[1. Form表单中用来处理文件上传 2](#_Toc418870955)

[2. 用Node和Express 来处理文件上传 2](#_Toc418870956)

[3. Multer 官网 3](#_Toc418870957)

[4. 安装multer 4](#_Toc418870958)

[5. Multer 的目标文件夹 6](#_Toc418870959)

[6. 打印出的req.files的内容 7](#_Toc418870960)

[7. How to Handle File Upload with Node and Express 4.0 7](#_Toc418870961)

[7-1. How to Handle File Upload with Node and Express 4.0 7](#_Toc418870962)

# Form表单中用来处理文件上传



# 用Node和Express 来处理文件上传

<http://stackoverflow.com/questions/23114374/file-uploading-with-express-4-0-req-files-undefined>



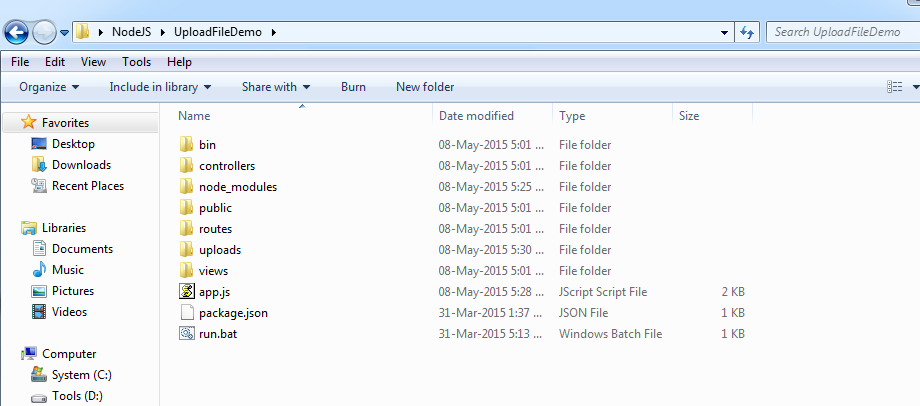
The [body-parser](https://github.com/expressjs/body-parser) module only handles JSON and urlencoded form submissions, not multipart (which would be the case if you're uploading files).

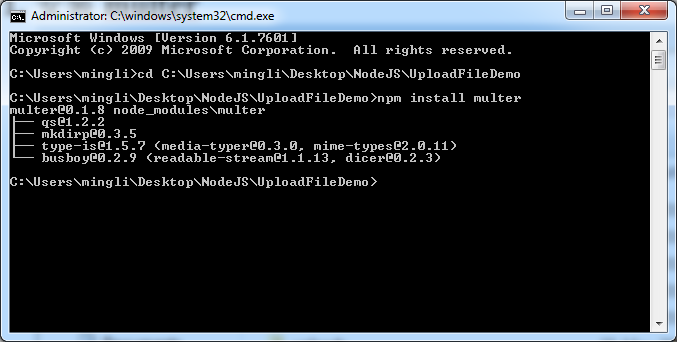
For multipart, you'd need to use something like [connect-busboy](https://github.com/mscdex/connect-busboy) or [multer](https://github.com/expressjs/multer) or [connect-multiparty](https://github.com/andrewrk/connect-multiparty)(multiparty/formidable is what was originally used in the express bodyParser middleware). Also FWIW, I'm working on an even higher level layer on top of busboy called [reformed](https://github.com/mscdex/reformed). It comes with an Express middleware and can also be used separately.

