三、几个框架

1.ejs

2.cookie-parser

3.expression-session

4.connect-mongo

5.connectflash

1. ejs模板引擎（类比asp.net mvc中的webform引擎和razor引擎）

模板引擎（Template Engine）是一个将页面模板和数据结合起来生成 html 的工具。上次课中，我们给浏览器返回了字符串，html文件，重定向。本次，利用模板引擎返回html 页面给浏览器。

安装ejs:npm i ejs –save

index.js:

var path = require('path');

var express = require('express');

var app = express();

var indexRouter = require('./routes/index');

var userRouter = require('./routes/users');

app.set('views', path.join(\_\_dirname, 'views'));// 设置存放模板文件的目录

app.set('view engine', 'ejs');// 设置模板引擎为 ejs

app.use('/', indexRouter);

app.use('/users', userRouter);

app.listen(3000);

views/users.ejs:

<!DOCTYPE html>

<html>

<head>

<style type="text/css">

body {padding: 50px;font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;}

</style>

</head>

<body>

<h1><%= name.toUpperCase() %></h1>

<p>hello, <%= name %></p>

</body>

</html>

routes/users.js:

var express = require('express');

var router = express.Router();

router.get('/:name', function(req, res) {

res.render('users', {

name: req.params.name

});

});

module.exports = router;

启动服务器，访问：localhost:3000/users/haha



1. <% code %>：运行 JavaScript 代码，不输出
2. <%= code %>：显示转义后的 HTML内容
3. <%- code %>：显示原始 HTML 内容

<% code %>使用例子：

**Data**

supplies: ['mop', 'broom', 'duster']

**Template**

<ul>

<% for(var i=0; i<supplies.length; i++) {%>

<li><%= supplies[i] %></li>

<% } %>

</ul>

**Result**

<ul>

<li>mop</li>

<li>broom</li>

<li>duster</li>

</ul>

模板更大的价值在于分拆成可复用的模板片段，然后组合使用：

**views/header.ejs**

<!DOCTYPE html>

<html>

<head>

<style type="text/css">

body {padding: 50px;font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;}

</style>

</head>

<body>

**views/footer.ejs**

</body>

</html>

**views/users.ejs**

<%- include('header') %>

<h1><%= name.toUpperCase() %></h1>

<p>hello, <%= name %></p>

<%- include('footer') %>

1. 理解restful架构：<http://www.ruanyifeng.com/blog/2011/09/restful>

post,put,get,delete 一切都是资源，对资源的操作

1. express-session（session中间件）: <https://www.npmjs.com/package/express-session>

http://www.cnblogs.com/chenchenluo/p/4197181.html

1. cookieParser: <http://www.cnblogs.com/coolicer/p/4191548.html>

asp.net中cookie对象；

var cookies = System.Web.HttpContext.Current.Response.Cookies;

express中的cookie处理：

res.cookie(**name**, **value** [, options]);

session是什么？session的工作流程

var express = require('express');

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

var cookieParser = require('cookie-parser');

var app = express();

app.use(cookieParser());

app.use(session({

secret: '12345',

cookie: {maxAge: 80000 }, //设置maxAge是80000ms，即80s后session和相应的cookie失效过期

resave: false,

saveUninitialized: true,

}));

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

if(req.session.lastPage) {

console.log('Last page was: ' + req.session.lastPage + ".");

}

req.session.lastPage = '/awesome'; //每一次访问时，session对象的lastPage会自动的保存或更新内存中的session中去。

res.send("You're Awesome. And the session expired time is: " + req.session.cookie.maxAge);

});

app.listen(5000);

在使用monogodb存储session时，最小的maxAge（session有效期不能小于60s），因为monogodb会每隔60s检查一次过期失效的session并清除。如果maxAge不设置，默认为null,这样的expire的时间就是浏览器的关闭时间，即每次关闭浏览器的时候，session都会失效。

Connect-mongo:用于将session存入mongo中

[https://www.npmjs.com/package/connect-mongo](http://www.npmjs.com/package/connect-mongo) 有详细的配置说明

