

1希望是附丽于存在的，有存在便有希望，有希望便是光明！

笔记本：	任务七		
创建时间：	2018/5/29 星期二 下午 5:42	更新时间：	2018/6/1 星期五 下午 1:42
作者：	18291893776@139.com		
URL：	file:///C:/Users/Administrator/Desktop/1/%E9%9D%9E%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93%E8%AF%BE%E4%BB%B6/db...		

--MySQL数据库基本操作：

--使用MySQL命令连接MySQL数据库

```
root@may-virtual-machine:/home/may# mysql -h localhost -u root -p123456
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.22-0ubuntu0.16.04.1 (Ubuntu)
```

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

--省略-h，默认localhost连接MySQL数据库

```
may@may-virtual-machine:~$ sudo su
[sudo] may 的密码：
root@may-virtual-machine:/home/may# mysql -u root -p123456
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.22-0ubuntu0.16.04.1 (Ubuntu)
```

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

数据库操作：

-- 查看当前MySQL下的所有数据库

```
mysql> show databases
-> ;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.18 sec)
```

-- 创建一个mydb2的数据库

```
mysql> create database mydb2;
Query OK, 1 row affected (0.08 sec)
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mydb2 |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

-- 删除数据库mydb2

```
mysql> drop database mydb2
-> ;
Query OK, 0 rows affected (0.68 sec)
```

```
mysql> shoe databases;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'shoe databases' at li
mysql> show databases;
```

```

+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.00 sec)

-- 创建一个mydb数据库
mysql> create database mydb;
Query OK, 1 row affected (0.06 sec)

-- 再次创建mydb数据库会报错
mysql> create database mydb;
ERROR 1007 (HY000): Can't create database 'mydb'; database exists

-- 尝试创建mydb数据库（若已存在则会报一个警告，不会报Error错误）
mysql> create database if not exists mydb;
Query OK, 1 row affected, 1 warning (0.00 sec)

-- 查看mydb的建库语句
mysql> show create database mydb;
+-----+-----+
| Database | Create Database |
+-----+-----+
| mydb | CREATE DATABASE `mydb` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+-----+
1 row in set (0.00 sec)

mysql> show create database mydb\G
***** 1. row *****
      Database: mydb
Create Database: CREATE DATABASE `mydb` /*!40100 DEFAULT CHARACTER SET latin1 */
1 row in set (0.00 sec)

-- 查看当前所在数据库位置：NULL表示没有在任何数据库中
mysql> select database();
+-----+
| database() |
+-----+
| NULL |
+-----+
1 row in set (0.00 sec)

-- 选择进入mydb数据库
mysql> use mydb;
Database changed

-- 查看当前所在数据库的位置
mysql> select database();
+-----+
| database() |
+-----+
| mydb |
+-----+
1 row in set (0.00 sec)

-- 数据表操作
=====
-- 查看当前数据库中的所有表
mysql> show tables;
Empty set (0.22 sec)

-- 创建一个uu表，内有三个字段id，name和age
mysql> create table uu(id int, name varchar(16),age int);
Query OK, 0 rows affected (1.17 sec)

-- 创建一个tt表，内有三个字段id，name和age
mysql> create table tt(
-> id int,
-> name varchar(16),
-> age int
-> );
Query OK, 0 rows affected (0.18 sec)

-- 查看当前库中有两个表
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| tt |
| uu |

```

```
+-----+
2 rows in set (0.06 sec)

-- 查看uu表的表结构
mysql> desc uu;
+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+
| id    | int(11)   | YES  |     | NULL    |       |
| name  | varchar(16) | YES  |     | NULL    |       |
| age   | int(11)   | YES  |     | NULL    |       |
+-----+
3 rows in set (2.68 sec)

-- 查看uu表的建表语句
mysql> show create table uu\G
***** 1. row *****
      Table: uu
Create Table: CREATE TABLE `uu` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(16) DEFAULT NULL,
  `age` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.03 sec)

-- 删除uu表
mysql> drop table uu;
Query OK, 0 rows affected (0.37 sec)

--查看库中的表
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| tt              |
+-----+
1 row in set (0.06 sec)

更改表名称：
      ALTER TABLE 旧表名 RENAME AS 新表名
更改AUTO_INCREMENT初始值:
      ALTER TABLE 表名称 AUTO_INCREMENT=1

更改表类型：
      ALTER TABLE 表名称 ENGINE="InnoDB"

mysql> show create table stu;
+-----+
| Table | Create
Table
+-----+
| stu   | CREATE TABLE `stu` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(8) NOT NULL,
  `age` tinyint(3) unsigned DEFAULT NULL,
  `sex` enum('m','w') NOT NULL DEFAULT 'm',
  `classid` char(8) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=latin1 |
+-----+
1 row in set (0.00 sec)

mysql> alter table stu ENGINE='MyISAM';
Query OK, 12 rows affected (1.69 sec)
Records: 12  Duplicates: 0  Warnings: 0

mysql> show create table stu;
+-----+
| Table | Create
Table
+-----+
| stu   | CREATE TABLE `stu` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(8) NOT NULL,
  `age` tinyint(3) unsigned DEFAULT NULL,
  `sex` enum('m','w') NOT NULL DEFAULT 'm',
  `classid` char(8) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`)
```

[illegible]

```
+-----+-----+
| name   | age   |
+-----+-----+
| zhangsan | 20   |
| lisi    | 22   |
| wangwu  | 25   |
| lisi    | 21   |
```

```
| xiaoli | 22 |
| xiaozhang | 19 |
+-----+-----+
6 rows in set (0.02 sec)
```

```
mysql> select * from stu;
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 2 | lisi | 22 | m | python02 |
| 3 | wangwu | 25 | w | python03 |
| 4 | li | 28 | m | python02 |
| 5 | zhang | 27 | w | python01 |
| 6 | zhao | 27 | m | python03 |
| 7 | uu01 | 18 | m | python03 |
| 8 | uu02 | 26 | m | python02 |
| 9 | uu03 | NULL | m | NULL |
| 10 | uu04 | 26 | m | python02 |
| 11 | uu06 | NULL | m | NULL |
| 12 | UU08 | 24 | m | python02 |
+-----+-----+-----+-----+
12 rows in set (0.01 sec)
```

```
mysql> select id,name,age from stu;
+-----+-----+
| id | name | age |
+-----+-----+
| 1 | zhangsan | 20 |
| 2 | lisi | 22 |
| 3 | wangwu | 25 |
| 4 | li | 28 |
| 5 | zhang | 27 |
| 6 | zhao | 27 |
| 7 | uu01 | 18 |
| 8 | uu02 | 26 |
| 9 | uu03 | NULL |
| 10 | uu04 | 26 |
| 11 | uu06 | NULL |
| 12 | UU08 | 24 |
+-----+-----+
12 rows in set (0.00 sec)
```

```
mysql> select id,name as username,age from stu;
+-----+-----+
| id | username | age |
+-----+-----+
| 1 | zhangsan | 20 |
| 2 | lisi | 22 |
| 3 | wangwu | 25 |
| 4 | li | 28 |
| 5 | zhang | 27 |
| 6 | zhao | 27 |
| 7 | uu01 | 18 |
| 8 | uu02 | 26 |
| 9 | uu03 | NULL |
| 10 | uu04 | 26 |
| 11 | uu06 | NULL |
| 12 | UU08 | 24 |
+-----+-----+
12 rows in set (0.00 sec)
```

```
mysql> select *,age+5 age5 from stu;
+-----+-----+-----+-----+-----+
| id | name | age | sex | classid | age5 |
+-----+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m | python03 | 25 |
| 2 | lisi | 22 | m | python02 | 27 |
| 3 | wangwu | 25 | w | python03 | 30 |
| 4 | li | 28 | m | python02 | 33 |
| 5 | zhang | 27 | w | python01 | 32 |
| 6 | zhao | 27 | m | python03 | 32 |
| 7 | uu01 | 18 | m | python03 | 23 |
| 8 | uu02 | 26 | m | python02 | 31 |
| 9 | uu03 | NULL | m | NULL | NULL |
| 10 | uu04 | 26 | m | python02 | 31 |
| 11 | uu06 | NULL | m | NULL | NULL |
| 12 | UU08 | 24 | m | python02 | 29 |
+-----+-----+-----+-----+-----+
12 rows in set (0.00 sec)
```

```
mysql> select *,'xi'an' as city from stu
-> ;
+-----+-----+-----+-----+-----+
| id | name | age | sex | classid | city |
+-----+-----+-----+-----+-----+
```

```

1 | zhangsan | 20 | m | python03 | xi'an |
2 | lisi | 22 | m | python02 | xi'an |
3 | wangwu | 25 | w | python03 | xi'an |
4 | li | 28 | m | python02 | xi'an |
5 | zhang | 27 | w | python01 | xi'an |
6 | zhao | 27 | m | python03 | xi'an |
7 | uu01 | 18 | m | python03 | xi'an |
8 | uu02 | 26 | m | python02 | xi'an |
9 | uu03 | NULL | m | NULL | xi'an |
10 | uu04 | 26 | m | python02 | xi'an |
11 | uu06 | NULL | m | NULL | xi'an |
12 | UU08 | 24 | m | python02 | xi'an |
+-----+-----+-----+-----+
12 rows in set (0.00 sec)

```

```

mysql> select *, 'xi'an' city from stu;
+-----+-----+-----+-----+-----+
| id | name | age | sex | classid | city |
+-----+-----+-----+-----+-----+
1 | zhangsan | 20 | m | python03 | xi'an |
2 | lisi | 22 | m | python02 | xi'an |
3 | wangwu | 25 | w | python03 | xi'an |
4 | li | 28 | m | python02 | xi'an |
5 | zhang | 27 | w | python01 | xi'an |
6 | zhao | 27 | m | python03 | xi'an |
7 | uu01 | 18 | m | python03 | xi'an |
8 | uu02 | 26 | m | python02 | xi'an |
9 | uu03 | NULL | m | NULL | xi'an |
10 | uu04 | 26 | m | python02 | xi'an |
11 | uu06 | NULL | m | NULL | xi'an |
12 | UU08 | 24 | m | python02 | xi'an |
+-----+-----+-----+-----+-----+
12 rows in set (0.00 sec)

```

```

mysql> select concat(classid, ':', name) from stu;
+-----+
| concat(classid, ':', name) |
+-----+
python03:zhangsan |
python02:lisi |
python03:wangwu |
python02:li |
python01:zhang |
python03:zhao |
python03:uu01 |
python02:uu02 |
NULL |
python02:uu04 |
NULL |
python02:UU08 |
+-----+
12 rows in set (0.06 sec)

```

```

mysql>
-- where条件查询
--1. 查询班级为python03期的所有学生信息
mysql> select * from stu where classid='python03';
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
1 | zhangsan | 20 | m | python03 |
3 | wangwu | 25 | w | python03 |
6 | zhao | 27 | m | python03 |
7 | uu01 | 18 | m | python03 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

```

--2. 查询班级为python03期，并且性别为m的所有学生信息
mysql> select * from stu where classid='python03'
and sex='m';
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
1 | zhangsan | 20 | m | python03 |
6 | zhao | 27 | m | python03 |
7 | uu01 | 18 | m | python03 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

```

--3. 查询年龄大于20，性别为w的所有信息
mysql> select * from stu where age>20 and sex='w';
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
3 | wangwu | 25 | w | python03 |
5 | zhang | 27 | w | python01 |

```

```
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

--4. 查询年龄是20~25的所有信息

```
mysql> select * from stu where age>20 and age<25;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 2 | lisi | 22 | m | python02 |
| 12 | UU08 | 24 | m | python02 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

--5. 查询年龄不在20~25的学生信息

```
mysql> select * from stu where age<20 or age>25;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 4 | li | 28 | m | python02 |
| 5 | zhang | 27 | w | python01 |
| 6 | zhao | 27 | m | python03 |
| 7 | uu01 | 18 | m | python03 |
| 8 | uu02 | 26 | m | python02 |
| 10 | uu04 | 26 | m | python02 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

--6. 查询id号为1,3,5,7,9的学生信息

```
mysql> select * from stu where id in(1,3,5,7,9);
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 3 | wangwu | 25 | w | python03 |
| 5 | zhang | 27 | w | python01 |
| 7 | uu01 | 18 | m | python03 |
| 9 | uu03 | NULL | m | NULL |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

--7. 查询classid不为null所有信息

```
mysql> select * from stu where classid is not null;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 2 | lisi | 22 | m | python02 |
| 3 | wangwu | 25 | w | python03 |
| 4 | li | 28 | m | python02 |
| 5 | zhang | 27 | w | python01 |
| 6 | zhao | 27 | m | python03 |
| 7 | uu01 | 18 | m | python03 |
| 8 | uu02 | 26 | m | python02 |
| 10 | uu04 | 26 | m | python02 |
| 12 | UU08 | 24 | m | python02 |
+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

--8. 查询班级为python01和python02期所有男生 (sex='m') 信息

```
mysql> select * from stu where (classid='python01' or classid='python02') and sex='m';
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 2 | lisi | 22 | m | python02 |
| 4 | li | 28 | m | python02 |
| 8 | uu02 | 26 | m | python02 |
| 10 | uu04 | 26 | m | python02 |
| 12 | UU08 | 24 | m | python02 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> select * from stu where classid in('python01','python02') and sex='m';
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 2 | lisi | 22 | m | python02 |
| 4 | li | 28 | m | python02 |
| 8 | uu02 | 26 | m | python02 |
| 10 | uu04 | 26 | m | python02 |
| 12 | UU08 | 24 | m | python02 |
+-----+-----+-----+-----+
5 rows in set (0.04 sec)
```

--9. 查询姓名中含有an子串的所有信息

-- like 模糊查询,支持俩个特殊符号: '%'和'_' %表示任意数量的任意字符, _表示任意一位字符

```
mysql> select * from stu where name regexp 'an';
```

```

+-----+-----+-----+-----+
| id | name   | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m   | python03 |
| 3 | wangwu   | 25 | w   | python03 |
| 5 | zhang    | 27 | w   | python01 |
+-----+-----+-----+-----+
3 rows in set (0.05 sec)

-- like 模糊查询，支持两个特殊符号：'%'和'_' %表示任意数量的任意字符，_表示任意一位字符
mysql> select * from stu where name like '%an%';
+-----+-----+-----+-----+
| id | name   | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m   | python03 |
| 3 | wangwu   | 25 | w   | python03 |
| 5 | zhang    | 27 | w   | python01 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

--10. 查询姓名是有4位任意小写字母或数字构成的信息
mysql> select * from stu where name like '____';
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 2 | lisi | 22 | m   | python02 |
| 6 | zhao | 27 | m   | python03 |
| 7 | uu01 | 18 | m   | python03 |
| 8 | uu02 | 26 | m   | python02 |
| 9 | uu03 | NULL | m   | NULL |
| 10 | uu04 | 26 | m   | python02 |
| 11 | uu06 | NULL | m   | NULL |
| 12 | UU08 | 24 | m   | python02 |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> select * from stu where name regexp '^[a-z0-9]{4}$';
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 2 | lisi | 22 | m   | python02 |
| 6 | zhao | 27 | m   | python03 |
| 7 | uu01 | 18 | m   | python03 |
| 8 | uu02 | 26 | m   | python02 |
| 9 | uu03 | NULL | m   | NULL |
| 10 | uu04 | 26 | m   | python02 |
| 11 | uu06 | NULL | m   | NULL |
| 12 | UU08 | 24 | m   | python02 |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)

-- 统计函数（聚合函数）max() min() sum() avg() count()
-- 获取最大年龄，最小年龄，年龄总和，平均年龄，总计条数
mysql> select max(age), min(age), sum(age), avg(age), count(id) from stu;
+-----+-----+-----+-----+
| max(age) | min(age) | sum(age) | avg(age) | count(id) |
+-----+-----+-----+-----+
| 28 | 18 | 243 | 24.3000 | 12 |
+-----+-----+-----+-----+
1 row in set (0.07 sec)

-- group by 字段名 分组
-- 按性别sex分组，并统计人数

mysql> select sex, count(*) from stu group by sex;
+-----+-----+
| sex | count(*) |
+-----+-----+
| m | 10 |
| w | 2 |
+-----+-----+
2 rows in set (0.00 sec)

-- 按班级分组统计每个班级的人数（排除班级信息为null的数据）
mysql> select classid, count(*) from stu where classid is not null group by classid;
+-----+-----+
| classid | count(*) |
+-----+-----+
| python01 | 1 |
| python02 | 5 |
| python03 | 4 |
+-----+-----+
3 rows in set (0.00 sec)

-- 按班级分组，并统计每个班级的男生和女生人数（排除班级信息为null的数据）
mysql> select classid, sex, count(*) from stu where classid is not null group by classid, sex;
+-----+-----+-----+
| classid | sex | count(*) |
+-----+-----+-----+

```



```
+-----+-----+
|python01|w|1|
|python02|m|5|
|python03|m|3|
|python03|w|1|
+-----+-----+
4 rows in set (0.00 sec)
```

-- 在上面的查询中，加入过滤条件(人数大于等于3的信息)

```
mysql> select classid,sex,count(*) num from stu where classid is not null group by classid,sex having num>=3;
```

```
+-----+-----+
|classid|sex|num|
+-----+-----+
|python02|m|5|
|python03|m|3|
+-----+-----+
2 rows in set (0.02 sec)
```

-- 排序：order by 字段名 asc(默认升序)|desc (降序)

```
mysql> select * from stu order by age;
```

```
+-----+-----+
|id|name|age|sex|classid|
+-----+-----+
|9|uu03|NULL|m|NULL|
|11|uu06|NULL|m|NULL|
|7|uu01|18|m|python03|
|1|zhangsan|20|m|python03|
|2|lisi|22|m|python02|
|12|UU08|24|m|python02|
|3|wangwu|25|w|python03|
|8|uu02|26|m|python02|
|10|uu04|26|m|python02|
|5|zhang|27|w|python01|
|6|zhao|27|m|python03|
|4|li|28|m|python02|
+-----+-----+
12 rows in set (0.06 sec)
```

```
mysql> select * from stu order by age asc;
```

```
+-----+-----+
|id|name|age|sex|classid|
+-----+-----+
|9|uu03|NULL|m|NULL|
|11|uu06|NULL|m|NULL|
|7|uu01|18|m|python03|
|1|zhangsan|20|m|python03|
|2|lisi|22|m|python02|
|12|UU08|24|m|python02|
|3|wangwu|25|w|python03|
|8|uu02|26|m|python02|
|10|uu04|26|m|python02|
|5|zhang|27|w|python01|
|6|zhao|27|m|python03|
|4|li|28|m|python02|
+-----+-----+
12 rows in set (0.00 sec)
```

```
mysql> select * from stu order by age desc;
```

```
+-----+-----+
|id|name|age|sex|classid|
+-----+-----+
|4|li|28|m|python02|
|5|zhang|27|w|python01|
|6|zhao|27|m|python03|
|8|uu02|26|m|python02|
|10|uu04|26|m|python02|
|3|wangwu|25|w|python03|
|12|UU08|24|m|python02|
|2|lisi|22|m|python02|
|1|zhangsan|20|m|python03|
|7|uu01|18|m|python03|
|9|uu03|NULL|m|NULL|
|11|uu06|NULL|m|NULL|
+-----+-----+
12 rows in set (0.00 sec)
```

```
mysql> select * from stu order by classid desc;
```

```
+-----+-----+
|id|name|age|sex|classid|
+-----+-----+
|1|zhangsan|20|m|python03|
|3|wangwu|25|w|python03|
|6|zhao|27|m|python03|
|7|uu01|18|m|python03|
|2|lisi|22|m|python02|
```

```
| 4 | li | 28 | m | python02 |
| 8 | uu02 | 26 | m | python02 |
| 10 | uu04 | 26 | m | python02 |
| 12 | UU08 | 24 | m | python02 |
| 5 | zhang | 27 | w | python01 |
| 9 | uu03 | NULL | m | NULL |
| 11 | uu06 | NULL | m | NULL |
+-----+-----+-----+-----+
```

12 rows in set (0.00 sec)

```
mysql> select * from stu order by classid asc;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 9 | uu03 | NULL | m | NULL |
| 11 | uu06 | NULL | m | NULL |
| 5 | zhang | 27 | w | python01 |
| 2 | lisi | 22 | m | python02 |
| 4 | li | 28 | m | python02 |
| 8 | uu02 | 26 | m | python02 |
| 10 | uu04 | 26 | m | python02 |
| 12 | UU08 | 24 | m | python02 |
| 1 | zhangsan | 20 | m | python03 |
| 3 | wangwu | 25 | w | python03 |
| 6 | zhao | 27 | m | python03 |
| 7 | uu01 | 18 | m | python03 |
+-----+-----+-----+-----+
```

12 rows in set (0.00 sec)

```
mysql> select * from stu order by classid asc, age desc;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 9 | uu03 | NULL | m | NULL |
| 11 | uu06 | NULL | m | NULL |
| 5 | zhang | 27 | w | python01 |
| 4 | li | 28 | m | python02 |
| 8 | uu02 | 26 | m | python02 |
| 10 | uu04 | 26 | m | python02 |
| 12 | UU08 | 24 | m | python02 |
| 2 | lisi | 22 | m | python02 |
| 6 | zhao | 27 | m | python03 |
| 3 | wangwu | 25 | w | python03 |
| 1 | zhangsan | 20 | m | python03 |
| 7 | uu01 | 18 | m | python03 |
+-----+-----+-----+-----+
```

12 rows in set (0.00 sec)

--获取部分数据：limit
-- 分页公式：limit (页号-1)*页大小,页大小

```
mysql> select * from stu limit 0,3;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 2 | lisi | 22 | m | python02 |
| 3 | wangwu | 25 | w | python03 |
+-----+-----+-----+-----+
```

3 rows in set (0.00 sec)

```
mysql> select * from stu limit 3,3;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 4 | li | 28 | m | python02 |
| 5 | zhang | 27 | w | python01 |
| 6 | zhao | 27 | m | python03 |
+-----+-----+-----+-----+
```

3 rows in set (0.00 sec)

```
mysql> select * from stu limit 6,3;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 7 | uu01 | 18 | m | python03 |
| 8 | uu02 | 26 | m | python02 |
| 9 | uu03 | NULL | m | NULL |
+-----+-----+-----+-----+
```

3 rows in set (0.00 sec)

```
mysql> select * from stu limit 9,3;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 10 | uu04 | 26 | m | python02 |
+-----+-----+-----+-----+
```

```
| 11 | uu06 | NULL | m | NULL |
| 12 | UU08 | 24 | m | python02 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

-- 综合查询练习

--1. 查询python03期所有学员，按年龄降序排序

```
mysql> select * from stu where classid='python03' order by age desc;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 6 | zhao | 27 | m | python03 |
| 3 | wangwu | 25 | w | python03 |
| 1 | zhangsan | 20 | m | python03 |
| 7 | uu01 | 18 | m | python03 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

--2. 查询python03期，年龄最大的3位学员信息

```
mysql> select * from stu where classid='python03' order by age desc limit 3;
```

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 6 | zhao | 27 | m | python03 |
| 3 | wangwu | 25 | w | python03 |
| 1 | zhangsan | 20 | m | python03 |
+-----+-----+-----+-----+
3 rows in set (0.03 sec)
```

--3. 统计每个班级人数，并按人数降序排序（排除班级信息为null的数据）

```
mysql> select classid,count(*) from stu where cla
```

```
+-----+-----+
| classid | count(*) |
+-----+-----+
| python02 | 5 |
| python03 | 4 |
| python01 | 1 |
+-----+-----+
3 rows in set (0.00 sec)
```

--4. 统计每个班级年龄在20~30的学员人数信息，并按人数降序排序（排除班级信息为null的数据）

```
mysql> select classid,count(*) from stu where classid is not null and age between 20 and 30
-> group by classid order by m desc
```

```
+-----+-----+
| classid | count(*) |
+-----+-----+
| python02 | 5 |
| python03 | 3 |
| python01 | 1 |
+-----+-----+
3 rows in set (0.02 sec)
```

--5. 统计每个班级男女生人数最多3条记录信息。（排除班级信息为null的数据）

```
mysql> select classid,sex,count(*) m from stu where classid is not null group by classid,sex order by m desc limit 3
-> ;
```

```
+-----+-----+-----+
| classid | sex | m |
+-----+-----+-----+
| python02 | m | 5 |
| python03 | m | 3 |
| python03 | w | 1 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

数据的DQL操作：数据查询

=====

格式：

```
select [字段列表]* from 表名
[where 搜索条件]
[group by 分组字段 [having 子条件]]
[order by 排序 asc|desc]
[limit 分页参数]
```

```
mysql> select classid,count(*) from stu group by classid order by count(*) desc;
```

```
+-----+-----+
| classid | count(*) |
+-----+-----+
| python02 | 5 |
| python03 | 4 |
| NULL | 2 |
| python01 | 1 |
+-----+-----+
4 rows in set (0.01 sec)
```

mysql>

mysql> select classid,count(*) from stu group by classid having count(*)>=3 order by count(*) desc;

注意：这种情况使用having语句主要是因为count(*) from stu group by classid 相当于一个整体，只有是group by classid 之后才会知道count(*)，所以千万不能在count(*)后面写where count(count(*)>=3。

```
+-----+-----+
| classid | count(*) |
+-----+-----+
| python02 | 5 |
| python03 | 4 |
+-----+-----+
2 rows in set (0.06 sec)
```

mysql> select ntid,count(*) from news where click>=100 group by ntid;

注意：这个语句和上个语句也是截然不同。

```
+-----+-----+
| ntid | count(*) |
+-----+-----+
| 4 | 2 |
+-----+-----+
1 row in set (0.00 sec)
```

mysql>

-- 多表查询：

-- 1. 嵌套查询（一个查询的结果是另外查询的条件）

-- 2. where关联查询，相当于join连接查询中的内联查询。都是要两个表中同时存在的才可以显示。

-- 3. join连接查询：内联inner join，左联left join，右联right join

-- 1. 嵌套查询：

-- 查询年龄最大的学生信息

mysql> select * from stu where age=(select max(age) from stu);

```
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 4 | li | 28 | m | python02 |
+-----+-----+-----+-----+
1 row in set (0.06 sec)
```

mysql>

mysql> select ntid,count(*) from news group by ntid;

```
+-----+-----+
| ntid | count(*) |
+-----+-----+
| 1 | 1 |
| 3 | 1 |
| 4 | 4 |
+-----+-----+
3 rows in set (0.14 sec)
```

mysql> select * from news;

```
+-----+-----+-----+-----+-----+-----+
| id | title | ntid | content | click | addtime |
+-----+-----+-----+-----+-----+-----+
| 1 | 骑共享单车犯法 | 1 | 这个事情官方还没有做出回应 | 20 | NULL |
| 2 | 武警特战排爆手 | 4 | 这个事情官方还没有做出回应 | 32 | NULL |
| 3 | 国外武装直升机型号发展与作战能力分析 | 4 | 这个事情官方还没有做出回应 | 234 | NULL |
| 4 | 俄战机过去一周在边境拦截外国侦查机11次 | 4 | 这个事情官方还没有做出回应 | 21 | NULL |
| 5 | 中国空军战机进行远洋训练 | 4 | 事情官方还没有做出回应 | 128 | NULL |
| 6 | 中国驻韩使馆举行中韩建交25周年纪念招待会 | 3 | 事情官方还没有做出回应 | 34 | NULL |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

对象		news @mydb (test) - 表		news @mydb (test) - 表			
开始事务		备注		筛选		排序	
导入		导出					
id	title	ntid	content	click	addtime		
1	骑共享单车犯法	1	这个事情官方还没有做出回应	20	(Null)		
2	武警特战排爆手	4	这个事情官方还没有做出回应	32	(Null)		
3	国外武装直升机型号发展与作战能力分析	4	这个事情官方还没有做出回应	234	(Null)		
4	俄战机过去一周在边境拦截外国侦查机11次	4	这个事情官方还没有做出回应	21	(Null)		
5	中国空军战机进行远洋训练	4	事情官方还没有做出回应	128	(Null)		
6	中国驻韩使馆举行中韩建交25周年纪念招待会	3	事情官方还没有做出回应	34	(Null)		

The screenshot shows the DBeaver interface with the 'news' table selected. The table structure and data are as follows:

id	name
1	娱乐新闻
2	体育新闻
3	国际新闻
4	军事新闻

```
mysql> select ntid,count(*) from news group by ntid order by count(*) desc;
```

```

+-----+-----+
| ntid | count(*) |
+-----+-----+
| 4 | 4 |
| 1 | 1 |
| 3 | 1 |
+-----+-----+
3 rows in set (0.06 sec)

```

```
mysql> select news.ntid,count(*),ntype.name from news,ntype where news.ntid=ntype.e.id group by news.ntid order by count(*) desc;
```

```
+-----+-----+-----+
| ntid | count(*) | name      |
+-----+-----+-----+
| 4    | 4        | 军事新闻 |
| 1    | 1        | 娱乐新闻 |
| 3    | 1        | 国际新闻 |
+-----+-----+-----+
3 rows in set (0.24 sec)
```

```
mysql>
```

2.where关联查询：

```
mysql> select n.id,n.title,t.name from news n,ntype t where n.ntid=t.id;
```

```

+-----+-----+-----+-----+
| id | title                                     | name |
+-----+-----+-----+-----+
| 1 | 骑共享单车犯法                         | 娱乐新闻 |
| 2 | 武警特战排爆手                         | 军事新闻 |
| 3 | 国外武装直升机型号发展与作战能力分析 | 军事新闻 |
| 4 | 俄战机过去一周在边境拦截外国侦查机11次 | 军事新闻 |
| 5 | 中国空军战机进行远洋训练               | 军事新闻 |
| 6 | 中国驻韩使馆举行中韩建交25周年纪念招待会 | 国际新闻 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)

```

和内联的查询结果是一样的，实际上它们就是一样的，只是写法不同。

