GBase 8c

JDBC 使用手册

www.gbase.cn

GBase 8c JDBC使用手册，南大通用数据技术股份有限公司
GBase 版权所有©2022，保留所有权利

版权声明

本文档所涉及的软件著作权及其他知识产权已依法进行了相关注册、登记，由南大通用数据技术股份有限公司合法拥有，受《中华人民共和国著作权法》、《计算机软件保护条例》、《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的南大通用数据技术股份有限公司的版权信息由南大通用数据技术股份有限公司合法拥有，受法律的保护，南大通用数据技术股份有限公司对本文档可能涉及到的非南大通用数据技术股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅，并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经南大通用数据技术股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将视为侵权，南大通用数据技术股份有限公司具有依法追究其责任的权利。

本文档中包含的信息如有更新，恕不另行通知。您对本文档的任何问题，可直接向南大通用数据技术股份有限公司告知或查询。

未经本公司明确授予的任何权利均予保留。

通讯方式

南大通用数据技术股份有限公司

天津市滨海高新区华苑产业园开华道22号普天大厦20-23层

电话：400-013-9696　　邮箱：info@gbase.cn

商标声明

GBASE® 是南大通用数据技术股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由南大通用数据技术股份有限公司合法拥有，受法律保护。未经南大通用数据技术股份有限公司书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯南大通用数据技术股份有限公司商标权的，南大通用数据技术股份有限公司将依法追究其法律责任。

目　录

目　录.....II

1 概述.....3

2 开发流程.....3

3 加载驱动.....4

4 连接数据库.....4

4.1 函数原型.....4

4.2 参数.....4

4.3 示例.....13

5 执行 SQL 语句.....15

5.1 执行普通 SQL 语句.....15

5.2 执行预编译 SQL 语句.....16

5.3 调用存储过程.....16

5.4 执行批处理.....17

6 处理结果集.....18

6.1 设置结果集类型.....18

6.2 在结果集中定位.....20

6.3 获取结果集中光标的位置.....20

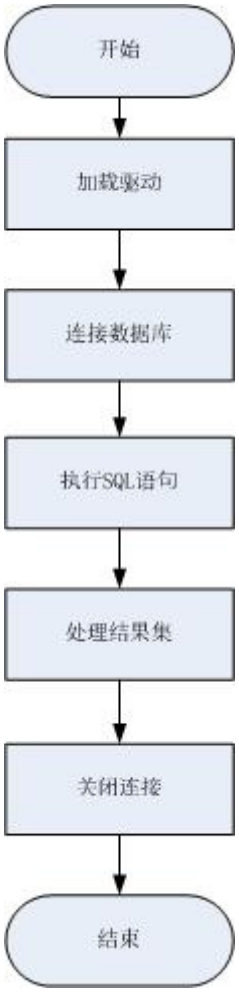
6.4 获取结果集中的数据.....21

7 关闭连接.....21

1 概述

JDBC（Java Database Connectivity，Java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问接口，应用程序可基于它操作数据。GBase8c 库提供了对 JDBC 4.0 特性的支持，不支持 JDBC 桥接 ODBC 方式。

2 开发流程



3 加载驱动

在创建数据库连接之前，需要先加载数据库驱动程序。

加载驱动的方法：

在 JVM 启动时参数传递：java -Djdbc.drivers=cn.gbase8c.Driver jdbctest

4 连接数据库

在创建数据库连接之后，才能使用它来执行 SQL 语句操作数据。

4.1 函数原型

JDBC 提供了三个方法，用于创建数据库连接。

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

4.2 参数

表 1 数据库连接参数

参数	描述
url	以 gbase8c-jdbc-3.0.0B01.jar 为例，gbase8c-jdbc-3.0.0B01.jar 的数据库连接描述符格式如下 jdbc:gbase8c:database jdbc:gbase8c://host/database jdbc:gbase8c://host:port/database jdbc:gbase8c://host:port/database?param1=value1&param2=value2 jdbc:gbase8c://host1:port1,host2:port2/database?param1=value1&param2=value2 说明：

