个人简介： wedo实验君, 数据分析师；热爱生活，热爱写作 微信公众号：wedo创客实验室

python任务调度利器-APScheduler

任务调度应用场景

所谓的任务调度是指安排任务的执行计划，即何时执行，怎么执行等。在现实项目中经常出现它们的身影； 特别是数据类项目，比如实时统计每5分钟网站的访问量，就需要每5分钟定时从日志数据分析访问量。

总结下任务调度应用场景：

- 离线作业调度： 按时间粒度执行某项任务
- 共享缓存更新： 定时刷新缓存，如redis缓存；不同进程间的共享数据

任务调度工具

- linux的crontab， 支持按照分钟/小时/天/月/周粒度，执行任务
- java的Quartz
- windows的任务计划

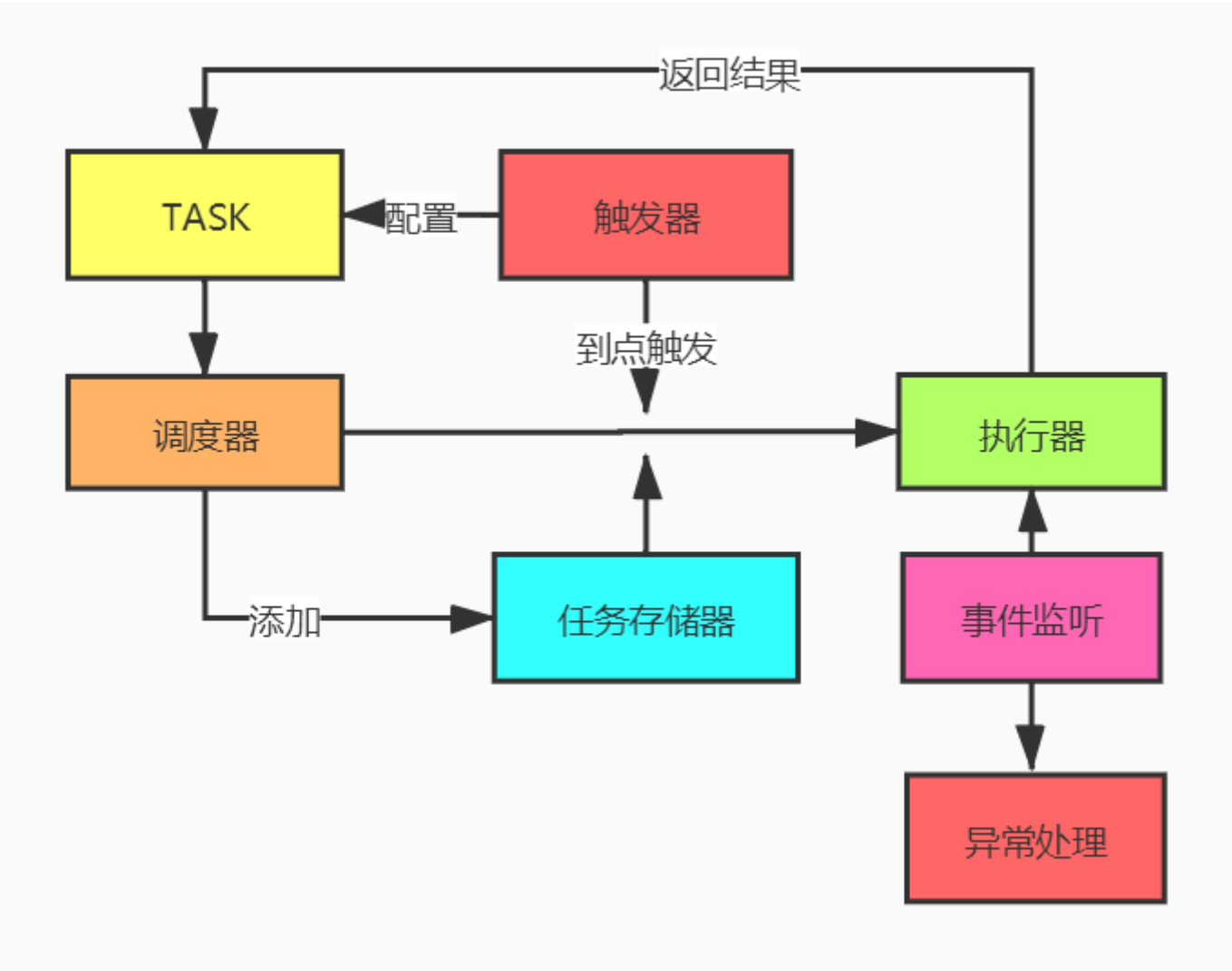
本文介绍的是python中的任务调度库，APScheduler（advance python scheduler ）。如果你了解Quartz的话，可以看出APScheduler是Quartz的python实现；APScheduler提供了基于时间，固定时间点和crontab方式的任务调用方案， 可以当作一个跨平台的调度工具来使用。

