



# Express+MongoDB搭建图片分享社区二

1024肥宅 2020-10-14 👁 657 ⌚ 阅读10分钟

关注

## 前言

在上一篇文章[Express+MongoDB搭建图片分享社区一](#)中我们实现了路由、页面和数据渲染。

本篇文章我们要实现图片上传、接入mongoDB数据库、评论、点赞、删除等功能。

## 图片上传功能

在根目录创建public/upload文件夹，用于存放用户上传的图片，然后安装 multer 中间件用于处理文件上传：

CSS

复制代码

```
1 npm i multer
```

在 server/routes.js 模块中，我们初始化 multer 中间件，然后将其添加到上传图片的路由中（即 POST /images），代码如下所示：

ini

复制代码

```
1 const express = require('express');
2 const multer = require('multer');
3 const path = require('path');
4
5 const router = express.Router();
6 const upload = multer({ dest: path.join(__dirname, '../public/upload/temp') });
7 const home = require('../controllers/home');
8 const image = require('../controllers/image');
9
10 module.exports = function(app) {
11   router.get('/', home.index);
12   router.get('/images/:image_id', image.index);
13   router.post('/images', upload.single('file'), image.create);
14   router.post('/images/:image_id/like', image.like);
15   router.post('/images/:image_id/comment', image.comment);
16 }
```

上述代码有两点需要讲解：

- 第 6 行，在初始化 upload 中间件时，传入 dest 选项指定保存上传文件的路径，这里我们选择在 public/upload 目录中再创建一个 temp 目录用于临时保存上传到的图片。
- 第 14 行，router.post 除第一个参数为 URL，后面可以跟任意多个中间件，这里我们将上传文件的中间件添加到 image.create 控制器的前面，确保先处理用户上传的文件。这里 upload.single('file') 表示只处理单个上传文件，并且字段名为 file，在后续中间件中就可以通过 req.file 进行获取。

## 注意

在dest选项这里图灵社区的路径是public/upload/temp，但是我在操作过程中发现这个路径不是项目根目录下，而是在当前server文件夹下，所以这里的路径要用../public/upload/temp

关于 multer 的详细用法，可以参考其[文档](#)

controller/image.js

▼ javascript

复制代码

```
1  const fs = require('fs');
2  const path = require('path');
3
4  module.exports = {
5    index: function(req, res) {
6      const viewModel = {
7        image: {
8          uniqueId: 1,
9          title: '示例图片1',
10         description: '这是张测试图片',
11         filename: 'sample1.jpg',
12         views: 0,
13         likes: 0,
14         timestamp: Date.now(),
15       },
16       comments: [
17         {
18           image_id: 1,
19           email: 'test@testing.com',
20           name: 'Test Tester',
21           comment: 'Test 1',
22           timestamp: Date.now(),
23         },
24         {
25           image_id: 1,
26           email: 'test@testing.com',
```

```
30     },
31   ],
32   };
33   res.render('image', viewModel);
34 },
35 create: function(req, res) {
36   var tempPath = req.file.path;
37   var imgUrl = req.file.filename;
38   var ext = path.extname(req.file.originalname).toLowerCase();
39   var targetPath = path.resolve('./public/upload/' + imgUrl + ext);
40
41   if (ext === '.png' || ext === '.jpg' || ext === '.jpeg' || ext === '.gif') {
42     fs.rename(tempPath, targetPath, function(err) {
43       if (err) throw err;
44       res.redirect('/images/' + imgUrl);
45     });
46   } else {
47     fs.unlink(tempPath, function(err) {
48       if (err) throw err;
49       res.json(500, { error: '只允许上传图片文件.' });
50     });
51   }
52 },
53 like: function(req, res) {
54   res.send('The image:like POST controller');
55 },
56 comment: function(req, res) {
57   res.send('The image:comment POST controller');
58 },
59 };
```

req.file 是一个 Multer 文件对象，包括 path（上传到服务器的路径）、filename（服务器存储的文件名）和 originalname（文件初始名，即保存在客户端的文件名）等有用的属性。我截取了一张输出 req.file 所有字段的图片如下：

```
{
  fieldname: 'file',
  originalname: 'tutture-logo-large.png',
  encoding: '7bit',
  mimetype: 'image/png',
  destination: '/Users/mRc/Tutorials/fake-instagram/server/public/upload/temp',
  filename: 'e4a8667aa0143972db07cd0fe9220007',
  path: '/Users/mRc/Tutorials/fake-instagram/server/public/upload/temp/e4a8667aa0143972db07cd0fe9220007',
  size: 216516
}
```

@稀土掘金技术社区

这里我们通过简单的后缀匹配来判断用户上传的是否为图片，如果是，则从临时目录 tempPath 存放到上传目录 targetPath 中，否则直接删除。上传成功后，通过 res.redirect 将页面重定向到刚刚上传的图片的详情页面。

首先安装MongoDB:

[MongoDB官网](#) [菜鸟教程MongoDB安装](#)

安装之后新开一个终端根据不同平台的对应方法打开数据库。

然后安装Mongoose:

CSS

复制代码

```
1 npm i mongoose
```

Mongoose 是 MongoDB 最流行的 ODM (Object Document Mapping, 对象文档映射), 使用起来要比底层的 MongoDB Node 驱动更方便。

