npm

sign up or log in

find packages

multer public

Middleware for handling <code>multipart/form-data</code>.

Multer is a node.js middleware for handling `multipart/form-data`, which is primarily used for uploading files. It is written on top of **busboy** for maximum efficiency.

NOTE: Multer will not process any form which is not multipart (`multipart/form-data`).

# Installation

```
$ npm install --save multer
```

# Usage

Multer adds a `body` object and a `file` or `files` object to the `request` object. The `body` object contains the values of the text fields of the form, the `file` or `files` object contains the files uploaded via the form.

Basic usage example:

npm Enterprise

Private npm for your enterprise. Start a free trial »

```
npm install multer
```

how? learn more

linusu published 6 mon...

1.1.0 is the latest of 23 relea...

github.com/expressjs/mult...

MIT

Collaborators

hac linu jpflu

Stats

14,087 downloads in the la...

79,085 downloads in the la...

307,690 downloads in the l...

43 open issues on GitHub

7 open pull requests on Git...

Try it out

```
var express = require('express')
var multer  = require('multer')
var upload = multer({ dest: 'uploads/' })

var app = express()

app.post('/profile', upload.single('avatar')
  // req.file is the `avatar` file
  // req.body will hold the text fields, if
})

app.post('/photos/upload', upload.array('pho
  // req.files is array of `photos` files
  // req.body will contain the text fields,
})

var cpUpload = upload.fields([{ name: 'avata
app.post('/cool-profile', cpUpload, function
  // req.files is an object (String -> Array
  //
  // e.g.
  //  req.files['avatar'][0] -> File
  //  req.files['gallery'] -> Array
  //
  // req.body will contain the text fields,
})
```

In case you need to handle a text-only multipart form, you can use any of the multer methods (`.single()`, `.array()`, `fields()`). Here is an example using `.array()`:

Test multer in your bro...

Keywords

middleware, express, formdata, form-data, multipart, post, form

Dependencies (8)

xtend, type-is, on-finished, object-assign, mkdirp, concat-stream, busboy, append-field

Dependents

oc, apiculi, structor, chen, widget-cms, ezajax, @nxus/data-loader, blueoak-server, gscan, ghost-node6, jii-httpserver, periodicjs, kila-app, we-core, newbeely-nodejs, keystone, webdriverio-server, jsonreststores, @onehilltech/blueprint, bolto, secc, micro-base, node-mydomoathome, codered-steganography, pk-app-pkgmgt, @m1r4ge/parse-server, autohost, hackable, better-transfer, nodecg, ghost, netbeast-cli, declaire, twreporter-keystone, triggers-service, blaze-mistar, koa-multer, jsreport-express, microgateway, carbon-framework, tsbot, parse-server-transform, apiconnect-microgateway, easyexpress,

```
var express = require('express')
var app = express()
var multer  = require('multer')
var upload = multer()

app.post('/profile', upload.array(), functio
  // req.body contains the text fields
})
```

ceres-framework, n-landing, eazyajax4js, Server, parse-server-hotfix, visionr-server, and more

microapps is hiring. View more…

# API

# File information

Each file contains the following information:

key	description	note
`fieldname`	Field name specified in the form	
`originalname`	Name of the file on the user's computer	
`encoding`	Encoding type of the file	
`mimetype`	Mime type of the file	
`size`	Size of the file in bytes	
`destination`	The folder to which the file has been saved	`DiskStorage`
`filename`	The name of the file within the `destination`	`DiskStorage`
`path`	The full path to the uploaded file	`DiskStorage`
`buffer`	A `Buffer` of the entire file	`MemoryStorage`

# multer(opts)

Multer accepts an options object, the most basic of which is the `dest` property, which tells Multer where to upload the files. In case you omit the options object, the files will be kept in memory and never written to disk.

By default, Multer will rename the files so as to avoid naming conflicts. The renaming function can be customized according to your needs.

The following are the options that can be passed to Multer.

key	description
`dest` or `storage`	Where to store the files
`fileFilter`	Function to control which files are accepted
`limits`	Limits of the uploaded data

In an average web app, only `dest` might be required, and configured as shown in the following example.

```
var upload = multer({ dest: 'uploads/' })
```

If you want more control over your uploads, you'll want to use the `storage` option instead of `dest`. Multer ships with storage engines `DiskStorage` and `MemoryStorage`; More engines are available from third parties.

