**Spring Boot 参考指南**

1.5.8.RELEASE

Phillip Webb , Dave Syer , Josh Long , Stéphane Nicoll , Rob Winch , Andy Wilkinson , Marcel Overdijk , Christian Dupuis , Sébastien Deleuze , Michael Simons

Copyright © 2012-2017

本文件的副本可供你自己使用，并分发给别人，但你不收取此类副本的人户费用，并进一步规定没分副本均包含本版权声明，不论是以书面形式还是以电子方式分发。

**目录**

1. [Spring Boot 文档](#_bookmark0) [1](#_bookmark0)
   1. 关于文档 [2](#_bookmark1)
   2. 获取帮助 [3](#_bookmark2)
   3. 第一步 [4](#_bookmark3)
   4. [使用 Spring Boot](#_bookmark4) [5](#_bookmark4)
   5. [了解 Spring Boot 功能](#_bookmark5) [6](#_bookmark5)
   6. 生产中使用 [7](#_bookmark6)
   7. 高级主题 [8](#_bookmark7)
2. 开始 [9](#_bookmark8)
3. [介绍 Spring Boot](#_bookmark9) [10](#_bookmark9)
4. 系统要求 [11](#_bookmark10)
   1. [Servlet 容器](#_bookmark11) [11](#_bookmark11)
5. [安装 Spring Boot](#_bookmark12) [12](#_bookmark12)
   1. Java开发人员安装说明 [12](#_bookmark13)

[Maven 安装](#_bookmark14) [12](#_bookmark14)

[Gradle 安装](#_bookmark15) [13](#_bookmark15)

* 1. [安装 Spring Boot CLI](#_bookmark16) [14](#_bookmark16)

手动安装 [14](#_bookmark17)

[安装与 SDKMAN!](#_bookmark18) [14](#_bookmark18)

[OSX 自带程序安装](#_bookmark19) [15](#_bookmark19)

[MacPorts 安装](#_bookmark20) [15](#_bookmark20)

命令行完成 [15](#_bookmark21)

[快速开始 Spring CLI 例子](#_bookmark22) [15](#_bookmark22)

* 1. [从较早版本的Spring Boot升级](#_bookmark23) [16](#_bookmark23)

1. [开发你的第一个 Spring Boot 应用](#_bookmark24) [17](#_bookmark24)
   1. [创建 POM](#_bookmark25) [17](#_bookmark25)
   2. [添加 dependencies](#_bookmark26) [18](#_bookmark26)
   3. 写代码 [18](#_bookmark27)

[@RestController 和 @RequestMapping 注解](#_bookmark28) [19](#_bookmark28)

[@EnableAutoConfiguration 注解](#_bookmark29) [19](#_bookmark29)

[“main” 方法](#_bookmark30) [19](#_bookmark30)

* 1. 运行示例 [19](#_bookmark31)
  2. [创建可执行的 jar](#_bookmark32) [20](#_bookmark32)

1. [What to read next](#_bookmark33) [22](#_bookmark33)
2. [使用 Spring Boot](#_bookmark34) [23](#_bookmark34)
3. 构建系统 [24](#_bookmark35)
   1. 依赖管理 [24](#_bookmark36)
   2. [Maven](#_bookmark37) [24](#_bookmark37)

[继承 starter parent](#_bookmark38) [24](#_bookmark38)

[没有父POM使用Spring Boot](#_bookmark39)  [25](#_bookmark39)

[变更Java 版本](#_bookmark40) [26](#_bookmark40)

[使用 Maven plugin Spring Boot](#_bookmark41)  [26](#_bookmark41)

* 1. [Gradle](#_bookmark42) [26](#_bookmark42)
  2. [Ant](#_bookmark43) [27](#_bookmark43)
  3. [Starters](#_bookmark44) [28](#_bookmark44)

1. [结构化你的代码](#_bookmark50) [33](#_bookmark50)
   1. [使用默认的包](#_bookmark51) [33](#_bookmark51)
   2. [定位 main 程序类](#_bookmark52) [33](#_bookmark52)
2. [配置 classes](#_bookmark53) [35](#_bookmark53)
   1. [导入外部配置 classes](#_bookmark54) [35](#_bookmark54)
   2. [导入 XML 配置](#_bookmark55) [35](#_bookmark55)
3. 自动装配 [36](#_bookmark56)
   1. [逐渐替换 auto-configuration](#_bookmark57) [36](#_bookmark57)
   2. [禁用特定的 auto-configuration](#_bookmark58) [36](#_bookmark58)
4. [Spring Beans 和 dependency injection](#_bookmark59) [37](#_bookmark59)
5. [使用 @SpringBootApplication 注解](#_bookmark60) [38](#_bookmark60)
6. [运行 application](#_bookmark61) [39](#_bookmark61)
   1. [从 IDE](#_bookmark62) 运行 [39](#_bookmark62)
   2. [打包运行](#_bookmark63)  [39](#_bookmark63)
   3. [使用 Maven plugin](#_bookmark64) [39](#_bookmark64)
   4. [使用 Gradle plugin](#_bookmark65) [40](#_bookmark65)
   5. [Hot swapping](#_bookmark66) [40](#_bookmark66)
7. 开发者工具 [41](#_bookmark67)
   1. 默认属性设置 [41](#_bookmark68)
   2. 自动重新启动 [42](#_bookmark69)

[Excluding resources](#_bookmark71) [43](#_bookmark71)

[监控外部 paths](#_bookmark72) [43](#_bookmark72)

禁止重启 [43](#_bookmark73)

[使用 trigger 文件](#_bookmark74) [44](#_bookmark74)

[自定义重新启动 classloader](#_bookmark75) [44](#_bookmark75)

已知限制 [44](#_bookmark76)

* 1. [LiveReload](#_bookmark77) [45](#_bookmark77)
  2. 全局设置 [45](#_bookmark78)
  3. 远程应用程序 [45](#_bookmark79)

[运行远程客户端应用程序](#_bookmark80) [46](#_bookmark80)

远程更新 [47](#_bookmark81)

远程调试 [47](#_bookmark82)

1. [打包应用程序用户生产](#_bookmark83) [49](#_bookmark83)
2. 下一步读的内容 [50](#_bookmark84)
3. [Spring Boot 特性](#_bookmark85) [51](#_bookmark85)
4. [SpringApplication](#_bookmark86) [52](#_bookmark86)
   1. 启动故障 [52](#_bookmark87)
   2. [自定义 Banner](#_bookmark88) [53](#_bookmark88)
   3. [自定义 SpringApplication](#_bookmark89) [54](#_bookmark89)
   4. [流利生成 API](#_bookmark90) [54](#_bookmark90)
   5. [应用 events and listeners](#_bookmark91) [55](#_bookmark91)
   6. [Web 环境](#_bookmark92) [55](#_bookmark92)
   7. 应用程序访问参数 [56](#_bookmark93)
   8. [使用 ApplicationRunner 或 CommandLineRunner](#_bookmark94) [56](#_bookmark94)
   9. 退出应用 [57](#_bookmark95)
   10. 管理功能 [57](#_bookmark96)
5. 外部化部署 [59](#_bookmark97)
   1. 配置随机值 [60](#_bookmark98)
   2. 命令行访问属性 [60](#_bookmark99)
   3. 应用属性文件 [61](#_bookmark100)
   4. [Profile-specific properties](#_bookmark101) [62](#_bookmark101)
   5. [Placeholders in properties](#_bookmark102) [63](#_bookmark102)
   6. [使用 YAML 代替 Properties](#_bookmark103) [63](#_bookmark103)

[加载 YAML](#_bookmark104) [63](#_bookmark104)

[Exposing YAML as properties in the Spring Environment](#_bookmark105) [64](#_bookmark105)

[Multi-profile YAML 文档](#_bookmark106) [64](#_bookmark106)

[YAML 缺陷](#_bookmark107) [65](#_bookmark107)

[Merging YAML lists](#_bookmark108) [65](#_bookmark108)

* 1. [Type-safe 配置文件](#_bookmark109) [66](#_bookmark109)

[Third-party 配置](#_bookmark110) [69](#_bookmark110)

[Relaxed binding](#_bookmark111) [69](#_bookmark111)

[Properties 转换](#_bookmark112) [70](#_bookmark112)

[@ConfigurationProperties 注解](#_bookmark113) [70](#_bookmark113)

[@ConfigurationProperties vs. @Value](#_bookmark114) [71](#_bookmark114)

1. [Profiles](#_bookmark115) [72](#_bookmark115)
   1. [Adding active profiles](#_bookmark116) [72](#_bookmark116)
   2. [Programmatically setting profiles](#_bookmark117) [72](#_bookmark117)
   3. [Profile-specific configuration files](#_bookmark118) [73](#_bookmark118)
2. [Logging](#_bookmark119) [74](#_bookmark119)
   1. [Log format](#_bookmark120) [74](#_bookmark120)
   2. [Console output](#_bookmark121) [74](#_bookmark121)

[Color-coded output](#_bookmark122) [75](#_bookmark122)

* 1. [File output](#_bookmark123) [76](#_bookmark123)
  2. [Log Levels](#_bookmark124) [76](#_bookmark124)
  3. 自定义日志配置文件 [77](#_bookmark125)
  4. [Logback extensions](#_bookmark126) [78](#_bookmark126)

[Profile-specific configuration](#_bookmark127) [79](#_bookmark127)

[Environment properties](#_bookmark128) [79](#_bookmark128)

1. [Developing web applications](#_bookmark129) [80](#_bookmark129)
   1. [The ‘Spring Web MVC framework’](#_bookmark130) [80](#_bookmark130)

[Spring MVC auto-configuration](#_bookmark131) [80](#_bookmark131)

[HttpMessageConverters](#_bookmark132) [81](#_bookmark132)

[Custom JSON Serializers and Deserializers](#_bookmark133) [81](#_bookmark133)

[MessageCodesResolver](#_bookmark134) [82](#_bookmark134)

[Static Content](#_bookmark135) [82](#_bookmark135)

[Custom Favicon](#_bookmark136) [84](#_bookmark136)

[ConfigurableWebBindingInitializer](#_bookmark137) [84](#_bookmark137)

[Template engines](#_bookmark138) [84](#_bookmark138)

[Error Handling](#_bookmark139) [84](#_bookmark139)

[Custom error pages](#_bookmark140) [85](#_bookmark140)

[Mapping error pages outside of Spring MVC](#_bookmark141) [86](#_bookmark141)

[Error Handling on WebSphere Application Server](#_bookmark142) [86](#_bookmark142)

[Spring HATEOAS](#_bookmark143) [87](#_bookmark143)

[CORS support](#_bookmark144) [87](#_bookmark144)

* 1. [JAX-RS and Jersey](#_bookmark145) [87](#_bookmark145)
  2. [Embedded servlet container support](#_bookmark146) [88](#_bookmark146)

