Express+MongoDB搭建图片分享社区一

1024肥宅 2020-10-13 303 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 复制代码

```bash
mkdir Instagrammy && cd Instagrammy

npm init
```

安装express

css 复制代码

```css
npm i express
```

最终生成的package.json文件如下：

swift 复制代码

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

```
14    }
15 }
```

然后在项目根目录新建入口文件server.js

javascript 复制代码

```javascript
const express = require('express');

app = express();

app.set('port', process.env.PORT || 3000);

app.get('/', function(req, res) {
    res.send('Hello World!');
})

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

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 复制代码

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

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

```javascript
function someMiddleware(req, res, next) {}

```

添加中间件的代码则非常简单：

```ini
app.use(middlewareA);
app.use(middlewareB);
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
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');
const morgan = require('morgan');
const methodOverride = require('method-override');
const errorHandler = require('errorhandler');

module.exports = function(app) {
    app.use(morgan('dev'));
    app.use(bodyParser.urlencoded({ extended: true }));
    app.use(bodyParser.json());
    app.use(methodOverride());
    app.use(cookieParser('secret-value'));
    app.use('/public/', express.static(path.join(__dirname, '../public')));
```

```
    }

    return app;
}
```

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 复制代码

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

app = express();
app = configure(app);

app.set('port', process.env.PORT || 3000);

app.get('/', function(req, res) {
    res.send('Hello World!');
})

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

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：

ini 复制代码

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

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 复制代码

```lua
module.exports = {
  index: function(req, res) {
    res.send('The image:index controller ' + req.params.image_id);
  },
  create: function(req, res) {
    res.send('The image:create POST controller');
  },
  like: function(req, res) {
    res.send('The image:like POST controller');
  },
  comment: function(req, res) {
    res.send('The image:comment POST controller');
  },
};
```

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：

ini 复制代码

```ini
const express = require('express');

const router = express.Router();
const home = require('../controllers/home');
const image = require('../controllers/image');

module.exports = function(app) {
  router.get('/', home.index);
  router.get('/images/:image_id', image.index);
  router.post('/images', image.create);
  router.post('/images/:image_id/like', image.like);
  router.post('/images/:image_id/comment', image.comment);
  app.use(router);
};
```

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。

ini 复制代码

```ini
const path = require('path');
const exphbs = require('express-handlebars');
const express = require('express');
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const morgan = require('morgan');
const methodOverride = require('method-override');
const errorHandler = require('errorhandler');

const routes = require('./routes');

module.exports = function(app) {
  app.use(morgan('dev'));
  app.use(bodyParser.urlencoded({ extended: true }));
  app.use(bodyParser.json());
  app.use(methodOverride());
  app.use(cookieParser('secret-value'));
  app.use('/public/', express.static(path.join(__dirname, '../public')));

