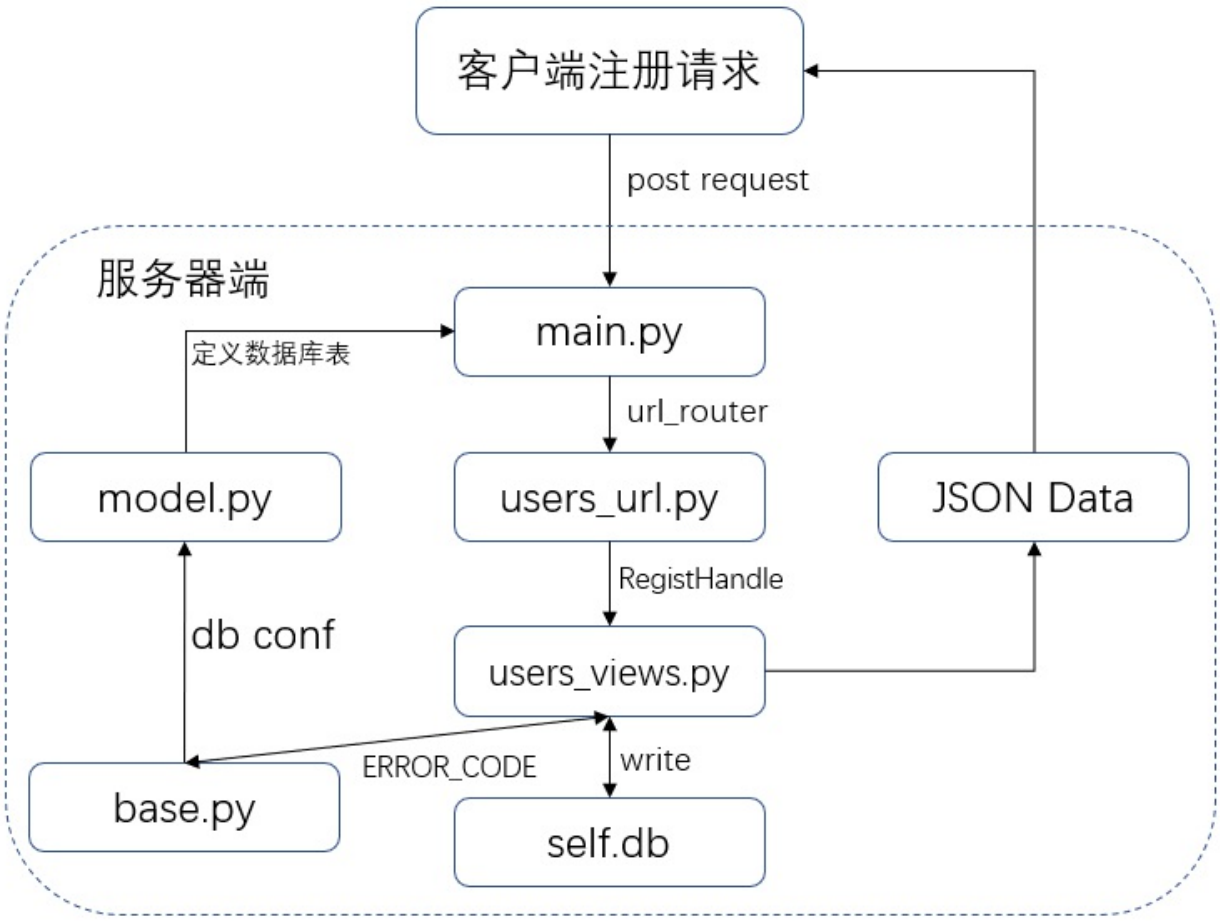
将用户信息写入 MySQL 数据库

上两小节已完成逻辑代码，这小节将学习使用 ORM 的方式将用户注册信息写入数据库中。

整个逻辑架构图

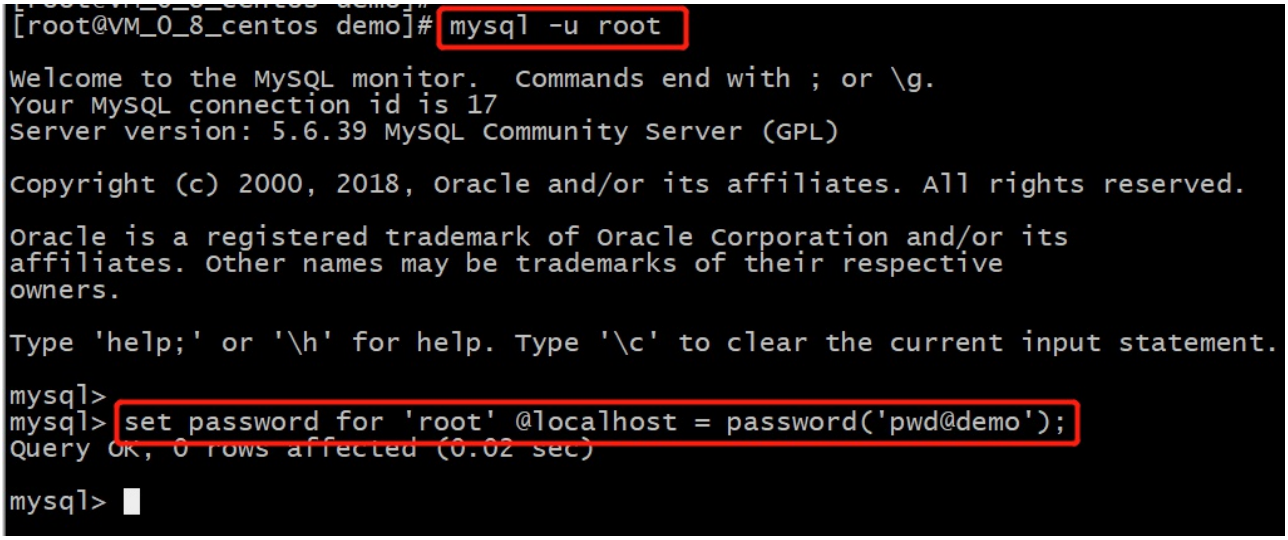


数据库的信息（如地址、端口、用户名和密码等）存放在 base.py 中，model.py 中定义了数据库表并从 base.py 中获取数据库信息。当 main.py 启动时，其将调用 model.py 初始化数据库。而 users_views.py 负责将客户端的请求数据写入数据库中，并返回注册成功信息。

配置数据用户名和密码

用户名为 root，密码为 pwd@demo，
在服务器端输入如下命令配置数据库。

```
mysql -u root
set password for 'root' @localhost =
password('pwd@demo');
```



创建数据库

在服务器端输入如下命令创建数据库。

```
CREATE DATABASE demo CHARACTER SET 'utf8' COLLATE
'utf8_general_ci';
```

创建完成后，使用 show databases 检查数据库是否创建成功。

```
mysql> CREATE DATABASE demo CHARACTER SET 'utf8' COLLATE 'utf8_general_ci';
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| demo               |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

代码中配置数据库

在配置文件 base.py 中指定数据库，需修改 conf/base.py，增加如下代码：

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import
declarative_base
engine =
create_engine('mysql://root:pwd@demo@localhost:33
06/demo?charset=utf8', encoding="utf8",
echo=False)
BaseDB = declarative_base()
```

