有很多方法，想起参照 jdk 的 doc 中 java.sql. DriverManager 的详细说明即可。

7.2自动装载 JDBC 驱动

本示例实现了使用 gbase-connector-java 的 JDBC4.0 新特性自动装载驱动功能。本样例需要 gbase-connector-java8.3.81.51 及以上版本，和 jre1.6 及以上版本。

示例如下：

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class AutomaticLoadingOfJavaSqlDriver {

    public static void main(String[] args) {
        AutomaticLoadingOfJavaSqlDriver
automaticLoadingOfJavaSqlDriver = new
AutomaticLoadingOfJavaSqlDriver();
```

```
            automaticLoadingOfJavaSqlDriver.userDriverManagerGetConn
ection();
        }
        /**
        * 使用自动装载驱动 DriverManager 获取连接.
        */
        public void userDriverManagerGetConnection() {
            Connection conn = null;
            try {
                conn =
DriverManager.getConnection("jdbc:gbase://192.168.111.915:5258/test?
user=sysdba&password=");

                System.out.println("AutomaticLoadingOfJavaSqlDriver
ok");
            } catch (SQLException ex) {
                // 处理错误
                System.out.println("SQLException: " +
ex.getMessage());
                System.out.println("SQLState: " + ex.getSQLState());
                System.out.println("VendorError: " +
ex.getErrorCode());
            } finally {
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
    }
```

7.3通过 JDBC 执行查询 SQL 语句

该章节分为 2 部分：

- 使用 Statement 执行 SQL 语句
- 使用 PreparedStatement 执行 SQL 语句

  ✧ executeSQLByStatement 方法使用 Statemen 执行 SQL 语句，并把 ResultSet 中的查询结果通过控制台打印出来。

  ✧ executeSQLByPreparedStatement 方法使用 PreparedStatement 执行 SQL 语句，并把 ResultSet 中的查询结果通过控制台打印出来。

示例如下：

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

public class ExecuteSQLByStatement {

    private static final String URL =
"jdbc:gbase://192.168.111.95:5258/test?user=sysdba&password=";

    public static void main(String[] args) {
        prepareTableAndData();
        ExecuteSQLByStatement executeSQLByStatement = new
ExecuteSQLByStatement();
        executeSQLByStatement.executeSQLByStatement();
```

```
            executeSQLByStatement.executeSQLByPreparedStatement();
            }

        /**
        * 通过 Statement 执行 SQL 语句,
        * 并把 ResultSet 中的结果通过控制台
        * 打印出来。
        */
        public void executeSQLByStatement() {
            Connection conn = null;
            Statement stm = null;
            ResultSet rs = null;
            ResultSetMetaData rsmd = null;
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn = DriverManager.getConnection(URL);
                stm = conn.createStatement();
                rs = stm.executeQuery("select * from user_info where
user_id in (2,3)");

                if (rs == null) {
                    return;
                }
                rsmd = rs.getMetaData();
                int rsColoumnCount = rsmd.getColumnCount();
                while (rs.next()) {
                    System.out.println("executeSQLByStatement
==========================");
                    for (int i = 0; i < rsColoumnCount; i++) {
                        System.out.print(rsmd.getColumnName(i +
1).concat(" = "));
                        System.out.println(rs.getObject(i + 1));
                    }
                    System.out.println("executeSQLByStatement
==========================");
                }
            } catch (ClassNotFoundException e) {
```

```
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            try {
                rs.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                rs = null;
            }
            try {
                stm.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                stm = null;
            }
            try {
                conn.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                conn = null;
            }
        }
    }

    /**
    * 通过 PreparedStatement 执行 SQL 语句,
    * 并把 ResultSet 中的结果通过控制台
    * 打印出来。
    */
    public void executeSQLByPreparedStatement() {
        Connection conn = null;
        PreparedStatement stm = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;
```

```
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn = DriverManager.getConnection(URL);
                stm = conn.prepareStatement("select * from user_info
where user_id in (?,?)");
                stm.setInt(1, 2);
                stm.setInt(2, 3);

                rs = stm.executeQuery();
                if (rs == null) {
                    return;
                }
                rsmd = rs.getMetaData();
                int rsColoumnCount = rsmd.getColumnCount();
                while (rs.next()) {
                    System.out.println("executeSQLByPreparedStatemen
t ============================");
                    for (int i = 0; i < rsColoumnCount; i++) {

                        System.out.print(rsmd.getColumnName(i +
1).concat(" = "));
                        System.out.println(rs.getObject(i + 1));
                    }
                    System.out.println("executeSQLByPreparedStatemen
t ============================");
                }
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    rs.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
```

```
            rs = null;
        }
        try {
            stm.close();
        } catch (NullPointerException e) {
        } catch (Exception e) {
            stm = null;
        }
        try {
            conn.close();
        } catch (NullPointerException e) {
        } catch (Exception e) {
            conn = null;
        }
    }
    }

/**
* 在 test 数据库中创建一个名称为
* “user_info”的表，包含三个字段，
* 并向表中插入三条数据。
*/
private static void prepareTableAndData() {
    Connection conn = null;
    Statement stm = null;
    try {
        Class.forName("com.gbase.jdbc.Driver");
        conn = DriverManager.getConnection(URL);
        stm = conn.createStatement();

        /*
        * create table user_info (
        user_id int(11) ,
        user_Name varchar(50),
        user_info varchar(200)
        )ENGINE=GsDB DEFAULT CHARSET=utf8
        */
```

```
                stm.executeUpdate("drop table if exists `user_info`");
                stm.executeUpdate("create table `user_info`
( `user_id` int(11) ,`user_Name` varchar(50),`user_info`
varchar(200))ENGINE=GsDB DEFAULT CHARSET=utf8");
                stm.executeUpdate("insert into `user_info`
(`user_id`,`user_name`, `user_info`) values (1,'张三','南大通用')");
                stm.executeUpdate("insert into `user_info`
(`user_id`,`user_name`, `user_info`) values (2,'张四','南大通用
-gbase8a')");
                stm.executeUpdate("insert into `user_info`
(`user_id`,`user_name`, `user_info`) values (3,'张五','南大通用
-gbase8d')");
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    stm.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    stm = null;
                }
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
    }
```

7.4通过 JDBC 执行 DDL 和 DML 语句

用例 executeDDLAndDMLSQLByStatement 方法实现以下功能:

- 使用 Statement 执行 DDL 语句创建一个表;
- 使用 Statement 执行 DML 语句向表中插入一条数据;
- 使用 Statement 执行 DML 语句修改 2 中插入的数据。

用例 executeDDLAndDMLSQLByPreparedStatement 方法实现以下功能:

- 使用 PreparedStatement 执行 DDL 语句创建一个表;
- 使用 PreparedStatement 执行 DML 语句向表中插入一条数据;
- 使用 PreparedStatement 执行 DML 语句修改 2 中插入的数据。

示例如下:

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;

public class ExecuteUpdateSQLByStatement {

    private static final String URL =
"jdbc:gbase://192.168.111.95:5258/test?user=sysdba&password=";

    /**
    * @param args
    */
    public static void main(String[] args) {
        ExecuteUpdateSQLByStatement executeUpdateSQLByStatement =
```

```
new ExecuteUpdateSQLByStatement();
            executeUpdateSQLByStatement.executeDDLAndDMLSQLByStateme
nt();
            executeUpdateSQLByStatement.executeDDLAndDMLSQLByPrepare
dStatement();
        }
        /**
        * 在 test 数据库中创建一个名称为
        * “user_info”的表，包含三个字段，
        * 并向表中插入三条数据。
        */
        public void executeDDLAndDMLSQLByStatement () {

            Connection conn = null;
            Statement stm = null;
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn = DriverManager.getConnection(URL);
                stm = conn.createStatement();

                /*
                * create table user_info (
                    user_id int(11) ,
                    user_Name varchar(50),
                    user_info varchar(200)
                    )ENGINE=GsDB DEFAULT CHARSET=utf8
                 */
                stm.executeUpdate("drop table if exists `user_info`");
                stm.executeUpdate("create table `user_info`
( `user_id` int(11) ,`user_Name` varchar(50),`user_info`
varchar(200))ENGINE=GsDB DEFAULT CHARSET=utf8");
                stm.executeUpdate("insert into `user_info`
(`user_id`,`user_name`, `user_info`) values (3,'张五','南大通用
-gbase8d')");
                stm.executeUpdate("update `user_info` set `user_name`
= '张五修改' where user_id='3' ");
                System.out.println("executeDDLAndDMLSQLByStatement
```