  if (app.get('env') === 'development') {
    app.use(errorHandler());
  }

  routes(app);
  return app;
};
```

最后我们去掉 server.js 中原来的首页路由。

ini 复制代码

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

app = express();
app = configure(app);
app.set('port', process.env.PORT || 3000);

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

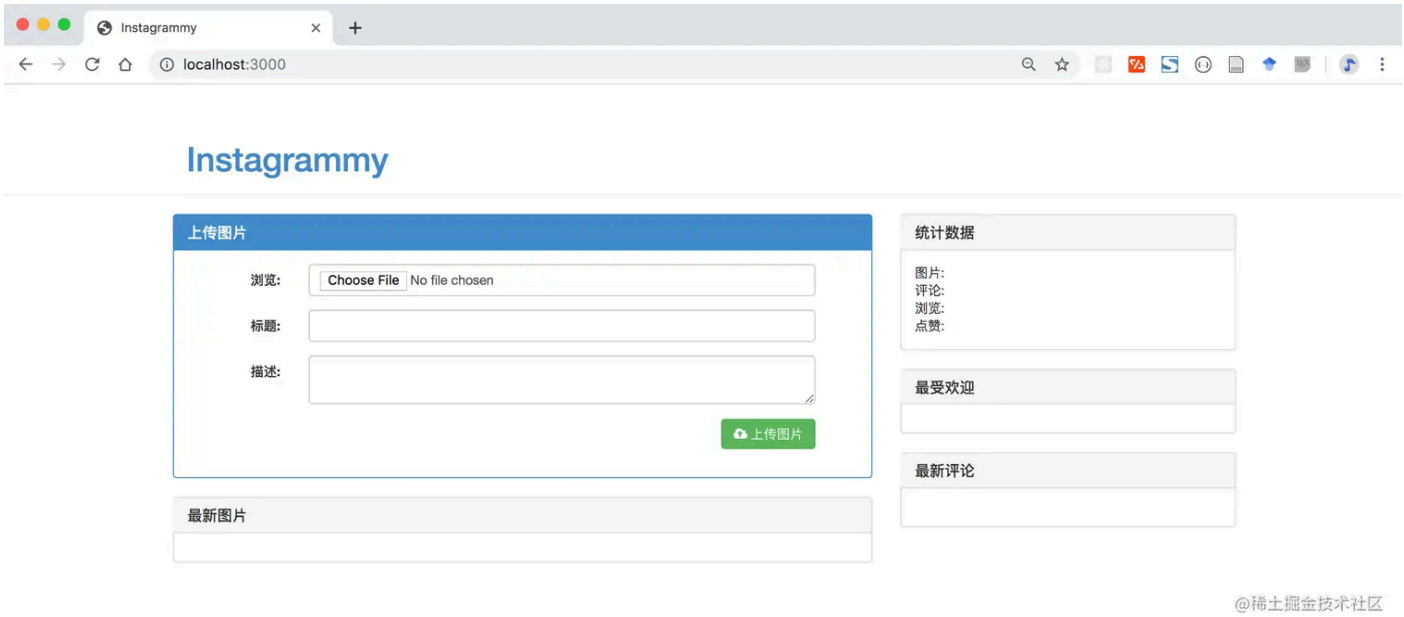
现在我们运行服务器

