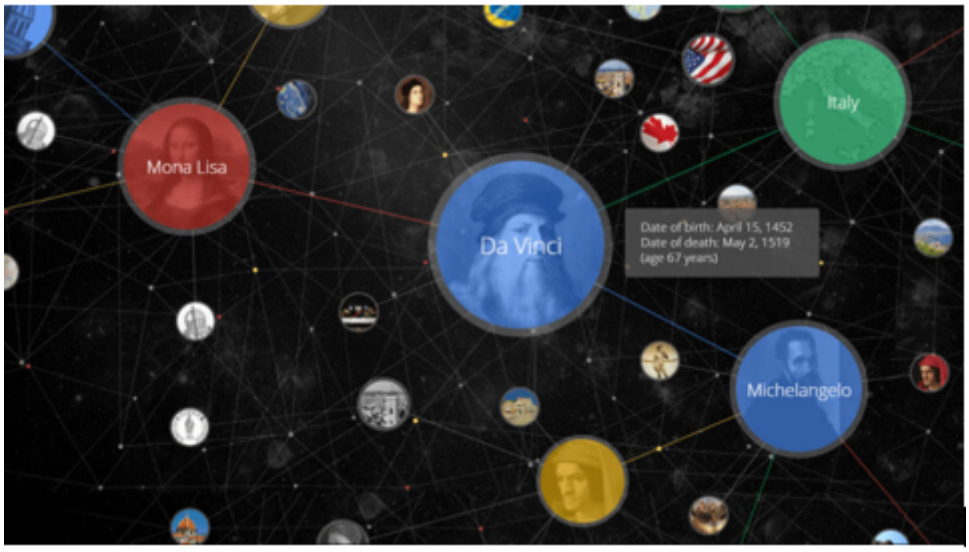
gStore0.9.0版本使用手册

1. 知识图谱与gStore介绍

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,BornOndate,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	“ReeseWitherspoon”
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。

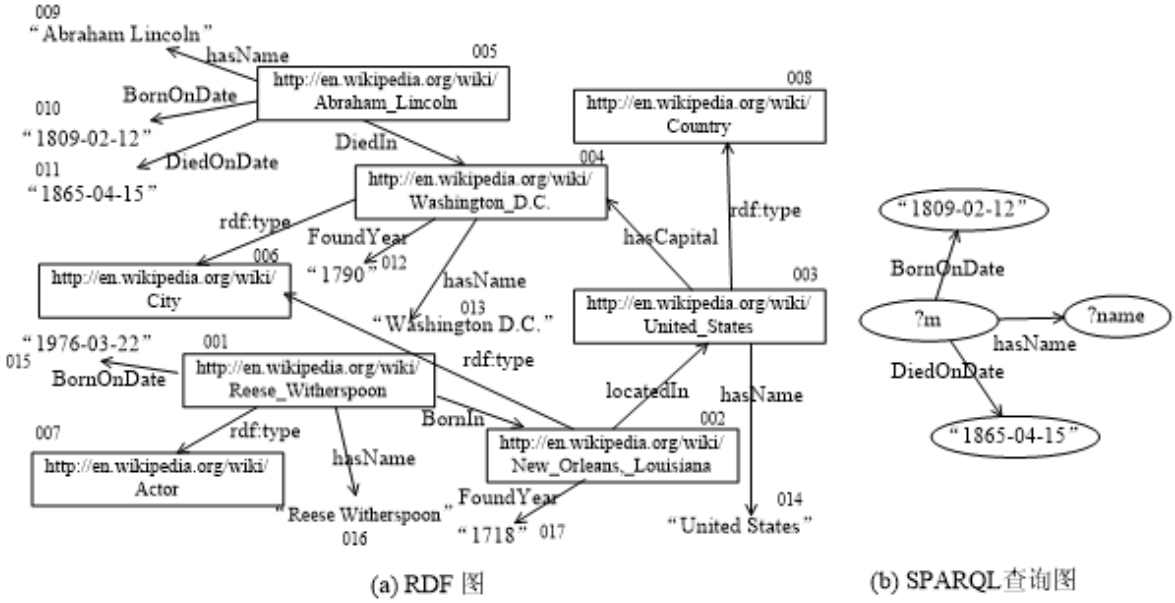


图 1-3 RDF图和SPARQL查询图

我们所研发的gStore系统属于后者，我们将RDF和SPARQL分别表示成图的形式，利用子图匹配的方法来回答SPARQL查询。例如在RDF中，主体和客体可以分别表示成RDF图中的节点，一条称述（即RDF三元组）可以表示成一条边，其中谓词是边的标签。SPARQL语句同样可以表示成一个查询图。 图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【支持SPARQL查询语法】）；单机可以支持30亿以上的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14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。

2. 安装指南

2.1 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
pthreads	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

2.2 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。

2.2.1 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
  ```

2.2.2 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
```

2.2.3 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
```

2.2.4 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::...`"），很可能是因为您使用低于 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
```

2.2.5 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 -xzvf  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
```

2.2.6 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
```

2.2.7 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
```

2.2.8 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
```

2.2.9 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
zip       #ubuntu系统
```

2.3 gStore获取

如果遇到权限问题，请在命令前加 `sudo` 。

2.3.1 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

2.3.2 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
```

2.4 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. 快速入门

3.1 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）1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。

3.2 构建数据库

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。

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。

3.2.1 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_name rdf_triple_file_path
```

参数含义：

```
db_name：以“.db”结尾的数据库名称
rdf_triple_file_path：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 lubm ./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

3.2.2 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】-【workbench】-【数据库管理】功能。

3.2.3 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】

3.3 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，用户可以通过如下三种方式使用gStore数据库查询功能。

3.3.1 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_name query_file
```

参数含义：

```
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 lubm ./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_name
```

程序显示命令提示符（“gsql>”），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 lubm

gsql>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>

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

gsql>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

3.3.2 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】-【workbench】-【数据库查询】功能。

3.3.3 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】

3.4 外部访问接口（ghttp）

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

3.4.1 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_name 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）
```

`ghttp` 支持GET和POST请求类型。

`ghttp` 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”）

** 注意：你最好不要只需输入命令“Ctrl + C”来停止ghttp，因为这不安全。

** 为了停止ghttp服务器，您可以输入 `bin/shutdown serverPort`

3.4.2 ghttp相关API

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

3.5 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。

3.5.1 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_name rdf_triple_file_path
```

参数含义：

```
rdf_triple_file_path：带".nt"或者".n3"后缀的文件路径
```

示例：

```
[bookug@localhost gStore]$ bin/gadd lubm ./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

3.5.2 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。

3.5.3 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并通过数据库管理模块可以创建图数据库，具体内容详见【开发文档】-【workbench】-【数据库查询】功能。

3.5.4 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`build`请求来构建图数据库，具体内容详见【开发文档】-【常用API】

3.6 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。

3.6.1 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
```

3.6.2 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。

3.6.3 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并通过数据库管理模块可以创建图数据库，具体内容详见【开发文档】-【workbench】-【数据库查询】功能。

3.6.4 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`build`请求来构建图数据库，具体内容详见【开发文档】-【常用API】

3.7 数据库状态查询

统计数据功能是获取指定数据库的统计信息，有如下几种方式。

3.7.1 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_name
```

参数含义：

```
db_name：数据库名称
```

示例：

```
[root@localhost gStore]$ bin/gmonitor 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
```

3.7.2 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创建图数据库，具体内容详见【开发文档】-【workbench】-【数据库管理】功能。

3.7.3 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`monitor`获取数据库统计信息，具体内容详见【开发文档】-【常用API】

3.8 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，有如下三种形式

3.8.1 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"
```

3.8.2 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创建图数据库，具体内容详见【开发文档】-【workbench】-【数据库管理】功能。

3.8.3 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`show`命令实现相关功能，具体内容详见【开发文档】-【常用API】

3.9 删除数据库

删除数据库功能可以删除指定数据库，有如下三种形式

3.9.1 命令行模式（gdrop）

gdrop用于删除某个数据库。

用法：

```
bin/gdrop db_name
```

命令参数：

```
db_name:数据库名称
```

示例：

```
[root@localhost gStore]$ bin/drop 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_name`。

3.9.2 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】-【workbench】-【数据库管理】功能。

3.9.3 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】

3.10 导出数据库

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：

3.10.1 命令行模式（gexport）

gexport用于导出某个数据库。

用法：

```
bin/gexport db_name [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：

```
(base) [root@iz8vb0u9hafhzz1mn5xcklz gStore]# bin/gexport 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.
```

3.10.2 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】-【workbench】-【数据库管理】功能。

3.10.3 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】

3.11 关闭数据库HTTP访问端口

关闭gStore HTTP访问端口可以停止gStore的ghttp服务

3.11.1 命令行模式（shutdown）

shutdown用于关闭某个ghttp服务

用法：

```
bin/shutdown port
```

命令参数：

```
port: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）
```

示例：

```
(base) [root@root gstore-test]# bin/shutdown 9990
[1]+  Done                    nohup bin/ghttp 9990
```

3.12 初始化系统数据库

`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

3.12.1 命令行模式（ginit）

ginit用于初始化数据库

用法：

```
bin/ginit -d [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`数据库中将没有其他数据库信息

示例：

```
[root@localhost gStore]$ bin/ginit -d 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 priviledge of update lock acquired
QueryCache cleared
Total time used: 97ms.
update num : 4
system.db is built successfully!
```

4. API使用

4.1 API介绍

gStore通过http服务向用户提供API服务，其组件为ghttp。

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。

4.1.1 启动API服务

命令：

```
bin/ghttp [db_name] [port]
nohup bin/ghttp [db_name] [port] & //后台启动
```

命令参数：

```
db_name: 启动的数据库名称（可选）（为空时默认为system数据库)
port:http服务监听端口（可选）（为空时默认为9000端口）
```

建议您仔细阅读示例代码以及相应的Makefile。这将帮助您理解API，特别是如果您想基于API接口编写自己的程序。

4.1.2 API服务地址

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

4.2 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)
- -

4.3 C++ 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/");
```

4.4 Java 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/");
```

4.5 Python 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

```
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/")
```

