# 实验16：制作一个简单的twitter

实验背景

利用已掌握的知识，完成实例。如图所示：



图1-1

实验目的

1、掌握工程入口及路由规划

2、掌握制作模板实现静态页

3、掌握实现模块功能

实验分析

本次实验要完成《制作一个简单的twitter》

实验步骤

1、安装MongoDB

mongod --dbpath ../blog/

2、连接MongoDB

在package.json的dependencies 中添加一行：

"mongodb": "2.1.19"

然后运行 npm install 更新依赖的模块

3、在工程的根目录中创建 settings.js 文件

module.exports = {

cookieSecret: 'myblog',

db: 'blog',

host: 'localhost',

port: 27017

};

其中 db 是数据库的名称，host 是数据库的地址，port是数据库的端口号，cookieSecret 用于 Cookie 加密与数据库无关

4、根目录下新建 models 文件夹，并在 models 文件夹下新建 db.js

var settings = require('../settings'),

Db = require('mongodb').Db,

Connection = require('mongodb').Connection,

Server = require('mongodb').Server;

module.exports = new Db(settings.db, new Server(settings.host, settings.port),

{safe: true});

其中通过 new Db(settings.db, new Server(settings.host, settings.port), {safe: true}); 设置数据库名、数据库地址和数据库端口创建了一个数据库连接实例，并通过 module.exports 导出该实例。

5、打开 app.js，在 var routes = require('./routes/index'); 下添加：

var settings = require('./settings');

6、会话支持

需要借助 express-session 和 connect-mongo 这两个第三方中间件，在 package.json 中添加：

"express-session": "1.13.0",

"connect-mongo": "1.2.0"

注意： 如报"error setting ttl index on collection : sessions"错误，把"mongodb"&"connect-mongo"版本号更到最新。

7、运行npm install安装模块,打开app.js，添加以下代码：

var session = require('express-session');

var MongoStore = require('connect-mongo')(session);

app.use(session({

secret: settings.cookieSecret,

key: settings.db,//cookie name

cookie: {maxAge: 1000 \* 60 \* 60 \* 24 \* 30},//30 days

store: new MongoStore({

db: settings.db,

host: settings.host,

port: settings.port

})

}));

注意： connect-mongo 最新版需要改成如：

app.use(session({

secret: settings.cookieSecret,

key: settings.db,//cookie name

cookie: {maxAge: 1000 \* 60 \* 60 \* 24 \* 30},//30 days

store: new MongoStore({

url: 'mongodb://localhost/blog'

})

}));

8.注册响应

在 models 文件夹下新建 user.js，添加如下代码：

var mongodb = require('./db');

function User(user) {

this.name = user.name;

this.password = user.password;

this.email = user.email;

};

module.exports = User;

//存储用户信息

User.prototype.save = function(callback) {

//要存入数据库的用户文档

var user = {

name: this.name,

password: this.password,

email: this.email

};

//打开数据库

mongodb.open(function (err, db) {

if (err) {

return callback(err);//错误，返回 err 信息

}

//读取 users 集合

db.collection('users', function (err, collection) {

if (err) {

mongodb.close();

return callback(err);//错误，返回 err 信息

}

//将用户数据插入 users 集合

collection.insert(user, {

safe: true

}, function (err, user) {

mongodb.close();

if (err) {

return callback(err);//错误，返回 err 信息

}

callback(null, user[0]);//成功！err 为 null，并返回存储后的用户文档

});

});

});

};

//读取用户信息

User.get = function(name, callback) {

//打开数据库

mongodb.open(function (err, db) {

if (err) {

return callback(err);//错误，返回 err 信息

}

//读取 users 集合

db.collection('users', function (err, collection) {

if (err) {

mongodb.close();

return callback(err);//错误，返回 err 信息

}

//查找用户名（name键）值为 name 一个文档

collection.findOne({

name: name

}, function (err, user) {

mongodb.close();

if (err) {

return callback(err);//失败！返回 err 信息

}

callback(null, user);//成功！返回查询的用户信息

});

});

});

};