	● database 为要连接的数据库名称。 ● host 为数据库服务器名称或 IP 地址。 连接 Administrator 的机器与 Administrator 不在同一网段时，host 指定的 IP 地址应为 Manager 界面上所设的 coo.cooListenIp2（应用访问 IP）的取值。 由于安全原因，数据库主节点禁止 Administrator 内部其他节点无认证接入。如果要在 Administrator 内部访问数据库主节点，请将 JDBC 程序部署在数据库主节点所在机器，host 使用 “127.0.0.1”。否则可能会出现 “FATAL: Forbid remote connection with trust method!” 错误。 建议业务系统单独部署在 Administrator 外部，否则可能会影响数据库运行性能。 缺省情况下，连接服务器为 localhost。 ● port 为数据库服务器端口。 缺省情况下，会尝试连接到 5432 端口的 database。 ● param 为参数名称，即数据库连接属性。 参数可以配置在 URL 中，以 “?” 开始配置，以 “=” 给参数赋值，以 “&” 作为不同参数的间隔。也可以采用 info 对象的属性方式进行配置，详细示例会在本节给出。 ● value 为参数值，即数据库连接属性值。 连接时需配置 connectTimeout、socketTimeout，如果未配置，默认为 0，即不会超时。在 DN 与客户端出现网络故障时，客户端一直未收到 DN 侧 ACK 确认报文，会启动超时重传机制，不断的进行重传。当超时时间达到系统默认的 600s 后才会报超时错误，这也就会导致 RTO 时间很高。
info	数据库连接属性（所有属性大小写敏感）。常用的属性如下： ● PGDBNAME：String 类型。表示数据库名称。（URL 中无需配置该参数，自动从 URL 中解析） ● PGHOST：String 类型。主机 IP 地址。详细示例见下。 ● PGPORT：Integer 类型。主机端口号。详细示例见下。 ● user：String 类型。表示创建连接的数据库用户。 ● password：String 类型。表示数据库用户的密码。 ● enable_ce：String 类型。其中 enable_ce=1 表示 JDBC 支持密态等值查询。 ● loggerLevel：String 类型。目前支持 4 种级别：OFF、INFO、DEBUG、

TRACE。设置为 OFF 关闭日志，设置为 INFO、DEBUG 和 TRACE 记录的日志信息详细程度不同。默认值为 INFO。该参数设置值不区分大小写。

- loggerFile：String 类型。Logger 输出的文件名。需要显示指定日志文件名，若未指定目录则生成在客户端运行程序目录。此参数已废弃，不再生效，如需使用可通过 java.util.logging 属性文件或系统属性进行配置。
- allowEncodingChanges：Boolean 类型。设置该参数值为“true”进行字符集类型更改，配合 characterEncoding=CHARSET 设置字符集，二者使用“&”分隔。
- currentSchema：String 类型。在 search-path 中指定要设置的 schema。
- hostRecheckSeconds：Integer 类型。JDBC 尝试连接主机后会保存主机状态：连接成功或连接失败。在 hostRecheckSeconds 时间内保持可信，超过则状态失效。缺省值是 10 秒。
- ssl：Boolean 类型。以 SSL 方式连接。

ssl=true 可支持 NonValidatingFactory 通道和使用证书的方式：

1、NonValidatingFactory 通道需要配置用户名和密码，同时将 SSL 设置为 true。

2、配置客户端证书、密钥、根证书，将 SSL 设置为 true。

- sslmode：String 类型。SSL 认证方式。取值范围为：disable、allow、prefer、require、verify-ca、verify-full。
  - disable：不使用 SSL 安全连接。
  - allow：如果数据库服务器要求使用，则可以使用 SSL 安全加密连接，但不验证数据库服务器的真实性。
  - prefer：如果数据库支持，那么首选使用 SSL 连接，但不验证数据库服务器的真实性。
  - require：只尝试 SSL 连接，如果存在 CA 文件，则应设置成 verify-ca 的方式验证。
  - verify-ca：只尝试 SSL 连接，并且验证服务器是否具有由可信任的证书机构签发的证书。
  - verify-full：只尝试 SSL 连接，并且验证服务器是否具有由可信任的证书机构签发的证书，以及验证服务器主机名是否与证书中的一致。
- sslcert：String 类型。提供证书文件的完整路径。客户端和服务端证书的类型为 End Entity。
- sslkey：String 类型。提供密钥文件的完整路径。使用时将客户端证书转

