# Express+MongoDB搭建图片分享社区一

1024肥宅 2020-10-13 ③ 303 ⑤ 阅读8分钟

关注

### 初始化项目结构

首先创建目录,并初始化(如果你想放在GitHub上的话,先在GitHub上建一个仓库然后clone下来): 初始化的时候可以一路enter下来,如果你不在意的话。

```
▼ bash

1 mkdir Instagrammy && cd Instagrammy
2
3 npm init

安装express
```

```
▼ css 

复制代码
```

1 npm i express

### 最终生成的package.json文件如下:

```
swift
1 {
     "name": "instagrammy",
     "version": "1.0.0",
3
    "description": "",
4
     "main": "index.js",
     "scripts": {
6
7
       "test": "echo \"Error: no test specified\" && exit 1"
8
     },
9
      "keywords": [],
10
     "author": "",
      "license": "ISC",
11
12
      "dependencies": {
    "express": "^4.17.1"
```







```
14 }
15 }
```

#### 然后在项目根目录新建入口文件server.js

```
javascript
   const express = require('express');
3
   app = express();
4
5
   app.set('port', process.env.PORT || 3000);
6
   app.get('/', function(req, res) {
7
       res.send('Hello World!');
8
   })
9
10
11 app.listen(app.get('port'), function() {
       console.log(`Server is running on http://localhost:${app.get('port')}`)
13 })
```

运行 node server.js , 然后在浏览器中访问http://localhost:3000 , 就可以在页面上看到服务器返回的Hello World!

这里为了接下去运行项目方便,我们在package.json中添加启动命令:

```
swift

1 "scripts": {
2    "start": "node server.js",
3    "test": "echo \"Error: no test specified\" && exit 1"
4 }
```

### 配置中间件

Express 本身是一个非常简洁的 web 框架,但是通过中间件这一设计模式,能够实现非常丰富的功能。一个 Express 中间件本质上是一个函数:

```
javascript 复制代码

function someMiddleware(req, res, next) {}

2
```







添加中间件的代码则非常简单:

```
1 app.use(middlewareA);
2 app.use(middlewareB);
3 app.use(middlewareC);
```

中间件 A、B、C 将会在处理每次请求时按顺序执行(这也意味着中间件的顺序是非常重要的)。接下来我们将添加以下基础中间件(也是几乎所有应用都会用到的中间件):

- morgan: 用于记录日志的中间件,对于开发调试和生产监控都很有用;
- bodyParser: 用于解析客户端请求的中间件,包括 HTML 表单和 JSON 请求;
- methodOverride: 为老的浏览器提供 REST 请求的兼容性支持;
- cookieParser: 用于收发 cookie;
- errorHandler: 用于在发生错误时打印调用栈, 仅在开发时使用;
- handlebars:用于渲染用户界面的模板引擎,会在后面细讲。

#### 我们通过 npm 安装这些中间件:

▼ arduino 复制代码

1 npm install express-handlebars body-parser cookie-parser morgan method-**override** errorhandler

在项目根目录创建server目录,在server目录下创建configure.js文件用于配置所有的中间件,文件内容如下:

php const path = require('path'); const exphbs = require('express-handlebars'); const express = require('express'); const bodyParser = require('body-parser'); const cookieParser = require('cookie-parser'); 5 const morgan = require('morgan'); const methodOverride = require('method-override'); 8 const errorHandler = require('errorhandler'); 9 10 module.exports = function(app) { 11 app.use(morgan('dev')); 12 app.use(bodyParser.urlencoded({ extended: true })); app.use(bodyParser.json()); 13 14 app.use(methodOverride()); app.use(cookieParser('secret-value')); 15 app.use('/public/', express.static(path.join(\_\_dirname, '../public'))); 16







```
20     }
21
22     return app;
23 }
```

express.static是express自带的静态资源中间件,用于向客户端发送图片、css等静态文件。最后,我们通过env变量来判断是否处于开发环境,如果是的话就添加errorHandler以便于调试代码。

