MinionPool

A worker pool for NodeJS

MinionPool - Overview

- Easily create worker pools in seconds
- Based on callbacks (hooks) into the lifecycle of the pool
- Let's your do any number and type of tasks in parallel
- ***ONLY*** if those tasks do not rely in the order of processing
- Best results when doing idempotent operations
- Actually a family of pools:
  - MinionPool
  - ArrayMinionPool
  - MysqlMinionPool
  - RabbitMQMinionPool
- Can poll for tasks or accept injected tasks
- Open Source

MinionPool - Proven in the field

- Log files processing
- Large mysql tables processing
- ElasticSearch reindexing
- S3 uploaders
- Web crawlers

MinionPool - Installing

- npm install minionpool
- npm install mysql_minionpool
- npm install rabbitmq_minionpool

MinionPool - As dependency

```
{
  "dependencies": {
    "minionpool": "*",
    "rabbitmq_minionpool": "*",
    "mysql_minionpool": "*"
  }
}
```

MinionPool - Concepts

- TaskSource: Used to produce and introduce tasks into the pool. A task source will be initialized, used, and then shutdown

- Minion: Each pool consists of N minions. Tasks will be assigned to the first available minion. Each minion will be initialized, used, and then shutdown when the TaskSource has reported that no more tasks are available.

MinionPool - How to use it

```
var minionpoolMod = require('minionpool');

var minionPool = new minionpoolMod.MinionPool(options);
minionPool.start();
```

MinionPool - Options

```
var options = {
  name: 'test',
  debug: true,
  concurrency: 5,
  logger: console.log,
  continueOnError: true,
  taskSourceStart: function(callback) { … callback(err, state); }
  taskSourceNext: function(state, callback) { callback(err, task); return state; },
  taskSourceEnd: function(state, callback) { callback(); },
  minionTaskHandler: function(task, state, callback) { callback(err, state); },
  minionStart: function(callback) { callback(err, state); },
  minionEnd: function(state, callback) { callback(); },
  poolEnd: function() { process.exit(0); }
};
```

MinionPool - ArrayMinionPool

```
var options = {
  name: 'test',
  concurrency: 5,
  minionTaskHandler: function(task, state, callback) {
    setTimeout(function() { callback(undefined, state); }, Math.floor(Math.random() * 500));
  },
  poolEnd: function() {
    process.exit(0);
  }
};
var data = [.., .., .., .., ...];
var minionPool = new minionsMod.ArrayMinionPool(options, data);
minionPool.start();
```

MinionPool - MysqlMinionPool

```
var pool = new mysqlMinionPoolMod.MysqlMinionPool({
  mysqlConfig: {
    host: '127.0.0.1',
    user: 'root',
    password: 'pass',
    database: 'db',
    port: 3306
  },
  concurrency: 5,    // How many pages to get concurrently...
  rowConcurrency: 1, // ... and how many concurrent rows processed PER query
  taskSourceStart: function(callback) { callback(undefined, {page: 0, pageSize: 10}); },
  minionTaskHandler: function(task, state, callback) { callback(undefined, state); }
});
pool.start();
```

MinionPool - MysqlMinionPool

```
taskSourceNext: function(state, callback) {
  var query = "SELECT * FROM `db`.`table` LIMIT ?,?";
  state.mysqlPool.getConnection(function(err, mysqlConnection) {
    if(err) {
      callback(err, undefined);
    } else {
      mysqlConnection.query(
        query, [state.page * state.pageSize, state.pageSize], function(err, rows) {
          mysqlConnection.release();
          if(err) {
            callback(err, undefined);
          } else if(rows.length === 0) {
            callback(undefined, undefined);
          } else {
            callback(undefined, rows);
          }
        }
      );
    }
  });
  state.page++;
  return state;
},
```

MinionPool - RabbitMQMinionPool

```
var options = {
  name: 'test',
  concurrency: 5,
  mqOptions: {
    exchangeName: 'workers',  // Will also create workers.retry
    queueName: 'myWorkers',   // Will also create myWorkers.retry
    routingKey: 'myWorkers',  // Optional. Equals to queueName if missing
  }
};
var pool = new minionsMod.RabbitMqMinionPool(options);
process.on('SIGINT', function() {
  pool.end();
});
pool.start();
```

MinionPool - RabbitMQMinionPool

```
minionTaskHandler: function(msg, state, callback) {
  var payload = msg.payload;
  var headers = msg.headers;
  var deliveryInfo = msg.deliveryInfo;
  var message = msg.message;
  var queue = msg.queue;
  console.log('got task: %s', util.inspect(payload));
  // See the node-amqp doc for more info.
  message.reject(); // or message.acknowledge();
  callback(undefined, state);
},
```

MinionPool - RabbitMQMinionPool

- Dead Letter eXchanges to support retrying failed (rejected) operations
- Channels in confirmation mode, so failed publishes will be notified
- Messages published as persistent
- Queues and exchanges marked as durable, autodelete = false

MinionPool - MultiCore

taskset: "*used to set or retrieve the CPU affinity of a running process given its PID or to launch a new COMMAND with a given CPU affinity*"

- http://linux.die.net/man/1/taskset
- https://github.com/karelzak/util-linux

$ taskset -c N /usr/node/bin/node bin/myworker # 0 =< N < number of cores

MinionPool - MultiCore

For mysql minion pools: you can divide the total number of rows per the number of cores available, and launch one pool per core that will process then given range of id's.

For rabbitmq minion pools: Just spawn as many pools per core needed, they will compete to consume tasks, rabbitmq will handle the complex part :)

MinionPool - Links

- https://github.com/marcelog/minionpool
- https://github.com/marcelog/mysql_minionpool
- https://github.com/marcelog/rabbitmq_minionpool

MinionPool

Thank you, Inakos!