我们首先实现图片有关的数据模型。创建 models 目录, 在其中添加 image.js 模块, 并添加实现 ImageSchema 的代码: model/image.js

javascript

复制代码

```
1 const mongoose = require('mongoose');
2 const path = require('path');
3
4 const Schema = mongoose.Schema;
5
6 const ImageSchema = new Schema({
7   title: { type: String },
8   description: { type: String },
9   filename: { type: String },
10  views: { type: Number, default: 0 },
11  likes: { type: Number, default: 0 },
12  timestamp: { type: Date, default: Date.now },
13 });
14
15 ImageSchema.virtual('uniqueId').get(function() {
16   return this.filename.replace(path.extname(this.filename), '');
17 });
18
19 module.exports = mongoose.model('Image', ImageSchema);
```

我们在第 6 行到第 13 行定义了一个 Schema, 即数据对象的模式, 描述了这个模型的所有字段及相应的属性。这里我们为 ImageSchema 定义了六个字段, 每个字段都有其类型(必须), views、likes 和 timestamp 还有相应的默认值(可选)。除了普通字段外, 我们还定义了虚字段uniqueId。虚字段(virtuals)和普通字段的最大区别是



接着我们在 home 控制器中调用 ImageModel 来从数据库中获取全部图片： controller/home.js

[复制代码](#)

```
ini

1  const ImageModel = require('../models/image');
2
3  module.exports = {
4    index: function(req, res) {
5      const viewModel = { images: [] };
6
7      ImageModel.find({}, {}, { sort: { timestamp: -1 } }, function(err, images) {
8        if (err) throw err;
9        viewModel.images = images;
10       res.render('index', viewModel);
11     });
12   },
13 };
```

在第 39 行中，我们用 find 方法查询图片，所有的查询方法可参考 [Mongoose 中文文档](#)。find 是查询多条数据记录的通用方法，其四个参数如下：

- filter：过滤器，是一个 JavaScript 对象，例如 { name: 'john' } 则限定返回所有名字为 john 的记录，这里我们用 {} 表示查询所有记录；
- projection（可选）：查询所返回的字段，可以是对象或字符串，我们用 {} 表示返回所有字段；
- options（可选）：查询操作的选项，用来指定查询操作的一些参数，比如我们用 sort 选项对返回结果进行排序（这里按照发布时间 timestamp 进行倒序排列，即把最新发布的放在最前面）；
- callback：回调函数，用于添加在查询完毕时的业务逻辑；

进一步，我们在 image 控制器中添加数据库操作的代码： controller/image.js

[复制代码](#)

```
ini

1  const fs = require('fs');
2  const path = require('path');
3  const ImageModel = require('../models/image');
4
5  module.exports = {
6    index: function(req, res) {
7      const viewModel = { image: {}, comments: [] };
8
9      ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(
10        err,
11        image,
12      ) {
```

```
16     image.views += 1;
17     viewModel.image = image;
18     image.save();
19     res.render('image', viewModel);
20   } else {
21     res.redirect('/');
22   }
23 });
24 },
25 create: function(req, res) {
26   var tempPath = req.file.path;
27   var imgUrl = req.file.filename;
28   var ext = path.extname(req.file.originalname).toLowerCase();
29   var targetPath = path.resolve('./public/upload/' + imgUrl + ext);
30
31   if (ext === '.png' || ext === '.jpg' || ext === '.jpeg' || ext === '.gif') {
32     fs.rename(tempPath, targetPath, function(err) {
33       if (err) throw err;
34       const newImg = new ImageModel({
35         title: req.body.title,
36         description: req.body.description,
37         filename: imgUrl + ext,
38       });
39       newImg.save(function(err, image) {
40         if (err) throw err;
41         res.redirect('/images/' + image.uniqueId);
42       });
43     });
44   } else {
45     fs.unlink(tempPath, function(err) {
46       if (err) throw err;
47       res.json(500, { error: '只允许上传图片文件.' });
48     });
49   }
50 },
51 like: function(req, res) {
52   res.send('The image:like POST controller');
53 },
54 comment: function(req, res) {
55   res.send('The image:comment POST controller');
56 },
57 };
```

在 `image.index` 和 `image.create` 两个控制器中，我们分别进行了单条数据记录的查询和插入。`findOne` 与之前的 `find` 参数格式完全一致，只不过仅返回一条数据。在插入新数据时，先创建一个 `ImageModel` 实例，然后再调用 `save` 方法进行保存即可。

ini

复制代码

```
1
2  const express = require('express');
3  const mongoose = require('mongoose');
4  const configure = require('./server/configure');
5
6  app = express();
7  app = configure(app);
8
9  // 建立数据库连接
10 mongoose.connect('mongodb://localhost/instagrammy');
11 mongoose.connection.on('open', function() {
12   console.log('Mongoose connected.');
```

## 运行过程中的问题

这里我们运行服务器的话会有如下的输出信息:

```
> node server.js
(node:2171) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass
option { useNewUrlParser: true } to MongoClient.connect.
(node:2171) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use
the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
Server is running on http://localhost:3000
Mongoose connected.
```