[Servlets, Filters, and listeners](#_bookmark147) [89](#_bookmark147)

[Registering Servlets, Filters, and listeners as Spring beans](#_bookmark148) [89](#_bookmark148)

[Servlet Context Initialization](#_bookmark149) [89](#_bookmark149)

[Scanning for Servlets, Filters, and listeners](#_bookmark150) [89](#_bookmark150)

[The EmbeddedWebApplicationContext](#_bookmark151) [89](#_bookmark151)

[Customizing embedded servlet containers](#_bookmark152) [90](#_bookmark152)

[Programmatic customization](#_bookmark153) [90](#_bookmark153)

[Customizing ConfigurableEmbeddedServletContainer directly](#_bookmark154) [91](#_bookmark154)

[JSP limitations](#_bookmark155) [91](#_bookmark155)

1. [Security](#_bookmark156) [92](#_bookmark156)
   1. [OAuth2](#_bookmark157) [93](#_bookmark157)

[Authorization Server](#_bookmark158) [93](#_bookmark158)

[Resource Server](#_bookmark159) [93](#_bookmark159)

* 1. [Token Type in User Info](#_bookmark160) [94](#_bookmark160)
  2. [Customizing the User Info RestTemplate](#_bookmark161) [94](#_bookmark161)

[Client](#_bookmark162) [95](#_bookmark162)

[Single Sign On](#_bookmark163) [96](#_bookmark163)

* 1. [Actuator Security](#_bookmark164) [97](#_bookmark164)

1. [Working with SQL databases](#_bookmark165) [98](#_bookmark165)
   1. [Configure a DataSource](#_bookmark166) [98](#_bookmark166)

[Embedded Database Support](#_bookmark167) [98](#_bookmark167)

[Connection to a production database](#_bookmark168) [99](#_bookmark168)

[Connection to a JNDI DataSource](#_bookmark169) [100](#_bookmark169)

* 1. [Using JdbcTemplate](#_bookmark170) [100](#_bookmark170)
  2. [JPA and ‘Spring Data’](#_bookmark171) [101](#_bookmark171)

[Entity Classes](#_bookmark172) [101](#_bookmark172)

[Spring Data JPA Repositories](#_bookmark173) [102](#_bookmark173)

[Creating and dropping JPA databases](#_bookmark174) [103](#_bookmark174)

[Open EntityManager in View](#_bookmark175) [103](#_bookmark175)

* 1. [Using H2’s web console](#_bookmark176) [103](#_bookmark176)

[Changing the H2 console’s path](#_bookmark177) [103](#_bookmark177)

[Securing the H2 console](#_bookmark178) [104](#_bookmark178)

* 1. [Using jOOQ](#_bookmark179) [104](#_bookmark179)

[Code Generation](#_bookmark180) [104](#_bookmark180)

[Using DSLContext](#_bookmark181) [104](#_bookmark181)

[Customizing jOOQ](#_bookmark182) [105](#_bookmark182)

1. [Working with NoSQL technologies](#_bookmark183) [106](#_bookmark183)
   1. [Redis](#_bookmark184) [106](#_bookmark184)

[Connecting to Redis](#_bookmark185) [106](#_bookmark185)

* 1. [MongoDB](#_bookmark186) [106](#_bookmark186)

[Connecting to a MongoDB database](#_bookmark187) [106](#_bookmark187)

[MongoTemplate](#_bookmark188) [107](#_bookmark188)

[Spring Data MongoDB repositories](#_bookmark189) [108](#_bookmark189)

[Embedded Mongo](#_bookmark190) [108](#_bookmark190)

[30.3. Neo4j](#_bookmark191) [109](#_bookmark191)

[Connecting to a Neo4j database](#_bookmark192) [109](#_bookmark192)

[Using the embedded mode](#_bookmark193) [109](#_bookmark193)

[Neo4jSession](#_bookmark194) [109](#_bookmark194)

[Spring Data Neo4j repositories](#_bookmark195) [110](#_bookmark195)

[Repository example](#_bookmark196) [110](#_bookmark196)

* 1. [Gemfire](#_bookmark197) [110](#_bookmark197)
  2. [Solr](#_bookmark198) [110](#_bookmark198)

[Connecting to Solr](#_bookmark199) [111](#_bookmark199)

[Spring Data Solr repositories](#_bookmark200) [111](#_bookmark200)

* 1. [Elasticsearch](#_bookmark201) [111](#_bookmark201)

[Connecting to Elasticsearch using Jest](#_bookmark202) [111](#_bookmark202)

[Connecting to Elasticsearch using Spring Data](#_bookmark203) [112](#_bookmark203)

[Spring Data Elasticsearch repositories](#_bookmark204) [112](#_bookmark204)

* 1. [Cassandra](#_bookmark205) [112](#_bookmark205)

[Connecting to Cassandra](#_bookmark206) [113](#_bookmark206)

[Spring Data Cassandra repositories](#_bookmark207) [113](#_bookmark207)

* 1. [Couchbase](#_bookmark208) [113](#_bookmark208)

[Connecting to Couchbase](#_bookmark209) [113](#_bookmark209)

[Spring Data Couchbase repositories](#_bookmark210) [114](#_bookmark210)

* 1. [LDAP](#_bookmark211) [115](#_bookmark211)

[Connecting to an LDAP server](#_bookmark212) [115](#_bookmark212)

[Spring Data LDAP repositories](#_bookmark213) [115](#_bookmark213)

[Embedded in-memory LDAP server](#_bookmark214) [115](#_bookmark214)

1. [Caching](#_bookmark215) [117](#_bookmark215)
   1. [Supported cache providers](#_bookmark216) [118](#_bookmark216)

[Generic](#_bookmark217) [119](#_bookmark217)

[JCache (JSR-107)](#_bookmark218) [119](#_bookmark218)

[EhCache 2.x](#_bookmark219) [120](#_bookmark219)

[Hazelcast](#_bookmark220) [120](#_bookmark220)

[Infinispan](#_bookmark221) [120](#_bookmark221)

[Couchbase](#_bookmark222) [120](#_bookmark222)

[Redis](#_bookmark223) [121](#_bookmark223)

[Caffeine](#_bookmark224) [121](#_bookmark224)

[Guava (deprecated)](#_bookmark225) [122](#_bookmark225)

[Simple](#_bookmark226) [122](#_bookmark226)

[None](#_bookmark227) [122](#_bookmark227)

1. [Messaging](#_bookmark228) [123](#_bookmark228)

[32.1. JMS](#_bookmark229) [123](#_bookmark229)

[ActiveMQ support](#_bookmark230) [123](#_bookmark230)

[Artemis support](#_bookmark231) [124](#_bookmark231)

[Using a JNDI ConnectionFactory](#_bookmark232) [124](#_bookmark232)

[Sending a message](#_bookmark233) [124](#_bookmark233)

[Receiving a message](#_bookmark234) [125](#_bookmark234)

* 1. [AMQP](#_bookmark235) [126](#_bookmark235)

[RabbitMQ support](#_bookmark236) [126](#_bookmark236)

[Sending a message](#_bookmark237) [126](#_bookmark237)

[Receiving a message](#_bookmark238) [127](#_bookmark238)

* 1. [Apache Kafka Support](#_bookmark239) [128](#_bookmark239)

[Sending a Message](#_bookmark240) [128](#_bookmark240)

[Receiving a Message](#_bookmark241) [129](#_bookmark241)

[Additional Kafka Properties](#_bookmark242) [129](#_bookmark242)

1. [Calling REST services](#_bookmark243) [131](#_bookmark243)
   1. [RestTemplate customization](#_bookmark244) [131](#_bookmark244)
2. [Validation](#_bookmark245) [133](#_bookmark245)
3. [Sending email](#_bookmark246) [134](#_bookmark246)
4. [Distributed Transactions with JTA](#_bookmark247) [135](#_bookmark247)
   1. [Using an Atomikos transaction manager](#_bookmark248) [135](#_bookmark248)
   2. [Using a Bitronix transaction manager](#_bookmark249) [135](#_bookmark249)
   3. [Using a Narayana transaction manager](#_bookmark250) [136](#_bookmark250)
   4. [Using a Java EE managed transaction manager](#_bookmark251) [136](#_bookmark251)
   5. [Mixing XA and non-XA JMS connections](#_bookmark252) [136](#_bookmark252)
   6. [Supporting an alternative embedded transaction manager](#_bookmark253) [137](#_bookmark253)
5. [Hazelcast](#_bookmark254) [138](#_bookmark254)
6. [Spring Integration](#_bookmark255) [139](#_bookmark255)
7. [Spring Session](#_bookmark256) [140](#_bookmark256)
8. [Monitoring and management over JMX](#_bookmark257) [141](#_bookmark257)
9. [Testing](#_bookmark258) [142](#_bookmark258)
   1. [Test scope dependencies](#_bookmark259) [142](#_bookmark259)
   2. [Testing Spring applications](#_bookmark260) [142](#_bookmark260)
   3. [Testing Spring Boot applications](#_bookmark261) [143](#_bookmark261)

[Detecting test configuration](#_bookmark262) [143](#_bookmark262)

[Excluding test configuration](#_bookmark263) [144](#_bookmark263)

[Working with random ports](#_bookmark264) [144](#_bookmark264)

[Mocking and spying beans](#_bookmark265) [145](#_bookmark265)

[Auto-configured tests](#_bookmark266) [146](#_bookmark266)

[Auto-configured JSON tests](#_bookmark267) [146](#_bookmark267)

[Auto-configured Spring MVC tests](#_bookmark268) [147](#_bookmark268)

[Auto-configured Data JPA tests](#_bookmark269) [149](#_bookmark269)

[Auto-configured JDBC tests](#_bookmark270) [150](#_bookmark270)

[Auto-configured Data MongoDB tests](#_bookmark271) [150](#_bookmark271)

[Auto-configured REST clients](#_bookmark272) [151](#_bookmark272)

[Auto-configured Spring REST Docs tests](#_bookmark273) [152](#_bookmark273)

[Using Spock to test Spring Boot applications](#_bookmark274) [153](#_bookmark274)

* 1. [Test utilities](#_bookmark275) [153](#_bookmark275)

[ConfigFileApplicationContextInitializer](#_bookmark276) [153](#_bookmark276)

[EnvironmentTestUtils](#_bookmark277) [154](#_bookmark277)

[OutputCapture](#_bookmark278) [154](#_bookmark278)

[TestRestTemplate](#_bookmark279) [154](#_bookmark279)

1. [WebSockets](#_bookmark280) [156](#_bookmark280)
2. [Web Services](#_bookmark281) [157](#_bookmark281)
3. [Creating your own auto-configuration](#_bookmark282) [158](#_bookmark282)
   1. [Understanding auto-configured beans](#_bookmark283) [158](#_bookmark283)
   2. [Locating auto-configuration candidates](#_bookmark284) [158](#_bookmark284)
   3. [Condition annotations](#_bookmark285) [158](#_bookmark285)