4.6 Nodejs 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/");
```

4.7 Php 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. gStore可视化工具Workbench

5.1 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。

5.2 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：

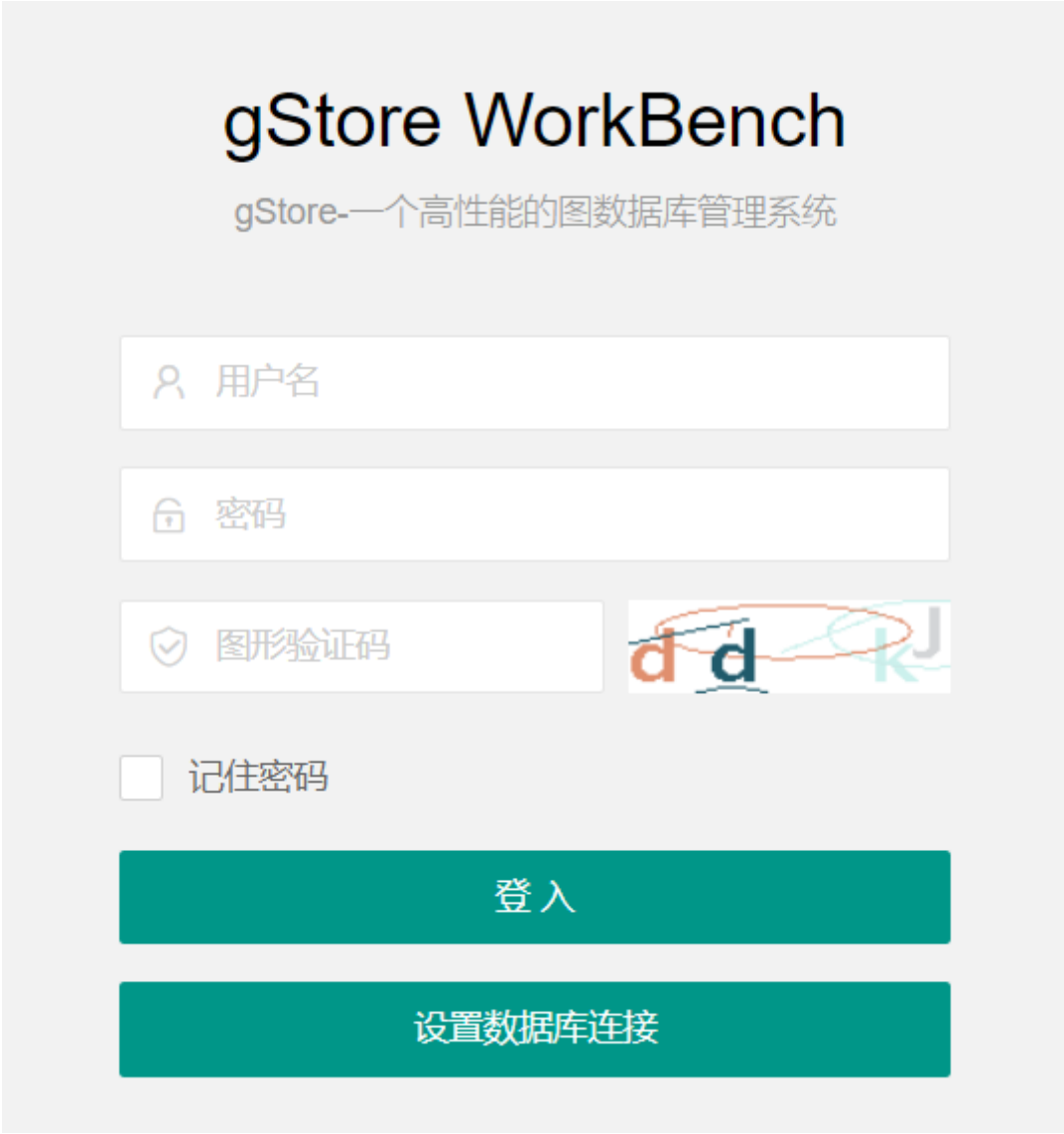
```
[root@node1 bin]# ./shutdown.sh
```

5.2 登录

5.2.1 浏览器访问系统

登录网址为：

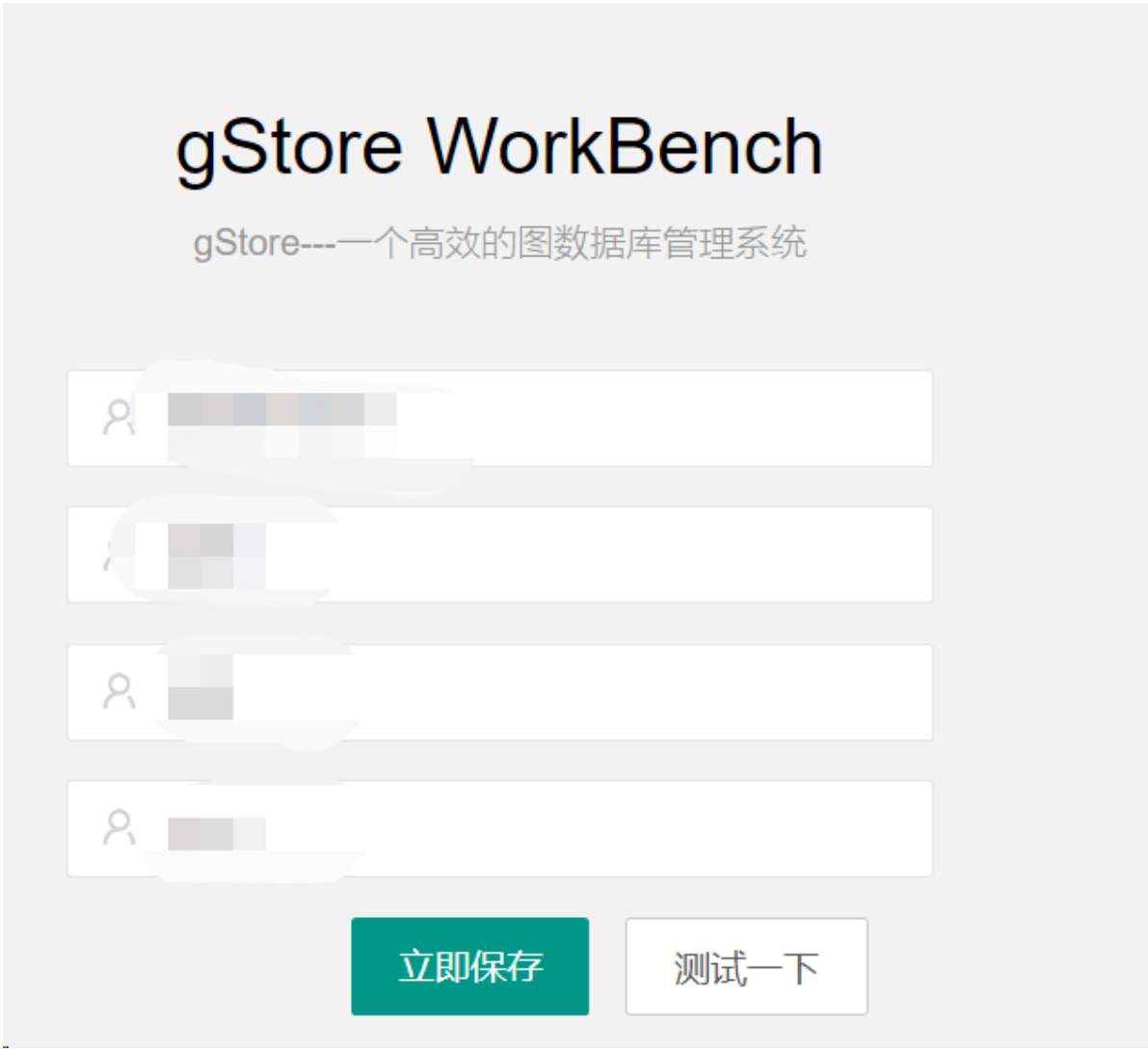
```
http://workbench自己部署的服务器ip:8080/gworkbench/views/user/login.html
```



5.2.2 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

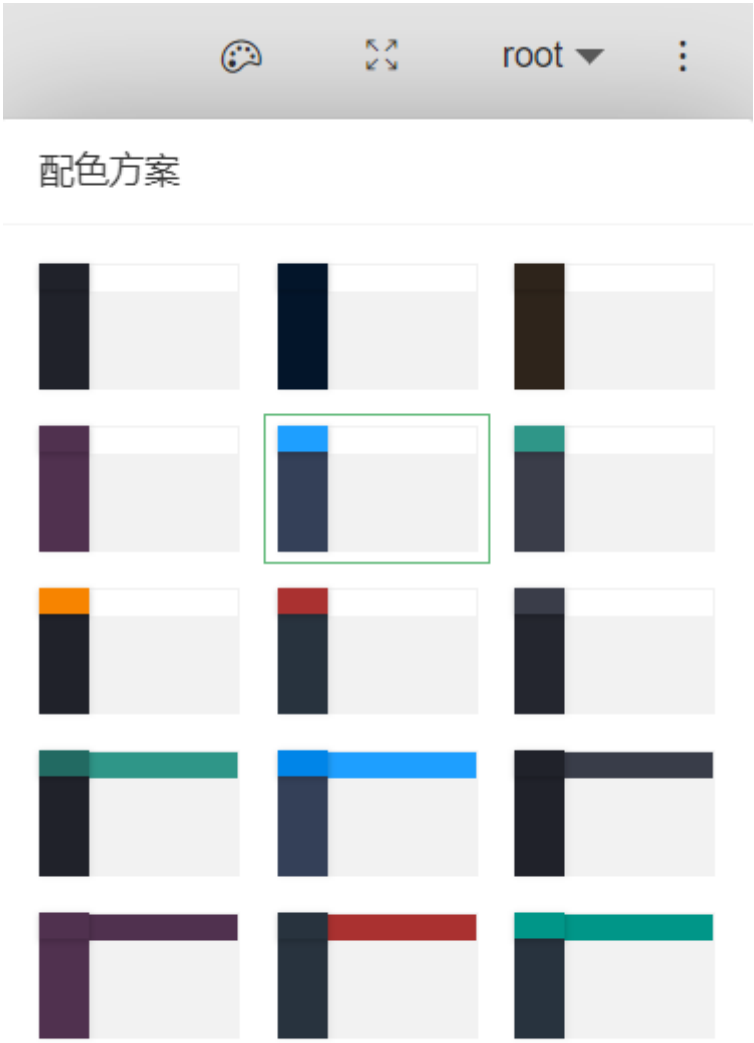
输入用户名、密码和验证码登录到已保存服务器上的gStore图数据库管理系统（gstore默认用户名为root，密码为12345）



5.3 用户管理

在网站右上角用户可以

- 调整界面配色
- 全屏操作
- 查看、更改用户基本资料和密码
- 退出gStore图数据库管理系统
- 提供数据库信息以及进入官网链接


root
配色方案


基本资料
修改密码
修改密码
当前密码
新密码
6到16个字符
确认新密码
确认修改

5.4 查询功能

5.4.1 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。

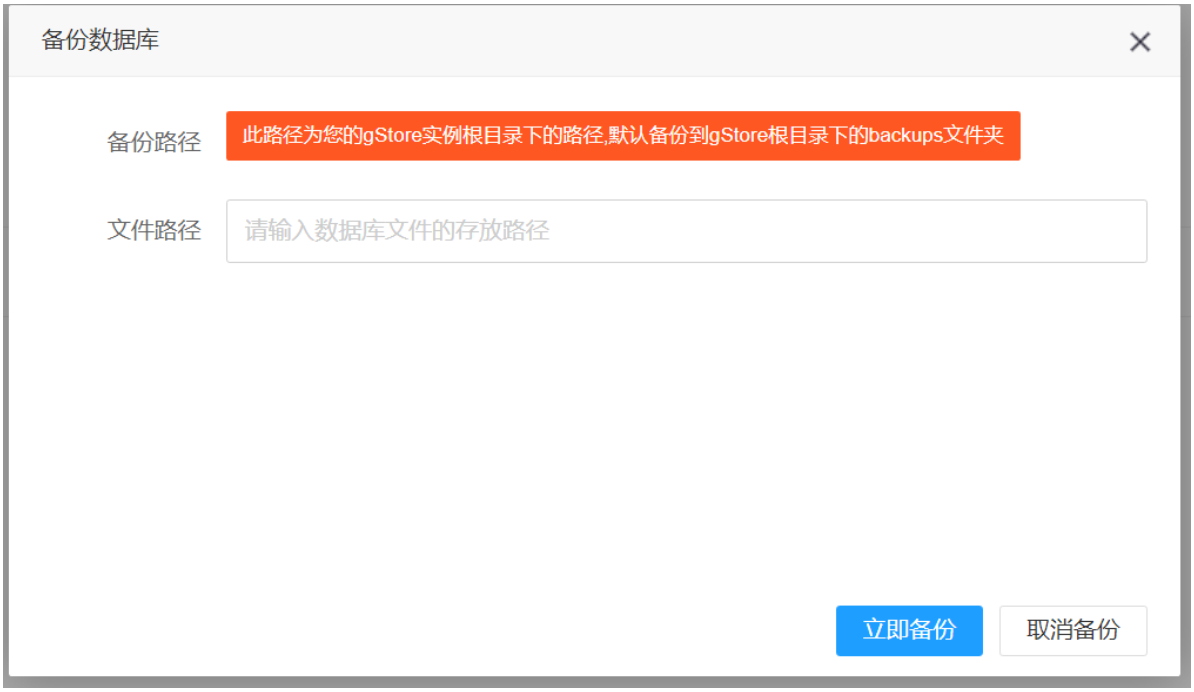
例如：路径选择