```
mysql> select n.id,n.title,t.name from news n inner join ntype t on n.ntid=t.id;
```

```

+-----+-----+-----+-----+
| id | title | name |
+-----+-----+-----+
| 1 | 骑共享单车犯法 | 娱乐新闻 |
| 2 | 武警特战排爆手 | 军事新闻 |
| 3 | 国外武装直升机型号发展与作战能力分析 | 军事新闻 |
| 4 | 俄战机过去一周在边境拦截外国侦查机11次 | 军事新闻 |
| 5 | 中国空军战机进行远洋训练 | 军事新闻 |
| 6 | 中国驻韩使馆举行中韩建交25周年纪念日招待会 | 国际新闻 |
+-----+-----+-----+
6 rows in set (0.00 sec)

```

```
mysql> select * from type;
```

```

+-----+-----+
| id | name | pid |
+-----+-----+
| 1 | 服装 | 0 |
| 2 | 数码 | 0 |
| 3 | 男装 | 1 |
| 4 | 手机 | 2 |
| 5 | 相机 | 2 |
| 6 | 电脑 | 2 |
| 7 | 女装 | 1 |
| 8 | 童装 | 1 |
| 9 | 食品 | 0 |
|10 | 零食 | 9 |
|11 | 特产 | 9 |
|12 | 休闲装 | 1 |
+-----+-----+
12 rows in set (0.00 sec)

```

```
mysql> desc type;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
name	varchar(16)	NO		NULL	
pid	int(10) unsigned	YES		NULL	

3 rows in set (0.00 sec)

-- 查询二级类别信息，并关联出他们的父类别名称
mysql> select t1.id,t1.name,t2.name from type t1,type t2 where t1.pid!=0 and t1.pid=t2.id;

id	name	name
3	男装	服装
4	手机	数码
5	相机	数码
6	电脑	数码
7	女装	服装
8	童装	服装
10	零食	食品
11	特产	食品
12	休闲装	服装

9 rows in set (0.01 sec)

--统计每个一级类别下都有多少个子类别。
mysql> select t1.id,t1.name,count(t2.id) from type t1,type t2 where t1.pid=0 and t1.id=t2.pid group by t1.id;

id	name	count(t2.id)
1	服装	4
2	数码	3
9	食品	2

3 rows in set (0.00 sec)

3.join查询---左联查询：

```
mysql> select n.id,n.title,t.name from news n left join ntype t on n.ntid=t.id;
```

id	title	name
1	骑共享单车犯法	娱乐新闻
6	中国驻韩使馆举行中韩建交25周年纪念招待会	国际新闻
2	武警特战排爆手	军事新闻
3	国外武装直升机型号发展与作战能力分析	军事新闻
4	俄战机过去一周在边境拦截外国侦查机11次	军事新闻
5	中国空军战机进行远洋训练	军事新闻

6 rows in set (0.00 sec)

mysql>
把news的这张表乱加了一条数据。再进行左联查询：
mysql> select n.id,n.title,t.name from news n left join ntype t on n.ntid=t.id;

id	title	name
1	骑共享单车犯法	娱乐新闻
6	中国驻韩使馆举行中韩建交25周年纪念招待会	国际新闻
2	武警特战排爆手	军事新闻
3	国外武装直升机型号发展与作战能力分析	军事新闻
4	俄战机过去一周在边境拦截外国侦查机11次	军事新闻
5	中国空军战机进行远洋训练	军事新闻
7	aaaaaaaaa	NULL

7 rows in set (0.00 sec)

右联查询也是同样的道理，将left换成right就OK！

--3. 修改数据

-- 格式：update 表名 set 字段名1=值1[字段名2=值2,...] [where 条件...]

```
mysql> select * from uu;
```

id	name	age
1	zhangsan	20
2	lisi	22
3	wangwu	25
4	lisi	21
5	xiaoli	22
6	xiaozhang	19

6 rows in set (1.14 sec)

```
-- 将id值为4的信息age改为30 (修改)
mysql> update uu set age=30 where id=4;
Query OK, 1 row affected (0.34 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from uu;
+-----+-----+
| id | name   | age |
+-----+-----+
| 1 | zhangsan | 20 |
| 2 | lisi    | 22 |
| 3 | wangwu  | 25 |
| 4 | lisi    | 30 |
| 5 | xiaoli  | 22 |
| 6 | xiaozhang | 19 |
+-----+-----+
6 rows in set (0.00 sec)
```

```
-- 将id为8,10和12的年龄age改为26, 班级classid改为python02
mysql> select * from stu;
```

```
+-----+-----+
| id | name   | age | sex | classid |
+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 2 | lisi    | 22 | m | python02 |
| 3 | wangwu  | 25 | w | python03 |
| 4 | li      | 28 | m | python02 |
| 5 | zhang   | 27 | w | python01 |
| 6 | zhao    | 27 | m | python03 |
| 7 | uu01    | 18 | m | python03 |
| 8 | uu02    | NULL | m | NULL |
| 9 | uu03    | NULL | m | NULL |
| 10 | uu04    | NULL | m | NULL |
| 11 | uu06    | NULL | m | NULL |
| 12 | UU08    | NULL | m | NULL |
+-----+-----+
12 rows in set (0.03 sec)
```

```
mysql> update stu set age=26, classid='python02' where id in(8,10,12);
Query OK, 3 rows affected (0.03 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

```
mysql> select * from stu;
+-----+-----+
| id | name   | age | sex | classid |
+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 2 | lisi    | 22 | m | python02 |
| 3 | wangwu  | 25 | w | python03 |
| 4 | li      | 28 | m | python02 |
| 5 | zhang   | 27 | w | python01 |
| 6 | zhao    | 27 | m | python03 |
| 7 | uu01    | 18 | m | python03 |
| 8 | uu02    | 26 | m | python02 |
| 9 | uu03    | NULL | m | NULL |
| 10 | uu04    | 26 | m | python02 |
| 11 | uu06    | NULL | m | NULL |
| 12 | UU08    | 26 | m | python02 |
+-----+-----+
12 rows in set (0.00 sec)
```

mysql>

```
--修改年龄的大小
mysql> update stu set age=age-2 where id=12;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from stu;
+-----+-----+
| id | name   | age | sex | classid |
+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 2 | lisi    | 22 | m | python02 |
| 3 | wangwu  | 25 | w | python03 |
| 4 | li      | 28 | m | python02 |
| 5 | zhang   | 27 | w | python01 |
| 6 | zhao    | 27 | m | python03 |
| 7 | uu01    | 18 | m | python03 |
| 8 | uu02    | 26 | m | python02 |
| 9 | uu03    | NULL | m | NULL |
| 10 | uu04    | 26 | m | python02 |
| 11 | uu06    | NULL | m | NULL |
| 12 | UU08    | 24 | m | python02 |
+-----+-----+
```

```
12 rows in set (0.00 sec)

mysql>

-- 2. 删除数据
-- 格式：delete from 表名 [where 条件 [分组、排序、limit]]
=====
-- 删除id为5的所有信息
mysql> delete from uu where id=5;
Query OK, 1 row affected (0.05 sec)

mysql> select * from uu;
+-----+-----+-----+
| id | name | age |
+-----+-----+-----+
| 1 | zhangsan | 20 |
| 2 | lisi | 22 |
| 3 | wangwu | 25 |
| 4 | lisi | 30 |
| 6 | xiaozhang | 19 |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
--查看stu表的所有数据
mysql> select * from stu;
+-----+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+-----+
| 1 | zhangsan | 20 | m | python03 |
| 2 | lisi | 22 | m | python02 |
| 3 | wangwu | 25 | w | python03 |
| 8 | xiaoli | 28 | m | python02 |
| 9 | zhang | 21 | w | python01 |
| 10 | zhao | 27 | m | python03 |
| 11 | uu01 | 18 | m | python03 |
+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

-- 删除id为22的信息
mysql> delete from stu where id=22
-> ;
Query OK, 0 rows affected (0.06 sec)

-- 删除id大于100的所有信息
mysql> delete from stu where id > 100;
Query OK, 0 rows affected (0.00 sec)

-- 删除id是100~200的所有信息
mysql> delete from stu where id>=100 and id<=200;
Query OK, 0 rows affected (0.00 sec)

-- 删除性别为w，年龄大于25的所有信息
mysql> delete from stu where sex='w' and age>25;
Query OK, 0 rows affected (0.00 sec)

mysql>

-- 数据类型：数值类型
=====
mysql> create table m1(
-> id int unsigned auto_increment primary key,
-> age tinyint unsigned,
-> num tinyint,
-> n1 int(4)
-> ),
-> n2 int(6) zerofill);
Query OK, 0 rows affected (0.56 sec)

mysql> desc m1;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| age | tinyint(3) unsigned | YES | | NULL | |
| num | tinyint(4) | YES | | NULL | |
| n1 | int(4) | YES | | NULL | |
| n2 | int(6) unsigned zerofill | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.14 sec)

mysql> insert into m1 values(1,88,-20,12345,5678);
Query OK, 1 row affected (0.07 sec)

mysql> select * from m1;
```



```
+-----+
| id | age | num | n1 | n2 |
+-----+
| 1 | 88 | -20 | 12345 | 005678 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> insert into m1 values(2,300,128,32423,2345678);
ERROR 1264 (22003): Out of range value for column 'age' at row 1
mysql> select * from m1;
+-----+
| id | age | num | n1 | n2 |
+-----+
| 1 | 88 | -20 | 12345 | 005678 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> insert into m1 values(1,23,34,100,34);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> insert into m1 values(null,23,34,100,34);
Query OK, 1 row affected (0.05 sec)
```

```
mysql> select * from m1;
+-----+
| id | age | num | n1 | n2 |
+-----+
| 1 | 88 | -20 | 12345 | 005678 |
| 2 | 23 | 34 | 100 | 000034 |
+-----+
2 rows in set (0.00 sec)
```

-- 子串类型实例
=====

```
mysql> create table m2(
-> c1 char(8),
-> c2 varchar(8),
-> c3 text,
-> c4 enum('y','n') not null default 'y'
-> );
Query OK, 0 rows affected (0.34 sec)
```

```
mysql> desc m2;
+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+
| c1    | char(8)   | YES  |     | NULL    |       |
| c2    | varchar(8)| YES  |     | NULL    |       |
| c3    | text      | YES  |     | NULL    |       |
| c4    | enum('y','n') | NO  |     | y       |       |
+-----+
4 rows in set (0.10 sec)
```

```
mysql> insert m2 values('qwertyuio','qwertyuio','we','n');
ERROR 1406 (22001): Data too long for column 'c1' at row 1
mysql> insert m2 values('qwertyui','qwertyui','we','n');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from m2;
+-----+
| c1    | c2      | c3   | c4 |
+-----+
| qwertyui | qwertyui | we  | n  |
+-----+
1 row in set (0.00 sec)
```

```
mysql> create table m3(
-> d1 date,
-> d2 datetime,
-> d3 timestamp);
Query OK, 0 rows affected (0.16 sec)
```

-- 时间日期类型
=====

```
mysql> desc m3;
+-----+
| Field | Type      | Null | Key | Default      | Extra      |
+-----+
| d1    | date      | YES  |     | NULL         |            |
| d2    | datetime  | YES  |     | NULL         |            |
| d3    | timestamp | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+-----+
3 rows in set (0.00 sec)
```

```
mysql> insert into m3 values('2018-05-26',now(),current_timestamp());
Query OK, 1 row affected (0.06 sec)
```

```
mysql> select * from m3;
+-----+-----+-----+
| d1      | d2      | d3      |
+-----+-----+-----+
| 2018-05-26 | 2018-05-26 13:21:08 | 2018-05-26 13:21:08 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql>
```

```
-- 表操作实战
```

```
mysql> create table stu(
-> id int unsigned not null auto_increment primary key,
-> name varchar(8) not null unique,
-> age tinyint unsigned,
-> sex enum('m','w') not null default 'm',
-> classid char(8)
-> );
Query OK, 0 rows affected (0.13 sec)
```

```
--表结构说明：
--id 整型int 无符号 非空 自增 主键约束
--name 可变子串8个长度 非空 唯一性约束
--age 非常小整型 无符号约束
--sex 性别枚举类型m或w值，非空，默认值m
--classid 定长子串8个长度
```

```
mysql> desc stu;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id     | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name   | varchar(8)      | NO   | UNI | NULL    |                |
| age    | tinyint(3) unsigned | YES  |     | NULL    |                |
| sex    | enum('m','w')   | NO   |     | m        |                |
| classid | char(8)         | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+
5 rows in set (0.06 sec)
```

```
mysql> show create table stu\G
***** 1. row *****
Table: stu
Create Table: CREATE TABLE `stu` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(8) NOT NULL,
  `age` tinyint(3) unsigned DEFAULT NULL,
  `sex` enum('m','w') NOT NULL DEFAULT 'm',
  `classid` char(8) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

```
-- 数据库备份与恢复
```

```
root@may-virtual-machine:/home/may# mysqldump -u root -p mydb > mydb.sql
Enter password:
root@may-virtual-machine:/home/may# ls
examples.desktop show 模板 图片 下载 桌面
mydb.sql          公共的 视频 文档 音乐
root@may-virtual-machine:/home/may# vim mydb.sql
root@may-virtual-machine:/home/may# mysqldump -u root -p mydb stu > mydb_stu.sql
Enter password:
root@may-virtual-machine:/home/may# mysqldump -u root -p mydb m1 > m1.sql
Enter password:
root@may-virtual-machine:/home/may# ls;
examples.desktop mydb_stu.sql 模板 文档 桌面
m1.sql           show      视频 下载
mydb.sql         公共的   图片 音乐

root@may-virtual-machine:/home/may# mysql -u root -p mydb < mydb.sql
Enter password:
root@may-virtual-machine:/home/may#
```

```
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| m1              |
| m2              |
| m3              |
```

```
| stu      |
| tt       |
| uu       |
+-----+
6 rows in set (0.00 sec)

mysql> select * from stu;
Empty set (0.00 sec)

mysql> select * from m1;
+-----+-----+-----+-----+
| id | age | num | n1 | n2 |
+-----+-----+-----+-----+
| 1 | 88 | -20 | 12345 | 005678 |
| 2 | 23 | 34 | 100 | 000034 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> drop table m1
-> ;
Query OK, 0 rows affected (0.15 sec)

mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| m2              |
| m3              |
| stu             |
| tt              |
| uu              |
+-----+
5 rows in set (0.00 sec)

mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| m2              |
| m3              |
| stu             |
| tt              |
| uu              |
+-----+
5 rows in set (0.00 sec)

mysql> create table m1;
ERROR 1113 (42000): A table must have at least 1 column
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| m1              |
| m2              |
| m3              |
| stu             |
| tt              |
| uu              |
+-----+
6 rows in set (0.00 sec)

mysql> select * from m1;
+-----+-----+-----+-----+
| id | age | num | n1 | n2 |
+-----+-----+-----+-----+
| 1 | 88 | -20 | 12345 | 005678 |
| 2 | 23 | 34 | 100 | 000034 |
+-----+-----+-----+-----+
2 rows in set (0.03 sec)

mysql>

-- 修改表结构实例
=====
mysql> show create table tt\G
***** 1. row *****

Table: tt
Create Table: CREATE TABLE `tt` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(16) DEFAULT NULL,
  `age` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql> desc tt;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id    | int(11)   | YES  |     |         |       |
| name  | varchar(16) | YES  |     |         |       |
| age   | int(11)   | YES  |     |         |       |
+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
| id | int(11) | YES | | NULL | |
| name | varchar(16) | YES | | NULL | |
| age | int(11) | YES | | NULL | |
+-----+-----+-----+-----+
```

3 rows in set (0.00 sec)

--1. 在tt表末尾添加一个phone字段，类型varchar(11)，无其他约束
mysql> alter table tt add phone varchar(11);
Query OK, 0 rows affected (0.29 sec)
Records: 0 Duplicates: 0 Warnings: 0

--2. 在tt表中age字段后添加一个address字段，类型varchar(100)，无其他约束
mysql> alter table tt add address varchar(100) after age;
Query OK, 0 rows affected (0.15 sec)
Records: 0 Duplicates: 0 Warnings: 0

--3. 在tt表首位插入一个mm字段，类型int
mysql> alter table tt add mm int first;
Query OK, 0 rows affected (0.10 sec)
Records: 0 Duplicates: 0 Warnings: 0

```
mysql> desc tt;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| mm | int(11) | YES | | NULL | |
| id | int(11) | YES | | NULL | |
| name | varchar(16) | YES | | NULL | |
| age | int(11) | YES | | NULL | |
| address | varchar(100) | YES | | NULL | |
| phone | varchar(11) | YES | | NULL | |
+-----+-----+-----+-----+
```

6 rows in set (0.00 sec)

mysql>

--4. 删除tt表的mm字段
mysql> alter table tt drop mm;
Query OK, 0 rows affected (0.12 sec)
Records: 0 Duplicates: 0 Warnings: 0

--5. 修改字段：tt表age字段类型改为tinyint类型，unsigned not null default 20
mysql> alter table tt modify age tinyint unsigned not null default 20;
Query OK, 0 rows affected (0.17 sec)
Records: 0 Duplicates: 0 Warnings: 0

--6. 修改name字段名为username
mysql> alter table tt change name username varchar(16);
Query OK, 0 rows affected (0.31 sec)
Records: 0 Duplicates: 0 Warnings: 0

```
mysql> desc tt;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id | int(11) | YES | | NULL | |
| username | varchar(16) | YES | | NULL | |
| age | tinyint(3) unsigned | NO | | 20 | |
| address | varchar(100) | YES | | NULL | |
| phone | varchar(11) | YES | | NULL | |
+-----+-----+-----+-----+
```

5 rows in set (0.00 sec)

mysql>

Windows操作数据库：
进入MySQL的配置文件，将bind-address注释掉
1.sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf这是配置文件的地址

```

21 # The following values assume you have at least 32M ram
22
23 [mysqld_safe]
24 socket          = /var/run/mysqld/mysqld.sock
25 nice            = 0
26
27 [mysqld]
28 #
29 # * Basic Settings
30 #
31 user            = mysql
32 pid-file        = /var/run/mysqld/mysqld.pid
33 socket          = /var/run/mysqld/mysqld.sock
34 port           = 3306
35 basedir        = /usr
36 datadir        = /var/lib/mysql
37 tmpdir          = /tmp
38 lc-messages-dir = /usr/share/mysql
39 skip-external-locking
40 #
41 # Instead of skip-networking the default is now to listen only on
42 # localhost which is more compatible and is not less secure.
43 #bind-address    = 127.0.0.1
44 #
45 # * Fine Tuning
-- 插入 --

```

(要搜索什么信息可以加一个正斜杠+查找内容。比如：/127 这样就会直接跳转到相应的那一行)

2.进行授权

```
mysql> grant all on *.* to root@'%' identified by '123456' with grant option;
```

Query OK, 0 rows affected, 1 warning (1.34 sec)

3.进行刷新 (如果不刷新的话就需要对数据库进行重启)

```
mysql> flush privileges;
```

Query OK, 0 rows affected (0.46 sec)

4.退出MySQL

```
mysql> exit;
```

Bye

5.重启MySQL (安全起见)

6.查看自己虚拟机的IP地址

```
root@may-virtual-machine:/home/may# ifconfig
```

```

ens33  Link encap:以太网 硬件地址 00:0c:29:f3:f7:8d
        inet 地址:192.168.220.129 广播:192.168.220.255 掩码:255.255.255.0
        inet6 地址: fe80::e61:809a:a6d1:cf4e/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
        接收数据包:20931 错误:0 丢弃:0 过载:0 帧数:0
        发送数据包:8368 错误:0 丢弃:0 过载:0 载波:0
        碰撞:0 发送队列长度:1000
        接收字节:27335354 (27.3 MB) 发送字节:650539 (650.5 KB)

```

```
lo      Link encap:本地环回
```

```
        inet 地址:127.0.0.1 掩码:255.0.0.0
```

```
        inet6 地址: ::1/128 Scope:Host
```

```
        UP LOOPBACK RUNNING  MTU:65536 跃点数:1
```

```
        接收数据包:268 错误:0 丢弃:0 过载:0 帧数:0
```

```
        发送数据包:268 错误:0 丢弃:0 过载:0 载波:0
```

```
        碰撞:0 发送队列长度:1000
```

```
        接收字节:22294 (22.2 KB) 发送字节:22294 (22.2 KB)
```

```
root@may-virtual-machine:/home/may#
```

7.进入数据库

```
root@may-virtual-machine:/home/may# mysql -u root -p123456
```

```
mysql: [Warning] Using a password on the command line interface can be insecure.
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 3
```

```
Server version: 5.7.22-0ubuntu0.16.04.1 (Ubuntu)
```

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
mysql> show databases;
```

```

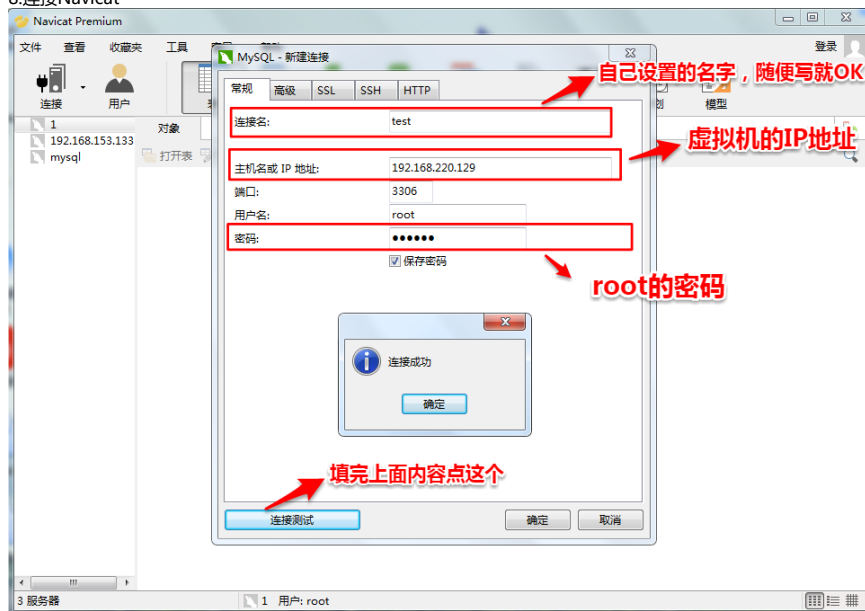
+-----+
| Database |
+-----+
| information_schema |
| mydb      |
| mysql     |
| performance_schema |

```

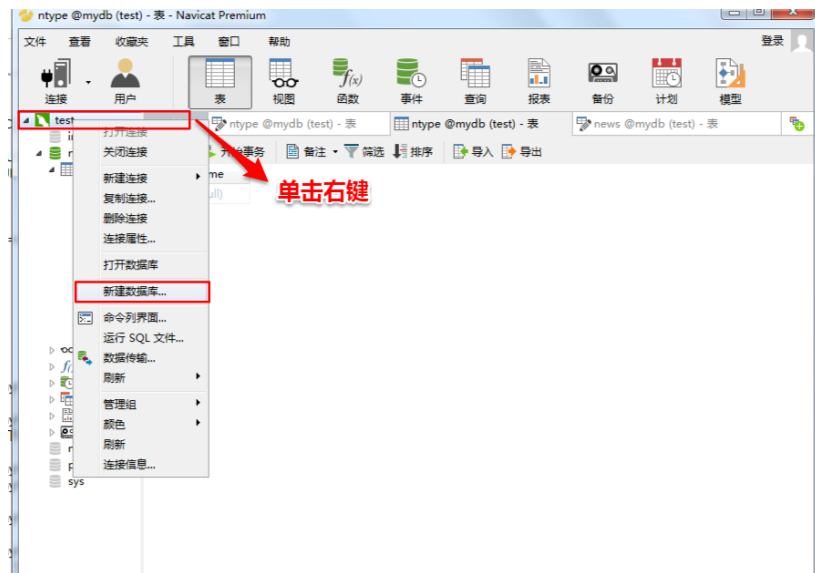
```
| sys |
+-----+
5 rows in set (0.16 sec)
```

mysql>

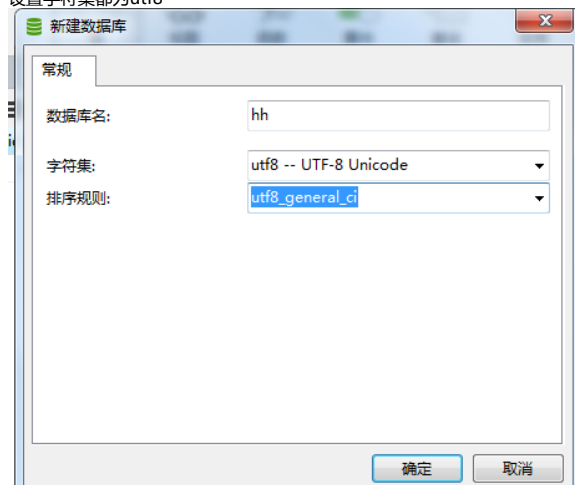
8.连接Navicat



在navicat创建数据库等操作



设置字符集都为utf8



查看字符集等信息：

```
mysql> \s
-----
mysql Ver 14.14 Distrib 5.7.22, for Linux (x86_64) using EditLine wrapper

Connection id:      3
Current database:   mydb
Current user:       root@localhost
SSL:                Not in use
Current pager:      stdout
Using outfile:      ''
Using delimiter:    ;
Server version:     5.7.22-0ubuntu0.16.04.1 (Ubuntu)
Protocol version:   10
Connection:         Localhost via UNIX socket
Server characterset: latin1
Db characterset:    latin1
Client characterset: utf8
Conn. characterset: utf8
UNIX socket:        /var/run/mysqld/mysqld.sock
Uptime:             47 min 0 sec

Threads: 4  Questions: 89  Slow queries: 0  Opens: 121  Flush tables: 1  Open tables: 40  Queries per second avg: 0.031
-----

mysql>
```

注意：字符集不统一是导致乱码的最主要的原因。乱码有两种情况，一种是数据完全被破坏，无法恢复。另一种是当前显示乱码，但是可以恢复。
例如：

```
mysql> set names latin1;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from ntype;
+-----+
| id      | name          |
+-----+
| 0000000001 | yu le xin wen |
| 0000000002 | ti yu xin wen |
| 0000000003 | guo ji xin wen |
| 0000000004 | jun shi xin wen |
| 0000000005 | ???          |
+-----+
5 rows in set (0.00 sec)
```

```
mysql> set names utf8;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> select * from ntype;
+-----+
| id      | name          |
+-----+
| 0000000001 | yu le xin wen |
| 0000000002 | ti yu xin wen |
| 0000000003 | guo ji xin wen |
| 0000000004 | jun shi xin wen |
| 0000000005 | 其他新闻      |
+-----+
5 rows in set (0.00 sec)
这就是属于可以恢复的。
mysql>
```

Server characterset: latin1 这个是MySQL服务器的字符集编码可以在MySQL的配置文件中国进行设置及修改。

1.首先进入MySQL的配置文件
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf

Ubuntu16下修改MySQL字符集

与Ubuntu14略有不同，主要是几个文件所在位置不同。修改方法如下：

1 修改mysql的配置文件

```
sudo vim /etc/mysql/conf.d/mysql.cnf
```

在[mysql]的下方加入如下语句。（注：这个文件下没有配置，只有【mysql】）

```
1 no-auto-rehash default-character-set=utf8
2
3 /etc/mysql/mysql.conf.d/mysqld.cnf
```

在[mysqld]下加入

```
1 /var/run/mysqld/mysqld.sock port = 3306 character-set-server=utf8 （这里是server，之前有的版本是set）
```

在配置文件中进行上图的操作，在port=3306的下一行插入character-set-server=utf8

2.保存退出之后重启MySQL服务器：

```
service mysql restart
```

3.进入MySQL查看：

```
mysql> use mydb
```

```
Reading table information for completion of table and column names
```

```
You can turn off this feature to get a quicker startup with -A
```

```
Database changed
```

```
mysql> \s
```

```
-----
```

```
mysql Ver 14.14 Distrib 5.7.22, for Linux (x86_64) using EditLine wrapper
```

