

NoSQL 数据库

简介

如今的网站对数据存储要求越来越灵活，在这种需求下 NoSQL 也就是非关系数据库越来越流行。所谓非关系数据库，是指不使用 SQL 语言进行数据操作的数据库的统称。这类数据库存储数据时没有固定的模式，不支持数据表 join 的操作，可以很方便的进行横向扩展。非关系数据库种类很多，其中 MongoDB 和 Redis 应用广泛。在本节实验中，我们将学习 MongoDB 和 Redis 的基本操作，以及怎么样使用 Python 代码访问这些数据库。

知识点

- MongoDB 的基础操作；
- Redis 基础操作；
- 使用 Python 访问 MongoDB 和 Redis；

MongoDB

MongoDB 是非常流行的 NoSQL 数据库，支持自动化的水平扩展，同时也被称为文档数据库，因为数据按文档的形式进行存储（BSON 对象，类似于 JSON）。在 MongoDB 中数据存储的组织方式主要分为四级：

- 数据库实例，比如一个 app 使用一个数据库；
- collection 文档集合，一个数据库包含多个文档集合，类似于 MySQL 中的表；
- document 文档，一个文档代表一项数据，类似于 JSON 对象，对应于 MySQL 表中的一条记录；
- 字段：一个文档包含多个字段；

MongoDB 存储的数据可以是无模式的，比如在一个集合中的所有文档不需要有一致的结构。也就是说往同一个表中插入不同的数据时，这些数据之间不必有同样的字段。这和 MySQL 彻底不同，在 MySQL 中创建表时就已经确定了数据项的字段，向其中插入数据时，必须是相同的结构。

本节实验基于 MongoDB 3.4 版本讲解。

环境准备

实验楼环境中已经安装了 MongoDB，每次启动实验后，需要启动 MongoDB 服务。在实验桌面上启动终端后，通过以下命令启动数据库：

🔗 [楼+之Python实战第10期 \(/courses/1190\)](/courses/1190)

数据库启动成功以后，通过以下命令链接到数据库：

```
$ mongo
```

mongo 是 MongoDB 的客户端 Shell，执行该命令时可以指定连接的 MongoDB 地址等信息，未指定时将连接默认地址，默认情况下 MongoDB 服务会监听在 127.0.0.1:27017 地址。

后文出现的所有 MongoDB 操作命令都将基于 mongo shell 输入。

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：

```
$ cd ~/Code
$ sudo pip install virtualenv
$ virtualenv -p /usr/bin/python3.5 env
$ source env/bin/activate
$ pip install pymongo ipython
```

注意这里使用 virtualenv 环境，并且指定了使用 Python 3.5 的版本，然后在其中安装所需的软件包可以使用 pip，pip 默认就是安装的 Python 3.5 对应的包。

以上命令在 Code 目录创建一个 virtualenv 环境，接着在这个虚拟环境中安装了本节实验需要的软件包。后续的交互式命令都通过 IPython 终端输入。可以通过以下命令启动 IPython 终端：

```
$ source env/bin/activate
$ ipython
```

后文出现的代码中，In [1] 类似样式的字符是 IPython 的提示符，不需要输入。

NOSQL 实验环境启动视频：

0:00 / 2:53

CRUD 操作

MongoDB 存储的文档记录是一个 BSON 对象，类似于 JSON 对象，由键值对组成。比如一条用户记录：

```
{
  name: "Aiden",
  age: 30,
  email: "luojin@simplecloud.cn"
}
```

每一个文档都有一个 `_id` 字段，该字段是主键，用于唯一的确定一条记录。如果往 MongoDB 中插入数据时没有指定 `_id` 字段，那么会自动产生一个 `_id` 字段，该字段的类型是 `ObjectId` (<https://docs.mongodb.com/manual/reference/bson-types/#objectid>)，长度是 12 个字节。在 MongoDB 文档的字段支持字符串，数字，时间戳等类型。一个文档最大可以达到 16M，可以存储相当多的数据。

先尝试往 MongoDB 中插入一条数据：

```
$ mongo
> use shiyanlou
> db.user.insertOne({name: "Aiden", age: 30, email: "luojin@simplecloud.cn", addr: ["CD", "SH"]})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("59a8034064e0acb13483d512")
}
> show databases;
admin      0.000GB
local      0.000GB
shiyanlou  0.000GB
> show collections;
user
```