[Class conditions](#_bookmark286) [159](#_bookmark286)

[Bean conditions](#_bookmark287) [159](#_bookmark287)

[Property conditions](#_bookmark288) [160](#_bookmark288)

[Resource conditions](#_bookmark289) [160](#_bookmark289)

[Web application conditions](#_bookmark290) [160](#_bookmark290)

[SpEL expression conditions](#_bookmark291) [160](#_bookmark291)

* 1. [Creating your own starter](#_bookmark292) [160](#_bookmark292)

[Naming](#_bookmark293) [160](#_bookmark293)

[Autoconfigure module](#_bookmark294) [161](#_bookmark294)

[Starter module](#_bookmark295) [161](#_bookmark295)

1. [What to read next](#_bookmark296) [162](#_bookmark296)
2. [Spring Boot Actuator: Production-ready features](#_bookmark297) [163](#_bookmark297)
3. [Enabling production-ready features](#_bookmark298) [164](#_bookmark298)
4. [Endpoints](#_bookmark299) [165](#_bookmark299)
   1. [Customizing endpoints](#_bookmark300) [166](#_bookmark300)
   2. [Hypermedia for actuator MVC endpoints](#_bookmark301) [167](#_bookmark301)
   3. [CORS support](#_bookmark302) [167](#_bookmark302)
   4. [Adding custom endpoints](#_bookmark303) [167](#_bookmark303)
   5. [Health information](#_bookmark304) [168](#_bookmark304)
   6. [Security with HealthIndicators](#_bookmark305) [168](#_bookmark305)

[Auto-configured HealthIndicators](#_bookmark306) [168](#_bookmark306)

[Writing custom HealthIndicators](#_bookmark307) [169](#_bookmark307)

* 1. [Application information](#_bookmark308) [170](#_bookmark308)

[Auto-configured InfoContributors](#_bookmark309) [170](#_bookmark309)

[Custom application info information](#_bookmark310) [170](#_bookmark310)

[Git commit information](#_bookmark311) [171](#_bookmark311)

[Build information](#_bookmark312) [171](#_bookmark312)

[Writing custom InfoContributors](#_bookmark313) [171](#_bookmark313)

1. [Monitoring and management over HTTP](#_bookmark314) [173](#_bookmark314)
   1. [Accessing sensitive endpoints](#_bookmark315) [173](#_bookmark315)
   2. [Customizing the management endpoint paths](#_bookmark316) [174](#_bookmark316)
   3. [Customizing the management server port](#_bookmark317) [174](#_bookmark317)
   4. [Configuring management-specific SSL](#_bookmark318) [175](#_bookmark318)
   5. [Customizing the management server address](#_bookmark319) [175](#_bookmark319)
   6. [Disabling HTTP endpoints](#_bookmark320) [175](#_bookmark320)
   7. [HTTP health endpoint format and access restrictions](#_bookmark321) [176](#_bookmark321)
2. [Monitoring and management over JMX](#_bookmark322) [178](#_bookmark322)
   1. [Customizing MBean names](#_bookmark323) [178](#_bookmark323)
   2. [Disabling JMX endpoints](#_bookmark324) [178](#_bookmark324)
   3. [Using Jolokia for JMX over HTTP](#_bookmark325) [178](#_bookmark325)

[Customizing Jolokia](#_bookmark326) [178](#_bookmark326)

[Disabling Jolokia](#_bookmark327) [178](#_bookmark327)

1. [Monitoring and management using a remote shell (deprecated)](#_bookmark328) [180](#_bookmark328)
   1. [Connecting to the remote shell](#_bookmark329) [180](#_bookmark329)

[Remote shell credentials](#_bookmark330) [181](#_bookmark330)

* 1. [Extending the remote shell](#_bookmark331) [181](#_bookmark331)

[Remote shell commands](#_bookmark332) [181](#_bookmark332)

[Remote shell plugins](#_bookmark333) [182](#_bookmark333)

1. [Loggers](#_bookmark334) [183](#_bookmark334)
   1. [Configure a Logger](#_bookmark335) [183](#_bookmark335)
2. [Metrics](#_bookmark336) [184](#_bookmark336)
   1. [System metrics](#_bookmark337) [184](#_bookmark337)
   2. [DataSource metrics](#_bookmark338) [185](#_bookmark338)
   3. [Cache metrics](#_bookmark339) [185](#_bookmark339)
   4. [Tomcat session metrics](#_bookmark340) [186](#_bookmark340)
   5. [Recording your own metrics](#_bookmark341) [186](#_bookmark341)
   6. [Adding your own public metrics](#_bookmark342) [186](#_bookmark342)
   7. [Special features with Java 8](#_bookmark343) [187](#_bookmark343)
   8. [Metric writers, exporters and aggregation](#_bookmark344) [187](#_bookmark344)

[Example: Export to Redis](#_bookmark345) [188](#_bookmark345)

[Example: Export to Open TSDB](#_bookmark346) [188](#_bookmark346)

[Example: Export to Statsd](#_bookmark347) [189](#_bookmark347)

[Example: Export to JMX](#_bookmark348) [189](#_bookmark348)

* 1. [Aggregating metrics from multiple sources](#_bookmark349) [190](#_bookmark349)
  2. [Dropwizard Metrics](#_bookmark350) [190](#_bookmark350)
  3. [Message channel integration](#_bookmark351) [191](#_bookmark351)

1. [Auditing](#_bookmark352) [192](#_bookmark352)
2. [Tracing](#_bookmark353) [193](#_bookmark353)
   1. [Custom tracing](#_bookmark354) [193](#_bookmark354)
3. [Process monitoring](#_bookmark355) [194](#_bookmark355)
   1. [Extend configuration](#_bookmark356) [194](#_bookmark356)
   2. [Programmatically](#_bookmark357) [194](#_bookmark357)
4. [Cloud Foundry support](#_bookmark358) [195](#_bookmark358)
   1. [Disabling extended Cloud Foundry actuator support](#_bookmark359) [195](#_bookmark359)
   2. [Cloud Foundry self signed certificates](#_bookmark360) [195](#_bookmark360)
   3. [Custom security configuration](#_bookmark361) [195](#_bookmark361)
5. [What to read next](#_bookmark362) [196](#_bookmark362)
6. [Deploying Spring Boot applications](#_bookmark363) [197](#_bookmark363)
7. [Deploying to the cloud](#_bookmark364) [198](#_bookmark364)
   1. [Cloud Foundry](#_bookmark365) [198](#_bookmark365)

[Binding to services](#_bookmark366) [199](#_bookmark366)

* 1. [Heroku](#_bookmark367) [199](#_bookmark367)
  2. [OpenShift](#_bookmark368) [201](#_bookmark368)
  3. [Amazon Web Services (AWS)](#_bookmark369) [201](#_bookmark369)

[AWS Elastic Beanstalk](#_bookmark370) [202](#_bookmark370)

[Using the Tomcat platform](#_bookmark371) [202](#_bookmark371)

[Using the Java SE platform](#_bookmark372) [202](#_bookmark372)

[Best practices](#_bookmark373) [202](#_bookmark373)

[Summary](#_bookmark374) [202](#_bookmark374)

* 1. [Boxfuse and Amazon Web Services](#_bookmark375) [202](#_bookmark375)
  2. [Google Cloud](#_bookmark376) [203](#_bookmark376)

1. [Installing Spring Boot applications](#_bookmark377) [205](#_bookmark377)
   1. [Supported operating systems](#_bookmark378) [205](#_bookmark378)
   2. [Unix/Linux services](#_bookmark379) [205](#_bookmark379)

[Installation as an init.d service (System V)](#_bookmark380) [205](#_bookmark380)

[Securing an init.d service](#_bookmark381) [206](#_bookmark381)

[Installation as a systemd service](#_bookmark382) [207](#_bookmark382)

[Customizing the startup script](#_bookmark383) [207](#_bookmark383)

[Customizing script when it’s written](#_bookmark384) [208](#_bookmark384)

[Customizing script when it runs](#_bookmark385) [209](#_bookmark385)

* 1. [Microsoft Windows services](#_bookmark387) [210](#_bookmark387)

1. [What to read next](#_bookmark388) [211](#_bookmark388)
2. [Spring Boot CLI](#_bookmark389) [212](#_bookmark389)
3. [Installing the CLI](#_bookmark390) [213](#_bookmark390)
4. [Using the CLI](#_bookmark391) [214](#_bookmark391)
   1. [Running applications using the CLI](#_bookmark392) [214](#_bookmark392)

[Deduced “grab” dependencies](#_bookmark393) [215](#_bookmark393)

[Deduced “grab” coordinates](#_bookmark394) [216](#_bookmark394)

[Default import statements](#_bookmark395) [216](#_bookmark395)

[Automatic main method](#_bookmark396) [216](#_bookmark396)

[Custom dependency management](#_bookmark397) [216](#_bookmark397)

* 1. [Testing your code](#_bookmark398) [217](#_bookmark398)
  2. [Applications with multiple source files](#_bookmark399) [217](#_bookmark399)
  3. [Packaging your application](#_bookmark400) [217](#_bookmark400)
  4. [Initialize a new project](#_bookmark401) [218](#_bookmark401)
  5. [Using the embedded shell](#_bookmark402) [218](#_bookmark402)
  6. [Adding extensions to the CLI](#_bookmark403) [219](#_bookmark403)

1. [Developing application with the Groovy beans DSL](#_bookmark404) [220](#_bookmark404)
2. [Configuring the CLI with settings.xml](#_bookmark405) [221](#_bookmark405)
3. [What to read next](#_bookmark406) [222](#_bookmark406)
4. [Build tool plugins](#_bookmark407) [223](#_bookmark407)
5. [Spring Boot Maven plugin](#_bookmark408) [224](#_bookmark408)
   1. [Including the plugin](#_bookmark409) [224](#_bookmark409)
   2. [Packaging executable jar and war files](#_bookmark410) [225](#_bookmark410)
6. [Spring Boot Gradle plugin](#_bookmark411) [226](#_bookmark411)
   1. [Including the plugin](#_bookmark412) [226](#_bookmark412)
   2. [Gradle dependency management](#_bookmark413) [226](#_bookmark413)
   3. [Packaging executable jar and war files](#_bookmark414) [226](#_bookmark414)
   4. [Running a project in-place](#_bookmark415) [227](#_bookmark415)
   5. [Spring Boot plugin configuration](#_bookmark416) [227](#_bookmark416)
   6. [Repackage configuration](#_bookmark417) [228](#_bookmark417)
   7. [Repackage with custom Gradle configuration](#_bookmark418) [229](#_bookmark418)

[Configuration options](#_bookmark419) [229](#_bookmark419)

[Available layouts](#_bookmark420) [230](#_bookmark420)

[Using a custom layout](#_bookmark421) [230](#_bookmark421)