@稀土掘金技术社区

解决方法: mongoose连接数据库时除了url参数外增加1个参数: {useNewUrlParser:true,useUnifiedTopology: true}  
server.js

javascript

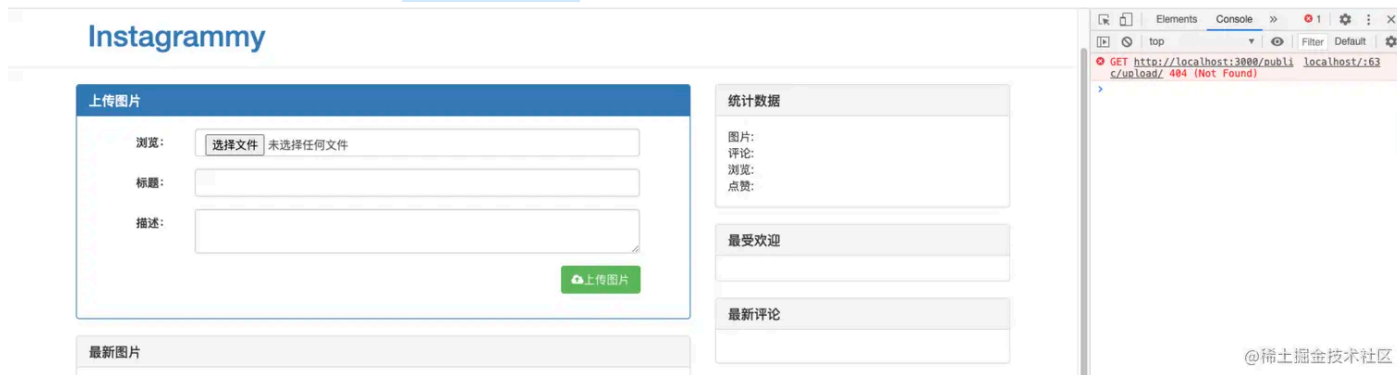
复制代码

```
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const configure = require('./server/configure');
4
5  app = express();
6  app = configure(app);
7
8  // 建立数据库连接
9  mongoose.connect('mongodb://localhost/instagrammy', {useNewUrlParser:true,useUnifiedTopology:true });
10 mongoose.connection.on('open', function() {
11   console.log('Mongoose connected.');
```

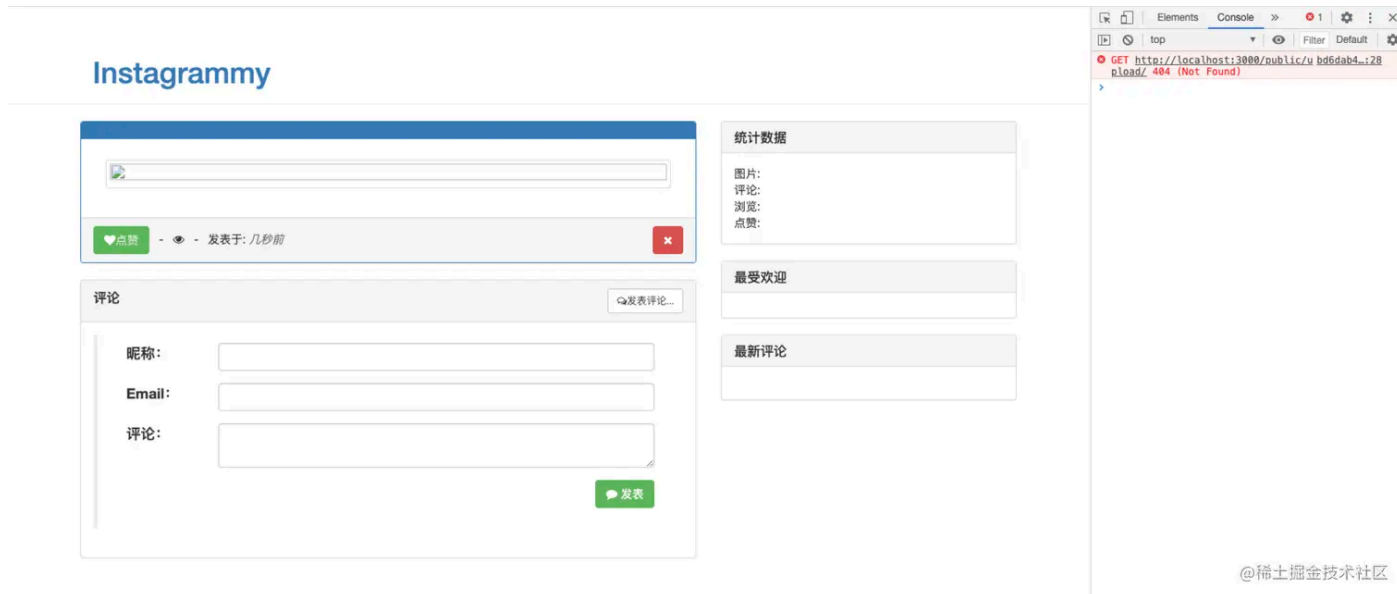


```
15
16 app.listen(app.get('port'), function() {
17   console.log(`Server is running on http://localhost:${app.get('port')}`)
18 })
```

然后我们重新运行服务器，打开localhost:3000，我们会发现console有报错信息：



我们走一下上传图片的过程，上传跳转到图片详情会发现页面是这个样子：



在服务器的日志上发现有如下所示信息：

```
Handlebars: Access has been denied to resolve the property "likes" because it is not an "own property" of its parent.
You can add a runtime option to disable the check or this warning:
See https://handlebarsjs.com/api-reference/runtime-options.html#options-to-control-prototype-access for details
Handlebars: Access has been denied to resolve the property "views" because it is not an "own property" of its parent.
You can add a runtime option to disable the check or this warning:
See https://handlebarsjs.com/api-reference/runtime-options.html#options-to-control-prototype-access for details
Handlebars: Access has been denied to resolve the property "timestamp" because it is not an "own property" of its parent.
You can add a runtime option to disable the check or this warning:
See https://handlebarsjs.com/api-reference/runtime-options.html#options-to-control-prototype-access for details
```

