Debug

Python任务调度模块 – APScheduler

2015年6月11日 by debugo · 22条评论

APScheduler是一个Python定时任务框架，使用起来十分方便。提供了基于日期、固定时间间隔以及crontab类型的任务，并且可以持久化任务、并以daemon方式运行应用。目前最新版本为3.0.x。
在APScheduler中有四个组件：
触发器**(trigger)**包含调度逻辑，每一个作业有它自己的触发器，用于决定接下来哪一个作业会运行。除了他们自己初始配置意外，触发器完全是无状态的。
作业存储**(job store)**存储被调度的作业，默认的作业存储是简单地把作业保存在内存中，其他的作业存储是将作业保存在数据库中。一个作业的数据讲在保存在持久化作业存储时被序列化，并在加载时被反序列化。调度器不能分享同一个作业存储。
执行器**(executor)**处理作业的运行，他们通常通过在作业中提交制定的可调用对象到一个线程或者进城池来进行。当作业完成时，执行器将会通知调度器。
调度器**(scheduler)**是其他的组成部分。你通常在应用只有一个调度器，应用的开发者通常不会直接处理作业存储、调度器和触发器，相反，调度器提供了处理这些的合适的接口。配置作业存储和执行器可以在调度器中完成，例如添加、修改和移除作业。
你需要选择合适的调度器，这取决于你的应用环境和你使用APScheduler的目的。通常最常用的两个：
– **BlockingScheduler**: 当调度器是你应用中唯一要运行的东西时使用。
– **BackgroundScheduler**: 当你不运行任何其他框架时使用，并希望调度器在你应用的后台执行。
安装APScheduler非常简单：

```
pip install apscheduler
```

选择合适的作业存储，你需要决定是否需要作业持久化。如果你总是在应用开始时重建job，你可以直接使用默认的作业存储（MemoryJobStore).但是如果你需要将你的作业持久化，以避免应用崩溃和调度器重启时，你可以根据你的应用环境来选择具体的作业存储。例如：使用Mongo或者SQLAlchemyJobStore （用于支持大多数RDBMS）
然而，调度器的选择通常是为你如果你使用上面的框架之一。然而，默认的ThreadPoolExecutor 通常用于大多数用途。如果你的工作负载中有较大的CPU密集型操作，你可以考虑用ProcessPoolExecutor来使用更多的CPU核。你也可以在同一时间使用两者，将进程池调度器作为第二执行器。

配置调度器

APScheduler提供了许多不同的方式来配置调度器，你可以使用一个配置字典或者作为参数关键字的方式传入。你也可以先创建调度器，再配置和添加作业，这样你可以在不同的环境中得到更大的灵活性。
下面是一个简单使用BlockingScheduler，并使用默认内存存储和默认执行器。(默认选项分别是MemoryJobStore和ThreadPoolExecutor，其中线程池的最大线程数为10)。配置完成后使用start()方法来启动。

```
1 from apscheduler.schedulers.blocking import BlockingScheduler
2 def my_job():
3     print 'hello world'
4
5 sched = BlockingScheduler()
6 sched.add_job(my_job, 'interval', seconds=5)
7 sched.start()
```

在运行程序5秒后，将会输出第一个Hello world。
下面进行一个更复杂的配置，使用两个作业存储和两个调度器。在这个配置中，作业将使用mongo作业存储，信息写入到MongoDB中。

```
from pymongo import MongoClient
from apscheduler.schedulers.blocking import BlockingScheduler
from apscheduler.jobstores.mongodb import MongoDBJobStore
from apscheduler.jobstores.memory import MemoryJobStore
from apscheduler.executors.pool import ThreadPoolExecutor, ProcessPoolExecutor


def my_job():
    print 'hello world'
host = '127.0.0.1'
port = 27017
client = MongoClient(host, port)

jobstores = {
    'mongo': MongoDBJobStore(collection='job', database='test', client=client),
    'default': MemoryJobStore()
}
executors = {
    'default': ThreadPoolExecutor(10),
    'processpool': ProcessPoolExecutor(3)
}
job_defaults = {
    'coalesce': False,
    'max_instances': 3
}
scheduler = BlockingScheduler(jobstores=jobstores, executors=executors, job_defaults=job_defa
scheduler.add_job(my_job, 'interval', seconds=5)

try:
    scheduler.start()
except SystemExit:
    client.close()
```

