# 为什么使用Sails?


JC_Huang (/u/843d2366f95b) + 关注
2015.04.25 16:35* 字数 2658 阅读 7013 评论 5 喜欢 26
(/u/843d2366f95b)

## 前言

入手Node.js半年，从用Express开发自己的博客到用Sails开发公司项目，深深被Sails震撼了。Sails是Balderdash团队的产品，快速的项目构建、优秀的框架结构还有众多的扩展，让我有种相见恨晚的感觉。在Koa流行之前，个人认为Sails的用户量还是挺可观的。今天，我想写一写Sails那些让我感动的地方，顺便理顺一下Sails的架构。

> **目录**
>
> 

## 一步搭建项目

> 在安装了Node.js 和 Sails的环境下，只需要一条命令，就能够搭建一个拥有完整架构的项目，尽管这很简单，我还是觉得有必要说一下。

在已经安装了Node.js和npm的前提下，首先你需要全局下安装Sails

```
$ sudo npm install sails -g
```

其次在一个空路径下，新建一个项目

```
$ sails new newApp
```

最后，只需要前往项目路径，把项目运行起来

```
$ cd testProject
$ sails lift
```

访问 `http://localhost:1337` 就能看到一个新的项目


new app

## 项目架构

```
.
├── api
│   ├── controllers
│   ├── models
│   ├── policies
│   ├── responses
│   └── services
├── views
├── assets
├── config
├── tasks
├── node_modules
├── package.json
├── Gruntfile.js
├── README.md
└── app.js
```

**api/**

`api` 目录下是你要构建应用的核心所在，常说的MVC的设计结构就体现在这里
`api/controllers` ： 控制层，该层是Http请求的入口。Sails官方建议该层只处理请求的转发和页面的渲染，具体的逻辑实现应该交给Service层。
`api/models`： 模型层，在Sails中，对于Model采用的是充血模型，除了可以在模型中定于属性之外，还可以定义包含逻辑处理的函数。在Sails中，所有Model都可以全局性访问。
`api/policies`： 过滤层，该层在Controller层之前对Http请求做处理，在这一层中，可以定于一些规则来过滤Http请求，比如身份认证什么的。
`api/responses`： http响应的方法都放这里，例如服务器错误、请求错误、404错误等，定义在responses文件夹里面的方法，都会赋值到controller层的req对象中。
`api/services`： 服务层，该层包含逻辑处理的方法，在Sails中，所有Service都可以全局性访问。

**views/**

视图层，存放视图模版文件的地方，Sails默认是提供ejs模版引擎的，如果你愿意，你可以换成jade、handlebars或者任何你喜欢的模版引擎。

**assets/**

资源文件夹，在Sails启动的时候，会启动某一个Grunt任务，把assets文件夹里的内容或压缩或编译或复制到根目录下的 `.tmp` 目录，这是前端可以直接通过路由访问的资源，HTML、JS、CSS以及图片等静态资源都放在这里了。

**config/**

配置文件夹，在Sails启动的时候，会加载该文件夹里的文件，并赋值在全局对象 `sails.config` 中，所以能够在任何一个地方都能用到。在用Sails开发，会经常跟这个文件夹里的文件打交道，从config的构成很容易知道Sails都提供哪方面的功能。

**tasks/**

Sails自带的项目自动化工具是 `Grunt` ，而Grunt的配置和任务注册都放在这个文件夹里了。这里已经提供了通常会用到的CSS编译、JS压缩、文件合并，更改检测等等任务，当然如果没有自己需要的，还能扩展。

**`app.js`**

Sails的启动文件，无论是 `$ sails lift` 命令或者 `$ npm start` 命令都会运行该文件。

## ORM

开发了Sails的团队Balderdash，还开发了一套ORM框架：Waterline。
Waterline在Sails主要的舞台是在 `/api/models` 目录里，在这里定义的模型文件，在Sails启动的时候，都要经由Waterline的洗礼。
Waterline 是通过Adapter关联数据库的，不同的Adapter关联不同的数据库。
Waterline 能适配绝对部分数据库，大致分类两类，一类是官方团队开发的 Adapter适配的，一类是民间开发者开发的Adapter适配的：
官方支持的:

