darq

Release v0.11.0

Nov 07, 2022

Contents:

Current Version: v0.11.0

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

1.1 Reference

class darq.app.**Darq**(**, queue_name: str = 'arq:queue', redis_settings: Optional[darq.connections.RedisSettings] = None, burst: bool = False, on_startup: Callable[[Dict[Any, Any]], Awaitable[None]] = None, on_shutdown: Callable[[Dict[Any, Any]], Awaitable[None]] = None, on_job_prerun: Optional[Callable[[Dict[Any, Any], Task, Sequence[Any], Mapping[str, Any]], Awaitable[None]]] = None, on_job_postrun: Optional[Callable[[Dict[Any, Any], Task, Sequence[Any], Mapping[str, Any], Any], Awaitable[None]]] = None, on_job_prepublish: Optional[Callable[[Dict[str, Any], Task, List[Any], Dict[str, Any], darq.types.JobEnqueueOptions], Awaitable[None]]] = None, on_scheduler_startup: Callable[[Dict[Any, Any]], Awaitable[None]] = None, on_scheduler_shutdown: Callable[[Dict[Any, Any]], Awaitable[None]] = None, max_jobs: int = 10, job_timeout: Union[int, float, datetime.timedelta] = 300, keep_result: Union[int, float, datetime.timedelta] = 3600, poll_delay: Union[int, float, datetime.timedelta] = 0.5, queue_read_limit: Optional[int] = None, max_tries: int = 5, health_check_interval: Union[int, float, datetime.timedelta] = 3600, health_check_key: Optional[str] = None, ctx: Optional[Dict[str, Any]] = None, retry_jobs: bool = True, max_burst_jobs: int = -1, job_serializer: Optional[Callable[[Dict[str, Any]], bytes]] = None, job_deserializer: Optional[Callable[[bytes], Dict[str, Any]]] = None, default_job_expires: Union[int, float, datetime.timedelta] = datetime.timedelta(days=1)*)

Darq application.

Parameters

- **queue_name** – queue name to get tasks from

- **redis_settings** – settings for creating a redis connection
- **burst** – whether to stop the worker once all tasks have been run
- **on_startup** – coroutine function to run at worker startup
- **on_shutdown** – coroutine function to run at worker shutdown
- **on_job_prerun** – coroutine function to run before task starts
- **on_job_postrun** – coroutine function to run after task finish
- **on_job_prepublish** – coroutine function to run before enqueue task
- **max_jobs** – maximum number of tasks to run at a time
- **job_timeout** – default task timeout (max run time)
- **keep_result** – default duration to keep task results for
- **poll_delay** – duration between polling the queue for new tasks
- **queue_read_limit** – the maximum number of tasks to pull from the queue each time it's polled; by default it equals `max_jobs`
- **max_tries** – default maximum number of times to retry a task
- **health_check_interval** – how often to set the health check key
- **health_check_key** – redis key under which health check is set
- **ctx** – dict object, data from it will be pass to hooks: `on_startup`, `on_shutdown` - can modify `ctx`; `on_job_prerun`, `on_job_postrun` - readonly
- **retry_jobs** – whether to retry tasks on Retry or CancelledError or not
- **max_burst_jobs** – the maximum number of tasks to process in burst mode (disabled with negative values)
- **job_serializer** – a function that serializes Python objects to bytes, defaults to pickle.dumps
- **job_deserializer** – a function that deserializes bytes into Python objects, defaults to pickle.loads
- **default_job_expires** – default task expires. If the task still hasn't started after this duration, do not run it

add_cron_jobs(**cron_jobs*) → None

Parameters **cron_jobs** – list of cron jobs to run, use *darq.cron.cron()* to create them

task(*func=None*, ***, *keep_result=None*, *timeout=None*, *max_tries=None*, *queue=None*, *expires=<object object>*, *with_ctx=False*)

