

gStore0.9.1版本使用手册

1.更新日志

1.1 版本更新记录

gStore 0.9.1 版本

- 更新时间：2021年11月23日
- 更新功能
 - 将gStore内核解析与执行进行分离，通过join order等技术进一步提升查询性能，在复杂查询中性能可以提升40%以上；
 - 重写gStore的http service组件ghttp，并增加了用户权限、心跳检测、批量导入、批量删除等功能，并编写了规范的ghttp api接口文档(见接口列表)，进一步丰富ghttp的功能，提升ghttp的健壮性；
 - 新增了Personalized PageRank (PPR) 自定义函数，Personalized PageRank自定义函数可用于计算实体间的相关度，从而在图中找出影响度最大的节点；
 - 新增Filter语句中对算术及逻辑运算的支持，如算术运算（如 $?x + ?y = 5$ ）；逻辑运算（如 $?x + ?y = 5 \ \&\& \ ?y > 0$ ）等；
 - 增加事务处理功能，支持begin/tquery/commit/rollback等事务操作；
 - 新增gServer组件，实现Socket API双向通信，用户除了通过ghttp组件远程访问gStore之外，还可以通过gServer组件远程访问gStore；
 - 规划本地操作指令格式，引入--help指令，用户可以查看各功能的详细指令格式，如bin/gbuild -h/--help可以详细查看gbuild命令的指令格式；
 - 修复一系列bug。

gStore 0.9.0版本

- 更新时间：： 2021-02-10
- 更新功能：
 - 将 SPARQL 解析器生成器从 ANTLR v3 升级到最新的、文档齐全且维护良好的 v4；
 - 支持在 SPARQL 查询中编写没有数据类型后缀的数字文字；
 - 支持 SELECT 子句中的算术和逻辑运算符；
 - 支持 SELECT 子句中的聚合 SUM、AVG、MIN 和 MAX；
 - 额外的支持内置在过滤器中，函数功能，包括 `datatype`，`contains`，`ucase`，`lcase`，`strstarts`，`now`，`year`，`month`，`day`，和 `abs`；
 - 支持路径相关功能作为SPARQL 1.1的扩展，包括环路检测、最短路径和K-hop可达性；
 - 支持数据库全量和增量备份和恢复，管理员配置可以开启自动全量备份；
 - 支持基于日志的回滚操作；
 - 支持具有三级隔离的事务：已提交读、快照隔离和可序列化；
 - 扩展数据结构以容纳多达 50 亿个三元组的大规模图。

1.2 文档更新记录

- 修改快速入门相关内容，同gStore 0.9.1版本匹配
- 修改常用API相关内容，同gStore 0.9.1版本匹配
- 修改Workbench控制台相关内容，同gStore 0.9.1版本匹配
- 增加更新日志，记录图数据库gStore的版本和相关文档更新
- 增加文档下载目录，用户可进行文档下载

2. 知识图谱与gStore介绍

2.1 知识图谱简介

近年来随着“人工智能”概念再度活跃，除了“深度学习”这个炙手可热的名词以外，“知识图谱”无疑也是研究者、工业界和投资人心目中的又一颗“银弹”。简单地说，“知识图谱”是一种数据模型，是以图形（Graph）的方式来展现“实体”、实体“属性”，以及实体之间的“关系”。下图是截取的Google的知识图谱介绍网页中的一个例子。在例子中有4个实体，分别是“达芬奇”，“意大利”，“蒙拉丽莎”和“米可朗基罗”。这个图明确地展示了“达芬奇”的逐个属性和属性值（例如名字、生日和逝世时间等），以及之间的关系（例如蒙拉丽莎是达芬奇的画作，达芬奇出生在意大利等）。



目前知识图谱普遍采用了语义网框架中RDF(Resource Description Framework,资源模式框架)模型来表示数据。语义网是万维网之父蒂姆·伯纳斯-李(Tim Berners-Lee)在1998年提出的概念，其核心是构建以数据为中心的网络，即Web of Data。其中RDF是W3C的语义网框架中的数据描述的标准，通常称之为RDF三元组<主体(subject)，谓词(predicate)，宾语(object)>。其中主体一定是一个被描述的资源，由URI来表示。谓词可以表示主体的属性，或者表示主体和宾语之间某种关系；当表示属性时，宾语就是属性值，通常是一个字面值(literal)；否则宾语是另外一个由URI表示的资源。

下图展示了一个人物类百科的RDF三元组的知识图谱数据集。例如y:Abraham_Lincoln表示一个实体URI（其中y表示前缀<http://en.wikipedia.org/wiki/>），其有三个属性(hasName,BornOdate,DiedOnDate)和一个关系(DiedIn)。

Prefix: y= http://en.wikipedia.org/wiki/

主体	属性	客体
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	BornOnDate	"1809-02-12"
y:Abraham_Lincoln	DiedOnDate	1865-04-15
y:Abraham_Lincoln	DiedIn	y:Washington_D.C
y:Washington_D.C	hasName	"Washington D.C."
y:Washington_D.C	FoundYear	1790
y:Washington_D.C	rdf:type	y:city
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_D.C
y:United_States	rdf:type	Country
y:Reese_Witherspoon	rdf:type	y:Actor
y:Reese_Witherspoon	BornOnDate	"1976-03-22"
y:Reese_Witherspoon	BornIn	y:New_Orleans_Louisiana
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:New_Orleans_Louisiana	FoundYear	1718
y:New_Orleans_Louisiana	rdf:type	y:city
y:New_Orleans_Louisiana	locatedIn	y:United_States

图 1-1 RDF数据的例子

面向RDF数据集，W3C提出了一种结构化查询语言SPARQL；它类似于面向关系数据库的查询语言SQL。和SQL一样，SPARQL也是一种描述性的结构化查询语言，即用户只需要按照SPARQL定义的语法规则去描述其想查询的信息即可，不需要明确指定如何进行查询的计算机的实现步骤。2008年1月，SPARQL成为W3C的正式标准。SPARQL中的WHERE子句定义了查询条件，其也是由三元组来表示。我们不过多的介绍语法细节，有兴趣的读者可以参考[1]。下面的例子解释了SPARQL语言。假设我们需要在上面的RDF数据中查询“在1809年2月12日出生，并且在1865年4月15日逝世的人的姓名？”这个查询可以表示成如下图的SPARQL语句。

```
SELECT ?name                                //查询返回的变量值 ↵
WHERE ↵
{ ?m <hasName> ?name.                        //查询条件 ↵
  ?m <BornOnDate> "1809-02-12" . ↵
  ?m <DiedOnDate> "1865-04-15" . ↵
} ↵
```

图 1-2 SPARQL查询的例子

知识图谱数据管理的一个核心问题是如何有效地存储RDF数据集和快速回答SPARQL查询。总的来说，有两套完全不同的思路。其一是我们可以利用已有的成熟的数据库管理系统（例如关系数据库系统）来存储知识图谱数据，将面向RDF知识图谱的SPARQL查询转换为面向此类成熟数据库管理系统的查询，例如面向关系数据库的SQL查询，利用已有的关系数据库产品或者相关技术来回答查询。这里面最核心的研究问题是如何构建关系表来存储RDF知识图谱数据，并且使得转换的SQL查询语句查询性能更高；其二是直接开发面向RDF知识图谱数据的Native的知识图谱数据存储和查询系统（Native RDF图数据库系统），考虑到RDF知识图谱管理的特性，从数据库系统的底层进行优化。

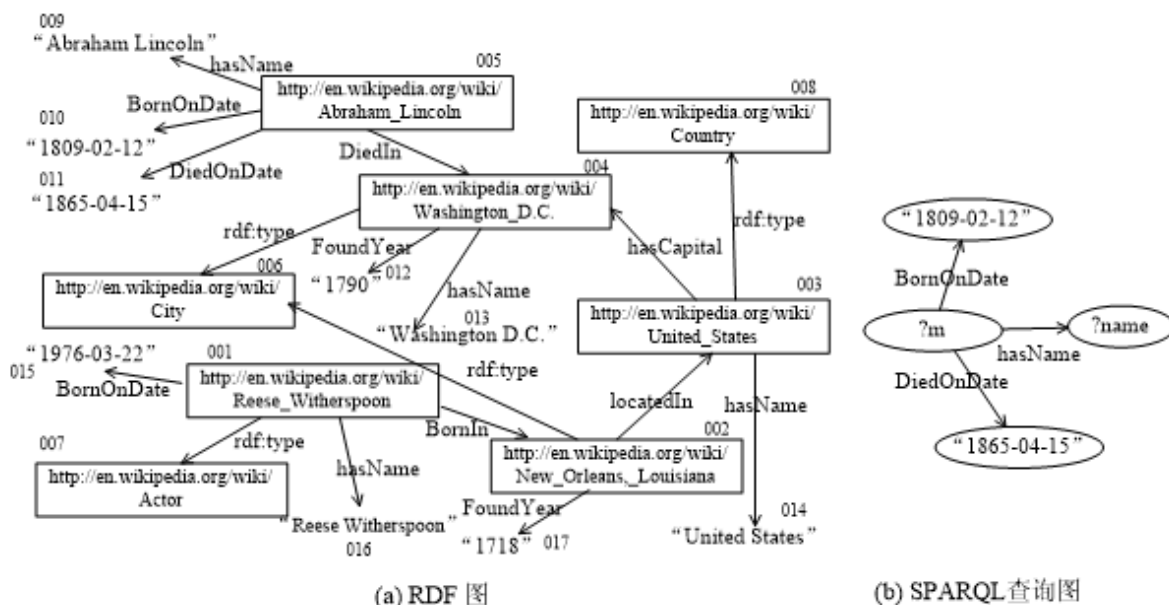


图 1-3 RDF图和SPARQL查询图

2.2 gStore简介

我们所研发的gStore系统属于后者，gStore是由北京大学王选计算机研究所数据管理实验室（PKUMOD）历经十年研发面向RDF数据模型的开源图数据库系统（通常称为Triple Store）。不同于传统基于关系数据库的知识图谱数据管理方法，gStore**原生基于图数据模型**(Native Graph Model)，维持了**原始RDF知识图谱的图结构**；其数据模型是有标签、有向的多边图，每个顶点对应着一个主体或客体。我们将面向RDF的SPARQL查询，转换为**面向RDF图的子图匹配查询**，利用我们所提出的**基于图结构的索引(VS-tree)来加速查询的性能**。图1-3显示了上例所对应的RDF图和SPARQL查询图结构。回答SPARQL查询本质上就是在RDF图中找到SPARQL查询图的子图匹配的位置，这就是基于图数据库的回答SPARQL查询的理论基础。在图1-3例子中，由节点005，009，010和011所推导的子图就是查询图的一个匹配，根据此匹配很容易知道SPARQL的查询结果是“Abraham Lincoln”。关于gStore的核心学术思路，请参考开发资源-论文和专利所发表的论文。

gStore开始于北京大学王选计算机研究所数据管理组（PKUMOD）邹磊教授与滑铁卢大学Tamer Ozsu教授、香港科技大学Lei Chen教授所撰写的VLDB 2011论文(Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Ozsu, Dongyan Zhao, gStore: Answering SPARQL Queries Via Subgraph Matching, Proc. VLDB 4(8): 482-493, 2011)，在论文中提出了利用子图匹配的方法回答SPARQL中的BGP (Basic Graph Pattern)语句的查询执行方案。该文章发表以后，PKUMOD实验室在中国自然科学基金委项目和中国科技部重点研发课题等资助下，持续从事gStore系统的开源、维护和系统优化工作。目前Github上开源的gStore系统可以支持W3C定义的SPARQL 1.1标准（具体可支持的SPARQL语法，请参考【SPARQL查询语言】）；

经过一系列的测试，测试结果表明gStore 在回答复杂查询（例如，包含圆圈）方面比其他数据库系统运行得更快。对于简单的查询，gStore 和其他数据库系统都运行良好。gStore单机版本可以支持**50亿以上**的RDF三元组存储和SPARQL查询，分布式系统gStore（分布式版本，目前未开源）具有非常好的可扩展性，根据“中国软件测评中心”给出的测试报告显示，分布式gStore系统在百亿规模的RDF三元组数据集上具有秒级查询时间。

gStore系统在Github上开源以来，一直采用开源社区中广泛使用的BSD 3-Clause开源协议，以促进gStore相关知识图谱技术生态的建设。根据该协议，我们要求使用者在充分需要尊重代码作者的著作权前提下，允许使用者自由的修改和重新发布代码，也允许使用者在gStore代码基础上自由地开发商业软件，以及发布和销售；但是以上的前提是必须满足第10章“法律条款”中，根据BSD 3-Clause开源协议，我们所拟定的相关法律条款。我们严格要求使用者，在其所发布的基于gStore代码基础上开发的软件上

标有“powered by gStore”和gStore标识（详见参考gStore标识）。我们强烈建议使用者在使用gStore前，参考“开源与法律条款”中有关规定。

3. 安装指南

3.1 系统要求

项目	需求
操作系统	Linux, 例如CentOS, Ubuntu等
架构	x86_64
磁盘容量	根据数据集的大小
内存大小	根据数据集的大小
glibc	必须安装 version >= 2.14
gcc	必须安装 version >= 5.0
g++	必须安装 version >= 5.0
make	必须安装
cmake	必须安装
pkg-config	必须安装
uuid	必须安装
boost	必须安装 version >= 1.56 && <= 1.59
readline	必须安装
readline-devel	必须安装
libcurl-devel	必须安装
openjdk	如果使用Java api, 则需要
openjdk-devel	如果使用Java api, 则需要
requests	如果使用Python http api, 则需要
node	如果使用Nodejs http api则需要 version >=10.9.0
curl-devel	如果使用php http api, 则需要
pthread	如果使用php http api, 则需要
realpath	如果使用gconsole, 则需要
ccache	可选, 用于加速编译

3.2 安装环境准备

根据您的操作系统运行 scripts/setup/ 中相应的脚本能够自动为您解决大部分问题。比如，若您是 Ubuntu 用户，可执行以下指令：

```
$ . scripts/setup/setup_ubuntu.sh
```

在运行脚本之前，建议先安装 5.0 以上版本的 gcc 和 g++。

当然，您也可以选择手动逐步准备环境；下面提供了各系统要求的详细安装指导。

3.2.1 gcc和g++安装

检查 g++ 版本：

```
$ g++ --version
```

若版本低于 5.0，则重新安装 5.0 以上版本。以安装 5.4.0 为例：（适用于 Ubuntu 和 CentOS）

```
$ wget http://ftp.tsukuba.wide.ad.jp/software/gcc/releases/gcc-5.4.0/gcc-5.4.0.tar.gz
$ tar xvf gcc-5.4.0.tar.gz
$ cd gcc-5.4.0
$ ./contrib/download_prerequisites
$ cd ..
$ mkdir gcc-build-5.4.0
$ cd gcc-build-5.4.0
$ ../gcc-5.4.0/configure --prefix=/opt/gcc-5.4.0 --enable-checking=release --enable-languages=c,c++ --disable-multilib
$ sudo make -j4    #允许4个编译命令同时执行，加速编译过程
$ sudo make install
```

Ubuntu 也可直接使用以下命令安装：

```
$ apt install -y gcc-5 g++-5
```

安装成功后，

- **需要修改 gcc 和 g++ 的默认版本：**假设 5.0 以上版本的 gcc 和 g++ 安装在了 /prefix/bin 路径下，则需要执行以下命令：

```
$ export PATH=/prefix:$PATH
```

- **需要修改动态链接库路径：**假设 5.0 以上版本的 gcc 和 g++ 动态链接库在 /prefix/lib 路径下，则需要执行以下命令：

```
$ export LD_LIBRARY_PATH=/prefix/lib:$LD_LIBRARY_PATH
```

3.2.2 jdk安装

判断 jdk 是否安装

```
$ java -version
```

如果没有安装，则安装

```
$ sudo yum install java-1.8.0-openjdk-devel.x86_64      #centos系统
$ sudo apt install -y openjdk-8-jdk                     #ubuntu系统
```

3.2.3 readline 安装

判断 readline 是否安装

```
$ yum list installed | grep readline      #centos系统
$ dpkg -s readline                       #ubuntu系统
```

如果没有安装，则安装

```
$ sudo yum install readline-devel        #centos系统
$ sudo apt install -y libreadline-dev    #ubuntu系统
```

3.2.4 boost 安装 (请使用1.56-1.59)

判断 boost 是否安装

```
$ yum list installed | grep boost        #centos系统
$ dpkg -s boost                          #ubuntu系统
```

如果没有安装，则安装：（以版本 1.56.0 为例）

版本:1.56.0

地址: http://sourceforge.net/projects/boost/files/boost/1.56.0/boost_1_56_0.tar.gz

安装脚本：（适用于 CentOS 和 Ubuntu）

```
$ wget
http://sourceforge.net/projects/boost/files/boost/1.56.0/boost_1_56_0.tar.gz$ tar
-xzvf boost_1_56_0.tar.gz$ cd boost_1_56_0$ ./bootstrap.sh$ sudo ./b2$ sudo
./b2 install
```

Ubuntu 也可直接使用以下命令安装：

```
$ sudo apt install -y libboost-all-dev
```

注意：请在确保 g++ 版本高于 5.0 后安装 boost。若在编译 gStore 时遇到与 boost 链接错误（形如 "undefined reference to boost::..."），很可能是由于您使用低于 5.0 的 gcc 版本编译 boost。此时，请使用以下步骤重新编译 boost：

- 清除旧文件： `./b2 --clean-all`
- 在 ./tools/build/src 下的 user-config.jam 文件中（若此路径下不存在此文件，请在 ./tools/build/example 或其他路径下找到一个示例 user-config.jam 文件并拷贝到

./tools/build/src 下) 添加: `using gcc : 5.4.0 : gcc-5.4.0的路径 ;`

- 在 ./ 下运行 `./bootstrap.sh --with-toolset=gcc`
- `sudo ./b2 install --with-toolset=gcc`

然后重新编译 gStore (请从 `make pre` 开始重做)。

安装成功后,

- **需要修改动态链接库路径:** 假设 boost 的动态链接库在 `/prefix/lib` 路径下, 则需要执行以下命令:

```
$ export LD_LIBRARY_PATH=/prefix/lib:$LD_LIBRARY_PATH
```

- **需要修改头文件路径:** 假设 boost 的头文件在 `/prefix/include` 路径下, 则需要执行以下命令:

```
$ export CPATH=/prefix/include:$CPATH
```

3.2.5 curl 安装

判断 curl 是否安装

```
$ curl --version #centos系统$ curl --version #ubuntu系统
```

如果没有安装, 则安装:

版本: 7.55.1

地址: <https://curl.haxx.se/download/curl-7.55.1.tar.gz>

安装脚本 (适用于 CentOS 和 Ubuntu)

```
$ wget https://curl.haxx.se/download/curl-7.55.1.tar.gz$ tar -xzf curl-7.55.1.tar.gz$ cd curl-7.55.1$ ./configure$ make$ make install
```

或者直接用下面命令安装

```
$ sudo yum install -y libcurl-devel libcurl-dev #centos系统$ sudo apt install -y curl libcurl4 libcurl4-openssl-dev #ubuntu系统
```

3.2.6 cmake 安装

判断 cmake 是否安装

```
$ cmake --version #centos系统$ cmake --version #ubuntu系统
```

如果没有安装, 则安装:

版本: 3.6.2

地址: <https://curl.haxx.se/download/curl-7.55.1.tar.gz>

安装脚本 (适用于 CentOS 和 Ubuntu)

```
$ wget https://cmake.org/files/v3.6/cmake-3.6.2.tar.gz$ tar -xvf cmake-3.6.2.tar.gz && cd cmake-3.6.2/$ ./bootstrap$ make$ make install
```

Ubuntu 也可直接使用以下命令安装:

```
$ sudo apt install -y cmake
```

3.2.7 pkg-config 安装

判断 pkg-config 是否安装

```
$ pkg-config --version          #centos系统$ pkg-config --version          #ubuntu系统
```

如果没有安装, 则安装

```
$ sudo yum install pkgconfig.x86_64          #centos系统$ sudo apt install -y pkg-config          #ubuntu系统
```

3.2.8 uuid 安装

判断 uuid 是否安装

```
$ uuid -m          #centos系统$ uuid -m          #ubuntu系统
```

如果没有安装, 则安装

```
$ sudo yum install libuuid-devel          #centos系统$ sudo apt install -y uuid-dev          #ubuntu系统
```

3.2.9 zip/unzip 安装

用于解压 gStore zip 包。

判断 zip/unzip 是否安装

```
$ yum list installed | grep unzip          #centos系统$ dpkg -s unzip          #ubuntu系统
```

如果没有安装, 则安装

```
$ sudo yum install -y unzip zip          #centos系统$ sudo apt-get install unzip          #ubuntu系统
```

3.3 gStore获取

如果遇到权限问题，请在命令前加 `sudo`。

3.3.1 方式一：download

打开 <https://github.com/pkumod/gStore>，下载gStore.zip；解压zip包。

gStore目前已经上传到gitee（码云），国内用户推荐使用码云下载，速度更快，网址为 <https://gitee.com/PKUMOD/gStore>

3.3.2 方式二：clone(推荐)

通过如下命令 clone：

```
$ git clone https://github.com/pkumod/gStore.git #github$ git clone  
https://gitee.com/PKUMOD/gStore.git #gitee(码云) 国内下载速度更快
```

注意：这一方法需要先安装 Git。

```
$ sudo yum install git #centos系统$ sudo apt-get install git #ubuntu系统
```

3.4 gStore编译

切换到 gStore 目录下：

```
$ cd gStore
```

执行如下指令：

```
$ make pre$ make #若编译顺利完成，最后会出现 Compilation ends  
successfully! 结果$ bin/ginit #初始化，如果顺利完成，最后会出现 system.db is built  
successfully! 结果
```

如果在已安装 5.0 以上版本的 gcc 后 `make pre` 仍报要求 5.0 以上版本 gcc 的错误，请先定位 5.0 以上版本 gcc 的路径，并在 gStore 目录下执行以下命令：

```
$ export CXX=<5.0以上版本gcc的路径>
```

然后重新 `make pre`。假如在这步操作后仍然报相同的错误，请手动删除 `tools/antlr4-cpp-runtime-4/` 下的 `CMakeCache.txt` 和 `CMakeFiles` 文件夹，再重新 `make pre`。

3.5 Docker方式部署gStore

我们提供两种方式通过容器部署gStore：

一种是通过项目根目录的Dockerfile文件自主构建，然后运行容器。

另一种是直接下载已经自动构建完成的镜像，然后直接运行。

3.5.1 环境准备

关于安装使用Docker，参考地址：[docker](#)

3.5.2 直接拉取镜像运行(推荐)

无需下载项目或自己构建，直接输入 `sudo docker pull pkumod/gstore:latest` 拉取已经在docker hub上自动构建完成的镜像。拉取完成后 `sudo docker run -p 9000:80 -it pkumod/gstore:latest` 即可直接启动并进入容器使用。

3.5.3 通过Dockerfile构建镜像

待调整

其他可能也有不少需要补充，所以目前只是**抛砖引玉**，添加了一个最基本的版本。基本的环境构建只是容器化的第一步

4. 快速入门

4.1 数据格式

`gstore` 是基于RDF模型的图数据库引擎，其数据格式也是遵循RDF模型的。RDF 是用于描述现实中资源的W3C 标准，它是描述信息的一种通用方法，使信息可以被计算机应用程序读取并理解。现实中任何实体都可以表示成RDF 模型中的资源，比如，图书的书名、作者、修改日期、内容以及版权信息。这些资源可以用作知识图谱中对客观世界的概念、实体和事件的抽象。每个资源的一个属性及属性值，或者它与其他资源的一条关系，都被称为一条知识。属性和关系能表示成三元组。

一个三元组包括三个元素：主体 (Subject)、属性 (Property) 及客体 (Object)，通常描述的是两个资源间的关系或一个资源的某种属性。当某个三元组描述了某个资源的属性时，其三个元素也被称为主体、属性及属性值 (Property Value)。比如，三元组<亚里士多德、出生地、Chalcis>表达了亚里士多德出生于Chalcis 的事实。

利用这些属性和关系，大量资源就能被连接起来，形成一个大RDF 知识图谱数据集。因此，一个知识图谱通常可以视作三元组的集合。这些三元组集合进而构成一个RDF 数据集。知识图谱的三元组集合可以选择关系型数据库或者图数据库进行存储。

RDF数据应以N-Triple格式提供（现在不支持XML），并且必须以SPARQL1.1语法提供查询。N-Triple格式文件示例如下：

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

三元组通常采用W3C定义的NT文件格式存储，如下表示了3条RDF数据，其中以<和>包裹的值是一个实体的URI，表示的是一个实体，而以""包裹的是字面值，表示的是实体某个属性的值，且后面通过^^表示该值的类型。如下3条RDF数据，分别表示了张三这个实体的两个属性性别和年龄，值分别为男和28，最后一条表示的是张三这个实体与李四这个实体之间存在着一个好友的关系。

```
<张三> <性别> "男"^^<http://www.w3.org/2001/XMLSchema#String> .
<张三> <年龄> "28"^^<http://www.w3.org/2001/XMLSchema#Int> .
<张三> <好友> <李四> .
```

关于N-Triple文件更详细的描述请参考[N-Triple](#)。并非SPARQL1.1中的所有语法都是在gStore中解析和回答的，例如，属性路径超出了gStore系统的能力。

4.2 初始化系统数据库

只要下载并编译gStore系统的代码，就会自动创建一个名为system（真实目录名称system.db）的数据库。这是管理系统统计信息的数据库，包括所有用户和所有数据库。您可以使用gquery命令查询此数据库，但禁止使用编辑器对其进行修改。

system数据库为gStore内置的系统数据库，该数据库无法删除，用于保存系统相关信息，尤其是已构建的数据库信息，如果system数据库损坏，可能导致ghttp无法启动，因此gStore提供了初始化系统数据库功能

4.2.1 命令行模式 (ginit)

ginit用于初始化数据库

用法：

```
bin/ginit -db [db_name1],[db_name2],[...]
```

命令参数：

db_name1: 数据库名称

如果没有写任何的数据库名称，则重新初始化的system数据库中将有其他数据库信息

示例：

```
[root@localhost gStore]$ bin/ginit -db lubm
=====
UPDATE
Insert:
{
  <system>    <built_time>    "2021-02-21 22:50:05".
  <lubm>  <database_status>    "already_built".
  <lubm>  <built_by>    <root>.
  <lubm>  <built_time>    "2021-02-21 22:50:05".
}
=====
parse query  successfully! .
unlock the query_parse_lock .
after Parsing, used 96ms.
write privilege of update lock acquired
QueryCache cleared
Total time used: 97ms.
update num : 4
system.db is built successfully!
```

4.3 创建数据库

创建数据库操作是gStore最重要的操作之一，也是用户安装gStore后需要做的第一个操作，gStore提供多种方式进行数据库创建操作。

4.3.1 命名行模式 (gbuild)

gbuild命令用于从RDF格式文件创建新的数据库，使用方式：

```
bin/gbuild -db dbname -f filename
```

参数含义：

dbname: 数据库名称
filename: 带“.nt”或者“.n3”后缀的文件所在的文件路径

例如，我们从lubm.nt构建一个名为“lubm.db”的数据库，可以在数据文件夹中找到。

```
[root@localhost gStore]$ bin/gbuild -db lubm -f ./data/lubm/lubm.nt
gbuild...
argc: 3 DB_store:lubm      RDF_data: ./data/lubm/lubm.nt
begin encode RDF from : ./data/lubm/lubm.nt ...
```