* 1. [Understanding how the Gradle plugin works](#_bookmark422) [231](#_bookmark422)
  2. [Publishing artifacts to a Maven repository using Gradle](#_bookmark423) [231](#_bookmark423)

[Configuring Gradle to produce a pom that inherits dependency management](#_bookmark424) [231](#_bookmark424)

[Configuring Gradle to produce a pom that imports dependency management](#_bookmark425) [232](#_bookmark425)

1. [Spring Boot AntLib module](#_bookmark426) [233](#_bookmark426)
   1. [Spring Boot Ant tasks](#_bookmark427) [233](#_bookmark427)

[spring-boot:exejar](#_bookmark428) [233](#_bookmark428)

[Examples](#_bookmark429) [233](#_bookmark429)

* 1. [spring-boot:findmainclass](#_bookmark430) [234](#_bookmark430)

[Examples](#_bookmark431) [234](#_bookmark431)

1. [Supporting other build systems](#_bookmark432) [235](#_bookmark432)
   1. [Repackaging archives](#_bookmark433) [235](#_bookmark433)
   2. [Nested libraries](#_bookmark434) [235](#_bookmark434)
   3. [Finding a main class](#_bookmark435) [235](#_bookmark435)
   4. [Example repackage implementation](#_bookmark436) [235](#_bookmark436)
2. [What to read next](#_bookmark437) [236](#_bookmark437)
3. [‘How-to’ guides](#_bookmark438) [237](#_bookmark438)
4. [Spring Boot application](#_bookmark439) [238](#_bookmark439)
   1. [Create your own FailureAnalyzer](#_bookmark440) [238](#_bookmark440)
   2. [Troubleshoot auto-configuration](#_bookmark441) [238](#_bookmark441)
   3. [Customize the Environment or ApplicationContext before it starts](#_bookmark442) [239](#_bookmark442)
   4. [Build an ApplicationContext hierarchy (adding a parent or root context)](#_bookmark443) [240](#_bookmark443)
   5. [Create a non-web application](#_bookmark444) [240](#_bookmark444)
5. [Properties & configuration](#_bookmark445) [241](#_bookmark445)
   1. [Automatically expand properties at build time](#_bookmark446) [241](#_bookmark446)

[Automatic property expansion using Maven](#_bookmark447) [241](#_bookmark447)

[Automatic property expansion using Gradle](#_bookmark448) [242](#_bookmark448)

* 1. [Externalize the configuration of SpringApplication](#_bookmark449) [242](#_bookmark449)
  2. [Change the location of external properties of an application](#_bookmark450) [243](#_bookmark450)
  3. [Use ‘short’ command line arguments](#_bookmark451) [243](#_bookmark451)
  4. [Use YAML for external properties](#_bookmark452) [244](#_bookmark452)
  5. [Set the active Spring profiles](#_bookmark453) [244](#_bookmark453)
  6. [Change configuration depending on the environment](#_bookmark454) [244](#_bookmark454)
  7. [Discover built-in options for external properties](#_bookmark455) [245](#_bookmark455)

1. [Embedded servlet containers](#_bookmark456) [246](#_bookmark456)
   1. [Add a Servlet, Filter or Listener to an application](#_bookmark457) [246](#_bookmark457)

[Add a Servlet, Filter or Listener using a Spring bean](#_bookmark458) [246](#_bookmark458)

[Disable registration of a Servlet or Filter](#_bookmark459) [246](#_bookmark459)

[Add Servlets, Filters, and Listeners using classpath scanning](#_bookmark460) [246](#_bookmark460)

* 1. [Change the HTTP port](#_bookmark461) [247](#_bookmark461)
  2. [Use a random unassigned HTTP port](#_bookmark462) [247](#_bookmark462)
  3. [Discover the HTTP port at runtime](#_bookmark463) [247](#_bookmark463)
  4. [Configure SSL](#_bookmark464) [247](#_bookmark464)
  5. [Configure Access Logging](#_bookmark465) [248](#_bookmark465)
  6. [Use behind a front-end proxy server](#_bookmark466) [248](#_bookmark466)

[Customize Tomcat’s proxy configuration](#_bookmark467) [249](#_bookmark467)

* 1. [Configure Tomcat](#_bookmark468) [249](#_bookmark468)
  2. [Enable Multiple Connectors with Tomcat](#_bookmark469) [249](#_bookmark469)
  3. [Use Tomcat’s LegacyCookieProcessor](#_bookmark470) [250](#_bookmark470)
  4. [Use Jetty instead of Tomcat](#_bookmark471) [250](#_bookmark471)
  5. [Configure Jetty](#_bookmark472) [251](#_bookmark472)
  6. [Use Undertow instead of Tomcat](#_bookmark473) [251](#_bookmark473)
  7. [Configure Undertow](#_bookmark474) [252](#_bookmark474)
  8. [Enable Multiple Listeners with Undertow](#_bookmark475) [252](#_bookmark475)
  9. [Use Tomcat 7.x or 8.0](#_bookmark476) [252](#_bookmark476)

[Use Tomcat 7.x or 8.0 with Maven](#_bookmark477) [252](#_bookmark477)

[Use Tomcat 7.x or 8.0 with Gradle](#_bookmark478) [253](#_bookmark478)

* 1. [Use Jetty 9.2](#_bookmark479) [253](#_bookmark479)

[Use Jetty 9.2 with Maven](#_bookmark480) [253](#_bookmark480)

[Use Jetty 9.2 with Gradle](#_bookmark481) [253](#_bookmark481)

* 1. [Use Jetty 8](#_bookmark482) [253](#_bookmark482)

[Use Jetty 8 with Maven](#_bookmark483) [254](#_bookmark483)

[Use Jetty 8 with Gradle](#_bookmark484) [254](#_bookmark484)

* 1. [Create WebSocket endpoints using @ServerEndpoint](#_bookmark485) [254](#_bookmark485)
  2. [Enable HTTP response compression](#_bookmark486) [255](#_bookmark486)

1. [Spring MVC](#_bookmark487) [256](#_bookmark487)
   1. [Write a JSON REST service](#_bookmark488) [256](#_bookmark488)
   2. [Write an XML REST service](#_bookmark489) [256](#_bookmark489)
   3. [Customize the Jackson ObjectMapper](#_bookmark490) [256](#_bookmark490)
   4. [Customize the @ResponseBody rendering](#_bookmark491) [258](#_bookmark491)
   5. [Handling Multipart File Uploads](#_bookmark492) [258](#_bookmark492)
   6. [Switch off the Spring MVC DispatcherServlet](#_bookmark493) [258](#_bookmark493)
   7. [Switch off the Default MVC configuration](#_bookmark494) [259](#_bookmark494)
   8. [Customize ViewResolvers](#_bookmark495) [259](#_bookmark495)
   9. [Use Thymeleaf 3](#_bookmark496) [260](#_bookmark496)
2. [HTTP clients](#_bookmark497) [261](#_bookmark497)
   1. [Configure RestTemplate to use a proxy](#_bookmark498) [261](#_bookmark498)
3. [Logging](#_bookmark499) [262](#_bookmark499)
   1. [Configure Logback for logging](#_bookmark500) [262](#_bookmark500)

[Configure logback for file only output](#_bookmark501) [263](#_bookmark501)

* 1. [Configure Log4j for logging](#_bookmark502) [263](#_bookmark502)

[Use YAML or JSON to configure Log4j 2](#_bookmark503) [264](#_bookmark503)

1. [Data Access](#_bookmark504) [265](#_bookmark504)
   1. [Configure a custom DataSource](#_bookmark505) [265](#_bookmark505)
   2. [Configure Two DataSources](#_bookmark506) [267](#_bookmark506)
   3. [Use Spring Data repositories](#_bookmark507) [268](#_bookmark507)
   4. [Separate @Entity definitions from Spring configuration](#_bookmark508) [268](#_bookmark508)
   5. [Configure JPA properties](#_bookmark509) [268](#_bookmark509)
   6. [Configure Hibernate Naming Strategy](#_bookmark510) [269](#_bookmark510)
   7. [Use a custom EntityManagerFactory](#_bookmark511) [269](#_bookmark511)
   8. [Use Two EntityManagers](#_bookmark512) [269](#_bookmark512)
   9. [Use a traditional persistence.xml](#_bookmark513) [270](#_bookmark513)
   10. [Use Spring Data JPA and Mongo repositories](#_bookmark514) [270](#_bookmark514)
   11. [Expose Spring Data repositories as REST endpoint](#_bookmark515) [271](#_bookmark515)
   12. [Configure a component that is used by JPA](#_bookmark516) [271](#_bookmark516)
2. [Database initialization](#_bookmark517) [272](#_bookmark517)
   1. [Initialize a database using JPA](#_bookmark518) [272](#_bookmark518)
   2. [Initialize a database using Hibernate](#_bookmark519) [272](#_bookmark519)
   3. [Initialize a database](#_bookmark520) [272](#_bookmark520)
   4. [Initialize a Spring Batch database](#_bookmark521) [273](#_bookmark521)
   5. [Use a higher-level database migration tool](#_bookmark522) [273](#_bookmark522)

[Execute Flyway database migrations on startup](#_bookmark523) [273](#_bookmark523)

[Execute Liquibase database migrations on startup](#_bookmark524) [274](#_bookmark524)

1. [Messaging](#_bookmark525) [275](#_bookmark525)
   1. [Disable transacted JMS session](#_bookmark526) [275](#_bookmark526)
2. [Batch applications](#_bookmark527) [276](#_bookmark527)
   1. [Execute Spring Batch jobs on startup](#_bookmark528) [276](#_bookmark528)
3. [Actuator](#_bookmark529) [277](#_bookmark529)
   1. [Change the HTTP port or address of the actuator endpoints](#_bookmark530) [277](#_bookmark530)
   2. [Customize the ‘whitelabel’ error page](#_bookmark531) [277](#_bookmark531)
   3. [Actuator and Jersey](#_bookmark532) [277](#_bookmark532)
4. [Security](#_bookmark533) [278](#_bookmark533)
   1. [Switch off the Spring Boot security configuration](#_bookmark534) [278](#_bookmark534)
   2. [Change the AuthenticationManager and add user accounts](#_bookmark535) [278](#_bookmark535)
   3. [Enable HTTPS when running behind a proxy server](#_bookmark536) [278](#_bookmark536)
5. [Hot swapping](#_bookmark537) [280](#_bookmark537)
   1. [Reload static content](#_bookmark538) [280](#_bookmark538)
   2. [Reload templates without restarting the container](#_bookmark539) [280](#_bookmark539)

[Thymeleaf templates](#_bookmark540) [280](#_bookmark540)

[FreeMarker templates](#_bookmark541) [280](#_bookmark541)

[Groovy templates](#_bookmark542) [280](#_bookmark542)

* 1. [Fast application restarts](#_bookmark543) [280](#_bookmark543)
  2. [Reload Java classes without restarting the container](#_bookmark544) [281](#_bookmark544)

