GaussDB
24.4.0
快速入门

文档版本 01

发布日期 2024-04-30

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

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

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

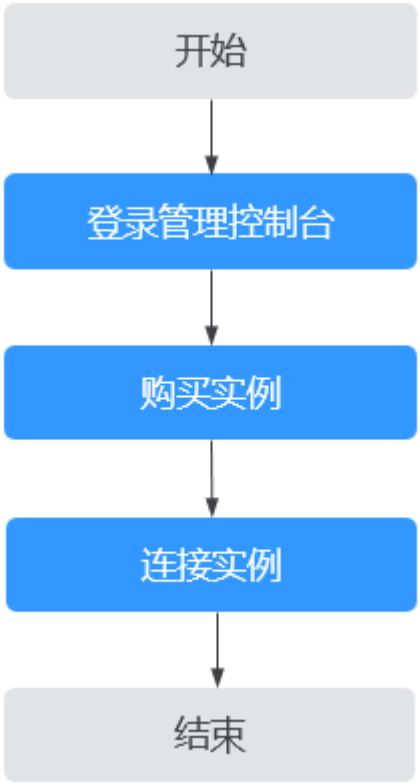
网址： https://www.huaweicloud.com/

目 录

1 操作指引

流程图

图 1-1 快速入门流程图



操作步骤

表 1-1 相关操作及参考手册

相关操作	参考章节
创建实例	购买实例

相关操作	参考章节
连接实例	根据业务场景选择连接方式： **使用客户端连接实例** **使用驱动连接实例**

2 购买实例

操作场景

本章将介绍在GaussDB的管理控制台购买实例。

目前，GaussDB支持“按需计费”和“包年/包月”计费方式购买。可以根据业务需要定制相应计算能力和存储空间的GaussDB实例。

前提条件

已注册华为账号并开通华为云。

操作步骤

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”，进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，单击“购买数据库实例”。

步骤5 在创建实例页面，选择计费模式，填写并选择实例相关信息后，单击“立即购买”。

图 2-1 计费模式和基本信息

表 2-1 基本信息

参数	描述
计费模式	云数据库GaussDB提供包年/包月、按需计费两种计费模式。 ● 包年/包月是一种预付费模式，即先付费再使用，按照订单的购买周期进行结算，因此在购买之前，必须确保账户余额充足。 ● 按需计费是一种后付费模式，即先使用再付费，即开即停，按照云数据库实际使用时长计费。以自然小时为单位整点计费，不足一小时按实际使用时长计费。
区域	租户当前所在区域，也可在页面左上角切换。 **说明** 不同区域内的产品内网不互通，且创建后不能更换，请谨慎选择。
实例名称	实例名称长度在4个到64个字符之间，必须以字母开头（区分大小写），可以包含字母、数字、中划线或下划线，不能包含其他特殊字符。
产品类型	云数据库GaussDB提供基础版和企业版两种类型。
数据库版本	GaussDB目前支持3.223、8.102版本。
实例类型	● 分布式版：分布式形态能够支撑较大的数据量，且提供横向扩展的能力，可以通过扩容的方式提高实例的数据容量和并发能力。 ● 主备版：适用于数据量较小，且长期来看数据不会大幅度增长，但是对数据的可靠性，以及业务的可用性有一定诉求的场景。

参数	描述
部署形态	● 分布式版 – 独立部署：将数据库组件部署在不同节点上。适用于可靠性、稳定性要求较高，实例规模较大的场景。仅企业版支持。 – 混合部署：采用一主两备三节点的部署模式，包含三个分片。仅基础版支持。 ● 主备版 – 高可用（1主2备）：采用一主两备三节点的部署模式，包含一个分片。 – 单副本：采用单节点的部署模式，仅包含一个CMS和一个DN组件。单副本实例仅支持2.2及以上版本实例。 – 1主1备1日志：采用一主一备一日志三节点的部署模式，包含一个分片，三个副本。 **注意** 单副本：由于部署在单台机器上，因此无法保障可用性（SLA）。 **说明** ● 混合部署形态有以下场景约束： – 仅支持3.223及以上版本。 – 不支持规格变更。 – 不支持磁盘自动扩容。 – 不支持包周期。
是否支持日志节点	仅分布式版实例支持该参数。 勾选后分布式实例将会支持一主一备一日志形态。
事务一致性	仅分布式版实例支持该参数。 ● 强一致性：应用更新数据时，用户都能查询到全部已经成功提交的数据，对性能有影响。 ● 最终一致性：应用更新数据时，不同用户查询到的数据可能不相同，有可能是更新后的值，也有可能是更新前的值，但经过一段时间后，查询到的数据均是更新后的值，该类型通常具有较高的性能。注意，不支持分布式事务强一致性读，不支持insert into select * from等依赖于查询结果的事务一致性，不支持拆分成多语句的写操作，不支持涉及多个节点执行的写操作。

参数	描述
切换策略	该参数仅针对特定用户开放，如需使用请联系客服人员申请。默认选择为数据高可靠，实例购买后如需修改切换策略，请参考**修改切换策略**。 ● 数据高可靠：对数据一致性要求高的系统推荐选择数据高可靠，在故障切换的时候优先保障数据一致性。 ● 业务高可用：对业务在线时间要求高的系统推荐使用业务高可用，在故障切换的时候优先保证数据库可用性。。 **说明** 在业务高可用场景下需要谨慎修改如下数据库参数，参数如何修改，可参见**修改实例参数**： – recovery_time_target：修改该参数会导致实例频繁进行强制切换，请在技术人员指导下进行修改。 – audit_system_object：修改该参数会导致丢失DDL审计日志，请在技术人员指导下进行修改。
副本集数量	仅分布式版实例可选。 每个分片下数据节点DN的数量，包括主DN和备DN。每个分片下3副本的部署方案，3副本就是1主2备的部署方式。
分片数量	仅分布式版实例可选。一个分片指的是一组DN副本集，分片内的DN数量与“副本集数量”参数有关，例如副本集数量为3，则一个分片就包含一主两备三个DN节点。
协调节点数量	仅分布式版实例可选。数据库中包含的协调节点（CN，Coordinator Node）数量。 协调节点的作用： ● 负责接收来自应用的访问请求，并向客户端返回执行结果。 ● 负责分解任务，并调度在各分片上并行执行。 **须知** 建议CN数量小于或等于分片数量的两倍。
可用区	可用区指在同一区域下，电力、网络隔离的物理区域，可用区之间内网互通，不同可用区之间物理隔离。 可用区只支持部署在一个或者三个可用区。
时区	由于世界各国家与地区经度不同，地方时也有所不同，因此会划分为不同的时区。时区可在购买实例时选择。

图 2-2 规格与存储

表 2-2 规格与存储

参数	描述
性能规格	实例的CPU和内存。不同性能规格对应不同连接数。 关于性能规格详情，请参见**数据库实例规格**。
专属云类型	M6。 **说明** 只有购买了专属计算集群（Dedicated Computing Cluster）的用户才有此选项。
资源类型	云硬盘。 **说明** 只有购买了专属计算集群（Dedicated Computing Cluster）的用户才有此选项。
存储类型	实例的存储类型决定实例的读写速度。最大吞吐量越高，读写速度越快。 GaussDB支持“超高IO”存储类型，最大吞吐量为350MB/s。
存储空间	申请的存储空间会有必要的文件系统开销，这些开销包括索引节点和保留块，以及数据库运行必需的空间。实例购买成功后可进行扩容，具体请参见**磁盘扩容**。 **说明** 创建实例时，磁盘空间支持单分片起配值为40GB，步长为4GB。
赠送备份空间	免费赠送存储空间等量的备份空间，超出免费备份空间部分采用按需计费方式。
磁盘加密	• 不加密：未开启加密功能。 • 加密：提高数据安全性，对性能有一定影响。 密钥名称：选择加密方式后，需要选择或创建密钥，该密钥是最终租户密钥。

图 2-3 网络和数据库配置

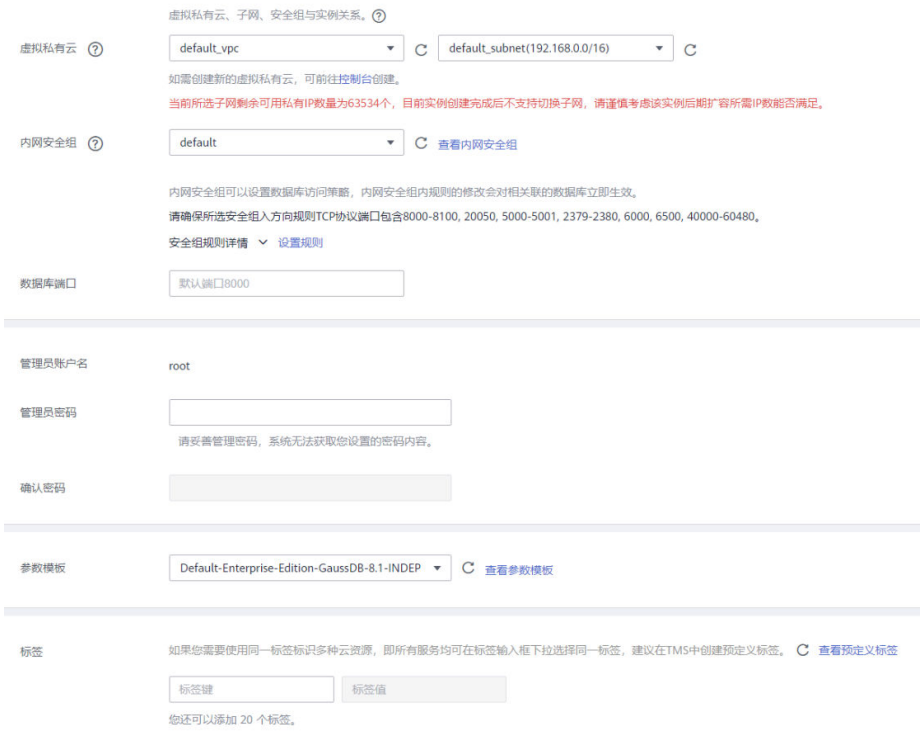


表 2-3 网络

参数	描述
虚拟私有云	GaussDB实例所在的虚拟专用网络，可以对不同业务进行网络隔离。需要创建或选择所需的虚拟私有云。如何创建虚拟私有云，请参见**创建虚拟私有云基本信息及默认子网**。 如果没有可选的虚拟私有云，GaussDB默认为用户分配资源。 **须知** GaussDB实例创建完成后不支持切换虚拟私有云，请谨慎选择所属虚拟私有云。
子网	通过子网提供与其他网络隔离的、可以独享的网络资源，以提高网络安全性。子网在可用区内才会有效，创建GaussDB实例的子网默认开启DHCP功能，不可关闭。创建实例时GaussDB会自动配置内网地址。 **说明** • 默认子网支持的IP数为256，分布式实例最大规模需要IP数为1286，推荐选用2048规模的子网。

参数	描述
内网安全组	控制网络出/入及端口的访问，默认添加了GaussDB实例所属的内网安全组访问。 ● 购买分布式版实例时，如果需要修改内网安全组，请确保入方向规则TCP协议端口包含：40000-60480,20050,5000-5001,2379-2380,6000,6500，<database port> - (<database port> + 100)。（例如设置的数据库端口为8000，则安全组中需要包含8000-8100）。 ● 购买主备版实例时，如果需要修改内网安全组，请确保入方向规则TCP协议端口包含：20050，5000-5001，2379-2380，6000，6500，<database port> - (<database port> + 100)。（例如设置的数据库端口为8000，则安全组中需要包含8000-8100）。 内网安全组限制实例的安全访问规则，加强GaussDB与其他服务间的安全访问。请确保所选取的内网安全组允许客户端访问数据库实例。如果创建时不需要指定安全组，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，提交开通白名单的申请。 如果没有可选的内网安全组，GaussDB默认为您分配内网安全组资源。
数据库端口	数据库对外开放的端口，默认为8000，可选范围为：1024-39998。如下端口被系统占用，不可设置：2378,2379,2380,4999,5000,5999,6000,6001,8097,8098,12016,12017,20049,20050,21731,21732,32122,32123,32124。
单浮动IP策略	是否开启单浮动IP策略，如果开启，实例将只有一个浮动IP绑定主节点，如果发生主备倒换，浮动IP不会发生变化；如果不开启，每个节点都会绑定一个浮动IP，如果发生主备倒换，浮动IP会发生变化。 单浮动IP约束限制如下： ● 仅支持3.206及以上的主备版实例。 ● 仅创建时支持修改，创建后不支持切换单浮动IP策略。

表 2-4 数据库配置

参数	描述
管理员账户名	数据库的登录名称默认为root。

参数	描述
管理员密码	需要输入高强度密码并定期修改，以提高安全性，防止出现密码被暴力破解等安全风险。 **须知** 设置的密码需满足以下几个条件： • 8~32个字符。 • 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#%^*-_=+?,）四类字符中的三类字符。 请妥善保管密码，因为系统将无法获取密码信息。 实例创建成功后，如需重置密码，请参见**重置管理员密码**。
确认密码	必须和管理员密码相同。

表 2-5 参数模板

参数	描述
参数模板	数据库参数模板就像是数据库引擎配置值的容器，参数模板中的参数可应用于一个或多个相同类型的数据库实例。实例创建成功后，参数模板可进行修改。 用户可以在实例创建完成之后根据业务需要进行调整。具体请参见**编辑参数模板**。
企业项目	对于已成功关联企业项目的用户，仅需在“企业项目”下拉框中选择目标项目。 如果需要自定义企业项目，请前往企业项目管理服务进行创建。关于如何创建项目，详见**《企业项目管理用户指南》**。

表 2-6 标签

参数	描述
标签	可选配置，对数据库的标识。使用标签可以方便识别和管理用户拥有的数据库服务资源。每个实例最多支持20个标签配额。 如用户的组织已经设定GaussDB的相关标签策略，则需按照标签策略规则为实例添加标签。标签如果不符合标签策略的规则，则可能会导致实例创建失败，请联系组织管理员了解标签策略详情。

如果对价格有疑问，可以单击页面底部“配置费用”处的“了解计费详情”来了解产品价格。

说明

GaussDB的性能，取决于用户申请GaussDB时所选择的配置。可供用户选择的硬件配置项为性能规格、存储类型以及存储空间。

步骤6 确认详细信息。

对于按需计费的实例，进行规格确认。

- 如果需要重新选择实例规格，单击“上一步”，回到上个页面修改实例信息。
- 如果规格确认无误，单击“提交”，完成购买实例的申请。

对于包年/包月模式的实例，进行订单确认。

- 如果需要重新选择实例规格，单击“上一步”，回到上个页面修改实例信息。
- 如果订单确认无误，单击“去支付”，进入“付款”页面。在付款页面，选择支付方式并完成付费。

步骤7 GaussDB实例创建任务提交后，用户可以在“实例管理”页面对其进行查看和管理。

- 创建GaussDB实例过程中，状态显示为“创建中”。
- 在实例列表的右上角，单击  刷新列表，可查看到创建完成的实例状态显示为“正常”。
- 创建完成后会自动进行一次全量备份，用于记录实例的初始状态。
- 数据库端口默认为8000，支持创建时设置，后期可修改。

----结束

相关操作

- **通过调用API创建数据库实例**
- **修改实例参数**

3 使用客户端连接实例

3.1 实例连接方式介绍

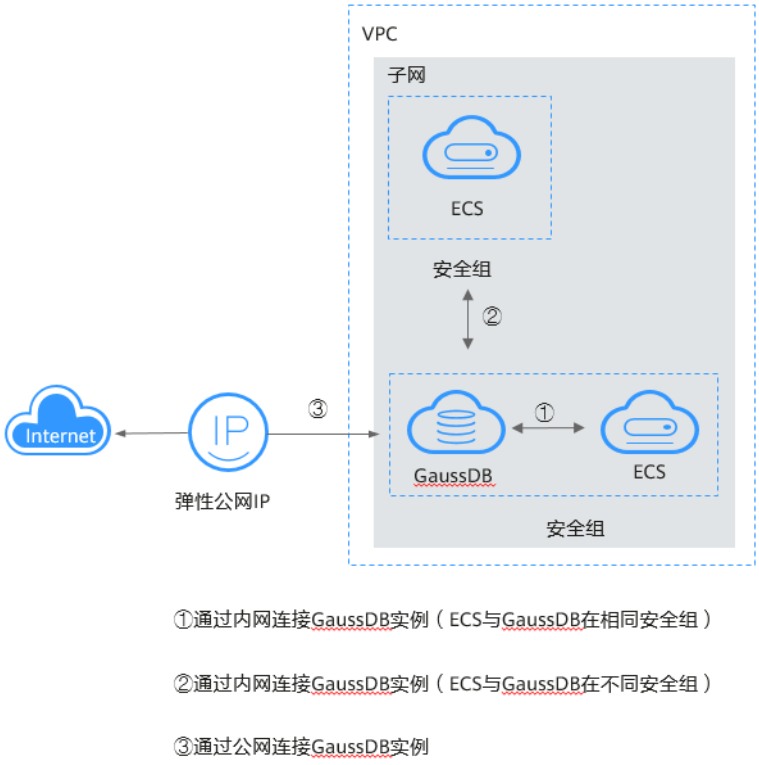
GaussDB提供使用gsql、DBeaver、Navicat和数据管理服务（Data Admin Service，简称DAS）连接实例的方式。

表 3-1 GaussDB 连接方式

连接方式	IP地址	使用场景	说明
DAS连接	无需使用IP地址	通过华为云数据管理服务（Data Admin Service，简称DAS）这款可视化的专业数据库管理工具，可获得执行SQL、高级数据库管理、智能化运维等功能，做到易用、安全、智能地管理数据库。GaussDB默认开通DAS连接权限。	易用、安全、高级、智能。
gsql连接	内网IP地址/弹性公网IP	gsql是GaussDB自带的客户端工具。使用gsql连接数据库，可以交互式地输入、编辑、执行SQL语句。	为了获得更快的传输速率和更高的安全性，建议将应用迁移到与GaussDB实例在同一子网，使用内网连接。
DBeaver连接	弹性公网IP	通过DBeaver这款可视化数据库管理工具可以查看数据库结构、执行SQL查询和脚本、浏览和导出数据、处理BLOB/CLOB数据以及修改数据库结构等。	开源、易用。
Navicat连接	弹性公网IP	Navicat数据库管理工具提供轻松、便捷的可视化数据查看和编辑功能。例如：数据增删改查、SQL或脚本处理、函数、数据生成等功能。	易用、稳定。

其中，通过gsql连接方式如图3-1所示。

图 3-1 gsql 连接



3.2 通过数据管理服务 DAS 连接实例（推荐）

操作场景

通过华为云数据管理服务（Data Admin Service，简称DAS）这款可视化的专业数据库管理工具，可获得执行SQL、高级数据库管理、智能化运维等功能，做到易用、安全、智能的管理数据库。

操作步骤

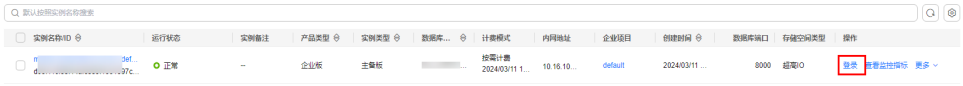
步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”，进入云数据库GaussDB信息页面。

步骤4 在“实例管理”页面，选择需要登录的目标数据库，单击操作列表中的“登录”，进入数据管理服务数据库登录界面。

图 3-2 登录实例



可以在“实例管理”页面，单击目标实例名称，进入实例的“基本信息”页面，在页面右上角，单击“登录”，进入数据管理服务实例登录界面。

图 3-3 登录实例

步骤5 正确输入数据库用户名和密码，单击“登录”，即可登录到数据库。

表 3-2 参数说明

参数	描述
登录用户名	GaussDB数据库账号，默认管理员账号为root。
数据库名称	需要连接的数据库名，默认管理数据库是postgres。
密码	数据库用户的密码。
定时采集	建议打开定时采集开关，若不开启，DAS只能实时的从数据库获取结构定义数据，将会影响数据库实时性能。 采集时间不支持自定义，启用“定时采集”后，系统会在UTC时间每天20:00点自动采集数据。如果用户本地不是UTC时间，请根据本地时区转换成对应时间即可。即使未启用“定时采集”，也可以根据需要随时单击“立即采集”采集数据。
SQL执行记录	建议开启SQL执行记录，开启后，便于在“SQL操作>SQL执行记录”查看，并可再次执行，无需重复输入。

登录后，通过DAS管理数据库的具体操作可参见**GaussDB 管理**。

----结束

后续操作

登录实例后，可以创建数据库，进行数据迁移等操作，具体请参见：

- **通过调用API创建GaussDB数据库**
- **迁移数据库**

3.3 通过 gsql 连接实例（Linux 方式）

本章介绍在管理控制台购买GaussDB实例后，如何使用gsql客户端连接GaussDB实例。

- **步骤一：购买ECS**

- **步骤二：查询需要连接实例的IP地址和端口号**
- **步骤三：测试连通性**
- **步骤四：获取驱动包**
- **步骤五：连接数据库**
  - **非SSL连接**
  - **SSL连接**

购买 ECS

GaussDB提供gsql工具帮助用户在命令行下连接数据库，用户需要提前创建一台弹性云服务器用于安装gsql工具。

1. **登录管理控制台**，查看是否有弹性云服务器。
   - 有弹性云服务器，执行**3**。
   - 无弹性云服务器，执行**2**。

   图 3-4 ECS 实例

2. 购买弹性云服务器时，选择Euler操作系统。

   购买Linux弹性云服务器请参考《弹性云服务器快速入门》中“**购买弹性云服务器**”章节。

3. 在ECS实例基本信息页，查看ECS实例的区域和VPC。

   图 3-5 ECS 基本信息

须知

操作系统需要选择Euler操作系统。gsql支持的操作系统版本如下：

X86：EulerOS V2.0SP5，Kylin V10 SP2 。

鲲鹏服务器：EulerOS V2.0SP8，Kylin V10 SP1 。

4. 在GaussDB实例基本信息页，查看GaussDB实例的区域和VPC。

图 3-6 GaussDB 基本信息

5. 确认ECS实例与GaussDB实例是否处于同一区域、同一VPC。
   - ECS与GaussDB实例在同一区域、同一VPC，则通过内网连接实例，内网地址如何获取，请参考**查询需要连接实例的IP地址**。
   - ECS与GaussDB实例不在同一VPC，则通过公网连接实例，公网地址如何获取，请参考**查询需要连接实例的IP地址**。请确保ECS和GaussDB实例都要有弹性IP。
     - ECS如何绑定弹性公网IP，请参考**绑定弹性公网IP**。
     - GaussDB实例如何绑定弹性公网IP，请参考**绑定弹性公网IP**。