注意：

- 不能以空的RDF数据集来创建数据库
- 注意不能直接 cd 到 bin 目录下，而要在gStore安装根目录执行gbuild操作

4.3.2 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具，通过gWorkbench可以连接上gStore并通过数据库管理模块可以创建图数据库，具体内容详见【[开发文档](#)】-【[可视化工具Workbench](#)】-【[查询功能](#)】-【[数据库管理](#)】功能。

4.3.3 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 `build` 请求来构建图数据库，具体内容详见【[开发文档](#)】-【[常用API](#)】-【[ghttp接口说明](#)】

4.3.4 Socket API (gServer)

gStore提供了gServer组件作为Socket API服务组件，用户可以通过向gServer发送socket请求实现相关功能，gServer中通过 `build` 请求来构建图数据库，具体内容详见【[开发文档](#)】-【[常用API](#)】-【[gServer接口说明](#)】

4.4 数据库列表

数据库列表功能是获取当前所有可用的数据库列表信息，有如下几种形式

4.4.1 命令行模式 (gshow)

gshow用于获取所有可用数据库列表信息。

用法：

```
bin/gshow
```

示例：

```
[root@localhost gStore]$ bin/gshow
=====
database: system
creator: root
built_time: "2019-07-28 10:26:00"
=====
database: lubm
creator: root
built_time: "2019-07-28 10:27:24"
```

4.4.2 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具，通过gWorkbench可以连接上gStore并通过数据库管理模块可以获取图数据库列表，具体内容详见【开发文档】-【可视化工具Workbench】-【查询功能】-【数据库管理】功能。

4.4.3 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 show 命令实现相关功能，具体内容详见【开发文档】-【常用API】-【ghttp接口说明】

4.4.4 Socket API (gServer)

gStore提供了gServer组件作为Socket API服务组件，用户可以通过向gServer发送socket请求实现相关功能，gServer中通过 show 请求来展示数据库列表，具体内容详见【开发文档】-【常用API】-【gServer接口说明】

4.5 数据库状态查询

数据库状态查询功能是获取指定数据库的统计信息，有如下几种方式。

4.5.1 命令行模式 (gmonitor)

gmonitor用于获取指定数据库的统计信息。

用法：

```
bin/gmonitor -db db_name
```

参数含义：

db_name: 数据库名称

示例：

```
[root@localhost gStore]$ bin/gmonitor -db lubm
database: lubm
creator: root
built_time: "2019-07-28 10:27:24"
triple num: 99550
entity num: 28413
literal num: 0
subject num: 14569
predicate num: 17
```

4.5.2 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具，通过gWorkbench可以连接上gStore并通过数据库管理模块可以查询图数据库，具体内容详见【开发文档】-【可视化工具Workbench】-【查询功能】-【数据库管理】功能。

4.5.3 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 `monitor` 获取数据库统计信息，具体内容详见【开发文档】-【常用API】-【ghttp接口说明】

4.6 数据库查询

数据库查询是gStore最重要的功能之一，gStore支持W3C定义的SPARQL 1.1查询语言，用户可以通过如下几种方式使用gStore数据库查询功能。

4.6.1 命令行模式 (gquery)

gquery用于使用包含SPARQL查询的文件查询现有数据库。（每个文件包含一个精确的SPARQL语句，SPARQL语句不仅可以进行查询操作，还可以进行增加和删除操作，详细的SPARQL语句使用请参考第八章）

1.查询名为db_name的数据库,输入以下命令：

```
bin/gquery -db db_name -q query_file
```

参数含义：

db_name：数据库名称
query_file：以“.sql”结尾的SPARQL语句存放的文件路径(其他后缀名也可以)

例如，我们执行./data/lubm/lubm_q0.sql中的SPARQL语句查询lubm数据库

查询结果为：

```
[root@localhost gStore]$ bin/gquery -db lubm -f ./data/lubm/lubm_q0.sql
There has answer: 15
final result is :
?x
<http://www.Department0.University0.edu/FullProfessor0>
<http://www.Department1.University0.edu/FullProfessor0>
<http://www.Department2.University0.edu/FullProfessor0>
<http://www.Department3.University0.edu/FullProfessor0>
<http://www.Department4.University0.edu/FullProfessor0>
<http://www.Department5.University0.edu/FullProfessor0>
<http://www.Department6.University0.edu/FullProfessor0>
<http://www.Department7.University0.edu/FullProfessor0>
<http://www.Department8.University0.edu/FullProfessor0>
<http://www.Department9.University0.edu/FullProfessor0>
<http://www.Department10.University0.edu/FullProfessor0>
<http://www.Department11.University0.edu/FullProfessor0>
<http://www.Department12.University0.edu/FullProfessor0>
<http://www.Department13.University0.edu/FullProfessor0>
<http://www.Department14.University0.edu/FullProfessor0>
```

2.了解gquery的详细使用，可以输入以下命令进行查看：

```
bin/gquery --help
```

3.进入gquery控制台命令：

```
bin/gquery -db dbname
```

程序显示命令提示符（“gsq|>”），您可以在此处输入命令

使用 `help` 看到所有命令的基本信息

输入 `quit` 以退出gquery控制台。

对于 `sparql` 命令, 使用 `sparql query_file` 执行SPARQL查询语句, `query_file`为存放SPARQL语句的文件路径。当程序完成回答查询时, 它会再次显示命令提示符。

我们也以lubm.nt为例。

```
(base) [root@iz8vb0u9hafhzz1mn5xcklz gstore]# bin/gquery -db lubm

gsq1>sparql ./data/lubm/lubm_q0.sql
... ..
Total time used: 4ms.
final result is :
<http://www.Department0.University0.edu/FullProfessor0>
<http://www.Department1.University0.edu/FullProfessor0>
<http://www.Department2.University0.edu/FullProfessor0>
<http://www.Department3.University0.edu/FullProfessor0>
<http://www.Department4.University0.edu/FullProfessor0>
<http://www.Department5.University0.edu/FullProfessor0>
<http://www.Department6.University0.edu/FullProfessor0>
<http://www.Department7.University0.edu/FullProfessor0>
<http://www.Department8.University0.edu/FullProfessor0>
<http://www.Department9.University0.edu/FullProfessor0>
<http://www.Department10.University0.edu/FullProfessor0>
<http://www.Department11.University0.edu/FullProfessor0>
<http://www.Department12.University0.edu/FullProfessor0>
<http://www.Department13.University0.edu/FullProfessor0>
<http://www.Department14.University0.edu/FullProfessor0>

gsq1>help
help - print commands message
quit - quit the console normally
sparql - load query from the second argument

gsq1>quit
```

注意:

- 如果没有答案, 将打印“[empty result]”, 并且在所有结果后面都有一个空行。
- 使用readline lib, 因此您可以使用键盘中的箭头键查看命令历史记录, 并使用和箭头键移动和修改整个命令。
- 实用程序支持路径完成。(不是内置命令完成)
- 注意不能直接 `cd` 到 `bin` 目录下, 而要在gStore安装根目录执行 `gquery` 操作

4.6.2 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具, 通过gWorkbench可以连接上gStore并通过数据库管理模块可以查询图数据库, 具体内容详见【开发文档】-【可视化工具Workbench】-【查询功能】-【图数据库查询】功能。

4.6.3 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 query 请求来查询图数据库包括查询、删除、插入，具体内容详见【开发文档】-【常用API】-【ghttp接口说明】

4.6.4 Socket API (gServer)

gStore提供了gServer组件作为Socket API服务组件，用户可以通过向gServer发送socket请求实现相关功能，gServer中通过 query 请求来查询图数据库包括查询、删除、插入，具体内容详见【开发文档】-【常用API】-【gServer接口说明】

4.7 数据库导出

导出数据库功能可以将数据库导出成.nt文件。有如下三种形式：

4.7.1 命令行模式 (gexport)

gexport用于导出某个数据库。

用法：

```
bin/gexport -db db_name -f path
```

命令参数：

db_name: 数据库名称

path: 导出到指定文件夹下（如果为空，则默认导出到gStore根目录下）

示例：

```
[root@localhost gStore]# bin/gexport -db lubm
after Handle, used 0 ms.
QueryCache didn't cache
after tryCache, used 0 ms.
in getFinal Result the first half use 0 ms
after getFinalResult, used 0ms.
Total time used: 1ms.
finish exporting the database.
```

4.7.2 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具，通过gWorkbench可以连接上gStore并通过数据库管理模块可以导出图数据库，具体内容详见【开发文档】 - 【可视化工具Workbench】 - 【查询功能】 - 【数据库管理】功能。

4.7.3 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 `export` 功能，具体内容详见【开发文档】 - 【常用API】 - 【ghttp接口说明】

4.8 数据库删除

删除数据库功能可以删除指定数据库，有如下三种形式

4.8.1 命令行模式 (gdrop)

gdrop用于删除某个数据库。

用法：

```
bin/gdrop -db db_name
```

命令参数：

db_name: 数据库名称

示例：

```
[root@localhost gStore]$ bin/drop -db lubm2
after tryCache, used 0 ms.
QueryCache cleared
Total time used: 97ms.
update num : 3
lubm2.db is dropped successfully!
```

为了删除数据库，您不应该只是输入 `rm -r db_name.db` 因为这不会更新名为的内置数据库 `system`。相反，你应该输入 `bin/gdrop -db db_name`。

4.8.2 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具，通过gWorkbench可以连接上gStore并通过数据库管理模块可以删除图数据库，具体内容详见【开发文档】-【可视化工具Workbench】-【查询功能】-【数据库管理】功能。

4.8.3 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 `drop` 命令实现相关功能，具体内容详见【开发文档】-【常用API】-【ghttp接口说明】

4.8.4 Socket API (gServer)

gStore提供了gServer组件作为Socket API服务组件，用户可以通过向gServer发送socket请求实现相关功能，gServer中通过 `drop` 请求来删除图数据库，具体内容详见【开发文档】-【常用API】-【gServer接口说明】

4.9 新增数据

插入RDF数据是gStore常规操作，用户可以通过如下几种方式来执行数据插入操作。

4.9.1 命令行模式 (gadd)--文件

gadd用于将文件中的三元组插入现有数据库。

用法：

```
bin/gadd -db db_name -f rdf_triple_file_path
```

参数含义：

db_name: 数据库名称
rdf_triple_file_path: 带".nt"或者".n3"后缀的文件路径

示例：

```
[bookug@localhost gStore]$ bin/gadd -db lubm -f ./data/lubm/lubm.nt
...
argc: 3 DB_store:lubm insert file:./data/lubm/lubm.nt
get important pre ID
...
insert rdf triples done.
inserted triples num: 99550
```

注意：

1. gadd主要用于RDF文件数据插入

2. 不能直接cd到bin目录下，而要在gStore安装根目录执行gadd操作

4.9.2 命令行模式 (gquery)---SPARQL语句

SPARQL定义中可以通过insert data指令来实现数据插入，基于此原理，用户也可以通过编写SPARQL插入语句，然后使用gStore的gquery工具来实现数据插入，其中SPARQL插入语句示例如下：

```
insert data {
  <张三> <性别> "男"^^<http://www.w3.org/2001/XMLSchema#String>.
  <张三> <年龄> "28"^^<http://www.w3.org/2001/XMLSchema#Int>.
  <张三> <好友> <李四>.
}
```

通过{}可以包含多条RDF数据，注意每条RDF数据都要以.结尾

由于可以使用数据库查询功能实现数据插入，因此也同样可以使用如下功能来进行数据插入。

4.9.3 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具，通过gWorkbench可以连接上gStore并通过数据库查询模块导入新增数据，具体内容详见【开发文档】 - 【可视化工具Workbench】 - 【查询功能】 - 【图数据库查询】功能。

4.9.4 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 `query` 请求来插入数据以及通过 `batchInsert` 来批量插入数据，具体内容详见【开发文档】 - 【常用API】 - 【ghttp接口说明】

4.9.5 Socket API (gServer)

gStore提供了gServer组件作为Socket API服务组件，用户可以通过向gServer发送socket请求实现相关功能，gServer中通过 `query` 请求来插入数据，具体内容详见【开发文档】 - 【常用API】 - 【gServer接口说明】

4.10 删除数据

删除RDF数据是gStore常规操作，用户可以通过如下几种方式来执行数据删除操作。

4.10.1 命令行模式 (gsub) --文件删除

gsub用于从现有数据库中删除文件中的三元组。

用法：

```
bin/gsub db_name rdf_triple_file_path
```

参数含义：

```
rdf_triple_file_path: 带".nt"或者以".n3"后缀的所要删除的数据文件路径
```

示例：

```
[root@localhost gStore]$ bin/gsub lubm ./data/lubm/lubm.nt
...
argc: 3 DB_store:lubm  remove file: ./data/lubm/lubm.nt
...
remove rdf triples done.
removed triples num: 99550
```

4.10.2 命令行模式 (gquery)---SPARQL语句

SPARQL定义中可以通过 `delete data` 指令来实现数据插入，基于此原理，用户也可以通过编写 SPARQL插入语句，然后使用gStore的 `gquery` 工具来实现数据插入，其中SPARQL插入语句示例如下：

```
delete data {
  <张三> <性别> "男"^^<http://www.w3.org/2001/XMLSchema#String>.
  <张三> <年龄> "28"^^<http://www.w3.org/2001/XMLSchema#Int>.
  <张三> <好友> <李四>.
}
```

通过 {} 可以包含多条RDF数据，注意每条RDF数据都要以 `;` 结尾

另外SPARQL中还可以通过 `delete where` 语句来实现根据子查询结构删除数据，如下所示。

```
delete where
{
  <张三> ?x ?y.
}
```

该语句表示删除 `张三` 实体的所有信息（包括属性和关系）

由于可以使用数据库查询功能实现数据插入，因此也同样可以使用如下功能来进行数据插入。

4.10.3 可视化工具 (gWorkbench)

gWorkbench是gStore的一个可视化管理工具，通过gWorkbench可以连接上gStore并通过数据库查询模块通过编写SPARQL语句可以删除数据，具体内容详见【开发文档】 - 【可视化工具Workbench】 - 【查询功能】 - 【图数据库查询】功能。

4.10.4 HTTP API (ghttp)

gStore提供了ghttp组件作为http api服务组件，用户可以通过向ghttp发送http请求实现相关功能，ghttp中通过 `query` 请求来删除数据以及通过 `batchRemove` 来批量删除数据，具体内容详见【开发文档】 - 【常用API】 - 【ghttp接口说明】

4.10.5 Socket API (gServer)

gStore提供了gServer组件作为Socket API服务组件，用户可以通过向gServer发送socket请求实现相关功能，gServer中通过 `query` 请求来删除数据，具体内容详见【开发文档】 - 【常用API】 - 【gServer接口说明】

4.11 HTTP API服务

ghttp是gStore提供的外部访问接口，是一个http API服务，用户通过向ghttp发送http请求，可以实现对gStore的远程连接和远程操作

4.11.1 开启ghttp服务

gStore编译后，在gStore的bin目录下会有一个ghttp服务，但该服务默认不启动，需要用户手动启动ghttp服务，启动命令如下：

```
bin/ghttp -db db_name -p serverPort
```

参数说明：

db_name：要启动ghttp的数据库名称（可选项，如果不填则默认为system数据库，该参数主要作用在于，启动ghttp时，将把该数据库相关信息load到内存中，如果后续查询其他数据库，服务器也将自动load所需数据库，因此该参数可以不填；

serverPort：ghttp监听端口，该端口需要手动指定，且需保证该端口不会被服务器防火墙禁止（可选项，如果不填则默认端口号为9000）

ghttp 支持GET和POST请求类型。

ghttp 支持并发只读查询，但是当包含更新的查询到来时，整个数据库将被锁定。在具有数十个内核线程的计算机上，建议并发运行查询的数量低于300，但我们可以在实验中同时运行13000个查询。要使用并发功能，最好将“打开文件”和“最大进程”的系统设置修改为65535或更大。

如果通过发送包含更新的查询 ghttp，您最好经常向控制台发送 checkpoint 命令 ghttp。否则，更新可能无法与磁盘同步，并且如果 ghttp 服务器异常停止则会丢失（例如，键入“Ctrl + C”）

4.11.2 关闭ghttp服务

关闭ghttp服务请使用以下指令进行关闭，最好不要只需输入命令 Ctrl + C 或 kill 来停止ghttp，因为这不安全。

```
bin/shutdown -p serverPort
```

参数说明：

serverPort：ghttp启动时设置的端口（如果启动时没有设置端口，则默认为9000）

4.11.3 HTTP API (ghttp)

ghttp提供了丰富的API接口以便用户可以远程操作gStore大部分功能，具体接口详见【开发文档】 - 【常用API】 - 【ghttp接口说明】

4.12 Socket API服务

gServer是gStore提供的外部访问接口，是一个socket API服务，用户通过socket双向通信，可以实现对gStore的远程连接和远程操作。

4.12.1 开启gServer服务

gStore编译后，在gStore的bin目录下会有一个gServer服务，但该服务默认不启动，需要用户手动启动gServer服务，启动命令如下：

```
bin/gserver -s
```

其他可选参数说明：

```
-t,--stop: 关闭gserver服务；
-r,--restart: 重启gserver服务；
-p,--port: 修改socket连接端口配置，默认端口为9000，修改后需要重启gserver服务
-P,--printport: 打印当前socket连接端口配置
-d,--debug: 启动debug模式（保持gserver服务在前台运行）
-k,--kill: 强制关闭服务，建议仅在无法正常关闭服务时使用
```

4.12.2 关闭gServer服务

关闭gServer服务请使用以下指令进行关闭，最好不要只需输入命令 `Ctrl + C` 或 `kill` 来停止gServer，因为这不安全。

```
bin/gserver -t
```

4.12.3 gServer相关API

gServer提供了丰富的API接口以便用户可以远程操作gStore大部分功能，具体接口详见【[开发文档](#)】 - 【[常用API](#)】 - 【[gServer 接口说明](#)】

5. API使用

5.1 API介绍

gStore通过http和Socket服务向用户提供API服务，其组件为ghttp和gserver。

5.1.1 HTTP API介绍

我们现在为ghttp提供c++、java、python、php和nodejs API。请参考 `api/http/cpp`、`api/http/java`、`api/http/python`、`api/http/php` 和 `api/http/nodejs` 中的示例代码。要使用这些示例，请确保已经生成了可执行文件。**接下来，使用 `bin/ghttp` 命令启动ghttp服务。**如果您知道一个正在运行的可用ghttp服务器，并尝试连接到它，也是可以的。然后，对于c++和java代码，您需要编译目录 `api/http/cpp/example` 和 `api/http/java/example` 中的示例代码。

具体启动和关闭ghttp可见【[开发文档](#)】-【[快速入门](#)】-【[HTTP API 服务](#)】。

API启动完成后，ghttp访问地址如下：

```
http://serverip:port/
```

其中 `serverip` 为gstore服务器所在的ip地址，`port` 为ghttp启动的端口

5.1.2 Socker API介绍

我们现在为gserver提供c++、java（后续会逐步完善python、php和nodejs）API。请参考 `api/socket/cpp`、`api/socket/java` 中的示例代码。要使用这些示例，请确保已经生成了可执行文件。**接下来，使用 `bin/gserver -s` 命令启动gServer服务。**如果您知道一个正在运行的可用gServer服务器，并尝试连接到它，也是可以的。然后，对于c++和java代码，您需要编译目录 `api/http/cpp/example` 和 `api/http/java/example` 中的示例代码。

具体启动和关闭gServer可见【[开发文档](#)】-【[快速入门](#)】-【[Socket API 服务](#)】。

Socket API启动完成后，就可以通过Socket进行连接了，gServer的默认端口为9000

5.2 HTTP API 结构

gStore的HTTP API放在gStore根目录的API/HTTP目录中，其内容如下：

- gStore/api/http/
 - cpp/ (the C++ API)
 - client.cpp (C++ API的源代码)
 - client.h
 - example/ (使用C++ API的示例程序)
 - GET-example.cpp
 - Benchmark.cpp
 - POST-example.cpp
 - Makefile
 - Makefile (编译和构建lib)
 - java/ (the Java API)
 - client.java
 - lib/
 - src/
 - Makefile
 - jgsc/
 - GstoreConnector.java (Java API的源代码)
 - example/ (使用Java API的示例程序)
 - Benckmark.java
 - GETexample.java
 - POSTexample.java
 - Makefile
 - python/ (the Python API)
 - example/ (python API的示例程序)
 - Benchmark.py
 - GET-example.py
 - POST-example.py
 - src/
 - GstoreConnector.py (使用python API的示例程序)
 - nodejs/ (the Nodejs API)
 - GstoreConnector.js (Nodejs API的源代码)
 - LICENSE
 - package.json
 - README.md
 - example/ (使用Nodejs API的示例程序)
 - POST-example.js
 - GET-example.js

- php/ (the Php API)
 - example/ (php API的示例程序)
 - Benchmark.php
 - POST-example.php
 - GET-example.php
 - src/
 - GstoreConnector.php (php API的源代码)

• ○ -

5.3 ghttp接口说明

5.3.1 接口对接方式

ghttp接口采用的是 http 协议，支持多种方式访问接口，如果ghttp启动的端口为 9000 ,则接口对接内容如下：

接口地址：

```
http://ip:9000/
```

接口支持 get 请求和 post 请求，其中 get 请求参数是放在url中， post 请求是将参数放在 body 请求

注意： GET 请求中各参数如果含有特殊字符，如? , @,&等字符时，需要采用urlencode进行编码，尤其是 sparql 参数必须进行编码

5.3.2 接口列表

接口名称	含义	备注
build	构建图数据库	数据库文件需在服务器本地
load	加载图数据库	将数据库加载到内存中
unload	卸载图数据库	将数据库从内存中卸载
monitor	统计图数据库	统计指定数据库相关信息（如三元组数量等）
drop	删除图数据库	可以逻辑删除和物理删除
show	显示数据库列表	显示所有数据库列表
usermanage（新增）	用户管理	新增、删除、修改用户信息
showuser	显示所有用户列表	显示所有用户列表信息
userprivilegemanage(新增)	用户权限管理	新增、删除、修改用户权限信息
backup	备份数据库	备份数据库信息
restore	还原数据库	还原数据库信息
query	查询数据库	包括查询、删除、插入
export	导出数据库	导出数据库为NT文件
login	登陆数据库	主要是用于验证用户名和密码
check（重写）	检测ghttp心跳信号	
init（弃用）	初始化system	该操作不应该通过ghttp能初始化
refresh（弃用）	重新load数据库	该操作可以由unload+load实现
parameter（弃用）	为备份设置参数	所有参数目前都已经归结到config.ini文件，不允许远程修改
begin	启动事务	事务启动，需要与tquery配合使用
tquery	查询数据库（带事务）	带事务模式的数据查询（仅限于insert和delete）
commit	提交事务	事务完成后提交事务
rollback	回滚事务	回滚事务到begin状态
txnlog	获取transaction的日志信息	以json返回transcation的日志信息
checkpoint	将数据写入磁盘	当对数据库进行了insert或delete操作后，需要手动执行checkpoint
testConnect	测试连接性	用于检测ghttp是否连接
getCoreVersion	获取gStore版本号	获取gStore版本号

接口名称	含义	备注
batchInsert	批量插入数据	批量插入NT数据
batchRemove	批量删除数据	批量删除NT数据
querylog（新增）	获取query的日志信息	以json返回query的日志信息
ipmanage（新增）	黑白名单管理	维护访问gstore的IP黑白名单

5.3.3 接口详细说明

该节中将详细阐述各个接口的输入和输出参数，假设ghttp server的ip地址为127.0.0.1，端口为9000

5.3.3.1 build 创建数据库

简要描述

- 根据已有的NT文件创建数据库
- 文件必须存在gStore服务器上

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以 `JSON` 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 build
db_name	是	string	数据库名称（不需要.db）
db_path	是	string	数据库文件路径（可以是绝对路径，也可以是相对路径，相对路径以gStore安装根目录为参照目录）
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Import RDF file to database done."
}
```

5.3.3.2 check

简要描述

- 检测ghttp服务是否在线

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 <code>**check**</code>

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "the ghttp server is running..."
}
```

5.3.3.3 load

简要描述

- 将数据库加载到内存中，load操作是很多操作的前置条件，如query，monitor等

请求URL

- `http://127.0.0.1:9000/`

请求方式

- **GET/POST**

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以 `JSON` 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 load
db_name	是	string	数据库名称（不需要.db）
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Database loaded successfully."
}
```

5.3.3.4 monitor

简要描述

- 获取数据库统计信息（需要先load数据库）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以 `JSON` 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 monitor
db_name	是	string	数据库名称（不需要.db）
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
database	string	数据库名称
creator	string	创建者
built_time	string	创建时间
triple num	string	三元组数量
entity num	int	实体数量
literal num	int	字符数量（属性值）
subject num	int	主语数量
predicate num	int	谓词数量
connection num	int	连接数量

返回示例

```
{  
  "StatusCode": 0,  
  "StatusMsg": "success",  
  "database": "test_lubm",  
  "creator": "root",  
  "built_time": "2021-08-27 21:29:46",  
  "triple num": "99550",  
  "entity num": 28413,  
  "literal num": 0,  
  "subject num": 14569,  
  "predicate num": 17,  
  "connection num": 0  
}
```

5.3.3.5 unload

简要描述

- 将数据库从内存中卸载（所有的更改都会刷回硬盘）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 unload
db_name	是	string	数据库名称（不需要.db）
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Database unloaded."
}
```

5.3.3.6 drop

简要描述

- 将数据库删除（可以逻辑删除，也可以物理删除）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 drop
db_name	是	string	数据库名称（不需要.db）
username	是	string	用户名
password	是	string	密码（明文）
is_backup	否	string	true:代表逻辑删除，false：代表物理删除（默认为true），如果是逻辑删除，将文件夹变成.bak文件夹，用户可以通过修改文件夹名为.db,然后调用bin/ginit -db 数据库名的方式将其加入system数据库中，从而实现恢复

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Database test_lubm dropped."
}
```

5.3.3.7 show

简要描述

- 显示所有数据库列表

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 show
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
ResponseBody	JSONArray	JSON数组（每个都是一个数据库信息）
----- database	string	数据库名称
-----creator	string	创建者
-----built_time	string	创建时间
-----status	string	数据库状态

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Get the database list successfully!",
  "ResponseBody": [
    {
      "database": "lubm",
      "creator": "root",
      "built_time": "2021-08-22 11:08:57",
      "status": "loaded"
    },
  ],
}
```

```
{
  "database": "movie",
  "creator": "root",
  "built_time": "2021-08-27 20:56:56",
  "status": "unloaded"
}
]
```

5.3.3.8 usermanage (新增)