```
Connection id: 3
```

```
Current database: mydb
```

```
Current user: root@localhost
```

```
SSL: Not in use
```

```
Current pager: stdout
```

```
Using outfile: ''
```

```
Using delimiter: ;
```

```
Server version: 5.7.22-0ubuntu0.16.04.1 (Ubuntu)
```

```
Protocol version: 10
```

```
Connection: Localhost via UNIX socket
```

```
Server character set: utf8
```

```
Db character set: latin1
```

```
Client character set: utf8
```

```
Conn. character set: utf8
```

```
UNIX socket: /var/run/mysqld/mysqld.sock
```

```
Uptime: 1 min 5 sec
```

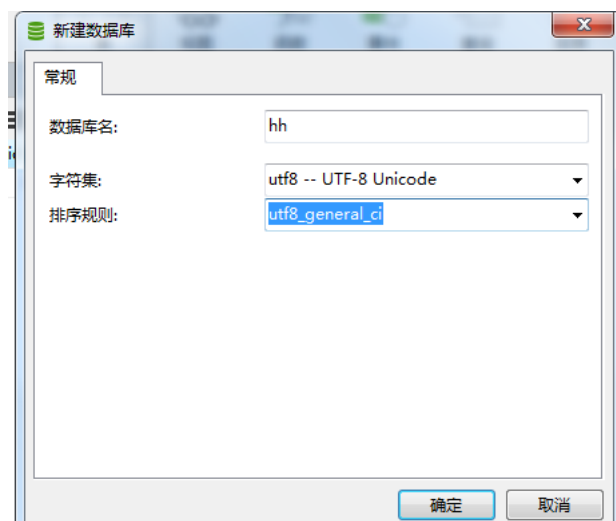
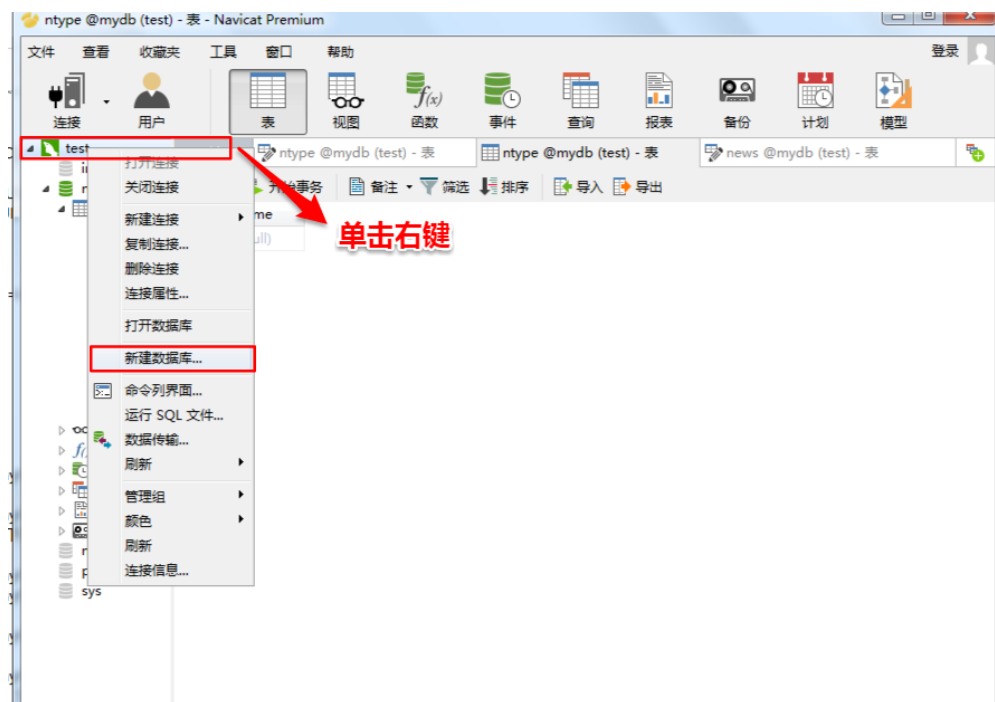
```
Threads: 1 Questions: 21 Slow queries: 0 Opens: 117 Flush tables: 1 Open tables: 36 Queries per second avg: 0.323
```

```
-----
```

```
mysql>
```

```
修改成功。
```

Db character set: latin1 这个是数据库的字符集编码，这就决定了默认创建的数据库的字符集编码

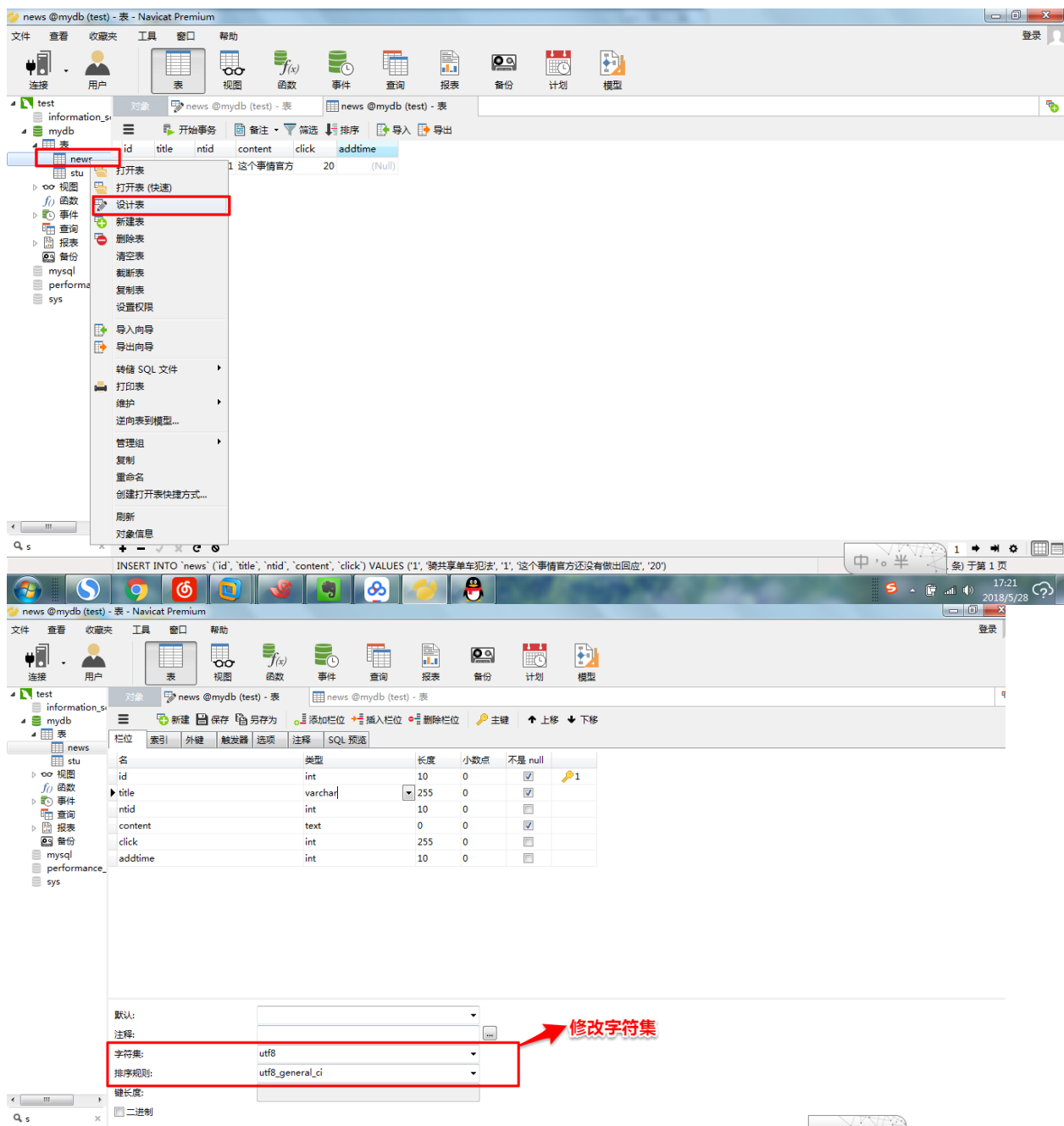


在创建数据库的时候就可以对其进行设置。

如果已经存在的数据库字符集编码不是utf8，那么就要利用命令对其进行修改，修改后，就算MySQL重启也会生效。

```
mysql> set character_set_database=utf8;
```

如果这样修改之后再Navicat中填入数据仍然显示字符集不正确，那么需要再进行一步操作：



```
mysql> \s
```

```
mysql Ver 14.14 Distrib 5.7.22, for Linux (x86_64) using EditLine wrapper
```

```
Connection id: 3
Current database: mydb
Current user: root@localhost
SSL: Not in use
Current pager: stdout
Using outfile: ''
Using delimiter: ;
Server version: 5.7.22-0ubuntu0.16.04.1 (Ubuntu)
Protocol version: 10
Connection: Localhost via UNIX socket
Server characterset: utf8
Db characterset: utf8
Client characterset: utf8
Conn. characterset: utf8
UNIX socket: /var/run/mysqld/mysqld.sock
Uptime: 1 hour 1 min 9 sec
```

```
Threads: 3 Questions: 442 Slow queries: 0 Opens: 149 Flush tables: 1 Open tables: 62 Queries per second avg: 0.120
```

```
Client characterset: utf8
```

Conn. charset: utf8

这两个字符集编码可以通过命令对其进行改变

```
mysql> set names latin1;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> \s
```

```
-----
mysql Ver 14.14 Distrib 5.7.22, for Linux (x86_64) using EditLine wrapper
```

```
Connection id:      3
Current database:   mydb
Current user:       root@localhost
SSL:                Not in use
Current pager:      stdout
Using outfile:      ''
Using delimiter:    ;
Server version:     5.7.22-0ubuntu0.16.04.1 (Ubuntu)
Protocol version:   10
Connection:         Localhost via UNIX socket
Server charset:     latin1
Db charset:         latin1
Client charset:     latin1
Conn. charset:      latin1
UNIX socket:        /var/run/mysqld/mysqld.sock
Uptime:             54 min 3 sec
```

```
Threads: 4 Questions: 93 Slow queries: 0 Opens: 121 Flush tables: 1 Open tables: 40 Queries per second avg: 0.028
```

```
-----
mysql> set names utf8
-> ;
Query OK, 0 rows affected (0.02 sec)
```

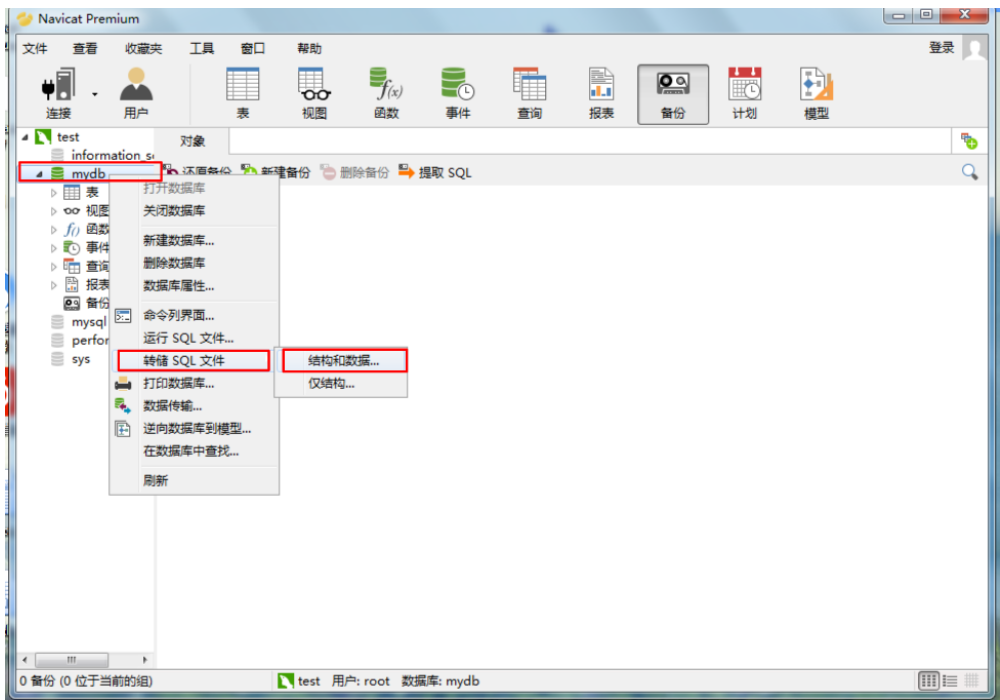
```
mysql> \s
```

```
-----
mysql Ver 14.14 Distrib 5.7.22, for Linux (x86_64) using EditLine wrapper
```

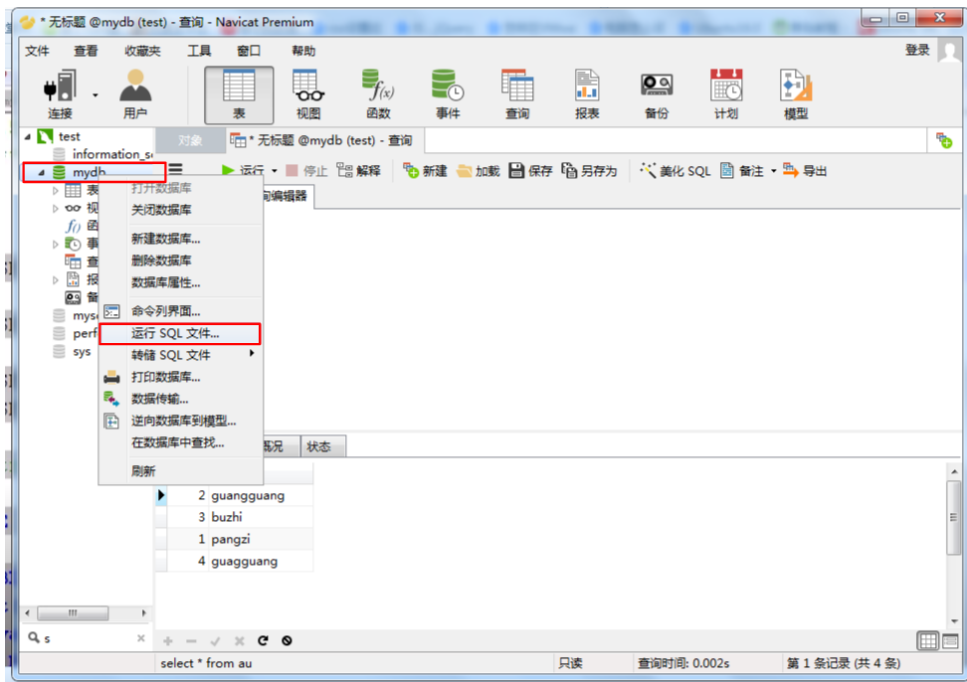
```
Connection id:      3
Current database:   mydb
Current user:       root@localhost
SSL:                Not in use
Current pager:      stdout
Using outfile:      ''
Using delimiter:    ;
Server version:     5.7.22-0ubuntu0.16.04.1 (Ubuntu)
Protocol version:   10
Connection:         Localhost via UNIX socket
Server charset:     latin1
Db charset:         latin1
Client charset:     utf8
Conn. charset:      utf8
UNIX socket:        /var/run/mysqld/mysqld.sock
Uptime:             55 min 7 sec
```

```
Threads: 4 Questions: 97 Slow queries: 0 Opens: 121 Flush tables: 1 Open tables: 40 Queries per second avg: 0.029
```

```
-----
mysql>
```



这样可以对数据进行备份。



这样可以恢复数据。

mysql> select ntid,count(*) from news group by ntid;

```
+-----+-----+
| ntid | count(*) |
+-----+-----+
| 1 | 1 |
| 3 | 1 |
| 4 | 4 |
+-----+-----+
```

3 rows in set (0.14 sec)

mysql> select * from news;

```
+-----+-----+-----+-----+-----+-----+
| id | title | ntid | content | click | addtime |
+-----+-----+-----+-----+-----+-----+
| 1 | 骑共享单车犯法 | 1 | 这个事情官方还没有做出回应 | 20 | NULL |
| 2 | 武警特战排爆手 | 4 | 这个这个事情官方还没有做出回应 | 32 | NULL |
| 3 | 国外武装直升机型号发展与作战能力分析 | 4 | 这个这个事情官方还没有做出回应 | 234 | NULL |
| 4 | 俄战机过去一周在边境拦截外国侦查机11次 | 4 | 这个事情官方还没有做出回应 | 21 | NULL |
| 5 | 中国空军战机进行远洋训练 | 4 | 事情官方还没有做出回应 | 128 | NULL |
| 6 | 中国驻韩使馆举行中韩建交25周年纪念招待会 | 3 | 事情官方还没有做出回应 | 34 | NULL |
+-----+-----+-----+-----+-----+-----+
```

6 rows in set (0.00 sec)

```
mysql> select ntid,count(*) from news group by ntid order by count(*) desc;
```

ntid	count(*)
4	4
1	1
3	1

3 rows in set (0.06 sec)

```
mysql> select news.ntid,count(*),ntype.name from news,ntype where news.ntid=ntype.e.id group by news.ntid order by count(*) desc;
```

ntid	count(*)	name
4	4	军事新闻
1	1	娱乐新闻
3	1	国际新闻

3 rows in set (0.24 sec)

mysql>

MySQL的高级操作：

特殊查询及表的操作

sql内置函数，事务处理和触发器

1. MySQL的表复制

复制表结构

```
mysql> create table 目标表名 like 原表名;
```

复制表数据

```
mysql> insert into 目标表名 select * from 原表名;
```

2. 数据表的索引

创建索引

```
CREATE INDEX index_name ON table_name (column_list)
```

```
CREATE UNIQUE INDEX index_name ON table_name (column_list)
```

删除索引

```
DROP INDEX index_name ON talbe_name
```

3. mysql视图

创建视图:

```
mysql> create view v_t1 as select * from t1 where id>4 and id<11;
```

Query OK, 0 rows affected (0.00 sec)

view视图的帮助信息:

```
mysql> ? view
```

ALTER VIEW

CREATE VIEW

DROP VIEW

查看视图:

```
mysql> show tables;
```

删除视图v_t1:

```
mysql> drop view v_t1;
```

4. MySQL的内置函数

字符串处理函数

*concat(s1,s2,...Sn) 连接s1,s2..Sn为一个字符串

insert(str,x,y,instr)将字符串str从第xx位置开始，y字符串的子字符串替换为字符串str

lower(str)将所有的字符串变为小写

upper(str)将所有的字符串变为大写

left(str,x)返回字符串中最左边的x个字符

right(str,y)返回字符串中最右边的x个字符

lpad(str,n,pad)用字符串pad对str最左边进行填充，直到长度为n个字符串长度

rpadd(str,n,pad)用字符串pad对str最右边进行填充，直到长度为n个字符串长度

trim(str) 去掉左右两边的空格

ltrim(str) 去掉字符串str左侧的空格

rtrim(str) 去掉字符串str右侧的空格

repeat(str,x) 返回字符串str重复x次

replace(str,a,b)将字符串的a替换成b
strcmp(s1,s2) 比较字符串s1和s2
substring(s,x,y)返回字符串指定的长度
*length(str) 返回值为字符串str 的长度

数值函数

*abs(x) 返回x的绝对值
ceil(x) 返回大于x的最小整数值
floor(x) 返回小于x的最大整数值
mod(x,y) 返回x/y的取余结果
rand() 返回0~1之间的随机数
*round(x,y)返回参数x的四舍五入的有y位小数的值
truncate(x,y) 返回x截断为y位小数的结果

日期和时间函数

curdate() 返回当前日期,按照' YYYY-MM-DD' 格式
curtime() 返回当前时间,当前时间以'HH:MM:SS'
*now() 返回当前日期和时间,
*unix_timestamp(date) 返回date时间的unix时间戳
from_unixtime(unix_timestamp[,format]) 返回unix时间的时间
week(date) 返回日期是一年中的第几周
year(date) 返回日期的年份
hour(time) 返回time的小时值
minute(time) 返回日time的分钟值
monthname(date) 返回date的月份
*date_format(date,fmt) 返回按字符串fmt格式化日期date值
date_add(date,INTERVAL,expr type) 返回一个日期或者时间值加上一个时间间隔的时间值
*datediff(expr,expr2) 返回起始时间和结束时间的间隔天数

//统计时间戳647583423距离当前时间相差天数（生日天数（不考虑年份））
mysql> select datediff(date_format(from_unixtime(647583423),"2017-%m-%d %h:%i:%s"),now());

其他常用函数

*database() 返回当前数据库名
version() 返回当前服务器版本
user() 返回当前登陆用户名
inet_aton 返回当前IP地址的数字表示 inet_aton("192.168.80.250");
inet_ntoa(num) 返回当前数字表示的ip inet_ntoa(3232256250);
*password(str) 返回当前str的加密版本
*md5(str) 返回字符串str的md5值

5. MySQL的事务处理

关闭自动提交功能（开启手动事务）

```
mysql> set autocommit=0;
```

从表t1中删除了一条记录

```
mysql> delete from t1 where id=11;
```

此时做一个p1还原点:

```
mysql> savepoint p1;
```

再次从表t1中删除一条记录:

```
mysql> delete from t1 where id=10;
```

再次做一个p2还原点:

```
mysql> savepoint p2;
```

此时恢复到p1还原点，当然后面的p2这些还原点会自动失效:

```
mysql> rollback to p1;
```

退回到最原始的还原点:

```
mysql> rollback;
```

回滚

开启自动事务提交（关闭手动事务）

```
mysql> set autocommit=1;
```

6. MySQL的触发器

格式：1、触发器的定义：

```
CREATE TRIGGER trigger_name trigger_time trigger_event  
ON tbl_name FOR EACH ROW trigger_stmt
```

说明：

trigger_name : 触发器名称
trigger_time:触发时间,可取值:BEFORE或AFTER
trigger_event : 触发事件,可取值:INSERT、UPDATE或DELETE。
tbl_name : 指定在哪个表上
trigger_stmt : 触发处理SQL语句。

示例:

```
mysql> delimiter $$
mysql> create trigger del_stu before delete on stu for each row
-> begin
-> insert into stu_bak values(old.id,old.name,old.sex,old.age,old.addtime);
-> end;
-> $$
Query OK, 0 rows affected (0.05 sec)

mysql> delimiter ;
```

7. mysql日志

日志:

数据库的日志必须开启着才可以恢复和找回数据。

sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf

这个是MySQL的配置文件位置。

查看是会不会计入到日志的。

show status like "com_%"

show status

主数据库主要进行增删改操作,从数据库主要进行查看操作,主要是通过bin-log日志进行跟踪数据的。

cd bin

ls

有一个mysqlbinlog 可以运行日志,可以实现数据的完整性恢复

开启日志: 在mysql配置文件中开启: log-bin=mysql-bin

查看bin-log日志:

mysql>show binary logs;

查看最后一个bin-log日志:

mysql>show master status;

此时就会多一个最新的bin-log日志

mysql>flush logs;

查看最后一个bin日志.

mysql>show master status;

mysql>reset master;

清空所有的bin-log日志

执行查看bin-log日志

备份数据:

mysqldump -uroot -pwei test -l -F '/tmp/test.sql'

其中: -F即flush logs,可以重新生成新的日志文件,当然包括log-bin日志

// Linux关闭MySQL的命令

\$mysql_dir/bin/mysqladmin -uroot -p shutdown

// linux启动MySQL的命令

\$mysql_dir/bin/mysqld_safe &

8、有关慢查询操作:

开户和设置慢查询时间:

vi /etc/my.cnf

log_slow_queries=slow.log

long_query_time=5

查看设置后是否生效

mysql> show variables like "%quer%";

慢查询次数:

mysql> show global status like "%quer%";

9 数据库的恢复

1. 首先恢复最后一次的备份完整数据

```
[root@localhost mnt]# mysql -u root -p mydemo < mydemo_2017-7-26.sql
```

Enter password:

2. 查看bin-log日志

```
[root@localhost data]# mysqlbinlog --no-defaults mysql-bin.000009;
```

查找到恢复的节点

3. 执行bin-log日志文件，恢复最后一块的增量数据。

```
[root@localhost data]# mysqlbinlog --no-defaults --stop-position="802" mysql-bin.000009 | mysql -u root -p123456 mydemo;
```

例子：

```
mysql> select * from stu;
```

```
+-----+
| id | name   | age | sex | classid |
+-----+
| 1 | zhangsan | 20 | w | 1 |
| 2 | lisi    | 25 | m | 2 |
| 3 | wangwu  | 22 | w | 5 |
| 4 | zhaoliu | 21 | m | 4 |
| 5 | uu01    | 27 | w | 1 |
| 6 | uu02    | 25 | m | 2 |
| 7 | uu03    | 28 | w | 2 |
| 8 | uu05    | 22 | m | 4 |
| 9 | xiaoli   | 29 | w | 2 |
|10 | xiaozhang | 19 | w | 1 |
|11 | xiaoyan  | 22 | m | 2 |
|12 | xiaoxin  | 28 | w | 4 |
|13 | wangwen  | 27 | w | 2 |
|14 | zhangle  | 29 | m | 5 |
+-----+
14 rows in set (0.00 sec)
```

```
mysql> create table stu2 like stu;
```

Query OK, 0 rows affected (0.10 sec)

```
mysql> desc stu;
```

```
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
| id    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name  | varchar(16)      | NO   | UNI | NULL    |                |
| age   | tinyint(3) unsigned | YES  |     | NULL    |                |
| sex   | enum('w','m')    | NO   |     | w       |                |
| classid | int(10) unsigned | YES  |     | NULL    |                |
+-----+
5 rows in set (0.00 sec)
```

```
mysql> desc stu2;
```

```
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
| id    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name  | varchar(16)      | NO   | UNI | NULL    |                |
| age   | tinyint(3) unsigned | YES  |     | NULL    |                |
| sex   | enum('w','m')    | NO   |     | w       |                |
| classid | int(10) unsigned | YES  |     | NULL    |                |
+-----+
5 rows in set (0.00 sec)
```

```
mysql> select * from stu2;
```

Empty set (0.01 sec)

```
mysql> insert into stu2 select * from stu limit 5;
```

Query OK, 5 rows affected (0.05 sec)

Records: 5 Duplicates: 0 Warnings: 0

```
mysql> select * from stu2;
```

```
+-----+
| id | name   | age | sex | classid |
+-----+
| 1 | zhangsan | 20 | w | 1 |
| 2 | lisi    | 25 | m | 2 |
| 3 | wangwu  | 22 | w | 5 |
| 4 | zhaoliu | 21 | m | 4 |
| 5 | uu01    | 27 | w | 1 |
+-----+
5 rows in set (0.00 sec)
```

```
mysql> desc stu;
```

```
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
```



```
+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(16) | NO | UNI | NULL | |
| age | tinyint(3) unsigned | YES | | NULL | |
| sex | enum('w','m') | NO | | w | |
| classid | int(10) unsigned | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

5 rows in set (0.00 sec)

```
mysql> create index index_age on stu(age);
Query OK, 0 rows affected (0.09 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> desc stu;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(16) | NO | UNI | NULL | |
| age | tinyint(3) unsigned | YES | MUL | NULL | |
| sex | enum('w','m') | NO | | w | |
| classid | int(10) unsigned | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

5 rows in set (0.01 sec)

```
mysql> drop index index_age on stu;
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> desc stu;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(16) | NO | UNI | NULL | |
| age | tinyint(3) unsigned | YES | | NULL | |
| sex | enum('w','m') | NO | | w | |
| classid | int(10) unsigned | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

5 rows in set (0.01 sec)

```
mysql> alter table stu add index index_age(age);
Query OK, 0 rows affected (0.09 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> desc stu;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(16) | NO | UNI | NULL | |
| age | tinyint(3) unsigned | YES | MUL | NULL | |
| sex | enum('w','m') | NO | | w | |
| classid | int(10) unsigned | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

5 rows in set (0.00 sec)

```
mysql> drop index index_age on stu;
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql>
mysql> select * from stu order by age desc limit 3;
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 14 | zhangle | 29 | m | 5 |
| 9 | xiaoli | 29 | w | 2 |
| 12 | xiaoxin | 28 | w | 4 |
+-----+-----+-----+-----+
```

3 rows in set (0.00 sec)

```
mysql> create view vstu as select * from stu order by age desc limit 3;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> select * from vstu;
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 14 | zhangle | 29 | m | 5 |
| 9 | xiaoli | 29 | w | 2 |
| 12 | xiaoxin | 28 | w | 4 |
+-----+-----+-----+-----+
```

3 rows in set (0.00 sec)

```
mysql> select * from vstu where sex="w";
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 9 | xiaoli | 29 | w | 2 |
| 12 | xiaoxin | 28 | w | 4 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> select * from stu;
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | w | 1 |
| 2 | lisi | 25 | m | 2 |
| 3 | wangwu | 22 | w | 5 |
| 4 | zhaoliu | 21 | m | 4 |
| 5 | uu01 | 27 | w | 1 |
| 6 | uu02 | 25 | m | 2 |
| 7 | uu03 | 28 | w | 2 |
| 8 | uu05 | 22 | m | 4 |
| 9 | xiaoli | 29 | w | 2 |
| 10 | xiaozhang | 19 | w | 1 |
| 11 | xiaoyan | 22 | m | 2 |
| 12 | xiaoxin | 28 | w | 4 |
| 13 | wangwen | 27 | w | 2 |
| 14 | zhangle | 29 | m | 5 |
+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

```
mysql> update stu set age=39 where id=10;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
mysql> select * from vstu;
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 10 | xiaozhang | 39 | w | 1 |
| 9 | xiaoli | 29 | w | 2 |
| 14 | zhangle | 29 | m | 5 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| classes |
| d1 |
| d2 |
| d3 |
| dd |
| news |
| ntype |
| stu |
| stu2 |
| type |
| uu |
| vstu |
+-----+
12 rows in set (0.00 sec)
```

```
mysql> drop table vstu;
ERROR 1051 (42S02): Unknown table 'vstu'
mysql> drop view vstu;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
```

事务：

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> delete from stu where classid=0;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> delete from stu where id>10;
Query OK, 4 rows affected (0.07 sec)
```

```
mysql> select * from stu;
+-----+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+-----+
| 1 | zhangsan | 20 | w | 1 |
| 2 | lisi | 25 | m | 2 |
| 3 | wangwu | 22 | w | 5 |
```

```

| 4 | zhaoliu | 21 | m | 4 |
| 5 | uu01 | 27 | w | 1 |
| 6 | uu02 | 25 | m | 2 |
| 7 | uu03 | 28 | w | 2 |
| 8 | uu05 | 22 | m | 4 |
| 9 | xiaoli | 29 | w | 2 |
+-----+-----+-----+
9 rows in set (0.00 sec)

```

```

mysql> update stu sex='m';
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '='m' at line 1
mysql> update stu set sex='m';
Query OK, 5 rows affected (0.00 sec)
Rows matched: 9 Changed: 5 Warnings: 0

```

```

mysql> select * from stu;
+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+
| 1 | zhangsan | 20 | m | 1 |
| 2 | lisi | 25 | m | 2 |
| 3 | wangwu | 22 | m | 5 |
| 4 | zhaoliu | 21 | m | 4 |
| 5 | uu01 | 27 | m | 1 |
| 6 | uu02 | 25 | m | 2 |
| 7 | uu03 | 28 | m | 2 |
| 8 | uu05 | 22 | m | 4 |
| 9 | xiaoli | 29 | m | 2 |
+-----+-----+-----+
9 rows in set (0.00 sec)

```

```

mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

```

```

mysql> select * from stu;
+-----+-----+-----+
| id | name | age | sex | classid |
+-----+-----+-----+
| 1 | zhangsan | 20 | w | 1 |
| 2 | lisi | 25 | m | 2 |
| 3 | wangwu | 22 | w | 5 |
| 4 | zhaoliu | 21 | m | 4 |
| 5 | uu01 | 27 | w | 1 |
| 6 | uu02 | 25 | m | 2 |
| 7 | uu03 | 28 | w | 2 |
| 8 | uu05 | 22 | m | 4 |
| 9 | xiaoli | 29 | w | 2 |
| 10 | xiaozhang | 22 | w | 0 |
| 11 | xiaoyan | 22 | m | 2 |
| 12 | xiaoxin | 28 | w | 4 |
| 13 | wangwen | 27 | w | 2 |
| 14 | zhangle | 29 | m | 5 |
+-----+-----+-----+
14 rows in set (0.00 sec)

```

```

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

```

```

mysql> rollback;

```

--随机函数的应用

```

mysql> select * from type order by rand() desc limit 3;
+-----+-----+-----+
| id | name | pid |
+-----+-----+-----+
| 9 | 食品 | 0 |
| 8 | 童装 | 1 |
| 12 | 休闲装 | 1 |
+-----+-----+-----+
3 rows in set (0.52 sec)

```

```

mysql> select * from type order by rand() desc limit 3;
+-----+-----+-----+
| id | name | pid |
+-----+-----+-----+
| 4 | 手机 | 2 |
| 7 | 女装 | 1 |
| 9 | 食品 | 0 |
+-----+-----+-----+
3 rows in set (0.00 sec)

```

```

mysql> select *,rand() from type order by rand() desc limit 3;
+-----+-----+-----+
| id | name | pid | rand() |
+-----+-----+-----+
| 5 | 相机 | 2 | 0.9624774539493746 |
| 2 | 数码 | 0 | 0.9208985110008144 |

```

```
| 11 | 特产 | 9 | 0.7281597077177462 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Python3 MySQL 数据库连接

1. 什么是 PyMySQL ?

PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个库，Python2中则使用mysqldb。
PyMySQL 遵循 Python 数据库 API v2.0 规范，并包含了 pure-Python MySQL 客户端库。

2. PyMySQL安装

PyMySQL下载地址：<https://github.com/PyMySQL/PyMySQL>。

2.1 使用pip命令进行安装：

```
$ pip install PyMySQL
```

2.2 使用 git 命令下载安装包安装(你也可以手动下载)：

```
$ git clone https://github.com/PyMySQL/PyMySQL
$ cd PyMySQL/
$ python3 setup.py install
```

3. 数据库连接

通过如下代码测试数据库连接

```
#!/usr/bin/python3

import pymysql

# 打开数据库连接
db = pymysql.connect("localhost","root","123456","mydb" )

# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()

# 使用 execute() 方法执行 SQL 查询
cursor.execute("SELECT VERSION()")

# 使用 fetchone() 方法获取单条数据.
data = cursor.fetchone()

print ("Database version : %s " % data)

# 关闭数据库连接
db.close()
```

4. 执行数据查询：

```
#!/usr/bin/python3

import pymysql

# 打开数据库连接
db = pymysql.connect("localhost","root","","mydemo" )

# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()

# SQL 查询语句
sql = "select * from stu limit %d" % (3)
#sql = "select * from stu"

try:
    # 执行SQL语句
    cursor.execute(sql)
    # 获取所有记录列表
    results = cursor.fetchall()
    for row in results:
        id = row[0]
        name = row[1]
        sex = row[2]
        age = row[3]
        classid = row[4]
        # 打印结果
        print ("id=%d,name=%s,sex=%s,age=%d,classid=%s" % (id,name,sex,age,classid))
except:
    print ("Error: unable to fetch data")

# 关闭数据库连接
db.close()
```

5. 执行数据添加

```
#!/usr/bin/python3
```

```
import pymysql

# 打开数据库连接
db = pymysql.connect("localhost","root","","mydemo" )

# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()

# SQL 插入语句
sql = "INSERT INTO stu(name,sex,age,classid) values('%s','%c','%d','%s') " % ('uu142','m',22,'lamp180')

try:
    # 执行sql语句
    cursor.execute(sql)
    # 执行sql语句
    db.commit()
    print("ok: %d " % (cursor.rowcount))
except:
    # 发生错误时回滚
    db.rollback()

# 关闭数据库连接
db.close()
```

6. 执行删除操作

```
#!/usr/bin/python3

import pymysql

# 打开数据库连接
db = pymysql.connect("localhost","root","","mydemo" )

# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()

# SQL 删除语句
sql = "delete from stu where id = '%d'" % (13)
try:
    # 执行SQL语句
    cursor.execute(sql)
    # 提交修改
    db.commit()
except:
    # 发生错误时回滚
    db.rollback()

# 关闭数据库连接
db.close()
```

数据库查询操作：

Python查询Mysql使用 fetchone() 方法获取单条数据, 使用fetchall() 方法获取多条数据。
 fetchone(): 该方法获取下一个查询结果集。结果集是一个对象
 fetchall(): 接收全部的返回结果行。
 rowcount: 这是一个只读属性，并返回执行execute()方法后影响的行数。

pip命令

列出已安装的包：

```
$ pip list
$ pip freeze # 查看自己安装的
```

安装软件（安装特定版本的package，通过使用==, >=, <=, <来指定一个版本号）**

```
$ pip install SomePackage
$ pip install 'Markdown<2.0'
$ pip install 'Markdown>2.0,<2.0.3'
```

卸载软件pip uninstall SomePackage

```
$ pip uninstall SomePackage
```

下载所需的软件包：

```
$ pip download SomePackage -d directory
例如下载PyMySQL软件包
$ pip download PyMySQL -d D:/pypackage
```

安装下载好的软件包文件

```
$ pip install 目录/软件包文件名
如安装PyMySQL软件包
$ pip3.6 install D:/pypackage/PyMySQL-0.7.11-py2.py3-none-any.whl
```

MySQL服务器的用户权限管理

=====

```
-- 授权一个用户 ( zhangsan ) 密码123 , 可以对所有的库 , 所有的表做所有操作。
mysql> grant all on *.* to zhangsan@'%' identified by '123';
Query OK, 0 rows affected (0.17 sec)
```

```
--刷新生效 , 否则就要重启MySQL服务才可以。
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

```
--浏览当前MySQL用户信息
mysql> select user,host,password from mysql.user;
+-----+-----+-----+
| user | host | password |
+-----+-----+-----+
| root | localhost | *23AE809DDACAF96AF0FD78ED04B6A265E05AA257 |
| root | 127.0.0.1 | |
| | localhost | |
| zhangsan | % | *23AE809DDACAF96AF0FD78ED04B6A265E05AA257 |
| admin | 192.168.112.132 | *23AE809DDACAF96AF0FD78ED04B6A265E05AA257 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
-- 移除一些权限
-- revoke:只删除了用户权限 , 但没有删除这个用户
mysql> revoke insert,delete on *.* from admin@192.168.112.132 identified by'123';
```

```
-- 查看指定用户的权限信息
mysql> show grants for xbb@localhost;
+-----+-----+
| Grants for xbb@localhost |
+-----+-----+
| GRANT USAGE ON *.* TO 'xbb'@'localhost' IDENTIFIED BY PASSWORD '*23AE809DDACAF96AF0FD78ED04B6A265E05AA257' |
+-----+-----+
```

```
--drop user:删除了整个用户及其权限 ( 包括数据字典中的数据 )
mysql> drop user 'xbb'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select user,host from mysql.user;
+-----+-----+
| user | host |
+-----+-----+
| root | 127.0.0.1 |
| debian-sys-maint | localhost |
| root | localhost |
| root | wangxg |
+-----+-----+
4 rows in set (0.00 sec)
```

1个字节代表8个位
int(3)指的是三位数
int(4)指的是四位数

当我们在选择使用int的类型的时候 , 不论是int (3) 还是int (11) , 它在数据库里面存储的都是4个字节的长度 , 在使用int (3) 的时候如果你输入的是10 , 会默认就是说这个3代表的是默认的一个长度 , 当你不足3位时 , 会帮你补全 , 当你超过3位时 , 就没有任何的影响。
() 是宽度显示 , 不是储存范围

double(6,2)表示的是总共6位数字 , 其中2位是小数位

tinyint(4)表示从-128到127
tinyint(3) unsigned
tinyint 型的字段如果设置为UNSIGNED类型 , 只能存储从0到255的整数,不能用来储存负数。
tinyint 型的字段如果不设置UNSIGNED类型,存储-128到127的整数。
1个tinyint型数据只占用一个字节;一个INT型数据占用四个字节。

电脑中 , 最基础的单位是位 (bit) , 只能为0或1 , 所有的数据由多个位的二进制组成。8位=1比特 (*Byte*) =1字节 , 1个字节的数据存量是2的8次方 , **4个字节** 42亿多的 **数字**。更大的就需要更多的存量。C语言中常用的数值类型其实有很多 , 比如最常用的int就是**4个字节** , -2147483648~2147483647; 但是也有其他的就用不到**4个字节**。

很多东西发生过 , 永远被保留!

用二进制类型储存视频、声音、图片
微博是采用varchar进存储 帖子什么的用text进行存储

数字比字符串快
一般自增都带着主键 主键可以没有自增

事务是共同进退 , 表示的是张三把钱转给李四 , 需要两个sql语句去完成 , 如果一个sql语句成功了 , 另一个语句没有完成 , 这样是不行的 , 必须两者都成功或都功一个失败了就不行 , 这样的情况就是事务。
事务是一个操作单元 , 必须同时进行。一般采用innodb格式的表单。

查询是不会计入日志的

编码不统一是导致乱码的最主要原因。

MySQL服务器有编码，建库建表有编码，连接库和表的时候也有编码，编码最好统一设置编码格式：

计算机集群，每个服务器贡献2个G，那加在一起就很多了，这样就可以将数据缓存进去。在需要数据的时候就现在缓存里面找，缓存里面没有就在数据库里面存里面缓存一份，这样速度就会加快了，提高性能。

数据特殊的对应关系：

比如微博上的互相关注的人，粉丝和楼主，全部用户都是同一张表中，因此会出现另一张关联表。另一种特殊情况是比如课程和学生之间，一个学生可以被很多学生选择。这个是多对多的关系，但凡存在多对多的关系时都需要在中间添加一张关联表。

利用索引查找数据是非常方便的，比如要查10000条数据，先从第5000条进行查看判断，再折半再折半后就会找到大致的位置。索引常常出现在有where判断索引相当于是在原表中添加了一系列索引，提高了查询的速度，但是相应的，增删改的速度也会下降。

视图是用在数据需要频繁更新的情况下，比如最新的新闻，销量最大的商品等等情况，一旦表发生改变，视图就会相应地发生变化。

触发器：

事件有修改，添加和删除。事件在修改前后都可以，一般添加是有之后的时间。事件是在删除之前。

这个案例就是监听，当把stu的数据删的时候就挪到stu2表里面。

日志：

数据库的日志必须开启着才可以恢复和找回数据。

```
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

这个是MySQL的配置文件位置。

查看是不会计入到日志的。

```
show status like "com_%"
```

```
show status
```

主数据库主要进行增删改操作，从数据库主要进行查看操作，主要是通过bin-log日志进行跟踪数据的。

```
cd bin
```

```
ls
```

有一个mysqlbinlog 可以运行日志，可以实现数据的完整性恢复

执行bin-log日志文件，恢复最后一块的增量数据。

```
/var/lib/mysql# mysqlbinlog --no-defaults --stop-datetime='2018-05-12 14:11:05' binlog.000002|mysql -u root -p mydb
```

权限的设置：

```
mysql> grant all on *.* to zhangsan@'%' identified by '123';
```

```
mysql> \;
```

```
mysql> grant all on mydb.* to lisi@'%' identified by '123';
```

```
mysql> flush privileges;
```

```
drop user zhangsan@'%';
```

先关掉mysql-server然后再开启。

pymysql

```
root@may-virtual-machine:~# python -V
```

```
Python 2.7.12
```

python -V 查看Python的版本

pip -V 查看pip的版本

pip list 查看用pip下载的软件有哪些

pip freeze 查看用pip下载的软件有哪些

使用pip命令进行安装：

```
$ pip install PyMySQL
```

```
root@may-virtual-machine:/var/lib/python/lesson03# ls
```

```
1.py
```

```
root@may-virtual-machine:/var/lib/python/lesson03# python2 1.py
```

```
root@may-virtual-machine:/var/lib/python/lesson03# python3 1.py
```

```
Traceback (most recent call last):
```

```
File "1.py", line 1, in <module>
```

```
    import pymysql
```

```
ImportError: No module named 'pymysql'
```

```
root@may-virtual-machine:/var/lib/python/lesson03# python2 1.py
root@may-virtual-machine:/var/lib/python/lesson03# python -V
Python 2.7.12
root@may-virtual-machine:/var/lib/python/lesson03# python3 -V
Python 3.5.2
这个部分用Python3无法操作MySQL，不知原因
root@may-virtual-machine:/var/lib/python/lesson03# python3 1.py
Traceback (most recent call last):
  File "1.py", line 1, in <module>
    import pymysql
ImportError: No module named 'pymysql'
```

同样的代码，用Python2可以，现在1.py只写了连接的内容，没有报错表示连接成功

```
root@may-virtual-machine:/var/lib/python/lesson03# python2 1.py
继续对1.py进行编辑
root@may-virtual-machine:/var/lib/python/lesson03# vi 1.py
```

成功执行出结果，pymysql在Python和MySQL之间起到了桥梁的作用

```
root@may-virtual-machine:/var/lib/python/lesson03# python2 1.py
Database info:5.7.22-0ubuntu0.16.04.1-log
```

下面的2py,3.py是在1.py的基础上进行修改，实现更多的功能

```
root@may-virtual-machine:/var/lib/python/lesson03# vi 1.py
root@may-virtual-machine:/var/lib/python/lesson03# cp 1.py 2.py
root@may-virtual-machine:/var/lib/python/lesson03# ls
1.py 2.py
root@may-virtual-machine:/var/lib/python/lesson03# vi 2.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 2.py
File "2.py", line 1
SyntaxError: Non-ASCII character '\xe6' in file 2.py on line 1, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details
root@may-virtual-machine:/var/lib/python/lesson03# vi 2.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 2.py
File "2.py", line 1
SyntaxError: Non-ASCII character '\xe6' in file 2.py on line 1, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details
root@may-virtual-machine:/var/lib/python/lesson03# ls
1.py 2.py
root@may-virtual-machine:/var/lib/python/lesson03# vi 2.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 2.py
File "2.py", line 1
SyntaxError: Non-ASCII character '\xe6' in file 2.py on line 1, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details
root@may-virtual-machine:/var/lib/python/lesson03# vi 2.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 2.py
File "2.py", line 1
SyntaxError: Non-ASCII character '\xe6' in file 2.py on line 1, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details
root@may-virtual-machine:/var/lib/python/lesson03# ls
1.py 2.py
root@may-virtual-machine:/var/lib/python/lesson03# rm -rf 2.py
root@may-virtual-machine:/var/lib/python/lesson03# ls
1.py
root@may-virtual-machine:/var/lib/python/lesson03# cp 1.py 2.py
root@may-virtual-machine:/var/lib/python/lesson03# ls
1.py 2.py
root@may-virtual-machine:/var/lib/python/lesson03# vi 2.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 2.py
File "2.py", line 1
SyntaxError: Non-ASCII character '\xe6' in file 2.py on line 1, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details
root@may-virtual-machine:/var/lib/python/lesson03# python3 2.py
Traceback (most recent call last):
  File "2.py", line 2, in <module>
    import pymysql
ImportError: No module named 'pymysql'
root@may-virtual-machine:/var/lib/python/lesson03# vi 2.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 2.py
Traceback (most recent call last):
  File "2.py", line 21, in <module>
    print("id=%d, name=%s, age=%d, sex=%c, classid=%d" % (id,name,age,sex,classid))
TypeError: %d format: a number is required, not str
root@may-virtual-machine:/var/lib/python/lesson03# vi 2.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 2.py
id=1, name=zsan, age=20, sex=m, classid=python03
id=2, name=lisi, age=22, sex=m, classid=python02
id=3, name=wang, age=25, sex=w, classid=python03
id=5, name=huai, age=21, sex=w, classid=python01
id=6, name=uu07, age=21, sex=w, classid=python03
id=7, name=uu01, age=18, sex=m, classid=python03
id=8, name=uu02, age=19, sex=m, classid=python02
id=9, name=uu03, age=23, sex=m, classid=python02
id=10, name=uu04, age=19, sex=m, classid=python03
id=12, name=uu06, age=25, sex=m, classid=python03
id=13, name=qq01, age=21, sex=m, classid=4
Traceback (most recent call last):
  File "2.py", line 22, in <module>
    print("Database info:%s" % data)
NameError: name 'data' is not defined
root@may-virtual-machine:/var/lib/python/lesson03# cp 2.py 3.py
root@may-virtual-machine:/var/lib/python/lesson03# ls
```



```
1.py 2.py 3.py
root@may-virtual-machine:/var/lib/python/lesson03# vi 3.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 3.py
Error
root@may-virtual-machine:/var/lib/python/lesson03# vi 3.py
root@may-virtual-machine:/var/lib/python/lesson03# python2 3.py
ok:1
```

1.py里面的代码

```
2.py里面的代码
#data analysis
import pymysql
```

```
db = pymysql.connect("localhost","root","123456","mydb")
```

```
cursor = db.cursor()
```

```
sql = "select * from stu"
```

```
cursor.execute(sql)
```

```
result = cursor.fetchall()
```

```
for row in result:
    id = row[0]
    name = row[1]
    age = row[2]
    sex = row[3]
    classid = row[4]
    print("id=%d, name=%s, age=%d, sex=%c, classid=%s" % (id,name,age,sex,classid))
"2.py" 25L, 410C
```

```
#data analysis
import pymysql
```

```
db = pymysql.connect("localhost","root","123456","mydb")
```

```
cursor = db.cursor()
```

```
sql = "insert into stu(name,age,sex,classid) values('%s', '%d', '%c', '%s') % ('qq10',22,'w','2')"
```

```
try:
    cursor.execute(sql)
    print("ok:%d" % (cursor.rowcount))
```

```
except:
    print("Error")
```

```
db.close()
```

```
~
"3.py" 20L, 315C
```

安装Samba

ubuntu16.04下安装samba

2017年07月05日 11:12:22

1、安装 samba软件包

```
sudo apt-get install samba
```

```
sudo apt-get install smbclient
```

2、启动或停止samba服务

```
sudo /etc/init.d/samba start
```

```
sudo /etc/init.d/samba stop
```

Ubuntu 16.04安装配置Samba服务

Samba是开源软件，用来让Linux系统与Windows系统的SMB/CIFS网络协定做连接，实现Windows主机与Linux服务器之间的资源共享。Samba服务为两种不同的操作系统架起了一座桥梁，使系统之间能够实现互相通信，为广泛的Linux爱好者提供了极大方便。

安装Samba

使用apt-get安装：

```
$ sudo apt-get install samba samba-common
```

如果你开启了防火墙，关闭：

```
$ sudo systemctl
```

配置Samba

编辑配置文件：

```
$ sudo vim /etc/samba/smb.conf
```

添加Samba共享目录：

```
[homes  
    browseable
```

添加一个用户：

```
$ sudo smbpasswd
```

我这里输入的是root用户，也可以输入其他的存在用户名。

重启samba服务生效：

```
$ sudo systemctl restart
```

测试：在Windows下运行窗口输入\\加上IP，例如：\\192.168.1.199\\root。在弹出的窗口，输入刚刚添加的用户名和密码，就可以访问Linux的文件目录了。

更多信息：<https://www.samba.org/>

注意：

在利用pymysql连接Python和MySQL数据库的时候，也许会连不上

1.首先看Windows下的Navicat能连接上不，如果连接不上，应该对配置文件等作出修改

2.进入mysql数据库,然后执行SELECT user,host,plugin FROM mysql.user;

```
mysql> SELECT user,host,plugin FROM mysql.user;  
+-----+-----+-----+  
| user      | host      | plugin      |  
+-----+-----+-----+  
| root      | localhost | auth_socket |  
| mysql.session | localhost | mysql_native_password |  
| mysql.sys  | localhost | mysql_native_password |  
| debian-sys-maint | localhost | mysql_native_password |  
| root      | %         | mysql_native_password |  
+-----+-----+-----+  
5 rows in set (0.07 sec)
```

从这可以看出 第一项root并不是密码登录

3.UPDATE mysql.user SET authentication_string=PASSWORD('Avalon'), plugin='mysql_native_password' WHERE user='root';

4.然后FLUSH PRIVILEGES;

然后重启mysql

5.进入mysql执行 SELECT user,host,plugin FROM mysql.user;

```
mysql> SELECT user,host,plugin FROM mysql.user;  
+-----+-----+-----+  
| user      | host      | plugin      |  
+-----+-----+-----+  
| root      | localhost | mysql_native_password |  
| mysql.session | localhost | mysql_native_password |  
| mysql.sys  | localhost | mysql_native_password |  
| debian-sys-maint | localhost | mysql_native_password |  
| root      | %         | mysql_native_password |  
+-----+-----+-----+  
5 rows in set (0.00 sec)
```

OK这样就可以了，写好Python文本 执行Python 1.py就可以了（说多了都是泪）

NoSQL简介

一、NoSQL简介

NoSQL(NoSQL = Not Only SQL) , 意即“不仅仅是SQL”。

在现代的计算系统上每天网络上都会产生庞大的数据量。

这些数据有很大一部分是由关系数据库管理系统 (RDBMSs) 来处理。 1970年 E.F.Codd's提出的关系模型的论文 "A relational model of data for large shared data banks" , 这使得数据建模和应用程序编程更加简单。

通过应用实践证明, 关系模型是非常适合于客户服务器编程, 远远超出预期的利益, 今天它是结构化数据存储在网络和商务应用的主导技术。

NoSQL 是一项全新的数据库革命性运动, 早期就有人提出, 发展至2009年趋势越发高涨。NoSQL的拥护者们提倡运用非关系型的数据存储, 相对于铺天盖地的关系型数据库运用, 这一概念无疑是一种全新的思维的注入。

关系型数据库遵循ACID规则

事务在英文中是transaction, 和现实世界中的交易很类似, 它有如下四个特性:

1、A (Atomicity) 原子性

原子性很容易理解, 也就是说事务里的所有操作要么全部做完, 要么都不做, 事务成功的条件是事务里的所有操作都成功, 只要有一个操作失败, 整个事务就失败, 需要回滚。

比如银行转账, 从A账户转100元至B账户, 分为两个步骤: 1) 从A账户取100元; 2) 存入100元至B账户。这两步要么一起完成, 要么一起不完成, 如果只完成第一步, 第二步失败, 钱会莫名其妙少了100元。

2、C (Consistency) 一致性

一致性也比较容易理解, 也就是说数据库要一直处于一致的状态, 事务的运行不会改变数据库原本的一致性约束。

例如现有有完整性约束 $a+b=10$, 如果一个事务改变了a, 那么必须得改变b, 使得事务结束后依然满足 $a+b=10$, 否则事务失败。

3、I (Isolation) 独立性

所谓的独立性是指并发的事务之间不会互相影响, 如果一个事务要访问的数据正在被另外一个事务修改, 只要另外一个事务未提交, 它所访问的数据就不受未提交事务的影响。

比如现有有个交易是从A账户转100元至B账户, 在这个交易还未完成的情况下, 如果此时B查询自己的账户, 是看不到新增加的100元的。

4、D (Durability) 持久性

持久性是指一旦事务提交后, 它所做的修改将会永久的保存在数据库上, 即使出现宕机也不会丢失。

什么是NoSQL?

NoSQL, 指的是非关系型的数据库。NoSQL有时也称作Not Only SQL的缩写, 是对不同于传统的关系型数据库的数据库管理系统的统称。

NoSQL用于超大规模数据的存储。(例如谷歌或Facebook每天为他们的用户收集万亿比特的数据)。这些类型的数据存储不需要固定的模式, 无需多余操作就可以横向扩展。

为什么使用NoSQL ?

