目录

[框架简介 2](#_Toc26888346)

[基本业务流程图 2](#_Toc26888347)

[项目目录 2](#_Toc26888348)

[任务队列的使用 4](#_Toc26888349)

[celery概况 4](#_Toc26888350)

[延时任务 6](#_Toc26888351)

[周期任务 7](#_Toc26888352)

[数据库的使用 8](#_Toc26888353)

[通过工具包db\_util操作数据库 8](#_Toc26888354)

[通过sqlalchemy操作数据库 8](#_Toc26888355)

[框架部署 9](#_Toc26888356)

[nginx配置 9](#_Toc26888357)

[gunicorn配置 11](#_Toc26888358)

[supervisord配置 11](#_Toc26888359)

[配置supervisor服务 12](#_Toc26888360)

# 框架简介

## 基本业务流程图

## 项目目录

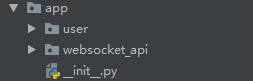
其中：

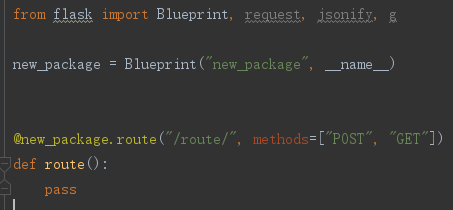
1. app为业务模块

user------------用户管理

websocket\_api---websocket接口

\_\_init\_\_----------业务应用和路由注册



在实际操作中，如需要新增业务模块则新建代码包，代码包下新建views文件（若数据连接采用orm模式还需新建model文件），views文件结合了controller和router功能，并在views文件中完成蓝图注册业务模块，如下图代码所示：

具体业务代码在router装饰器下新建的函数中编写代码。

1. config为框架基础配置

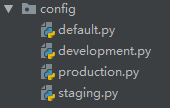
default---------默认配置

development---开发配置

production-----生产配置

staging--------测试配置

配置文件在app的\_\_init\_\_中完成加载



1. env为框架运行环境包，通过source命令激活env/bin/activate，即可切换至当前框架python环境。
2. instance为实例配置

config.py-----------业务相关配置

gunicorn\_config.py---网关服务器相关配置

1. logs为框架日志文件夹

日志的默认配置：

日志文件最大占用空间---20MB

日志文件备份数---------10

1. supervisor.d为项目进程管理配置文件夹

backend.conf-----后台进程管理配置

celery\_beat.conf---celery调度程序进程管理配置

celery\_flower------celery监控程序进程管理配置

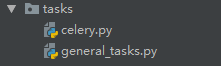
celery\_tasks-------celery任务队列进程管理配置

1. tasks为异步任务模块

celery---------celery应用程序

general\_tasks---任务集

使用celery作为任务队列调度工具，消息代理为redis。定时任务的新增方法非常简单，只需新增由celery.task装饰器装饰的函数，即可将任务注册到celery。



1. tets为测试模块
2. utils为项目工具包

celery\_sqlalchemy\_scheduler---周期任务辅助工具包

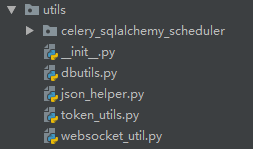
db\_util----------数据库工具包，实现了各类数据库的连接池和连接对象的创建