简要描述

- 对用户进行管理（包括增、删、改）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 usermanage
type	是	string	操作类型（1: adduser , 2: deleteUser 3: alterUserPassword）
username	是	string	用户名
password	是	string	密码（明文）
op_username	是	string	操作的用户名
op_password	是	string	操作的密码（如果是修改密码，该密码为要修改的密码）（如果包含特殊字符，且采用get请求，需要对其值进行URLEncode编码）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "StatusCode": 1004,  "StatusMsg": "username already existed, add user failed."}
```

5.3.3.9 showuser

简要描述

- 显示所有用户信息

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 showuser
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
ResponseBody	JSONArray	JSON对象数组
-----username	string	用户名
-----password	string	密码
-----query_privilege	string	查询权限（数据库名以逗号分隔）
----update_privilege	string	更新权限（数据库名以逗号分隔）
----load_privilege	string	加载权限（数据库名以逗号分隔）
---unload_privilege	string	卸载权限（数据库名以逗号分隔）
----backup_privilege	string	备份权限（数据库名以逗号分隔）
----restore_privilege	string	还原权限（数据库名以逗号分隔）
---export_privilege	string	导出权限（数据库名以逗号分隔）

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "success",    "ResponseBody": [      {        "username": "liwenjie",        "password": "shuaige1982",        "query privilege": "",        "update privilege": "",        "load privilege": "",        "unload privilege": "",        "backup privilege": "",        "restore privilege": "",        "export privilege": ""      },      {        "username": "liwenjie2",        "password": "shuaige19888&&",        "query privilege": "lubm,movie,",        "update privilege": "lubm,movie,",        "load privilege": "lubm,movie,",        "unload privilege": "lubm,movie,",        "backup privilege": "lubm,movie,",        "restore privilege": "",        "export privilege": ""      },      {        "username": "root",        "password": "123456",        "query privilege": "all",        "load privilege": "all",        "unload privilege": "all",        "backup privilege": "all",        "restore privilege": "all",        "export privilege": "all"      }    ]  }
```

5.3.3.10 userprivilegemanage (新增)

简要描述

- 对用户权限进行管理（包括增、删、改）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为userprivilegemanage
type	是	string	操作类型（1: add privilege, 2: delete privilege 3: clear Privilege）
username	是	string	用户名
password	是	string	密码（明文）
op_username	是	string	操作的用户名
privileges	否	string	需要操作的权限（多个权限使用逗号分隔）（如果是clear Privilege 可以为空）1:query,2:load,3:unload,4:update,5:backup,6:restore,7:export, you can set multi privileges by using , to split.
db_name	否	string	需要操作的数据库（如果是clearPrivilege可以为空）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "StatusCode": 0,  "StatusMsg": "add privilege query successfully. \r\nadd privilege load successfully. \r\nadd privilege unload successfully. \r\nadd privilege update successfully. \r\nadd privilege backup successfully. \r\n"}

```

5.3.3.11 backup

简要描述

- 对数据库进行备份

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 backup
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	需要操作的数据库
backup_path	否	string	备份文件路径（可以是相对路径，也可以是绝对路径,相对路径以gStore根目录为参考），默认为gStore根目录下的backup目录

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
backupfilepath	string	备份文件路径（该值可以作为restore的输入参数值）

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "Database backup successfully.",  "backupfilepath": "testbackup/lubm.db_210828211529"}
```

5.3.3.12 restore

简要描述

- 对备份数据库进行还原

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 restore
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	需要操作的数据库
backup_path	否	string	备份文件完整路径【带时间戳的】（可以是相对路径，也可以是绝对路径,相对路径以gStore根目录为参考），默认为gStore根目录下的backup目录

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "Database restore successfully."}
```

5.3.3.13 query

简要描述

- 对数据库进行查询

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 query
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	需要操作的数据库
format	否	string	结果集返回格式（可选值有：json，html和file），默认是json
sparql	是	string	要执行的sparql语句（如果是get请求的话，sparql需要进行url编码）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
head	JSON	头部信息
results	JSON	结果信息（详情请见返回示例）

返回示例

```
{  "head": {    "link": [],    "vars": [      "x"    ]  },  "results": {    "bindings": [      {        "x": {          "type": "uri",          "value": "十面埋伏"        }      },      {        "x": {          "type": "uri",          "value": "投名状"        }      },      {        "x": {          "type": "uri",          "value": "如花"        }      }    ]  },  "statusCode": 0,  "StatusMsg": "success"}
```

5.3.3.14 export

简要描述

- 对数据库进行导出

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 restore
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	需要操作的数据库
db_path	否	string	导出路径（默认为gstore根目录）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
filepath	string	导出文件的路径

返回示例

```
{  "StatusCode": 0,  "StatusMsg": "Export the database successfully.",  "filepath": "export/lubm_210828214603.nt"}
```

5.3.3.15 login

简要描述

- 登陆用户（验证用户名和密码）

请求URL

- http://127.0.0.1:9000/

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，httprequest 中的 body 中的 raw，以JSON 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为login
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "StatusCode": 1001,    "StatusMsg": "wrong password."}
```

5.3.3.16 begin

简要描述

- 开始事务

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 begin
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	数据库名称
isolevel	是	string	事务隔离等级 1:RC(read committed) 2:SI(snapshot isolation) 3:SR(seriablizable)

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
TID	string	事务ID(该ID非常重要，需要以这个作为参数)

返回示例

```
{  "StatusCode": 1001,  "StatusMsg": "wrong password."}
```

5.3.3.17 tquery

简要描述

- 事务型查询

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 tquery
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	数据库名称
tid	是	string	事务ID
sparql	是	string	sparql语句

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "result": "",  "StatusCode": 0,  "StatusMsg": "success"}
```

5.3.3.18 commit

简要描述

- 事务提交

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以 `JSON` 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 commit
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	数据库名称
tid	是	string	事务ID

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "StatusCode": 0,  "StatusMsg": "transaction commit success. TID: 1"}
```

5.3.3.19 rollback

简要描述

- 事务回滚

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 rollback
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	数据库名称
tid	是	string	事务ID

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "StatusCode": 0,  "StatusMsg": "transaction rollback success. TID: 2"}
```

5.3.3.20 txnlog

简要描述

- 获取事务日志（该功能只对root用户生效）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 txnlog
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
list	JSONArray	日志JSON数组

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "Get Transaction log success",    "list":
[    {
        "db_name": "lubm2",        "TID": "1",
        "user": "root",        "begin_time": "1630376221590",        "state":
"COMMITTED",        "end_time": "1630376277349"    },    {
        "db_name": "lubm2",        "TID": "2",        "user": "root",
        "begin_time": "1630376355226",        "state": "ROLLBACK",
        "end_time": "1630376379508"    }    ]}
```

5.3.3.21 checkpoint

简要描述

- 收到将数据刷回到硬盘（使得数据最终生效）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 checkpoint
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	数据库名称

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{  "StatusCode": 0,  "StatusMsg": "Database saved successfully."}
```

5.3.3.22 testConnect

简要描述

- 测试服务器可否连接（用于workbench）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 testConnect
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
CoreVersion	string	内核版本号
licensetype	string	授权类型（开源版还是企业版）

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "success",    "CoreVersion": "0.9.1",  
  "licensetype": "\"opensource\""}}
```

5.3.3.23 getCoreVersion

简要描述

- 获取服务器版本号（用于workbench）

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以 `JSON` 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 getCoreVersion
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
CoreVersion	string	内核版本号

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "success",    "CoreVersion": "0.9.1"}
```

5.3.3.24 batchInsert

简要描述

- 批量插入数据

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以JSON结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 batchInsert
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	数据库名
file	是	string	要插入的数据nt文件（可以是相对路径也可以是绝对路径）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
success_num	string	执行成功的数量

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "Batch Insert Data Successfully.",  "success_num": "25"}
```

5.3.3.25 batchRemove

简要描述

- 批量插入数据

请求URL

- `http://127.0.0.1:9000/`

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以 JSON 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 batchRemove
username	是	string	用户名
password	是	string	密码（明文）
db_name	是	string	数据库名
file	是	string	要删除的数据nt文件（可以是相对路径也可以是绝对路径）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
success_num	int	执行成功的记录数

返回示例

```
{  "StatusCode": 0,    "StatusMsg": "Batch Remove Data Successfully.",  "success_num": "25"}
```

5.3.3.26 shutdown

简要描述

- 关闭ghttp

请求URL

- `http://127.0.0.1:9000/shutdown` 【注意，地址变化】

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，`httprequest` 中的 `body` 中的 `raw`，以 `JSON` 结构传递

参数

参数名	必选	类型	说明
username	是	string	用户名（该用户名默认是system）
password	是	string	密码（该密码需要到服务器的system.db/password[port].txt文件中查看，如端口假设是9000，则password9000.txt文件中查看）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

无返回值，成功则默认收不到信息（该处要完善），失败返回错误JSON信息

5.3.3.27 querylog

简要描述

- 获取查询日志

请求URL

http://127.0.0.1:9000

请求方式

- GET/POST

参数传递方式

- GET请求，参数直接以URL方式传递
- POST请求，httprequest 中的 body 中的 raw，以JSON 结构传递

参数

参数名	必选	类型	说明
operation	是	string	操作名称，固定值为 querylog
username	是	string	用户名
password	是	string	密码（明文）
date	是	string	日期，格式为yyyyMMdd
pageNo	是	int	页号，取值范围1-N，默认1
pageSize	是	int	每页数，取值范围1-N，默认10

返回值

参数	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
totalSize	int	总数
totalPage	int	总页数
pageNo	int	当前页号
pageSize	int	每页数
list	Array	日志数组
QueryDateTime	string	查询时间
Sparql	string	SPARQL语句
Format	string	查询返回格式
RemoteIP	string	请求IP
FileName	string	查询结果集文件
QueryTime	int	耗时(毫秒)
AnsNum	int	结果数

返回示例

```
{  "StatusCode":0,    "StatusMsg":"Get query log success", "totalSize":64,
"totalPage":13, "pageNo":2, "pageSize":5,  "list":[      {
"QueryDateTime":"2021-11-16 14:55:52:90ms:467microseconds",
"Sparql":"select ?name where { ?name <不喜欢> <Eve>. }",
"Format":"json",          "RemoteIP":"183.67.4.126",
"FileName":"140163774674688_20211116145552_847890509.txt",
"QueryTime":0,           "AnsNum":2      }      .....  ]}
```

5.3.3.28 ipmanage

简要描述

- 黑白名单管理

请求URL

http://127.0.0.1:9000

请求方式

- GET/POST

参数传递方式

- GET请求, 参数直接以URL方式传递
- POST请求, httprequest 中的 body 中的 raw, 以 JSON 结构传递

参数

查询黑白名单

参数名	必选	类型	说明
operation	是	string	操作名称, 固定值为ipmanage
username	是	string	用户名
password	是	string	密码 (明文)
type	是	string	操作类型, 固定值为1

保存黑白名单

参数名	必选	类型	说明
operation	是	string	操作名称, 固定值为ipmanage
username	是	string	用户名
password	是	string	密码 (明文)
type	是	string	操作类型, 固定值为2
whiteIPs	是	string	白名单 (多个用,分割, 支持范围配置, 使用-连接如: ip1-1p2)
blackIPs	是	string	黑名单 (多个用,分割, 支持范围配置, 使用-连接如: ip1-1p2)

```
//保存POST示例{  "operation": "ipmanage",  "username": "root",  "password": "123456",  "type": "2",  "whiteIPs": "127.0.0.1,183.67.4.126-183.67.4.128",  "blackIPs": "192.168.1.141"}
```

返回值

参数	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
ResponseBody	Object	返回数据（只有查询时才返回）
whiteIPs	array	白名单列表
blackIPs	array	黑名单列表

返回示例

```
// 查询黑白名单返回{      "StatusCode": 0,      "StatusMsg": "success",      "ResponseBody": {          "whiteIPs": [              "127.0.0.1",              "183.67.4.126-183.67.4.128"          ],          "blackIPs": [              "192.168.1.141"          ]      } }// 保存黑白名单返回{      "StatusCode": 0,      "StatusMsg": "success"}
```

附表1 返回值代码表

代码值	涵义
0	Success
1000	The method type is not support
1001	Authentication Failed
1002	Check Privilege Failed
1003	Param is illegal
1004	The operation conditions are not satisfied
1005	Operation failed
1006	Add privilege Failed
1007	Loss of lock
1008	Transcation manage Failed
1100	The operation is not defined
1101	IP Blocked

5.4 C++ HTTP API

要使用C++ API，请将该短语 `#include "client.h"` 放在cpp代码中，具体使用如下：

构造初始化函数

```
GstoreConnector(std::string serverIP, int serverPort, std::string username,
std::string password);
```

功能：初始化

参数含义：[服务器IP]，[服务器上ghttp端口]，[用户名]，[密码]

使用示例：`GstoreConnector gc("127.0.0.1", 9000, "root", "123456");`

构建数据库：build

```
std::string build(std::string db_name, std::string rdf_file_path, std::string
request_type);
```

功能：通过RDF文件新建一个数据库

参数含义：[数据库名称]，[.nt文件路径]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.build("lubm", "data/lubm/lubm.nt");`

加载数据库：load

```
std::string load(std::string db_name, std::string request_type);
```

功能：加载你建立的数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.load("lubm");`

停止加载数据库：unload

```
std::string unload(std::string db_name, std::string request_type);
```

功能：停止加载数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.unload("lubm");`

用户管理：user

```
std::string user(std::string type, std::string username2, std::string addition,
std::string request_type);
```

功能：添加、删除用户或修改用户的权限，必须由根用户执行操作

1. 添加、删除用户：

参数含义：["add_user"添加用户，"delete_user"删除用户]，[用户名]，[密码]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：gc.user("add_user", "user1", "111111");

2. 修改用户的权限：

参数含义：["add_query"添加查询权限，"delete_query"删除查询权限，"add_load"添加加载权限，"delete_load"删除加载权限，"add_unload"添加不加载权限，"delete_unload"删除不加载权限，"add_update"添加更新权限，"delete_update"删除更新权限，"add_backup"添加备份权限，"delete_bakup"删除备份权限，"add_restore"添加还原权限，"delete_restore"删除还原权限，"add_export"添加导出权限，"delete_export"删除导出权限]，[用户名]，[数据库名]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：gc.user("add_query", "user1", "lubm");

显示用户：showUser

```
std::string showUser(std::string request_type);
```

功能：显示所有用户

参数含义：[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：gc.showUser();

数据库查询：query

```
std::string query(std::string db_name, std::string format, std::string sparql,
std::string request_type);
```

功能：查询数据库

参数含义：[数据库名称]，[查询结果类型json,html或text]，[sparql语句]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
std::string res = gc.query("lubm", "json", sparql);
```

```
std::cout << res << std::endl; //输出结果
```

删除数据库：drop

```
std::string drop(std::string db_name, bool is_backup, std::string request_type);
```

功能：直接删除数据库或删除数据库同时留下备份

参数含义：[数据库名称]，[false不备份，true备份]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：gc.drop("lubm", false); //直接删除数据库不留下备份

监控数据库：monitor

```
std::string monitor(std::string db_name, std::string request_type);
```

功能：显示特定数据库的信息
参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]
使用示例：gc.monitor("lubm");

保存数据库：checkpoint

```
std::string checkpoint(std::string db_name, std::string request_type);
```

功能：如果更改了数据库，保存数据库
参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]
使用示例：gc.checkpoint("lubm");

展示数据库: show

`std::string show(std::string request_type);`功能: 显示所有已创建的数据库参数含义: [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]使用示例: `gc.show()`;

显示内核版本信息: getCoreVersion

`std::string getCoreVersion(std::string request_type);`功能: 得到内核版本信息参数含义: [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]使用示例: `gc.getCoreVersion()`;

显示API版本信息: getAPIVersion

`std::string getAPIVersion(std::string request_type);`
功能: 得到API版本信息
参数含义: [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]
使用示例: `gc.getAPIVersion()`;

查询数据库并保存文件: fquery

`void fquery(std::string db_name, std::string format, std::string sparql, std::string filename, std::string request_type);`
功能: 查询数据库并保留结果到文件
参数含义: [数据库名称], [查询结果类型json,html或text], [sparql语句], [文件名称], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]
使用示例: `gc.fquery("lubm", "json", sparql, "ans.txt")`;

导出数据库

`std::string exportDB(std::string db_name, std::string dir_path, std::string request_type);`
功能: 导出数据库到文件夹下
参数含义: [数据库名称], [数据库导出的文件夹路径], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]
使用示例: `gc.exportDB("lubm", "/root/gStore/")`;

5.5 Java HTTP API

要使用Java API，请参阅[gStore/api/http/java/src/jgsc/GstoreConnector.java](#)。具体使用如下：

构造初始化函数

```
public class GstoreConnector(String serverIP, int serverPort, String username, String password);
```

功能：初始化

参数含义：[服务器IP]，[服务器上ghttp端口]，[用户名]，[密码]

使用示例：`GstoreConnector gc = new GstoreConnector("127.0.0.1", 9000, "root", "123456");`;

构建数据库：build

```
public String build(String db_name, String rdf_file_path, String request_type);
```

功能：通过RDF文件新建一个数据库

参数含义：[数据库名称]，[.nt文件路径]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.build("lubm", "data/lubm/lubm.nt");`;

加载数据库：load

```
public String load(String db_name, String request_type);
```

功能：加载你建立的数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.load("lubm");`;

停止加载数据库：unload

```
public String unload(String db_name, String request_type);
```

功能：停止加载数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.unload("lubm");`;

用户管理：user

```
public String user(String type, String username2, String addition, String request_type);
```

功能：添加、删除用户或修改用户的权限，必须由根用户执行操作

1. 添加、删除用户：

参数含义：["add_user"添加用户，"delete_user"删除用户]，[用户名]，[密码]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.user("add_user", "user1", "111111");`;

2. 修改用户的权限：

参数含义：["add_query"添加查询权限，"delete_query"删除查询权限，"add_load"添加加载权限，"delete_load"删除加载权限，"add_unload"添加不加载权限，"delete_unload"删除不加载权限，"add_update"添加更新权限，"delete_update"删除更新权限，"add_backup"添加备份权限，"delete_bakup"删除备份权限，"add_restore"添加还原权限，"delete_restore"删除还原权限，"add_export"添加导出权限，"delete_export"删除导出权限]，[用户名]，[数据库名]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`gc.user("add_query", "user1", "lubm");`;

显示用户: showUser

```
public String showUser(String request_type);
```

功能: 显示所有用户

参数含义: [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例: gc.showUser();

数据库查询: query

```
public String query(String db_name, String format, String sparql, String request_type);
```

功能: 查询数据库

参数含义: [数据库名称], [查询结果类型json,html或text], [sparql语句], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例:

```
String res = gc.query("lubm", "json", sparql);  
System.out.println(res); //输出结果
```

删除数据库: drop

```
public String drop(String db_name, boolean is_backup, String request_type);
```

功能: 直接删除数据库或删除数据库同时留下备份

参数含义: [数据库名称], [false不备份, true备份], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例: gc.drop("lubm", false); //直接删除数据库不留下备份

监控数据库: monitor

```
public String monitor(String db_name, String request_type);
```

功能: 显示特定数据库的信息

参数含义: [数据库名称], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例: gc.monitor("lubm");

保存数据库: checkpoint

```
public String checkpoint(String db_name, String request_type);
```

功能: 如果更改了数据库,保存数据库

参数含义: [数据库名称], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例: gc.checkpoint("lubm");

展示数据库: show

```
public String show(String request_type);
```

功能: 显示所有已创建的数据库

参数含义: [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例: gc.show();

显示内核版本信息: getCoreVersion

```
public String getCoreVersion(String request_type);
```

功能: 得到内核版本信息

参数含义: [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例: gc.getCoreVersion();

显示API版本信息: getAPIVersion

```
public String getAPIVersion(String request_type);
```

功能: 得到API版本信息

参数含义: [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]

使用示例: gc.getAPIVersion();

查询数据库并保存文件: fquery

```
public void fquery(String db_name, String format, String sparql, String filename, String request_type);
```

功能: 查询数据库并保留结果到文件
参数含义: [数据库名称], [查询结果类型 json,html或text], [sparql语句], [文件名称], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]
使用示例: `gc.fquery("lubm", "json", sparql, "ans.txt");`

导出数据库

```
public String exportDB(String db_name, String dir_path, String request_type);
```

功能: 导出数据库到文件夹下
参数含义: [数据库名称], [数据库导出的文件夹路径], [请求类型"GET"和"post",如果请求类型为“GET”,则可以省略]
使用示例: `gc.exportDB("lubm", "/root/gStore/");`

5.6 Python HTTP API

要使用Python API，请参阅gStore/api/http/python/src/GstoreConnector.py。具体使用如下：

构造初始化函数

```
public class GstoreConnector(self, serverIP, serverPort, username, password):
```

功能：初始化

参数含义：[服务器IP]，[服务器上ghttp端口]，[用户名]，[密码]

使用示例：`gc = GstoreConnector.GstoreConnector("127.0.0.1", 9000, "root", "123456")`

构建数据库：build

```
def build(self, db_name, rdf_file_path, request_type):
```

功能：通过RDF文件新建一个数据库

参数含义：[数据库名称]，[.nt文件路径]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.build("lubm", "data/lubm/lubm.nt")`

加载数据库：load

```
def load(self, db_name, request_type):
```

功能：加载你建立的数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.load("lubm")`

停止加载数据库：unload

```
def unload(self, db_name, request_type):
```

功能：停止加载数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.unload("lubm")`

用户管理：user

```
def user(self, type, username2, addition, request_type):
```

功能：添加、删除用户或修改用户的权限，必须由根用户执行操作

1. 添加、删除用户：

参数含义：["add_user"添加用户，"delete_user"删除用户]，[用户名]，[密码]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.user("add_user", "user1", "111111")`

2. 修改用户的权限：

参数含义：["add_query"添加查询权限，"delete_query"删除查询权限，"add_load"添加加载权限，"delete_load"删除加载权限，"add_unload"添加不加载权限，"delete_unload"删除不加载权限，"add_update"添加更新权限，"delete_update"删除更新权限，"add_backup"添加备份权限，"delete_bakup"删除备份权限，"add_restore"添加还原权限，"delete_restore"删除还原权限，"add_export"添加导出权限，"delete_export"删除导出权限]，[用户名]，[数据库名]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.user("add_query", "user1", "lubm")`

显示用户：showUser

```
def showUser(self, request_type):
```

功能：显示所有用户

参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.showUser()`

数据库查询：query

```
def query(self, db_name, format, sparql, request_type):
```

功能：查询数据库

参数含义：[数据库名称]，[查询结果类型json,html或text]，[sparql语句]，[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：

```
res = gc.query("lubm", "json", sparql)
print(res) //输出结果
```

删除数据库：drop

```
def drop(self, db_name, is_backup, request_type):
```

功能：直接删除数据库或删除数据库同时留下备份

参数含义：[数据库名称]，[false不备份，true备份]，[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.drop("lubm", false)` //直接删除数据库不留下备份

监控数据库：monitor

```
def monitor(self, db_name, request_type):
```

功能：显示特定数据库的信息
参数含义：[数据库名称]，[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.monitor("lubm")`

保存数据库：checkpoint

```
def checkpoint(self, db_name, request_type):
```

功能：如果更改了数据库，保存数据库
参数含义：[数据库名称]，[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.checkpoint("lubm")`

展示数据库：show

```
def show(self, request_type):
```

功能：显示所有已创建的数据库
参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.show()`

显示内核版本信息：getCoreVersion

```
def getCoreVersion(self, request_type):
```

功能：得到内核版本信息
参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.getCoreVersion()`

显示API版本信息：getAPIVersion

```
def getAPIVersion(self, request_type):
```

功能：得到API版本信息
参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.getAPIVersion()`

查询数据库并保存文件：fquery

`def fquery(self, db_name, format, sparql, filename, request_type):`功能: 查询数据库并保留结果到文件
参数含义: [数据库名称], [查询结果类型`json,html`或`text`], [`sparql`语句], [文件名], [请求类型`"GET"`和`"post"`,如果请求类型为“`GET`”,则可以省略]
使用示例: `gc.fquery("lubm", "json", sparql, "ans.txt")`

导出数据库

`def exportDB(self, db_name, dir_path, request_type):` 功能: 导出数据库到文件夹下
参数含义: [数据库名称], [数据库导出的文件夹路径], [请求类型`"GET"`和`"post"`,如果请求类型为“`GET`”,则可以省略]
使用示例: `res = gc.exportDB("lubm", "/root/gStore/")`

5.7 Nodejs HTTP API

在使用Nodejs API之前，键入 `npm install request` 并 `npm install request-promise` 在nodejs文件夹下添加所需的模块。

要使用Nodejs API，请参阅[gStore/api/http/nodejs/GstoreConnector.js](#)。具体使用如下：

构造初始化函数

```
class GstoreConnector(ip = '', port, username = '', password = '');
```

功能：初始化

参数含义：[服务器IP]，[服务器上ghttp端口]，[用户名]，[密码]

使用示例：`gc = new GstoreConnector("127.0.0.1", 9000, "root", "123456");`

构建数据库：build

```
async build(db_name = '', rdf_file_path = '', request_type);
```

功能：通过RDF文件新建一个数据库

参数含义：[数据库名称]，[.nt文件路径]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.build("lubm", "data/lubm/lubm.nt");`

加载数据库：load

```
async load(db_name = '', request_type);
```

功能：加载你建立的数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.load("lubm");`

停止加载数据库：unload

```
async unload(db_name = '', request_type);
```

功能：停止加载数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.unload("lubm");`

用户管理：user

```
async user(type = '', username2 = '' , addition = '' , request_type);
```

功能：添加、删除用户或修改用户的权限，必须由根用户执行操作

1. 添加、删除用户：

参数含义：["add_user"添加用户，"delete_user"删除用户]，[用户名]，[密码]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.user("add_user", "user1", "111111");`

2. 修改用户的权限：

参数含义：["add_query"添加查询权限，"delete_query"删除查询权限，"add_load"添加加载权限，"delete_load"删除加载权限，"add_unload"添加不加载权限，"delete_unload"删除不加载权限，"add_update"添加更新权限，"delete_update"删除更新权限，"add_backup"添加备份权限，"delete_bakup"删除备份权限，"add_restore"添加还原权限，"delete_restore"删除还原权限，"add_export"添加导出权限，"delete_export"删除导出权限]，[用户名]，[数据库名]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.user("add_query", "user1", "lubm");`

显示用户：showUser

```
async showUser(request_type);
```

功能：显示所有用户

参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.showUser();`

数据库查询：query

```
async query(db_name = '', format = '' , sparql = '' , request_type);
```

功能：查询数据库

参数含义：[数据库名称], [查询结果类型json,html或text], [sparql语句], [请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：

```
res = gc.query("lubm", "json", sparql);  
console.log(JSON.stringify(res,"")); //输出结果
```

删除数据库：drop

```
async drop(db_name = '', is_backup , request_type);
```

功能：直接删除数据库或删除数据库同时留下备份

参数含义：[数据库名称], [false不备份, true备份], [请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.drop("lubm", false);` //直接删除数据库不留下备份

监控数据库：monitor

```
async monitor(db_name = '', request_type);
```

功能：显示特定数据库的信息
参数含义：[数据库名称], [请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.monitor("lubm");`

保存数据库：checkpoint

```
async checkpoint(db_name = '', request_type);
```

功能：如果更改了数据库，保存数据库
参数含义：[数据库名称], [请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.checkpoint("lubm");`

展示数据库：show