```
/root/gStore/data    绝对路径
./data               相对路径
```

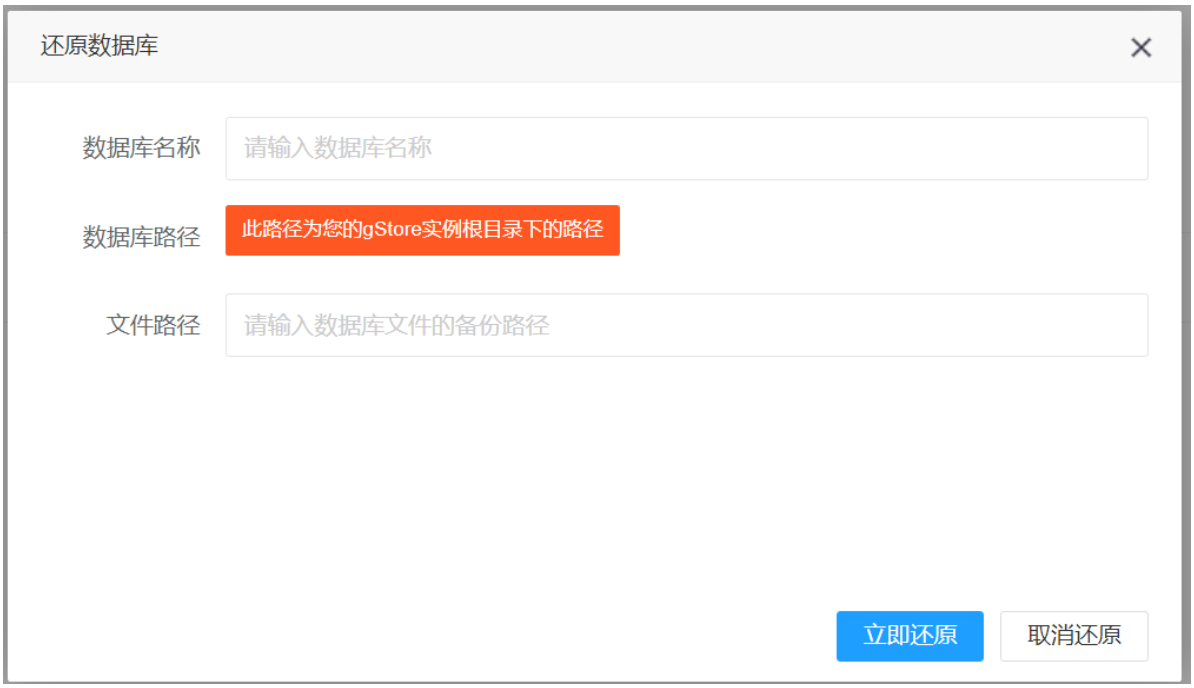
输入正确的路径后点击立即导出。**system数据库不能导出。**

- **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

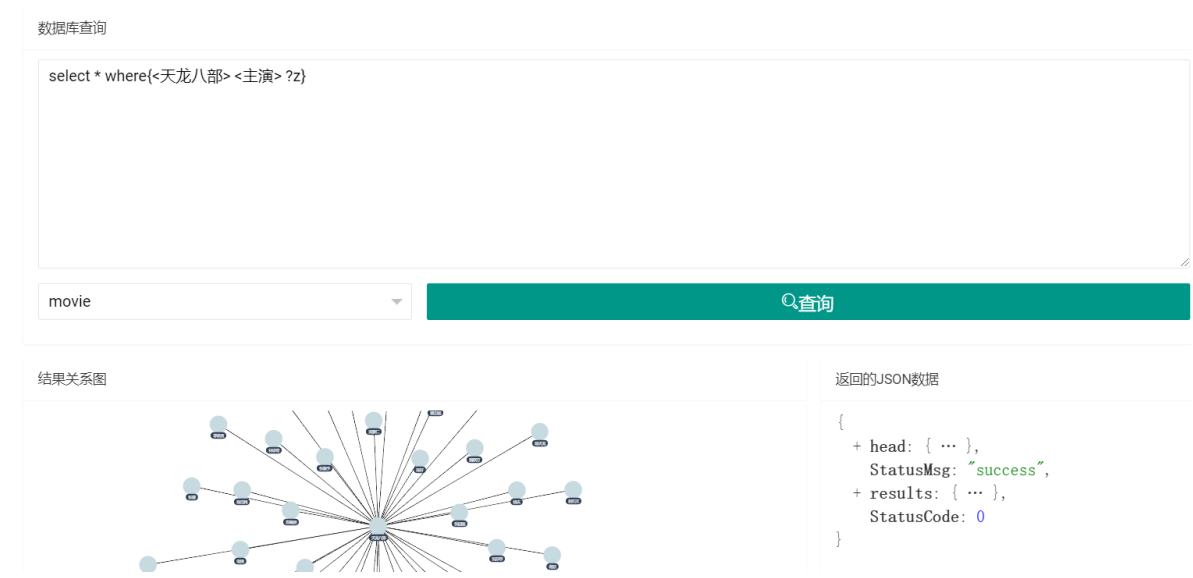


- **还原数据库**



5.4.2 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，返回结果图和json数据文件



5.6 用户管理（只有root用户有该权限）

5.6.1 用户列表

- 显示当前用户列表

查看用户的用户名、密码和具有的使用权限

当前所有用户

用户名	密码	查询权限	更新权限	加载权限	卸载权限	备份	恢复	导出	操作
root	123456	all	all	all	all	all	all	all	删除
shasha	1	lubm1 lubm2 lubm4 lubm5	lubm1	lubm1 lubm4 lubm5	lubm1 lubm4 lubm5	lubm1 lubm4 lubm5	lubm1 lubm4 lubm5	lubm1 lubm4 lubm5	删除
zlj	0								删除

5.6.2 添加用户

- 添加新用户

输入用户名和密码添加用户

5.6.3 授权用户

- 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

用户授权

请选择用户

shasha

请选择数据库

lubm2

查询权限

加载权限

卸载权限

更新权限

备份权限

还原权限

导出权限

立即授权

6. Docker安装与部署

> 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.

6.1 环境准备

关于安装使用Docker，参考地址：docker

6.2 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。

6.3 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

7. SPARQL查询语法

7.1 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。**

7.1.1 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。

7.1.2 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>

7.1.3 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>

7.2 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高豆瓣评分：

```
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>

7.3 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 ?scoreWHERE{  ?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>