代码中定义数据库表

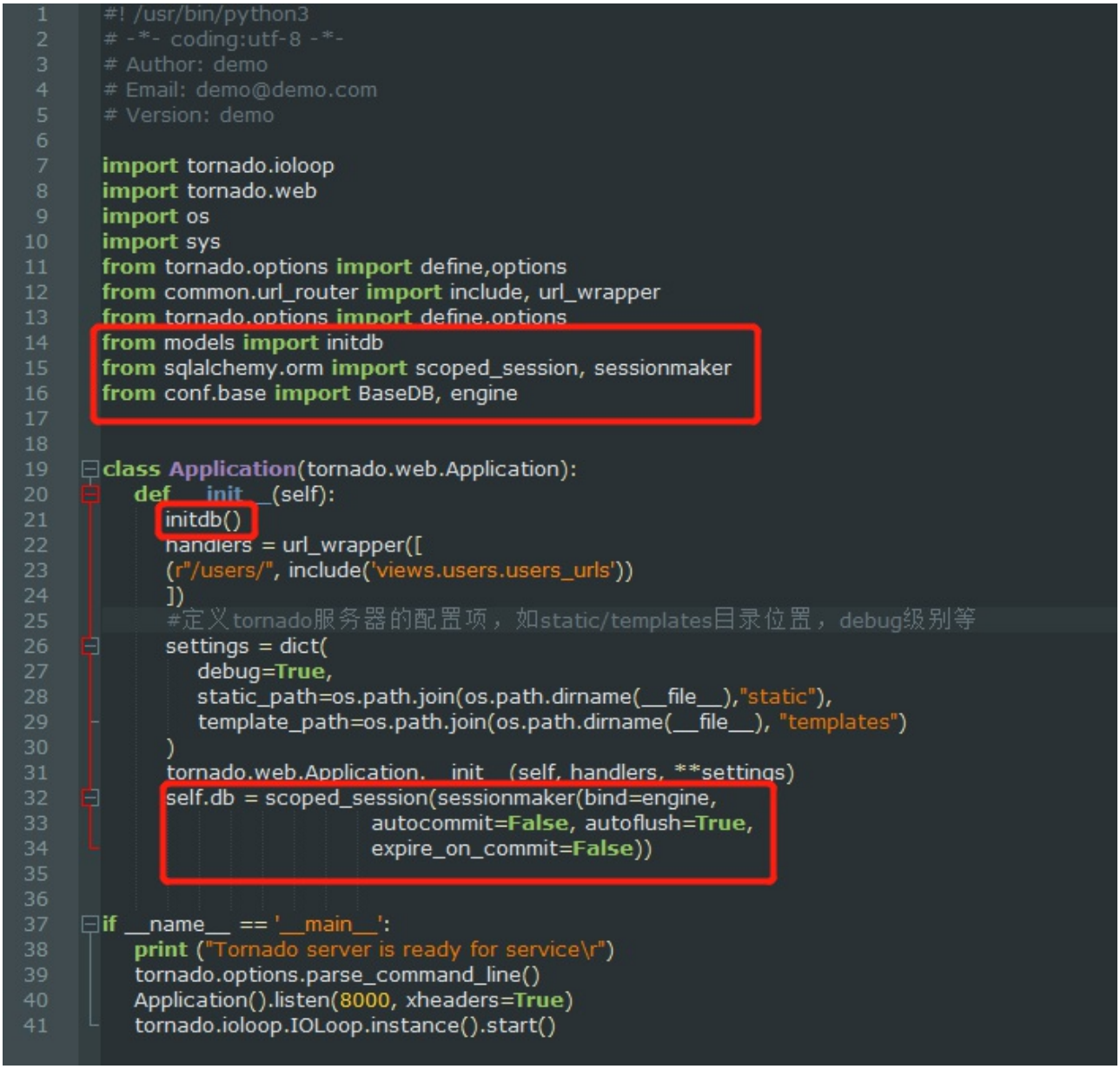
在前面的介绍中，我们提到，models.py 这个文件主要包含数据库表的定义及初始化。从第 6 小节中看到，用户注册信息包含手机号、密码和验证码。这里需要记录在数据库中的有手机号（phone）和密码（password），当然还包括创建的时间（createTime）。这些信息作为数据库表项，在 models.py 中定义，在 models.py 文件中输入如下代码：

```
#! /usr/bin/python3
# -*- coding:utf-8 -*-

from conf.base import BaseDB, engine
import sys
from sqlalchemy import (
Column,
Integer,
    String,
    DateTime
)


class Users(BaseDB):
    """table for users
    """
    __tablename__ = "users"
    #定义表结构，包括id，phone，password，createTime
    id = Column(Integer, primary_key=True)
    phone = Column(String(50), nullable=False)
    password = Column(String(50), nullable=True)
    createTime = Column(DateTime, nullable=True)

    def __init__(self, phone, password,
createTime):
        self.phone = phone
```

```
        self.password = password
        self.createTime = createTime


def initdb():
    BaseDB.metadata.create_all(engine)

if __name__ == '__main__':
    print ("Initialize database")
    initdb()
```

