

# 第一次数据请求，服务器接收用户注册信息

本小节将是我们编写服务器端代码的开始。现在假设有这样一个App（见下图），用户需要通过该界面提交注册信息。服务器端在接收到客户端的注册请求后，返回注册成功信息，并将该用户写入数据库表用户信息中。



## 注册

手机号 请输入手机号码

密码 请输入密码

验证码 请输入验证码

获取验证码

注册

# 客户端模拟

考虑到本小册讲解的是服务器端，这里不作 App 端的介绍，我们将使用 HTTP 发包工具来模拟上面的 App 注册信息的提交。

HTTP 发包工具：[Getman \(https://getman.cn/\)](https://getman.cn/)

约定服务器端 HTTP server 的端口号为 8000，服务器端和客户端定义的请求是 /users/regist，那么完整的 URL 为 `http://150.109.33.132:8000/users/regist?`（请用自己的云虚拟机 IP 替换其中的 IP）。

参数为手机号（phone）、密码（password）及验证码（code），参数放入 HTTP 的 body 中，具体为：  
`{"phone": "18866668888", "password": "demo123456", "code": "123456"}`

注：

1. 确保服务器端 8000 端口已放通；
2. 在实际的项目中，密码不会明文的传输，一般会在客户端先使用 md5 进行加密，服务器端存储的也是加密后的密码字符串。本小册作为学习示例，将使用明文讲解。