7.4 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>

7.5 高级功能

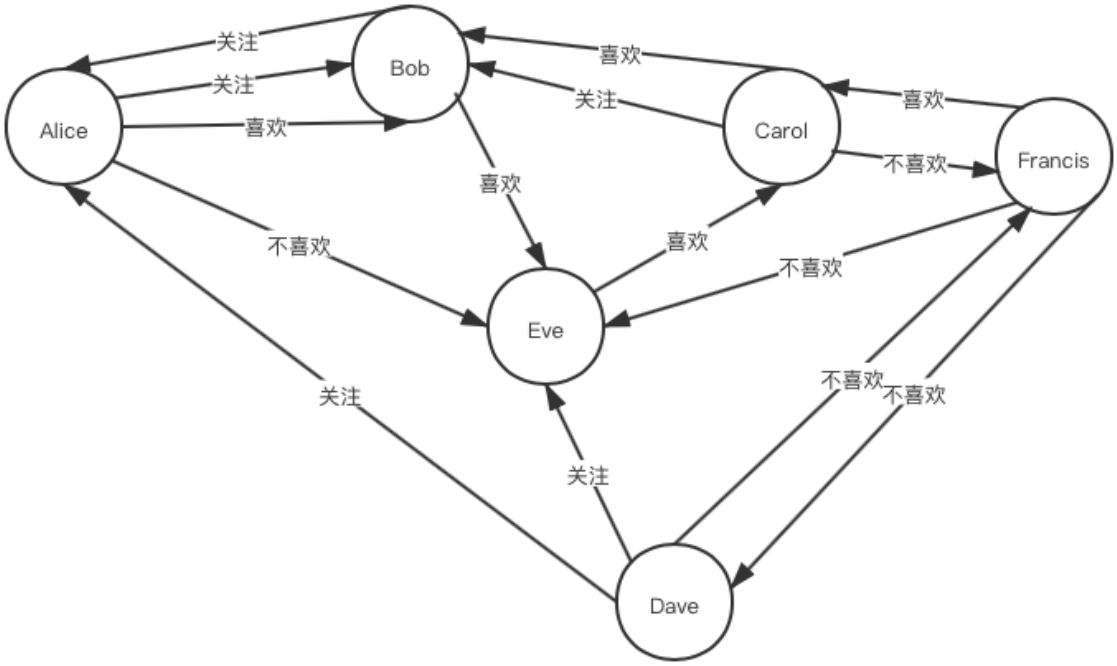
7.5.1 路径相关查询

在**内核版本 v0.9** 中，gStore 加入了与数据图中结点间的路径相关的一系列查询，目前包括环路查询和最短路径查询。

为了更好地演示路径相关查询的功能，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": [{...}, {...}, ...] }
```

7.5.1.1 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#>

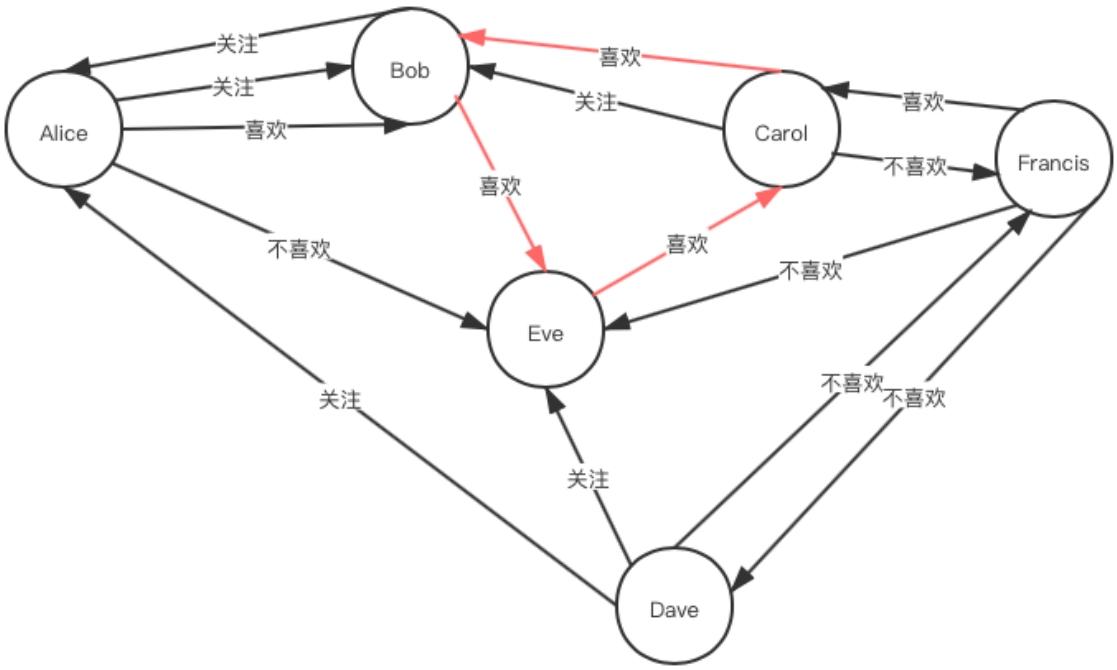
如果希望输出一个满足以上条件的环，则使用下面的查询：

```
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：



7.5.1.2 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

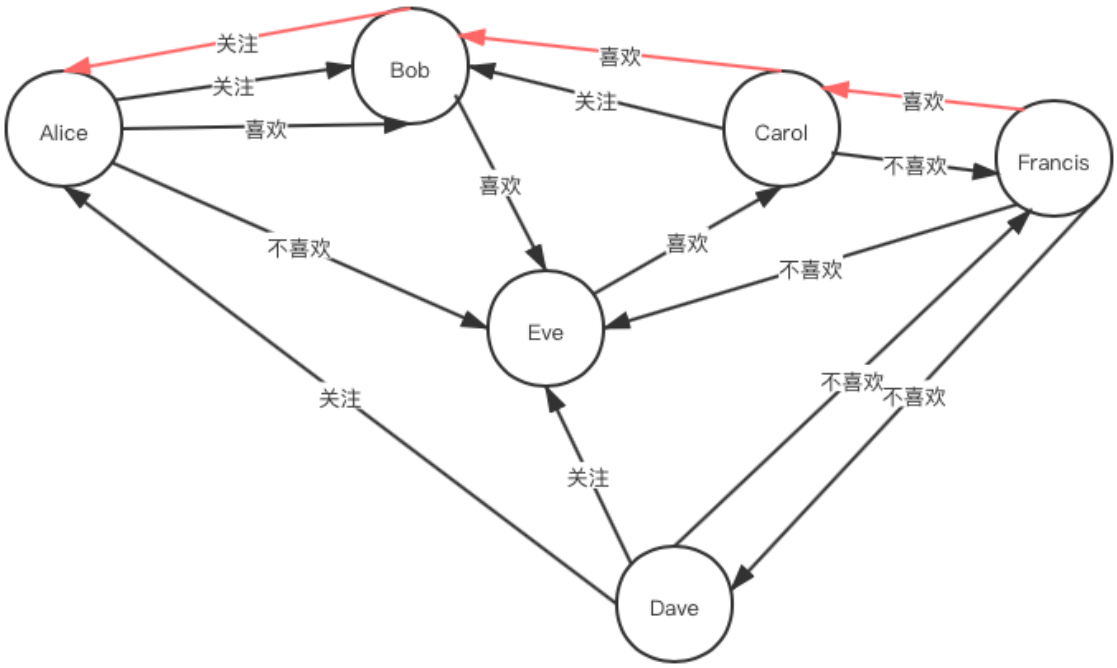
返回值

- `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}]}
```

7.5.1.3 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

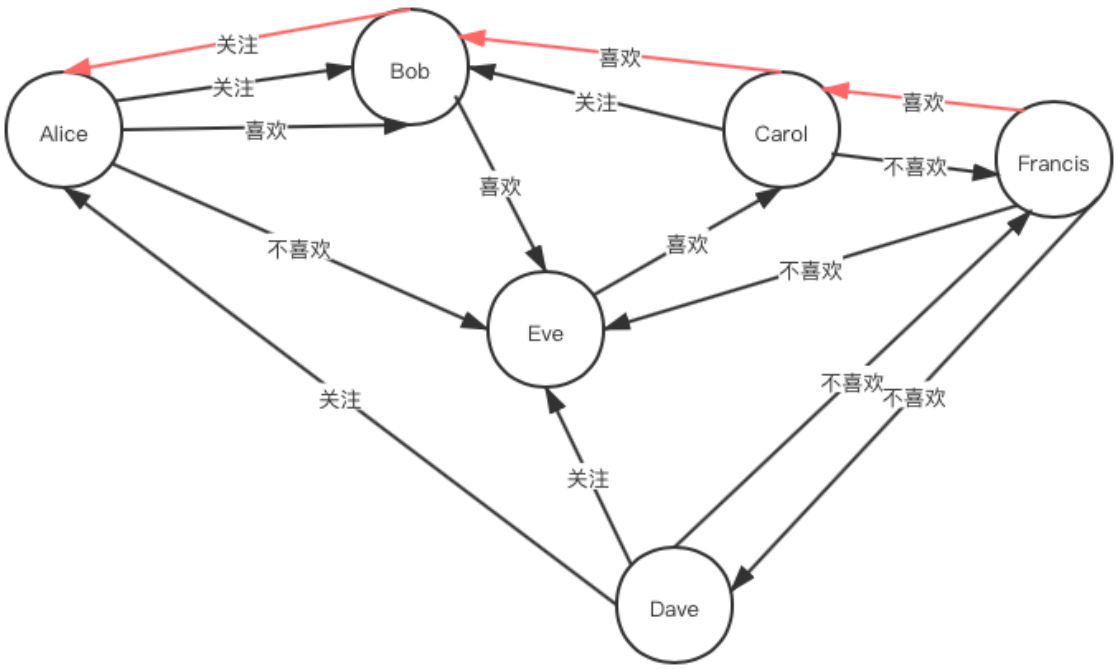
返回值

- `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"}]}
```