查询需要连接实例的 IP 地址和端口号

1. **登录管理控制台**。
2. 单击管理控制台左上角的，选择区域和项目。
3. 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”，进入云数据库GaussDB信息页面。
4. 在“实例管理”页面，选择指定的实例，单击实例名称，进入实例基本信息页面。
5. 在“连接信息”模块处，查看IP地址和端口号。
   - ECS实例与GaussDB实例在同一VPC，请获取内网地址和数据库端口号。
   - ECS实例与GaussDB实例不在同一VPC，请获取弹性公网IP和数据库端口号。

测试连通性

1. 登录ECS实例，请参见《弹性云服务器用户指南》中“**Linux弹性云服务器远程登录（VNC方式）**”。
2. 在ECS上测试是否可以正常连接到GaussDB实例地址的端口，连接地址和端口通过**查询需要连接实例的IP地址和端口号**获取。

   telnet *IP地址 端口*

   示例：

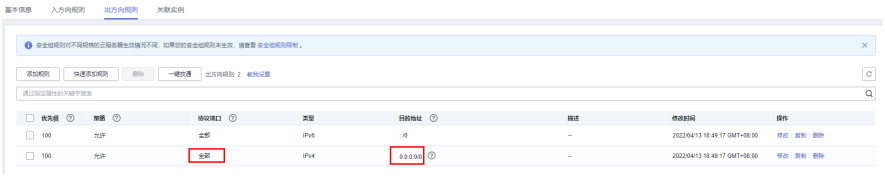
   telnet 192.168.0.16 8000

   说明

   如果提示command not found，请根据ECS使用的操作系统不同，自行安装telnet工具。

   - 如果可以通信，说明网络正常。
   - 如果无法通信，请检查安全组规则。
     - 查看ECS的安全组的出方向规则，如果目的地址不为“0.0.0.0/0”且协议端口不为“全部”，需要将GaussDB实例的IP地址和端口添加到出方向规则。
       - ECS实例与GaussDB实例在同一VPC，将GaussDB实例的内网IP地址和端口添加到出方向规则。
       - ECS实例与GaussDB实例不在同一VPC，将GaussDB实例的公网IP地址和端口添加到出方向规则。

     图 3-7 ECS 的安全组

     - 查看GaussDB的安全组的入方向规则，如果源地址不为“0.0.0.0/0”且协议端口不为“全部”，需要将ECS实例IP地址和端口添加到入方向规则。
       - ECS实例与GaussDB实例在同一VPC，需要将ECS实例的私有IP地址和端口添加到入方向规则。
       - ECS实例与GaussDB实例不在同一VPC，将ECS实例的弹性公网IP地址和端口添加到入方向规则。

     具体操作请参见**设置安全组规则**。

     图 3-8 GaussDB 的安全组

获取驱动包

根据不同版本的实例，下载不同版本的发布包，如表3-3所示。

表 3-3 驱动包下载列表

版本	下载地址
3.x	驱动包 驱动包校验包
2.x	驱动包 驱动包校验包

为了防止软件包在传递过程或存储期间被恶意篡改，下载软件包时需下载对应的校验包对软件包进行校验，校验方法如下：

1. 上传软件包和软件包校验包到虚拟机（Linux操作系统）的同一目录下。
2. 执行如下命令，校验软件包完整性。

   cat *GaussDB_driver.zip.sha256* **| sha256sum --check**

   如果回显OK，则校验通过。

   ```
   GaussDB_driver.zip: OK
   ```

连接数据库

- **非SSL连接**

  a. 以root用户登录申请的弹性云服务器。

  b. 上传客户端工具包并配置gsql的执行环境变量。

  i. 执行以下命令创建“/tmp/tools”目录，用以存放客户端工具包。

  ```
  mkdir /tmp/tools
  ```

  ii. 参考获取驱动包，下载对应版本“GaussDB_driver.zip”驱动包，并将其上传到申请的弹性云服务器“/tmp/tools”路径下。

  iii. 执行以下命令解压“GaussDB_driver.zip”驱动包。

  ```
  cd /tmp/tools
  unzip GaussDB_driver.zip
  ```

  iv. 执行以下命令将解压出来的“GaussDB-Kernel_V***R***C**_EULER_64bit-Gsql.tar.gz”客户端工具包拷贝到“/tmp/tools”路径下。

  说明

  客户端工具包相对位置为解压后位置，请根据实际情况填写。此处以主备版实例Euler2.5_x86_64系统的gsql工具包为例。

  ```
  cd /tmp/tools/GaussDB_driver/Centralized/Euler2.5_X86_64/
  cp GaussDB-Kernel_V***R***C**_EULER_64bit-Gsql.tar.g /tmp/tools
  ```

  v. 执行以下命令解压文件。

  ```
  cd /tmp/tools
  tar -zxvf GaussDB-Kernel_V***R***C**_EULER_64bit-Gsql.tar.g
  ```

vi. 配置环境变量。

执行以下命令打开“~/.bashrc”文件。

```
vim ~/.bashrc
```

按“G”将光标移至最后一行，按“i”进入INSERT模式，输入如下内容后，单击“ESC”退出INSERT模式，输入“:wq”命令保存并退出。

```
export PATH=/tmp/tools/bin:$PATH
export LD_LIBRARY_PATH=/tmp/tools/lib:$LD_LIBRARY_PATH
```

执行以下命令使环境变量配置永久生效。

```
source ~/.bashrc
```

c. 执行如下指令，根据提示输入密码，连接数据库。

数据库创建成功后，会默认生成名称为postgres的数据库，此处以postgres库为例。

```
gsql -d postgres -h 10.0.0.0 -U root -p 8000
Password for user root:
```

postgres为需要连接的数据库名称，10.0.0.0为实例的IP地址，通过**查询需要连接实例的IP地址**获取，root为登录数据库的用户名，8000为数据库的端口号，通过**查询需要连接实例的端口号**获取。

- **SSL连接**

a. **登录管理控制台**。

b. 单击管理控制台左上角的，选择区域和项目。

c. 在页面左上角单击，选择“数据库 > 云数据库 GaussDB”，进入云数据库 GaussDB信息页面。

d. 在“实例管理”页面，单击实例名称进入“基本信息”页面，单击“数据库信息”模块“SSL”处的，下载根证书或捆绑包。

e. 将根证书上传至需连接GaussDB实例的弹性云服务器，或保存到可访问数据库实例的设备。

将根证书导入弹性云服务器Linux操作系统，请参见**将根证书导入Windows/Linux操作系统**。

f. 连接GaussDB实例。

以Linux系统为例，在弹性云服务器设置环境变量，执行如下命令。

```
export PGSSLMODE=<sslmode>
export PGSSLROOTCERT=<ca-file-directory>
```

gsql -h *<host>* **-p** *<port>* **-d** *<database>* **-U** *<user>*

表 3-4 参数说明

参数	说明
<host>	主机IP，在“实例管理”页面单击实例名称，进入“基本信息”页面。“连接信息”模块的“内网地址”（通过弹性云服务器访问）。
<port>	端口，默认8000，当前端口，即在“实例管理”页面单击实例名称，进入“基本信息”页面，“连接信息”模块的“数据库端口”。
<database>	需要连接的数据库名，默认管理数据库是postgres。
<user>	用户名，即GaussDB数据库账号，默认管理员账号为root。
<ca-file-directory>	SSL连接CA证书路径。
<sslmode>	SSL连接模式，设置为“verify-ca”，通过检查证书链（Certificate Chain，以下简称CA）来验证服务是否可信任。

在弹性云服务器设置环境变量，使用root用户SSL连接postgres数据库实例，具体示例如下：

```
export PGSSLMODE="verify-ca"
export PGSSLROOTCERT="/home/Ruby/ca.pem"
```

gsql -d *postgres* **-h** *10.0.0.0* **-U** *root* **-p** *8000*

```
Password for user root:
```

g. 登录数据库后，出现如下信息，表示通过SSL连接成功。

```
SSL connection (cipher: DHE-RSA-AES256-GCM-SHA384, bits: 256)
```

相关链接

有关gsql的命令参考和更多信息，请参见《工具参考》。

3.4 通过 DBeaver 连接实例（Windows 方式）

DBeaver是一个通用的数据库客户端，可以通过配置不同驱动连接各种不同的数据库。本章介绍如何通过DBeaver连接GaussDB实例。

步骤一：获取驱动包

1. 获取驱动包和驱动包校验包。

   根据不同版本的实例，下载对应版本的驱动包和驱动包校验包到本地任意目录，如表3-5所示。

表 3-5 驱动包下载列表

版本	下载地址
3.x	**驱动包** **驱动包校验包**
2.x	**驱动包** **驱动包校验包**

2. 校验驱动包。

   为了防止驱动包在传递过程或存储期间被恶意篡改，需要对驱动包进行校验，校验方法如下：

   a. 使用快捷键“Win+R”打开“运行”窗口。 在“打开”栏，输入“cmd”，按“Enter”回车，打开命令行页面。

   b. 执行以下命令，获取驱动包的Hash值。

      certutil -hashfile *{驱动包本地目录}\{驱动包名}* sha256

      - {*驱动包本地目录*}：请根据实际下载目录进行替换。例如：C:\Users
      - {*驱动包*名}：请根据实际下载的驱动包名进行替换。例如：GaussDB_driver.zip

      示例：certutil -hashfile C:\Users\GaussDB_driver.zip sha256

   c. 将**2.b**获取到的Hash值和**1**中获取到的驱动包校验包的Hash值进行比较。

      - 若一致则通过校验。
      - 若不一致，请重新下载驱动包，重复**2.a**~**2.c**进行校验。

3. 解压驱动包。

   将**1**中获取到的驱动包解压到本地， 找到gsjdbc4.jar包，放在本地任意目录下。

步骤二：获取 DBeaver 客户端安装包

DBeaver官网提供了针对不同操作系统的客户端安装包，单击**此处**访问DBeaver官网下载系统对应的DBeaver客户端安装包并完成安装 。

步骤三：创建新驱动

1. 打开DBeaver客户端。
2. 单击“数据库 > 驱动管理”，打开驱动管理器界面。
3. 单击“新建”，打开创建新驱动界面。
4. 在“设置”页正确输入驱动名称、类名、URL模板、默认端口、Default Database、Default User，选择驱动类型，单击“确定”创建驱动。

表 3-6 参数说明

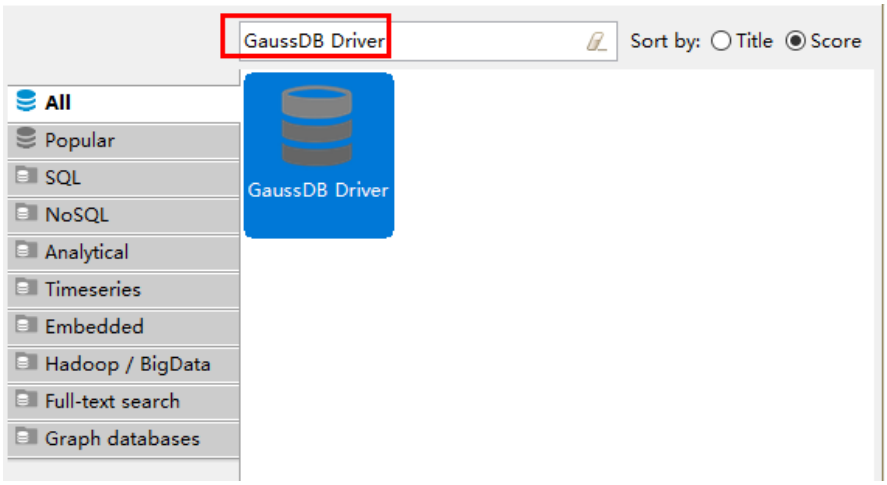
参数	描述
驱动名称	命名为便于识别的名称，例如GaussDB Driver。
驱动类型	驱动类型选择Generic。
类名	类名为org.postgresql.Driver。
URL模板	URL模板为jdbc:postgresql://{host}[:{port}]/[{database}]。
默认端口	需要连接的数据库端口。创建实例时自定义的端口，GaussDB实例的默认端口为8000。
Default Database	需要连接的数据库名。实例创建成功后，会默认生成名称为postgres的数据库。
Default User	需要访问GaussDB实例的账号名称。默认root。

5. 在“库”页中，单击添加文件，添加3中的gsjdbc4.jar。
6. 添加后驱动类为空，需要单击“找到类”。识别出来的驱动类，需要与“设置”页的“类名”一致。
7. 单击“确定”，驱动设置完成。

步骤四：连接数据库

1. 在DBeaver客户端单击  ，打开创建连接界面。
2. 搜索步骤三中创建的驱动，选中驱动，单击“下一步”。

图 3-9 选择驱动



3. 输入主机IP地址，端口，数据库名，用户名和密码。

表 3-7 参数说明

参数	描述
主机	目标实例的内网地址。查看目标实例的内网地址及端口信息的步骤如下： 1. 登录云数据库GaussDB的管理控制台。 2. 选择目标实例所在区域。 3. 单击目标实例名称，进入“基本信息”页面。 4. 在“连接信息”模块，查看“弹性公网IP”信息。如果未绑定弹性公网IP，则需要为数据库实例绑定弹性公网IP。具体操作请参考**绑定弹性公网IP**。
端口	需要连接的数据库端口。创建实例时自定义的端口，GaussDB实例的默认端口为8000。
数据库/模式	需要连接的数据库名。实例创建成功后，会默认生成名称为postgres的数据库。
用户名	需要访问GaussDB实例的账号名称。默认root。
密码	要访问GaussDB实例的账号所对应的密码。

4. 单击“测试链接”。若弹框中显示“已连接”，则说明可正常连接，单击“确定”。
5. 单击“完成”，即可连接到数据库。在“数据库导航”栏可查看到连接的数据库信息。

3.5 通过 Navicat 连接实例（Windows 方式）

最新 Navicat Premium 16.2.8 Windows版已支持对GaussDB的管理和开发。本章介绍如何通过Navicat连接GaussDB实例。

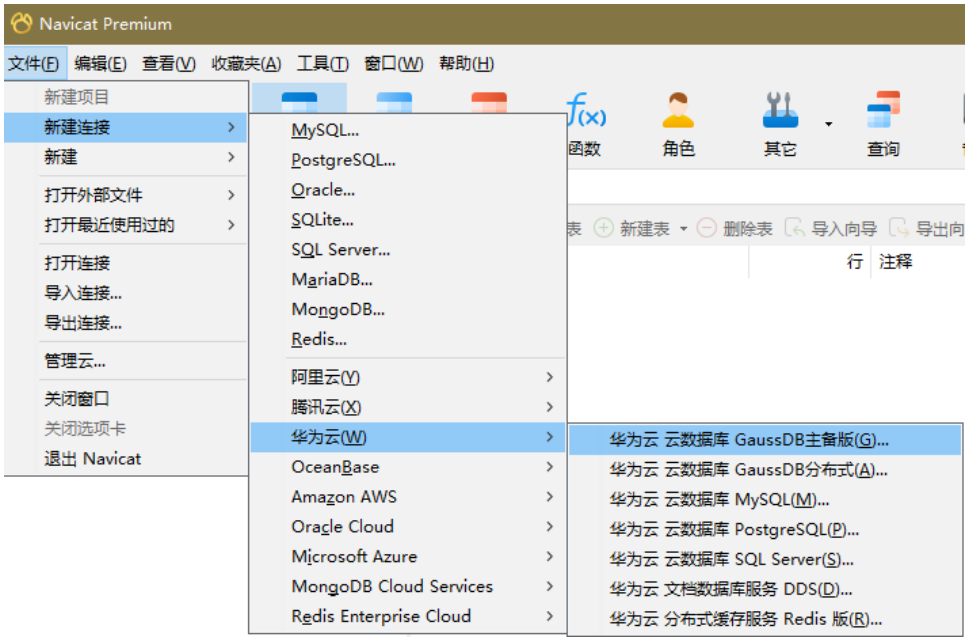
前提条件

单击**此处**下载或购买Navicat Premium并完成安装。

操作步骤

步骤1 打开Navicat Premium，单击文件 > 新建连接 > 华为云 > 华为云 云数据库 GaussDB 主备版/华为云 云数据库 GaussDB分布式。

图 3-10 新建连接



步骤2 在“新建连接”界面，输入正确的连接名、主机，端口，初始数据库，用户名和密码。

图 3-11 主备版连接信息

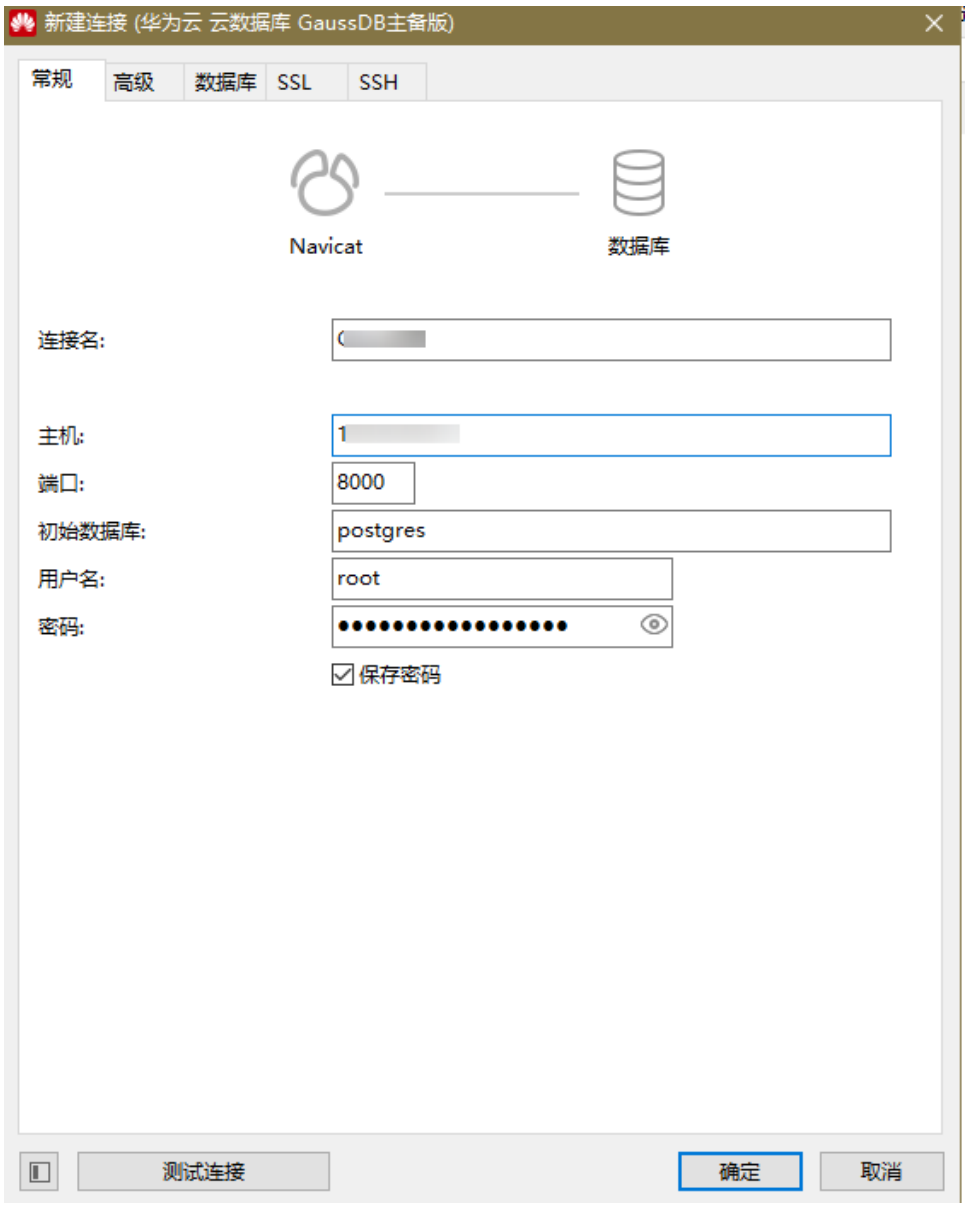


图 3-12 分布式连接信息

表 3-8 参数说明

参数	描述
连接名	命名为便于识别的名称。

参数	描述
主机	目标实例的内网地址。查看目标实例的内网地址及端口信息的步骤如下： 1. 登录云数据库GaussDB的管理控制台。 2. 选择目标实例所在区域。 3. 单击目标实例名称，进入“基本信息”页面。 4. 在“连接信息”模块，查看“弹性公网IP”信息。如果未绑定弹性公网IP，则需要为数据库实例绑定弹性公网IP。具体操作请参考**绑定弹性公网IP**。
端口	需要连接的数据库端口。创建实例时自定义的端口，GaussDB实例的默认端口为8000。
初始数据库	需要连接的数据库名。实例创建成功后，会默认生成名称为postgres的数据库。
用户名	需要访问GaussDB实例的账号名称。默认root。
密码	要访问GaussDB实例的账号所对应的密码。

步骤3 单击“测试连接”，若弹框中显示“连接成功”，则说明可正常连接，单击“确定”关闭弹框。

步骤4 单击“确定”，即可建立数据库连接，此时连接默认为关闭状态，需要手动打开连接。

步骤5 在连接名处右键，单击“打开连接”。

步骤6 在数据库名处右键，单击“打开数据库”。

----结束

4 使用驱动连接实例

本章主要介绍如何使用JDBC、ODBC等驱动连接实例。

4.1 分布式

4.1.1 开发规范

如果用户在APP的开发过程中，使用了连接池机制，那么需要遵循如下规范：

- 如果在连接中设置了GUC参数，那么在将连接归还连接池之前，必须使用“SET SESSION AUTHORIZATION DEFAULT;RESET ALL;”将连接的状态清空。
- 如果使用了临时表，那么在将连接归还连接池之前，必须将临时表删除。