代码中初始化数据库

在 main.py 中，调用 models.py 初始化数据库并启用数据库



```
#! /usr/bin/python3
# -*- coding:utf-8 -*-
# Author: demo
# Email: demo@demo.com
# Version: demo

import tornado.ioloop
import tornado.web
import os
import sys
from tornado.options import define,options
from common.url_router import include, url_wrapper
from tornado.options import define,options
from models import initdb
from sqlalchemy.orm import scoped_session, sessionmaker
from conf.base import BaseDB, engine


class Application(tornado.web.Application):
    def __init__(self):
        initdb()
        handlers = url_wrapper([
        (r"/users/", include('views.users.users_urls'))
        ])
        #定义tornado服务器的配置项，如static/templates目录位置，debug级别等
        settings = dict(
            debug=True,
            static_path=os.path.join(os.path.dirname(__file__),"static"),
            template_path=os.path.join(os.path.dirname(__file__), "templates")
        )
        tornado.web.Application.__init__(self, handlers, **settings)
        self.db = scoped_session(sessionmaker(bind=engine,
                                 autocommit=False, autoflush=True,
                                 expire_on_commit=False))


if __name__ == '__main__':
    print ("Tornado server is ready for service\r")
    tornado.options.parse_command_line()
    Application().listen(8000, xheaders=True)
    tornado.ioloop.IOLoop.instance().start()
```

具体代码如下：

```
#! /usr/bin/python3
# -*- coding:utf-8 -*-
# Author: demo
# Email: demo@demo.com
# Version: demo

import tornado.ioloop
import tornado.web
import os
```

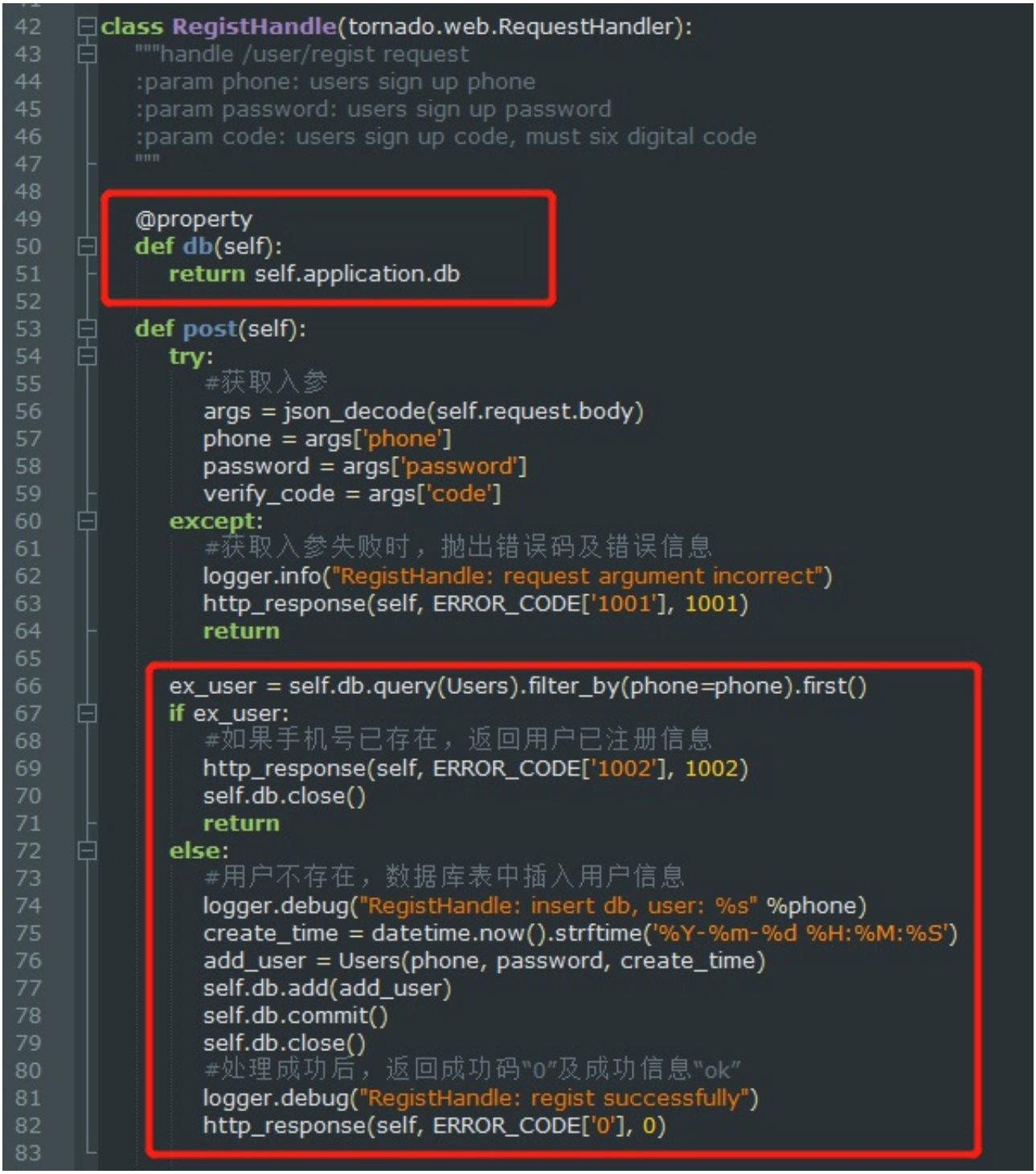
```
import sys
from tornado.options import define, options
from common.url_router import include,
url_wrapper
from tornado.options import define, options
from models import initdb
from sqlalchemy.orm import scoped_session,
sessionmaker
from conf.base import BaseDB, engine


class Application(tornado.web.Application):
    def __init__(self):
        initdb()
        handlers = url_wrapper([
        (r"/users/",
include('views.users.users_urls'))
        ])
        #定义tornado服务器的配置项，如
static/templates目录位置，debug级别等
        settings = dict(
            debug=True,

static_path=os.path.join(os.path.dirname(__file__
), "static"),

template_path=os.path.join(os.path.dirname(__file
__), "templates")
        )
        tornado.web.Application.__init__(self,
handlers, **settings)
        self.db =
scoped_session(sessionmaker(bind=engine,
```

```
autocommit=False, autoflush=True,

expire_on_commit=False))


if __name__ == '__main__':
    print ("Tornado server is ready for
service\r")
    tornado.options.parse_command_line()
    Application().listen(8000, xheaders=True)
    tornado.ioloop.IOLoop.instance().start()
```

