[![Build Status](https://secure.travis-ci.org/tim-kos/node-retry.png?branch=master)](http://travis-ci.org/tim-kos/node-retry "Check this project's build status on TravisCI") [![codecov](https://codecov.io/gh/tim-kos/node-retry/branch/master/graph/badge.svg)](https://codecov.io/gh/tim-kos/node-retry) # retry Abstraction for exponential and custom retry strategies for failed operations. ## Installation npm install retry ## Current Status This module has been tested and is ready to be used. ## Tutorial The example below will retry a potentially failing `dns.resolve` operation `10` times using an exponential backoff strategy. With the default settings, this means the last attempt is made after `17 minutes and 3 seconds`. ``` javascript var dns = require('dns'); var retry = require('retry'); function faultTolerantResolve(address, cb) { var operation = retry.operation(); operation.attempt(function(currentAttempt) { dns.resolve(address, function(err, addresses) { if (operation.retry(err)) { return; } cb(err ? operation.mainError() : null, addresses); }); }); } faultTolerantResolve('nodejs.org', function(err, addresses) { console.log(err, addresses); }); ``` Of course you can also configure the factors that go into the exponential backoff. See the API documentation below for all available settings. currentAttempt is an int representing the number of attempts so far. ``` javascript var operation = retry.operation({ retries: 5, factor: 3, minTimeout: 1 \* 1000, maxTimeout: 60 \* 1000, randomize: true, }); ``` ## API ### retry.operation([options]) Creates a new `RetryOperation` object. `options` is the same as `retry.timeouts()`'s `options`, with two additions: \* `forever`: Whether to retry forever, defaults to `false`. \* `unref`: Whether to [unref](https://nodejs.org/api/timers.html#timers\_unref) the setTimeout's, defaults to `false`. \* `maxRetryTime`: The maximum time (in milliseconds) that the retried operation is allowed to run. Default is `Infinity`. ### retry.timeouts([options]) Returns an array of timeouts. All time `options` and return values are in milliseconds. If `options` is an array, a copy of that array is returned. `options` is a JS object that can contain any of the following keys: \* `retries`: The maximum amount of times to retry the operation. Default is `10`. Seting this to `1` means `do it once, then retry it once`. \* `factor`: The exponential factor to use. Default is `2`. \* `minTimeout`: The number of milliseconds before starting the first retry. Default is `1000`. \* `maxTimeout`: The maximum number of milliseconds between two retries. Default is `Infinity`. \* `randomize`: Randomizes the timeouts by multiplying with a factor between `1` to `2`. Default is `false`. The formula used to calculate the individual timeouts is: ``` Math.min(random \* minTimeout \* Math.pow(factor, attempt), maxTimeout) ``` Have a look at [this article][article] for a better explanation of approach. If you want to tune your `factor` / `times` settings to attempt the last retry after a certain amount of time, you can use wolfram alpha. For example in order to tune for `10` attempts in `5 minutes`, you can use this equation: ![screenshot](https://github.com/tim-kos/node-retry/raw/master/equation.gif) Explaining the various values from left to right: \* `k = 0 ... 9`: The `retries` value (10) \* `1000`: The `minTimeout` value in ms (1000) \* `x^k`: No need to change this, `x` will be your resulting factor \* `5 \* 60 \* 1000`: The desired total amount of time for retrying in ms (5 minutes) To make this a little easier for you, use wolfram alpha to do the calculations: [article]: http://dthain.blogspot.com/2009/02/exponential-backoff-in-distributed.html ### retry.createTimeout(attempt, opts) Returns a new `timeout` (integer in milliseconds) based on the given parameters. `attempt` is an integer representing for which retry the timeout should be calculated. If your retry operation was executed 4 times you had one attempt and 3 retries. If you then want to calculate a new timeout, you should set `attempt` to 4 (attempts are zero-indexed). `opts` can include `factor`, `minTimeout`, `randomize` (boolean) and `maxTimeout`. They are documented above. `retry.createTimeout()` is used internally by `retry.timeouts()` and is public for you to be able to create your own timeouts for reinserting an