HAPIDAYS: `SERVER.INJECT` AND REGISTERED SERVER METHODS

Farrin A. Reid

Director of Engineering @ MovingWorlds

handle: blakmatrix

slides: http://github.com/blakmatrix/hapiday2014talk

OVERVIEW

I am going to discuss when and when to not use `server.inject` and when to use `server.method` in the Hapi.js framework.

I hope by then end of the talk you will learn when and how to use both.

INSPIRATION

This talk was inspired by discovering issues during the development of our public launch product.

tl;dw: used `server.inject` as stopgap; found bugs.

SERVER.INJECT

Uses the shot module for performing injections without making a socket connection

SERVER.INJECT(*OPTIONS*, CALLBACK)

- Options (`String` with the requested URI || `Object`)
  - method - (GET|POST|PUT|...)
  - url - request URL (foo.com:8080) **required**
  - headers - {key:value}
  - payload - (`String` || `buffer`) payload objects need to be converted to strings first
  - credentials - (`Object`) contains authentication data
  - simulate - (`Object`)
    - error - if true, emits an 'error' event after payload transmission (if any)
    - close - if true, emits a 'close' event after payload transmission (if any)
    - end - if false, does not end the stream.

SERVER.INJECT(OPTIONS, *CALLBACK*)

- callback - `function(res)`, **required**
  - `res.statusCode` - HTTP status code
  - `res.headers` - HTTP headers
  - `res.payload` - `String` of response payload
  - `res.rawPayload` - raw response payload buffer
  - `res.raw` - `{req:{},res:{}}` `Object` with the injection request and response objects
  - `res.result` - The raw handler response before it's turned int what is returned via `res.payload` **not always available**

WHEN/WHY `SERVER.INJECT` SHOULD BE USED:

- Testing:

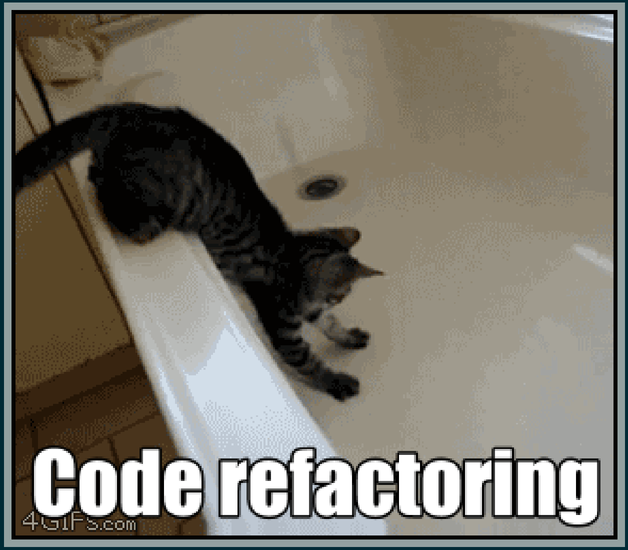
```
var Lab = require("lab");
var lab = exports.lab = Lab.script();

lab.experiment( "Test username existence", function() {
  // tests

  lab.test("Expect status code 200 for name that is taken", function(done){

    var options = {
      method: 'GET',
      url: '/timmy'
    };
    //server inject lets you simulate an http request
    server.inject(options, function(response){
      Lab.expect(response.statusCode).to.equal(200);
      Lab.expect(response.result.available).to.equal("no");
      done();
    });
```

WHEN/WHY `SERVER.INJECT` SHOULD BE USED:

- Doing things that server methods cannot, eg. complex routing logic that cannot easily be abstracted
- Avoiding overhead/imitations of the network stack
  - you can isolate networking issues
- `credentials` by seting this you can bypass default authenticated strategies
  - testing keeping some routes "internal only" with perhaps higher privilages

WHEN/WHY `SERVER.INJECT` SHOULD NOT BE USED:

- `res.result` is not always available, meaning `res.payload` may need to be used which leads to potentially needing to use `JSON.parse` on the string, costing both the overhead of encoding the response into a string, then parsing it (`JSON.parse(res.payload)`) and potentially reencoding to finally send to the client.
  - There is an implicit assumption (by the developer) that the route injected will return the expected `res.statusCode` depending on the situation. This can lead to confusion when assumptions are wrong.


Code refactoring
4GIFS.com

REGISTERED SERVER METHODS

- Methods are awesome because you can use them to share common functions by attaching them to the `server` object, this means you do not have to require your common modules everywhere you need them (essentially helps you to make your code more DRY).
- You can cache your methods usign hapi's native caching.
- You can change the `this` context within your methods.

WHAT

- You need to give your method a name, you can access via `server.methods[name]` later

- Neat Feature: You can register a name such as `'user.get'` and it will register it as a nested object (eg. `{user:{get: foo(){}}}`)
  - This can be a great way to organize your server methods and used with bind can serve as a method namespace accessible through `this` in the method.

```
server.method([{
name: 'users.findAndCountAll',
method: function(options, next) {
  this.findAndCountAll(options).complete(function (err, result) {
    if (err) {
      return next(err);
    }
    return next(null, result);
  });
},
options: {
  bind: server.plugins['hapi-sequelize'].models.User
}
}]);
```

- You need to give your method a function of course

```
{
  method: function(arg1, arg2, ..., argn, next){
    next(err, result, ttl);
    }
}
```

- You may also give your method some [options]
  - `bind` - the context object sent back to the method via `this`
  - `cache` - cache configurations (same as `server.cache()`)
  - `callback` - set to false if your method is synchronous (callback in method takes the form `function(err, result, cached, report)`)
  - `generateKey` - for generating a unique key for your caching, must produce a `number`, `boolean`, or `string`

HOW

You can register server methods in two ways:

- As seperate parameters

```
var foo = function (x, y, next
    next(null, x + y);
};

server.method('foo', foo, {});
```

- As an object

```
var foo = function (x, y, next
    next(null, x + y);
};

server.method({
    name: 'foo',
    method: foo,
    options: {}
});
```

- (or an array of objects)

```
var foo = function (x, y, next
    next(null, x + y);
};
var bar = function (x, y, next
    next(null, x * y);
};

server.method([
  {
    name: 'foo',
    method: foo,
    options: {}
  },
  {
    name: 'bar',
    method: bar,
    options: {}
```

WHEN/WHY SERVER METHODS SHOULD BE USED:

- If you want to use DRY principles, avoids having to require common modules everywhere in your code when you have access to the `server` object
- If want to cache a string/number/boolean/[json] object (buffers and stream support in the future?)

WHEN/WHY SERVER METHODS SHOULD NOT BE USED:

- If you only perform an operation once (slight overhead to call the method)
- It doesn't fit your use case

CLOSE/SUMMARY

Use `server.inject` for tests, elsewhere if and only if routing logic is too complex to be abstracted

Use server methods if you need to use the same pieces of code in multiple locations, ++points for speeding up responses with caching!

THANKS

Farrin A. Reid

@blakmatrix - on all the things