可以看到，在插入数据前使用 `use` 指令，切换到了 `shiyanlou` 数据库，尽管该数据库暂时不存在，但当我们插入数据后，该数据库就被自动创建了。`show databases` 和 `show collections` 分别显示了当前存在的数据库和当前数据库的所有文档集合。而且数据插入后，自动添加了 `_id` 字段。插入多条数据，可以使用 `db.collection.insertMany` 方法：

🔗 楼+之Python实战第10期 (/courses/1190)

```
... {name: "lxttx", age: 28, email: "lxttx@simplecloud.cn", addr: ["BJ", "CD"]},
... {name: "jin", age: 31, email: "jin@simplecloud.cn", addr: ["GZ", "SZ"]},
... {name: "nan", age: 26, email: "nan@simplecloud.cn", addr: ["NJ", "AH"]}
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("59a8034564e0acb13483d513"),
    ObjectId("59a8034564e0acb13483d514"),
    ObjectId("59a8034564e0acb13483d515")
  ]
}
```

查询数据可以使用 `db.collection.find` 方法，可以指定查询过滤条件：

```
> db.user.find()
{ "_id" : ObjectId("59a8034064e0acb13483d512"), "name" : "Aiden", "age" : 30, "email" : "luo
jin@simplecloud.cn", "addr" : [ "CD", "SH" ] }
{ "_id" : ObjectId("59a8034564e0acb13483d513"), "name" : "lxttx", "age" : 28, "email" : "lxt
tx@simplecloud.cn", "addr" : [ "BJ", "CD" ] }
{ "_id" : ObjectId("59a8034564e0acb13483d514"), "name" : "jin", "age" : 31, "email" : "jin@s
implecloud.cn", "addr" : [ "GZ", "SZ" ] }
{ "_id" : ObjectId("59a8034564e0acb13483d515"), "name" : "nan", "age" : 26, "email" : "nan@s
implecloud.cn", "addr" : [ "NJ", "AH" ] }

> db.user.find({name: "jin"})
{ "_id" : ObjectId("59a8034564e0acb13483d514"), "name" : "jin", "age" : 31, "email" : "jin@s
implecloud.cn", "addr" : [ "GZ", "SZ" ] }

> db.user.find({age: {$gt: 30}})
{ "_id" : ObjectId("59a8034564e0acb13483d514"), "name" : "jin", "age" : 31, "email" : "jin@s
implecloud.cn", "addr" : [ "GZ", "SZ" ] }

> db.user.find({addr: "CD"})
{ "_id" : ObjectId("59a8034064e0acb13483d512"), "name" : "Aiden", "age" : 30, "email" : "luo
jin@simplecloud.cn", "addr" : [ "CD", "SH" ] }
{ "_id" : ObjectId("59a8034564e0acb13483d513"), "name" : "lxttx", "age" : 28, "email" : "lxt
tx@simplecloud.cn", "addr" : [ "BJ", "CD" ] }
```

上面例子中，我们先通过 `db.user.find()` 获取到了之前插入的全部数据。接着使用不同的过滤条件进行了查询，其中有一些查询如 `{age: {$gt: 30}}` 表示查询年龄大于 30 的用户。还可以发现查询数组中是否存在某一元素也非常方便，上面的例子中查询出了所有地址含有 CD 用户。

MongoDB 的查询功能非常强大，可以组合各种查询条件，更多的使用方法可以学习实验楼的其他课程。更新数据主要通过 `db.user.updateOne` 或者 `db.user.updateMany` 方法，前者更新一条记录，后者更新多条记录：

楼+之Python实战第10期 (/courses/1190)

```
... {name: "Aiden"},
... {$set: {age: 29, addr: ["CD", "SH", "BJ"]}}
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.user.find({name: "Aiden"})
{ "_id" : ObjectId("59a8034064e0acb13483d512"), "name" : "Aiden", "age" : 29, "email" : "luo
jin@simplecloud.cn", "addr" : [ "CD", "SH", "BJ" ] }
```