换为 DER 格式：

openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt

- sslrootcert：String 类型。SSL 根证书的文件名。根证书的类型为 CA。
- sslpassword：String 类型。提供给 ConsoleCallbackHandler 使用。
- sslpasswordcallback：String 类型。SSL 密码提供者的类名。缺省值：cn.gbase8c.ssl.LibPQFactory.ConsoleCallbackHandler。
- sslfactory：String 类型。提供的值是 SSLSocketFactory 在建立 SSL 连接时用的类名。
- sslfactoryarg：String 类型。此值是上面提供的 sslfactory 类的构造函数的可选参数（不推荐使用）。
- sslhostnameverifier：String 类型。主机名验证程序的类名。接口实现 javax.net.ssl.HostnameVerifier，默认使用 cn.gbase8c.ssl.PGjdbcHostnameVerifier。
- loginTimeout：Integer 类型。指建立数据库连接的等待时间。超时时间单位为秒。
- connectTimeout：Integer 类型。用于连接服务器操作的超时值。如果连接到服务器花费的时间超过此值，则连接断开。超时时间单位为秒，值为 0 时表示已禁用，timeout 不超时。
- socketTimeoutInConnecting：Integer 类型。用于 socket 读取操作的超时值。在建立 PgConnection 到可以执行 sql 语句之前的 socket 读操作超时时间受该参数影响。如果从服务器读取所花费的时间超过此值，则连接关闭。超时时间单位为秒，值为 0 时表示已禁用，timeout 不超时。默认值为 5。
- socketTimeout：Integer 类型。用于 socket 读取操作的超时值。当连接可以执行 sql 语句之后，socket 读操作超时时间受该参数影响。如果从服务器读取所花费的时间超过此值，则连接关闭。超时时间单位为秒，值为 0 时表示已禁用，timeout 不超时。默认值为 0。
- cancelSignalTimeout：Integer 类型。发送取消消息本身可能会阻塞，此属性控制用于取消命令的“connect 超时”和“socket 超时”。超时时间单位为秒，默认值为 10 秒。
- tcpKeepAlive：Boolean 类型。启用或禁用 TCP 保活探测功能。默认为 false。
- logUnclosedConnections：Boolean 类型。客户端可能由于未调用 Connection 对象的 close () 方法而泄漏 Connection 对象。最终这些对象将被垃圾回收，并且调用 finalize () 方法。如果调用者自己忽略了此

	操作，该方法将关闭 Connection。 ● assumeMinServerVersion：String 类型。客户端会发送请求进行 float 精度设置。该参数设置要连接的服务器版本，如 assumeMinServerVersion=9.0，可以在建立时减少相关包的发送。 ● ApplicationName：String 类型。设置正在使用连接的 JDBC 驱动的名称。通过在数据库主节点上查询 pg_stat_activity 表可以看到正在连接的客户端信息，JDBC 驱动名称显示在 application_name 列。缺省值为 PostgreSQL JDBC Driver。 ● connectionExtraInfo：Boolean 类型。表示驱动是否上报当前驱动的部署路径、进程属主用户到数据库。 取值范围：true 或 false，默认值为 false。设置 connectionExtraInfo 为 true，JDBC 驱动会将当前驱动的部署路径、进程属主用户、url 连接配置信息上报到数据库中，记录在 connection_info 参数里；同时可以在 PG_STAT_ACTIVITY 中查询到。 ● autosave：String 类型。共有 3 种：“always”、“never”、“conservative”。如果查询失败，指定驱动程序应该执行的操作。在 autosave=always 模式下，JDBC 驱动程序在每次查询之前设置一个保存点，并在失败时回滚到该保存点。在 autosave=never 模式（默认）下，无保存点。在 autosave=conservative 模式下，每次查询都会设置保存点，但是只会在“statement XXX 无效”等情况下回滚并重试。 ● protocolVersion：Integer 类型。连接协议版本号，目前仅支持 3。注意：设置该参数时将采用 md5 加密方式，需要同步修改数据库的加密方式：gs_guc set -N all -I all -c "password_encryption_type=1" ，重启 Administrator 生效后需要创建用 md5 方式加密口令的用户。同时修改 pg_hba.conf，将客户端连接方式修改为 md5。用新建用户进行登录（不推荐）。 说明： MD5 加密算法安全性低，存在安全风险，建议使用更安全的加密算法。 ● prepareThreshold：Integer 类型。控制 parse 语句何时发送。默认值是 5。第一次 parse 一个 SQL 比较慢，后面再 parse 就会比较快，因为有缓存了。如果一个会话连续多次执行同一个 SQL，在达到 prepareThreshold 次数以上时，JDBC 将不再对这个 SQL 发送 parse 命令。 ● preparedStatementCacheQueries：Integer 类型。确定每个连接中缓存的查询数，默认情况下是 256。若在 prepareStatement () 调用中使用超过 256 个不同的查询，则最近最少使用的查询缓存将被丢弃。0 表示禁用缓存。 ● preparedStatementCacheSizeMiB：Integer 类型。确定每个连接可缓存的最大值（以兆字节为单位），默认情况下是 5。若缓存了超过 5MB 的