```
npm start
```

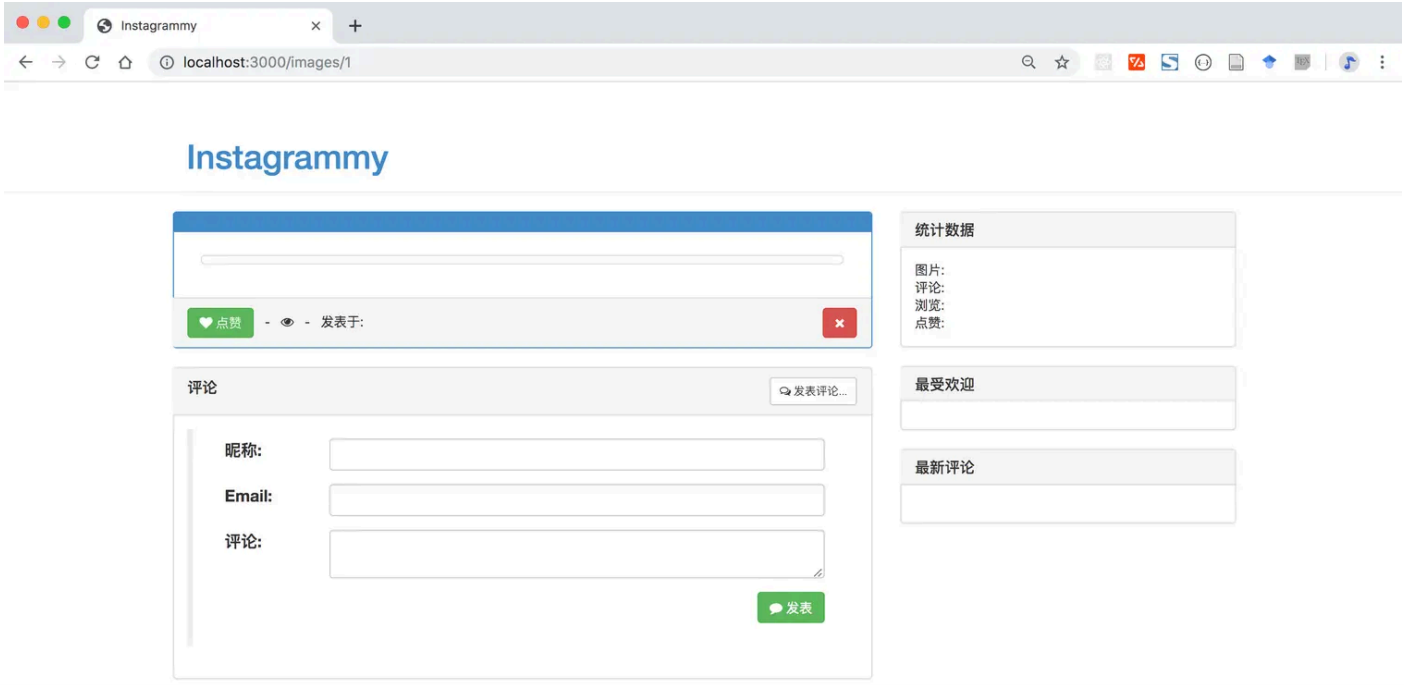
打开浏览器访问localhost:3000可以看到页面展示的The home:index controller。访问localhost:3000/images/test123，则是The image:index controller test123。

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：



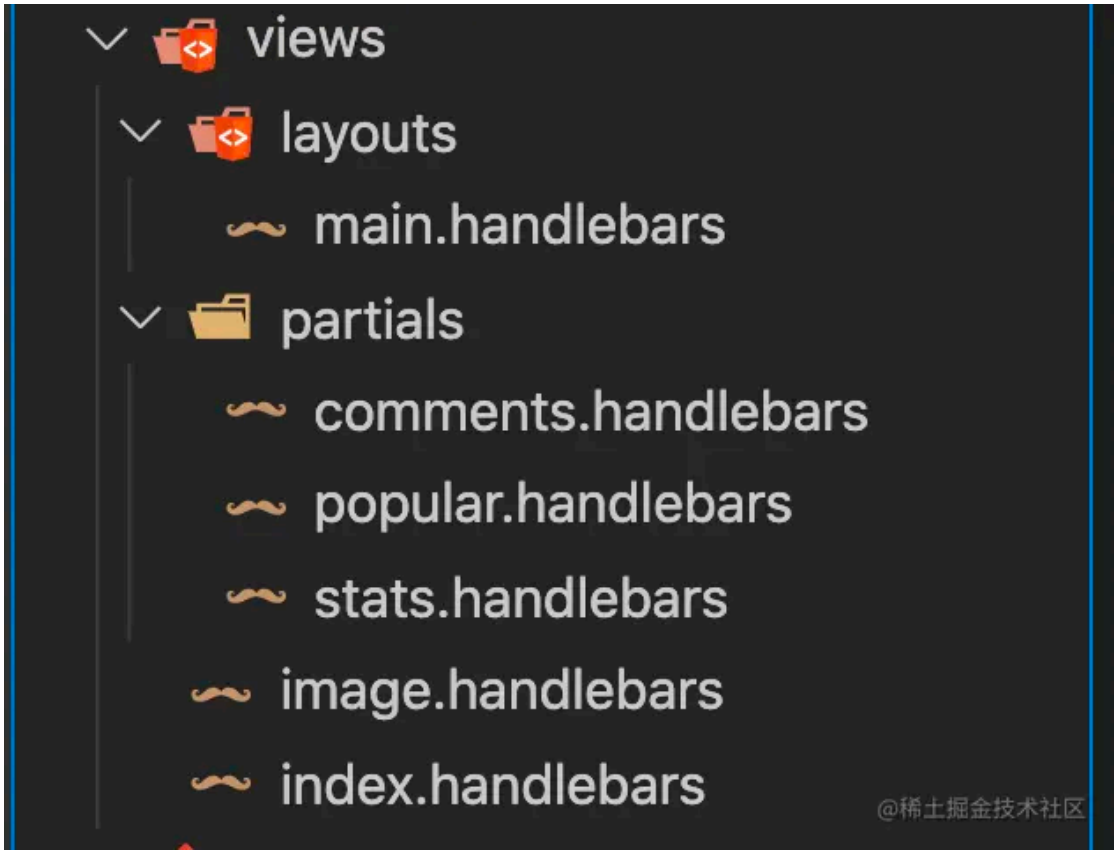
图片详情页的效果如下图:



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：

xml 复制代码

```xml
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Instagrammy</title>
```

```
    <div class="page-header">
        <div class="container">
            <div class="col-md-6">
                <h1><a href="/">Instagrammy</a></h1>
            </div>
        </div>
    </div>

    <div class="container">
        <div class="row">
            <div class="col-sm-8">{{{body}}}</div>
        </div>
    </div>
</body>
</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:

```css
<h1>Index Page</h1>
```