否则，连接池里面的连接就是有状态的，会对用户后续使用连接池进行操作的正确性带来影响。

应用程序开发驱动兼容性说明如表4-1所示：

表 4-1 兼容性说明

驱动	兼容性说明
JDBC、Go、ODBC、libpq、Psycopg、ecpg	驱动前向兼容数据库，若需使用驱动与数据库同步增加的新特性，须升级数据库。

须知

- 原则上，兼容性参数应在创建数据库后就设置，不应在使用过程中来回切换。
- 涉及使用以下场景的特性需要配合将JDBC驱动升级到503.1.0及以上的配套版本：
  - 开启s2兼容性参数，设置sessiontimezone的合法性校验。

在多线程环境下使用驱动：

JDBC驱动程序线程不是安全的，无法保证连接上的方法同步。由调用者来同步对驱动程序的调用。

4.1.2 使用 JDBC 连接数据库

JDBC（Java Database Connectivity，java数据库连接）是用于执行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。GaussDB库提供了对JDBC 4.2特性的支持，需要使用JDK1.8版本编译程序代码，不支持JDBC桥接ODBC方式。

前提条件

计算机已安装Java JDK8版本。

JDBC 包

包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Jdbc.tar.gz。

解压后JDBC的驱动jar包：

- gaussdbjdbc.jar：主类名为“com.huawei.gaussdb.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，推荐使用此驱动包。本章的Java代码示例默认使用gaussdbjdbc.jar包。
- gscejdbc.jar：主类名为“com.huawei.gaussdb.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，此驱动包打包了密态数据库需要加载的加解密相关的依赖库，密态场景推荐使用此驱动包。目前仅支持EulerOS操作系统。
- gaussdbjdbc-JRE7.jar：主类名为“com.huawei.gaussdb.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，在JDK1.7环境使用gaussdbjdbc-JRE7.jar包。

⚠ 注意

- 使用gscejdbc.jar驱动包时，需要先设置环境变量LD_LIBRARY_PATH。具体使用方式见《特性指南》中“设置密态等值查询 > 使用JDBC操作密态数据库”章节。
- 在JDK1.8环境中使用gaussdbjdbc.jar，不推荐使用gaussdbjdbc-JRE7.jar。
- 其他JDBC的jar包介绍请参考《开发指南》中“应用程序开发教程 > JDBC兼容性包”章节。

驱动类

在创建数据库连接之前，需要加载数据库驱动类“com.huawei.gaussdb.jdbc.Driver”。

说明

1. 由于GaussDB在JDBC的使用上与PG的使用方法保持兼容，所以同时在同一进程内使用两个JDBC驱动的时候，可能会造成类名冲突。
2. 本版本JDBC不再支持IAM认证功能。
3. GaussDB JDBC驱动主要做了以下特性的增强：
    1. 支持SHA256加密方式登录。
    2. 支持对接实现sf4j接口的第三方日志框架。
    3. 支持连接级别的分布式负载均衡。
    4. 支持容灾切换。

环境类

客户端需配置JDK1.8。JDK是跨平台的，支持Windows、Linux等多种平台，下面以Windows为例，介绍JDK配置流程：

步骤1 DOS窗口（windows下的命令提示符）输入“java -version”，查看JDK版本，确认为JDK1.8版本。如果未安装JDK，请下载安装包并安装。

步骤2 右键单击“我的电脑”，选择“属性”。

步骤3 在“系统”页面左侧导航栏单击“高级系统设置”。

步骤4 在“系统属性”页面，“高级”页签上单击“环境变量”。

步骤5 在“环境变量”页面上，“系统变量”区域单击“新建”或“编辑”，设置如下变量名和变量值。变量说明如表4-2所示。

表 4-2 变量说明

变量名	操作	变量值
JAVA_HOME	• 若存在，则单击“编辑”。 • 若不存在，则单击“新建”。	JAVA的安装目录。 例如：C:\Program Files\Java\jdk1.8.0_131。
Path	单击“编辑”。	• 若配置了JAVA_HOME，则在变量值的最前面加上：%JAVA_HOME%\bin。 • 若未配置JAVA_HOME，则在变量值的最前面加上JAVA安装的全路径：C:\Program Files\Java\jdk1.8.0_131\bin。
CLASSPATH	单击“新建”。	%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar。

步骤6 单击“确定”，并依次关闭各窗口。

----结束

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。

加载驱动有两种方法：

- 在代码中创建连接之前任意位置隐含装载：
  Class.forName("com.huawei.gaussdb.jdbc.Driver")
- 在JVM启动时参数传递：java -Djdbc.drivers=com.huawei.gaussdb.jdbc.Driver jdbctest

  说明

  上述jdbctest为测试用例程序的名称。

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)

参数

表 4-3 数据库连接参数

参数	描述
url	gaussdbjdbc.jar数据库连接描述符。 host为服务器名称或IPv4时，格式如下： ● jdbc:gaussdb:(数据库名称缺省则与用户名一致) ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database ● jdbc:gaussdb://host:port/database ● jdbc:gaussdb://host:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://host1:port1,host2:port2/database?param1=value1&param2=value2 host为IPv6时，格式如下： ● jdbc:gaussdb:(数据库名称缺省则与用户名一致) ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database 或 jdbc:gaussdb://[host]/database ● jdbc:gaussdb://[host]:port/database ● jdbc:gaussdb://[host]:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://[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，同时支持IPv4和IPv6。 由于安全原因，数据库CN禁止集群内部其他节点无认证接入。如果要在集群内部访问CN，请将JDBC程序部署在CN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集群外部，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。 ● 当host为IPv6且在url中指定port时，需要通过“[]”分隔IP，如：[IP]:port。 ● 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，详情见**示例**。 ● 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，这会导致RTO时间较高。 ● 建议使用JDBC标准接口建立连接时，确保url格式的合法性，不合法的url会导致异常，且异常中包含原始url字符串，可能造成敏感信息泄漏。

参数	描述
info	info常用属性详情请参见《开发指南》中“应用程序开发教程>基于JDBC开发>连接数据库”章节参数说明。
user	数据库用户。
password	数据库用户的密码。

说明

uppercaseAttributeName参数开启后，如果数据库中有小写、大写和大小写混合的元数据，则只能查询出小写部分的元数据，并以大写的形式输出。使用前请务必确认元数据的存储是否全为小写以避免数据出错。

示例

示例1：连接数据库

```
// 以下用例以gaussdbjdbc.jar为例。
// 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 = "com.huawei.gaussdb.jdbc.Driver";
    // 数据库连接描述符。
    String sourceURL = "jdbc:gaussdb://$ip:$port/database";
    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;
  }
```

示例2：使用Properties对象作为参数建立连接

```
// 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 = "com.huawei.gaussdb.jdbc.Driver";
```



```
    // 数据库连接描述符。
    String sourceURL = "jdbc:gaussdb://$ip:$port/database?autoBalance=true";
    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;
}
```

常用参数详细请见《 开发指南 》中“应用程序开发教程>基于JDBC开发>JDBC常用参数参考”。

示例3：使用流式读功能

```
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// 建立连接。
public static Connection getConnection(String username, String passwd) {
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    String sourceURL = "jdbc:gaussdb://$ip.$port/database?enableStreamingQuery=true";
    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;
}

// 执行普通SQL语句，创建t_user表。
public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();
```



```
        // 执行普通SQL语句。
        stmt.executeUpdate("DROP TABLE IF EXISTS t_user");
        stmt.executeUpdate("CREATE TABLE t_user(id int, name VARCHAR(20));");
        stmt.close();
    } catch (SQLException e) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// 执行预处理语句，批量插入数据。
public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
        // 生成预处理语句。
        pst = conn.prepareStatement("INSERT INTO t_user VALUES (?,?)");
        for (int i = 0; i < 20; i++) {
            // 添加参数。
            pst.setInt(1, i + 1);
            pst.setString(2, "name " + (i + 1));
            pst.addBatch();
        }
        // 执行批处理。
        pst.executeBatch();
        pst.close();
    } catch (SQLException e) {
        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// 开启流式读，查询t_user表中内容。
public static void StreamingQuery(Connection conn) {
    PreparedStatement pst = null;
    ResultSet resultSet = null;

    try {
        // 查询表t_user中的所有值。
        pst = conn.prepareStatement("SELECT * FROM t_user");
        pst.setFetchSize(Integer.MIN_VALUE);// ((PgStatement)statement).enableStreamingResults(); 两者作用
一致。
        resultSet = pst.executeQuery();
        while (resultSet.next()) {
            System.out.println(resultSet.getInt(1));
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
```



```
        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

public static void main(String[] args) throws Exception {
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = getConnection(userName , password);

    CreateTable(conn);

    BatchInsertData(conn);

    StreamingQuery(conn);

    // 关闭数据库连接。
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

须知

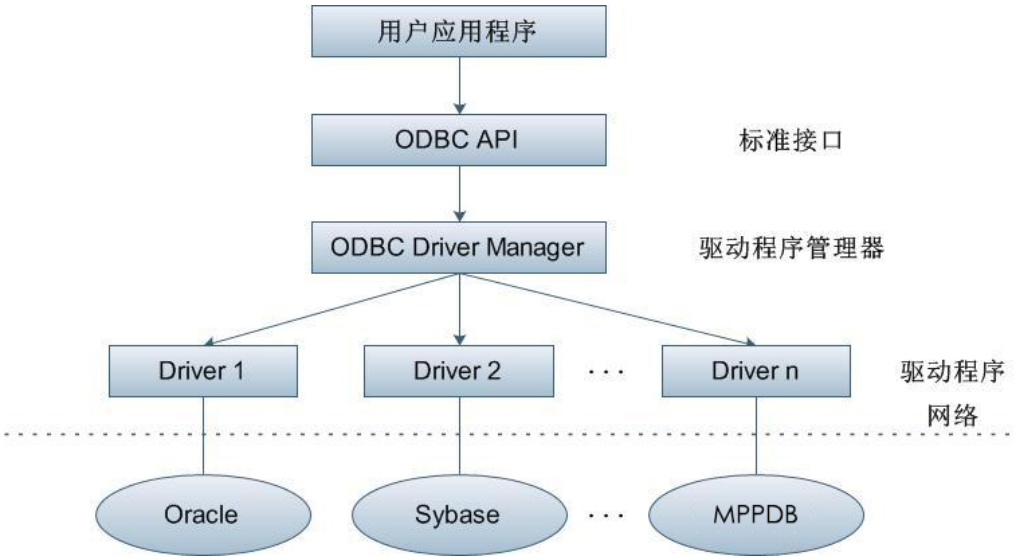
使用流式读功能时，结果集使用完之后，需要执行resultSet.close()或者statement.close()操作，否则当前连接将不可用。

4.1.3 使用 ODBC 连接数据库

ODBC（Open Database Connectivity，开放数据库互连）是由Microsoft公司基于X/OPEN CLI提出的用于访问数据库的应用程序编程接口。应用程序通过ODBC提供的API与数据库进行交互，在避免了应用程序直接操作数据库系统的同时，增强了应用程序的可移植性、扩展性和可维护性。

ODBC的系统结构参见**图1 ODBC系统结构**。

图 4-1 ODBC 系统结构



GaussDB目前在以下环境中提供对ODBC3.5的支持。

表 4-4 ODBC 支持平台

操作系统	平台
EulerOS V2.0SP5	x86_64位
EulerOS V2.0SP9	ARM64位
EulerOS V2.0SP10	x86_64位
EulerOS V2.0SP10	ARM64位
Windows 7	x86_32位
Windows 7	x86_64位
Windows Server 2008	x86_32位
Windows Server 2008	x86_64位
Kylin V10	x86_64位
Kylin V10	ARM64位
UnionTech V20	x86_64位
UnionTech V20	ARM64位
Huawei Cloud EulerOS 2.0	x86_64位
Huawei Cloud EulerOS 2.0	ARM64位

UNIX/Linux系统下的驱动程序管理器主要有unixODBC和iODBC，在这选择驱动管理器unixODBC-2.3.7作为连接数据库的组件。

Windows系统自带ODBC驱动程序管理器，在控制面板->管理工具中可以找到数据源（ODBC）选项。

说明

当前数据库ODBC驱动基于开源版本，对于华为自研的数据类型，tinyint、smalldatetime、nvarchar2在获取数据类型的时候，可能会出现不兼容的情况。

ODBC 相关约束说明

- ODBC不支持容灾切换。
- 当数据库开启proc_outparam_override参数时，ODBC无法正常调用带有out参数的存储过程。

前提条件

已下载Linux版本的ODBC驱动包和Windows版本的ODBC驱动包。

- Linux版本包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Odbc.tar.gz。Linux环境下，开发应用程序要用到unixODBC提供的头文件（sql.h、sqlext.h等）和库libodbc.so。这些头文件和库可从unixODBC-2.3.7的安装包中获得。
- Windows版本包名为GaussDB-Kernel_数据库版本号_Windows_Odbc_X64.tar.gz（64位）。Windows环境下，开发应用程序用到的相关头文件和库文件都由系统自带。

在 Linux 下使用 ODBC 连接数据库

步骤1 获取unixODBC源码包。

获取参考地址：**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz**

下载后请先按照社区提供的完整性校验算法进行完整性校验。下载**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5**，查看MD5值，对比MD5值是否与源码包一致。

步骤2 安装unixODBC。如果机器上已经安装了其他版本的unixODBC，可以直接覆盖安装。

以unixODBC-2.3.7版本为例，在客户端执行如下命令安装unixODBC。

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no #如果要在ARM服务器上编译，请追加一个configure参数： --build=aarch64-unknown-linux-gnu
make
#安装可能需要root权限
make install
```

说明

- 目前不支持unixODBC-2.2.1版本。
- 默认安装到“/usr/local”目录下，生成数据源文件到“/usr/local/etc”目录下，库文件生成在“/usr/local/lib”目录。
- 通过编译带有--enable-fastvalidate=yes选项的unixODBC来获得更高性能。但此选项可能会导致向ODBC API传递无效句柄的应用程序发生故障，而不是返回SQL_INVALID_HANDLE错误。

步骤3 替换客户端GaussDB驱动程序。

将GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Odbc.tar.gz解压。解压后会得到两个文件夹：lib与odbc，在odbc文件夹中还会有一个lib文件夹。将解压后得到的/lib文件夹与/odbc/lib文件夹中的所有动态库都复制到“/usr/local/lib”目录下。

步骤4 配置数据源。

1. 配置ODBC驱动文件。

   在“/usr/local/etc/odbcinst.ini”文件中追加以下内容。

   ```
   [GaussMPP]
   Driver64=/usr/local/lib/gsqlodbcw.so
   setup=/usr/local/lib/gsqlodbcw.so
   ```

   odbcinst.ini文件中的配置参数说明如表4-5所示。

   表 4-5 odbcinst.ini 文件配置参数

参数	描述	示例
[DriverName]	驱动器名称，对应数据源DSN中的驱动名。	[DRIVER_N]
Driver64	驱动动态库的路径。	Driver64=/usr/local/lib/gsqlodbcw.so
setup	驱动安装路径，与Driver64中动态库的路径一致。	setup=/usr/local/lib/gsqlodbcw.so

2. 配置数据源文件。

   在“/usr/local/etc/odbc.ini ”文件中追加以下内容。

   ```
   [gaussdb]
   Driver=GaussMPP
   Servername=127.0.0.1 #数据库Server IP
   Database=db1 # 数据库名
   Username=omm  #数据库用户名
   Password=  #数据库用户密码
   Port=8000 #数据库侦听端口
   Sslmode=allow
   ```

   odbc.ini文件配置参数说明如表4 odbc.ini文件配置参数所示。

   表 4-6 odbc.ini 文件配置参数

参数	描述	示例
[DSN]	数据源的名称。	[gaussdb]
Driver	驱动名，对应odbcinst.ini中的DriverName。	Driver=DRIVER_N
Servername	服务器的IP地址。可配置多个IP地址。支持IPv4和IPv6。	Servername=127.0.0.1
Database	要连接的数据库的名称。	Database=db1
Username	数据库用户名称。	Username=omm

参数	描述	示例
Password	数据库用户密码。 **说明** ODBC驱动本身已经对内存密码进行过清理，以保证用户密码在连接后不会再在内存中保留。 但是如果配置了此参数，由于UnixODBC对数据源文件等进行缓存，可能导致密码长期保留在内存中。 推荐在应用程序连接时，将密码传递给相应API，而非写在数据源配置文件中。同时连接成功后，应当及时清理保存密码的内存段。 **注意** 配置文件中填写密码时，需要遵循http规则： 1. 字符应当采用URL编码规范，如"!"应写作"%21"，"%"应写作"%25"，因此应当特别注意字符。 2. "+"会被替换为空格""。	Password=********
Port	服务器的端口号。当开启负载均衡时，可配置多个端口号，且需与配置的多IP一一对应。如果开启负载均衡配置多个IP时，仍只配置一个端口号，则默认所有IP共有同一个端口号，即为配置的端口号。	Port=8000
Sslmode	开启SSL模式	Sslmode=allow

其中关于Sslmode的选项的允许值，具体信息如**表3 sslmode的可选项及其描述**所示。

表 4-7 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。

sslmode	是否会启用SSL加密	描述
allow	可能	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	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。如果不一致，需要使用root用户修改/etc/hosts文件，将连接数据库节点的IP地址和主机名加入。 **说明** 此模式不支持产品默认证书，生成证书请联系管理员处理。

步骤5 SSL模式。具体操作请联系数据库管理员。

步骤6 配置数据库服务器。具体操作请联系数据库管理员。

步骤7 在客户端配置环境变量。

```
vim ~/.bashrc
```

在配置文件中追加以下内容。

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

步骤8 执行如下命令使设置生效。

```
source ~/.bashrc
```

步骤9 执行以下命令，开始连接数据库。

isql -v *GaussODBC*

GaussODBC为数据源名称

- 如果显示如下信息，表明配置正确，连接成功。

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
```

- 若显示ERROR信息，则表明配置错误。请检查上述配置是否正确。
- 若是集群环境，需要在所有机器上都复制配置一份unixODBC。

----结束

在 Windows 下使用 ODBC 连接数据库

Windows操作系统自带ODBC数据源管理器，无需用户手动安装管理器便可直接进行配置。

步骤1 替换客户端GaussDB驱动程序。

根据需要，将包名为GaussDB-Kernel_数据库版本号_Windows_X64_Odbc.tar.gz的64位驱动或包名为GaussDB-Kernel_数据库版本号_Windows_X86_Odbc.tar.gz的32位驱动解压后，单击gsqlodbc.exe进行驱动安装。

步骤2 打开驱动管理器。

在配置数据源时，请使用ODBC版本对应的ODBC驱动管理器（如果使用64位ODBC驱动，必须要使用64位的ODBC驱动管理器，假设操作系统安装盘符为C盘，如果是其他盘符，请对路径做相应修改）。

- **如果需要在64**位操作系统使用**32**位ODBC驱动请使用：C:\Windows**SysWOW64**\odbcad32.exe，请勿直接使用“控制面板 > 管理工具 > 数据源(ODBC)”。

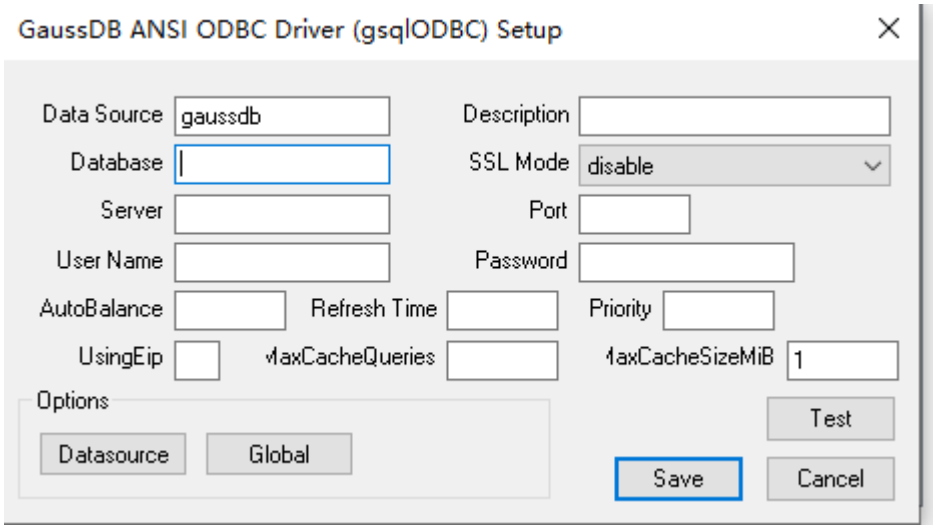
  说明

  WOW64的全称是"Windows 32-bit on Windows 64-bit"，C:\Windows\SysWOW64\存放的是64位系统上的32位运行环境。而C:\Windows\System32\存放的是与操作系统一致的运行环境，具体的技术信息请查阅Windows的相关技术文档。

- **32**位操作系统请使用：C:\Windows\System32\odbcad32.exe，或者单击“计算机 > 控制面板 > 管理工具 > 数据源(ODBC)”打开驱动管理器。
- 64位操作系统请使用：控制面板 > 管理工具 > 数据源(ODBC) 打开驱动管理。

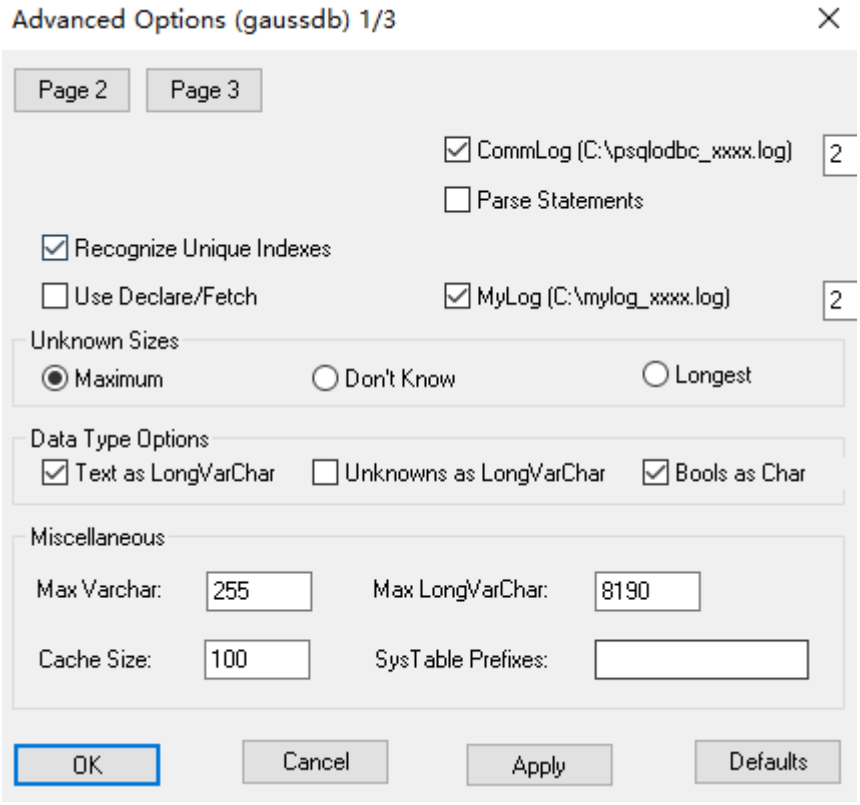
步骤3 配置数据源。

在打开的驱动管理器上，选择“用户DSN > 添加 > GaussDB Unicode”，然后进行配置：



参数说明参考**在Linux下使用ODBC连接数据库**文件参数配置。

其中单击Datasource可以选择配置是否打印日志：



须知

此界面上配置的用户名及密码信息，将会被记录在Windows注册表中，再次连接数据库时不再需要输入认证信息。但是出于安全考虑，建议在单击"Save"按钮保存配置信息前，清空相关敏感信息，在使用ODBC的连接API时，再传入所需的用户名、密码信息。

步骤4 SSL模式。

将3中的设置窗口的“SSL Mode”选项调整至“require”。

表 4-8 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	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	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，并不验证数据库服务器的真实性。

sslmode	是否会启用SSL加密	描述
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。当前windows ODBC不支持cert方式认证。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。当前windows ODBC不支持cert方式认证。

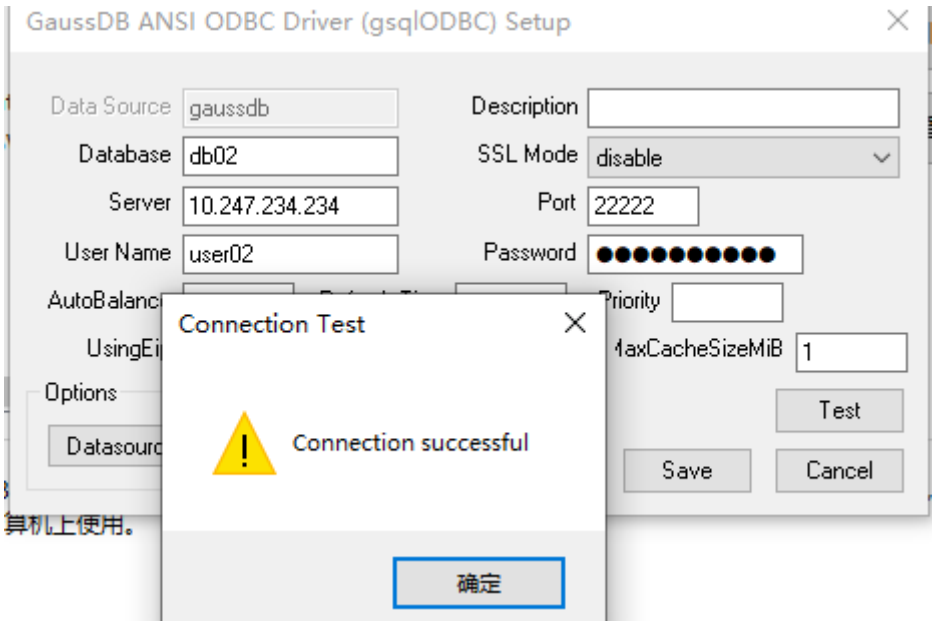
步骤5 配置GaussDB服务器。具体操作请联系管理员。

步骤6 执行如下命令重启集群。

