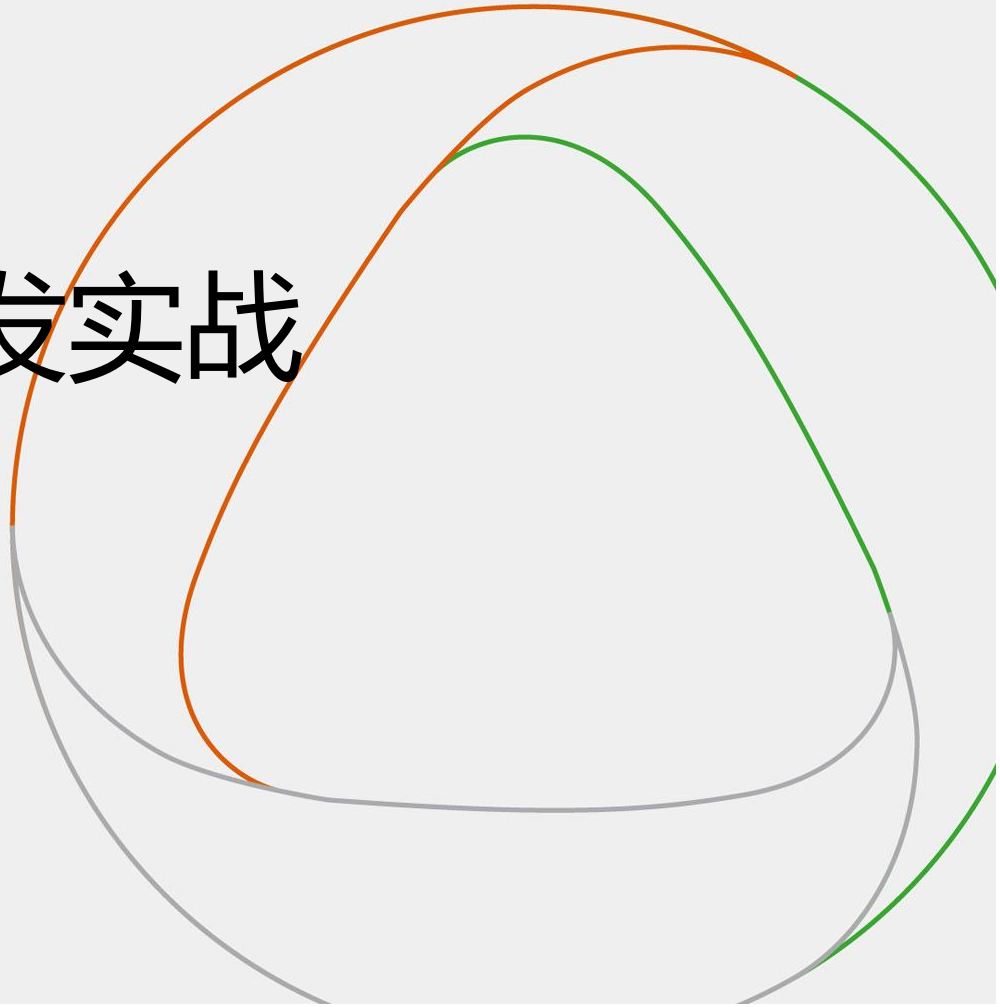
AsiaInfo 亚信

NodeJS Web开发实战

by 齐飞

qifei3@asiainfo.com



AsiaInfo 亚信

个人博客 - 以登陆/注册功能为例讲解

领航产业互联网
Leading the Business Internet

运行环境

RedHat® OpenShift PaaS 云平台

使用技术

1. Node.js >= 0.6.0
2. Express = 3.4.4
3. MongoDB >= 2.4.8
4. mongojs = 0.15.1
5. express3-handlebars = 0.5.0

主要技术介绍

Express 是一个小巧且灵活的 Node.js Web应用框架，它有一套健壮的特性，可用于开发单页、多页和混合Web应用。

MongoJS是一个轻量级的Node.js模块，它提供了一个JavaScript风格的API，帮助我们轻松连接到MongoDB！

express3-handlebars可以对Express进行扩展，使我们可以在NodeJS后端使用Handlebars模版对页面进行渲染。

代码结构 - 主要文件列表


```
├─ blogapp/
│  ├── server.js          // 主应用
│  ├─ routes/             // 控制层：路由响应逻辑
│  │  ├── index.js
│  │  └── user.js
│  ├─ views/              // 视图层：Handlebars模版
│  │  ├── userInfo.handlebars
│  │  └─ layouts/         // 模版框架
│  │     └── mainframe.handlebars
│  ├─ public/             // 静态文件、公用信息
│  │  ├── pages/
│  │  └── css, js, images ...
│  └─ misc/               // 工具模块
│     ├── db.js
│     └── util.js
```

