基于 NodeJS & MongoDB 的拉丁培训管理系统设计、实现

目录

1. 前言

虽然此文的题目是跟舞蹈相关的，但是内容本身却是纯粹的技术讨论，而拉丁舞只是个"引子"或者"噱头"。

之所以用 Node 开发这个系统一方面固然是简化拉丁舞培训报名， 最重要的还是因为最近学了 Nodejs 和 MongoDB，而学习的最好方式莫过于实践， 而且用 Node 做这个内部需求也不会有业务上的压力，比较适合技术积累。

目前应用已经初具雏形，仅供测试，感兴趣的同学可以通过：http://10.19.70.7:3000/ 访问，这个是系统的测试环境，管理员权限是开放的。如果您发现什么问题可以向我反馈。

2. 概述

本文主要介绍如何利用 NodeJS 以及相关模块和 MongoDB 等来构建一个虽小却"五脏俱全"的完整 Web 应用——拉丁培训管理系统。

NodeJS 基于 Chrome V8 Javascript 引擎，由于其具有事件驱动、非阻塞 I/O 模型、支持异步回调等特性使其非常适合用于构建快速、可扩展、轻量、高效的数据密集型实时应用。

先说说整个系统的实现方案：

- 前端： **FDEV4/jQuery**
- 后台： **NodeJS**
- Web Framework： **Express**
- Html 渲染模板引擎： **Jade/markdown**
- 数据持久化存储： **MongoDB ***
- 数据库驱动： **MongoSkin**(基于 node-mongodb-native)
- Node 应用持续可用性保证： **Forever**
- 静态资源提供/请求压缩、访客日志记录等： **Nginx**
- 部署环境： **Mac / Ubuntu**

3. 从一个基本的应用开始

这部分的内容需要达到的一个目标是能够利用 Node 和 Express Web Framework 搭建一个最简单的 Web 应用，可以：

1. 提供静态资源路由：加载相应的 js、css；
2. 利用 Jade 模板引擎渲染出一个 Html 页面；
3. 连接并读取 MongoDB 数据库，将查询的数据与模板一起合成相应的 Html 页面，在浏览器中显

示出来；

4. 用户填写相应的表单数据，并将数据提交给相应的 Node Controller 加工处理后存入数据库；

这些是完成一个 web 应用最基本的操作，我们先要完成以上基本功能而后在此基础上进一步的完善和扩展，最终实现更为复杂的系统功能。

3.1 环境准备

工欲善其事，必先利其器。首先要配置好开发环境，这个过程相对来说还是比较简单的：

1. **NodeJS.** 首先安装 NodeJS：可以在 http://nodejs.org/ 上下载最新的安装文件进行安装。这个过程应该会比较顺利的。在 Ubuntu 下安装步骤：

    1. sudo apt-get remove nodejs #卸载旧版本 Node，可选

        (如果通过 source 安装 node 可以直接在 source 目录执行 sudo make uninstall 就可以了)

    2. wget http://nodejs.org/dist/v0.6.14/node-v0.6.14.tar.gz
    3. tar -xzvf node-v0.6.14.tar.gz; cd node-v0.6.14/
    4. ./configure
    5. make
    6. sudo make install

        node –v #可以通过此命令查看是否安装成功及 Node 的版本号

2. **NPM**(Node Package Manager). NPM 默认会随同 NodeJS 一起安装的。如果没有,可以参考这里的说明进行安装 :http://npmjs.org/doc/README.html 。在 Unix like 系统上通过 curl http://npmjs.org/install.sh | sh 命令应该就可以搞定的。

3. **MongoDB.** 由于需要将数据进行持久化存储所以还需要一个数据库。对于后台工程师来说采用 Oracle 或者 Mysql 可能是很自然的想法，不过我个人觉得采用 MongoDB 跟 Nodejs 搭配是非常完美的，这俩简直天生一对儿。MongoDB 天生对 JS 开发人员友好，而且性能、稳定性、扩展性都很好，是一个非常有潜力的 Nosql 解决方案。具体安装可以参考：http://www.mongodb.org/display/DOCS/Quickstart

    在 Ubuntu 下可以采用如下方式安装：

    1. sudo vi /etc/apt/sources.list

        加上这个软件源:

        deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen
    2. sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
    3. sudo apt-get update

sudo apt-cache search mongodb

4. sudo apt-get install mongodb-10gen

安装好后配置文件路径： /etc/mongodb.conf

mongod –version #可以通过此命令检查是否成功及 mongodb 版本号

4. **Express.** NodeJS 虽然很适合构建快速、可扩展的 web 应用，不过单纯利用 NodeJS 来构建 Web 应用还是有很多事情需要做的，重造轮子的成本很大，而且带来的后续可维护性、性能、安全性等都会大打折扣。所以推荐使用 Express web framework 来帮助搭建应用的基本框架。Express 具体使用指南参考：http://expressjs.com/guide.html

Express 的安装：$ npm install express –g

在 express 安装完成后通过简单的命令即可生成一个基本的 web application：

1. **Create The App：** $ express /github/latinode && cd /github/latinode

该步骤会在 latinode 目录下生成如下目录结构：

```
├── app.js            // web 应用启动文件
├── package.json      // Json 格式的应用描述信息：应用名、版本、作者、依赖、license 等
├── public            // 静态资源存放目录
│   ├── images        // 静态图片存放目录
│   ├── javascripts   // 前台 JS 存放路径
│   └── stylesheets   // 样式文件存放路径
│       └── style.css
├── routes            // 后台 Nodejs 代码存放目录
│   └── index.js      // index 首页相关代码
└── views             // 前端 Jade 模板存放路径
    ├── index.jade    // 首页模板对应 Jade 文件
    └── layout.jade   // 公用 layout 对应 jade 模板
```

2. **Install Dependencies:** $ npm install –d #该命令会在 latinode 应用里的 node_modules 里安装 express module。

3. **Start the Server:** $ node app.js

安装好依赖，启动应用后可以访问：http://localhost:3000/ 即可看到页面输出：

Express

Welcome to Express

看到这个信息说明应用启动成功了。还是很顺利哈。

至此我们已经搭建好了一个完整的最基本的 web 应用：可以加载静态资源、可以渲染出 html 页面。是不是很容易？哈哈。不过终究还是缺了点什么？嗯，一个真正的应用没有数据哪行！？所以接下来才正式切入主题——打通数据库。

