**用Spring Boot Java应用开发**

## Spring Boot是颠覆性的

* 为什么说Spring Boot是颠覆性的呢？因为它彻底改变了Java应用开发的模式。以往我们做Spring应用开发，要知道配置哪些类来让Hibernate和Spring一起工作，要知道如何配置 来控制哪个模版进行视图层的展示。经常写了一大堆代码之后发现只是在处理Spring框架本身的配置，根本一行业务逻辑都没有写。开发完成之后，我们还要考虑部署的问题，且不说部署到非常笨重的应用服务器，比如WebSphere, Weblogic或者JBoss，即使部署到Tomcat或者Jetty这种轻量级容器上面，我们要知道如何配置容器，如何修改配置文件等等。而且在多应用部署到同一个Tomcat的时候，经常会出现冲突。就算我们花了很大力气解决了这些问题，程序部署成功之后，我们很难去了解这个程序的运行状态。有可能我们要配置很多第三方工具来去知道这个应用程序运行状态如何，有哪些参数，环境变量是什么。尽管Spring帮我们解决了依赖注入的问题，简化了一些MVC的流程，但是Spring框架本身集成了越来越多东西，导致其越来越难配置，维护成本成直线上升。很多时候Java程序员们看到Python或者JavaScript程序员敲几个命令安装一些库，然后简单的敲几行代码，引入一些框架比如flask，然后直接一个简单的API就可以跑起来了。这时候Java程序员可能还在研究该使用Maven里面的哪个库，如何在代码里面进行配置呢。
* 。

新建springBoot Hollewold

1,新建一个maven项目

2, 创建pom.xml文件

<?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>rabbit</groupId>

<artifactId>springBootDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>springBootDemo</name>

<url>http://maven.apache.org</url>

<parent>

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

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

<version>1.4.0.RELEASE</version>

</parent>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<!-- <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-freemarker</artifactId>

</dependency> -->

<!-- mysql连接 -->

<!-- <dependency> <groupId>mysql</groupId> <artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope> </dependency> -->

<!-- sqlserver连接 -->

<dependency>

<groupId>com.microsoft.sqlserver</groupId>

<artifactId>jdbc</artifactId>

<version>4.0.2206.100</version>

</dependency>

<!-- mybatis -->

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>1.1.1</version>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

新建 启动入口

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Demo1Application.**class**, args);

}

**和mybatis整合:**

@SpringBootApplication//等价于//@ComponentScan+@EnableAutoConfiguration+@Configuration

@MapperScan("com.example.mapper")

**public** **class** Demo1Application {

**protected** **static** Logger *logger*=LoggerFactory.*getLogger*(Demo1Application.**class**);

@Bean

@ConfigurationProperties(prefix="spring.datasource")

**public** DataSource dataSource() {

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

}

@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());

}

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Demo1Application.**class**, args);

}

}

**application.properties:**

spring.datasource.url=jdbc:sqlserver://192.168.6.135:1433;databaseName=ActionDBV2

spring.datasource.username=user\_app\_Test

spring.datasource.password=AD19EFAC580DF3

spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver

spring.datasource.max-active=10

spring.datasource.max-idle=5

spring.datasource.min-idle=0

security.user.name=admin

security.user.password=test

#指定log的配置文件，以及记录Spring Boot的log级别

#logging.config=logback1-spring.xml

#logging.level.org.springframework.web: INFO

# 项目contextPath，一般在正式发布版本中，我们不配置

# server.context-path=/myspringboot

# 错误页，指定发生错误时，跳转的URL。请查看BasicErrorController源码便知

# server.error.path=/error

# 服务端口

# server.port=9090

* 1. 在src/main/java包下的第一层结构中，是必须放一个含有main方法的主启动的类，而且只能有一个main方法，如果再出现其他的main方法，在使用maven编译打包时，会报编译错误，当然在src/test/java包下，可以出现多个，但建议最好使用Junit进行单元测试.
  2. Spring Boot支持大部分的log配置，其中包括：   
     （1）java util logging   
     （2）log4j   
     （3）log4j2   
     （4）logbak   
     默认的情况下spring boot会选择logback作为日志记录的载体，当然要想它正常的工作，需要依赖 Commons Logging， Java Util Logging， Log4J 或 SLF4J， Spring Boot推荐的logbak作为日志的配置文件 默认: logback-spring.xml

介绍下Spring Boot中几个常用的注解，其中大部分都是来自Spring MVC的注解,这些注解使用的频率一般都在90%之上。   
  
（1）@RestController和@Controller指定一个类，作为控制器的注解   
（2）@RequestMapping方法级别的映射注解，这一个用过Spring MVC的小伙伴相信都很熟悉   
（3）@EnableAutoConfiguration和@SpringBootApplication是类级别的注解，根据maven依赖的jar来自动猜测完成正确的spring的对应配置，只要引入了spring-boot-starter-web的依赖，默认会自动配置Spring MVC和tomcat容器   
（4）@Configuration类级别的注解，一般这个注解，我们用来标识main方法所在的类,完成元数据bean的初始化。   
（5）@ComponentScan类级别的注解，自动扫描加载所有的Spring组件包括Bean注入，一般用在main方法所在的类上   
（6）@ImportResource类级别注解，当我们必须使用一个xml的配置时，使用@ImportResource和@Configuration来标识这个文件资源的类。   
（7）@Autowired注解，一般结合@ComponentScan注解，来自动注入一个Service或Dao级别的Bean   
（8）@Component类级别注解，用来标识一个组件，比如我自定了一个filter，则需要此注解标识之后，Spring Boot才会正确识别

**java -jar spring\_boot\_interface-0.0.1-SNAPSHOT.jar --spring.config.location=/application.properties &**