**兼容 Spring Boot 1.x 和 2.x 配置类参数绑定的工具类 SpringBootBin**

为了让我提供的通用 Mapper 的 boot-s [tar](http://www.liuhaihua.cn/archives/tag/tar) ter 同时兼容 [Spring Boot](http://www.liuhaihua.cn/archives/tag/spring-boot) 1.x 和 2.x，增加了这么一个工具类。

在 Spring Boot 中，能够直接注入 XXProperties 类的地方不需要使用这个工具类。

但是在 Spring 的接口和启动流程设计中，有些情况下只能通过 EnvironmentAware   
接口得到 Environment   
对象，此时你想得到 XXProperties 类没有更好的办法。

也许有人直接从 Environment   
对象中 [遍历](http://www.liuhaihua.cn/archives/tag/%e9%81%8d%e5%8e%86) 获取所有的 [配置](http://www.liuhaihua.cn/archives/tag/%e9%85%8d%e7%bd%ae) 信息，但是有一个无法完美解决的问题就是 relax 值，例如 first-name   
， firstName   
, FIRST\_NAME   
都可以代表同一个 [参数](http://www.liuhaihua.cn/archives/tag/%e5%8f%82%e6%95%b0) ，在自己 [代码](http://www.liuhaihua.cn/archives/tag/%e4%bb%a3%e7%a0%81) 中很难处理这种情况。

通用 Mapper 在兼容两者过程中遇到过很多 BUG，这一次通过一个工具类解决了这个问题。

在 Spring Boot 1.x 中，可以通过下面代码绑定参数到对象：

try {

RelaxedPropertyResolver resolver = new RelaxedPropertyResolver(environment);

Map<String, Object> properties = resolver.getSubProperties("");

//targetClass 目标类型，例如 MapperProperties

T target = targetClass.newInstance();

RelaxedDataBinder binder = new RelaxedDataBinder(target, prefix);

binder.bind(new MutablePropertyValues(properties));

return target;

} catch (Exception e) {

throw new RuntimeException(e);

}

Spring Boot 2.x 中，绑定更简单，如下：

Binder binder = Binder.get(environment);

return binder.bind(prefix, targetClass).get();

上面这两段代码也是最近才找到，要不然这个功能会出现的更早。

由于上面的两处代码都在 [spring](http://www.liuhaihua.cn/archives/tag/spring) -boot.jar 中，因此 [编译](http://www.liuhaihua.cn/archives/tag/compile) 时不能同时依赖两个不同的版本，而且为了方便以后项目依赖从 1.x 升级到 2.x，因此针对上面两处代码全部使用反射实现。

[源码](http://www.liuhaihua.cn/archives/tag/nativecode) 地址： [https](http://www.liuhaihua.cn/archives/tag/https) :// [git](http://www.liuhaihua.cn/archives/tag/git) hub.com/abel533/ [mapper](http://www.liuhaihua.cn/archives/tag/mapper) -boot-starter/blob/master/ [map](http://www.liuhaihua.cn/archives/tag/map) per-spring-boot-autoconfigure/ [src](http://www.liuhaihua.cn/archives/tag/src) /main/ [java](http://www.liuhaihua.cn/archives/tag/java-2) /tk/ [mybatis](http://www.liuhaihua.cn/archives/tag/mybatis) /spring/mapper/SpringBootBindUtil.java

简单用法如下：

MapperProperties mapperProperties = SpringBootBindUtil.bind(

environment,

MapperProperties.class,

MapperProperties.PREFIX);

至此通过 environment 就能得到想要的配置类了。

*原文*

[http](http://www.liuhaihua.cn/archives/tag/http) ://blog.csdn.net/isea533/article/details/79121981