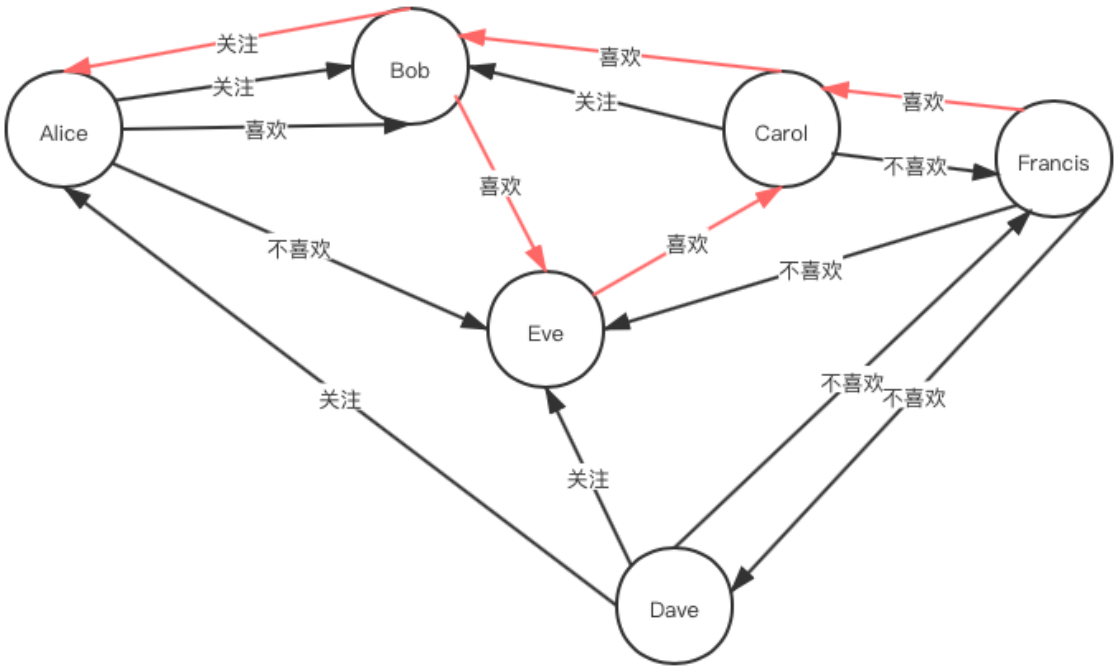
若希望返回一条两人之间满足条件的路径，则可以调用 `kHopReachablePath` 函数：

```
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：



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。

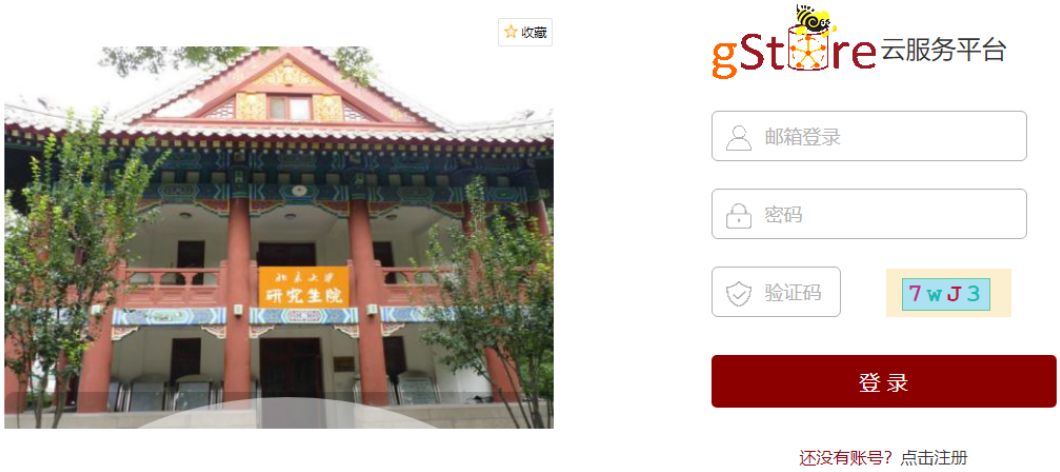
8.1.4 gStore如何在以上事务中发挥作用?

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



如为第一次使用gStore云平台，需要进行注册,注册界面如下：

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，还有包括gStore官网、团队等一些常用链接信息等。

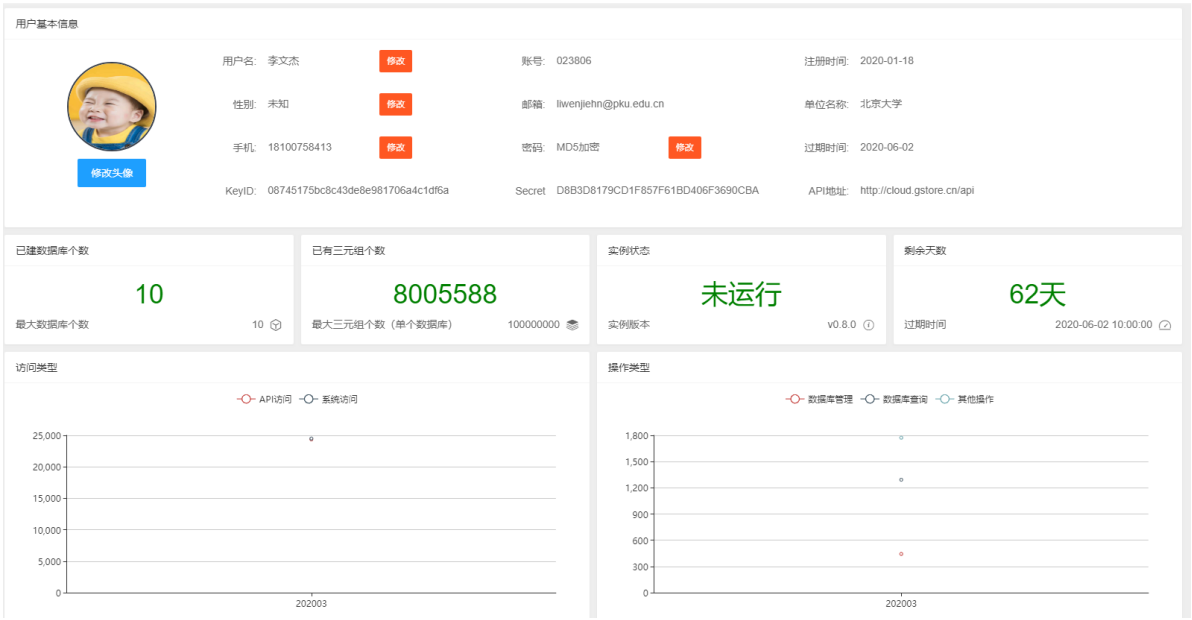
8.2.3 个人中心

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。

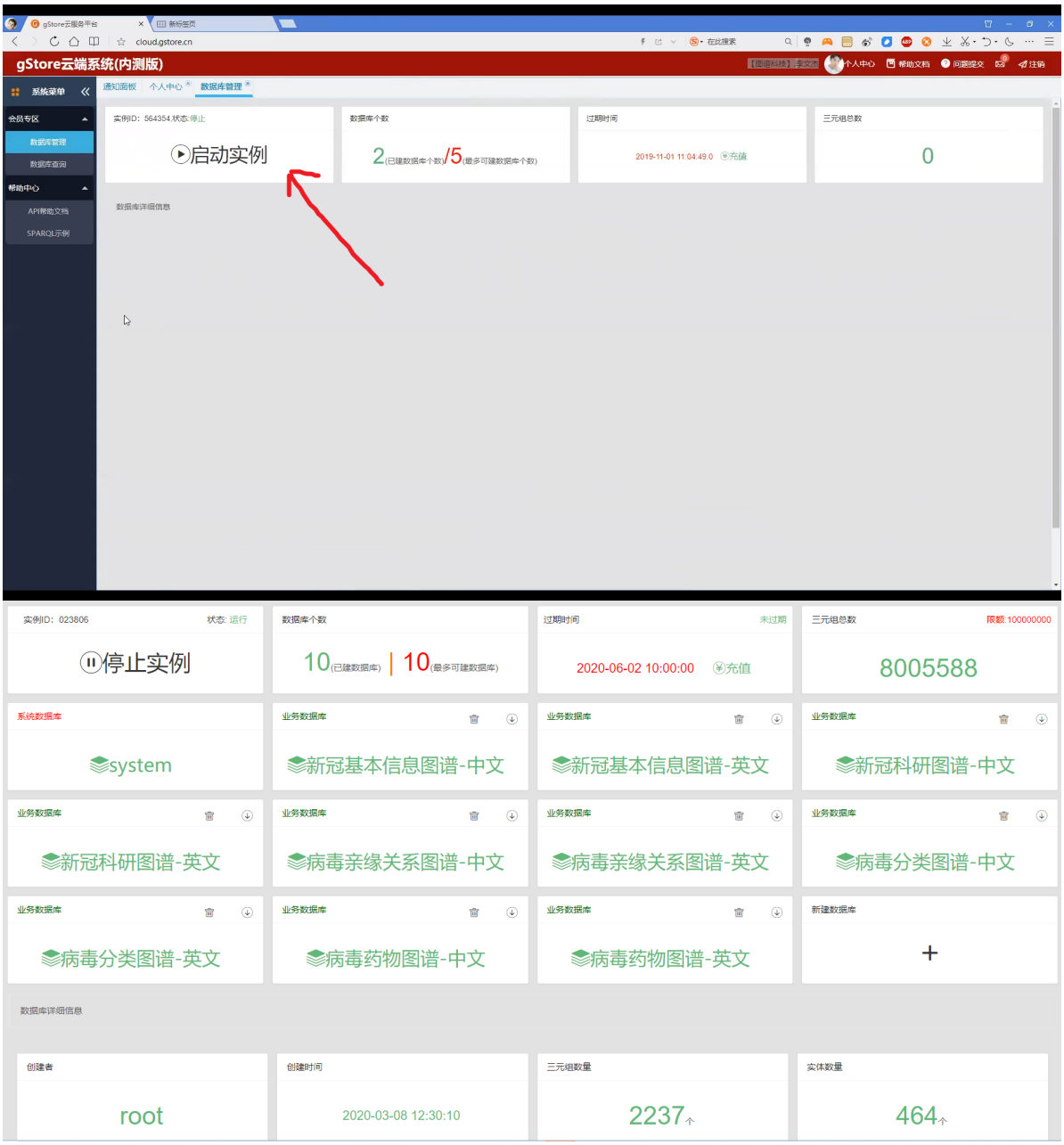


8.2.4 数据库管理

