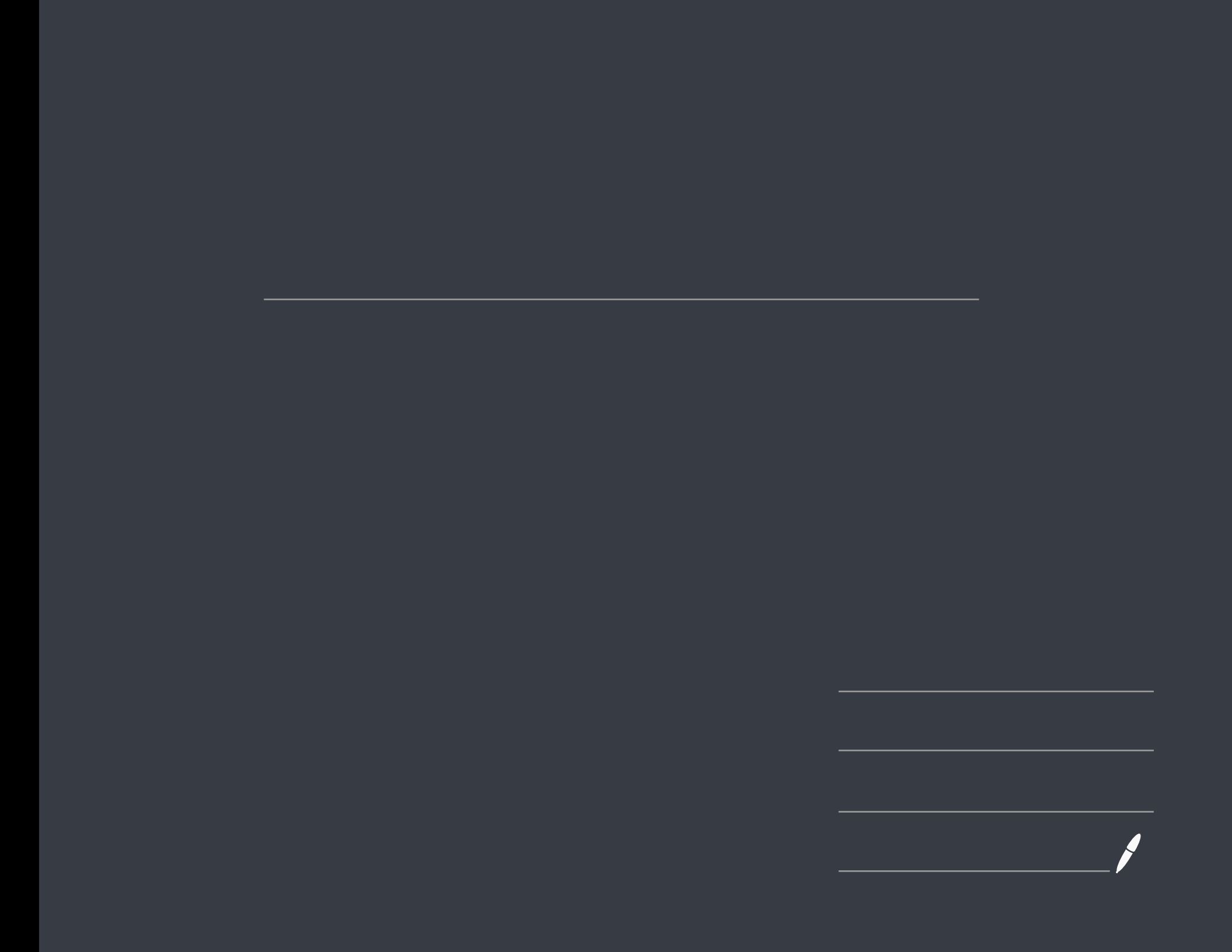

How bodyParser() works

Adam Zerner Jul 2, 2015 · 4 min read

When I first was learning Express, I had `app.js` written for me and I just had to fill in a few routes. Eg.

```
app.get('/', function(req, res) {
  // get posts
  res.json(posts)
});
```

Then, one day I was applying to a job and had to set up Node and Express *from scratch!*

I tried writing the create route like I normally would…

```
app.post('/', function(req, res) {
  Post.create(req.body)
});
```

… but it wasn't working! It said something about `req.body` not existing. Huh? Why is that? I could have sworn that I did everything properly.

. . .

The next thing I did was I tried to inspect `req` itself. I tried to find the data that was POSTed in the `req` object. But the `req` object was big and confusing and I couldn't really find the data I was looking for. That approach didn't work.