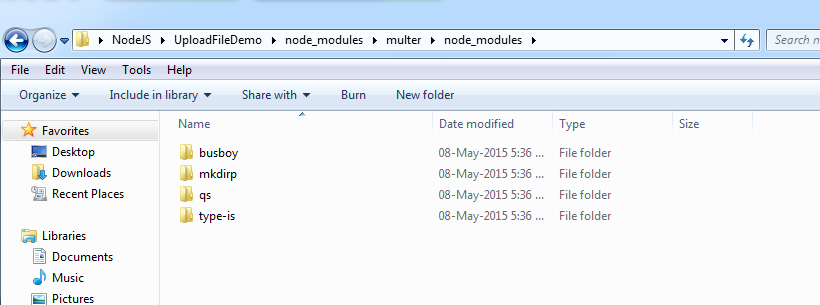
# Multer 官网

https://github.com/expressjs/multer

# 安装multer



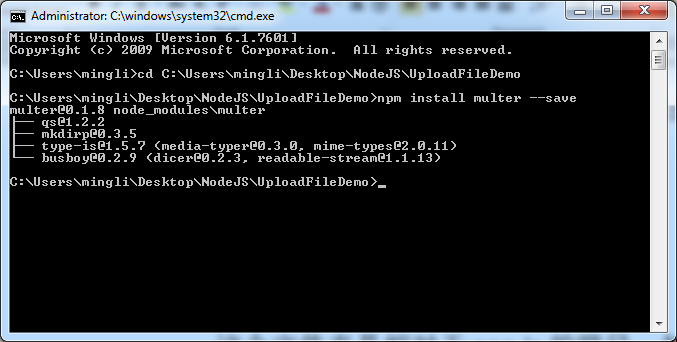


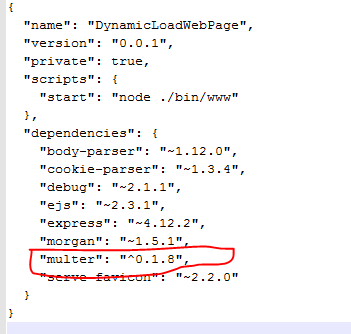


直接cmd进到跟目录，即app.js所在的目录

执行 npm install multer即可

也可以加上- - save 表示，这样可以把multer自动加入到package.json的dependencies中去

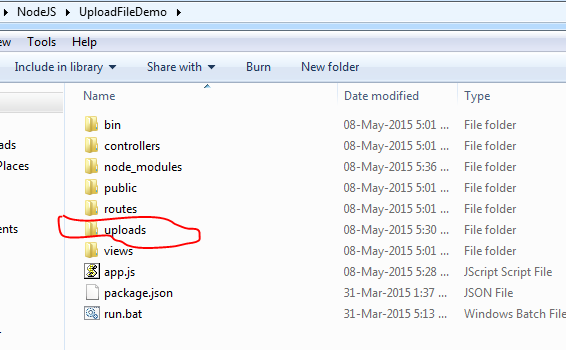




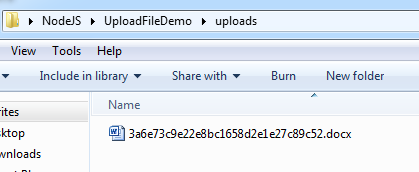
# Multer 的目标文件夹

app.use(multer({ dest: './uploads/'}))

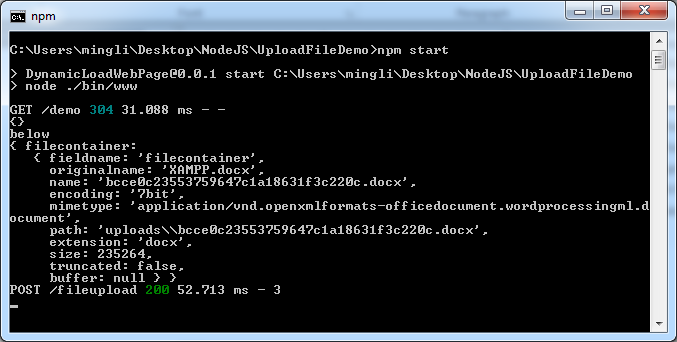
这个文件夹是相对于app.js的路径。如果不存在，会自动创建。



上传的文件会自动传到这个目录下



# 打印出的req.files的内容



其中filecontainer是form中input 标签的name，原先上传的文件是xampp.docx

# How to Handle File Upload with Node and Express 4.0

## [How to Handle File Upload with Node and Express 4.0](http://www.e-zest.net/blog/how-to-handle-file-upload-with-node-and-express-4-0/)

File upload is a critical part of many web applications. While it was easy to handle uploads with Express, the most popular web framework for Node, some breaking changes in Express 4.0 make it a little difficult. What adds to it is that most guides on express predate this change! Here in this blog, we shall see how to deal with uploads using the newer version.

**The Problem:**

In Express 4.0, Connect is no more a dependency. Besides Connect has already marked multipart () middleware deprecated and be removed in v3.0. But other middleware (don’t bother, it’s just a fancy word for a module that does some processing on the request) in Connect are available as separate modules which we can install individually and use in our app. Well, that may streamline a lot of things for us like individual module updates, but that also makes all the previous methods of working with Express for file upload obsolete.

**The Solution:**

This ‘How to’, expects very basic knowledge of Node.js and none of Express! We shall begin with installing Express, create an express app and then run it. After which we shall add file upload capability using multer package.

So let’s begin.

Assumptions:

1. Node and npm are installed and setup on your machine.
2. Dragons and Unicorns don’t exist.

Now open terminal and hit:

|  |
| --- |
| npm install -g express-generator |

Ok, now in the good old days you would install express, not the generator, but then the express CLI (command line interface) used to come bundled with it, but now, no more. Adding that -g is going to install it globally i.e. typing express in your terminal won’t ask you “huh?”

Now switch to your working directory, a directory you wish to contain your express web app in and type

|  |
| --- |
| express myApp |

This should create a directory by the name “myApp” and a few files and folders under it, should also show you all the files and folders it created in terminal output. Well, it also tells us what to do next. So let us take the hint:

|  |
| --- |
| cd myApp && npm install |

Install what? Nah, it’s not gonna ask you, cause the generator has already created the package.json file and added all the express and express dependencies in it. So npm will just pick up the list and install all it was asked about and show you a tree of every dependency it installed.