是的，要让数据流动起来，从数据库经由 MongoSkin 驱动流向 Node Controller，经过必要的加工处理，放入模板引擎，与 Jade 模板合成对应的 html，发送给客户端，在用户浏览器里展现出来。用户输入必要的信息，经过前端 JS 验证，然后由 express 路由给指定的 Node Controller 逻辑，这里可以进行后端验证、权限鉴别、其他处理等然后存入 MongoDB 数据库。因此我们要安装 Mongoskin：

5. **MongoSkin.** 通过 NodeJS 连接 MongoDB 数据库需要相应的驱动。node-mongodb-native 是最强大的 MongoDB 驱动之一，不过这个驱动用起来不太方便，回调太多。Mongoskin 是基于 node-mongodb-native 的，并在其上进一步封装使其更易于使用。封装后的使用语法类似 mongo shell，而且像 node-mongodb-native 一样强大，另外还支持 javascript 方法绑定。可以像使用 MVC Model 一样使用 Mongoskin 实例。

   MongoSkin 安装方法:

   在 node 应用目录/github/latinode 内执行：npm install mongoskin

   这样会在 node 应用目录 node_modules 里面安装 mongoskin 模块。

   其他相关细节可以参考：https://github.com/guileen/node-mongoskin

接下来看看如何来完成对数据库的操作——

3.2 通过 MongoSkin 操作数据库

3.2.1 创建测试数据

为了测试利用 Node 读取 MongoDB 数据是否成功，我们需要先向数据库中插入测试数据，这一步可以通过 mongo shell 来完成：

1. 安装完 mongoDB 后通过 mongod 命令启动数据库(如果在 ubuntu 上通过前文所述方法安装则会自动启动 mongod，可以通过 ps aux|grep mongo 看是否有相应进程)，Mongod 会默认监听来自 27017 端口的数据库连接请求；
2. 通过 mongo 命令启动 mongo shell，Mongo Shell 是一个完整的 Javascript 解释器，我们可以在其中执行各种 js 脚本语句，同时也可以通过该终端对 mongoDB 数据库进行操作；
3. 在数据库中插入数据：
   a) 默认情况下启动 mongo shell 后会连接 test 数据库，可以切换到 latin 数据库：执行 use latin 命令即可.注意一开始虽然没有 latin db，但是当执行 use latin 命令时会根据情况自动创建 latin db。可以通过 db.getName()命令显示当前数据库名。

b) 执行命令 db.latin.insert({'dancerID': '29411', "dancerName" : "M.J."});

c) 该命令会在数据库中创建一个名字为 latin 的 collection（相当于传统数据库的表 table），然后在该 collection 中新建一个 document（相当于传统数据库中的记录 row）。Document 的内容查询出来后为一个 json 对象。不过存储的是 Bson 格式的 document ,即 json-like Document 序列化后对其进行二进制编码存储。

d) 我们可以通过 db.latin.find();命令来验证之前的数据插入是否成功。

说明：MongoDB 不是此处的重点，先完成最基本的操作即可，后面会再进一步加以说明。

3.2.2 通过 Get 请求读取 MongoDB 数据

要想从数据库中读取数据先要通过 MongoSkin 连接数据库。这个过程相对来说还是挺简单的。

1. 由于之前已经为应用安装了 mongoskin 模块 ,我们可以在 routes 目录里面添加一个文件 database.js, 内容如下：

```
var db = exports.db = require('mongoskin').db('localhost:27017/latin');
// 基本 dancer 操作 DAO 接口
var dancerDAO = exports.dancerDAO = {
  /**
   * 根据 dancerID 查询其基本会员信息
   * @param dancerID          待查询的会员的 dancerID
   * @param fn                fn 为执行查询成功后的回调
   */
  findDancerByID: function(dancerID, fn){
    this.findOne({'dancerID':dancerID}, fn);
  }
}
```

2. Collection 操作绑定

使用 MongoSkin 的一个便利之处就是可以将 Collection 和其对应的操作进行绑定。这样一来 DAO 接口的逻辑就可以复用，不必重复编写相同的数据库读写代码逻辑，维护起来也容易很多。由于数据库操作是后面很多逻辑的基础，所以这个绑定操作也应当在读写数据库之前进行，放在 app.js 里面是比较合适的，可以在 app.js 中加入如下代码：

```
var    db   = require("./routes/database.js").db,
dancerOp    = require("./routes/database.js").dancerDAO;
```

在应用 configure（app.configure()）之后，Routes 初始化代码：

```
app.get('/', routes.index);
```

之前加入 Collection 绑定逻辑：

```
db.bind("latin", dancerOp);
```

进行如上绑定后就可以在其他 routes 逻辑中通过：db.latin.findDancerByID('29411',function(){});来查询数据了。

3. Node index 逻辑中读取数据

将 index.js 中代码改成如下：

```
var db = require("./database.js").db;
exports.index = function(req, res){
   db.latin.findDancerByID('29411', function(err, result) {

      if (err) throw err;
      if (!!result){
         res.render('index',
          {    title:    'Express',
             dancer: result
          });
      }
   });
};
```

通过以上逻辑将会从数据库中查询满足 dancerID 为 29411 的数据，然后将其放入 result 变量里面。

4. 模板数据合成展示：

修改 index 对应的模板文件：index.jade，添加如下内容：

```
h1= title
p Welcome to #{title}
p dancerID: #{dancer.dancerID}
p dancerName: #{dancer.dancerName}
```

Jade 模板引擎会用 title 和 dancer 中的数据替换 index.jade 中的相应变量，合成最终将在浏览器中展示的 html，最后合成出来的 html 代码如下：

```
<body><h1>Express</h1><p>Welcome to Express</p><p>dancerID:
29411</p><p>dancerName: MJ</p></body>
```

这样一来我们就将数据从数据库中取出来进行展示了。生成出来的 html 是去除空格后的压缩代码，阅读起来不太方便，不过倒是相当节约带宽的。注意 Jade 模板语言的具体详情现在不做描述，后面会进一步说明。

现在我们已经可以从数据库中读取数据并显示了，不过这里的逻辑很简单，而且不足的是 dancerID 也是写死的，一种很自然的想法是通过"/user/29411"…"/user/29555"等 URL 来访问对应的会员信息。这

个需要用到 express 的 Routing 功能。稍后再说。

接下来说说如何将用户提交的表单数据保存到数据库。

3.2.3 通过 Post 请求将数据存入 MongoDB

1. 首先需要一个提交数据的表单，可以直接对 index.jade 进行修改如下：

```
h1= title
p Welcome to #{title}
p dancerID: #{dancer.dancerID}
p dancerName: #{dancer.dancerName}
form(name="updateForm", id="updateForm", method="post", action="/update")
  table.apply-table-a
    tbody
      tr.em
        th
          label 工 号：
        td
          input#dancerID.comm-input(type="text", name="dancerID" )
      tr
        th
          label 姓 名：
        td
          input#dancerName.comm-input(type="text", name="dancerName")
  button(class="comm-button", id="update-btn") 提 交
```

