# 说明：

手册中有详细内容，详细内容查看手册

# 一、单文件开发流程

1、pip install tornado #安装

2、在指定目录创建文件manage.py

3、manage.py引入tornado

import tornado.web

import tornado.ioloop

import tornado.options #获取参数

1. 在main函数中编写

if \_\_name\_\_ == “\_\_main\_\_”:

tarnado.options.parse\_command\_line() #获取命令行参数

app = tornado.web.Application(

[(r“/”, IndexOne), #配置路由

(r”/one”, IndexTwo)] #配置的路由

)

app.listen(8000) #监听端口

tornado.ioloop.IOLoop,current(),start() #开启项目

1. 编写一个类，响应路由方法

class IndexOne(tronado.web.RequestHandler): #路由配置调用

def get(self) #对应Get请求

self.write(“这里是返回的数据”)

def post(self): # 对应http的post请求方式

self.write("Hello Itcast!")

# 系统类

tornado.web 基础web请求框架

|->RequestHandler 包含所有请求方法,当没有对应请求方式的成员方 | 法时，会返回“405: Method Not Allowed”错误。

|->Application 核心应用类，基础配置

|->start() 开启 **单进程模式**

|->listen() 监听端口 **单进程模式**

tornado.ioloop 核心循环模块--主要是程序运行

|->current() 当前线程的实例

|->start() 开启当前线程的实例

tornado.httpserver 服务器模块 **多进程注意端口**

|->HTTPServer( web应用 ) 实例一个应用

|->bind() 绑定端口 **多进程**

|->start( num ) num代表开启几个进程 **多进程**

tornado.options 配置项

# 自定义目录结构

/-----------

|->manage.py #入口

|->url.py #路由

|->appliction #应用，可多个目录

| |->index.py #应用类

| |->father.py #公共类

|->statics

|->expand #扩展

| |->session.py #redis实现session的扩展

| |->mysql.py #pymysql连接mysql

|->templates

# 定义路由

**路由文件的引入，根据路由文件位置变化，建议与入口文件在同级目录**

1. **新建路由文件url.py**
2. **url.py**

#引入视图

from appliction import index

urls = [

#定义路由

(r"/", index.IndexHandler)

]

1. **manage.py**

#程序入口文件

from url import urls # 导入路由

app = tornado.web.Application(urls) #加载路由

# 获取请求参数

只有form才能通过get\_argument等场所获取，其他数据通过get\_body\_argument获取

json

self.request.headers.get(“content-type”) .startswith(“application/json”)

返回真

json\_data = json.loads(self.request.body)

上传

from-data:<multipart/form-data>

self.request.files # 这是一个列表

self.request.files[‘key’][0][‘body’] # 这是上传的文件

路由匹配方式

(r”/one/two”, indexHandler)

def post(self, one, two):

**特殊一：**

key1=value1&key2=value2

HTTP报文头Header中的"Content-Type"为：

============

application/x-www-form-urlencoded

multipart/form-data

=============

原因 -> 以get方式传值，请求头标注为POST；无法获取（GET/POST）

**特殊二：**

对于请求体数据为json或xml的。

**--------------------------------------**

1. **GET**

def get(self):

**self.get\_query\_argument(name, default=””, strip=True)**

#获取key为name的值，strip去掉空格

**self.get\_query\_arguments(name, strip=True)**

#获取多个值

1. **POST**

def post(self):

**get\_body\_argument(name, default=””, strip=True)**

#获取key为name的值，strip去掉空格

**get\_body\_arguments(name, strip=True)**

#获取多个值

#获取url地址中key为greeting的值，没有为hello

self.get\_argument('greeting', 'Hello') #获取的数据

self.wirte(“sss”) #返回的数据

1. **输出错误**

self.write\_error(500)

1. 获取所有

self.get\_argument(name, default=””, strip=True)

self.get\_arguments(name, strip=True)

# 使用session

Tornado没有定义session，通过redis实现

1、pip install redis

1. 安装redis
2. 使用单例模式，防止重复连接
3. 使用单例模式，防止重复初始化\_\_init\_\_()
4. 使用tornado.options
5. 在要使用的方法中使用

------------------------------------

单例：

class SesActive(object):

\_\_interface = None

\_\_first\_init = True

def \_\_new\_\_(cls):

#单例模式

if cls.\_\_interface == None:

cls.\_\_interface = object.\_\_new\_\_(cls)

return cls.\_\_interface

else:

return cls.\_\_interface

def \_\_init\_\_(self):