Well, the generator may have told us a complex command to run the app, but we don’t listen to computers all the time, so let’s just type:

|  |
| --- |
| npm start |

And the application should now start. Open your browser and go to the URL: http://localhost:3000 and express should welcome you. But how did that work? The generator mentions in the package .json that our app has a script called ‘start’, and also the command for it is “node ./bin/www”. If you open /bin/www, it’s just like any other node module, but declares that it is a node module using hashbang directive. The file also mentions that the app by default shall listen on port 3000.

Well, that was easy. But hey, you have an app running! Congratulations!!

But we still have a long way to go:

Go to URL: http://localhost:3000/users, what happened? How? No, I am not gonna explain that, it’s dead easy, just open app.js file in myApp folder and read through it. You may also have to open a couple of other files, like routes/users.js, but that is good. If we are going to build further, we should know what we already have!

Let’s move on to our next objective, file upload.

We shall create a new route, /upload for handling files, include it in our app. But a file is not uploaded with the regular content type “text/html” of get or “application/x-www-form-urlencoded” of post requests, but with “multipart/form-data” for post requests. Now what this means for us is: the request will be received in parts, shall contain binary or non-ASCII data as a stream. So either we can write the code for handling each part of the request as it is received and process it further to make sense out of it, or we can use a library that does all of this for us.

So we are going to use “[multer](https://www.npmjs.org/package/multer" \t "blank)”, a middleware (!) that handles “multipart/form-data” and magically (Is our assumption correct then?) makes the uploaded files and form data available to us in request as request.files and request.body. So let’s go to the myApp directory in terminal and type:

|  |
| --- |
| npm install multer --save |

This tells npm to install multer and also ‘save’ it as a dependency of our application, i.e. package.json.

Now let’s tell express app to use multer to process requests. Open the app.js file in a text editor and add the following line among other require statements already present.

|  |
| --- |
| var multer = require('multer'); |

Now that we have the module, let’s use it:

|  |
| --- |
| app.use(multer({  dest: “./uploads/”  })); |

This line goes immediately after the line where we say app.use(bodyParser.urlencoded()). It tells the app to use multer to process multi-part requests and multer to save our files by default to /uploads/ directory. There are many other options that multer allows us to pass, some are important from a security perspective, to block an attacker from uploading huge files and crash the application, but for now, we are going to keep it simple. But I urge you to read through the documentation and take necessary steps before you take this app to production.

So far, so good. But we still haven’t written anything as to what should happen when we visit the /uploads URL. First let’s create a page we want to be shown when a user visits that URL, let’s call it uploadPage. Create a file: uploadPage.jade in folder /views/ and add the following code to it:

|  |
| --- |
| **extends** layout  block content  h1= title  form#fileUpload(method="post" action="/uploads/upload" enctype="multipart/form-data")  label(**for**="payload") Select a file to upload:  input#payload(type="file" name="myFile" accept="image/\*")  br  button#upload Upload |

What we are saying here is, this page uses the contents of the layout.jade file and has a form and a field for uploading file and a button that submits the form. (Yes, I know, &nbsp; is not elegant.)

But how will the user see this page? Let us create a router, create a file “uploads.js” in /routes/ directory and the following code to it.

|  |
| --- |
| var express = require('express');  var router = express.Router();  var util = require("util");  var fs = require("fs");    router.get('/', function(req, res) {  res.render("uploadPage", {title: "I love files!"});  });    router.post("/upload", function(req, res, next){  **if** (req.files) {  console.log(util.inspect(req.files));  **if** (req.files.myFile.size === 0) {  **return** next(**new** Error("Hey, first would you select a file?"));  }  fs.exists(req.files.myFile.path, function(exists) {  **if**(exists) {  res.end("Got your file!");  } **else** {  res.end("Well, there is no magic for those who don’t believe in it!");  }  });  }  });  module.exports = router; |

Now we have to tell the app to use this router. In app.js, add the following line in require statements:

|  |
| --- |
| var uploads = require('./routes/uploads'); |

And after route for /users, add:

|  |
| --- |
| app.use(“/uploads”, uploads); |

Cool, now let’s start the app (npm start) and visit: http://localhost:3000/uploads in the browser. See the upload form? Well, upload a file, what are you waiting for?

Oh wait, did we create the “/uploads/” directory? Nope!! But don’t worry, the directory shall be created and file uploaded.

The first, get the method in the router caters to the browser’s request to show the page and paints our uploadPage, with upload form. The action for the upload form is catered to by the next, post method in the router. It checks if the request.files exists. Now, it shall exist even if no file was uploaded, only that it will have some null values. You can see the details in both cases on the console. What if a notorious user just clicked the upload button without actually selecting a file? For that, we check the size, if 0, we have caught them! Now if the file was uploaded, we can check that with the fs.exists() check. Now how you handle either scenario is up to your requirements.

But hey, am sure you know how to handle file upload with express 4.0+.