今天我们可以通过第三方平台(如: Google, Facebook等)可以很容易的访问和抓取数据。用户的个人信息, 社交网络, 地理位置, 用户生成的数据和用户操作日志已经成倍的增加。我们如果要对这些用户数据进行挖掘, 那SQL数据库已经不适合这些应用了, NoSQL数据库的发展也却能很好的处理这些大的数据。

RDBMS vs NoSQL

RDBMS

- 高度组织化结构化数据
- 结构化查询语言 (SQL) (SQL)
- 数据和关系都存储在单独的表中。
- 数据操纵语言, 数据定义语言
- 严格的一致性
- 基础事务

NoSQL

- 代表着不仅仅是SQL
- 没有声明性查询语言
- 没有预定义的模式
 - 键 - 值对存储, 列存储, 文档存储, 图形数据库
- 最终一致性, 而非ACID属性
- 非结构化和不可预知的数据
- CAP定理
- 高性能, 高可用性和可伸缩性

二、Redis

Redis 一个内存数据库, 通过 Key-Value 键值对的方式存储数据。由于 Redis 的数据都存储在内存中, 所以访问速度非常快, 因此 Redis 大量用于缓存系统, 存储热点数据, 可以极大的提高网站的响应速度。

优点

- 支持数据的持久化，通过配置可以将内存中的数据保存在磁盘中，Redis 重启以后再将数据加载到内存中；
- 支持列表，哈希，有序集合等数据结构，极大的扩展了 Redis 用途；
- 原子操作，Redis 的所有操作都是原子性的，这使得基于 Redis 实现分布式锁非常简单；
- 支持发布/订阅功能，数据过期功能；

环境准备

<http://www.runoob.com/redis/redis-install.html>

Ubuntu 下安装

在 Ubuntu 系统安装 Redi 可以使用以下命令：

```
$sudo apt-get update
$sudo apt-get install redis-server
```

启动 Redis

```
$ redis-server
```

查看 redis 是否启动？

```
$ redis-cli
```

以上命令将打开以下终端：

```
redis 127.0.0.1:6379>
```

127.0.0.1 是本机 IP ，6379 是 redis 服务端口。现在我们输入 PING 命令。

```
redis 127.0.0.1:6379> ping
PONG
```

以上说明我们已经成功安装了redis。

三、基本操作

Redis 是 Key-Value 内存数据库，操作是通过各种指令进行的，比如 `SET` 指令可以设置键值对，而 `GET` 指令则获取某一个键的值。不同的数据结构，Redis 有不同的指令，这样指令一共有几十个，下面主要介绍一些常用的指令。

Redis 对 Key 也就是键有各种各样的指令，主要有下面的指令（下面的指令中小写字串都是参数，可以自定义）：

```
>keys * //返回键（key）

>keys list* //返回名以list开头的所有键（key）

>exists list1 //判断键名为list1的是否存在 存在返回1， 不存在返回0

>del list1 //删除一个键（名为list1）

>expire list1 10 //设置键名为list1的过期时间为10秒后

>ttl list1 //查看键名为list1的过期时间，若为-1表示以过期 或 永不过期

>move age 1 //将键名age的转移到1数据库中。

>select 1 //表示进入到1数据库中，默认在0数据库

>persist age //移除age的过期时间

>flushdb:删除所有的数据 清除当前所在库的所有数据

>flushall 清空所有数据
```

Redis 配置

Redis 的配置文件位于 Redis 安装目录下，文件名为 redis.conf。

你可以通过CONFIG命令查看或设置配置项。

```
redis 127.0.0.1:6379> CONFIG GET CONFIG_SETTING_NAME

redis 127.0.0.1:6379> CONFIG GET loglevel
```

```
1) "loglevel"  
2) "notice"
```

Redis 配置参数说明

redis.conf 配置项说明如下：

1. Redis默认不是以守护进程的方式运行，可以通过该配置项修改，使用yes启用守护进程
daemonize no
2. 当Redis以守护进程方式运行时，Redis默认会把pid写入/var/run/redis.pid文件，可以通过pidfile指定
pidfile /var/run/redis.pid
3. 指定Redis监听端口，默认端口为6379，作者在自己的一篇博文中解释了为什么选用6379作为默认端口，
因为6379在手机按键上MERZ对应的号码，而MERZ取自意大利歌女Alessia Merz的名字
port 6379
4. 绑定的主机地址
bind 127.0.0.1
5. 当 客户端闲置多长时间后关闭连接，如果指定为0，表示关闭该功能
timeout 300
6. 指定日志记录级别，Redis总共支持四个级别：debug、verbose、notice、warning，默认为verbose
loglevel verbose
7. 日志记录方式，默认为标准输出，如果配置Redis为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null
logfile stdout
8. 设置数据库的数量，默认数据库为0，可以使用SELECT <dbid>命令在连接上指定数据库id
databases 16
9. 指定在多长时间內，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合
save <seconds> <changes>
Redis默认配置文件中提供了三个条件：
save 900 1
save 300 10
save 60 10000
分别表示900秒（15分钟）内有1个更改，300秒（5分钟）内有10个更改以及60秒内有10000个更改。

10. 指定存储至本地数据库时是否压缩数据，默认为yes，Redis采用LZF压缩，如果为了节省CPU时间，可以关闭该选项，
但会导致数据库文件变的巨大
rdbcompression yes
11. 指定本地数据库文件名，默认值为dump.rdb
dbfilename dump.rdb
12. 指定本地数据库存放目录
dir ./
13. 设置当本机为slav服务时，设置master服务的IP地址及端口，在Redis启动时，它会自动从master进行数据同步
slaveof <masterip> <masterport>
14. 当master服务设置了密码保护时，slav服务连接master的密码
masterauth <master-password>
15. 设置Redis连接密码，如果配置了连接密码，客户端在连接Redis时需要通过AUTH <password>命令提供密码，默认关闭
requirepass foobared
16. 设置同一时间最大客户端连接数，默认无限制，Redis可以同时打开的客户端连接数为Redis进程可以打开的最大文件描述符数，
如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，
Redis会关闭新的连接并向客户端返回max number of clients reached错误信息
maxclients 128
17. 指定Redis最大内存限制，Redis在启动时会把数据加载到内存中，达到最大内存后，Redis会先尝试清除已到期或即将到期的Key，
当此方法处理 后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。
Redis新的vm机制，会把Key存放在内存，Value会存放在swap区
maxmemory <bytes>
18. 指定是否在每次更新操作后进行日志记录，Redis在默认情况下是异步的把数据写入磁盘，如果不开启，
可能会在断电时导致一段时间内的数据丢失。因为 redis本身同步数据文件是按上面save条件来同步的，
所以有的数据会在一段时间内只存在于内存中。默认为no
appendonly no
19. 指定更新日志文件名，默认为appendonly.aof
appendfilename appendonly.aof
20. 指定更新日志条件，共有3个可选值：
no：表示等操作系统进行数据缓存同步到磁盘（快）
always：表示每次更新操作后手动调用fsync()将数据写到磁盘（慢，安全）
everysec：表示每秒同步一次（折衷，默认值）
appendfsync everysec

21. 指定是否启用虚拟内存机制，默认值为no，简单的介绍一下，VM机制将数据分页存放，由Redis将访问量较少的页即冷数据swap到磁盘上，
访问多的页面由磁盘自动换出到内存中（在后面的文章我会仔细分析Redis的VM机制）
vm-enabled no
22. 虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个Redis实例共享
vm-swap-file /tmp/redis.swap
23. 将所有大于vm-max-memory的数据存入虚拟内存，无论vm-max-memory设置多少，所有索引数据都是内存存储的(Redis的索引数据 就是keys)，
也就是说，当vm-max-memory设置为0的时候，其实是所有value都存在于磁盘。默认值为0
vm-max-memory 0
24. Redis swap文件分成了很多的page，一个对象可以保存在多个page上面，但一个page上不能被多个对象共享，
vm-page-size是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page大小最好设置为32或者64bytes；
如果存储很大大对象，则可以使用更大的page，如果不 确定，就使用默认值
vm-page-size 32
25. 设置swap文件中的page数量，由于页表（一种表示页面空闲或使用的bitmap）是在放在内存中的，，在磁盘上每8个pages将消耗1byte的内存。
vm-pages 134217728
26. 设置访问swap文件的线程数，最好不要超过机器的核数，如果设置为0，那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。
默认值为4
vm-max-threads 4
27. 设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启
glueoutputbuf yes
28. 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法
hash-max-zipmap-entries 64
hash-max-zipmap-value 512
29. 指定是否激活重置哈希，默认为开启（后面在介绍Redis的哈希算法时具体介绍）
activerehashing yes
30. 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件
include /path/to/local.conf

四、数据类型

Redis的数据类型

Redis通常被称为数据结构服务器，因为值（value）可以是 字符串(String), 哈希(Hash), 列表(list), 集合(sets) 和 有序集合(sorted sets)等类型。

String（子串类型）

```
set 命令：设置一个键和值，键存在则只覆盖，返回ok
> set 键 值 例如： >set name zhangsan

get 命令：获取一个键的值，返回值
> get 键 例如：>get name

setnx命令：设置一个不存在的键和值（防止覆盖），
> setnx 键 值 若键已存在则返回0表示失败

setex命令：设置一个指定有效期的键和值（单位秒）
> setex 键 [有效期] 值 例如：>setex color 10 red
不写有效期则表示永久有效，等价于set

setrange命令：替换子字符串（替换长度由子串长度决定）
> setrange 键 位置 子字符串
> setrange name 4 aa 将name键对应值的第4个位置开始替换

mset命令：批量设置键和值,成功则返回ok
> mset 键1 值1 键2 值2 键3 值3 ....

msetnx命令：批量设置不存在的键和值,成功则返回ok
> msetnx 键1 值1 键2 值2 键3 值3 ....

getset命令：获取原值，并设置新值

getrange命令：获取指定范围的值
>getrange 键 0 4 //获取指定0到4位置上的值

mget命令： 批量获取值
>mget 键1 键2 键3....

incr命令： 指定键的值做加操作，返回加后的结果。
> 键 例如： >incr kid
incrby命令： 设置某个键加上指定值
> incrby 键 m //其中m可以是正整数或负整数

decr命令： 指定键的值做减操作，返回减后的结果。
> decr 键 例如： >decr kid
decrby命令： 设置某个键减上指定值
> decrby 键 m //其中m可以是正整数或负整数

append命令：给指定key的字符串追加value，返回新字符串的长度
>append 键 追加字符串

strlen求长度 >strlen 键名 //返回对应的值。
```

redis string类型增删改查代码练习：

--查看redis是否启动

may@may-virtual-machine:~\$ redis-cli

--返回键

```
127.0.0.1:6379> keys *
```

(empty list or set)

返回名以list开头的所有键（key）

```
127.0.0.1:6379> keys list*
```

(empty list or set)

--通过**CONFIG**命令查看或设置配置项，配置参数等情况请查看上部分笔记

```
127.0.0.1:6379> CONFIG GET CONFIG_SETTING_NAME
```

(empty list or set)

```
127.0.0.1:6379> CONFIG GET loglevel
```

1) "loglevel"

2) "notice"

--string类型数据操作

--set 命令：设置一个键和值，键存在则只覆盖，返回ok

```
127.0.0.1:6379> set name zhangsan
```

OK

get 命令：获取一个键的值，返回值

```
127.0.0.1:6379> get name
```

"zhangsan"

--setex命令：设置一个指定有效期的键和值（单位秒）

```
127.0.0.1:6379> setex color 10 red
```

OK

--有效期内就会返回值

```
127.0.0.1:6379> get color
```

"red"

--有效期过后就没有返回值了

```
127.0.0.1:6379> get color
```

(nil)

--setex命令：设置一个指定有效期的键和值（单位秒）将name键对应值的第4个位置开始替换，从0开始。

```
127.0.0.1:6379> setrange name 4 aa
```

(integer) 8

```
127.0.0.1:6379> get name
```

"zhanaaan"

```

127.0.0.1:6379> keys *
1) "name"
127.0.0.1:6379> keys n*
1) "name"

--判断键名为list1的是否存在 存在返回1， 不存在返回0
127.0.0.1:6379> exists list1
(integer) 0
del list1 //删除一个键（名为name）
127.0.0.1:6379> del name
(integer) 1
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379> set name zhangsan
OK
127.0.0.1:6379> keys *
1) "name"
--表示进入到1数据库中，默认在0数据库
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> get name
(nil)
127.0.0.1:6379[1]> keys *
(empty list or set)
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> keys *
1) "name"
127.0.0.1:6379> set classid 'py02'
OK
127.0.0.1:6379> get classid
"py02"
127.0.0.1:6379> set id 1
OK
127.0.0.1:6379> get id
"1"
127.0.0.1:6379> keys *
1) "classid"
2) "name"
3) "id"
127.0.0.1:6379> keys i*
1) "id"
--setnx命令：设置一个不存在的键和值（防止覆盖） 若键已存在则返回0表示失败
127.0.0.1:6379> setnx id 2
(integer) 0
127.0.0.1:6379> get id
"1"
--setex 键 [有效时间] 值,设置love键对应的值IloveU，有效时间是6秒
127.0.0.1:6379> setex love 6 IloveU
OK
127.0.0.1:6379> get love
(nil)
127.0.0.1:6379> setex love 6 IloveU
OK
127.0.0.1:6379> get love
"IloveU"
127.0.0.1:6379> get love
"IloveU"
--6秒后love键就没有值了
127.0.0.1:6379> get love
(nil)
127.0.0.1:6379> keys *
1) "classid"
2) "name"
3) "id"

--设置键名为name的过期时间为500秒后,返回1则表示设置成功
127.0.0.1:6379> expire name 500
(integer) 1
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> expire name 10
(integer) 1
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> get name
"zhangsan"
--10秒后就失效了
127.0.0.1:6379> get name
(nil)
127.0.0.1:6379> get name

```

```
(nil)
127.0.0.1:6379> get name
(nil)
127.0.0.1:6379> get name
(nil)
127.0.0.1:6379> get name
(nil)

--查看键名为name的过期时间，若为-2表示以过期
127.0.0.1:6379> ttl name
(integer) -2
127.0.0.1:6379> ttl name
(integer) -2

--移除name的过期时间，但是name已经过期了，则无法移除，返回0
127.0.0.1:6379> persist name
(integer) 0
127.0.0.1:6379> get name
(nil)
127.0.0.1:6379> ttl name
(integer) -2
127.0.0.1:6379> set name lisi
OK
127.0.0.1:6379> get name
"lisi"
127.0.0.1:6379> expire name 40
(integer) 1
127.0.0.1:6379> ttl name
(integer) 33
127.0.0.1:6379> ttl name
(integer) 27
127.0.0.1:6379> ttl name
(integer) 15
127.0.0.1:6379> ttl name
(integer) 4
127.0.0.1:6379> ttl name
(integer) -2
127.0.0.1:6379> ttl name
(integer) -2

--查看键名为id的过期时间，若为-1表示永不过期
127.0.0.1:6379> ttl id
(integer) -1

127.0.0.1:6379> set name lisi
OK

127.0.0.1:6379> expire name 50
(integer) 1
127.0.0.1:6379> ttl name
(integer) 44

--移除name的过期时间，由于时效还没过，所以设置成功返回1
127.0.0.1:6379> persist name
(integer) 1
127.0.0.1:6379> ttl name
(integer) -1
127.0.0.1:6379> get name
"lisi"

--替换子字符串（替换长度由子串长度决定）
将name键对应值的第4个位置开始替换，从0开始数
127.0.0.1:6379> setrange name 4 aa
(integer) 6
127.0.0.1:6379> get name
"lisiaa"
127.0.0.1:6379> set name sa
OK
127.0.0.1:6379> get name
"sa"
127.0.0.1:6379> setrange name 4 yy
(integer) 6

--不够的位数用\x00补齐
127.0.0.1:6379> get name
"sa\x00\x00yy"

--批量设置键和值，成功则返回ok
127.0.0.1:6379> mset uname zhangsan upass 123456 uemail zhangsan@qq.com
OK

-- 批量设置不存在的键和值，成功则返回ok
127.0.0.1:6379> msetnx uname zhangsan haoyou wangwu
(integer) 0
127.0.0.1:6379> keys *
1) "classid"
2) "uemail"
```



```
3) "uname"
4) "upass"
5) "name"
6) "id"
127.0.0.1:6379> get name
"sa\x00\x00yy"

--getset命令：获取原值"sa\x00\x00yy"，并设置新值"wangwu"
127.0.0.1:6379> getset name wangwu
"sa\x00\x00yy"
127.0.0.1:6379> get name
"wangwu"

--/获取指定0到4位置上的值
127.0.0.1:6379> getrange name 0 2
"wan"

--指定键的值做加法操作，返回加后的结果。
127.0.0.1:6379> incr uid
(integer) 1
127.0.0.1:6379> incr uid
(integer) 2
127.0.0.1:6379> incr uid
(integer) 3

--设置某个键加上指定值，就是当前的uid值3加上指定值10
127.0.0.1:6379> incrby uid 10
(integer) 13
127.0.0.1:6379> incr uid
(integer) 14
127.0.0.1:6379> decr uid
(integer) 13

--指定键的值做减减操作，返回减后的结果。就是当前的uid值13减去指定值10
127.0.0.1:6379> decrby uid 10
(integer) 3
127.0.0.1:6379> decr uid
(integer) 2
127.0.0.1:6379> decr uid
(integer) 1
127.0.0.1:6379> decr uid
(integer) 0
127.0.0.1:6379> decr uid
(integer) -1
127.0.0.1:6379> get name
"wangwu"

--给指定key的字符串追加value，返回新字符串值的长度
127.0.0.1:6379> append name ww
(integer) 8
127.0.0.1:6379> get name
"wanguwuw"

--拼写错误
127.0.0.1:6379> srrlen name
(error) ERR unknown command 'srrlen'

--返回对应的值的长度
127.0.0.1:6379> strlen name
(integer) 8
127.0.0.1:6379>
```

Hash类型

Redis hash 是一个string类型的field和value的映射表，hash特别适合于存储对象。

hset命令： 设置一个哈希表的键和值
>hset hash名 键 值
如：>hset user:001 name zhangsan
hget命令： 获取执行哈希名中的键对应值

hsetnx命令： 设置一个哈希表中不存在的键和值
>hsetnx hash名 键 值 //成功返回1，失败返回0
如：>hsetnx user:001 name zhangsan

hmset命令：hmset user:001 username zhangsan age 20 sex 1 批量设置
hmgget user:001 username age sex:批量获取值

>hexists user:001 name //是否存在， 若存在返回1

>hlen user:001 //获取某哈希user001名中键的数量

>hdel user:001 name //删除哈希user:001 中name键

```
>hkeys user:002 //返回哈希名为user:002中的所有键。
>hvals user:002 //返回哈希名为user:002中的所有值。
>hgetall user:002 //返回哈希名为user:002中的所有键和值。
```

hashes类型代码练习

may@may-virtual-machine:~\$ redis-cli

--设置一个哈希表的键和值

hset hash名 键 值

127.0.0.1:6379> hset user:001 username zhangsan

(integer) 1

--获取执行哈希名中的键对应值

127.0.0.1:6379> hget user:001 username

"zhangsan"

--批量设置

127.0.0.1:6379> hmset user:002 username lisi password 123 email qq@qq.com

OK

--hget命令： 获取执行哈希名中的键password对应值

127.0.0.1:6379> hget user:002 password

"123"

127.0.0.1:6379> hget user:002 email

"qq@qq.com"

--返回哈希名为user:002中的所有键。

127.0.0.1:6379> hkeys user:002

1) "username"

2) "password"

3) "email"

--返回哈希名为user:001中的所有键。

127.0.0.1:6379> hkeys user:001

1) "username"

--返回哈希名为user:001中的所有值。

127.0.0.1:6379> hvals user:001

1) "zhangsan"

--返回哈希名为user:002中的所有值。

127.0.0.1:6379> hvals user:002

1) "lisi"

2) "123"

3) "qq@qq.com"

--返回哈希名为user:002中的所有键和值。

127.0.0.1:6379> hgetall user:002

1) "username"

2) "lisi"

3) "password"

4) "123"

5) "email"

6) "qq@qq.com"

--返回哈希名为user:001中的所有键和值。

127.0.0.1:6379> hgetall user:001

1) "username"

2) "zhangsan"

127.0.0.1:6379>

List列表（双向链表结构）

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

list即可以作为“栈”也可以作为“队列”。

操作：

>lpush list1 "world" //在list1头部压入一个字符串

>lpush list1 "hello" // 在list1头部压入一个字符串

>lrange list1 0 -1 //获取list1中内容

0:表示开头 -1表示结尾。

>rpush list2 "world" //在list2尾部压入一个字符串

>rpush list2 "hello" // 在list2尾部压入一个字符串

>lrange list2 0 -1 //获取list2中内容

0:表示开头 -1表示结尾。

```
>linsert list2 before hello there
在key对应list的特定位置前或后添加字符串

>lset list2 1 "four"
修改指定索引位置上的值

>lrem list2 2 "hello" //删除前两个hello值
>lrem list2 -2 "hello" //删除后两个hello值
>lrem list2 0 "hello" //删除所有hello值

>ltrim mylist8 1 3 //删除此范围外的值

>lpop list2 //从list2的头部删除元素，并返回删除元素
>rpop list2 //从list2的尾部删除元素，并返回删除元素
>rpoplpush list1 list2 //将list1的尾部一个元素移出到list2头部。并返回

>lindex list2 1 //返回list2中索引位置上的元素
>llen list2 //返回list2上长度
```

list类型代码练习

放一本书再放一本书再放再放，最后放进去的那本书也是最先拿出来的，构造和析构，构造方法是当使用new关键字实例化对象的时候会自动触发，

析构方法是当脚本结束或者对象要消失的时候，即对象要在内存中销毁的时候，脚本运行完毕会释放内存，自动调用析构方法，也可以输出一句话。
最后创建的是最先销毁的，这就是“栈”的概念。
队列是先进去的先出去，后进去的后出去。

0 -1表示的从头到尾取，先进的先出来。

```
--在list1头部压入一个字符串world
127.0.0.1:6379> lpush list1 'world'
(integer) 1
127.0.0.1:6379> lpush list1 'hello'
(integer) 2
127.0.0.1:6379> lpush list1 '!'
(integer) 3
```

```
--获取list1中内容，结果和输入呈现一个倒影镜像的感觉
127.0.0.1:6379> lrange list1 0 -1
1) "!"
2) "hello"
3) "world"
```

```
--在list2尾部压入一个字符串
127.0.0.1:6379> rpush list2 '!'
(integer) 1
127.0.0.1:6379> rpush list2 'world'
(integer) 2
127.0.0.1:6379> rpush list2 'hello'
(integer) 3
```

```
--获取list1中内容，结果和输入呈现同方向
127.0.0.1:6379> lrange list2 0 -1
1) "!"
2) "world"
3) "hello"
```

```
--在key对应list2的特定位置前添加字符串 there
127.0.0.1:6379> linsert list2 before hello there
(integer) 4
127.0.0.1:6379> lrange list2 0 -1
1) "!"
2) "world"
3) "there"
4) "hello"
```

```
--修改指定索引2位置上的值，从0开始
127.0.0.1:6379> lset list2 2 'two'
OK
127.0.0.1:6379> lrange list2 0 -1
1) "!"
2) "world"
3) "two"
4) "hello"
127.0.0.1:6379> lrange list2 0 -1
1) "!"
2) "world"
3) "two"
4) "hello"
127.0.0.1:6379> lset list2 0 'world'
OK
127.0.0.1:6379> lrange list2 0 -1
1) "world"
2) "world"
3) "two"
```

```
4) "hello"
127.0.0.1:6379> lset list2 1 'one'
Invalid argument(s)
127.0.0.1:6379> lset list2 1 'one'
OK
127.0.0.1:6379> lset list2 1 'one'
OK
```

--在hello前插入there

```
127.0.0.1:6379> linsert list2 before 'hello' there
(integer) 5
127.0.0.1:6379> lrange list2 0 -1
1) "world"
2) "one"
3) "two"
4) "there"
5) "hello"
127.0.0.1:6379> rpush list2 one
(integer) 6
127.0.0.1:6379> rpush list2 one
(integer) 7
127.0.0.1:6379> rpush list2 one
(integer) 8
127.0.0.1:6379> lrange list2 0 -1
1) "world"
2) "one"
3) "two"
4) "there"
5) "hello"
6) "one"
7) "one"
8) "one"
```

--删除前一个hello值

```
127.0.0.1:6379> lrem list2 1 'one'
(integer) 1
127.0.0.1:6379> lrange list2 0 -1
1) "world"
2) "two"
3) "there"
4) "hello"
5) "one"
6) "one"
7) "one"
```

--将list1的尾部一个元素移出到list2头部，对于这种情况也许会觉得很麻烦，很难分清楚哪是头哪是尾。因此只要用 `0 -1` 的方法输出，最后的一个就是尾。第一个就是头，不管是何种插入方式！

```
127.0.0.1:6379> rpoplpush list1 list2
"world"
127.0.0.1:6379> lrange list2 0 -1
1) "world"
2) "world"
3) "two"
4) "there"
5) "hello"
6) "one"
7) "one"
8) "one"
```

```
127.0.0.1:6379> lrange list1 0 -1
1) "!"
2) "hello"
127.0.0.1:6379>
```

--从list1的尾部删除元素，并返回删除元素

```
127.0.0.1:6379> rpop list1
"hello"
127.0.0.1:6379> lrange list1 0 -1
1) "!"
127.0.0.1:6379> rpop list2
"one"
127.0.0.1:6379> lrange list2 0 -1
1) "world"
2) "world"
3) "two"
4) "there"
5) "hello"
6) "one"
7) "one"
127.0.0.1:6379>
```

--从list2的尾部删除元素，并返回删除元素

```
127.0.0.1:6379> lpop list2
"world"
127.0.0.1:6379> lrange list2 0 -1
1) "world"
```

```
2) "two"
3) "there"
4) "hello"
5) "one"
6) "one"
127.0.0.1:6379>
```

Redis 集合(Set)

Redis的Set是string类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

```
>sadd myset "hello" //向myset中添加一个元素
成功返回1，失败(重复)返回0

>smembers myset //获取myset中的所有元素(结果是无序的)

>srem myset "one" //从myset中删除一个one
成功返回1，失败(不存在)返回0

>spop myset //随机返回并删除myset中的一个元素
>randmember myset //随机获取myset中的一个元素，但是不删除

> smove myset1 myset2 zhangsan:将myset1中zhangsan移动到myset2中
> scard myset1 返回myset1的个数
> sismember myset zhangsan:判断张三是否在myset中

>sdiff myset1 myset2 //返回两个集合的差集
以myset1为标准，获取myset2中不存在的。
>sdiffstore dstset myset1 myset2 ...// 返回所有集合的差集，并保存到dstset中

>sinter myset1 myset2 myset3... // 返回N个集合中的交集
>sinterstore dstset myset1 myset2 ... // 返回N个集合的交集并存储到dstset中

> sunion myset1 myset2 ...//返回所有集合的并集
> sunionstore dstset myset1 myset2// 返回所有集合的并集，并存储到dstset中
```

无序集合类型代码练习

--向myset1中添加一个元素two

```
127.0.0.1:6379> sadd myset1 two
(integer) 1
从myset1中删除一个one
127.0.0.1:6379> srem myset1 two
(integer) 1
127.0.0.1:6379> smembers myset1
(empty list or set)
127.0.0.1:6379> sadd myset1 one
(integer) 1
127.0.0.1:6379> sadd myset1 two
(integer) 1
127.0.0.1:6379> sadd myset three
(integer) 1
获取myset中的所有元素(结果是无序的)可是这个每次出来的结果都是一样的，我也不知道为什么
127.0.0.1:6379> smembers myset1
1) "two"
2) "one"
127.0.0.1:6379> smembers myset1
1) "two"
2) "one"
127.0.0.1:6379> smembers myset1
1) "two"
2) "one"
127.0.0.1:6379> smembers myset1
1) "two"
2) "one"
127.0.0.1:6379> smembers myset1
1) "two"
2) "one"
127.0.0.1:6379> sadd myset1 three
(integer) 1
127.0.0.1:6379> smembers myset1
1) "three"
2) "two"
3) "one"
127.0.0.1:6379> smembers myset1
1) "three"
2) "two"
3) "one"
127.0.0.1:6379> smembers myset1
1) "three"
2) "two"
3) "one"
127.0.0.1:6379> smembers myset1
```

```
1) "three"
2) "two"
3) "one"
127.0.0.1:6379> sadd myset1 a
(integer) 1
127.0.0.1:6379> sadd myset1 b
(integer) 1
127.0.0.1:6379> sadd myset1 c
(integer) 1
127.0.0.1:6379> smembers myset1
1) "two"
2) "a"
3) "one"
4) "three"
5) "c"
6) "b"
127.0.0.1:6379> smembers myset1
1) "two"
2) "a"
3) "one"
4) "three"
5) "c"
6) "b"
127.0.0.1:6379> smembers myset1
1) "two"
2) "a"
3) "one"
4) "three"
5) "c"
6) "b"
```

```
--随机获取myset1中的一个元素，但是不删除
127.0.0.1:6379> srandmember myset1
"two"
127.0.0.1:6379> srandmember myset1
"a"
127.0.0.1:6379> srandmember myset1
"three"
```

```
--将myset1中a移动到myset2中
127.0.0.1:6379> smove myset1 myset2 a
(integer) 1
127.0.0.1:6379> smember myset2
(error) ERR unknown command 'smember'
127.0.0.1:6379> smembers myset2
1) "a"
127.0.0.1:6379> smembers myset1
1) "one"
2) "three"
3) "two"
4) "c"
5) "b"
```

```
--判断abc是否在myset1中,返回0表示不存在
127.0.0.1:6379> sismember myset1 abc
(integer) 0
--判断one是否在myset1中,返回1表示存在
127.0.0.1:6379> sismember myset1 one
(integer) 1
```

```
--返回myset1和myset2两个集合的差集
127.0.0.1:6379> sdiff myset1 myset2
1) "three"
2) "two"
3) "c"
4) "one"
5) "b"
--返回myset1和myset2两个集合的差集，并保存到myset3中
127.0.0.1:6379> sdiffstore myset3 myset1 myset2
(integer) 5
127.0.0.1:6379> smembers myset1
1) "one"
2) "three"
3) "two"
4) "c"
5) "b"
127.0.0.1:6379> smembers myset2
1) "a"
127.0.0.1:6379> smembers myset3
1) "three"
2) "two"
3) "c"
4) "one"
5) "b"
--返回所有集合myset1和myset2的并集
127.0.0.1:6379> sunion myset1 myset2
1) "two"
```

```
2) "c"
3) "one"
4) "three"
5) "a"
6) "b"
127.0.0.1:6379> sunion myset1 myset2 myset3
1) "one"
2) "three"
3) "two"
4) "c"
5) "a"
6) "b"

127.0.0.1:6379> smembers myset1
1) "one"
2) "three"
3) "two"
4) "c"
5) "b"
127.0.0.1:6379> smembers myset2
1) "a"
```

```
--返回所有集合的并集，并存储到myset4中
127.0.0.1:6379> sunionstore myset4 myset1 myset2
(integer) 6
127.0.0.1:6379> smembers myset4
1) "two"
2) "c"
3) "one"
4) "three"
5) "a"
6) "b"
127.0.0.1:6379>
```

Redis 有序集合Sset (sorted set)

Redis 有序集合和集合一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

有序集合的成员是唯一的,但分数(score)却可以重复。

```
> zadd zset 1 one 向zset中添加one，排序为1排序
> zrem zset one:删除zset中one

> zincrby zset 2 one:如果one存在，则顺序增加2，如果one不存在，那么就是2

> zrank zset one:返回one在zset中排名(从小到大的排序)
> zrevrank zset one:返回one在zset中排名(从大到小的排序)

> zrange zset 0 -1 withscores:根据score排序（根据score从小到大排序）
> zrevrange zset 0 -1 withscores:根据score排序（根据score从大到小排序）

> zrangebyscore zset 2 3 withscores:返回集合中score在给定区间的元素（包含2和3）
> zcount zset 2 3:返回集合中给定区间的数量
> zcard zset:返回集合中元素的个数
> zscore zset one:返回one元素的score
> zremrangebyrank zset 3 3:删除集合中排名在给定区间的元素
```

有序集合类型代码练习

```
--向zset中添加one，排序为1排序
127.0.0.1:6379> zadd zset1 1 one
(integer) 1
--向zset中添加one，排序为2排序
127.0.0.1:6379> zadd zset1 2 two
(integer) 1
127.0.0.1:6379> zadd zset1 3 a
(integer) 1
--返回one在zset1中排名(从小到大的排序)
127.0.0.1:6379> zrank zset1 one
(integer) 0
127.0.0.1:6379> zrank zset1 two
(integer) 1
127.0.0.1:6379> zrank zset1 a
(integer) 2

--根据score排序（根据score从小到大排序）
127.0.0.1:6379> zrange zset1 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "a"
6) "3"
127.0.0.1:6379> zadd zset1 10 c
(integer) 1
127.0.0.1:6379> zadd zset1 8 d
(integer) 1
```

```
127.0.0.1:6379> zrange zset 0 -1 withscores
(empty list or set)
127.0.0.1:6379> zrange zset1 0 -1 withscores
1) "one"
2) "1"
3) "two"
4) "2"
5) "a"
6) "3"
7) "d"
8) "8"
9) "c"
10) "10"
--如果one存在，则顺序增加2，如果one不存在，那么就是2
127.0.0.1:6379> zincrby zset1 2 one
"3"
```

```
--返回one在zset中排名(从大到小的排序)
127.0.0.1:6379> zrevrank zset1 one
(integer) 2
127.0.0.1:6379> zrevrank zset1 two
(integer) 4
```

```
127.0.0.1:6379> zrank zset1 two
(integer) 0
--返回集合中score在给定区间的元素（包含3和10）
127.0.0.1:6379> zrangebyscore zset1 3 10 withscores
1) "a"
2) "3"
3) "one"
4) "3"
5) "d"
6) "8"
7) "c"
8) "10"
127.0.0.1:6379> zrangebyscore zset1 3 8 withscores
1) "a"
2) "3"
3) "one"
4) "3"
5) "d"
6) "8"
127.0.0.1:6379> zrangebyscore zset1 3 5 withscores
1) "a"
2) "3"
3) "one"
4) "3"
--返回集合中元素的个数
127.0.0.1:6379> zcard zset1
(integer) 5
127.0.0.1:6379>
```