注意：空格缩进对于 Jade 模板很重要，缩进代表了 DOM 结构的层级关系。子元素的缩进要深一级。

2. 接下来我们需要修改 database.js，在其中添加一个方法:

```
updateDancerByID: function(dancerID, dancerName, fn){
  this.update({dancerID:dancerID}, {dancerName:dancerName}, {upsert:true}, fn);
}
```

在 mongoDB 语法中有一种特殊的更新叫 upsert，该操作如果没有找到满足条件的文档就会以该条件和更新的文档为基础创建一个新的文档，如果找到了满足条件的文档就会正常更新。如此同一套代码既可以创建又可以更新文档。Update 的第三个参数为 true 表示这是一个 upsert 操作。注意在 upsert 参数传递上 mongoskin 与 mongoshell 有些差别。

3. 添加用户提交表单数据处理逻辑：

由于在前面 index.jade 里面我们的表单提交到的 action 是 update，所以我们可以在

routes/index.js 中添加该数据保存逻辑，新增代码如下：

```
exports.update = function(req, res){
  var dancerID    = req.body.dancerID,
    dancerName    = req.body.dancerName;   // 表单提交的数据可以通过 req.body 取得
  // 简单起见，此处忽略表单数据校验逻辑
  db.latin.updateDancerByID(dancerID, dancerName, function(err) {
    if (err) throw err;
    console.log('Dancer Updated With ID:',dancerID,' DancerName:', dancerName);
    // 表单提交成功后返回首页
    res.redirect('back');
  });
};
```

4. 在 app.js 里配置 post 请求的路由信息，修改 app.js 在 get 请求下添加新的路由如下：

```
// Routes
app.get('/', routes.index);
 app.post('/update', routes.update);
```

5. 重启应用，现在再来试试提交表单，数据应该可以成功保存了。可以通过 mongo shell 执行查询：

```
db.latin.find({dancerID:'29411'});
```

来查看表单提交操作执行后数据的变化情况。

从 updateDancerByID 方法我们不难发现目前只能修改用户 dancerName 信息，而 dancerID 是不能修改的，除非该会员不存在，则执行新增操作。

到此为止我们已经完成了 mongodb 数据的读取和保存。主干流程算是通了。不过一个实际的应用要远比这个复杂的多 ,好在基本原理上是相同的。而且迄今为止我们所掌握的 MongoDB、MongoSkin、Jade、express 技能等都是九牛一毛。NodeJS 的水还是很深的，而且涉及到的周边模块也很多，为了完成一个更为复杂的"过得去"的应用我们还需要了解更多……。

4. 拉丁培训管理系统需求概述

这个培训管理系统的功能远比上面的例子要复杂，细节上要考虑的也相当多。不过使用 NodeJS 开发的好处是前端、后端都用 JS，而且接口可以自己定义，沟通成本很小，同一个功能实现的方式可能很多，全在自己发挥。

先说说这个系统的基本功能：

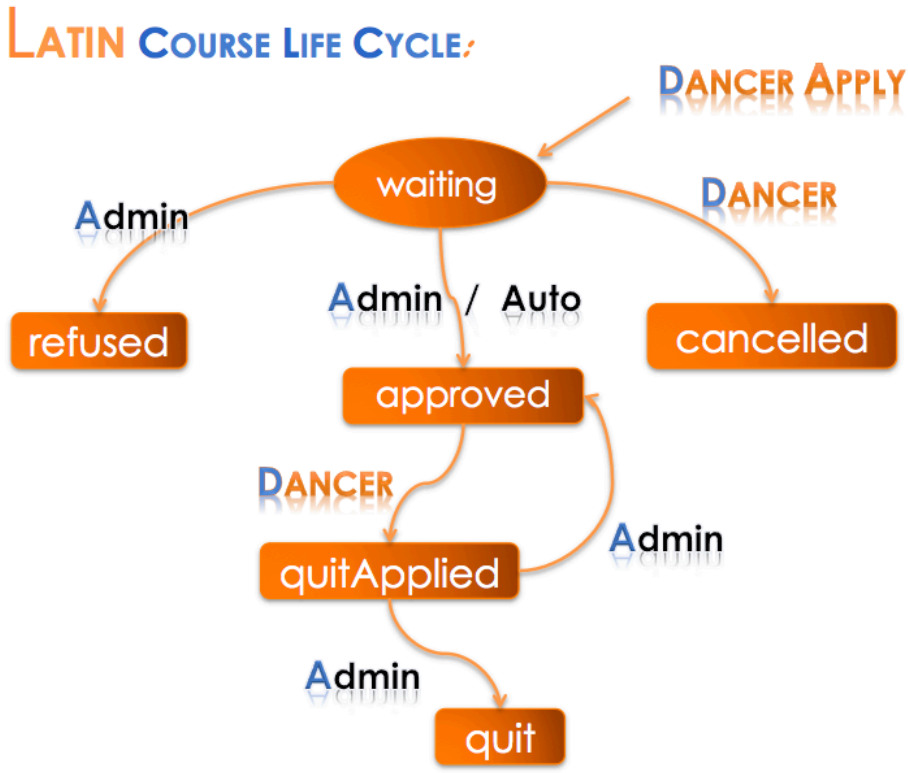
目前已经实现的功能大体上分两部分——用户前台和管理员后台。

4.1 用户前台功能描述

1. 实现**网上报名。**
   a) 新用户：填写基本信息（姓名、工号、旺旺、支付宝账号、分机、公司邮箱等）；选择培训课程；提交报名；系统存入数据库。
   b) 老用户：根据用户填写的工号**自动获取显示其个人信息**；然后选择相应课程提交报名；也可以修改个人信息。
2. 用户可以查看个人信息及报名情况等。
3. 报名页显示**当前开设课程实时信息**（舞种、额定人数，已申请人数、报名成功人数等）。
4. 新开课程配置简化 :开设新培训课程时代码逻辑不需要变更 ,只需要更改配置即可迅速满足要求。
5. 申请报名后需管理员手工审核报名是否通过，也可以根据一定规则由系统进行自动审核。
6. 管理员审核通过前用户可取消课程报名，审核通过后用户可申请退课。
7. 退课需管理员审核：可以拒绝退课；或者线下退费、退课。
8. 用户可以**查询报名情况：**根据工号、部门、性别、课程、报名状态、缴费状态等进行组合查询。
9. 查询结果分页展示，点击表格头部时可以对某些字段进行排序。

