How Nest takes controller method and assign it specific path (how pipes, guards … are applied)?

Nestjs defines and store routes using router-resolver and httpAdapter (express)

Nest takes modules, all controllers and each method in controllers and create a path till that method and a handler that will be assigned to it’s path

Our handler is our controller method which applies it’s guards, pipes and interceptors.

Below is presented how nest stores path and its method and how cached object is looking:

Also all code explanation is stored on:

Github: <https://github.com/dimapascal/learn-nestjs/tree/explore_routing_table>

Photos folder:

Path object:

const entrypointDefinition: Entrypoint<HttpEntrypointMetadata> = {

type: 'http-endpoint',

methodName, *// controller method name*

className: instanceWrapper.name, *// controller.name*

classNodeId: instanceWrapper.id, *// controller.id*

metadata: {

key: path, *// final path to method*

path, *// final path to method*

requestMethod: RequestMethod[

requestMethod *// post, get, delete ...*

] as keyof typeof RequestMethod,

methodVersion: routePathMetadata.methodVersion as VersionValue,

controllerVersion:

routePathMetadata.controllerVersion as VersionValue,

},

};

Storing of path object:

this.graphInspector.insertEntrypointDefinition<HttpEntrypointMetadata>(

entrypointDefinition,

instanceWrapper.id,

);

Below is very important line as hear we register in httpAdapter (express) our final path and handler which already applies pipes guards and interceptors (our controller method)

*// Hear i assign my handler method to appropriate path*

routerMethodRef(path, routeHandler);

Let’s define steps that are required to register route:

* Initially when we create nest app by calling **NestFactory.create(…)** we create **NestApplication**
* In constructor of our NestApplication we define some class named RoutesResolver
* RoutesResolver explore all of our modules, controller and register their routes using other class named RouterExplorer
* RouterExplorer is doing exactly as it is named, he explore routes and more exactly routes of our controller, below are some preparation that RouterExplorer do before registering routes:
  + Get instance of our controller
  + Extract all routes and methods defined in controller
* Once route-explorer have defined all methods he starts to parse each method, below are some transformations controller method (method that transform -> createCallbackProxy(…)):
  + Extract required metdata related to controller method and it’s params

*const handlerMetadata: HandlerMetadata = {*

*argsLength, // number of args of method*

*fnHandleResponse, // factory function for template method*

*paramtypes, // possible param types of method (@Body, @Query)*

*getParamsMetadata, // add factory to param which i call in pipes to get data*

*httpStatusCode, // code to return*

*hasCustomHeaders, // as named*

*responseHeaders, // as named*

*};*

* + Get all instances (local, global) of pipes, guards and interceptors associated to current controller method
  + Create a functions that will apply guards and pipes
  + Finally return a function that implements all interceptos header and status decorators, if exist template decorators and others and then calls our controller method
* Once our proxy method (controller method = handler) is defined we also extract all exception filters and join then in another handler that is our final function that applies all pipes (to method and it’s params), guards interceptors and exception filters.
* Once we get our proxy method we log all it’s possible paths (could be versioned or contain multiple paths) and finally register our method and it’s path
* At this moment we do next this (Code of each step is provided above):
  + Create **entrypointDefinition** object which will be cached in **graphInspector**
  + And register our route in it’s http adapter by providing path and our proxy (handler)
  + Under the hood of nestjs for express it looks like this:

app.get('/our-path, function ourHandler(req, res){

res.send("I am proxy function that!");

});