1.虚拟环境

1.1 虚拟环境的创建

```
mkvirtualenv tornado_py3 -p python3
```

1.2 安装tornado

```
pip install tornado
```

1.3 虚拟环境其他的操作

```
# 虚拟环境
mkvirtualenv  # 创建虚拟环境
rmvirtualenv  # 删除虚拟环境
workon  # 进入虚拟环境、查看所有虚拟环境
deactivate  # 退出虚拟环境

# pip
pip install  # 安装依赖包
pip uninstall  # 卸载依赖包
pip list  # 查看已安装的依赖库
```

2. 搭建项目

```
# 导包
import tornado.ioloop
import tornado.web

# 类似django中视图
class MainHandler(tornado.web.RequestHandler):
        # 这里写请求的方式
    def get(self):

        self.write("Hello, world")

# 程序配置
def make_app():
        # 这里配置路由功能
    return tornado.web.Application([
        (r"/", MainHandler),
    ])

# 程序入口
if __name__ == "__main__":
        # 加载配置
    app = make_app()
    # 设置监听
    app.listen(8888)
    # 开启服务, ioloop 实际上是对 epoll 的封装
    tornado.ioloop.IOLoop.current().start()
```

3.完成请求的方式

```
# 类似django中视图
class MainHandler(tornado.web.RequestHandler):
    # 得到数据
    def get(self):
        # 这里就是返回的内容
        self.write("得到数据")
    # 添加新的数据
    def post(self):
        self.write("添加新的数据")

    # 修改数据
    def put(self):
        self.write("修改数据")

    # 删除数据
    def delete(self):
        self.write("删除数据")
```

4. 设置静态文件

```
# 程序配置
def make_app():
    return tornado.web.Application([
        (r"/books/", MainHandler),
    ],
        static_path="./static" # 配置静态文件夹路径
    )
```

5. 配置模板

5.1 加载模板配置

```
def make_app():
return tornado.web.Application([
    (r"/books/", MainHandler),
],
    static_path="./static", # 配置静态文件夹路径
    template_path = "./templates",# 配置模板路径
)
```

5.2 定义模板

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>{{ show }}</h1>
</body>
</html>
```

5.3 渲染模板

模板内容使用render返回内容

```
# 类似django中视图
class MainHandler(tornado.web.RequestHandler):
    # 得到数据
    def get(self):
        # 这里就是返回的内容
        self.render("index.html",show='显示内容')
```

6 加载前端给的数据

6.1 加载前端给个模板文件

6.2 加载前端给的前端静态资源

7. 数据库操作

7.1 数据库初始化

```
-- 创建数据库
create database book_manager charset=utf8;

-- 使用数据库
use book_manager;

-- 创建表
CREATE TABLE books(id int UNSIGNED PRIMARY KEY AUTO_INCREMENT ,btitle VARCHAR(30)
not NULL ,bauthor VARCHAR(30) NOT NULL ,bperson VARCHAR(30),bpub_date DATE NOT NUL
L ,bread INT UNSIGNED,bcomment INT UNSIGNED);


-- 插入数据
insert into books(btitle, bauthor, bperson, bpub_date, bread, bcomment) VALUES
('红楼梦','曹雪芹','宝玉','1980-5-1',12,34),
('西游记','施耐安','悟空','1986-7-24',36,50),
('水浒传','吴承恩','林冲','1995-12-24',20,80),
('三国演义','罗贯中','曹操','1980-5-1',58,24);
```

7.2 安装pymysql

```
pip install pymysql
```

8. 使用同步的方式操作数据

8.1 get请求

8.1.1get请求处理

```
class MainHandler(tornado.web.RequestHandler):
    # 得到数据
    # 异步方法
    def get(self):

        # 1. 从数据库得到数据
        # 1.1连接数据库
        # 创建Connection连接
        conn = connect(host='localhost', port=3306, database='book_manager', user=
'root', password='mysql', charset='utf8')
        # 获得Cursor对象
        cs1 = conn.cursor()