4.2 管理员后台功能描述

- **课程生命周期管理：**



1. 新申请报名课程状态为待审核----------(waiting)；
2. 管理员审核前用户可以自助取消----------(cancelled)；
3. 管理员可以审核拒绝----------(refused)；
4. 或者审核通过则报名成功----------(approved)；
5. 报名成功后会员可以申请退课----------(quitApplied)；
6. 退课需要管理员审核，审核拒绝则回到报名成功----------(approved)；
7. 退课审核通过则处于退课成功----------(quit)
8. 对于课程生命周期状态转换系统会验证其是否满足前置条件 :比如 waiting 状态不能直接转为 quit，但是可以转换为 cancelled，只有 waiting 或者 quiteApplied 状态的可以转换为 approved，而 approved 不能转为 waiting ,只有 quitApplied 的课程可以转为 quit，并且需要先退费等。

- **课程缴费状态设置：**

为了简化逻辑，课程的缴费状态跟其生命周期是分开的：

i. 新报名的会员其缴费状态为未缴费。

ii. 报名成功的会员在缴费后管理员可以将其设为已缴费。

iii. 申请退课的会员管理员可以线下退费并修改其状态为未缴费，然后退课。

- **管理员修改会员信息：**

  会员可以修改自身基本信息，管理员可以修改其基本及高级属性比如：level、vip 等级、blacklist 状态、lock 状态等。

5. 系统的具体设计实现

5.1 Document 模型设计

首先说说会员模型设计。如果用传统的 Mysql 数据库来实现系统所需功能，由于会员和课程之间是多对多的关系，所以除了需要建会员表、课程表之外还需要增加一个培训表，把会员和课程关联起来，如此以来系统就相对复杂化了，而且为了保证操作的原子性可能还需要引入事务。

采用 MongoDB 后就完全不同了，由于 MongoDB 是 schema free 的 Nosql 数据库，在 MongoDB 中集合(collection)相当于传统数据库的表(table)，文档(document)相当于传统数据库的记录(row)，而且文档里面可以嵌套文档。这样以来可以在会员文档里面嵌入其参加的舞蹈课程信息文档，如此一个文档就能很自然地表达出会员和课程的关系了。具体文档模型如下：

dancer:

```
{
  _id : ObjectId("4f5c6c80359dda5b98000034"),   // mongodb 自动生成的
  dancerID : "29411",                           // 工号，唯一
  dancerName : "MJ",                            // 姓名
  department : "tech",                          // 部门
  email : "latin@alibaba-inc.com",              // 公司邮箱
  extNumber : "76211",                          // 分机号码
  gender : "male",                              // 性别
  wangWang : "hustcer",                         // 旺旺号码
  alipayID : "hustcer@gmail.com",               // 支付宝账号
  level:5,                                      // 舞蹈水平 9 is the highest level.
  vip:5,                                        // 根据上课积极程度以及参加演出次数确定,最高为 5
  performance:[],                               // 参加的演出信息
  ps:"",                                        // 个人备注、个性签名之类
  blacklist:true,                               // 加入黑名单的会员不能报名参加培训
  locked:true,                                  // 个人信息锁定后不能修改
  dancerDNA: 110101,                            // 以后可以考虑利用二进制标志位来定义用户是否具备某些属性
  password:password,                            // 用户密码，暂不支持
  courses : [
    {
      courseVal : "13RI",          // 前面的数字代表期次，接下来的字母表示舞种，尾字母表示课程等级
      gmtPayChanged :    ISODate("2012-03-11T21:44:54.017Z"),   // 缴费状态改变时间
      gmtStatusChanged : ISODate("2012-03-17T06:23:17.858Z"),   // 课程状态改变时间
      applyTime :        ISODate("2012-03-17T06:23:17.858Z"),   // 课程申请时间
```

```
            paid :      false,          // 如果已缴费则为缴费金额
            status :    "approved"      // status: waiting, cancelled, approved,refused,quitApplied, quit
        },
        {
            courseVal :     "13CE",
            gmtPayChanged :         ISODate("2012-03-17T06:23:53.124Z"),
            gmtStatusChanged :      ISODate("2012-03-18T08:00:29.880Z"),
            applyTime :             ISODate("2012-03-17T06:23:17.858Z"),        // 课程申请时间
            paid :          true,
            status :        "approved"
        }
    ],
    gmtCreated : ISODate("2012-03-11T09:12:32.878Z"),
    gmtModified : ISODate("2012-03-18T08:00:29.880Z")
}
```

需要说明的是：

1. courses 是一个数组，而数组里面的每一个元素又是一个内嵌文档，表示会员参与的舞蹈培训信息。
2. courseVal 的命名遵循特定规则 :比如"13RI"其中 13 代表培训的期次 ;R 代表舞种为伦巴(Rumba)，其他还有 C(恰恰，ChaCha)、S(桑巴、Samba)、J(牛仔，Jive)、P(斗牛，Paso doble)；I 代表课程等级为中级(intermediate), E 代表基础班：Elementary，A 代表高级班 Advanced；
3. 会员文档会有创建和修改时间属性，课程会有缴费状态变更时间、课程状态变化时间和报名申请时间属性，以跟进信息变更。

在建立以上文档模型后，由于用户和管理员的所有操作最终都要反映到数据库的数据变化上，所以我们还需要掌握基本的 MongoDB 数据库操作技能。

5.2 MongoDB 基础

前面已经说过，在 MongoDB 中集合(collection)相当于传统数据库的表(table)，文档(document)相当于传统数据库的记录(row)，而且文档里面可以嵌套文档。所以为了完成一个典型的 web 应用我们需要掌握基本的 MongoDB 文档增删改查技巧。

从实用角度出发，本文以拉丁培训管理系统中用到过的基本操作为例进行说明。由于应用本身也是用 Node 开发的，所以每一种操作会以原生 mongo shell 命令及 Node + mongoskin 接口调用方式给出。

项目 gitHub 地址: https://github.com/hustcer/latinode

5.2.1 Mongo shell 基本使用

MongoDB 安装完毕之后通过 ***mongod*** 命令即可启动，默认会监听来自 27017 端口的请求。同时还可以在浏览器里面通过 http://localhost:28017/ 监视数据库运行状态。

