SpringBoot教程梳理

By huangnx

首先明确的说：

SpringBoot并不是一个全新的体系，全新的架构。

他只是在Spring的基础上，根据大家使用spring这么多年积累的经验，将一些最佳实践或者习惯性的使用方式总结了下，重新做了一层封装，以便我们能更快速的构建一个spring应用。

简单的说SpringBoot其实就是默认配置了很多框架的使用方式。

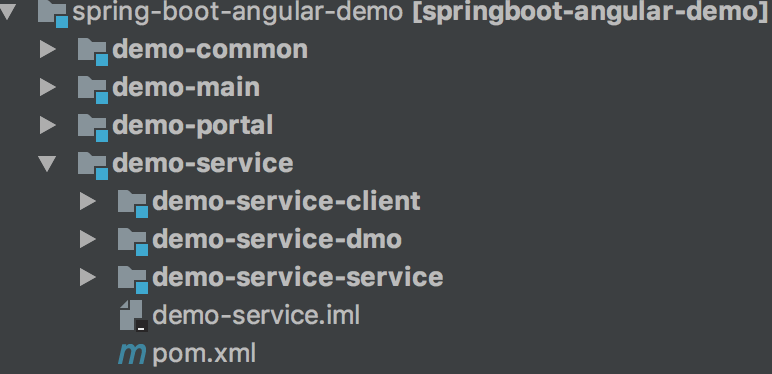
具体体现在哪里嘞？我举一些例子来帮助大家理解：

1. 内嵌servlet容器，无需以war包的形式部署项目。可以以jar包的形式运行
2. 简化maven配置
3. 自动配置Spring
4. 应用监控

大家都是老手了。 Spring的东西就那么多，不详细的一一讲了。这里以demo为例。通过demo取了解springboot。

# 跟着demo学习springboot

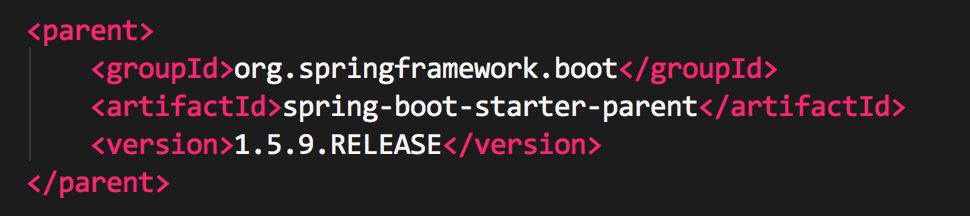
## 工程结构。如图：



## 5个module:

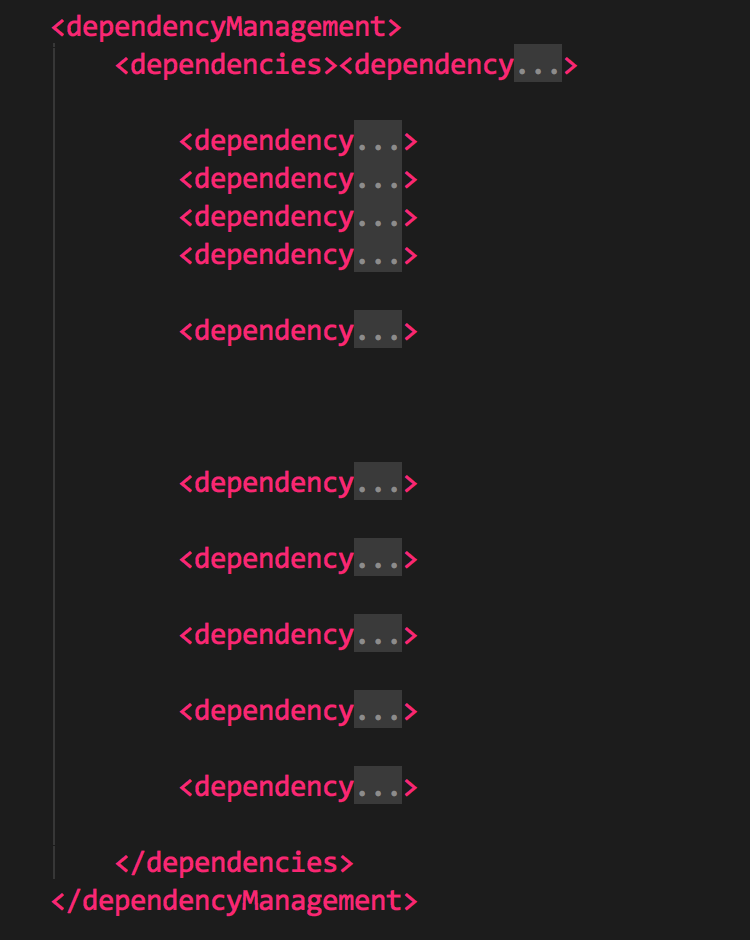
* demo-portal：controller层的东西
* demo-service：
* demo-service-client：对外暴露服务的接口定义
* demo-service-dmo：数据库实体定义
* demo-service-service：服务实现
* demo-common：常量、工具类、全局配置
* demo-main：可执行程序
* demo-test：

## 父pom：



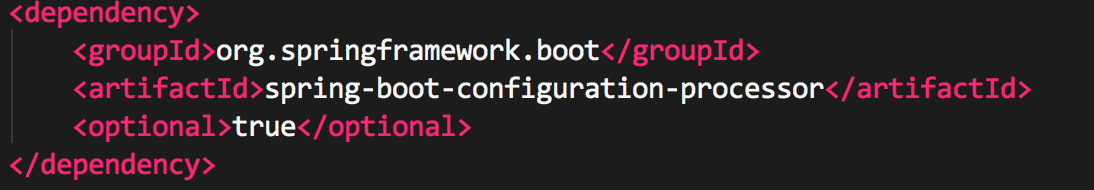
需要注明parent。这里用的是springboot 1.5.9.RELEASE 的版本。