查询，则最近最少使用的查询缓存将被丢弃。0 表示禁用缓存。

- databaseMetadataCacheFields：Integer 类型。默认值是 65536。指定每个连接可缓存的最大值。“0” 表示禁用缓存。
- databaseMetadataCacheFieldsMiB：Integer 类型。默认值是 5。每个连接可缓存的最大值，单位是 MB。“0” 表示禁用缓存。
- stringtype：String 类型，可选字段为：false、 "unspecified"、 "varchar"。设置通过 setString () 方法使用的 PreparedStatement 参数的类型，如果 stringtype 设置为 VARCHAR（默认值），则这些参数将作为 varchar 参数发送给服务器。若 stringtype 设置为 unspecified，则参数将作为 untyped 值发送到服务器，服务器将尝试推断适当的类型。
- batchMode：Boolean 类型。用于确定是否使用 batch 模式连接。默认值为 on，表示开启 batch 模式。
- fetchsize：Integer 类型。用于设置数据库连接所创建 statement 的默认 fetchsize。默认值为 0，表示一次获取所有结果。与 defaultRowFetchSize 等价。
- reWriteBatchedInserts：Boolean 类型。批量导入时，该参数设置为 on，可将 N 条插入语句合并为一条：insert into TABLE_NAME values (values1, ..., valuesN), ..., (values1, ..., valuesN); 使用该参数时，需设置 batchMode=off。
- unknownLength：Integer 类型，默认为 Integer.MAX_VALUE。某些 postgresql 类型（例如 TEXT）没有明确定义的长度，当通过 ResultSetMetaData.getColumnDisplaySize 和 ResultSetMetaData.getPrecision 等函数返回关于这些类型的数据时，此参数指定未知长度类型的长度。
- defaultRowFetchSize：Integer 类型。确定一次 fetch 在 ResultSet 中读取的行数。限制每次访问数据库时读取的行数可以避免不必要的内存消耗，从而避免 OutOfMemoryException。缺省值是 0，这意味着 ResultSet 中将一次获取所有行。没有负数。
- binaryTransfer：Boolean 类型。使用二进制格式发送和接收数据，默认值为 “false”。
- binaryTransferEnable：String 类型。启用二进制传输的类型列表，以逗号分隔。OID 编号和名称二选一，例如 binaryTransferEnable=Integer4_ARRAY,Integer8_ARRAY。

  比如：OID 名称为 BLOB，编号为 88，可以如下配置：

  binaryTransferEnable=BLOB 或 binaryTransferEnable=88
- binaryTransferDisEnable：String 类型。禁用二进制传输的类型列表，以

