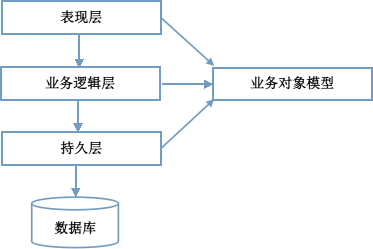
基于SpringBoot + Mybatis实现SpringMVC Web项目【原创】

**一、热身**

一个现实的场景是：当我们开发一个Web工程时，架构师和开发工程师可能更关心项目技术结构上的设计。而几乎所有结构良好的软件（项目）都使用了分层设计。分层设计是将项目按技术职能分为几个内聚的部分，从而将技术或接口的实现细节隐藏起来。



从另一个角度上来看，结构上的分层往往也能促进了技术人员的分工，可以使开发人员更专注于某一层业务与功能的实现，比如前端工程师只关心页面的展示与交互效果（例如专注于HTML，JS等），而后端工程师只关心数据和业务逻辑的处理（专注于Java，Mysql等）。两者之间通过标准接口（协议）进行沟通。

在实现分层的过程中我们会使用一些框架，例如SpringMVC。但利用框架带来了一些使用方面的问题。我们经常要做的工作就是配置各种XML文件，然后还需要搭建配置Tomcat或者Jetty作为容器来运行这个工程。每次构建一个新项目，都要经历这个流程。更为不幸的是有时候前端人员为了能在本地调试或测试程序，也需要先配置这些环境，或者需要后端人员先实现一些服务功能。这就和刚才提到的“良好的分层结构”相冲突。

每一种技术和框架都有一定的学习曲线。开发人员需要了解具体细节，才知道如何把项目整合成一个完整的解决方案。事实上，一个整合良好的项目框架不仅仅能实现技术、业务的分离，还应该关注并满足开发人员的“隔离”。

为了解决此类问题，便产生了Spring Boot这一全新框架。Spring Boot就是用来简化Spring应用的搭建以及开发过程。该框架致力于实现免XML配置，提供便捷，独立的运行环境，实现“一键运行”满足快速应用开发的需求。

与此同时，一个完整的Web应用难免少不了数据库的支持。利用JDBC的API需要编写复杂重复且冗余的代码。而使用O/RM（例如Hibernate）工具需要基于一些假设和规则，例如最普遍的一个假设就是数据库被恰当的规范了。这些规范在现实项目中并非能完美实现。由此，诞生了一种混合型解决方案——Mybatis。Mybatis是一个持久层框架，它从各种数据库访问工具中汲取大量优秀的思想，适用于任何大小和用途的数据库。根据官方文档的描述：MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以对配置和原生Map使用简单的 XML 或注解，将接口和 Java 的 POJOs(Plain Old Java Objects，普通的 Java对象)映射成数据库中的记录。

最后，再回到技术结构分层上，目前主流倡导的设计模式为MVC，即模型(model)－视图(view)－控制器(controller)。实现该设计模式的框架有很多，例如Struts。而前文提到的SpringMVC是另一个更为优秀，灵活易用的MVC框架。 SpringMVC是一种基于Java的以请求为驱动类型的轻量级Web框架，其目的是将Web层进行解耦，即使用“请求-响应”模型，从工程结构上实现良好的分层，区分职责，简化Web开发。

目前，对于如何把这些技术整合起来形成一个完整的解决方案，并没有相关的最佳实践。将Spring Boot和Mybatis两者整合使用的资料和案例较少。因此，本文提供（介绍）一个完整利用SpringBoot和Mybatis来构架Web项目的案例。该案例基于SpringMVC架构提供完整且简洁的实现Demo，便于开发人员根据不同需求和业务进行拓展。

补充提示，Spring Boot 推荐采用基于 Java 注解的配置方式，而不是传统的 XML。只需要在主配置 Java 类上添加“@EnableAutoConfiguration”注解就可以启用自动配置。Spring Boot 的自动配置功能是没有侵入性的，只是作为一种基本的默认实现。开发人员可以通过定义其他 bean 来替代自动配置所提供的功能，例如在配置本案例数据源（DataSource）时，可以体会到该用法。