然后定义了下其它pom依赖以及内部module依赖的版本管理

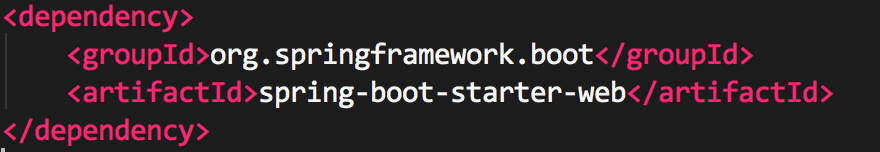


## 几个核心的pom依赖介绍

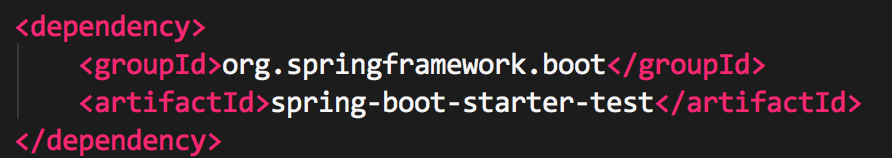
加载自定义配置类所需：



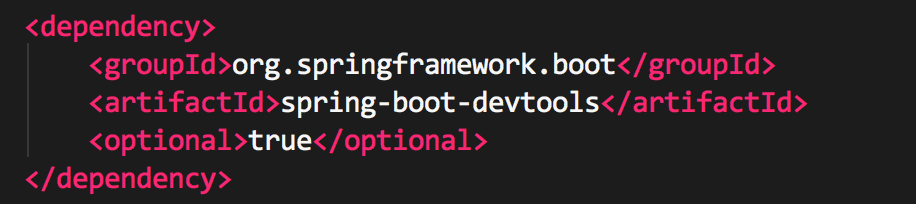
SpringBoot，Web工程必需



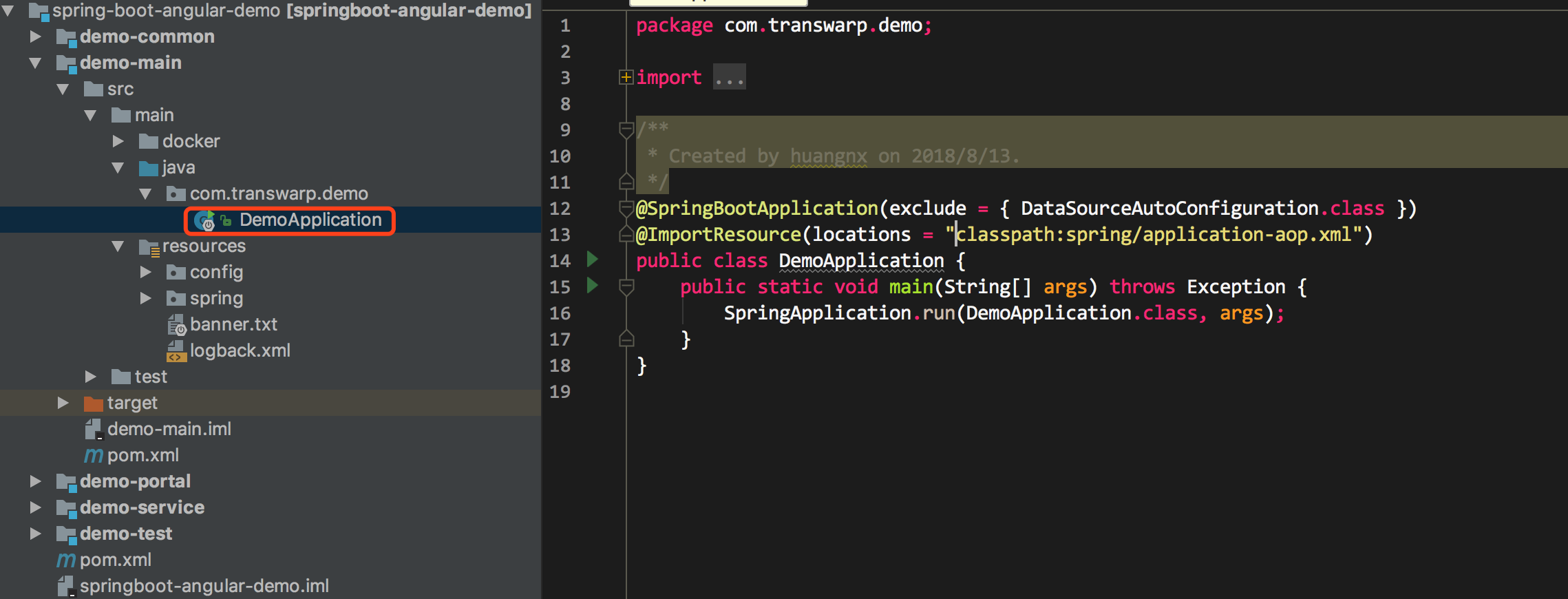
SpringBoot，单元测试必需



SpringBoot，开发热部署必需



## springboot应用入口类：



### class上加注解：@SpringBootApplication，标明这是一个springboot的应用。

* 注解：@SpringBootApplication是多个springboot注解的聚合。
  + 其中三个比较重要
    - @Configuration（@SpringBootConfiguration点开查看发现里面还是应用了@Configuration）
    - @EnableAutoConfiguration
    - @ComponentScan

@ Configuration是标识一个class是IoC容器的配置类（SpringBoot鼓励基于JavaConfig的配置形式）

@ComponentScan 动扫描并加载符合条件的组件（比如@Component和@Repository等）或者bean定义，最终将这些bean定义加载到IoC容器中。默认是从声明@ComponentScan所在类的package进行扫描。（也可通过参数自定义）

@EnableAutoConfiguration 借助@Import的帮助，将所有符合自动配置条件的bean定义加载到IoC容器

PS：Demo中的配置如下。把数据库配置的自动装载给去掉了。原因是：demo中用到了多数据源，用的是自定义的数据库配置

@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })

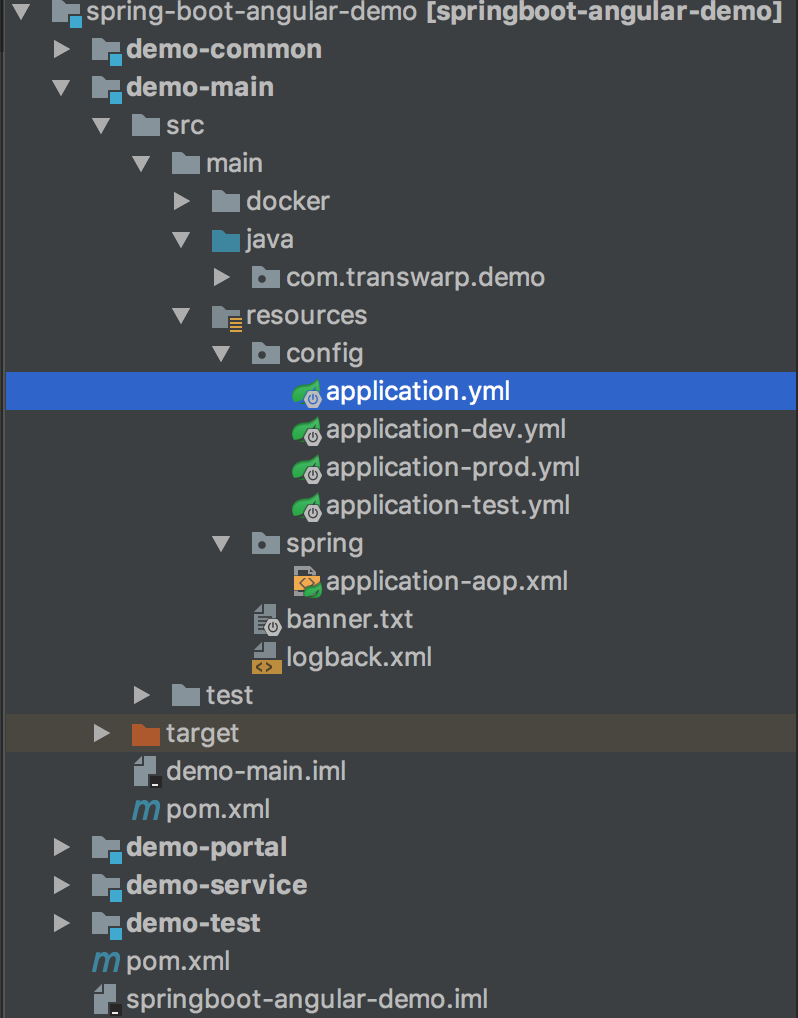
### 装载自定义的xml配置

如下，加上这个注解即可（按照springbot的理念（去xml化），此种方式不推荐）

@ImportResource(locations = "classpath:spring/application-aop.xml")

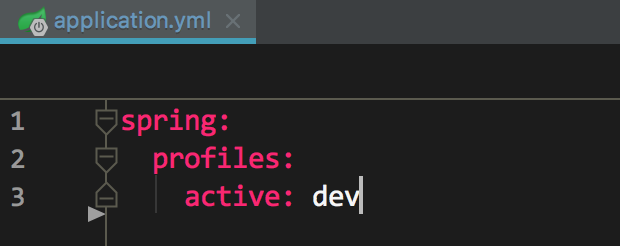
## SpringBoot应用配置文件

如下图，application.yml是SpringBoot应用启动加载的配置文件



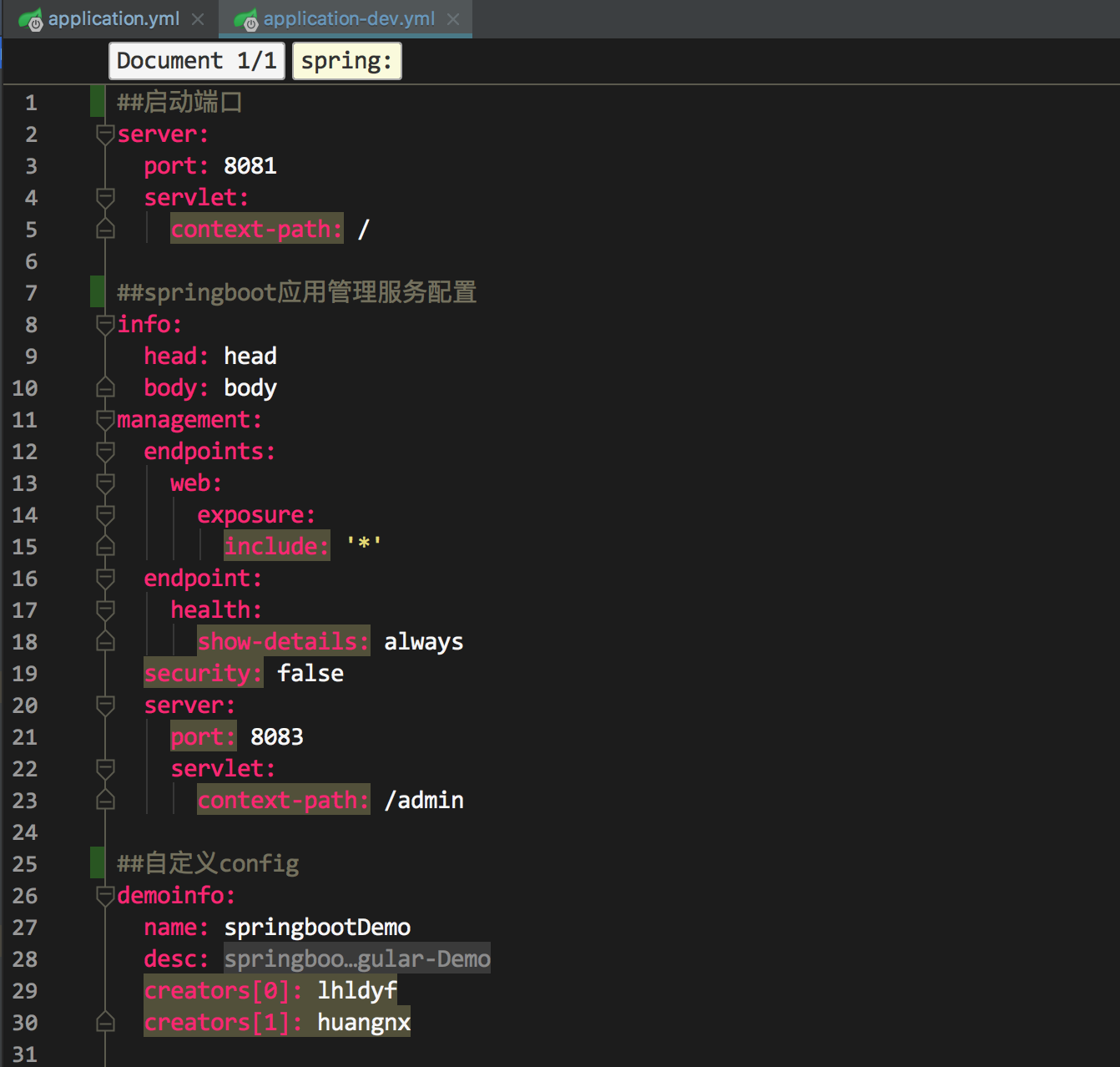
有两种properties的配置文件格式\*.properties; **.yml。这里采用的是**\*.yml的格式。

在demo中，利用profile做了不同环境propertis文件配置的分离



如图。选用的是dev环境的配置文件。Springboot应用启用时会去加载application-dev.yml的配置文件

接下来我们看看application-dev.yml配置文件的内容。和\*.properties配置文件相似，是key values结构的。不同的是该种配置方式支持树状结构，结构更清晰