"""

只初始化1次\_\_init\_\_初始化，连接方法

"""

if self.\_\_first\_init:

host = "127.0.0.1"

port = "6379"

self.\_\_redis = redis.StrictRedis(host, port)

SesActive.\_\_first\_init = False

# websocket

1. 前端轮询（有无数据，服务器都响应）
2. 长轮询（有数据服务器响应）
3. websocket

-->开始使用http报文(使用101状态码)

-->升级为websocket

1. 使用方法

websocket没有get、post等请求

websocket中有request有http的request属性

self.request.remte\_ip # 客户端IP

1. 引入/继承

from tornado.websocket import webSocketHandler

class IndexHandler(websocketHandler)

1. 创建类属性

ws\_opt = [] //类属性，保存连接用户，全局

1. 创建连接属性

def open( self ): # 方法名，固定

for user in self.ws\_opt:

ws\_opt.write\_message(“这是广播”)

# 将当前对象放入，因继承 websocketHandler，所有websocket方法 # 都可使用

self.ws\_opt.append( self )

1. 发送信息

def on\_message(self, msg):

1. 关闭

def on\_close(self):

self.ws\_opt.remove(self) # 清楚下线的websocket对象

1. 跨域名/跨源

def check\_origin(self)

return true # 允许跨源

# 项目开启

1、创建启动文件

**/server.py 服务文件**

import tornado.web

import tornado.ioloop

import tornado.options

import tornado.httpserver

import pymysql

import redis

from url import urls // 导入同级目录下的路由文件

from config import configs //导入配置文件

// 获取终端配置信息

from tornado.options import define, option

define(“port”, type=int, default=8000, help=””)

// 添加mysql、redis配置

class Application(tornado.web.Application）：

def \_\_init\_\_(self, \*args, \*\*kargs):

super(Application, self).\_\_init\_\_(\*args, \*\*kargs)

self.db = pymysql.Connect( //加载mysql

\*\*configs.mysql\_options[‘host’]

)

self.redis = redis.StrictRedis( //加载redis

\*\*configs.redis\_options[‘host’] // 解包

)

// 主方法

def main():

options.logging = “waring” //日志等级

options.log\_file\_prefix = configs.log\_file

tornado.options.parse\_command\_line() //读取配置参数

app=tornado.web.Application({

urls, \*\*configs

})

http\_server = tornado.httpserver.HTTPServer(app)

http\_server.listen( options.port )

tornado.ioloop.IOLoop.current().start()

if \_\_name\_\_ == “\_\_main\_\_”:

main()

**/url.py 路由文件**

from handler.IndexHandler import IndexHandler

from handler.BaseHandler import StaticFileHandler

urls =[

(r”/”,IndexHandler),

//静态文件

(r”/(.\*)”, StaticFileHandler, dict(

path=os.path.join(

os.path.dirname(\_\_FILE\_\_),”html”,default\_filename=”index.html”)

]

**/config.py 配置文件**

import os

settings = {

“static\_path”:os.path.join(

os.path.dirname(\_\_FILE\_\_),”static”),

“template\_path”:os.path.join(

os.path.dirname(\_\_FILE\_\_),”template”),

“debug”:True,

“cookie\_secret”:”安全字符串uuid”,

“xsrf\_cookie”:”True”

}

mysql\_options = dick{

host=”127.0.0.1”,

database=”home”

}

redis\_options = dick{

host=”127.0.0.1”,

port=”6369”

}

log\_file = “路径” // 日志位置

session\_expires\_seconds = 86400

**/handler/BaseHandler.py**

**from**

class BaseHandler(RequestHandler):

@property

def db(self):

return self.application.db

@property

def redis(self):

return self.application.redis

def prepare(self):

self.xsrf\_token //手动触发一下 ,放大下面执行

if self.request.headers.get(“Content-Type”, “”)

.startwith(“application/json”):

self.json\_args = json.loads(self.request.body)

else:

self.json\_args = None

def write\_error(self):

def set\_default\_headers(self)

def initialize(self)

def on\_finish(self):

def get\_current\_user(self):

self.session = session(self)

return self.session.data

class StaticFileHandler(tornado.web.StaticFileHandler):

def \_\_init\_\_(self, \*args, \*\*kargs):

super(StaticFileHandler, self)