**二、实践**

一些说明：

项目IDE采用Intellij（主要原因在于Intellij颜值完爆Eclipse，谁叫这是一个看脸的时代）

工程依赖管理采用个人比较熟悉的Maven（事实上SpringBoot与Groovy才是天生一对）

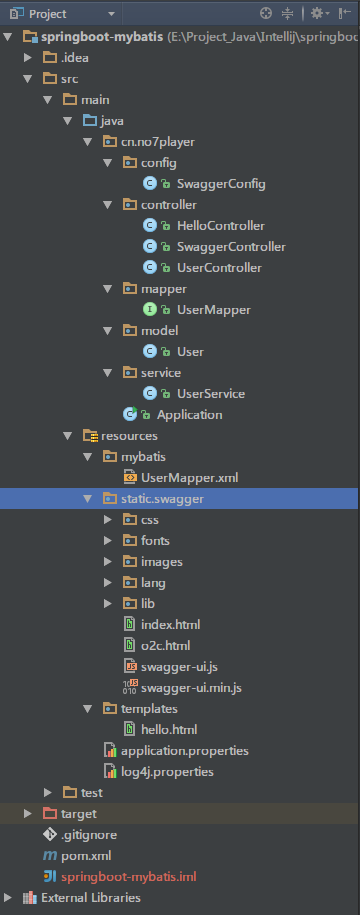
**1.预览：**

（1）github地址

<https://github.com/djmpink/springboot-mybatis>

git :  <https://github.com/djmpink/springboot-mybatis.git>

（2）完整项目结构



（3）数据库

数据库名：test

【user.sql】

SET FOREIGN\_KEY\_CHECKS=0;

-- ----------------------------

-- Table structure for user

-- ----------------------------

DROP TABLE IF EXISTS `user`;

CREATE TABLE `user` (

`id` int(11) NOT NULL,

`name` varchar(255) DEFAULT NULL,

`age` int(11) DEFAULT NULL,

`password` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', '7player', '18', '123456');

**2.Maven配置**

完整的【pom.xml】配置如下：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>cn.7player.framework</groupId>

<artifactId>springboot-mybatis</artifactId>

<version>1.0-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.2.5.RELEASE</version>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.7</java.version>

</properties>

<dependencies>

<!--Spring Boot-->

<!--支持 Web 应用开发，包含 Tomcat 和 spring-mvc。 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--模板引擎-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<!--支持使用 JDBC 访问数据库-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<!--添加适用于生产环境的功能，如性能指标和监测等功能。 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<!--Mybatis-->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.2</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.2.8</version>

</dependency>

<!--Mysql / DataSource-->

<dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-jdbc</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!--Json Support-->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.1.43</version>

</dependency>

<!--Swagger support-->

<dependency>

<groupId>com.mangofactory</groupId>

<artifactId>swagger-springmvc</artifactId>

<version>0.9.5</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestone</id>

<url>https://repo.spring.io/libs-release</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-milestone</id>

<url>https://repo.spring.io/libs-release</url>

</pluginRepository>

</pluginRepositories>

</project>

**3.主函数**

【Application.java】包含main函数，像普通java程序启动即可。

此外，该类中还包含和数据库相关的DataSource，SqlSeesion配置内容。

注：@MapperScan(“cn.no7player.mapper”) 表示Mybatis的映射路径（package路径）

package cn.no7player;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.log4j.Logger;

import org.mybatis.spring.SqlSessionFactoryBean;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.core.io.support.PathMatchingResourcePatternResolver;

import org.springframework.jdbc.datasource.DataSourceTransactionManager;

import org.springframework.transaction.PlatformTransactionManager;

import javax.sql.DataSource;

@EnableAutoConfiguration

@SpringBootApplication

@ComponentScan

@MapperScan("cn.no7player.mapper")

public class Application {

private static Logger logger = Logger.getLogger(Application.class);