Mongo shell 命令可以在 mongod 启动后，通过 ***mongo*** 命令连接数据库，然后在其中输入对应的命令即可执行。Mongo shell 可以执行基本的 js 语句，同时可以对数据库进行操作。如果有数据库操作命令记不得了，最简单的办法就是从 ***help*** 命令入手。

help 指令会列出 shell 支持的一些操作命令；根据 ***help*** 的输出提示可以顺藤摸瓜有所发现——

db.help()可用于获得数据库 db 操作的可用命令；

db.mycoll.help()可以用于获得 collection 支持的操作命令，当然'mycoll'可以被换成其他任意的 collection 的名字。

一句话：mongodb 的帮助文档还是很齐全的，所以如果想在 mongodb 上完成一些类似 sql 数据库上的功能的话可以 ***help*** 下，然后根据输出提示总是能找到些线索的。

启动 mongo shell 后默认会链接 test 数据库；

可以通过 ***show dbs*** 命令查询所有可用数据库；

然后用 ***use*** 命令切换到对应的库，比如 '***use dance***'可以切换到 dance 数据库；

然后 ***show collections*** 查询该数据库里的 collection 名，相当于传统数据库的表名；

最后可以通过 ***db.collectionName*** 引用对应的 collection 并对其进行操作，比如：***db.latin.count();*** 命令可以用于查询当前数据库(已经切换到 dance)下 latin collection 里面所有文档(document，相当于传统数据库中的 row)数目。

而通过 mongoskin 连接数据库可以通过如下方式实现：

```
var dbMongo    = exports.db = require('mongoskin').db('localhost:27017/dance');
```

前提是你已经安装好 mongoskin 模块。该操作将连接本地 27017 端口的 dance 数据库。

5.2.2 Mongodb 基本文档操作

下面就来说说如何通过 mongo shell(以下简称 shell)和 Node + mongoskin 驱动(以下简称 node)实现对 document 的基本增删改查操作，我们先假设您已经根据前面的操作切换到 dance 数据库，并且计划对 latin collection 进行操作。我们可以对前面的文档模型进行简化为：{'dancerID': '', "dancerName" : "", level:5} 则：

Shell 插入文档：

```
db.latin.insert({'dancerID': '29411', "dancerName" : "M.J.01", level:5})
db.latin.insert({'dancerID': '29455', "dancerName" : "M.J.02", level:5})
```

Node 插入文档：

```
db.collection('latin').insert({'dancerID': '29466', 'dancerName' : 'M.J.01', level:5}, function(err, result) {
  if (err) throw err;
  if (result) console.log('M.J.01 Added!');
});
```

注：db.collection('latin')用于获得当前 db（此处为 dance）内的 latin collection， insert 传入的第一个参数为待插入的文档对象，后面一个参数为文档插入成功后执行的回调函数。

Shell 删除文档：

```
// 删除数据库中 dancerID 为 29477 的文档，如果不止一个则所有满足条件的都会被删除
> db.latin.remove({dancerID:'29477'})
```

Node 删除文档：

为了简化操作我们可以把 db.collection('latin') 赋值给 db.latin，以后就可以通过 db.latin 进行调用了（下同）：

```
db.latin = db.collection('latin');
db.latin.remove({dancerID:'29477'}, function(err) {
  if (!err) console.log('Dancer deleted!');
});
```

Shell 查询文档：

```
// 从数据库中查询所有 dancerName 为 MJ,level 大于等于 5 的文档，如果有多条则全部显示
> db.latin.find({dancerName:'MJ',level:{$gte:5}})

// NOTICE: 比较筛选除了有 $gte 外还有 $lt, $gt, $lte 分别相当于：>=,<,>,<=。

// 从数据库中查询一条 dancerID 为 29411 的文档，如果有多条只显示一条
> db.latin.findOne({dancerID:'29411'})

// 如果只想返回特定字段的查询结果，可以在第二个参数里将相应的字段的 value 设为 1，

// 如果不想取出该字段也可以将其设为 0(_id 是默认被返回的，可以显示设置为 0 则不返回)，如下：
> db.latin.findOne({dancerID:'29411'},{dancerName:1,_id:0});
```

Node 查询文档：

```
// 如果 dancerName 为 'MJ'的文档不止一个，可以将其转换为一个数组
db.latin.find({dancerName: 'MJ',level:{$gte:5}}).toArray(function(err, result) {
  if (err) throw err;
  console.log(result[0]);
});
// findOne 会确保只返回一条满足条件的数据
db.latin.findOne({dancerID: '29411'}, function(err, result) {
  if (err) throw err;
  console.log(result);
});
```

Shell 统计满足条件的文档数目：

```
// 查询数据库 latin 集合中所有文档的数目
> db.latin.count()
// 查询 latin 集合中 level 大于 5 的所有文档的数目
> db.latin.count({level: {$gt:5}})
```

Node 统计满足条件的文档数目：

```
db.latin.count(function(err, count) {
  console.log('There are ' + count + 'dancer in the database');
});
db.latin.count({level:{$gt:5}}, function(err, count) {
  console.log(count + ' dancer whose level is greater than 5');
});
```

Shell 修改文档：

```
// 更新数据库中 dancerID 为 29477 的文档，将其 dancerName 设置为 MJ07
> db.latin.update({dancerID:'29477'}, {$set:{ dancerName: 'MJ07'}})
```

Node 修改文档：

```
db.latin.update({dancerID:'29477'}, {$set:{ dancerName: 'MJ07'}}, function(err) {
  if (!err) console.log('Level updated!');
});
```

mongodb 中对于文档的修改有多种形式，这里例举的只是其中最简单的一种，后面会有其他的修改方法示例。总得来说 mongoskin 的 api 很直观、自然，跟 mongo shell 的操作命令很类似。基本上可以举一反三的，可以先猜测，如果不对再查查 mongoskin 的 api。

5.2.3 Mongodb 文档内嵌数组操作

Mongodb 文档的 value 有可能是一个数组，对于数组 mongodb 也有一些特殊的操作方法，不过整体来说大多数情况下可以将数组中的每一个元素分别当做全局 key 的 value 来处理，假设文档模型为：

```
{'dancerID':'29411','dancerName':'MJ', 'courses':['c1', 'c2', 'c3'] }
```

可以将其视为：

```
{'dancerID':'29411','dancerName':'MJ', 'courses':'c1', 'courses':'c2', 'courses':'c3' }
```

