# 说明：

手册中有详细内容，详细内容查看手册

# 一、基础

## 安装

pip install tornado #安装

## 目录

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

|-> manage.py / server.py #入口

|-> url.py #路由

|-> appliction #应用，可多个目录

| |-> index.py #应用类

| |-> base.py #公共类

|-> statics #静态文件

|-> lib #别人的包，库

|-> utils #自己的扩展

| |-> session.py #redis实现session的扩展

| |-> mysql.py #pymysql连接mysql

|-> templates #模板

## 导入

在包里导入

from application.BaseHandler import BaseHandler

from .BaseHandler import BaseHandler

导入非包的当前文件

from BaseHandler import BaseHandler

# 原理

1. ioloop是tornado的核心
2. ioloop封装了操作管理epoll的工作
3. 当ioloop实例启动时，ioloop将服务器监听的socket添加到epoll容器中，然后循环等待epoll返回可处理的socket
4. 当有客户端发起连接后，ioloop从epoll容器中拿到了服务器监听的socket，并调用服务器处理该监听socket的方法，接收连接请求，并将新的与客户端连接的socket添加到epoll容器中，然后，循环等待epoll返回可处理的socket
5. 当客户端发来请求数据后，ioloop从epoll中拿到了接收数据的socket，并调用服务器处理实例处理该传输的 socket方法，从socket中读取出http报文，解析后通过apllaction的实例方法，进行路由分发，实例化具体的requestHandler，执行其中的http方法，生成相应数据并打包报文写入缓冲区中
6. 当与客户端对应的socket可写时，ioloop从中拿到对应可写的socket，将缓冲区中对应的响应报文数据，写入到socket传回客户端，完成请求处理。

# aplication

tornado.options.parse\_command\_line() //读取配置参数, 还会输出请求信息，像django一样

app = tornado.web.Application({ // 主应用

urls, \*\*configs

})

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

http\_server.listen( 9000 ) //监听端口

tornado.ioloop.IOLoop.current().start() // 主循环

# 四、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、url.py**

from appliction import index #引入控制器

urls = [

(r"/", index.IndexHandler) #定义路由

（r”/(?P<subject>.+)/(?P<date>\d+)”, index.IndexHandler)

]

**2、manage.py**

from url import urls # 导入路由

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

1. **inde.py**

def index(self, subject, date): # 由路由传递参数的接收

# 五、接收参数

**条件1**：url：www.xx.com/a?key1=value1&key2=value2

**条件2**：HTTP报文头Header中的"Content-Type"为以下任意一种

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

application/x-www-form-urlencoded #POST方式

multipart/form-data #POST方式

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

tornado解析是会根据header确定**值**的位置，因Header标注为POST，所以会从body中取值

## 1、POST

def post(self):

#获取key为name的值，strip去掉空格

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

#获取多个值

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

## 2、GET

def get(self):

#获取key为name的值，strip去掉空格

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

#获取多个值

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

## 3、POST与GET全获取

获取url / form 表单传来的值，如有相同的变量名，**获取最后一个**

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

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

## 4、上传

发送请求的格式：<multipart/form-data>

接收方法：

self.request.files # 这是一个列表

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

**通过 get\_argument也可获取**

## 5、json

发送请求的格式：<application/json>

self.request.headers.get(“content-type”).startswith(“application/json”) #判断格式为json

json\_data = json.loads(self.request.body) #json序列化加接收

## 6、RequestHandler.request

对象存储了关于请求的相关信息，具体属性有：

**method：**HTTP的请求方式，如GET或POST;

**host**：被请求的主机名；

**uri：**请求的完整资源标示，包括路径和查询字符串；

**path：**请求的路径部分；

**query:**请求的查询字符串部分；

**version**:使用的HTTP版本；

**headers:**请求的协议头，是类字典型的对象，支持关键字索引的方式获取特定协议头信息， 例如：request.headers["Content-Type"]

**body:**请求体数据；

**remote\_ip:**客户端的IP地址；

**files:**用户上传的文件，为字典类型，型如：

# 项目开启

## 1、**/server.py | /manage.py服务文件**

1. **导入包**

import tornado.web

import tornado.ioloop

# import tornado.options

import tornado.httpserver

import pymysql

import redis

1. **导入配置文件**

**在 包 里导入，需要使用 from .url import urls (同级）**

from url import urls // 导入同级目录下的路由文件

from config import setting //导入配置文件

from config import mysql\_options//导入配置文件

from config import redis\_options //导入配置文件

// 导入option，主要用于获取 命令行 参数

from tornado.options import define, option

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

1. **添加mysql，redis**

class **Application**(tornado.web.Application）：

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

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

self.db = pymysql.Connect( //连接mysql

\*\*mysql\_options

)

self.redis = redis.StrictRedis( //连接redis

\*\*redis\_options // 解包

)

**（4）主方法**

def main():

options.logging = configs.log\_level # 设置日志等级

options.log\_file\_prefix = configs.log\_file # 设置读取配置文件中的

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

app=Application(

urls, \*\*settings

)

http\_server = tornado.httpserver.HTTPServer(app) #多进程

http\_server.listen( options.port ) #简写

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

**app.listen(8000)** 相当于：<单进程>

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

http\_server.bind(8000)

http\_server.start(1)