[Configuring Spring Loaded for use with Maven](#_bookmark545) [281](#_bookmark545)

[Configuring Spring Loaded for use with Gradle and IntelliJ IDEA](#_bookmark546) [281](#_bookmark546)

1. [Build](#_bookmark547) [283](#_bookmark547)
   1. [Generate build information](#_bookmark548) [283](#_bookmark548)
   2. [Generate git information](#_bookmark549) [283](#_bookmark549)
   3. [Customize dependency versions](#_bookmark550) [284](#_bookmark550)
   4. [Create an executable JAR with Maven](#_bookmark551) [284](#_bookmark551)
   5. [Use a Spring Boot application as a dependency](#_bookmark552) [285](#_bookmark552)
   6. [Extract specific libraries when an executable jar runs](#_bookmark553) [286](#_bookmark553)
   7. [Create a non-executable JAR with exclusions](#_bookmark554) [286](#_bookmark554)
   8. [Remote debug a Spring Boot application started with Maven](#_bookmark555) [287](#_bookmark555)
   9. [Remote debug a Spring Boot application started with Gradle](#_bookmark556) [287](#_bookmark556)
   10. [Build an executable archive from Ant without using spring-boot-antlib](#_bookmark557) [288](#_bookmark557)
   11. [How to use Java 6](#_bookmark558) [289](#_bookmark558)

[Embedded servlet container compatibility](#_bookmark559) [289](#_bookmark559)

[Jackson](#_bookmark560) [289](#_bookmark560)

[JTA API compatibility](#_bookmark561) [289](#_bookmark561)

1. [Traditional deployment](#_bookmark562) [290](#_bookmark562)
   1. [Create a deployable war file](#_bookmark563) [290](#_bookmark563)
   2. [Create a deployable war file for older servlet containers](#_bookmark564) [291](#_bookmark564)
   3. [Convert an existing application to Spring Boot](#_bookmark565) [291](#_bookmark565)
   4. [Deploying a WAR to WebLogic](#_bookmark566) [293](#_bookmark566)
   5. [Deploying a WAR in an Old (Servlet 2.5) Container](#_bookmark567) [293](#_bookmark567)
2. [Appendices](#_bookmark568) [295](#_bookmark568)
3. [Common application properties](#_bookmark569) [296](#_bookmark569)
4. [Configuration meta-data](#_bookmark571) [318](#_bookmark571)
   1. [Meta-data format](#_bookmark572) [318](#_bookmark572)

[Group Attributes](#_bookmark573) [319](#_bookmark573)

[Property Attributes](#_bookmark574) [320](#_bookmark574)

[Hint Attributes](#_bookmark575) [322](#_bookmark575)

[Repeated meta-data items](#_bookmark576) [322](#_bookmark576)

* 1. [Providing manual hints](#_bookmark577) [323](#_bookmark577)

[Value hint](#_bookmark578) [323](#_bookmark578)

[Value provider](#_bookmark579) [323](#_bookmark579)

[Any](#_bookmark580) [324](#_bookmark580)

[Class reference](#_bookmark581) [325](#_bookmark581)

[Handle As](#_bookmark582) [325](#_bookmark582)

[Logger name](#_bookmark583) [326](#_bookmark583)

[Spring bean reference](#_bookmark584) [327](#_bookmark584)

[Spring profile name](#_bookmark585) [328](#_bookmark585)

* 1. [Generating your own meta-data using the annotation processor](#_bookmark586) [328](#_bookmark586)

[Nested properties](#_bookmark587) [329](#_bookmark587)

[Adding additional meta-data](#_bookmark588) [329](#_bookmark588)

1. [Auto-configuration classes](#_bookmark589) [331](#_bookmark589)
   1. [From the “spring-boot-autoconfigure” module](#_bookmark590) [331](#_bookmark590)
   2. [From the “spring-boot-actuator” module](#_bookmark591) [334](#_bookmark591)
2. [Test auto-configuration annotations](#_bookmark592) [336](#_bookmark592)
3. [The executable jar format](#_bookmark593) [338](#_bookmark593)
   1. [Nested JARs](#_bookmark594) [338](#_bookmark594)

[The executable jar file structure](#_bookmark595) [338](#_bookmark595)

[The executable war file structure](#_bookmark596) [338](#_bookmark596)

* 1. [Spring Boot’s “JarFile” class](#_bookmark597) [339](#_bookmark597)

[Compatibility with the standard Java “JarFile”](#_bookmark598) [339](#_bookmark598)

* 1. [Launching executable jars](#_bookmark599) [339](#_bookmark599)

[Launcher manifest](#_bookmark600) [340](#_bookmark600)

[Exploded archives](#_bookmark601) [340](#_bookmark601)

* 1. [PropertiesLauncher Features](#_bookmark602) [340](#_bookmark602)
  2. [Executable jar restrictions](#_bookmark603) [342](#_bookmark603)

[Zip entry compression](#_bookmark604) [342](#_bookmark604)

[System ClassLoader](#_bookmark605) [342](#_bookmark605)

* 1. [Alternative single jar solutions](#_bookmark606) [342](#_bookmark606)

1. [Dependency versions](#_bookmark607) [343](#_bookmark607)

**第一部分. Spring Boot 文档**

本章节简要介绍Spring Boot参考文档。将其视为该文档的其余部分的地图。可以以线性方式阅读本参考文档，或者如果你不感兴趣，可以跳过本分内容。

# 关于文档

Spring Boot参考指南可作为html,pdf和epub文档。最新的副本可在官网获得。

本文件以副本可供自己使用，此类副本的额任何费用的情况下可以分发发给他人，，并进一步规定，不论是以书面形式还是以电子邮件方式分发，每个副本均包含本版权声明。

# 获取帮助

有关Spring Boot的问题，我们愿意提供帮助！

* [stackoverflow.com](http://stackoverflow.com/)
* [github.com/spring-projects/spring-boot/issues](https://github.com/spring-projects/spring-boot/issues).

# 第一步

如果你刚开始使用Spring Boot，或者通常是Spring，这是一个开始使用Spring Boot 的地方。

* **从头开始:** [概述](#_bookmark9) | [要求](#_bookmark10) | [安装](#_bookmark12)
* **教程:** [Part 1](#_bookmark24) | [Part 2](#_bookmark27)
* **运行示例:** [Part 1](#_bookmark31) | [Part 2](#_bookmark32)

# 使用 Spring Boot

准备好开始使用Spring Boot了吗？

Ready to actually start using Spring Boot? [我们默认已经准备好](#_bookmark34).

* **构建系统:** [Maven](#_bookmark37) | [Gradle](#_bookmark42) | [Ant](#_bookmark43) | [Starters](#_bookmark44)
* **最佳做法:** [Code Structure](#_bookmark50) | [@Configuration](#_bookmark53) | [@EnableAutoConfiguration](#_bookmark56) | [Beans and Dependency Injection](#_bookmark59)
* **运行你的代码** [IDE](#_bookmark62) | [Packaged](#_bookmark63) | [Maven](#_bookmark64) | [Gradle](#_bookmark65)
* **打包你的应用程序:** [Production jars](#_bookmark83)
* **启动Spring Boot:** [Using the CLI](#_bookmark389)

# 学习 Spring Boot 特性

需要更多关于Spring Boot的核心特性？这里都有。

* **核心特性:** [SpringApplication](#_bookmark86) | [External Configuration](#_bookmark97) | [Profiles](#_bookmark115) | [Logging](#_bookmark119)
* **Web 应用程序:** [MVC](#_bookmark130) | [Embedded Containers](#_bookmark146)
* **数据处理:** [SQL](#_bookmark165) | [NO-SQL](#_bookmark183)
* **消息:** [Overview](#_bookmark228) | [JMS](#_bookmark229)
* **测试:** [Overview](#_bookmark258) | [Boot Applications](#_bookmark261) | [Utils](#_bookmark275)
* **扩展:** [Auto-configuration](#_bookmark282) | [@Conditions](#_bookmark285)

# 发布到生产环境

当你准备将Spring Boot应用程序部署到生产环境，[技巧](#_bookmark297) 你可能会喜欢。

* **管理端点:** [Overview](#_bookmark299) | [Customization](#_bookmark300)
* **连接选项:** [HTTP](#_bookmark314) | [JMX](#_bookmark322) | [SSH](#_bookmark328)
* **监控:** [Metrics](#_bookmark336) | [Auditing](#_bookmark352) | [Tracing](#_bookmark353) | [Process](#_bookmark355)

# 高级主题

最后，我们为更高级的用户提供了一些主题。

* **部署Spring Boot应用程序:** [云部署](#_bookmark364) | [OS服务](#_bookmark379)
* **构建工具插件:** [Maven](#_bookmark408) | [Gradle](#_bookmark411)
* **附录:** [Application Properties](#_bookmark569) | [Auto-configuration classes](#_bookmark589) | [Executable Jars](#_bookmark593)

**第二部分. 入门**

如果你刚开始使用Spring Boot，或者一般来说就是Spring，那么这部分是你所需要了解的！在这里我们回答基本“what？”，“how?”,”why?”的问题。你会发现一个优美的介绍，以及安装说明。然后，我们将构建我们的第一个Spring Boot应用程序，并讨论一些核心原则。

# Spring Boot 介绍

Spring Boot 是你可以轻松创建单机的生产级别的基于Spring的可运行的应用程序。我们队Spring平台和第三方库有自己的一些做法，所以你可以从最小的功能开始。大多数Spring Boot 应用程序值需要很少的Spring配置。

你可以使用Spring Boot创建可以用java –jar或更传统的war部署来启动的应用程序。我们还提供了一个运行“spring scripts”的命令行工具。

我们的主要目标是：

* 为所有Spring开发提供一个更快，更广泛的入门体验。
* 开箱即用，但随着需求开始偏离默认值，快速开始。
* 提供大量项目(如嵌入式服务器，安全性，指标，运行状况检查，外部配置)通用的一些列非功能性功能。
* 绝对不生成代码，不需要XML配置

# 系统要求

默认情况下，Spring Boot1.5.8.RELEASE需要Java 7 和 Spring Framework 4.3.12.RELEASE或更高版本。你可以使用带有Java 6的Spring Boot以及其他一些配置。有关更多详细信息请参见84.11节如何使用Java 6 。Maven(3.2+)和Gradle 2(2.9或更高版本)和3提供了明确的构建支持。

**Tip**

虽然可以使用Java 6 或 7 的Spring Boot，但是我们通常推荐使用Java 8 。

## Servlet 容器

开箱即用支持一下嵌入式servlet容器:

|  |  |
| --- | --- |
| **Name** | **Servlet Version Java Version** |
| Tomcat 8  Tomcat 7 | 3.1 Java 7+  3.0 Java 6+ |
| Jetty 9.3 | 3.1 Java 8+ |
| Jetty 9.2 | 3.1 Java 7+ |
| Jetty 8 | 3.0 Java 6+ |
| Undertow 1.3 | 3.1 Java 7+ |