.\_\_init\_\_(\*args,\*\*kargs)

self.xsrf\_token

**/handler/IndexHandler.py**

from .BaseHandler import BaseHandler

class IndexHandler(BaseHandle):

def get(self):

self.write(“index”)

**/handler/\_\_init\_\_.py 创建包**

**日志功能**

import logging

logging.error/debug/info/waring/(“输出信息”)

tail -f 日志名

**/libs 别人的包，库**

**/utils 自己的包，库**

**/contants.py 常量文件**

# 验证码

描述：生成图片验证码

Url

method：get

@param codeID int 验证码编号

前端发送验证码编号，后端接收，生成验证码，保存编号，与文本

编号放在前端，是防止刷新二维码

import logging

from .BaseHandler import BaseHandler

class ImageCodeHandler(Basehandler):

def get(self):

code\_id = self.get\_argument(“codeID”)

pre\_code\_id = self.get\_argument(“pcodeID”)

if pre\_code\_id:

try:

self.redis.delect(“code\_id”)

except Exception as e:

logging.error(e) # redis错误

// 调用验证码类

// 存放redis中缓存

// 返回前端

手机验证码

1. 参数要判断手机号格式

r”1\d{10}

# session

可以保存在数据库，缓存，内存中，文件

放到内存中，开启多个进程，进程独享空间，内存中

import uuid

import loggin

import json

import config

class Session (object):

def \_\_init\_\_(self, request\_handler):

self.request\_handler = request\_handler

self.session\_id = self.request\_handler.get\_secure\_cookie(“session”)

if not self.session\_id:

# 用户第一次访问，生成一个ID

self.sessin\_id = uuid.uuid4().get\_hex() //唯一

self.data = {}

else:

# 获取session

try:

data = self.redis.get(“sess\_%s” % session\_id)

except Exception e:

loggin.error(e)

self.data = {}

if not data:

self.data = {}

else:

self.data = json.loads(data)

def save(self):

json\_data = json.dump(self.data) # 序列化

try:

self.redis.setex(“sess\_%s” % json\_data, config.session\_expires, json\_data)

except Exception e:

logging.error(e)

raise Exception(“save session failed”)

def clear(self):

self.request\_handler.clear\_cookie(“session\_id”)

try:

self.redis.delete(“sess\_%s” % self.session\_id)

except Exception as e:

logging.error(e)

# 模板

## 设置模板

application(

template\_path=”路径”

)

## 使用变量标注位置

**index.html**

{{static\_url(“css/index”)}}/xxx/js

## 设置render

/controller/indexHandler.py

indexHandler(self):

self.render(“index.html”) #加载模板

## 使用变量

**indexHandler.py**

index(self):

self.render(“index.html”,{“a”=11,”b”=22})

**index.html**

<h1>{{a}}</h1>{{b}}

## 5、控制静态文件handler

1. from tornado.web import StaticFileHandler

# 操作数据库

/manage.py

class Application(tornado.web.Application):

def \_\_init\_\_(self, \*args, \*\*kargs):

super(Application.self).\_\_init\_\_(\*args, \*\*args)

# 连接数据库方法

self.db = pymysql(

host="127.0.0.1"

)

# 用户登录验证

/controller/index.py

1. 添加装饰

@tornado.web.authenticated

index(self):

定义get\_current\_usr方法

def get\_current\_usr(self):

< @tornado.web.authenticated会自动调用本方法，返回真，验证成功，返回假验证失败>

1. 定义登陆失败后的路由

<application>

login\_url=”/login”

<url.py> 路由中设置

(r“/login”, loginHandler) # 自定义跳转控制器

# 开启cookie

## 前置设置

**/Application.py**

cookie\_secret = “uuid生成的字符串”

xsrf\_cookie = true # 开启

## 使用模板cookie

1. **前置设置**

**/Application.py**

cookie\_secret = “uuid生成的字符串”

xsrf\_cookie = true # 开启

1. **/static/index.html**

{%module xsrf\_form\_html() %}

## ajax / 不使用模板

set\_secure\_cookie

get\_secure\_cookie

**（1）通用设置  
/Application.py**

cookie\_secret = “uuid生成的字符串”

xsrf\_cookie = true # 开启

**（2）/controller/indexHandler.py**

def get(self):

self.xsrf\_token

**（3）application**

class staticFileHandler(tornado.web.StaticFileHandler):

def \_\_init\_\_(self, \*args, \*\*kargs):

super(staticFileHandler.self).\_\_init\_\_(\*args, \*\*kargs)

self.xsrf.token