json\_helper---json编码工具包，解决部分数据无法正常通过json序列化的问题

token\_util---token工具包，实现了对token的加密方式以及存活时间等相关问题

websocket\_util------websocket工具包，实现了socket的创建和蓝图注册功能

除此以外，工具包均可完善，亦可新增。



1. manage为框架入口模块

manage相当于整个框架的开关，便于运行框架或被其他Web服务器调用。

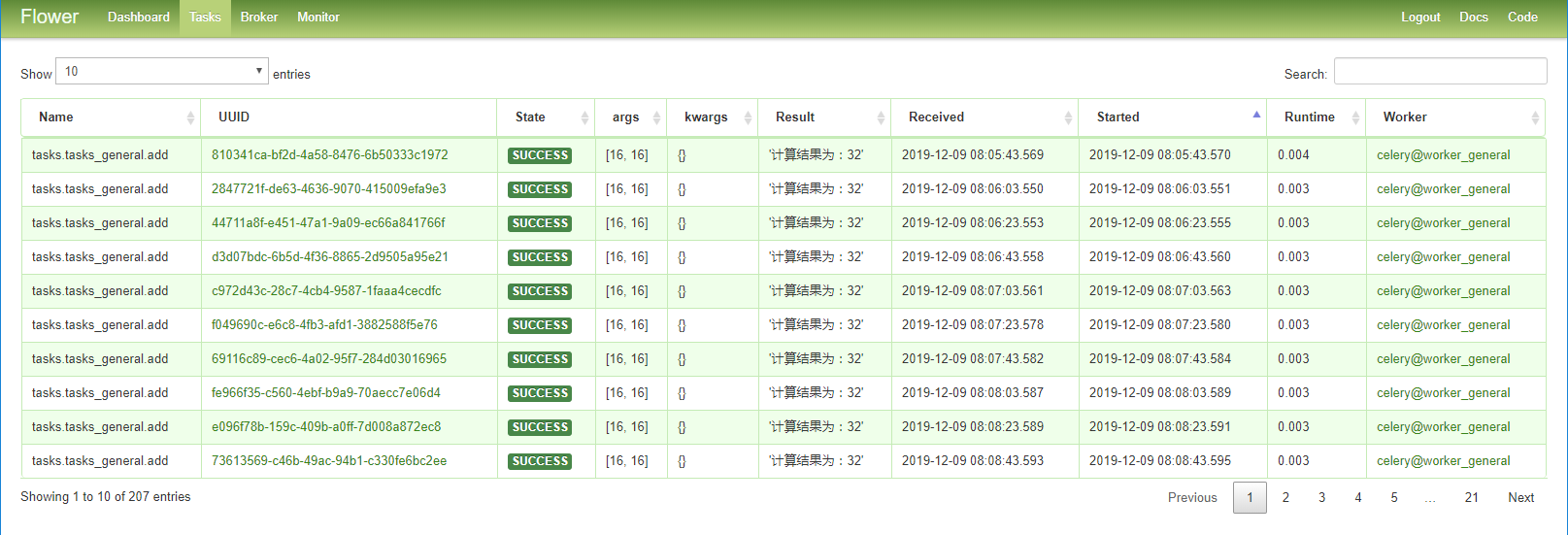
1. supervisord.conf

supervisord配置文件。

# 任务队列的使用

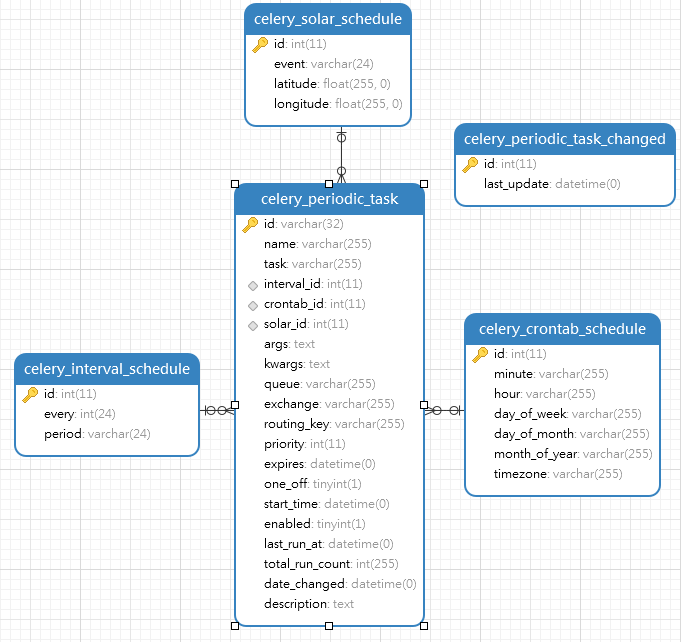
## celery概况

任务队列采用celery工具，项目文档地址<https://docs.celeryproject.org/en/latest/>，任务队列监控工具flower，如下图为flower监控web界面。



任务application在项目路径为takas.celery中实现，任务实体在项目路径为tasks.tasks\_general中实现，任务相关配置在项目路径为instance.config文件中。

Celery在supervisord托管下的启动文件有三个：celery\_beat、celery\_flower、celery\_tasks，如果项目中无需周期任务功能，则无需启动调度器celery\_beat。