你也可以将Spring Boot应用程序部署到任何与Servlet 3.0+兼容的容器。

# 安装 Spring Boot

Spring Boot可以与经典Java开发工具一期使用，也可以作为命令行工具安装。无论如何，你将需要Java SDK V1.6 或更高版本。在空时之前，你应该检查你当前的Java安装:

$ java -version

如果你对Java开发不熟悉，或者你知识向尝试Spring Boot, 则可能需要先尝试Spring Boot CLI，否则请阅读[Spring Boot CLI](#_bookmark16) 安装说明。

**Tip**

尽管Spring Boot与Java 6兼容，但如果可能的话，应该考虑使用最新版本的Java

## Java开发人员的安装说明

你可以像使用任何标准Java库一样使用Spring Boot。只需要在你的类路径中包含相应的spring-boot.-\*.jar文件即可。Spring Boot不需要任何特殊的工具集成，所以可以使用任何IDE或文本编辑器;Spring Boot应用程序没有什么特别之处，所以你可以像运行其他Java程序一样运行和调试。

尽管你可以复制Spring Boot jars，但我们通常建议你使用支持依赖管理的构建工具(如Maven或Gradle)。

### Maven 安装

Spring Boot与Apache Maven3.2或更高版本兼容。如果你还没有安装maven，你可以按照[maven.apache.org](http://maven.apache.org/)上的说进行操作。

**Tip**

在许多操作系统上，maven可以通过包管理器来安装。如果你是OSX 用户，请尝试安装maven。ubuntu用于而已运行sudo apt-get install maven。

Spring Boot 依赖使用org.springframework.boot groupId。通常，你的maven POM文件将从spring-boot-starter-parent项目继承，并向一个或多个starter声明依赖关系。Spring Boot还提供一个可选的maven插件来创建可执行的jar文件。

这是一个典型的 pom.xml 文件:

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

**<project xmlns**=[**"http://maven.apache.org/POM/4.0.0"**](http://maven.apache.org/POM/4.0.0) **xmlns:xsi**=[**"http://www.w3.org/2001/XMLSchema-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xsi:schemaLocation**=[**"http://maven.apache.org/POM/4.0.0**](http://maven.apache.org/POM/4.0.0)[**http://maven.apache.org/xsd/maven-4.0.0.xsd"**](http://maven.apache.org/xsd/maven-4.0.0.xsd)**>**

**<modelVersion>**4.0.0**</modelVersion>**

**<groupId>**com.example**</groupId>**

**<artifactId>**myproject**</artifactId>**

**<version>**0.0.1-SNAPSHOT**</version>**

*<!-- Inherit defaults from Spring Boot -->*

**<parent>**

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

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

**<version>**1.5.8.RELEASE**</version>**

**</parent>**

*<!-- Add typical dependencies for a web application -->*

**<dependencies>**

**<dependency>**

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

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

**</dependency>**

**</dependencies>**

*<!-- Package as an executable jar -->*

**<build>**

**<plugins>**

**<plugin>**

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

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

**</plugin>**

**</plugins>**

**</build>**

**</project>**

**Tip**

spring-boot-starter-parent 是使用Spring Boot的好方法，但他可能并不适合所有的情况。有时呢可能需要从不同的父POM继承，或者你可能不喜欢我们的默认设置。有关使用import scope 的替代解决方案，请参考[POM的Spring Boot章节](#_bookmark39)

## 安装 Spring Boot CLI

Spring Boot CLI是一个命令行工具，可以使用它快速创建Spring原型。 它允许你运行Groovy脚本，这意味着你有一个数据的类Java语法，没有太多的样板代码。

你不需要使用CLI来使用Spring Boot，但它绝对是让Spring应用程序实现最快速快捷的方式。

### 手动安装

你可以从Spring软件存储库下载Spring CLI发行版：

* [spring-boot-cli-1.5.8.RELEASE-bin.zip](http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.5.8.RELEASE/spring-boot-cli-1.5.8.RELEASE-bin.zip)
* [spring-boot-cli-1.5.8.RELEASE-bin.tar.gz](http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.5.8.RELEASE/spring-boot-cli-1.5.8.RELEASE-bin.tar.gz)

最新版本 [snapshot distributions](http://repo.spring.io/snapshot/org/springframework/boot/spring-boot-cli/) 也是可以使用的。

下载之后，请按照解压缩归档中的[INSTALL.txt](https://raw.github.com/spring-projects/spring-boot/v1.5.8.RELEASE/spring-boot-cli/src/main/content/INSTALL.txt) 说明进行操作。总结：在.zip文件的bin目录下有一个spring脚本(用于Windows的spring.bat)，或可以使用带有.jar文件的java –jar(该脚本可以帮助你确保类路径设置正确)。

### 使用 SDKMAN安装!

SDKMAN!(软件开发工具包管理器)可用于管理各种二进制SDK的多个版本，包括Groovy和Spring Boot CLI。获取[SDKMAN](http://sdkman.io/)! 从sdkman.io安装Spring Boot。

$ sdk install springboot

$ spring --version

Spring Boot v1.5.8.RELEASE

如果你正在开发CLI的功能并希望轻松访问你刚刚构建的版本，请按照这些额外说明进行操作。

$ sdk install springboot dev /path/to/spring-boot/spring-boot-cli/target/spring-boot-cli-1.5.8.RELEASE- bin/spring-1.5.8.RELEASE/

$ sdk default springboot dev

$ spring --version

Spring CLI v1.5.8.RELEASE

这将安装一个名为dev实例的spring的本地实例。它指向你的目标构建位置，所以每次重建Spring Boot时，spring都是最新的。

你可以如下这样做::

$ sdk ls springboot

================================================================================

Available Springboot Versions

================================================================================

> + dev

\* 1.5.8.RELEASE

================================================================================

+ - local version

\* - installed

> - currently in use

================================================================================

### OSX Homebrew 安装

如果你在Mac上并使用Homebrew，则只需要安装Spring Boot CLI即可：

$ brew tap pivotal/tap

$ brew install springboot

Homebrew 将会将spring安装到 /usr/local/bin.

**Note**

如果你没有看到该公式，那么你的brew的安装可能会过期。值需要执行brew更新并重启。

### MacPorts 安装

如果你在Mac上并使用[MacPorts](http://www.macports.org/)，则只需安装Spring Boot CLI即可：

$ sudo port install spring-boot-cli

### 命令行完成

Spring Boot CLI提供了为[BASH](http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29) 和[zsh](http://en.wikipedia.org/wiki/Zsh) shells命令完成脚本。你可以在任何shell中获取脚本(也称为spring)的源代码，或者将其放入个人或系统范围的bash完成初始化。在Debian系统上，系统范围的焦恩在/shell-completion/bash中，当一个新的shell启动时，该目录中的所有脚本都会被执行。要手动运行脚本，例如:如果你已经安装使用SDKMAN!

$ . ~/.sdkman/candidates/springboot/current/shell-completion/bash/spring

$ spring <HIT TAB HERE>

grab help jar run test version

**Note**

如果你使用Homebrew或MacPorts安装Spring Boot CLI，则命令行完成脚本会自动在您的shell中注册。

### 快速启动 Spring CLI 示例

这是一个非常建档的web应用程序，你可以使用它来测试你的安装。创建一个名为app.groovy的文件。

@RestController

**class** ThisWillActuallyRun {

@RequestMapping("/") String home() {

***"Hello World!"***

}

}

然后简单地从一个shell运行它：

$ spring run app.groovy

**Note**

首次运行应用程序需要一段时间，因为依赖关系将会被下载。后续运行将会更快。

在你喜欢的浏览器中打开localhost:8080 ，你应该看到下面的输出。

Hello World!

## 从早期版本的Spring Boot升级

如果你是从早期版本的Spring Boot进行升级，请查看[project wiki](https://github.com/spring-projects/spring-boot/wiki)发行说明。你会发现升级说明以及每个版本的新功能和指的注意功能列表。

要升级现有的CLI安装，请使用响应的package manager命令(如brew升级)，或者如果你手动安装了CLI，请按照标准说明(standard instructions)记住更新PATH环境变量以及删除所有旧的引用。

# 开发你的第一个Spring Boot 应用程序

我们用Java开发一个简单的“Hello World！”web应用程序，重点介绍Spring Boot的一些主要特性。我们将使用Maven来构建这个项目，因为大多说IDE都支持。

**Tip**

spring.io网站包含许多使用Spring Boot 的入门指南。如果你想解决一个待定的问题，现在哪里查看。

你可以通过转到[start.spring.io](https://start.spring.io/) 并从依赖关系搜索器中选择web starter来快速执行以下步骤。这将自动商城一个新的项目结构，以便你可以立即开始编码([start coding right away](#_bookmark27))。检查文档以获取更多详细信息([documentation for more details](https://github.com/spring-io/initializr))。

在开始之前，请打开终端，检查是否安装了Java和Maven的有效版本。

$ java -version

java version "1.7.0\_51"

Java(TM) SE Runtime Environment (build 1.7.0\_51-b13)

Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)

$ mvn -v

Apache Maven 3.2.3 (33f8c3e1027c3ddde99d3cdebad2656a31e8fdf4; 2014-08-11T13:58:10-07:00) Maven home: /Users/user/tools/apache-maven-3.1.1

Java version: 1.7.0\_51, vendor: Oracle Corporation

**Note**

此示例需要在自己的文件中创建。后续的说明嘉定你已经创建了一个合适的文件夹，并且它是你的当前目录。

## 创建 POM

我们需要从创建一个maven pom.xml文件开始。pom.xml是用来构建项目的配方。打开你喜欢的文本编辑器并添加一下内容。

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

**<project xmlns**=[**"http://maven.apache.org/POM/4.0.0"**](http://maven.apache.org/POM/4.0.0) **xmlns:xsi**=[**"http://www.w3.org/2001/XMLSchema-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xsi:schemaLocation**=[**"http://maven.apache.org/POM/4.0.0**](http://maven.apache.org/POM/4.0.0)[**http://maven.apache.org/xsd/maven-4.0.0.xsd"**](http://maven.apache.org/xsd/maven-4.0.0.xsd)**>**

**<modelVersion>**4.0.0**</modelVersion>**

**<groupId>**com.example**</groupId>**

**<artifactId>**myproject**</artifactId>**

**<version>**0.0.1-SNAPSHOT**</version>**

**<parent>**

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

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

**<version>**1.5.8.RELEASE**</version>**

**</parent>**

*<!-- Additional lines to be added here... -->*

**</project>**

这应该给你一个工作的构建，你可以通过运行mvn(你可以忽略jar僵尸空的 – 没有内容被标记为包含警告)来测试它。

**Note**

此时，你可以将项目导入IDE(大多数现代Java IDE包含多maven的内置支持)。为了简单起见，我们将继续在这个例子中使用纯文本编辑器。

## 添加 classpath 依赖

Spring Boot 提供了许多“starters”，可以方便地将jar添加到你的classpath中。我们的示例应用程序已经在pom的父节点中使用了spring-boot-starter-parent。spring-boot-starter-parent是一个特别的启动器，提供了有用的maven默认值。它还提供了一个依赖管理部分，以便你可以省略版本标记获得“blessed”的依赖关系。

其他入门者知识提供开发特定类型的应用程序时可能需要的依赖关系。由于我们正在开发一个web应用程序，我们将添加一个spring-boot-start-web依赖项 – 但在此之前，让我们看看我们目前有什么。

.mvn 依赖项：tree命令打印项目依赖项的属性结构。

**<dependencies>**

**<dependency>**

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

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

**</dependency>**

**</dependencies>**

$ mvn dependency:tree

[INFO] com.example:myproject:jar:0.0.1-SNAPSHOT

你可以看到spring-boot-starter-parent本身不提供依赖关系。 让我们编辑我们的pom.xml，并在父节的下方添加spring-boot-starter-web依赖项：

如果再次运行mvn dependency：tree，则会看到现在还有许多附加依赖项，包括Tomcat Web服务器和Spring Boot本身。.