查询MongoDB可以看到作业的运行情况如下：

```
{
  "_id" : "55ca54ee4bb744f8a5ab08cc4319bc24",
  "next_run_time" : 1434017278.797,
  "job_state" : new BinData(0, "gAJ9cQEoVQRhcmdzcQIpVQhleGVjdXRvcnEDVQdkZWZhdWx0cQRVDW1heF9pb
}
```

操作作业

1. 添加作业

上面是通过add_job()来添加作业，另外还有一种方式是通过scheduled_job()修饰器来修饰函数。

```
@sched.scheduled_job('cron', id='my_job_id', day='last sun')

def some_decorated_task():
    print("I am printed at 00:00:00 on the last Sunday of every month!")
```

2. 移除作业

```
job = scheduler.add_job(myfunc, 'interval', minutes=2)
job.remove()
```

```
Same, using an explicit job ID:

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

3. 暂停和恢复作业

暂停作业:
– apscheduler.job.Job.pause()
– apscheduler.schedulers.base.BaseScheduler.pause_job()
恢复作业:
– apscheduler.job.Job.resume()
– apscheduler.schedulers.base.BaseScheduler.resume_job()

4. 获得**job**列表

获得调度作业的列表，可以使用get_jobs()来完成，它会返回所有的job实例。或者使用print_jobs()来输出所有格式化的作业列表。

5. 修改作业

```
def some_decorated_task():
    print("I am printed at 00:00:00 on the last Sunday of every month!")</pre>
```

6. 关闭调度器

默认情况下调度器会等待所有正在运行的作业完成后，关闭所有的调度器和作业存储。如果你不想等待，可以将wait选项设置为False。

```
scheduler.shutdown()
scheduler.shutdown(wait=False)
```

作业运行的控制

add_job的第二个参数是trigger，它管理着作业的调度方式。它可以为date, interval或者cron。对于不同的trigger，对应的参数也相同。

(1). cron定时调度

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)

timezone (datetime.tzinfo|str) – time zone to use for the date/time calculations (defaults to scheduler timezone)
和Linux的Crontab一样，它的值格式为：

Expression	Field	Description
*	any	Fire on every value
*/a	any	Fire every a values, starting from the minimum
a-b	any	Fire on any value within the a-b range (a must be smaller than b)
a-b/c	any	Fire every c values within the a-b range
xth y	day	Fire on the x -th occurrence of weekday y within the month
last x	day	Fire on the last occurrence of weekday x within the month
last	day	Fire on the last day within the month
x,y,z	any	Fire on any matching expression; can combine any number of any of the above expressions

几个例子如下：

```
chedules job_function to be run on the third Friday
f June, July, August, November and December at 00:00, 01:00, 02:00 and 03:00
ed.add_job(job_function, 'cron', month='6-8,11-12', day='3rd fri', hour='0-3')
uns from Monday to Friday at 5:30 (am) until 2014-05-30 00:00:00
ed.add_job(job_function, 'cron', day_of_week='mon-fri', hour=5, minute=30, end_date='2014-05-30')
```

(2). interval 间隔调度

它的参数如下：
weeks (int) – number of weeks to wait
days (int) – number of days to wait
hours (int) – number of hours to wait
minutes (int) – number of minutes to wait
seconds (int) – number of seconds to wait
start_date (datetime|str) – starting point for the interval calculation
end_date (datetime|str) – latest possible date/time to trigger on
timezone (datetime.tzinfo|str) – time zone to use for the date/time calculations
例子：

```
1 # Schedule job_function to be called every two hours
2 sched.add_job(job_function, 'interval', hours=2)
```

(3). date 定时调度

最基本的一种调度，作业只会执行一次。它的参数如下：
run_date (datetime|str) – the date/time to run the job at
timezone (datetime.tzinfo|str) – time zone for run_date if it doesn't have one already
例子：

```
1 # The job will be executed on November 6th, 2009
2 sched.add_job(my_job, 'date', run_date=date(2009, 11, 6), args=['text'])
```