代码将用户信息写入数据库

修改 users_views.py，将用户数据写入数据库中，修改内容包括从 models 中导入 Users 类表，并判断用户是否在数据库中。如果存在，返回注册失败信息；如果不存在，将用户信息写入数据库，并返回注册成功信息。

```python
#! /usr/bin/python3
# -*- coding:utf-8 -*-

import tornado.web
import sys
from tornado.escape import json_decode
import logging
from logging.handlers import TimedRotatingFileHandler
from datetime import datetime


#从commons中导入http_response方法
from common.commons import (
    http_response,
)

#从配置文件中导入错误码
from conf.base import (
    ERROR_CODE,
)

from models import (
    Users
)


########## Configure logging #############
logFilePath = "log/users/users.log"
logger = logging.getLogger("Users")
```



```
class RegistHandle(tornado.web.RequestHandler):
    """handle /user/regist request
    :param phone: users sign up phone
    :param password: users sign up password
    :param code: users sign up code, must six digital code
    """

    @property
    def db(self):
        return self.application.db

    def post(self):
        try:
            #获取入参
            args = json_decode(self.request.body)
            phone = args['phone']
            password = args['password']
            verify_code = args['code']
        except:
            #获取入参失败时，抛出错误码及错误信息
            logger.info("RegistHandle: request argument incorrect")
            http_response(self, ERROR_CODE['1001'], 1001)
            return

        ex_user = self.db.query(Users).filter_by(phone=phone).first()
        if ex_user:
            #如果手机号已存在，返回用户已注册信息
            http_response(self, ERROR_CODE['1002'], 1002)
            self.db.close()
            return
        else:
            #用户不存在，数据库表中插入用户信息
            logger.debug("RegistHandle: insert db, user: %s" %phone)
            create_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            add_user = Users(phone, password, create_time)
            self.db.add(add_user)
            self.db.commit()
            self.db.close()
            #处理成功后，返回成功码"0"及成功信息"ok"
            logger.debug("RegistHandle: regist successfully")
            http_response(self, ERROR_CODE['0'], 0)
```