```
ok");
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    stm.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    stm = null;
                }
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
        /**
        * 在 test 数据库中创建一个名称为
        * “user_info”的表，包含三个字段，
        * 并向表中插入三条数据。
        */
        public void executeDDLAndDMLSQLByPreparedStatement () {

            Connection conn = null;
            PreparedStatement stm = null;
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn = DriverManager.getConnection(URL);
                stm = conn.prepareStatement("drop table if exists
`user_info`");
```

```
                /*
                * create table user_info (
                    user_id int(11) ,
                    user_Name varchar(50),
                    user_info varchar(200)
                    )ENGINE=GsDB DEFAULT CHARSET=utf8
                */
                stm.addBatch("create table `user_info-2` ( `user_id` int(11) ,`user_Name` varchar(50),`user_info` varchar(200))ENGINE=GsDB DEFAULT CHARSET=utf8");
                stm.executeBatch();
                stm = conn.prepareStatement("insert into `user_info-2` (`user_id`,`user_name`, `user_info`) values (?,?,?)");
                stm.setInt(1, 3);
                stm.setString(2, "张五");
                stm.setString(3, "南大通用-gbase8d");
                stm.executeUpdate();
                stm = conn.prepareStatement("update `user_info-2` set `user_name` = ? where user_id=? ");
                stm.setString(1, "张五修改 Prepared");
                stm.setInt(2, 3);
                stm.executeUpdate();

                System.out.println("executeDDLAndDMLSQLByPreparedStatement ok");
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    stm.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    stm = null;
```

```
            }
            try {
                conn.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                stm = null;
            }
        }
    }
}
```

7.5通过 JDBC 调用存储过程

该用例分为 4 部分，分别实现了如下功能：

- callProcedureWhitNoParamByCallableStatement 调用没有参数的存储过程；
- callProcedureWhitINParamByCallableStatement 调用只有 IN 参数的存储过程；
- callProcedureWhitOUTParamByCallableStatement 调用只有 OUT 参数的存储过程；
- callProcedureWhitInOutParamByCallableStatement 调用有 IN、OUT 参数的存储过程。

示例如下：

```
package com.gbase.jdbc.simple;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
    import java.sql.Statement;
    import java.sql.Types;

    public class CallProcByJdbc {

        private static final String URL =
"jdbc:gbase://192.168.111.95:5258/test?user=sysdba&password=";
        /**
        * @param args
        */
        public static void main(String[] args) {

            //创建存储过程
            prepareProc();

            CallProcByJdbc callProcByJdbc = new CallProcByJdbc();

            //调用没有参数的存储过程
            callProcByJdbc.callProcedureWhitNoParamByCallableStateme
nt();

            //调用只有 IN 参数的存储过程
            callProcByJdbc.callProcedureWhitINParamByCallableStateme
nt();

            //调用只有 OUT 参数的存储过程
            callProcByJdbc.callProcedureWhitOUTParamByCallableStatem
ent();

            //调用有 IN/OUT 参数的存储过程
            callProcByJdbc.callProcedureWhitInOutParamByCallableStat
ement();
        }

        /**
        * 通过 CallableStatement 调用没有参数的
```

```
* 存储过程。
*/
public void callProcedureWhitNoParamByCallableStatement() {
    Connection conn = null;
    CallableStatement cstm = null;
    ResultSet rs = null;
    try {
        Class.forName("com.gbase.jdbc.Driver");
        conn = DriverManager.getConnection(URL);
        cstm = conn.prepareCall("call procNoParam()");
        rs = cstm.executeQuery();
        rs.next();
        System.out.println(rs.getString(1));
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            rs.close();
        }catch (NullPointerException e) {
        } catch (Exception e) {
            cstm = null;
        }
        try {
            cstm.close();
        } catch (NullPointerException e) {
        } catch (Exception e) {
            cstm = null;
        }
        try {
            conn.close();
        } catch (NullPointerException e) {
        } catch (Exception e) {
            conn = null;
        }
    }
```

```
    }

    /**
    * 通过 CallableStatement 调用 IN 参数的
    * 存储过程。
    */
    public void callProcedureWhitINParamByCallableStatement() {
        Connection conn = null;
        CallableStatement cstm = null;
        ResultSet rs = null;
        try {
            Class.forName("com.gbase.jdbc.Driver");
            conn = DriverManager.getConnection(URL);
            cstm = conn.prepareCall("{call procInParam(?)}");
            cstm.setString(1, "InParam Call Works!");
            rs = cstm.executeQuery();
            rs.next();
            System.out.println(rs.getString(1));
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                rs.close();
            }catch (NullPointerException e) {
            } catch (Exception e) {
                cstm = null;
            }
            try {
                cstm.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                cstm = null;
            }
            try {
                conn.close();
```

```
            } catch (NullPointerException e) {
            } catch (Exception e) {
                conn = null;
            }
        }
    }

    /**
    * 通过 CallableStatement 调用 OUT 参数的
    * 存储过程。
    */
    public void callProcedureWhitOUTParamByCallableStatement() {
        Connection conn = null;
        CallableStatement cstm = null;
        try {
            Class.forName("com.gbase.jdbc.Driver");
            conn = DriverManager.getConnection(URL);
            cstm = conn.prepareCall("call procOutParam(?)");
            cstm.setString(1, "@outParam");
            cstm.registerOutParameter(1, Types.VARCHAR);
            cstm.execute();
            System.out.println(cstm.getString(1));
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                cstm.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                cstm = null;
            }
            try {
                conn.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
```

```
                conn = null;
            }
        }
    }

    /**
    * 通过 CallableStatement 调用 IN/OUT 参数的
    * 存储过程。
    */
    public void callProcedureWhitInOutParamByCallableStatement() {
        Connection conn = null;
        CallableStatement cstm = null;
        try {
            Class.forName("com.gbase.jdbc.Driver");
            conn = DriverManager.getConnection(URL);
            cstm = conn.prepareCall("{call procInOutParam(?,?)}");
            cstm.setString(1, "aaaaa");
            cstm.setString(2, "@outParam");
            cstm.registerOutParameter(2, Types.VARCHAR);
            cstm.execute();
            System.out.println(cstm.getString(2));
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                cstm.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                cstm = null;
            }
            try {
                conn.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                conn = null;
```

```
                }
            }
        }

        /**
        * 创建 4 个存储过程;
        * 1、没有参数
        * 2、只有 IN 参数
        * 3、只有 OUT 参数
        * 4、有 IN、OUT 参数
        */
        private static void prepareProc() {
            Connection conn = null;
            Statement stm = null;
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn = DriverManager.getConnection(URL);
                stm = conn.createStatement();

                stm.executeUpdate("DROP PROCEDURE IF EXISTS
`test`.`procNoParam`");
                stm.executeUpdate("DROP PROCEDURE IF EXISTS
`test`.`procInParam`");
                stm.executeUpdate("DROP PROCEDURE IF EXISTS
`test`.`procOutParam`");
                stm.executeUpdate("DROP PROCEDURE IF EXISTS
`test`.`procInOutParam`");
                stm.executeUpdate("CREATE PROCEDURE
`test`.`procNoParam` () begin select 'procNoParamTest works'; end");
                stm.executeUpdate("CREATE PROCEDURE
`test`.`procInParam` (IN inParam Varchar(100)) begin select inParam;
end");
                stm.executeUpdate("CREATE PROCEDURE
`test`.`procOutParam` (OUT outParam Varchar(100)) begin SET outParam =
'outParamTest works'; end");
                stm.executeUpdate("CREATE PROCEDURE
`test`.`procInOutParam` (IN inParam Varchar(100), OUT outParam
```

```
Varchar(200)) begin set outParam = CONCAT(\'InOutParam \',inParam,\ '
works!\ '); end");
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    stm.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    stm = null;
                }
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
    }
```

7.6NATIONAL CHARACTER 相关操作

本示例实现了如下功能:

- NationalCharacterSimple.simpleGetNClob 获取 Nclob 类型内容。
- NationalCharacterSimple.simpleSetNClob()通过

PreparedStatement.SetNClob 实现 NClob 的存入数据库。

- nationalCharacterSimple.simpleUpdateNClob()通过

PreparedStatement.UpdateNClob 实现 Nclob 字段的更新。

更多关于 NATIONAL CHARACTER 操作示例请参照工程。

示例如下:

```
package com.gbase.jdbc.simple;

import java.io.Reader;
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.NClob;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class NationalCharacterSimple {

    private static final String URL =
"jdbc:gbase://192.168.111.95:5258/test?user=sysdba&password=";