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

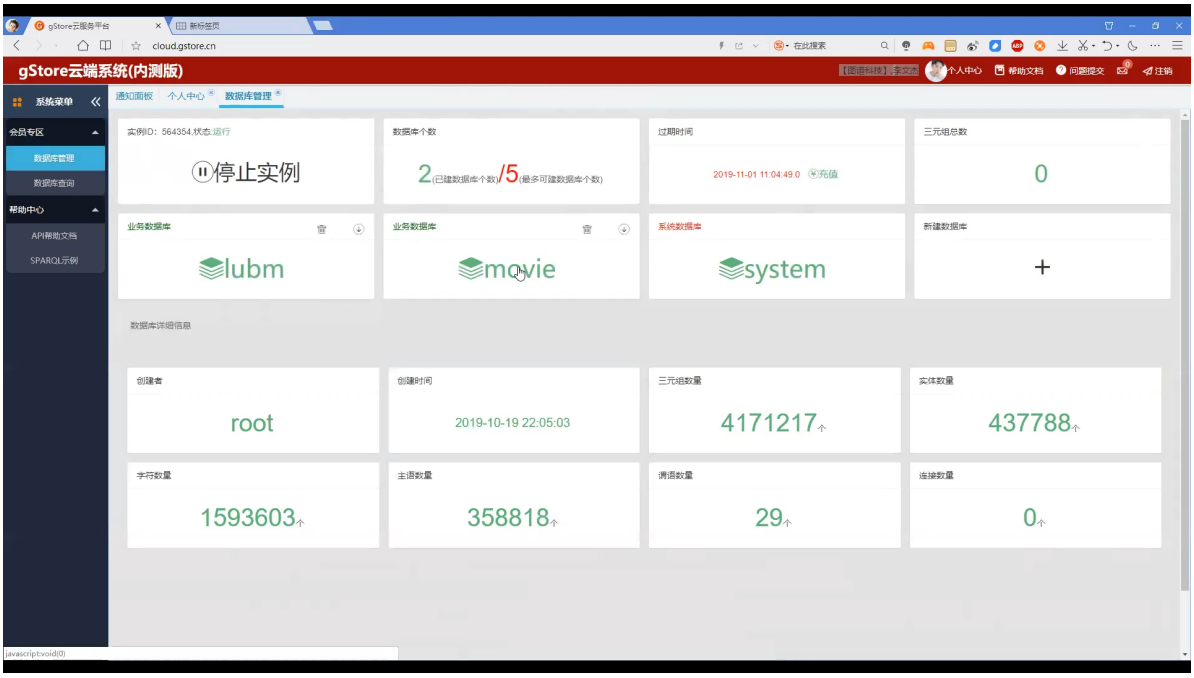
在数据库管理中有一个很重要的功能：实例。在第一次进入系统时，实例可能是停止状态，需要手动启动。



8.2.4.2 查看数据库信息

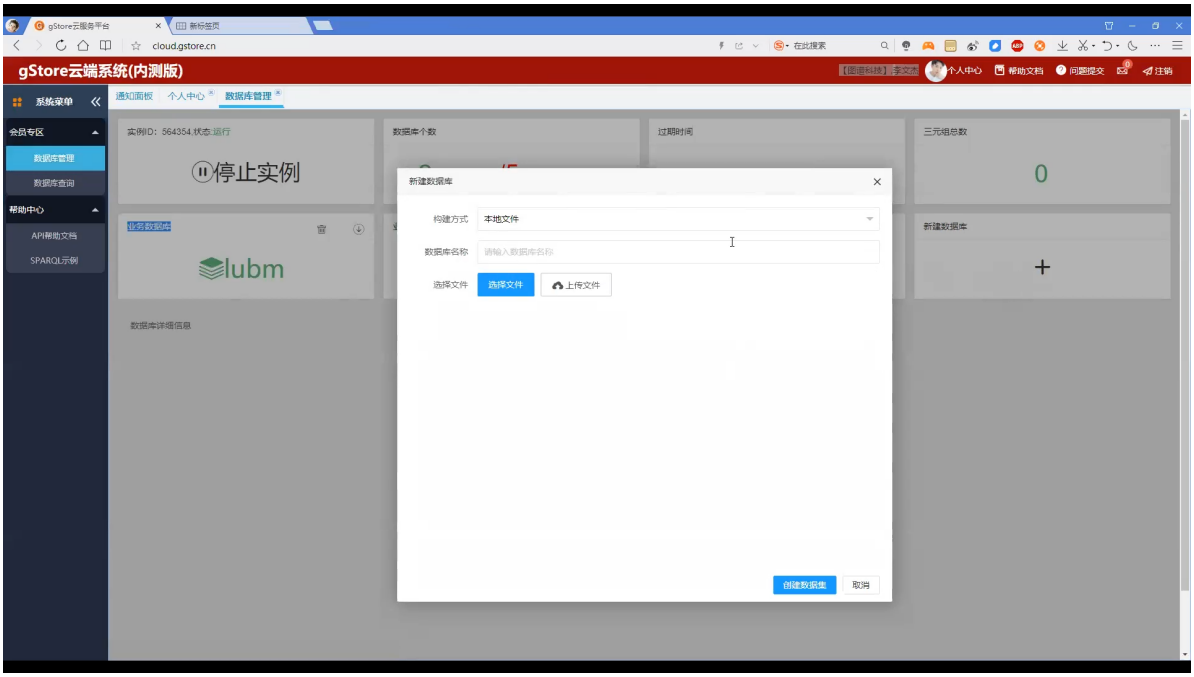
启动实例后，可以从上面一行中看到实例的状态，已创建数据库与最多可创建数据库的个数，gStore过期时间，三元组的总数等信息。
在下方，可以看到现在已创建的数据库，其中包括一个系统数据库（系统创建，不能操作，不算在最多可创建数据库个数里）和若干业务数据库（自己创建，可以操作）。
对业务数据库可以进行创建数据库，删除数据库，导出数据库，获取数据库相关信息等操作。
点击某个数据库可以获取其相关信息，包括创建者、创建时间、三元组数量、实体数量、字符数量、主

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：

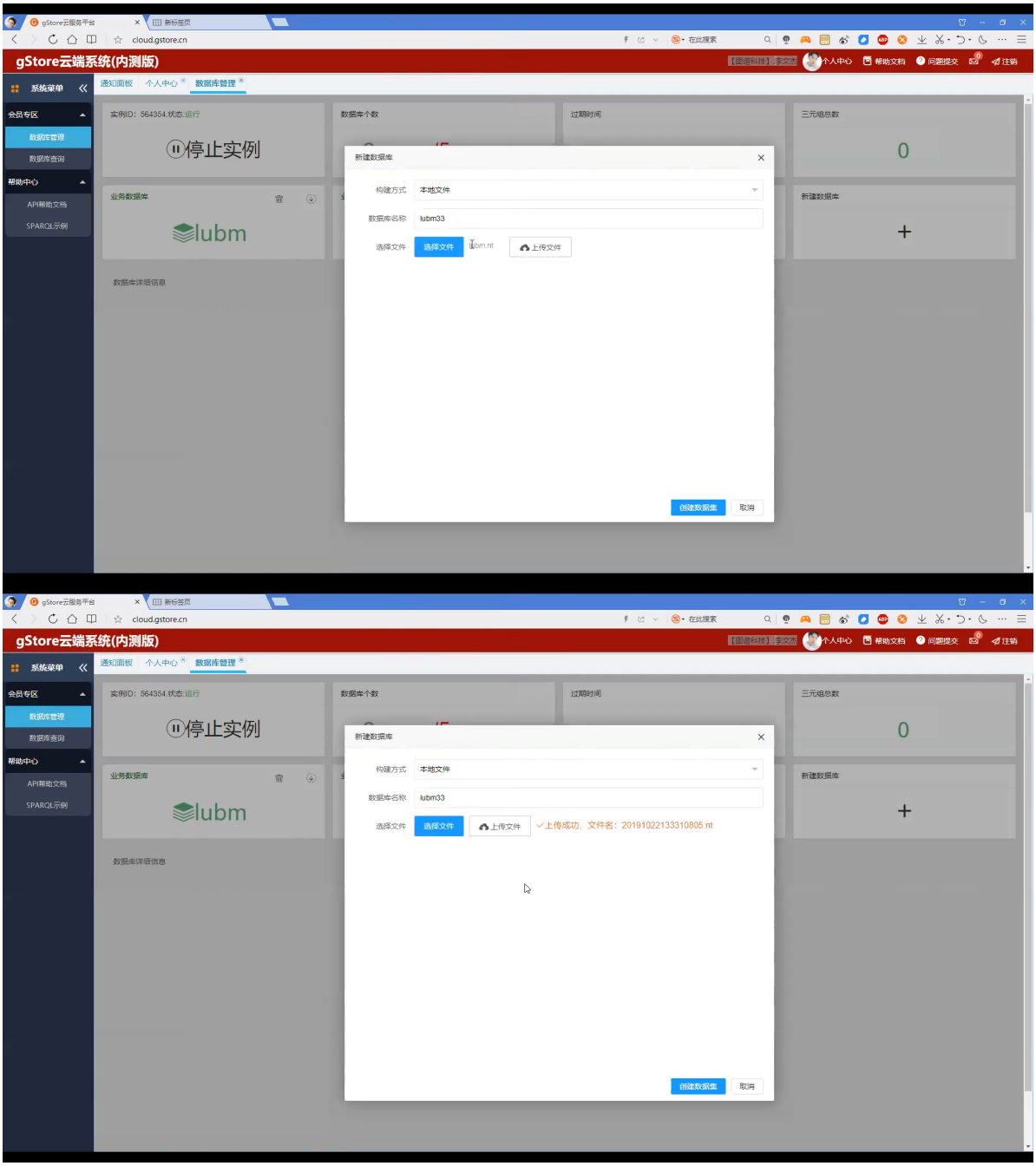


8.2.4.3 创建数据库

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**）

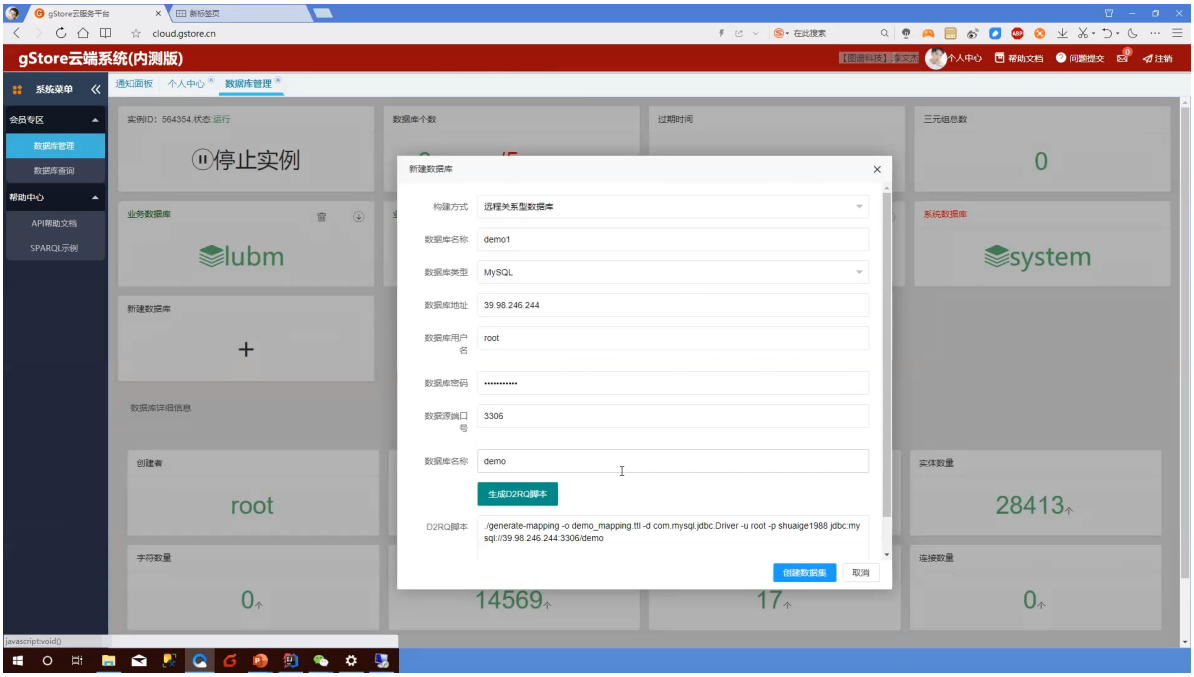