9.页面通知

flash 即 connect-flash 模块，flash 是一个在 session 中用于存储信息的特定区域。信息写入 flash ，下一次显示完毕后即被清除。典型的应用是结合重定向的功能，确保信息是提供给下一个被渲染的页面。

在 package.json 添加一行代码：

"connect-flash": "0.1.1"

修改 app.js ，在 var settings = require('./settings'); 后添加：

var flash = require('connect-flash');

在 app.set('view engine', 'ejs'); 后添加：

app.use(flash());

10.打开 index.js，在最前面添加如下代码：

var crypto = require('crypto')；

var User = require('../models/user.js');

通过 require() 引入 crypto 模块和 user.js 用户模型文件，crypto 是 Node.js 的一个核心模块，我们用它生成散列值来加密密码。

修改 index.js 中 app.post('/reg') 如下：

app.post('/reg', function (req, res) {

var name = req.body.name,

password = req.body.password,

password\_re = req.body['password-repeat'];

//检验用户两次输入的密码是否一致

if (password\_re != password) {

req.flash('error', '两次输入的密码不一致!');

return res.redirect('/reg');//返回注册页

}

//生成密码的 md5 值

var md5 = crypto.createHash('md5'),

password = md5.update(req.body.password).digest('hex');

var newUser = new User({

name: name,

password: password,

email: req.body.email

});

//检查用户名是否已经存在

User.get(newUser.name, function (err, user) {

if (err) {

req.flash('error', err);

return res.redirect('/');

}

if (user) {

req.flash('error', '用户已存在!');

return res.redirect('/reg');//返回注册页

}

//如果不存在则新增用户

newUser.save(function (err, user) {

if (err) {

req.flash('error', err);

return res.redirect('/reg');//注册失败返回主册页

}

req.session.user = newUser;//用户信息存入 session

req.flash('success', '注册成功!');

res.redirect('/');//注册成功后返回主页

});

});

});

11.修改 header.ejs，将 <nav></nav> 修改如下：

<nav>

<span><a title="主页" href="/">home</a></span>

<% if (user) { %>

<span><a title="发表" href="/post">post</a></span>

<span><a title="登出" href="/logout">logout</a></span>

<% } else { %>

<span><a title="登录" href="/login">login</a></span>

<span><a title="注册" href="/reg">register</a></span>

<% } %>

</nav>

12.在 <article> 后添加如下代码：

<% if (success) { %>

<div><%= success %></div>

<% } %>

<% if (error) { %>

<div><%= error %> </div>

<% } %>

13.修改 index.js ，将 app.get('/') 修改如下：

app.get('/', function (req, res) {

res.render('index', {

title: '主页',

user: req.session.user,

success: req.flash('success').toString(),

error: req.flash('error').toString()

});

});

14.将 app.get('reg') 修改如下：

app.get('/reg', function (req, res) {

res.render('reg', {

title: '注册',

user: req.session.user,

success: req.flash('success').toString(),

error: req.flash('error').toString()

});

});

15登录功能

打开 index.js ，将 app.post('/login') 修改如下：

app.post('/login', function (req, res) {

//生成密码的 md5 值

var md5 = crypto.createHash('md5'),

password = md5.update(req.body.password).digest('hex');

//检查用户是否存在

User.get(req.body.name, function (err, user) {

if (!user) {

req.flash('error', '用户不存在!');

return res.redirect('/login');//用户不存在则跳转到登录页

}

//检查密码是否一致

if (user.password != password) {

req.flash('error', '密码错误!');

return res.redirect('/login');//密码错误则跳转到登录页

}

//用户名密码都匹配后，将用户信息存入 session

req.session.user = user;

req.flash('success', '登陆成功!');

res.redirect('/');//登陆成功后跳转到主页

});

});

将 app.get('/login') 修改如下：

app.get('/login', function (req, res) {

res.render('login', {

title: '登录',

user: req.session.user,

success: req.flash('success').toString(),

error: req.flash('error').toString()});

});

16.登出功能

接下来我们实现登出响应。修改 app.get('/logout') 如下：

app.get('/logout', function (req, res) {

req.session.user = null;

req.flash('success', '登出成功!');

res.redirect('/');//登出成功后跳转到主页

});