所以如果要查询选修了 c1 课程的学员可以通过如下方式：

```
> db.latin.find({courses: 'c1'});
```

内嵌数组查询：

现在 collection 中插入一些测试数据：

```
> db.latin.insert({"dancerID":"29711","dancerName":"MJ", "courses":["c1", "c2", "c3"] });
> db.latin.insert({"dancerID":"29511","dancerName":"M0", "courses":[ "c2", "c3"] });
> db.latin.insert({"dancerID":"29611","dancerName":"M1", "courses":["c1", "c3"] });
```

此时如果需要查询所有参加了 c1 和 c3 课程的同学不能通过：

```
> db.latin.find({ 'courses':['c1', 'c3'] });
```

因为这时已经指定了 courses 的具体值，属于精确匹配，只能查出一条文档即 dancerID:29611 的，第一条文档没有命中，正确的方式应该是：

Shell 查询数组：

```
> db.latin.find({courses:{$all:["c1", "c3"]}})
```

此时会命中两条文档。

Node 查询数组：

```
db.latin = db.collection('latin');
db.latin.find({courses:{$all:["c1", "c3"]}}, function(err, result) {
  if (err) throw err;
  console.log(result);
});
```

上面的查询假定数组是无序的，也就是说：{ "courses":["c1", "c3"] }和{ "courses":["c3", "c1"] }都满足条件，如果想查询数组特定位置的元素是否为指定值，比如查询参与第二期培训，且课程为 c2 的会员可以通过如下 ***key.index*** 方式查询：

```
> db.latin.find({'courses.1':'c2' });
```

此时会命中第一条文档，但是不会命中第二条文档。Index 的值从 0 开始。

数组子集查询：

前面说过通过设定查询操作的第二个参数的各个 key 的 value 为 1 或者 0 来控制是否返回对应字段，对于数组查询还可以在第二个参数中通过 $slice 设定需要返回的数组 value 的子集，比如：

Shell 查询数组子集：

```
// 查询所有会员返回其对应 dancerID 及所参与的前两门课程。
> db.latin.find({},{dancerID:1,_id:0,courses:{$slice:2}});
// 注 :如果{$slice:-2}则返回后两门课程 ,如果{$slice:[10,20]} 则跳过前 10 个数组元素 ,返回接下来的 20 个元素。
```

Node 查询数组子集：

```
// 查询所有会员返回其对应 dancerID 及所参与的前两门课程。
db.latin.find({},{dancerID:1,_id:0,courses:{$slice:2}}, function(err, result) {
  if (err) throw err;
  console.log(result);
});
```

内嵌数组修改：

可以用$push 为数组末尾增添一个元素， 如果对应的数组的 key 不存在的话则创建该 key 并加入这个元素。比如对于新插入的文档：

```
> db.latin.insert({'dancerID': '667788', "dancerName" : "latino"});
```

如果我们需要为其增加 course：c1,可以这样(假设 dancerID 为 667788 的会员只有一个)：

```
> db.latin.update({dancerID:'667788'}, {$push:{courses:'c1'}});
```

不过$push 存在一个问题：他不会检查元素在数组中是否存在，这意味着如果你多次 push 同一个元素那么数组中就有可能存在多个重复的元素，这可能并不是你所希望的，在这种情况下可以用 $ne 检查下，如果不存在再 push，如下：

```
> db.latin.update({dancerID:'667788', courses:{$ne:'c1'}}, {$push:{courses:'c1'}});
```

对于这种情况先要判断，不存在了再 push 比较麻烦，实际上可以通过 $addToSet 来取代，如下：

```
> db.latin.update({dancerID:'667788'}, {$addToSet:{courses:'c1'}});
```

这样以来系统会自动检查会员是否已经参与了该课程，如果没有则加入，反之不做修改。

假如会员同时报名了好几个课程：c1/c2/c3，分三次更新固然可以达到目的，不过效率会下降很多，这时可以将$addToSet 与 $each 配合起来使用：

```
> db.latin.update({dancerID:'667788'}, {$addToSet:{courses:{$each:['c1','c2','c3']}}});
```

如果需要删除会员课程有几种方法：

{$pop : {key : 1}}可以从 courses 尾部删除一个课程：

```
db.latin.update({dancerID:'667788'}, {$pop:{courses:1}});
```

{$pop : {key : -1}} 可以从课程头部删除一个课程记录：

```
db.latin.update({dancerID:'667788'}, {$pop:{courses:-1}});
```

或者课程的具体位置不清楚，但是课程值知道的话可以用 $pull 比如：

```
> db.latin.update({dancerID:'667788'},{$pull:{courses:'c1'}});
```

可以删除会员 667788 的 c1 课程，如果该课程存在的话。

也可以对数组指定位置的元素进行修改：比如对于 { "courses":["c1", "c3","c4"] }将其修改为 { "courses":["c1", "c2","c4"] }可以采用如下方案：

```
> db.latin.update({dancerID:'667788'}, {$set:{'courses.1':'c2'}});
```

其中".1"代表要修改的是课程数组中的 index 为 1 的元素。不过问题是我们并不总是可以知道要修改的元素的 index 是多少，所以我需要查询下，这个时候可以用位置操作符："$" 来表示匹配元素的 index。如下：

```
// 需要注意的是如果有多条满足条件的记录只会更新最先匹配上的一条，而不是所有满足条件的记录
> db.latin.update({dancerID:'667788', courses:'c3'},{$set:{'courses.$':'c2'}});
```

5.2.4 MongoDB 文档内嵌文档操作

以上数组操作示例中，数组元素为简单的字符串类型，在实际应用场景中可能要复杂得多——有可能数组的每一个元素都是一个文档。Mongodb 内嵌文档操作比较麻烦，尤其是当数组里内嵌文档的时候，复杂度更增加了不少，下面以舞蹈培训管理系统的实际模型为例进行说明：

假如新的文档模型为：

```
{
   "dancerID" : "29411",
   "dancerName" : "MJ",
   "wangWang" : "hustcer",
   "courses" : [
      {
         "courseVal" : "13CI",
```

```
            "paid" : true,
            "status" : "approved"
        },
        {
            "courseVal" : "13SE",
            "paid" : true,
            "status" : "waiting"
        }
    ]
}
```

课程报名：

对于新报名的会员数据库中没有该会员对应的课程记录，这个时候可以直接给课程赋值为相应的值，然后插入数据库，比如：

```
var dancerModel = {};
dancerModel.dancerID = '29411';
dancerModel.dancerName = 'MJ';
dancerModel.wangWang = 'hustcer';
dancerModel.couses.push({
        'courseVal' : '13CI',
        'paid' : true,
        'status' : 'approved'
    });
dancerModel.couses.push({
        'courseVal' : '13SE',
        'paid' : true,
        'status' : 'waiting'
    });
db.latin.insert(dancerModel);
```