创建数据库有三种方法，第一种是本地文件，即从本地上传一个文件到服务器上。目前系统只支持nt文件，将来可能会支持n3文件。



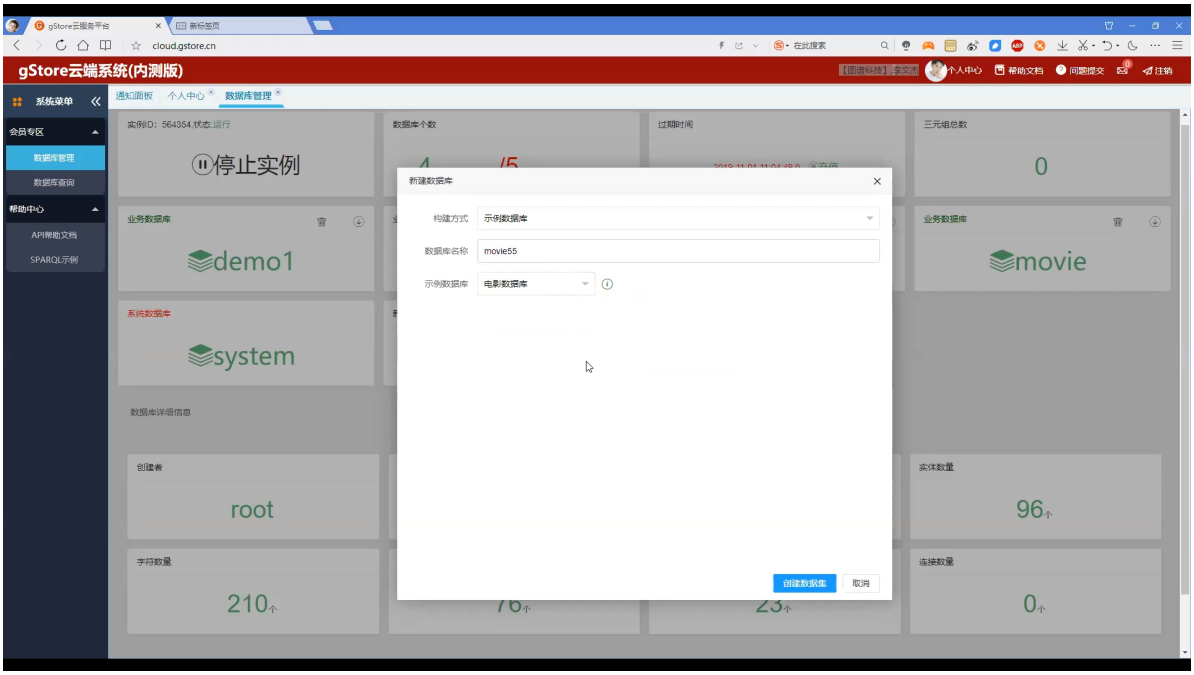
注意，文件大小不能超过2GB，行数不能超过一百万

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。



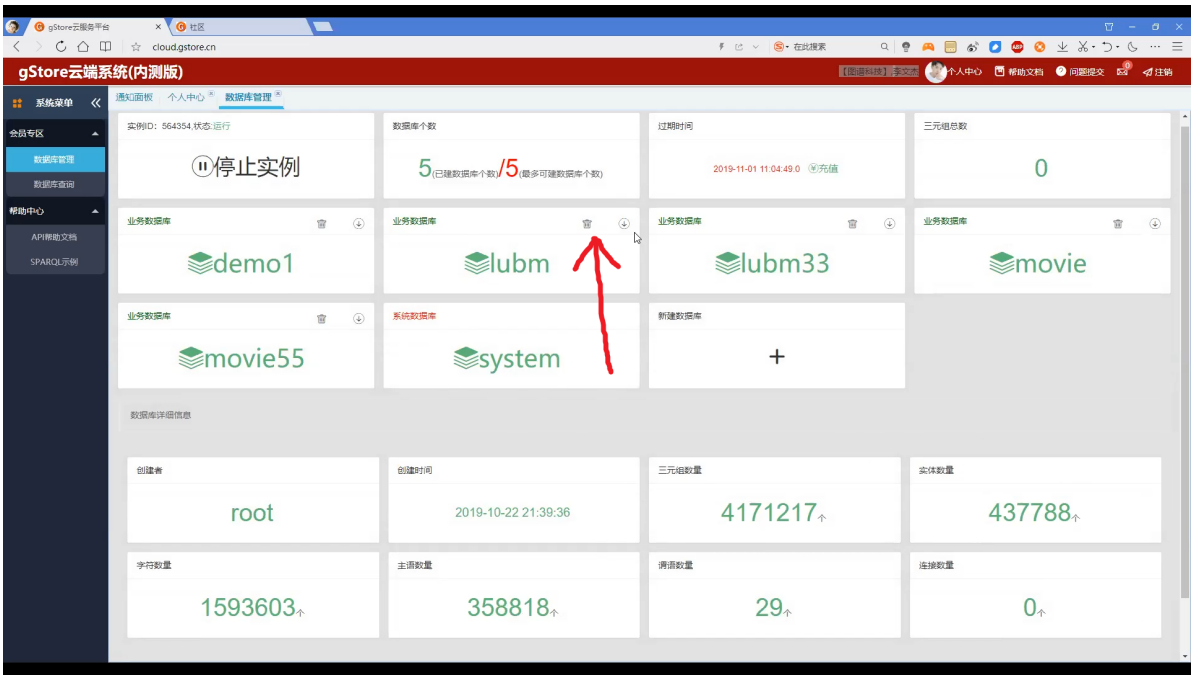
注意，需要输入两个名称，第一个名称是你创建的数据库的名称，第二个名称是你连接的数据库的名称。

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**）

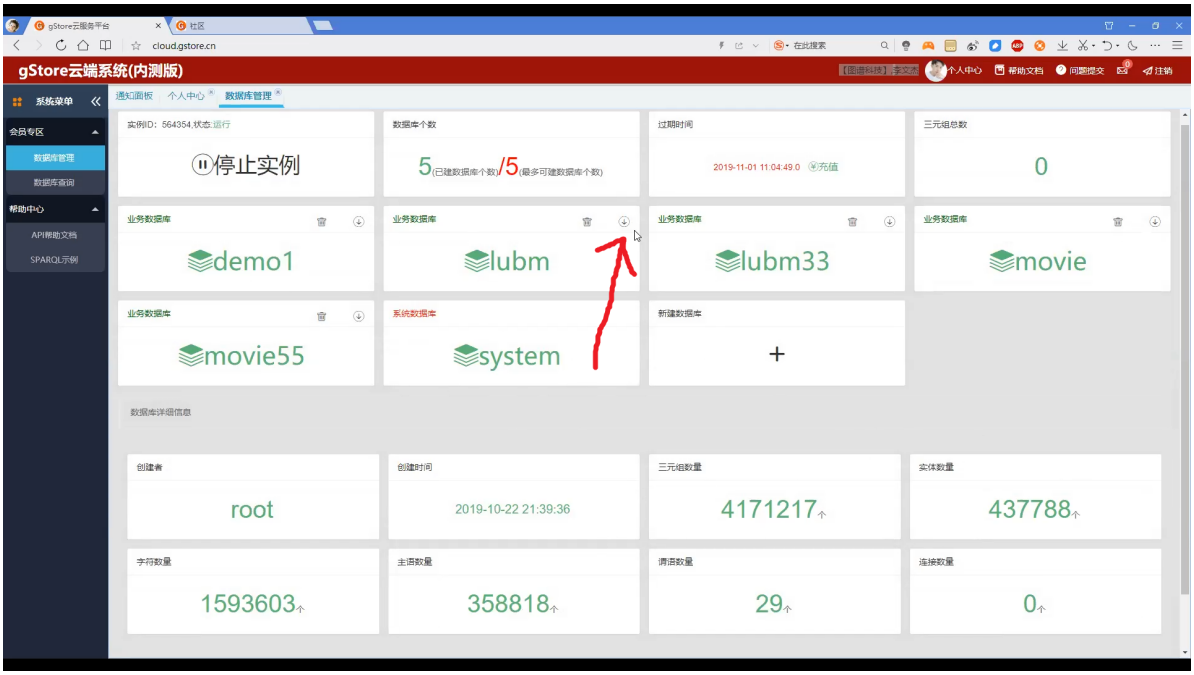


8.2.4.4 删除数据库

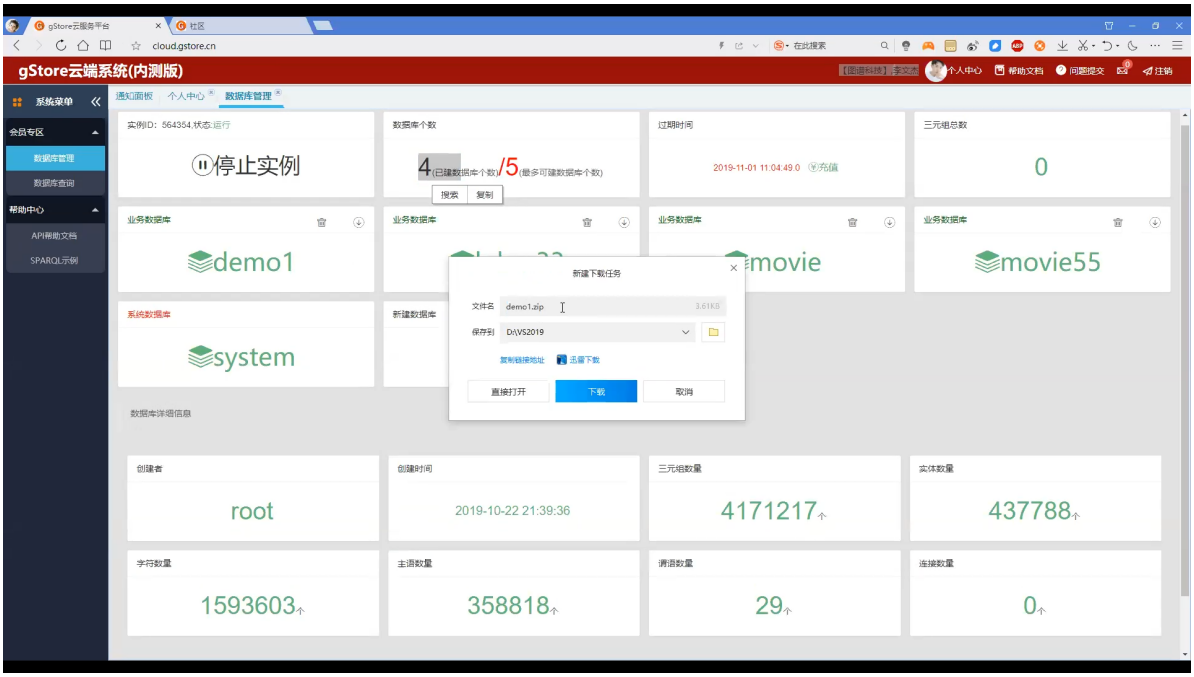
点击某个数据库右上角左边的垃圾桶标志，可以删除该数据库。
系统会为所有被删除的数据库提供15天的恢复期，以防止误删。



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。



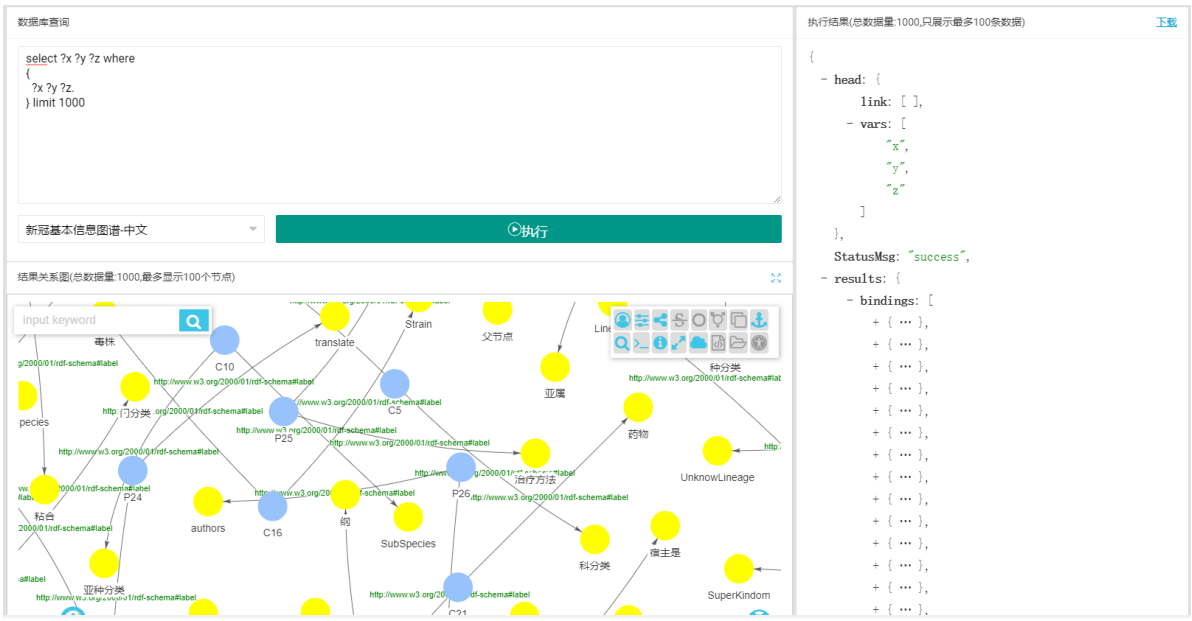
