japp 介绍

By @sogyf

Table of Contents

japp 介绍

`japp` 是在 @JFinal 框架的基础上，以及 @JFinal-ext 的扩展补充，参考 `Play 1.2` 系列的方式， 以约定配置的方式，一个快速、高效的web开发框架。

主要特点如下：

1. 在 `JFinal` 基础上，参考 `Play 1.2` 配置方式，将大部分的配置整合到一个配置文件，并支持动态启动相关插件等；
2. 需要使用 `JDK-1.6` 以及 支持 `Servlet 3.0` 以上版本的Web容器；
3. 通过 `javax.servlet.ServletContainerInitializer` (需要 `Servlet3.0` 以上容器)的方式去掉了 `web.xml` 的配置；
4. 通过 `jfgen` 工具支持那些不使用 `Maven` 的开发人员；
5. 整合 `job` 、 `mongodb` 、 `shiro` 、 `redis` 等常用插件；
6. 大量 `WEB` 开发中，常用的功能脚手架程序，比如： excel处理、文件上传处理、登录授权功能处理等；
7. 集成常用的工具包比如 `Google Guava` 等，并使用常用函数的示例；
8. 路由 COC、Model COC、Controller COC等；
9. `Freemarker` 支持 模版 继承，部分常用的标签等；
10. 测试自动执行 `misc/sql` 目录下的 `sql` 文件，默认按照文件名字母数字排序依次执行；
11.

`japp`框架概览

框架概览

核心：jfinal

> JFinal 是基于 Java 语言的极速 WEB + ORM 框架，其核心设计目标是开发迅速、代码量少、学习简单、功能强大、轻量级、易扩展、Restful。

`japp` 核心就是 `jFinal`，只是在 `jfinal` 的基础上，为常见的插件、组件以及相关代码和组织习惯而设定的一个WEB框架。

`Jfinal` 非常优秀，很好的插件体系，良好、简单的代码设计、MVC轻巧的构思、Java中应该是最方面的 `ActiveRecord` 数据库操作等等优点，但是由于之前我对 `PlayFramework` 的使用经历，感觉 `jfinal` 对于其配置方式，采用了比较宽松自由的方式来处理，只需要继承 `JfinalConfig` 类，设定路由、控制器、插件等；

在这一点，我由于受 `PlayFramework` 影响较深，感觉对于过多的暴露出配置方法和项目，对于团队开发的沟通起来，比较费时费力，如果参考 `PlayFramework` 的配置方法，只需要将配置项说明沟通好，就一切ok了！（当然，是否真的如此，还需要看团队吧）。

由于这样的原因，就用了 `japp`，为啥叫japp呢？ `japp=Jfinal Application`，其含义一目了然了，就是在Jfinal的基础上，为应用本身提供一个脚手架的基础架构。

5分钟指南

> 相关 `Maven` 的配置和指南，请参考 Apache Maven

1. 建立Maven WEB工程

   使用Maven命令创建一个WEB工程，这里作为示例，使用 `todo` 名称，如下所示：

   ```
   mvn archetype:generate\
       -DgroupId=todo\
       -DartifactId=todo\
       -DarchetypeArtifactId=maven-archetype-webapp
   ```

   `DarchetypeArtifactId=maven-archetype-webapp` 指定用Webapp的方式来创建一个Maven工程

   > 这里建议，groupId 和 artifactId保持一样吧，很多的都是区分开来，感觉有些多余，当然，这只是个建议

2. 增加 `japp` 的项目依赖

   - 项目创建成果后，修改 `pom` 项目依赖，增加japp的依赖
   - 在 `dependencies` 节点上，增加以下依赖

     ```
     <dependency>
         <groupId>com.github.sogyf</groupId>
         <artifactId>jfinal-app</artifactId>
         <version>0.0.9</version>
     </dependency>
     ```