图片详情页面内容:

```css
<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');
  },
};
```

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

lua 复制代码

```lua
module.exports = {
  index: function(req, res) {
    res.render('image');
  },
  create: function(req, res) {
    res.send('The image:create POST controller');
  },
  like: function(req, res) {
    res.send('The image:like POST controller');
  },
  comment: function(req, res) {
    res.send('The image:comment POST controller');
  },
};
```

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

ini 复制代码

```ini
const path = require('path');
const exphbs = require('express-handlebars');
const express = require('express');
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const morgan = require('morgan');
const methodOverride = require('method-override');
const errorHandler = require('errorhandler');

const routes = require('./routes');

module.exports = function(app) {
  app.engine('handlebars', exphbs());
  app.set('view engine', 'handlebars');

  app.use(morgan('dev'));
  app.use(bodyParser.urlencoded({ extended: true }));
  app.use(bodyParser.json());
  app.use(methodOverride());
  app.use(cookieParser('secret-value'));
  app.use('/public/', express.static(path.join(__dirname, '../public')));

  if (app.get('env') === 'development') {
    app.use(errorHandler());
  }
```

```
  return app;
};
```

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 复制代码

```
{{!-- views/index.handlebars --}}

<div class="panel panel-primary">
  <div class="panel-heading">
    <h3 class="panel-title">
      上传图片
    </h3>
  </div>
  <form action="/images" method="post" enctype="multipart/form-data">
    <div class="panel-body form-horizontal">
      <div class="form-group col-md-12">
        <label for="file" class="col-sm-2 control-label">浏览:</label>
        <div class="col-md-10">
          <input type="file" class="form-control" name="file" id="file">
        </div>
      </div>
      <div class="form-group col-md-12">
        <label for="title" class="col-md-2 control-label">标题:</label>
        <div class="col-md-10">
          <input type="text" class="form-control" name="title">
        </div>
      </div>
      <div class="form-group col-md-12">
        <label for="description" class="col-md-2 control-label">描述:</label>
        <div class="col-md-10">
          <textarea name="description" rows="2" class="form-control"></textarea>
        </div>
      </div>
      <div class="form-group col-md-12">
        <div class="col-md-12 text-right">
          <button type="submit" id="login-btn" class="btn btn-success">
            <i class="fa fa-cloud-upload"> 上传图片</i>
          </button>
        </div>
      </div>
```

```
</div>

<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">
      最新图片
    </h3>
  </div>
  <div class="panel-body">
    {{#each images}}
      <div class="col-md-4 text-center" style="padding-bottom: 1em;">
        <a href="/images/{{uniqueId}}">
          <img src="/public/upload/{{filename}}" alt="{{title}}"
            style="width: 175px; height: 175px;" class="img-thumbnail">
        </a>
      </div>
    {{/each}}
  </div>
</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 uniqueId 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。

xml 复制代码

```
{{!-- views/image.handlebars --}}
<div class="panel panel-primary">
  <div class="panel-heading">
    <h2 class="panel-title">{{image.title}}</h2>
  </div>
  <div class="panel-body">
    <p>{{image.description}}</p>
    <div class="col-md-12 text-center">
      <img src="/public/upload/{{image.filename}}" alt="{{image.title}}"
        class="thumbnail">
    </div>
  </div>
  <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>   -  
        <i class="fa fa-eye"></i>
        <strong>{{image.views}}</strong>
```

```
        <button class="btn btn-danger" id="btn-delete" data-id="{{image.uniqueId}}">
          <i class="fa fa-times"></i>
        </button>
      </div>
    </div>
  </div>
</div>