对于数据库中已经存在的会员，可以先查出其相关信息，然后对其进行修改最后保存，如下：

```
var dancer = db.latin.find({dancerID: '29411'});
dancer.dancerName = 'Mongod';
// 遍历 dancer.courses 数组，如果不存在对应的课程则加入该课程，然后保存
.....
db.latin.save(dancer);
```

“遍历 dancer.courses 数组，如果不存在对应的课程则加入该课程”这个需求可能会让你想到前面提到过的 $ne + $push ,或者 $addToSet。不过实际上这两种操作都不适用于当前的场景因为这两种操作都要求文档是精确匹配的，也就是说：{"courseVal" : "13SE","paid" : true,"status" : "waiting"}和{"courseVal" : "13SE","paid" : false,"status" : "waiting"}是两个不同的课程，而实际上是同一课程，只不过处在不同的状态罢了：是否缴费过。

所以在这种情况下我们只能采用看似老土却很有效的 for 循环来遍历数组：只要对应 courseVal 为 13SE 的课程存在就可以确定会员曾经报名过该课程，数据库中有对应的记录，顶多是状态的差别而已。所以上述给已有会员增加课程的操作应该类似这样的：

```
.....................
    var courseExist = false;
    for(int i = 0, l = dancer.courses.length; i < l; i ++){
        if(dancer.courses[i].courseVal==="13SE"){
            courseExist = true;
            break;  // 课程已经存在则不再增加
        }
    }
    if(!courseExist){
    dancerModel.couses.push({
          "courseVal" : "13SE",
          "paid" : false,
          "status" : "waiting"
        });
    }
```

修改课程的报名状态

由于课程为会员 courses 数组的内嵌文档，显然这个数组里面可能会有很多课程，所以修改指定会员的特定课程到某一个状态需要三个参数：会员唯一性标记(dancerID)、课程对应 value、修改后的状态，当然还可以添加一个可选参数：修改后执行的回调函数，最终对应的 node 代码如下：

```
    /**
    * 设置会员课程状态
    * @param dancerID     待设置的会员的 dancerID
    * @param courseValue     待设置的课程的值
    * @param status        课程新的状态。
    */
    updateDancerCourseStatus: function(dancerID, courseValue, status, fn){
       this.update({'dancerID':dancerID.toUpperCase(), 'courses.courseVal':courseValue}, {  $set:
          {'courses.$.status':status, 'courses.$.gmtStatusChanged':new Date(), 'gmtModified': new Date()}
       }, fn);
    }
```

以上代码表示查询ID为dancerID的会员，对于其课程值为courseValue的课程进行修改，修改其状态为status，同时更新其修改时间等属性。

修改课程的缴费状态

修改课程缴费状态跟上述修改课程报名状态类似，逻辑如下：

```
/**
   * 设置会员缴费状态
   * @param dancerID          待设置的会员的 dancerID
```

```
 * @param courseValue     待设置的课程的值
 * @param isPaid          会员是否缴费，缴费为 true，反之为 false
 */
updateDancerPayStatus: function(dancerID, courseValue, isPaid, fn){
    this.update({'dancerID':dancerID.toUpperCase(), 'courses.courseVal':courseValue}, {  $set:
        {'courses.$.paid':isPaid, 'courses.$.gmtPayChanged':new Date(), 'gmtModified': new Date()}
    }, fn);
}
```

会员筛选

会员条件筛选的操作比较复杂，工号、性别、部门，报名课程及课程的缴费状态、报名状态等都可以进行组合查询，而课程相关条件是要对内嵌文档进行查询的，可以参考如下方式实现（req.body 里面包含用户后台表单提交过来的查询条件）：

```
/*
* 会员列表筛选/搜索接口.
*/
exports.search = function(req, res){
    var col        = db.collection('latin');
    var dancerModel = {};
    // 根据课程状态，是否缴费来进行查询
    if (!!req.body.dancerID)    {dancerModel.dancerID    = req.body.dancerID;};
    if (!!req.body.gender)      {dancerModel.gender      = req.body.gender;};
    if (!!req.body.department)  {dancerModel.department = req.body.department;};

    // 内嵌文档精确匹配
    dancerModel.courses = {};
    dancerModel.courses.$elemMatch = {};
    if (!!req.body.course)
        dancerModel.courses.$elemMatch.courseVal = req.body.course;
    if (!!req.body.status)
        dancerModel.courses.$elemMatch.status     = req.body.status;
    if (!!req.body.paid)
        dancerModel.courses.$elemMatch.paid      = JSON.parse(req.body.paid);
    // FIXME: 如果课程没有任何匹配条件就把该条件完全去掉,这种方法挺猥琐的感觉，以后可以改进下
    if (JSON.stringify(dancerModel.courses.$elemMatch) === '{}')
        delete dancerModel.courses;
    col.findDancerByCondition(dancerModel, function(err, result) {
        if (err) throw err;
        res.contentType('application/json');
        res.send({data:result});
    });
};
……省略部分代码……
/**
```

```
 * 根据条件查询其基本会员信息，不含创建，修改时间，_id 等
 * @param condition     Json 对象，待查询的会员所满足的条件
 *                  eg. {dancerID:'29411', 'courses.courseVal':'13R', 'courses.paid':true}
 */
findDancerByCondition: function(condition, fn){
  if (!!condition.dancerID) { condition.dancerID = condition.dancerID.toUpperCase();};
  db.collection('latin').find(condition, {gmtCreated:0, gmtModified:0, _id:0}).toArray(fn);
}
```

5.2.5 Mongoskin MVC Helper

文档增删查改的操作比较普遍，如果在每一个需要进行此操作的地方都写上类似的逻辑维护起来简直是个噩梦，所以很自然的想法就是像后台那样：把数据库操作相关逻辑单独地抽取出来成为一个 DAO 层，其它 service 层调用 DAO 层提供的接口就可以了。

在 Node 里面我们也可以用类似的思路来封装对数据库增删改查的操作，只提供相应的对外接口以减少逻辑冗余，提高可维护性。在 mongoskin 里面可以将 collection 上所有支持的操作与 collection 本身进行绑定，这样一来 collection 好比一个 java 类，包含一系列的数据，同时对外提供了对这些数据进行操作的方法。