item, see [issue #13](https://github.com/tim-kos/node-retry/issues/13). ### retry.wrap(obj, [options], [methodNames]) Wrap all functions of the `obj` with retry. Optionally you can pass operation options and an array of method names which need to be wrapped. ``` retry.wrap(obj) retry.wrap(obj, ['method1', 'method2']) retry.wrap(obj, {retries: 3}) retry.wrap(obj, {retries: 3}, ['method1', 'method2']) ``` The `options` object can take any options that the usual call to `retry.operation` can take. ### new RetryOperation(timeouts, [options]) Creates a new `RetryOperation` where `timeouts` is an array where each value is a timeout given in milliseconds. Available options: \* `forever`: Whether to retry forever, defaults to `false`. \* `unref`: Wether to [unref](https://nodejs.org/api/timers.html#timers\_unref) the setTimeout's, defaults to `false`. If `forever` is true, the following changes happen: \* `RetryOperation.errors()` will only output an array of one item: the last error. \* `RetryOperation` will repeatedly use the `timeouts` array. Once all of its timeouts have been used up, it restarts with the first timeout, then uses the second and so on. #### retryOperation.errors() Returns an array of all errors that have been passed to `retryOperation.retry()` so far. The returning array has the errors ordered chronologically based on when they were passed to `retryOperation.retry()`, which means the first passed error is at index zero and the last is at the last index. #### retryOperation.mainError() A reference to the error object that occured most frequently. Errors are compared using the `error.message` property. If multiple error messages occured the same amount of time, the last error object with that message is returned. If no errors occured so far, the value is `null`. #### retryOperation.attempt(fn, timeoutOps) Defines the function `fn` that is to be retried and executes it for the first time right away. The `fn` function can receive an optional `currentAttempt` callback that represents the number of attempts to execute `fn` so far. Optionally defines `timeoutOps` which is an object having a property `timeout` in miliseconds and a property `cb` callback function. Whenever your retry operation takes longer than `timeout` to execute, the timeout callback function `cb` is called. #### retryOperation.try(fn) This is an alias for `retryOperation.attempt(fn)`. This is deprecated. Please use `retryOperation.attempt(fn)` instead. #### retryOperation.start(fn) This is an alias for `retryOperation.attempt(fn)`. This is deprecated. Please use `retryOperation.attempt(fn)` instead. #### retryOperation.retry(error) Returns `false` when no `error` value is given, or the maximum amount of retries has been reached. Otherwise it returns `true`, and retries the operation after the timeout for the current attempt number. #### retryOperation.stop() Allows you to stop the operation being retried. Useful for aborting the operation on a fatal error etc. #### retryOperation.reset() Resets the internal state of the operation object, so that you can call `attempt()` again as if this was a new operation object. #### retryOperation.attempts() Returns an int representing the number of attempts it took to call `fn` before it was successful. ## License retry is licensed under the MIT license. # Changelog 0.10.0 Adding `stop` functionality, thanks to @maxnachlinger. 0.9.0 Adding `unref` functionality, thanks to @satazor. 0.8.0 Implementing retry.wrap. 0.7.0 Some bug fixes and made retry.createTimeout() public. Fixed issues [#10](https://github.com/tim-kos/node-retry/issues/10), [#12](https://github.com/tim-kos/node-retry/issues/12), and [#13](https://github.com/tim-kos/node-retry/issues/13). 0.6.0 Introduced optional timeOps parameter for the attempt() function which is an object having a property timeout in milliseconds and a property cb callback function. Whenever your retry operation takes longer than timeout to execute, the timeout callback function cb is called. 0.5.0 Some minor refactoring. 0.4.0 Changed retryOperation.try() to retryOperation.attempt(). Deprecated the aliases start() and try() for it. 0.3.0 Added retryOperation.start() which is an alias for retryOperation.try(). 0.2.0 Added attempts() function and parameter to retryOperation.try() representing the number of attempts it took to call fn().