    /**
    * Runs all test cases in this test suite
    *
    * @param args
    */
    public static void main(String[] args) {
        NationalCharacterSimple nationalCharacterSimple = new
NationalCharacterSimple();
        try {
            nationalCharacterSimple.simpleGetNClob();
            nationalCharacterSimple.simpleSetNClob();
            nationalCharacterSimple.simpleUpdateNClob();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
            }
        }

        /**
        * Simple for ResultSet.getNClob()
        *
        * @throws Exception
        */
        public void simpleGetNClob() throws Exception {
            Connection conn = null;
            Statement stm = null;
            ResultSet rs = null;
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn = DriverManager.getConnection(URL);
                stm = conn.createStatement();
                createTable("simpleGetNClob", "(c1 NATIONAL
CHARACTER(10), c2 NATIONAL CHARACTER(10))", stm);
                stm.executeUpdate("INSERT INTO simpleGetNClob (c1, c2)
VALUES (_utf8 'aaa', _utf8 'bbb')");
                rs = stm.executeQuery("SELECT c1, c2 FROM
simpleGetNClob");
                rs.next();
                char[] c1 = new char[3];

                rs.getNClob(1).getCharacterStream().read(c1);
                System.out.println(new String(c1));
                char[] c2 = new char[3];

                rs.getNClob("c2").getCharacterStream().read(c2);
                System.out.println(new String(c2));

            }catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            } finally {
                try {
                    rs.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    rs = null;
                }
                try {
                    stm.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    stm = null;
                }
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
        /**
        * Simple for PreparedStatement.setNClob()
        *
        * @throws Exception
        */
        public void simpleSetNClob() throws Exception {

            Connection conn = null;
            Statement stm = null;
            PreparedStatement pstm = null;
            ResultSet rs = null;
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn =
DriverManager.getConnection(URL+"&useServerPrepStmts=false&useUnicod
```

```
e=true&characterEncoding=utf-8");
                stm = conn.createStatement();

                createTable("simpleSetNClob", "(c1 NATIONAL
CHARACTER(10), c2 NATIONAL CHARACTER(10), " +
                "c3 NATIONAL CHARACTER(10))", stm);
                pstm = conn.prepareStatement("INSERT INTO
simpleSetNClob (c1, c2, c3) VALUES (?, ?, ?)");
                pstm.setNClob(1, (NClob)null);
                NClob nclob2 = conn.createNClob();
                nclob2.setString(1, "aaa");
                pstm.setNClob(2, nclob2);              // for
setNClob(int, NClob)

                Reader reader3 = new StringReader("\'aaa\'");
                reader3 = new StringReader("\'aaa\'");
                pstm.setNClob(3, reader3, 5);          // for
setNClob(int, Reader, long)
                pstm.execute();
                rs = stm.executeQuery("SELECT c1, c2, c3 FROM
simpleSetNClob");
                rs.next();

                //null
                System.out.println(rs.getString(1));

                //"aaa"
                System.out.println(rs.getString(2));

                //"\'aaa\'"
                System.out.println(rs.getString(3));
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
```

```
        }finally {
            try {
                rs.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                rs = null;
            }
            try {
                stm.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                stm = null;
            }
            try {
                pstm.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                pstm = null;
            }
            try {
                conn.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                conn = null;
            }
        }
    }
    /**
    * Simple for ResultSet.updateNClob()
    *
    * @throws Exception
    */
    public void simpleUpdateNClob() throws Exception {

        Connection conn = null;
        Statement stm = null;
        PreparedStatement pstm = null;
```

```
            ResultSet rs = null;
            ResultSet rs2 = null;
            try {
                Class.forName("com.gbase.jdbc.Driver");
                conn =
DriverManager.getConnection(URL+"&useUnicode=true&characterEncoding=
utf-8");
                stm =
conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_UPDATABLE);

                createTable("simpleUpdateNChlob",
                                    "(c1 CHAR(10) PRIMARY KEY, c2
NATIONAL CHARACTER(10)) default character set utf8", stm);
                pstm = conn.prepareStatement("INSERT INTO
simpleUpdateNChlob (c1, c2) VALUES (?, ?)");
                pstm.setString(1, "1");
                NClob nClob1 = conn.createNClob();
                nClob1.setString(1, "aaa");
                pstm.setNClob(2, nClob1);
                pstm.execute();
                rs = stm.executeQuery("SELECT c1, c2 FROM
simpleUpdateNChlob");
                rs.next();
                NClob nClob2 = conn.createNClob();
                nClob2.setString(1, "bbb");
                rs.updateNClob("c2", nClob2);
                rs.updateRow();(移动到这行然后修改改行数据)
                rs.moveToInsertRow();
                rs.updateString("c1", "2");
                NClob nClob3 = conn.createNClob();
                nClob3.setString(1, "ccc");
                rs.updateNClob("c2", nClob3);
                rs.insertRow(); (移动到插入点然后插入新数据)
                rs2 = stm.executeQuery("SELECT c1, c2 FROM
simpleUpdateNChlob");
                rs2.next();
```

```
            //"1"
            System.out.println(rs2.getString("c1"));

            //"bbb"

            System.out.println(rs2.getNString("c2"));
            rs2.next();

            //"2"
            System.out.println(rs2.getString("c1"));

            //"ccc"

            System.out.println(rs2.getNString("c2"));
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                rs.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                rs = null;
            }
            try {
                rs2.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                rs2 = null;
            }
            try {
                stm.close();
            } catch (NullPointerException e) {
```

```
                } catch (Exception e) {
                    stm = null;
                }
                try {
                    pstm.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    pstm = null;
                }
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
        private void createTable(String objectName,
                String columnsAndOtherStuff, Statement stmt) throws
    SQLException {
            createSchemaObject("TABLE", objectName,
columnsAndOtherStuff, stmt);
        }
        private void createSchemaObject(String objectType, String
objectName,
                String columnsAndOtherStuff, Statement stmt) throws
    SQLException {
            dropSchemaObject(objectType, objectName, stmt);

            StringBuffer createSql = new
StringBuffer(objectName.length() + objectType.length() +
columnsAndOtherStuff.length() + 10);
            createSql.append("CREATE  ");
            createSql.append(objectType);
            createSql.append(" ");
            createSql.append(objectName);
```

```
            createSql.append(" ");
            createSql.append(columnsAndOtherStuff);

            try {
                stmt.executeUpdate(createSql.toString());
            } catch (SQLException sqlEx) {
                throw sqlEx;
            }
        }
        private void dropSchemaObject(String objectType, String
objectName, Statement stmt)
        throws SQLException {
            stmt.executeUpdate("DROP " + objectType + " IF EXISTS "
            + objectName);
        }
    }
```

7.7大对象类型使用

本样例实现了使用 PreparedStatement 相关方法实现 Blob 和 Clob 类型数据的插入和读取。

示例如下：

```
package com.gbase.jdbc.simple;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.sql.Blob;
import java.sql.Clob;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class BlobAndClob {

    private static final String URL =
"jdbc:gbase://192.168.111.95:5258/test?user=sysdba&password=";

    /**
    * @param args
    */
    public static void main(String[] args) {
        prepareTableAndData();
        BlobAndClob blobAndClob = new BlobAndClob();
        blobAndClob.SimpleBlob();
    }

    public void SimpleBlob() {

        Connection conn = null;
        PreparedStatement stm = null;
        ResultSet rs = null;
        OutputStream out = null;
        FileInputStream bIn = null;
        BufferedReader br = null;
        BufferedReader brForRead = null;
        try {
            bIn = new FileInputStream(
            new File("blobTest.txt"));

            Class.forName("com.gbase.jdbc.Driver");
            conn = DriverManager.getConnection(URL);
```

```
stm = conn.prepareStatement("insert into" +
        " `test`.`blogAndClobTest` VALUES (?, ?, ?)");
stm.setLong(1, System.currentTimeMillis());

//创建一个 Blob 对象
Blob blob = conn.createBlob();

//把内容写入到对象中
out = blob.setBinaryStream(1);
int i = -1;
while ((i = bIn.read()) != -1) {
    out.write(i);
}
out.flush();
out.close();

//把设置好的 Blob 的设置到 PrepareStatement 中
stm.setBlob(2, blob);

//创建一个 Clob 对象
Clob clob = conn.createClob();
clob.setString(1, "南大通用安全数据库-Clob");
stm.setClob(3, clob);
stm.executeUpdate();
blob.free();
clob.free();

rs = stm.executeQuery("select * from
    `test`.`blogAndClobTest`");

rs.next();