```
async show(request_type);
```

功能：显示所有已创建的数据库
参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.show();`

显示内核版本信息：getCoreVersion

```
async getCoreVersion(request_type);
```

功能：得到内核版本信息
参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]
使用示例：`res = gc.getCoreVersion();`

显示API版本信息：getAPIVersion

```
async getAPIVersion(request_type);
```

功能：得到API版本信息

参数含义：[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.getAPIVersion();`

查询数据库并保存文件：fquery

```
async fquery(db_name = '', format = '' , sparql = '' , filename = '' ,  
request_type);
```

功能：查询数据库并保留结果到文件

参数含义：[数据库名称]，[查询结果类型json,html或text]，[sparql语句]，[文件名称]，[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：`gc.fquery("lubm", "json", sparql, "ans.txt");`

导出数据库

```
async exportDB(db_name = '' , dir_path = '' , request_type);
```

功能：导出数据库到文件夹下

参数含义：[数据库名称]，[数据库导出的文件夹路径]，[请求类型"GET"和"post",如果请求类型为“GET”，则可以省略]

使用示例：`res = gc.exportDB("lubm", "/root/gStore/");`

5.8 PHP HTTP API

要使用Php API，请参阅[gStore/api/http/php/src/GstoreConnector.php](#)。具体使用如下：

构造初始化函数

```
class GstoreConnector($ip, $port, $username, $password)
```

功能：初始化

参数含义：[服务器IP]，[服务器上ghttp端口]，[用户名]，[密码]

使用示例：`$gc = new GstoreConnector("127.0.0.1", 9000, "root", "123456");`

构建数据库：build

```
function build($db_name, $rdf_file_path, $request_type)
```

功能：通过RDF文件新建一个数据库

参数含义：[数据库名称]，[.nt文件路径]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$res = $gc->build("lubm", "data/lubm/lubm.nt");
```

```
echo $res . PHP_EOL;
```

加载数据库：load

```
function load($db_name, $request_type)
```

功能：加载你建立的数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$ret = $gc->load("test");
```

```
echo $ret . PHP_EOL;
```

停止加载数据库：unload

```
function unload($db_name, $request_type)
```

功能：停止加载数据库

参数含义：[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$ret = $gc->unload("test");
```

```
echo $ret . PHP_EOL;
```

用户管理：user

```
function user($type, $username2, $addition, $request_type)
```

功能：添加、删除用户或修改用户的权限，必须由根用户执行操作

1. 添加、删除用户：

参数含义：["add_user"添加用户, "delete_user"删除用户]，[用户名]，[密码]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$res = $gc->user("add_user", "user1", "111111");
```

```
echo $res . PHP_EOL;
```

2. 修改用户的权限：

参数含义：["add_query"添加查询权限, "delete_query"删除查询权限, "add_load"添加加载权限, "delete_load"删除加载权限, "add_unload"添加不加载权限, "delete_unload"删除不加载权限, "add_update"添加更新权限, "delete_update"删除更新权限, "add_backup"添加备份权限, "delete_bakup"删除备份权限, "add_restore"添加还原权限, "delete_restore"删除还原权限, "add_export"添加导出权限, "delete_export"删除导出权限]，[用户名]，[数据库名]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$res = $gc->user("add_user", "user1", "lubm");
```

```
echo $res . PHP_EOL;
```

显示用户：showUser

```
function showUser($request_type)
```

功能：显示所有用户

参数含义：[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$res = $gc->showUser();
```

```
echo $res . PHP_EOL;
```

数据库查询：query

```
function query($db_name, $format, $sparql, $request_type)
```

参数含义：[数据库名称]，[查询结果类型json,html或text]，[sparql语句]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$res = $gc->query("lubm", "json", $sparql);
```

```
echo $res . PHP_EOL; //输出结果
```

删除数据库：drop

```
function drop($db_name, $is_backup, $request_type)
```

功能：直接删除数据库或删除数据库同时留下备份

参数含义：[数据库名称]，[false不备份, true备份]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]

使用示例：

```
$res = $gc->drop("lubm", false); //直接删除数据库不留下备份
```

```
echo $res . PHP_EOL;
```

监控数据库：monitor

```
function monitor($db_name, $request_type)功能：显示特定数据库的信息参数含义：[数据库名称]
```

[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]使用示例：\$res = \$gc->monitor("lubm");echo \$res . PHP_EOL;

保存数据库：checkpoint

```
function checkpoint($db_name, $request_type)功能：如果更改了数据库，保存数据库参数含义：  
[数据库名称]，[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]使用示例：$res = $gc-  
>checkpoint("lubm");echo $res. PHP_EOL;
```

展示数据库：show

```
function show($request_type)功能：显示所有已创建的数据库参数含义：[请求类型"GET"和"post"，  
如果请求类型为“GET”，则可以省略]使用示例：$res = $gc->show();echo $res. PHP_EOL;
```

显示内核版本信息：getCoreVersion

```
function getCoreVersion($request_type)功能：得到内核版本信息参数含义：[请求类  
型"GET"和"post"，如果请求类型为“GET”，则可以省略]使用示例：$res = $gc-  
>getCoreVersion();echo $res. PHP_EOL;
```

显示API版本信息：getAPIVersion

```
function getAPIVersion($request_type)  
功能：得到API版本信息  
参数含义：[请求类型"GET"和"post"，如果请求类型为“GET”，则可以省略]  
使用示例：  
$res = $gc->getAPIVersion();  
echo $res. PHP_EOL;
```

查询数据库并保存文件：fquery

```
function fquery($db_name, $format, $sparql, $filename, $request_type)  
功能：查询数据库并保留结果到文件  
参数含义：[数据库名称]，[查询结果类型json,html或text]，[sparql语句]，[文件名称]，[请求类  
型"GET"和"post"，如果请求类型为“GET”，则可以省略]  
使用示例：$gc->fquery("lubm", "json", $sparql, "ans.txt");
```

导出数据库

```
function exportDB($db_name, $dir_path, $request_type)  
功能：导出数据库到文件夹下  
参数含义：[数据库名称]，[数据库导出的文件夹路径]，[请求类型"GET"和"post"，如果请求类型为  
“GET”，则可以省略]  
使用示例：$res = $gc->exportDB("lubm", "/root/gStore/");
```

5.9 gServer接口说明

5.9.1 接口对接方式

gServer接口采用的是 `socket` 协议，支持多种方式访问接口，如果Main目录下的gserver启动的端口为 `9000` ,则接口对接内容如下：

接口地址：

```
http://ip:9000/
```

接口支持输入一个json格式的参数字表，如下所示：

```
{"op": "[op_type]", "[paramname1]": "[paramvalue1]", "[paramname2]": "[paramvalue2]".....}
```

5.9.2 接口列表

接口名称	含义	备注
build	构建图数据库	数据库文件需在服务器本地
load	加载图数据库	将数据库加载到内存中
unload	卸载图数据库	将数据库从内存中卸载
drop	删除图数据库	可以逻辑删除和物理删除
show	显示数据库列表	显示所有数据库列表
query	查询数据库	包括查询、删除、插入
stop	关闭服务端	只有root用户可以操作
close	关闭客户端连接	处理客户端关闭连接请求
login	登陆数据库	主要是用于验证用户名和密码

5.9.3 接口详细说明

该节中将详细阐述各个接口的输入和输出参数，假设gserver的ip地址为127.0.0.1，端口为9000

5.9.3.1 build 创建数据库

简要描述

- 根据已有的NT文件创建数据库
- 文件必须存在gStore服务器上

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为 build
db_name	是	string	数据库名称（不需要.db）
db_path	是	string	数据库文件路径（可以是绝对路径，也可以是相对路径，相对路径以gStore安装根目录为参照目录）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Import RDF file to database done."
}
```

5.9.3.2 load

简要描述

- 将数据库加载到内存中，load操作是很多操作的前置条件，如query等

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为load
db_name	是	string	数据库名称（不需要.db）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Load database successfully."
}
```

5.9.3.3 unload

简要描述

- 将数据库从内存中卸载（所有的更改都会刷回硬盘）

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为 unload
db_name	是	string	数据库名称（不需要.db）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Unload database done."
}
```

5.9.3.4 drop

简要描述

- 将数据库删除

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为 drop
db_name	是	string	数据库名称（不需要.db）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Drop database done."
}
```

5.9.3.5 show

简要描述

- 显示所有数据库列表

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为 show

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
ResponseBody	JSONArray	JSON数组（每个都是一个数据库信息）
----- database	string	数据库名称
-----status	string	数据库状态

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "success",
  "ResponseBody": [
    "lubm": "loaded",
    "lubm10k": "unloaded"
  ]
}
```

5.9.3.6 query

简要描述

- 对数据库进行查询

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为 query
db_name	是	string	需要操作的数据库
format	否	string	结果集返回格式，默认是json
sparql	是	string	要执行的sparql语句

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息
head	JSON	头部信息
results	JSON	结果信息（详情请见返回示例）

返回示例

```
{
  "head": {
    "link": [],
    "vars": [
      "x"
    ]
  },
  "results": {
    "bindings": [
      {
        "x": {
          "type": "uri",
          "value": "十面埋伏"
        }
      }
    ]
  }
}
```

```
{
  "x": {
    "type": "uri",
    "value": "投名状"
  },
  {
    "x": {
      "type": "uri",
      "value": "如花"
    }
  }
],
"StatusCode": 0,
"StatusMsg": "success"
}
```

5.9.3.7 login

简要描述

- 登陆用户（验证用户名和密码）

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为 login
username	是	string	用户名
password	是	string	密码（明文）

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 1001,
  "StatusMsg": "wrong password."
}
```

5.9.3.8 stop

简要描述

- 关闭服务端

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为stop

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "server stopped."
}
```

5.9.3.9 close

简要描述

- 关闭与客户端的连接

请求ip

- 127.0.0.1

请求端口号

- 9000

参数传递方式

- 以JSON结构传递

参数

参数名	必选	类型	说明
op	是	string	操作名称，固定值为 close

返回值

参数名	类型	说明
StatusCode	int	返回值代码值（具体请参考附表：返回值代码表）
StatusMsg	string	返回具体信息

返回示例

```
{
  "StatusCode": 0,
  "StatusMsg": "Connection disconnected."
}
```

附表1 返回值代码表

代码值	涵义
0	Success
1000	The method type is not support
1001	Authentication Failed
1002	Check Privilege Failed
1003	Param is illegal
1004	The operation conditions are not satisfied
1005	Operation failed
1006	Add privilege Failed
1007	Loss of lock
1008	Transcation manage Failed
1100	The operation is not defined
1101	IP Blocked

6. SPARQL查询语法

6.1 图模式 (Graph Patterns)

本文档主要参考了 [SPARQL 1.1 标准文档](#)，同时也增加了 gStore 自身定制化的内容，如果想要详细了解 gStore 支持的 SPARQL 语句，请仔细阅读我们的文档吧！

除特殊说明处，本文档将持续使用以下的 RDF 数据实例作为查询的对象：

```
<刘亦菲> <姓名> "刘亦菲" .
<刘亦菲> <姓名> "Crystal Liu" .
<刘亦菲> <性别> "女" .
<刘亦菲> <星座> "处女座" .
<刘亦菲> <职业> "演员" .

<林志颖> <姓名> "林志颖" .
<林志颖> <性别> "男" .
<林志颖> <职业> "演员" .
<林志颖> <职业> "导演" .

<胡军> <姓名> "胡军" .
<胡军> <性别> "男" .
<胡军> <星座> "双鱼座" .
<胡军> <职业> "演员" .
<胡军> <职业> "配音" .
<胡军> <职业> "制片" .
<胡军> <职业> "导演" .

<天龙八部> <主演> <林志颖> .
<天龙八部> <主演> <刘亦菲> .
<天龙八部> <主演> <胡军> .
<天龙八部> <类型> <武侠片> .
<天龙八部> <类型> <古装片> .
<天龙八部> <类型> <爱情片> .
<天龙八部> <豆瓣评分> "8.3"^^<http://www.w3.org/2001/XMLSchema#float> .
<天龙八部> <上映时间> "2003-12-11T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .

<恋爱大赢家> <主演> <林志颖> .
<恋爱大赢家> <主演> <刘亦菲> .
<恋爱大赢家> <类型> <爱情片> .
<恋爱大赢家> <类型> <剧情片> .
<恋爱大赢家> <豆瓣评分> "6.1"^^<http://www.w3.org/2001/XMLSchema#float> .
<恋爱大赢家> <上映时间> "2004-11-30T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
```

由于 SPARQL 1.1 标准文档暂无官方中文译本，因此下文中术语首次出现时将注明其英文原文。

按照标准，SPARQL 查询中的**关键词均不区分大小写**。

6.1.1 最简单的图模式

我们先给出一个最简单的查询：

```
SELECT ?movie
WHERE
{
    ?movie <主演> <刘亦菲> .
}
```

查询由两部分组成：**SELECT 语句**指定需要输出查询结果的变量，**WHERE 语句**提供用来与数据图匹配的图模式。上面的查询中，图模式由单条**三元组** `?movie <主演> <刘亦菲>` 构成，其中作为主语的 `?movie` 是**变量**，作为谓词的 `<主演>` 和作为宾语的 `<刘亦菲>` 是 **IRI** (International Resource Identifier, 国际资源标识符)。这个查询将返回由刘亦菲主演的所有影视作品，在示例数据上运行结果如下：

?movie
<天龙八部>
<恋爱大赢家>

三元组的主语、谓词、宾语都可以是 IRI；宾语还可以是 **RDF 字面量 (RDF Literal)**。以下查询将给出示例数据中所有职业为导演的人物：

```
SELECT ?person
WHERE
{
    ?person <职业> "导演" .
}
```

其中 `"导演"` 是一个 RDF 字面量。

结果如下：

?person
<胡军>
<林志颖>

当前 gStore 版本下，带有数据类型的 RDF 字面量在查询中需要添加与数据文件中相应的后缀。例如，以下查询将给出豆瓣评分为 8.3 的影视作品：

```
SELECT ?movie
WHERE
{
    ?movie <豆瓣评分> "8.3"^^<http://www.w3.org/2001/XMLSchema#float> .
}
```

结果如下：

?movie
<天龙八部>

其他的常见数据类型包括 `<http://www.w3.org/2001/XMLSchema#integer>`（整数类型），`<http://www.w3.org/2001/XMLSchema#decimal>`（定点类型），`xsd:double`（双精度浮点类型），`<http://www.w3.org/2001/XMLSchema#string>`（字符串类型），`<http://www.w3.org/2001/XMLSchema#boolean>`（布尔类型），`<http://www.w3.org/2001/XMLSchema#dateTime>`（日期时间类型）。数据文件中也可能出现其他数据类型，只需在查询中使用 `^^<数据类型后缀>` 的形式即可。

6.1.2 基本图模式 (Basic Graph Pattern)

基本图模式即为三元组的集合；上一节中的两个查询的 `WHERE` 语句均只有最外层大括号，因此属于**基本图模式**；加上最外层大括号，即为由单个基本图模式构成的**组图模式 (Group Graph Pattern)**。

上一节的两个查询中基本图模式都由单个三元组构成。以下查询使用了由多个三元组构成的基本图模式，将给出示例数据中天龙八部的所有男性主演：

```
SELECT ?person
WHERE
{
    <天龙八部> <主演> ?person .
    ?person <性别> "男" .
}
```

结果如下：

?person
<胡军>
<林志颖>

6.1.3 组图模式 (Group Graph Pattern)

组图模式以配对的大括号分隔。组图模式既可以像上一节介绍的那样由单个基本图模式构成，也可以由多个子组图模式和以下的 **OPTIONAL**, **UNION**, **MINUS** 三种运算嵌套而成。**FILTER** 则在一个组图模式的范围内过滤结果。

OPTIONAL

关键词 `OPTIONAL` 使用的语法如下：

```
pattern1 OPTIONAL { pattern2 }
```

查询结果必须匹配 `pattern1`，并选择性地匹配 `pattern2`。`pattern2` 被称为 `OPTIONAL` 图模式。如果 `pattern2` 存在匹配，则将其加入 `pattern1` 的匹配结果；否则，仍然输出 `pattern1` 的匹配结果。因此，`OPTIONAL` 常用于应对部分数据缺失的情况。

下面的查询给出示例数据中人物的性别和星座信息。其中，只要存在性别信息的人物都会被返回，不论是否同时存在该人物的星座信息；若同时存在，则额外返回。

```

SELECT ?person ?gender ?horoscope
WHERE
{
    ?person <性别> ?gender .
    OPTIONAL
    {
        ?person <星座> ?horoscope .
    }
}

```

结果如下：

?person	?gender	?horoscope
<刘亦菲>	"女"	"处女座"
<林志颖>	"男"	
<胡军>	"男"	"双鱼座"

UNION

关键词 UNION 的使用语法与 OPTIONAL 类似。以 UNION 相连的图模式中，只要存在一个与某数据匹配，该数据就与以 UNION 相连的整体匹配。因此，UNION 可以理解为对它所连接的各图模式的匹配结果集合求并集（由于允许重复结果，实际上采用多重集语义）。

下面的查询给出示例数据中类别是古装片或剧情片的影视作品：

```

SELECT ?movie
WHERE
{
    {?movie <类型> <古装片> .}
    UNION
    {?movie <类型> <剧情片> .}
}

```

结果如下：

?movie
<天龙八部>
<恋爱大赢家>

MINUS

关键词 MINUS 的使用语法与 OPTIONAL, UNION 类似。MINUS 左边和右边的图模式的匹配均会被计算，从左边的图模式的匹配结果中移除能与右边的图模式匹配的部分作为最终结果。因此，MINUS 可以理解为对它所连接的两个图模式的匹配结果集合求差（左为被减集合，多重集语义）。

下面的查询将给出示例数据中主演了天龙八部但没有主演恋爱大赢家的人物：

```
SELECT ?person
WHERE
{
    <天龙八部> <主演> ?person .
    MINUS
    {<恋爱大赢家> <主演> ?person .}
}
```

结果如下：

?person
<胡军>

FILTER

关键词 FILTER 之后紧随着一个约束条件，当前组图模式中不满足此条件的结果将被过滤掉，不被返回。FILTER 条件中可以使用等式、不等式以及各种内建函数。

下面的查询将给出示例数据中豆瓣评分高于 8 分的影视作品：

```
SELECT ?movie
WHERE
{
    ?movie <豆瓣评分> ?score .
    FILTER (?score > "8"^^<http://www.w3.org/2001/XMLSchema#float>)
}
```

结果如下：

?movie
<天龙八部>

无论 FILTER 放置在一个组图模式中的什么位置，只要仍然处于同一个嵌套层，则其语义不变，约束条件的作用范围仍然是当前组图模式。比如以下的查询就与前一个查询等价：

```
SELECT ?movie
WHERE
{
    FILTER (?score > "8"^^<http://www.w3.org/2001/XMLSchema#float>)
    ?movie <豆瓣评分> ?score .
}
```

常用于 FILTER 条件的一个内建函数是正则表达式 **REGEX**。下面的查询将给出示例数据中的刘姓人物：

```
SELECT ?person
WHERE
{
    ?person <姓名> ?name .
    FILTER REGEX(?name, "刘.*")
}
```

结果如下：

?person
<刘亦菲>

6.2 聚合函数 (Aggregates)

聚合函数用在 SELECT 语句中，语法如下：

```
SELECT (AGGREGATE_NAME(?x) AS ?y)
WHERE
{
    ...
}
```

其中，`AGGREGATE_NAME` 是聚合函数的名称，变量 `?x` 是聚合函数作用的对象，变量 `?y` 是最终结果中聚合函数值的列名。

聚合函数作用于各组结果。返回的全部结果默认作为一组。gStore支持的聚合函数如下所示：

COUNT

用于计数的聚合函数。

下面的查询将给出示例数据中职业为演员的人物的数目：

```
SELECT (COUNT(?person) AS ?count_person)
WHERE
{
    ?person <职业> "演员" .
}
```

结果如下：

?count_person
"3"^^< http://www.w3.org/2001/XMLSchema#integer >

SUM

用于求和的聚合函数。

下面的查询将给出示例数据中所有电影的豆瓣评分之和：

```
SELECT (SUM(?score) AS ?sum_score)
WHERE
{
    ?movie <豆瓣评分> ?score .
}
```

结果如下：

?sum_score
"14.400000"^^ http://www.w3.org/2001/XMLSchema#float

AVG

用于求平均值的聚合函数。

下面的查询将给出示例数据中所有电影的平均豆瓣评分：

```
SELECT (AVG(?score)) AS ?avg_score)
WHERE
{
    ?movie <豆瓣评分> ?score .
}
```

结果如下：

?avg_score
"7.200000"^^ http://www.w3.org/2001/XMLSchema#float

MIN

用于求最小值的聚合函数。

下面的查询将给出示例数据中所有电影的最低豆瓣评分：

```
SELECT (MIN(?score)) AS ?min_score)
WHERE
{
    ?movie <豆瓣评分> ?score .
}
```

结果如下：

?min_score
"6.1"^^ http://www.w3.org/2001/XMLSchema#float

MAX

用于求最大值的聚合函数。

下面的查询将给出示例数据中所有电影的最低豆瓣评分：

```
SELECT (MAX(?score)) AS ?max_score)
WHERE
{
    ?movie <豆瓣评分> ?score .
}
```

结果如下：

?max_score
"8.3"^^ http://www.w3.org/2001/XMLSchema#float

GROUP BY

如果希望按照某一个变量的值对结果分组，可以使用关键词 GROUP BY 。例如，下面的查询将给出示例数据中的所有职业及对应的人数：

```
SELECT ?occupation (COUNT(?person) AS ?count_person)
WHERE
{
    ?person <职业> ?occupation .
}
GROUP BY ?occupation
```

结果如下：

?occupation	?count_person
"演员"	"3"^^< http://www.w3.org/2001/XMLSchema#integer >
"导演"	"2"^^< http://www.w3.org/2001/XMLSchema#integer >
"配音"	"1"^^< http://www.w3.org/2001/XMLSchema#integer >
"制片"	"1"^^< http://www.w3.org/2001/XMLSchema#integer >

6.3 结果序列修饰符 (Solution Sequences and Modifiers)

以下的关键词均属于结果序列修饰符，它们对查询结果做后处理，以形成最终返回的结果。

DISTINCT: 去除重复结果

SELECT 语句不带关键词 DISTINCT 的查询会在最终结果中保留重复的结果。例如下面的查询给出示例数据中所有的职业：

```
SELECT ?occupation
WHERE
{
    ?person <职业> ?occupation .
}
```

结果如下：

?occupation
"演员"
"演员"
"演员"
"导演"
"导演"
"制片"
"配音"

如果希望查看不重复的职业种类，则可以在 SELECT 语句中添加关键词 DISTINCT：

```
SELECT DISTINCT ?occupation
WHERE
{
    ?person <职业> ?occupation .
}
```

结果如下：

?occupation
"演员"
"导演"
"制片"
"配音"

DISTINCT 也可以在聚合函数 COUNT 中使用。下面的查询给出示例数据中的职业种类数目：

```
SELECT (COUNT(DISTINCT ?occupation) AS ?count_occupation)
WHERE
{
    ?person <职业> ?occupation .
}
```

结果如下：

?count_occupation
"4"^^< http://www.w3.org/2001/XMLSchema#integer >

ORDER BY: 排序

查询结果默认是无序的。如果希望根据某些变量的值对结果进行排序，可以在 WHERE 语句后面添加 ORDER BY 语句。例如下面的查询将示例数据中的影视作品按照豆瓣评分排序，未指定顺序时默认为升序：

```

SELECT ?movie ?score
WHERE
{
    ?movie <豆瓣评分> ?score
}
ORDER BY ?score

```

结果如下：

?movie	?score
<恋爱大赢家>	"6.1"^^< http://www.w3.org/2001/XMLSchema#float >
<天龙八部>	"8.3"^^< http://www.w3.org/2001/XMLSchema#float >

如果希望降序排序，需要用关键词 DESC 修饰变量名：

```

SELECT ?movie ?score
WHERE
{
    ?movie <豆瓣评分> ?score
}
ORDER BY DESC(?score)

```

结果如下：

?movie	?score
<天龙八部>	"8.3"^^< http://www.w3.org/2001/XMLSchema#float >
<恋爱大赢家>	"6.1"^^< http://www.w3.org/2001/XMLSchema#float >

ORDER BY 语句可以包含多个以空格分隔的变量，每个变量都可用 DESC 修饰。gStore 暂不支持在 ORDER BY 语句中使用含四则运算的表达式及内建函数。

OFFSET: 跳过一定数量的结果

OFFSET 语句放在 WHERE 语句之后，其语法如下：

```

OFFSET nonnegative_integer

```

其中 `nonnegative_integer` 须为非负整数，表示需要跳过的结果数量。`OFFSET 0` 符合语法，但不会对结果产生影响。由于查询结果默认无序，SPARQL 语义不保证跳过的结果满足任何确定性的条件。因此，OFFSET 语句一般与 ORDER BY 语句配合使用。

下面的查询将示例数据中的影视作品按豆瓣评分从低到高排序，并跳过评分最低的影视作品：

```

SELECT ?movie ?score
WHERE
{
    ?movie <豆瓣评分> ?score .
}
ORDER BY ?score
OFFSET 1

```

结果如下：

?movie	?score
<天龙八部>	"8.3"^^< http://www.w3.org/2001/XMLSchema#float >

LIMIT: 限制结果数量

LIMIT 语句的语法与 OFFSET 语句类似：

```
LIMIT nonnegative_integer
```

其中 `nonnegative_integer` 须为非负整数，表示允许的最大结果数量。与 OFFSET 类似，由于查询结果默认无序，LIMIT 语句一般与 ORDER BY 语句配合使用。

下面的查询给出示例数据中豆瓣评分最高的影视作品：

```
SELECT ?movie ?score WHERE { ?movie <豆瓣评分> ?score . } ORDER BY DESC(?score) LIMIT 1
```

结果如下：

?movie	?score
<天龙八部>	"8.3"^^< http://www.w3.org/2001/XMLSchema#float >

6.4 图更新

通过 **INSERT DATA**，**DELETE DATA** 和 **DELETE WHERE** 查询，我们可以向数据库中插入或从数据库中删除三元组。

INSERT DATA

INSERT DATA 用于向数据库中插入三元组。其语法与 SELECT 查询类似，区别在于构成组图模式的三元组中不能含有变量。

下面的查询向示例数据中插入影视作品仙剑奇侠传的相关信息：

```
INSERT DATA
{
    <仙剑奇侠传> <主演> <胡歌> .
    <仙剑奇侠传> <主演> <刘亦菲> .
    <仙剑奇侠传> <类型> <武侠片> .
    <仙剑奇侠传> <类型> <古装片> .
    <仙剑奇侠传> <类型> <爱情片> .
    <仙剑奇侠传> <豆瓣评分> "8.9"^^<http://www.w3.org/2001/XMLSchema#float> .
}
```

“图模式-最简单的图模式”一节中出现过的查询

```
SELECT ?movie
WHERE
{
    ?movie <主演> <刘亦菲> .
}
```

在插入上述数据后，结果变为：

?movie
<天龙八部>
<恋爱大赢家>
<仙剑奇侠传>

DELETE DATA

DELETE DATA 用于从数据库中删除三元组。其用法与 INSERT DATA 完全类似。

DELETE WHERE

DELETE DATA 用于从数据库中删除符合条件的三元组；相比起 DELETE DATA，它的 WHERE 语句与 SELECT 查询的 WHERE 语句是完全相同的，也就是说三元组中允许含有变量。例如，下面的查询删除示例数据中所有武侠片的相关信息：