Session存储在monogodb中：

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

app.use(cookieParser());

app.use(session({

secret: '12345',

name: 'testapp',

cookie: {maxAge: 80000 },

resave: false, //don’t save session if unmodified

saveUninitialized: true,

store: new MongoStore({ //创建新的mongodb数据库

~~host: 'localhost/testdb',~~ //数据库的地址，本机的话就是127.0.0.1，也可以是网络主 机

url: 'mongodb://localhost/testdb', //不同的connect-mongo版本写法不同，早期的版

本写host，最新的版本要按照url的写法

port: 27017, //数据库的端口号

~~db: 'test-app' //数据库的名称。~~

})

}));

跟session的内存存储一样，只需增加红色部分的store选项即可，app会自动替我们把session存入到mongodb数据库，而非内存中。

注意：详细的配置参数参考给出的资料链接，都比较详细。

关于ISO时间：

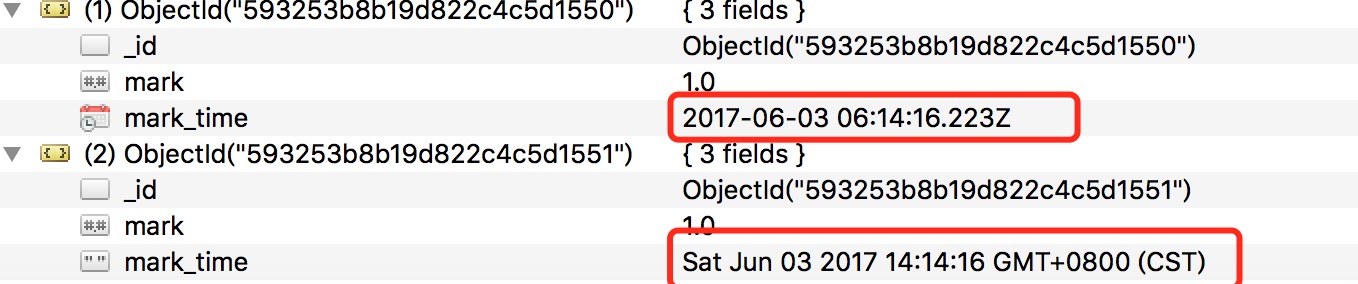
db.tianyc04.insert({mark:1,mark\_time:new Date()})

db.tianyc04.insert({mark:1,mark\_time: Date()})

db.tianyc04.find()

上边代码的意思：新建一个名为tianyc04的collection，这个collection里有两个键值对。然后打印这两个键值对。

打印出来的结果是：



大约相差8个小时

connect-flash:页面通知提示的中间件，基于session实现

<http://yunkus.com/connect-flash-usage>

flash是session中一个用于存储信息的特殊区域，消息写入到flash中，在跳转目标页中显示该消息。Flash是配置redirect一同使用的，以确保消息在目标页面中可用。

Flash可用于一次性的消息提示，比如注册，登录页面，当你再次刷新时，flash就没有提示消息了。

1.配置app.js文件：

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

var cookieParser = require('cookie-parser');

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

app.use(cookieParser('12345'));

app.use(session({

secret: '12345',

//name: 'testapp', //这里的name值得是cookie的name，默认cookie的name是：connect.sid

cookie: {maxAge: 80000 }, //设置maxAge是80000ms，即80s后session和相应的cookie失效过期

resave: false,

saveUninitialized: true,

}));

app.use(flash());

// set flash

app.use(function (req, res, next) {

res.locals.errors = req.flash('error');

res.locals.infos = req.flash('info');

next();

});

2.编写路由和逻辑

router.get('/', function(req, res, next) {

if(req.session.user)

{

req.flash('info','登录成功');

res.redirect('login');

}

else

{

res.send('未登录，请先登录！')

}

});

// 登录

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

res.render('index', { title: '欢迎登录' });

});

router.get('/loginfirst',function(req,res,next){

req.session.user = 'lichao';

res.send('登录好了');

});

npm start运行，访问localhost:3000，因为没有session，提示先登录

访问localhost:3000:loginfirst模拟登录过程，写入session信息

再次访问localhost:3000,提示已经登录成功，并且，跳转到新页面时，flash已经将登录成功的提示信息带入到了新页面。

问题：

在使用mongodb存储session信息时不可以把过期时间设置的小于60秒。