//读取 Blob 字段
Blob readBlob = rs.getBlob(2);
br = new BufferedReader(new InputStreamReader
        (readBlob.getBinaryStream()));
```

```
            String line = null;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
            readBlob.free();

            //读取 Clob 字段
            Clob readClob = rs.getClob(3);
            brForRead = new BufferedReader
        (readClob.getCharacterStream());
            line = null;
            while ((line = brForRead.readLine()) != null) {
                System.out.println(line);
            }
            readClob.free();

        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                brForRead.close();
            } catch (IOException e4) {
            }
            try {
                br.close();
            } catch (IOException e3) {
            }
            try {
                bIn.close();
            } catch (IOException e2) {
            }
```

```
            try {
                rs.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                rs = null;
            }
            try {
                stm.clearParameters();
            } catch (SQLException e1) {
            }
            try {
                stm.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                stm = null;
            }
            try {
                conn.close();
            } catch (NullPointerException e) {
            } catch (Exception e) {
                conn = null;
            }
        }
    }
    /**
    * 在 test 数据库中创建一个名称为
    * “blogAndClobTest”的表，包含三个字段，
    */
    private static void prepareTableAndData() {
        Connection conn = null;
        Statement stm = null;
        try {
            Class.forName("com.gbase.jdbc.Driver");
            conn = DriverManager.getConnection(URL);
            stm = conn.createStatement();

            /*
```

```
                * create table user_info (
                    user_id int(11) ,
                    user_Name varchar(50),
                    user_info varchar(200)
                    )ENGINE=GsDB DEFAULT CHARSET=utf8
                */
                stm.executeUpdate("drop table if exists
            `blogAndClobTest`");
                stm.executeUpdate("create table `blogAndClobTest` "
            +"( `data_id` bigint ,`blob_data` blob,`clob_data`
    " +"text)ENGINE=GsDB DEFAULT CHARSET=utf8");
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } finally {
                try {
                    stm.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    stm = null;
                }
                try {
                    conn.close();
                } catch (NullPointerException e) {
                } catch (Exception e) {
                    conn = null;
                }
            }
        }
    }
```

7.8 获取 AUTO_INCREMENT 列值方法 1

本示例实现了使用 Statement.getGeneratedKeys()获取 AUTO_INCREMENT 列的值。

示例如下：

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class SampleGetGeneratedKeys {
    public static void main(String[] args) {
        try {
            (new SampleGetGeneratedKeys()).test();
        } catch (Exception ex) {

        }
    }

    public void test() throws Exception {
        Connection conn = null;
        try {
            Class.forName("com.gbase.jdbc.Driver").newInstance();
            conn = DriverManager

    .getConnection("jdbc:gbase://192.168.5.210:5258/test?user=root&p
assword=");
```

```
            Statement stmt = null;
            ResultSet rs = null;
            try {
                // 创建 Statement 对象
                stmt = conn.createStatement(
                        java.sql.ResultSet.TYPE_FORWARD_ONLY,
                        java.sql.ResultSet.CONCUR_UPDATABLE);

                // 创建表
                stmt.executeUpdate("DROP TABLE IF EXISTS
autoIncTutorial");
                stmt.executeUpdate("CREATE TABLE autoIncTutorial ("
                        + "priKey INT NOT NULL AUTO_INCREMENT, "
                        +  "dataField VARCHAR(64), PRIMARY KEY
(priKey))");

                // 插入一条数据
                stmt.executeUpdate("INSERT INTO autoIncTutorial
(dataField) "
                        + "values ('Can I Get the Auto Increment
Field?')",
                        Statement.RETURN_GENERATED_KEYS);

                // 使用 Statement.getGeneratedKeys()获取自增一字段
                int autoIncKeyFromApi = -1;
                rs = stmt.getGeneratedKeys();
                if (rs.next()) {
                    autoIncKeyFromApi = rs.getInt(1);
                }
                rs.close();
                rs = null;
                System.out.println("Key returned from
```

```
getGeneratedKeys():"
                        + autoIncKeyFromApi);
            } finally {
                if (rs != null) {
                    try {
                        rs.close();
                    } catch (SQLException ex) {

                    }
                }
                if (stmt != null) {
                    try {
                        stmt.close();
                    } catch (SQLException ex) {

                    }
                }
            }
        } catch (SQLException ex) {
            // 处理错误
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
        } finally {
            conn.close();
        }
    }
}
```

7.9 获取 AUTO_INCREMENT 列值方法 2

本示例实现了使用 SELECT LAST_INSERT_ID()获取 AUTO_INCREMENT 列的值

示例如下：

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class SampleLastInsertID {

    private static final String URL =
"jdbc:gbase://192.168.111.95:5258/test?user=sysdba&password=";
    public static void main(String[] args) {
        try {
            (new SampleLastInsertID()).test();
        } catch (Exception ex) {

        }
    }

    public void test() throws Exception {
        Connection conn = null;
        try {
            Class.forName("com.gbase.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(URL);

            Statement stmt = null;
            ResultSet rs = null;
            try {
                // 创建 Statement 对象
                stmt = conn.createStatement();
```

```
                // 创建表
                stmt.executeUpdate("DROP      TABLE      IF      EXISTS
autoIncTutorial");
                stmt.executeUpdate("CREATE TABLE autoIncTutorial ("
                        + "priKey INT NOT NULL AUTO_INCREMENT, "
                        +   "dataField   VARCHAR(64),   PRIMARY   KEY
(priKey))");

                // 插入一条数据
                stmt.executeUpdate("INSERT    INTO    autoIncTutorial
(dataField) "
                        + "values  ('Can  I  Get  the  Auto  Increment
Field?')");

                // 使用 LAST_INSERT_ID()获取自增一字段值
                int autoIncKeyFromFunc = -1;
                rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");
                if (rs.next()) {
                    autoIncKeyFromFunc = rs.getInt(1);
                } else {

                }
                rs.close();
                System.out.println("Key returned from "
                        +    "'SELECT    LAST_INSERT_ID()':    "    +
autoIncKeyFromFunc);
            } finally {
                if (rs != null) {
                    try {
                        rs.close();
                    } catch (SQLException ex) {
```

```
                }
            }
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException ex) {

                }
            }
        }
    } catch (SQLException ex) {
        // 处理错误
        System.out.println("SQLException: " + ex.getMessage());
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("VendorError: " + ex.getErrorCode());
    } finally {
        conn.close();
    }
  }
}
```

7.10 获取 AUTO_INCREMENT 列值方法 3

本示例实现了使用可更新结果集获取 AUTO_INCREMENT 列的值。

示例如下：

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;

public class SampleUpdatableResultSet {

    private static final String URL =
"jdbc:gbase://192.168.111.95:5258/test?user=sysdba&password=";

    public static void main(String[] args) {
        try {
            (new SampleUpdatableResultSet()).test();
        } catch (Exception ex) {

        }
    }

    public void test() throws Exception {
        Connection conn = null;
        try {
            Class.forName("com.gbase.jdbc.Driver").newInstance();
            conn = DriverManager
                    .getConnection(URL);

            Statement stmt = null;
            ResultSet rs = null;
            try {
                stmt = conn.createStatement(
                        java.sql.ResultSet.TYPE_FORWARD_ONLY,
                        java.sql.ResultSet.CONCUR_UPDATABLE);

                // 创建表
                stmt.executeUpdate("DROP TABLE IF EXISTS
```

```
autoIncTutorial");
                stmt.executeUpdate("CREATE TABLE autoIncTutorial ("
                        + "priKey INT NOT NULL AUTO_INCREMENT, "
                        +   "dataField   VARCHAR(64),   PRIMARY   KEY
(priKey))");

                // 获取自增一字段值
                rs = stmt.executeQuery("SELECT priKey, dataField "
                        + "FROM autoIncTutorial");
                rs.moveToInsertRow();
                rs.updateString("dataField", "AUTO INCREMENT here?");
                rs.insertRow();

                rs.last();

                int autoIncKeyFromRS = rs.getInt("priKey");
                rs.close();
                rs = null;
                System.out.println("Key returned for inserted row: "
                        + autoIncKeyFromRS);
            } finally {
                if (rs != null) {
                    try {
                        rs.close();
                    } catch (SQLException ex) {

                    }
                }
                if (stmt != null) {
                    try {
                        stmt.close();
                    } catch (SQLException ex) {
```

```
                    }
                }
            }
        } catch (SQLException ex) {
            // 处理错误
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
        } finally {
            conn.close();
        }
    }
}
```

