Getting binary content in node.js with http.request

Asked 11 years, 5 months ago Modified 3 years, 9 months ago Viewed 103k times

76

I would like to retrieve binary data from an https request.

I found a **similar question** that uses the request method, Getting binary content in Node.js using request, is says setting **encoding** to **null** should work, but it doesn't.

```
options = {
    hostname: urloptions.hostname,
    path: urloptions.path,
    method: 'GET',
    rejectUnauthorized: false,
    encoding: null
};

req = https.request(options, function(res) {
    var data;
    data = "";
    res.on('data', function(chunk) {
        return data += chunk;
    });
    res.on('end', function() {
        return loadFile(data);
    });
    res.on('error', function(err) {
        console.log("Error during HTTP request");
        console.log(err.message);
    });
})
```

Edit: setting encoding to **'binary'** doesn't work either

node.js http binary

Share

Improve this question

Follow

edited Dec 11, 2018 at 11:12

asked Jul 24, 2013 at 14:09

edi9999

20.5k ● 15 ● 97 ● 134

If you know the encoding you're attempting to apply to the data can you not fairly easily convert it to binary? I mean, it's a computer, you have no choice to but receive binary data... – MobA11y Jul 24, 2013 at 14:14

7 Answers

Sorted by: Highest score (default)

109

The accepted answer did not work for me (i.e., setting encoding to binary), even the user who asked the question mentioned it did not work.

Here's what worked for me, taken from: http://chad.pantherdev.com/node-js-binary-http-streams/

```
http.get(url.parse('http://myserver.com:9999/package'), function(res) {
    var data = [];

    res.on('data', function(chunk) {
        data.push(chunk);
    }).on('end', function() {
        //at this point data is an array of Buffers
        //so Buffer.concat() can make us a new Buffer
        //of all of them together
        var buffer = Buffer.concat(data);
        console.log(buffer.toString('base64'));
    });
});
```

Edit: Update answer following a suggestion by Semicolon

Share

Improve this answer

Follow

edited Sep 17, 2014 at 14:05

answered Jan 9, 2014 at 15:46

Guaycuru

1,340 ● 1 ● 11 ● 14

Came across this; for me, setting the encoding to null with the get() options does actually work. For reference, trying to set the encoding through defaults for the request module doesn't work. – TheDiveO Jan 31, 2018 at 17:07

This didn't work for me until I swapped 'end' with 'finish' – cpres Jan 31, 2019 at 0:42

1 If you don't have Buffer.concat() in your version of node, Buffer.from() accepts an array as well. – Spechal Jul 18, 2019 at 20:55

28

Running on NodeJS 6.10(and 8.10, tested in Feb 2019) in the AWS Lambda environment, none of the solutions above worker for me.

What did work for me was the following:

```
https.get(opt, (res) => {
    res.setEncoding('binary');
    let chunks = [];

    res.on('data', (chunk) => {
        chunks.push(Buffer.from(chunk, 'binary'));
    });

    res.on('end', () => {
```

```
        let binary = Buffer.concat(chunks);
        // binary is now a Buffer that can be used as Uint8Array or as
        // any other TypedArray for data processing in NodeJS or
        // passed on via the Buffer to something else.
    });
});
```

Take note the res.setEncoding('binary'); and Buffer.from(chunk, 'binary') lines. One sets the response encoding and the other creates a Buffer object from the string provided in the encoding specified previously.

Share
Improve this answer
Follow

edited Feb 22, 2019 at 7:35

answered Apr 1, 2018 at 17:51

Pärt Johanson
1,650 ● 1 ● 15 ● 15

21

You need to set encoding to response, not request:

```
req = https.request(options, function(res) {
    res.setEncoding('binary');

    var data = [ ];

    res.on('data', function(chunk) {
        data.push(chunk);
    });
    res.on('end', function() {
        var binary = Buffer.concat(data);
        // binary is your data
    });
    res.on('error', function(err) {
        console.log("Error during HTTP request");
        console.log(err.message);
    });
});
```

Here is useful answer: Writing image to local server

Share
Improve this answer
Follow

edited May 23, 2017 at 12:17

Community Bot
1 ● 1

answered Jul 24, 2013 at 14:15

moka
23k ● 4 ● 52 ● 67

3 this code says data needs to be an array of buffers, but there are strings (– makc May 1, 2016 at 22:26