发包器模拟如下：

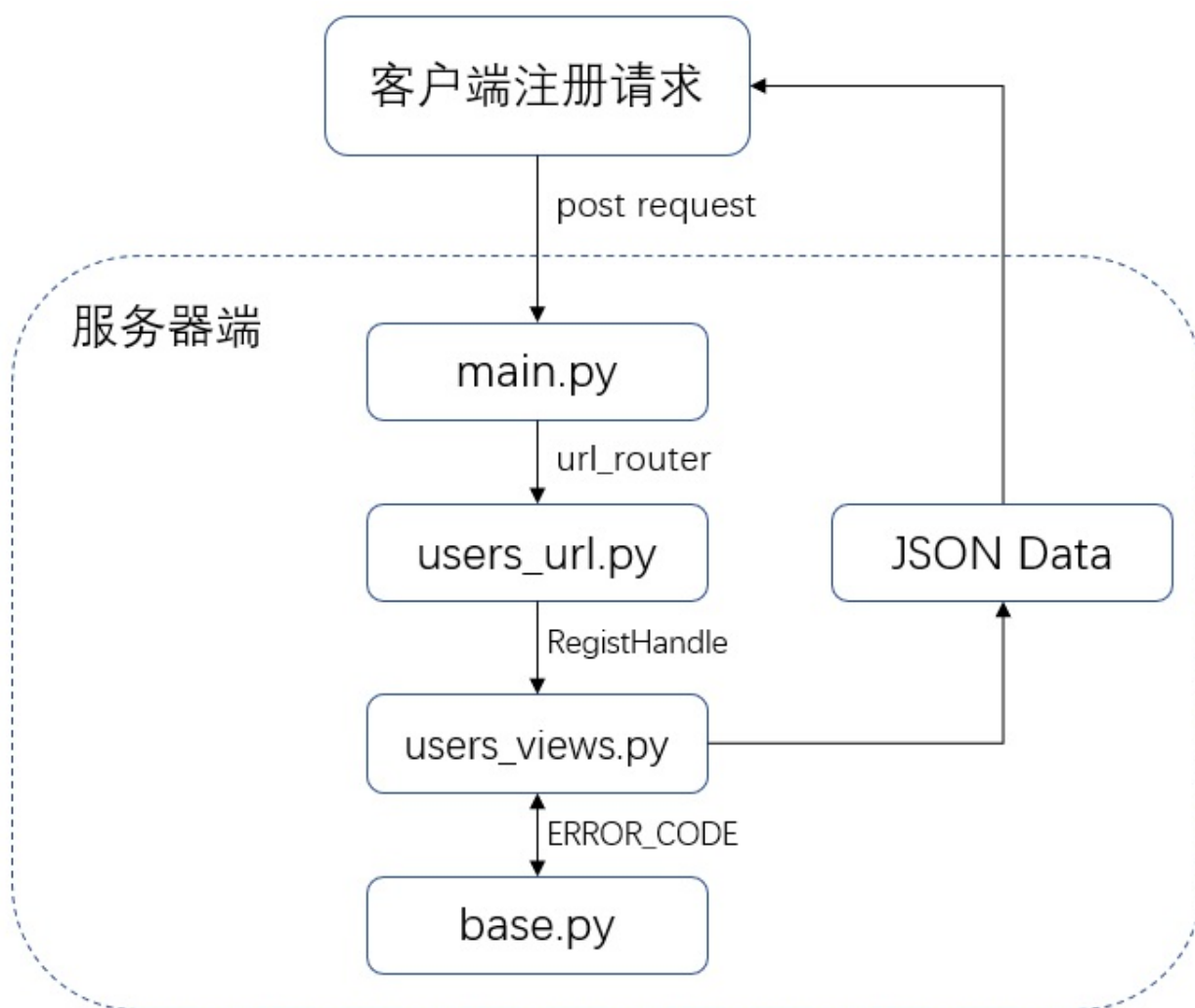


The screenshot shows the Getman HTTP client interface. At the top, there is a dropdown menu set to 'POST' and a text field containing the URL 'http://150.109.33.132:8000/users/regist?'. Below this, the 'Request' section is visible, with tabs for 'Body', 'Header', and 'Content-Type'. The 'Body' tab is selected, and the text field below it contains the JSON payload: `{"phone": "18866668888", "password": "demo123456", "code": "123456"}`. Red boxes highlight the POST method, the URL, and the JSON body in the original image.

客户端的请求至此已初步完成，现在，服务器端接收到客户端这个请求后，将如何处理呢？

## 服务器端处理

### 调用逻辑



客户端以 POST 的方式，发送注册请求至服务器端，请求进入服务器端的 main.py 后，将调用 url\_router 转发到 users\_url.py 中，在 users\_urls.py 中，对应的 URL 将调用 users\_views.py 的 RegistHandle 类，RegistHandle 为真正的代码处理逻辑，在校验用户信息正确的情况下，返回 JSON 格式的注册成功信息给客户端。

## 编写服务器端入口函数

main.py 是 Tornado 作为 HTTP 服务器的统一入口，根据前面的约定，Tornado 对外服务的端口号为 8000。

```
#!/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

class Application(tornado.web.Application):
    def __init__(self):
        #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,
            **settings)
```

```
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()
```

保存 main.py 代码后，在服务器端运行此段代码

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

此时再次点击 HTTP 发包模拟器发送注册信息

URL: http://150.109.33.132:8000/users/regist?

入

参: {"phone": "18866668888", "password": "demo123456",

再次查看服务器端

```
[root@VM_0_8_centos demo]# ls  
common conf log main.py models.py static templates views  
[root@VM_0_8_centos demo]# ./main.py  
Tornado server is ready for service  
  
[w 180407 08:29:33 web:2106] 404 POST /user/regist (101.105.54.156) 1.33ms
```

此条打印说明，客户端的 HTTP 请求已到达服务器，服务器接收成功但处理失败了，原因为找不到路径 /users/regist。下面在服务器端编写针对 /users/regist 的处理代码。

## 编写路由转发

首先，服务器端从 main.py 收到客户端的请求后，需要将其转发给对应的处理模块。进入 common 目录，创建 url\_router.py 文件

```
[root@VM_0_8_centos demo]# ls
common conf log main.py models.py static templates views
[root@VM_0_8_centos demo]# cd common/
[root@VM_0_8_centos common]# touch url_router.py
[root@VM_0_8_centos common]# ls
url_router.py
[root@VM_0_8_centos common]#
```

在 url\_router.py 中输入如下代码。

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

from __future__ import unicode_literals
from importlib import import_module

def include(module):
    '''根据传入的字符串，调用相应的模块,如 module 为字符串
    regist 时,
    调用views.users.users_views.RegistHandle 模块'''
    res = import_module(module)
    urls = getattr(res, 'urls', res)
    return urls

def url_wrapper(urls):
    '''拼接请求 url，调用对应的模块，如拼接 users 和
    regist 成 url /users/regist,
    调用 views.users.users_views.RegistHandle 模块'''
    wrapper_list = []
```