7.11 GBase JDBC 在 Jboss 应用中使用示例

该示例主要内容为在 JBOSS 服务器上配置 GBase 数据源。对 JBOSS 本身的安装以及 Web 工程的创建不做讨论。

本示例基本信息如下:

JBOSS: jboss-4.0.3

GBase JDBC: GBase JDBC 驱动

JDK: JDK1.6

JBOSS 安装路径为: $JBOSS_HOME

步骤如下:

1) 将 GBase JDBC 驱动包 GBase JDBC 驱动 拷贝至 jboss 目录$JBOSS_HOME \server\default\lib 下。

2) 进入目录$JBOSS_HOME \server\default\deploy，并在该目录下创建 gbasedb-ds.xml 文件，文件内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
- <datasources>
- <local-tx-datasource>
- <!--
 This connection pool will be bound into JNDI with the name "java:/GBaseDB"
  -->
  <jndi-name>GBaseDB</jndi-name>

<connection-url>jdbc:gbase://localhost:5258/test</connection-url>
  <driver-class>com.gbase.jdbc.Driver</driver-class>
  <user-name>root</user-name>
  <password>123456</password>
  <min-pool-size>5</min-pool-size>
- <!--
 Don't set this any higher than max_connections on your
        GBase server, usually this should be a 10 or a few 10's
        of connections, not hundreds or thousands
  -->
  <max-pool-size>20</max-pool-size>
- <!--
```

```
    Don't allow connections to hang out idle too long,
            never longer than what wait_timeout is set to on the
            server...A few minutes is usually okay here,
            it depends on your application
            and how much spikey load it will see
     -->
     <idle-timeout-minutes>5</idle-timeout-minutes>
    - <!--
     If you're using GBase 8a MPP Cluster JDBC 3.1.8 or newer, you can use
                our implementation of these to increase the robustness of the connection pool.
     -->

<exception-sorter-class-name>com.gbase.jdbc.integration.jboss.ExtendedGBaseExceptionSorter</exception-sorter-class-name>

<valid-connection-checker-class-name>com.gbase.jdbc.integration.jboss.GBaseValidConnectionChecker</valid-connection-checker-class-name>
    </local-tx-datasource>
    </datasources>
```

3) 在 Web 工程的 WebRoot\META-INF 目录下添加 jbosscmp-jdbc.xml 文件，内容如下：

```
    <?xml version="1.0" encoding="UTF-8"?>
    <jbosscmp-jdbc>
```

```
<defaults>
<datasource>java:/GBaseDB</datasource>
</defaults>
</jbosscmp-jdbc>
```

4) 创建测试 servlet，代码如下：

```
package gbasejboss;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class gbaseServlet extends HttpServlet {

    public gbaseServlet() {
        super();
    }
    public void destroy() {
        super.destroy();

    }
    public void doGet(HttpServletRequest request,
HttpServletResponse response)
```

```
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            System.out.println("come");
            //获得连接池
            Context initCtx = new InitialContext();
            DataSource ds =
(DataSource)initCtx.lookup("java:/GBaseDB");
            //获得连接
            conn = ds.getConnection();
            if (conn != null){
                out.println("The Gbase connection is ok!!");
                System.out.println("ok");
            }else{
                out.println("The Gbase connection occur error!");
                System.out.println("dddd");
            }
            //测试 SQL 语句
            Statement st = conn.createStatement();
            ResultSet rs1= st.executeQuery("select cust_name from
customers where cust_id=1");
            while(rs1.next()){
                //输出 SQL 语句结果
                out.println(rs1.getString(1));
                System.out.println(rs1.getString(1));
            }
        }catch(Exception e){
            System.out.println("Exception"+e);
        }
    }
```

```
        public void doPost(HttpServletRequest request,
HttpServletResponse response)
                throws ServletException, IOException {
            this.doGet(request, response);
        }

        public void init() throws ServletException {

        }
    }
```

5) 修改 Web 工程 WebRoot\WEB-INF 目录下的 web.xml 文件，添加如下内容:

```
    <servlet-mapping>
        <servlet-name>gbaseServlet</servlet-name>
        <url-pattern>/gbaseServlet</url-pattern>
    </servlet-mapping>
```

6) 将 Web 工程部署到 Jboss 服务器，启动服务器，通过 url:

http://localhost:8080/GbaseJboss/gbaseServlet 测试 servlet

结果应显示如下:

The Gbase connection is ok!! GBase 8a

7.12 GBase JDBC 在 Tomcat 应用中使用示例

该示例主要内容为在 Tomcat 服务器上配置 GBase 数据源。对 Tomcat 本身的安装以及 Web 工程的创建不做讨论。

本示例基本信息如下:

Tomcat: tomcat-5.5.30

GBase JDBC: GBase JDBC 驱动

JDK： JDK1.6

Tomcat 安装路径假设为：$TOMCAT_HOME

步骤如下：

1） 将 GBase JDBC 驱动包 GBase JDBC 驱动拷贝至 tomcat 目录 $TOMCAT_HOME\common\lib 。

2） 通过在$TOMCAT_HOME\conf\Catalina\localhost 目录下增加声明资源文件，该文件以 Web 应用名称为名 (例:GBaseapp.xml)，配置 JNDI DataSource，内容部分如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Resource
      name="jdbc/GBaseDB"
      type="javax.sql.DataSource"
      password="somepassword"
      driverClassName="com.gbase.jdbc.Driver"
      maxIdle="2"
      maxWait="50"
      username="user"
      url="jdbc:gbase://localhost:5258/test"
      maxActive="4"/>
</Context>
```

3） 修改 Web 应用目录 WebRoot\WEB-INF 下的 web.xml，添加如下内容：

```
<servlet>
```

```
        <servlet-name>gbasetomcatServlet</servlet-name>
        <servlet-class>gbasetomcat.gbasetomcatServlet
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>gbasetomcatServlet</servlet-name>
        <url-pattern>/gbasetomcatServlet</url-pattern>
      </servlet-mapping>
    <resource-ref>
          <description>DB Connection</description>
          <res-ref-name>jdbc/GBaseDB</res-ref-name>
          <res-type>javax.sql.DataSource</res-type>
          <res-auth>Container</res-auth>
    </resource-ref>
```

4) 创建 Servlet 测试连接池，代码如下：

```
package gbasetomcat;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class gbasetomcatServlet extends HttpServlet {

    public gbasetomcatServlet() {
        super();
    }
    public void destroy() {
        super.destroy();
    }

    public void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            System.out.println("come");
            Context initCtx = new InitialContext();
            //获得连接池
            DataSource ds =
            (DataSource)initCtx.lookup("java:comp/env/jdbc/GBase
DB");
            //创建连接
            conn = ds.getConnection();
            if (conn != null){
                out.println("The Gbase connection is ok!!");
                System.out.println("ok");
```

```
                }else{
                    out.println("The Gbase connection occur error");
                    System.out.println("error");
                }
                //使用连接创建 Statement 对象执行 SQL 语句
                Statement st = conn.createStatement();
                ResultSet rs1= st.executeQuery("select cust_name from
customers  where cust_id=1");
                //获得执行结果，输出结果
                while(rs1.next()){
                    out.println(rs1.getString(1));
                    System.out.println(rs1.getString(1));
                }
            }catch(Exception e){
                System.out.println("Exception"+e);
            }
        }

        public void doPost(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException {
            this.doGet(request, response);
        }

        public void init() throws ServletException {
        }
    }
```

启动 Tomcat 服务器，通过 http://localhost:8080/GbaseTomcat/gbasetomcatServlet 访问，如果输出结果如下，说明 GBase 连接池配置成功。结果:

```
The Gbase connection is ok!! GBase 8a
```

7.13 GBase JDBC 集群高可用性示例

本样例代码适用于 GBaseJDBC8.3.81.53 及以上版本。

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class SampleAutoTransHost {

    static String dbUrl =
"jdbc:gbase://192.168.2.136:5258/test?user=root&password=root&useUni
code=true&characterEncoding=utf8&failoverEnable=true&hostList=192.16
8.2.138,192.168.2.139";

    public static void main(String[] args) {

        Connection conn = null;
        Statement stm = null;
        ResultSet rs = null;
        try {
            Class.forName("com.gbase.jdbc.Driver");

            conn = DriverManager.getConnection(dbUrl);

            stm = conn.createStatement();

            rs = stm.executeQuery("select 1");

            rs.next();

            System.out.println(rs.getObject(1));
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
```