在server.js中调用刚才配置中间件的代码:

```
javascript
   const express = require('express');
   const configure = require('./server/configure');
3
   app = express();
   app = configure(app);
6
   app.set('port', process.env.PORT || 3000);
7
8
   app.get('/', function(req, res) {
10
       res.send('Hello World!');
11 })
12
13
   app.listen(app.get('port'), function() {
        console.log(`Server is running on http://localhost:${app.get('port')}`)
15 })
```

### 搭建路由和控制器

在上面的步骤中我们配置好了基础的中间件,但是只有主页(/)可以访问,下面我们要实现以下路由:

- GET /: 网站主页。
- GET /images/image\_id: 展示单张图片。
- POST /images: 上传图片。
- POST /images/image\_id/like: 点赞图片。
- POST /images/image\_id/comment: 评论图片。

我们采用前端最熟悉的MVC模式来搭建下面的内容。

首先创建controllers目录,在该目录下新建home.js文件定义index控制器,代码如下:







```
3 res.send('The home:index controller');
4 },
5 };
```

每个控制器实际上都是一个 Express 中间件(只不过不需要 next 函数,因为是最后一个中间件)。这里我们暂时用 res.send 发一条文字来代表这个 controller 已经实现。

再在 controllers 目录下创建 image.js, 实现与图片处理相关的控制器:

```
lua
   module.exports = {
1
2
      index: function(req, res) {
3
       res.send('The image:index controller ' + req.params.image_id);
4
     },
5
     create: function(req, res) {
6
       res.send('The image:create POST controller');
7
     },
8
     like: function(req, res) {
       res.send('The image:like POST controller');
9
10
11
      comment: function(req, res) {
12
       res.send('The image:comment POST controller');
13
     },
14 };
```

然后在 server 目录下创建路由模块 routes.js,建立从 URL 到控制器之间的映射:

```
const express = require('express');
1
2
3
   const router = express.Router();
   const home = require('../controllers/home');
   const image = require('../controllers/image');
5
6
7
   module.exports = function(app) {
      router.get('/', home.index);
8
9
     router.get('/images/:image_id', image.index);
10
      router.post('/images', image.create);
      router.post('/images/:image_id/like', image.like);
11
      router.post('/images/:image_id/comment', image.comment);
12
13
      app.use(router);
14 };
```

这里我们用到了 Express 自带的路由类 Router,可以很方便地定义路由,并且 Router 本身也是一个中间件,可以







接着在 server/configure.js 模块中调用路由模块。

```
const path = require('path');
    const exphbs = require('express-handlebars');
2
    const express = require('express');
   const bodyParser = require('body-parser');
    const cookieParser = require('cookie-parser');
    const morgan = require('morgan');
6
    const methodOverride = require('method-override');
7
    const errorHandler = require('errorhandler');
8
9
    const routes = require('./routes');
10
11
12
    module.exports = function(app) {
      app.use(morgan('dev'));
13
      app.use(bodyParser.urlencoded({ extended: true }));
14
15
      app.use(bodyParser.json());
16
      app.use(methodOverride());
17
      app.use(cookieParser('secret-value'));
      app.use('/public/', express.static(path.join(__dirname, '../public')));
18
19
20
      if (app.get('env') === 'development') {
        app.use(errorHandler());
21
22
      }
23
24
      routes(app);
25
      return app;
26 };
```

最后我们去掉 server.is 中原来的首页路由。

```
const express = require('express');
   const configure = require('./server/configure');
2
```

```
3
4
   app = express();
5
   app = configure(app);
   app.set('port', process.env.PORT || 3000);
6
7
   app.listen(app.get('port'), function() {
8
9
      console.log(`Server is running on http://localhost:${app.get('port')}`);
10 });
```

现在我们运行服务器

ini









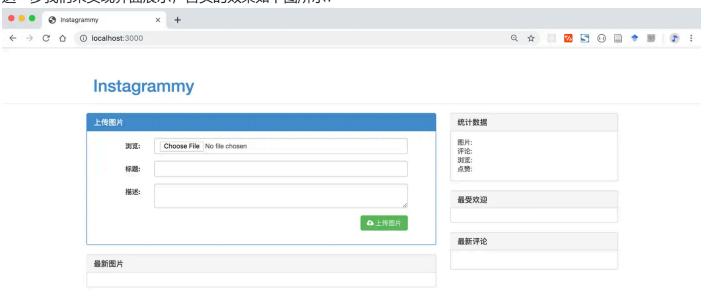
6/23

1 npm start

打开浏览器访问localhost:3000可以看到页面展示的The home:index controller。访问 localhost:3000/images/test123,则是The image:index controller test123。