APScheduler

组件介绍

APScheduler由5个部分组成：触发器、调度器、任务存储器、执行器和任务事件。

- 任务job： 任务id和任务执行func
- 触发器triggers：确定任务何时开始执行
- 任务存储器job stores: 保存任务的状态
- 执行器executors：确定任务怎么执行
- 任务事件event：监控任务执行异常情况
- 调度器schedulers：串联任务的整个生命周期，添加编辑任务到任务存储器,在任务的执行时间到来时，把任务交给执行器执行返回结果；同时发出事件监听，监控任务事件 。



安装

```
pip install apscheduler
```

简单例子

```
from apscheduler.schedulers.background import BackgroundScheduler
from apscheduler.executors.pool import ThreadPoolExecutor, ProcessPoolExecutor
from apscheduler.jobstores.sqlalchemy import SQLAlchemyJobStore
from apscheduler.events import EVENT_JOB_EXECUTED, EVENT_JOB_ERROR
import logging
import datetime

# 任务执行函数
def job_func(job_id):
    print('job %s is runed at %s' % (job_id, datetime.datetime.now().strftime('%Y-
%m-%d %H:%M:%S')))

# 事件监听
def job_exception_listener(event):
```

```
    if event.exception:
        # todo： 异常处理，告警等
        print('The job crashed :(')
    else:
        print('The job worked :)')

# 日志
logging.basicConfig()
logging.getLogger('apscheduler').setLevel(logging.DEBUG)

# 定义一个后台任务非阻塞调度器
scheduler = BackgroundScheduler()
# 添加一个任务到内存中
# 触发器：trigger='interval' seconds=10 每10s触发执行一次
# 执行器：executor='default' 线程执行
# 任务存储器：jobstore='default' 默认内存存储
# 最大并发数：max_instances
scheduler.add_job(job_func, trigger='interval', args=[1], id='1', name='a test
job', max_instances=10, jobstore='default', executor='default', seconds=10)
# 设置任务监听
scheduler.add_listener(job_exception_listener, EVENT_JOB_EXECUTED |
EVENT_JOB_ERROR)

# 启动调度器
scheduler.start()
```

运行情况：

```
job 1 is runed at 2020-03-21 20:00:38
The job worked :)
job 1 is runed at 2020-03-21 20:00:48
The job worked :)
job 1 is runed at 2020-03-21 20:00:58
The job worked :)
```

触发器

触发器决定何时执行任务，APScheduler支持的触发器有3种

- trigger='interval'： 按固定时间周期执行，支持weeks, days, hours, minutes, seconds, 还可指定时间范围

  ```
  sched.add_job(job_function, 'interval', hours=2, start_date='2010-10-10
  09:30:00', end_date='2014-06-15 11:00:00')
  ```

- trigger='date': 固定时间，执行一次

```
sched.add_job(my_job, 'date', run_date=datetime(2009, 11, 6, 16, 30, 5),
args=['text'])
```

- trigger='cron': 支持crontab方式，执行任务
  - 参数：分钟/小时/天/月/周粒度，也可指定时间范围

```
year (int|str) – 4-digit year
month (int|str) – month (1-12)
day (int|str) – day of the (1-31)
week (int|str) – ISO week (1-53)
day_of_week (int|str) – number or name of weekday (0-6 or
mon,tue,wed,thu,fri,sat,sun)
hour (int|str) – hour (0-23)
minute (int|str) – minute (0-59)
second (int|str) – second (0-59)
start_date (datetime|str) – earliest possible date/time to trigger on
(inclusive)
end_date (datetime|str) – latest possible date/time to trigger on
(inclusive)
```

  - 例子

```
   # 星期一到星期五，5点30执行任务job_function，直到2014-05-30 00:00:00
   sched.add_job(job_function, 'cron', day_of_week='mon-fri', hour=5,
minute=30, end_date='2014-05-30')

   # 按照crontab格式执行， 格式为：分钟 小时 天 月 周，*表示所有
   # 5月到8月的1号到15号，0点0分执行任务job_function
   sched.add_job(job_function, CronTrigger.from_crontab('0 0 1-15 may-
aug *'))
```

执行器

执行器决定如何执行任务；APScheduler支持4种不同执行器，常用的有pool(线程/进程)和gevent(io多路复用，支持高并发)，默认为pool中线程池， 不同的执行器可以在调度器的配置中进行配置（见调度器）

- apscheduler.executors.asyncio： 同步io，阻塞
- apscheduler.executors.gevent：io多路复用，非阻塞
- apscheduler.executors.pool: 线程ThreadPoolExecutor和进程ProcessPoolExecutor
- apscheduler.executors.twisted： 基于事件驱动