利用 Mongoskin 对数据库操作进行封装很简单：

1. 首先要定义某 collection 对外提供的方法；

2. 将这些方法与 collection 进行绑定。

这样一来在 control 业务逻辑层就可以通过 collection 实例来调用绑定在其上的方法，从而达到逻辑复用。举例如下：

可以在 database/dancer.js 里面定义：

```
exports.commonDancerOp = {
  /**
   * 根据条件查询其基本会员信息，不含创建，修改时间，_id 等
   * @param condition     Json 对象，待查询的会员所满足的条件
   *                  eg. {dancerID:'29411', 'courses.courseVal':'13R', 'courses.paid':true}
   */
  findDancerByCondition: function(condition, fn){
    if (!!condition.dancerID) { condition.dancerID = condition.dancerID.toUpperCase();};
    this.find(condition, {gmtCreated:0, gmtModified:0, _id:0}).toArray(fn);
  },
  /**
   * 查询满足指定条件的会员数目
   * @param condition     Json 对象，待查询的会员所满足的条件
   */
  countDancerByCondition: function(condition, fn){
```

```
        if (!!condition.dancerID) { condition.dancerID = condition.dancerID.toUpperCase();};
        this.count(condition, fn);
    },
    /**
     * 根据 dancerID 查询其基本会员信息，不含创建，修改时间，_id 等
     * @param dancerID        待查询的会员的 dancerID
     */
    findDancerByID: function(dancerID, fn){
        this.findOne({'dancerID':dancerID.toUpperCase()}, {gmtCreated:0, gmtModified:0, _id:0, vip:0, level:
0}, fn);
    },
    /**
     * 设置会员缴费状态
     * @param dancerID          待设置的会员的 dancerID
     * @param courseValue       待设置的课程的值
     * @param isPaid            会员是否缴费，缴费为 true，反之为 false
     */
    updateDancerPayStatus: function(dancerID, courseValue, isPaid, fn){

        this.update({'dancerID':dancerID.toUpperCase(), 'courses.courseVal':courseValue}, {  $set:
            {'courses.$.paid':isPaid, 'courses.$.gmtPayChanged':new Date(), 'gmtModified': new Date()}
        }, fn);
    }
};
```

然后在 node 应用启动逻辑里面将以上数据库操作方法与相应的 collection 进行绑定 ,在 app.js 里面加入类似如下逻辑：

```
var     db              = require('mongoskin').db('localhost:27017/dance'),
        dancerOp        = require("./database/dancer.js").commonDancerOp;
db.bind('latin' , dancerOp);
```

随后在 control 里面取得 collection 示例即可调用所绑定的接口，比如在 index.js 里面这样调用：

```
db.collection('latin'). findDancerByCondition(....);
db.collection('latin'). findDancerByID (....);
db.collection('latin'). updateDancerPayStatus (....);
.....................
```

如此以来即可达到把数据库持久化相关逻辑进行抽象、公用的目的，维护起来也更方便。

5.2.6 MongoDB 访问权限控制

通过 mongo shell 设置数据库访问权限

通过 ***mongod*** 命令以默认配置启动数据库服务是不会对数据库进行访问权限控制的，这显然太不安全，这意味着任何一个知道数据库地址的人都可以访问数据，所以最好给数据库设置访问权限，mongodb 的权

限设置比较简单：

1) 先以默认配置启动数据库，此时无权限控制，然后通过 mongo shell 切换到 admin 数据库添加用户名和密码：

```
> use admin;
> db.addUser('admin','123456');
```

2) 安全关闭数据库服务，退出 shell

```
> db.shutdownServer();
> exit
```

3) 以—auth 模式重启数据库服务：

```
$ sudo mongod --auth --dbpath /mnt/db/
```

4) 此时对 admin 数据库设置的用户名和密码已经生效，可以通过两种方式登录该数据库：

i) 先启动 mongo shell 然后授权：

```
$ mongo localhost:27107
> use admin
> db.auth('admin','123456');
```

ii) 在 shell 终端直接启动 mongo 同时授权：

```
$ mongo localhost:27107/admin -u admin -p 123456
```

5) 同理如果需要对其他数据库添加用户名和密码可以按如下步骤：

```
> use dance
> db.addUser('admin', '123456');
```

此时已经给 dance 数据库设置好了用户名和密码，不过前提是你已经完成了第 4 步的 admin 的授权。还有一些系统级的命令比如: show dbs 的执行也需要先获得 admin 的授权，否则会报类似于：'uncaught exception: listDatabases failed:{ "errmsg" : "need to login", "ok" : 0 }'的错误。

* 注意：此处的用户名和密码设置对于 ***mongodump*** 和 ***mongoexport*** 命令同样生效。

通过 mongoskin 链接数据库并进行权限验证

设置好数据库用户名和密码之后通过 mongoskin 建立数据库链接也需要做相应调整：

```
exports.db = require('mongoskin').db('mongo://admin:123456@localhost:27017/dance');
```

不过你的项目可能是通过 github 之类的进行代码管理的，这样也会有些问题：

1. 如果直接将这样的代码传到 github 上任何人都可以看到数据库密码的，也就没什么意义了；

2. 当然你可以写个空的或者假的用户名、密码提交上去，然后代码 pull 下来后再修改……，最后提交，这样很麻烦，而且难保你哪天会忘记修改……；

这个问题不难解决：可以建个配置文件，系统启动的时候从配置文件里读取鉴权信息，而这个配置文件在开发、测试、生产环境可以自定义，但不必提交到 github 上，同时在没有配置文件的时候以无需授权模式连接数据库（默认启动 mongodb 的时候也的确是无需授权即可访问的），也不会造成其他人 clone 你的代码并启动应用之后出现无法访问数据的情况，具体如下：

```
var fs              = require('fs'),
    path            = require('path');
var confFile        = path.normalize(__dirname + '/../conf/conf.json');
// 数据库配置文件不存在则尝试采用匿名方式访问
if ( !path.existsSync(confFile) ){
    console.log('[INFO]----Can not find database config file ' + confFile + ', using default config...');
    exports.db      = require('mongoskin').db('localhost:27017/dance');
}else{
    console.log('[INFO]----Connect to database using config file ' + confFile);
    var data        = JSON.parse(fs.readFileSync(confFile));
    var connection = 'mongo://' + data.dbAuth.username + ':' + data.dbAuth.password +
'@localhost:27017/dance';
    exports.db      = require('mongoskin').db(connection);
}
```

其中 conf.json 文件格式如下：

```
{
    "dbAuth"    :{
        "username"    : "",
        "password"    : ""
    }
}
```

这样以来就可以达到上述目标了。