### 配置handlebars模板引擎

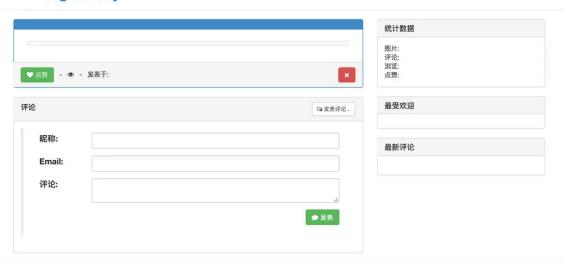
这一步我们来实现界面展示,首页的效果如下图所示:



### 图片详情页的效果如下图:



#### Instagrammy





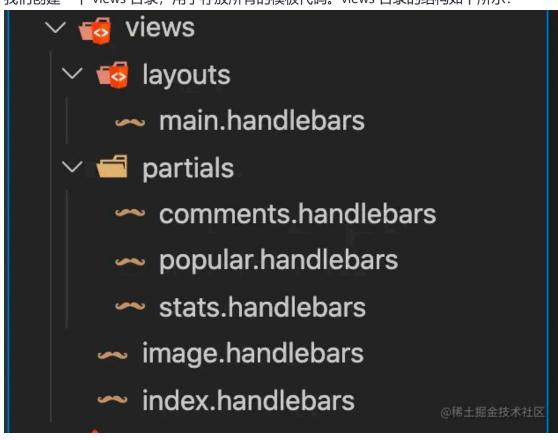


@稀土掘金技术社区

尽管如今前后端分离已经是大势所趋,但是通过模板引擎在服务器端渲染页面也是有用武之地的,特别是快速地开发一些简单的应用。在模板引擎中,Handlebars 和 Pug 当属其中的佼佼者。由于 Handlebars 和普通的 HTML 文档几乎完全一致,容易上手,因此这篇教程中我们选用 Handlebars,并且选用 Bootstrap 样式。

与普通的 HTML 文档相比,模板最大的特点即在于提供了数据的接入。例如 handlebars,可以在双花括号 {{}} 之间填写任何数据,当服务器渲染页面时只需传入相应的数据即可渲染成对应的内容。除此之外,handlebars 还 支持条件语法、循环语法和模板嵌套等高级功能,下面将详细描述。

我们创建一个 views 目录,用于存放所有的模板代码。views 目录的结构如下所示:



其中 image.handlebars 和 index.handlebars 是页面模板, layouts/main.handlebars 则是整个网站的布局模板(所有页面共享), partials 目录则用于存放页面之间共享的组件模板,例如评论、数据等等。

首先完成布局模板 layouts/main.handlebars:

```
10
        <div class="page-header">
11
            <div class="container">
12
                <div class="col-md-6">
                    <h1><a href="/">Instagrammy</a></h1>
13
14
                </div>
15
            </div>
        </div>
16
17
18
        <div class="container">
19
            <div class="row">
                <div class="col-sm-8">{{{body}}}</div>
20
21
            </div>
        </div>
23 </body>
24 </html>
```

main.handlebars 本身是一个完整的 HTML 文档,包括 head 和 body 部分。在 head 部分,我们定义了网站的一些元数据,还加入了 Bootstrap 的 CDN 链接;在 body 部分,包含两个容器:网站头部(header)和每个页面的自定义内容(即 {{{body}}}})。

接下来编写主页内容index.handlebars,页面内容先用标题占位:

```
1 <h1>Index Page</h1>
```

#### 图片详情页面内容:

```
▼ css 

复制代码
```

1 <h1>Image Page</h1>

index.handlebars 和 image.handlebars 的内容将替换布局模板中的 {{{body}}} 部分。在用户访问某个页面时,页面内容 = 布局模板 + 页面模板。