//DataSource配置

@Bean

@ConfigurationProperties(prefix="spring.datasource")

public DataSource dataSource() {

return new org.apache.tomcat.jdbc.pool.DataSource();

}

//提供SqlSeesion

@Bean

public SqlSessionFactory sqlSessionFactoryBean() throws Exception {

SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();

sqlSessionFactoryBean.setDataSource(dataSource());

PathMatchingResourcePatternResolver resolver = new PathMatchingResourcePatternResolver();

sqlSessionFactoryBean.setMapperLocations(resolver.getResources("classpath:/mybatis/\*.xml"));

return sqlSessionFactoryBean.getObject();

}

@Bean

public PlatformTransactionManager transactionManager() {

return new DataSourceTransactionManager(dataSource());

}

/\*\*

\* Main Start

\*/

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

logger.info("============= SpringBoot Start Success =============");

}

}

**4.Controller**

请求入口Controller部分提供三种接口样例：视图模板，Json，restful风格

（1）视图模板

返回结果为视图文件路径。视图相关文件默认放置在路径 resource/templates下：

package cn.no7player.controller;

import org.apache.log4j.Logger;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

@Controller

public class HelloController {

private Logger logger = Logger.getLogger(HelloController.class);

/\*

\* http://localhost:8080/hello?name=cn.7player

\*/

@RequestMapping("/hello")

public String greeting(@RequestParam(value="name", required=false, defaultValue="World") String name, Model model) {

logger.info("hello");

model.addAttribute("name", name);

return "hello";

}

}

（2）Json

返回Json格式数据，多用于Ajax请求。

package cn.no7player.controller;

import cn.no7player.model.User;

import cn.no7player.service.UserService;

import org.apache.log4j.Logger;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class UserController {

private Logger logger = Logger.getLogger(UserController.class);

@Autowired

private UserService userService;

/\*

\* http://localhost:8080/getUserInfo

\*/

@RequestMapping("/getUserInfo")

@ResponseBody

public User getUserInfo() {

User user = userService.getUserInfo();

if(user!=null){

System.out.println("user.getName():"+user.getName());

logger.info("user.getAge():"+user.getAge());

}

return user;

}

}

（3）restful

REST 指的是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是 RESTful。

此外，有一款RESTFUL接口的文档在线自动生成+功能测试功能软件——Swagger UI，具体配置过程可移步《Spring Boot 利用 Swagger 实现restful测试》

package cn.no7player.controller;

import cn.no7player.model.User;

import com.wordnik.swagger.annotations.ApiOperation;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping(value="/users")

public class SwaggerController {

/\*

\* http://localhost:8080/swagger/index.html

\*/

@ApiOperation(value="Get all users",notes="requires noting")

@RequestMapping(method=RequestMethod.GET)

public List<User> getUsers(){

List<User> list=new ArrayList<User>();

User user=new User();

user.setName("hello");

list.add(user);

User user2=new User();

user.setName("world");

list.add(user2);

return list;

}

@ApiOperation(value="Get user with id",notes="requires the id of user")

@RequestMapping(value="/{name}",method=RequestMethod.GET)

public User getUserById(@PathVariable String name){

User user=new User();

user.setName("hello world");

return user;

}

}

**5.Mybatis**

配置相关代码在Application.java中体现。

（1）【application.properties】

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=gbk&zeroDateTimeBehavior=convertToNull

spring.datasource.username=root

spring.datasource.password=123456

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

注意，在Application.java代码中，配置DataSource时的注解

@ConfigurationProperties(prefix=“spring.datasource”)

表示将根据前缀“spring.datasource”从application.properties中匹配相关属性值。

（2）【UserMapper.xml】

Mybatis的sql映射文件。Mybatis同样支持注解方式，在此不予举例了。

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="cn.no7player.mapper.UserMapper">

<select id="findUserInfo" resultType="cn.no7player.model.User">

select name, age,password from user;