Parameters

- **func** – coroutine function
- **keep_result** – duration to keep the result for, if 0 the result is not kept
- **timeout** – maximum time the task should take
- **max_tries** – maximum number of tries allowed for the task, use 1 to prevent retrying
- **queue** – queue of the task, can be used to send task to different queue
- **expires** – if the task still hasn't started after this duration, do not run it
- **with_ctx** – pass context to the task as first argument

exception darq.app.**DarqConfigError**

exception darq.app.**DarqConnectionError**

exception darq.app.**DarqException**

class darq.connections.**ArqRedis**(*pool_or_conn: Union[aioredis.pool.ConnectionsPool, aioredis.connection.RedisConnection], job_serializer: Optional[Callable[[Dict[str, Any]], bytes]] = None, job_deserializer: Optional[Callable[[bytes], Dict[str, Any]]] = None, **kwargs*)

Thin subclass of `aioredis.Redis` which adds `darq.connections.enqueue_job()`.

Parameters

- **redis_settings** – an instance of `darq.connections.RedisSettings`.
- **job_serializer** – a function that serializes Python objects to bytes, defaults to pickle.dumps
- **job_deserializer** – a function that deserializes bytes into Python objects, defaults to pickle.loads
- **kwargs** – keyword arguments directly passed to `aioredis.Redis`.

all_job_results() → List[darq.jobs.JobResult]

Get results for all jobs in redis.

enqueue_job(*function: str, args: Sequence[Any], kwargs: Mapping[str, Any], *, job_id: Optional[str] = None, queue_name: Optional[str] = None, defer_until: Optional[datetime.datetime] = None, defer_by: Union[None, int, float, datetime.timedelta] = None, expires: Union[None, int, float, datetime.timedelta] = None, job_try: Optional[int] = None*) → Optional[darq.jobs.Job]

Enqueue a job.

Parameters

- **function** – Name of the function to call
- **args** – args to pass to the function
- **kwargs** – kwargs to pass to the function
- **job_id** – ID of the job, can be used to enforce job uniqueness
- **queue_name** – queue of the job, can be used to create job in different queue
- **defer_until** – datetime at which to run the job
- **defer_by** – duration to wait before running the job
- **expires** – if the job still hasn't started after this duration, do not run it
- **job_try** – useful when re-enqueueing jobs within a job

Returns *darq.jobs.Job* instance or `None` if a job with this ID already exists

queued_jobs(*, *queue_name: str = 'arq:queue'*) → List[darq.jobs.JobDef]

Get information about queued, mostly useful when testing.

class darq.connections.**RedisSettings**(*host: Union[str, List[Tuple[str, int]]] = 'localhost', port: int = 6379, database: int = 0, password: Optional[str] = None, ssl: Union[bool, None, ssl.SSLContext] = None, conn_timeout: int = 1, conn_retries: int = 5, conn_retry_delay: int = 1, sentinel: bool = False, sentinel_master: str = 'mymaster', sentinel_timeout: float = 0.2*)

No-Op class used to hold redis connection redis_settings.

Used by *darq.connections.create_pool()* and *darq.worker.Worker*.

darq.connections.**create_pool**(*settings: darq.connections.RedisSettings = None, *, retry: int = 0, job_serializer: Optional[Callable[[Dict[str, Any]], bytes]] = None, job_deserializer: Optional[Callable[[bytes], Dict[str, Any]]] = None*) → darq.connections.ArqRedis

Create a new redis pool, retrying up to `conn_retries` times if the connection fails.

Similar to `aioredis.create_redis_pool` except it returns a *darq.connections.ArqRedis* instance, thus allowing job enqueuing.

exception darq.worker.**Retry**(*defer: Union[int, float, datetime.timedelta, None] = None*)

Special exception to retry the job (if `max_retries` hasn't been reached).

Parameters **defer** – duration to wait before rerunning the job

class darq.worker.**Worker**(*app: Darq, **replace_kwargs*)

Main class for running jobs.

Parameters

- **app** – instance of *darq.app.Darq()*
- **worker_settings** – instance of `darq.worker.WorkerSettings`

async_run() → None

Asynchronously run the worker, does not close connections. Useful when testing.

run() → None

Sync function to run the worker, finally closes worker connections.

run_check(*retry_jobs: Optional[bool] = None, max_burst_jobs: Optional[int] = None*) → int

Run *darq.worker.Worker.async_run()*, check for failed jobs and raise `darq.worker.FailedJobs` if any jobs have failed.