```
gs_om -t stop
gs_om -t start
```

步骤7 单击Test进行测试连接。

- 如果显示如下，则表明配置正确，连接成功。



- 若显示ERROR信息，则表明配置错误。请检查上述配置是否正确。

----结束

4.1.4 使用 libpq 连接数据库

libpq是GaussDB C应用程序接口。libpq是一套允许客户程序向GaussDB服务器服务进程发送查询并且获得查询返回值的库函数。同时也是其他几个GaussDB应用接口下面的引擎，如ODBC等依赖的库文件。本章给出了示例显示如何利用libpq编写代码。

前提条件

本地已安装C语言开发环境。

基于libpq编译开发源程序，主要包括如下步骤：

- 解压GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Libpq.tar.gz文件，其中include文件夹下的头文件为所需的头文件，lib文件夹中为所需的libpq库文件。

  说明

  除libpq-fe.h外，include文件夹下默认还存在头文件postgres_ext.h、gs_thread.h、gs_threadlocal.h，这三个头文件是libpq-fe.h的依赖文件。

- 开发源程序testlibpq.c，源码文件中需引用libpq提供的头文件。
  例如：#include <libpq-fe.h>
- gcc编译libpq源程序，需要通过-I *directory*选项，提供头文件的安装位置（有些时候编译器会查找缺省的目录，因此可以忽略这些选项）。如：
  gcc -I *(头文件所在目录)* -L *(libpq库所在目录)* testlibpq.c -lpq
  例如： gcc -I $(GAUSSHOME)/include/libpq -L $(GAUSSHOME)/lib -lpq testlibpq.c -o testlibpq
- 如果要使用制作文件(makefile)，向CPPFLAGS、LDFLAGS、LIBS变量中增加如下选项：
  CPPFLAGS += -I *(头文件所在目录)*
  LDFLAGS += -L *(libpq库所在目录)*
  LIBS += -lpq
  例如：
  CPPFLAGS += -I$(GAUSSHOME)/include/libpq
  LDFLAGS += -L$(GAUSSHOME)/lib

常用功能示例代码

示例1：

```
/*
 * testlibpq.c
 * 说明： testlibpq.c源程序，提供libpq基本且常见的使用场景。
 * 使用libpq提供的PQconnectdb、PQexec、PQntuples、PQfinish等接口实现数据库建连，执行sql，获取返回结果以及资源清理。
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串 */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *host = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * 用户在命令行上提供了conninfo字符串的值时使用该值
     * 否则环境变量或者所有其它连接参数
     * 都使用缺省值。
```



```
 */
if (argc > 1)
    conninfo = argv[1];
else
    sprintf(conninfo,
        "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
        dbname, port, host, username, passwd);

/* 连接数据库 */
conn = PQconnectdb(conninfo);

/* 检查后端连接成功建立 */
if (PQstatus(conn) != CONNECTION_OK)
{
    fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
    exit_nicely(conn);
}

/*
 * 测试实例涉及游标的使用时候必须使用事务块。
 *把全部放在一个  "select * from pg_database"
 * PQexec() 里，过于简单，不推荐使用。
 */

/* 开始一个事务块 */
res = PQexec(conn, "BEGIN");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/*
 * 在结果不需要的时候PQclear PGresult，以避免内存泄漏
 */
PQclear(res);

/*
 * 从系统表 pg_database（数据库的系统目录）里抓取数据
 */
res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
PQclear(res);

res = PQexec(conn, "FETCH ALL in myportal");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/* 打印属性名称 */
nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
    printf("%-15s", PQfname(res, i));
printf("\n\n");

/* 打印行 */
for (i = 0; i < PQntuples(res); i++)
{
```



```
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    PQclear(res);

    /* 关闭入口 ... 不用检查错误 ... */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* 结束事务 */
    res = PQexec(conn, "END");
    PQclear(res);

    /* 关闭数据库连接并清理 */
    PQfinish(conn);

    return 0;
}
```

示例2：

```
/*
 * testlibpq2.c 测试PQprepare
 *PQprepare： 创建一个给定参数的预备语句，用于PQexecPrepared执行预备语句。
 * 在运行这个例子之前, 可以参考用下面的命令进行建表和插入数据
 * create table t01(a int, b int);
 * insert into t01 values(1, 23);
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
可直接赋值字符串 */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /* PQconnectdb连接数据库, 详细的连接信息为connstr*/
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
             hostaddr, dbname, port, username, paswswd);
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
```



```
    }

    /* 创建表t01 */
    res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01
values(1, 23);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        printf("Command failed: %s.\n", PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }

    /* cmd_s
    sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
    /* cmd_sql中$1对应的参数 */
    paramTypes[0] = 23;
    /* PQprepare创建一个给定参数的预备语句 */
    res = PQprepare(conn,
                    "pre_name",
                    cmd_sql,
                    1,
                    paramTypes);
   if( PQresultStatus(res) != PGRES_COMMAND_OK )
    {
        printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }
    PQclear(res);
    paramValues[0] = cid;
    for (k=0; k<2; k++)
    {
        sprintf(cid, "%d", 1);
        paramLengths[0] = 6;
        paramFormats[0] = 0;
        /* 执行预备语句 */
        res = PQexecPrepared(conn,
                             "pre_name",
                             1,
                             paramValues,
                             paramLengths,
                             paramFormats,
                             0);
        if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
        {
            printf("%s\n",PQerrorMessage(conn));
            PQclear(res);
            PQfinish(conn);
            return -1;
        }
        cnt = PQntuples(res);
        printf("return %d rows\n", cnt);
        for (i=0; i<cnt; i++)
        {
            printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
        }
        PQclear(res);
    }
    /* 执行结束 关闭连接 */
    PQfinish(conn);
    return 0;
}
```

示例3：

```
/*
 * testlibpq3.c
 * 测试PQexecParams
 * PQexecParams：执行一个绑定参数的命令，并以二进制格式请求查询结果。
 * 在运行这个例子之前，用下面的命令填充一个数据库
```



```
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
 * INSERT INTO test1 values (2, 'ho there');
 *
 * 期望的输出是：
 *
 *
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * 这个函数打印查询结果，这些结果是二进制格式，从上面的
 * 注释里面创建的表中抓取出来的。
 */
static void
show_binary_results(PGresult *res)
{
    int         i;
    int         i_fnum,
                t_fnum;

    /* 使用 PQfnumber 来避免对结果中的字段顺序进行假设 */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        int         ival;

        /* 获取字段值（忽略可能为空的可能） */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * INT4 的二进制表现形式是网络字节序，
         * 建议转换成本地字节序。
         */
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * TEXT 的二进制表现形式是文本，因此libpq能够给它附加一个字节零，
         * 把它看做 C 字符串。
         *
         */

        printf("tuple %d: got\n", i);
```



```
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
可直接赋值字符串 */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
    int         paramFormats[1];
    uint32_t    binaryIntVal;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * 如果用户在命令行上提供了参数，
     * 那么使用该值为conninfo 字符串；否则
     * 使用环境变量或者缺省值。
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, hostaddr, username, passwd);

    /* 和数据库建立连接 */
    conn = PQconnectdb(conninfo);

    /* 检查与服务器的连接是否成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);
```



```
    /* 把整数值 "2" 转换成网络字节序 */
    binaryIntVal = htonl((uint32_t) 2);

    /* 为 PQexecParams 设置参数数组 */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1;        /* 二进制 */
    /* PQexecParams执行一个绑定参数的命令 */
    res = PQexecParams(conn,
                       "SELECT * FROM test1 WHERE i = $1::int4",
                       1,       /* 一个参数 */
                       NULL,    /* 让后端推导参数类型 */
                       paramValues,
                       paramLengths,
                       paramFormats,
                       1);      /* 要求二进制结果 */

    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    /* 显示二进制结果 */
    show_binary_results(res);

    PQclear(res);

    /* 关闭与数据库的连接并清理 */
    PQfinish(conn);

    return 0;
}
```

4.1.5 使用 Psycopg 连接数据库

Psycopg是一种用于执行SQL语句的PythonAPI，可以为GaussDB数据库提供统一访问接口，应用程序可基于它进行数据操作。Psycopg2是对libpq的封装，主要使用C语言实现，既高效又安全。它具有客户端游标和服务器端游标、异步通信和通知、支持“COPY TO/COPY FROM”功能。支持多种类型Python开箱即用，适配GaussDB数据类型。通过灵活的对象适配系统，可以扩展和定制适配。Psycopg2兼容Unicode。

GaussDB数据库提供了对Psycopg2特性的支持，并且支持psycopg2通过SSL模式连接。

表 4-9 Psycopg 支持平台

操作系统	平台	Python版本
EulerOS V2.0SP5	• ARM64位 • x86_64位	3.8.5
EulerOS V2.0SP9	• ARM64位 • x86_64位	3.7.4
EulerOS V2.0SP10、Kylin v10、UnionTech20	• ARM64位 • x86_64位	3.7.9
EulerOS V2.0SP11、Suse 12.5	• ARM64位 • x86_64位	3.9.11

操作系统	平台	Python版本
Huawei Cloud EulerOS 2.0	• ARM64位 • x86_64位	3.9.9

须知

psycopg2在编译过程中，会链接（link）GaussDB的openssl，GaussDB的openssl与操作系统自带的openssl可能不兼容。如果遇到不兼容现象，例如提示"version 'OPENSSL_1_1_1f' not found"，请使用环境变量LD_LIBRARY_PATH进行隔离，以避免混用操作系统自带的openssl与GaussDB依赖的openssl。

例如，在执行某个调用psycopg2的应用软件client.py时，将环境变量显性赋予该应用软件：

```
export LD_LIBRARY_PATH=/path/to/gaussdb/libs:$LD_LIBRARY_PATH python client.py
```

其中，/path/to/pyscopg2/lib 表示GaussDB依赖的openssl库所在目录，需根据文件实际存储路径修改。

前提条件

本地已安装python语言运行环境。

连接数据库

步骤1 准备相关驱动和依赖库。可以从发布包中获取，包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Python.tar.gz。

解压后有两个文件夹：

- psycopg2：psycopg2库文件。
- lib：lib库文件。

步骤2 加载驱动。

- 在使用驱动之前，需要做如下操作：
  a. 先解压版本对应驱动包。
  ```
  tar zxvf xxxx-Python.tar.gz
  ```
  b. 使用root用户将psycopg2复制到python安装目录下的site-packages文件夹下。
  ```
  su root
  cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
  ```
  c. 修改psycopg2目录权限为755。
  ```
  chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
  ```
  d. 将psycopg2目录添加到环境变量$PYTHONPATH，并使之生效。
  ```
  export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
  ```
  e. 对于非数据库用户，需要将解压后的lib目录，配置在LD_LIBRARY_PATH中。
  ```
  export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
  ```
- 在创建数据库连接之前，需要先加载如下数据库驱动程序：
  ```
  import  psycopg2
  ```

步骤3 连接数据库。

非SSL方式连接数据库：

1. 使用psycopg2.connect函数获得connection对象。
2. 使用connection对象创建cursor对象。

SSL方式连接数据库：

用户通过psycopy2连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯。在使用SSL时，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参见Openssl相关文档和命令。

1. 使用*.ini文件（python的configparser包可以解析这种类型的配置文件）保存数据库连接的配置信息。
2. 在连接选项中添加SSL连接相关参数：sslmode、sslcert、sslkey、sslrootcert。
   a. sslmode：可选项见表4-10。
   b. sslcert：客户端证书路径。
   c. sslkey：客户端密钥路径。
   d. sslrootcert：根证书路径。
3. 使用psycopg2.connect函数获得connection对象。
4. 使用connection对象创建cursor对象。

⚠ 注意

使用SSL安全连接数据库，需保证所使用的python解释器为生成动态链接库（.so）文件的方式编译，可通过如下步骤确认python解释器的连接方式。

1. 在python解释器命令行中输入import ssl，导入SSL。
2. 执行ps ux查询python解释器运行的pid（假设pid为******）。
3. 在shell命令行中执行pmap -p ****** | grep ssl，查看返回结果中是否包含libssl.so的相关路径。如果有，则python解释器为动态链接方式编译。

表 4-10 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	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	可能	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	是	必须使用SSL安全连接，但是仅进行数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且校验服务端CA有效性。

sslmode	是否会启用SSL加密	描述
verify-full	是	必须使用SSL安全连接，目前GaussDB暂不支持。

----结束

4.1.6 使用 Hibernate 连接数据库

Hibernate是一种Java语言下的对象关系映射（ORM）解决方案。它提供了一个使用方便的持久化框架，可以自动将数据库表和Java对象之间建立映射关系，使得开发者可以使用面向对象的方式来操作数据库。使用Hibernate，开发者就不需要手动编写大量的SQL和JDBC代码，从而大大简化了数据访问层的开发工作。

本章节指导用户使用Hibernate连接GaussDB进行建表，修改表以及增删改查操作示例。

配置 pom 依赖

```
<dependency>
   <groupId>com.huaweicloud.gaussdb</groupId>
   <artifactId>opengaussjdbc</artifactId>
   <version>503.2.T35</version>
</dependency>
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>5.3.7.Final</version>
</dependency>
```

⚠ 注意

配置pom依赖时，需要提前配置好maven环境。

配置 hibernate.cfg.xml 资源文件

```
<?xml version="1.0" encoding="utf-8"?>
      <!DOCTYPE hibernate-configuration PUBLIC
            "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
            "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <!--GaussDB 连接信息-->
      <property name="connection.driver_class">com.huawei.opengauss.jdbc.Driver</property>
      <property name="connection.url">jdbc:opengauss://***.***.***.***（需要替换为数据库ip）:20000（需要替换为数据库端口）/test?currentSchema=test（test需要替换为对应的database和schema）</property>
      <property name="connection.username">***（需要替换为正确的用户名）</property>
      <property name="connection.password">******（需要替换为正确的密码）</property>

      <!-- 以下为可选配置 -->

      <!--是否支持方言 -->
      <!--在 postgresql 兼容模式下，必须有如下配置-->
<property name="dialect">org.hibernate.dialect.PostgreSQL82Dialect</property>
      <!--执行CURD时是否打印sql 语句  -->
      <property name="show_sql">true</property>

      <!--自动建表(修改表)-->
```

```
    <!-- <property name="hbm2ddl.auto">update</property>-->
    <!-- <property name="hbm2ddl.auto">create</property>-->

    <!-- 资源注册 (实体类映射文件)-->
    <mapping resource="./student.xml"/>
  </session-factory>
</hibernate-configuration>
```

准备实体类和实体类映射表 xml 文件

以Student类以及student.xml文件为例，文件路径为com.howtodoinjava.hibernate.test.dto。

1. 实体类。

```
public class Student implements Serializable {
    int id;
    String name;
//  String age;
    // 此处省略构造器，get，set方法
}
```

2. student.xml。

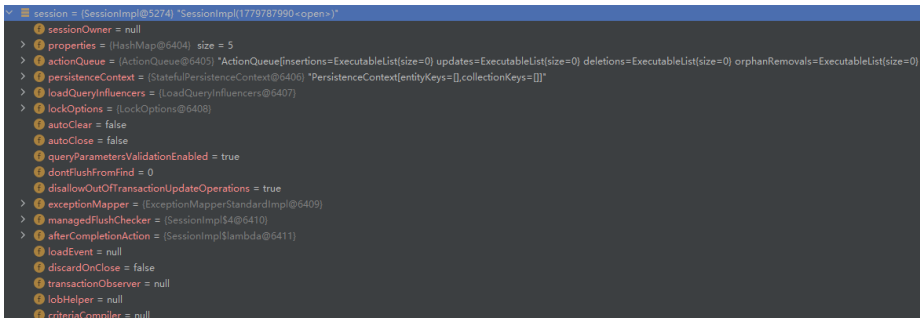
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        '-//Hibernate/Hibernate Mapping DTD 3.0//EN'
        'http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd'>
<hibernate-mapping>
    <!-- 类与表的映射制作在class元素上 -->
    <!-- name:全路径类名 -->
    <!-- table:表名 -->
    <class name="com.howtodoinjava.hibernate.test.dto.Student" table="student">
        <!-- 主键的映射制作在 id 元素上 -->
        <!-- name:对象中用于作为主键的属性名 -->
        <!-- column:表中主键字段名 -->
        <!-- 如果 name 与 column 值相同，可以省略 column -->
        <id name="id" column="id">
            <!-- 将 generator 元素 class 属性设置为 "assigned" 手动生成,必须给 id -->
            <generator class="assigned"  />

            <!--**要注意 Hibernate 主键生成策略**-->

        </id>
        <!-- 属性与字段的映射制作在 property 元素上 -->
        <!-- name:类中的属性名 -->
        <!-- column:表中的字段名 -->
        <!-- 如果 name 与 column 值相同，可以省略 column -->
        <property name="name" />
        <!--此处同实体类-->
        <!--<property name="age"/>-->
    </class>
</hibernate-mapping>
```

功能测试示例

1. 测试连接功能。
   a. 测试方法：

```
@Test
    public void testConnection() {
        // 加载配置信息
        Configuration conf = new Configuration().configure();
        // 基于配置信息,创建 SessionFactory 对象
        SessionFactory sessionFactory = conf.buildSessionFactory();
        // 打开一个与数据库相关的 session 对象
        Session session = sessionFactory.openSession();
        System.out.println(session);
    }
```


b. 打断点可见成功建立连接：



2. 自动建表。

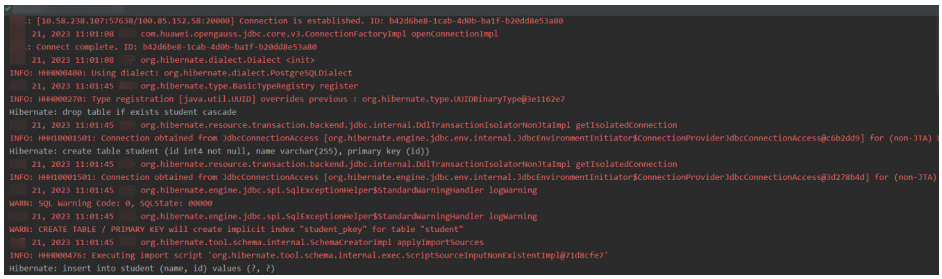
a. 将配置文件本行注释打开：

```
<property name="hbm2ddl.auto">create</property>
```

b. 将数据库中 student 表删除后，执行如下测试方法：

```
@Test
    public void testCreateTableAndInsertData() {
        // 创建要测试的对象
        Student student = new Student();
        student.setId(16);
        student.setName("xiaoming");
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // 通过 session 保存数据
        session.save(student);
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

c. 查看控制台打印出来的所执行 sql 语句以及 GaussDB 数据库中的结果：




