Node Hero – Node.js Request Module Tutorial

9 months ago

This is the 6th part of the tutorial series called Node Hero – in these chapters, you can learn how to get started with Node.js and deliver software products using it.

In the following tutorial, you will learn the basics of HTTP, and how you can fetch resources from external sources using the Node.js request module.

Upcoming and past chapters:

1. Getting started with Node.js
2. Using NPM
3. Understanding async programming
4. Your first Node.js HTTP server
5. Node.js database tutorial
6. Node.js request module tutorial *[you are reading it now]*
7. Node.js project structure tutorial
8. Node.js authentication using Passport.js
9. Node.js unit testing tutorial
10. Debugging Node.js applications
11. Node.js Security Tutorial
12. How to Deploy Node.js Applications
13. Monitoring Node.js Applications

What's HTTP?

HTTP stands for *Hypertext Transfer Protocol*. HTTP functions as a request–response protocol in the client–server computing model.

HTTP Status Codes

Before diving into the communication with other APIs, let's review the HTTP status codes we may encounter during the process. They describe the outcome of our requests and are essential for error handling.

- 1xx - Informational
- 2xx - Success: These status codes indicate that our request was received and processed correctly. The most common success codes are `200 OK`, `201 Created` and `204 No Content`.
- 3xx - Redirection: This group shows that the client had to do an additional action to complete the request. The most common redirection codes are `301 Moved Permanently`, `304 Not Modified`.
- 4xx - Client Error: This class of status codes is used when the request sent by the client was faulty in some way. The server response usually contains the explanation of the error. The most common client error codes are `400 Bad Request`, `401 Unauthorized`, `403 Forbidden`, `404 Not Found`, `409 Conflict`.
- 5xx - Server Error: These codes are sent when the server failed to fulfill a valid request due to some error. The cause may be a bug in the code or some temporary or permanent incapability. The most common server error codes

ests to External APIs

APIs is easy in Node. You can just require the core HTTP ing requests.

uch better ways to call an external endpoint. On NPM you ules that can make this process easier for you. For example, ones are the request and superagent modules.

Both of these modules have an error-first callback interface that can lead to some issues (I bet you've heard about Callback-Hell), but luckily we have access to the promise-wrapped versions.

Using the Node.js Request Module

Using the request-promise module is simple. After installing it from NPM, you just have to require it:

```
const request = require('request-promise')
```

If you are calling a JSON API, you may want the request-promise to parse the response automatically. In this case, just add this to the request options:

```
json: true
```

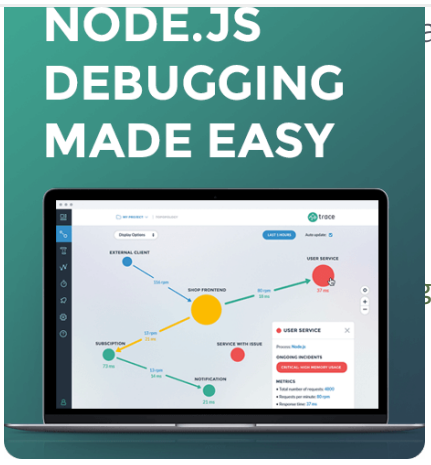
POST requests work in a similar way:

```
const options = {
  method: 'POST',
  uri: 'https://risingstack.com/login',
  body: {
    foo: 'bar'
  },
  json: true
    // JSON stringifies the body automatically
}

request(options)
  .then(function (response) {
    // Handle the response
  })
```


arameters you just have to add the `qs` property to the

gstack.com',




```
  }
}
```

This will make your request URL: `https://risingstack.com?limit=10&skip=20&sort=asc`. You can also define any header the same way we added the query parameters:

```
const options = {
  method: 'GET',
  uri: 'https://risingstack.com',
  headers: {
    'User-Agent': 'Request-Promise',
    'Authorization': 'Basic QWxhZGRpbjpPcGVuU2VzYW1l'
  }
}
```

Error handling

Error handling is an essential part of making requests to external APIs, as we can never be sure what will happen to them. Apart from our client errors the server may respond with an error or just send data in a wrong or inconsistent format. Keep these in mind when you try handling the response. Also, using `catch` for every request is a good way to avoid the external service crashing our server.

e going to create a small Express application that can render
nditions based on city names.

r API key, please visit their developer site)

```
re('express')
equest-promise')
('express-handlebars')

const app = express()
```

```
app.engine('.hbs', exphbs({
  defaultLayout: 'main',
  extname: '.hbs',
  layoutsDir: path.join(__dirname, 'views/layouts')
}))
app.set('view engine', '.hbs')
app.set('views', path.join(__dirname, 'views'))

app.get('/:city', (req, res) => {
  rp({
    uri: 'http://apidev.accuweather.com/locations/v1/search',
    qs: {
      q: req.params.city,
      apiKey: 'api-key'
         // Use your accuweather API key here
    },
    json: true
  })
    .then((data) => {
      res.render('index', data)
    })
    .catch((err) => {
```


es the following:

s server

ebars structure – for the `.hbs` file please refer to the Node.js

o the external API

is ok, it renders the page

shows the error page and logs the error

Get the whole **Node Hero** series as a single pdf and read it later.

Next up

In the next chapter of Node Hero you are going to learn how to structure your Node.js projects correctly.

In the meantime try out integrating with different API providers and if you run into problems or questions, don't hesitate to share them in the comment section!

Node.js nodejs nodehero npm tutorial node.js request request module

Read more from us:

- What's new in Node v6?
- Introducing Distributed Tracing for Microservices Monitoring

27 Comments RisingStack Blog 1 Login

Recommend 3 Share Sort by Best

Join the discussion…

daviddavis • 9 months ago

Hello, I'm also really enjoying this series. If it's not too much to ask could we get a look at what your index.hbs looks like that you render onto? Thanks.