Returns number of completed jobs

darq.cron.**cron**(*task: Union[str, darq.types.DarqTask[typing.Callable[..., typing.Coroutine[typing.Any, typing.Any, typing.Any]]][Callable[[...], Coroutine[Any, Any, Any]]]], *, name: Optional[str] = None, month: Union[None, Set[int], List[int], Tuple[int], int] = None, day: Union[None, Set[int], List[int], Tuple[int], int] = None, weekday: Union[None, Set[int], List[int], Tuple[int], int, str] = None, hour: Union[None, Set[int], List[int], Tuple[int], int] = None, minute: Union[None, Set[int], List[int], Tuple[int], int] = None, second: Union[None, Set[int], List[int], Tuple[int], int] = 0, microsecond: int = 123456, run_at_startup: bool = False, unique: bool = True, timeout: Union[int, float, datetime.timedelta, None] = None, keep_result: Optional[float] = 0, max_tries: Optional[int] = 1*) → darq.cron.CronJob

Create a cron job, eg. it should be executed at specific times.

Workers will enqueue this job at or just after the set times. If `unique` is true (the default) the job will only be run once even if multiple workers are running.

Parameters

- **task** – task function to run
- **name** – name of the job, if None, the name of the task is used
- **month** – month(s) to run the job on, 1 - 12
- **day** – day(s) to run the job on, 1 - 31
- **weekday** – week day(s) to run the job on, 0 - 6 or mon - sun
- **hour** – hour(s) to run the job on, 0 - 23
- **minute** – minute(s) to run the job on, 0 - 59
- **second** – second(s) to run the job on, 0 - 59
- **microsecond** – microsecond(s) to run the job on, defaults to 123456 as the world is busier at the top of a second, 0 - 1e6
- **run_at_startup** – whether to run as worker starts
- **unique** – whether the job should be only be executed once at each time
- **timeout** – job timeout
- **keep_result** – how long to keep the result for
- **max_tries** – maximum number of tries for the job

class darq.jobs.**JobStatus**

Enum of job statuses.

complete = 'complete'

job is complete, result is available

deferred = 'deferred'

job is in the queue, time it should be run not yet reached

in_progress = 'in_progress'

job is in progress

not_found = 'not_found'

job not found in any way

queued = 'queued'

job is in the queue, time it should run has been reached

class darq.jobs.**Job**(*job_id: str, redis: ArqRedis, _queue_name: str = 'arq:queue', _deserializer: Optional[Callable[[bytes], Dict[str, Any]]] = None*)

Holds data a reference to a job.

info() → Optional[darq.jobs.JobDef]

All information on a job, including its result if it's available, does not wait for the result.

result(*timeout: Optional[float] = None*, *, *pole_delay: float = 0.5*) → Any

Get the result of the job, including waiting if it's not yet available. If the job raised an exception, it will be raised here.

Parameters

- **timeout** – maximum time to wait for the job result before raising TimeoutError, will wait forever
- **pole_delay** – how often to poll redis for the job result

result_info() → Optional[darq.jobs.JobResult]
Information about the job result if available, does not wait for the result. Does not raise an exception even if the job raised one.

status() → darq.jobs.JobStatus
Status of the job.

1.2 Changelog

1.2.1 0.11.1 (unreleased)

- Remove `pydantic` dependency

1.2.2 0.11.0 (2022-08-03)

- Added ability to optionally pass `ctx` to the task, like this:

```
@task(with_ctx=True)
def foobar(ctx):
    log.info('Foobar try %s', ctx['job_try'])
```

`ctx` contains: `job_id`, `job_try`, `enqueue_time`, `score`, `metadata` + all worker's `ctx` (including custom context which can be passed via `on_startup`). Thanks to @kindermax (https://github.com/seedofjoy/darq/pull/426) !