```
test=> select * from test.student;
 id |   name
----+----------
 16 | xiaoming
(1 row)
```

成功创建student表并插入字段（id = 16, name = "xiaoming"）。

3. 修改表（增加字段）。

a. 首先修改配置文件，需要配置需要修改表所在的schema路径在url中。

如本用例中，student 表在名为 test 的 databse 下的名为 test 的 schema 下，即该表路径为test.test.student：

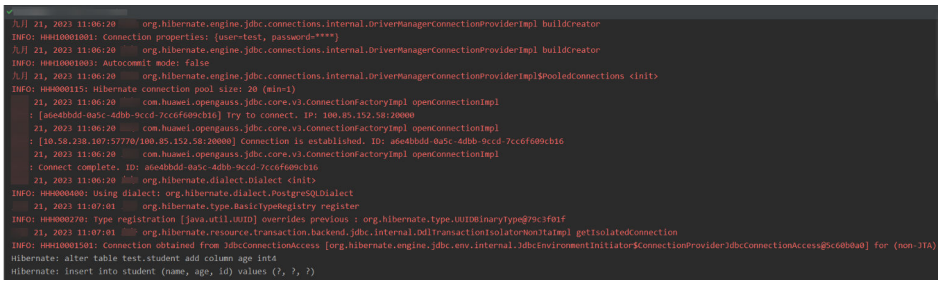
```
<property name="connection.url">jdbc:opengauss://xxx.xxx.xxx.xxx（需要替换为数据库ip）:20000（需要替换为数据库端口）/test?currentSchema=test（test需要替换为对应的database和schema）
```

```
</property>
<!-- 打开update注释 -->
<property name="hbm2ddl.auto">update</property>
```

b. 将实体类和xml文件中关于age属性的注释打开，执行如下方法：

```
@Test
    public void testAlterTable() {
        // 创建要测试的对象
        Student student = new Student();
        student.setId(15);
        student.setName("xiaohong");
        student.setAge("20");
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // 通过 session 保存数据
        session.save(student);
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

c. 查看控制台打印出来的所执行 sql 语句以及 GaussDB 数据库中的结果：




```
test=> select * from test.student;
 id |   name    | age
----+-----------+-----
 16 | xiaoming  |
 15 | xiaohong  |  20
(2 rows)
```

在原有表的基础上，框架根据实体类及xml文件，自动对表进行了新增字段处理。

4. 持久化数据。

a. 测试方法：

```
@Test
    public void testInsert() {
        Student s1 = new Student(1,"q");
        Student s2 = new Student(2,"w");
        Student s3 = new Student(3,"e");
        ArrayList<Student> students = new ArrayList<>();
        students.add(s1);
        students.add(s2);
        students.add(s3);
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // 通过 session 保存数据
        for (Student student : students) {
            session.save(student);
        }
```

```
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

b. 执行结果如下：

```
INFO: HHH10001001: Connection properties: {user=test, password=****}
    21, 2023 11:10:53     org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
    21, 2023 11:10:53     org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
    21, 2023 11:10:53     com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
     [d8970650-ea76-4e3e-999b-02fd926e4f41] Try to connect. IP: 100.85.152.58:20000
    21, 2023 11:10:53     com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
     [10.58.238.107:57875/100.85.152.58:20000] Connection is established. ID: d8970650-ea76-4e3e-999b-02fd926e4f41
    21, 2023 11:10:53     com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
     Connect complete. ID: d8970650-ea76-4e3e-999b-02fd926e4f41
    21, 2023 11:10:54     org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
    21, 2023 11:11:33     org.hibernate.type.BasicTypeRegistry register
INFO: HHH000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@3e1162e7
    21, 2023 11:11:34     org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@7bca6fac] for (non-JTA)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
```

```
test=> select * from test.student;
 id |   name    | age
----+-----------+-----
 16 | xiaoming  |
 15 | xiaohong  |  20
  1 | q         |   0
  2 | w         |   0
  3 | e         |   0
(5 rows)
```

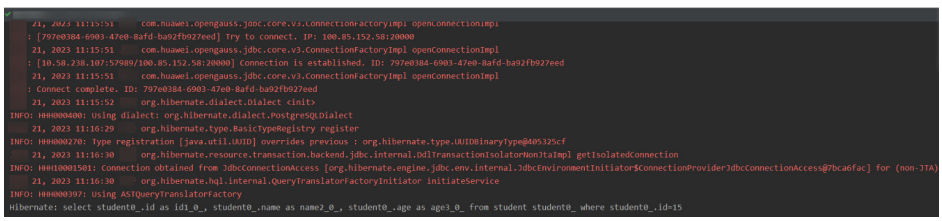
成功插入三行数据。

5. HQL模式查询。

a. 测试方法：

```
@Test
    public void testHQL() {
        //HQL模式
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        // 创建会话
        Session session = sessionFactory.openSession();
        // 开始事务
        Transaction tx = session.beginTransaction();

        // 创建HQL查询
        String hql = "FROM Student S WHERE S.id = 15";
        Query query = session.createQuery(hql);

        // 执行查询并获取结果
        List results = query.list();

        // 提交事务
        tx.commit();

        // 关闭会话
        session.close();
    }
```

b. 执行结果如下：



```
    21, 2023 11:15:51     com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
    : [797e0384-6903-47e0-8afd-ba92fb927eed] Try to connect. IP: 100.85.152.58:20000
    21, 2023 11:15:51     com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
    : [10.58.238.107:57989/100.85.152.58:20000] Connection is established. ID: 797e0384-6903-47e0-8afd-ba92fb927eed
    21, 2023 11:15:51     com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
    : Connect complete. ID: 797e0384-6903-47e0-8afd-ba92fb927eed
    21, 2023 11:15:52     org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
    21, 2023 11:16:29     org.hibernate.type.BasicTypeRegistry register
INFO: HHH000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@405325cf
    21, 2023 11:16:30     org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@7bca6fac] for (non-JTA)
    21, 2023 11:16:30     org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select student0_.id as id1_0_, student0_.name as name2_0_, student0_.age as age3_0_ from student student0_ where student0_.id=15
```


6. SQL模式查询。
   a. 测试方法：

```
@Test
    public void testQuery() {
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // SQL模式
        List<Student> students = session.createSQLQuery("select * from test.student where id =
1").addEntity(Student.class).list();
        for (int i = 0; i < students.size(); i++) {
            System.out.println(students.get(i));
        }
        students.get(0).setAge("20");
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

   b. 执行结果如下：

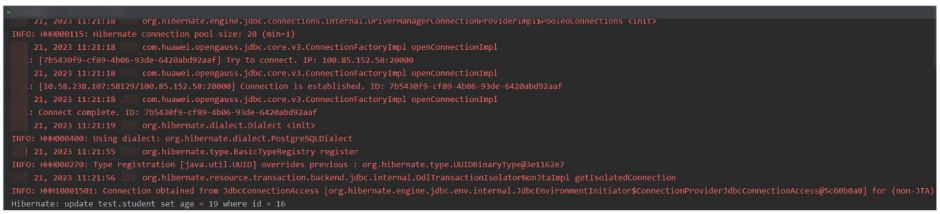
```
test=> select * from test.student;
 id |   name    | age
----+-----------+-----
 16 | xiaoming  |
 15 | xiaohong  |  20
  1 | q         |  20
  2 | w         |   0
  3 | e         |   0
(5 rows)
```

   成功查询到id=1的学生数据并修改其age字段为20。

7. 修改操作。
   - 测试方法：

```
@Test
    public void testUpdate() {
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // SQL模式
        session.createSQLQuery("update test.student set age = 19 where id =
16").executeUpdate();
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

   - 执行结果如下：




```
test=> select * from test.student;
 id |   name    | age
----+-----------+-----
 15 | xiaohong  |  20
  1 | q         |  20
  2 | w         |   0
  3 | e         |   0
 16 | xiaoming  |  19
(5 rows)
```

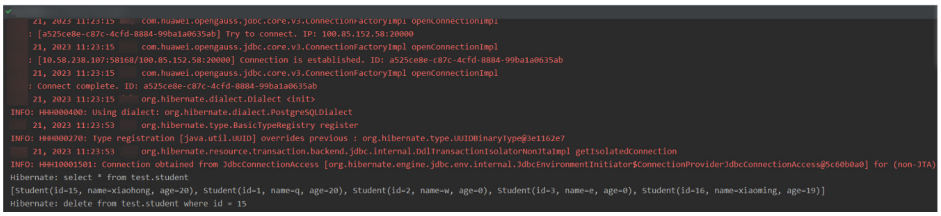
成功将id=16的学生的age字段修改为19。

8. 删除操作。

   a. 测试方法：

```
@Test
   public void testDelete() {
      // 开启事务,基于 session 得到
      Configuration conf = new Configuration().configure();
      SessionFactory sessionFactory = conf.buildSessionFactory();
      Session session = sessionFactory.openSession();
      Transaction transaction = session.beginTransaction();
      // SQL模式
      List<Student> students = session.createSQLQuery("select * from
test.student").addEntity(Student.class).list();
      System.out.println(students);
      session.createSQLQuery("delete from test.student where id = " +
students.get(0).getId()).executeUpdate();
      // 提交事务
      transaction.commit();
      // 操作完毕,关闭 session 连接对象
      session.close();
   }
```

   b. 执行结果如下：




```
test=> select * from test.student;
 id |   name    | age
----+-----------+-----
  1 | q         |  20
  2 | w         |   0
  3 | e         |   0
 16 | xiaoming  |  19
(4 rows)
```

student表中id=15的字段已被删除。

4.1.7 使用 MyBatis 连接数据库

MyBatis是一个优秀的持久层框架，它支持定制化SQL、存储过程以及高级映射，其避免了几乎所有的JDBC代码和手动设置参数以及获取结果集。MyBatis可以使用简单的XML或注解进行配置和原始映射，将接口和Java的POJOs（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

本章节指导用户使用MyBatis连接GaussDB示例的配置依赖和配置文件。

配置 pom 依赖

```
<dependency>
    <groupId>com.huaweicloud.gaussdb</groupId>
    <artifactId>opengaussjdbc</artifactId>
    <version>503.2.T35</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.6</version>
</dependency>
```

⚠ 注意

配置pom依赖时，需要提前配置好maven环境。

配置文件

配置mybatis-config.xml资源文件。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<!-- 配置文件的根元素 -->
<configuration>
    <!--配置全局属性-->
    <settings>
        <!--使用jdbc的getGeneratedKeys获取数据库自增主键值-->
        <setting name="useGeneratedKeys" value="true"/>
        <!--使用列标签替换列别名 默认为true-->
        <setting name="useColumnLabel" value="true" />
        <!--开启驼峰式命名转换：Table{create_time} -> Entity{createTime}-->
        <setting name="mapUnderscoreToCamelCase" value="true" />
        <setting name="logImpl" value="STDOUT_LOGGING" />
    </settings>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.huawei.opengauss.jdbc.Driver"/>
                <property name="url" value="jdbc:opengauss://***.***.***.***（需要替换为数据库ip）:20000（需要替换为数据库端口）/test?（test需要替换为对应的database）connectionTimeout=10"/>
                <property name="username" value="***（需要替换为正确的用户名）"/>
                <property name="password" value="*******（需要替换为正确的密码）"/>
            </dataSource>
        </environment>
    </environments>
    <!--注册 mapper（mapper.xml 所在地址）-->
    <mappers>
        <mapper resource="mapper/StudentDaoMapper.xml"></mapper>
    </mappers>
</configuration>
```

示例

1. 测试实体类StudentEntity.java（com.huawei.entity路径下）：

```
public class StudentEntity {
    Integer id;
    String name;
}
```

2. 实体类对应的StudentDaoMapper.xml文件（resources.mapper路径下）：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="StudentMapper">
    <!-- 根据主键查询-->
    <select id="getList" resultType="com.huawei.entity.StudentEntity" >
        select * from student;
    </select>
</mapper>
```

3. 查询表测试方法：

```
@Test
public void mainTest() throws IOException {
    // 1.读取mybatis的核心配置文件(mybatis-config.xml)
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
    // 2.通过配置信息获取一个SqlSessionFactory工厂对象
    SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
    // 3.通过工厂获取一个SqlSession对象
    SqlSession session = fac.openSession();
    // 4.通过namespace+id找到要执行的sql语句并执行sql语句
    List<StudentEntity> list = session.selectList("StudentMapper.getList");
    // 5.输出结果
    list.forEach(i -> {
        System.out.println(i.toString());
    });
}
```

4. 查询结果日志：

```
Setting autocommit to false on JDBC Connection [com.huawei.opengauss.jdbc.jdbc.PgConnection@47caedad]
==>  Preparing: select * from student;
==> Parameters:
<==    Columns: id, name
<==        Row: 1, test
<==      Total: 1
StudentEntity(id=1, name=test)
```

注意

目前MybatisPlus插件中PaginationInnerInterceptor未对GaussDB驱动做适配。创建PaginationInnerInterceptor对象时指定DbType为POSTGRE_SQL即可解决。eg: PaginationInnerInterceptor innerInterceptor = new PaginationInnerInterceptor(DbType.POSTGRE_SQL)。

4.1.8 使用 JayDebeApi 连接数据库

JayDebeApi是一个Python模块，其提供了一种方便、高效的方式，让Python开发者能够利用Java的JDBC驱动程序来连接和操作各种数据库。

本章节指导用户如何使用JayDebeApi连接GaussDB数据库。

环境配置

1. 配置GaussDB开发环境。

准备好基本的 GaussDB 开发环境，获取数据库连接参数，例如：

```
gsql -h ***.***.***.*** -p 20000 -U *** -W ****** -d test
```

参数说明：

-h：GaussDB服务器IP。

-p：GaussDB连接端口。

-U：连接用户名。

-W：用户密码。

-d：连接的DATABASE名。

2. 安装JayDeBeApi驱动。
   a. 在计算机上安装Java JDK 8版本以及Python3版本，可以通过如下命令确认版本：
      ```
      java -version
      python --version
      pip --version
      ```
   b. 如果服务器能连接到Python源（Python的软件包索引PyPI），可以通过pip命令安装JayDeBeApi：
      ```
      pip install jaydebeapi
      ```
      如果不能，可以通过下载离线安装包然后进行本地安装的方式安装JayDeBeApi。
3. 获取GaussDB驱动程序安装包。

   根据不同版本的实例，下载不同版本的发布包，如表4-11所示。

   表 4-11 驱动包下载列表

版本	下载地址
3.x	**驱动包** **驱动包校验包**
2.x	**驱动包** **驱动包校验包**

   为了防止软件包在传递过程或存储期间被恶意篡改，下载软件包时需下载对应的校验包对软件包进行校验，校验方法如下：

   a. 上传软件包和软件包校验包到虚拟机（Linux操作系统）的同一目录下。
   b. 执行如下命令，校验软件包完整性。

      cat *GaussDB_driver.zip.sha256* **| sha256sum --check**

      如果回显OK，则校验通过。

      ```
      GaussDB_driver.zip: OK
      ```

使用示例

1. 新建脚本文件。
   – 新建一个test_jaydebeapi.py文件，写入代码如下：
      ```
      #!/usr/bin/env python3.x
      # -*- coding: UTF-8 -*-
      encoding = "utf8"
      ```



```
import jaydebeapi

def test_jaydebeapi():
    #需要配置的参数
    url = 'jdbc:opengauss://***.***.***.***:20000/test'
    user = '***'
    password = '******'
    driver = 'com.huawei.opengauss.jdbc.Driver'
    jarFile = './opengaussjdbc.jar'

    conn = jaydebeapi.connect(driver, url, [user, password], jarFile)
    cur = conn.cursor()

    #创建表students
    sql = 'create table students (id int, name varchar(20))'
    cur.execute(sql)

    #往students表中插入三组数据
    sql = "insert into students values(1,'xiaoming'),(2,'xiaohong'),(3,'xiaolan')"
    cur.execute(sql)

    #查询students表中的所有数据
    sql = 'select * from students'
    cur.execute(sql)
    ans = cur.fetchall()
    print(ans)

    #更新students表中的数据
    sql = 'update students set name = \'xiaolv\' where id = 1'
    cur.execute(sql)

    #再次查询students表中的所有数据
    sql = 'select * from students'
    cur.execute(sql)
    ans = cur.fetchall()
    print(ans)

    #删除表students
    sql = 'drop table students'
    cur.execute(sql)

    cur.close()
    conn.close()

test_jaydebeapi()
```

- 配置代码中的参数：

```
#连接url，设置数据库服务器IP、端口、数据库名
url = 'jdbc:opengauss://***.***.***.***:20000/test'
#设置用户名
user = '***'
#设置密码
password = '******'
#JDBC驱动类路径
driver = 'com.huawei.opengauss.jdbc.Driver'
#JDBC驱动jar包路径（默认放在与test_jaydebeapi.py文件同目录下）
jarFile = './opengaussjdbc.jar'
```

2. 执行程序。

   使用以下命令执行test_jaydebeapi.py文件：

   ```
   python ./test_jaydebeapi.py
   ```

3. 预期结果。

   成功连接GaussDB数据库，并返回两次查询结果，如下所示：


```
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: [f42f7374-3bfd-4f91-b370-4bbd007aea16] Try to connect. IP: 127.0.0.1:20000
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: [127.0.0.1:48426/127.0.0.1:20000] Connection is established. ID: f42f7374-3bfd-4f91-b370-4bbd007aea16
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: Connect complete. ID: f42f7374-3bfd-4f91-b370-4bbd007aea16
[(1, 'xiaoming'), (2, 'xiaohong'), (3, 'xiaolan')]
[(1, 'xiaoly'), (2, 'xiaohong'), (3, 'xiaolan')]
```

4.2 主备版

4.2.1 开发规范

如果用户在APP的开发过程中，使用了连接池机制，那么需要遵循如下规范：

- 如果在连接中设置了GUC参数，那么在将连接归还连接池之前，必须使用“SET SESSION AUTHORIZATION DEFAULT;RESET ALL;”将连接的状态清空。
- 如果使用了临时表，那么在将连接归还连接池之前，必须将临时表删除。

否则，连接池里面的连接就是有状态的，会对用户后续使用连接池进行操作的正确性带来影响。

应用程序开发驱动兼容性说明如表4-12所示。

表 4-12 兼容性说明

驱动	兼容性说明
JDBC、Go、ODBC、libpq、Psycopg、ecpg	驱动前向兼容数据库，若需使用驱动与数据库同步增加的新特性，须升级数据库。

须知

- behavior_compat_options='proc_outparam_override' 重载参数仅在A兼容模式可用。
- 原则上，兼容性参数应在创建数据库后就设置，不应在使用过程中来回切换。
- 涉及使用以下场景的特性需要配合将JDBC驱动升级到503.1.0及以上的配套版本。
  1. 全密态内存解密逃生通道。
  2. 通过JDBC使用record、array、tableof嵌套自定义类型。
  3. 开启s2兼容性参数，设置sessiontimezone的合法性校验。

在多线程环境下使用驱动：

JDBC驱动程序线程不是安全的，无法保证连接上的方法同步。由调用者同步对驱动程序的调用。

4.2.2 使用 JDBC 连接数据库

JDBC（Java Database Connectivity，Java数据库连接）是用于执行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。

GaussDB库提供了对JDBC 4.2特性的支持，需要使用JDK1.8版本编译程序代码，不支持JDBC桥接ODBC方式。

前提条件

计算机已安装Java JDK8版本。

JDBC 包

包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Jdbc.tar.gz。解压后JDBC的驱动jar包：

- gaussdbjdbc.jar：主类名为“com.huawei.gaussdb.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，推荐使用此驱动包。本章的Java代码示例默认使用gaussdbjdbc.jar包。
- gscejdbc.jar：主类名为“com.huawei.gaussdb.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，此驱动包打包了密态数据库需要加载的加解密相关的依赖库，密态场景推荐使用此驱动包。目前仅支持EulerOS操作系统。
- gaussdbjdbc-JRE7.jar：主类名为“com.huawei.gaussdb.jdbc.Driver”，数据库连接的url前缀为“jdbc:gaussdb”，在JDK1.7环境使用gaussdbjdbc-JRE7.jar包。

⚠ 注意

- 使用gscejdbc.jar驱动包时，需要先设置环境变量LD_LIBRARY_PATH。具体使用方式见《特性指南》中“设置密态等值查询 > 使用JDBC操作密态数据库”章节。
- 在JDK1.8环境中使用gaussdbjdbc.jar，不推荐使用gaussdbjdbc-JRE7.jar。
- 其他JDBC的jar包介绍请参考《开发指南》中“应用程序开发教程 > JDBC兼容性包”章节。

驱动类

在创建数据库连接之前，需要加载数据库驱动类“com.huawei.gaussdb.jdbc.Driver”。

说明

1. 由于GaussDB在JDBC的使用上与PG的使用方法保持兼容，所以同时在同一进程内使用两个JDBC驱动的时候，可能会造成类名冲突。
2. GaussDB JDBC驱动主要做了以下特性的增强：
    1. 支持SHA256加密方式登录。
    2. 支持对接实现sf4j接口的第三方日志框架。
    3. 支持容灾切换。

环境类

客户端需配置JDK1.8。JDK是跨平台的，支持Windows、Linux等多种平台，下面以Windows为例，配置方法如下：

步骤1 DOS窗口（windows下的命令提示符）输入“java -version”，查看JDK版本，确认为JDK1.8版本。如果未安装JDK，请从官方网站下载安装包并安装。

步骤2 根据如下步骤配置系统环境变量。

1. 右键单击“我的电脑”，选择“属性”。
2. 在“系统”页面左侧导航栏单击“高级系统设置”。
3. 在“系统属性”页面，“高级”页签上单击“环境变量”。
4. 在“环境变量”页面上，“系统变量”区域单击“新建”或“编辑”配置系统变量。变量说明如表4-13所示。

表 4-13 变量说明

变量名	操作	变量值
JAVA_HOME	– 若存在，则单击“编辑”。 – 若不存在，则单击“新建”。	JAVA的安装目录。 例如：C:\Program Files\Java\jdk1.8.0_131。
Path	单击“编辑”。	– 若配置了JAVA_HOME，则在变量值的最前面加上： %JAVA_HOME%\bin。 – 若未配置JAVA_HOME，则在变量值的最前面加上 JAVA安装的全路径：C:\Program Files\Java\jdk1.8.0_131\bin。
CLASSPATH	单击“新建”。	%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar。

----结束

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。

加载驱动有两种方法：

- 在代码中创建连接之前任意位置隐含装载：Class.forName("com.huawei.gaussdb.jdbc.Driver");。
- 在JVM启动时参数传递：java -Djdbc.drivers=com.huawei.gaussdb.jdbc.Driver jdbctest。

  说明

  上述jdbctest为测试用例程序的名称。

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)。

参数

表 4-14 数据库连接参数

参数	描述
url	gaussdbjdbc.jar数据库连接描述符。 host为服务器名称或IPv4时，格式如下： ● jdbc:gaussdb:(数据库名称缺省则与用户名一致) ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database ● jdbc:gaussdb://host:port/database ● jdbc:gaussdb://host:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://host1:port1,host2:port2/database?param1=value1&param2=value2 host为IPv6时，格式如下： ● jdbc:gaussdb:(数据库名称缺省则与用户名一致) ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database 或 jdbc:gaussdb://[host]/database ● jdbc:gaussdb://[host]:port/database ● jdbc:gaussdb://[host]:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://[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，同时支持IPv4和IPv6。 由于安全原因，数据库主节点禁止数据库内部其他节点无认证接入。如果要在数据库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数据库外部，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。 ● 当host为IPv6且在url中指定port时，需要通过“[]”分隔IP，如：[IP]:port。 ● 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，详情见**示例**。 ● 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，这会导致RTO时间很高。 ● 建议使用JDBC标准接口建立连接时，确保url格式的合法性，不合法的url会导致异常，且异常中包含原始url字符串，可能造成敏感信息泄漏。

参数	描述
info	info常用属性详情请参见《 开发指南 》应用程序开发教程>基于JDBC开发>连接数据库章节参数说明。
user	数据库用户。
password	数据库用户的密码。

说明

uppercaseAttributeName参数开启后，如果数据库中有小写、大写和大小写混合的元数据，只能查询出小写部分的元数据，并以大写的形式输出，使用前请务必确认元数据的存储是否全为小写以避免数据出错。

示例

示例1：连接数据库