```
            } finally {
                if (rs != null) {
                    try {
                        rs.close();
                    } catch (SQLException e) {
                    }
                }
                if (stm != null) {
                    try {
                        stm.close();
                    } catch (SQLException e) {
                    }
                }
                if (conn != null) {
                    try {
                        conn.close();
                    } catch (SQLException e) {
                    }
                }
            }
        }
    }
```

7.14 GBase JDBC 集群高可用负载均衡

用例假设如下场景:

1、 集群环境

集群节点 ip 192.168.111.96, 192.168.5.212, 12.168.7.174

2、 需求

在创建连接时，把链接请求按照轮询方式均摊到各个节点

本样例代码适用于 GBaseJDBC8.3.81.53 及以上版本。

```
package com.gbase.jdbc.simple;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class SampleGBaseJDBCLoadbalance {

    /**
    * 数据库连接串
    * failoverEnable = true
    * hostList=192.168.5.212,192.168.7.174
    * gclusterId=gcl1
    * 启用高可用负载均衡
    */
    public static final String URL =
"jdbc:gbase://192.168.111.96:5258/test?user=gbase&password=gbase2011
0531&failoverEnable=true&hostList=192.168.5.212,192.168.7.174&gclust
erId=gcl1";

    /**
    * @param args
    */
    public static void main(String[] args) {
        try {
            Class.forName("com.gbase.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        //准备样例代码用到的表及数据
        prepareTable();
```

```
            //创建线程池
            ExecutorService executorService =
Executors.newCachedThreadPool();
            for (int i = 0; i < 50; i++) {
                executorService.execute(new
SelectThread(String.valueOf(i)));
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            executorService.shutdown();
            try {
                Thread.sleep(50000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("main thread finished");
        }

        private static void prepareTable() {
            Connection conn = null;
            Statement stm = null;
            try {
                conn = DriverManager.getConnection(URL);
                stm = conn.createStatement();
                stm.executeUpdate("drop table if exists
`loadbalance`");
                stm.executeUpdate("create table loadbalance(a varchar
(10), b varchar (12))");
                stm.executeUpdate("insert into test.loadbalance
values('a1', 'b1')");
                stm.executeUpdate("insert into test.loadbalance
values('a2', 'b2')");
            } catch (SQLException e) {
```

```
                e.printStackTrace();
            } finally {
                if (stm != null) {
                    try {
                        if (!stm.isClosed())
                            stm.close();
                    } catch (Exception e) {
                    }
                }
            }
        }
    }
    class SelectThread implements Runnable {
        private String threadName;

        SelectThread(String n) {
            threadName = n;
        }
        int i =0;
        public void run() {
            while (i < 20) {
                i++;
                work();
                try {
                        Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("end = " + threadName);
        }
        public void work() {
            try {
                Class.forName("com.gbase.jdbc.Driver");
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
```

```
            Connection conn = null;
            Statement stm = null;
            ResultSet rs = null;
            String ip= null;
            try {
                conn =
DriverManager.getConnection(SampleGBaseJDBCLoadbalance.URL);
                ip = (com.gbase.jdbc.GBaseConnection)conn).getHost();
                System.out.println(this.threadName + " 线程: 获取连接
on " + ip);
                stm = conn.createStatement();
                rs = stm.executeQuery("select loadbalance.*, sleep(5)
from test.loadbalance");
                while(rs.next()) {
                    System.out.println(rs.getString(1));
                }
            } catch (SQLException e) {
                System.out.println(this.threadName + " 线程: 异常 on
"+ip);
                e.printStackTrace();
            } finally {
                try {
                    rs.close();
                } catch (SQLException e) {
                }
                try {
                    stm.close();
                } catch (SQLException e) {
                }
                try {
                    conn.close();
                } catch (SQLException e) {
                }
            }
        }
    }
```

7.15 GBase JDBC 流式读取使用示例

本示例展示了通过 JDBC 流模式逐行读取的实现方式：

示例如下：

```
package com.gbase.jdbc.simple;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
public class StatementReadBigData {
public void testBigData() throws Exception {
        try {

            Connection Conn = getConnectionWithProps();

            Statement streamStmt = null;

            try {
                streamStmt = Conn.createStatement(
                        java.sql.ResultSet.TYPE_FORWARD_ONLY,
                        java.sql.ResultSet.CONCUR_READ_ONLY);
                streamStmt.setFetchSize(Integer.MIN_VALUE); //必须设置为 Integer.MIN_VALUE,以流式读取；也可以通过修改 jdbc url,通过 defaultFetchSize 参数设置

                this.rs  =  streamStmt.executeQuery("SELECT  DUMMYID, DUMMYNAME "
```

```
                    + "FROM testbigdata ORDER BY DUMMYID");

            while (this.rs.next()) {
                this.rs.getString(1);
            }

        } finally {
            if (streamStmt != null) {
                streamStmt.close();
            }
            if(rs != null) {
                rs.close();
            }
        }
    } finally {
        if(Conn != null) {
            Conn.close();
        }
    }
}
```

7.16 GBase JDBC 获取加载任务信息示例

Jdbc 新增了用于获取加载任务 ID 号，加载数据跳过行数的功能。因为 jdbc 标准接口并不包含该方法定义，故用户在使用时需要将标准的 Statement 转化为 com.gbase.jdbc.StatementImpl 类型方可使用。

因为加载 sql 语法是在 8611 版本中引入的，故该功能仅支持 8611 版本以上集群。

1） 获取数据跳过行数示例

该功能需要 gbase-connector-java-8.3.81.53-build54.1 以上版本支持（含 54.1）

```
String loadSql="load data infile 'ftp://test:123456@192.168.7.182/1.txt' into table loadtest fields terminated by ','";
Connection conn =DriverManager.getConnection(URL);
StatementImpl stmt = (StatementImpl) conn.createStatement();
stmt.execute("drop table if exists loadtest");
stmt.execute("create table loadtest(a int, b varchar(100))");
stmt.executeUpdate(loadSql);
long skippedLines = stmt.getSkippedLines();
```

2） 获取任务 ID 示例

该功能需要 gbase-connector-java-8.3.81.53-build54.2 以上版本支持(含 54.2)

```
String loadSql="load data infile 'ftp://test:123456@192.168.7.182/1.txt' into table loadtest fields terminated by ','";
Connection conn =DriverManager.getConnection(URL);
StatementImpl stmt = (StatementImpl) conn.createStatement();
stmt.execute("drop table if exists loadtest");
stmt.execute("create table loadtest(a int, b varchar(100))");
stmt.executeUpdate(loadSql);
long taskID = stmt.getLoadTaskID();
```

7.17 Ipv6 使用示例

从 jdbc 版本 build54.4.8 开始支持使用 ipv6 与数据库连接。使用方式除了 url 设置有区别外，其他操作完全一样。

使用步骤如下：

1) 首先安装支持 ipv6 对应的集群版本，安装后假设 ip 地址为：2001:da8:e000::1:1:1

2) 使用 jdbc 对集群进行操作,支持如下两种配置方式

设置 url 为:

jdbc:gbase://

（2001:da8:e000::1:1:1）:5258/bht?user=gbase&password=gbase20110531

或者

jdbc:gbase://[2001:da8:e000::1:1:1]:5258/bht?user=gbase&password=gbase20110531

即只需要将协议中 ip 地址用 () 或者[]括起来，其他使用方式同原来一样。

注意:

1. 在使用 hostList 参数时，该参数对应的 ip 不需要使用 () 或者[]进行包围。
2. 使用[]时需要 jdbc 55.2.1 版本支持。

7.18 Utf8mb4 编码使用示例

从 jdbc 版本 build54.4.8 开始支持 utf8mb4 编码。如果数据库设置使用了 utf8mb4 编码，相应的 jdbc 也要对应支持。

使用步骤如下:

1) 集群已安装支持 utf8mb4 版本，并且设置了字符集编码为 utf8mb4

2) 设置 url 为如下:

jdbc:gbase://192.168.7.126:5258/bht?user=gbase&password=gbase20110531&useUnicode=true&characterEncoding=utf8

即只需要在 url 上设置 useUnicode=true&characterEncoding=utf8 即可。

7.19 Gb18030 编码使用示例

从 jdbc 版本 build55.2.1 开始支持 gb18030 编码。如果数据库设置使用了 gb18030 编码，相应的 jdbc 也要对应支持。

使用步骤如下：

3）集群已安装支持 gb18030 版本，并且设置了字符集编码为 gb18030

4）设置 url 为如下：

jdbc:gbase://192.168.7.126:5258/bht?user=gbase&password=gbase20110531&useUnicode=true&characterEncoding= gb18030

即只需要在 url 上设置 useUnicode=true&characterEncoding= gb18030 即可。

7.20 连接虚拟集群

jdbc 驱动从 build55 版本开始支持虚拟集群的连接，如果 8a 集群支持虚拟集群，请申请 jdbc build55 以上版本。

使用方式非常简单，通过 url 参数配置 vcName 为具体的虚拟集群名称即可。注意该参数必须配置，且url必须明确指定数据库名称。

具体配置样例如下：

jdbc:gbase://192.168.8.22/gbase?user=gbase&password=gbase20110531&vcName=vc2

7.21 Jdbc 使用 ssl 加密传输数据

如果使用 jdbc 进行 ssl 数据传输，前提是必须 server 支持，必须先开启 server 端的 ssl 功能。

开启集群步骤如下（可参考集群手册，以下只给出开启步骤）

1）生成 ssl 文件（直接在 linux 下执行即可）