入口文件 - Server.js

基本结构 ：

```
var express = require('express'); // 使用require声明依赖模块
var app = express();              // 生成运行实例

app.use(app.router);              // 使用路由中间件

app.get('/', function(req, res){  // 部署路由规则
    res.send('hello world');
});

app.listen(3000);                 // 开始监听
```

托管静态资源，配置登录页面路由


```
var express = require('express'),      // 增加http、path模块依赖
    http = require('http'),
    path = require('path'),
    app = express();

app.use(express.static(                // 使用static中间件托管静态资源
    path.join(__dirname, 'public')));  // 获取运行环境下的真实路径
app.use(app.router);

app.get('/', function(req, res){       // 访问根路径时显示登录页
   res.sendfile(
       path.join(__dirname, 'public/pages/login.html'));
});

// ...
```

注册功能 - 模块引用


```
var express = require('express'),
// ...
var mongojs = require('mongojs');          // 引用mongojs模块

var dbName = "/blogapp";                   // 数据库名
var connection_string =                    // 连接字符串
    "127.0.0.1" + dbName;

var crypto = require('crypto');            // 哈希加密模块

// ...
```

注册功能 - 路由器/控制层


```
var express = require('express'),
// ...

app.post('/ACTION_USER_REGISTER',          // 仅接受Post方式访问
  function(req, res){
    var username = req.param('name'),      // 从request中获取入参
    password = req.param('password'),
    email = req.param('email');

    var sha1 = crypto.createHash('sha1'); // 使用SHA1算法对密码进
    sha1.update(plain);                    行加密
    password = sha1.digest('hex');

// 下页继续 ...
```

注册功能 - 用户信息持久化 - 1


```
// 接上页
mongojs(connection_string, ['users'])          // 使用mongojs模块连接数据库
  .users.find({username: username},            // 以当前用户名进行查询
    function(err, rows){
      if(rows.length == 0){                    // 如果该用名未被使用
        mongojs(connection_string, ['users'])  // 保存当前用户
        .users.save({username: username,
          password: password,email: email
        }, function(err){                      // 回调函数
          if(!err){
            res.send({status: '200',message: '恭喜您注册成功！'});
            res.end();                         // 保存成功，向页面返回JSON
          }                                    数据
        })
      } else {

// 下页继续...
```

注册功能 - 用户信息持久化 - 2


```
// 接上页
                                                    // 如果当前用户名已存在
      } else {
        res.send({                                  // 返回注册失败的JSON数据
          status: '-1',
          message: '该用户名太流行，已被人捷足先登，再换一个吧！'
        }); res.end();                              // 关闭response流
      }
   })
});
```

用户信息 - 配置Handlebars模版支持


```
var exphbs = require('express3-handlebars');      // 模块引用声明，通过该模块使Express支持Handlebars模版

app.set('views', path.join(__dirname, 'views')); // 配置模版所在路径

app.engine('handlebars',                          // 设置Handlebars的默认母板文件
  exphbs({defaultLayout: 'mainframe'}));

app.set('view engine', 'handlebars');             // 设置Express默认模版为Handlebars

app.enable('view cache');                         // 对模版文件启用缓存
```

用户信息 - 模版文件信息


```
├─ blogapp/
│  ├── server.js
│  ├─┬ views/                          // Handlebars模版目录
│  │ ├── userInfo.handlebars           // 用户信息模版
│  │ └─┬ layouts/
│  │   └── mainframe.handlebar         // 母板文件
│  └── ...
```

母板可以将模版文件生成的HTML片段包装为完整的HTML页面。

```
<!DOCTYPE HTML>
<html>
<head>
  <title>{{title}}</title>
  <meta charset="UTF-8">
</head>
  {{{body}}}
</html>
```