```
//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 = "com.huawei.gaussdb.jdbc.Driver";
    //数据库连接描述符。
    String sourceURL = "jdbc:gaussdb://$ip:$port/database";
    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;
  }
```

示例2：使用Properties对象作为参数建立连接

```
// 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 = "com.huawei.gaussdb.jdbc.Driver";
    //数据库连接描述符。
```



```
    String sourceURL = "jdbc:gaussdb://$ip:$port/database?";
    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;
}
```

常用参数详细请见《 开发指南 》中“ 应用程序开发教程>基于JDBC开发>JDBC常用参数参考”。

示例3：使用流式读功能

```
// 认证用的用户名和密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中存放（密码应密文存放，使用时解密），确保安全。
// 本示例以用户名和密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量（环境变量名称请根据自身情况进行设置）EXAMPLE_USERNAME_ENV和EXAMPLE_PASSWORD_ENV。
// 建立连接。
public static Connection getConnection(String username, String passwd) {
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableStreamingQuery=true";
    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;
}

// 执行普通SQL语句，创建t_user表。
public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();
```



```
        //执行普通SQL语句。
        stmt.executeUpdate("DROP TABLE IF EXISTS t_user");
        stmt.executeUpdate("CREATE TABLE t_user(id int, name VARCHAR(20));");
        stmt.close();
    } catch (SQLException e) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

//执行预处理语句，批量插入数据。
public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
        //生成预处理语句。
        pst = conn.prepareStatement("INSERT INTO t_user VALUES (?,?)");
        for (int i = 0; i < 20; i++) {
            //添加参数。
            pst.setInt(1, i + 1);
            pst.setString(2, "name " + (i + 1));
            pst.addBatch();
        }
        //执行批处理。
        pst.executeBatch();
        pst.close();
    } catch (SQLException e) {
        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// 开启流式读，查询t_user表中内容。
public static void StreamingQuery(Connection conn) {
    PreparedStatement pst = null;
    ResultSet resultSet = null;

    try {
        // 查询表t_user中的所有值。
        pst = conn.prepareStatement("SELECT * FROM t_user");
        pst.setFetchSize(Integer.MIN_VALUE);// ((PgStatement)statement).enableStreamingResults(); 两者作用
一致。
        resultSet = pst.executeQuery();
        while (resultSet.next()) {
            System.out.println(resultSet.getInt(1));
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
```



```
        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

public static void main(String[] args) throws Exception {
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");

    Connection conn = getConnection(userName, password);

    CreateTable(conn);

    BatchInsertData(conn);

    StreamingQuery(conn);

    //关闭数据库连接。
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

须知

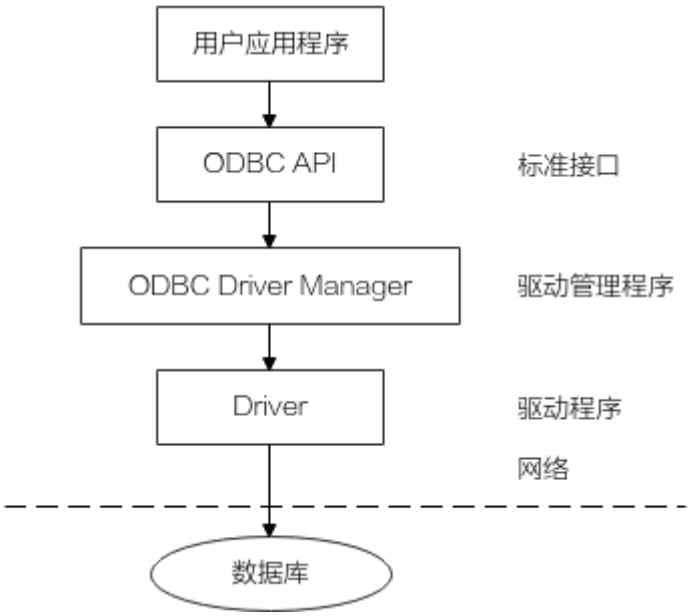
使用流式读功能时，结果集使用完之后，需要执行resultSet.close()或者statement.close()操作，否则当前连接将不可用。

4.2.3 使用 ODBC 连接数据库

ODBC（Open Database Connectivity，开放数据库互连）是由Microsoft公司基于X/OPEN CLI提出的用于访问数据库的应用程序编程接口。应用程序通过ODBC提供的API与数据库进行交互，增强了应用程序的可移植性、扩展性和可维护性。

ODBC的系统结构参见**图1 ODBC系统结构**。

图 4-2 ODBC 系统结构



GaussDB目前在以下环境中提供对ODBC的支持。

表 4-15 ODBC 支持平台

操作系统	平台
EulerOS V2.0SP5	x86_64位
EulerOS V2.0SP9	ARM64位
EulerOS V2.0SP10	x86_64位
EulerOS V2.0SP10	ARM64位
Windows 7	x86_32位
Windows 7	x86_64位
Windows Server 2008	x86_32位
Windows Server 2008	x86_64位
Kylin V10	x86_64位
Kylin V10	ARM64位
UnionTech V20	x86_64位
UnionTech V20	ARM64位
Huawei Cloud EulerOS 2.0	x86_64位
Huawei Cloud EulerOS 2.0	ARM64位

UNIX/Linux系统下的驱动程序管理器主要有unixODBC和iODBC，在这选择驱动管理器unixODBC-2.3.7作为连接数据库的组件。

Windows系统自带ODBC驱动程序管理器，在控制面板->管理工具中可以找到数据源（ODBC）选项。

说明

当前数据库ODBC驱动基于开源版本，对于tinyint、smalldatetime、nvarchar、nvarchar2类型，在获取数据类型的时候，可能会出现不兼容的情况。

ODBC 相关约束说明

- ODBC不支持备机读。
- ODBC不支持自定义类型，不支持在存储过程中使用自定义类型参数。
- ODBC不支持容灾切换。
- 当数据库开启proc_outparam_override参数时，ODBC无法正常调用带有out参数的存储过程。

前提条件

已下载Linux版本的ODBC驱动包和Windows版本的ODBC驱动包。

- Linux版本包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Odbc.tar.gz。Linux环境下，开发应用程序要用到unixODBC提供的头文件（sql.h、sqlext.h等）和库libodbc.so。这些头文件和库可从unixODBC-2.3.7的安装包中获得。
- Windows版本包名为GaussDB-Kernel_数据库版本号_Windows_Odbc_X64.tar.gz（64位）。Windows环境下，开发应用程序用到的相关头文件和库文件都由系统自带。

在 Linux 下使用 ODBC 连接数据库

步骤1 获取unixODBC源码包。

获取参考地址：**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz**

下载后请先按照社区提供的完整性校验算法进行完整性校验。下载**https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5**，查看MD5值，对比MD5值是否与源码包一致。

步骤2 安装unixODBC。如果机器上已经安装了其他版本的unixODBC，可以直接覆盖安装。

以unixODBC-2.3.7版本为例，在客户端执行如下命令安装unixODBC。

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no #如果要在ARM服务器上编译，请追加一个configure参数： --build=aarch64-unknown-linux-gnu
make
#安装可能需要root权限
make install
```


说明

- 目前不支持unixODBC-2.2.1版本。
- 默认安装到“/usr/local”目录下，生成数据源文件到 “/usr/local/etc”目录下，库文件生成在“/usr/local/lib”目录。
- 通过编译带有--enable-fastvalidate=yes选项的unixODBC来获得更高性能。但此选项可能会导致向ODBC API传递无效句柄的应用程序发生故障，而不是返回SQL_INVALID_HANDLE错误。

步骤3 替换客户端GaussDB驱动程序。

将GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Odbc.tar.gz解压。解压后会得到两个文件夹：lib与odbc，在odbc文件夹中还会有一个lib文件夹。将解压后得到的/lib文件夹与/odbc/lib文件夹中的所有动态库都复制到“/usr/local/lib”目录下。

步骤4 配置数据源。

1. 配置ODBC驱动文件。

   在“/usr/local/etc/odbcinst.ini”文件中追加以下内容。

   ```
   [GaussMPP]
   Driver64=/usr/local/lib/gsqlodbcw.so
   setup=/usr/local/lib/gsqlodbcw.so
   ```

   odbcinst.ini文件中的配置参数说明如表4-16所示。

   表 4-16 odbcinst.ini 文件配置参数

参数	描述	示例
[DriverName]	驱动器名称，对应数据源DSN中的驱动名。	[DRIVER_N]
Driver64	驱动动态库的路径。	Driver64=/usr/local/lib/gsqlodbcw.so
setup	驱动安装路径，与Driver64中动态库的路径一致。	setup=/usr/local/lib/gsqlodbcw.so

2. 配置数据源文件。

   在“/usr/local/etc/odbc.ini”文件中追加以下内容。

   ```
   [MPPODBC]
   Driver=GaussMPP
   Servername=127.0.0.1 #数据库Server IP
   Database=db1  #数据库名
   Username=omm  #数据库用户名
   Password=  #数据库用户密码
   Port=8000  #数据库侦听端口
   Sslmode=allow
   ```

   odbc.ini文件配置参数说明如表4 odbc.ini文件配置参数所示。

   表 4-17 odbc.ini 文件配置参数

参数	描述	示例
[DSN]	数据源的名称。	[MPPODBC]

参数	描述	示例
Driver	驱动名，对应odbcinst.ini中的DriverName。	Driver=DRIVER_N
Servername	服务器的IP地址。可配置多个IP地址。支持IPv4和IPv6。	Servername=127.0.0.1
Database	要连接的数据库的名称。	Database=db1
Username	数据库用户名称。	Username=omm
Password	数据库用户密码。 **说明** ODBC驱动本身已经对内存密码进行过清理，以保证用户密码在连接后不会再在内存中保留。 但是如果配置了此参数，由于UnixODBC对数据源文件等进行缓存，可能导致密码长期保留在内存中。 推荐在应用程序连接时，将密码传递给相应API，而非写在数据源配置文件中。同时连接成功后，应当及时清理保存密码的内存段。 **注意** 配置文件中填写密码时，需要遵循http规则： 1. 字符应当采用URL编码规范，如"!"应写作"%21"，"%"应写作"%25"，因此应当特别注意字符。 2. "+"会被替换为空格""。	Password=********
Port	服务器的端口号。	Port=8000
Sslmode	开启SSL模式。	Sslmode=allow

其中关于Sslmode的选项的允许值，具体信息如**表3 Sslmode的可选项及其描述**所示。

表 4-18 Sslmode 的可选项及其描述

Sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	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	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。GaussDB不支持此模式。

步骤5 SSL模式。具体操作请联系数据库管理员。

步骤6 配置数据库服务器。具体操作请联系数据库管理员。

步骤7 在客户端配置环境变量。

```
vim ~/.bashrc
```

在配置文件中追加以下内容。

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

步骤8 执行如下命令使设置生效。

```
source ~/.bashrc
```

步骤9 执行以下命令，开始连接数据库。

isql -v *GaussODBC*

GaussODBC为数据源名称

- 如果显示如下信息，表明配置正确，连接成功。

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
```

- 若显示ERROR信息，则表明配置错误。请检查上述配置是否正确。

----结束

在 Windows 下使用 ODBC 连接数据库

Windows操作系统自带ODBC数据源管理器，无需用户手动安装管理器便可直接进行配置。

步骤1 替换客户端GaussDB驱动程序。

根据需要，将包名为GaussDB-Kernel_数据库版本号_Windows_X64_Odbc.tar.gz的64位驱动或包名为GaussDB-Kernel_数据库版本号_Windows_X86_Odbc.tar.gz的32位驱动解压后，单击gsqlodbc.exe进行驱动安装。

步骤2 打开驱动管理器。

在配置数据源时，请使用ODBC版本对应的ODBC驱动管理器（如果使用64位ODBC驱动，必须要使用64位的ODBC驱动管理器，假设操作系统安装盘符为C盘，如果是其他盘符，请对路径做相应修改）。

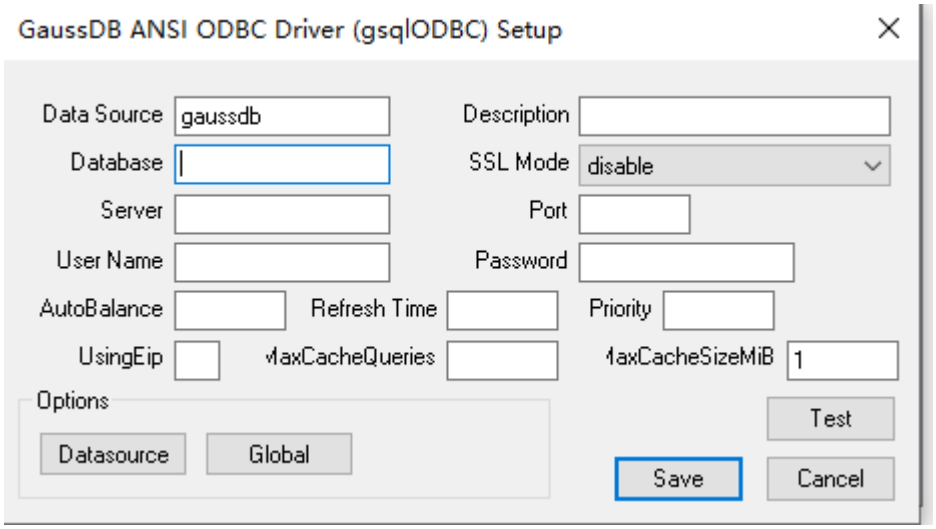
- 如果需要在64位操作系统使用32位ODBC驱动请使用：C:\Windows**SysWOW64**\odbcad32.exe，请勿直接使用“控制面板 > 管理工具 > 数据源(ODBC)”。

  说明

  WOW64的全称是"Windows 32-bit on Windows 64-bit"，C:\Windows\SysWOW64\存放的是64位系统上的32位运行环境。而C:\Windows\System32\存放的是与操作系统一致的运行环境，具体的技术信息请查阅Windows的相关技术文档。
- 32位操作系统请使用：C:\Windows\System32\odbcad32.exe，或者单击“计算机 > 控制面板 > 管理工具 > 数据源(ODBC)”打开驱动管理器。
- 64位操作系统请使用：控制面板 > 管理工具 > 数据源(ODBC) 打开驱动管理。

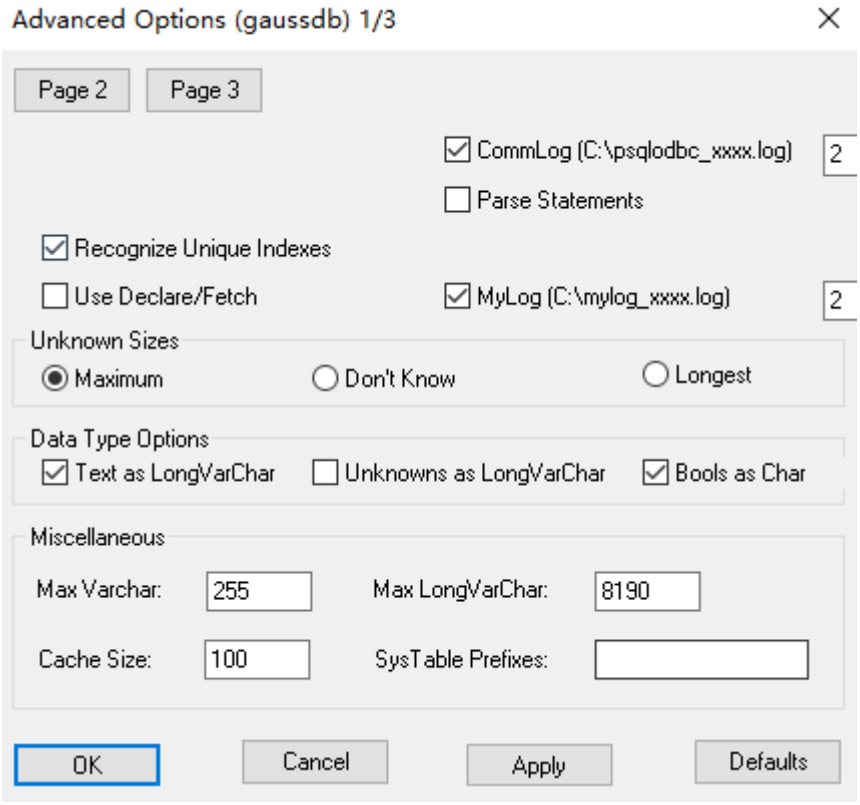
步骤3 配置数据源。

在打开的驱动管理器上，选择“用户DSN > 添加 > GaussDB Unicode”，然后进行配置：



参数说明参考**在Linux下使用ODBC连接数据库**文件参数配置。

其中单击Datasource可以选择配置是否打印日志：



须知

此界面上配置的用户名及密码信息，将会被记录在Windows注册表中，再次连接数据库时不再需要输入认证信息。但是出于安全考虑，建议在单击"Save"按钮保存配置信息前，清空相关敏感信息，在使用ODBC的连接API时，再传入所需的用户名、密码信息。

步骤4 SSL模式。

将3中的设置窗口的“SSL Mode”选项调整至“require”。

表 4-19 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	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	可能	如果数据库支持，那么首选使用SSL安全加密连接，但不验证数据库服务器的真实性。
require	是	必须使用SSL安全连接，但是只做了数据加密，并不验证数据库服务器的真实性。

sslmode	是否会启用SSL加密	描述
verify-ca	是	必须使用SSL安全连接，并且验证数据库是否具有可信证书机构签发的证书。当前windows ODBC不支持cert方式认证。
verify-full	是	必须使用SSL安全连接，在verify-ca的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。当前windows ODBC不支持cert方式认证。

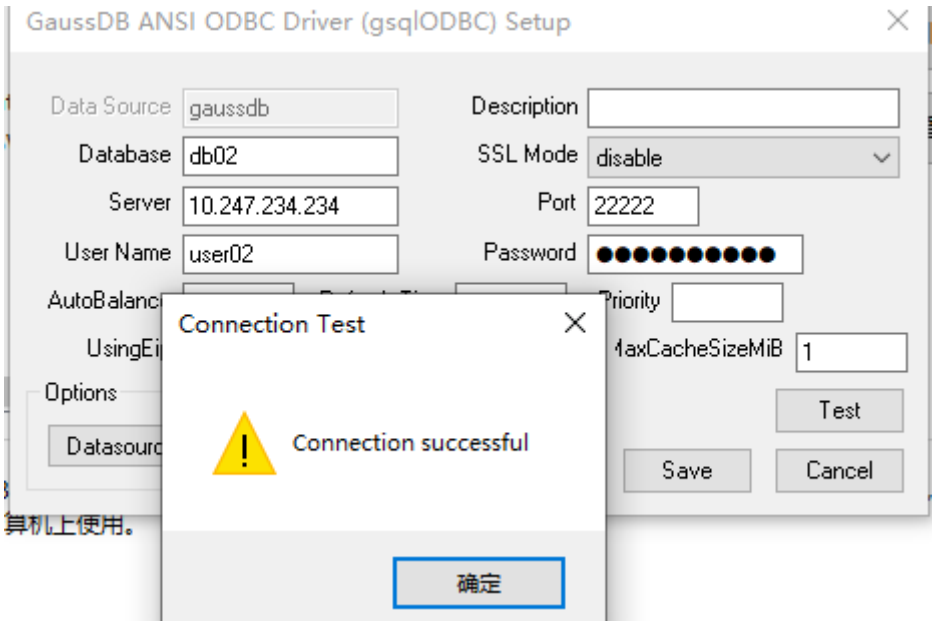
步骤5 配置GaussDB服务器。具体操作请联系管理员。

步骤6 执行如下命令重启数据库实例。

```
gs_om -t stop
gs_om -t start
```

步骤7 单击Test进行测试连接。

- 如果显示如下，则表明配置正确，连接成功。



- 若显示ERROR信息，则表明配置错误。请检查上述配置是否正确。

----结束

4.2.4 使用 libpq 连接数据库

libpq是GaussDB C应用程序接口。libpq是一套允许客户程序向GaussDB服务器服务进程发送查询并且获得查询返回值的库函数。同时也是其他几个GaussDB应用接口下面的引擎，如ODBC等依赖的库文件。本章给出了示例显示如何利用libpq编写代码。

前提条件

本地已安装c语言开发环境。

基于libpq编译开发源程序，主要包括如下步骤：

1. 解压GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Libpq.tar.gz文件，其中include文件夹下的头文件为所需的头文件，lib文件夹中为所需的libpq库文件。

   说明

   除libpq-fe.h外，include文件夹下默认还存在头文件postgres_ext.h、gs_thread.h、gs_threadlocal.h，这三个头文件是libpq-fe.h的依赖文件。