有差别，但差别不大。基本上基于SpringBoot的代码不需要改动，但有些配置属性和配置类，可能要改动，改动原因是

* 配置已经不存在或者改名
* 类已经不存在改名

听着挺吓人，但我实际切换过程中改动的地方很少。一般正常的MVC，数据库访问这些都不需要改动,下面按照本书章节说明我曾碰到的区别

* 第1章，SpringBoot 2基于Spring5和JDK8，而Spring 1x则用的是降低版本，带来的为可能是你的应用服务必须支持JDK8
* 第2章，无区别，使用SpringBoo2，建议使用较新的Maven版本，以及较新的JDK,尤其是IDE工具，以免Maven在IDE里的视图报出警告信息
* 第3章：MVC部分，有些定制类改动了，比如WebMvcConfiguer,由抽象类改为接口，这是因为JDK8对接口有新的支持形式，3.8章提到的统一错误处理，基类AbstarctErrorController也改动非常大，请参考书中描述的知识点。MVC里的视图渲染Freemaker视图解析器也有改动，默认情况下，它会自动加上ftl来来寻找模板
* 第4章：关于Beetl，无改动，请使用新版本即可
* 第5章，无改动，JDBCTemplate和BeetlSQL均可以使用
* 第6章，JPA中，findById 返回了一个Optional对象，**改动较大，会直接影响所有业务代码**
* 第7章，SpringBoot配置，web应用的ContextPath 配置属性已经改动，可以参考本文章的7.1.1 说明。另外配置文件的中文可以直接读取，而不需要转码。另外，自动装配里Boot提供的JavaVersion类报名改动了。根据jdk版本自动装配的需要调整代码
* 第8章，部署SpringBoot无改动
* 第9章，测试SpringBoot无改动
* 第10章，无改动，但Sagger UI变化较大。除了参考本书外，还需要密切注意Sawgger的最新特性
* 第11章， MongDB，无变化
* 第12章，Redis 无变化
* 第13章，ElasticSearch，无改动，应该说SpringBoot2做的更好了。需要密切注意Elastic Search本身版本变化，它也是版本帝，Spring Boot略有点根本上。因此不能担保在因为Elastci Search的RESTAPI变化导致Spring Boot不兼容，所以一定要使用Spring Boot指定的版本
* 第14章,基本无改动，但内部CacheManager和Cache的接口改动，因此缓存机制内部变化交大，书里提到的一二级分布式缓存实现方式变化就比较大
* 第15章，SpringSession 无变化
* 第16章，由于本书将的是用Curator集成Spring Boot，所以无变化
* 第17章，Acutator，本章SpringBoot1.x和2变化较大，默认情况，不再启用所有监控，另外编写自己监控信息，完全需要重写，HealthIndicator,EndPoint 变化很大

# spring boot框架出现java.lang.IllegalArgumentException: jdbcUrl is required with driverClassName. 错误

在搭建spring boot框架是出现了如下错误

在网上花费大半天的时间都没有找到方法解决。最后经过一番查找找到了问题的关键所在。

因为之前我的spring boot使用的是

<version>1.5.7.RELEASE</version>

1.5.7的版本，我把之前的配置文件是直接粘贴拷贝过来的，而我新建的工程使用的版本是

<version>2.0.2.RELEASE</version>

因为不同版本之间配置文件的差异还是很大的。

在1.5.7版本中application.properties文件的配置是

**spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.url**=**jdbc:mysql://localhost:3306/bian  
spring.datasource.username**=**root  
spring.datasource.password**=**ls0611**

而在2.0.2版本中需要在datasource后面加上hikari；具体配置如下：

**spring.datasource.hikari.bian.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.hikari.bian.jdbc-url**=**jdbc:mysql://localhost:3306/bian  
spring.datasource.hikari.bian.username**=**root  
spring.datasource.hikari.bian.password**=**ls0611**