任务存储器

任务存储器决定任务的保存方式， 默认存储在内存中（MemoryJobStore），重启后就没有了。APScheduler支持的任务存储器有：

- apscheduler.jobstores.memory： 内存
- apscheduler.jobstores.mongodb： 存储在mongodb
- apscheduler.jobstores.redis： 存储在redis
- apscheduler.jobstores.rethinkdb：存储在rethinkdb
- apscheduler.jobstores.sqlalchemy：支持sqlalchemy的数据库如mysql，sqlite等
- apscheduler.jobstores.zookeeper：zookeeper

不同的任务存储器可以在调度器的配置中进行配置（见调度器）

调度器

APScheduler支持的调度器方式如下，比较常用的为BlockingScheduler和BackgroundScheduler

- BlockingScheduler：适用于调度程序是进程中唯一运行的进程，调用start函数会阻塞当前线程，不能立即返回。
- BackgroundScheduler：适用于调度程序在应用程序的后台运行，调用start后主线程不会阻塞。
- AsyncIOScheduler：适用于使用了asyncio模块的应用程序。
- GeventScheduler：适用于使用gevent模块的应用程序。
- TwistedScheduler：适用于构建Twisted的应用程序。
- QtScheduler：适用于构建Qt的应用程序。

从前面的例子，我们可以看到，调度器可以操作任务（并为任务指定触发器、任务存储器和执行器）和监控任务。

```
scheduler.add_job(job_func, trigger='interval', args=[1], id='1', name='a test
job', max_instances=10, jobstore='default', executor='default', seconds=10)
```

我们来详细看下各个部分

- 调度器配置： 在add_job我们看到jobstore和executor都是default，APScheduler在定义调度器时可以指定不同的任务存储和执行器，以及初始的参数

```
from pytz import utc

from apscheduler.schedulers.background import BackgroundScheduler
from apscheduler.jobstores.mongodb import MongoDBJobStore
from apscheduler.jobstores.sqlalchemy import SQLAlchemyJobStore
from apscheduler.executors.pool import ThreadPoolExecutor,
ProcessPoolExecutor

# 通过dict方式执行不同的jobstores、executors和默认的参数
jobstores = {
    'mongo': MongoDBJobStore(),
    'default': SQLAlchemyJobStore(url='sqlite:///jobs.sqlite')
}
executors = {
    'default': ThreadPoolExecutor(20),
    'processpool': ProcessPoolExecutor(5)
```

```
}
job_defaults = {
    'coalesce': False,
    'max_instances': 3
}
# 定义调度器
scheduler = BackgroundScheduler(jobstores=jobstores, executors=executors,
job_defaults=job_defaults, timezone=utc)

def job_func(job_id):
    print('job %s is runed at %s' % (job_id,
datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')))
# 添加任务
scheduler.add_job(job_func, trigger='interval', args=[1], id='1', name='a
test job', jobstore='default', executor='processpool', seconds=10)
# 启动调度器
scheduler.start()
```

- 操作任务： 调度器可以增加，删除，暂停，恢复和修改任务。需要注意的是这里的操作只是对未执行的任务起作用，已经执行和正在执行的任务不受这些操作的影响。
    - add_job

      ```
      scheduler.add_job(job_func, trigger='interval', args=[1], id='1',
      name='a test job', max_instances=10, jobstore='default',
      executor='default', seconds=10)
      ```

    - remove_job: 通过任务唯一的id，删除的时候对应的任务存储器里记录也会删除

      ```
      scheduler.add_job(myfunc, 'interval', minutes=2, id='my_job_id')
      scheduler.remove_job('my_job_id')
      ```

    - Pausing and resuming jobs： 暂停和重启任务

      ```
      scheduler.add_job(myfunc, 'interval', minutes=2, id='my_job_id')
      scheduler.pause_job('my_job_id')
      scheduler.resume_job('my_job_id')
      ```

    - Modifying jobs： 修改任务的配置

      ```
      job = scheduler.add_job(myfunc, 'interval', minutes=2, id='my_job_id',
      max_instances=10)
      # 修改任务的属性
      job.modify(max_instances=6, name='Alternate name')
      ```

```
# 修改任务的触发器
scheduler.reschedule_job('my_job_id', trigger='cron', minute='*/5')
```

- 监控任务事件类型，比较常用的类型有：
    - EVENT_JOB_ERROR: 表示任务在执行过程的出现异常触发
    - EVENT_JOB_EXECUTED： 任务执行成功时
    - EVENT_JOB_MAX_INSTANCES： 调度器上执行的任务超过配置的参数时

```
scheduler.add_listener(job_exception_listener, EVENT_JOB_EXECUTED |
EVENT_JOB_ERROR)
```

> 参考文档：https://apscheduler.readthedocs.io/en/stable/userguide.html