aiorq

发布 *v1*

wt

2022 年 04 月 08 日

Contents

Aiorq 是一个包含 asyncio 和 redis 的分布式任务队列，它从 arq 重写以进行改进，并包含 web 接口。

CHAPTER 1

快速而轻量的

aiorq 继承了 arq 库的基础功能（异步的）。

由 python3 的 asyncio 构建的，允许非阻塞作业排队和执行。可以使用 asyncio 池同时运行多个作业（可能有数百个）Tasks。延迟执行、轻松重试作业和悲观执行（见下文）意味着非常适合必须完成的关键作业。

目前 aiorq 只有大约 750 行，不会有太大变化。aiorq 被设计得更简单、更清晰、更强大。

CHAPTER 2

依赖

- `redis>=5.0`
- `aioredis>=2.0.0`

CHAPTER 3

安装

```
pip install aiorq
pip install aioredis
```

CHAPTER 4

用法

4.1 快速开始

```
# demo.py
# -*- coding: utf-8 -*-

import asyncio
import os
from aiorq import create_pool
from aiorq.connections import RedisSettings
from aiorq.cron import cron


async def say_hello(ctx, name) -> None:
    await asyncio.sleep(5)
    print(f"Hello {name}")


async def say_hi(ctx, name) -> None:
    await asyncio.sleep(3)
    print(f"Hi {name}")


async def startup(ctx):
```

(下页继续)

(续上页)

```
    print("starting... done")


async def shutdown(ctx):
    print("ending... done")


async def run_cron(ctx, time_='2021-11-16 10:26:05'):
    print(time_)


class WorkerSettings:
    redis_settings = RedisSettings(
        host=os.getenv("REDIS_HOST", "127.0.0.1"),
        port=os.getenv("REDIS_PORT", 6379),
        database=os.getenv("REDIS_DATABASE", 0),
        password=os.getenv("REDIS_PASSWORD", None)
    )

    functions = [say_hello, say_hi, run_cron]

    on_startup = startup
    on_shutdown = shutdown

    cron_jobs = [
        cron(coroutine=run_cron, name="x100", minute=40, second=50, keep_result_
↪forever=True)
    ]

    # allow_abort_jobs = True
    # worker_name = "ohuo"
    # queue_name = "ohuo"
```

以 demo.py 文件为例，文件中声明了 say_hello、say_hi、run_cron 方法、其中 run_cron 作为定时任务。

CHAPTER 5

启动 worker

```
> aiorq demo.WorkerSettings
15:08:50: Starting Queue: ohuo
15:08:50: Starting Worker: ohuo@04dce85c-1798-43eb-89d8-7c6d78919feb
15:08:50: Starting Functions: say_hello
15:08:50: redis_version=5.0.10 mem_usage=731.12K clients_connected=2 db_keys=9
starting...
```

CHAPTER 6

启动 Integration

```
> aiorq tasks.WorkerSettings server
INFO:     Started server process [4524]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
```

CHAPTER 7

任务

7.1 实时执行任务

有时您希望一项作业一次只运行一次（例如备份）或针对给定参数运行一次（例如为特定公司生成发票）。

aiorq 通过自定义作业 ID 支持这一点，请参阅 connections.ArqRedis.enqueue_job(). 它保证具有特定 ID 的作业在执行完成之前不能再次入队。

```
from aiorq import create_pool
from aiorq.connections import RedisSettings
import os
import asyncio

redis_settings = RedisSettings(
    host=os.getenv("REDIS_HOST", "127.0.0.1"),
    port=os.getenv("REDIS_PORT", 6379),
    database=os.getenv("REDIS_DATABASE", 0),
    password=os.getenv("REDIS_PASSWORD", None)
)


async def main():
    redis = await create_pool(redis_settings)
    job = await redis.enqueue_job('say_hi', name="pai", _queue_name="pai:queue")
    await job.info()
```

(下页继续)

(续上页)

```
if __name__ == '__main__':
    asyncio.run(main())
```

- 通过 enqueue_job 方法将任务发布到 redis 队列中

7.2 延时执行任务

您可以安排作业在未来运行，无论是在给定的持续时间 (_defer_by) 还是在特定时间 _defer_until

```
# 延迟 100s 执行
await redis.enqueue_job('say_hi', name="pai", _queue_name="pai:queue", _defer_by=100)

# 延迟 1 分钟执行
await redis.enqueue_job('say_hi', name="pai", _queue_name="pai:queue", _defer_
↪by=timedelta(minutes=1))

# 长时间等待
await redis.enqueue_job('say_hi', name="pai", _queue_name="pai:queue", _defer_
↪until=datetime(2032, 1, 28))
```