可以看到成功的更新了一条记录。删除数据也非常简单，可以通过 `db.user.deleteMany` 或 `db.user.deleteOne` 方法：

```
> db.user.deleteMany({addr: "CD"})
{ "acknowledged" : true, "deletedCount" : 2 }
> db.user.find()
{ "_id" : ObjectId("59a8034564e0acb13483d514"), "name" : "jin", "age" : 31, "email" : "jin@simplecloud.cn", "addr" : [ "GZ", "SZ" ] }
{ "_id" : ObjectId("59a8034564e0acb13483d515"), "name" : "nan", "age" : 26, "email" : "nan@simplecloud.cn", "addr" : [ "NJ", "AH" ] }
```

上面的命令成功的删除所有地址包含 "CD" 的用户，共删除了两条记录。

MongoDB 基本操作视频：

0:00 / 8:12

Python 操作 MongoDB

在 Python 中访问 MongoDB 数据库，主要通过 PyMongo (<https://api.mongodb.com/python/current/tutorial.html>) 软件包。该软件包含一个 `MongoClient` 对象，可以用于建立 MongoDB 客户端。在 IPython 中输入下面的示例代码，创建客户端：

```
In [2]: from pymongo import MongoClient

In [3]: client = MongoClient('127.0.0.1', 27017)

In [4]: db = client.shiyanlou
```

前文中，已经知道 MongoDB 默认监听在 `127.0.0.1:27017` 地址上，所以在创建 `client` 时，使用了该地址。客户端创建成功后，我们通过 `client.shiyanlou` 方式选择了 `shiyanlou` 数据库。接着就可以查询所有的文档了：

```

In [13]: for user in db.user.find():
...:     print(user)
...:
{'_id': ObjectId('59a8034564e0acb13483d514'), 'name': 'jin', 'age': 31.0, 'email': 'jin@simplecloud.cn', 'addr': ['GZ', 'SZ']}
{'_id': ObjectId('59a8034564e0acb13483d515'), 'name': 'nan', 'age': 26.0, 'email': 'nan@simplecloud.cn', 'addr': ['NJ', 'AH']}

```

通过 PyMongo 插入数据也非常简单，直接通过 `insert_one` 方法：

```

In [14]: user = {'name': 'Aiden', 'age': 30, 'addr': ['CD', 'SH', 'BJ']}

In [15]: db.user.insert_one(user)
Out[15]: <pymongo.results.InsertOneResult at 0x10730aa08>
In [17]: db.user.find_one({'name': 'Aiden'})
Out[17]:
{'_id': ObjectId('59a80988a75acb3615913dc6'),
 'addr': ['CD', 'SH', 'BJ'],
 'age': 30,
 'name': 'Aiden'}

```

数据插入以后，我们使用 `find_one` 方法，查询了该记录，查询方法几乎和 mongo shell 查询方法相同。查询发现，没有设置 `email` 字段，可以通过 `update_one` 方法更新记录：

```

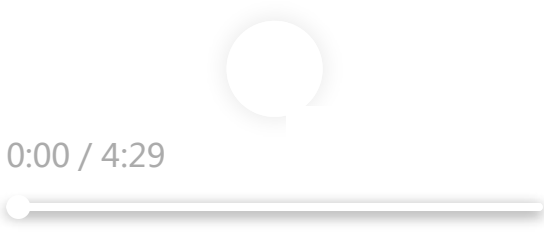
In [19]: db.user.update_one({'name': 'Aiden'}, {'$set': {'email': 'aiden@simplecloud.cn'}})
Out[19]: <pymongo.results.UpdateResult at 0x1070dce08>

In [20]: db.user.find_one({'name': 'Aiden'})
Out[20]:
{'_id': ObjectId('59a80988a75acb3615913dc6'),
 'addr': ['CD', 'SH', 'BJ'],
 'age': 30,
 'email': 'aiden@simplecloud.cn',
 'name': 'Aiden'}

```

可以发现 PyMongo 的很多操作类似于 mongo shell 的操作，比较简单。

Python 操作 MongoDB 视频：



0:00 / 4:29

- 拓展阅读：MongoDB 官方文档（英文）(<https://docs.mongodb.com/>)

- 拓展阅读：MongoDB 手册（中文）(<https://mongodb-lou-plus.com/courses/1190/documentation.readthedocs.io/en/latest/>)