用户信息 - 后端生成页面


```
app.get('/about/:username', function(req, res){  // 在路由中使用自定义参数
  var name = req.params.username;                // 获取自定义参数
  mongojs(connection_string, ['users'])

    .users.find({username: name},                // 查询用户信息
      function(err, rows){
        res.render('userInfo', {                 // 使用Handlebars模版生成页面
         title: "查看用户信息",
         username: rows[0].username,
         email: rows[0].email
      });
  });
});
```

代码重构

问题：

1. 大部分代码都在Server.js中

2. 访问数据库、对用户密码进行哈希加密等操作步骤过于繁琐。

解决方法：

1. 采用MVC思想对代码分层解耦

2. 对数据库、加密运算等常用工具，进行模块化封装

代码重构 - 分层

/routes/user.js：

```
exports.register = function(req, res){          // 注册模块
  var username = req.param('name');
  // ...
}
exports.about = function(req, res){
  var name = req.params.username;               // 用户信息模块
  // ...
}
// ...
```

/server.js：

```
app.post('/ACTION_USER_REGISTER', user.register);  // server.js中仅保留路由定义，回调函数中的代码逻辑转义到user.js中

app.get('/about/:username', user.about);
```

代码重构 - 封装工具模块

/misc/db.js:

```
var mongojs = require('mongojs');                          // 配置数据库连接信息
// ...
exports.getCollection = function(collections){             // 对mongojs进行封装，
  if(collections){                                         // 直接提供对mongoDB中
    return mongojs(connection_string, collections);        // 集合的访问能力
  } else {
    return mongojs(connection_string);
  }
}
```

/misc/util.js:

```
var crypto = require('crypto');
exports.encrypt = function(plain){
  var sha1 = crypto.createHash('sha1');     // 使用SHA1算法进行加密
  sha1.update(plain);
  return sha1.digest('hex');
}
```

登陆功能 - user.js模块


```
var db = require('../misc/db'),                    // 引入我们刚刚封装好的db.js
  util = require('../misc/util');                  和util.js模块

exports.login = function(req, res){                // 未模块暴露login方法
  var username = req.param('name')
  var password = req.param('password');

  password = util.encrypt(password);               // 利用util模块加密用户密码

  var allCollections = db.getCollection();         // 利用db模块获取数据集合
  allCollections.collection('users').find({username: username},
function(err, rows){
  if(!err && rows.length>0 && password == rows[0].password){
    // ...                                         // 判断用户输入的密码和根据
  })                                               用户名从数据库中的查出密码
}                                                  记录进行对比，判断登陆信息
                                                   是否正确，并返回相应结果。
```

登陆功能 - server.js路由配置

/server.js:

```
user = require('./routes/user');                    // 引用user模块

app.post('/ACTION_USER_LOGIN', user.login);         // 为登陆功能指定响应函数
```

移植到OpenShift - 配置端口和IP

/server.js:

```
// 在云端运行时，从环境信息中获取主机端口号和IP信息；本地调试时使用80端口
app.set('server_port', process.env.OPENSHIFT_NODEJS_PORT || 80);
app.set('server_ip', process.env.OPENSHIFT_NODEJS_IP || '127.0.0.1');
// 使用对应的端口和IP启动服务器
http.createServer(app).listen(app.get('server_port'), app.get('server_ip'), function(){
  console.log('Server is listening on port ' + app.get('server_port'));
});
```

移植到OpenShift - 配置数据库连接

/db.js:

```
var dbName = "/blogapp";  // 数据库名
var connection_string = "127.0.0.1" + dbName;  // 数据库连接字符串

// 当运行在Openshift平台时，需要使用Openshift分配的用户名/密码/地址访问
MongoDB数据库
if(process.env.OPENSHIFT_MONGODB_DB_HOST){
  connection_string =
    process.env.OPENSHIFT_MONGODB_DB_USERNAME + ":"   // 用户名
      + process.env.OPENSHIFT_MONGODB_DB_PASSWORD + "@"    // 密码
        + process.env.OPENSHIFT_MONGODB_DB_HOST   // 数据库主机地址
          + dbName;  // 数据库名
}
```

移植到OpenShift - 提交代码

执行Shell命令：

```
$ git add *                                  // 将当前目录下所有文件添加到本地Git仓库
$ git commit -a -m 'Share Meeting Demo'// 提交所有变更信息到本地Git仓库
$ git push                                   // 将本地Git仓库推送到OpenShift云端
```

AsiaInfo 亚信

项目演示：

http://qfei.wicp.net/