   - 只需要增加 `jfinal-app` 的依赖即可，Maven会自动将其相关依赖传播到自己的应用中。

3. 建立相关COC配置文件

   - 在 `japp` 中，采用了如下的约定形式，以不同包（注意，是固定包名）的方式，分别代表不同的作用；

```
.
├── logs                                # 日志文件目录
├── misc                                # 相关构建脚本和数据库脚本的文件夹存放目录
├── pom.xml                             # Maven的依赖文件
├── readme.md                           # 项目描述MarkDown文件
├── src                                 # Java WEB 工程源文件目录，包括视图和Java类
以及单元测试类
|   ├── main                            # Java WEB 工程源文件目录，包括视图和Java类
|   |   ├── java                        # Java 源代码目录
|   |   |   ├── app                     # 应用包名（不能使用其他的）
|   |   |   |   ├── Const.java          # 应用常量类
|   |   |   |   ├── controllers         # jfinal 控制器目包
|   |   |   |   ├── dtos                # 数据传输DTO包
|   |   |   |   ├── events              # 系统启动后或者异步事件包
|   |   |   |   ├── filters             # 过滤器,Servlet3.1规范
|   |   |   |   ├── interceptors        # jfinal 拦截器包，包括全局或者单个的拦截器
|   |   |   |   ├── jobs                # 定时任务包
|   |   |   |   ├── kits                # 常用工具包
|   |   |   |   ├── models              # jfinal activieRecord 数据库映射包
|   |   |   |   └── validators          # 表单验证器包
|   |   ├── resources                   # WEB相关配置和资源包
|   |   |   ├── application.conf        # japp的配置文件（非常重要，名字不能更换）
|   |   |   ├── ehcache.xml             # 缓存配置
|   |   |   ├── logback.xml             # 日志配置
|   |   |   ├── shiro.ini               # Apache Shiro身份登录等配置
|   |   |   └── sqlcnf                  # 数据库映射SQL 配置文件目录
|   |   └── webapp                      # WEB 工程目录，包括视图和静态资源等
|   |       ├── WEB-INF
|   |       |   └── views               # 视图，freemarker文件目录
|   |       └── static                  # 静态资源CSS、JS以及图片的存储目录
|   └── test                            # 单元测试目录
|       ├── java                        # 单元测试类包
└──────── └── resources                 # 单元测试资源
```

   - 如果您是Linux操作系统，可通过如下命令直接创建：

```
mkdir -p src/main/java/app/{controllers,dtos,interceptors,events,jobs,kits,models,
validators}
mkdir -p src/main/resources/sqlcnf/
mkdir -p src/main/webapp/WEB-INF/views
mkdir -p src/main/webapp/static
```

   - 然后删除 `webapp` 目录下的 `jsp` 文件和 `web.xml` 文件

```
rm -rf src/main/webapp/index.jsp
rm -rf src/main/webapp/WEB-INF/web.xml
```

   到这里，相关的约定文件即配置完毕。

4. 新建首页控制器

   在 `app.controllers` 目录下新建 `IndexController` 类，并使得 `IndexController` 类继承 `com.github.sog.controller.BasicController` ,如下所示：

```
package app.controllers;

import com.github.sog.controller.BasicController;

/**
 * <p>
 * Default index Controller.
 * </p>
 */
public class IndexController extends BasicController {

    public void index(){
        render("index.ftl")
    }
}
```

- 首先，继承 `BasicController` 表示当前的类为控制器;
- 其次，这个控制器处理的路由请求是 `\index`，路由引擎会自动根据 控制器类名 小写驼峰开头的方式 + 去掉 `Controller` 后的字符串作为路由地址，比如：这里的 `IndexController` 路由地址就是 `\index`，这里我们称之为 `controllerkey`。

> 同时，需要注意的是，如果 `IndexController` 是属于 `app.controllers` 包下的某个子包的话，那么路由应该是 `子包名称` + `\index`，例如：如果 `IndexController` 在包 `app.controllers.admin` 包下，那么对应的路由地址是 `\admin\index`，其他依次类推；