另外配了logback，和普通的spring应用配起来没啥区别

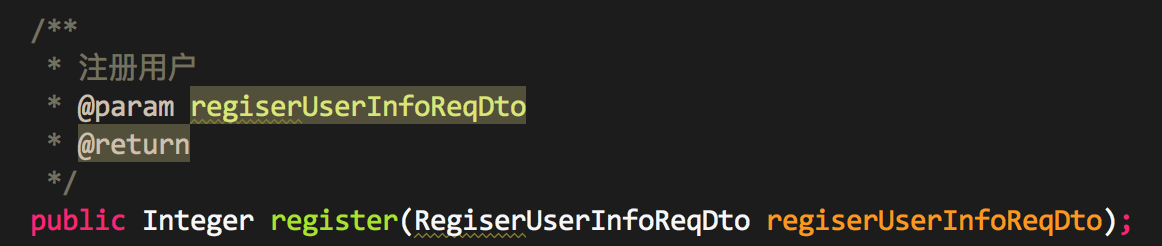
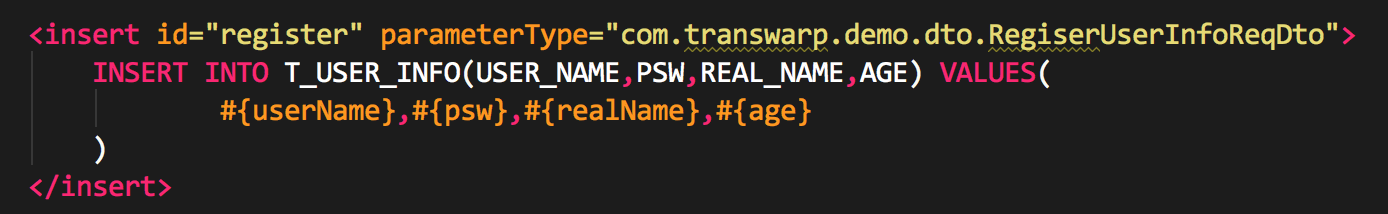
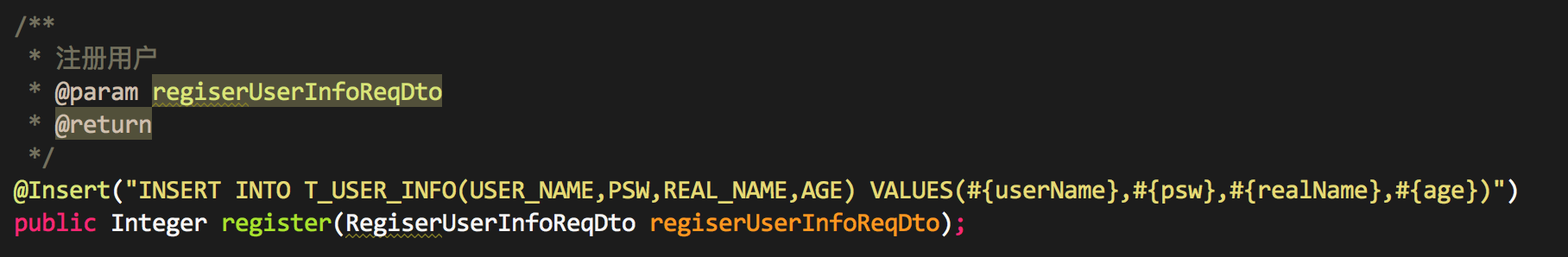
## demo-portal