模板写好之后,我们修改控制器controller/home.js相应的代码,通过 res.render 函数渲染模板:

```
ini

module.exports = {
  index: function(req, res) {
    res.render('index');
  },
};
```







ini

同样地,我们修改 image 控制器controllers/image.js的代码:

```
module.exports = {
1
      index: function(req, res) {
2
3
        res.render('image');
4
      create: function(req, res) {
        res.send('The image:create POST controller');
6
7
     },
8
     like: function(req, res) {
9
        res.send('The image:like POST controller');
10
     },
      comment: function(req, res) {
11
12
        res.send('The image:comment POST controller');
13
      },
14 };
```

最后在server/configure.js中配置handlebars中间件:

```
1 const path = require('path');
2 const exphs = require('express-handlehars');
```

```
const exphbs = require('express-handlebars');
   const express = require('express');
3
   const bodyParser = require('body-parser');
   const cookieParser = require('cookie-parser');
6
   const morgan = require('morgan');
   const methodOverride = require('method-override');
7
   const errorHandler = require('errorhandler');
8
9
10
   const routes = require('./routes');
11
   module.exports = function(app) {
12
13
      app.engine('handlebars', exphbs());
      app.set('view engine', 'handlebars');
14
15
16
      app.use(morgan('dev'));
      app.use(bodyParser.urlencoded({ extended: true }));
17
18
      app.use(bodyParser.json());
19
      app.use(methodOverride());
      app.use(cookieParser('secret-value'));
20
      app.use('/public/', express.static(path.join(__dirname, '../public')));
21
22
      if (app.get('env') === 'development') {
23
        app.use(errorHandler());
24
25
      }
```









```
28    return app;
29  };
```

到这里我们这一步就完成了,现在 npm start 运行服务器,访问主页localhost:3000和图片详情页面 localhost:3000/images/test就可以看到对应的页面了,虽然没有数据.

### 完善页面代码

首先在index.handlebars中添加上传图片的表单和展示最新图片的容器

ini {{!-- views/index.handlebars --}} 2 3 <div class="panel panel-primary"> 4 <div class="panel-heading"> 5 <h3 class="panel-title"> 上传图片 6 7 </h3> 8 </div> 9 <form action="/images" method="post" enctype="multipart/form-data"> <div class="panel-body form-horizontal"> 10 11 <div class="form-group col-md-12"> 12 <label for="file" class="col-sm-2 control-label">浏览:</label> <div class="col-md-10"> 13 <input type="file" class="form-control" name="file" id="file"> 14 15 </div> </div> 16 17 <div class="form-group col-md-12"> <label for="title" class="col-md-2 control-label">标题:</label> 18 <div class="col-md-10"> 19 20 <input type="text" class="form-control" name="title"> </div> 21 22 </div> 23 <div class="form-group col-md-12"> 24 <label for="description" class="col-md-2 control-label">描述:</label> <div class="col-md-10"> 25 <textarea name="description" rows="2" class="form-control"></textarea> 26 27 </div> </div> 28 29 <div class="form-group col-md-12"> 30 <div class="col-md-12 text-right"> <button type="submit" id="login-btn" class="btn btn-success"> 31 <i class="fa fa-cloud-upload"> 上传图片</i> 32 33 </button> 34 </div> 35 </div>







```
38
   </div>
39
40
   <div class="panel panel-default">
41
      <div class="panel-heading">
42
        <h3 class="panel-title">
43
          最新图片
        </h3>
44
      </div>
45
46
      <div class="panel-body">
        {{#each images}}
47
          <div class="col-md-4 text-center" style="padding-bottom: 1em;">
48
49
            <a href="/images/{{uniqueId}}">
              <img src="/public/upload/{{filename}}" alt="{{title}}"</pre>
50
                style="width: 175px; height: 175px;" class="img-thumbnail">
51
52
            </a>
          </div>
53
        {{/each}}
54
      </div>
55
56
   </div>
```