```
DELETE WHERE { ?movie <类型> <武侠片> . ?movie ?y ?z . }
```

此时再次运行“图模式-最简单的图模式”一节中出现过的查询：

```
SELECT ?movie
WHERE
{
    ?movie <主演> <刘亦菲> .
}
```

结果变为：

?movie
<恋爱大赢家>

6.5 高级功能

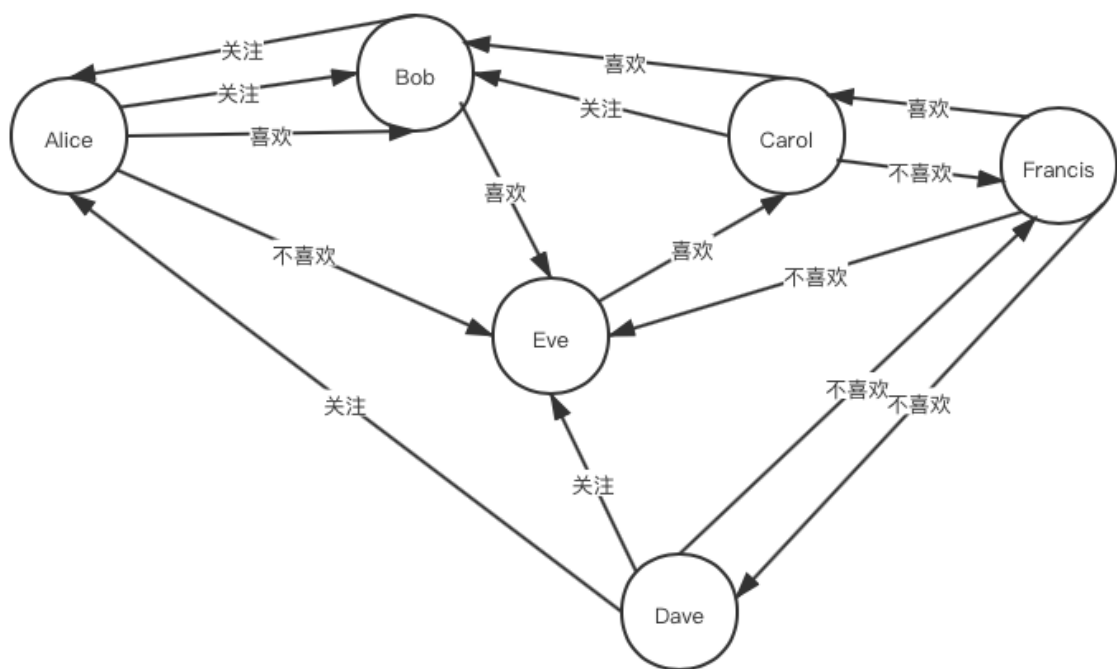
在**内核版本 v0.9.1** 中，gStore 加入了与数据图中结点间的路径和中心度相关的一系列查询，目前包括环路查询、最短路径查询、K跳可达性查询和Personalized PageRank查询。

6.5.1 示例数据

为了更好地演示高级功能，使用以下的社交关系数据作为示例数据：

```
<Alice> <关注> <Bob> .  
<Alice> <喜欢> <Bob> .  
<Alice> <不喜欢> <Eve> .  
<Bob> <关注> <Alice> .  
<Bob> <喜欢> <Eve> .  
<Carol> <关注> <Bob> .  
<Carol> <喜欢> <Bob> .  
<Carol> <不喜欢> <Francis> .  
<Dave> <关注> <Alice> .  
<Dave> <关注> <Eve> .  
<Dave> <不喜欢> <Francis> .  
<Eve> <喜欢> <Carol> .  
<Francis> <喜欢> <Carol> .  
<Francis> <不喜欢> <Dave> .  
<Francis> <不喜欢> <Eve> .
```

上述数据的图示如下：



如无特殊说明，返回路径的函数均以如下 JSON 格式字符串表示一条路径/一个环/一个子图：

```
{  
  "src": "<src_IRI>", "dst": "<dst_IRI>",  
  "edges": [  
    { "fromNode": 0, "toNode": 1, "predIRI": "<pred>" }  
  ],  
  "nodes": [  
    { "nodeIndex": 0, "nodeIRI": "<src_IRI>" },  
    { "nodeIndex": 1, "nodeIRI": "<dst_IRI>" }  
  ]  
}
```

最终返回值以如下形式表示一组路径/一组环/一组子图：（其中 `paths` 的元素为上述格式）

```
{ "paths": [{...}, {...}, ...] }
```

6.5.2 路径相关查询

6.5.1.1 环路查询

查询是否存在包含结点 `u` 和 `v` 的一个环。

```
cyclePath(u, v, directed, pred_set)
cycleBoolean(u, v, directed, pred_set)
```

用于 SELECT 语句中，与聚合函数使用语法相同。

参数

`u, v`：变量或结点 IRI

`directed`：布尔值，为真表示有向，为假表示无向（图中所有边视为双向）

`pred_set`：构成环的边上允许出现的谓词集合。若设置为空 `{}`，则表示允许出现数据中的所有谓词

返回值

- `cyclePath`：以 JSON 形式返回包含结点 `u` 和 `v` 的一个环（若存在）。若 `u` 或 `v` 为变量，对变量的每组有效值返回一个环。
- `cycleBoolean`：若存在包含结点 `u` 和 `v` 的一个环，返回真；否则，返回假。

下面的查询询问是否存在包含 Carol、一个 Francis 不喜欢的人（示例数据中即为 Dave 或 Eve），且构成它的边只能由“喜欢”关系标记的有向环：

```
select (cycleBoolean(?x, <Carol>, true, {<喜欢>}) as ?y) where {      <Francis> <不喜欢> ?x . }
```

结果如下：

`?y`

"true"^^<<http://www.w3.org/2001/XMLSchema#>>

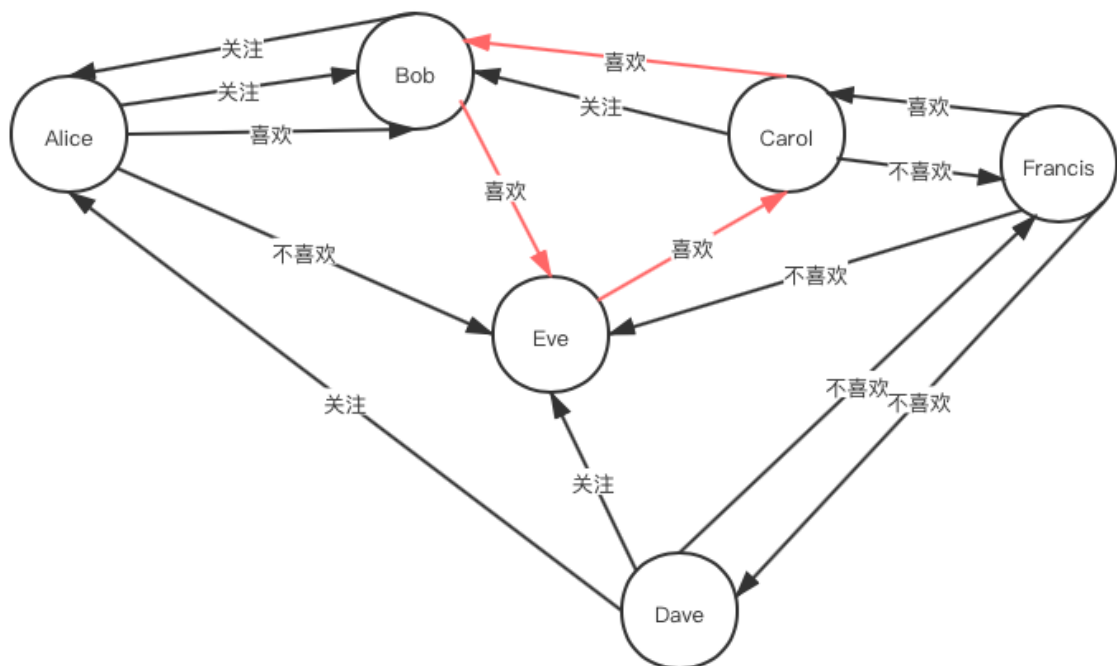
如果希望输出一个满足以上条件的环，则使用下面的查询：

```
SELECT (cyclePath(?x, <Carol>, true, {<喜欢>}) as ?y)
WHERE
{
    <Francis> <不喜欢> ?x .
}
```

结果如下，可见其中一个满足条件的环由 Eve 喜欢 Carol - Carol 喜欢 Bob - Bob 喜欢 Eve 顺次构成：（为方便阅读，省略了字符串最外层的双引号和内部双引号的转义）

```
{
  "paths": [{
    "src": "<Eve>",
    "dst": "<Carol>",
    "edges":
      [{"fromNode": 2, "toNode": 3, "predIRI": "<喜欢>"},
      {"fromNode": 3, "toNode": 1, "predIRI": "<喜欢>"}, {"fromNode": 1, "toNode": 2, "predIRI": "<喜欢>"}],
    "nodes":
      [{"nodeIndex": 1, "nodeIRI": "<Bob>"}, {"nodeIndex": 3, "nodeIRI": "<Carol>"},
      {"nodeIndex": 2, "nodeIRI": "<Eve>"}]
  }]
}
```

下图标红的部分即为这个环：



6.5.1.2 最短路径查询

查询从结点 **u** 到结点 **v** 的最短路径。

```
shortestPath(u, v, directed, pred_set)
shortestPathLen(u, v, directed, pred_set)
```

用于 SELECT 语句中，与聚合函数使用语法相同。

参数

u, **v** : 变量或结点 IRI

directed : 布尔值，为真表示有向，为假表示无向（图中所有边视为双向）

pred_set : 构成最短路径的边上允许出现的谓词集合。若设置为空 **{}**，则表示允许出现数据中的所有谓词

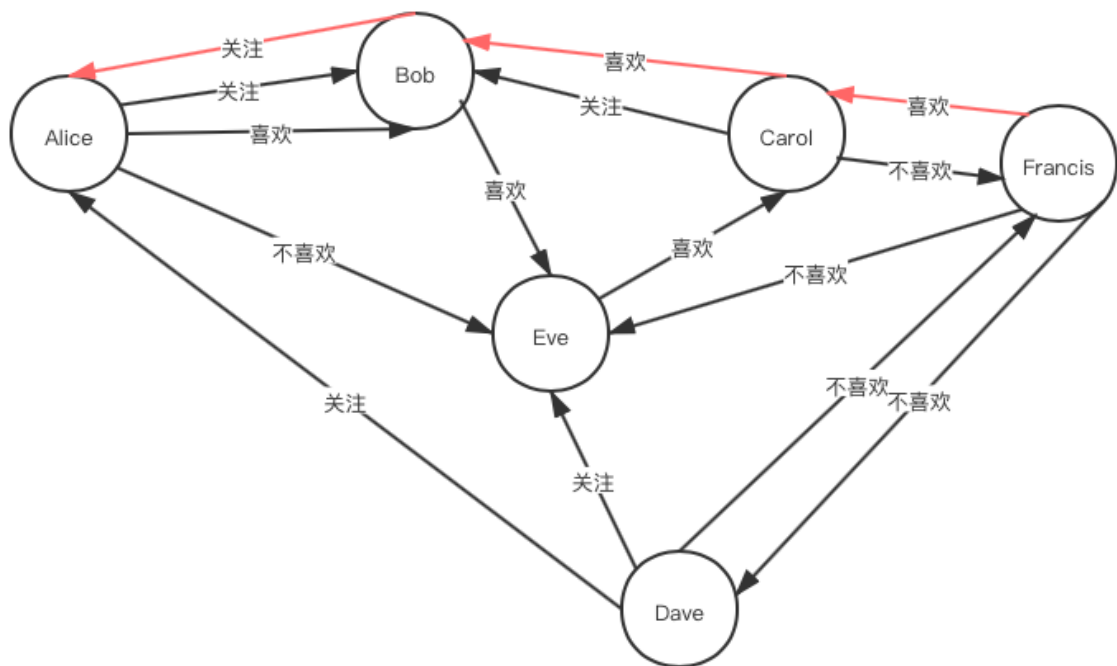
返回值

- `shortestPath` : 以 JSON 形式返回从结点 `u` 到 `v` 的一条最短路径 (若可达)。若 `u` 或 `v` 为变量, 对变量的每组有效值返回一条最短路径。
- `shortestPathLen` : 返回从结点 `u` 到 `v` 的最短路径长度 (若可达)。若 `u` 或 `v` 为变量, 对变量的每组有效值返回一个最短路径长度数值。

下面的查询返回从 Francis 到一个 Bob 喜欢、关注或不喜欢, 且没有被 Francis 不喜欢的人 (示例数据中即为 Alice) 的最短路径, 边上的关系可以是喜欢或关注:

```
SELECT (shortestPath(<Francis>, ?x, true, {<喜欢>, <关注>})) AS ?y)WHERE{    <Bob>
?pred ?x .    MINUS { <Francis> <不喜欢> ?x . }}
```

下图标红的部分即为这条最短路径:



结果如下: (为方便阅读, 省略了字符串最外层的双引号和内部双引号的转义)

```
{  "paths": [{    "src": "<Francis>",    "dst": "<Alice>",    "edges":    [{ "fromNode": 4, "toNode": 3, "predIRI": "<喜欢>" },    { "fromNode": 3, "toNode": 1, "predIRI": "<喜欢>" }, { "fromNode": 1, "toNode": 0, "predIRI": "<关注>" }],    "nodes":    [{ "nodeIndex": 0, "nodeIRI": "<Alice>" },    { "nodeIndex": 1, "nodeIRI": "<Bob>" }, { "nodeIndex": 3, "nodeIRI": "<Carol>" },    { "nodeIndex": 4, "nodeIRI": "<Francis>" } ]}]}
```

如果希望只输出最短路径长度, 则使用下面的查询:

```
SELECT (shortestPathLen(<Francis>, ?x, true, {<喜欢>, <关注>})) AS ?y)WHERE{ <Bob>
?pred ?x .    MINUS { <Francis> <不喜欢> ?x . }}
```

结果如下: (为方便阅读, 省略了字符串最外层的双引号和内部双引号的转义)

```
{ "paths": [{ "src": "<Francis>", "dst": "<Alice>", "length": 3 } ] }
```

6.5.1.3 可达性 / K 跳可达性查询

查询从结点 **u** 到结点 **v** 是否可达 / 是否 K 跳可达（即存在以 **u** 为起点、以 **v** 为终点，长度小于或等于 **k** 的路径）。

```
kHopReachable(u, v, directed, k, pred_set) kHopReachablePath(u, v, directed, k, pred_set)
```

参数

u, v : 变量或结点 IRI

k : 若置为非负整数，则为路径长度上限（查询 K 跳可达性）；若置为负数，则查询可达性

directed : 布尔值，为真表示有向，为假表示无向（图中所有边视为双向）

pred_set : 构成路径的边上允许出现的谓词集合。若设置为空 **{}**，则表示允许出现数据中的所有谓词

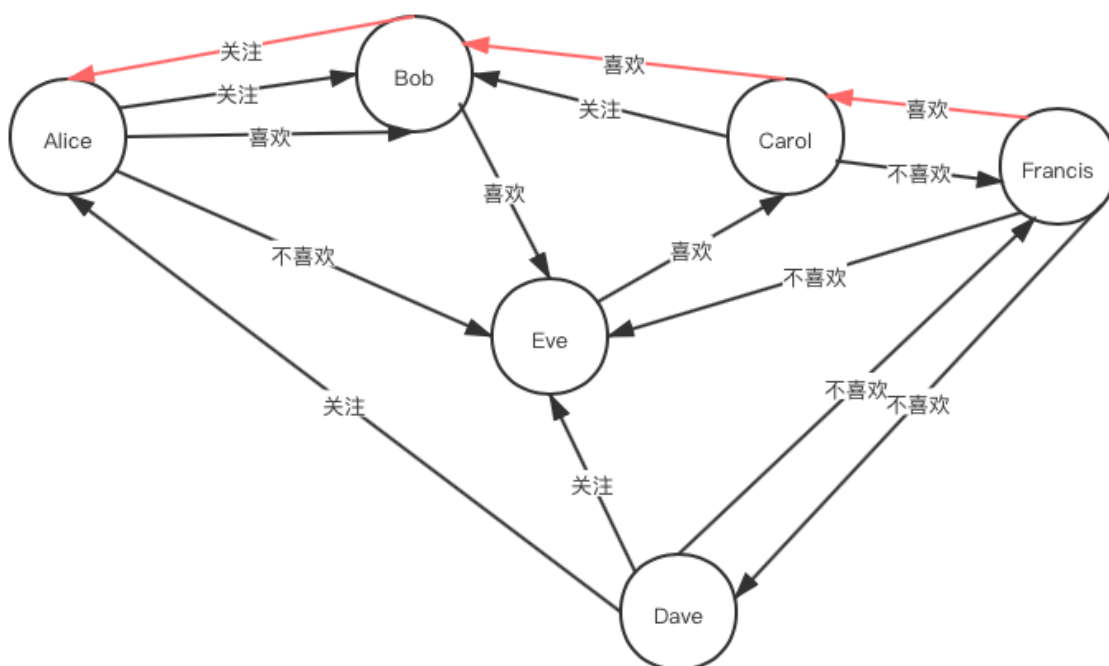
返回值

- kHopReachable**: 若从结点 **u** 到结点 **v** 可达（或 K 跳可达，取决于参数 **k** 的取值），返回真；否则，返回假。若 **u** 或 **v** 为变量，对变量的每组有效值返回一个真/假值。
- kHopReachablePath**: 返回任意一条从结点 **u** 到结点 **v** 的路径（若可达）或 K 跳路径，即长度小于或等于 **k** 的路径（若 K 跳可达，取决于参数 **k** 的取值）。若 **u** 或 **v** 为变量，对变量的每组有效值返回一条路径（若可达）或 K 跳路径（若 K 跳可达）。

下面的查询效仿上一节“最短路径查询”中的示例查询：起点为 Francis，终点为一个 Bob 喜欢、关注或不喜欢，且没有被 Francis 不喜欢的人（示例数据中即为 Alice）。询问这两人之间是否通过喜欢或关注关系 2 跳或以内可达。

```
SELECT (kHopReachable(<Francis>, ?x, true, 2, {<喜欢>, <关注>})) AS ?y WHERE {  
<Bob> ?pred ?x . MINUS { <Francis> <不喜欢> ?x . }}
```

由于已知满足条件的最短路径长度为 3：



因此上述查询的结果为假：

```
{"paths":[{"src":"<Francis>","dst":"<Alice>","value":"false"}]}
```

另一方面，Francis 和 Alice 之间是可达的，只是最短路径长度超出了上述限制。因此若查询可达性（将 `k` 设置为负数），则会返回真：

```
SELECT (kHopReachable(<Francis>, ?x, true, -1, {<喜欢>, <关注>}) AS ?y)WHERE{<Bob> ?pred ?x . MINUS { <Francis> <不喜欢> ?x . }}
```

结果如下：

```
{"paths":[{"src":"<Francis>","dst":"<Alice>","value":"true"}]}
```

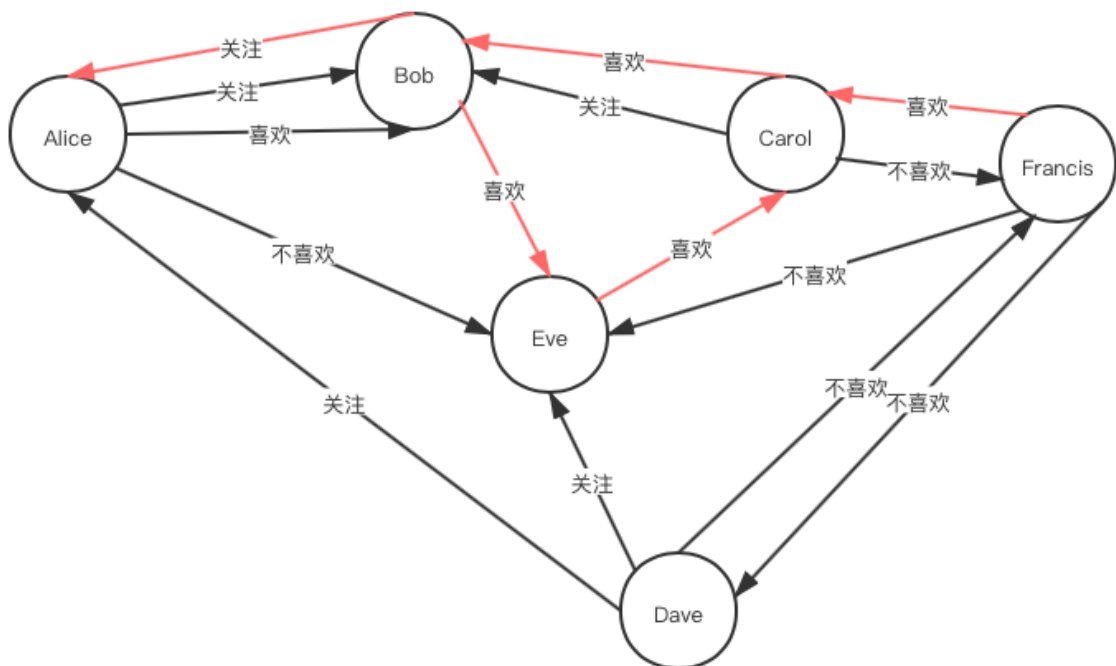
若希望返回一条两人之间满足条件的路径，则可以调用 `kHopReachablePath` 函数：

```
SELECT (kHopReachablePath(<Francis>, ?x, true, -1, {<喜欢>, <关注>}) AS ?y)WHERE{<Bob> ?pred ?x . MINUS { <Francis> <不喜欢> ?x . }}
```

此时结果可能为上述最短路径：

```
{  "paths":[{"src":"<Francis>","dst":"<Alice>","edges":
[{"fromNode":4,"toNode":3,"predIRI":"<喜欢>"},
{"fromNode":3,"toNode":1,"predIRI":"<喜欢>"},{"fromNode":1,"toNode":0,"predIRI":"
<关注>"}]},  "nodes":    [{"nodeIndex":0,"nodeIRI":"<Alice>"},
{"nodeIndex":1,"nodeIRI":"<Bob>"}, {"nodeIndex":3,"nodeIRI":"<Carol>"},
{"nodeIndex":4,"nodeIRI":"<Francis>"}]  }}}
```

也可能是下图中含有环的、同样满足条件的非最短路径：



6.5.2 Personalized PageRank

计算相对于 u 的 PPR 值。

```
PPR(u, {v1, v2, ...}, pred_set)
PPR(u, k, pred_set)
```

参数

u：变量或 IRI，表示源结点

{v1, v2, ...}：变量或 IRI 列表，表示目标结点。若列表为空，默认为图中所有结点均为目标结点

k：整数，表示返回 PPR 值为 Top-k 的结点 IRI 及其对应的 PPR（若总结点数不足 k 个，则返回所有结点 IRI 及其对应的 PPR）

pred_set：考虑的谓词集合（若给出空，则默认为考虑所有谓词）

返回值

返回值为以下形式，其中 src 为 u 对应的 IRI 或变量查询出的结果； dst 含有哪些目标结点取决于函数的第二个参数；对应的 PPR 值为双精度浮点数。

```
{ "paths":
  [
    { "src": "<Francis>",
      [ { "dst": "<Alice>", "PPR": 0.1 }, { "dst": "<Bob>", "PPR": 0.01 }, ... ]
    }
  ],
  ...
}
```

例子

```
PREFIX ns:    <http://example.org/ns#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>

SELECT (PPR(ns:a, 10, {ns:pred2}) AS ?z)
WHERE
{
  ?y ns:pred1 ns:b .
}
```

7. gStore可视化工具Workbench

7.1 安装和部署

gStore Workbench是gStore团队开发用于在线管理gStore图数据库及对gStore进行查询可视化的web工具，目前gStore官网提供workbench下载，下载链接为<http://www.gstore.cn>，选择【产品】-【gstore workbench】，填入相关信息后，您将获取一个workbench 压缩包，但需要安装和部署，下面将详细介绍安装部署的步骤。

7.2 下载tomcat

workbench 是一个web网站，需要一个web服务器作为web容器来运行，我们推荐采用tomcat8作为web服务器，下载地址为<https://tomcat.apache.org/download-80.cgi>。下载压缩包之后要解压。

- 把workbench压缩包放到tomcat的webapps目录并解压
- 到tomcat的bin目录下
- 启动tomcat:
[root@node1 bin]# ./startup.sh
- 停止tomcat:

```
[root@node1 bin]# ./shutdown.sh
```

7.3 登录

7.3.1 浏览器访问系统

登录网址为：

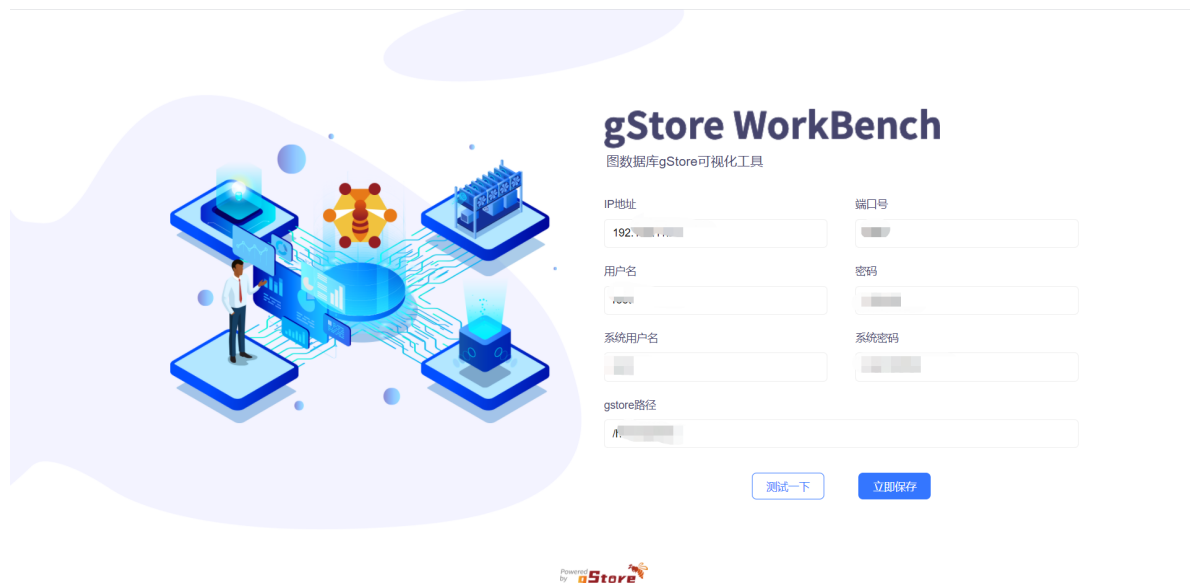
```
http://workbench自己部署的服务器ip:8080/gworkbench/views/user/login.html
```



7.3.2 连接gStore实例

设置远端服务器ip和端口保存到远端的gStore图数据库管理系统，注意远端服务器要安装gStore并启动ghttp服务

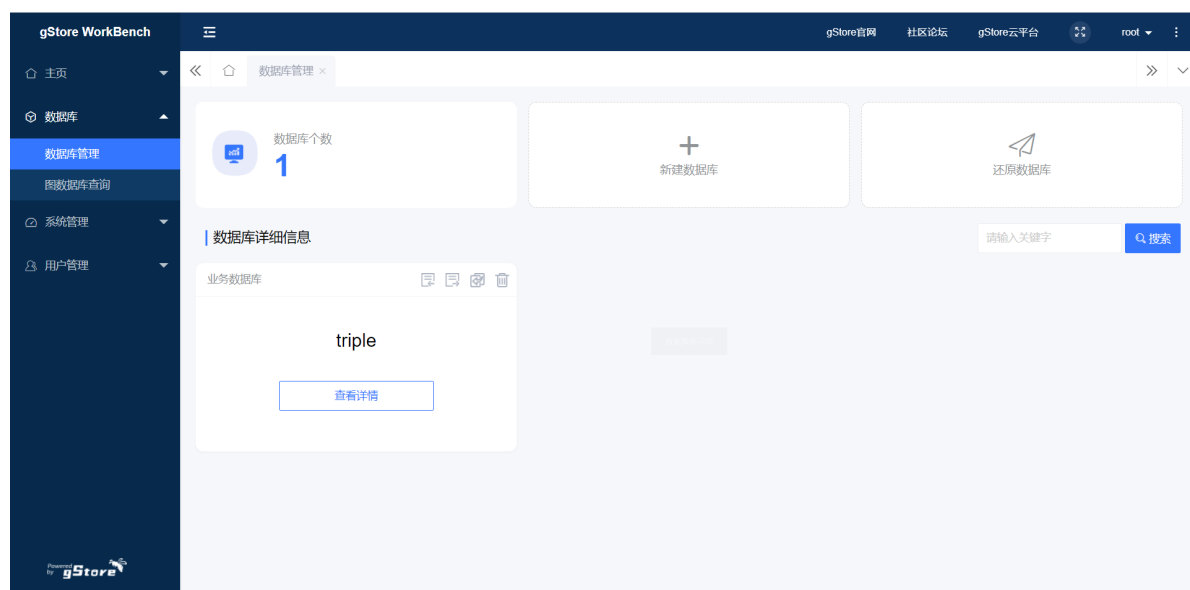
输入用户名、密码和验证码登录到已保存服务器上的gStore图数据库管理系统（gstore默认用户名为root，密码为12345）



7.4 查询功能

7.4.1 数据库管理

- 查看已加载数据库的信息



点击数据库，会看到数据库的具体信息

- 新建数据库

1.输入新建的数据库名称，如lubm

2.有两种方式上传文件：

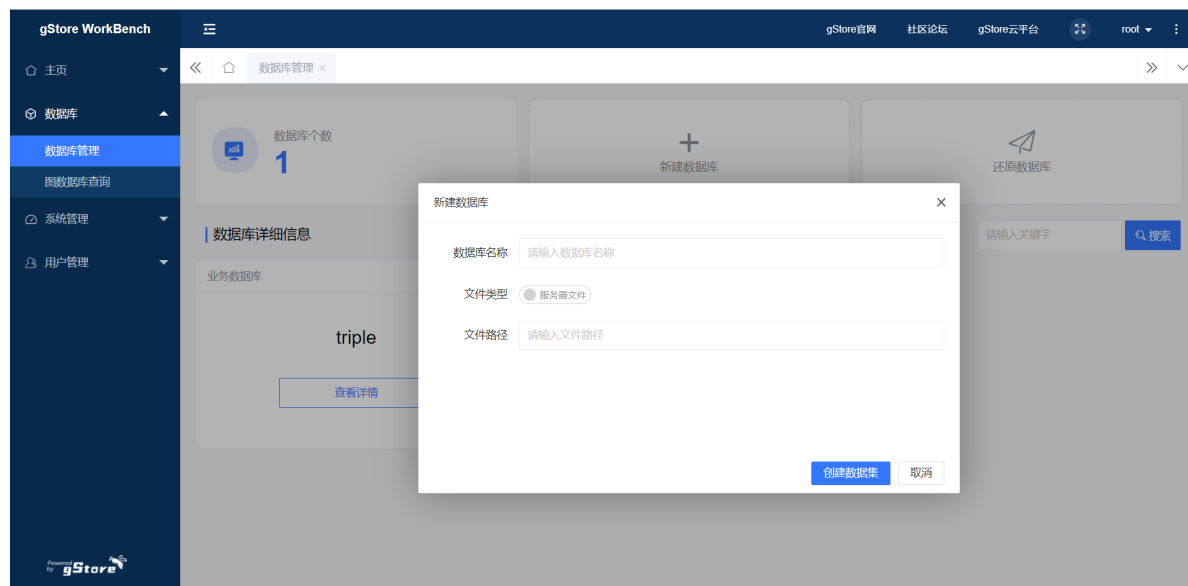
一种是从服务器上传，输入正确的nt文件或n3文件路径，可以输入绝对路径或相对路径，若是想输入相对路径，注意当前路径为安装gstore的根目录。

例如：路径选择

/root/gStore/data/lubm.nt	绝对路径
./data/lubm.nt	相对路径

另外一种是从本地上传，注意使用这种方式必须保证**workbench部署的服务器与安装gStore的服务器是同一台**。首先从本地选择nt或n3文件,然后点击上传文件。

3.点击创建数据集

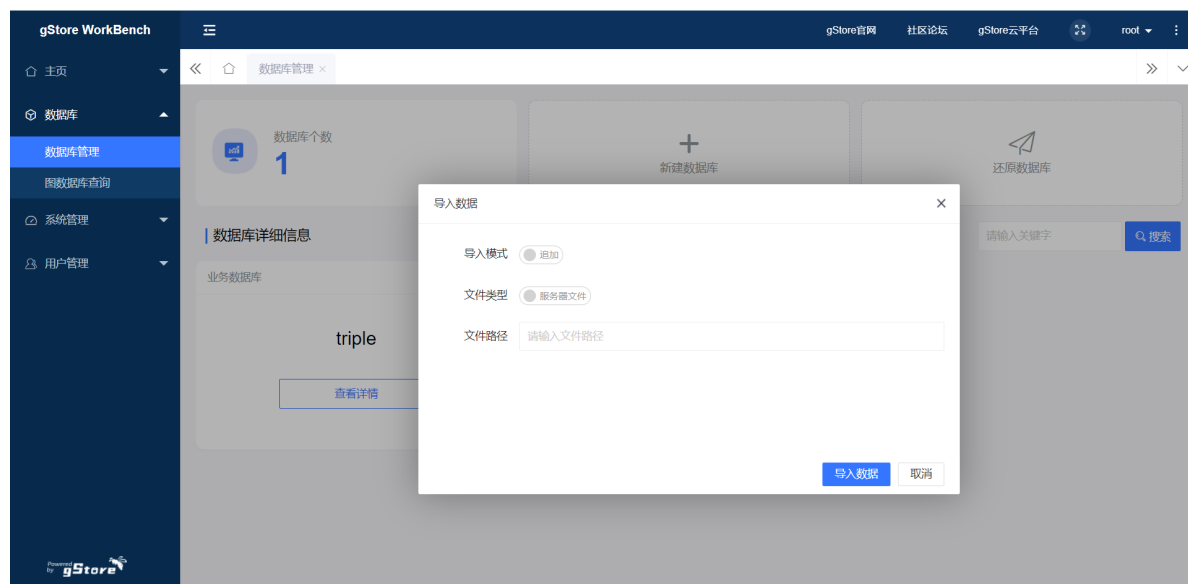


• 删除数据库

点击数据库右上角的删除按钮，选择删除或者完全删除都会删除数据库。**system数据库不能删除。**

• 导入数据

点击【数据库管理】，选择要导入的数据库，点击右上角导入图标，导入模式可选择【追加】和【覆盖】两种；文件类型可选择服务器文件和本地文件两种；入的本地文件需要选择nt或者n3格式的文件，点击【上传文件】后，继续点击【导入数据】即可。



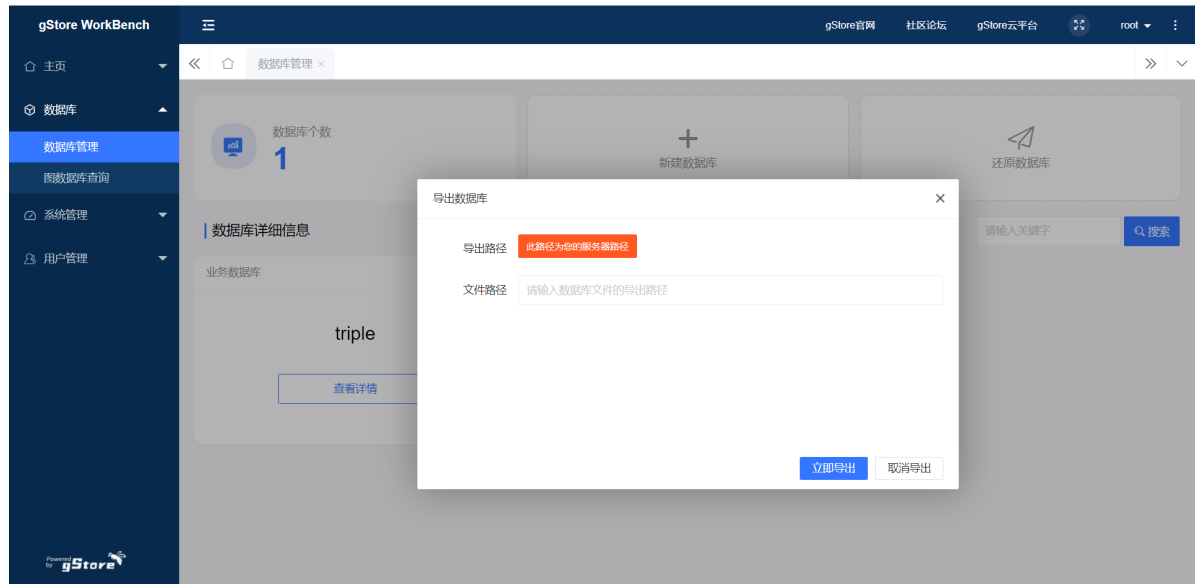
- 导出数据库

把数据库导出为nt文件，点击数据库右上角的导出按钮，选择导出的nt文件所在文件夹路径，可以输入绝对路径或相对路径，若是想输入相对路径，注意当前路径为安装gstore的根目录。

例如：路径选择

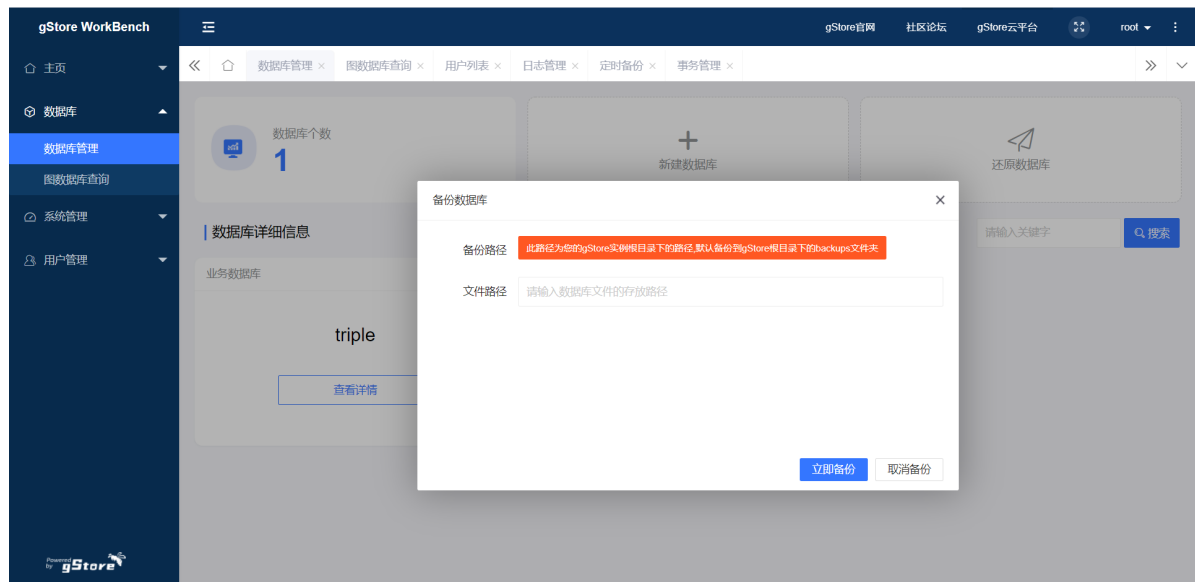
/root/gStore/data	绝对路径
./data	相对路径

输入正确的路径后点击立即导出。**system数据库不能导出。**

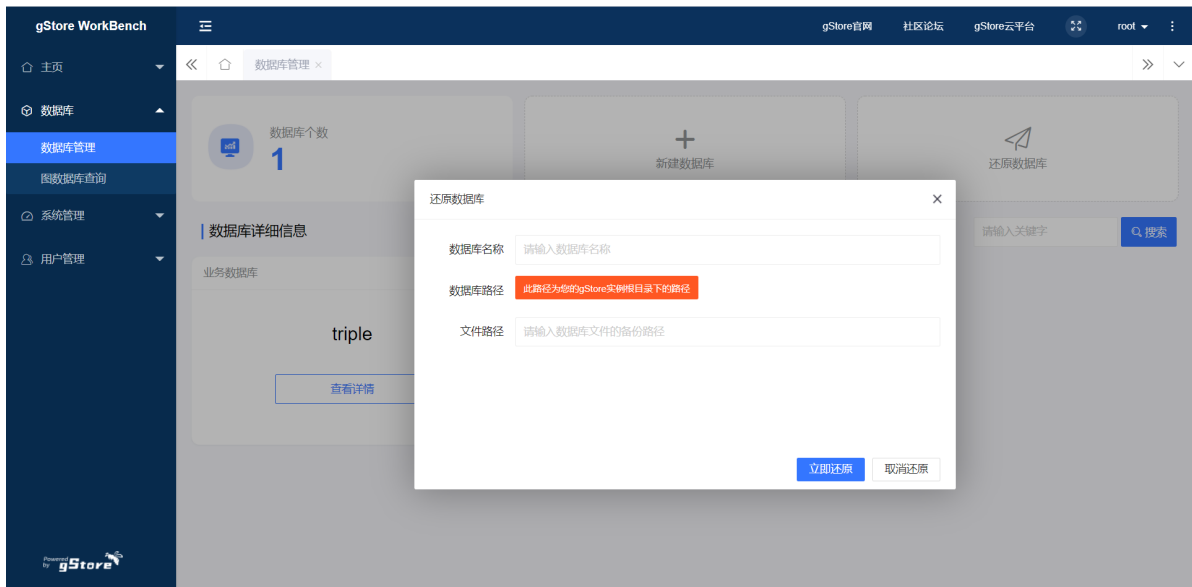


- 备份数据库

点击想要备份的数据库上的“备份”按钮，弹出如下对话框



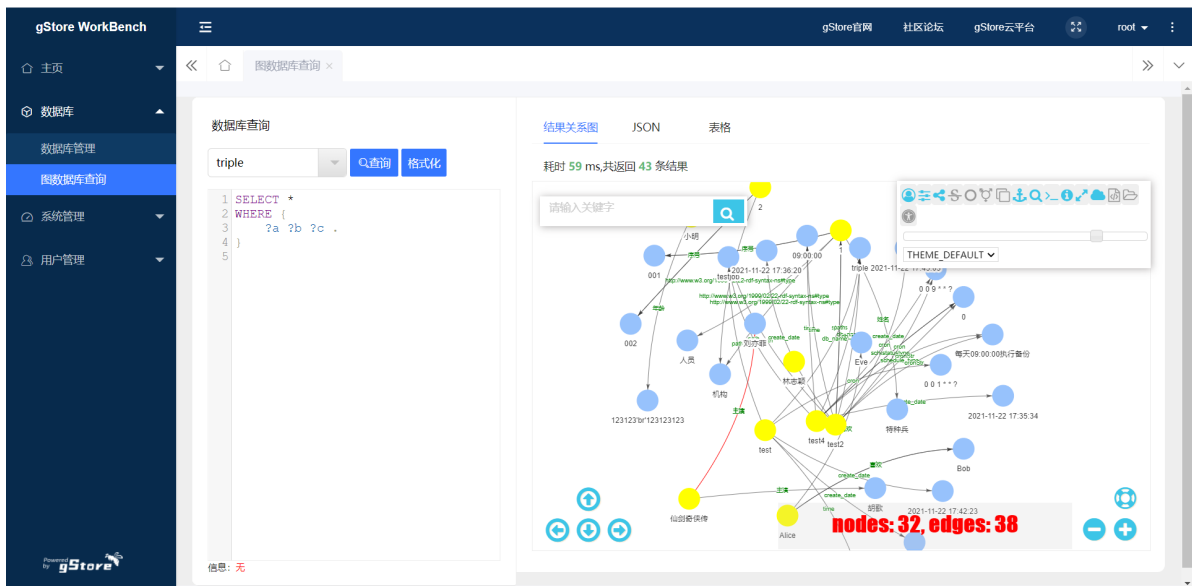
- 还原数据库



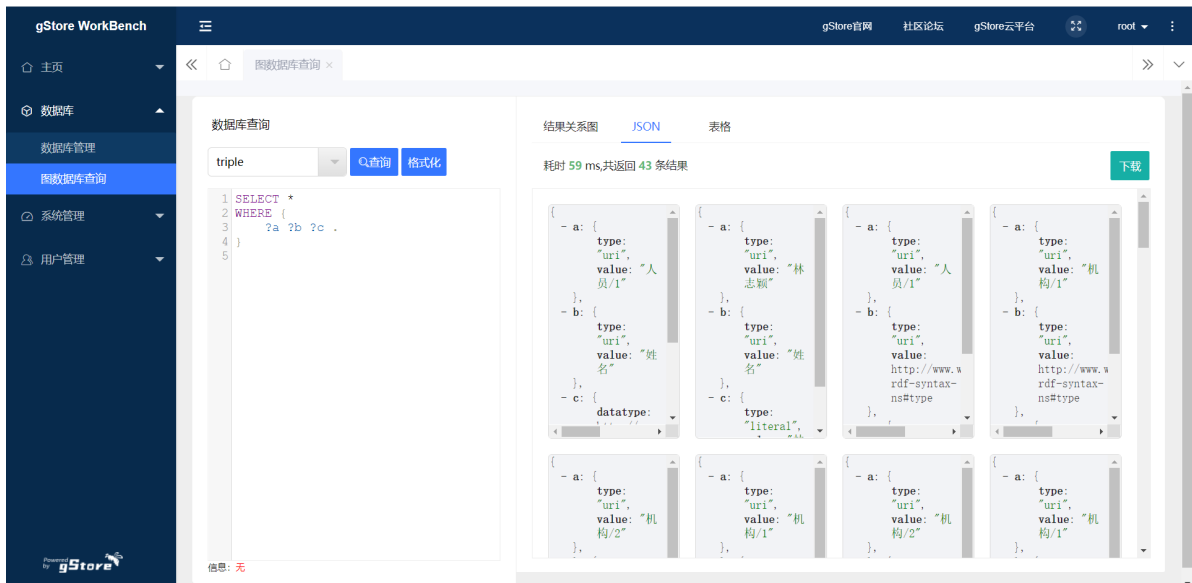
7.4.2 图数据库查询

- 选择要查询的数据库
- 按照sparql文档输入查询语句，返回结果图、json和表格三种方式

图形化展示



Json展示



表格展示

The screenshot shows the gStore WorkBench interface. On the left is a sidebar with navigation options: 主页, 数据库, 数据库管理, 图数据库查询 (selected), 系统管理, and 用户管理. The main area is titled '图数据库查询' and contains a query editor with a triple query:

```
1 SELECT *
2 WHERE {
3   ?a ?b ?c .
4 }
5
```

 Below the editor is a 'Q查询' button and a '格式化' button. To the right of the editor is a '结果关系图' tab and a 'JSON' tab, with the '表格' tab selected. The table displays 11 rows of query results with columns: 序号, a_..., a_value, b_..., b_value, c_..., and c_value. The data includes various URIs and values like '人员/1', '林志颖', 'http://www.w3...', '内码', '序号', '年龄', and '机构/1'.

序号	a_...	a_value	b_...	b_value	c_...	c_value
1	uri	人员/1	uri	姓名	ty...	特种兵
2	uri	林志颖	uri	姓名	lit...	林志颖
3	uri	人员/1	uri	http://www.w3...	uri	人员
4	uri	机构/1	uri	http://www.w3...	uri	机构
5	uri	机构/2	uri	http://www.w3...	uri	机构
6	uri	机构/1	uri	内码	ty...	001
7	uri	机构/2	uri	内码	ty...	002
8	uri	机构/1	uri	序号	ty...	001
9	uri	机构/2	uri	序号	ty...	002
10	uri	小明	uri	年龄	uri	123123'br'1231...
11	uri	人员/1	uri	属于	uri	机构/1

7.5 用户管理（只有root用户有该权限）

7.5.1 新增用户

- 添加新用户

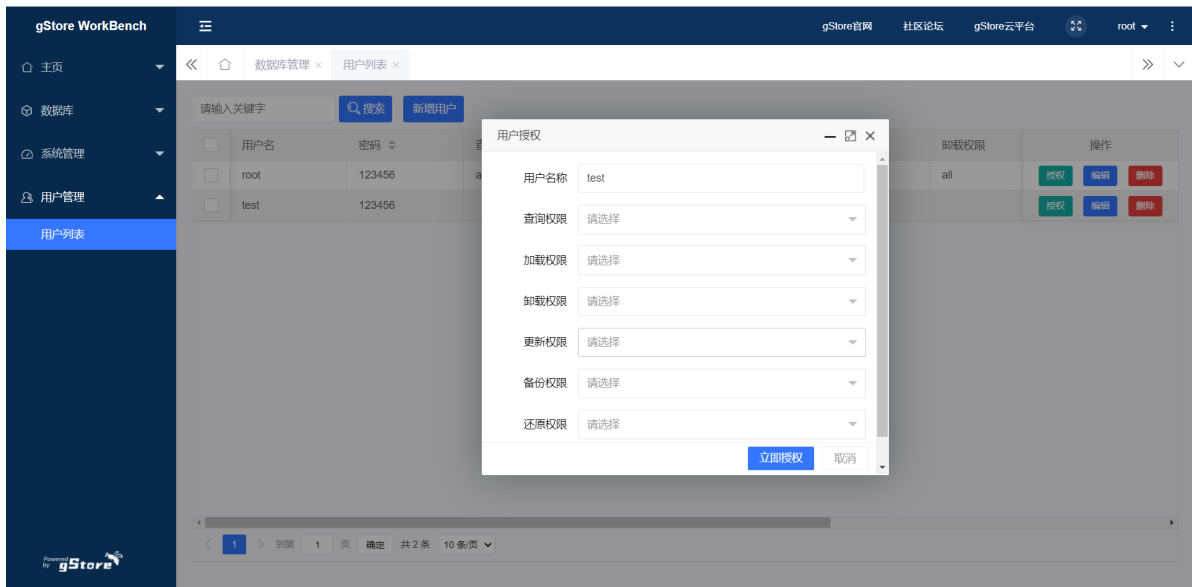
输入用户名和密码添加用户

The screenshot shows the gStore WorkBench interface with the '用户列表' tab selected. A '新增用户' dialog box is open in the center, prompting the user to enter a '用户名' (username), '密码' (password), and '确认密码' (confirm password). The background shows a table of existing users with columns for '用户名' and '密码'. The table contains two entries: 'root' with password '123456' and 'test' with password '123456'. The dialog box has '提交' (submit) and '取消' (cancel) buttons at the bottom.

7.5.2 用户授权

- 对用户进行功能授权

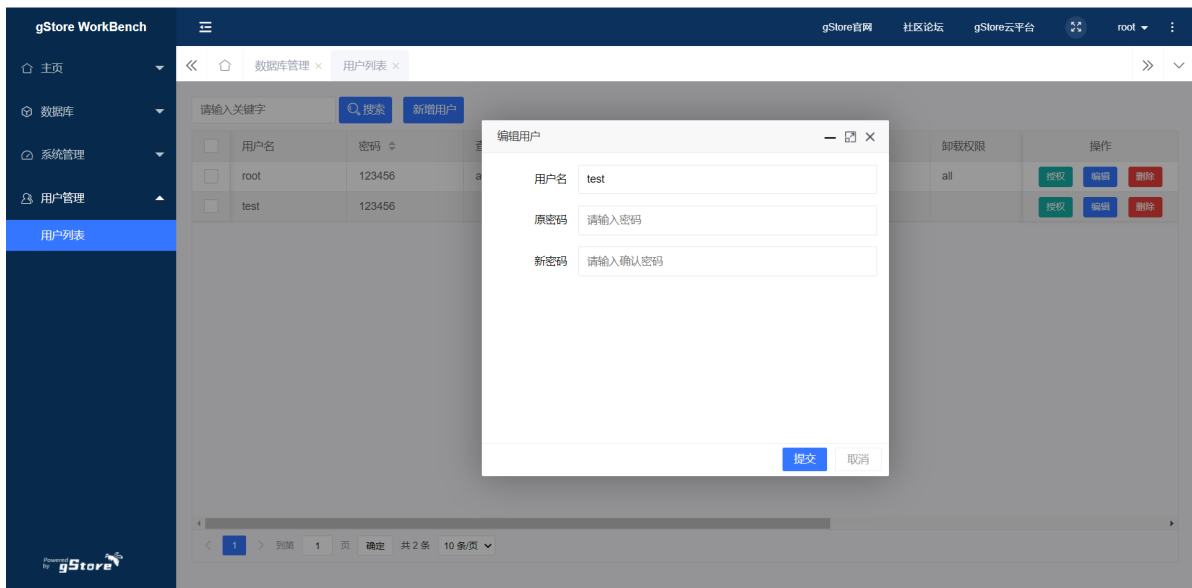
选择需要授权的用户和数据库，添加或删除查询、加载、卸载、更新、备份、还原和导出权限



7.5.3 账户编辑

- 对用户账户的具体信息进行编辑

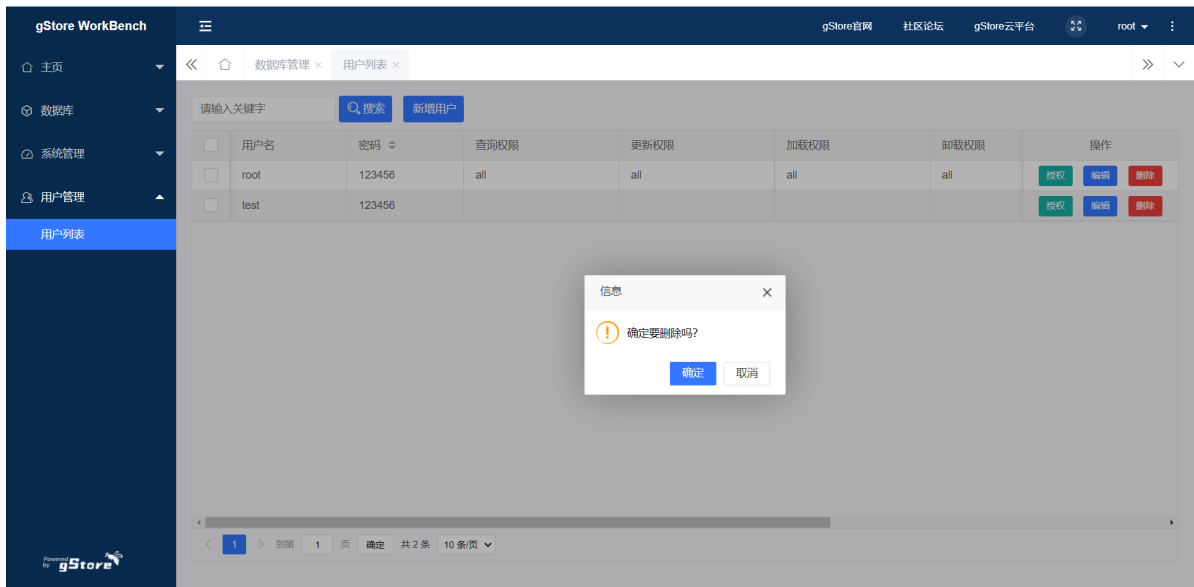
点击【用户管理】，选择某一用户账户，点击操作栏下的【编辑】，输入相关信息后点击【提交】，即可对用户密码进行修改



7.5.4 账户删除

- 对用户账户进行删除

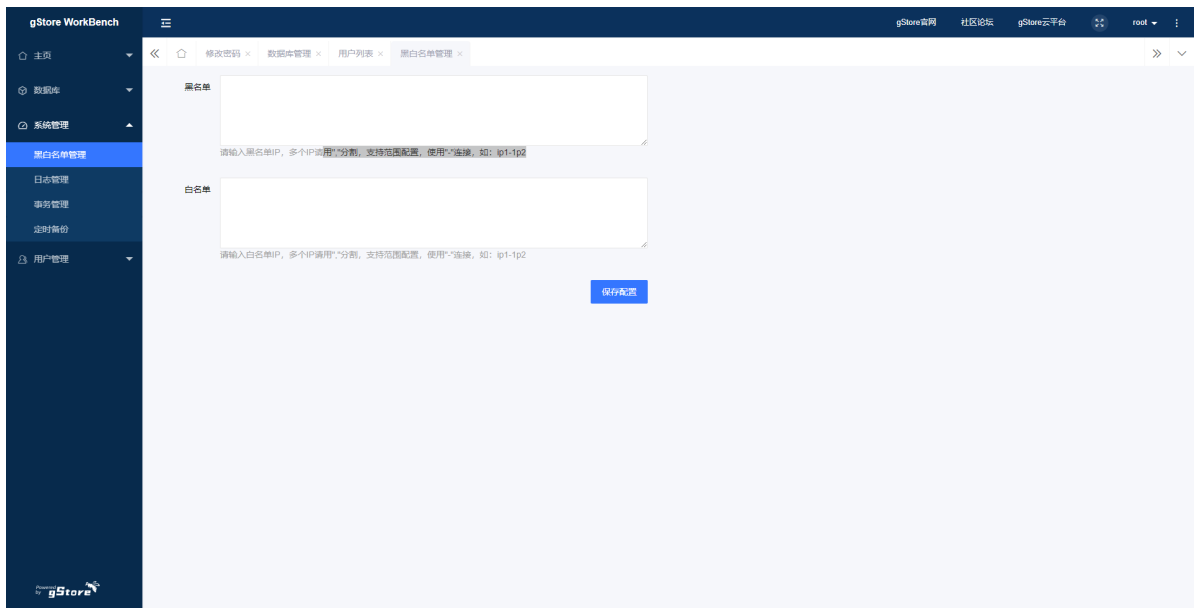
点击【用户管理】，选择某一用户账户，点击操作栏下的【删除】即可删除该用户



7.6 系统管理

7.6.1 黑白名单管理

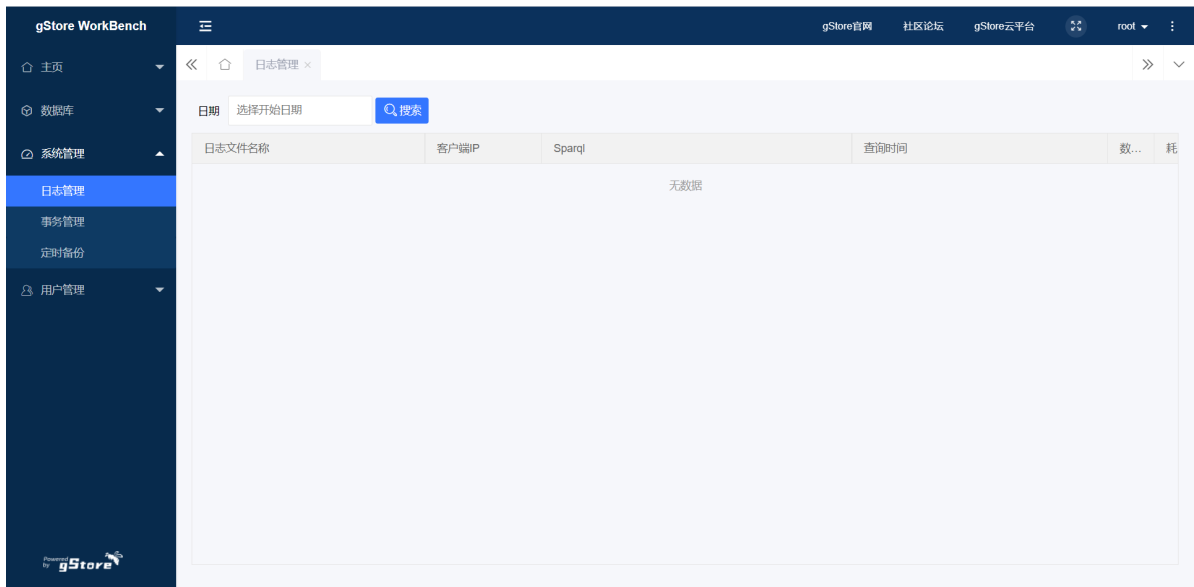
输入黑白名单IP，用","分割，支持范围配置，使用 "-" 连接，如：ip1-1p2



7.6.2 日志管理

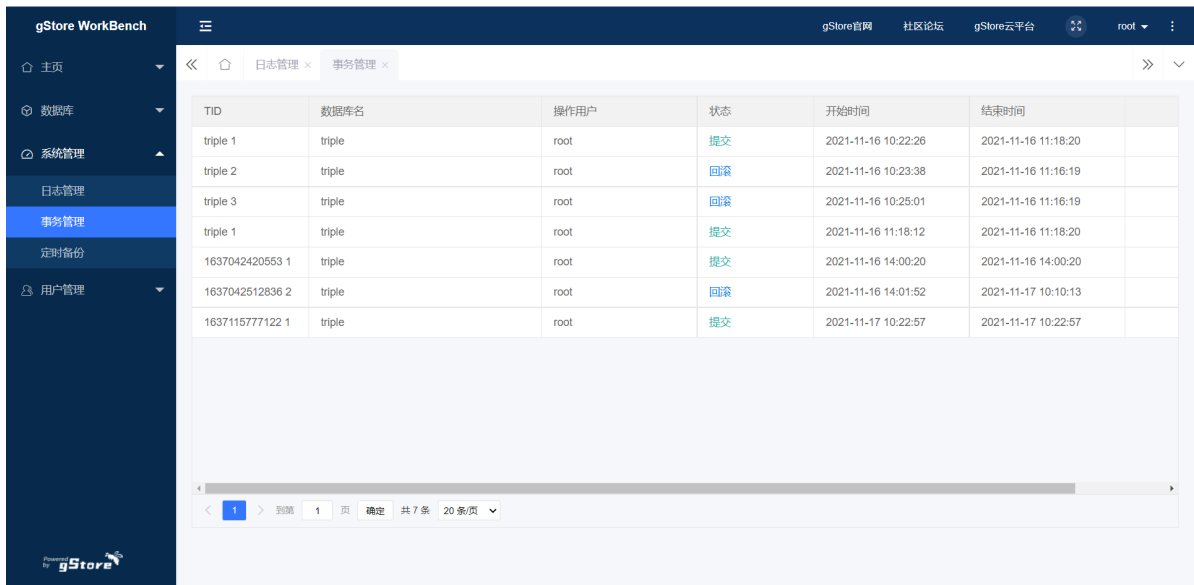
- 用户可以在web界面上查看系统日志

点击【系统管理】—【日志管理】，就可以查看到具体的日志信息，包括日志文件名称、客户端IP、SPARQL、查询时间、数据格式、耗时（毫秒）和结果数择需要授权的用户和数据库，添加或删除查询、加载、卸载、更新、备份、还原和导出权限



7.6.3 事务管理

点击【系统管理】—【事务管理】查看具体事务信息，包括TID、数据库名、操作用户、状态、开始时间、结束时间等。



7.6.4 定时备份

- 对用户账户进行删除

点击【定时备份】，然后点击【新增任务】，依次填入定时方式、任务名称、数据库名称和备份路径，就可以增加新的备份任务

gStore WorkBench

≡

gStore官网

社区论坛

gStore云平台

root

🏠 主页

🗄️ 数据库

🔧 系统管理

📖 日志管理

📋 事务管理

🕒 定时备份

👤 用户管理

定时备份 ×

请输入任务名称

🔍 搜索

新增任务

任务名称	数据库名称	状态	备份路径	定时备份...	执行时间	创建时间	操作
yq-task	triple	运行中	/data/backup/yq-task/	定时备份	2021-11-22 16:3...	2021-11-22 16:3...	启动 编辑
task-2222	triple	运行中	/data/backup/task-2222/	定时备份	2021-11-23 09:4...	2021-11-23 09:4...	启动 编辑

新增备份任务

定时方式

选择定时方式

任务名称

请输入备份任务名称

数据库名称

请选择数据库

备份路径

请输入备份文件保存路径,格式如: /""/"" 或者 /""/""

提交

取消

<

1

>

到第

1

页

确定

共 2 条

20 条/页

8. gStore云平台用户使用手册

8.1 简介

8.1.1 gStore是什么？

gStore是一个由北京大学王选计算机所数据管理实验室研发的，基于图的RDF三元组存储的数据管理系统，可以用来管理庞大的互相联系的数据，拥有源头创新、标准系统、性能优越、自主可控四大优点。

8.1.2 gStore云平台是什么？

gStore云平台是gStore系统的云端服务版本，可以在网上注册并审核通过后使用，不需要下载安装。

8.1.3 gStore有什么用？

gStore可以用于大规模数据的处理，这让其拥有很广的用途，包括但不限于政府大数据、金融科技、智慧医疗、人工智能等。

8.1.4 gStore如何在以上事务中发挥作用？

以金融科技为例，该系统可以通过图数据库的方式进行多级股权的查询，在本例中最多可查出五层的股权关系数据。

8.2 使用方式

8.2.1 注册与登录

云平台网址: <http://cloud.gstore.cn>



★ 收藏

gStore 云服务平台

邮箱登录

密码

验证码

7wJ3

登录

[还没有账号? 点击注册](#)

Copyright 北京大学王选计算机研究所
Powered By **gStore** An Efficient Graph Database

如为第一次使用gStore云平台, 需要进行注册, 注册界面如下:

gStore 注册 已经有账号? [马上登录](#)

账号信息

账号	邮箱地址	请使用可以接收的真实邮箱作为登录账号
密码	请输入密码	6-20位字符, 可使用字母、数字或符号的组合, 不建议使用纯字母、纯数字、纯符号
重复密码	再次输入密码	

注册用户信息

姓名	请输入您的姓名
电话	请输入您的电话

单位和用途

单位名称	请输入你的所属单位
用途	请输入您的用途
验证码	请输入验证码 RRsm
☐ 同意 《gStore云服务条款》	
提交	

Copyright 北京大学王选计算机研究所
Powered By **gStore** An Efficient Graph Database

注册后经审核通过, 即可登录云端系统。

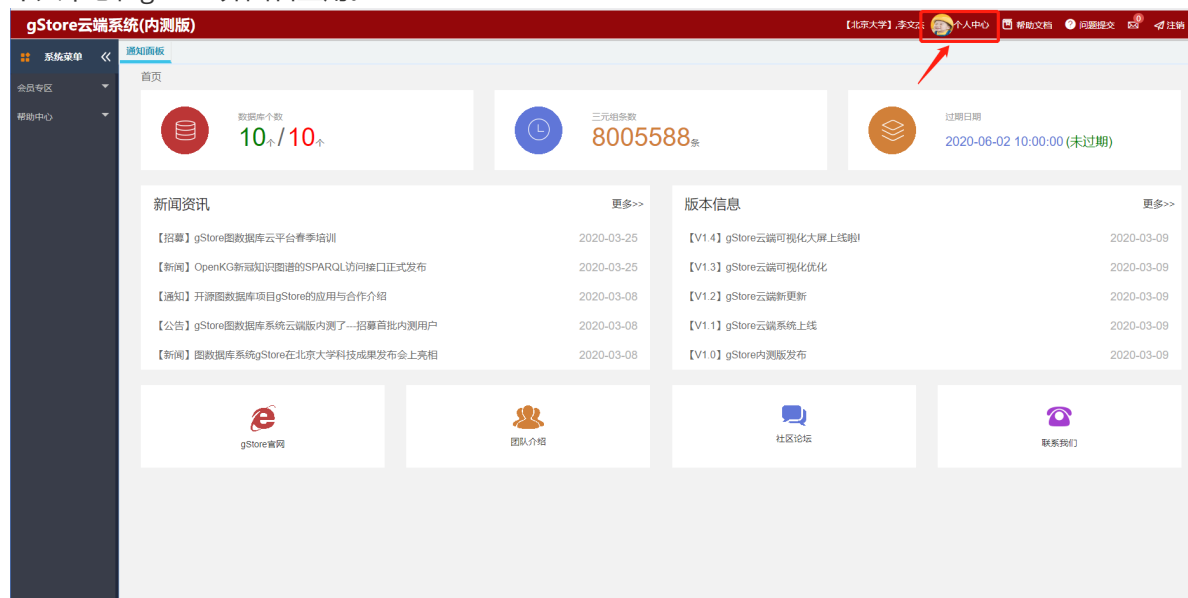
8.2.2 平台首页

平台首页如下图所示，将展示当前已构建数据库数量，三元组总数以及到期时间等，以及平台相关资讯信息（包括新闻资讯和版本信息等），点击相关资讯可以查看详情，还包括gStore官网、团队等一些常用链接信息等。



8.2.3 个人中心

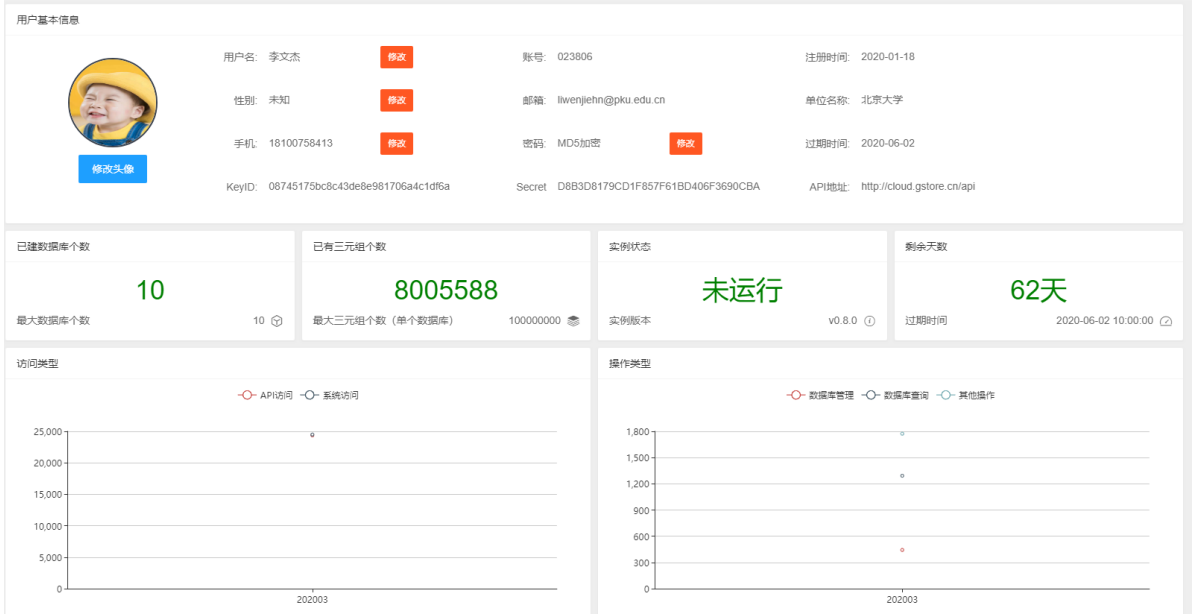
个人中心在gStore界面右上角。



进入个人中心后可以查看用户基本信息和本周操作日志。

用户基本信息中包括KeyID和Secret，这两个值用来在其他程序与gStore云平台的对接中作为密钥使

用。

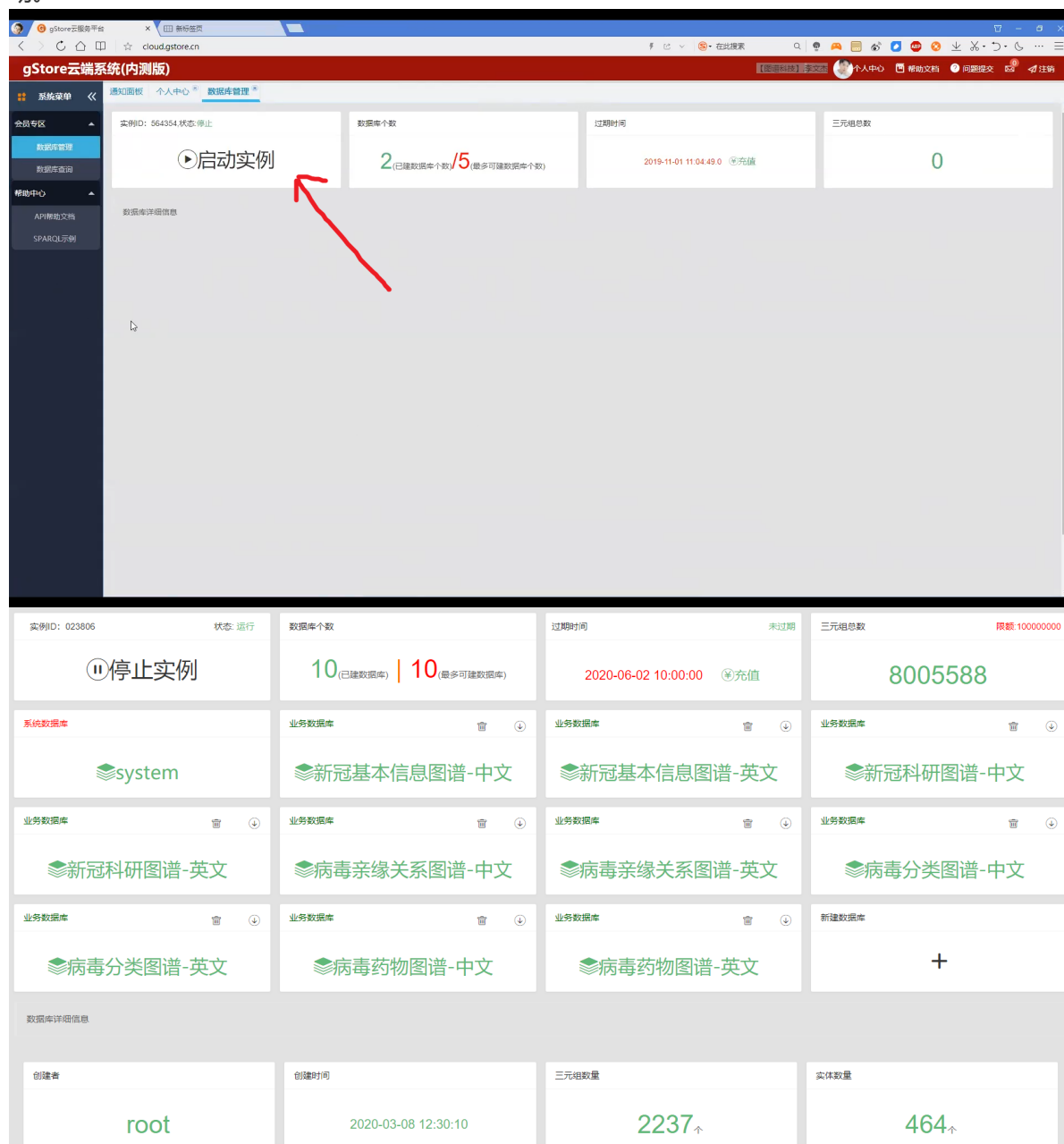


8.2.4 数据库管理

左边区域是系统菜单，包括会员专区和帮助中心两大区域。会员专区又分为两大功能模块，数据库管理和数据库查询。

8.2.4.1 启动数据库实例

在数据库管理中有一个很重要的功能：实例。在第一次进入系统时，实例可能是停止状态，需要手动启动。



8.2.4.2 查看数据库信息

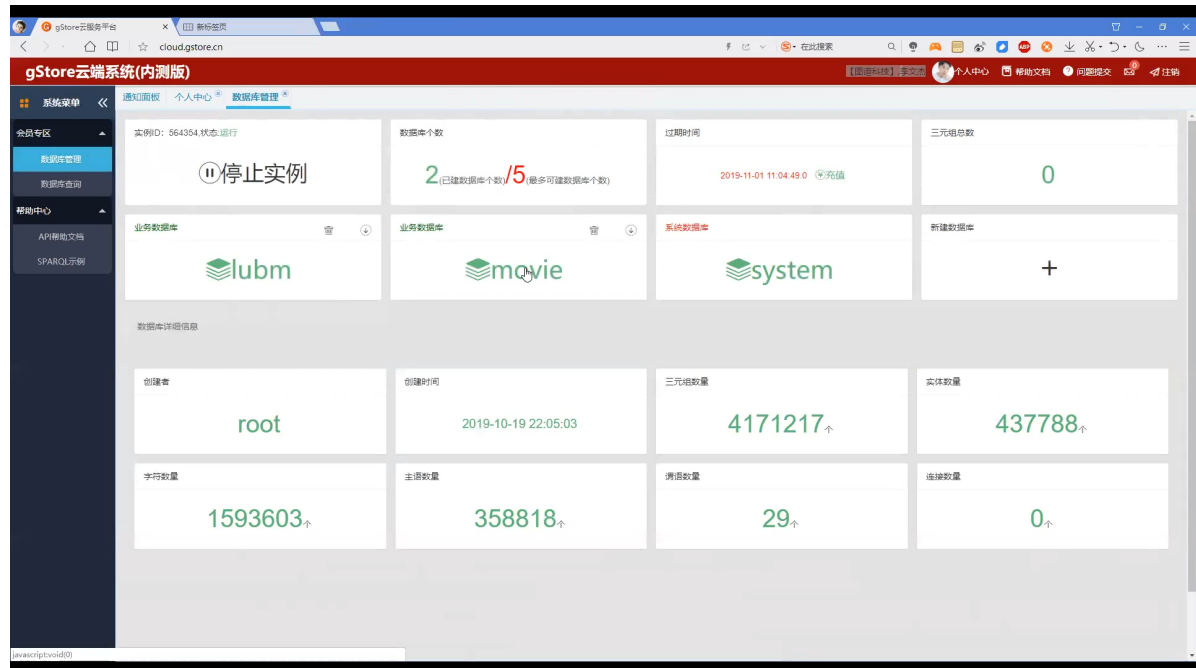
启动实例后，可以从上面一行中看到实例的状态，已创建数据库与最多可创建数据库的个数，gStore过期时间，三元组的总数等信息。

在下方，可以看到现在已创建的数据库，其中包括一个系统数据库（系统创建，不能操作，不算在最多可创建数据库个数里）和若干业务数据库（自己创建，可以操作）。

对业务数据库可以进行创建数据库，删除数据库，导出数据库，获取数据库相关信息等操作。

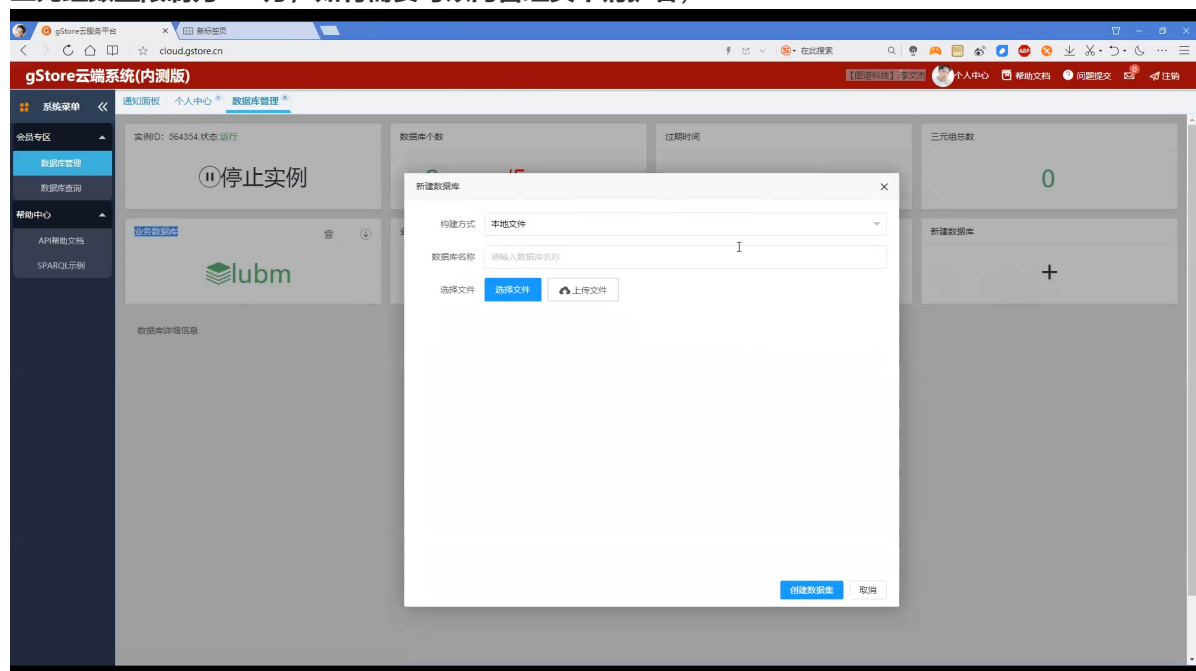
点击某个数据库可以获取其相关信息，包括创建者、创建时间、三元组数量、实体数量、字符数量、主

语数量、谓语句数量、连接数量等。如下图为movie数据库的相关信息：