## 写 code

为了完成我们的应用程序，我们需要创建一个Java文件。 Maven默认会从src / main / java编译源代码，所以你需要创建这个文件夹结构，然后添加一个名为src / main / java / Example.java的文件：

**import** org.springframework.boot.\*;

**import** org.springframework.boot.autoconfigure.\*;

**import** org.springframework.stereotype.\*;

**import** org.springframework.web.bind.annotation.\*;

@RestController @EnableAutoConfiguration **public class** Example {

@RequestMapping("/") String home() {

**return *"Hello World!"***;

}

**public static void** main(String[] args) **throws** Exception { SpringApplication.run(Example.**class**, args);

}

}

虽然这里没有太多的代码，但还是有很多。 我们来看看重要的部分。

### @RestController 和 @RequestMapping 注解

我们的Example类的第一个注解是@RestController。 这被称为*stereotype注解*。 它为阅读代码提供了线索，对于Spring来说，这个类扮演着特定的角色。 在这种情况下，我们的类是一个web @Controller，所以Spring在处理传入的Web请求时会考虑它。

@RequestMapping注释提供了“路由”信息。 它告诉Spring，任何具有路径“/”的HTTP请求都应该映射到home方法。 @RestController注释告诉Spring将结果字符串直接返回给调用者。

**Tip**

@RestController和@RequestMapping注解是Spring MVC注释（它们不是Spring Boot特有的）。 有关更多详细信息，请参阅[MVC section](https://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#mvc)参考资料中的MVC部分。

### @EnableAutoConfiguration 注解

第二个类级注释是@EnableAutoConfiguration。 这个注解告诉Spring Boot根据你添加的jar依赖来“猜测”你将如何配置Spring。 由于spring-boot-starter-web添加了Tomcat和Spring MVC，所以自动配置会假定你正在开发一个Web应用程序并相应地设置Spring。

|  |
| --- |
| **Starters and Auto-Configuration**  自动配置旨在与“启动器”配合使用，但这两个概念并不直接相关。 您可以自由选择和选择初学者之外的jar依赖项，并且Spring Boot将尽其所能地自动配置您的应用程序。 |

### “main” 方法

我们的应用程序的最后一部分是主要的方法。 这只是一个遵循Java约定的应用程序入口点的标准方法。 我们的主要方法通过调用run来委托Spring Boot的SpringApplication类。 SpringApplication将引导我们的应用程序，从Spring开始，它将启动自动配置的Tomcat Web服务器。 我们需要将Example.class作为参数传递给run方法，以告知SpringApplication是Spring的主要组件。 args数组也被传递以暴露任何命令行参数。.

## 运行这个示例

在这一点上我们的应用程序应该工作 由于我们使用了spring-boot-starter-parent POM，所以我们有一个有用的运行目标，我们可以使用它来启动应用程序。 键入mvn spring-boot：run 运行以启动应用程序：

\_ \_

\_

.

$ mvn spring-boot:run

/\\ / '\_ \_ \_(\_)\_ \_ \ \ \ \ ( ( )\ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ )| |\_)| | | | | || (\_| | ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | / / / /

=========|\_|==============| /=/\_/\_/\_/

:: Spring Boot :: (v1.5.8.RELEASE)

....... . . .

....... . . . (log output here)

....... . . .

........ Started Example in 2.222 seconds (JVM running for 6.514)

如果你打开一个web浏览器到localhost：8080，你应该看到如下输出：

Hello World!

要正常退出应用程序点击ctrl-c。

## 创建一个可执行的 jar

让我们通过创建一个完全独立的可执行jar文件来完成我们的例子，我们可以在生产环境中运行它。 可执行jar（有时也称为“fat jars”）是包含您编译的类以及您的代码需要运行的所有jar依赖项的归档文件。

|  |
| --- |
| **Executable jars and Java**  Java不提供任何标准的方法来加载嵌套的jar文件（即jar文件本身包含在jar中）。 如果您正在分发自包含的应用程序，这可能会有问题。  为了解决这个问题，许多开发者使用“uber”jar。 “uber”jar只是将所有jar里的所有类打包成一个单一的档案。 这种方法的问题在于，很难在应用程序中看到实际使用的库。 如果在多个罐子中使用相同的文件名（但具有不同的内容），则也可能是有问题的。  Spring Boot采用了不同的方法，可以直接嵌入jar。 |

要创建一个可执行的jar文件，我们需要将spring-boot-maven-plugin添加到我们的pom.xml。 在依赖关系部分下面插入以下几行：

**<build>**

**<plugins>**

**<plugin>**

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

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

**</plugin>**

**</plugins>**

**</build>**

**Note**

spring-boot-starter-parent POM包含<executions>配置来绑定重新打包目标。 如果您不使用父POM，则需要自行声明此配置。 有关详细信息，请参阅插件文档([plugin documentation](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/usage.html) )。

保存你的pom.xml并从命令行运行mvn package：

$ mvn package

[INFO] Scanning for projects... [INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building myproject 0.0.1-SNAPSHOT

[INFO] ------------------------------------------------------------------------ [INFO] .... ..

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myproject ---

[INFO] Building jar: /Users/developer/example/spring-boot-example/target/myproject-0.0.1-SNAPSHOT.jar [INFO]

[INFO] --- spring-boot-maven-plugin:1.5.8.RELEASE:repackage (default) @ myproject --- [INFO] ------------------------------------------------------------------------ [INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

如果您查看目标目录，则应该看到myproject-0.0.1-SNAPSHOT.jar。 该文件大小应该在10 MB左右。 如果你想偷看里面，你可以使用jar tvf：

$ jar tvf target/myproject-0.0.1-SNAPSHOT.jar

您还应该在目标目录中看到一个名为myproject-0.0.1-SNAPSHOT.jar.original的小得多的文件。 这是Maven在被Spring Boot重新包装之前创建的原始jar文件。

使用 java –jar 命 令 运 行 jj a r 文 件 。

/\\ / '\_ \_ \_(\_)\_ \_ \ \ \ \ ( ( )\ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ )| |\_)| | | | | || (\_| | ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | / / / /

=========|\_|==============| /=/\_/\_/\_/

:: Spring Boot :: (v1.5.8.RELEASE)

....... . . .

....... . . . (log output here)

....... . . .

........ Started Example in 2.536 seconds (JVM running for 2.864)

\_ \_

\_

.

$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

像以前一样，要优雅地退出应用程序点击ctrl-c。

# 接下来读什么

希望本节为您提供了一些Spring Boot基础知识，并帮助您编写自己的应用程序。 如果您是面向任务的开发人员，则可能需要跳过[spring.io](https://spring.io/)，查看一些入门指南(getting started)，以解决具体的“如何使用Spring”问题; 我们也有Spring Boot特定( [*How-to*](#_bookmark438) )的操作指南文档。

Spring Boot版本([Spring Boot repository](http://github.com/spring-projects/spring-boot) )也有一些你可以运行的样本([bunch of samples](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples) )。 样本独立于代码的其余部分（也就是说，您不需要构建其余的代码来运行或使用样本）。

另外，下一个逻辑步骤是阅读[*Part III, “Using Spring Boot”*](#_bookmark34)。 如果你真的不耐烦，也可以跳过来阅读Spring Boot的特([*Spring Boot features*](#_bookmark85))。

**第三部分. 使用 Spring Boot**

本节将更详细地介绍如何使用Spring Boot。 它涵盖了构建系统，自动配置以及如何运行应用程序等主题。 我们还介绍了一些Spring Boot的最佳实践。 尽管Spring Boot没有什么特别的地方（它只是你可以使用的另一个库），但是有一些建议，如果遵循这些建议，将使开发过程变得更容易一些。

如果您刚刚开始使用Spring Boot，则可能需要先阅读[*Getting Started*](#_bookmark8) ，然后再深入本节。

# 构建 systems

强烈建议您选择支持依赖管理的构建系统([*dependency management*](#_bookmark36),)，并且可以使用发布到“Maven Central”存储库的工件。 我们建议您选择Maven或Gradle。 Spring Boot可以与其他构建系统（例如Ant）一起工作，但是它们不会得到特别好的支持。

## Dependency 管理

Spring Boot的每个发行版都提供了一个支持的依赖列表。 在实践中，您不需要为构建配置中的任何这些依赖项提供一个版本，因为Spring Boot正在为您进行管理。 当您升级Spring Boot本身时，这些依赖关系也将以一致的方式升级。

**Note**

如果您觉得有必要，您仍然可以指定一个版本并覆盖Spring Boot的建议。

策划的列表包含您可以使用Spring Boot的所有spring模块以及第三方库的精炼列表。 该列表可用作标准物料清单（[Bills of Materials (spring-boot-](#_bookmark39) [dependencies)](#_bookmark39) ，并且还提供[Maven](#_bookmark38) 和[Gradle](#_bookmark413) 的其他专用支持。

**Warning**

Spring Boot的每个版本都与Spring框架的基础版本相关联，所以我们

强烈建议您不要自行指定版本。

## Maven

Maven用户可以继承spring-boot-starter-parent项目以获得合理的默认值。 父项目提供以下功能：

* Java 1.6 作为默认的编译器级别
* UTF-8 源代码格式
* [Dependency Management section](#_bookmark36),，允许您省略继承自Spring-Boot-Dependency POM的通用依赖项的<version>标记。
* 智能 [resource filtering](https://maven.apache.org/plugins/maven-resources-plugin/examples/filter.html).
* 智能 plugin configuration ([exec plugin](http://www.mojohaus.org/exec-maven-plugin/), [surefire](http://maven.apache.org/surefire/maven-surefire-plugin/), [Git commit ID](https://github.com/ktoso/maven-git-commit-id-plugin), [shade](http://maven.apache.org/plugins/maven-shade-plugin/)).
* 智能 resource filtering for application.properties and application.yml 包括 profile-specific files (e.g. application-foo.properties and application-foo.yml)