在[stackoverflow](https://stackoverflow.com/questions/40404040)有这个问题的 相关讨论，有兴趣的可以去看一下

解决方法：



```
1 npm i @handlebars/allow-prototype-access
```

然后修改server/configure.js的内容如下所示:

```
1  const path = require('path');
2  const Handlebars = require('handlebars');
3  const expbs = require('express-handlebars');
4  const express = require('express');
5  const bodyParser = require('body-parser');
6  const cookieParser = require('cookie-parser');
7  const morgan = require('morgan');
8  const methodOverride = require('method-override');
9  const errorHandler = require('errorhandler');
10 const moment = require('moment');
11
12 const { allowInsecurePrototypeAccess } = require('@handlebars/allow-prototype-access');
13
14 const routes = require('./routes');
15
16 module.exports = function(app) {
17     // 定义 moment 全局语言
18     moment.locale('zh-cn');
19     app.engine('handlebars', expbs.create({
20         helpers: {
21             timeago: function(timestamp) {
22                 return moment(timestamp).startOf('minute').fromNow();
23             }
24         },
25         handlebars: allowInsecurePrototypeAccess(Handlebars)
26     }).engine)
27     app.set('view engine', 'handlebars');
28
29     app.use(morgan('dev'));
30     app.use(bodyParser.urlencoded({ extended: true }));
31     app.use(bodyParser.json());
32     app.use(methodOverride());
33     app.use(cookieParser('secret-value'));
34     app.use('/public/', express.static(path.join(__dirname, '../public')));
35
36     if (app.get('env') === 'development') {
37         app.use(errorHandler());
38     }
39
40     routes(app);
```

现在我们来重新运行服务器，尝试上传图片，可以发现不仅能上传成功，还可以在首页看到新添加的图片了！

## 实现评论功能

这一步我们来实现网站的评论功能。按照 MVC 模式，我们将依次实现评论的模型（M）、视图（V）和控制器（C）。首先，仿照 `models/image.js`，我们实现评论的数据模型：`models/comment.js`

▼ javascript

复制代码

```
1  const mongoose = require('mongoose');
2
3  const Schema = mongoose.Schema;
4  const ObjectId = Schema.ObjectId;
5
6  const CommentSchema = new Schema({
7    image_id: { type: ObjectId },
8    email: { type: String },
9    name: { type: String },
10   gravatar: { type: String },
11   comment: { type: String },
12   timestamp: { type: Date, default: Date.now },
13 });
14
15 CommentSchema.virtual('image')
16   .set(function(image) {
17     this._image = image;
18   })
19   .get(function() {
20     return this._image;
21   });
22
23 module.exports = mongoose.model('Comment', CommentSchema);
```

CommentSchema 有两个字段需要补充说明一下：

- `image_id`：由于图片和评论是一对多的关系（即一张图片包括多个评论），因此我们需要在记录每个评论所属的图片，即通过 `image_id` 字段进行记录；
- `gravatar`：用 MD5 对电子邮箱加密后得到的字符串，用于访问 [Gravatar 服务](#)。Gravatar 提供了跨网站的头像服务，如果你在集成了 Gravatar 服务的网站通过邮箱注册并上传了头像，那么别的网站也可以通过 Gravatar 访问你的头像。这里请通过 `npm install md5` 安装 MD5 加密的包。

我们对评论有关的界面代码进行细微的调整，将提交按钮的 `type` 从 `button` 改为 `submit`：

`views/image.handlebars`

```

49     <div class="form-group col-sm-12">
50         <label for="name" class="col-sm-2 control-label">昵称:</label>
51         <div class="col-sm-10"><input type="text" class="form-control" name="name"></div>
52     </div>
53     <div class="form-group col-sm-12">
54         <label for="email" class="col-sm-2 control-label">Email:</label>
55         <div class="col-sm-10"><input type="text" class="form-control" name="email"></div>
56     </div>
57     <div class="form-group col-sm-12">
58         <label for="comment" class="col-sm-2 control-label">评论:</label>
59         <div class="col-sm-10">
60             <textarea name="comment" class="form-control" rows="2"></textarea>
61         </div>
62     </div>
63     <div class="form-group col-sm-12">
64         <div class="col-sm-12 text-right">
65             <button class="btn btn-success" id="comment-btn" type="submit">
66                 <i class="fa fa-comment"></i> 发表
67             </button>
68         </div>
69     </div>
70 </form>
71 </div>
72 </blockquote>

```

@稀土掘金技术社区

最后是评论有关的 controller 代码。包括在 image.comment 中实现创建评论，以及在 image.index 中实现对单张图片所有评论的查询： controllers/image.js

ini

复制代码

```

1  const fs = require('fs');
2  const path = require('path');
3  const md5 = require('md5');
4  const ImageModel = require('../models/image');
5  const CommentModel = require('../models/comment');
6
7  module.exports = {
8      index: function(req, res) {
9          const viewModel = { image: {}, comments: [] };
10
11          ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(
12              err,
13              image,
14          ) {
15              if (err) throw err;
16              if (image) {
17                  // 增加该图片的访问量
18                  image.views += 1;
19                  viewModel.image = image;
20                  image.save();
21