users_views.py 完整代码如下：

```
#! /usr/bin/python3
# -*- coding:utf-8 -*-

import tornado.web
import sys
```

```
from tornado.escape import json_decode
import logging
from logging.handlers import
TimedRotatingFileHandler
from datetime import datetime


#从commons中导入http_response方法
from common.commons import (
    http_response,
)

#从配置文件中导入错误码
from conf.base import (
    ERROR_CODE,
)

from models import (
    Users
)


########## Configure logging #############
logFilePath = "log/users/users.log"
logger = logging.getLogger("Users")
logger.setLevel(logging.DEBUG)
handler = TimedRotatingFileHandler(logFilePath,
                                   when="D",
                                   interval=1,

backupCount=30)
formatter = logging.Formatter('%(asctime)s \
%(filename)s[line:%(lineno)d] %(levelname)s %
```

```
(message)s',)
handler.suffix = "%Y%m%d"
handler.setFormatter(formatter)
logger.addHandler(handler)


class RegistHandle(tornado.web.RequestHandler):
    """handle /user/regist request
    :param phone: users sign up phone
    :param password: users sign up password
    :param code: users sign up code, must six
digital code
    """

    @property
    def db(self):
        return self.application.db

    def post(self):
        try:
            #获取入参
            args = json_decode(self.request.body)
            phone = args['phone']
            password = args['password']
            verify_code = args['code']
        except:
            #获取入参失败时，抛出错误码及错误信息
            logger.info("RegistHandle: request
argument incorrect")
            http_response(self,
ERROR_CODE['1001'], 1001)
            return
```

```
        ex_user =
self.db.query(Users).filter_by(phone=phone).first
()
        if ex_user:
            #如果手机号已存在，返回用户已注册信息
            http_response(self,
ERROR_CODE['1002'], 1002)
            self.db.close()
            return
        else:
            #用户不存在，数据库表中插入用户信息
            logger.debug("RegistHandle: insert
db, user: %s" %phone)
            create_time =
datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            add_user = Users(phone, password,
create_time)
            self.db.add(add_user)
            self.db.commit()
            self.db.close()
            #处理成功后，返回成功码“0”及成功信息“ok”
            logger.debug("RegistHandle: regist
successfully")
            http_response(self, ERROR_CODE['0'],
0)
```

增加错误码处理

修改 base.py，增加错误码 1002：

```
"1002": "用户已注册，请直接登录",
```

```python
#! /usr/bin/python3
# -*- coding:utf-8 -*-

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
engine = create_engine('mysql://root:pwd@demo@localhost:3306/demo?charset=utf8', encoding="utf8", echo=False)
BaseDB = declarative_base()

ERROR_CODE = {
    "0": "ok",
    #Users error code
    "1001": "入参非法",
    "1002": "用户已注册，请直接登录",
}
```