逗号分隔。OID 编号和名称二选一。覆盖 binaryTransferEnable 的设置。

- blobMode：String 类型。用于设置 setBinaryStream 方法为不同类型的数据赋值，设置为 on 时表示为 blob 类型数据赋值，设置为 off 时表示为 bytea 类型数据赋值，默认为 on。
- socketFactory：String 类型。用于创建与服务器 socket 连接的类的名称。该类必须实现了接口“javax.net.SocketFactory”，并定义无参或单 String 参数的构造函数。
- socketFactoryArg：String 类型。此值是上面提供的 socketFactory 类的构造函数的可选参数，不推荐使用。
- receiveBufferSize：Integer 类型。该值用于设置连接流上的 SO_RCVBUF。
- sendBufferSize：Integer 类型。该值用于设置连接流上的 SO_SNDBUF。
- preferQueryMode：String 类型。共有 4 种：“extended”、“extendedForPrepared”、“extendedCacheEverything” 和 “simple”。用于指定执行查询的模式，simple 模式会 excute，不 parse 和 bind；extended 模式会 bind 和 excute；extendedForPrepared 模式为 prepared statement 扩展使用；extendedCacheEverything 模式会缓存每个 statement。
- targetServerType： String 类型。该参数识别主备数据节点是通过查询 URL 连接串中，数据节点是否允许写操作来实现的，默认为 “any”。共有四种：“any”、“master”、“slave”、“preferSlave”：
  - master 则尝试连接到 URL 连接串中的主节点，如果找不到就抛出异常。
  - slave 则尝试连接到 URL 连接串中的备节点，如果找不到就抛出异常。
  - preferSlave 则尝试连接到 URL 连接串中的备数据节点（如果有可用的话），否则连接到主数据节点。
  - any 则尝试连接 URL 连接串中的任何一个数据节点。
- heartbeatPeriod：是否开启对集群主节点进行心跳维护。该功能主要针对连接主备集群并且配置连接主节点的场景，当主节点宕机后，jdbc 会在其备机中查找新的主机，使下次建连的时候可以快速定位到新的主机。当且仅当设置了 targetServerType=master 且连接串的节点数大于 1，如果可以建连成功，则开启或者加入对集群主节点的心跳维护，该参数配置小于等于 0 时，没有心跳间隔按照不开启，或者当前连接节点不加入心跳维护，Integer 类型，默认值 5000，单位毫秒
- masterFailureHeartbeatTimeout: 在开启主节点心跳维护的场景下，当集群主节点宕机，设置对集群探寻主节点的超时阈值，在这个超时时间内

	还没有探测到主节点的存在，则认为集群一直没有主节点，不在对当前集群进行维护，这个时间应该包含主节点的 RTO 时间，Integer 类型，单位：毫秒，默认值：30000 ● priorityServers：Integer 类型。此值用于指定 url 上配置的前 n 个节点作为主集群被优先连接。默认值为 null。该值为数字，大于 0，且小于 url 上配置的 DN 数量。 例如：jdbc:gbase8c://host1:port1,host2:port2,host3:port3,host4:port4,/database?priorityServers=2。即表示 host1 与 host2 为主集群节点，host3 与 host4 为容灾集群节点。 ● forceTargetServerSlave：Boolean 类型。此值用于控制是否开启强制连接备机功能，并在集群发生主备切换时，禁止已存在的连接在升主备机上继续使用。默认值为 false，表示不开启强制连接备机功能。true，表示开启强制连接备机功能。 ● loadBalanceHosts：Boolean 类型。在默认模式下（禁用），默认顺序连接 URL 中指定的多个主机。如果启用，则使用洗牌算法从候选主机中随机选择一个主机建立连接。集中式环境下，如果使用此参数需要保证业务中没有写操作。 ● autoBalance：jdbc 可以通过 URL 中设置多个数据库节点，实现对主备集群的访问。通过设置负载均衡算法，jdbc 可以在建立连接时，依照特定规则将客户端与主备集群的连接依次建立在 URL 中配置的各个节点上，以此实现连接的负载均衡。默认值为 "false"，此时 jdbc 始终与 URL 中配置的同一个满足建连条件的节点建立连接。连接主备集群时，使用此参数需要保证业务中没有写操作，或者与 targetServerType=slave 搭配，限制客户端只连接备机。目前，jdbc 提供了 roundrobin、priority roundrobin、leastconn、shuffle 四种负载均衡模式，具体说明如下： ■ roundrobin：轮询模式，即与各候选节点轮流建立连接。取值："roundrobin"、"true"、"balance"。 ■ priority roundrobin：带优先级的轮询模式，优先对前 n 个候选节点做轮询建连，取值："proprity [n]"，n 为非负整数。 ■ leastconn：最小连接模式，对候选节点依照各节点的有效连接数做优先级排序，优先与连接数少的节点建立连接，该方法只会统计通过当前驱动在当前集群内使用 leastconn 模式建立的连接。取值："leastconn"。 ■ shuffle：随机模式，即随机选择候选节点建立连接，取值："shuffle"。loadBalanceHosts 设置为 "true" 等同于 autoBalance 设置为 "shuffle"。 如果 loadBalanceHosts 设置为 "true" 的同时，autoBalance 设置为以上