17.发表文章

在 views 文件夹下新建 post.ejs ，添加如下代码：

<%- include header %>

<form method="post">

标题：<br />

<input type="text" name="title" /><br />

正文：<br />

<textarea name="post" rows="20" cols="100"></textarea><br />

<input type="submit" value="发表" />

</form>

<%- include footer %>

18.文章模型

仿照用户模型，我们将文章模型命名为 Post 对象，它拥有与 User 相似的接口，分别是 Post.get 和 Post.prototype.save 。Post.get 的功能是从数据库中获取文章，可以按指定用户获取，也可以获取全部的内容。Post.prototype.save 是 Post 对象原型的方法，用来将文章保存到数据库。

在 models 文件夹下新建 post.js ，添加如下代码：

var mongodb = require('./db');

function Post(name, title, post) {

this.name = name;

this.title = title;

this.post = post;

}

module.exports = Post;

//存储一篇文章及其相关信息

Post.prototype.save = function(callback) {

var date = new Date();

//存储各种时间格式，方便以后扩展

var time = {

date: date,

year : date.getFullYear(),

month : date.getFullYear() + "-" + (date.getMonth() + 1),

day : date.getFullYear() + "-" + (date.getMonth() + 1) + "-" + date.getDate(),

minute : date.getFullYear() + "-" + (date.getMonth() + 1) + "-" + date.getDate() + " " +

date.getHours() + ":" + (date.getMinutes() < 10 ? '0' + date.getMinutes() : date.getMinutes())

}

//要存入数据库的文档

var post = {

name: this.name,

time: time,

title: this.title,

post: this.post

};

//打开数据库

mongodb.open(function (err, db) {

if (err) {

return callback(err);

}

//读取 posts 集合

db.collection('posts', function (err, collection) {

if (err) {

mongodb.close();

return callback(err);

}

//将文档插入 posts 集合

collection.insert(post, {

safe: true

}, function (err) {

mongodb.close();

if (err) {

return callback(err);//失败！返回 err

}

callback(null);//返回 err 为 null

});

});

});

};

//读取文章及其相关信息

Post.get = function(name, callback) {

//打开数据库

mongodb.open(function (err, db) {

if (err) {

return callback(err);

}

//读取 posts 集合

db.collection('posts', function(err, collection) {

if (err) {

mongodb.close();

return callback(err);

}

var query = {};

if (name) {

query.name = name;

}

//根据 query 对象查询文章

collection.find(query).sort({

time: -1

}).toArray(function (err, docs) {

mongodb.close();

if (err) {

return callback(err);//失败！返回 err

}

callback(null, docs);//成功！以数组形式返回查询的结果

});

});

});

};

20.发表响应

接下来我们给发表文章注册响应，打开 index.js ，在 User = require('../models/user.js') 后添加一行代码：

var Post = require('../models/post.js');

修改 app.post('/post') 如下：

app.post('/post', checkLogin);

app.post('/post', function (req, res) {

var currentUser = req.session.user,

post = new Post(currentUser.name, req.body.title, req.body.post);

post.save(function (err) {

if (err) {

req.flash('error', err);

return res.redirect('/');

}

req.flash('success', '发布成功!');

res.redirect('/');//发表成功跳转到主页

});

});

最后，我们修改 index.ejs ，让主页右侧显示发表过的文章及其相关信息。

打开 index.ejs ，修改如下：

<%- include header %>

<% posts.forEach(function (post, index) { %>

<p><h2><a href="#"><%= post.title %></a></h2></p>

<p class="info">

作者：<a href="#"><%= post.name %></a> |

日期：<%= post.time.minute %>

</p>

<p><%- post.post %></p>

<% }) %>

<%- include footer %>

打开 index.js ，修改 app.get('/') 如下：

app.get('/', function (req, res) {

Post.get(null, function (err, posts) {

if (err) {

posts = [];

}

res.render('index', {

title: '主页',

user: req.session.user,

posts: posts,

success: req.flash('success').toString(),

error: req.flash('error').toString()

});

});

});

扩展练习

1.思考并实现，完成《通知页面》并实现测试

实验思考

1.思考并实现，独立设计一个小应用。