# 极速开发框架SpringBoot及源码分析

主讲：Cat 老师

**北京动力节点教育科技有限公司**  
**2020 • 北京**

**动力节点版权所有，禁止私自传播**

## 快速回顾

What is SpringBoot？

SpringBoot开发运行环境要求

springboot程序的几种创建方式

SpringBoot开发java程序

SpringBoot开发web程序

SpringBoot main方法分析

SpringBoot项目约定的代码结构

SpringBoot 项目pom.xml继承结构

SpringBoot 项目pom.xml修改默认版本

SpringBoot 整合mybatis

SpringBoot 整合jsp视图展示（也支持一些模板技术）

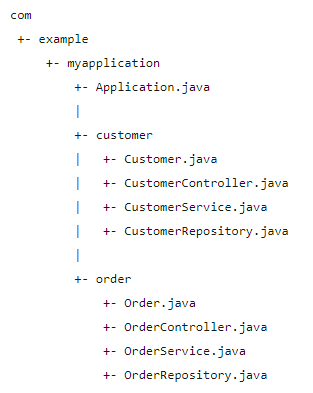
SpringBoot 项目pom.xml热部署插件devtools

JRebel （这是idea插件，收费的，需要破解）

## SpringBoot开发最佳实践

1、SpringBoot 程序，每个类应该有一个包，没有包的话会出现问题；

2、一个合理的包结构：



3、我们通常建议您将主应用程序类放在其他类之上的根包中，主类上需要有一个@SpringBootApplication注解，默认是扫描当前包及子类下的标有@Component, @Service, @Repository, @Controller等注解的bean；

4、如果main类上不使用@SpringBootApplication注解，可以使用@EnableAutoConfiguration、@ComponentScan、@Configuration 注解代替；

5、main方法的类也是一个**配置类，也是ioc容器中的一个bean**，Springboot不推荐使用xml配置方式；

6、还可以通过@Import导入其他bean对象（组件），或者通过@Configuration加入bean组件，或者@Configuration+@Bean，或者@Configuration+@ComponentScan加入bean对象（组件）；

7、也可以通过@ImportResource(locations = "classpath:applicationContext.xml")加入xml配置的bean对象（组件）；

8、禁止某些自动配置：@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})

或者在配置文件中 spring.autoconfigure.exclude= 进行排除，可以达到优化程序的效果；

9、springboot内嵌了web服务器，所以打成一个jar包即可运行，方便我们的开发和调试；

10、延迟初始化bean：spring.main.lazy-initialization=true

ApplicationContext context = new SpringApplicationBuilder()  
 .lazyInitialization(true)  
 .sources(com.bjpowernode.Application.class)  
 .bannerMode(Banner.Mode.*CONSOLE*)  
 .run(args);

SpringApplication springApplication = new SpringApplication(Application.class);  
springApplication.setLazyInitialization(true);

如果要禁用某些bean的延迟初始化，则可以使用@Lazy(value=false)注解将它们的延迟属性显式设置为false；

SpringApplication启用延迟初始化后，将根据需要创建bean，而不是在应用程序启动期间创建bean,因此启用延迟初始化可以减少启动应用程序所需的时间。在Web应用程序中，启用延迟初始化将导致许多与Web相关的Bean直到收到HTTP请求后才被初始化。延迟初始化的缺点是，它可能会延迟发现应用程序的问题。如果错误地配置了Bean的延迟初始化，则可能导致启动时没有问题，在访问调用的时候出现问题，另外还必须注意确保JVM有足够的内存来容纳所有应用程序的bean，而不仅仅是启动过程中初始化的bean，由于这些原因，默认情况下SpringBoot不会开启延迟初始化；

11、CommandLineRunner, ApplicationRunner在springboot启动后回调着两个接口里面的run方法；

## SpringBoot敏感配置信息加密

配置信息加密主要利用了<http://www.jasypt.org/> 这个类库，Jasypt是一个Java库，允许开发人员以最小的努力将基本的加解密功能添加到你的项目中，而无需对密码加解密的工作原理有深入的了解；