        # 1.2 执行查询的sql语句
        cs1.execute("select * from books;")
        # 得到数据库的数据
        data = cs1.fetchall()

        # 1.3 关闭
        cs1.close()
        conn.close()

        # 操作
        #for temp in data:
        #  print(temp)



        # 返回数据
        self.render('index.html',show_list = data)
```

8.1.2模板数据展示

```
     <!--这里动态添加数据-->
{% for temp in show_list%}
<tr>
    <td><input class="idInput" type="text" value="{{temp[0]}}"></td>
    <td><input type="text" value="{{temp[1]}}"></td>
    <td><input type="text" value="{{temp[2]}}"></td>
    <td><input type="text" value="{{temp[3]}}"></td>
    <td><input type="text" value="{{temp[4]}}"></td>
    <td><input type="text" value="{{temp[5]}}"></td>
    <td><input type="text" value="{{temp[6]}}"></td>
    <td class="del"><input type="button" value=" 删 除 "></td>
    <td class="update"><input type="button" value=" 修 改 "></td>
</tr>
{%end%}
```

8.2 post请求

```
def post(self):
    # 得到请求的数据
    # 使用元组
    params_list = list()
    params_list.append(self.get_argument('btitle'))
    params_list.append(self.get_argument('bauthor'))
    params_list.append(self.get_argument('bperson'))
    params_list.append(self.get_argument('bpub_date'))
    params_list.append(self.get_argument('bread'))
    params_list.append(self.get_argument('bcomment'))

    print(params_list)

    # 1. 从数据库得到数据
    # 1.1连接数据库
    # 创建Connection连接
    conn = connect(host='localhost', port=3306, database='book_manager', user='roo
t', password='mysql',
                   charset='utf8')
    # 获得Cursor对象
    cs1 = conn.cursor()

    # 1.2 执行查询的sql语句
    cs1.execute("insert into books(btitle,bauthor,bperson,bpub_date,bread,bcomment
) values(%s,%s,%s,%s,%s,%s)",
                params_list)
    # 得到数据库的数据
    conn.commit()

    # 1.3 关闭
    cs1.close()
    conn.close()

    self.write({"data": "success"})
```

8.3put请求

```
def put(self):
    # 得到数据
    body_data = self.request.body.decode("utf-8")

    # 解析成字典
    params_dict = json.loads(body_data)

    # 1. 从数据库得到数据
    # 1.1连接数据库
    # 创建Connection连接
    conn = connect(host='localhost', port=3306, database='book_manager', user='roo
t', password='mysql',
                   charset='utf8')
    # 获得Cursor对象
    cs1 = conn.cursor()

    # btitle, bauthor, bperson, bpub_date, bread, bcomment
    # 1.2 执行查询的sql语句

    cs1.execute(
        "update books set btitle = %(btitle)s,bauthor = %(bauthor)s,bperson = %(bp
erson)s,bpub_date =%(bpub_date)s,bread = %(bread)s,bcomment = %(bcomment)s where i
d = %(id)s",
        params_dict
        )
    # 得到数据库的数据
    conn.commit()

    # 1.3 关闭
    cs1.close()
    conn.close()

    self.write({"data":"success"})
```

8.4 delete 请求

```
def delete(self):
        # 得到数据
        body_data = self.request.body.decode("utf-8")

        # 解析成字典
        params_dict = json.loads(body_data)

        # 1. 从数据库得到数据
        # 1.1连接数据库
        # 创建Connection连接
        conn = connect(host='localhost', port=3306, database='book_manager', user=
'root', password='mysql',
                       charset='utf8')
        # 获得Cursor对象
        cs1 = conn.cursor()

        # 1.2 执行查询的sql语句
        cs1.execute("delete from books where id = %(id)s", params_dict)

        conn.commit()