没什么好讲的。和spring MVC差不多

如果是restful风格的接口，在controller class上加注解：@RestController。普通的接口，那还是@Controller

## demo-service

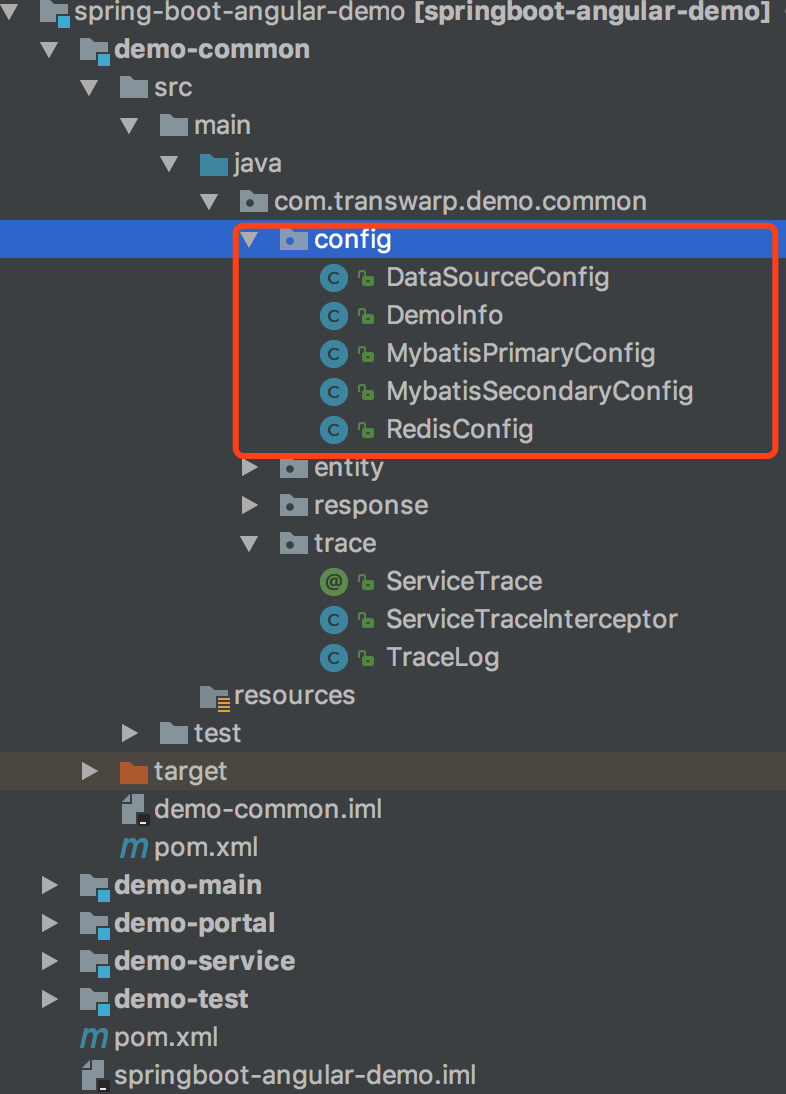
Dao层和普通的spring应用有些区别

* 可以选用mapper的方式使用mybatis
  + 
  + 
* 也可以使用注解的方式，在Java class中用接口的方式定义服务
  + 

PS：使用spring-data-jpa，只需要定义接口，都不需要写SQL（通过接口名适配），是很方便的。

## Demo-common

这里着重将一些config

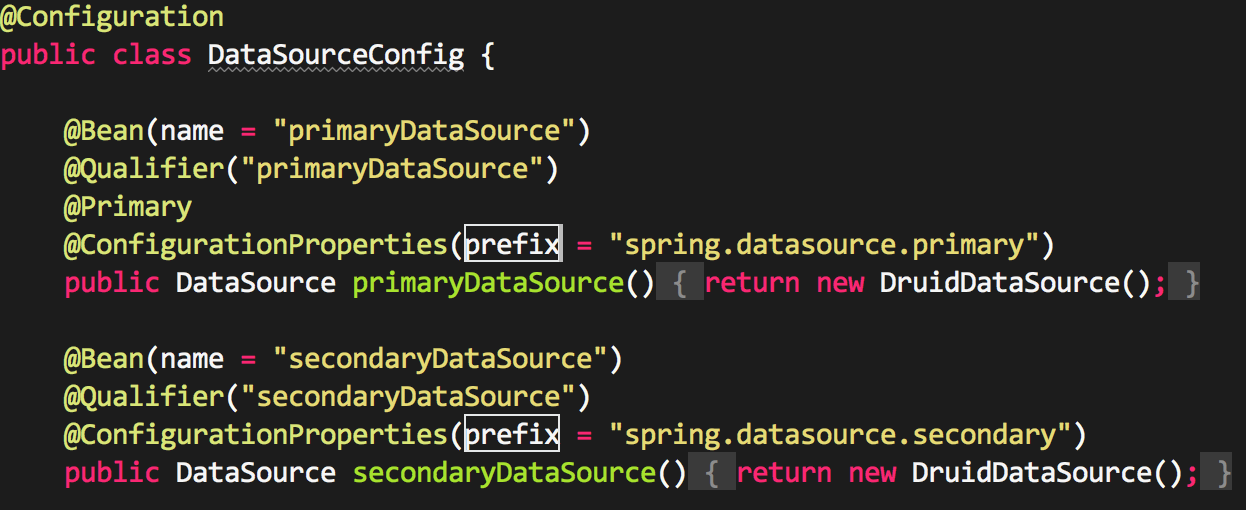


前面提到了。Springboot推荐使用JavaConfig的配置形式。Demo中配置了：

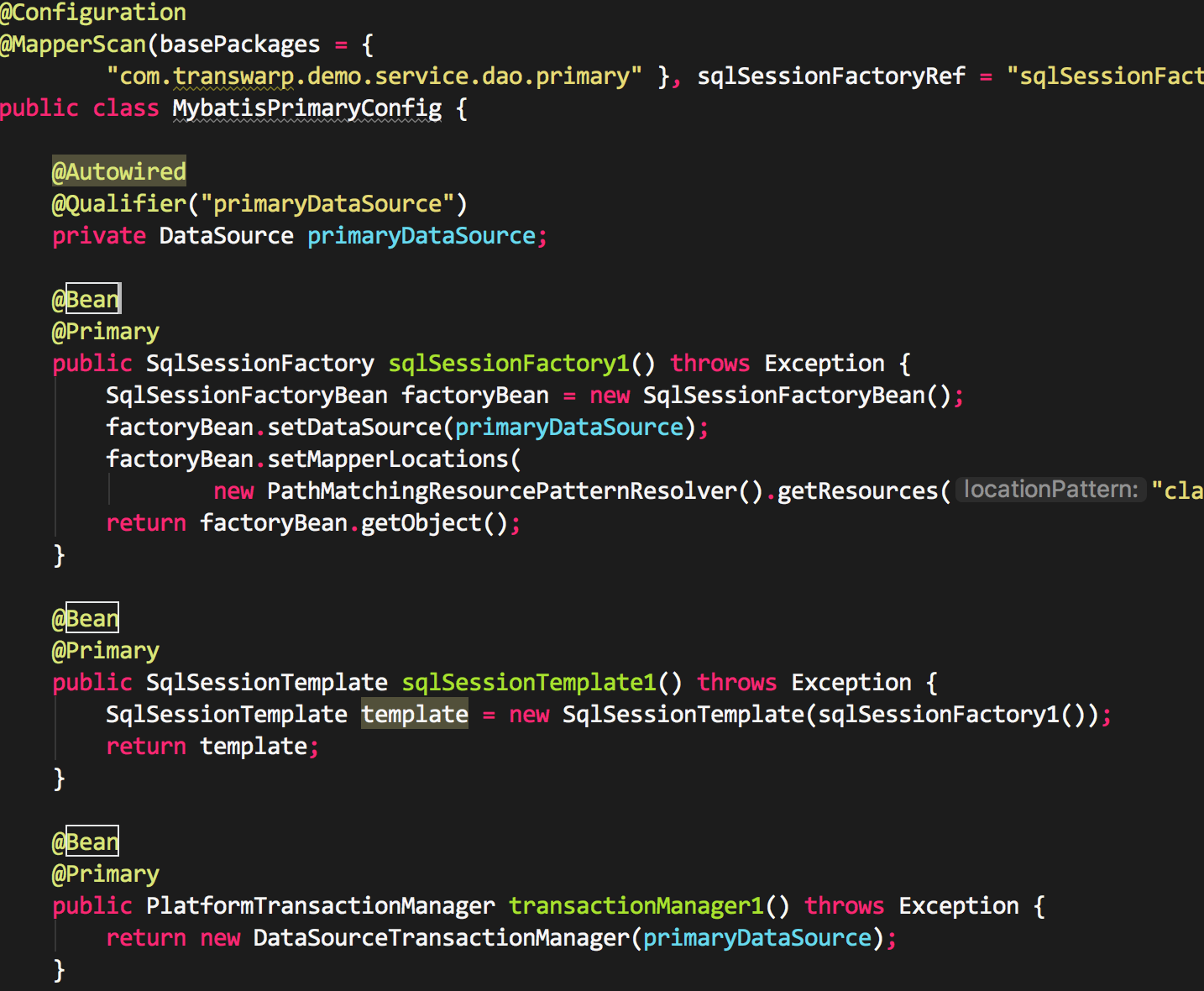
* 数据库配置（DataSourceConfig、MybatisPrimaryConfig、MybatisSecondaryConfig）
* Redis的配置：RedisConfig
* 自定义属性类：DemoInfo

#### 关于多数据源的配置

该类配置了两个数据库的DataSource的bean。通过@ConfigurationProperties注解注明加载的是application.yml中的哪个配置项