**http\_server.listen(8000)** 相当于：<单进程>

http\_server.bind(8000)

http\_server.start(1)

<多进程>

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

http\_server.bind(8000)

http\_server.start(0) # 少于或等于0，根据CPU核数开启进程

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

tornado.ioloop.IOLoop.current().start() **#主循环**

## 2、/url.py 路由文件

from handler.IndexHandler import IndexHandler

**#导入staticFileHandler类**

**from handler.BaseHandler import StaticFileHandler**

urls =[

(r”/”,**IndexHandler**),

(r”/(.\*)”, **StaticFileHandler**, dict( //静态文件

path=os.path.join(

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

]

## 3、/config.py 配置文件

import os

**settings** = {

“static\_path”:os.path.join( # 静态文件路径

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

“template\_path”:os.path.join( # 模板路径

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

“debug”:True, #bug

“cookie\_secret”:”安全字符串uuid”,

“xsrf\_cookie”:”True”

}

**mysql\_options** = dict(host=”127.0.0.1”, database=”home”)

**redis\_options** = dict(host=”127.0.0.1”, port=”6369”)

log\_file = “路径” // 日志位置

session\_expires\_seconds = 86400 #session过期时间

## 4、/handler/BaseHandler.py

**from tornado.web import RequestHandler**

**from util import SESSION #导入session文件**

class BaseHandler(**RequestHandler**):

**@property #作为属性，self.db即可调用**

def db(self):

return self.application.db

**@property #作为属性，self.redis即可调用**

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):

self.set\_header( # 设置请求头

“Content-Type”,”application/json;charset=UTF-8”)

def initialize(self)

def on\_finish(self):

def get\_current\_user(self):

**# 通过session判断用户是否为成功登录**

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

## 5、/handler/IndexHandler.py

from .BaseHandler import BaseHandler

class IndexHandler(BaseHandle):

def get(self):

self.write(“index”)

**/libs 别人的包，库**

**/utils 自己的包，库**

**/contants.py 常量文件**

## 6、日志功能

from tornado.options import option

options.logging = configs.log\_level # 设置日志等级

options.log\_file\_prefix = configs.log\_file # 设置读取配置文件中的

import logging

logging.error(“输出信息”)

logging.debug(“输出信息”)

logging.info(“输出信息”)

logging.waring(“输出信息”)

tail -f 日志名 //cmd中查看

# 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

# websocket

## 以前的方式

前端轮询（有无数据，服务器都响应）、长轮询（有数据服务器响应）、websocket

## 原理

-->开始使用http报文(使用101状态码)

-->升级为websocket

## 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，class Indexhandler

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): #固定方法 class Indexhandler

1. **关闭**

def on\_close(self): # class Indexhandler

self.ws\_opt.remove(self) # 用户关闭连接后从容器中移除用户

1. **跨域名/跨源**

def check\_origin(self) # class Indexhandler

return true # 允许跨源

# SESSION

通过操作redis实现session方法，

## 1、导入

import uuid

import loggin

import json

import config

## 2、类

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)

## 3、保存

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”)

## 4、删除

def clear(self):

#删除客户端的cookie

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

try:

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

except Exception as e:

logging.error(e)

## 5、使用

def get(self):

self.session = session(self)

# 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

# 用户登录验证

/controller/index.py

1. **添加装饰**

@tornado.web.authenticated

index(self):

**2、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>

**3、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

# 注册/登录

html

$.ajax({

contentType:”application/json”,

headers:{ // 请求头一定要携带

“X-XSRFTOKEN”:getCookie(“\_xsrf”) }

})

在用户第一次登录首页时，先判断登录状态ajax--check\_login

baseHandler,

prepare,

self.xsrf\_token

# 异步

## 1、异步回调

对用户访问不会有本质的速度提升，因为他会堵塞，但他提升并发数，可以同时处理多个请求

1. **导入模块**

from tornado.httpclient import AsyncHTTPClient

1. **新建客户端**

**@tornado.web.asynchronous** #不关闭连接，等待异步执行完成

def get(self):

clint = AsyncHTTPClient() #建立异步客户端

clint.fetch(“url地址”，callback=self.on\_response） #发送异步请求

1. **定义回调函数**

def on\_response(self, resp): # 回调函数

# 程序处理代码

# resp是url返回的数据数据, self由fetch传递的参数

self.finish() # 完成请求，终止

## 协程回调

***--> 不复用-----------------------------***

1. **导入模块**

import tornado.gen

1. **新建异步客户端**

@tornado.gen.coroutine

def get(self):

client = AsynHTTPClint()

resp = yield client.fetch(“URL 地址”)

# 正常处理

***--> 可以复用------------------------------***

1. **导入模块**

import tornado.gen

1. **新建异步客户端**

@tornado.gen.coroutine

def get(self):

res = yield self.one(“1.1.1.1”)

self.write(res)

@tornado.gen.coroutine

def one(self,ip):

client = AsynHTTPClint() # 异步客户端

resp = yield client.fetch(“URL 地址”)

# return resp #python2.7会出错

raise tornado.gen.Return(resp) # 此处需要注意

## 3、并行协程

## 4、使用HTTPRequest

req = tornado.HTTPCline.HTTPRequest(

url=url,method=post,body=body

);

clint.fetch(req)