</select>

</mapper>

（3）接口UserMapper

package cn.no7player.mapper;

import cn.no7player.model.User;

public interface UserMapper {

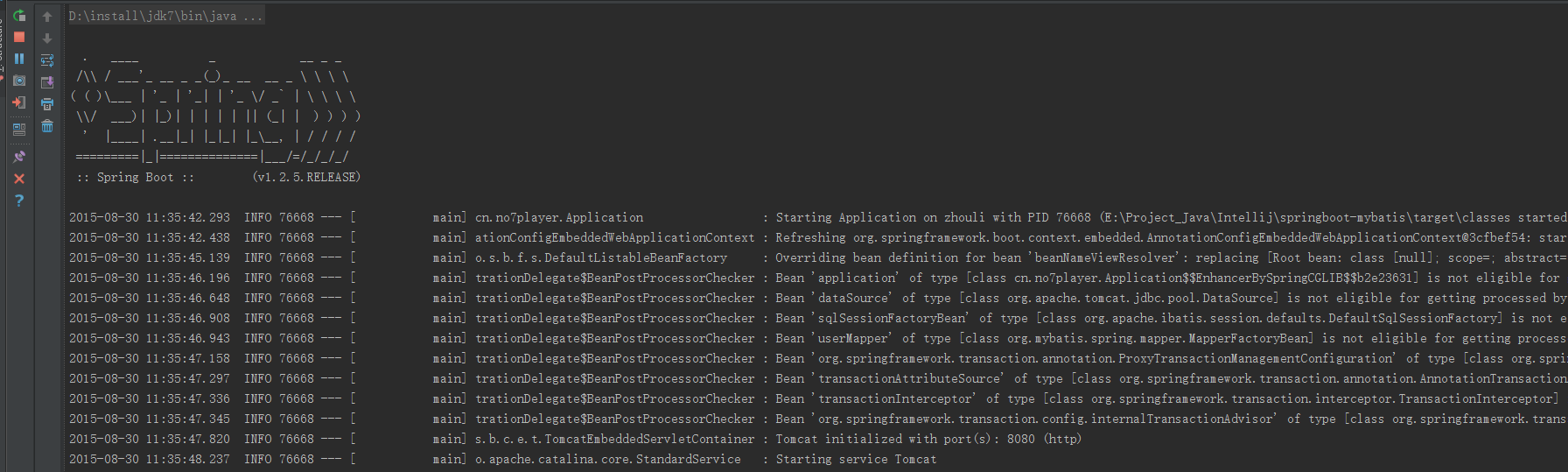
public User findUserInfo();