该类配置了主库SessioFactory、SessionTemplate、事务的配置。@MapperScan注解标明了包路径‘com.transwarp.demo.service.dao.primary’下的mapper走的是primary库。

同理另外一个配置类MybatisSecondaryConfig 是包路径‘com.transwarp.demo.service.dao.secondary’下的mapper走的是secondary 库

#### 关于自定义配置类

定义一个class与application.yml中的属性对应。然后class上加上如下注解。就可以通过@Autowired的方式使用它了

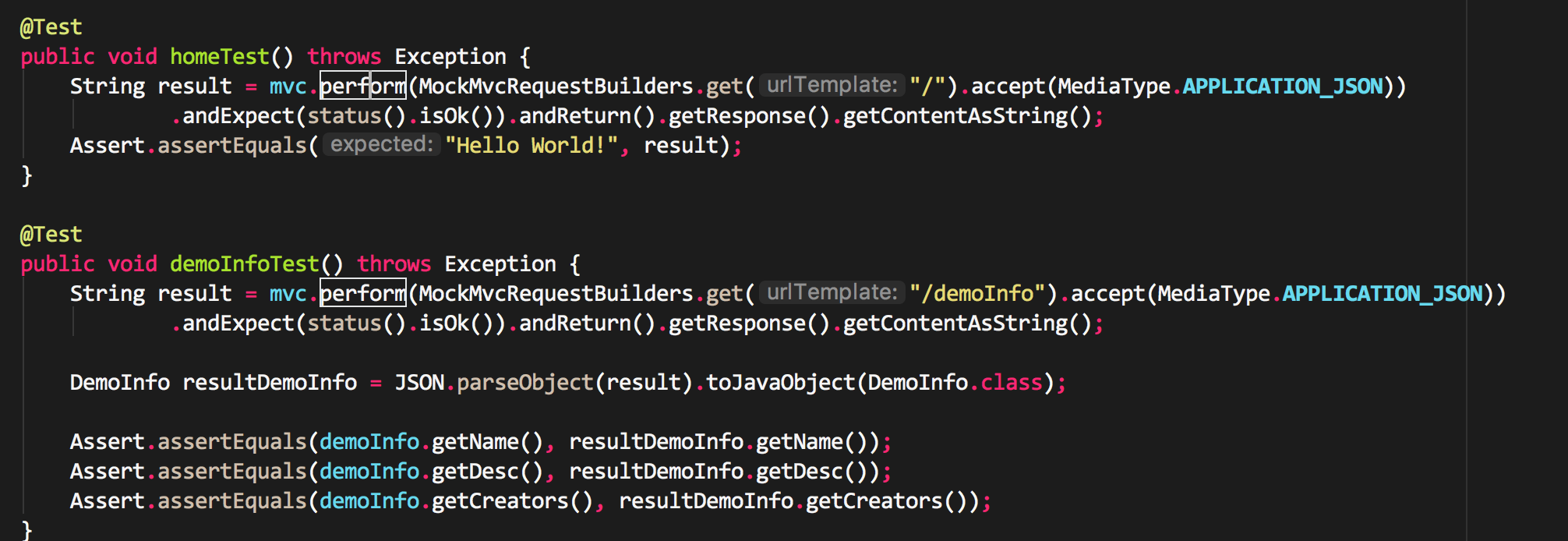


****

## Demo-test

使用起来和junitTest区别不大

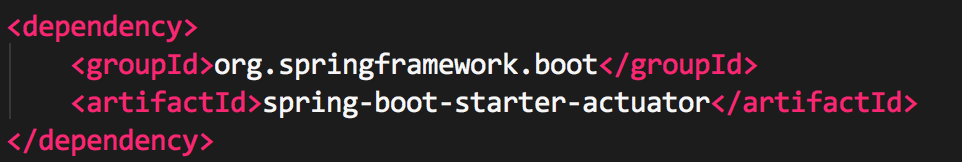
还可以test url



# SpringBoot应用监控

## Actuator监控

加上pom依赖



加上监控的配置

可以通过下述restful 接口查看监控信息

Actuator 提供了 13 个接口，具体如下表所示。

| **HTTP 方法** | **路径** | **描述** |
| --- | --- | --- |
| GET | /autoconfig | 提供了一份自动配置报告，记录哪些自动配置条件通过了，哪些没通过 |
| GET | /configprops | 描述配置属性(包含默认值)如何注入Bean |
| GET | /beans | 描述应用程序上下文里全部的Bean，以及它们的关系 |
| GET | /dump | 获取线程活动的快照 |
| GET | /env | 获取全部环境属性 |
| GET | /env/{name} | 根据名称获取特定的环境属性值 |
| GET | /health | 报告应用程序的健康指标，这些值由HealthIndicator的实现类提供 |
| GET | /info | 获取应用程序的定制信息，这些信息由info打头的属性提供 |
| GET | /mappings | 描述全部的URI路径，以及它们和控制器(包含Actuator端点)的映射关系 |
| GET | /metrics | 报告各种应用程序度量信息，比如内存用量和HTTP请求计数 |
| GET | /metrics/{name} | 报告指定名称的应用程序度量值 |
| POST | /shutdown | 关闭应用程序，要求endpoints.shutdown.enabled设置为true |
| GET | /trace | 提供基本的HTTP请求跟踪信息(时间戳、HTTP头等) |