2. 开发源程序testlibpq.c，源码文件中需引用libpq提供的头文件。
   例如： #include <libpq-fe.h>
3. gcc编译libpq源程序，需要通过-I *directory*选项，提供头文件的安装位置（有些时候编译器会查找缺省的目录，因此可以忽略这些选项）。如：
   gcc -I *(头文件所在目录)* -L *(libpq库所在目录)*testlibpq.c -lpq
4. 如果要使用制作文件(makefile)，向CPPFLAGS、LDFLAGS、LIBS变量中增加如下选项：
   CPPFLAGS += -I *(头文件所在目录)*
   LDFLAGS += -L *(libpq库所在目录)*
   LIBS += -lpq
   例如：
   CPPFLAGS += -I$(GAUSSHOME)/include/libpq
   LDFLAGS += -L$(GAUSSHOME)/lib

常用功能示例代码

示例1：

```
/*
 * testlibpq.c
 * 说明： testlibpq.c源程序，提供libpq基本且常见的使用场景。
 * 使用libpq提供的PQconnectdb、PQexec、PQntuples、PQfinish等接口实现数据库建连，执行sql，获取返回结果以及资源清理。
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串 */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *host = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * 用户在命令行上提供了conninfo字符串的值时使用该值
     * 否则环境变量或者所有其它连接参数
     * 都使用缺省值。
     */
```



```
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, host, username, passwd);

    /* 连接数据库 */
    conn = PQconnectdb(conninfo);

    /* 检查后端连接成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * 测试实例涉及游标的使用时候必须使用事务块
     *把全部放在一个  "select * from pg_database"
     * PQexec() 里，过于简单，不推荐使用
     */

    /* 开始一个事务块 */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
     * 在结果不需要的时候PQclear PGresult，以避免内存泄漏
     */
    PQclear(res);

    /*
     * 从系统表 pg_database（数据库的系统目录）里抓取数据
     */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);

    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /* 打印属性名称 */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* 打印行 */
    for (i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++)
```



```
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    PQclear(res);

    /* 关闭入口 ... 不用检查错误 ... */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* 结束事务 */
    res = PQexec(conn, "END");
    PQclear(res);

    /* 关闭数据库连接并清理 */
    PQfinish(conn);

    return 0;
}
```

示例2：

```
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
/* 测试PQconnectStart + loop PQconnectStart接口实现多IP配置下的自动选主 */

static void exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    char *conninfo = "postgresql://10.442.13.173:53600,10.442.13.177:54600/postgres?
user=bot&password=Gaussdba@Mpp&target_session_attrs=read-write";
    //PGconn *conn = PQconnectdb(conninfo);
    PGconn *conn = PQconnectStart(conninfo);
    if (conn == NULL) {
        fprintf(stderr, "Connection initialization failed\n");
        return 1;
    }

    ConnStatusType status;
    int sock;

    while (1) {
        PQconnectPoll(conn);
        status = PQstatus(conn);
        fprintf(stderr, "\n--------conn->status: %d\n", PQstatus(conn));
        if (status == CONNECTION_BAD) {
            fprintf(stderr, "Connection failed\n");
            PQfinish(conn);
            return 1;
        }

        if ((int)status == 0) {
            fprintf(stderr, "Connection established\n");
            break;
        }

        /* 获取底层 socket 描述符，以便进行非阻塞 IO 操作 */
        sock = PQsocket(conn);

        /* 在此处可以进行其他的非阻塞操作，比如等待事件发生 */

        /* 通过 sleep 模拟非阻塞过程中的其他操作 */
        sleep(1);
```



```
    }
    fprintf(stderr, "\n--------conn->status: %d\n", PQstatus(conn));

    PGresult *res;
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s", PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "show transaction_read_only;");
    printf("\n----------trancaction_read_only is %-15s\n", PQgetvalue(res, 0, 0));
    PQclear(res);

    res = PQexec(conn, "select 1;");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "select failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    printf("%-15s\n", PQgetvalue(res, 0, 0));
    PQclear(res);

    PQfinish(conn);
    return 0;
}
```

示例3：

```
/*
 * testlibpq3.c 测试PQprepare
 * PQprepare： 创建一个给定参数的预备语句，用于PQexecPrepared执行预备语句。
 * 在运行这个例子之前,可以参考下面的命令进行建表
 * create table t01(a int, b int);
 * insert into t01 values(1, 23);
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下
可直接赋值字符串 */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /* PQconnectdb连接数据库, 详细的连接信息为connstr */
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
             hostaddr, dbname, port, username, paswswd);
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
```



```
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
    }

    /* 创建表t01 */
    res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01
values(1, 23);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        printf("Command failed: %s.\n", PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }

    /* cmd_sql查询 */
    sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
    /* cmd_sql中$1对应的参数 */
    paramTypes[0] = 23;
    /* PQprepare创建一个给定参数的预备语句 */
    res = PQprepare(conn,
                    "pre_name",
                    cmd_sql,
                    1,
                    paramTypes);
   if( PQresultStatus(res) != PGRES_COMMAND_OK )
    {
        printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }
    PQclear(res);
    paramValues[0] = cid;
    for (k=0; k<2; k++)
    {
        sprintf(cid, "%d", 1);
        paramLengths[0] = 6;
        paramFormats[0] = 0;
        /* 执行预备语句 */
        res = PQexecPrepared(conn,
                             "pre_name",
                             1,
                             paramValues,
                             paramLengths,
                             paramFormats,
                             0);
        if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
        {
            printf("%s\n",PQerrorMessage(conn));
            PQclear(res);
            PQfinish(conn);
            return -1;
        }
        cnt = PQntuples(res);
        printf("return %d rows\n", cnt);
        for (i=0; i<cnt; i++)
        {
            printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
        }
        PQclear(res);
    }
    /* 执行结束，关闭连接 */
    PQfinish(conn);
```



```
    return 0;
}
```

示例4：

```
/*
 * testlibpq4.c
 * 测试PQexecParams
 * PQexecParams： 执行一个绑定参数的命令，并以二进制格式请求查询结果。
 * 在运行这个例子之前，用下面的命令填充一个数据库
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
 * INSERT INTO test1 values (2, 'ho there');
 *
 * 期望的输出如下
 *
 *
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * 这个函数打印查询结果，这些结果是二进制格式，从上面的
 * 注释里面创建的表中抓取出来的
 */
static void
show_binary_results(PGresult *res)
{
    int         i;
    int         i_fnum,
                t_fnum;

    /* 使用 PQfnumber 来避免对结果中的字段顺序进行假设 */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        int         ival;

        /* 获取字段值（忽略可能为空的可能） */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * INT4 的二进制表现形式是网络字节序
         * 建议转换成本地字节序
         */
```



```
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * TEXT 的二进制表现形式是文本，因此libpq能够给它附加一个字节零
         * 把它看做 C 字符串
         *
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    /* 此处user、passwd等变量应从环境变量或配置文件读取，环境变量需用户自己按需配置；非环境变量情况下可直接赋值字符串 */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
    int         paramFormats[1];
    uint32_t    binaryIntVal;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * 如果用户在命令行上提供了参数，
     * 那么使用该值为conninfo 字符串；否则
     * 使用环境变量或者缺省值。
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s password=%s",
            dbname, port, hostaddr, username, passwd);

    /* 和数据库建立连接 */
    conn = PQconnectdb(conninfo);

    /* 检查与服务器的连接是否成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);
```



```
    res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    /* 把整数值 "2" 转换成网络字节序 */
    binaryIntVal = htonl((uint32_t) 2);

    /* 为 PQexecParams 设置参数数组 */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1;        /* 二进制 */

    /* PQexecParams执行一个绑定参数的命令 */
    res = PQexecParams(conn,
                       "SELECT * FROM test1 WHERE i = $1::int4",
                       1,       /* 一个参数 */
                       NULL,    /* 让后端推导参数类型 */
                       paramValues,
                       paramLengths,
                       paramFormats,
                       1);      /* 要求二进制结果 */

    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    /* 显示二进制结果 */
    show_binary_results(res);

    PQclear(res);

    /* 关闭与数据库的连接并清理 */
    PQfinish(conn);

    return 0;
}
```

4.2.5 使用 Psycopg 连接数据库

Psycopg是一种用于执行SQL语句的PythonAPI，可以为GaussDB数据库提供统一访问接口，应用程序可基于它进行数据操作。Psycopg2是对libpq的封装，主要使用C语言实现，既高效又安全。它具有客户端游标和服务器端游标、异步通信和通知、支持“COPY TO/COPY FROM”功能。支持多种类型Python开箱即用，适配GaussDB数据类型。通过灵活的对象适配系统，可以扩展和定制适配。Psycopg2兼容Unicode。

GaussDB数据库提供了对Psycopg2特性的支持，并且支持psycopg2通过SSL模式连接。

表 4-20 Psycopg 支持平台

操作系统	平台	Python版本
EulerOS V2.0SP5	• ARM64位 • x86_64位	3.8.5

操作系统	平台	Python版本
EulerOS V2.0SP9	• ARM64位 • x86_64位	3.7.4
EulerOS V2.0SP10、Kylin v10、UnionTech20	• ARM64位 • x86_64位	3.7.9
EulerOS V2.0SP11、Suse 12.5	• ARM64位 • x86_64位	3.9.11
Huawei Cloud EulerOS 2.0	• ARM64位 • x86_64位	3.9.9

须知

psycopg2在编译过程中，会链接（link）GaussDB的openssl，GaussDB的openssl与操作系统自带的openssl可能不兼容。如果遇到不兼容现象，例如提示"version 'OPENSSL_1_1_1f' not found"，请使用环境变量LD_LIBRARY_PATH进行隔离，以避免混用操作系统自带的openssl与GaussDB依赖的openssl。

例如，在执行某个调用psycopg2的应用软件client.py时，将环境变量显性赋予该应用软件：

```
export LD_LIBRARY_PATH=/path/to/gaussdb/libs:$LD_LIBRARY_PATH python client.py
```

其中，/path/to/pyscopg2/lib 表示GaussDB依赖的openssl库所在目录，需根据文件实际存储路径修改。

前提条件

本地已安装python语言运行环境。

连接数据库

步骤1 准备相关驱动和依赖库。可以从发布包中获取，包名为GaussDB-Kernel_数据库版本号_操作系统版本号_64bit_Python.tar.gz。

解压后有两个文件夹：

- psycopg2：psycopg2库文件。
- lib：lib库文件。

步骤2 加载驱动。

- 在使用驱动之前，需要做如下操作：
  a. 先解压版本对应驱动包。
  ```
  tar zxvf xxxx-Python.tar.gz
  ```
  b. 使用root用户将psycopg2复制到python安装目录下的site-packages文件夹下。
  ```
  su root
  cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
  ```

c. 修改psycopg2目录权限为755。
```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```
d. 将psycopg2目录添加到环境变量$PYTHONPATH，并使之生效。
```
export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
```
e. 对于非数据库用户，需要将解压后的lib目录，配置在LD_LIBRARY_PATH中。
```
export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
```

- 在创建数据库连接之前，需要先加载如下数据库驱动程序：
```
import  psycopg2
```

步骤3 连接数据库。

非SSL方式连接数据库：

1. 使用psycopg2.connect函数获得connection对象。
2. 使用connection对象创建cursor对象。

SSL方式连接数据库：

用户通过psycopy2连接GaussDB服务器时，可以通过开启SSL加密客户端和服务器之间的通讯。在使用SSL时，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参见Openssl相关文档和命令。

1. 使用*.ini文件（python的configparser包可以解析这种类型的配置文件）保存数据库连接的配置信息。
2. 在连接选项中添加SSL连接相关参数：sslmode、sslcert、sslkey、sslrootcert。
   a. sslmode：可选项见表4-21。
   b. sslcert：客户端证书路径。
   c. sslkey：客户端密钥路径。
   d. sslrootcert：根证书路径。
3. 使用psycopg2.connect函数获得connection对象。
4. 使用connection对象创建cursor对象。

注意

使用SSL安全连接数据库，需保证所使用的python解释器为生成动态链接库（.so）文件的方式编译，可通过如下步骤确认python解释器的连接方式。

1. 在python解释器命令行中输入import ssl，导入SSL。
2. 执行ps ux查询python解释器运行的pid（假设pid为******）。
3. 在shell命令行中执行pmap -p ****** | grep ssl，查看返回结果中是否包含libssl.so的相关路径。如果有，则python解释器为动态链接方式编译。

表 4-21 sslmode 的可选项及其描述

sslmode	是否会启用SSL加密	描述
disable	否	不使用SSL安全连接。
allow	可能	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。

sslmode	是否会启用SSL加密	描述
prefer	可能	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	是	必须使用SSL安全连接，但是仅进行数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用SSL安全连接，并且校验服务端CA有效性。
verify-full	是	必须使用SSL安全连接，目前GaussDB暂不支持。

----结束

4.2.6 使用 Hibernate 连接数据库

Hibernate是一种Java语言下的对象关系映射（ORM）解决方案。它提供了一个使用方便的持久化框架，可以自动将数据库表和Java对象之间建立映射关系，使得开发者可以使用面向对象的方式来操作数据库。使用Hibernate，开发者就不需要手动编写大量的SQL和JDBC代码，从而大大简化了数据访问层的开发工作。

本章节指导用户使用Hibernate连接GaussDB进行建表，修改表以及增删改查操作示例。

配置 pom 依赖

```
<dependency>
   <groupId>com.huaweicloud.gaussdb</groupId>
   <artifactId>opengaussjdbc</artifactId>
   <version>503.2.T35</version>
</dependency>
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>5.3.7.Final</version>
</dependency>
```

⚠ 注意

配置pom依赖时，需要提前配置好maven环境。

配置 hibernate.cfg.xml 资源文件

```
<?xml version="1.0" encoding="utf-8"?>
      <!DOCTYPE hibernate-configuration PUBLIC
            "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
            "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <!--GaussDB 连接信息-->
      <property name="connection.driver_class">com.huawei.opengauss.jdbc.Driver</property>
      <property name="connection.url">jdbc:opengauss://***.***.***.***（需要替换为数据库ip）:20000（需要替换为数据库端口）/test?currentSchema=test（test需要替换为对应的database和schema）</property>
      <property name="connection.username">***（需要替换为正确的用户名）</property>
```

```
        <property name="connection.password">******（需要替换为正确的密码）</property>

        <!-- 以下为可选配置 -->

        <!--是否支持方言 -->
        <!--在 postgresql 兼容模式下，必须有如下配置-->
<property name="dialect">org.hibernate.dialect.PostgreSQL82Dialect</property>
        <!--执行CURD时是否打印sql 语句  -->
        <property name="show_sql">true</property>

        <!--自动建表(修改表)-->
        <!-- <property name="hbm2ddl.auto">update</property>-->
        <!-- <property name="hbm2ddl.auto">create</property>-->

        <!-- 资源注册 (实体类映射文件)-->
        <mapping resource="./student.xml"/>
    </session-factory>
</hibernate-configuration>
```

准备实体类和实体类映射表 xml 文件

以Student类以及student.xml文件为例，文件路径为com.howtodoinjava.hibernate.test.dto。

1. 实体类。

```
public class Student implements Serializable {
    int id;
    String name;
//  String age;
    // 此处省略构造器，get，set方法
}
```

2. student.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        '-//Hibernate/Hibernate Mapping DTD 3.0//EN'
        'http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd'>
<hibernate-mapping>
    <!-- 类与表的映射制作在class元素上 -->
    <!-- name:全路径类名 -->
    <!-- table:表名 -->
    <class name="com.howtodoinjava.hibernate.test.dto.Student" table="student">
        <!-- 主键的映射制作在 id 元素上 -->
        <!-- name:对象中用于作为主键的属性名 -->
        <!-- column:表中主键字段名 -->
        <!-- 如果 name 与 column 值相同，可以省略 column -->
        <id name="id" column="id">
            <!-- 将 generator 元素 class 属性设置为 "assigned" 手动生成,必须给 id -->
            <generator class="assigned"  />

            <!--**要注意 Hibernate 主键生成策略**-->

        </id>
        <!-- 属性与字段的映射制作在 property 元素上 -->
        <!-- name:类中的属性名 -->
        <!-- column:表中的字段名 -->
        <!-- 如果 name 与 column 值相同，可以省略 column -->
        <property name="name" />
        <!--此处同实体类-->
        <!--<property name="age"/>-->
    </class>
</hibernate-mapping>
```

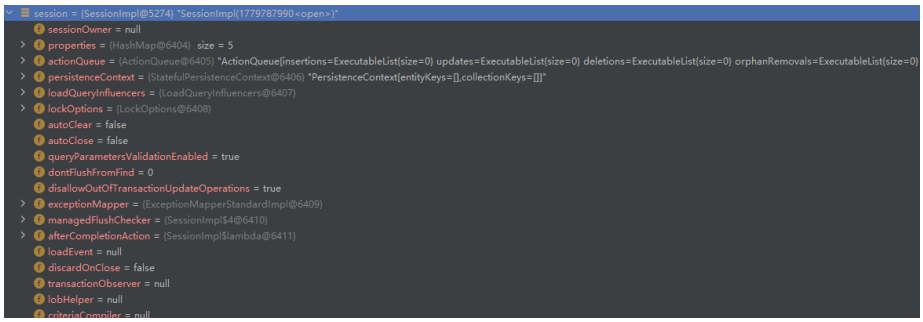
功能测试示例

1. 测试连接功能。

a. 测试方法：

```
@Test
    public void testConnection() {
        // 加载配置信息
        Configuration conf = new Configuration().configure();
        // 基于配置信息,创建 SessionFactory 对象
        SessionFactory sessionFactory = conf.buildSessionFactory();
        // 打开一个与数据库相关的 session 对象
        Session session = sessionFactory.openSession();
        System.out.println(session);
    }
```

b. 打断点可见成功建立连接：



2. 自动建表。

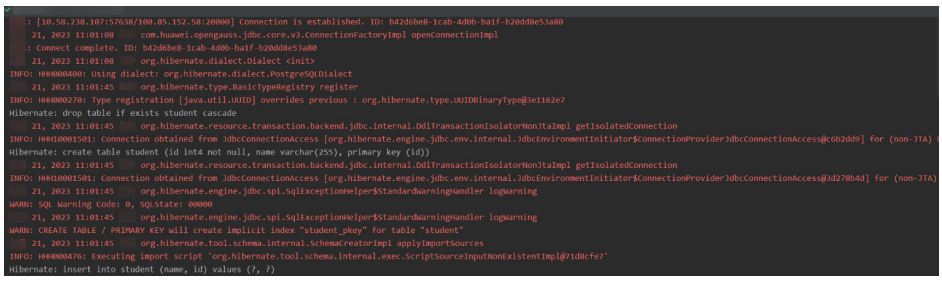
a. 将配置文件本行注释打开：

```
<property name="hbm2ddl.auto">create</property>
```

b. 将数据库中 student 表删除后，执行如下测试方法：

```
@Test
    public void testCreateTableAndInsertData() {
        // 创建要测试的对象
        Student student = new Student();
        student.setId(16);
        student.setName("xiaoming");
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // 通过 session 保存数据
        session.save(student);
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

c. 查看控制台打印出来的所执行 sql 语句以及 GaussDB 数据库中的结果：




```
test=> select * from test.student;
 id |   name
----+----------
 16 | xiaoming
(1 row)
```


成功创建student表并插入字段（ id = 16, name = "xiaoming" ）。

3. 修改表（增加字段）。

   a. 首先修改配置文件，需要配置需要修改表所在的schema路径在url中。

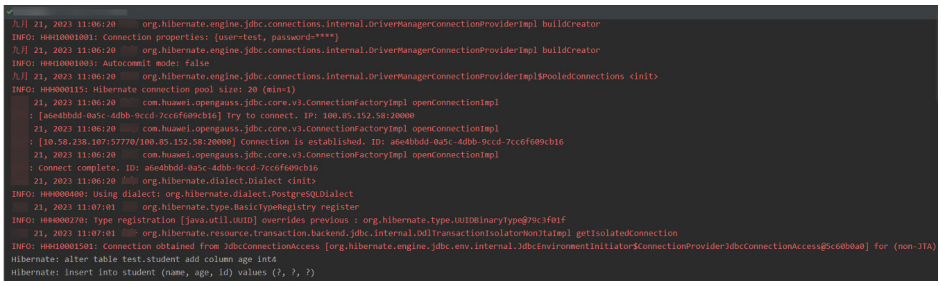
   如本用例中，student 表在名为 test 的 databse 下的名为 test 的 schema 下，即该表路径为test.test.student：

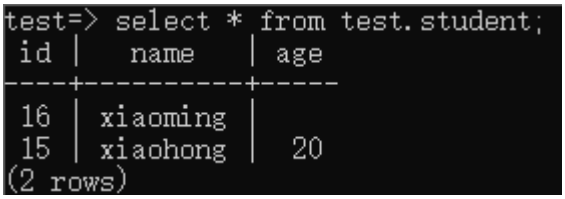
   ```
   <property name="connection.url">jdbc:opengauss://xxx.xxx.xxx.xxx（需要替换为数据库ip）:20000（需要替换为数据库端口）/test?currentSchema=test（test需要替换为对应的database和schema）
   </property>
   <!-- 打开update注释 -->
   <property name="hbm2ddl.auto">update</property>
   ```

   b. 将实体类和xml文件中关于age属性的注释打开，执行如下方法：

   ```
   @Test
       public void testAlterTable() {
           // 创建要测试的对象
           Student student = new Student();
           student.setId(15);
           student.setName("xiaohong");
           student.setAge("20");
           // 开启事务,基于 session 得到
           Configuration conf = new Configuration().configure();
           SessionFactory sessionFactory = conf.buildSessionFactory();
           Session session = sessionFactory.openSession();
           Transaction transaction = session.beginTransaction();
           // 通过 session 保存数据
           session.save(student);
           // 提交事务
           transaction.commit();
           // 操作完毕,关闭 session 连接对象
           session.close();
       }
   ```

   c. 查看控制台打印出来的所执行 sql 语句以及 GaussDB 数据库中的结果：

   在原有表的基础上，框架根据实体类及xml文件，自动对表进行了新增字段处理。

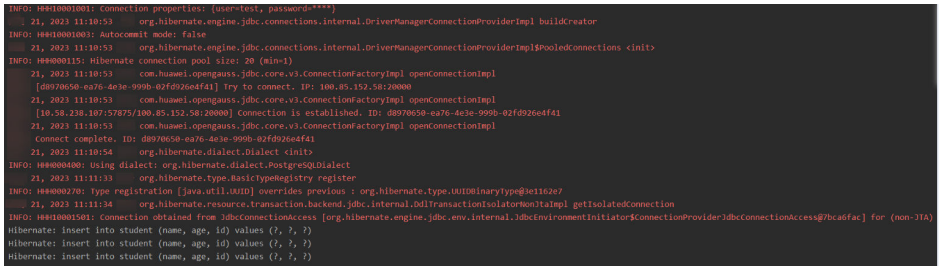
4. 持久化数据。

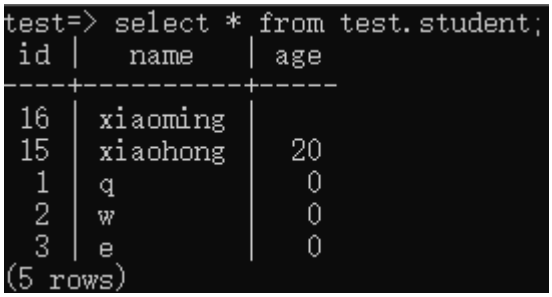
   a. 测试方法：

   ```
   @Test
       public void testInsert() {
           Student s1 = new Student(1,"q");
           Student s2 = new Student(2,"w");
           Student s3 = new Student(3,"e");
           ArrayList<Student> students = new ArrayList<>();
           students.add(s1);
           students.add(s2);
   ```