8

1. Don't call `setEncoding()` method, because by default, no encoding is assigned and stream data will be returned as `Buffer` objects

2. Call `Buffer.from()` in `on.data` callback method to convert the `chunk` value to a `Buffer` object.

```
http.get('my_url', (response) => {
  const chunks = [];
  response.on('data', chunk => chunks.push(Buffer.from(chunk))) // Converte
`chunk` to a `Buffer` object.
    .on('end', () => {
      const buffer = Buffer.concat(chunks);
      console.log(buffer.toString('base64'));
    });
});
```

Share Improve this answer Follow

answered Apr 15, 2019 at 10:37

Naijia

338 ● 4 ● 6

8

Pärt Johanson I wish I could comment just to thank you for saving me from the recursive loop I've been in all day of ripping my hair out and then reading the (incredibly unhelpful) node docs on this, over, and over. Upon finding your answer, I went to dig into the docs, and I can't even find the `res.setEncoding` method documented anywhere! It's just shown as part of two examples, wherein they call `res.setEncoding('utf8');` Where did you find this or how did you figure it out!?

Since I don't have enough reputation to comment, I'll at least contribute something useful with my answer: Pärt Johanson's answer worked 100% for me, I just tweaked it a bit for my needs because I'm using it to download and eval a script hosted on my server (and compiled with nwjc) using `nw.Window.get().evalNWBin()` on NWJS 0.36.4 / Node 11.11.0:

```
let opt = {...};
let req = require('https').request(opt, (res) => {
  // server error returned
  if (200 !== res.statusCode) {
    res.setEncoding('utf8');
    let data = '';
    res.on('data', (strData) => {
      data += strData;
    });
    res.on('end', () => {
      if (!res.complete) {
        console.log('Server error, incomplete response: ' + data);
      } else {
        console.log('Server error, response: ' + data);
      }
    });
  }
  // expected response
  else {
    res.setEncoding('binary');
```

```
    let data = [];
    res.on('data', (binData) => {
      data.push(Buffer.from(binData, 'binary'));
    });
    res.on('end', () => {
      data = Buffer.concat(data);
      if (!res.complete) {
        console.log('Request completed, incomplete response, ' + data.length +
' bytes received');
      } else {
        console.log('Request completed, ' + data.length + ' bytes received');
        nw.Window.get().evalNWBin(null, data);
      }
    });
  }
};
```

Edit: P.S. I posted this just in case anyone wanted to know how to handle a non-binary response -- my actual code goes a little deeper and checks response content type header to parse JSON (intended failure, i.e. 400, 401, 403) or HTML (unexpected failure, i.e. 404 or 500)

Share

Improve this answer

Follow

edited Mar 18, 2021 at 13:05

Matt Hudson

7,349 ● 7 ● 51 ● 68

answered Mar 15, 2019 at 21:12

caffeinatedbits

491 ● 5 ● 9

3

Everyone here is on the right track, but to put the bed the issue, you **cannot** call `.setEncoding()` EVER.

If you call `.setEncoding()`, it will create a `StringDecoder` and set it as the default decoder. If you try to pass `null` or `undefined`, then it will still create a `StringDecoder` with its default decoder of `UTF-8`. Even if you call `.setEncoding('binary')`, it's the same as calling `.setEncoding('latin1')`. Yes, seriously.

I wish I could say you set `._readableState.encoding` and `_readableState.decoder` back to `null`, but when you call `.setEncoding()` buffer gets wiped and replaced with a binary encoding of the decoded string of what was there before. That means your data has already been changed.

If you want to "undo" the decoding, you have to re-encode the data stream back into binary like so:

```
  req.on('data', (chunk) => {
      let buffer;
      if (typeof chunk === 'string') {
        buffer = Buffer.from(chunk, req.readableEncoding);
      } else {
```

```
        buffer = chunk;
      }
      // Handle chunk
  });
```

Of course, if you never call `.setEncoding()`, then you don't have to worry about the chunk being returned as a `string`.