1 • Reply • Share ›

jaybird1905 • 24 days ago

Can someone please provide the html needed to render this data blob using express-handlebars??? I'm getting the data back from the accuweather API but can't get it to display...

• Reply • Share ›

Nikhil Jauhari • 2 months ago

Writing server code in route/same file is not a good way here?

• Reply • Share ›

eva • 6 months ago

Cannot GET /

ylo ➚ eva • 5 months ago

alhost:3000, but you have to hit localhost:3000/city-name to get rid of the ET / because express doesn't know what to do with / request.
now what to do with /city-name request.

Reply • Share ›

6 months ago

myself just in case other folks have the same issue. Unlike npm, node ve a "global variable" where proxy settings can be, well, set.

e done application by application. I've found something on StackOverflow that works: add this to the sample code (and obviously install the required npm package).

```
var globalTunnel = require('global-tunnel');
```

```
globalTunnel.initialize({
host: 'proxy.example.com',
port: 8080
});
```

http://stackoverflow.com/quest...

I haven't managed to make the Express application work but at least I get error messages from Accuweather, meaning I managed to go through the proxy.

One step at a a time! (or keep swimming, as Dory would say)

Reply • Share ›

Ignatius Gonsalves • 8 months ago

path is not defined

Reply • Share ›

Mickola m ➔ Ignatius Gonsalves • 7 months ago

const path = require('path')

Reply • Share ›

Simmo • 9 months ago

Hi!
How would you solve the problem of retrieving (downloading) 1000 images ? :)
Thx

mmo ➔ Francis Kim • 8 months ago

anks Francis.. I think I would go for it ;)

Reply • Share ›

meth ➔ Simmo • 9 months ago

se the async module for that for example. With that you will have control urrency.

Reply • Share ›

nmo ➔ Gergely Németh • 9 months ago

hanks Gergely!
My challenge is to collect those URLs and schedule a download in chunck of X items.

Reply • Share ›

Redsmin {Redis GUI} • 9 months ago

You may also want to take a look at request-retry (https://github.com/FGRibreau/n...), it wraps request, adds promise support (works with any promise library, bluebird, whenjs, kew, RSVP, Q...) and handles network errors for you :)

Reply • Share ›

Poetro • 9 months ago

The request module can do so much more, then this request-promise. With the original request module you can pipe back data, process data as it is streaming and all other things that you could do with streams.

And for larger datasets it is always better to process data as it streams back, then wait for it for fully load, as it requires less memory, and the client can see the response as the data is loaded from the other end.

Reply • Share ›

Aditya Srivastava ➚ Poetro • 6 months ago

Hii poetro.. I had a doubt if you could help
I want to modify the response body content using request module before it is send to browser.
While using request module, can we give our localhost as url. (http://localhost:8000/index.html)???
I tried giving localhost url, it goes into infinite loop and keeps on printing the response body again and again..

➚ Poetro • 9 months ago

o, would you suggest using the request module with a promise in this
ry I'm a node noob.

Reply • Share ›

etro ➚ daviddavis • 9 months ago

sorry but I don't really like Promises.

Reply • Share ›

daviddavis ➚ Poetro • 9 months ago

Hey Poetro, would a solution for you look like this then, short and sweet?

request({

url: 'http://apidev.accuweather.com/...,

```
headers: {

'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87
Safari/537.36'

},

qs: {

q: req.params.city,

apiKey: 'api-key'
```

see more

• Reply • Share ›

Poetro ➚ daviddavis • 9 months ago

No, it would look something like:

```
request(options)
.on('error', handleError)
.on('response', handleResponseHeader)
.pipe(writableStream)
.on('error', handlePipeError)
.on('finish', handleFinish)
```

• Reply • Share ›

RisingStack

Never miss an update! your email Subscribe

…o

…long does it take to come a new episode?

…Share ›

…meth ➚ Saqib • 9 months ago

…ou can expect them coming on a weekly-basis / once in every two

…Reply • Share ›

…qib ➚ Gergely Németh • 9 months ago

…ould like them to come as soon as possible, once in every two weeks will be too late.

• Reply • Share ›

Philip J Cox • 9 months ago

Thanks for the tutorial, I am really liking this series. I understand that a quick overview with contrived examples can not always go into to detail, so as much as I enjoy reading them I am left thinking there is still a lot to consider for real world apps. If any notes and links can be added for other things we may want to take into consideration when building real world apps would be great, for example security issues and best practices. Thanks again!

Reply • Share ›

Ferenc Hamori ➚ Philip J Cox • 9 months ago

Hello Philip! Thanks for your kind words :) We'll try to cover the topics you mentioned - actually the 11th part of the Node Hero series will be about application security.

Also, check out our "How to be a better Node.js developer in 2016 blogpost" at https://blog.risingstack.com/h... if you want to deepen your knowledge of building real world app. It is based on our enterprise Node.js consulting and development experiences and filled with actionable know-how and good links.

Thanks, Ferenc

Reply • Share ›

Philip J Cox ➚ Ferenc Hamori • 9 months ago

Hi Ferenc. Thank you, for the reply and the link! I'll definitely check this out.

1 Reply • Share ›

Blog

Best articles

Node Hero Tutorials

Node.js at Scale Tutorials

Resources

Node.js Maturity Checklist

Trace by RisingStack – Node.js Monitoring

Node.js is Enterprise Ready

RisingStack Community

Node.js Daily

Connect

RisingStack.com

Twitter

Github

Facebook

RisingStack

Never miss an update! your email Subscribe