```
# The job will be executed on November 6th, 2009 at 16:30:05
sched.add_job(my_job, 'date', run_date=datetime(2009, 11, 6, 16, 30, 5), args=['text'])
```

^^

Ref:
http://apscheduler.readthedocs.org/en/latest/modules/triggers
http://apscheduler.readthedocs.org/en/3.0/userguide.html

Posted in Python|R.
← MongoDB RS优先级设置
Flask-RESTful构建小型REST服务 →

22 条评论 最新 最早 最热

非洲白白
顶一个，很有用
2015年12月28日 回复 顶 转发

db浆糊

2015年12月28日 回复 顶 转发

vince
发
1月9日 回复 顶 转发

vince
暂停和恢复怎么做。
1月9日 回复 顶 转发

db浆糊
找下文档哦亲~~ http://apscheduler.readthedocs.org/en/latest/modules/triggers pause和resume~
1月22日 回复 顶 转发

lee
您好，请问在windows下可以持久定时运行吗？谢谢
1月11日 回复 顶 转发

db浆糊
当然可以。作为一个后台任务一直运行就行。
1月22日 回复 顶 转发

Karis Ankedy
apscheduler可以执行异步定时任务吗， 比如说两个任务在同一时刻运行的情况
1月15日 回复 顶 转发

db浆糊

两个job设置同一时间就行了

1月22日 回复 顶 转发

奋斗的小蚂蚁最老实

对多个任务每次都需要调用start()方法啊？

1月20日 回复 顶 转发

db浆糊

对于整个调度器来启动就行了。

1月22日 回复 顶 转发

文

我试了第二个例子，怎么mogodb中没有写入数据呢

3月1日 回复 顶 转发

Noisyguy

>>> from apscheduler.schedulers.blocking import BlockingScheduler
Traceback (most recent call last):
File "", line 1, in
File "/Library/Python/2.7/site-packages/apscheduler/__init__.py", line 3, in
version_info = tuple(int(x) if x.isdigit() else x for x in parsed_version.public.split('.'))
AttributeError: 'tuple' object has no attribute 'public'

mac EI Captain 环境，出现这个，怎么破，难道还要什么依赖吗

4月6日 回复 顶 转发

Noisyguy

解决了，setuptools too old, 更新到新版就可以了

4月6日 回复 顶 转发

lol

pip install setuptools --upgrade

5月12日 回复 顶 转发

Laul

hello，有没有遇到使用默认内存存储做后台任务执行时，申请的virt越来越大的情况?

5月24日 回复 顶 转发

刘震

请问 想 执行多次后停止 怎么操作

9月3日 回复 顶 转发

有课

9月21日 回复 顶 转发

uPxSz

全都到碗里来！美臀/丝袜/美熟女乱伦精品大合集！！！HTTp://uVU.cc/inRB

9月24日 回复 顶 转发

McrnM

全都到碗里来！美臀/丝袜/美熟女乱伦精品大合集！！！HTTp://uVU.cc/inRB

10月4日 回复 顶 转发

cJrzY

全都到碗里来！美臀/丝袜/美熟女乱伦精品大合集！！！【 v.ht/7UrM 】

10月9日 回复 顶 转发

kllpM

■■■■■无毒爽站【htTP://v.ht/aS8H】，在线爽，夫妇必备 ■■■■■■

11月6日 回复 顶 转发

社交帐号登录: 微信 微博 QQ 人人 更多»

说点什么吧...

发布

Debugo正在使用多说

- 搜索： 搜索
- 近期文章
  - C++自动类型推断和基于范围的for循环
  - R入门笔记2
  - Nginx重定向&静动资源分离
  - Flask-RESTful构建小型REST服务
  - Python任务调度模块 – APScheduler
  - MongoDB RS优先级设置
  - MongoDB的权限管理
  - Docker Compose—简化复杂应用的利器
  - 在Windows下构建Docker开发环境
  - Gitbucket—快速建立自己的Github
- 热评文章
  - 使用Python操作Redis

- 使用fio进行I/O性能测试
- Spark as a Service之JobServer部署
- Docker 1.5 - 容器运行监控和只读容器
- Sqoop常用操作

- 文章归档

选择月份 ▼

- 分类目录

- BigData (48)
  - Hadoop (21)
  - HBase|Hive (13)
  - Spark (11)
- Database (79)
  - MySQL (21)
  - NoSQL (26)
  - Oracle (36)
- Dev (59)
  - C|C++ (17)
  - JavaScript (10)
  - Java|Scala (19)
  - Python|R (13)
- Ops (90)
  - Linux (51)
  - Tools (27)
  - Virtualization (12)

- 友链

geewaza
Jlog
Bananalighter
粉丝日志

- 功能

- 登录
- 文章RSS
- 评论RSS
- WordPress.org