	四种负载均衡模式，优先使 autoBalance 生效；如果同时配置了 autoBalance 和 targetServerType，jdbc 会在满足 targetServerType 的前提下负载均衡；如果客户端使用多个指定了相同节点集合的 URL 分别通过同一驱动建立连接，jdbc 会将其视为同一集群上的连接，并整体进行负载均衡。 ● enableQuickAutoBalance：是否开启集群状态变化时，jdbc 快速负载均衡功能。该功能主要针对使用连接池连接主备集群并配置负载均衡的场景。当功能开启时，jdbc 会定时检测 URL 中配置的节点的状态，当检测到节点恢复时，jdbc 会筛选集群内其他节点上的空闲连接并关闭。连接池感知到缓存连接减少后，会基于 leastconn 模式在新恢复节点上创建连接，使集群重新负载均衡。该功能在筛选可关闭连接时，只会筛选由当前驱动创建的当前集群内的设置了 enableQuickAutoBalance="true" 的连接。该功能必须与最小连接模式配合使用：设置为 "true" 且 autoBalance 设置为 "leastconn" 时，功能开启。 取值范围：{"true", "false"}。 默认值："false"。 ● maxIdleTimeBeforeTerminal：jdbc 触发快速负载均衡时，可保留连接的最大空闲时长。jdbc 快速负载均衡功能筛选空闲连接时，会将处于 idle 状态并且持续时间大于等于 maxIdleTimeBeforeTerminal 的连接视为空闲连接，这些连接可能会被关闭。该参数只在 autoBalance 设置为 "leastconn" 且 enableQuickAutoBalance 设置为 "true" 时才生效。 单位：秒。 取值范围：[0, 9223372036854775) 内的整数。 默认值：30。 ● minReservedConPerCluster：jdbc 触发快速负载均衡时，一个集群内空闲连接的最小保留百分比。默认情况下，jdbc 触发快速负载均衡时，所有被筛选出来的空闲连接都会被关闭。设置该参数可以在触发快速负载均衡时，jdbc 会至少保留一个集群内空闲连接的 minReservedConPerCluster%。该参数针对集群生效，如果存在多个配置了相同节点的 URL 都设置了该参数，则取最小值。 取值范围：[0, 100] 内的整数。 默认值：0。 ● minReservedConPerDatanode：jdbc 触发快速负载均衡时，一个节点内空闲连接的最小保留百分比。设置该参数可以在触发快速负载均衡时，jdbc 会至少保留每个节点内空闲连接的 minReservedConPerDatanode%。如果 URL 串同时设置了 minReservedConPerDatanode 和

	minReservedConPerCluster，jdbc 会保证最终筛选出来的待关闭连接同时满足两个参数。该参数针对集群生效，如果存在多个配置了相同节点的 URL 都设置了该参数，则取最小值。 取值范围：[0, 100] 内的整数。 默认值：0。 ● options：String 类型。该参数用于在 jdbc 连接串中设置 guc 参数。 例如： jdbc:gbase8c://host:port/database?options=-c statement_timeout=10000。
user	数据库用户。
password	数据库用户的密码。

4.3 示例

```
//以下代码将获取数据库连接操作封装为一个接口，可通过给定用户名和密码来连接数据库。
public static Connection getConnect(String username, String passwd)
    {
        //驱动类。
        String driver = "cn.gbase8c.Driver";
        //数据库连接描述符。
        String sourceURL = "jdbc:gbase8c://10.10.0.13:8000/test";
        Connection conn = null;

        try
        {
            //加载驱动。
            Class.forName(driver);
        }
        catch( Exception e )
        {
            e.printStackTrace();
            return null;
```

```
        }

        try
        {
             //创建连接。
            conn = DriverManager.getConnection(sourceURL, username,
passwd);
            System.out.println("Connection succeed!");
        }
        catch(Exception e)
        {
            e.printStackTrace();
            return null;
        }

        return conn;
    };
// 以下代码将使用 Properties 对象作为参数建立连接
public static Connection getConnectUseProp(String username, String passwd)
    {
        //驱动类。
        String driver = "cn.gbase8c.Driver";
        //数据库连接描述符。
        String sourceURL = "jdbc:gbase8c://10.10.0.13:8000/test";
        Connection conn = null;
        Properties info = new Properties();

        try
        {
            //加载驱动。
            Class.forName(driver);
        }
        catch( Exception e )
        {
            e.printStackTrace();
            return null;
        }

        try
        {
             info.setProperty("user", username);
```

```
            info.setProperty("password", passwd);
            //创建连接。
            conn = DriverManager.getConnection(sourceURL, info);
            System.out.println("Connection succeed!");
        }
        catch(Exception e)
        {
            e.printStackTrace();
            return null;
        }

        return conn;
    };
```