<div class="panel panel-default">
  <div class="panel-heading">
    <div class="row">
      <div class="col-md-8">
        <strong class="panel-title">评论</strong>
      </div>
      <div class="col-md-4 text-right">
        <button class="btn btn-default btn-sm" id="btn-comment" data-id="{{image.uniqueId}}">
          <i class="fa fa-comments-o"> 发表评论...</i>
        </button>
      </div>
    </div>
  </div>
  <div class="panel-body">
    <blockquote id="post-comment">
      <div class="row">
        <form action="/images/{{image.uniqueId}}/comment" method="post">
          <div class="form-group col-sm-12">
            <label for="name" class="col-sm-2 control-label">昵称:</label>
            <div class="col-sm-10"><input type="text" class="form-control" name="name"></div>
          </div>
          <div class="form-group col-sm-12">
            <label for="email" class="col-sm-2 control-label">Email:</label>
            <div class="col-sm-10"><input type="text" class="form-control" name="email"></div>
          </div>
          <div class="form-group col-sm-12">
            <label for="comment" class="col-sm-2 control-label">评论:</label>
            <div class="col-sm-10">
              <textarea name="comment" class="form-control" rows="2"></textarea>
            </div>
          </div>
          <div class="form-group col-sm-12">
            <div class="col-sm-12 text-right">
              <button class="btn btn-success" id="comment-btn" type="button">
                <i class="fa fa-comment"></i> 发表
              </button>
            </div>
          </div>
        </form>
      </div>
    </blockquote>
    <ul class="media-list">
```

```
          <img src="http://www.gravatar.com/avatar/{{gravatar}}?d=monsterid&s=45"
            class="media-object img-circle">
        </a>
        <div class="media-body">
          {{comment}}
          <br/>
          <strong class="media-heading">{{name}}</strong>
          <small class="text-muted">{{timestamp}}</small>
        </div>
      </li>
      {{/each}}
    </ul>
  </div>
</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：

ini 复制代码

```
{{!-- views/partials/stats.handlebars --}}
<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">统计数据</h3>
  </div>
  <div class="panel-body">
    <div class="row">
      <div class="col-md-4 text-left">图片:</div>
      <div class="col-md-8 text-right">{{sidebar.stats.images}}</div>
    </div>
    <div class="row">
      <div class="col-md-4 text-left">评论:</div>
      <div class="col-md-8 text-right">{{sidebar.stats.comments}}</div>
    </div>
    <div class="row">
      <div class="col-md-4 text-left">浏览:</div>
      <div class="col-md-8 text-right">{{sidebar.stats.views}}</div>
    </div>
    <div class="row">
      <div class="col-md-4 text-left">点赞:</div>
      <div class="col-md-8 text-right">{{sidebar.stats.likes}}</div>
    </div>
  </div>
</div>
```

ini 复制代码

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

最新评论组件（comments.handlebars）：

ini 复制代码

```ini
{{!-- 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">
    <ul class="media-list">
      {{#each sidebar.comments}}
      <li class="media">
        <a href="/images/{{image.uniqueId}}" class="pull-left">
          <img src="/public/upload/{{image.filename}}" class="media-object"
            height="45" width="45">
        </a>
        <div class="media-body">
          {{comment}}<br/>
          <strong class="media-heading">{{name}}</strong>
          <small class="text-muted">{{timestamp}}</small>
        </div>
      </li>
      {{/each}}
    </ul>
  </div>
</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 复制代码

```xml
{{!-- views/layouts/main.handlebars --}}

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Instagrammy</title>
  <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">
</head>

<body>
  <div class="page-header">
    <div class="container">
      <div class="col-md-6">
        <h1><a href="/">Instagrammy</a></h1>
      </div>
    </div>
  </div>

  <div class="container">
    <div class="row">
      <div class="col-sm-8">{{{body}}}</div>
      <div class="col-sm-4">
        {{> stats this}}
        {{> popular this}}
        {{> comments this}}
      </div>
    </div>
  </div>
</body>

</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

```
module.exports = {
  index: function(req, res) {
    const viewModel = {
      images: [
        {
          uniqueId: 1,
          title: '示例图片1',
          description: '',
          filename: 'sample1.jpg',
          views: 0,
          likes: 0,
          timestamp: Date.now(),
        },
        {
          uniqueId: 2,
          title: '示例图片2',
          description: '',
          filename: 'sample2.jpg',
          views: 0,
          likes: 0,
          timestamp: Date.now(),
        },
        {
          uniqueId: 3,
          title: '示例图片3',
          description: '',
          filename: 'sample3.jpg',
          views: 0,
          likes: 0,
          timestamp: Date.now(),
        },
      ],
    };
    res.render('index', viewModel);
  },
};
```

javascript 复制代码

```
{{!-- controllers/image.js --}}

module.exports = {
  index: function(req, res) {
    const viewModel = {
      image: {
        uniqueId: 1,
        title: '示例图片1',
        description: '这是张测试图片',
        filename: 'sample1.jpg',
        views: 0
```

```
      },
      comments: [
        {
          image_id: 1,
          email: 'test@testing.com',
          name: 'Test Tester',
          comment: 'Test 1',
          timestamp: Date.now(),
        },
        {
          image_id: 1,
          email: 'test@testing.com',
          name: 'Test Tester',
          comment: 'Test 2',
          timestamp: Date.now(),
        },
      ],
    };
    res.render('image', viewModel);
  },
  create: function(req, res) {
    res.send('The image:create POST controller');
  },
  like: function(req, res) {
    res.send('The image:like POST controller');
  },
  comment: function(req, res) {
    res.send('The image:comment POST controller');
  },
};
```