```

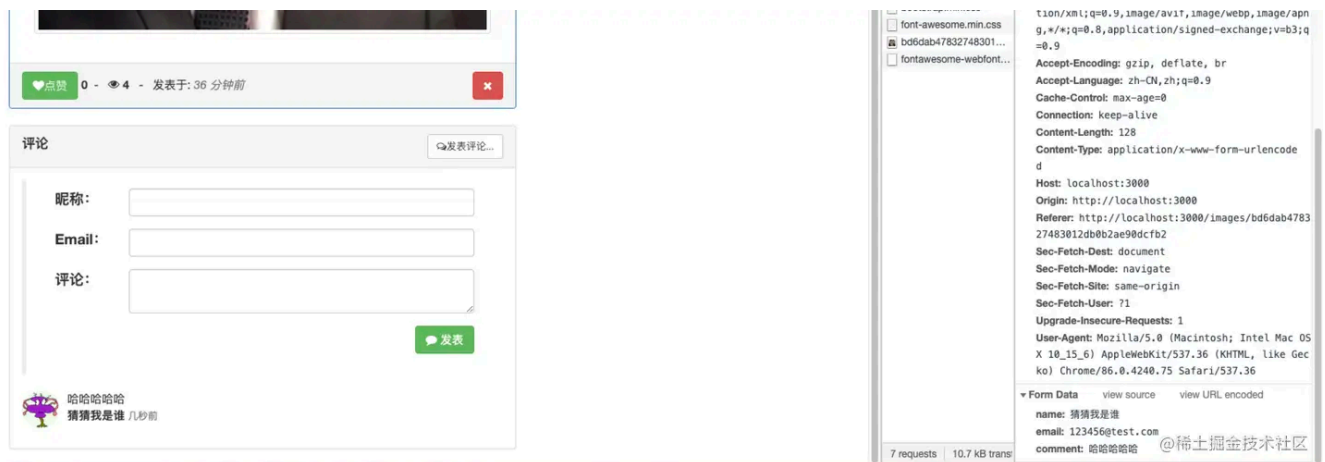


```
25         { sort: { timestamp: 1 } } },
26         function(err, comments) {
27             if (err) throw err;
28             viewModel.comments = comments;
29             res.render('image', viewModel);
30         },
31     );
32 } else {
33     res.redirect('/');
34 }
35 });
36 },
37 create: function(req, res) {
38     var tempPath = req.file.path;
39     var imgUrl = req.file.filename;
40     var ext = path.extname(req.file.originalname).toLowerCase();
41     var targetPath = path.resolve('./public/upload/' + imgUrl + ext);
42
43     if (ext === '.png' || ext === '.jpg' || ext === '.jpeg' || ext === '.gif') {
44         fs.rename(tempPath, targetPath, function(err) {
45             if (err) throw err;
46             const newImg = new ImageModel({
47                 title: req.body.title,
48                 description: req.body.description,
49                 filename: imgUrl + ext,
50             });
51             newImg.save(function(err, image) {
52                 if (err) throw err;
53                 res.redirect('/images/' + image.uniqueId);
54             });
55         });
56     } else {
57         fs.unlink(tempPath, function(err) {
58             if (err) throw err;
59             res.json(500, { error: '只允许上传图片文件.' });
60         });
61     }
62 },
63 like: function(req, res) {
64     res.send('The image:like POST controller');
65 },
66 comment: function(req, res) {
67     ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(
68         err,
69         image,
70     ) {
71         if (!err && image) {
72             const newComment = new CommentModel(req.body);
73             newComment.gravatar = md5(newComment.email);
74             newComment.image_id = image._id;
```

```
78     });  
79   } else {  
80     res.redirect('/');  
81   }  
82   });  
83 },  
84 };
```

查询与创建评论的代码和之前操作图片的代码大部分都是一致的，最大的差别在于查询时需要根据所属的图片ID，创建时需要记录图片的ID。这里我们约定使用 MongoDB 为每一条数据默认创建的 `_id` 字段。

接下来我们重启服务器，然后找一张图片点进详情测试一下评论功能



## 实现图片的点赞和删除

这一步我们来实现图片的点赞和删除。首先在控制器中添加点赞和删除的代码: `controller/image.js`

ini

复制代码

```
1  const fs = require('fs');  
2  const path = require('path');  
3  const md5 = require('md5');  
4  const ImageModel = require('../models/image');  
5  const CommentModel = require('../models/comment');  
6  
7  module.exports = {  
8    index: function(req, res) {  
9      const viewModel = { image: {}, comments: [] };  
10  
11      ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(  
12        err,  
13        image,  
14      ) {
```

```
18     image.views += 1;
19     viewModel.image = image;
20     image.save();
21
22     CommentModel.find(
23       { image_id: image._id },
24       {},
25       { sort: { timestamp: 1 } },
26       function(err, comments) {
27         if (err) throw err;
28         viewModel.comments = comments;
29         res.render('image', viewModel);
30       },
31     );
32   } else {
33     res.redirect('/');
34   }
35 });
36 },
37 create: function(req, res) {
38   var tempPath = req.file.path;
39   var imgUrl = req.file.filename;
40   var ext = path.extname(req.file.originalname).toLowerCase();
41   var targetPath = path.resolve('./public/upload/' + imgUrl + ext);
42
43   if (ext === '.png' || ext === '.jpg' || ext === '.jpeg' || ext === '.gif') {
44     fs.rename(tempPath, targetPath, function(err) {
45       if (err) throw err;
46       const newImg = new ImageModel({
47         title: req.body.title,
48         description: req.body.description,
49         filename: imgUrl + ext,
50       });
51       newImg.save(function(err, image) {
52         if (err) throw err;
53         res.redirect('/images/' + image.uniqueId);
54       });
55     });
56   } else {
57     fs.unlink(tempPath, function(err) {
58       if (err) throw err;
59       res.json(500, { error: '只允许上传图片文件.' });
60     });
61   }
62 },
63 like: function(req, res) {
64   ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(
65     err,
66     image,
67   ) {
```

```
71         if (err) res.json(err);
72         else res.json({ likes: image.likes });
73     });
74 }
75 });
76 },
77 remove: function(req, res) {
78     ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(
79         err,
80         image,
81     ) {
82         if (err) throw err;
83         fs.unlink(path.resolve('./public/upload/' + image.filename), function(
84             err,
85         ) {
86             if (err) throw err;
87             CommentModel.remove({ image_id: image._id }, function(err) {
88                 image.remove(function(err) {
89                     if (!err) {
90                         res.json(true);
91                     } else {
92                         res.json(false);
93                     }
94                 });
95             });
96         });
97     });
98 },
99 comment: function(req, res) {
100     ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(
101         err,
102         image,
103     ) {
104         if (!err && image) {
105             const newComment = new CommentModel(req.body);
106             newComment.gravatar = md5(newComment.email);
107             newComment.image_id = image._id;
108             newComment.save(function(err, comment) {
109                 if (err) throw err;
110                 res.redirect('/images/' + image.uniqueId + '#' + comment._id);
111             });
112         } else {
113             res.redirect('/');
114         }
115     });
116 },
117 };
```