## spring-boot-admin监控

参考：

<https://github.com/codecentric/spring-boot-admin>

<https://www.jianshu.com/p/e20a5f42a395>

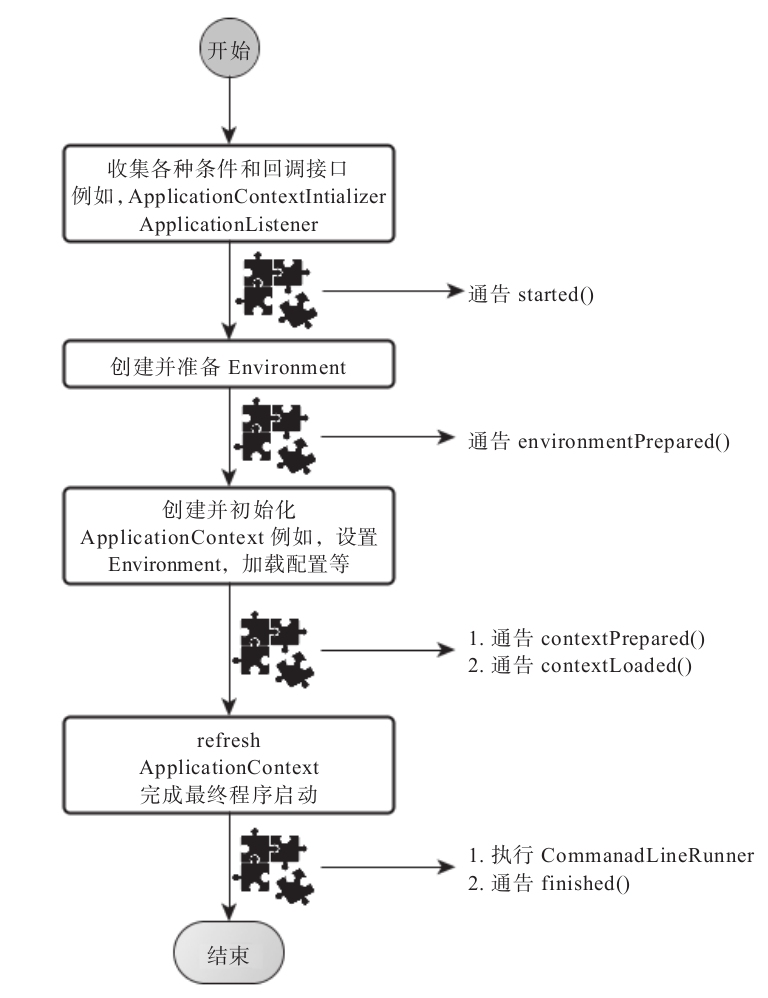
# SpringBoot的一些核心依赖

| **Name** | **Description** | **备注** |
| --- | --- | --- |
| spring-boot-starter-thymeleaf | 使MVC Web applications 支持Thymeleaf | Thymeleaf是一个JAVA库，一个XML/XHTML/HTML5的可扩展的模板引擎，同类事物：Jsp |
| spring-boot-starter-data-couchbase | 使用Couchbase 文件存储数据库、Spring Data Couchbase | Spring Data是一个用于简化数据库访问，并支持云服务的开源框架 |
| spring-boot-starter-artemis | 为JMS messaging使用Apache Artemis | JMS是Java消息服务；HornetQ代码库捐献给 Apache ActiveMQ 社区，它现在成为ActiveMQ旗下的一个子项目，名为 “Artemis” |
| spring-boot-starter-web-services | 使用Spring Web Services | Spring Web Services是基于Spring框架的Web服务框架，主要侧重于基于文档驱动的Web服务，提供SOAP服务开发，允许通过多种方式创建 Web 服务。 |
| spring-boot-starter-mail | 使用Java Mail、Spring email发送支持 | Java Mail、Spring email为邮件发送工具 |
| spring-boot-starter-data-redis | 通过Spring Data Redis 、Jedis client使用Redis键值存储数据库 | Jedis 是 Redis 官方首选的 Java 客户端开发包 |
| spring-boot-starter-web | 构建Web，包含RESTful风格框架SpringMVC和默认的嵌入式容器Tomcat | RESTful是一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件 |
| spring-boot-starter-activemq | 为JMS使用Apache ActiveMQ | ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线 |
| spring-boot-starter-data-elasticsearch | 使用Elasticsearch、analytics engine、Spring Data Elasticsearch | ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口 |
| spring-boot-starter-integration | 使用Spring Integration | Spring Integration是Spring框架创建的一个API，面向企业应用集成（EAI） |
| spring-boot-starter-test | 测试 Spring Boot applications包含JUnit、 Hamcrest、Mockito | JUnit、 Hamcrest、Mockito为测试框架 |
| spring-boot-starter-jdbc | 通过 Tomcat JDBC 连接池使用JDBC |  |
| spring-boot-starter-mobile | 通过Spring Mobile构建Web应用 | Spring Mobile 是 Spring MVC 的扩展,用来简化手机上的Web应用开发 |
| spring-boot-starter-validation | 通过Hibernate Validator使用 Java Bean Validation | Bean Validation 是一个数据验证的规范；Hibernate Validator是一个数据验证框架 |
| spring-boot-starter-hateoas | 使用Spring MVC、Spring HATEOAS构建 hypermedia-based RESTful Web 应用 | hypermedia-based似乎是专业术语，博主表示不会翻译；Spring HATEOAS 是一个用于支持实现超文本驱动的 REST Web 服务的开发库 |
| spring-boot-starter-jersey | 通过 JAX-RS、Jersey构建 RESTful web applications；spring-boot-starter-web的另一替代方案 | JAX-RS是JAVA EE6 引入的一个新技术；Jersey不仅仅是一个JAX-RS的参考实现，Jersey提供自己的API，其API继承自JAX-RS，提供更多的特性和功能以进一步简化RESTful service和客户端的开发 |
| spring-boot-starter-data-neo4j | 使用Neo4j图形数据库、Spring Data Neo4j | Neo4j是一个高性能的,NOSQL图形数据库，它将结构化数据存储在网络上而不是表中 |
| spring-boot-starter-websocket | 使用Spring WebSocket构建 WebSocket 应用 | Websocket是一个持久化的协议，相对于HTTP这种非持久的协议来说 |
| spring-boot-starter-aop | 通过Spring AOP、AspectJ面向切面编程 | AspectJ是一个面向切面的框架，它扩展了Java语言 |
| spring-boot-starter-amqp | 使用Spring AMQP、Rabbit MQ | Spring AMQP 是基于 Spring 框架的 AMQP 消息解决方案,提供模板化的发送和接收消息的抽象层,提供基于消息驱动的 POJO；RabbitMQ是一个在AMQP基础上完整的，可复用的企业消息系统 |
| spring-boot-starter-data-cassandra | 使用Cassandra分布式数据库、Spring Data Cassandra | Apache Cassandra是一套开源分布式NoSQL数据库系统 |
| spring-boot-starter-social-facebook | 使用 Spring Social Facebook | Facebook提供用户使用第三方社交网络的账号API，同类事物：QQ第三方登录接口 |
| spring-boot-starter-jta-atomikos | 为 JTA 使用 Atomikos | JTA，即Java Transaction API，JTA允许应用程序执行分布式事务处理；Atomikos 是一个为Java平台提供增值服务的并且开源类事务管理 |
| spring-boot-starter-security | 使用 Spring Security | Spring Security是一个能够为基于Spring的企业应用系统提供声明式的安全访问控制解决方案的安全框架 |
| spring-boot-starter-mustache | 使MVC Web applications 支持Mustache | Mustache是基于JavaScript实现的模版引擎，类似于jQuery Template，但是这个模版更加的轻量级，语法更加的简单易用，很容易上手 |
| spring-boot-starter-data-jpa | 通过 Hibernate 使用 Spring Data JPA （Spring-data-jpa依赖于Hibernate） | JPA全称Java Persistence API.JPA通过JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中 |
| spring-boot-starter | Core starter,包括 自动配置支持、 logging and YAML | logging是指的Starter的专有框架；YAML是“另一种标记语言”的外语缩写，它参考了其他多种语言，包括：XML、C语言、Python、Perl以及电子邮件格式RFC2822 |
| spring-boot-starter-groovy-templates | 使MVC Web applications 支持Groovy Templates | Groovy Templates是模视图模板，同类事物：Jsp |
| spring-boot-starter-freemarker | 使MVC Web applications 支持 FreeMarker | FreeMarker是模视图模板，同类事物：Jsp |
| spring-boot-starter-batch | 使用Spring Batch | Spring Batch是一个轻量级的,完全面向Spring的批处理框架,可以应用于企业级大量的数据处理系统 |
| spring-boot-starter-social-linkedin | 使用Spring Social LinkedIn | LinkedIn提供用户使用第三方社交网络的账号API，同类事物：QQ第三方登录接口 |
| spring-boot-starter-cache | 使用 Spring caching 支持 | Spring caching是Spring的提供的缓存框架 |
| spring-boot-starter-data-solr | 通过 Spring Data Solr 使用 Apache Solr | Apache Solr 是一个开源的搜索服务器。Solr 使用 Java 语言开发，主要基于 HTTP 和 Apache Lucene 实现 |
| spring-boot-starter-data-mongodb | 使用 MongoDB 文件存储数据库、Spring Data MongoDB | Spring Data是一个用于简化数据库访问，并支持云服务的开源框架 |
| spring-boot-starter-jooq | 使用JOOQ链接SQL数据库；spring-boot-starter-data-jpa、spring-boot-starter-jdbc的另一替代方案 | jOOQ（Java Object Oriented Querying，即面向Java对象查询）是一个高效地合并了复杂SQL、类型安全、源码生成、ActiveRecord、存储过程以及高级数据类型的Java API的类库。 |
| spring-boot-starter-jta-narayana | Spring Boot Narayana JTA Starter | 似乎和jboss.narayana.jta有关 |
| spring-boot-starter-cloud-connectors | 用连接简化的 Spring Cloud 连接器进行云服务就像Cloud Foundry、Heroku那样 | Cloud Foundry是VMware推出的业界第一个开源PaaS云平台；Heroku是一个支持多种编程语言的云平台即服务 |
| spring-boot-starter-jta-bitronix | 为JTA transactions 使用 Bitronix | Bitronix Transaction Manager (BTM) 是一个简单但完整实现了 JTA 1.1 API 的类库，完全支持 XA 事务管理器，提供 JTA API 所需的所有服务，并让代码保持简洁 |
| spring-boot-starter-social-twitter | 使用 Spring Social Twitter | Twitter提供用户使用第三方社交网络的账号API，同类事物：QQ第三方登录接口 |
| spring-boot-starter-data-rest | 使用Spring Data REST 以 REST 方式暴露 Spring Data repositories | 博主也不是很明白。原文：exposing Spring Data repositories over REST using Spring Data REST |