5 执行 SQL 语句

5.1 执行普通 SQL 语句

应用程序通过执行 SQL 语句来操作数据库的数据（不用传递参数的语句），需要按以下步骤执行：

1. 调用 Connection 的 createStatement 方法创建语句对象。

```
Connection conn = DriverManager.getConnection("url","user","password");
Statement stmt = conn.createStatement();
```

2. 调用 Statement 的 executeUpdate 方法执行 SQL 语句。

```
int rc = stmt.executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER,
c_customer_name VARCHAR(32));");
```

3. 关闭语句对象。

```
stmt.close();
```

5.2 执行预编译 SQL 语句

预编译语句是只编译和优化一次，然后可以通过设置不同的参数值多次使用。由于已经预先编译好，后续使用会减少执行时间。因此，如果多次执行一条语句，请选择使用预编译语句。可以按以下步骤执行：

1. 调用 Connection 的 prepareStatement 方法创建预编译语句对象。

```
PreparedStatement pstmt = con.prepareStatement("UPDATE customer_t1 SET
c_customer_name = ? WHERE c_customer_sk = 1");
```

2. 调用 PreparedStatement 的 setShort 设置参数。

```
pstmt.setShort(1, (short)2);
```

3. 调用 PreparedStatement 的 executeUpdate 方法执行预编译 SQL 语句。

```
int rowcount = pstmt.executeUpdate();
```

4. 调用 PreparedStatement 的 close 方法关闭预编译语句对象。

```
pstmt.close();
```

5.3 调用存储过程

支持通过 JDBC 直接调用事先创建的存储过程，步骤如下：

1. 调用 Connection 的 prepareCall 方法创建调用语句对象。

```
Connection myConn = DriverManager.getConnection("url","user","password");
CallableStatement cstmt = myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
```

2. 调用 CallableStatement 的 setInt 方法设置参数。

```
cstmt.setInt(2, 50);
```

```
cstmt.setInt(1, 20);
cstmt.setInt(3, 90);
```

3. 调用 CallableStatement 的 registerOutParameter 方法注册输出参数。

```
cstmt.registerOutParameter(4, Types.INTEGER);  //注册 out 类型的参数，类型为整型。
```

4. 调用 CallableStatement 的 execute 执行方法调用。

```
cstmt.execute();
```

5. 调用 CallableStatement 的 getInt 方法获取输出参数。

```
int out = cstmt.getInt(4);  //获取 out 参数
示例：
//在数据库中已创建了如下存储过程，它带有 out 参数。
create or replace procedure testproc
(
    psv_in1 in integer,
    psv_in2 in integer,
    psv_inout in out integer
)
as
begin
    psv_inout := psv_in1 + psv_in2 + psv_inout;
end;
```

6. 调用 CallableStatement 的 close 方法关闭调用语句。

```
cstmt.close();
```

5.4 执行批处理

用一条预处理语句处理多条相似的数据，数据库只创建一次执行计划，节省了语句的编译和优化时间。可以按如下步骤执行：

1. 调用 Connection 的 prepareStatement 方法创建预编译语句对象。

```
Connection conn = DriverManager.getConnection("url","user","password");
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO customer_t1 VALUES
(?)");
```

2. 针对每条数据都要调用 setShort 设置参数，以及调用 addBatch 确认该条设置完毕。

```
pstmt.setShort(1, (short)2);
pstmt.addBatch();
```

3. 调用 PreparedStatement 的 executeBatch 方法执行批处理。

```
int[] rowcount = pstmt.executeBatch();
```