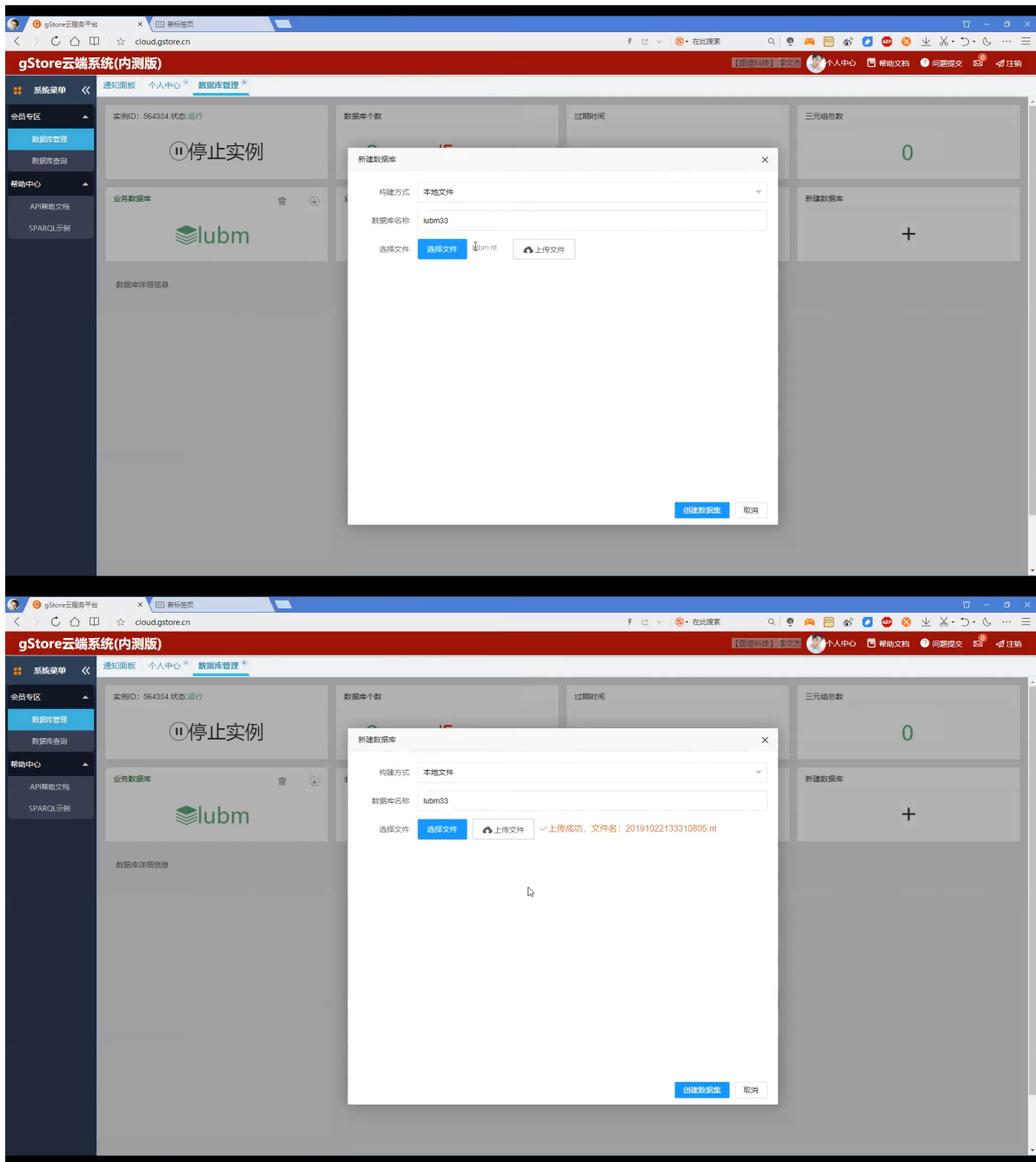


8.2.4.3 创建数据库

点击加号，可以创建数据库：（由于资源有限，目前每个用户创建数据库数量限制为5个，每个数据库三元组数量限制为100万，如有需要可以向管理员申请扩容）

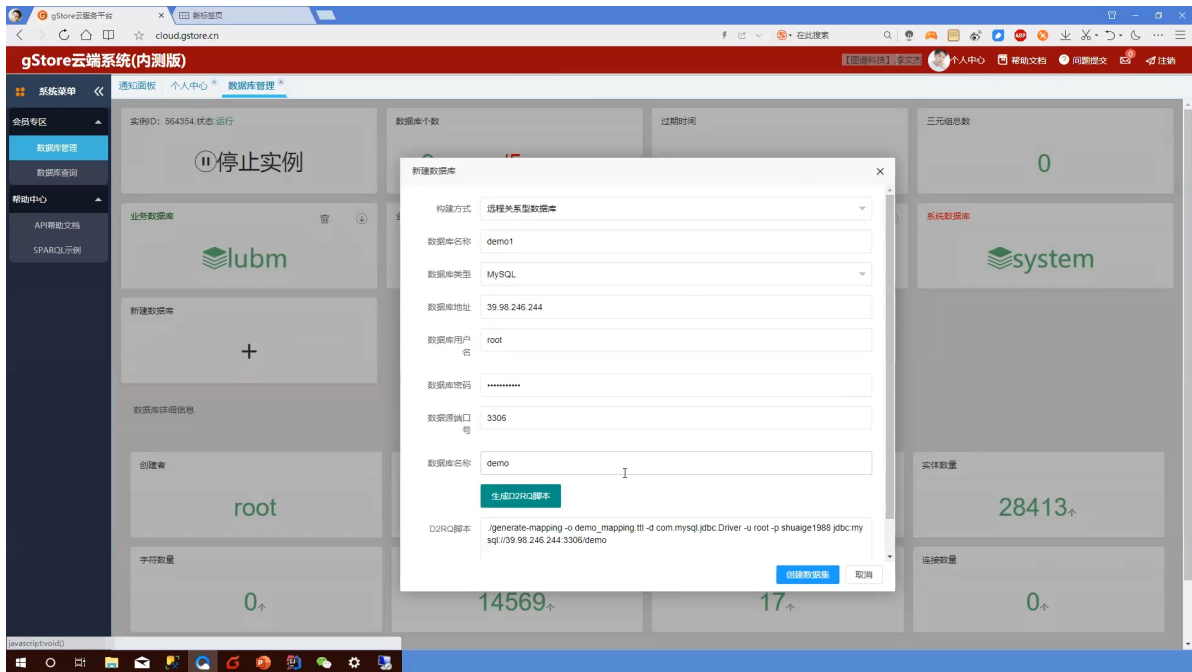


创建数据库有三种方法，第一种是本地文件，即从本地上传一个文件到服务器上。目前系统只支持nt文件，将来可能会支持n3文件。



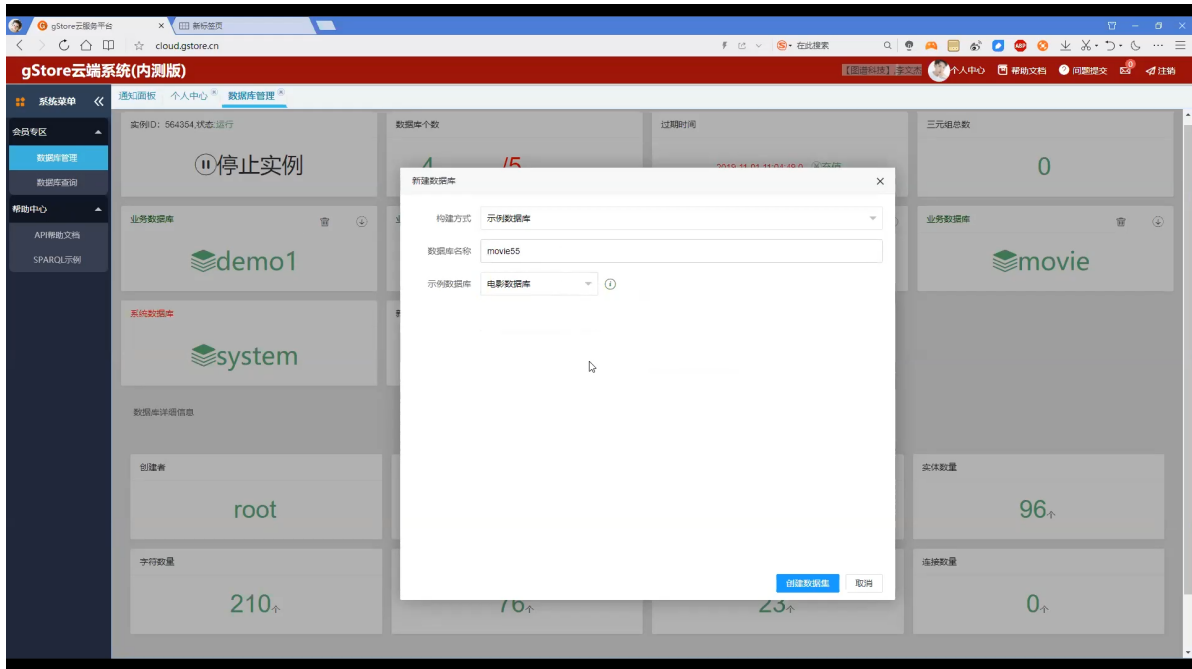
注意，文件大小不能超过2GB，行数不能超过一百万

第二种创建数据库的方法是远程关系型数据库，即远程访问网络上的数据库，将其导入到云平台上。目前云平台支持MySQL、Oracle、SQLServer、Postgre四种关系型数据库。创建数据库时需输入其相关信息，再生成D2RQ脚本，即可生成数据库。



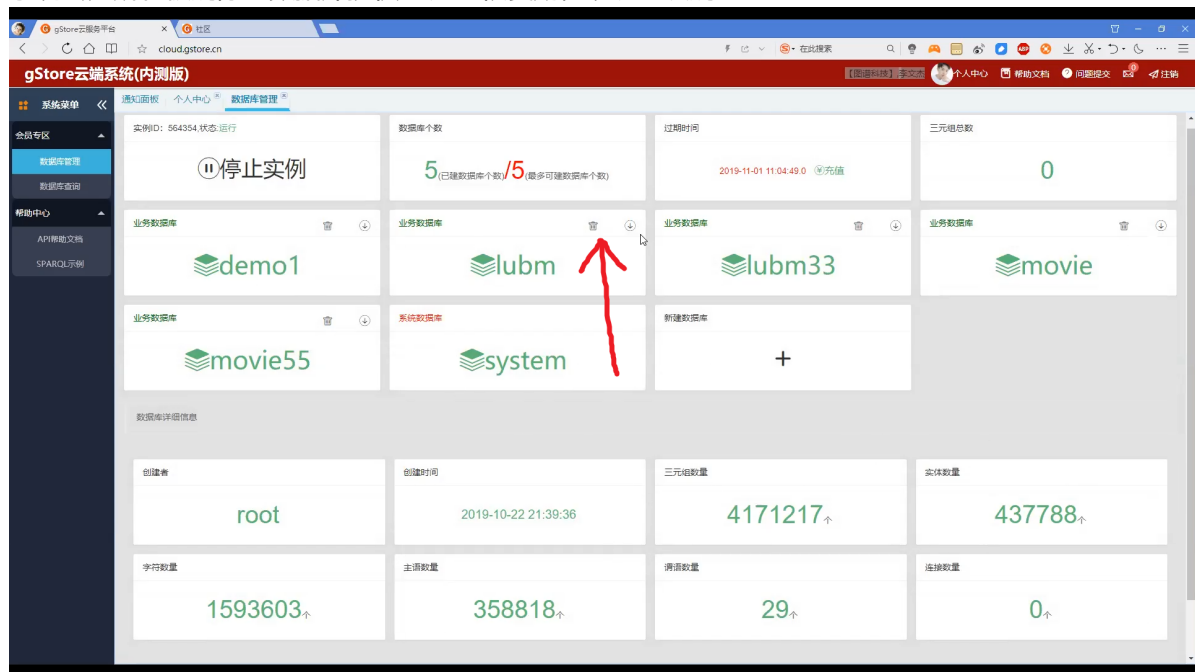
注意，需要输入两个名称，第一个名称是你创建的数据库的名称，第二个名称是你连接的数据库的名称。

第三种创建数据库的方法是使用示例数据库。现在云平台的示例数据库中只有电影数据库，但之后会不断增加。以电影数据库为例，含有400多万三元组，里面包含电影、导演、演员、上映时间、电影评分等相关信息。（示例数据库三元组数量不受单个数据库100万条三元组数量限制）

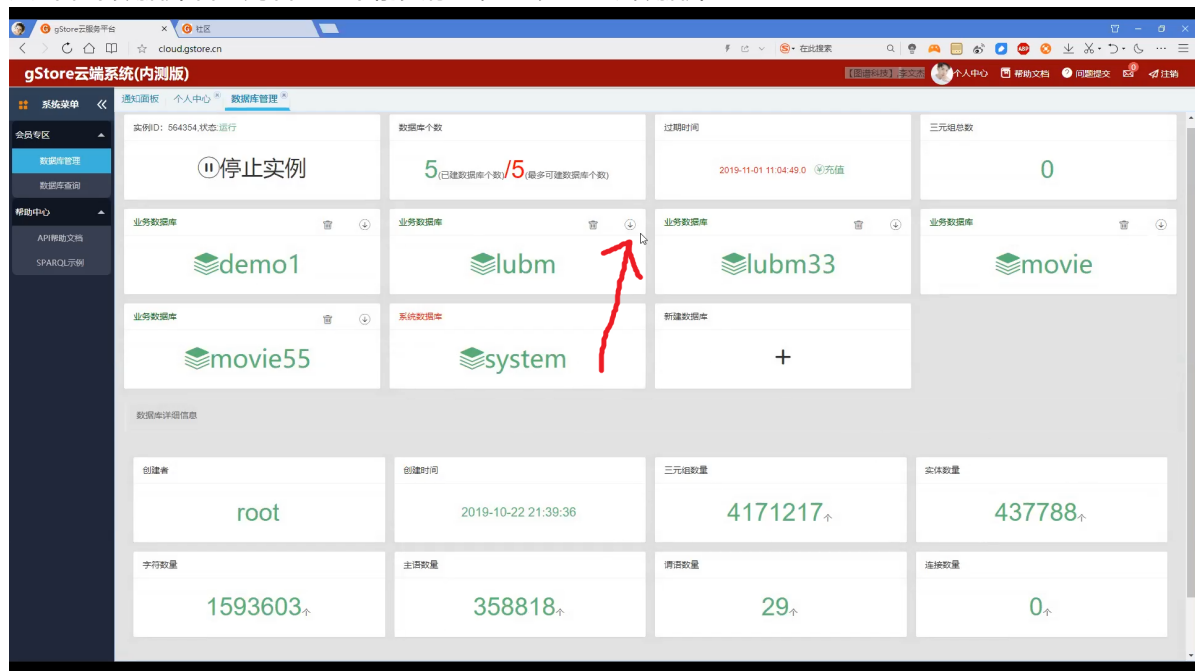


8.2.4.4 删除数据库

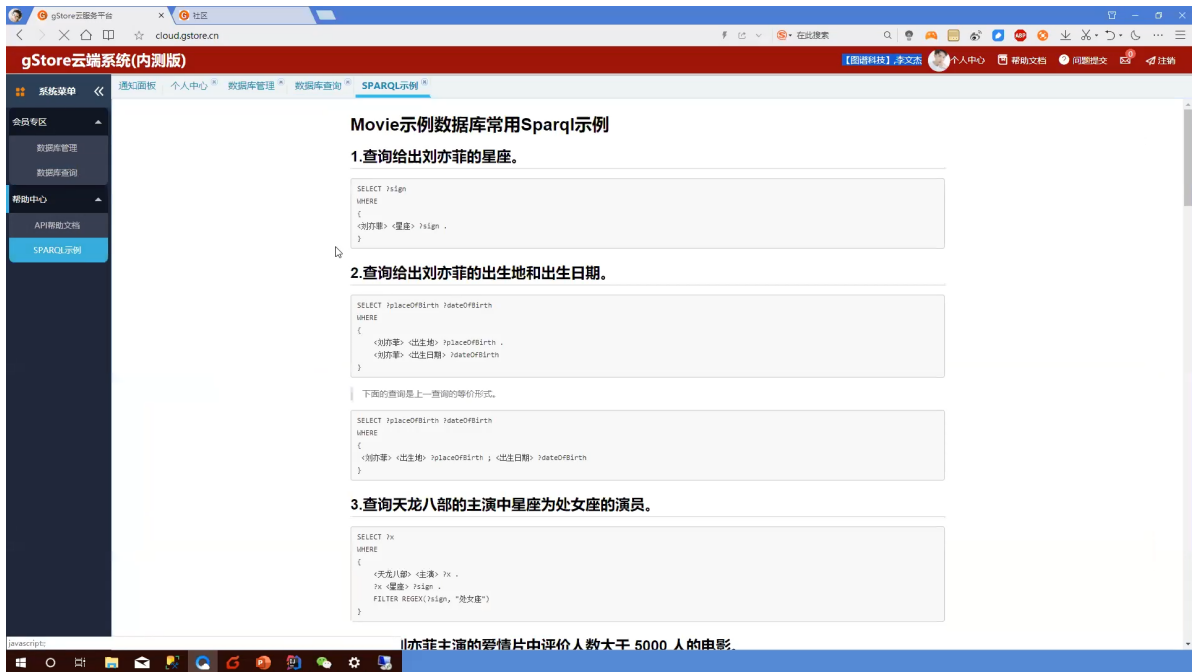
点击某个数据库右上角左边的垃圾桶标志，可以删除该数据库。
系统会为所有被删除的数据库提供15天的恢复期，以防止误删。



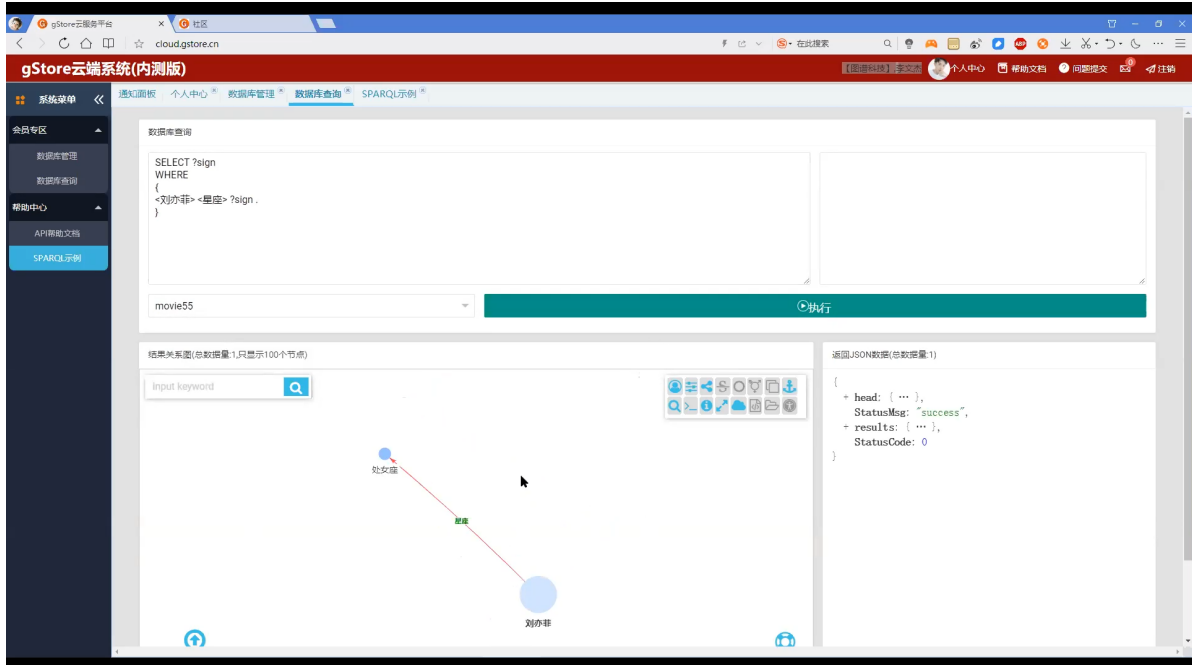
点击某个数据库右上角右边的下箭头标志，可以导出该数据库。



点击导出后，会创建一个zip压缩文件，下载后解压便得到对应该数据库的nt文件。



以示例中第一个问题为例，查询刘亦菲的星座，得到结果如下：



可以看到，左边是可视化的图形结果，右边是JSON数据的文字结果。同时，也可以用Sparql语句插入或删除数据库中的数据。

8.2.6 帮助中心

为了更好的为用户提供服务，平台为用户提供了多种帮助文档信息，并在后续将不断完善和丰富文档信息。目前主要提供了如下文档信息：

8.2.6.1 平台使用手册

平台使用手册主要是对gStore云平台的使用进行说明，让用户了解平台使用的相关问题

8.2.6.2 API帮助文档

用户除了直接使用gStore云平台对图数据进行管理和查询以外，还可以使用API方式直接访问数据，API帮助文档中详细介绍了接口参数和返回值信息。

8.2.6.3 SPARQL示例

部分用户可能对SPARQL不是很熟悉，为此平台提供了SPARQL示例文档，该文档以示例数据库Movie为例，详细介绍了目前平台支持的主要SPARQL语句，用户可以直接在该帮助文档中复杂SPARQL语句并在数据库查询功能中进行测试。

8.2.7 API

应部分用户要求，我们将数据库查询操作封装成API接口，用户可以通过该接口实现远程数据库访问，方便用户嵌入其他系统，对接中需要使用KeyID和Secret。现在平台拥有三个数据接口，分别为获取当前数据列表、获取数据库详细信息、查询数据库。具体操作可参见帮助中心中的API帮助文档模块。

8.3 结束

gStore云平台的帮助手册就此结束，如果您有什么使用上的问题，可以点击云平台右上角的“问题提交”，在社区中提出意见。

9. gStore大事记

2021年

- 11月, gStore0.9.1版本发布
- 10月, gBuilder 2.0版本发布
- 2月, gStore产品完成了中国信息通信研究院的《图数据库基础能力测试》项目;
- 2月, gStore新版官网上线;

2020年

- 12月, gStore新增最短/最长路径, K跳可达性查询, 环路检测等高级查询函数, 进一步丰富gStore算法库;
- 12月, gStore beta版 (v0.9) 和gStore 稳定版(v0.8) 在github和gitee上正式发布;
- 11月, 知识图谱自动化构建平台gBuilder V0.1版本上线;
- 10月, gStore 分布式版本gMaster在中科院计算所相关项目中进行应用示范;
- 7月, gStore与统信UOS操作系统、鲲鹏/海光/兆芯/飞腾国产CPU适配成功;

2019年

- 12月,北京大学图数据库系统gStore上线中国科技云2.0;
- 11月,中国软件测评中心对gStore分布式系统进行性能测试, 测试结果表明gStore分布式系统在106亿规模数据存储条件下平均查询响应时间为1.79秒;
- 10月, 北京大学图数据库系统gStore云平台部署上线;
- 9月, 图数据库系统与国产"PK"体系(飞腾CPU+麒麟操作系统) 适配成功;

2018年

- PKUMOD团队发表论文 Multi-Query Optimization in Federated RDF Systems, 获得23rd International Conference on Database Systems for Advanced Applications (DASFAA)最佳论文奖 (BEST PAPER AWARD)
- gStore系统的相关理论研究工作“大规模图结构数据管理”, 获得中国教育部自然科学二等奖(邹磊排名第一)
- PKUMOD研究团队基于知识图谱的自然语言问答研究工作gAnswer系统在Github上正式开源, 版本号V0.1
- gAnswer系统参加欧盟举办的知识库自然语言问答比赛QALD-9, 斩获第一名

2017年

- PKUMOD研究团队在Github上发布gStore里程碑版本 V0.5版

2016年

- PKUMOD研究团队获得中国科技部重点研发课题“图数据管理关键技术及系统”资助

2015年

- gStore相关代码在Github上正式开源，版本0.1

2014年

- PKUMOD图数据管理相关理论研究工作“海量图结构数据存储和查询优化理论研究”，获得中国计算机学会自然科学二等奖（邹磊排名第一）
- 基于知识图谱的自然语言问答研究工作gAnswer第一篇相关学术论文发表
Lei Zou, Ruizhe Huang, Haixun Wang, Jeffery Xu Yu, Wenqiang He, Dongyan Zhao, Natural Language Question Answering over RDF ---- A Graph Data Driven Approach, SIGMOD 2014
- PKUMOD研究团队获得中国自然科学基金委面上项目“基于图的大规模异质信息网络的匹配查询关键技术研究”资助

2011年

- gStore第一篇相关学术论文发表
Lei Zou., et al: gStore: Answering SPARQL Queries via Subgraph Matching. PVLDB 4(8): 482-493 (2011)
- PKUMOD研究团队获得中国自然科学基金委青年基金项目“基于图数据库理论的海量RDF 数据存储和查询方法研究”资助

10. 开源与法律条款

10.1 开源与社区

gStore系统从2015年1月开始在Github上开源，遵从BSD 3-Clause开源协议，开源地址是<https://github.com/pkumod/gStore>；我们倡导用户在尊重代码作者的著作权前提下，自由地使用和修改gStore，开发各种基于gStore的知识图谱行业应用，推动知识图谱行业软件的健康和可持续发展。我们鼓励用户积极地使用gStore系统、报告问题、提出建议，并且向gStore开源项目贡献代码，加入我们，使gStore系统变得更好。

在使用过程中遇到任何问题，如果你愿意告诉我们你的姓名、机构、使用gStore目的和邮箱，通过发邮件至service@gstore.cn，我们将及时回复您。我们保证不会泄露您和您单位的隐私，只用于提高gStore系统本身。

10.2 法律条款

gStore系统一直采用开源社区中广泛使用的BSD 3-Clause开源协议；根据该协议，用户在遵从下面的条款情况下，使用者可以自由地修改和重新发布代码，也允许使用者在gStore代码基础上自由地开发商业软件，以及发布和销售。具体条款如下：

Copyright (c) 2016 gStore team All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Peking University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

中文条款声明如下（具有同等法律效应）：

版权所有(c) 2016 gStore团队保留所有权利。

在遵守以下条件的前提下，可以以源代码及二进制形式再发布或使用软件，包括进行修改或不进行修改：

- 源代码的再发布必须保持上述版权通知，本条件列表和以下声明。

- 以二进制形式再发布软件时必须在文档和/或发布提供的其他材料中复制上述版权通知，本条件列表和以下声明。
- 未经事先书面批准的情况下，不得利用北京大学或贡献者的名字用于支持或推广该软件的衍生产品。

本软件为版权所有人和贡献者“按现状”为根据提供，不提供任何明确或暗示的保证，包括但不限于本软件针对特定用途的可售性及适用性的暗示保证。在任何情况下，版权所有人或其贡献者均不对因使用本软件而以任何方式产生的任何直接、间接、偶然、特殊、典型或因此而生的损失（包括但不限于采购替换产品或服务；使用价值、数据或利润的损失；或业务中断）而根据任何责任理论，包括合同、严格责任或侵权行为（包括疏忽或其他）承担任何责任，即使在已经提醒可能发生此类损失的情况下。

我们严格要求使用者，在其所发布的基于gStore代码基础上开发的软件和基于gStore的应用软件产品上标有“powered by gStore”和gStore标识（标识参考开发文档中的“gStore标识”）。

11. gStore标识

11.1 gStore的图片标识如下



11.2 Powered by gStore 推荐标识如下