在展示最新图片时,我们用到了 handlebars 提供的循环语法(第 46 行到 53 行)。对于传入模板的数据对象 images 进行遍历,每个循环中可以访问单个 image 的全部属性,例如 uniqueld 等等。

接着完善 image.handlebars 的内容,包括展示图片的详细内容、发表评论的表单和展示所有评论的容器。

```
{{!-- views/image.handlebars --}}
    <div class="panel panel-primary">
      <div class="panel-heading">
4
        <h2 class="panel-title">{{image.title}}</h2>
      </div>
5
      <div class="panel-body">
6
7
        {{image.description}}
        <div class="col-md-12 text-center">
8
          <img src="/public/upload/{{image.filename}}" alt="{{image.title}}"</pre>
9
10
            class="thumbnail">
11
        </div>
      </div>
12
13
      <div class="panel-footer">
14
        <div class="row">
15
          <div class="col-md-8">
            <button class="btn btn-success" id="btn-like" data-id="{{image.uniqueId}}">
16
              <i class="fa fa-heart"> 点赞</i>
17
18
            </button>
19
            <strong class="likes-count">{{image.likes}}</strong> &nbsp; - &nbsp;
20
            <i class="fa fa-eye"></i></i>
21
            <strong>{{image.views}}</strong>
```

xml





```
25
            <button class="btn btn-danger" id="btn-delete" data-id="{{image.uniqueId}}">
26
              <i class="fa fa-times"></i></i>
27
            </button>
28
          </div>
29
        </div>
30
      </div>
31 </div>
32
33
    <div class="panel panel-default">
      <div class="panel-heading">
34
        <div class="row">
35
36
          <div class="col-md-8">
            <strong class="panel-title">评论</strong>
37
          </div>
38
          <div class="col-md-4 text-right">
39
            <button class="btn btn-default btn-sm" id="btn-comment" data-id="{{image.uniqueId}}">
40
              <i class="fa fa-comments-o"> 发表评论...</i>
41
            </button>
42
43
          </div>
        </div>
44
45
      </div>
46
      <div class="panel-body">
        <blockquote id="post-comment">
47
48
          <div class="row">
49
            <form action="/images/{{image.uniqueId}}/comment" method="post">
              <div class="form-group col-sm-12">
50
                <label for="name" class="col-sm-2 control-label">昵称:</label>
51
52
                <div class="col-sm-10"><input type="text" class="form-control" name="name"></div>
53
              </div>
              <div class="form-group col-sm-12">
54
                <label for="email" class="col-sm-2 control-label">Email:</label>
55
56
                <div class="col-sm-10"><input type="text" class="form-control" name="email"></div>
57
58
              <div class="form-group col-sm-12">
                <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
66
                  <button class="btn btn-success" id="comment-btn" type="button">
67
                    <i class="fa fa-comment"></i> 发表
68
                  </button>
69
                </div>
70
              </div>
71
            </form>
72
          </div>
        </blockquote>
73
        74
```



https://juejin.cn/post/6882986837346877448





```
78
              <img src="http://www.gravatar.com/avatar/{{gravatar}}?d=monsterid&s=45"</pre>
79
                class="media-object img-circle">
80
            </a>
            <div class="media-body">
81
82
              {{comment}}
83
              <br/>
              <strong class="media-heading">{{name}}</strong>
84
              <small class="text-muted">{{timestamp}}</small>
85
86
            </div>
87
          {{/each}}
88
89
        </div>
91 </div>
```