若项目采用周期任务的功能，需另外配置数据库做调度信息存储，表模型见下图：

相关celery配置说明：

BROKER\_URL-----------------------代理url

CELERY\_RESULT\_BACKEN------------任务结果存储url

CELERY\_INCLUDE-------------------任务队列包含的任务实体列表

CELERY\_TIMEZONE------------------时区

CELERY\_QUEUES--------------------任务队列配置

CELERY\_ROUTES--------------------任务路由配置

CELERYD\_MAX\_TASKS\_PER\_CHILD-----每个worker执行的最大任务数

CELERYBEAT\_MAX\_LOOP\_INTERVAL---任务调度器beat休眠的最大秒数

CELERYBEAT\_SYNC\_EVERY-----------与数据库同步之前可以调用的周期任务数

CELERY\_TRACE\_APP-----------------是否抛出异常，生产环境关闭

CELERY\_ENABLE\_UTC----------------时间是否转换为UTC时间

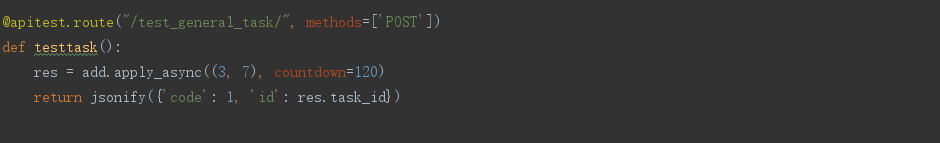
BEAT\_DB---------------------------周期任务beat同步数据库

beat\_dburi--------------------------周期任务beat同步数据库url

## 延时任务

基本操作：

1. 发布任务

选择任务实体函数，如tasks.tasks\_general.add，直接在业务代码中用apply\_async方法发布任务，如下图。任务发布之后celery会自动生成一个唯一识别码，在下图中通过res.task\_id获得识别码。