就这样给解决了！

## Spring Boot 2.0 新特性和发展方向

**以Java 8 为基准**

Spring Boot 2.0 要求Java 版本必须8以上， Java 6 和 7 不再支持。

**内嵌容器包结构调整**

为了支持reactive使用场景，内嵌的容器包结构被重构了的幅度有点大。EmbeddedServletContainer被重命名为WebServer，并且org.springframework.boot.context.embedded 包被重定向到了org.springframework.boot.web.embedded包下。举个例子，如果你要使用TomcatEmbeddedServletContainerFactory回调接口来自定义内嵌Tomcat容器，你现在应该使用TomcatServletWebServerFactory。

**Servlet-specific 的server properties调整**

大量的Servlet专属的server.\* properties被移到了server.servlet下：



由此可以看出一些端倪，那就是server不再是只有servlet了，还有其他的要加入。

**Actuator 默认映射**

Actuator的端点（endpoint）现在默认映射到/application，比如，/info 端点现在就是在/application/info。但你可以使用management.context-path来覆盖此默认值。

**@ConditionalOnBean**

@ConditionalOnBean现在的判断条件由OR变为了AND。

**Remote CRaSH shell**

1.5的时候此远程工具被标为deprecated，2.0的时候将会把这个支持彻底remove掉，以及基于此功能的项目也会被删除掉。

**Spring Loaded不再支持**

由于Spring Loaded项目已被移到了attic了，所以不再支持Spring Loaded了。现在建议你去使用Devtools。Spring Loaded不再支持了。

**Hazelcast自动配置不再支持**

不能再自动配置HazelcastInstance进行缓存。 因此，spring.cache.hazelcast.config属性不再可用。

**默认connection pool变了**

默认的连接池已经由Tomcat切换到了HikariCP。如果你过去使用spring.datasource.type在基于Tomcat的应用程序中强制使用Hikari，现在你可以删除这个覆盖了。同样的，如果想要使用Tomcat的连接池，只需要简单的加入以下配置就可以了：

spring.datasource.type=org.apache.tomcat.jdbc.pool.DataSource

**Servlet Filters**

针对一个Filter默认的dispatcher类型现在为 DipatcherType.REQUEST，这样就保证了Spring Boot和Servlet默认的配置统一了。

**Spring Security**

Spring Security的 filter 现在可以自动配置如下类型了： ASYNC, ERROR, and REQUEST 。这样就让Spring Boot中的配置和Spring Security默认配置保持一致了。

**Spring Session**

Spring Session的 filter 现在支持自动配置如下dispatcher类型 ASYNC, ERROR, and REQUEST 。同样是为了让Spring Boot的配置与Spring Session的默认配置保持一致。值得注意的是从Spring Session 2.0起, Mongo和GemFire 将不再被支持。

**Jetty**

要求Jetty最低版本为9.4。

**Tomcat**

要求Tomcat最低版本为8.5。

**Hibernate**

要求Hibernate最低版本为5.2。

**Gradle**

要求Gradle最低版本为3.4。

**SendGrid**

SendGrid最低支持版本是3.2。为了支持这次升级，username和password已经被干掉了。因为API key现在是唯一支持的认证方式。

**Starter 的传递依赖的改变**

以前有几个Spring Boot starter是依靠Spring MVC和spring-boot-starter-web传递的。 为了对Spring WebFlux的支持，spring-boot-starter-mustache和spring-boot-starter-thymeleaf不再依赖spring-boot-starter-web。现在你要自己选择并添加spring-boot-starter-web或spring-boot-starter-webflux作为依赖。

**记住：**web和webflux是平行的关系。我们之前的文章中提到过spring webflux。你可以移步查看：[Spring 5 新增全新的reactive web框架：webflux](http://mp.weixin.qq.com/s?__biz=MzA5MzQ2NTY0OA==&mid=2650796733&idx=1&sn=b54cd0c3e18a6fcf04821047e22f3119&chksm=88562fe2bf21a6f4debef4ba31df2629f83a83e481aee0883e648776056028ad54e8d8269017&scene=21#wechat_redirect)。