在展示所有评论的代码中,我们同样用到了 handlebars 的循环语法,非常方便。

然后,我们将分别实现网站右边栏中的统计数据、最受欢迎图片和最新评论组件。

#### 首先是统计数据组件模板:

{{!-- views/partials/stats.handlebars --}} <div class="panel panel-default"> <div class="panel-heading"> 3 <h3 class="panel-title">统计数据</h3> 4 5 </div> <div class="panel-body"> 6 <div class="row"> 7 8 <div class="col-md-4 text-left">图片:</div> 9 <div class="col-md-8 text-right">{{sidebar.stats.images}}</div> </div> 10 11 <div class="row"> <div class="col-md-4 text-left">评论:</div> 12 13 <div class="col-md-8 text-right">{{sidebar.stats.comments}}</div> </div> 14 <div class="row"> 15 <div class="col-md-4 text-left">浏览:</div> 16 17 <div class="col-md-8 text-right">{{sidebar.stats.views}}</div> 18 </div> 19 <div class="row"> <div class="col-md-4 text-left">点赞:</div> 20 21 <div class="col-md-8 text-right">{{sidebar.stats.likes}}</div> 22 </div> 23 </div> 24 </div>







```
{{!-- views/partials/popular.handlebars --}}
1
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">最受欢迎</h3>
4
      </div>
5
6
      <div class="panel-body">
        {{#each sidebar.popular}}
8
          <div class="col-md-4 text-center" style="padding-bottom: .5em;">
9
            <a href="/images/{{uniqueId}}">
              <img src="/public/upload/{{filename}}" style="width: 75px; height: 75px;"</pre>
10
                class="img-thumbnail">
11
12
            </a>
13
          </div>
14
        {{/each}}
15
      </div>
16 </div>
```

#### 最新评论组件 (comments.handlebars):

```
{{!-- views/partials/comments.handlebars --}}
   <div class="panel panel-default">
     <div class="panel-heading">
3
       <h3 class="panel-title">最新评论</h3>
4
     </div>
5
6
     <div class="panel-body">
7
       8
         {{#each sidebar.comments}}
9
         10
           <a href="/images/{{image.uniqueId}}" class="pull-left">
             <img src="/public/upload/{{image.filename}}" class="media-object"</pre>
11
               height="45" width="45">
12
13
           </a>
14
           <div class="media-body">
             {{comment}}<br/>
15
             <strong class="media-heading">{{name}}</strong>
16
             <small class="text-muted">{{timestamp}}</small>
17
18
           </div>
19
         20
         {{/each}}
21
       </div>
22
23 </div>
```

最后,我们在布局模板 lavouts/main.handlebars 中加入所有组件模板,加入模板的语法为 {{> component







xml

₩ 4.1 / D.T.T.

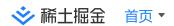
```
{{!-- views/layouts/main.handlebars --}}
2
   <!DOCTYPE html>
3
4
    <html lang="en">
   <head>
6
7
      <meta charset="UTF-8">
8
      <title>Instagrammy</title>
9
      <link href="http://netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" rel="stylesheet">
      <link href="http://netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css" rel="stylesheet">
10
  </head>
11
12
13
   <body>
14
      <div class="page-header">
15
        <div class="container">
          <div class="col-md-6">
16
17
            <h1><a href="/">Instagrammy</a></h1>
18
          </div>
        </div>
19
      </div>
20
21
22
      <div class="container">
23
        <div class="row">
          <div class="col-sm-8">{{{body}}}</div>
24
25
          <div class="col-sm-4">
            {{> stats this}}
26
            {{> popular this}}
27
28
            {{> comments this}}
29
          </div>
30
        </div>
31
      </div>
32
   </body>
33
34 </html>
```

### 将数据传入模板视图

如果没有数据传入,那么模板相应的数据部分将全都是空白。在这一步中,我们将用一些假数据来演示如何从控制器将数据传入模板视图。

首先在 home 控制器中构造一个 viewModel 对象,并在 render 函数中作为第二参数传入。可以看到 viewModel 对象与模板中的数据接口是完全一致的。