```
openssl genrsa 2048 > ca-key.pem

openssl req -sha1 -new -x509 -nodes -days 3650 -key ca-key.pem > ca-cert.pem

openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout server-key.pem > server-req.pem

openssl rsa -in server-key.pem -out server-key.pem

openssl x509 -sha1 -req -in server-req.pem -days 730 -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > server-cert.pem

openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout client-key.pem > client-req.pem

openssl rsa -in client-key.pem -out client-key.pem

openssl x509 -sha1 -req -in client-req.pem -days 730 -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > client-cert.pem
```

此步骤可能出现提示，直接忽略即可

2）拷贝三个文件到某个目录下，在gcluster*.cnf下设置如下内容

```
ssl-ca=/usr/local/myssl/ca-cert.pem

ssl-cert=/usr/local/myssl/server-cert.pem

ssl-key=/usr/local/myssl/server-key.pem
```

3）重启集群然后通过show variables like '%SSL%'查看是否开启ssl功能。如下为开启：

```
show variables like '%ssl%';
+---------------+---------------------------------+
| Variable_name | Value                           |
+---------------+---------------------------------+
| have_openssl  | YES                             |
| have_ssl      | YES                             |
| ssl_ca        | /usr/local/myssl/ca-cert.pem    |
| ssl_capath    |                                 |
| ssl_cert      | /usr/local/myssl/server-cert.pem |
| ssl_cipher    |                                 |
| ssl_key       | /usr/local/myssl/server-key.pem |
+---------------+---------------------------------+
```

经过前面三步集群已经开启ssl功能，针对jdbc按照如下使用步骤

1）生成jdbc连接用密钥

```
keytool -import -alias GBaseCACert -file ca-cert.pem -keystore truststore
```

说明：ca-cert.pem为生成ssl文件时生成的文件，执行该步骤后会提示输入认证，即密码，比如输入password1（jdbc会用到）

```
openssl x509 -outform DER -in client-cert.pem -out client.cert

keytool -import -file client.cert -keystore keystore -alias GBaseClientCertificate
```

说明：client.cert为生成ssl文件时生成的文件，执行该步骤后会提示输入认证，即密码，比如输入password1，（jdbc会用到）

2）上述步骤会生成两个文件truststore，keystore，将这两个文件拷贝到jdbc可以访问的路径下

3） 按照如下样例编写代码

```
String url = "jdbc:gbase://192.168.8.27:5258/gbase?user=root&useSSL=true&requireSSL=true";
            String trustStorePath = "D:\\JDBCTest\\src\\test-certs\\truststore";
            String keyStorePath = "D:\\JDBCTest\\src\\test-certs\\keystore";
            System.setProperty("javax.net.ssl.keyStore", keyStorePath);
            System.setProperty("javax.net.ssl.keyStorePassword", "password1");
            System.setProperty("javax.net.ssl.trustStore", trustStorePath);
            System.setProperty("javax.net.ssl.trustStorePassword", "password1");
```

```
Connection conn = DriverManager.getConnection(url);
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("show status like '%SSL%'");
while(rs.next()){
      System.out.println(rs.getString(1)+"-----"+rs.getString(2));
}
```

以上就是使用jdbc ssl功能步骤，注意黄色背景设置

7.22 通过 Jdbc 修改过期密码

jdbc 版本从 build55.3.1 开始支持通过 jdbc 修改数据库密码。使用方式为通过新增方法 PasswordChangeUtil.changePassword(String url, String newPassword);其中参数 1 为合理的 url,且 url 中包含原始的用户名密码信息,参数 2 为新的密码。

样例如下:

```
String url="jdbc:gbase://192.168.7.126:5258/gbase?user=bht&password=111111";
PasswordChangeUtil.changePassword(url, "222222");
```

7.23 通过 url 参数控制取出的列信息是否进行大小写转换。

jdbc 版本从 build55.4.1 开始支持通过 jdbc url 参数控制对 getColumnName 和 getColumnLabel 方法返回的结果进行大小写转换。

样例如下:

1) 转换为大写

```
String url="jdbc:gbase://192.168.7.126:5258/gbase?user=bht&password=111111&
```

caseSensitiveFlag=2″;

如下代码label,name为大写

```
String sql = "select col1 tzm ,Col2 列a,col3 Hj from bht2";
        ResultSet rs = this.stm.executeQuery(sql);
        ResultSetMetaData rsmd = rs.getMetaData();
        int count = rsmd.getColumnCount();
        while(rs.next()){
            for(int i=1;i<=count;i++){
                String label = rsmd.getColumnLabel(i);
                String name = rsmd.getColumnLabel(i);
            }
        }
        rs.close();
```

8 采用 kerberos 认证方式与 GCluster 或 8a 单机连接

Jdbc 从 build54.4.7 版本开始支持支持 kerberos 认证方式与集群或者 8a 单机进行连接。使用前必须确保数据库版本支持 kerberos 认证并且已经配置好 kerberos 认证相关服务。具体使用可以参考数据库手册要求。

以下操作仅在认为集群或者 8a 已经配置好 kerberos 相关服务基础上进行的操作。

1:需要从集群获取keytab文件，拷贝到某一目录

2: 需要执行kinit命令初始化票据凭证，最后一个参数为kerberos client端 principal name

1) linux: kinit -kt /opt/gcluster/ktest1.keytab ktest1@gbase.cn

2) window: kinit -k -t D:\\svn\\kerberos\\ktest1.keytab ktest1@gbase.cn

3: url进行设置

enableKerberosFlag=true : 开启认证方式

clientPrincipalName=ktest1 : 设置客户端principal name，注意不需要输入@gbse.cn

kerberosKeyTab="D:\\svn\\kerberos\\ktest1.keytab",设置keytab路径

4: 从kerberos服务器获取krb5.conf文件 (linux默认安装在/etc下面)，如果jdbc部署在linxu服务器下，直接拷贝到/etc下即可；如果部署在windows

将其修改名字为 krb5.ini 然后复制到 C:\Windows 下面，注意里面的路径信息请修改为实际存在的路径。

5: 确保 jdbc 所在机器的时间与 kerberos 服务器时间一致

以上配置正确后可以使用如下方式进行连接

//是否输出认证过程中重要参数信息。生产环境可以不设置。

```
System.setProperty("sun.security.krb5.debug","true");
System.setProperty("sun.security.jgss.debug","true");

String url
="jdbc:gbase://192.168.6.122:5258/test?user=ktest&traceProtocol=false"
        + "&profileSql=false"
        + "&enableKerberosFlag=true"
        + "&clientPrincipalName=ktest1"
        +"&kerberosKeyTab=D:\\svn\\kerberos\\ktest1.keytab"
        ;
Class.forName("com.gbase.jdbc.Driver");
Connection conn = DriverManager.getConnection(url);
Statement st =    conn.createStatement();