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

npm i moment
```

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

```
ini                                                    复制代码

{{!-- server/configure.js --}}

const path = require('path');
const exphbs = require('express-handlebars');
```

```
const cookieParser = require('cookie-parser');
const morgan = require('morgan');
const methodOverride = require('method-override');
const errorHandler = require('errorhandler');
const moment = require('moment');

const routes = require('./routes');

module.exports = function(app) {
  // 定义 moment 全局语言
  moment.locale('zh-cn');

  app.engine(
    'handlebars',
    exphbs.create({
      helpers: {
        timeago: function(timestamp) {
          return moment(timestamp).startOf('minute').fromNow();
        },
      },
    }).engine,
  );
  app.set('view engine', 'handlebars');

  app.use(morgan('dev'));
  app.use(bodyParser.urlencoded({ extended: true }));
  app.use(bodyParser.json());
  app.use(methodOverride());
  app.use(cookieParser('secret-value'));
  app.use('/public/', express.static(path.join(__dirname, '../public')));

  if (app.get('env') === 'development') {
    app.use(errorHandler());
  }

  routes(app);
  return app;
};
```

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>  -  
                <i class="fa fa-eye"></i>
                <strong>{{image.views}}</strong>
                  -   发表于: <em class="text-muted">{{timeago image.timestamp}}</em>
            </div>
            <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>
    </div>
</div>
```



```
        <ul class="media-list">
            {{#each comments}}
            <li class="media">
                <a href="#" class="pull-left">
                    <img src="http://www.gravatar.com/avatar/{{gravatar}}?d=monsterid&s=45" alt="" cla
                </a>
                <div class="media-body">
                    {{comment}}
                    <br/>
                    <strong class="media-heading">{{name}}</strong>
                    <small class="text-muted">{{timeago timestamp}}</small>
                </div>
            </li>
            {{/each}}
        </ul>
```

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">
        <ul class="media-list">
            {{#each sidebar.comments}}
            <li class="media">
                <a href="/images/{{image.uniqueId}}" class="pull-left">
                    <img src="/public/upload/{{image.filename}}" class="media-object" height="45" widt
                </a>
                <div class="media-body">
                    {{comment}}<br/>
                    <strong class="media-heading">{{name}}</strong>
                    <small class="text-muted">{{timeago timestamp}}</small>
                </div>
            </li>
            {{/each}}
        </ul>
    </div>
</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 发送

暂无评论数据

目录 收起

搜索建议 搜索关键词

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 2.1k 72 18 前端

Typora+Vue主题+Gitee图床轻松写文章

[LeetCode0304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 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 7 4 算法

[LeetCode513题找树左下角的值] | 刷题打卡

1024肥宅 3年前 1.0k 10 2 算法

Webpack上传腾讯云 | 项目复盘

1024肥宅 3年前 978 8 2 Webpack

Docker简单上手01

1024肥宅 3年前 876 13 1 Docker

[LeetCode1200. 最小绝对差] | 刷题打卡

1024肥宅 3年前 892 7 2 算法

Express+MongoDB搭建图片分享社区二

1024肥宅 3年前 656 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 6 6 算法

[LeetCode209题长度最小的子数组] | 刷题打卡

1024肥宅 3年前 726 8 评论 算法

前端迅速上手Linux基础教程

1024肥宅 3年前 417 18 评论 Linux

[LeetCode141题环形链表] | 刷题打卡

1024肥宅 3年前 658 3 评论 算法

[LeetCode1550题存在连续三个奇数的数组] | 刷题打卡

1024肥宅 3年前 454 8 评论 算法

[LeetCode1438题绝对差不超过限制的最长连续子数组] | 刷题打卡

1024肥宅 3年前 503 5 评论 算法