结果检查

上面的几大步骤，从配置数据库，到代码指定数据库，再到将用户信息写入数据库，我们已完成了数据库部分代码的编写，下面执行 main.py 文件，查看是否运行正常。

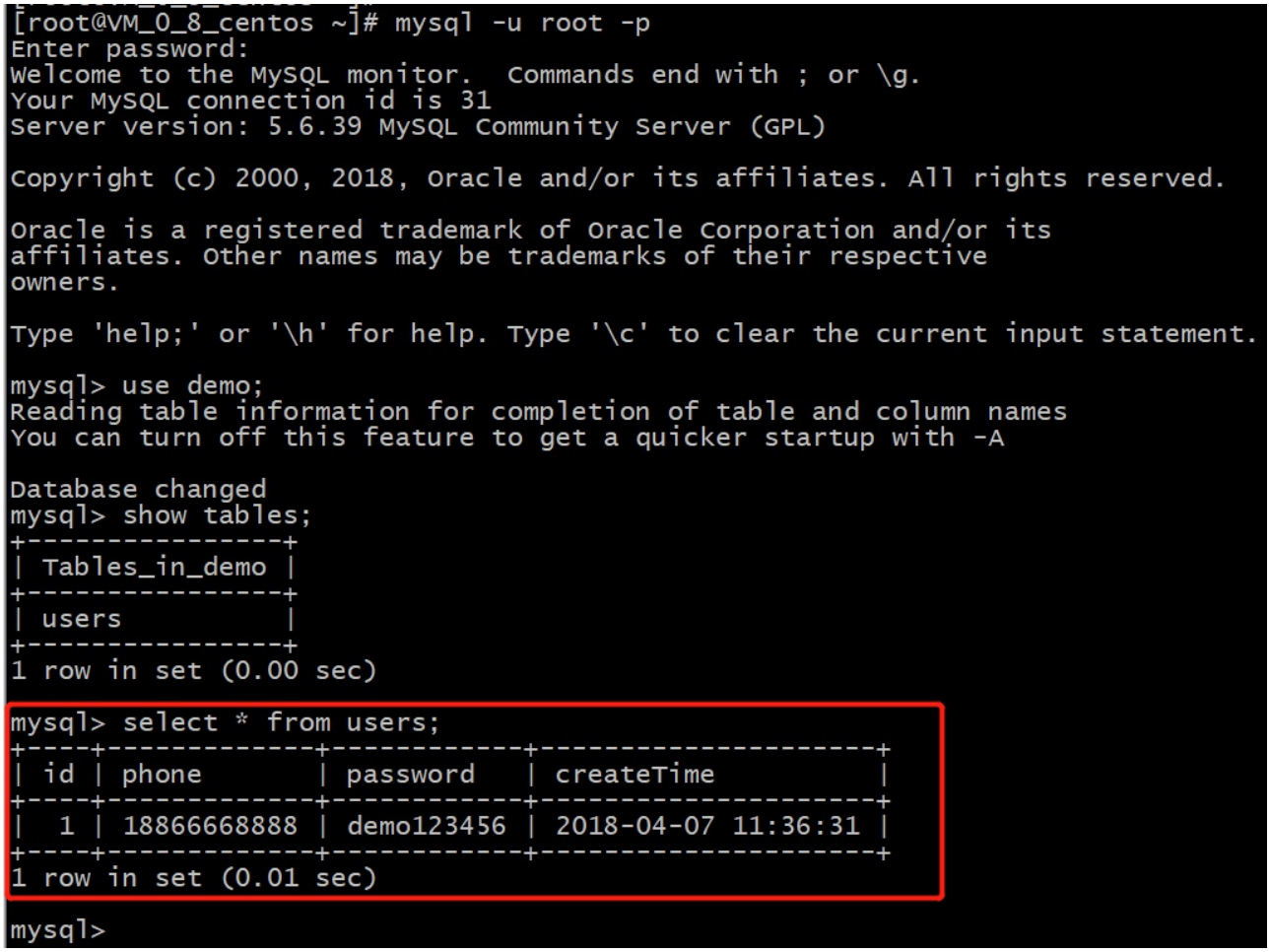
```
[root@VM_0_8_centos demo]#
[root@VM_0_8_centos demo]# ls
common  conf  log  main.py  models.py  models.pyc  static  templates  views
[root@VM_0_8_centos demo]# ./main.py
Tornado server is ready for service
```

HTTP 发包模拟器再次请求注册信息

查看控制台

```
Tornado server is ready for service
[D 180407 11:36:31 users_views:76] RegistHandle: insert db, user: 18866668888
[D 180407 11:36:31 users_views:83] RegistHandle: regist successfully
[I 180407 11:36:31 web:2106] 200 POST /users/regist (101.105.54.156) 38.87ms
```

查看数据库



在 HTTP 发包模拟器上再次点击注册

可以看到，服务器端返回的错误信息提示该用户已注册。

代码下载

到目前为止，服务器端代码如下：
[demo8 (https://github.com/Jawish185/demo8.git)](https://github.com/Jawish185/demo8.git)

小结

至此，我们已完成了数据库的写入，加上前两节的逻辑处理和 log 处理，客户端与服务器端的第一条消息请求交互已完成。这里只是使用到了 SQLAlchemy 很有限的功能，SQLAlchemy 具有很强大的功能，感兴趣的同学可以访问 [SQLAlchemy 官网 (http://docs.sqlalchemy.org/en/latest/)](http://docs.sqlalchemy.org/en/latest/)学习。