练习题：存储楼+的学习数据

在 /home/shiyanlou 目录下实现 mongotest.py 脚本，用来向 MongoDB 中的 louplus 数据库中存储楼+用户的数据，执行 python3 mongotest.py 之后 MongoDB 里具备如下的数据：

1. MongoDB 里有 louplus 数据库
2. louplus 数据库下有 users 集合
3. users 集合中包括下列两个用户的数据：

```
[{
  'user_id': 1000,
  'name': 'Shiyan',
  'pass': 10,
  'study_time': 50,
},
{
  'user_id': 2000,
  'name': 'Lou',
  'pass': 15,
  'study_time': 171,
}]
```

完成代码后，需要在终端执行下面命令才可以点击下一步：

```
$ python3 /home/shiyanlou/mongotest.py
```

完成后点击 下一步 ，系统将自动检测完成结果。

Redis

Redis 是一个内存数据库，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，可以极大地提高网站的响应速度。相对于其它内存数据库，Redis 具有以下几个优点：

- 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

实验楼环境中已经安装了Redis，每次启动实验后，需要手动启动 Redis。在实验桌面上启动终端后，通过以下命令启动数据库：

```
$ sudo service redis-server start
```

数据库启动成功以后，通过以下命令链接到数据库：

```
$ redis-cli  
127.0.0.1:6379>
```

redis-cli 是 Redis 的客户端 Shell，执行该命令时可以指定连接的 Redis 服务地址等信息，未指定时将连接默认地址，Redis 服务默认监听在 127.0.0.1:6379 地址。后文出现的所有 Redis 操作命令都将基于 redis-cli 输入。

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：

```
$ cd ~/Code  
$ sudo pip install virtualenv  
$ virtualenv -p /usr/bin/python3.5 env  
$ source env/bin/activate  
$ pip install redis ipython
```

以上命令在 Code 目录创建一个 virtualenv 环境，接着在这个虚拟环境中安装了本节实验需要的软件包。后续的交互式命令都通过 IPython 终端输入。可以通过以下命令启动 IPython 终端：

```
$ source env/bin/activate  
$ ipython
```


后文出现的代码中，In [1] 类似样式的字符是 IPython 的提示符，不需要输入。

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符串都是参数，可以自定义）：

- SET key value 设置键值；
- EXISTS key 判断键是否存在；
- EXPIRE key seconds 设置 Key 的过期时间，过期以后Key 将被自动删除；

- TTL key 获取 Key 的剩余生存时间；
-  楼+之Python实战第10期 (/courses/1190) DEL key 删除 Key；
- TYPE key 获取 Key 对应的 Value 的类型；

通过 redis-cli 演示以上的指令如下：

```
127.0.0.1:6379> exists user
(integer) 0
127.0.0.1:6379> set user aiden
OK
127.0.0.1:6379> get user
"aiden"
127.0.0.1:6379> type user
string
127.0.0.1:6379> expire user 5
(integer) 1
127.0.0.1:6379> ttl user
(integer) 4
127.0.0.1:6379> ttl user
(integer) 2
127.0.0.1:6379> ttl user
(integer) -2
127.0.0.1:6379> exists user
(integer) 0
```

上面例子中，首先判断 user 键是否存在，接着通过 SET 设置了值，接着还使用 EXPIRE 指令设置了过期时间为 5 秒。可以看到 5 秒后，user 键就被自动删除了。

有的时候会看到输出的字符串前有 b 这样的前缀，这其实不是字符串，而是字符串经过某种编码之后生成的字节码，对应的还有 u'xxxxx' 这类 unicode 编码字节码。

上文中已提到，Redis 还支持其他的数据结构，不仅仅是简单的字符串键值对，比如支持哈希类型的键值，这种数据结构中 Key 对应于一个哈希，而哈希又包含多个字段和相应的值。对于这种类型主要有下面的操作指令：

- HSET key field value 设置名称为 key 的哈希的字段 field 为值 value；
- HGET key field 获取名为 key 的哈希的字段 field；
- HGETALL key 获取名为 Key 的哈希所有字段和 Value；
- HKEYS key 获取名为 Key 的哈希的所有字段；
- HLEN key 获取名为 Key 的哈希的字段数量；