```

for url in urls:
    path, handles = url
    if isinstance(handles, (tuple, list)):
        for handle in handles:
            #分离获取字符串（如regist）和调用类（如
views.users.users_views.RegistHandle）
            pattern, handle_class = handle
            #拼接url，新的url调用模块
            wrap = ('{0}{1}'.format(path,
pattern), handle_class)
            wrapper_list.append(wrap)
        else:
            wrapper_list.append((path, handles))
    return wrapper_list

```

接下来修改 main.py，调用 url\_router.py 将用户请求的路径转发给对应的请求模块。

增加如下几行，从 common 目录的 url\_router 导入所需函数（`from common.url_router import include, url_wrapper`），并在 Application 的类中，拼接转发路由。



```

1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # Author: demo
4  # Email: demo@demo.com
5  # Version: demo
6
7  import tornado.ioloop
8  import tornado.web
9  import os
10 import sys
11 import shutil
12 from tornado.options import define, options
13 from common.url_router import include, url_wrapper
14
15
16 reload(sys)
17 sys.setdefaultencoding("utf-8")
18
19
20 class Application(tornado.web.Application):
21     def __init__(self):
22         handlers = url_wrapper([
23             (r"/users/", include('views.users.users_urls'))
24         ])
25         settings = dict(
26             debug=True,
27             static_path=os.path.join(os.path.dirname(__file__), "static"),
28             template_path=os.path.join(os.path.dirname(__file__), "templates")
29         )
30         tornado.web.Application.__init__(self, handlers, **settings)
31
32
33
34 if __name__ == '__main__':
35     print "Tornado server is ready for service\r"
36     tornado.options.parse_command_line()
37     Application().listen(8000, xheaders=True)
38     tornado.ioloop.IOLoop.instance().start()
39

```

完成后的代码如下：

```

#!/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

class Application(tornado.web.Application):
    def __init__(self):
        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)

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()
```

至此，main.py 的路由转发已完成，接下来将编写真正的处理模块。

进入 views 目录，创建 users 目录，该目录将存放所有跟用户信息处理相关的代码。在该目录下，创建 users\_urls.py、users\_views.py。

```
[root@VM_0_8_centos demo]# cd views/
[root@VM_0_8_centos views]# mkdir users
[root@VM_0_8_centos views]# cd users/
[root@VM_0_8_centos users]# touch users_urls.py
[root@VM_0_8_centos users]# touch users_views.py
[root@VM_0_8_centos users]# ls
users_urls.py  users_views.py
[root@VM_0_8_centos users]#
```

其中，users\_urls.py 处理针对 users 相关的路由及调用类之间的路由，users\_views.py 为真正的逻辑处理。在 users\_urls.py 中输入如下代码：

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

from __future__ import unicode_literals
from .users_views import (
    RegistHandle
)

urls = [
    #从 /users/regist 过来的请求, 将调用 users_views
    里面的 RegistHandle 类
    (r'regist', RegistHandle)
]
```

在 users\_views.py 文件中, 输入如下代码:

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

import tornado.web
from tornado.escape import json_decode

# 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,
)
```

```

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
    """

    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
            http_response(self,
ERROR_CODE['1001'], 1001)
            return

        # 处理成功后，返回成功码“0”及成功信息“ok”
        http_response(self, ERROR_CODE['0'], 0)

```

在users\_views.py 中看到，我们从公共方法库（commons）中导入了方法，并从配置文件中导入了错误码定义。接下来编写 commons 及 base 配置文件。

进入 common 目录，并创建 commons.py 文件，在 commons.py 中输入如下代码：

```
[root@VM_0_8_centos demo]# cd common/  
[root@VM_0_8_centos common]# touch commons.py  
[root@VM_0_8_centos common]#
```

```
#!/usr/bin/python3  
# -*- coding:utf-8 -*-  
  
import json  
  
def http_response(self, msg, code):  
    self.write(json.dumps({"data": {"msg": msg,  
"code": code}}))  
  
if __name__ == "__main__":  
    http_response()
```

在 conf 目录下，创建 base.py 文件：

```
[root@VM_0_8_centos demo]# ls  
common conf log main.py models.py static templates views  
[root@VM_0_8_centos demo]#  
[root@VM_0_8_centos demo]#  
[root@VM_0_8_centos demo]# cd conf/  
[root@VM_0_8_centos conf]# touch base.py  
[root@VM_0_8_centos conf]# ls  
base.py  
[root@VM_0_8_centos conf]#
```

在 base.py 文件中，输入如下代码：

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

ERROR_CODE = {
    "0": "ok",
    #Users error code
    "1001": "入参非法"
}
```