- PostgreSQL
- MySQL
- MongoDB
- Redis
- Disk
- Memory

民间开发的：

- SQLServer
- OrientDB
- Oracle
- Cassandra

关于Waterline的更多信息可以关注：
github:waterline (https://link.jianshu.com?t=https://github.com/balderdashy/waterline)
github:waterline-docs (https://link.jianshu.com?t=https://github.com/balderdashy/waterline)

## MVC的实现

在这一段我想不仅仅要谈论到Model层、View层和Controller层，我认为还有必要谈到Service层、和Policy层。

### Model层

模型文件定义到 `/api/models` 中，由Waterline驱动，所有model都能全局访问。Sails提供命令行创建model的命令： `$ sails generate model MODEL_NAME`

### View层

实现在 `/views` 中，除了默认提供的ejs模版引擎之外，还能更换成jade、handlebars等模版引擎

### Controller层

在 `/api/controller` 目录里，Sails中提供创建controller的命令： `$ sails generate controller CONTROLLER_NAME` 。Sails也提供同时创建model和对应的Controller的命令： `$ sails generate api API_NAME` 。

### Service层

在 `/api/services` 目录里，存放自定义的服务，所有service都能够全局访问，Sails官方的建议是把逻辑处理都放在该层中，Controller层只做路由的分发和轻逻辑的处理。

### Policy层

在 `/api/policies` 目录里，存放自定义的过滤器。该层是一条请求在到达Controller之前根据需求过滤请求的中间层。在 `/api/policies` 目录中定义的文件，还需要在 `config/policies.js` 文件中为需求应用到某一过滤器的Action配置。

## 路由

Sails中要理解路由，首先要记得这个名词 `blueprint` ，中文翻译为：蓝图。我不知道官方是否解释过为什么要用个单词，但以我的理解，Sails的blueprint是负责指挥每一条客户端请求应该分配到服务器端的哪个Action去，所以叫蓝图吧。
blueprint主要分为三种： `RESTful routes` 、 `Shortcut routes` 、 `Action routes` 。

### RESTful routes

当路径诸如： `/:modelIdentity` 或者 `/:modelIdentity/:id` 的时候，blueprint会根据HTTP的动作（GET、POST、DELETE、PUT等）来分配到相应的Controller下相应的Action来处理。例如一个POST请求 `/user` 会创建一个用户，一个DELETE请求 `/user/123` 会删除 `id` 为123的用户。

### Shortcut routes

这种路由主要是方便开发，请求的参数可以直接写在请求路径中，例如 `/user/create?name=joe` 会创建一个新的用户， `/user/update/1?name=mike` 会更新 `id` 为1的用户的名字。
shortcut routes在开发环境很便利，但是在生产环境下需要关闭。

### Action routes

这种路由会自动的为Controller层的每一个Action创建一个路由，例如你的Controller层有一个 `FooController.js` ，里面有一个Action `bar` ，那么请求 `/foo/bar` 就会分配到 `bar` Action。

当然Sails也会提供自定义的路由，用户可以在 `config/routes.js` 和 `config/polices.js` 这两个配置文件中选择关闭或者打开blueprint提供的路由，和定义自己的路由。

## 安全

要确保产品的安全性，要对几种常见的攻击和安全策略了如指掌，诸如CORS、CSRF、DDOS、XSS等。Sails对于常见的安全策略都有提供支持，且只需要通过相关的配置文件就可以控制安全策略的等级。深入探讨Web的安全策略，并不在本文的范畴内，日后我会以这个为题写一篇文章聊聊Web的安全。
想要了解更多Sails的安全策略可以看看这里：sails: security (https://link.jianshu.com?t=http://sailsjs.org/#!/documentation/concepts/Security)

