AVA Hook Documentation

Release 0.0.0

Ryan Leonard

Sep 22, 2017

Contents

CHAPTER 1

Documentation

Introduction

A formal interface for test helpers that mock and cleanup variables in *test.beforeEach* and *test.afterEach* sections.

This is an unofficial format for the AVA library, and has no connection with the AVA project.

Features

For developers writing AVA hooks, *ava-hook* provides a well-featured and tested container to add your setup and cleanup code to. Instead of re-inventing the wheel in every library, *ava-hook* provides a user-focused interface, generic documentation for using the hook, and extra configuration to control the functionality of the hook.

- Simple syntax for defining the stages of your hook (both for setup and for cleanup)
- Variable registration system to ensure that variables in *t.context* are free from conflict
- *(soon)* User-visible configuration variables
- *(soon)* Dependency system to use other *ava-hook* modules for setup

ava-hook is also focused on being flexible both for the hook developer as well as the end user. Most of the features above are customizable, based on options provided. Additionally, most parts of *ava-hook* are overridable, both by the hook developer and the end user.

Next

Want to learn more conceptually? Start with the *Overview*.

Ready to start using *ava-hook* in your own projects? Read the *ava-hook for Developers*.

Are you trying to use an AVA hook module that is using *ava-hook*? Check out the *Generic ava-hook User Guide*.

See the *API* reference for other technical details.

Overview

AVA hooks have three core sections:

- **Dependencies** allow hooks to create instances of other hooks. For instance, a file-system based database may need a temporary directory to be created.
- **Config** variables allow basic properties to be tweaked. For instance, the database may have an "in-memory" option.
- **Variables** define the sections that will be added to *t.context* in the AVA test instance. AVA hooks will make sure that variables don't conflict, and can either throw errors or rename variables when conflicts occur.

AVA hooks are designed so the hook developer provides the basic sections that they intend to run. The end-user can override any dependencies defined (or provide different configuration to them), configure the variables that will be added to *t.context*, and also configure how the hook is implemented:

- Each stage of the hook (creating a database instance, loading temporary data, etc.) can be given their own hidden *beforeEach()*, to provide easy debugging if the library breaks. *(default)*
- All stages of the hook can be grouped together into a single *beforeEach()* hook.
- The user can create reach inside the AVA hook and create a *beforeEach()* hook for each stage in the AVA hook.

Guides

ava-hook for Developers

Writing a Hook

This is intended to be a quick-start guide to get you writing an AVA hook using the *ava-hook* library. We'll be creating an *AVAHook* that sets up a simple Express server.

For a more in-depth overview, please see the *Overview*.

Environment

This guide assumes that you are writing a Node.js module that is only focused on exporting this single AVA hook. These instructions would need to be adapted for use inside a larger program.

Listing 1.1: Sample Module Setup

```
mkdir ava-express-server
cd ava-express-server
npm init
```

Installation

Install *ava-hook* via NPM:

```
npm install --save ava-hook
```

Extend *ava-hook*

Start by inheriting the basic methods from *AVAHook*. .. TODO: link to API

Listing 1.2: AVAExpressServer.js

```
const AVAHook = require("ava-hook");

class AVAExpressServer extends AVAHook {

}

module.exports = AVAExpressServer;
```

Declare Variables

We'll want to add the Express server (returned from *express()*) to the test context (*t.context*).

By declaring the variable through *ava-hook*, the library will ensure that no other hooks are already using our variable.

When variable conflicts are detected, *ava-hook* will automatically assign us an unused variable.

```
class AVAExpressServer extends AVAHook {

  static get variables() {
    return [
      "app",
    ];
  }

}
```

Define Hooks

Next we'll add methods to the class for each setup or cleanup hook we want.

When interacting with *t.context*, we'll use *this.variable()*, which may provide a slightly-modified variable to avoid conflicts.

```
const express = require("express");

class AVAExpressServer extends AVAHook {

  createServer(t) {
    t.context[this.variable("app")] = express();
  }

  cleanupServer(t) {
    delete t.context[this.variable("app")];
  }

}
```

We don't really need the *cleanupServer* stage (as garbage collection should take care of it), but we'll use it for this guide.

Register Hooks

Now that we've got the methods written, we can register them with *AVAHook* so *test.beforeEach()* hooks will be automatically created.

```
class AVAExpressServer extends AVAHook {

  static get setup() {
    return {
      createServer: "create Express server",
    };
  }

  static get cleanup() {
    return {
      cleanupServer: "teardown Express server",
    };
  }

}
```

setup() and *cleanup()* take a list of the methods to call in *beforeEach()* and *afterEach()* (respectively), and also include the title to provide for each stage.

Conclusion

Entire File

First, here's the entire file:

Listing 1.3: AVAExpressServer.js

```
const AVAHook = require("ava-hook");
const express = require("express");

class AVAExpressServer extends AVAHook {

  static get variables() {
    return [
      "app",
    ];
  }

  static get setup() {
    return {
      createServer: "create Express server",
    };
  }

  static get cleanup() {
    return {
      cleanupServer: "teardown Express server",
    };
  }

  createServer(t) {
```

```
    t.context[this.variable("app")] = express();
  }

  cleanupServer(t) {
    delete t.context[this.variable("app")];
  }

}

module.exports = AVAExpressServer;
```

Example Usage

```
const test = require("ava");
const AVAExpressServer = require("ava-express-server");
const request = require("supertest");

let server = new AVAExpressServer();
server.register();

test("default Express routing", t => {
  return request(t.context.app)
    .get("/undefined-path/")
    .expect(404);
});
```

Summary

We've integrated several different features of *ava-hook* in the above example. We'll briefly break them down.