至此，我们已经完成了基本的用户注册以及服务器端处理逻辑代码，重新运行 main.py，查看是否启动正常。

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

现在再从 HTTP 发包模拟器

 POST `http://150.109.33.132:8000/users/regist?`

### Request

Body

Header

Content-Type ▾

`{"phone":"18866668888","password":"demo123456","code":"123456"}`

### Response

Body

Header

```
{
  "data": {
    "msg": "ok",
    "code": 0
  }
}
```


此时看到返回的 JSON 消息已成功。

再次查看服务器端，此时控制台打印的 log 提示 HTTP 200，表示该条 URL 请求已正确处理并返回。

```
[root@VM_0_8_centos demo]# ls
common  conf  log  main.py  models.py  static  templates  views
[root@VM_0_8_centos demo]# ./main.py
Tornado server is ready for service
[I 180407 08:40:06 web:2106] 200 POST /users/regist (101.105.54.156) 0.59ms
```

假如此时 HTTP 发包模拟器入参少了 code 参数，将提示错误信息。



 POST ▼ http://150.109.33.132:8000/users/regist?

## Request

Body

Header

Content-Type ▼

```
{"phone":"18866668888","password":"demo123456"}
```

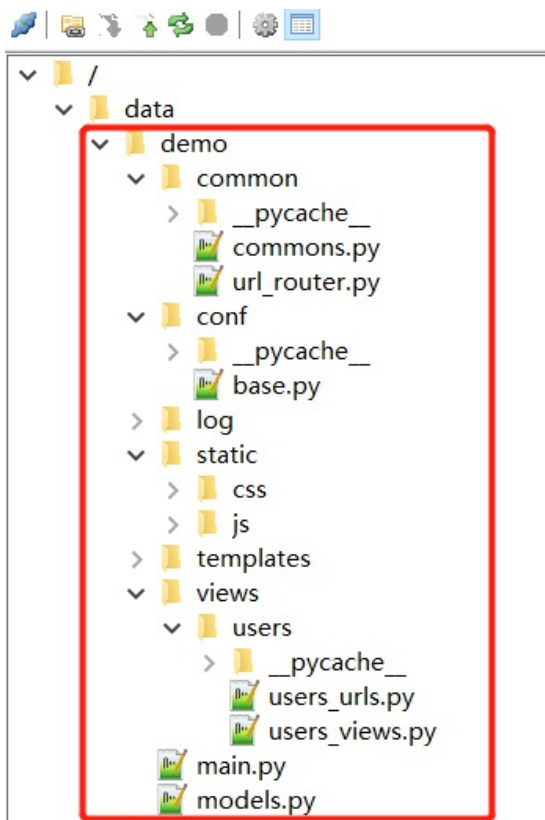
## Response

Body

Header

```
{
  "data": {
    "msg": "入参非法",
    "code": 1001
  }
}
```

至此，我们第一次客户端与服务器端的数据请求及回复已讲解完毕。完成后的目录结构及文件如下。



```
1  #!/usr/bin/python3
2  # -*- coding:utf-8 -*-
3  # Author: demo
4  # Email: demo@demo.com
5  # Version: demo
6
7  import tornado.ioloop
8  import tornado.web
9  import os
10 import sys
11 from tornado.options import define, options
12 from common.url_router import include, url_wrapper
13 from tornado.options import define, options
14
15
16 class Application(tornado.web.Application):
17     def __init__(self):
18         handlers = url_wrapper([
19             (r"/users/", include('views.users.users_urls'))
20         ])
21         #定义tornado服务器的配置项，如static/templates目录
22         settings = dict(
23             debug=True,
24             static_path=os.path.join(os.path.dirname(__file__),
25                                     'static'),
26             template_path=os.path.join(os.path.dirname(__file__),
27                                       'templates'),
28         )
29         tornado.web.Application.__init__(self, handlers, **settings)
30
31 if __name__ == '__main__':
32     print ("Tornado server is ready for service\r")
```

## 代码下载

到目前为止，服务器端代码如下：

[demo6 \(https://github.com/Jawish185/demo6.git\)](https://github.com/Jawish185/demo6.git)

## 小结

本小节讲解了客户端与服务器端的第一次数据请求及回复。代码比较简单，重点在于理解其中的 URL 路由转发，以达到触类旁通的效果。代码还有很多待完善的地方，如增加 log 管理，进一步抽象类和方法等。下一小节，我们将为代码加入 log 管理。