五、redis 高级应用

Redis高级实用特性

Redis 安全

我们可以通过 redis 的配置文件设置密码参数，这样客户端连接到 redis 服务就需要密码验证，这样可以让你的 redis 服务更安全。

查看是否设置了密码验证：

```
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) ""
```

默认情况下 requirepass 参数是空的，这就意味着你无需通过密码验证就可以连接到 redis 服务。

临时修改密码(redis服务重启后不再生效)

```
127.0.0.1:6379> CONFIG set requirepass "runoob"
OK
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) "runoob"
```

配置文件为Redis添加密码

修改配置文件


```
sudo vim /etc/redis/redis.conf

设置: requirepass redis的密码

# requirepass foobared

requirepass abc123

重启服务
sudo service redis restart

登录（两种）
1.# ./redis-cli -a 密码 //连接时指定密码来进行授权
2.redis-cli 进入后发现操作不了时
auth 密码
OK
-----
windows 下设置密码生效

修改配置文件

启动服务 加载配置文件redis-server redis.conf \((加载一次即可\))

启动客户端
```

为redis设置密码：

redis的配置文件位置：
127.0.0.1:6379> info
config_file:/etc/redis/redis.conf

直接修改redis密码，密码需要高两个等级，因为redis太快了。每秒钟11万次读。
127.0.0.1:6379> config set requirepass 123456
OK
127.0.0.1:6379> config set requirepass
(error) NOAUTH Authentication required.
127.0.0.1:6379> exit
root@may-virtual-machine:/# redis-cli
127.0.0.1:6379> get name
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> get name
"wangwuwu"

这种直接改的方法即 config set requirepass是一次性的，当redis服务重启后，密码就不再管用，所以想要永久修改就需要在配置文件中添加，详情看下载的课件 C:\Users\Administrator\Desktop\1\非关系型数据库课件\db\Py-NoSQL-17.10.25\Redis\5 pwd redisgao-ji.html 即上面所述！

Redis主从复制

```
操作步骤：

1. 先将linux虚拟机关闭，之后克隆一个。

2. 启动两个虚拟机：master（主）和slave（从）

3. 在slave（从）中配置一下ip地址

\# ifconfig eth0 192.168.128.229

\# ping 一下看看通不通。

4. 配置从机

进入：配置文件

slaveof 192.168.128.228 6379 //配置连接主机的Redis的ip和端口

masterauth 密码 //配置连接密码

最后启动slave（从）机的Redis服务。
```

其他：可以通过info命令中的role属性查看自己角色是master、slave

Redis事务

```
>multi //开启一个事务
>set age 10 //暂存指令队列
>set age 20
>exec //开始执行（提交事务）
或>discard //清空指令队列（事务回滚）
```

Redis乐观锁

Redis Watch 命令用于监视一个(或多个) key，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断
在事务前对被操作的属性做一个：

```
> watch age
>multi //开启一个事务(在此期间有其他修改，则此处会失败)
>set age 10 //暂存指令队列
>set age 20
>exec //开始执行（提交事务）
或>discard //清空指令队列（事务回滚）
```

Redis持久化机制(通过修改配置文件做设置)

redis的配置文件位置：
127.0.0.1:6379> info
config_file:/etc/redis/redis.conf

1.snapshotting(快照)默认方式
配置 save
save 900 1 #900秒内如果超过1个key被修改，则发起快照保存
save 300 10 #300秒内容如超过10个key被修改，则发起快照保存
save 60 10000
2.Append-only file (aof方式)
配置 appendonly on 改为yes
会在bin目录下产生一个.aof的文件
关于aof的配置
appendonly yes //启用aof 持久化方式

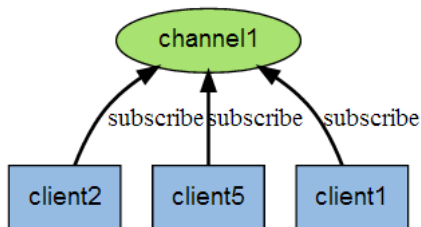
```
# appendfsync always //收到写命令就立即写入磁盘，最慢，但是保证完全的持久化
appendfsync everysec //每秒钟写入磁盘一次，在性能和持久化方面做了很好的折中
# appendfsync no //完全依赖os，性能最好,持久化没保证
```

Redis发布订阅

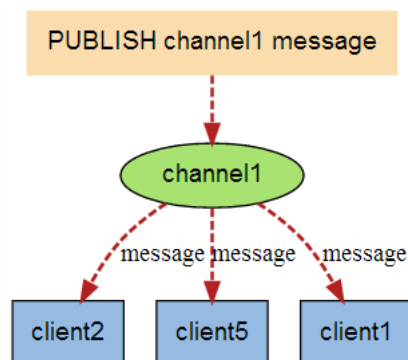
Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 客户端可以订阅任意数量的频道。

下图展示了频道 channel1，以及订阅这个频道的三个客户端—— client2、client5 和 client1 之间的关系：



当有新消息通过 PUBLISH 命令发送给频道 channel1 时，这个消息就会被发送给订阅它的三个客户端：



```
我们创建了订阅频道名为 redisChat:
redis 127.0.0.1:6379> SUBSCRIBE redisChat
```

```
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "redisChat"
3) (integer) 1
```

现在，我们先重新开启个 redis 客户端，然后在同一个频道 redisChat 发布两次消息，订阅者就能接收到消息。

```
redis 127.0.0.1:6379> PUBLISH redisChat "Redis is a great caching technique"
(integer) 1
```

```
redis 127.0.0.1:6379> PUBLISH redisChat "Learn redis by runoob.com"
(integer) 1
```

```
# 订阅者的客户端会显示如下消息
1) "message"
2) "redisChat"
3) "Redis is a great caching technique"
1) "message"
2) "redisChat"
3) "Learn redis by runoob.com"
```

```
-----
序号 命令及描述
1 PSUBSCRIBE pattern [pattern ...] 订阅一个或多个符合给定模式的频道。
2 PUBSUB subcommand [argument [argument ...]] 查看订阅与发布系统状态。
3 PUBLISH channel message 将信息发送到指定的频道。
4 PUNSUBSCRIBE [pattern [pattern ...]] 退订所有给定模式的频道。
5 SUBSCRIBE channel [channel ...] 订阅给定的一个或多个频道的信息。
6 UNSUBSCRIBE [channel [channel ...]] 指退订给定的频道。
```

发布及订阅消息

用两个端口进行操作

第一个端口：

```
127.0.0.1:6379> subscribe tv1 tv2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "tv1"
3) (integer) 1
1) "subscribe"
2) "tv2"
3) (integer) 2
1) "message"
2) "tv1"
3) "hello"
1) "message"
2) "tv2"
3) "world"
```

第二个端口：

```
127.0.0.1:6379> publish tv1 'hello'
(integer) 1
127.0.0.1:6379> publish tv2 'world'
(integer) 1
```

网络配置文件位置：

```
vi /etc/network/interfaces
```

xshell连接不上Ubuntu了，首先要看安装ssh没，22端口打开没，最主要的就是通过Windows下的cmd看看IP地址改变没，还要通过ifconfig看看虚拟机的IP地址发生改变没！（i

安装sublime text3

目前最简单的方法是通过ppa安装，打开终端，输入以下命令：

```
sudo add-apt-repository ppa:webupd8team/sublime-text-3
```

```
sudo apt-get update
```

```
sudo apt-get install sublime-text-installer
```

卸载 sublime text 命令：

```
sudo apt-get remove sublime-text-installer
```

Redis配置虚拟内存

和大多NoSQL数据库一样，Redis同样遵循了Key/Value数据存储模型。在有些情况下，Redis会将Keys/Values保存在内存中以提高数据查询和数据修改的效率，然而这样的做法并非总是很好的选择。鉴于此，我们可以将之进一步优化，即尽量在内存中只保留Keys的数据，这样可以保证数据检索的效率，而Values数据在很少使用的时候则可以被换出到磁盘。

在实际的应用中，大约只有10%的Keys属于相对比较常用的键，这样Redis就可以通过虚存将其余不常用的Keys和Values换出到磁盘上，而一旦这些被换出的Keys或Values需要被读取时，Redis则将其再次读回到主内存中。

在redis配置文件中设置

vm-enabled yes #开启vm功能

vm-swap-file /tmp/redis.swap #交换出来的value保存的文件路径

vm-max-memory 1000000 #redis使用的最大内存上限

vm-page-size 32 #每个页面的大小32字节

vm-pages 134217728 #最多使用多少页面

vm-max-threads 4 #用于执行value对象换入换出的工作线程数量

六、Python操作redis

对于使用 Python 访问 Redis，我们需要先安装`redis-py` 软件包，该包实现了 Python 的 Redis 驱动。通过以下命令建立工作环境，安装软件包：

```
$ sudo pip3 install redis
```

Python 操作 Redis

Python 中访问 Redis 可以通过[redis-py](#)软件包进行。

类似于 PyMongo，也是需要先创建一个 Redis 客户端，如下代码：

```
import redis
r = redis.Redis(host='127.0.0.1', port=6379, decode_responses=True)
r.set('name', 'OK')
print(r.get('name'))
```

1. 千万要谨记，在写代码的时候，要么全用vim写，要么全用sublime写，不能一会这个，一会那个地胡乱用，推荐就用sublime写。
2. 常常会涉及到改变行结束符，在sublime中的行结束符的修改：view->line endings

Python 操作 Redis实验：

做Python操作redis的时候，首先是在自己建立的文件夹下有server.py这个文件，这个文件用 Python3 server.py这个命令运行之后，相当于打开了8000这个端口，因此就可以在浏览器中通过输入127.0.0.1:8000得到访问。server.py的代码如下所示：

```
1 from http.server import HTTPServer, CGIHTTPRequestHandler
2
3 port = 8000
4
5 httpd = HTTPServer(('', port), CGIHTTPRequestHandler)
6 print("Starting simple_httpd on port: " + str(httpd.server_port))
7 httpd.serve_forever()
~
~
```

同时在这个文件夹下再建立一个名为cgi-bin的文件夹（必须要用这个名字），新建一个名为index.py的文件，里面的代码如下所示：

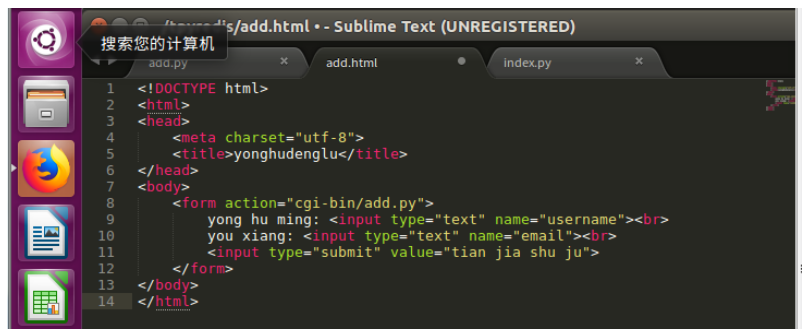
```
1#!/usr/bin/env python3
2#coding:utf8
3import cgi,cgitb,redis
4cgitb.enable()
5
6
7 print("Content-Type: text/html;charset=utf-8")
8 print()
9
10 #接收数据
11 fs=cgi.FieldStorage()
12
13 r=redis.Redis(host='127.0.0.1', port=6379, decode_responses=True)
14 r.set('name', '你好')
15 a=r.get('name')
16 print(a)
17
```



这里特别要注意的是，需要在虚拟机里面的浏览器输入127.0.0.1:8000。

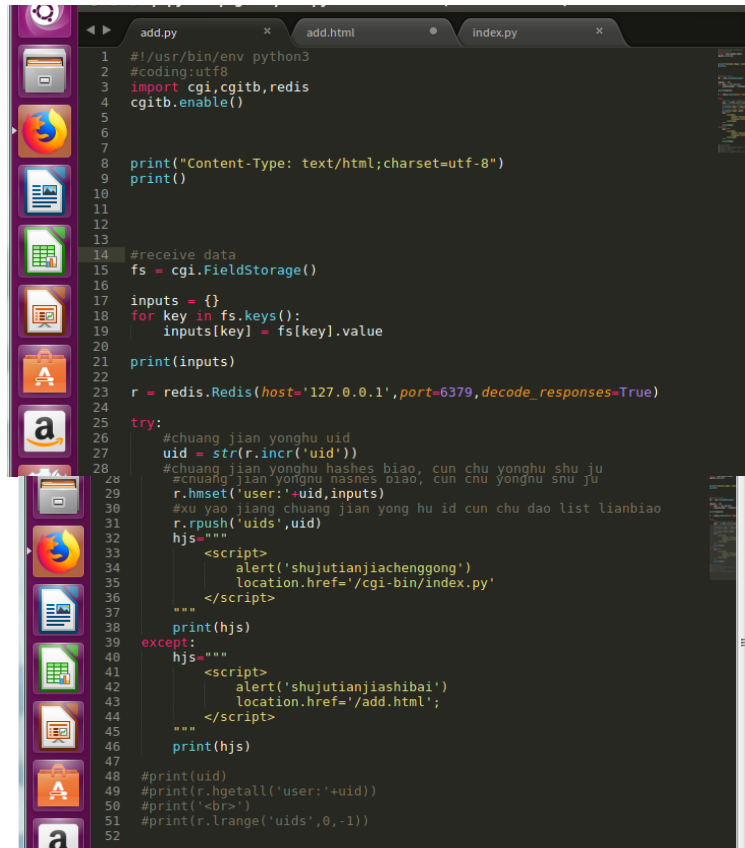
接下来将要更加复杂的内容：

在做一页，需要将用户输入的信息添加入数据库，那么主要涉及到三个文件，add.html,add.py和index.py。（server.py肯定也是需要的，同上，此处不再赘述）。add.html是一个页面给用户提交信息，提交的信息传到add.py,add.py已经连接了redis，所以相当于是说add.html这个页面和redis数据库已经建立起了连接。为了存储数据，在add.py中创建了uid，这样有助于创建hashes表，因为创建的uid是采用自增的方式，不会重复。同时还需要将uid存储到list链表中去（这是为了防止当某个uid对应的数据被删了之后，用for循环查找数据太麻烦了。因此有了list链表之后，当数据被删时，同时也要将list链表中的uid删掉。）这些表都创建好之后，那么数据成功存储到了数据库。之后链接到/cgi-bin/index.py，就可以提取数据了。index.py当然需要连接redis数据库。首先获取uid,再通过uid获取用户的数据。为了让获取的数据以更美观的形式展示出来，就需要用html来布置页面。



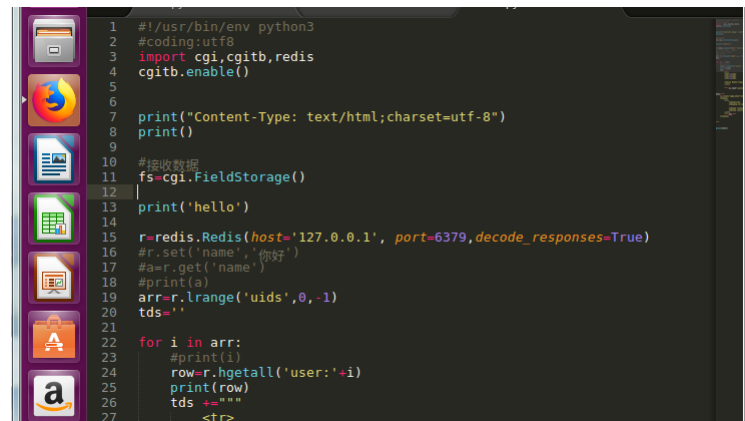
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>yonghudenglou</title>
6 </head>
7 <body>
8   <form action="cgi-bin/add.py">
9     yong hu ming: <input type="text" name="username"><br>
10    you xiang: <input type="text" name="email"><br>
11    <input type="submit" value="tian jia shu ju">
12  </form>
13 </body>
14 </html>
```

add.html代码



```
1 #!/usr/bin/env python3
2 #coding:utf8
3 import cgi,cgitb,redis
4 cgitb.enable()
5
6
7
8 print("Content-Type: text/html;charset=utf-8")
9 print()
10
11
12
13
14 #receive data
15 fs = cgi.FieldStorage()
16
17 inputs = {}
18 for key in fs.keys():
19     inputs[key] = fs[key].value
20
21 print(inputs)
22
23 r = redis.Redis(host='127.0.0.1',port=6379,decode_responses=True)
24
25 try:
26     #chuang jian yonghu uid
27     uid = str(r.incr('uid'))
28     #chuang jian yonghu hashes biao, cun chu yonghu shu ju
29     #cnuang jian yonghu nasnes biao, cun cnu yonghu shu ju
30     r.hmset('user:'+uid,inputs)
31     #xu yao jiang chuang jian yong hu id cun chu dao list lianbiao
32     r.rpush('uids',uid)
33     hjs="""
34         <script>
35             alert('shujutianjiachenggong')
36             location.href='/cgi-bin/index.py'
37         </script>
38     """
39     print(hjs)
40 except:
41     hjs="""
42         <script>
43             alert('shujutianjiashibai')
44             location.href='/add.html';
45         </script>
46     """
47     print(hjs)
48
49 #print(uid)
50 #print(r.hgetall('user:'+uid))
51 #print('<br>')
52 #print(r.lrange('uids',0,-1))
```

add.py代码



```
1 #!/usr/bin/env python3
2 #coding:utf8
3 import cgi,cgitb,redis
4 cgitb.enable()
5
6
7 print("Content-Type: text/html;charset=utf-8")
8 print()
9
10 #接收数据
11 fs=cgi.FieldStorage()
12 |
13 print('hello')
14
15 r=redis.Redis(host='127.0.0.1', port=6379,decode_responses=True)
16 #r.set('name','你好')
17 #a=r.get('name')
18 #print(a)
19 arr=r.lrange('uids',0,-1)
20 tds=''
21
22 for i in arr:
23     #print(i)
24     row=r.hgetall('user:'+i)
25     print(row)
26     tds +=<tr>
```

```
28 <td>%s</td>
29 <td>%s</td>
30 <td>%s</td>
31
32 <td><a href="/cgi-bin/del.py?id='%s'">delete</a>|<a href="/cgi-b
33 </tr>
34
35 """%(i,row['username'],row['email'],i,i)
36
37
38 html="""
39 <a href="/add.html">tian jia shu ju</a>
40 <table>
41 <tr>
42 <td>id</td>
43 <td>yong hu ming</td>
44
45 <td>you xiang</td>
46 <td>cao zuo</td>
47 </tr>
48 """+tds+"""
49 </table>
50 """
51 print(html)
```

index.py代码

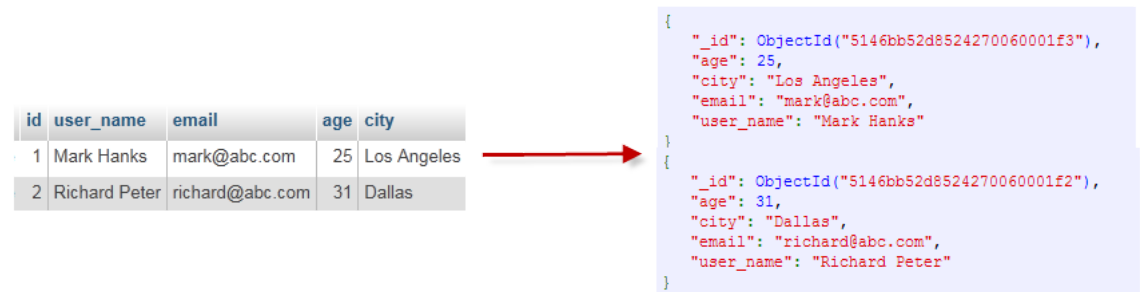
以上是课上老师一起做的实验，更加完整的代码参考课件E:\python系统班课件\1\非关系型数据库课件\db\redis-web

一、MongoDB概念解析

在mongodb中基本的概念是文档、集合、数据库，下面我们挨个介绍。

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

通过下图实例，我们也可以更直观的的了解Mongo中的一些概念：



数据库

一个mongodb中可以建立多个数据库。

MongoDB的默认数据库为"db"，该数据库存储在data目录中。

MongoDB的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中。

"show dbs"命令可以显示所有数据的列表。

```
$ ./mongo
MongoDB shell version: 3.0.6
connecting to: test
> show dbs
local 0.078GB
test 0.078GB
>
```

执行"db"命令可以显示当前数据库对象或集合。

```
$ ./mongo
MongoDB shell version: 3.0.6
connecting to: test
```

```
> db
test
>
```

运行"use"命令，可以连接到一个指定的数据库。

```
> use local
switched to db local
> db
local
>
```

以上实例命令中，"local" 是你要链接的数据库。

在下一个章节我们将详细讲解MongoDB中命令的使用。

数据库也通过名字来标识。数据库名可以是满足以下条件的任意UTF-8字符串。

- 不能是空字符串 ("")。
- 不得含有' ' (空格)、.、\$、/、\和\0 (空字符)。
- 应全部小写。
- 最多64字节。

有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。

- **admin**：从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
- **local**: 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
- **config**：当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相关信息。

文档

文档是一组键值(key-value)对(即BSON)。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。

一个简单的文档例子如下：

```
{ "site": "www.runoob.com", "name": "菜鸟教程" }
```

下表列出了 RDBMS 与 MongoDB 对应的术语：

RDBMS	MongoDB
数据库	数据库
表格	集合
行	文档
列	字段
表联合	嵌入文档
主键	主键 (MongoDB 提供了 key 为 _id)
	数据库服务和客户端
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

需要注意的是：

1. 文档中的键/值对是有序的。
2. 文档中的值不仅可以在双引号里面的字符串，还可以是其他几种数据类型（甚至可以是整个嵌入的文档）。
3. MongoDB区分类型和大小写。
4. MongoDB的文档不能有重复的键。
5. 文档的键是字符串。除了少数例外情况，键可以使用任意UTF-8字符。

文档键命名规范：

- 键不能含有\0 (空字符)。这个字符用来表示键的结尾。
- .和\$有特别的意义，只有在特定环境下才能使用。
- 以下划线"_"开头的键是保留的(不是严格要求的)。

集合

集合就是 MongoDB 文档组，类似于 RDBMS （关系数据库管理系统：Relational Database Management System)中的表格。

集合存在于数据库中，集合没有固定的结构，这意味着你在对集合可以插入不同格式和类型的数据，但通常情况下我们插入集合的数据都会有一定的关联性。

比如，我们可以将以下不同数据结构的文档插入到集合中：

```
{ "site": "www.baidu.com" }
{ "site": "www.google.com", "name": "Google" }
{ "site": "www.runoob.com", "name": "菜鸟教程", "num": 5 }
```

当第一个文档插入时，集合就会被创建。

合法的集合名

- 集合名不能是空字符串""。
- 集合名不能含有\0字符（空字符），这个字符表示集合名的结尾。
- 集合名不能以"system."开头，这是为系统集合保留的前缀。
- 用户创建的集合名字不能含有保留字符。有些驱动程序的确支持在集合名里面包含，这是因为某些系统生成的集合中包含该字符。除非你要访问这种系统创建的集合，否则千万不要在名字里出现\$。

如下实例：

```
db.col.findOne()
```

MongoDB

MongoDB 是一个基于分布式文件存储的数据库。由 C++ 语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。

什么是MongoDB？

MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。

在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为WEB应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

```
{
  name: "sue",           ← field: value
  age: 26,               ← field: value
  status: "A",           ← field: value
  groups: [ "news", "sports" ] ← field: value
}
```

主要特点

- MongoDB的提供了一个面向文档存储，操作起来比较简单和容易。
- 你可以在MongoDB记录中设置任何属性的索引(如：FirstName="Sameer",Address="8 Gandhi Road")来实现更快的排序。
- 你可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。
- 如果负载的增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上这就是所谓的分片。
- Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。
- MongoDb 使用update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段。
- Mongodb中的Map/reduce主要是用来对数据进行批量处理和聚合操作。
- Map和Reduce。Map函数调用emit(key,value)遍历集合中所有的记录，将key与value传给Reduce函数进行处理。
- Map函数和Reduce函数是使用Javascript编写的，并可以通过db.runCommand或mapreduce命令来执行MapReduce操作。
- GridFS是MongoDB中的一个内置功能，可以用于存放大量小文件。
- MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。
- MongoDB支持各种编程语言:RUBY, PYTHON, JAVA, C++, PHP, C#等多种语言。
- MongoDB安装简单。

ubuntu16.04通过apt-get方式安装MongoDB

虽然Ubuntu本身也提供MongoDB安装包，但往往官网的安装包版本更新。

```
~$ apt-cache show mongodb-clients
Package: mongodb-clients
Priority: optional
Section: universe/database
Installed-Size: 160066
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Laszlo Boszormenyi (GCS) <gcs@debian.org>
Architecture: amd64
Source: mongodb
Version: 1:2.6.10-0ubuntu1 # 版本号
```


安装:

1.导入包管理系统使用的公钥

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6
```

2.为MongoDB创建一个列表文件

根据版本创建/etc/apt/sources.list.d/mongodb-org-3.4.list 列表文件

Ubuntu 16.04

```
echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list
```

3.更新本地包数据库

```
sudo apt-get update
```

4.安装最新版本的MongoDB

```
sudo apt-get install -y mongodb-org
```

5.查看配置文件

配置文件mongod.conf所在路径:

/etc/mongod.conf

内容:

```
# mongod.conf

# for documentation of all options, see:
# http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: /var/lib/mongodb #数据库存储路径
  journal:
    enabled: true
  # engine:
  # mmapv1:
  # wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true #以追加的方式写入日志
  path: /var/log/mongodb/mongod.log #日志文件路径

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1 #绑定监听的ip 127.0.0.1只能监听本地的连接，可以改为0.0.0.0

#processManagement:

#security:

#operationProfiling:

#replication:

#sharding:

## Enterprise-Only Options:

#auditLog:

#snmp:
```

6.启动和关闭MongoDB

```
sudo service mongod start # 启动
sudo service mongod stop # 关闭
```

```
hupeng@hupeng-vm:~$ ps aux | grep mongod # 查看守护进程mongod的运行状态
mongodb 18454 9.5 1.5 292152 61952 ? Ssl 12:27 0:00 /usr/bin/mongod --quiet --config /etc/mongod.conf
hupeng 18475 0.0 0.0 15964 936 pts/4 R+ 12:27 0:00 grep --color=auto mongod
```

卸载

1.关闭守护进程mongod

```
sudo service mongod stop
```

2.卸载安装的软件包

```
sudo apt-get purge mongodb-org*
```

3.移除数据库和日志文件（数据库和日志文件的路径取决于/etc/mongod.conf文件中的配置）

```
sudo rm -r /var/log/
mongodb
sudo rm
-r /var/lib/mongodb
```

参考文档: <https://docs.mongodb.com/master/tutorial/install-mongodb-on-ubuntu/>

MongoDB 创建数据库

语法

MongoDB 创建数据库的语法格式如下：

```
use DATABASE_NAME
```

如果数据库不存在，则创建数据库，否则切换到指定数据库。

实例

以下实例我们创建了数据库 runoob:

```
> use runoob
switched to db runoob
> db
runoob
>
```

如果你想查看所有数据库，可以使用show dbs命令：

```
> show dbs
local 0.078GB
test 0.078GB
>
```

可以看到，我们刚创建的数据库 runoob 并不在数据库的列表中，要显示它，我们需要向 runoob 数据库插入一些数据。

```
> db.runoob.insert({"name":"菜鸟教程"})
WriteResult({ "nInserted" : 1 })
> show dbs
local 0.078GB
runoob 0.078GB
test 0.078GB
>
```

MongoDB 中默认的数据库为 test，如果你没有创建新的数据库，集合将存放在 test 数据库中。

MongoDB 删除数据库

语法

MongoDB 删除数据库的语法格式如下：

```
db.dropDatabase()
```

删除当前数据库，默认为 test，你可以使用 db 命令查看当前数据库名。

实例

以下实例我们删除了数据库 runoob。

首先，查看所有数据库：

```
>show dbs
local 0.078GB
runoob 0.078GB
```

```
test 0.078GB
```

接下来我们切换到数据库 runoob :

```
>use runoob
switched to db runoob

>
```

执行删除命令 :

```
>db.dropDatabase()
{ "dropped" : "runoob", "ok" : 1 }
```

最后, 我们再通过 show dbs 命令数据库是否删除成功 :

```
>show dbs
local 0.078GB
test 0.078GB

>
```

删除集合

集合删除语法格式如下 :

```
db.collection.drop()
```

以下实例删除了 runoob 数据库中的集合 site :