Github：<https://github.com/jasypt/jasypt> 这个库是java语言写的；

不过直接使用上面这个类库进行开发稍微有些麻烦，所以在SpringBoot中有人对Jasypt制作了一个starter依赖，方便开发，maven依赖如下：

*<!-- jasypt-spring-boot-starter -->*<dependency>  
 <groupId>com.github.ulisesbocchio</groupId>  
 <artifactId>jasypt-spring-boot-starter</artifactId>  
 <version>3.0.3</version>  
</dependency>

加解密代码

@SpringBootTest  
class ApplicationTests {  
 @Autowired  
 private StringEncryptor stringEncryptor;  
 @Test  
 public void encryptPwd() {  
 *//加密* String username = stringEncryptor.encrypt("root");  
 System.*out*.println("加密username: " + username);  
  
 String decUsername = stringEncryptor.decrypt(username);  
 System.*out*.println("解密username: " + decUsername);  
  
 *//加密* String password = stringEncryptor.encrypt("123456");  
 System.*out*.println("password: " + password);  
 String decPassword = stringEncryptor.decrypt(password);  
 System.*out*.println("解密password: " + decPassword);  
 }

}

## SpringBoot中的数据源

SpringBoot默认采用的是**HikariCP**数据库连接池，该数据库连接池性能极好；

若要更换为阿里巴巴Druid数据库连接池，首先把HikariCP的依赖排除掉：

<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>2.1.3</version>  
 <exclusions>  
 *<!--排除掉HikariCP数据库连接池-->* <exclusion>  
 <groupId>com.zaxxer</groupId>  
 <artifactId>HikariCP</artifactId>  
 </exclusion>  
 </exclusions>  
</dependency>

然后添加druid依赖

*<!-- druid -->*<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.1.23</version>  
</dependency>

然后需要在配置文件中指定：

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

另外一种更简洁的方式是通过starter更换为Druid数据库连接池，直接加一个依赖即可：

*<!-- druid-spring-boot-starter -->*<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid-spring-boot-starter</artifactId>  
 <version>1.1.23</version>  
</dependency>

除此之外还有其他一些数据库连接池：

DBCP

DBCP2

Tomcat JDBC

C3P0

等等

## SpringBoot跨域资源共享CORS支持

Spring 4.2.x版本开始支持CORS

我们知道，传统的Ajax请求只能获取在同一个域名下的资源，但是HTML5规范中打破了这种限制，允许Ajax发起跨域的请求；

浏览器本身是可以发起跨域请求的，比如你可以链接一个外域的图片或者脚本，但是javascript脚本是不能获取这些外域资源内容的；

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing），它允许浏览器向跨源服务器发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制；

这种CORS使用了一个额外的HTTP响应头来赋予当前user-agent（浏览器）获得非同源资源的权限，这里的非同源就是Cross-Origin的概念，这里的权限就是访问非同源的资源权限；

CORS是现在HTML5标准中的一部分，在大部分现代浏览器中都有所支持，可能在某些老版本的浏览器不支持CORS，如果要兼容一些老的浏览器版本，则需要采用JSONP进行跨域请求；

## 同源与非同源的定义

如果 **访问协议、端口（如果指定了端口）、host**都相同，则称之为同源，否则为非同源；

比如源链接： <http://store.company.com/dir/page.html>

| **URL** | **是否同源** | **原因** |
| --- | --- | --- |
| <http://store.company.com/dir2/other.html> | 是 |  |
| <http://store.company.com/dir/inner/another.html> | 是 |  |
| <https://store.company.com/secure.html> | 否 | 协议不同 |
| <http://store.company.com:81/dir/etc.html> | 否 | 端口不同 |
| <http://news.company.com/dir/other.html> | 否 | host不同 |