apply\_async所需参数：

args--------传递给任务函数的参数

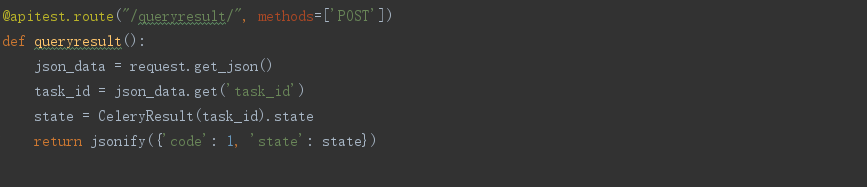
kwargs------传递给任务函数的键值参数

countdown---任务延时执行秒数

eta----------任务延时执行的具体时间和日期

expires------任务过期的具体时间和日期

1. 任务状态

通过任务唯一识别码，就可以在任意位置任意时间查看任务的执行状态，如下图借助CeleryResult类实现异步获取任务执行状态。

Celery内置的任务状态：

PENDING---任务挂起状态

STARTED---任务启动状态

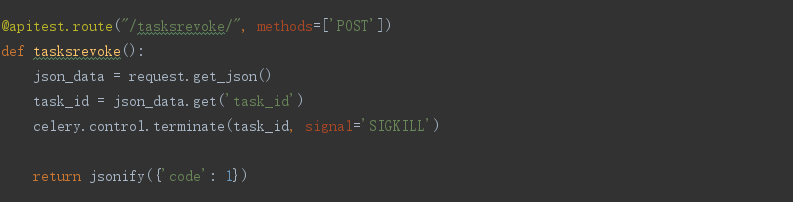
SUCCESS—任务成功执行状态

FAILURE---任务执行失败状态

RETRY---任务正在执行状态

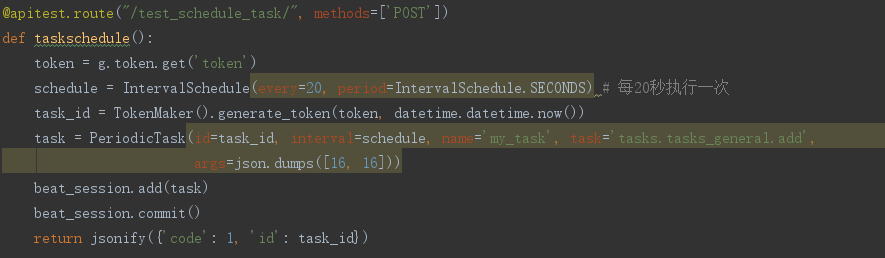
REVOKED---任务已被撤销状态

1. 任务撤销

通过方法celery.control.terminate即可根据任务唯一识别码撤销任务，具体代码示例见下图：

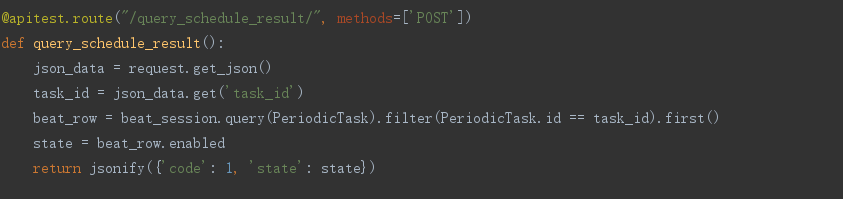
## 周期任务

1. 发布任务

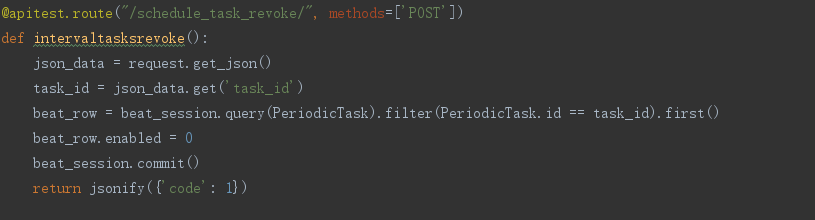
选择任务实体函数，具体代码示例见下图，如tasks.tasks\_general.add，首先创建schedule对象，定义任务执行的周期类型和频率；接着定义任务条目对象task，最后执行add添加至数据库，commit提交。下图示例中任务的唯一识别码通过TokenMaker类，输入token和时间戳参数生成的。

1. 任务状态

通过beat\_session查询PeriodciTask对应唯一任务识别码的enable字段，值为1，任务正常执行，值为0，任务已关闭。



3、任务撤销

通过beat\_session修改PeriodciTask对应唯一任务识别码的enable字段为0即可撤销任务。

# 数据库的使用

## 通过工具包db\_util操作数据库

1. 首先导入数据库连接对象。

from utils.db\_util import mysqlpool

1. 通过with代码块获取连接游标，退出with代码块自动提交事务并关闭连接游标。

conn = mysqlpool.get\_conn()

with conn.swich\_db(config.WOWRKSHEET01) as cursor:

pass

1. 也可在with代码块中提前提交事务。

conn.commit()

1. 若执行查询语句，成功返回查询结果集，失败返回0。
2. 若执行增删改语句，返回受影响行数。

## 通过sqlalchemy操作数据库

1. 在app的\_\_init\_\_中注册sqlalchemy。
2. 业务模块的model中通过类建立与数据库的映射关系。
3. 在业务配置文件中写入sqlalchemy相关配置。

|  |  |
| --- | --- |
| 配置名称 | 介绍 |
| SQLALCHEMY\_DATABASE\_URI | 将要被用于数据库链接的URI。例如: mysql://username:password@server/db mysql+pymysql://root:123456@localhost:3306/TTC |
| SQLALCHEMY\_BINDS | 一个将会绑定多种数据库的字典 |
| SQLALCHEMY\_ECHO | 如果设置为True，SQLAlchemy会将记录所有标准错误声明，这对调试非常有用。 |
| SQLALCHEMY\_RECORD\_QUERIES | 可以用于禁用或启用查询记录的显示 。 查询记录会自动的在调试或测试模式下进行 。 |
| SQLALCHEMY\_NATIVE\_UNICODE | 可以用来启用或禁用本地对 unicode 的支持。 |
| SQLALCHEMY\_POOL\_SIZE | 数据库池的大小。 默认与数据库引擎的值相同 (通常为 5) |
| SQLALCHEMY\_POOL\_TIMEOUT | 指定池的连接超时（以秒为单位）。 |
| SQLALCHEMY\_POOL\_RECYCLE | 自动循环连接的秒数。这是MySQL所必须的，默认情况下，闲置8小时后会删除连接。如果使用MySQL，SQLAlchemy会自动将其设置为2小时，一些后端可能使用不同的默认超时值。 |
| SQLALCHEMY\_MAX\_OVERFLOW | 控制连接池达到最大大小后还可以创建的连接数，当这些附加连接返回到连接池时，它们将会被断开并丢弃。 |
| SQLALCHEMY\_TRACK\_MODIFICATIONS | 如果设置为True，Flask-SQLAlchemy将跟踪对对象的修改，并发出信号。默认值为None，他可以启用跟踪功能，但会发出警告，表明它在将来会被默认禁用。这需要额外的内存，如果不需要，应该禁用。 |