轻松实现。remove 的使用方法与之前的 find 几乎一模一样，只不过 find 会返回符合条件的结果，而 remove 则会直接将符合条件的记录从数据库中删除。

我们在路由模块 server/routes.js 中添加刚刚写好的 image.remove 控制器：

ini

复制代码

```
1  const express = require('express');
2  const multer = require('multer');
3  const path = require('path');
4
5  const router = express.Router();
6  const upload = multer({ dest: path.join(__dirname, 'public/upload/temp') });
7  const home = require('../controllers/home');
8  const image = require('../controllers/image');
9
10 module.exports = function(app) {
11   router.get('/', home.index);
12   router.get('/images/:image_id', image.index);
13   router.post('/images', upload.single('file'), image.create);
14   router.post('/images/:image_id/like', image.like);
15   router.post('/images/:image_id/comment', image.comment);
16   router.delete('/images/:image_id', image.remove);
17   app.use(router);
18 };
```

我们来重新运行服务器，然后试一下点赞和删除发现并没有任何作用，因为上面的只是服务端的逻辑，前端页面显示并没有对应的逻辑，所以我们用jQuery来实现前端的点赞和删除请求。 views/layouts/main.handlebars

xml

复制代码

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>Instagrammy</title>
7    <link href="http://netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" rel="stylesheet">
8    <link href="http://netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css" rel="stylesheet">
9  </head>
10
11 <body>
12   <div class="page-header">
13     <div class="container">
14       <div class="col-md-6">
15         <h1><a href="/">Instagrammy</a></h1>
16       </div>
```



```
20 <div class="container">
21   <div class="row">
22     <div class="col-sm-8">{{{body}}}</div>
23     <div class="col-sm-4">
24       {{> stats this}}
25       {{> popular this}}
26       {{> comments this}}
27     </div>
28   </div>
29 </div>
30 </body>
31
32 <script src="https://cdn.bootcss.com/jquery/3.4.1/jquery.min.js"></script>
33 <script>
34   $(function () {
35     // to do...
36   });
37   $('#btn-like').on('click', function (event) {
38     event.preventDefault();
39     var imgId = $(this).data('id');
40     $.post('/images/' + imgId + '/like').done(function (data) {
41       $('.likes-count').text(data.likes);
42     });
43   });
44   $('#btn-delete').on('click', function (event) {
45     event.preventDefault();
46     var $this = $(this);
47     var remove = confirm('确定要删除这张图片吗?');
48     if (remove) {
49       var imgId = $(this).data('id');
50       $.
51         .ajax({
52           url: '/images/' + imgId,
53           type: 'DELETE'
54         })
55         .done(function (result) {
56           if (result) {
57             $this.removeClass('btn-danger').addClass('btn-success');
58             $this.find('i').removeClass('fa-times').addClass('fa-check');
59             $this.append('<span> 已删除!</span>');
60             alert('删除成功');
61             window.location.href=document.referrerr;
62           }
63         });
64     }
65   });
66 </script>
67 </html>
```

## 完善用户界面

这一步我们将实现侧边栏所有数据的同步更新。首先先创建helpers目录用于存放侧边栏数据获取的相关代码。然后分析一下数据同步逻辑（例如统计数据），我们发现要进行的查询非常多：图片总数、评论总数、图片所有的访问量、图片所有的点赞数。如果按照普通的写法，我们也许会这样写：

▼ javascript

复制代码

```
1 queryA(function(err, resultsA) {
2   queryB(function(err, resultsB) {
3     queryC(function(err, resultsC) {
4       queryD(function(err, resultsD) {
5         // some code ...
6       }
7     }
8   }
9 })
```

这样的代码不仅十分丑陋，难以维护（即大家常说的“回调地狱”），而且性能也十分糟糕——所有查询都是链式执行。但其实所有的查询都是相互独立的，完全可以并发进行，那我们应该怎么写呢？

答案就是 [async](#) 库。async 是在 ECMAScript 6 的 Promise 体系出现之前最流行的异步组件库，凭借其强大的性能、丰富且设计良好的接口成为 Node 和前端开发中解决异步的最佳选择之一。这里我们也用 async 来解决并发获取数据的问题。安装 async 包：

▼ csharp

复制代码