点击导出后，会创建一个zip压缩文件，下载后解压便得到对应该数据库的nt文件。



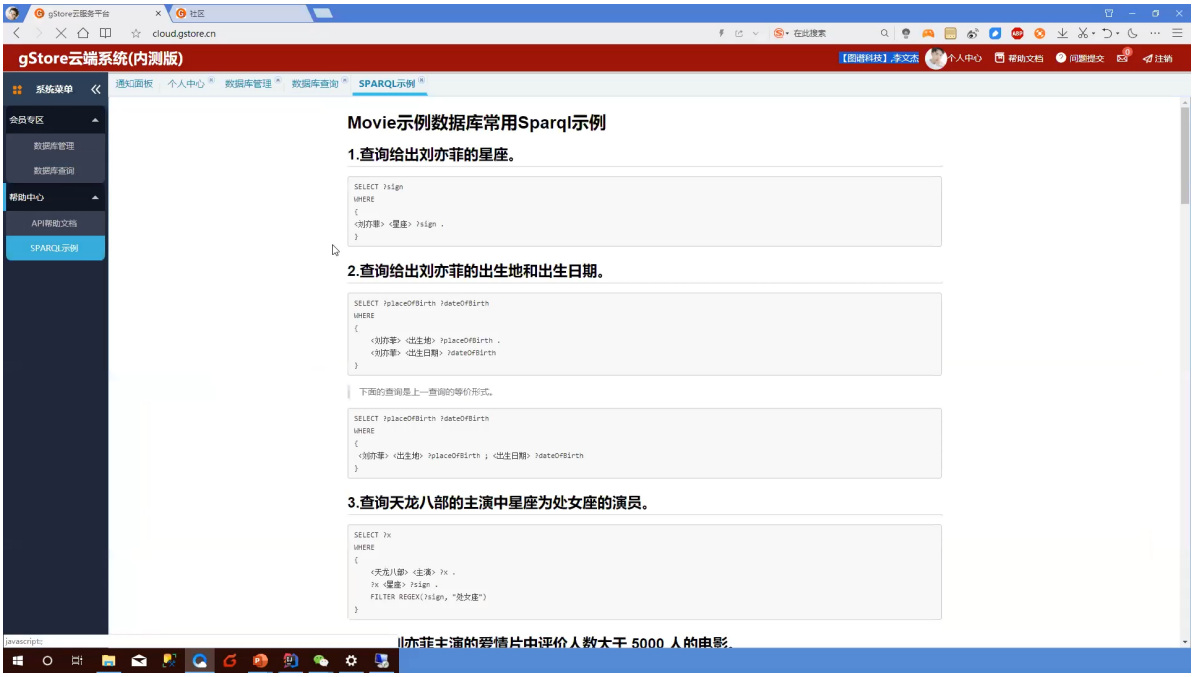
之后，数据库更名，数据库备份等更多的数据库功能也将可能上线。

8.2.5 数据库查询

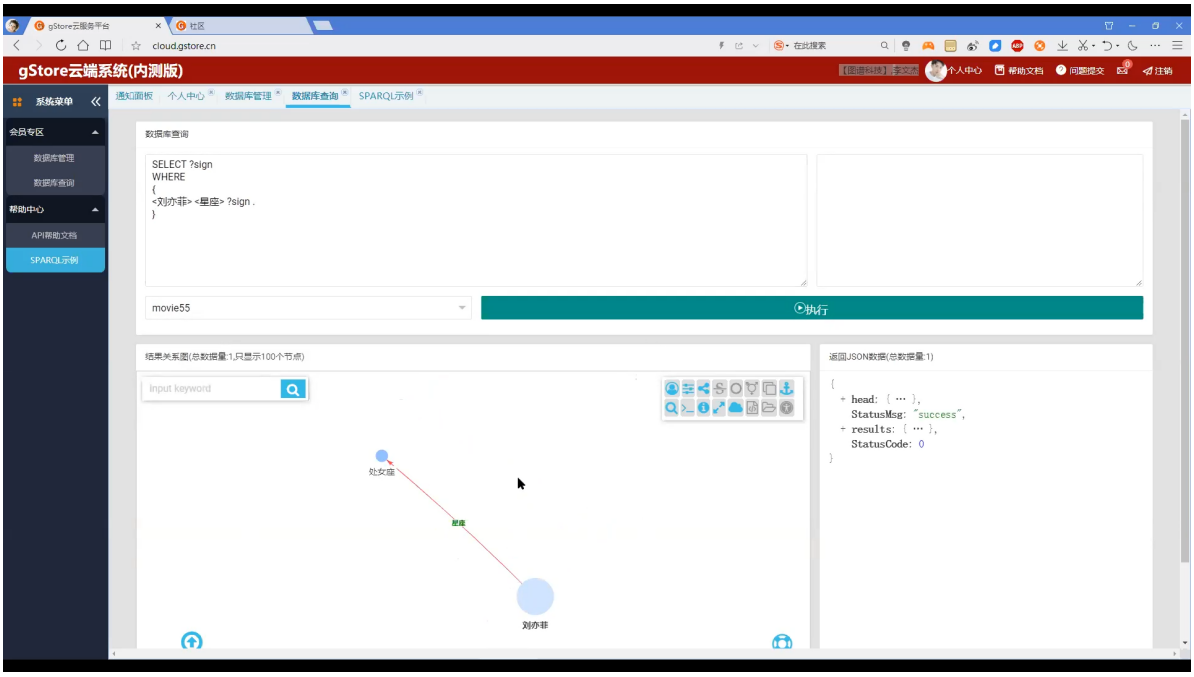
在数据库查询方面，gStore云端系统提供了一种可视化的查询界面，在下图上面的文本框中输入Sparql语句以得到结果。（**注意：考虑系统性能，关系图和JSON数据只展示100条数据，可以通过点击JSON旁边的"下载"按钮获取所有返回数据，结果关系图可以点击右边全屏按钮进行全屏显示**）



对Sparql语句不太熟悉的用户可以通过帮助中心区域中的SPARQL示例功能模块了解Sparql语句。



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 Conferenceon 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：

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）：

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


Powered
by
gStore