▼ yaml 复制代码







```
3
   module.exports = {
4
      index: function(req, res) {
5
        const viewModel = {
6
          images: [
7
            {
8
              uniqueId: 1,
              title: '示例图片1',
9
              description: '',
10
11
              filename: 'sample1.jpg',
              views: 0,
12
              likes: 0,
13
14
              timestamp: Date.now(),
15
            },
            {
16
              uniqueId: 2,
17
              title: '示例图片2',
18
19
              description: '',
20
              filename: 'sample2.jpg',
              views: 0,
21
              likes: 0,
22
23
              timestamp: Date.now(),
24
            },
25
            {
              uniqueId: 3,
26
27
              title: '示例图片3',
              description: '',
28
              filename: 'sample3.jpg',
29
              views: ∅,
30
31
              likes: 0,
              timestamp: Date.now(),
32
33
            },
34
          ],
35
        };
        res.render('index', viewModel);
36
      },
37
38 };
     javascript
1
   {{!-- controllers/image.js --}}
2
3
    module.exports = {
4
      index: function(req, res) {
5
        const viewModel = {
6
          image: {
7
            uniqueId: 1,
            title: '示例图片1',
8
9
            description: '这是张测试图片',
            filename: 'sample1.jpg',
10
            views. 0
```





```
14
15
          comments: [
            {
16
17
              image_id: 1,
18
              email: 'test@testing.com',
19
              name: 'Test Tester',
              comment: 'Test 1',
20
              timestamp: Date.now(),
21
22
            },
23
              image_id: 1,
24
25
              email: 'test@testing.com',
              name: 'Test Tester',
26
              comment: 'Test 2',
27
              timestamp: Date.now(),
28
29
            },
30
          ],
31
        };
32
        res.render('image', viewModel);
33
      },
34
      create: function(req, res) {
35
        res.send('The image:create POST controller');
36
      },
37
      like: function(req, res) {
38
        res.send('The image:like POST controller');
39
      },
40
      comment: function(req, res) {
41
        res.send('The image:comment POST controller');
42
      },
43 };
```

在传入数据时,我们可以自定义一些 helper 函数在模板中使用。例如 timestamp 时间戳,Date.now() 返回的是一串数字,显然用户体验很不友好,因此我们需要将其转换为方便用户阅读的时间,例如 "几秒前""两小时前"。这里我们选用 JavaScript 最流行的处理时间的库 moment.js,并通过 npm 安装:

```
▼ css 复制代码

1 npm i moment
```

然后在 server/configure.js 中配置 handlebars 的 helper 函数 timeago:

```
1 {{!-- server/configure.js --}}
2 const path = require('path');
4 const exphbs = require('express-handlebars');
```







```
const cookieParser = require('cookie-parser');
8
    const morgan = require('morgan');
    const methodOverride = require('method-override');
9
    const errorHandler = require('errorhandler');
    const moment = require('moment');
11
12
   const routes = require('./routes');
13
14
15
    module.exports = function(app) {
      // 定义 moment 全局语言
16
17
      moment.locale('zh-cn');
18
19
      app.engine(
20
        'handlebars',
        exphbs.create({
21
22
          helpers: {
23
            timeago: function(timestamp) {
              return moment(timestamp).startOf('minute').fromNow();
24
25
            },
26
          },
27
        }).engine,
28
      );
29
      app.set('view engine', 'handlebars');
30
31
      app.use(morgan('dev'));
      app.use(bodyParser.urlencoded({ extended: true }));
32
33
      app.use(bodyParser.json());
34
      app.use(methodOverride());
35
      app.use(cookieParser('secret-value'));
36
      app.use('/public/', express.static(path.join(__dirname, '../public')));
37
38
      if (app.get('env') === 'development') {
39
        app.use(errorHandler());
40
      }
41
42
      routes(app);
43
      return app;
44 };
```

接着在相应用到时间戳的地方加入 timeago 函数:





views/image.handlebars

```
<div class="panel-footer">
   <div class="row">
       <div class="col-md-8">
           <button class="btn btn-success" id="btn-like" data-id="{{image.uniqueId}}">
               <i class="fa fa-heart">点赞</i>
           </button>
           <strong class="likes-count">{{image.likes}}</strong>&nbsp; - &nbsp;
           <i class="fa fa-eye"></i>
           <strong>{{image.views}}</strong>
            -   发表于: <em class="text-muted">{{timeago image.timestamp}}</em>
       <div class="col-md-4 text-right">
           | button class="btn btn-danger" id="btn-delete" data-id="{{image.uniqueId}}
               <i class="fa fa-times"></i>
           </button>
       </div>
   </div>
```





views/partials/comments.handlebars

```
<div class="panel panel-default">
   <div class="panel-heading">
       <h3 class="panel-title">最新评论</h3>
   </div>
   <div class="panel-body">
       {{#each sidebar.comments}}
           class="media">
               <a href="/images/{{image.uniqueId}}" class="pull-left">
                   <img src="/public/upload/{{image.filename}}" class="media-object" height="45" widt</pre>
               <div class="media-body">
                   {{comment}}<br/>>
                  <strong class="media-heading">{{name}}</strong>
                <small class="text-muted">{{timeago timestamp}}</small>
           {{/each}}
   </div>
```