```
1 npm i async
```

然后创建helpers/stats.js用户获取网站统计数据：

▼ php

复制代码

```
1 const async = require('async');
2 const ImageModel = require('../models/image');
3 const CommentModel = require('../models/comment');
4
5 module.exports = function(callback) {
6   async.parallel(
7     [
8       function(next) {
9         // 统计图片总数
```



```
13      // 统计评论总数
14      CommentModel.countDocuments({}, next);
15  },
16  function(next) {
17      // 对图片所有访问量求和
18      ImageModel.aggregate(
19          [
20              {
21                  $group: {
22                      _id: '1',
23                      viewsTotal: { $sum: '$views' },
24                  },
25              },
26          ],
27          function(err, result) {
28              if (err) {
29                  return next(err);
30              }
31              var viewsTotal = 0;
32              if (result.length > 0) {
33                  viewsTotal += result[0].viewsTotal;
34              }
35              next(null, viewsTotal);
36          },
37      );
38  },
39  function(next) {
40      // 对所有点赞数求和
41      ImageModel.aggregate(
42          [
43              {
44                  $group: {
45                      _id: '1',
46                      likesTotal: { $sum: '$likes' },
47                  },
48              },
49          ],
50          function(err, result) {
51              if (err) {
52                  return next(err);
53              }
54              var likesTotal = 0;
55              if (result.length > 0) {
56                  likesTotal += result[0].likesTotal;
57              }
58              next(null, likesTotal);
59          },
60      );
61  },
62  ],
```



```
66     comments: results[1],
67     views: results[2],
68     likes: results[3],
69   });
70 },
71 );
72 };
```

这里我们用到了 `async.parallel` 接口，它接受两个参数：

- `tasks`：一个函数数组，每个函数对应一个异步任务（所有任务将并发执行），并且接受一个回调函数用于返回任务执行的结果；
- `callback`：整个任务组的回调函数，可以获取所有异步任务执行完成后的所有结果。

我们将四个数据查询任务包装成四个函数作为 `async.parallel` 的第一个参数，在最后的 `callback` 中返回所有查询结果。非常简洁、优雅。

接下来实现侧边栏中的最新图片模块，一个简单的数据库查询即可：`helpers/images.js`

▼ javascript

复制代码

```
1  const ImageModel = require('../models/image');
2
3  module.exports = {
4    popular: function(callback) {
5      ImageModel.find({}, {}, { limit: 9, sort: { likes: 01 } }, function(err, images) {
6        if (err) return callback(err);
7        callback(null, images);
8      })
9    }
10 }
```

然后是创建获取最新评论的代码。不过简单地查询评论模型是不够的，我们还需要获取到每个评论对应的图片，这时候用 `async.each` 函数对一个数组中所有对象进行异步操作最为合适不过。整个模块的代码如下：

▼ javascript

复制代码

```
1  const async = require('async');
2  const ImageModel = require('../models/image');
3  const CommentModel = require('../models/comment');
4
5  module.exports = {
6    newest: function(callback) {
7      CommentModel.find({}, {}, { limit: 5, sort: { timestamp: -1 } }, function(err, comments) {
```

```
11         if (err) throw err;
12         comment.image = image;
13         next(err);
14     })
15 }
16 async.each(comments, attachImage, function(err) {
17     if (err) throw err;
18     callback(err, comments);
19 })
20 })
21 }
22 }
```

async.each 函数接受的三个参数如下：

- collection：用于接收异步操作的集合，这里是评论集；
- iteratee：异步操作函数，这里是 attachImage 函数；
- callback：全部操作执行完成的回调函数；

然后将前面三个 helper 函数放到一起，创建一个 sidebar 模块，并发获取三个模块的数据。这里我们还是用 async.parallel 函数，因为三个模块本质上也是异步查询：helpers/sidebars.js

ini

复制代码

```
1  const async = require('async');
2  const Stats = require('./stats');
3  const Images = require('./images');
4  const Comments = require('./comments');
5
6  module.exports = function(viewModel, callback) {
7      async.parallel(
8          [
9              function(next) {
10                  Stats(next);
11              },
12              function(next) {
13                  Images.popular(next);
14              },
15              function(next) {
16                  Comments.newest(next);
17              }
18          ],
19          function(err, results) {
20              viewModel.sidebar = {
21                  stats: results[0],
22                  popular: results[1],
23                  comments: results[2]
```

```
27     )
28 }
```

最后将sidebar 模块用到 home 和 image 控制器中: controllers/home.js

ini

复制代码

```
1  const sidebar = require('../helpers/sidebar');
2  const ImageModel = require('../models/image');
3
4  module.exports = {
5    index: function(req, res) {
6      const viewModel = { images: [] };
7
8      ImageModel.find({}, {}, { sort: { timestamp: -1 } }, function(err, images) {
9        if (err) throw err;
10       viewModel.images = images;
11       sidebar(viewModel, function(viewModel) {
12         res.render('index', viewModel);
13       });
14     });
15   },
16   };
```

controllers/image.js

javascript

复制代码

```
1  const fs = require('fs');
2  const path = require('path');
3  const md5 = require('md5');
4  const sidebar = require('../helpers/sidebar');
5  const ImageModel = require('../models/image');
6  const CommentModel = require('../models/comment');
7
8  module.exports = {
9    index: function(req, res) {
10      const viewModel = { image: {}, comments: [] };
11      ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(err, image) {
12        if (err) throw err;
13        if (image) {
14          // 增加该图片的访问量
15          image.views += 1;
16          viewModel.image = image;
17          image.save();
18        }
19        CommentModel.find(
```