}

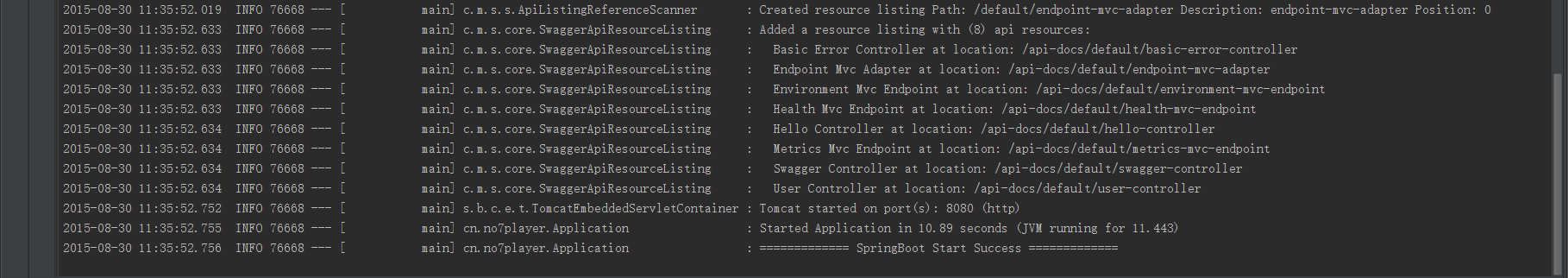
**三、总结**

（1）运行 Application.java

（2）控制台输出：



…..（略过无数内容）

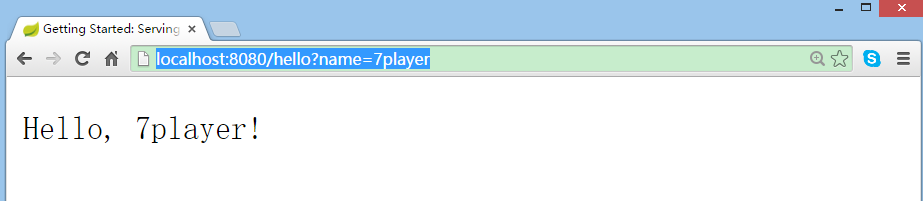


（3）访问：

针对三种控制器的访问分别为：

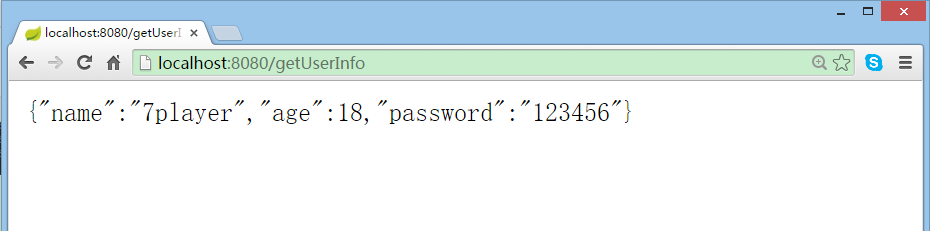
视图：

<http://localhost:8080/hello?name=7player>



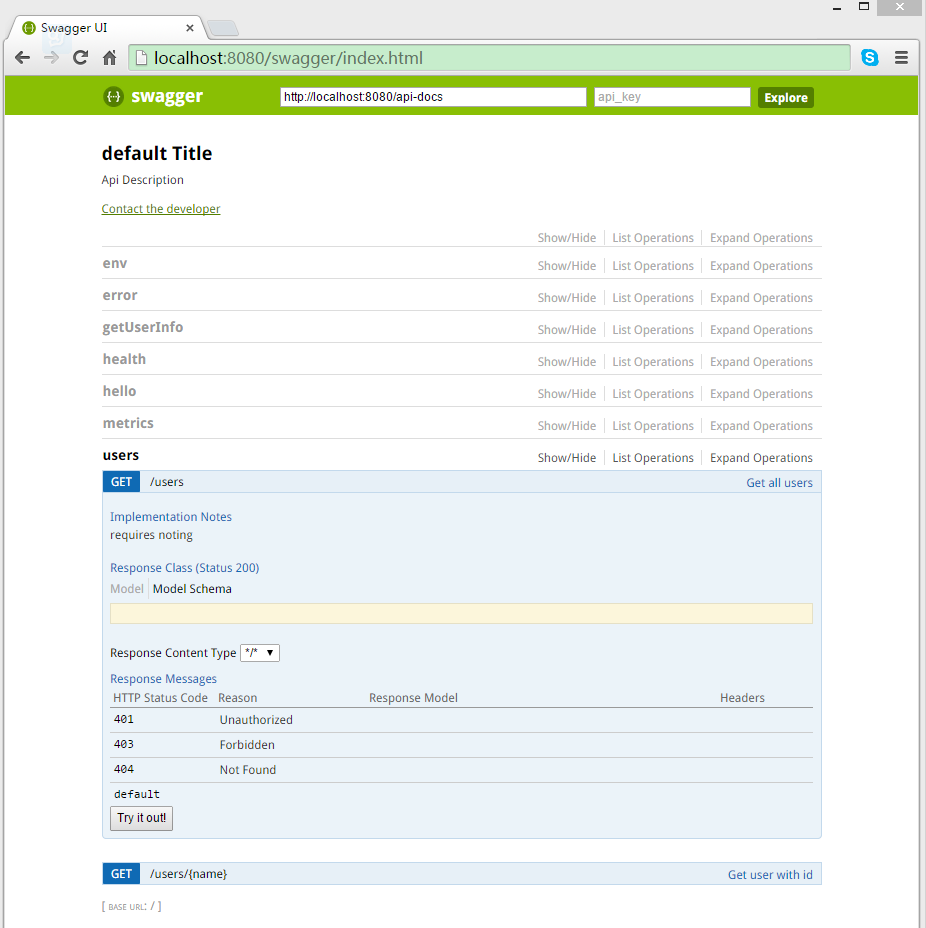
Json：

<http://localhost:8080/getUserInfo>



Restful（使用了swagger）：

<http://localhost:8080/swagger/index.html>



**四、参阅**

《Spring Boot – Quick Start》

[http://projects.spring.io/spring-boot/#quick-start](http://7player.cn/2015/08/30/%E3%80%90%E5%8E%9F%E5%88%9B%E3%80%91%E5%9F%BA%E4%BA%8Espringboot-mybatis%E5%AE%9E%E7%8E%B0springmvc-web%E9%A1%B9%E7%9B%AE/#quick-start)

《mybatis》

<http://mybatis.github.io/mybatis-3/>

《使用 Spring Boot 快速构建 Spring 框架应用》

<http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/>

《Using @ConfigurationProperties in Spring Boot》

<http://www.javacodegeeks.com/2014/09/using-configurationproperties-in-spring-boot.html?utm_source=tuicool>

《Springboot-Mybatis-Mysample》

<https://github.com/mizukyf/springboot-mybatis-mysample>

《Serving Web Content with Spring MVC》

<http://spring.io/guides/gs/serving-web-content/>

《理解RESTful架构》

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

**附录：**

Spring Boot 推荐的基础 POM 文件

|  |  |
| --- | --- |
| **名称** | **说明** |