## 日志

Sails提供了一个全局对象 `sails.log` 用来处理日志信息的输出，日志是分level的，在 `config/log.js` 中配置日志输出的level，而level的作用看下表：

| Priority | level | Log fns visible |
| --- | --- | --- |
| 0 | silent | N/A |
| 1 | error | .error() |
| 2 | warn | .warn(), .error() |
| 3 | debug | .debug(), .warn(), .error() |

| Priority | level | Log fns visible |
| --- | --- | --- |
| 4 | info | .info(), .debug(), .warn(), .error() |
| 5 | verbose | .verbose(), .info(), .debug(), .warn(), .error() |
| 6 | silly | .silly(), .verbose(), .info(), .debug(), .warn(), .error() |

Sails的日志管理默认是info层的，既会输出.info(), .debug(), .warn(), .error()的信息。

## 单元测试

Sails使用了mocha进行单元测试，在新建Sails项目的时候，没有创建单元测试的文件夹，需要自己手动构造单元测试目录，官方建议的目录是这样的：

```
.
├── api
├── assets
├── ...
├── test
│   ├── unit
│   │   ├── controllers
│   │   │   └── UsersController.test.js
│   │   ├── models
│   │   │   └── Users.test.js
│   │   └── ...
│   ├── fixtures
│   ├── ...
│   ├── bootstrap.test.js
│   └── mocha.opts
└── views
```

而我在单元测试常用的组合是：mocha (https://link.jianshu.com?t=https://github.com/mochajs/mocha)、should (https://link.jianshu.com?t=https://github.com/tj/should.js)、supertest (https://link.jianshu.com?t=https://github.com/visionmedia/supertest)、 istanbul (https://link.jianshu.com?t=https://github.com/gotwarlost/istanbul)
其中should是提供断言，supertest是用于测试Controller层的时候伪造http请求的，而istanbul则是提供测试代码覆盖率的。
关于怎么在Sails中编写测试代码，可以参考 sails:testing (https://link.jianshu.com?t=http://sailsjs.org/#!/documentation/concepts/Testing)

## WebSocket

对于有即时性通讯需求的Web应用，我们会用Socket，Sails也为这方面提供了支持。在客户端提供js文件： `sails.io.js` ，而在服务器端提供全局对象： `sails.sockets` 。通过这两个对象，就可以进行客户端和服务器端即时性通讯的开发了。
Sails默认会启动WebSocket功能，在客户端访问服务器端的时候，会自动尝试在同域名下连接socket。
值得注意的是，这样会对AngularJS、EmberJS等前端MVVC开发产生一些障碍。
比如进行AngularJS开发的时候，我们在 `http://localhost:9000` 跑AngularJS项目，而服务器端却跑在 `http://localhost:1337` 。
当访问 `http://localhost:9000` 的时候， `sails.io.js` 会尝试于当前路径下进行socket连接，也就是 `http://localhost:9000` ，这时会出错，因为服务器是跑在 `http://localhost:1337` 的。
在开发的时候要解决这样的问题的时候，我们只需要在AngularJS这边引入 `sails.io.js` 之后定义连接路径就行了：

```
<script src="scripts/lib/sails.io.js"></script>
<script>io.sails.url = "http://localhost:1337";</script>
```

## 结语

可以说Sails涵盖了Web开发中会遇到的绝大部分需求和问题，如果深入研究Sails的话，是受益匪浅的。

---

如果本文对您有用
请不要吝啬你们的Follow与Start
这会大大支持我们继续创作

**「Github」**
MZMonster： @MZMonster (https://link.jianshu.com?t=https://github.com/MZMonster/)
JC_Huang： @JerryC8080 (https://link.jianshu.com?t=https://github.com/JerryC8080)

**小礼物走一走，来简书关注我**

赞赏支持



JC_Huang (/u/843d2366f95b)
写了 78093 字，被 926 人关注，获得了 1223 个喜欢
(/u/843d2366f95b)
+关注

君子，厚积薄发也 个人主页 | BLueSun [http://huang-jerryc.com]