```
23     function(err, comments) {
24         if (err) throw err;
25         viewModel.comments = comments;
26         sidebar(viewModel, function(viewModel) {
27             res.render('image', viewModel);
28         })
29     }
30 }
31 } else {
32     res.redirect('/');
33 }
34 })
35 },
36 create: function(req, res) {
37     var tempPath = req.file.path;
38     var imgUrl = req.file.filename;
39     var ext = path.extname(req.file.originalname).toLowerCase();
40     var targetPath = path.resolve('./public/upload/' + imgUrl + ext);
41
42     if (ext === '.png' || ext === '.jpg' || ext === '.jpeg' || ext === '.gif') {
43         fs.rename(tempPath, targetPath, function(err) {
44             if (err) throw err;
45             const newImg = new ImageModel({
46                 title: req.body.title,
47                 description: req.body.description,
48                 filename: imgUrl + ext
49             })
50             newImg.save(function(err, image) {
51                 if (err) throw err;
52                 res.redirect('/images/' + image.uniqueId);
53             })
54         })
55     } else {
56         fs.unlink(tempPath, function(err) {
57             if (err) throw err;
58             res.json(500, { error: '只允许上传图片文件.' });
59         })
60     }
61 },
62 like: function(req, res) {
63     ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(err, image) {
64         if (!err && image) {
65             image.likes += 1;
66             image.save(function(err) {
67                 if (err) res.json(err);
68                 else res.json({ likes: image.likes })
69             })
70         }
71     })
72 },
```

```
76     fs.unlink(path.resolve('./public/upload/' + image.filename), function(err) {
77         if (err) throw err;
78         CommentModel.remove({ image_id: image._id }, function(err) {
79             image.remove(function(err) {
80                 if (!err) {
81                     res.json(true);
82                 } else {
83                     Response.json(false);
84                 }
85             })
86         })
87     })
88 })
89 },
90 comment: function(req, res) {
91     ImageModel.findOne({ filename: { $regex: req.params.image_id } }, function(err, image) {
92         if (!err && image) {
93             const newComment = new CommentModel(req.body);
94             newComment.gravatar = md5(newComment.email);
95             newComment.image_id = image._id;
96             newComment.save(function(err, comment) {
97                 if (err) throw err;
98                 res.redirect('/images/' + image.uniqueId + '#' + comment._id);
99             })
100         } else {
101             res.redirect('/')
102         }
103     })
104 }
105 }
```

现在我们来重新运行服务器看下侧边栏的数据有没有同步更新吧!

到这里我们今天要实现的功能已经全部实现了, 你可以在这个的基础上自己再拓展。

标签: Express

评论 0



抢首评, 友善交流



暂无评论数据

目录

收起 ^

- 前言
- 图片上传功能
- 接入MongoDB数据库
- 运行过程中的问题
- 实现评论功能
- 实现图片的点赞和删除
- 完善用户界面

搜索建议

搜索关键词 Q

- Express+MongoDB搭建图片分享社区一
- Reactjs + Nodejs + Express + Mongoddb搭建[图片上传/预览]项目(下)
- 后端实战手把手教你写文件上传接口：如何使用 Node.js + MongoDB 开发 RESTful API 接口（Node.js + Express + Mo
- vue+express+mongodb——个人全栈博客项目（二）
- Vue3+TypeScript+Vite+Express+MongoDB搭建博客管理系统
- mongodb+express+React 框架搭建 一
- node+express+mongodb （二）nodejs服务初始化
- docker-compose搭建mongodb



## 为你推荐

### 浅谈multer（出错勿喷）

朝依 2年前  864  3  3

前端

### NestJs文件上传由浅入深

求知若饥 5月前  833  6  1

前端 后端 NestJS

### 【0-1搭建网站】（三）安装、使用MongoDB

小小小七 24天前  239  点赞  评论




MongoDB koa 前端

### php7对mongodb的操作 以及存储图片技术gridfs

禾下月 2年前  571  1  评论

MongoDB 后端 图片资源

### Node（八）之MongoDB简单应用

阿宇的编程之旅 3年前  204  4  评论


MongoDB

### MongoDB 学习干货分享

已注销 9月前  3.1k  22  3

Mongo... 数据库 后端

### 博客功能篇：文章图片上传和生成缩略图处理

开发小程序的之朴 3年前  1.8k  5  评论

Go JavaScript

### 第二十天、文件上传下载实现

空尽欢 2年前  849  4  1

Node.js 前端

### 全面打通MongoDB+mongoose

勇宝趣学前端 4月前  518  4  评论

前端 数据库

### 看完即入门express+mongoose 搭建数据接口

\_荒 3年前  976  9  5

Node.js

### Linux服务器的MongoDB的安装和使用

守望时空33 2年前  1.0k  2  评论

MongoDB

### express图片上传

灵感火花 4月前  1.1k  2  评论

Express Node.js

### 如何使用 multer 中间件来处理上传到 Node.js 服务器的文件？

【架构师（第十四篇）】基于Node egg.js 和 mongodb 的使用

一尾流莺    2年前    👁 1.3k    👍 18    💬 21

前端    架构    前端框架

Nest成长足迹（五）：如何上传文件

晚风予星    5月前    👁 427    👍 5    💬 1

NestJS    前端    后端