
node-persist

(localStorage on the server)

Super-easy asynchronous persistent data structures in Node.js, modeled after HTML5 localStorage

Node-persist doesn't use a database. Instead, JSON documents are stored in the file system for persistence. Because there is no network overhead, node-persist is just about as fast as a database can get. Node-persist uses the HTML5 localStorage API, so it's easy to learn.

This is still a work in progress. Send pull requests please. ## Note

- This is **not** designed for large amounts of data, you can do way more than the 5MB limit imposed by the browsers but don't stretch it, in some cases you might need to load the whole storage into RAM, but normal `getItem/setItem` does not.
- If you're looking for the version that supports both `synchronous` and `asynchronous` use `node-persist@2.1.0`

Install

```
1 $ npm install node-persist
```

Basic Example

```
1 const storage = require('node-persist');
2
3 //you must first call storage.init or initSync
4 storage.initSync( /* options ... */ );
5 // or
6 // storage.init( /* options ... */ );
7
8 // then anywhere else in your code
9 await storage.setItem('name','yourname')
10 console.log(await storage.getItem('name')); // yourname
```

Run the counter example:

```
1 $ cd examples/counter
2 $ node counter.js
3 $ open up localhost:8080
```

3.1.1 change logs

backward changes

- Added the `writeQueue`* options, trying to resolve issue#108, see the API Documentation below.

3.0.0 change logs

Non-backward changes

- All the `*Sync` functions were removed, **every** operation is now **asynchronous**
- All the `persist`* functions were removed
- **Nothing** is held up in **RAM** use your own memory caching module, i.e. nano-cache
- Node 7.6+ is required now, we're using `async/await`
- `continuous` and `interval` options were removed, since we immediately persist to disk now, **asynchronously**
- `forEach` callback now accepts an object `callback({key, value})` instead of 2 arguments `callback(key, value)`

2.0.0 change logs

Non-backward changes

- filenames on the file system are now md5 hashed now and the structure of the saved data has changed to include the ttl in them.
- no longer need/support a `options.ttlDir`, since the `ttls` are now stored in the same file as each value
- added `expiredInterval` option
- added `forgiveParseErrors` option

1.0.0 change logs

Mostly non-backward changes

- `storage.getItem()` now returns a promise
- `storage.valuesWithKeyMatch()` no longer accepts a callback
- `storage.values()` no longer accepts a callback
- `storage.key()` is gone

-
- The default `dir` is now `process.cwd() + (dir || '.node-persist/storage')`, unless you use an absolute path
 - added `storage.get()`, alias to `getItem()`
 - added `storage.set()`, alias to `setItem()`
 - added `storage.del()`, `storage.rm()`, as aliases to `removeItem()`
 - Keys, on the file system are base64 encoded with the replacement of the /

API Documentation

async init(options, [callback]) if the storage dir is new, it will create it ##### Options
You can pass `init()` an options object to customize the behavior of node-persist

These are the defaults

```
1 await storage.init({
2   dir: 'relative/path/to/persist',
3
4   stringify: JSON.stringify,
5
6   parse: JSON.parse,
7
8   encoding: 'utf8',
9
10  // can also be custom logging function
11  logging: false,
12
13  // ttl* [NEW], can be true for 24h default or a number in
    MILLISECONDS or a valid Javascript Date object
14  ttl: false,
15
16  // every 2 minutes the process will clean-up the expired cache
17  expiredInterval: 2 * 60 * 1000,
18
19  // in some cases, you (or some other service) might add non-valid
    storage files to your
20  // storage dir, i.e. Google Drive, make this true if you'd like to
    ignore these files and not throw an error
21  forgiveParseErrors: false,
22
23  // instead of writing to file immediately, each "file" will have
    its own mini queue to avoid corrupted files, keep in mind that
    this would not properly work in multi-process setting.
24  writeQueue: true,
25
26  // how often to check for pending writes, don't worry if you feel
    like 1s is a lot, it actually tries to process every time you
    setItem as well
27  writeQueueIntervalMs: 1000,
```

```
28
29     // if you setItem() multiple times to the same key, only the last
        one would be set, BUT the others would still resolve with the
        results of the last one, if you turn this to false, each one
        will execute, but might slow down the writing process.
30     writeQueueWriteOnlyLast: true,
31 });
```

async getItem(key) This function will get the value for that key stored on disk

```
1 let value = await storage.getItem('obj');
```

async setItem(key, value, [options]) This function sets 'key' in your database to 'value'

```
1 await storage.setItem('fibonacci',[0,1,1,2,3,5,8]);
2 await storage.setItem(42,'the answer to life, the universe, and
    everything.');
```

```
3 await storage.setItem(42,'the answer to life, the universe, and
    everything.', {ttl: 1000*60 /* 1 min */ });
```

* The only option available when calling `setItem(key, value, option)` is `{ttl: Number | Date}`

async updateItem(key, value, [options]) This function updates a 'key' in your database with a new 'value' without touching the `ttl`, however, if the `key` was not found or if it was `expired` a new item will get set

```
1 await storage.updateItem(42,'the answer to life, the universe, and
    everything.', {ttl: 1000*60*10 /* 10 minutes */ });
2 await storage.updateItem(42,'means nothing, do not trust wikipedia');
    // ttl is still the same, will expired in 10 minutes since it was
    first set
```

* The only option available when calling `updateItem(key, value, option)` is `{ttl: Number | Date}`

async removeItem(key) This function immediately deletes it from the file system asynchronously

```
1 await storage.removeItem('me');
```

async clear() This function immediately deletes all files from the file system asynchronously.

```
1 await storage.clear();
```

async values() This function returns all of the values

```
1 await storage.setItem("batman", {name: "Bruce Wayne"});
2 await storage.setItem("superman", {name: "Clark Kent"});
3 console.log(await storage.values()); //output: [{name: "Bruce Wayne"}, {
  name: "Clark Kent"}]
```

async valuesWithKeyMatch(match) This function returns all of the values matching a string or RegExp

```
1 await storage.setItem("batman", {name: "Bruce Wayne"});
2 await storage.setItem("superman", {name: "Clark Kent"});
3 await storage.setItem("hulk", {name: "Bruce Banner"});
4 console.log(await storage.valuesWithKeyMatch('man')); //output: [{name:
  "Bruce Wayne"}, {name: "Clark Kent"}]
5 // also accepts a Regular Expression
6 console.log(await storage.valuesWithKeyMatch(/man/)); //output: [{name:
  "Bruce Wayne"}, {name: "Clark Kent"}]
```

async keys() this function returns an array of all the keys in the database.

```
1 console.log(await storage.keys()); // ['batman', 'superman']
```

async length() This function returns the number of keys stored in the database.

```
1 console.log(await storage.length()); // 2
```

async forEach(callback) This function iterates over each key/value pair and executes an asynchronous callback as well

```
1 storage.forEach(async function(datum) {
2   // use datum.key and datum.value
3 });
```

Factory method

create(options) - synchronous, static method If you choose to create multiple instances of storage, you can. Just avoid using the same `dir` for the storage location. **You still have to call `init` after `create`** - you can pass your configs to either `create` or `init`

```
1 const storage = require('node-persist');
2 const myStorage = storage.create({dir: 'myDir', ttl: 3000});
3 await myStorage.init();
```

Tests

```
1 npm install
2 npm test
```

Simon Last