st.execute("insert into bht1 values(1,'bht')");
st.setMaxRows(2);
ResultSet rs =st.executeQuery("select * from bht1");
while(rs.next()){
    System.out.println(rs.getString(1)+rs.getString("b"));
}
conn.close();
System.out.println("fin");
```

从上代码可以看到连接集群不再需要使用 password，而仅仅只需要 kerberos 服务器认证过的 user 即可。

9 GBase JDBC 常见问题和解决办法

有一些问题看起来是 GBase JDBC 用户经常会遇到的。本节讲述它们的症状和解决办法。

9.1GBase JDBC 连接数据库异常

通过 GBase JDBC 试图连接一个数据库时，得到如下的异常：

SQLException: Server configuration denies access to data source

SQLState: 08001

VendorError: 0

什么原因？通过 GBase 命令行管理工具连接正常。

回答：

GBase JDBC 必须使用 TCP/IP 套接字来连接 GBase，由于 Java 不支持 Unix Domain 套接字。因此，当 GBase JDBC 连接 GBase 时，GBase 中的安全管理器会使用它的授权表来决定连接是否被允许。

用户必须增加授权来允许这种情况。下面的例子会告诉用户如何来做（不是最安全的）。

从 sqlcli 命令行客户端，作为一个可以授权的用户登录，利用如下的命令：

GRANT ALL PRIVILEGES ON [dbname].* to '[user]'@'[hostname]' identified by '[password]'

以用户的数据库的名字代替[dbname]，以用户的用户名代替[user]，用要连接 GBase JDBC 的主机名称代替[hostname]，并用用户密码代替[password]。注意，对于从本地主机进行连接的主机名部分，RedHat Linux 将失败。在这种情况下，对于[hostname]值用户需要使用 localhost.localdomain。在这之后使用 FLUSH PRIVILEGES 命令。

注意:

除非添加了“—host”标志，并为主机使用了不同于 localhost 的其他设置，否则将无法使用 GBase 命令行客户端测试连通性。如果用户使用特定的主机名 localhost，那么 sqlcli 命令行客户端会使用 Unix Domain 套接字。如果用户测试与 localhost 的连通性，那么使用“127.0.0.1”来作为主机名。

警告:

如果用户不明白 GRANT 是做什么的，或它如何工作，用户应该在试图改变权限之前阅读并理解 GBase 手册中的“GBase 访问权限系统”以及“常规安全问题”两节。

在 GBase 中不恰当地改变权限和许可，可能会使服务器不再具有最佳的安全性能。

9.2No Suitable Driver 问题

我的应用程序抛出一个 SQL 异常“No Suitable Driver”，这是为什么？

回答:

有可能发生了两种情况之一，或者驱动不在用户的 CLASSPATH 中。或者用户的 URL 格式不正确（参见 2.2 章节）。

9.3关于--skip-networking 问题

我想在一个小程序或应用程序中使用 GBase JDBC，但是得到一个类似如下的异常:

```
SQLException: Cannot connect to GBase server on host:5258.
Is there a GBase server running on the machine/port you
are trying to connect to?
(java.security.AccessControlException)
```

SQLState: 08S01

VendorError: 0

回答:

或许是因为用户正在运行 Applet，用户的 GBase server 安装时设置了“--skip-networking”选项，或者是由于 GBase server 运行在防火墙之后。

Applet 仅能使网络连接返回运行 Web 服务器的机器，该 Web 服务器提供了用于 Applet 的.class 文件。这意味着，要想使其工作，GBase 必须运行在相同的机器上（或必须使某类端口重定向）。这也意味着，你无法通过你的本地文件系统来测试 Java 程序，你必须将它们放在 Web 服务器上。

由于 Java 不支持 Unix Domain 套接字，GBase JDBC 只能使用 TCP/IP 与 GBase 通信。如果 GBase 启动时带有“--skip-networking”标志，或有防火墙，与 GBase 的 TCP/IP 通信可能会受到影响。

如果 GBase 启动时设置了“--skip-networking”选项，用户可以在文件 /etc/gs.cnf 中把它注释掉。如果用户的 GBase server 在防火墙后，那么用户需要修改防火墙配置使得允许运行用户的 Java 代码的主机在 GBase 侦听的端口（缺省为 5258）上给 GBase server 进行连接。

9.4GBase 自动关闭连接问题

我有一个小服务程序/应用程序正常地工作了一白天，然后晚上就停止工作了:

回答:

GBase 在无活动 8 小时后就自动关闭连接。你或许需要使用能处理失效连接的连接池，或使用 autoReconnect 参数。

此外，用户应该在应用程序中捕获 SQL 异常并处理它们，而不是一直传播它们直到用户的程序退出，这只是一个好的编程习惯。当在处理查询过程中遇到网络连通性问题时，GBase JDBC 会把 SQLState （参见用户的 APIDOCS 中的

java.sql.SQLException.getSQLState()) 置为 “08S01” 。这时用户的程序应该重新连接 GBase 。

下面（简单的）例子说明了什么样的代码可以处理这理这些异常： 使用重试逻辑的事务例子

```
public void doBusinessOp() throws SQLException
{
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    //
    // How many times do you want to retry the transaction
    // (or at least _getting_ a connection)?
    //
    int retryCount = 5;
    boolean transactionCompleted = false;
    do {
        try {
            conn = getConnection(); // assume getting this from a
            // javax.sql.DataSource, or the
            // java.sql.DriverManager
            conn.setAutoCommit(false);
            //
            // Okay, at this point, the 'retry-ability' of the
            // transaction really depends on your application logic,
            // whether or not you're using autocommit (in this case
            // not), and whether you're using transacational storage
            // engines
            //
            // For this example, we'll assume that it's _not_ safe
            // to retry the entire transaction, so we set retry count
            // to 0 at this point
            //
            // If you were using exclusively transaction-safe
tables,
            // or your application could recover from a connection
```

```
going
                    // bad in the middle of an operation, then you would not
                    // touch 'retryCount' here, and just let the loop repeat
                    // until retryCount == 0.
                    //
                    retryCount = 0;
                    stmt = conn.createStatement();
                    String query = "SELECT foo FROM bar ORDER BY baz";
                    rs = stmt.executeQuery(query);
                    while (rs.next()) {}
                    rs.close();
                    rs = null;
                    stmt.close();
                    stmt = null;
                    conn.commit();
                    conn.close();
                    conn = null;
                    transactionCompleted = true;
                } catch (SQLException sqlEx) {
                    //
                    // The two SQL states that are 'retry-able' are 08S01
                    // for a communications error, and 41000 for deadlock.
                    //
                    // Only retry if the error was due to a stale connection,
                    // communications problem or deadlock
                    //
                    String sqlState = sqlEx.getSQLState();
                    if ("08S01".equals(sqlState) ||
"41000".equals(sqlState)) {
                        retryCount--;
                    }else {
                        retryCount = 0;
                    }
                } finally {
                    if (rs != null) {
                        try {
                            rs.close();
```

```
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this . . .
                }
            }
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this as well . . .
                }
            }
            if (conn != null) {
                try {
                    //
                    // If we got here, and conn is not null, the
                    // transaction should be rolled back, as not
                    // all work has been done
                    try {
                        conn.rollback();
                    } finally {
                        conn.close();
                    }
                } catch (SQLException sqlEx) {
                    //
                    // If we got an exception here, something
                    // pretty serious is going on, so we better
                    // pass it up the stack, rather than just
                    // logging it. . .
                    throw sqlEx;
                }
            }
        }
    } while (!transactionCompleted && (retryCount > 0));
}
```

9.5使用 JDBC 来更新结果集问题

我使用 JDBC 来更新结果集，但是却得到一个异常说我的结果集没有更新。

回答:

因为 GBase 没有行标识， GBase JDBC 只能更新来自于至少有一个主键的表上的查询的结果集，这个查询必须选择所有的主键且这个查询只能跨越一个表(也就是没有连接)，这是 JDBC 规范中的要点。

9.6使用 jdbc 获取 HH:MM:SS.xxxxxx 格式的时间

Jdbc 规范对与 Time 类型的定义并没有处理微妙的部分,当实际情况真的需要处理如下类似语句时 time(now() + interval 1 microsecond) 结果集的 getTime 方法是报格式错误的,针对此做了一个增强，通过 getTime 方法可以获取 HH:MM:SS 的 time 类型，如果确实需要获取完整值，可以通过 getObject 获取，然后转成 String 型。

10 JDBC 使用注意事项

10.1 Jdbc 通过 execute*(sql)方法执行特殊 sql 语句

1. 在 GBase JDBC 中，不要执行“set names”来设置字符集，因为驱动不会发现字符集已经改变，而是会继续使用在初始化连接设置时使用的字符集。
2. 在 GBase JDBC 中，不要执行“use database”来试图改变当前 Statement 使用的数据库，因为驱动不会发现你已经切换了数据库，针对使用人员不能通过接口明确知道当前使用的数据库，容易导致将目标 sql 执行到非预期的数据库上。

正确做法是通过 Connection.setCatalog()方式切换数据库，且只能在 Statement 创建之前改变数据库，一旦 Statement 创建了就不能修改其使用的数据库。

正确用例参考如下：

```
conn.setCatalog("test1");
Statement stm1= conn.createStatement();
conn.setCatalog("test2");
Statement stm2= conn.createStatement();
```

这样 stm1,stm2 分别使用 test1,test2。

GBASE®

南大通用数据技术股份有限公司
General Data Technology Co., Ltd.

官方微信

GBase 8a 技术社区

技术支持热线：400-013-9696