到这里我们实现了项目的视图和控制器完成了第一部分,下一部分我们来接入MongoDB数据库实现图片上传、点赞、删除、评论等功能。

原文地址图灵社区。

标签: Express

#### 评论 0



抢首评, 友善交流





0 / 1000 ②

发送









#### 暂无评论数据

初始化项目结构

配置中间件

搭建路由和控制器

配置handlebars模板引擎

完善页面代码

搜索建议

将数据传入模板视图

## Express+MongoDB搭建图片分享社区二

mongoDB+express+React 框架搭建一

Reactjs + Nodejs + Express + Mongodb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

Vue3+TypeScript+Vite+Express+MongoDB搭建博客管理系统

docker-compose搭建mongodb

使用Node + Express + mongodb 搭建API服务端

Reactjs + Nodejs + Express + Mongodb搭建[图片上传/预览]项目(上)

express+nodeJs 搭建后端服务一

搭建Express+MongoDB后台

#### 为你推荐

#### 前端上手Docker超详细基础教程

1024肥宅 3年前 ② 2.1k 🖒 72 💬 18

Typora+Vue主题+Gitee图床轻松写文章

※ 稀土掘金 首页 ▼

探索稀土掘金

٦

搜索关键词

Q



前端

[LeetCod	e0304题	二维区域	和检索	- 矩阵不可变]   刷题打卡	
1024肥宅	3年前	⊚ 1.4k	<b>I</b> △ 10	;;; 1	算法
[LeetCode0303题区域和检索 - 数组不可变]   刷题打卡					
1024肥宅	3年前	⊚ 1.2k	<u>ı</u> ∆ 7	<b>₩</b> 4	算法
[LeetCode513 <b>题找树左下角的值</b> ]   刷题打卡					
1024肥宅	3年前	◎ 1.0k	<b>I</b> △ 10	<b>∵</b> 2	算法
Webpack	上传腾讯	マイス   项目	复盘		
1024肥宅	3年前	⊚ 978	<b>1</b> △8	··· 2	Webpack
Docker <b>简单上手</b> 01					
1024肥宅	3年前	⊚ 876	<b> </b> △ 13	□ 1	Docker
[LeetCode1200. 最 <b>小绝对差</b> ]   刷题打卡					
1024肥宅	3年前	⊚ 892	<b> </b> △ 7	<b>₩</b> 2	算法
Express+MongoDB搭建图片分享社区二					
1024肥宅	3年前	◎ 656	<b>I</b> ∆ 16	○ 评论	Express
[LeetCode94题二叉树的中序遍历]   刷题打卡					
1024肥宅	3年前	⊚ 691	<b>I</b> △ 6	<b>₩</b> 6	算法
[LeetCode209题长度最小的子数组]   刷题打卡					
1024肥宅	3年前	⊚ 726	<b>l</b> ∆ 8	□ 评论	算法
前端迅速上手Linux基础教程					
1024肥宅	3年前	⊚ 417	<b> </b> △ 18	○ 评论	Linux
[LeetCode141 <b>题环形链表</b> ]   <b>刷题打卡</b>					
1024肥宅	3年前	⊚ 658	<b>I</b> △ 3		算法
[LeetCode1550题 <b>存在连续三个奇数的数组</b> ]   刷题打卡					
1024肥宅	3年前	⊚ 454	<b>1</b> △ 8	· 评论	算法
[LeetCode1438 <mark>题绝对差不超过限制的最长连续子数组</mark> ]   刷题打卡					
1024肥宅	3年前	⊚ 503	<b>I</b> △ 5	□ 评论	算法