SpringBoot解决跨域问题，只需要配置如下代码即可：

@Bean  
public WebMvcConfigurer corsConfigurer() {  
 return new WebMvcConfigurer() {  
 @Override  
 public void addCorsMappings(CorsRegistry registry) {  
 registry.addMapping("/web/\*\*");  
 registry.addMapping("/boot/\*\*");  
 }  
 };  
}

JSONP 是 JSON with padding（填充式 JSON 或参数式 JSON）的简写；

JSONP实现跨域请求的原理简单来说，就是动态创建<script>标签，然后利用<script>的src 不受同源策略约束来跨域获取数据；

JSONP 由两部分组成：回调函数和数据，回调函数是当响应到来时应该在页面中调用的函数，回调函数的名字一般是在请求中指定的，而数据就是传入回调函数中的 JSON 数据，例如；

$.ajax({  
 type:"GET",  
 url:"http://192.168.0.104:8080/boot/test",  
 dataType : "jsonp", *//返回的数据类型，设置为JSONP方式* jsonp : 'callback', *//覆盖默认的 jsonp回调参数名callback* jsonpCallback: 'handleResponse', *//设置回调函数名* success: function(response, status, xhr){  
 console.log('状态为：' + status + ',状态是：' + xhr.statusText);  
 console.log(response);  
 },  
 error: function (XMLHttpRequest, textStatus, errorThrown) {  
 *// 状态码* console.log(XMLHttpRequest.status);  
 *// 状态* console.log(XMLHttpRequest.readyState);  
 *// 错误信息* console.log(textStatus);  
 console.log(errorThrown);  
 }  
 });  
});  
function *handleResponse*(response){  
 console.log(response);  
}

## CORS与JSONP比较

CORS与JSONP都是为了跨域，但是CORS比JSONP更强大；

JSONP只支持GET请求，CORS支持所有类型的HTTP请求；

JSONP的优势在于支持老式浏览器；

## SpringBoot静态资源处理

静态资源：js, css, html, 图片，音视频等；

静态资源路径：是指系统可以直接访问的路径，且路径下的所有文件均可被用户直接访问；

Spring Boot默认静态资源目录位置位于classpath下，目录名需符合如下规则：

有4个目录下的静态资源可以直接访问：

/static

/public

/resources

/META-INF/resources

在SpringBoot的源码中有定义：

private static final String[] *CLASSPATH\_RESOURCE\_LOCATIONS* = {

"classpath:/META-INF/resources/",  
"classpath:/resources/",

"classpath:/static/",

"classpath:/public/"

};

可以通过spring.resources.static-locations=classpath:/mysource/进行修改；

## SpringBoot核心配置文件

1、.properties文件

键值对的properties属性文件配置方式

SpringBoot默认读取该文件作为项目配置文件

2、.yml文件

yml 也是一种配置文件格式，主要采用空格、换行、冒号等格式排版进行配置；

yml 后缀也可以使用 yaml 后缀；

配置的值与前面的冒号必须要有一个空格；

只要在类路径上具有SnakeYAML的jar包，SpringApplication类就会自动支持YAML作为配置文件，默认的spring-boot-starter自动依赖了SnakeYAML；

具体使用可以参考一下视频演示；

3、多环境配置文件

#比如配置测试环境

spring.profiles.active=dev

application-dev.properties

#比如配置生产环境

spring.profiles.active=product

application-product.properties

配置随机值，通过配置文件得到一个随机值，springboot提供RandomValuePropertySource可用于注入随机值（例如，注入一些随机值或测试用例），它可以产生整数，longs，uuid或字符串，如下示例所示：

my.secret=${random.value}  
my.number=${random.int}  
my.bignumber=${random.long}  
my.uuid=${random.uuid}  
my.number.less.than.ten=${random.int(10)}  
my.number.in.range=${random.int[1024,65536]}

配置文件中可以使用占位符：

my.name = cat  
my.desc = ${my.name} is a teacher.