# 深入探索SpringApplication执行流程

SpringApplication的run方法的实现是我们本次旅程的主要线路，该方法的主要流程大体可以归纳如下：

1. 如果我们使用的是SpringApplication的静态run方法，那么，这个方法里面首先要创建一个SpringApplication对象实例，然后调用这个创建好的SpringApplication的实例方法。在SpringApplication实例初始化的时候，它会提前做几件事情：
   1. 根据classpath里面是否存在某个特征类（org.springframework.web.context.ConfigurableWebApplicationContext）来决定是否应该创建一个为Web应用使用的ApplicationContext类型。
   2. 使用SpringFactoriesLoader在应用的classpath中查找并加载所有可用的ApplicationContextInitializer。
   3. 使用SpringFactoriesLoader在应用的classpath中查找并加载所有可用的ApplicationListener。
   4. 推断并设置main方法的定义类。
2. SpringApplication实例初始化完成并且完成设置后，就开始执行run方法的逻辑了，方法执行伊始，首先遍历执行所有通过SpringFactoriesLoader可以查找到并加载的SpringApplicationRunListener。调用它们的started()方法，告诉这些SpringApplicationRunListener，“嘿，SpringBoot应用要开始执行咯！”。
3. 创建并配置当前Spring Boot应用将要使用的Environment（包括配置要使用的PropertySource以及Profile）。
4. 遍历调用所有SpringApplicationRunListener的environmentPrepared()的方法，告诉他们：“当前SpringBoot应用使用的Environment准备好了咯！”。
5. 如果SpringApplication的showBanner属性被设置为true，则打印banner。
6. 根据用户是否明确设置了applicationContextClass类型以及初始化阶段的推断结果，决定该为当前SpringBoot应用创建什么类型的ApplicationContext并创建完成，然后根据条件决定是否添加ShutdownHook，决定是否使用自定义的BeanNameGenerator，决定是否使用自定义的ResourceLoader，当然，最重要的，将之前准备好的Environment设置给创建好的ApplicationContext使用。
7. ApplicationContext创建好之后，SpringApplication会再次借助Spring-FactoriesLoader，查找并加载classpath中所有可用的ApplicationContext-Initializer，然后遍历调用这些ApplicationContextInitializer的initialize（applicationContext）方法来对已经创建好的ApplicationContext进行进一步的处理。
8. 遍历调用所有SpringApplicationRunListener的contextPrepared()方法。
9. 最核心的一步，将之前通过@EnableAutoConfiguration获取的所有配置以及其他形式的IoC容器配置加载到已经准备完毕的ApplicationContext。
10. 遍历调用所有SpringApplicationRunListener的contextLoaded()方法。
11. 调用ApplicationContext的refresh()方法，完成IoC容器可用的最后一道工序。
12. 查找当前ApplicationContext中是否注册有CommandLineRunner，如果有，则遍历执行它们。
13. 正常情况下，遍历执行SpringApplicationRunListener的finished()方法、（如果整个过程出现异常，则依然调用所有SpringApplicationRunListener的finished()方法，只不过这种情况下会将异常信息一并传入处理）  
    去除事件通知点后，整个流程如下：

[](http://7xqch5.com1.z0.glb.clouddn.com/springboot3-3.jpg)

# 附学习资源文档：

* SpringBoot各版本官方文档：
* <https://docs.spring.io/spring-boot/docs/>
* SpringBoot各种学习demo源码
  + <https://github.com/JeffLi1993/springboot-learning-example>