代码练习 :

```
root@may-virtual-machine:/home/may# sudo service mongod start
```

```
root@may-virtual-machine:/home/may# mongo
```

```
MongoDB shell version v3.4.15
```

```
connecting to: mongodb://127.0.0.1:27017
```

```
MongoDB server version: 3.4.15
```

```
Server has startup warnings:
```

```
2018-05-31T19:29:53.833+0800 I STORAGE [initandlisten]
```

```
2018-05-31T19:29:53.833+0800 I STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
```

```
2018-05-31T19:29:53.833+0800 I STORAGE [initandlisten] ** See http://dochub.mongodb.org/core/prodnotes-filesystem
```

```
2018-05-31T19:29:55.123+0800 I CONTROL [initandlisten]
```

```
2018-05-31T19:29:55.123+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
```

```
2018-05-31T19:29:55.123+0800 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
```

```
2018-05-31T19:29:55.123+0800 I CONTROL [initandlisten]
```

```
> show dbs
```

```
admin 0.000GB
```

```
local 0.000GB
```

```
py 0.007GB
```

```
test 0.000GB
```

```
> db
```

```
test
```

```
> use py4
```

```
switched to db py4
```

```
> show dbs
```

```
admin 0.000GB
```

```
local 0.000GB
```

```
py 0.007GB
```

```
py4 0.000GB
```

```
test 0.000GB
```

```
> db.user.insert({"name":"zhangsan","age":22,"sex":1})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.user.find()
```

```
{ "_id" : ObjectId("5b0fdd4c9e840a70905bd639"), "name" : "zhangsan", "age" : 22, "sex" : 1 }
```

```
> show dbs
```

```
admin 0.000GB
```

```
local 0.000GB
```

```
py 0.007GB
```

```
py4 0.000GB
```

```
test 0.000GB
```

```
> use py4
```

```
switched to db py4
```

```
> show tables
```

```
user
```

```
> db
```

```
py4
```

```
> use test
```

```
switched to db test
```

```
> db.user.insert({"a":"aa"})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> show collections
```

```
user
```

```
> db.user.find()
```

```

{ "_id" : ObjectId("5b0027ceae18a81beda47ef4"), "class" : "py01", "name" : "zhangsan" }
{ "_id" : ObjectId("5b0027d9ae18a81beda47ef5"), "class" : "py01", "name" : "lisi" }
{ "_id" : ObjectId("5b0027feae18a81beda47ef6"), "class" : "py02", "name" : "wangwu" }
{ "_id" : ObjectId("5b0fdea49e840a70905bd63a"), "a" : "aa" }
>
> show dbs
admin  0.000GB
local  0.000GB
py     0.007GB
py4    0.000GB
test   0.000GB
> db
test
> db.dropDatabase()
{ "dropped" : "test", "ok" : 1 }
> show dbs
admin  0.000GB
local  0.000GB
py     0.007GB
py4    0.000GB
> use py4
switched to db py4
> show tables
user
> db.user.find()
{ "_id" : ObjectId("5b0fdd4c9e840a70905bd639"), "name" : "zhangsan", "age" : 22, "sex" : 1 }
> db.user.drop()
true
> show tables
>

```

总结一下：

这里其实就用到几个常用的命令，记住就可以了，整理如下：

0.sudo mongod start 启动MongoDB，千万不要大意，将mongod写成mongodb或者mongo

1.mongo 输入mongo就可以进入MongoDB

2.show dbs 显示所有的数据库

3.db 显示当前所在的数据库，默认是test

4.use py4 切换到py4数据库

5.db.user.insert({"name":"zhangsan","age":22,"sex":1}) 向当前数据库中的名为user的collection即table添加数据。

6.db.user.find() 显示当前user表中的数据

7.db.dropDatabase() 当切换到某个数据库，输入这个命令就是表示要将这个数据库删掉。

8.db.collection.drop() 指的就是删掉这个名为collection的表。

可以在配置文件里面写的路径里面找到数据存储的位置。

/etc/mongod.conf配置文件位置

```
root@may-virtual-machine:/home/may# vim /etc/mongod.conf
```

```
root@may-virtual-machine:/home/may# cd /var/lib/mongodb
```

```
root@may-virtual-machine:/var/lib/mongodb# ls
```

```
collection-0--7568270994765050244.wt _mdb_catalog.wt
```

```
collection-2--7568270994765050244.wt mongod.lock
```

```
collection-5--7568270994765050244.wt sizeStorer.wt
```

```
diagnostic.data storage.bson
```

```
index-1--7568270994765050244.wt WiredTiger
```

```
index-3--7568270994765050244.wt WiredTigerLAS.wt
```

```
index-4--7568270994765050244.wt WiredTiger.lock
```

```
index-6--7568270994765050244.wt WiredTiger.turtle
```

```
journal WiredTiger.wt
```

二、MongoDB插入文档

本章节中我们将向大家介绍如何将数据插入到MongoDB的集合中。

文档的数据结构和JSON基本一样。

所有存储在集合中的数据都是BSON格式。

BSON是一种类json的一种二进制形式的存储格式,简称Binary JSON。

插入文档

MongoDB 使用 insert() 或 save() 方法向集合中插入文档，语法如下：

```
db.COLLECTION_NAME.insert(document)
```

实例

以下文档可以存储在 MongoDB 的 runoob 数据库 的 col 集合中：

```
>db.col.insert({title: 'MongoDB 教程',
description: 'MongoDB 是一个 Nosql 数据库',
by: '菜鸟教程',
url: 'http://www.runoob.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
})
```

以上实例中 col 是我们的集合名，如果该集合不在该数据库中， MongoDB 会自动创建该集合并插入文档。

查看已插入文档：

```
> db.col.find()
{ "_id" : ObjectId("56064886ade2f21f36b03134"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库",
"by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
>
```

我们也可以将数据定义为一个变量，如下所示：

```
> document=({title: 'MongoDB 教程',
description: 'MongoDB 是一个 Nosql 数据库',
by: '菜鸟教程',
url: 'http://www.runoob.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
});
```

执行后显示结果如下：

```
{
"title" : "MongoDB 教程",
"description" : "MongoDB 是一个 Nosql 数据库",
"by" : "菜鸟教程",
"url" : "http://www.runoob.com",
"tags" : [
"mongodb",
"database",
"NoSQL"
],
"likes" : 100
}
```

执行插入操作：

```
> db.col.insert(document)
WriteResult({ "nInserted" : 1 })
>
```

插入文档你也可以使用 db.col.save(document) 命令。如果不指定 _id 字段 save() 方法类似于 insert() 方法。如果指定 _id 字段，则会更新该 _id 的数据。

代码练习：

```
> show dbs
admin 0.000GB
local 0.000GB
py 0.007GB
> use test
switched to db test
> db.col.insert({title: 'MongoDB 教程',
... description: 'MongoDB 是一个 Nosql 数据库',
... by: '菜鸟教程',
... url: 'http://www.runoob.com',
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
> db.col.find()
{ "_id" : ObjectId("5b0fe8189e840a70905bd63b"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" :
"http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
> document=({title: 'MongoDB 教程',
```

```
... description: 'MongoDB 是一个 Nosql 数据库',
... by: '菜鸟教程',
... url: 'http://www.runoob.com',
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... });
}
{
  "title": "MongoDB 教程",
  "description": "MongoDB 是一个 Nosql 数据库",
  "by": "菜鸟教程",
  "url": "http://www.runoob.com",
  "tags": [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes": 100
}
> db.col.insert(document)
WriteResult({ "nInserted" : 1 })
> db.col.find()
{ "_id" : ObjectId("5b0fe8189e840a70905bd63b"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
{ "_id" : ObjectId("5b0fe8649e840a70905bd63c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
>
```

三、MongoDB 更新文档

MongoDB 使用**update()**和**save()**方法来更新集中的文档。接下来让我们详细来看下两个函数的应用及其区别。

update() 方法

update() 方法用于更新已存在的文档。语法格式如下：

```
db.collection.update(
  <query>,
  <update>,
  {
    upsert: <boolean>,
    multi: <boolean>,
    writeConcern: <document>
  }
)
```

参数说明：

- **query** : update的查询条件，类似sql update查询内where后面的。
- **update** : update的对象和一些更新的操作符（如\$,\$inc...）等，也可以理解为sql update查询内set后面的
- **upsert** : 可选，这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。
- **multi** : 可选，mongodb 默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。
- **writeConcern** :可选，抛出异常的级别。

注意啦注意啦注意啦！！！！

upsert指的是这个筛选中，即update后面的筛选条件在这个数据表中是否会有数据满足这个要求。

比如在这行代码中db.coo.update({"count":{"gt:40}},{"set":{"miss":"Imissyou"}},true,true)

如果所有数据中的“count”都不大于40，那么就会产生一条新数据，这条数据中就会有“miss”：“Imissyou”。

所以千万不要理解成这个表中的数据里面有没有"miss":"Imissyou"这个元素，如果没有就插入，不是这个意思啊喂！！！！

总之upsert的false和true就是问你要不要添加数据。记住这个就好了！

如果复习的时候你又懵了，看下面“更多实例”部分，那里我将把练习的代码复制粘贴在那。如果看了还不明白的话，你就是笨蛋！不接受反驳（摊手）。

实例

我们在集合 col 中插入如下数据：

```
>db.col.insert({
  title: 'MongoDB 教程',
  description: 'MongoDB 是一个 Nosql 数据库',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
})
```

接着我们通过 update() 方法来更新标题(title):

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }) # 输出信息
> db.col.find().pretty()
{
  "_id" : ObjectId("56064f89ade2f21f36b03136"),
  "title" : "MongoDB",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

可以看到标题(title)由原来的 "MongoDB 教程" 更新为了 "MongoDB"。

以上语句只会修改第一条发现的文档，如果你要修改多条相同的文档，则需要设置 multi 参数为 true。

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}},{multi:true})
```

代码练习：

```
> db.col.insert({
... title: 'MongoDB 教程',
... description: 'MongoDB 是一个 Nosql 数据库',
... by: '菜鸟教程',
... url: 'http://www.runoob.com',
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
> db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.col.find().pretty()
{
  "_id" : ObjectId("5b0feaf29e840a70905bd63e"),
  "title" : "MongoDB",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
>

> show tables
col
> db.col.drop()
true
> doc=({'name':"zhangsan","age":40,"sex":0})
{ "name" : "zhangsan", "age" : 40, "sex" : 0 }
> db.co.insert(doc)
WriteResult({ "nInserted" : 1 })
> db.co.find()
{ "_id" : ObjectId("5b0fec8e9e840a70905bd63f"), "name" : "zhangsan", "age" : 40, "sex" : 0 }
> db.co.update({'name':"zhangsan"},{$set:{'name':"lisi","age":35,"sex":1}},{multi:true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.co.find()
{ "_id" : ObjectId("5b0fec8e9e840a70905bd63f"), "name" : "lisi", "age" : 35, "sex" : 1 }
>
```

save() 方法

save() 方法通过传入的文档来替换已有文档。语法格式如下：

```
db.collection.save(
<document>,
{
  writeConcern: <document>
}
)
```

参数说明：

- **document** : 文档数据。
- **writeConcern** :可选，抛出异常的级别。

实例

以下实例中我们替换了 _id 为 56064f89ade2f21f36b03136 的文档数据：

```
>db.col.save({
  "_id" : ObjectId("56064f89ade2f21f36b03136"),
  "title" : "MongoDB",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "Runoob",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "NoSQL"
  ],
  "likes" : 110
})
```

替换成功后，我们可以通过 find() 命令来查看替换后的数据

```
>db.col.find().pretty()
{
  "_id" : ObjectId("56064f89ade2f21f36b03136"),
  "title" : "MongoDB",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "Runoob",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "NoSQL"
  ],
  "likes" : 110
}
>
```

更多实例

只更新找到符合条件的第一条
db.col.update({ "count" : { \$gt : 1 } } , { \$set : { "sex" : "女" } });

全部更新： 找到符合条件的全部更新,如果没有符合条件的不会进行添加数据 如果符合条件的数据中没有对应字段,就会添加字段和数据
db.col.update({ "count" : { \$gt : 20 } } , { \$set : { "sex" : "OK","love":"iloveyou" } },false,true);

如果能找到第一个符合条件的数据 就进行修改 如果没有找到符合条件的就是进行添加数据
db.col.update({ "count" : { \$lt : 2 } } , { \$set : { "test5" : "OK" } },true,false);

找符合条件的所有数据进行修改.,如果没有符合条件的数据 就进行添加
db.col.update({ "count" : { \$lt : 5 } } , { \$set : { "test6" : "OK" } },true,true);

练习代码：

```
> db.coo.insert({"name":"zhangsan","age":40,"sex":"man","count":15})
WriteResult({ "nInserted" : 1 })
> db.coo.insert({"name":"lisi","age":30,"sex":"man","count":20})
WriteResult({ "nInserted" : 1 })
> db.coo.insert({"name":"wangwu","age":20,"sex":"man","count":30})
WriteResult({ "nInserted" : 1 })
> db.coo.find()
{ "_id" : ObjectId("5b0feea39e840a70905bd640"), "name" : "zhangsan", "age" : 40, "sex" : "man", "count" : 15 }
{ "_id" : ObjectId("5b0feebc9e840a70905bd641"), "name" : "lisi", "age" : 30, "sex" : "man", "count" : 20 }
{ "_id" : ObjectId("5b0feede9e840a70905bd642"), "name" : "wangwu", "age" : 20, "sex" : "man", "count" : 30 }
> db.coo.find().pretty()
{
  "_id" : ObjectId("5b0feea39e840a70905bd640"),
  "name" : "zhangsan",
  "age" : 40,
  "sex" : "man",
  "count" : 15
}
{
  "_id" : ObjectId("5b0feebc9e840a70905bd641"),
  "name" : "lisi",
  "age" : 30,
  "sex" : "man",
  "count" : 20
}
{
  "_id" : ObjectId("5b0feede9e840a70905bd642"),
```



```

    "name" : "wangwu",
    "age" : 20,
    "sex" : "man",
    "count" : 30
}
--只更新找到符合条件的第一条
> db.coo.update({"count":{$gt:1}},{$set:{"sex":"nv"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.coo.find()
{ "_id" : ObjectId("5b0feea39e840a70905bd640"), "name" : "zhangsan", "age" : 40, "sex" : "nv", "count" : 15 }
{ "_id" : ObjectId("5b0feebc9e840a70905bd641"), "name" : "lisi", "age" : 30, "sex" : "man", "count" : 20 }
{ "_id" : ObjectId("5b0feede9e840a70905bd642"), "name" : "wangwu", "age" : 20, "sex" : "man", "count" : 30 }

--全部更新: 找到符合条件的全部更新,如果没有符合条件的不会进行添加数据 (这个例子中存在满足条件的数据, 因此正常修改)
> db.coo.update({"count":{$gt:20}},{$set:{"sex":"ok","love":"Iloveyou"}},false,true)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.coo.find()
{ "_id" : ObjectId("5b0feea39e840a70905bd640"), "name" : "zhangsan", "age" : 40, "sex" : "nv", "count" : 15 }
{ "_id" : ObjectId("5b0feebc9e840a70905bd641"), "name" : "lisi", "age" : 30, "sex" : "man", "count" : 20 }
{ "_id" : ObjectId("5b0feede9e840a70905bd642"), "name" : "wangwu", "age" : 20, "sex" : "ok", "count" : 30, "love" : "Iloveyou" }
> db.coo.find().pretty()
{
  "_id" : ObjectId("5b0feea39e840a70905bd640"),
  "name" : "zhangsan",
  "age" : 40,
  "sex" : "nv",
  "count" : 15
}
{
  "_id" : ObjectId("5b0feebc9e840a70905bd641"),
  "name" : "lisi",
  "age" : 30,
  "sex" : "man",
  "count" : 20
}
{
  "_id" : ObjectId("5b0feede9e840a70905bd642"),
  "name" : "wangwu",
  "age" : 20,
  "sex" : "ok",
  "count" : 30,
  "love" : "Iloveyou"
}
--找符合条件的所有数据进行修改,如果没有符合条件的数据 就进行添加
> db.coo.update({"count":{$gt:20}},{$set:{"hate":"Ihateyou"}},true,true)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.coo.find().pretty()
{
  "_id" : ObjectId("5b0feea39e840a70905bd640"),
  "name" : "zhangsan",
  "age" : 40,
  "sex" : "nv",
  "count" : 15
}
{
  "_id" : ObjectId("5b0feebc9e840a70905bd641"),
  "name" : "lisi",
  "age" : 30,
  "sex" : "man",
  "count" : 20
}
{
  "_id" : ObjectId("5b0feede9e840a70905bd642"),
  "name" : "wangwu",
  "age" : 20,
  "sex" : "ok",
  "count" : 30,
  "love" : "Iloveyou",
  "hate" : "Ihateyou"
}
> db.coo.update({"count":{$gt:40}},{$set:{"miss":"Imissyou"}},true,true)
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("5b0ff4fdcd8bce564bd95c0")
})
> db.coo.find().pretty()
{
  "_id" : ObjectId("5b0feea39e840a70905bd640"),
  "name" : "zhangsan",
  "age" : 40,
  "sex" : "nv",
  "count" : 15
}
{
  "_id" : ObjectId("5b0feebc9e840a70905bd641"),
  "name" : "lisi",
  "age" : 30,
  "sex" : "man",
  "count" : 20
}
{
  "_id" : ObjectId("5b0feede9e840a70905bd642"),
  "name" : "wangwu",
  "age" : 20,
  "sex" : "ok",

```

```
    "count" : 30,
    "love" : "Iloveyou",
    "hate" : "Ihateyou"
}
{ "_id" : ObjectId("5b0ff4fdcdca8bce564bd95c0"), "miss" : "Imissyou" }
>
```

总结：

着重记住一个例子就OK！

```
db.coo.update({"count":{$gt:40}},{$set:{"miss":"Imissyou"}},true,true)
```

照搬模板就行

四、MongoDB 删除文档

在前面的几个章节中我们已经学习了MongoDB中如何为集合添加数据和更新数据。在本章节中我们将继续学习MongoDB集合的删除。

MongoDB remove()函数是用来移除集合中的数据。

MongoDB数据更新可以使用update()函数。在执行remove()函数前先执行find()命令来判断执行的条件是否正确，这是一个比较好的习惯。

语法

remove() 方法的基本语法格式如下所示：

```
db.collection.remove(
  <query>,
  <justOne>
)
```

如果你的 MongoDB 是 2.6 版本以后的，语法格式如下：

```
db.collection.remove(
  <query>,
  {
    justOne: <boolean>,
    writeConcern: <document>
  }
)
```

参数说明：

- **query**：(可选) 删除的文档的条件。
- **justOne**：(可选) 如果设为 true 或 1，则只删除一个文档。
- **writeConcern**：(可选) 抛出异常的级别。

实例

以下文档我们执行两次插入操作：

```
>db.col.insert({title: 'MongoDB 教程',
description: 'MongoDB 是一个 Nosql 数据库',
by: '菜鸟教程',
url: 'http://www.runoob.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
})
```

使用 find() 函数查询数据：

```
> db.col.find()
{ "_id" : ObjectId("56066169ade2f21f36b03137"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["mongodb", "database", "NoSQL"], "likes" : 100 }
{ "_id" : ObjectId("5606616dade2f21f36b03138"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["mongodb", "database", "NoSQL"], "likes" : 100 }
```

接下来我们移除 title 为 'MongoDB 教程' 的文档：

```
>db.col.remove({'title':'MongoDB 教程'})
WriteResult({ "nRemoved" : 2 }) # 删除了两条数据
>db.col.find()
..... # 没有数据
```

如果你只想删除第一条找到的记录可以设置 justOne 为 1，如下所示：

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

如果你想删除所有数据，可以使用以下方式（类似常规 SQL 的 truncate 命令）：

```
>db.col.remove({})
>db.col.find()
```

>

代码练习

```
> db.coo.find()
{ "_id" : ObjectId("5b0feea39e840a70905bd640"), "name" : "zhangsan", "age" : 40, "sex" : "nv", "count" : 15 }
{ "_id" : ObjectId("5b0feebc9e840a70905bd641"), "name" : "lisi", "age" : 30, "sex" : "man", "count" : 20 }
{ "_id" : ObjectId("5b0feede9e840a70905bd642"), "name" : "wangwu", "age" : 20, "sex" : "ok", "count" : 30, "love" : "Iloveyou", "hate" : "Ihateyou" }
> db.coo.remove({"name":"zhangsan"})
WriteResult({ "nRemoved" : 1 })
> db.coo.find()
{ "_id" : ObjectId("5b0feebc9e840a70905bd641"), "name" : "lisi", "age" : 30, "sex" : "man", "count" : 20 }
{ "_id" : ObjectId("5b0feede9e840a70905bd642"), "name" : "wangwu", "age" : 20, "sex" : "ok", "count" : 30, "love" : "Iloveyou", "hate" : "Ihateyou" }
> db.coo.insert({"name":"zhangsan","age":20,"sex":1})
WriteResult({ "nInserted" : 1 })
> db.coo.insert({"name":"zhangsan","age":20,"sex":1})
WriteResult({ "nInserted" : 1 })
> db.coo.find()
{ "_id" : ObjectId("5b0feebc9e840a70905bd641"), "name" : "lisi", "age" : 30, "sex" : "man", "count" : 20 }
{ "_id" : ObjectId("5b0feede9e840a70905bd642"), "name" : "wangwu", "age" : 20, "sex" : "ok", "count" : 30, "love" : "Iloveyou", "hate" : "Ihateyou" }
{ "_id" : ObjectId("5b10b1d15426fe9f694e4891"), "name" : "zhangsan", "age" : 20, "sex" : 1 }
{ "_id" : ObjectId("5b10b1d45426fe9f694e4892"), "name" : "zhangsan", "age" : 20, "sex" : 1 }
> db.coo.remove({"name":"zhangsan"},1)
WriteResult({ "nRemoved" : 1 })
> db.coo.find()
{ "_id" : ObjectId("5b0feebc9e840a70905bd641"), "name" : "lisi", "age" : 30, "sex" : "man", "count" : 20 }
{ "_id" : ObjectId("5b0feede9e840a70905bd642"), "name" : "wangwu", "age" : 20, "sex" : "ok", "count" : 30, "love" : "Iloveyou", "hate" : "Ihateyou" }
{ "_id" : ObjectId("5b10b1d45426fe9f694e4892"), "name" : "zhangsan", "age" : 20, "sex" : 1 }
> db.user.remove({})
WriteResult({ "nRemoved" : 0 })
> db.coo.remove({})
WriteResult({ "nRemoved" : 3 })
> db.coo.find({})
>
```

总结：

只要记住下面三个例子就OK

--删除满足条件的全部

```
> db.coo.remove({"name":"zhangsan"})
```

--删除满足条件的第一条

```
> db.coo.remove({"name":"zhangsan"},1)
```

--删除此表中的全部数据

```
> db.user.remove({})
```

五、MongoDB 查询文档

MongoDB 查询文档使用 find() 方法。

find() 方法以非结构化的方式来显示所有文档。

语法

MongoDB 查询数据的语法格式如下：

```
db.collection.find(query, projection)
```

- **query**：可选，使用查询操作符指定查询条件
- **projection**：可选，使用投影操作符指定返回的键。查询时返回文档中所有键值，只需省略该参数即可（默认省略）。

如果你需要以易读的方式来读取数据，可以使用 pretty() 方法，语法格式如下：

```
>db.col.find().pretty()
```

pretty() 方法以格式化的方式来显示所有文档。

实例

以下实例我们查询了集合 col 中的数据：

```
> db.col.find().pretty()
{
  "_id" : ObjectId("56063f17ade2f21f36b03133"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

除了 find() 方法之外，还有一个 findOne() 方法，它只返回一个文档。

MongoDB 与 RDBMS Where 语句比较

如果你熟悉常规的 SQL 数据，通过下表可以更好的理解 MongoDB 的条件语句查询：

操作	格式	范例	RDBMS中的类似语句
等于	{<key>:<value>}	db.col.find({"by":"菜鸟教程"}).pretty()	where by = '菜鸟教程'
小于	{<key>:{\$lt:<value>}}	db.col.find({"likes":{\$lt:50}}).pretty()	where likes < 50
小于或等于	{<key>:{\$lte:<value>}}	db.col.find({"likes":{\$lte:50}}).pretty()	where likes <= 50
大于	{<key>:{\$gt:<value>}}	db.col.find({"likes":{\$gt:50}}).pretty()	where likes > 50
大于或等于	{<key>:{\$gte:<value>}}	db.col.find({"likes":{\$gte:50}}).pretty()	where likes >= 50
不等于	{<key>:{\$ne:<value>}}	db.col.find({"likes":{\$ne:50}}).pretty()	where likes != 50

MongoDB AND 条件

MongoDB 的 find() 方法可以传入多个键(key)，每个键(key)以逗号隔开，及常规 SQL 的 AND 条件。

语法格式如下：

```
>db.col.find({key1:value1, key2:value2}).pretty()
```

实例

以下实例通过by和title键来查询菜鸟教程中MongoDB 教程的数据

```
> db.col.find({"by":"菜鸟教程", "title":"MongoDB 教程"}).pretty()
{
  "_id" : ObjectId("56063f17ade2f21f36b03133"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

以上实例中类似于 WHERE 语句：**WHERE by='菜鸟教程' AND title='MongoDB 教程'**

MongoDB OR 条件

MongoDB OR 条件语句使用了关键字\$or,语法格式如下：

```
>db.col.find(
{
  $or: [
    {key1: value1}, {key2:value2}
  ]
}
).pretty()
```

实例

以下实例中，我们演示了查询键by值为 菜鸟教程 或键title值为MongoDB 教程的文档。

```
>db.col.find({$or:[{"by":"菜鸟教程"}, {"title": "MongoDB 教程"}]}).pretty()
{
  "_id" : ObjectId("56063f17ade2f21f36b03133"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
```

```
"mongodb",
"database",
"NoSQL"
],
"likes" : 100
}
>
```

AND 和 OR 联合使用

以下实例演示了 AND 和 OR 联合使用，类似常规 SQL 语句为：`'where likes>50 AND (by = '菜鸟教程' OR title = 'MongoDB 教程')`

```
>db.col.find({"likes": {$gt:50}, $or: [{"by": "菜鸟教程"}, {"title": "MongoDB 教程"}]}).pretty()
{
  "_id" : ObjectId("56063f17ade2f21f36b03133"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

总结：

--查询user集合中 `uname="张三"` **and** `age=21` 的所有文档

```
> db.user.find({"uname":"zhangsan", "age":21}).pretty()
```

--查询user集合中 `uname=zhangsan` 或者 `uname="张三"` 的数据

```
>db.user.find({$or:[{"uname":"zhangsan"}, {"uname": "张三"}]}).pretty()
```

--查询user集合中 `age=21` `uname=张三`或`uname=zhangsan` (**and** 和**or**一起使用)

```
>db.user.find({"age":21,$or:[{"uname":"zhangsan"}, {"uname": "张三"}]}).pretty()
```

--限制返回的数据字段 **其中键值后面的1表示需要显示，0表示不需要显示。值为0的也可以不需要写出来**

```
> db.user.find({"uname":"zhangsan", "age":21}, {"_id":1, "uname":1}).pretty()
```

```
{ "_id" : ObjectId("59f02a93efa271499313b6a4"), "uname" : "zhangsan" }
```

```
{ "_id" : ObjectId("59f02abbefa271499313b6a5"), "uname" : "zhangsan" }
```

代码练习：

```
> db.coo.find()
```

```
> show dbs
```

```
admin 0.000GB
```

```
local 0.000GB
```

```
py 0.007GB
```

```
test 0.000GB
```

```
> use py
```

```
switched to db py
```

```
> show tables
```

```
co
```

```
col
```

```
coo
```

```
coo.insertdb.col
```

```
user
```

```
> db.user.find()
```

```
{ "_id" : ObjectId("5affb549502273576a543e13"), "name" : "lrh0", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e14"), "name" : "lrh1", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e15"), "name" : "lrh2", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e16"), "name" : "lrh3", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e17"), "name" : "lrh4", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e18"), "name" : "lrh5", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e19"), "name" : "lrh6", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e1a"), "name" : "lrh7", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e1b"), "name" : "lrh8", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e1c"), "name" : "lrh9", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e1d"), "name" : "lrh10", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e1e"), "name" : "lrh11", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e1f"), "name" : "lrh12", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e20"), "name" : "lrh13", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e21"), "name" : "lrh14", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e22"), "name" : "lrh15", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e23"), "name" : "lrh16", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e24"), "name" : "lrh17", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e25"), "name" : "lrh18", "age" : 18 }
```

```
{ "_id" : ObjectId("5affb549502273576a543e26"), "name" : "lrh19", "age" : 18 }
```

```
Type "it" for more
```

```
--只查找第一条数据
> db.user.findOne()
{
  "_id" : ObjectId("5affb549502273576a543e13"),
  "name" : "lrh0",
  "age" : 18
}
--查找满足条件的
> db.user.find({"name":"lrh18"})
{ "_id" : ObjectId("5affb549502273576a543e25"), "name" : "lrh18", "age" : 18 }
> db.user.find({"name":"lrh18","age":18})
{ "_id" : ObjectId("5affb549502273576a543e25"), "name" : "lrh18", "age" : 18 }
--or的使用
> db.user.find($or:[{"name":"lrh18"}, {"name":"lrh19"}])
{ "_id" : ObjectId("5affb549502273576a543e25"), "name" : "lrh18", "age" : 18 }
{ "_id" : ObjectId("5affb549502273576a543e26"), "name" : "lrh19", "age" : 18 }
--and 和or的联合使用
> db.user.find("age":18,$or:[{"name":"lrh18"}, {"name":"lrh19"}])
{ "_id" : ObjectId("5affb549502273576a543e25"), "name" : "lrh18", "age" : 18 }
{ "_id" : ObjectId("5affb549502273576a543e26"), "name" : "lrh19", "age" : 18 }
--限制返回的数据字段
> db.user.find({"name":"lrh5"}, {"_id":0,"name":1})
{ "name" : "lrh5" }
>
```