1.2.3 0.10.2 (2022-02-03)

- Add proper typing for functions wrapped with the @task decorator. Mypy will now check that parameters are passed correctly when calling `func()` and `func.delay()`

1.2.4 0.10.1 (2021-07-29)

- Add `sentinel_timeout` (defaults to 0.2) param to `RedisSettings`

1.2.5 0.10.0 (2021-07-09)

- **Breaking change**: Rename `darq.worker.Function` to `darq.worker.Task`
- Made `job` to `task` naming migration
- Add max_jobs parameter to CLI (thanks to @antonmyronyuk)
- Fixed bug with `expires` argument: `default_job_expires` could not be replaced with `None` in `@task` or `.apply_async`

1.2.6 0.9.0 (2020-06-24)

- **Breaking change**: Add `scheduler_ctx` param to `on_scheduler_startup` and `on_scheduler_shutdown` to share data between this callbacks. It already has `ctx['redis']` - instance of `ArqRedis`

1.2.7 0.8.0 (2020-06-22)

- **Breaking change**: Changed CLI command format. Before: `darq some_project.darq_app.darq`. Now: `darq -A some_project.darq_app.darq worker`
- **Breaking change**: Scheduler (cron jobs) now run's seperate from worker (see `darq scheduler` command)
- **Breaking change**: Changed some function signatures (rename arguments)
- **Breaking change**: Remove `redis_pool` param from `Darq` app
- Add `on_scheduler_startup` and `on_scheduler_shutdown` callbacks

1.2.8 0.7.2 (2020-06-18)

- Fix some types (cron, OnJobPrepublishType)
- `on_job_prerun` now runs before "task started" log and `on_job_postrun` now runs after "task finished" log

1.2.9 0.7.1 (2020-05-25)

- `.apply_async`: Make `args` and `kwargs` arguments optional

1.2.10 0.7.0 (2020-05-25)

- Fork `arq` to project and merge it with `darq` (It was easier to rewrite `arq` than to write a wrapper)
- **Breaking change**: Remove "magic" params from `.delay`. For enqueue job with special params added `.apply_async`.
- Add `watch`-mode to CLI.
- Fix: Now worker will not run cronjob if it's functions queue not match with worker's

1.2.11 0.6.0 (2020-03-08)

- **Breaking change**: Changed *Darq* constructor from single *config* param to separate params.
- *arq_function.coroutine* now has *.delay* method.

1.2.12 0.5.0 (2020-03-03)

- Add `on_job_prepublish(metadata, arq_function, args, kwargs)` callback. `metadata` is mutable dict, which will be available at `ctx['metadata']`.

1.2.13 0.4.0 (2020-03-03)

- Add `default_job_expires` param to Darq (if the job still hasn't started after this duration, do not run it). Default - 1 day
- Add *expires* param to `@task` (if set - overwrites `default_job_expires`)

1.2.14 0.3.1 (2020-03-02)

- Rewrite warm shutdown: now during warm shutdown cron is disabled, on second signal the warm shutdown will be canceled

1.2.15 0.3.0 (2020-02-27)

- **Breaking change**: `on_job_prerun` and `on_job_postrun` now accepts `arq.worker.Function` instead of the original function (it can still be accessed at `arq_function.coroutine`)

1.2.16 0.2.1 (2020-02-26)

- Fix `add_cron_jobs` method. Tests added.

1.2.17 0.2.0 (2020-02-26)

- Add `on_job_prerun(ctx, function, args, kwargs)` and `on_job_postrun(ctx, function, args, kwargs, result)` callbacks.

1.2.18 0.1.0 (2020-02-26)

- **Breaking change**: Jobs no longer explicitly get `JobCtx` as the first argument, as in 99.9% cases it doesn't need it. In future release will be possible to optionally pass `JobCtx` in some way.
- **Breaking change**: All cron jobs should be wrapped in `@task` decorator
- Directly pass `functions` to `arq.Worker`, not names.

1.2.19 0.0.3 (2020-02-25)

- `.delay()` now returns `arq_redis.enqueue_job` result (`Optional[Job]`)
- Add `py.typed` file
- Fixed `add_cron_jobs` typing

1.2.20 0.0.2 (2020-02-24)

- Add `add_cron_jobs` method

1.2.21 0.0.1 (2020-02-21)

First release

Python Module Index

d

Index