**Solr健康指标**

针对Solr的健康检查信息不再是solrStatus这个属性了。现在是status property，是一个标准状态属性，对应于一个的整型值。

**默认代理策略**

Spring Boot现在默认是使用CGLIB代理，同时包含AOP支持。如果你需要基于代理（proxy-based）的代理策略，你需要把spring.aop.proxy-target-class设置为false。

**基于CLI的测试**

从Spring Boot的CLI中删除了测试支持，现在建议你使用Maven或Gradle构建应用程序，去使用它们提供的丰富的测试支持吧。

**@ConfigurationProperties**

@ConfigurationProperties里的ignoreNestedProperties属性已被删除。

**Multipart配置改变**

为了更好的反映Servlet的特性,  spring.http.multipart. 属性已经被命名为 spring.servlet.multipart。

ps：其实就是因为webflux来了，只能把servlet和webflux区分开来。

**Mustache 模板默认文件扩展名**

过去Mustache模板的默认的文件扩展名是.html。现在.mustache成了官方指定的扩展名，而且很多的IDE插件已经支持了此后缀。你可以通过spring.mustache.suffix来覆盖现在的默认的支持。

**Spring Framework 5.0**

Spring Boot 2.0 是建立在Spring Framework 5.0之上的（最低要求）。Spring 5最大的亮点就是reactive。你可以去看看这篇：[Spring 5 新增全新的reactive web框架：webflux](http://mp.weixin.qq.com/s?__biz=MzA5MzQ2NTY0OA==&mid=2650796733&idx=1&sn=b54cd0c3e18a6fcf04821047e22f3119&chksm=88562fe2bf21a6f4debef4ba31df2629f83a83e481aee0883e648776056028ad54e8d8269017&scene=21#wechat_redirect)。

**松绑定改善**

松绑定已经得到改善。这里简单的举个例子：

简单的属性在删除特殊字符然后转成小写后进行绑定。比如：

下面的属性最终都会被映射为spring.jpa.databaseplatform=mysql:

spring.jpa.database-platform=mysql

spring.jpa.databasePlatform=mysql

spring.JPA.database\_platform=mysql

有关详细信息，我们可以查阅：https://github.com/spring-projects/spring-boot/wiki/Relaxed-Binding-2.0

**WebFlux 和 WebFlux.fn支持**

Spring Boot 2.0 提供了一个新的starter ，用来支持Reactive Spring web frameworks。该starter为spring-boot-starter-webflux。其中Reactor Netty是默认的web引擎(spring-boot-starter-reactor-netty)。

**Reactive数据库支持**

Spring Boot 2.0 对一下的数据库提供了自动配置（ auto-configuration ）的reactive的支持：

* MongoDB (spring-boot-starter-data-mongodb-reactive)
* Redis (spring-boot-starter-data-redis-reactive)
* Cassandra (spring-boot-starter-data-cassandra-reactive)

@DataMongoTest测试注解也是支持reactive的。

**@WebFluxTest支持**

Reactive controller现在可以使用@WebFluxTest来测试。它提供的功能和 @WebMvcTest类似。而且WebTestClient可以直接使用，已经被自动配置（auto-configured）了。

**支持使用@SpringBootTest自动配置WebTestClient**

当你使用@SpringBootTest with an actual server (that is, either DEFINED\_PORT or RANDOM\_PORT), a WebTestClient is available the same way TestRestTemplate is.

使用@SpringBootTest进行WebTestClient自动配置

将@SpringBootTest用于实际服务器（即DEFINED\_PORT或RANDOM\_PORT）时，WebTestClient的可用方式与TestRestTemplate相同。

**Gradle插件**

Spring Boot的Gradle插件已在很大程度上被重写，以实现一些重大的改进。有关这方面的就不细讲了，感兴趣的同学可以去一探究竟。

**/loggers端点POST的状态码改变**

针对/loggers端点上POST操作的状态码已又200改为204。

**Elasticsearch**

Elasticsearch已升级到5.4。 与Elastic公司宣布不再支持嵌入式Elasticsearch的情况一样，NodeClient的自动配置已被删除。就是这么的节奏一致。现在你可以通过使用spring.data.elasticsearch.cluster-nodes自动配置TransportClient，具体value可以是一个或多个要连接的节点的地址。