        # 1.3 关闭
        cs1.close()
        conn.close()

        self.write({"data":"success"})
```

9. 异步请求改造

```
class MainHandler(tornado.web.RequestHandler):
# 得到数据
# 异步方法
async def get(self):
    print('get请求')
    # 等待3秒
    await asyncio.sleep(3)

    # 这里就是返回的内容
    self.render("index.html",show='显示内容')
```

10异步

10.1 安装模块

```
pip3 install aiomysql
```

把所有请求改成异步请求

只需要方法前加async 再耗时返回时使用await返回

```
class MainHandler(tornado.web.RequestHandler):
# 得到数据
# 异步方法
async def get(self):
    # 1. 从数据库得到数据
    # 1.1连接数据库
    # 创建Connection连接
    conn = await aiomysql.connect(host='localhost', port=3306, db='book_manager',
user='root', password='mysql',
                      charset='utf8')
    # 获得Cursor对象
    cs1 =  await conn.cursor()

    # 1.2 执行查询的sql语句
    await cs1.execute("select * from books;")
    # 得到数据库的数据
    data = await cs1.fetchall()

    # 1.3 关闭
    await cs1.close()
    conn.close()

    # 操作
    # for temp in data:
    #   print(temp)


    self.render('index.html', show_list=data)

async def post(self):
    # 得到请求的数据
    # 使用元组
    params_list = list()
    params_list.append(self.get_argument('btitle'))
    params_list.append(self.get_argument('bauthor'))
    params_list.append(self.get_argument('bperson'))
    params_list.append(self.get_argument('bpub_date'))
    params_list.append(self.get_argument('bread'))
    params_list.append(self.get_argument('bcomment'))
```

```
    print(params_list)

    # 1. 从数据库得到数据
    # 1.1连接数据库
    # 创建Connection连接
    conn = await aiomysql.connect(host='localhost', port=3306, db='book_manager',
user='root', password='mysql',
                      charset='utf8')
    # 获得Cursor对象
    cs1 = await  conn.cursor()

    # 1.2 执行查询的sql语句
    await cs1.execute("insert into books(btitle,bauthor,bperson,bpub_date,bread,bc
omment) values(%s,%s,%s,%s,%s,%s)",
                  params_list)
    # 得到数据库的数据
    await conn.commit()

    # 1.3 关闭
    await cs1.close()
    conn.close()

    self.write({"data": "success"})

async def put(self):
    # 得到数据
    body_data = self.request.body.decode("utf-8")

    # 解析成字典
    params_dict = json.loads(body_data)

    # 1. 从数据库得到数据
    # 1.1连接数据库
    # 创建Connection连接
    conn = await aiomysql.connect(host='localhost', port=3306, db='book_manager',
user='root', password='mysql',
                      charset='utf8')
    # 获得Cursor对象
    cs1 = await  conn.cursor()

    # btitle, bauthor, bperson, bpub_date, bread, bcomment
    # 1.2 执行查询的sql语句

    await cs1.execute(
        "update books set btitle = %(btitle)s,bauthor = %(bauthor)s,bperson = %(bp
erson)s,bpub_date =%(bpub_date)s,bread = %(bread)s,bcomment = %(bcomment)s where i
d = %(id)s",
        params_dict
```

```
    )
    # 得到数据库的数据
    await conn.commit()

    # 1.3 关闭
    await cs1.close()
    conn.close()

    self.write({"data": "success"})

async def delete(self):
    # 得到数据
    body_data = self.request.body.decode("utf-8")

    # 解析成字典
    params_dict = json.loads(body_data)

    # 1. 从数据库得到数据
    # 1.1连接数据库
    # 创建Connection连接
    conn = await aiomysql.connect(host='localhost', port=3306, db='book_manager',
user='root', password='mysql',
                      charset='utf8')
    # 获得Cursor对象
    cs1 = await  conn.cursor()

    # 1.2 执行查询的sql语句
    await cs1.execute("delete from books where id = %(id)s", params_dict)

    await conn.commit()

    # 1.3 关闭
    await cs1.close()
    conn.close()

    self.write({"data":"success"})
```