Variables

We're using "Hook Variables" twice in this example.

First, we override *AVAHook.variables* to define the custom variables that we want to set.

Then in our "Hook Stages", we use *this.variable()* to retrieve the context variable name. Normally this will be the same as the requested variable name (e.g. *this.variable("app")* will normally return *"app"*), but the variable name may be altered to avoid a collision in the context variables.

Hook Stages

We defined instance methods that contain the body of our setup and cleanup hooks. In our example, we registered these stages with *AVAHook* so that the user can call *AVAExpressServer#register*, but the user could also pluck individual stages in custom *test.beforeEach* statements:

```
const test = require("ava");
const AVAExpressServer = require("ava-express-server");

const server = new AVAExpressServer();

```

```
test.beforeEach("create server", server.createServer);

// ...
```

Most users will likely use *#register* to setup all hooks at once instead of dealing with each individual hook, but having the ability to access each hook individually enables better debugging or customization.

Stage Registration

Finally, we overrode *AVAHook.setup* and *AVAHook.cleanup* to register each stage with the hook, and provided names for each step.

Using Hook Dependencies

We *previously covered* setting up a basic AVA hook, where we created a hook that constructed a bare Express server.

If we wanted to create a more complex hook, such as constructing a GraphQL API inside an Express server, it would be nice to re-use the Express hook instead of doing all that work again.

AVA Hook has a basic syntax for adding dependencies for new AVA Hooks.

Environment

This guide will assume that you already have the AVA Hook that creates an Express server.

We'll create a second NPM package for this GraphQL API layer.

Listing 1.4: NPM Setup

```
mkdir ava-express-graphql
cd ava-express-graphql
npm init
```

Installation

We'll be using *ava-hook*, the *ava-express-server* we created before, and the *express-graphql* package.

Listing 1.5: Package Installation

```
npm install --save ava-hook ava-express-server express-graphql
```

Hook Interface

Listing 1.6: ExpressGraphQLHook.js

```
const AVAHook = require("ava-hook");
const AVAExpressServer = require("ava-express-server");
const graphql = require("express-graphql");

class ExpressGraphQLHook extends AVAHook {

```

```
  static get dependencies() {

  }

  static get setup() {
    return {
      addMiddleware: "add GraphQL middleware",
    };
  }

  addMiddleware(t) {

  }

  addGet(t) {

  }

}
```

Declare Dependencies

To use the Express hook that we already created, we need to mark it as a dependency.

```
const AVAExpressServer = require("ava-express-server");

class ExpressGraphQLHook extends AVAHook {

  static get dependencies() {
    return {
      app: AVAExpressServer,
    };
  }

}
```

dependencies takes the list of AVA hooks, indexed by a custom name. By using unique custom names, we can use the same hook as a dependency twice. This could be useful if we wanted two different temporary files.

Now that the dependency has been declared, the dependent's setup/cleanup hooks will be registered before our own hooks stages.

Use Dependencies

Now that the Express setup/cleanup hooks have been registered, we can use the server in our own hooks.

```
const graphql = require("express-graphql");

class ExpressGraphQLHook extends AVAHook {

  addMiddleware(t) {
    const app = t.context[this.dependency("app").variable("app")];
    app.use("/graphql", graphql({
      schema: {},
      graphiql: true,
```

```
10      }));
11    }
12
13  }
```

Note: Simplifed Example This is a basic example of hook dependencies, and the *graphql* configuration has been simplifed.

You should probably use a variable for the Express URL, GraphQL schema, and the other variables.

We can use *this.dependency* to get the AVAHook instance for the Express hook. This ensures that we use the correct variable name for the Express server, even if the variable name was changed to avoid a conflict.

Generic *ava-hook* User Guide

API

AVA Hooks publishes a JavaScript API, intended for hook developers to use.

AVAHook

class AVAHook(*opts*)

Abstract interface for test helpers.

Create a new test helper that can later be connected to AVA.

Arguments

- **opts** (*AVAHookOptions*) – options to configure this {@link AVAHook}.

afterEach()

Register each *afterEach()* stage with AVA.

beforeEach()

Register each *beforeEach()* stage with AVA.

cleanup

The list of stages to cleanup, with the description to provide *t.afterEach()*.

dependencies

Dependencies that should be run before this hook.

dependency(*name*)

Get a local dependency by name.

Arguments

- **name** (*String*) – the name of the dependency

Returns **AVAHook** – the hook instance.

register()

Register all stage hooks with AVA, and reserve variables.

registerDependencies()

Register all dependencies for this hook.

registerList (*list*, *type*)
Register each stage hook with AVA from the given list.

Arguments

- **list** (*StageList*) – the stages to register
- **type** (*String*) – the AVA hook to register stages with.

registerVariable (*varName*, *instance*, *requested*)
Register a global variable for use in {@link AVAHook} instances. Provides an alternative name if the name is already taken.

Arguments

- **varName** (*String*) – the name of the variable to reserve.
- **instance** (AVAHook) – the instance of {@link AVAHook} (for debugging, etc.)
- **requested** (*String*) – an alternative name requested by the user.

Returns **String** – the final name of the variable that is reserved for this instance.

reserveVariables ()
Reserve all variables needed for this instance. If variables are already claimed, alternatives will be found.

setup
The list of stages to setup, with the description to provide *t.beforeEach()*.

variable (*name*)
Get the globally reserved name for a variable from the non-unique name used for this instnace.

Arguments

- **name** (*String*) – the local name of the variable.

variables
The names of variables that will be used inside this hook, and exported when the hook is complete.

- genindex

Index