文件上传

multer

中文介绍

Multer 是一个 node.js 中间件，用于处理 `multipart/form-data` 类型的表单数据，它主要用于上传文件。它是写在 busboy 之上非常高效。

注意: Multer 不会处理任何非 `multipart/form-data` 类型的表单数据。

使用

Multer 会添加一个 `body` 对象 以及 `file` 或 `files` 对象 到 express 的 `request` 对象中。 `body` 对象包含表单的文本域信息， `file` 或 `files` 对象包含对象表单上传的文件信息。

最基本使用方法 接收用户上传的所有文件

```
const multer=require('multer');
var objMulter=multer({dest: './www/upload/'});
var app=express();
// 接受任何文件
app.use(objMulter.any());
server.post('/', function (req, res){
    //所有上传的文件信息都保存在req.files中
    console.log(req.files);
})
```

基本使用方法:

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

var app = express()

app.post('/profile', upload.single('avatar'), function (req, res, next) {
  // req.file 是 `avatar` 文件的信息
  // req.body 将具有文本域数据，如果存在的话
})

app.post('/photos/upload', upload.array('photos', 12), function (req, res, next) {
  // req.files 是 `photos` 文件数组的信息
  // req.body 将具有文本域数据，如果存在的话
})

var cpUpload = upload.fields([{ name: 'avatar', maxCount: 1 }, { name: 'gallery',
maxCount: 8 }])
app.post('/cool-profile', cpUpload, function (req, res, next) {
  // req.files 是一个对象 (String -> Array) 键是文件名，值是文件数组
  //
  // 例如:
  //  req.files['avatar'][0] -> File
  //  req.files['gallery'] -> Array
  //
  // req.body 将具有文本域数据，如果存在的话
})
```

API

文件信息

每个文件具有下面的信息:

Key	Description	Note
`fieldname`	Field name 由表单指定	
`originalname`	用户计算机上的文件的名称	
`encoding`	文件编码	
`mimetype`	文件的 MIME 类型	
`size`	文件大小（字节单位）	
`destination`	保存路径	`DiskStorage`
`filename`	保存在 `destination` 中的文件名	`DiskStorage`
`path`	已上传文件的完整路径	`DiskStorage`
`buffer`	一个存放了整个文件的 `Buffer`	`MemoryStorage`

`multer(opts)`

Multer 接受一个 options 对象，其中最基本的是 `dest` 属性，这将告诉 Multer 将上传文件保存在哪。如果你省略 options 对象，这些文件将保存在内存中，永远不会写入磁盘。

为了避免命名冲突，Multer 会修改上传的文件名。这个重命名功能可以根据您的需要定制。

以下是可以传递给 Multer 的选项。

Key	Description
`dest` or `storage`	在哪里存储文件
`fileFilter`	文件过滤器，控制哪些文件可以被接受
`limits`	限制上传的数据
`preservePath`	保存包含文件名的完整文件路径

通常，只需要设置 `dest` 属性 像这样：

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

如果你想在上传时进行更多的控制，你可以使用 `storage` 选项替代 `dest`。Multer 具有 `DiskStorage` 和 `MemoryStorage` 两个存储引擎；另外还可以从第三方获得更多可用的引擎。

`.single(fieldname)`

接受一个以 `fieldname` 命名的文件。这个文件的信息保存在 `req.file`。

`.array(fieldname[, maxCount])`

接受一个以 `fieldname` 命名的文件数组。可以配置 `maxCount` 来限制上传的最大数量。这些文件的信息保存在 `req.files`。

`.fields(fields)`

接受指定 `fields` 的混合文件。这些文件的信息保存在 `req.files`。

`fields` 应该是一个对象数组，应该具有 `name` 和可选的 `maxCount` 属性。

Example:

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

`.none()`

只接受文本域。如果任何文件上传到这个模式，将发生 "LIMIT_UNEXPECTED_FILE" 错误。这和 `upload.fields([])` 的效果一样。

`.any()`

接受一切。文件数组将保存在 `req.files`。

警告: 确保你总是处理了用户的文件上传。 永远不要将 multer 作为全局中间件使用，因为恶意用户可以上传文件到一个你没有预料到的路由，应该只在你需要处理上传文件的路由上使用。

`storage`

`DiskStorage`

磁盘存储引擎可以让你控制文件的存储。

```
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 })
```

有两个选项可用， `destination` 和 `filename`。他们都是用来确定文件存储位置的函数。

`destination` 是用来确定上传的文件应该存储在哪个文件夹中。也可以提供一个 `string` (例如 `'/tmp/uploads'`)。如果没有设置 `destination`，则使用操作默认的临时文件夹

注意: 如果你提供的 `destination` 是一个函数，你需要负责创建文件夹。当提供一个字符串，multer 将确保这个文件夹是你创建的。

`filename` 用于确定文件夹中的文件名的确定。 如果没有设置 `filename`，每个文件将设置为一个随机文件名，并且是没有扩展名的

注意: Multer 不会为你添加任何扩展名，你的程序应该返回一个完整的文件名。