通过 redis-cli 演示如下：

🔗 楼之Python实战第10期 (/courses/1190)

```
(integer) 0
127.0.0.1:6379> hset user name aiden
(integer) 1
127.0.0.1:6379> hset user age 30
(integer) 1
127.0.0.1:6379> hmset user email luojin@simplecloud.cn addr chengdu
OK
127.0.0.1:6379> hgetall user
1) "name"
2) "aiden"
3) "age"
4) "30"
5) "email"
6) "luojin@simplecloud.cn"
7) "addr"
8) "chengdu"
127.0.0.1:6379> hkeys user
1) "name"
2) "age"
3) "email"
4) "addr"
127.0.0.1:6379> hget user addr
"chengdu"
127.0.0.1:6379> hlen user
(integer) 4
```

上面的例子中，设置了一个名为 `user` 的哈希。先使用 `HSET` 为单个字段赋值，接着使用 `HMSET` 为多个字段赋值。使用 `HGETALL` 能一次获取全部的字段和值。注：在 4.0 版本的 `redis-server` 中，`hset` 也可以设置多组键值对，实验环境中的 `redis-server` 就是此版本。

Redis 还支持有序集合，有序集合可以用于快速实现排名功能，主要的操作指令如下：

- `ZADD key score member` 将成员和对应的评分添加到有序集合中；
- `ZREVRANK key member` 获取 `member` 在有序集合 `key` 中的排名；

通过 `redis-cli` 演示如下：

```
127.0.0.1:6379> zadd rank 100 aiden
(integer) 1
127.0.0.1:6379> zadd rank 120 lxtttx
(integer) 1
127.0.0.1:6379> zadd rank 80 jin
(integer) 1
127.0.0.1:6379> zrevrank rank aiden
(integer) 1
127.0.0.1:6379> zrevrank rank lxtttx
(integer) 0
127.0.0.1:6379> zrevrank rank jin
(integer) 2
127.0.0.1:6379> zrevrank rank not_exist
(nil)
```

上面例子中，我们通过 `ZADD` 往 `rank` 中添加了三个成员，最后通过 `ZREVRANK` 依次获取了成员的排名，可以发现排名是从 0 开始计算的，排第 0 的成员得分最高。有序集合的排名，还有 `zrank` 方法，请大家自行试验结果。Redis 还有其他各种操作指令，篇幅有限就不一一介绍了。

Redis 基本操作视频：



下面补充两个数据类型：

数据类型 - 列表：

```
127.0.0.1:6379> lpush num one two three
(integer) 3
127.0.0.1:6379> type num # 查看数据类型
list
127.0.0.1:6379> lrange num 0 -1 # 打印全部元素
1) "three"
2) "two"
3) "one"
127.0.0.1:6379> llen num # 列表长度
(integer) 3
```

数据类型 - 集合：

127.0.0.1:6379> sadd player James Kobe Wall Wade Bosh # 创建集合
(integer) 5
127.0.0.1:6379> smembers player # 查看集合内全部成员
1) "Kobe"
2) "Wade"
3) "Wall"
4) "James"
5) "Bosh"
127.0.0.1:6379> scard player # 集合内成员数量
(integer) 5
127.0.0.1:6379> srem player Kobe # 删除某个成员
(integer) 1
127.0.0.1:6379> smembers player
1) "Wade"
2) "Wall"
3) "James"
4) "Bosh"
127.0.0.1:6379> spop player # 随机删除某个成员并返回
"James"
127.0.0.1:6379> smembers player
1) "Wade"
2) "Bosh"
3) "Wall"

Python 操作 Redis

Python 中访问 Redis 可以通过 `redis-py` (<https://github.com/andymccurdy/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:

```
In [1]: import redis

In [2]: r = redis.StrictRedis(host='127.0.0.1', db=0)

In [3]: r.ping()
Out[3]: True
```

上面的代码中, 通过 `redis.StrictRedis` 创建了一个 Redis 客户端, 其中 `db` 参数指定了链接的逻辑数据库编号为 0。不同编号的数据库, 可以有同名的 Key。客户端创建成功后, 就可以进行各种指令操作了。先尝试了 `ping` 方法, 该方法返回 `True` 表示数据库工作正常。`redis-py` 客户端有各种和 Redis 指令同名的方法, 调用这些方法就可以完成各种操作, 比如获取上文中创建的哈希键 `user` :

```
In [4]: r.hgetall('user')
Out[4]:
{b'addr': b'chengdu',
 b'age': b'30',
 b'email': b'luojin@simplecloud.cn',
 b'name': b'aiden'}
```