# .single(fieldname)

Accept a single file with the name `fieldname`. The single file will be stored in `req.file`.

# .array(fieldname[, maxCount])

Accept an array of files, all with the name `fieldname`. Optionally error out if more than `maxCount` files are uploaded. The array of files will be stored in `req.files`.

# .fields(fields)

Accept a mix of files, specified by `fields`. An object with arrays of files will be stored in `req.files`.

`fields` should be an array of objects with `name` and optionally a `maxCount`. Example:

```
[
  { name: 'avatar', maxCount: 1 },
  { name: 'gallery', maxCount: 8 }
]
```

# .any()

Accepts all files that comes over the wire. An array of files will be stored in `req.files`.

WARNING: Make sure that you always handle the files that a user uploads. Never add multer as a global middleware since a malicious user could upload files to a route that you didn't anticipate. Only use this function on routes where you are handling the uploaded files.

# storage

# DiskStorage

The disk storage engine gives you full control on storing files to disk.

```
var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, '/tmp/my-uploads')
  },
  filename: function (req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now
  }
})

var upload = multer({ storage: storage })
```

There are two options available, `destination` and `filename`. They are both functions that determine where the file should be stored.

`destination` is used to determine within which folder the uploaded files should be stored. This can also be given as a `string` (e.g. `'/tmp/uploads'`). If no `destination` is given, the operating system's default directory for temporary files is used.

Note: You are responsible for creating the directory when providing `destination` as a function. When passing a string, multer will make sure that the directory is created for you.

`filename` is used to determine what the file should be named inside the folder. If no `filename` is given, each file will be given a random name that doesn't include any file extension.

Note: Multer will not append any file extension for you, your function should return a filename complete with an file extension.

Each function gets passed both the request (`req`) and some information about the file (`file`) to aid with the decision.

Note that `req.body` might not have been fully populated yet. It depends on the order that the client transmits fields and files to the server.

# MemoryStorage

The memory storage engine stores the files in memory as `Buffer` objects. It doesn't have any options.

```
var storage = multer.memoryStorage()
var upload = multer({ storage: storage })
```

When using memory storage, the file info will contain a field called `buffer` that contains the entire file.

WARNING: Uploading very large files, or relatively small files in large numbers very quickly, can cause your application to run out of memory when memory storage is used.

# limits

An object specifying the size limits of the following optional properties. Multer passes this object into busboy directly, and the details of the properties can be found on busboy's page.

The following integer values are available:

key	description	default
`fieldNameSize`	Max field name size	100 bytes
`fieldSize`	Max field value size	1MB
`fields`	Max number of non-file fields	Infinity
`fileSize`	For multipart forms, the max file size (in bytes)	Infinity
`files`	For multipart forms, the max number of file fields	Infinity
`parts`	For multipart forms, the max number of parts (fields + files)	Infinity
`headerPairs`	For multipart forms, the max number of header key=>value pairs to parse	2000

Specifying the limits can help protect your site against denial of service (DoS) attacks.

# fileFilter

Set this to a function to control which files should be uploaded and which should be skipped. The function should look like this:

```
function fileFilter (req, file, cb) {

  // The function should call `cb` with a bo
  // to indicate if the file should be accep

  // To reject this file pass `false`, like
  cb(null, false)

  // To accept the file pass `true`, like so
  cb(null, true)

  // You can always pass an error if somethi
  cb(new Error('I don\'t have a clue!'))

}
```

# Error handling

When encountering an error, multer will delegate the error to express. You can display a nice error page using the standard express way.

If you want to catch errors specifically from multer, you can call the middleware function by yourself.

```
var upload = multer().single('avatar')

app.post('/profile', function (req, res) {
  upload(req, res, function (err) {
    if (err) {
      // An error occurred when uploading
      return
    }

    // Everything went fine
  })
})
```

# Custom storage engine

See the documentation here if you want to build your own storage engine.

# License

MIT