# fastq

ci [npm version](https://badge.fury.io/js/fastq)

Fast, in memory work queue.

Benchmarks (1 million tasks):

* setImmediate: 812ms
* fastq: 854ms
* async.queue: 1298ms
* neoAsync.queue: 1249ms

Obtained on node 12.16.1, on a dedicated server.

If you need zero-overhead series function call, check out [fastseries](http://npm.im/fastseries). For zero-overhead parallel function call, check out [fastparallel](http://npm.im/fastparallel).

[js-standard-style](https://github.com/feross/standard)

* Installation
* Usage
* API
* Licence & copyright

## Install

npm i fastq --save

## Usage (callback API)

'use strict'

const queue = require('fastq')(worker, 1)

queue.push(42, function (err, result) {

if (err) { throw err }

console.log('the result is', result)

})

function worker (arg, cb) {

cb(null, arg \* 2)

}

## Usage (promise API)

const queue = require('fastq').promise(worker, 1)

async function worker (arg) {

return arg \* 2

}

async function run () {

const result = await queue.push(42)

console.log('the result is', result)

}

run()

### Setting "this"

'use strict'

const that = { hello: 'world' }

const queue = require('fastq')(that, worker, 1)

queue.push(42, function (err, result) {

if (err) { throw err }

console.log(this)

console.log('the result is', result)

})

function worker (arg, cb) {

console.log(this)

cb(null, arg \* 2)

}

### Using with TypeScript (callback API)

'use strict'

import \* as fastq from "fastq";

import type { queue, done } from "fastq";

type Task = {

id: number

}

const q: queue<Task> = fastq(worker, 1)

q.push({ id: 42})

function worker (arg: Task, cb: done) {

console.log(arg.id)

cb(null)

}

### Using with TypeScript (promise API)

'use strict'

import \* as fastq from "fastq";

import type { queueAsPromised } from "fastq";

type Task = {

id: number

}

const q: queueAsPromised<Task> = fastq.promise(asyncWorker, 1)

q.push({ id: 42}).catch((err) => console.error(err))

async function asyncWorker (arg: Task): Promise<void> {

// No need for a try-catch block, fastq handles errors automatically

console.log(arg.id)

}

## API

* fastqueue()
* queue#**push()**
* queue#**unshift()**
* queue#**pause()**
* queue#**resume()**
* queue#**idle()**
* queue#**length()**
* queue#**getQueue()**
* queue#**kill()**
* queue#**killAndDrain()**
* queue#**error()**
* queue#**concurrency**
* queue#**drain**
* queue#**empty**
* queue#**saturated**
* fastqueue.promise()

### fastqueue([that], worker, concurrency)

Creates a new queue.

Arguments:

* that, optional context of the worker function.
* worker, worker function, it would be called with that as this, if that is specified.
* concurrency, number of concurrent tasks that could be executed in parallel.

### queue.push(task, done)

Add a task at the end of the queue. done(err, result) will be called when the task was processed.

### queue.unshift(task, done)

Add a task at the beginning of the queue. done(err, result) will be called when the task was processed.

### queue.pause()

Pause the processing of tasks. Currently worked tasks are not stopped.

### queue.resume()

Resume the processing of tasks.

### queue.idle()

Returns false if there are tasks being processed or waiting to be processed. true otherwise.

### queue.length()

Returns the number of tasks waiting to be processed (in the queue).

### queue.getQueue()

Returns all the tasks be processed (in the queue). Returns empty array when there are no tasks

### queue.kill()

Removes all tasks waiting to be processed, and reset drain to an empty function.

### queue.killAndDrain()

Same than kill but the drain function will be called before reset to empty.

### queue.error(handler)

Set a global error handler. handler(err, task) will be called each time a task is completed, err will be not null if the task has thrown an error.

### queue.concurrency

Property that returns the number of concurrent tasks that could be executed in parallel. It can be altered at runtime.

### queue.drain

Function that will be called when the last item from the queue has been processed by a worker. It can be altered at runtime.

### queue.empty

Function that will be called when the last item from the queue has been assigned to a worker. It can be altered at runtime.

### queue.saturated

Function that will be called when the queue hits the concurrency limit. It can be altered at runtime.

### fastqueue.promise([that], worker(arg), concurrency)

Creates a new queue with Promise apis. It also offers all the methods and properties of the object returned by [fastqueue](#fastqueue) with the modified [push](#pushPromise) and [unshift](#unshiftPromise) methods.

Node v10+ is required to use the promisified version.

Arguments:

* that, optional context of the worker function.
* worker, worker function, it would be called with that as this, if that is specified. It MUST return a Promise.
* concurrency, number of concurrent tasks that could be executed in parallel.

#### queue.push(task) => Promise

Add a task at the end of the queue. The returned Promise will be fulfilled (rejected) when the task is completed successfully (unsuccessfully).

This promise could be ignored as it will not lead to a 'unhandledRejection'.

#### queue.unshift(task) => Promise

Add a task at the beginning of the queue. The returned Promise will be fulfilled (rejected) when the task is completed successfully (unsuccessfully).

This promise could be ignored as it will not lead to a 'unhandledRejection'.

#### queue.drained() => Promise

Wait for the queue to be drained. The returned Promise will be resolved when all tasks in the queue have been processed by a worker.

This promise could be ignored as it will not lead to a 'unhandledRejection'.

## License

ISC