Gunicorn 部署 Flask-Apscheduler 之踩坑记录

作者：YYJeffrey
原文链接：https://ld246.com/article/1582878621206
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

flask-apscheduler 是一款flask的定时任务框架，其本质上是和 apscheduler 一样的，具体的使用操和其他的flask组件一样。在开发环境上定时任务跑起来很顺利，但是到了生产环境用Gunicorn部署时候出现了各种问题。

踩坑一：TimeZone offset does not match system offset

部署生产环境时，一上去就给我丢了一个大大的异常： “TimeZone offset does not match system ffset”，大致意思是说我的运行的时区和系统时区不匹配。

```
  File "/usr/local/lib/python3.6/site-packages/flask_apscheduler/scheduler.py", line 36, in __init__
    self._scheduler = scheduler or BackgroundScheduler()
  File "/usr/local/lib/python3.6/site-packages/apscheduler/schedulers/base.py", line 87, in __init__
    self.configure(gconfig, **options)
  File "/usr/local/lib/python3.6/site-packages/apscheduler/schedulers/base.py", line 126, in configure
    self._configure(config)
  File "/usr/local/lib/python3.6/site-packages/apscheduler/schedulers/background.py", line 29, in _configure
    super(BackgroundScheduler, self)._configure(config)
  File "/usr/local/lib/python3.6/site-packages/apscheduler/schedulers/base.py", line 697, in _configure
    self.timezone = astimezone(config.pop('timezone', None)) or get_localzone()
  File "/usr/local/lib/python3.6/site-packages/tzlocal/unix.py", line 165, in get_localzone
    _cache_tz = _get_localzone()
  File "/usr/local/lib/python3.6/site-packages/tzlocal/unix.py", line 90, in _get_localzone
    utils.assert_tz_offset(tz)
  File "/usr/local/lib/python3.6/site-packages/tzlocal/utils.py", line 38, in assert_tz_offset
    raise ValueError(msg)
ValueError: Timezone offset does not match system offset: 0 != 28800. Please, check your config files.
[2020-02-28 12:56:42 +0800] [1063] [INFO] Worker exiting (pid: 1063)
```

读了下flask-apscheduler的源码发现，他会读取SCHEDULER_TIMEZONE这个值，作为当前运行的区。

```
timezone = self.app.config.get('SCHEDULER_TIMEZONE')
if timezone:
    options['timezone'] = timezone
```

心想这简单，我在配置文件里把这个时区配置上应该就完事了。

```
# config.py
SCHEDULER_TIMEZONE = "Asia/Shanghai"
```

配置了时区为“Asia/Shanghai”后，发现依然报这个错，既然运行环境的时区正确了，那会不会我产环境的时区也有问题?

生产环境是docker容器，进入容器用date查看了他的当前时间，发现时间是和系统时间一致的，再看cat /etc/timezone的时区，发现这里出了问题，显示的是Etc/UTC，解决的思路是修改Dockerfile配置正确的时区，在Dockerfile中加入此行。

```
RUN echo "Asia/Shanghai" > /etc/timezone
```

修改之后重新运行，已经没有出现上述报错了，这个坑算是到这就排完了。

踩坑二：Flask-Apscheduler 多进程环境重复运行

排完第一个坑的时候项目可以启动运行了，这时当然要测试一下定时任务的分发是否正常，此时出现下面的情况。

有三条任务是一模一样的，Flask-Apscheduler 的 add_job() 方法需要传一个id值，这个值代表了每任务，且是不能重复的，不然会抛出

ConflictingIdError 异常，那为什么测试的时候会发出几条一模一样的任务?

其实这主要和生产环境使用Gunicorn部署有关，Gunicorn 可以指定一个worker 参数，指的是开启进程数，而每次开一个worker，都会启动一个scheduler，这就导致了这些定时任务是由不同的进程建的。

这个问题网上的解决方法五花八门，不如利用阻塞某个socket端口，来实现进程的创建时只创建单一个，或者给Gunicorn 加 --preload 参数等等，这时还是看官方的issue保险，发现还真有，按其中的决方式我尝试了几种。

1. 配置环境变量以控制开启的scheduler数量

通过读取环境变量的方式，判断SCHEDULER_LOCK的值是否为False，此时开启scheduler，并且修环境变量值。

```
if os.environ.get("SCHEDULER_LOCK") == "False":
    scheduler.start()
    os.environ["SCHEDULER_LOCK"] = "True"
```