**Quartz Scheduler**

Spring Boot 2 针对Quartz调度器提供了支持。你可以加入spring-boot-starter-quartz starter来启用。而且支持基于内存和基于jdbc两种存储。

**Spring Data Web配置**

Spring Boot公开了一个新的spring.data.web配置名称空间，可以轻松配置分页和排序。

**Json starter**

一个全新的spring-boot-starter-json starter 聚合了很多常用的json工具，可以支持对json的读写。

它不仅仅提供了jackson-databind而且当你使用java8时候，还提供了有用的模块：jackson-datatype-jdk8, jackson-datatype-jsr310 和 jackson-module-parameter-names。

之前使用jackson-databind的地方就可以使用这个新的starter了。

**Thymeleaf starter**

Thymeleaf starter现在包含了thymeleaf-extras-java8time，开箱即用。

**InfluxDB**

如果InfluxDB java client 和 the spring.influx.url 被设置, 一个InfluxDB client现在就会被自动配置。而且现在支持认证。

**JdbcTemplate**

Spring Boot自动配置（auto-configuration）的JdbcTemplate现在可以通过spring.jdbc.template命名空间进行定制。 此外，自动配置（auto-configuration）的NamedParameterJdbcTemplate在底层就是JdbcTemplate。

**jOOQ**

Spring Boot现在可以根据DataSource自动检测出jOOQ方言（类似于JPA方言）。

此外，还引入了@JooqTest用于只有jOOQ才能hold的测试场景。

**@DataRedisTest**

新加了一个测试的时候针对redis的新注解。

**强大的Mongo客户端自定义**

现在可以通过MongoClientSettingsBuilderCustomizer这个bean来高度定制化Spring Boot的Mongo客户端。

**Cassandra**

spring.data.cassandra 现在支持池化（pooling）。

**Kafka listener支持批量消费**

现在支持一次性批量消费多个ConsumerRecord，你可以创建一批的监听器（listener），这样设置：

spring.kafka.listener.type=batch

**Web filters 初始化**

Web filters 现在在所有支持的容器中都会被立刻初始化。也就是eagerly，急加载。

**Auto-configuration 报告**

现在不满足条件（unconditional）的class也会被包含进来，在自动配置（auto-configuration）的Actuator端点的response中一并返回。

**重置logger操作**

现在Loggers端点（endpoint） 支持reset日志级别到默认设置。

**Maven 插件属性**

插件的配置属性现在的暴露方式有所改变，现在所有的都是以spring-boot为前缀，这是为了避免和其他插件冲突而导致错误。

比如，以下命令行可以启用profile foo：

mvn spring-boot:run -Dspring-boot.run.profiles=foo

**Devtools 远程调试**

已经从Devtools中删除了通过HTTP进行远程调试的支持。（＃9489）

**Jetty**

为了和Tomcat和Undertow看齐，现在对Jetty的所有的http method的请去都进行压缩，而不是之前那样只对GET request进行压缩(#8184)。

**Reactive server 自定义**

当配置一个reactive web server时，针对Jetty，Tomcat，Undertow的定制化器现在会被调用 (#9572)。

**Jolokia**

Jolokia不再是一个endpoint。并且默认是禁用的，这已和其他的web端点保持了一直。它的配置已转移到了management.jolokia。如果想启用 Jolokia, 向你的配置加入management.jolokia.enabled=true就可以了。

**数据库迁移**

Liquibase 和 Flyway 配置的key 已被转移到了spring 的命名空间下：(比如： spring.liquibase and spring.flyway )。

**Auto-configuration排序**

@AutoConfigureOrder 默认值由Ordered.LOWEST\_PRECEDENCE变为0。 (#10142)

