libnpmhook

`libnpmhook` is a Node.js library for programmatically managing the npm registry's server-side hooks.

For a more general introduction to managing hooks, see the introductory blog post.

Table of Contents

Example

```
const hooks = require('libnpmhook')

console.log(await hooks.ls('mypkg', {token: 'deadbeef'}))
// array of hook objects on `mypkg`.
```

Install

```
$ npm install libnpmhook
```

API

`opts` for `libnpmhook` commands

`libnpmhook` uses `npm-registry-fetch`. All options are passed through directly to that library, so please refer to its own `opts` documentation for options that can be passed in.

A couple of options of note for those in a hurry:

- `opts.token` - can be passed in and will be used as the authentication token for the registry. For other ways to pass in auth details, see the n-r-f docs.
- `opts.otp` - certain operations will require an OTP token to be passed in. If a `libnpmhook` command fails with `err.code === EOTP`, please retry the request with `{otp: <2fa token>}`

`> hooks.add(name, endpoint, secret, [opts]) -> Promise`

`name` is the name of the package, org, or user/org scope to watch. The type is determined by the name syntax: `'@foo/bar'` and `'foo'` are treated as packages, `@foo` is treated as a scope, and `~user` is treated as an org name or scope. Each type will attach to different events.

The `endpoint` should be a fully-qualified http URL for the endpoint the hook will send its payload to when it fires. `secret` is a shared secret that the hook will send to that endpoint to verify that it's actually coming from the registry hook.

The returned Promise resolves to the full hook object that was created, including its generated `id`.

See also: `POST /v1/hooks/hook`

Example

```
await hooks.add('~zkat', 'https://example.com/api/added', 'supersekrit', {
  token: 'myregistrytoken',
  otp: '694207'
})

=>

{ id: '16f7xoal',
  username: 'zkat',
  name: 'zkat',
  endpoint: 'https://example.com/api/added',
  secret: 'supersekrit',
  type: 'owner',
  created: '2018-08-21T20:05:25.125Z',
  updated: '2018-08-21T20:05:25.125Z',
  deleted: false,
  delivered: false,
  last_delivery: null,
  response_code: 0,
  status: 'active' }
```

`> hooks.find(id, [opts]) -> Promise`

Returns the hook identified by `id`.

The returned Promise resolves to the full hook object that was found, or error with `err.code` of `'E404'` if it didn't exist.

See also: `GET /v1/hooks/hook/:id`

Example

```
await hooks.find('16f7xoal', {token: 'myregistrytoken'})

=>

{ id: '16f7xoal',
  username: 'zkat',
  name: 'zkat',
```

```
  endpoint: 'https://example.com/api/added',
  secret: 'supersekrit',
  type: 'owner',
  created: '2018-08-21T20:05:25.125Z',
  updated: '2018-08-21T20:05:25.125Z',
  deleted: false,
  delivered: false,
  last_delivery: null,
  response_code: 0,
  status: 'active' }
```

`> hooks.rm(id, [opts]) -> Promise`

Removes the hook identified by `id`.

The returned Promise resolves to the full hook object that was removed, if it existed, or `null` if no such hook was there (instead of erroring).

See also: `DELETE /v1/hooks/hook/:id`

Example

```
await hooks.rm('16f7xoal', {
  token: 'myregistrytoken',
  otp: '694207'
})

=>

{ id: '16f7xoal',
  username: 'zkat',
  name: 'zkat',
  endpoint: 'https://example.com/api/added',
  secret: 'supersekrit',
  type: 'owner',
  created: '2018-08-21T20:05:25.125Z',
  updated: '2018-08-21T20:05:25.125Z',
  deleted: true,
  delivered: false,
  last_delivery: null,
  response_code: 0,
  status: 'active' }

// Repeat it...
await hooks.rm('16f7xoal', {
  token: 'myregistrytoken',
  otp: '694207'
})

=> null
```

`> hooks.update(id, endpoint, secret, [opts]) -> Promise`

The `id` should be a hook ID from a previously-created hook.

The `endpoint` should be a fully-qualified http URL for the endpoint the hook will send its payload to when it fires. `secret` is a shared secret that the hook will send to that endpoint to verify that it's actually coming from the registry hook.

The returned Promise resolves to the full hook object that was updated, if it existed. Otherwise, it will error with an `'E404'` error code.

See also: `PUT /v1/hooks/hook/:id`

Example

```
await hooks.update('16fxoal', 'https://example.com/api/other', 'newsekrit', {
  token: 'myregistrytoken',
  otp: '694207'
})

=>

{ id: '16f7xoal',
  username: 'zkat',
  name: 'zkat',
  endpoint: 'https://example.com/api/other',
  secret: 'newsekrit',
  type: 'owner',
  created: '2018-08-21T20:05:25.125Z',
  updated: '2018-08-21T20:14:41.964Z',
  deleted: false,
  delivered: false,
  last_delivery: null,
  response_code: 0,
  status: 'active' }
```

`> hooks.ls([opts]) -> Promise`

Resolves to an array of hook objects associated with the account you're authenticated as.

Results can be further filtered with three values that can be passed in through `opts`:

- `opts.package` - filter results by package name
- `opts.limit` - maximum number of hooks to return
- `opts.offset` - pagination offset for results (use with `opts.limit`)

See also:

- `hooks.ls.stream()`
- `GET /v1/hooks`

Example

```
await hooks.ls({token: 'myregistrytoken'})
```

```
=>
[
  { id: '16f7xoal', ... },
  { id: 'wnyf98a1', ... },
  ...
]
```

`> hooks.ls.stream([opts]) -> Stream`

Returns a stream of hook objects associated with the account you're authenticated as. The returned stream is a valid `Symbol.asyncIterator` on `node@>=10`.

Results can be further filtered with three values that can be passed in through `opts`:

- `opts.package` - filter results by package name
- `opts.limit` - maximum number of hooks to return
- `opts.offset` - pagination offset for results (use with `opts.limit`)

See also:

- `hooks.ls()`
- `GET /v1/hooks`

Example

```
for await (let hook of hooks.ls.stream({token: 'myregistrytoken'})) {
  console.log('found hook:', hook.id)
}

=>
// outputs:
// found hook: 16f7xoal
// found hook: wnyf98a1
```