六、MongoDB查询条件

MongoDB条件操作符

描述

条件操作符用于比较两个表达式并从mongoDB集合中获取数据。

在本章节中，我们将讨论如何在MongoDB中使用条件操作符。

MongoDB中条件操作符有：

```
$gt ----大于 ---- greater than >

$gte ----小于----- gt equal >=

$lt ---大于等于----- less than <

$lte ---小于等于----- lt equal <=

$ne ---不等于----- not equal !=

$eq ----等于---- equal =
```

\$type操作符

在本章节中，我们将继续讨论MongoDB中条件操作符 \$type。

\$type操作符是基于BSON类型来检索集合中匹配的数据类型，并返回结果

MongoDB 中可以使用的类型如下表所示：

类型	数字	备注
Double	1	
String	2	
Object	3	
Array	4	
Binary data	5	
Undefined	6	已废弃。
Object id	7	
Boolean	8	
Date	9	
Null	10	
Regular Expression	11	

JavaScript	13	
Symbol	14	
JavaScript (with scope)	15	
32-bit integer	16	
Timestamp	17	
64-bit integer	18	
Min key	255	Query with-1.
Max key	127	

我们使用的数据库名称为"runoob" 我们的集合名称为"col"，以下为我们插入的数据。

```
>db.col.insert({
  title: 'PHP 教程',
  description: 'PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['php'],
  likes: 200
})
>db.col.insert({title: 'Java 教程',
  description: 'Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['java'],
  likes: 150
})
>db.col.insert({title: 'MongoDB 教程',
  description: 'MongoDB 是一个 Nosql 数据库',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['mongodb'],
  likes: 100
})
```

使用find()命令查看数据：

```
> db.col.find()
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["php"], "likes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["java"], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["mongodb"], "likes" : 100 }
```

\$type 实例

如果想获取 "col" 集合中 title 为 String 的数据，你可以使用以下命令：

```
db.col.find({"title" : {$type : 2}})

{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["php"], "likes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["java"], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["mongodb"], "likes" : 100 }
```

MongoDB Limit() 方法

如果你需要在MongoDB中读取指定数量的数据记录，可以使用MongoDB的Limit方法，limit()方法接受一个数字参数，该参数指定从MongoDB中读取的记录条数。

语法

limit()方法基本语法如下所示：

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

实例

集合 col 中的数据如下：

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["php"], "likes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["java"], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : ["mongodb"], "likes" : 100 }
```

以上实例为显示查询文档中的两条记录：

```
> db.col.find({},{"title":1,_id:0}).limit(2)
{ "title" : "PHP 教程" }
{ "title" : "Java 教程" }
```

```
>
```

注：如果你们没有指定limit()方法中的参数则显示集合中的所有数据。

MongoDB Skip() 方法

我们除了可以使用limit()方法来读取指定数量的数据外，还可以使用skip()方法来跳过指定数量的数据，skip方法同样接受一个数字参数作为跳过的记录条数。

语法

skip() 方法脚本语法格式如下：

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

实例

以上实例只会显示第二条文档数据

```
>db.col.find({},{"title":1,_id:0}).limit(1).skip(1)
{ "title" : "Java 教程" }
>
```

注:skip()方法默认参数为 0 。

MongoDB 排序 sort()方法

在MongoDB中使用使用sort()方法对数据进行排序，sort()方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而-1是用于降序排列。

语法

sort()方法基本语法如下所示：

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

实例

col 集合中的数据如下：

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

以下实例演示了 col 集合中的数据按字段 likes 的降序排列：

```
>db.col.find({},{"title":1,_id:0}).sort({"likes":-1})
{ "title" : "PHP 教程" }
{ "title" : "Java 教程" }
{ "title" : "MongoDB 教程" }
>
```

MongoDB 排序 count()方法

在MongoDB中使用使用count()方法对数据进行统计

类似sql中 select count(*) from mycol

```
> db.mycol.count()
3
```

代码练习：

```
> db.col.find()
{"_id": ObjectId("5aff9b3544572610e69eb688"), "title": "PHP 教程", "description": "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by": "菜鸟教程", "url": "http://www.runoob.com", "tags": [ "php" ], "likes": 200 }
{"_id": ObjectId("5aff9b3f44572610e69eb689"), "title": "Java 教程", "description": "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by": "菜鸟教程", "url": "http://www.runoob.com", "tags": [ "java" ], "likes": 150 }
{"_id": ObjectId("5aff9b4a44572610e69eb68a"), "title": "MongoDB 教程", "description": "MongoDB 是一个 Nosql 数据库", "by": "菜鸟教程", "url": "http://www.runoob.com", "tags": [ "mongodb" ], "likes": 100 }
{"_id": ObjectId("5aff9ba544572610e69eb68b"), "title": 1, "name": "xiaokeai" }
> db.col.remove({})
WriteResult({ "nRemoved": 4 })
```



```

> db.col.find()
> db.col.insert({
... title: 'PHP 教程',
... description: 'PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。',
... by: '菜鸟教程',
... url: 'http://www.runoob.com',
... tags: ['php'],
... likes: 200
... })
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'Java 教程',
... description: 'Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。',
... by: '菜鸟教程',
... url: 'http://www.runoob.com',
... tags: ['java'],
... likes: 150
... })
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'MongoDB 教程',
... description: 'MongoDB 是一个 Nosql 数据库',
... by: '菜鸟教程',
... url: 'http://www.runoob.com',
... tags: ['mongodb'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
> db.col.find()
{"_id" : ObjectId("5b10bf135426fe9f694e4893"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }
{"_id" : ObjectId("5b10bf245426fe9f694e4894"), "title" : "Java 教程", "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }
{"_id" : ObjectId("5b10bf335426fe9f694e4895"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
> db.col.find().pretty()
{
  "_id" : ObjectId("5b10bf135426fe9f694e4893"),
  "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "php"
  ],
  "likes" : 200
}
{
  "_id" : ObjectId("5b10bf245426fe9f694e4894"),
  "title" : "Java 教程",
  "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "java"
  ],
  "likes" : 150
}
{
  "_id" : ObjectId("5b10bf335426fe9f694e4895"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb"
  ],
  "likes" : 100
}
}
--选择title的数据类型为string的数据
> db.col.find({'title':{$type:2}}).pretty()
{
  "_id" : ObjectId("5b10bf245426fe9f694e4894"),
  "title" : "Java 教程",
  "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "java"
  ],
  "likes" : 150
}
{
  "_id" : ObjectId("5b10bf335426fe9f694e4895"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [

```

```

    "mongodb"
  ],
  "likes" : 100
}
{
  "_id" : ObjectId("5b10bf135426fe9f694e4893"),
  "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "php"
  ],
  "likes" : 200
}
--限制读取的记录条数，并且只显示title。id为0不显示，其他没写也不显示
> db.col.find({},{"title":1,"_id":0}).limit(2)
{ "title" : "PHP 教程" }
{ "title" : "Java 教程" }
> db.col.find({},{"title":1,"_id":0}).limit(3)
{ "title" : "PHP 教程" }
{ "title" : "Java 教程" }
{ "title" : "MongoDB 教程" }
--用skip跳过1条数据，显示1条数据
> db.col.find({},{"title":1}).limit(1).skip(1)
{ "_id" : ObjectId("5b10bf245426fe9f694e4894"), "title" : "Java 教程" }
> db.col.find({},{"title":1,"_id":0}).limit(2).skip(1)
{ "title" : "Java 教程" }
{ "title" : "MongoDB 教程" }
> db.col.find({},{"title":1,"_id":0}).skip(1).limit(2)
{ "title" : "Java 教程" }
{ "title" : "MongoDB 教程" }
--sort()方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而-1是用于降序排列。
通过“likes”进行降序排序
> db.col.find({},{"title":1,"_id":0,"likes":1}).sort({"likes":-1})
{ "title" : "PHP 教程", "likes" : 200 }
{ "title" : "Java 教程", "likes" : 150 }
{ "title" : "MongoDB 教程", "likes" : 100 }
--升序方式
> db.col.find({},{"title":1,"_id":0,"likes":1}).sort({"likes":1})
{ "title" : "MongoDB 教程", "likes" : 100 }
{ "title" : "Java 教程", "likes" : 150 }
{ "title" : "PHP 教程", "likes" : 200 }
--使用count()方法对数据进行统计
> db.col.count()
3

```

总结：

```

db.col.find({"title":{$type:2}}).pretty()
db.col.find({},{"title":1,"_id":0}).limit(2)
db.col.find({},{"title":1}).limit(1).skip(1)
db.col.find({},{"title":1,"_id":0,"likes":1}).sort({"likes":-1})
db.col.count()

```

MongoDB 聚合

MongoDB中聚合(aggregate)主要用于处理数据(诸如统计平均值,求和等)，并返回计算后的数据结果。有点类似sql语句中的 count(*)。

aggregate() 方法

MongoDB中聚合的方法使用aggregate()。

语法

aggregate() 方法的基本语法格式如下所示：

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

实例

集合中的数据如下：

```

{
  _id: ObjectId("7df78ad8902c"),
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',

```

```
by_user: 'runoob.com',
url: 'http://www.runoob.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
},
{
  _id: ObjectId(7df78ad8902d)
title: 'NoSQL Overview',
description: 'No sql database is very fast',
by_user: 'runoob.com',
url: 'http://www.runoob.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 10
},
{
  _id: ObjectId(7df78ad8902e)
title: 'Neo4j Overview',
description: 'Neo4j is no sql database',
by_user: 'Neo4j',
url: 'http://www.neo4j.com',
tags: ['neo4j', 'database', 'NoSQL'],
likes: 750
},
}
```

现在我们通过以上集合计算每个作者所写的文章数，使用aggregate()计算结果如下：

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
  "result" : [
    {
      "_id" : "runoob.com",
      "num_tutorial" : 2
    },
    {
      "_id" : "Neo4j",
      "num_tutorial" : 1
    }
  ],
  "ok" : 1
}
```

以上实例类似sql语句：*select by_user, count(*) from mycol group by by_user*

在上面的例子中，我们通过字段by_user字段对数据进行分组，并计算by_user字段相同值的总和。

下表展示了一些聚合的表达式:

表达式	描述	实例
\$sum	计算总和。	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$sum : "\$likes"}}}])
\$avg	计算平均值	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$avg : "\$likes"}}}])
\$min	获取集合中所有文档对应值得最小值。	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$min : "\$likes"}}}])
\$max	获取集合中所有文档对应值得最大值。	db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$max : "\$likes"}}}])

代码练习：

```
aggregate ( )

> db.user.find()
{ "_id" : ObjectId("5b0027ceae18a81beda47ef4"), "class" : "py01", "name" : "zhangsan" }
{ "_id" : ObjectId("5b0027d9ae18a81beda47ef5"), "class" : "py01", "name" : "lisi" }
{ "_id" : ObjectId("5b0027feae18a81beda47ef6"), "class" : "py02", "name" : "wangwu" }
> db.user.aggregate([{$group:{_id:"$class",num:{$sum:1}}}])
{ "_id" : "py02", "num" : 1 }
{ "_id" : "py01", "num" : 2 }
```

七、MongoDB 索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

ensureIndex() 方法

MongoDB使用 ensureIndex() 方法来创建索引。

语法

ensureIndex()方法基本语法格式如下所示：

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

语法中 Key 值为你要创建的索引字段，1为指定按升序创建索引，如果你想按降序来创建索引指定为-1即可。

实例

```
添加普通索引
db.col.ensureIndex({"title":1})

查询索引
db.col.getIndexes()

删除索引
db.col.dropIndex({"title:1})

添加唯一索引 注意 字段保持唯一 否则报错,删除后可以添加
db.lis.ensureIndex({name:1},{unique:true})

查询分析
db.lis.find({"name":'a'}).explain()
```

ensureIndex() 方法中你也可以设置使用多个字段创建索引（关系型数据库中称作复合索引）。

```
>db.col.ensureIndex({"title":1,"description":-1})
>
```

ensureIndex() 接收可选参数，可选参数列表如下：

Parameter	Type	Description
background	Boolean	建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引，即增加 "background" 可选参数。"background" 默认值为 false 。
unique	Boolean	建立的索引是否唯一。指定为true创建唯一索引。默认值为 false 。
name	string	索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。

实例

在后台创建索引：

```
db.values.ensureIndex({open: 1, close: 1}, {background: true})
```

通过在创建索引时加background:true 的选项，让创建工作在后台执行

代码练习：

```
> db.col.find().pretty()
{
  "_id" : ObjectId("5b10bf135426fe9f694e4893"),
  "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "php"
  ],
  "likes" : 200
}
{
  "_id" : ObjectId("5b10bf245426fe9f694e4894"),
  "title" : "Java 教程",
  "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "java"
  ],
  "likes" : 150
}
```

```

    "likes" : 150
  }
  {
    "_id" : ObjectId("5b10bf335426fe9f694e4895"),
    "title" : "MongoDB 教程",
    "description" : "MongoDB 是一个 Nosql 数据库",
    "by" : "菜鸟教程",
    "url" : "http://www.runoob.com",
    "tags" : [
      "mongodb"
    ],
    "likes" : 100
  }
}
> db.col.insert({"title":1,"name":"xiaokeai"})
WriteResult({ "nInserted" : 1 })
> db.col.find().pretty()
{
  "_id" : ObjectId("5b10bf135426fe9f694e4893"),
  "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "php"
  ],
  "likes" : 200
}
{
  "_id" : ObjectId("5b10bf245426fe9f694e4894"),
  "title" : "Java 教程",
  "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "java"
  ],
  "likes" : 150
}
{
  "_id" : ObjectId("5b10bf335426fe9f694e4895"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb"
  ],
  "likes" : 100
}
{
  "_id" : ObjectId("5b10c8205426fe9f694e4896"),
  "title" : 1,
  "name" : "xiaokeai"
}
--创建索引，1代表的是按照升序的顺序
> db.col.ensureIndex({"title":1})
{
  "ok" : 0,
  "errmsg" : "Index with name: title_1 already exists with different options",
  "code" : 85,
  "codeName" : "IndexOptionsConflict"
}
--查看索引（数据本身默认就会以id存在索引，现在我们创建了一个索引，所以总共有两个索引）
> db.col.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "py.col"
  },
  {

```

```

    "v" : 2,
    "unique" : true,
    "key" : {
        "title" : 1
    },
    "name" : "title_1",
    "ns" : "py.col"
}
]
--删除索引
> db.col.dropIndex({title:1})
{ "nIndexesWas" : 2, "ok" : 1 }
> db.col.getIndexes()
[
  {
    "v" : 2,
    "key" : {
        "_id" : 1
    },
    "name" : "_id_",
    "ns" : "py.col"
  }
]
> db.col.find().pretty()
{
  "_id" : ObjectId("5b10bf135426fe9f694e4893"),
  "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "php"
  ],
  "likes" : 200
}
{
  "_id" : ObjectId("5b10bf245426fe9f694e4894"),
  "title" : "Java 教程",
  "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "java"
  ],
  "likes" : 150
}
{
  "_id" : ObjectId("5b10bf335426fe9f694e4895"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb"
  ],
  "likes" : 100
}
{
  "_id" : ObjectId("5b10c8205426fe9f694e4896"),
  "title" : 1,
  "name" : "xiaokeai"
}
--创建唯一索引
> db.col.ensureIndex({title:1},{unique:true})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.col.getIndexes()
[
  {
    "v" : 2,

```

```

    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "py.col"
  },
  {
    "v" : 2,
    "unique" : true,
    "key" : {
      "title" : 1
    },
    "name" : "title_1",
    "ns" : "py.col"
  }
]
--因为又创建了一个唯一性索引，所以不可以再插入相同的数据，当插入的时候会报错
> db.col.insert({"title":1,"name":"xiaokeai"})
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "E11000 duplicate key error collection: py.col index: title_1 dup key: { : 1.0 }"
  }
})
> db.col.find().pretty()
{
  "_id" : ObjectId("5b10bf135426fe9f694e4893"),
  "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "php"
  ],
  "likes" : 200
}
{
  "_id" : ObjectId("5b10bf245426fe9f694e4894"),
  "title" : "Java 教程",
  "description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设计语言。",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "java"
  ],
  "likes" : 150
}
{
  "_id" : ObjectId("5b10bf335426fe9f694e4895"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb"
  ],
  "likes" : 100
}
{
  "_id" : ObjectId("5b10c8205426fe9f694e4896"),
  "title" : 1,
  "name" : "xiaokeai"
}
>

```

查询分析是查询语句性能分析的重要工具。

MongoDB 中查询分析用 explain() 和 hint() 方法

添加20万条数据

```

> for(var i = 0; i < 200000; i++){db.user.insert({"name":"lrh"+i,"age":18})}
WriteResult({ "nInserted" : 1 })

```

```
> db.user.dropIndexes()
{
  "nIndexesWas" : 1,
  "msg" : "non-_id indexes dropped for collection",
  "ok" : 1
}
```

删除 user 集合中字段 name 上的索引，然后查询 name = "lrh100000"，利用explain("executionStats")查询此时执行的时间。

```
> db.user.find({"name":"lrh100000"}).explain("executionStats")
```

```
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "py4.user",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "name" : {
        "$eq" : "lrh100000"
      }
    },
    "winningPlan" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "name" : {
          "$eq" : "lrh100000"
        }
      },
      "direction" : "forward"
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 1,
    "executionTimeMillis" : 1308,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 200000,
    "executionStages" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "name" : {
          "$eq" : "lrh100000"
        }
      },
      "nReturned" : 1,
      "executionTimeMillisEstimate" : 106,
      "works" : 200002,
      "advanced" : 1,
      "needTime" : 200000,
      "needYield" : 0,
      "saveState" : 1566,
      "restoreState" : 1566,
      "isEOF" : 1,
      "invalidates" : 0,
      "direction" : "forward",
      "docsExamined" : 200000
    }
  },
  "serverInfo" : {
    "host" : "may-virtual-machine",
    "port" : 27017,
    "version" : "3.4.15",
    "gitVersion" : "52e5b5fb3a2a5b1a217f5e647b5061817475f9"
  },
  "ok" : 1
}
```

没用索引查询用到的时间是 1308毫秒

给 user 集合中 name 字段添加索引，然后再查询同一个条件，看执行查询所用了多久时间

```
> db.user.ensureIndex({"name":1})
```

```
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```



```

}
> db.user.find({"name":"lrh100000"}).explain("executionStats")
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "py4.user",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "name" : {
        "$eq" : "lrh100000"
      }
    },
    "winningPlan" : {
      "stage" : "FETCH",
      "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
          "name" : 1
        },
        "indexName" : "name_1",
        "isMultiKey" : false,
        "multiKeyPaths" : {
          "name" : [ ]
        },
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 2,
        "direction" : "forward",
        "indexBounds" : {
          "name" : [
            ["lrh100000\","lrh100000\"]
          ]
        }
      },
      "rejectedPlans" : [ ]
    },
    "executionStats" : {
      "executionSuccess" : true,
      "nReturned" : 1,

```

如果用到了索引，explain() 方法会返回 winningPlan，标识用到的索引名称 indexName

我们可以清楚到处，用了索引，执行时间只有 620毫秒

```

    "executionTimeMillis" : 620,
    "totalKeysExamined" : 1,
    "totalDocsExamined" : 1,
    "executionStages" : {
      "stage" : "FETCH",
      "nReturned" : 1,

      "executionTimeMillisEstimate" : 21,
      "works" : 2,
      "advanced" : 1,
      "needTime" : 0,
      "needYield" : 0,
      "saveState" : 1,
      "restoreState" : 1,
      "isEOF" : 1,
      "invalidates" : 0,
      "docsExamined" : 1,
      "alreadyHasObj" : 0,
      "inputStage" : {
        "stage" : "IXSCAN",
        "nReturned" : 1,
        "executionTimeMillisEstimate" : 21,
        "works" : 2,
        "advanced" : 1,
        "needTime" : 0,
        "needYield" : 0,
        "saveState" : 1,
        "restoreState" : 1,
        "isEOF" : 1,
        "invalidates" : 0,

```

```
    "keyPattern" : {
      "name" : 1
    },
    "indexName" : "name_1",
    "isMultiKey" : false,
    "multiKeyPaths" : {
      "name" : [ ]
    },
    "isUnique" : false,
    "isSparse" : false,
    "isPartial" : false,
    "indexVersion" : 2,
    "direction" : "forward",
    "indexBounds" : {
      "name" : [
        ["\lrh100000\","\lrh100000\"]
      ]
    },
    "keysExamined" : 1,
    "seeks" : 1,
    "dupsTested" : 0,
    "dupsDropped" : 0,
    "seenInvalidated" : 0
  }
}
},
"serverInfo" : {
  "host" : "may-virtual-machine",
  "port" : 27017,
  "version" : "3.4.15",
  "gitVersion" : "52e5b5fb3a2a5b1a217f5e647b5061817475f9"
},
"ok" : 1
}
```

八、MongoDB 备份(mongodump)与恢复(mongorestore)

MongoDB数据备份

在MongoDb中我们使用mongodump命令来备份MongoDB数据。该命令可以导出所有数据到指定目录中。

mongodump命令可以通过参数指定导出的数据量级转存的服务器。

语法

mongodump命令脚本语法如下：

```
> mongodump -h dbhost -d dbname -o dbdirectory
```

- **-h**：B所在服务器地址，例如：127.0.0.1，当然也可以指定端口号：127.0.0.1:27017
- **-d**：

需要备份的数据库实例，例如：test

- **-o**：
- 备份的数据存放位置，例如：c:\data\dump，当然该目录需要提前建立，在备份完成后，系统自动在dump目录下建立一个test目录，这个目录里面存放该数据库实例的备份数据。

实例

先退出mongo客户端

```
mongodump -d py4 -o /home/yc/data/
```

MongoDB数据恢复

mongodb使用 mongorestore 命令来恢复备份的数据。

语法

mongorestore命令脚本语法如下：

```
> mongorestore -h <hostname> <:port> -d dbname <path>
退出mongo客户端执行
mongorestore -d py /home/yc/data/py4
```

--host <:port>, -h <:port> : MongoDB所在服务器地址，默认为：localhost:27017

--db, -d : 需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如test2

--drop : 恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，慎用哦！

<path> : mongorestore 最后的一个参数，设置备份数据所在位置，例如：c:\data\dump\test。

你不能同时指定 <path> 和 --dir 选项，--dir也可以设置备份目录。

--dir : 指定备份的目录 你不能同时指定 <path> 和 --dir 选项。

接下来我们执行以下命令：

```
> mongorestore
```

执行以上命令输出结果如下：

代码练习：

```
root@may-virtual-machine:/# mkdir data2
```

```
root@may-virtual-machine:/# cd data2
```

```
root@may-virtual-machine:/data2# ls
```

--备份数据到/data2中

```
root@may-virtual-machine:/data2# mongodump -d py -o /data2
```

```
2018-06-01T13:03:09.821+0800   writing py.user to
2018-06-01T13:03:10.060+0800   writing py.coo to
2018-06-01T13:03:10.068+0800   writing py.col to
2018-06-01T13:03:10.069+0800   writing py.coo.insertdb.col to
2018-06-01T13:03:10.174+0800   done dumping py.coo (8 documents)
2018-06-01T13:03:10.189+0800   writing py.co to
2018-06-01T13:03:10.196+0800   done dumping py.col (4 documents)
2018-06-01T13:03:10.234+0800   done dumping py.coo.insertdb.col (1 document)
2018-06-01T13:03:10.249+0800   done dumping py.co (0 documents)
2018-06-01T13:03:11.756+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:14.656+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:17.653+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:20.659+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:23.649+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:26.688+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:29.689+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:32.690+0800   [.....] py.user 101/200000 (0.1%)
2018-06-01T13:03:35.893+0800   [#####.....] py.user 55833/200000 (27.9%)
2018-06-01T13:03:38.874+0800   [#####.....] py.user 156774/200000 (78.4%)
2018-06-01T13:03:40.529+0800   [#####.....] py.user 200000/200000 (100.0%)
2018-06-01T13:03:40.535+0800   done dumping py.user (200000 documents)
root@may-virtual-machine:/data2# ls
```

```
py
```

```
root@may-virtual-machine:/data2# cd py
```

```
root@may-virtual-machine:/data2/py# ls
```

```
co.bson      coo.bson      user.bson
```

```
col.bson     coo.insertdb.col.bson  user.metadata.json
```

```
col.metadata.json  coo.insertdb.col.metadata.json
```

```
co.metadata.json  coo.metadata.json
```

```
> show dbs
```

```
admin 0.000GB
```

```
local 0.000GB
```

```
py 0.007GB
```

```
test 0.000GB
```

```
> use py
```

```
switched to db py
```

```
> show tables
```

```
co
```

```
col
```

```
coo
```

```
coo.insertdb.col
```

```
user
```

--删除py数据库

```
> db.dropDatabase()
```

```
{ "dropped" : "py", "ok" : 1 }
```

```
> show dbs
admin 0.000GB
local 0.000GB
test 0.000GB
--恢复数据
root@may-virtual-machine:/data2/py# mongorestore -d py /data2/py
2018-06-01T13:22:52.815+0800 the --db and --collection args should only be used when restoring from a BSON file. Other uses are deprecated and will not exist in the
future; use --nsInclude instead
2018-06-01T13:22:52.816+0800 building a list of collections to restore from /data2/py dir
2018-06-01T13:22:52.831+0800 reading metadata for py.user from /data2/py/user.metadata.json
2018-06-01T13:22:52.910+0800 reading metadata for py.col from /data2/py/col.metadata.json
2018-06-01T13:22:53.094+0800 restoring py.col from /data2/py/col.bson
2018-06-01T13:22:53.175+0800 restoring indexes for collection py.col from metadata
2018-06-01T13:22:53.183+0800 reading metadata for py.coo from /data2/py/coo.metadata.json
2018-06-01T13:22:53.183+0800 reading metadata for py.coo.insertdb.col from /data2/py/coo.insertdb.col.metadata.json
2018-06-01T13:22:53.196+0800 restoring py.user from /data2/py/user.bson
2018-06-01T13:22:53.326+0800 finished restoring py.col (4 documents)
2018-06-01T13:22:53.326+0800 reading metadata for py.co from /data2/py/co.metadata.json
2018-06-01T13:22:53.362+0800 restoring py.coo from /data2/py/coo.bson
2018-06-01T13:22:53.422+0800 restoring py.coo.insertdb.col from /data2/py/coo.insertdb.col.bson
2018-06-01T13:22:54.354+0800 restoring py.co from /data2/py/co.bson
2018-06-01T13:22:54.392+0800 no indexes to restore
2018-06-01T13:22:54.392+0800 finished restoring py.coo (8 documents)
2018-06-01T13:22:54.393+0800 no indexes to restore
2018-06-01T13:22:54.393+0800 finished restoring py.co (0 documents)
2018-06-01T13:22:54.435+0800 no indexes to restore
2018-06-01T13:22:54.435+0800 finished restoring py.coo.insertdb.col (1 document)
2018-06-01T13:22:55.773+0800 [#.....] py.user 517KB/10.4MB (4.9%)
2018-06-01T13:22:58.804+0800 [###.....] py.user 1.48MB/10.4MB (14.3%)
2018-06-01T13:23:01.826+0800 [####.....] py.user 1.84MB/10.4MB (17.8%)
2018-06-01T13:23:04.768+0800 [#####.....] py.user 2.41MB/10.4MB (23.2%)
2018-06-01T13:23:07.763+0800 [#####.....] py.user 3.44MB/10.4MB (33.1%)
2018-06-01T13:23:10.755+0800 [#####.....] py.user 4.42MB/10.4MB (42.6%)
2018-06-01T13:23:13.759+0800 [#####.....] py.user 7.08MB/10.4MB (68.2%)
2018-06-01T13:23:16.749+0800 [#####.....] py.user 7.19MB/10.4MB (69.2%)
2018-06-01T13:23:20.104+0800 [#####.....] py.user 7.81MB/10.4MB (75.3%)
2018-06-01T13:23:22.749+0800 [#####.....] py.user 10.4MB/10.4MB (100.0%)
2018-06-01T13:23:22.751+0800 [#####.....] py.user 10.4MB/10.4MB (100.0%)
2018-06-01T13:23:22.751+0800 restoring indexes for collection py.user from metadata
2018-06-01T13:23:33.627+0800 finished restoring py.user (200000 documents)
2018-06-01T13:23:33.627+0800 done
root@may-virtual-machine:/data2/py#
```

```
> show dbs
admin 0.000GB
local 0.000GB
py 0.007GB
test 0.000GB
>
啦啦~py又回来了！就是酱紫的。
```

总结：
备份：
mongodump -d py -o /data2
恢复：
mongorestore -d py /data2/py

九、Python 访问 MongoDB

对于使用 Python 访问 MongoDB，需要先安装 `PyMongo` 软件包，该包实现了 Python 的 MongoDB 驱动。

通过以下命令建立工作环境，安装软件包：

```
$ sudo pip install pymongo
安装后使用 pip3 list查看是否存在pymongo
```

创建脚本或使用命令行模式

```
#链接到 127.0.0.1 的mongo
client = pymongo.MongoClient('127.0.0.1', 27017)
#选择test库
db = client.test
# print(db)
```

查询所有数据 db.test.find()

```
for user in db.test.find():
    print(user)
    print('<br>')
```

通过 PyMongo 插入数据也非常简单，直接通过 `insert` 方法：

```
# 定义数据
data = {'name': 'Aiden', 'age': 30}
# 执行数据添加
db.test.insert(data)
```