**Auto-configuration测试工具**

一个新的ApplicationContextRunner 测试工具让我们测试自动配置变得容易。 未来将会把所有的测试套件都迁移到这个模型上。

**Java 9 支持**

基本支持了Java 9。之所以是“基本”，是因为还没得到用户的验证。

**Jedis变为了Lettuce**

Redis客户端驱动现在由Jedis变为了Lettuce。使用Jedis的同学们，大概知道方向了吧。

但仍然支持Jedis，那么exclude掉io.lettuce:lettuce-core，然后添加 redis.clients:jedis就是了。

**OAuth 2.0 支持**

Spring Security OAuth 项目中的功能将会迁移到Spring Security中。将会OAuth 2.0。

**Mockito 1.x**

Mockito 1.x不再支持@MockBean和@SpyBean。 如果你不使用spring-boot-starter-test来管理依赖关系，则应升级到Mockito 2.x。

**JSON-B 支持**

除了Jackson 和 Gson，现在还支持了JSON-B 。JSON测试支持也已更新为新的JsonbTester类。

**Session Endpoint**

Spring Session 用现在可以通过/application/sessions Actuator Endpoint进行查找和删除session。

**ConfigurationProperties 验证**

如果你希望@ConfigurationProperties对象上开启验证，那么只需要添加 @Validated就可以了。

**Spring Mobile**

针对Spring Mobile的自动配置和依赖不再支持，已被删除。汗，我们之前的文章中还介绍过这个项目：[使用Spring Boot开发一个Spring Mobile程序](http://mp.weixin.qq.com/s?__biz=MzA5MzQ2NTY0OA==&mid=2650797144&idx=1&sn=ae284568128443afede1468949af002a&chksm=88562907bf21a01107c4c2979931217a325a5a910fe50895457e194c83081510bab56fd3097d&scene=21#wechat_redirect)。

**Spring WebFlux支持错误约定**

Spring Boot现在让WebFlux的错误约定和MVC保持一致就像使用MVC一样：默认视图和JSON响应错误，自定义错误视图等等。

**TLS 配置 和 HTTP/2 支持**

你现在可以为你的WebFlux应用配置SSL，使用server.ssl.\* 配置属性。Tomcat, Jetty, Undertow 和 Reactor Netty都支持。

你现在也可以为你的MVC 或 WebFlux应用配置HTTP/2：

使用server.http2.enabled。

**@KafkaListener支持使用@SendTo**

使用了自动配置工厂的Kafka listener现在支持@SendTo。

**Kotlin扩展**

Spring Boot 2.0 发布了Kotlin runApplication扩展：

package com.example.demo

import org.springframework.boot.autoconfigure.SpringBootApplication

import org.springframework.boot.runApplication

@SpringBootApplication

class DemoApplication

fun main(args: Array<String>) {

    runApplication<DemoApplication>(\*args)

}

**Job command line runner 顺序**

CommandLineRunner 现在执行批处理job的启动order是0。

**支持自动配置的模板化欢迎页**

Spring Boot 2.0 现在支持静态和模板化两种欢迎页类型。它会首先去配置好的静态内容的目录下查找index.html文件，如果没找到，然后就去查找index模板。 只要找到了一个就会被用作欢迎页。

**Context path会在启动时被打印**

在Spring Boot之前的版本中，对context path貌似并没有那么的重视一样。现在当我们使用内嵌容器的时候， context path 会被打印在HTTP 端口的旁边，像这样：

Tomcat started on port(s): 8080 (http) with context path '/foo'

**测试优化，自动扫描Converter和GenericConverter**

Converter 和 GenericConverter beans现在可以被@WebMvcTest和@WebFluxTest自动扫描到了。

**Health vs. Status**

status endpoint 已经被删除了，现在改成了health。这个health 端点既可以展示status也可以有更多的细节。 health 端点现在默认是被暴露的（只展示status）。如果你希望展示更多的细节信息，可以通过修改属性management.endpoints.health.show-details来实现：

management.endpoints.health.show-details=true