- 路由引导后，其它的就是按照 `JFinal` 的默认路由规则处理了。详细的可通过 官方指导pdf，这里摘录比较重要的路由规则表，如下：

```
  {
      "url": "controllerkey", "Controller": "ControllerkeyController.index()",
      "url": "controllerkey/method", "Controller": "ControllerkeyController.method
()",
      "url": "controllerKey/method/v0-v1", "Controller": "ControllerkeyController.
method(),所带 url 参数值为：v0-v1",
      "url": "controllerKey/v0-v1", "Controller": "ControllerkeyController.index()
,所带 url 参数值为：v0-v1",
  }
```

5. 增加应用配置文件 `application.conf`

在 `src/main/resources` 目录下新建 `application.conf` 文件，这里，我们先不管其具体的配置项的作用（虽然看起来就能够明白），文件内容如下：

```
app=todo
app.version=0.1

domain=http://127.0.0.0.1:8080/todo
dev.mode=true
cache=true

security=false

# Start the database plug-in configuration db.url said.
#db.url=jdbc:mysql://127.0.0.1:3306/todo?useUnicode=true&characterEncoding=utf-8
#db.username=todo
```

```
#db.password=todo@japp
#db.sqlinxml=true

#job=true

# Enable MongdoDB plugin
#mongo.host=192.168.1.210
#mongo.port=27017
#mongo.db=todo
# Enable MongoDB ORM plugin.
#mongo.morphia=

# Enable Redis Plugin.
#redis.url=
```

6. 增加视图文件

   在 第4步 的 控制器中 `IndexController.index()` 方法中，使用了 `render` 方法，这里， `japp` 中的约定，会直接渲染 `WEB-INF\view\controllerkey` 下的指定文件，如果不明确调用 `render` 方法来指定的话，那么会自动根据方法名称选择对应的模版文件，比如，在这里的 `IndexController.index()` 会自动渲染 `WEB-INF\view\index\index.ftl` 文件出来

   ```
   <html>
   <head>
       <title>japp example</title>
   </head>
   <body>
       <h1>japp has start.</h1>
   </body>
   </html>
   ```

   同理，如果 `app.controllers` 控制器包下，存在子包，那么在 `WEB-INF\view\` 也应该存在同名的文件目录。

   增加 `index.ftl` 文件，然后在 `index.ftl` 中增加点内容，比如 Like japp等

7. 运行程序

   - 配置Maven 的Tomcat 插件 启动

     在项目的Maven的 `pom.xml` 中的 `build` 节点，增加如下配置

     ```
     <plugin>
         <groupId>org.apache.tomcat.maven</groupId>
         <artifactId>tomcat7-maven-plugin</artifactId>
         <version>2.2</version>
         <configuration>
             <port>9000</port>
             <path>/todo</path>
             <uriEncoding>${file.encoding}</uriEncoding>
         </configuration>
     </plugin>
     ```

     在命令行下使用 `mvn tomcat:run` 命令启动，后访问http://localhost:9000/todo/index

     到这里，基本上一个普通的WEB页面就配置ok了!

`japp`安装指南

jfapp 命令行工具介绍

`jfapp` 是参考 `Play Framework` 的命令行工具，主要特点：

```
- 创建工程
- 构建应用，生产WAR包
- 自动生成数据库映射实体
- 转换`IDEA`工程
- 转换`Maven`工程
```

jfapp 命令行安装

jfapp 使用

jfapp 创建工程

`japp`中的主要概念

路由

控制器

模版`Freemarker`介绍

表单验证

`ActiviteRecord`数据库操作介绍

常用插件

缓存

发送电子邮件

测试您的控制器

齐全的工具帮助类

日志

`ii8`国际化

`japp`使用的库

部署到生产环境