4. 调用 PreparedStatement 的 close 方法关闭预编译语句对象。

```
pstmt.close();
```

6 处理结果集

6.1 设置结果集类型

不同类型的结果集有各自的应用场景，应用程序需要根据实际情况选择相应的结果集类型。在执行 SQL 语句过程中，都需要先创建相应的语句对象，而部分创建语句对象的方法提供了设置结果集类型的功能。具体的参数设置如表 1 所示。涉及的 Connection 的方法如下：

```
//创建一个 Statement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。
createStatement(int resultSetType, int resultSetConcurrency);

//创建一个 PreparedStatement 对象，该对象将生成具有给定类型和并发性的 ResultSet
对象。
```

```
prepareStatement(String sql, int resultSetType, int
resultSetConcurrency);

//创建一个 CallableStatement 对象，该对象将生成具有给定类型和并发性的 ResultSet
对象。
prepareCall(String sql, int resultSetType, int resultSetConcurrency);
```

表 1 结果集类型

参数	描述
resultSetType	表示结果集的类型，具体有三种类型： ● ResultSet.TYPE_FORWARD_ONLY：ResultSet 只能向前移动。是缺省值。 ● ResultSet.TYPE_SCROLL_SENSITIVE：在修改后重新滚动到修改所在行，可以看到修改后的结果。 ● ResultSet.TYPE_SCROLL_INSENSITIVE：对可修改例程所做的编辑不进行显示。 说明： 结果集从数据库中读取了数据之后，即使类型是 ResultSet.TYPE_SCROLL_SENSITIVE，也不会看到由其他事务在这之后引起的改变。调用 ResultSet 的 refreshRow() 方法，可进入数据库并从其中取得当前游标所指记录的最新数据。
resultSetConcurrency	表示结果集的并发，具体有两种类型： ● ResultSet.CONCUR_READ_ONLY：如果不从结果集中的数据建立一个新的更新语句，不能对结果集中的数据进行更新。 ● ResultSet.CONCUR_UPDATEABLE：可改变的结果集。对于可滚动的结果集，可对结果集进行适当的改变。

6.2 在结果集中定位

ResultSet 对象具有指向其当前数据行的光标。最初，光标被置于第一行之前。next 方法将光标移动到下一行；因为该方法在 ResultSet 对象没有下一行时返回 false，所以可以在 while 循环中使用它来迭代结果集。但对于可滚动的结果集，JDBC 驱动程序提供更多的定位方法，使 ResultSet 指向特定的行。定位方法如表 2 所示。

表 2 在结果集中定位的方法

方法	描述
next()	把 ResultSet 向下移动一行。
previous()	把 ResultSet 向上移动一行。
beforeFirst()	把 ResultSet 定位到第一行之前。
afterLast()	把 ResultSet 定位到最后一行之后。
first()	把 ResultSet 定位到第一行。
last()	把 ResultSet 定位到最后一行。
absolute(int)	把 ResultSet 移动到参数指定的行数。
relative(int)	通过设置为 1 向前（设置为 1，相当于 next ()）或者向后（设置为 - 1，相当于 previous ()）移动参数指定的行。

6.3 获取结果集中光标的位置

对于可滚动的结果集，可能会调用定位方法来改变光标的位置。JDBC 驱动程序提供了获取结果集中光标所处位置的方法。获取光标位置的方法如表 3 所示

表 3 获取结果集光标的位置

方法	描述
isFirst()	是否在一行。
isLast()	是否在最后一行。
isBeforeFirst()	是否在第一行之前。
isAfterLast()	是否在最后一行之后。
getRow()	获取当前在第几行。

6.4 获取结果集中的数据

ResultSet 对象提供了丰富的方法，以获取结果集中的数据。获取数据常用的方法如表 4 所示，其他方法请参考 JDK 官方文档。

表 4 **ResultSet** 对象的常用方法

方法	描述
int getInt(int columnIndex)	按列标获取 int 型数据。
int getInt(String columnLabel)	按列名获取 int 型数据。
String getString(int columnIndex)	按列标获取 String 型数据。
String getString(String columnLabel)	按列名获取 String 型数据。
Date getDate(int columnIndex)	按列标获取 Date 型数据。
Date getDate(String columnLabel)	按列名获取 Date 型数据。

7 关闭连接

在使用数据库连接完成相应的数据操作后，需要关闭数据库连接。

关闭数据库连接可以直接调用其 close 方法。例如：

```
Connection conn = DriverManager.getConnection("url","user","password") ;
conn.close();
```

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

微信二维码