After you have a your chunk as `Buffer`, then you can work with it as you chose. In the interested of thoroughness, here's how to use with a preset buffer size, while also checking `Content-Length`:

```
const BUFFER_SIZE = 4096;

/**
 * @param {IncomingMessage} req
 * @return {Promise<Buffer>}
 */
function readEntireRequest(req) {
  return new Promise((resolve, reject) => {
    const expectedSize = parseInt(req.headers['content-length'], 10) || null;
    let data = Buffer.alloc(Math.min(BUFFER_SIZE, expectedSize ||
BUFFER_SIZE));
    let bytesWritten = 0;
    req.on('data', (chunk) => {
      if ((chunk.length + bytesWritten) > data.length) {
        // Buffer is too small. Double it.
        let newLength = data.length * 2;
        while (newLength < chunk.length + data.length) {
          newLength *= 2;
        }
        const newBuffer = Buffer.alloc(newLength);
        data.copy(newBuffer);
        data = newBuffer;
      }
      bytesWritten += chunk.copy(data, bytesWritten);
      if (bytesWritten === expectedSize) {
        // If we trust Content-Length, we could return immediately here.
      }
    });
    req.on('end', () => {
      if (data.length > bytesWritten) {
        // Return a slice of the original buffer
        data = data.subarray(0, bytesWritten);
      }
      resolve(data);
    });
    req.on('error', (err) => {
      reject(err);
    });
  });
}
```

The choice to use a buffer size here is to avoid immediately reserving a large amount of memory, but instead only fetch RAM as needed. The `Promise` functionality is just

for convenience.

Share

Improve this answer

Follow

edited May 26, 2020 at 20:45

answered May 26, 2020 at 20:16

ShortFuse
6,734 ● 3 ● 38 ● 39

2

As others here, I needed to process binary data chunks from Node.js HTTP response (aka `http.IncomingMessage`).

None of the existing answers really worked for my Electron 6 project (bundled with Node.js 12.4.0, at the time of posting), besides Pärt Johanson's answer and its variants.

Still, even with that solution, the chunks were always arriving at the `response.on('data', ondata)` handler as `string` objects (rather than expected and desired `Buffer` objects). That incurred extra conversion with `Buffer.from(chunk, 'binary')`. I was getting strings regardless of whether I explicitly specified binary encoding with `response.setEncoding('binary')` or `response.setEncoding(null)`.

The only way I managed to get the original `Buffer` chunks was **to pipe the `response` to an instance of `stream.Writable` where I provide a custom `write` method**:

```
const https = require('https');
const { Writable } = require('stream');

async function getBinaryDataAsync(url) {
  // start HTTP request, get binary response
  const { request, response } = await new Promise((resolve, reject) => {
    const request = https.request(url, {
      method: 'GET',
        headers: {
          'Accept': 'application/pdf',
          'Accept-Encoding': 'identity'
        }
      }
    );

    request.on('response', response =>
      resolve({request, response}));
    request.on('error', reject);
    request.end();
  });

  // read the binary response by piping it to stream.Writable
  const buffers = await new Promise((resolve, reject) => {

    response.on('aborted', reject);
    response.on('error', reject);

    const chunks = [];

    const stream = new Writable({
```

```
      write: (chunk, encoding, notifyComplete) => {
        try {
          chunks.push(chunk);
          notifyComplete();
        }
        catch(error) {
          notifyComplete(error);
        }
      }
    });

    stream.on('error', reject);
    stream.on('finish', () => resolve(chunks));
    response.pipe(stream);
  });

  const buffer = Buffer.concat(buffers);
  return buffer.buffer; // as ArrayBuffer
}

async function main() {
  const arrayBuff = await
getBinaryDataAsync('https://download.microsoft.com/download/8/A/4/8A48E46A-
C355-4E5C-8417-E6ACD8A207D4/VisualStudioCode-TipsAndTricks-Vol.1.pdf');
  console.log(arrayBuff.byteLength);
};

main().catch(error => console.error(error));
```

Updated, as it turns, this behavior only manifests for our Web API server. So, `response.on('data')` actually works well for the sample URL I use in the above code snippet and the stream is not needed for it. It's weird though this is sever-specific, I'm investigating it further.

Share

Improve this answer

Follow

edited Sep 18, 2019 at 6:00

answered Sep 15, 2019 at 22:13

noseratio

61.6k ● 36 ● 223 ● 500