我尝试了这种方法，显然并不行，通过环境变量共享多个进程的方式，是不太可取的。

2. 通过全局锁，控制scheduler只运行一次

首次创建进程时，会创建一个scheduler.lock文件，并加上非阻塞互斥锁，此时scheduler可以成功启，如果文件加锁失败抛出异常，则表示当前scheduler已经开启了，最后再注册一个退出事件，此时flask退出的话，就释放文件锁。

```
def register_scheduler():
    """
    注册定时任务
    """
    f = open("scheduler.lock", "wb")
    # noinspection PyBroadException
    try:
        fcntl.flock(f, fcntl.LOCK_EX | fcntl.LOCK_NB)
        scheduler.start()
    except:
        pass

    def unlock():
        fcntl.flock(f, fcntl.LOCK_UN)
        f.close()

    atexit.register(unlock)
```

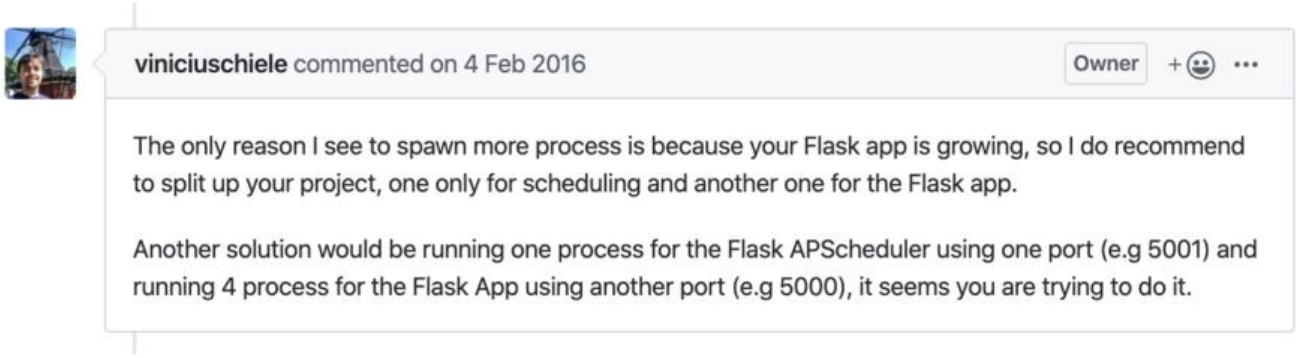
这个方法看似完美，其在运行开始时确实没有出现问题，但这还没完，且看下个坑。

踩坑三：集群环境下，Flask-Apscheduler 多进程环境重复行

第二个问题确实可以通过文件锁的方式解决，但是我的生产环境是集群环境，也就是有多个应用实例每个应用实例部署于docker容器之中，这样他们是互相隔离的，并通过Nginx做负载均衡。

此时再调用定时任务时，显然还会有问题，当定时任务调用接口打在同一个实例时，是没有问题的，第二次调用接口时就可以抛出异常并捕获，但对于不同的实例，当第三次，接口请求打在了另外的实上，此时按正常的业务逻辑，任务下发应该需要失败，但它依然成功了，也就是说上面通过文件全局的方式，并不适用于集群环境。

此时只能再次阅读其他issue，确实也有人遇到了此类问题，但我看到作者的建议其实是拆分业务。



拆分业务确实给了我不少提示，但我的定时任务的功能不算是一个微服务，如果独立成一个flask实例需要一点时间，此时我想到了这个问题的本质是由于多进程导致 Flask-Apscheduler 开启多个 scheduler 而造成的，那么我可以构建一个单进程的实例，并让特定的定时任务接口流量从这个集群的实例过可以了。

修改Gunicorn的配置文件，通过获取WORKERS_COUNT这个环境变量来判断需要开启几个进程，并进程数赋值给workers。

```
if os.environ.get("WORKERS_COUNT"):
    workers = int(os.environ.get("WORKERS_COUNT"))
```

```
else:
    workers = multiprocessing.cpu_count() * 2 + 1
```

在docker-compose.yml配置文件中指定其中一个实例的进程数为单进程。

```
environment:
    - WORKERS_COUNT=1
```

此时再通过Nginx的正则规则，将定时任务的接口路由到该实例就可以了，Nginx的匹配标识符可以考下表。

标识符	描述
=	**精确匹配：**用于标准uri前，要求请求字符和uri严格匹配。如果匹配成功就停止匹配，立即执行该location里面的请求。
~	**正则匹配：**用于正则uri前，表示uri里面包正则，并且区分大小写。
~*	**正则匹配：**用于正则uri前，表示uri里面含正则，不区分大小写。
^~	**非正则匹配；**用于标准uri前，nginx服务匹配到前缀最多的uri后就结束，该模式匹配成功后，不会使用正则匹配。
无	**普通匹配（最长字符匹配）；**与location序无关，是按照匹配的长短来取匹配结果。若完全匹配，就停止匹配。

到这里算是终于解决了这个问题，过程虽然艰辛，但也算颇有收获。