可以看到，redis-py 将返回的结果自动转换成了字典。

🔗 楼+之Python实战第10期 (/courses/1190)

看到输出的字符串前有 `b` 这样的前缀，表示字节编码的字符串，对应的还有 `u'xxxxx'` 这类 unicode 编码的字符串，都比较常见。

redis-py 基本操作视频：

0:00 / 3:14

Redis 也支持发布订阅消息模式。该功能使发布者和订阅者解耦，不需要对方的存在，只需要简单的往某一频道上发送数据就行了，订阅了该频道的订阅者会自动收到消息。下面我们进行演示，首先在 redis-py 客户端中订阅 `labreport-channel` 频道，并监听消息：

```
In [5]: p = r.pubsub()

In [6]: p.subscribe('labreport-channel')

In [7]: for msg in p.listen():
...:     print(msg)
...:
{'type': 'subscribe', 'pattern': None, 'channel': b'labreport-channel', 'data': 1}
```

然后在 redis-cli 客户端中，通过 `PUBLISH channel message` 指令往频道中发布消息 `message`：

```
127.0.0.1:6379> publish labreport-channel "1 msg from redis-cli"
(integer) 1
127.0.0.1:6379> publish labreport-channel "2 msg from redis-cli"
(integer) 1
127.0.0.1:6379>
```

消息发布以后，就可以看到 IPython 终端中马上收到了消息：

```
In [7]: for msg in p.listen():
...:     print(msg)
...:
{'type': 'subscribe', 'pattern': None, 'channel': b'labreport-channel', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'labreport-channel', 'data': b'1 msg from r
edis-cli'}
{'type': 'message', 'pattern': None, 'channel': b'labreport-channel', 'data': b'2 msg from r
edis-cli'}
```

- 拓展阅读：Redis 订阅发布消息 (<https://www.jianshu.com/p/7f1902685a22>)

- 拓展阅读：Redis 命令参考 (<http://redisdoc.com/>)
- 楼+之Python实战第10期 ([/courses/1190](http://courses/1190))
- 拓展阅读：Redis 中文网 (<http://www.redis.cn/>)

练习题：存储楼+的学习数据

在 /home/shiyanlou 目录下实现 redistest.py 脚本，用来向 Redis 中数据库存储楼+用户的数据，执行 python3 redistest.py 之后 Redis 里具备如下的数据：

```
127.0.0.1:6379> hgetall user1
1) "id"
2) "1000"
3) "name"
4) "Shiyan"
5) "pass"
6) "10"
7) "study_time"
8) "50"
127.0.0.1:6379> hgetall user2
1) "id"
2) "2000"
3) "name"
4) "Lou"
5) "pass"
6) "15"
7) "study_time"
8) "171"
127.0.0.1:6379> 
```



注意会用到 hmset 操作，该操作可以设置多个 key/value 键值对，例如：

```
>>> r = redis.StrictRedis(host='127.0.0.1', db=0)
>>> r.hmset('name1', {'key1': 'value1', 'key2': 'value2'})
```

完成代码后，需要在终端执行下面命令才可以点击下一步：

```
$ python3 /home/shiyanlou/redistest.py
```

完成后点击 下一步，系统将自动检测完成结果。

总结

本节实验中，我们学习了 MongoDB 和 Redis 的基础知识，在后续的项目实战中，碰到这方面的主题我们再深入讲解。

本节的知识点：

🔗 楼+之Python实战第10期 (/courses/1190)

- MongoDB 的基础操作；
 - Redis 基础操作；
 - 使用 Python 访问 MongoDB 和 Redis；
-

NoSQL 应用的非常广泛，尤其在互联网项目中对于非结构化数据的存储基本都会采用 NoSQL 来存储。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：挑战：从数据库中读取内容 (/courses/1190/labs/8537/document)

下一节：挑战：为文章增加标签 (/courses/1190/labs/8539/document)