- 通过指定 _defer_by 延时 100s 执行任务

7.3 重试任务

```
job = await redis.enqueue_job('say_hi', name="pai", _queue_name="pai:queue", _job_
↪try=2, _defer_by=100)
```

- 通过指定 _job_try 参数指定该任务的重试次数为 2

```
  aiorq demo.WorkerSettings
12:42:50: Starting worker for 1 functions: say_hi
12:42:50: redis_version=4.0.9 mem_usage=904.61K clients_connected=4 db_keys=4
12:42:50:  21.78s → c3dd4acc171541b9ac10b1d791750cde:the_task() try=2 delayed=21.78s
12:42:55:   5.00s ← c3dd4acc171541b9ac10b1d791750cde:say_hi ●
^C12:42:57: shutdown on SIGINT ◆ 1 jobs complete ◆ 0 failed ◆ 0 retries ◆ 0 ongoing␣
↪to cancel
```

在 aiorq 中引发 Retry 异常来重试作业

```
import asyncio
from aiohttp import ClientSession
from aiorq import create_pool, Retry
from aiorq.connections import RedisSettings

async def download_content(ctx, url):
    session: ClientSession = ctx['session']
    async with session.get(url) as response:
        if response.status != 200:
            raise Retry(defer=ctx['job_try'] * 5)
        content = await response.text()
    return len(content)
```

7.4 任务取消

要中止工作，请调用 Job.abort()

如果作业已经在运行，它将中止它，如果它当前在队列中，则阻止它运行

如果需要在程序中取消作业，请在 WorkerSettings 中设置 allow_abort_jobs = True

```
class WorkerSettings:
    functions = []
    allow_abort_jobs = True
```

```
async def main():
    redis = await create_pool(RedisSettings())
    job = await redis.enqueue_job('do_stuff')
    await asyncio.sleep(1)
    await job.abort()
```

```
  arq demo.WorkerSettings
12:42:38: Starting worker for 1 functions: say_hi
12:42:38: redis_version=4.0.9 mem_usage=904.50K clients_connected=4 db_keys=3
12:42:38:  10.23s → c3dd4acc171541b9ac10b1d791750cde:say_hi() delayed=10.23s
^C12:42:40: shutdown on SIGINT ◆ 0 jobs complete ◆ 0 failed ◆ 0 retries ◆ 1 ongoing␣
↪to cancel
12:42:40:   1.16s ↻ c3dd4acc171541b9ac10b1d791750cde:say_hi cancelled, will be run␣
↪again
```

CHAPTER 8

获取结果和状态

```
async def main():
    redis = await create_pool(redis_settings)
    job = await redis.enqueue_job('say_hi', name="pai", _queue_name="pai:queue")
    await job.job_id
    await job.info()
    await job.status()
    await job.result(timeout=5)
```

- job_id 为任务唯一 id，由 uuid 生成
- info() 方法返回一个 JobDef
- status() 方法返回 Status
- result() 方法返回 Result 对象，timeout 为获取结果的超时时间，超时返回 None

CHAPTER 9

定时任务

```python
from aiorq import cron

async def run_regularly(ctx):
    print('run foo job at 9.12am, 12.12pm and 6.12pm')

class WorkerSettings:
    cron_jobs = [
        cron(run_regularly, hour={9, 12, 18}, minute=12)
    ]
```

CHAPTER 10

健康检查

```
aiorq --check demo.WorkerSettings
```

每个 worker 在 redis 中都有一个健康检查的键，用于统计任务状态。

```
Mar-01 17:41:22 j_complete=0 j_failed=0 j_retried=0 j_ongoing=0 queued=0
```

CHAPTER 11

关于命令行

```
aiorq --help
arq --check demo.WorkerSettings
```

CHAPTER 12

如何优雅的杀死 worker 进程

需要注意的是，当 worker 后台常驻的时候 (可能是 nohup 或者 supervisor)，使用 kill -9 强制终端信号的时候，很有可能无法执行 redis 会话的 close() 回调。这跟 ctrl + c 中断信号是非一致的，比较推荐的做法是使用 kill -15 优雅的阻塞等待任务完成后杀死进程。这可能作为一个 Fix 在下一个版本中修复、唉。

CHAPTER 13

Reference

CHAPTER 14

致谢

- Arq and FastAPI

CHAPTER 15

License

MIT