```
        students.add(s3);
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // 通过 session 保存数据
        for (Student student : students) {
            session.save(student);
        }
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

b. 执行结果如下：





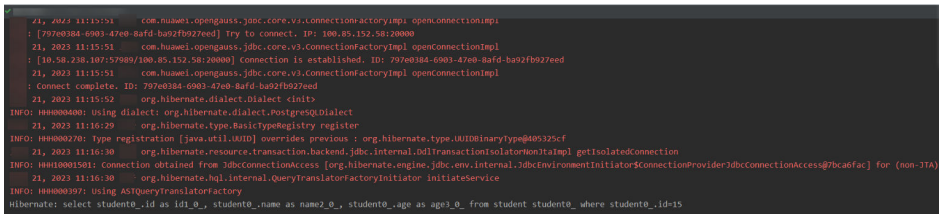
成功插入三行数据。

5. HQL模式查询。

a. 测试方法：

```
@Test
    public void testHQL() {
        //HQL模式
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        // 创建会话
        Session session = sessionFactory.openSession();
        // 开始事务
        Transaction tx = session.beginTransaction();

        // 创建HQL查询
        String hql = "FROM Student S WHERE S.id = 15";
        Query query = session.createQuery(hql);

        // 执行查询并获取结果
        List results = query.list();

        // 提交事务
        tx.commit();

        // 关闭会话
        session.close();
    }
```

b. 执行结果如下：



6. SQL模式查询。
   a. 测试方法：

```
@Test
    public void testQuery() {
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // SQL模式
        List<Student> students = session.createSQLQuery("select * from test.student where id =
1").addEntity(Student.class).list();
        for (int i = 0; i < students.size(); i++) {
            System.out.println(students.get(i));
        }
        students.get(0).setAge("20");
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

   b. 执行结果如下：

```
test=> select * from test.student;
 id |   name    | age
----+-----------+-----
 16 | xiaoming  |
 15 | xiaohong  |  20
  1 | q         |  20
  2 | w         |   0
  3 | e         |   0
(5 rows)
```

   成功查询到id=1的学生数据并修改其age字段为20。

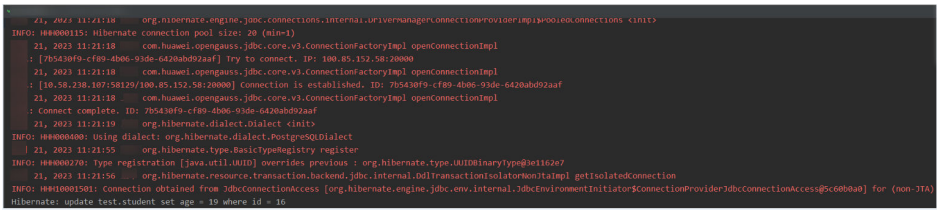
7. 修改操作。
   - 测试方法：

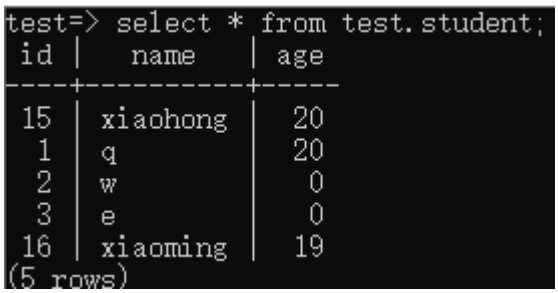
```
@Test
    public void testUpdate() {
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // SQL模式
        session.createSQLQuery("update test.student set age = 19 where id =
16").executeUpdate();
        // 提交事务
```



```
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

- 执行结果如下：





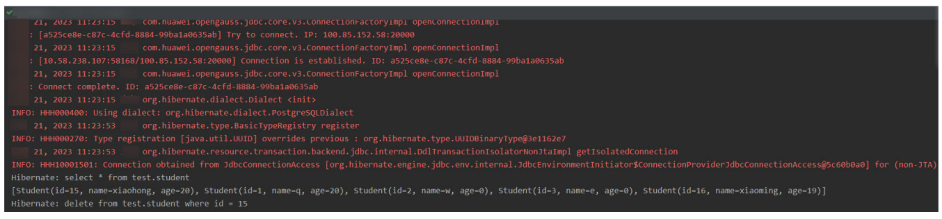
成功将id=16的学生的age字段修改为19。

8. 删除操作。

a. 测试方法：

```
@Test
    public void testDelete() {
        // 开启事务,基于 session 得到
        Configuration conf = new Configuration().configure();
        SessionFactory sessionFactory = conf.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        // SQL模式
        List<Student> students = session.createSQLQuery("select * from
test.student").addEntity(Student.class).list();
        System.out.println(students);
        session.createSQLQuery("delete from test.student where id = " +
students.get(0).getId()).executeUpdate();
        // 提交事务
        transaction.commit();
        // 操作完毕,关闭 session 连接对象
        session.close();
    }
```

b. 执行结果如下：




```
test=> select * from test.student;
 id |   name    | age
----+-----------+-----
  1 | q         |  20
  2 | w         |   0
  3 | e         |   0
 16 | xiaoming  |  19
(4 rows)
```

student表中id=15的字段已被删除。

4.2.7 使用 MyBatis 连接数据库

MyBatis是一个优秀的持久层框架，它支持定制化SQL、存储过程以及高级映射，其避免了几乎所有的JDBC代码和手动设置参数以及获取结果集。MyBatis可以使用简单的XML或注解进行配置和原始映射，将接口和Java的POJOs（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

本章节指导用户使用MyBatis连接GaussDB示例的配置依赖和配置文件。

配置 pom 依赖

```
<dependency>
    <groupId>com.huaweicloud.gaussdb</groupId>
    <artifactId>opengaussjdbc</artifactId>
    <version>503.2.T35</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.6</version>
</dependency>
```

⚠ 注意

配置pom依赖时，需要提前配置好maven环境。

配置文件

配置mybatis-config.xml资源文件。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<!-- 配置文件的根元素 -->
<configuration>
    <!--配置全局属性-->
    <settings>
        <!--使用jdbc的getGeneratedKeys获取数据库自增主键值-->
        <setting name="useGeneratedKeys" value="true"/>
        <!--使用列标签替换列别名 默认为true-->
        <setting name="useColumnLabel" value="true" />
        <!--开启驼峰式命名转换：Table{create_time} -> Entity{createTime}-->
        <setting name="mapUnderscoreToCamelCase" value="true" />
        <setting name="logImpl" value="STDOUT_LOGGING" />
    </settings>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
```

```
        <dataSource type="POOLED">
            <property name="driver" value="com.huawei.opengauss.jdbc.Driver"/>
            <property name="url" value="jdbc:opengauss://***.***.***.***（需要替换为数据库ip）:20000（需要替换为数据库端口）/test?（test需要替换为对应的database）connectionTimeout=10"/>
            <property name="username" value="***（需要替换为正确的用户名）"/>
            <property name="password" value="*******（需要替换为正确的密码）"/>
        </dataSource>
    </environment>
  </environments>
  <!--注册 mapper（mapper.xml 所在地址）-->
  <mappers>
    <mapper resource="mapper/StudentDaoMapper.xml"></mapper>
  </mappers>
</configuration>
```

示例

1. 测试实体类StudentEntity.java（com.huawei.entity路径下）：

```
public class StudentEntity {
    Integer id;
    String name;
}
```

2. 实体类对应的StudentDaoMapper.xml文件（resources.mapper路径下）：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="StudentMapper">
    <!-- 根据主键查询-->
    <select id="getList" resultType="com.huawei.entity.StudentEntity" >
        select * from student;
    </select>
</mapper>
```

3. 查询表测试方法：

```
@Test
public void mainTest() throws IOException {
    // 1.读取mybatis的核心配置文件(mybatis-config.xml)
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
    // 2.通过配置信息获取一个SqlSessionFactory工厂对象
    SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
    // 3.通过工厂获取一个SqlSession对象
    SqlSession session = fac.openSession();
    // 4.通过namespace+id找到要执行的sql语句并执行sql语句
    List<StudentEntity> list = session.selectList("StudentMapper.getList");
    // 5.输出结果
    list.forEach(i -> {
        System.out.println(i.toString());
    });
}
```

4. 查询结果日志：

```
Setting autocommit to false on JDBC Connection [com.huawei.opengauss.jdbc.jdbc.PgConnection@47caedad]
==>  Preparing: select * from student;
==> Parameters:
<==    Columns: id, name
<==        Row: 1, test
<==      Total: 1
StudentEntity(id=1, name=test)
```


> **⚠ 注意**
>
> 目前MybatisPlus插件中PaginationInnerInterceptor未对GaussDB驱动做适配。创建PaginationInnerInterceptor对象时指定DbType为POSTGRE_SQL即可解决。eg: PaginationInnerInterceptor innerInterceptor = new PaginationInnerInterceptor(DbType.POSTGRE_SQL)。

4.2.8 使用 JayDebeApi 连接数据库

JayDebeApi是一个Python模块，其提供了一种方便、高效的方式，让Python开发者能够利用Java的JDBC驱动程序来连接和操作各种数据库。

本章节指导用户如何使用JayDebeApi连接GaussDB数据库。

环境配置

1. 配置GaussDB开发环境。

   准备好基本的 GaussDB 开发环境，获取数据库连接参数，例如：

   ```
   gsql -h ***.***.***.*** -p 20000 -U *** -W ****** -d test
   ```

   参数说明：

   -h：GaussDB服务器IP。

   -p：GaussDB连接端口。

   -U：连接用户名。

   -W：用户密码。

   -d：连接的DATABASE名。

2. 安装JayDeBeApi驱动。

   a. 在计算机上安装Java JDK 8版本以及Python3版本，可以通过如下命令确认版本：

   ```
   java -version
   python --version
   pip --version
   ```

   b. 如果服务器能连接到Python源（Python的软件包索引PyPI），可以通过pip命令安装JayDeBeApi：

   ```
   pip install jaydebeapi
   ```

   如果不能，可以通过下载离线安装包然后进行本地安装的方式安装JayDeBeApi。

3. 获取GaussDB驱动程序安装包。

   根据不同版本的实例，下载不同版本的发布包，如表4-22所示。

   表 4-22 驱动包下载列表

版本	下载地址
3.x	驱动包 驱动包校验包
2.x	驱动包 驱动包校验包

为了防止软件包在传递过程或存储期间被恶意篡改，下载软件包时需下载对应的校验包对软件包进行校验，校验方法如下：

a. 上传软件包和软件包校验包到虚拟机（Linux操作系统）的同一目录下。

b. 执行如下命令，校验软件包完整性。

cat *GaussDB_driver.zip.sha256* **| sha256sum --check**

如果回显OK，则校验通过。

```
GaussDB_driver.zip: OK
```

使用示例

1. 新建脚本文件。

   – 新建一个test_jaydebeapi.py文件，写入代码如下：

```
#!/usr/bin/env python3.x
# -*- coding: UTF-8 -*-
encoding = "utf8"
import jaydebeapi

def test_jaydebeapi():
    #需要配置的参数
    url = 'jdbc:opengauss://***.***.***.***:20000/test'
    user = '***'
    password = '******'
    driver = 'com.huawei.opengauss.jdbc.Driver'
    jarFile = './opengaussjdbc.jar'

    conn = jaydebeapi.connect(driver, url, [user, password], jarFile)
    cur = conn.cursor()

    #创建表students
    sql = 'create table students (id int, name varchar(20))'
    cur.execute(sql)

    #往students表中插入三组数据
    sql = "insert into students values(1,'xiaoming'),(2,'xiaohong'),(3,'xiaolan')"
    cur.execute(sql)

    #查询students表中的所有数据
    sql = 'select * from students'
    cur.execute(sql)
    ans = cur.fetchall()
    print(ans)

    #更新students表中的数据
    sql = 'update students set name = \'xiaolv\' where id = 1'
    cur.execute(sql)

    #再次查询students表中的所有数据
    sql = 'select * from students'
    cur.execute(sql)
    ans = cur.fetchall()
    print(ans)

    #删除表students
    sql = 'drop table students'
    cur.execute(sql)

    cur.close()
    conn.close()

test_jaydebeapi()
```

   – 配置代码中的参数：

```
#连接url，设置数据库服务器IP、端口、数据库名
url = 'jdbc:opengauss://***.***.***.***:20000/test'
```



```
#设置用户名
user = '***'
#设置密码
password = '******'
#JDBC驱动类路径
driver = 'com.huawei.opengauss.jdbc.Driver'
#JDBC驱动jar包路径（默认放在与test_jaydebeapi.py文件同目录下）
jarFile = './opengaussjdbc.jar'
```

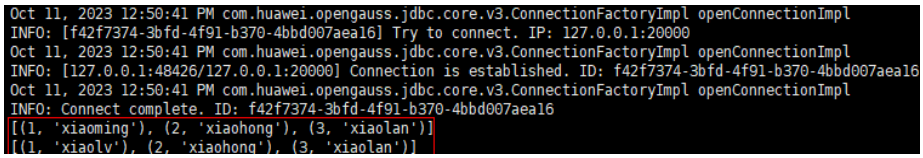
2. 执行程序。

   使用以下命令执行test_jaydebeapi.py文件：

   ```
   python ./test_jaydebeapi.py
   ```

3. 预期结果。

   成功连接GaussDB数据库，并返回两次查询结果，如下所示：



5 示例：使用 DAS 连接实例并执行 SQL

本章指导用户创建并试用最小规格的GaussDB按需计费实例，并执行基本的SQL语法。

- 购买实例
- 使用DAS连接实例
- SQL使用入门

购买实例

步骤1 登录华为云控制台。

步骤2 单击管理控制台左上角的，选择区域。

步骤3 单击左侧的服务列表图标，选择“数据库 > 云数据库 GaussDB”。

步骤4 在左侧导航栏选择GaussDB > 实例管理。

步骤5 单击“购买数据库实例”。

步骤6 配置实例名称、计费信息等实例基本信息。

图 5-1 计费模式和基本信息

步骤7 选择实例规格。

步骤8 选择实例所属的VPC和安全组、配置数据库端口。

建议选择子网数量充足的虚拟私有云，确保实例创建成功。

步骤9 配置实例密码、参数模板、企业项目等信息。

步骤10 单击“立即购买”，核对实例信息，单击“提交”。

步骤11 返回实例列表。

当实例运行状态为“正常”时，表示实例创建完成。

----结束

使用 DAS 连接实例

步骤1 登录华为云控制台。

步骤2 单击管理控制台左上角的 ，选择区域。

步骤3 单击左侧的服务列表图标，选择“数据库 > 数据管理服务 DAS”。

步骤4 在数据管理服务DAS左侧导航栏，单击“开发工具”，进入开发工具数据库登录列表页面。

步骤5 单击“新增数据库登录”，打开新增数据库登录窗口。

须知

创建数据库后，会默认新增root用户的登录，不需要再次手动创建。

步骤6 “数据库引擎”选择“GaussDB”、“数据库来源”保持默认、目标实例，填写数据库名称、登录用户名、密码以及描述（非必填项）信息。

建议开启定时采集、SQL执行记录功能。

如果提示已有连接，可以跳过创建连接步骤，直接执行**步骤9**。

步骤7 可根据需要选择“测试连接”（必选操作步骤）。

如测试连接成功，将提示“连接成功”，可继续新增操作。如测试连接失败，将提示连接失败原因，需根据提示信息进行修改，以便新增数据库登录成功。

步骤8 设置完登录信息，单击“立即新增”。

步骤9 新增完成后，单击新增登录的“登录”，登录当前数据库。

步骤10 进入SQL查询页面。

----结束

SQL 使用入门

步骤1 创建数据库用户。

默认只有集群安装时创建的管理员用户可以访问初始数据库，还可以创建其他数据库用户账号。

CREATE USER *joe* **WITH PASSWORD** "xxxxxxxxx";

当结果显示为如下信息，则表示创建成功。

如上创建了一个用户名为joe，密码为xxxxxxx的用户。

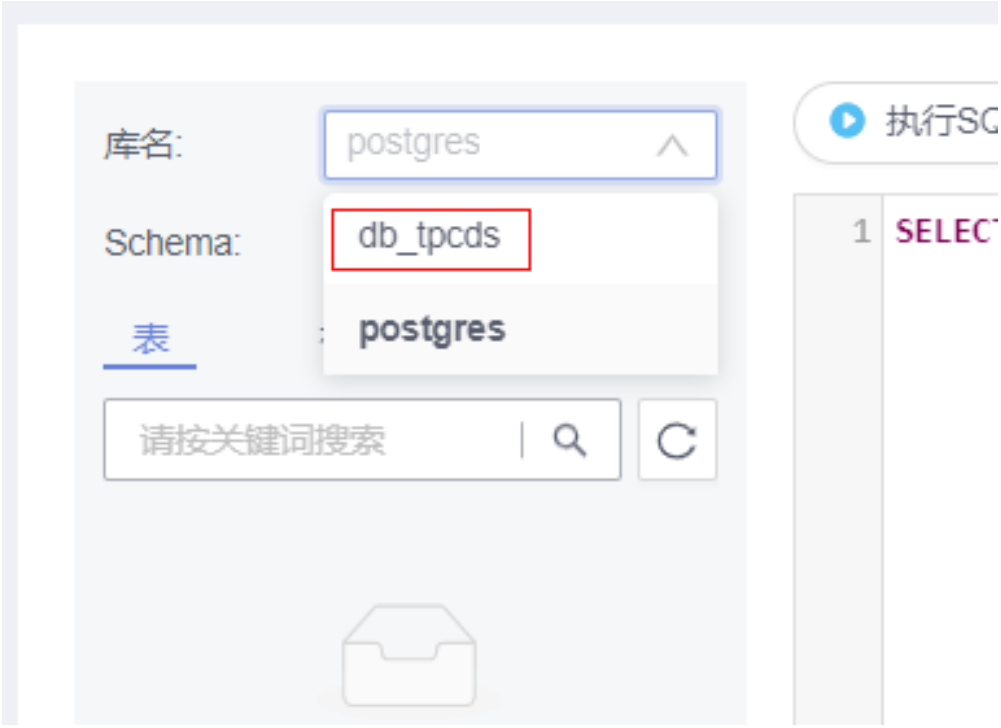
引申信息：关于数据库用户的更多信息请参考**用户及权限**。

步骤2 创建数据库。

CREATE DATABASE *db_tpcds*;

当结果显示为如下信息，则表示创建成功。

创建完db_tpcds数据库后，可以在左上方切换到新创建的库中。



步骤3 创建表。

- 执行如下命令来创建一个schema。
  CREATE SCHEMA *myschema*;
- 创建一个名称为mytable，只有一列的表。字段名为firstcol，字段类型为integer。
  CREATE TABLE myschema.mytable *(firstcol int)*;
- 向表中插入数据：
  INSERT INTO myschema.mytable values (100);
- 查看表中数据：
  SELECT * FROM myschema.mytable;

引申信息：

- 默认情况下，新的数据库对象是创建在“$user”模式下的，例如刚刚新建的表。关于模式的更多信息请参考**创建和管理schema**。
- 关于创建表的更多信息请参见**创建和管理表**。
- 除了创建的表以外，数据库还包含很多系统表。这些系统表包含集群安装信息以及GaussDB上运行的各种查询和进程的信息。可以通过查询系统表来收集有关数据库的信息。请参见**查看系统表**。

步骤4 在db_tpcds库中，root用户下执行如下语句，将新创建的库db_tpcds的所有权限赋予新用户joe。

GRANT ALL ON DATABASE db_tpcds **TO** joe;

GRANT USAGE ON schema myschema **TO** joe;

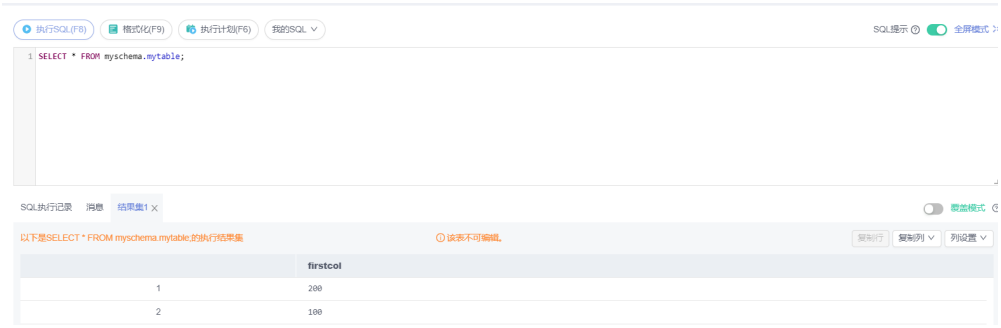
GRANT ALL ON TABLE myschema.mytable **TO** joe;

步骤5 新增joe用户登录数据库db_tpcds。

步骤6 登录之后，在表中插入数据并验证。

INSERT INTO myschema.mytable values (200);

SELECT * FROM myschema.mytable;



----结束