Eventually, I came across `bodyParser()` !!! I learned that:

> *You need to use bodyParser() if you want the form data to be available in req.body.*

Top highlight

But I had no clue how this happens. I still don't really understand, so I'm writing this article to help me figure it out.

app.use()

First things first — to use `bodyParser()` you do something like this:

```
app.use(bodyParser.json());
```

To understand how this works, you have to understand how middleware works in Express. In particular, when app.use() is called with a function as it's only argument:

```
app.use(function(req, res) {
  // stuff
});
```

1. It'll match every request.
2. That function will act as middleware.

Contrast this with when `app.use()` is called with the a signature of the form:

```
app.use('/test', cb);
```

It'll only match requests that start with `/test`.

Anyway, `bodyParser.json()` returns a function, and when that function is passed into app.use, it acts just like any other middleware. It may be helpful to think about it like this:

```
var cb = bodyParser.json();
app.use(cb);
```

. . .

Under the Hood

> *Ok Adam, enough with the BS. I understand how middlware works. But when I don't use bodyParser and I log out the* `req` *object, I just get this huge thing and can't find my data anywhere. I understand how a middlware could do something like* `req.body = constructBody(req.data)`*, but I can't find* `req.data`*!!*

If you're asking that question, you've come to the right place! It confused me for a while too. If you're like me, this is the kind of thing that you can't "just accept". I could accept that if the data is there, you could perform some sort of manipulations on it to get `req.body`. I don't have to understand exactly what those manipulations are. But if the data *isn't even there*, how the hell can you perform the manipulations and get `req.body`?!!

Let me show you.

The short answer is that it works something like this:

```
app.use(function( req, res, next ) {
  var data = '';
  req.on('data', function( chunk ) {
    data += chunk;
  });
  req.on('end', function() {
    req.rawBody = data;
    console.log( 'on end: ', data )
    if (data && data.indexOf('{') > -1 ) {
      req.body = JSON.parse(data);
    }
    next();
  });
});
```

The long answer is that you have to understand *streams*. To understand streams, you have to understand that information is sent across the internet via *packets*.

I assume that you watched a bit of the video and have an idea how packets work. The next step is to understand the stream abstraction that you'll work with in Node.

The idea with streams is that you could do something like this:

```
req.on('data', function(chunk) {
  // here's the chunk
});
```

Each time a chunk of the data comes in, you get to use it. So the string

```
abcdefghijklmnopqrstuvwxyz
```

Might come in 5 chunks

```
abcde
fghij
klmno
pqrst
uvwxyz
```

and you'd be able to access each chunk.

To my understanding, if you want to access the post data, you have to get it from the stream. Ie. it won't just be available on the `req` object.

Check out https://www.github.com/substack/stream-adventure to get a better understanding of how streams work.

. . .

bodyParser

At this point, you should have a solid idea of bodyParser does. It returns a function that acts as middleware. The function listens for `req.on('data')` and constructs `req.body` from the chunks of data it gets.

But there are a few caveats. Basically, there are a bunch of different ways to format the data you POST to the server:

- `application/x-www-form-urlencoded`
- `multipart/form-data`
- `application/json`
- `application/xml`
- maybe some others

In brief, bodyParser has to parse the data differently depending on its type (read here about the differences between the first two). And so you need to do something like this (from Express docs):

```
app.use(bodyParser.json()); // for parsing application/json
app.use(bodyParser.urlencoded({ extended: true })); // for parsing
application/x-www-form-urlencoded
app.use(multer()); // for parsing multipart/form-data
```

Note that there used to be a bodyParser() constructor you could call that would do it all for you, but now you have to call specific methods.

```
You, a day ago | 1 author (You)
const fs = require('fs');

const requestHandler = (req, res) => {
  const url = req.url;
  const method = req.method;
  if (url === '/') {
    res.write('<html>');
    res.write('<head><title>Enter Message</title></head>');
    res.write(
      '<body><form action="/message" method="POST"><input type="text" name="message"><button type="submit">Send</button></form></body>'
    );
    res.write('</html>');
    return res.end();
  }

  if (url === '/message' && method === 'POST') {
    const body = [];
    req.on('data', chunk => {
      console.log(chunk);
      body.push(chunk);
    });
    return req.on('end', () => {
      const parsedBody = Buffer.concat(body).toString();
      const message = parsedBody.split('=')[1];
      fs.writeFile('message.txt', message, err => {
        res.statusCode = 302;
        res.setHeader('Location', '/');
        return res.end();
      });
    });
  }      You, a day ago • added missing section 3 code
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>My First Page</title><head>');
  res.write('<body><h1>Hello from my Node.js Server!</h1></body>');
  res.write('</html>');
  res.end();
};

```

Because Internet send the data chunk by chunk, you have to combine them together and then either buffer them or Json.parse like blog article says. Or you can use parse-body

```
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({extended: false}));

// /admin/add-product => POST
router.post('/add-product', (req, res, next) => {
  products.push({ title: req.body.title });
  res.redirect('/');
});
```