| spring-boot-starter | 核心 POM，包含自动配置支持、日志库和对 YAML 配置文件的支持。 |
| spring-boot-starter-amqp | 通过 spring-rabbit 支持 AMQP。 |
| spring-boot-starter-aop | 包含 spring-aop 和 AspectJ 来支持面向切面编程（AOP）。 |
| spring-boot-starter-batch | 支持 Spring Batch，包含 HSQLDB。 |
| spring-boot-starter-data-jpa | 包含 spring-data-jpa、spring-orm 和 Hibernate 来支持 JPA。 |
| spring-boot-starter-data-mongodb | 包含 spring-data-mongodb 来支持 MongoDB。 |
| spring-boot-starter-data-rest | 通过 spring-data-rest-webmvc 支持以 REST 方式暴露 Spring Data 仓库。 |
| spring-boot-starter-jdbc | 支持使用 JDBC 访问数据库。 |
| spring-boot-starter-security | 包含 spring-security。 |
| spring-boot-starter-test | 包含常用的测试所需的依赖，如 JUnit、Hamcrest、Mockito 和 spring-test 等。 |
| spring-boot-starter-velocity | 支持使用 Velocity 作为模板引擎。 |
| spring-boot-starter-web | 支持 Web 应用开发，包含 Tomcat 和 spring-mvc。 |
| spring-boot-starter-websocket | 支持使用 Tomcat 开发 WebSocket 应用。 |
| spring-boot-starter-ws | 支持 Spring Web Services。 |
| spring-boot-starter-actuator | 添加适用于生产环境的功能，如性能指标和监测等功能。 |
| spring-boot-starter-remote-shell | 添加远程 SSH 支持。 |
| spring-boot-starter-jetty | 使用 Jetty 而不是默认的 Tomcat 作为应用服务器。 |
| spring-boot-starter-log4j | 添加 Log4j 的支持。 |
| spring-boot-starter-logging | 使用 Spring Boot 默认的日志框架 Logback。 |
| spring-boot-starter-tomcat | 使用 Spring Boot 默认的 Tomcat 作为应用服务器。 |