# 框架部署

## nginx配置

server {

listen {port};

server\_name localhost;

charset utf-8;

location / {

proxy\_pass http://127.0.0.1:{port}; # 监听的IP端口

proxy\_connect\_timeout 600;

proxy\_send\_timeout 600;

proxy\_read\_timeout 1800;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Connection "Upgrade";

proxy\_set\_header Host $host:$server\_port;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header REMOTE-HOST $remote\_addr;

add\_header Access-Control-Allow-Origin $http\_origin always;

add\_header Access-Control-Allow-Credentials true always;

add\_header Access-Control-Allow-Methods 'GET,POST,PUT,DELETE,OPTIONS' always;

add\_header Access-Control-Allow-Headers 'Authorization,X-Requested-With,Content-Type,Origin,Accept,Credentials,token' always;

}

location /websocket {

proxy\_pass http://127.0.0.1:{port}/websocket; # 监听的IP端口

proxy\_connect\_timeout 600;

proxy\_send\_timeout 600;

proxy\_read\_timeout 1800;

proxy\_http\_version 1.1;

proxy\_buffering off;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Origin "";

proxy\_set\_header Connection "Upgrade";

proxy\_set\_header Host $host:$server\_port;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header REMOTE-HOST $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

add\_header Access-Control-Allow-Origin $http\_origin always;

add\_header Access-Control-Allow-Methods 'GET,POST,PUT,DELETE,OPTIONS' always;

add\_header Access-Control-Allow-Headers 'Authorization,X-Requested-With,Content-Type,Origin,Accept,Credentials,token' always;

}

}

## gunicorn配置

## server mechanics

chdir = '/usr/local/flask\_houtai/flaskhoutai'

pidfile = '%s/gunicorn.pid'%chdir

pythonpath = '%s/env/python/bin'%chdir

## server socket

bind = '127.0.0.1:833'

backlog = 1024

## worker 进程

workers = 7

worker\_class = 'utils.websocket\_util.worker'

# worker\_connections = 1000

max\_requests = 5000

max\_requests\_jitter = 1000

keepalive = 60

## security

limit\_request\_line = 4094

limit\_request\_fields = 100

limit\_request\_field\_size = 5120

## debugging

reload = True

## supervisord配置

[unix\_http\_server]

file=/usr/local/project/supervisor.sock

chmod=0700

[inet\_http\_server]

port=0.0.0.0:9001

username=zjx

password=123456

[supervisord]

logfile=/usr/local/project/logs/supervisord.log

logfile\_maxbytes=20MB

logfile\_backups=10

loglevel=info

pidfile=/usr/local/supervisord.pid

nodaemon=false

[rpcinterface:supervisor]

supervisor.rpcinterface\_factory=supervisor.rpcinterface:make\_main\_rpcinterface

[supervisorctl]

serverurl=unix:///usr/local/project/supervisor.sock

[include]

files = /usr/local/project//supervisor.d/\*.conf

## 配置supervisor服务

1. 进入/usr/lib/systemd/system目录，并创建supervisor.service文件

[Unit]

Description=supervisor

After=network.target syslog.target

[Service]

Type=forking

ExecStart=/usr/local/project/env/bin/supervisord -c /usr/local/ project /supervisord.conf

ExecReload=/usr/local/ project /env/bin/supervisorctl reload

ExecStop=/usr/local/ project /env/bin/supervisorctl shutdown

KillMode=process

Restart=on-failure

RestartSec=42s

[Install]

WantedBy=multi-user.target

1. 设置开机启动

命令行：

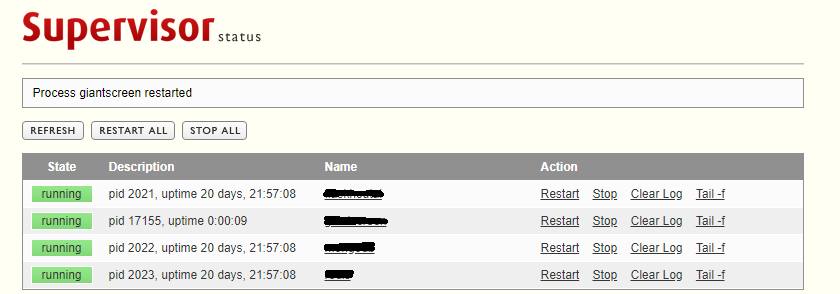
systemctl enable supervisor.service

chmod 766 supervisor.service

systemctl daemon-reload

1. 通过web端控制框架进程

浏览器打开框架supervisor配置中inet\_http\_server项的IP端口，以及用户名密码登陆，如下图所示：

这样就很方便的通过可视化的方式控制各个项目的进程存活，并在项目意外挂掉后自动帮助重启项目。