最后一点：由于默认配置文件接受Spring样式占位符（$ {...}），Maven过滤被改为使用@ .. @占位符（您可以用Maven属性resource.delimiter覆盖）。

### 继承 starter parent

要将项目配置为从spring-boot-starter-parent继承，只需设置父项：

*<!-- Inherit defaults from Spring Boot -->*

**<parent>**

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

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

**<version>**1.5.8.RELEASE**</version>**

**</parent>**

**Note**

您应该只需要在此依赖项上指定Spring Boot版本号。 如果您导入更多的启动器，则可以安全地省略版本号。

通过该设置，您还可以通过在自己的项目中重写属性来覆盖各个依赖项。 例如，要升级到另一个Spring Data发行版，您需要将以下内容添加到您的pom.xml中

**Tip**

**<properties>**

**<spring-data-releasetrain.version>**Fowler-SR2**</spring-data-releasetrain.version>**

**</properties>**

检查 [spring-boot-dependencies pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-dependencies/pom.xml) 以获取支持的列表。

### 使用没有父 POM 的 Spring Boot

不是每个人都喜欢从spring-boot-starter-parent POM继承。 你可能有你自己的企业标准的父母，你需要使用，或者你可能只是喜欢显式声明所有的Maven配置。

如果您不想使用spring-boot-starter-parent，那么仍然可以通过使用scope = import dependency来保留依赖关系管理（而不是插件管理）的。

**<dependencyManagement>**

**<dependencies>**

**<dependency>**

*<!-- Import dependency management from Spring Boot -->*

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

**<artifactId>**spring-boot-dependencies**</artifactId>**

**<version>**1.5.8.RELEASE**</version>**

**<type>**pom**</type>**

**<scope>**import**</scope>**

**</dependency>**

**</dependencies>**

**</dependencyManagement>**

如上所述，该设置不允许您使用属性覆盖单个依赖项。 为了达到同样的结果，你需要在spring-boot-dependencies条目之前在项目的dependencyManagement中添加一个条目。 例如，要升级到另一个Spring Data发行版，您需要将以下内容添加到您的pom.xml中。

**<dependencyManagement>**

**<dependencies>**

*<!-- Override Spring Data release train provided by Spring Boot -->*

**<dependency>**

**<groupId>**org.springframework.data**</groupId>**

**<artifactId>**spring-data-releasetrain**</artifactId>**

**<version>**Fowler-SR2**</version>**

**<scope>**import**</scope>**

**<type>**pom**</type>**

**</dependency>**

**<dependency>**

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

**<artifactId>**spring-boot-dependencies**</artifactId>**

**<version>**1.5.8.RELEASE**</version>**

**<type>**pom**</type>**

**<scope>**import**</scope>**

**</dependency>**

**</dependencies>**

**</dependencyManagement>**

**Note**

在上面的例子中，我们指定了一个BOM，但是任何依赖类型都可以被覆盖。

### 更改 Java 版本

spring-boot-starter-parent选择相当保守的Java兼容性。 如果您想遵循我们的建议并使用较新的Java版本，则可以添加一个java.version属性：

**<properties>**

**<java.version>**1.8**</java.version>**

**</properties>**

### 使用 Spring Boot Maven 插件

Spring Boot包含一个Maven插件，可以将项目打包为可执行的jar文件。 如果要使用它，请将插件添加到<plugins>部分

**<build>**

**<plugins>**

**<plugin>**

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

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

**</plugin>**

**</plugins>**

**</build>**

**Note**

如果你使用Spring Boot starter parent pom POM，你只需要添加插件，除非你想改变在父代中定义的设置，否则不需要进行配置。

## Starters

Starters是一套方便的依赖描述符，可以包含在应用程序中。 您可以获得所需的所有Spring及相关技术的一站式服务，无需搜索示例代码，也不需要粘贴大量依赖描述符。 例如，如果你想开始使用Spring和JPA来访问数据库，只需在你的项目中加入spring-boot-starter-data-jpa依赖项，你就可以开始了。

初学者包含很多依赖项，您需要快速启动并运行一个项目，并使用一组支持的传递依赖项。

|  |
| --- |
| **What’s in a name**  所有官方首发者都遵循类似的命名模式; spring-boot-starter- \*，其中\*是特定类型的应用程序。 这种命名结构旨在帮助您找到启动器。 许多IDE中的Maven集成允许您按名称搜索依赖项。 例如，安装适当的Eclipse或STS插件后，只需在POM编辑器中点击ctrl-space并键入“spring-boot-starter”即可获得完整列表。  正如创建自己的启动器([Creating your own starter](#_bookmark292) )部分中所解释的，第三方启动器不应该以spring-boot为开始，因为它是为官方Spring Boot工件保留的。 acme的第三方初学者通常被命名为acme-spring-boot-starter。 |

下面的应用程序启动器是由Spring Boot提供的

org.springframework.boot group:

*Table 13.1. Spring Boot application starters*

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Pom** |
| spring-boot-starter | 核心启动器，包括自动启动器， | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter/pom.xml) |
|  | 配置支持，日志记录 |  |
|  | 和YAML |  |
| spring-boot-starter- activemq | 使用Apache ActiveMQ启动JMS消息传递 | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-activemq/pom.xml) |
| spring-boot-starter- amqp | 使用Spring AMQP和Rabbitmq | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-amqp/pom.xml) |
| spring-boot-starter-aop | 面向切面 | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-aop/pom.xml) |
|  | 用Spring AOP编程 |  |
|  | 和AspectJ |  |
| spring-boot-starter- | JMS消息传递 | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-artemis/pom.xml) |
| artemis | 使用Apache Artemis |  |

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Pom** |
| spring-boot-starter- | Starter for using Spring Batch | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-batch/pom.xml) |
| batch |  |  |
| spring-boot-starter- | Starter for using Spring | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-cache/pom.xml) |
| cache | Framework’s caching support |  |
| spring-boot-starter- cloud-connectors | Starter for using Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-cloud-connectors/pom.xml) |
| spring-boot-starter- data-cassandra | Starter for using Cassandra distributed database and Spring Data Cassandra | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-cassandra/pom.xml) |
| spring-boot-starter- data-couchbase | Starter for using Couchbase document-oriented database and Spring Data Couchbase | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-couchbase/pom.xml) |
| spring-boot-starter- data-elasticsearch | Starter for using Elasticsearch search and analytics engine and Spring Data Elasticsearch | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-elasticsearch/pom.xml) |
| spring-boot-starter- data-gemfire | Starter for using GemFire distributed data store and Spring Data GemFire | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-gemfire/pom.xml) |
| spring-boot-starter- | Starter for using Spring Data | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-jpa/pom.xml) |
| data-jpa | JPA with Hibernate |  |
| spring-boot-starter- | Starter for using Spring Data | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-ldap/pom.xml) |
| data-ldap | LDAP |  |
| spring-boot-starter- data-mongodb | Starter for using MongoDB document-oriented database and Spring Data MongoDB | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-mongodb/pom.xml) |
| spring-boot-starter- data-neo4j | Starter for using Neo4j graph database and Spring Data Neo4j | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-neo4j/pom.xml) |
| spring-boot-starter- data-redis | Starter for using Redis key- value data store with Spring Data Redis and the Jedis client | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-redis/pom.xml) |
| spring-boot-starter- data-rest | Starter for exposing Spring Data repositories over REST using Spring Data REST | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-rest/pom.xml) |
| spring-boot-starter- data-solr | Starter for using the Apache Solr search platform with Spring Data Solr | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-data-solr/pom.xml) |

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Pom** |
| spring-boot-starter- freemarker | Starter for building MVC web applications using FreeMarker views | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-freemarker/pom.xml) |
| spring-boot-starter- groovy-templates | Starter for building MVC web applications using Groovy Templates views | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-groovy-templates/pom.xml) |
| spring-boot-starter- hateoas | Starter for building hypermedia- [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-hateoas/pom.xml) based RESTful web application  with Spring MVC and Spring HATEOAS | |
| spring-boot-starter- integration | Starter for using Spring Integration | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-integration/pom.xml) |
| spring-boot-starter- jdbc | Starter for using JDBC with the [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-jdbc/pom.xml) Tomcat JDBC connection pool | |
| spring-boot-starter- | Starter for building RESTful | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-jersey/pom.xml) |
| jersey | web applications using JAX-RS | |
|  | and Jersey. An alternative to | |
|  | [spring-boot-starter-web](#_bookmark47) | |
| spring-boot-starter- | Starter for using jOOQ to | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-jooq/pom.xml) |
| jooq | access SQL databases. An | |
|  | alternative to [spring-boot-](#_bookmark45) | |
|  | [starter-data-jpa](#_bookmark45) or | |
|  | [spring-boot-starter-](#_bookmark46) | |
|  | [jdbc](#_bookmark46) | |
| spring-boot-starter- | Starter for JTA transactions | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-jta-atomikos/pom.xml) |
| jta-atomikos | using Atomikos | |
| spring-boot-starter- | Starter for JTA transactions | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-jta-bitronix/pom.xml) |
| jta-bitronix | using Bitronix | |
| spring-boot-starter- jta-narayana | Spring Boot Narayana JTA Starter | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-jta-narayana/pom.xml) |
| spring-boot-starter- mail | Starter for using Java Mail and Spring Framework’s email sending support | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-mail/pom.xml) |
| spring-boot-starter- mobile | Starter for building web applications using Spring Mobile | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-mobile/pom.xml) |
| spring-boot-starter- mustache | Starter for building MVC web applications using Mustache views | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-mustache/pom.xml) |

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Pom** |
| spring-boot-starter- | Starter for using Spring Security [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-security/pom.xml) | |
| security |  | |
| spring-boot-starter- | Starter for using Spring Social | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-social-facebook/pom.xml) |
| social-facebook | Facebook | |
| spring-boot-starter- | Stater for using Spring Social | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-social-linkedin/pom.xml) |
| social-linkedin | LinkedIn | |
| spring-boot-starter- | Starter for using Spring Social | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-social-twitter/pom.xml) |
| social-twitter | Twitter | |
| spring-boot-starter- | Starter for testing Spring Boot | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-test/pom.xml) |
| test | applications with libraries | |
|  | including JUnit, Hamcrest and | |
|  | Mockito | |
| spring-boot-starter- thymeleaf | Starter for building MVC web applications using Thymeleaf views | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-thymeleaf/pom.xml) |
| spring-boot-starter- validation | Starter for using Java Bean Validation with Hibernate Validator | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-validation/pom.xml) |
| spring-boot-starter-web | Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-web/pom.xml) |
| spring-boot-starter- web-services | Starter for using Spring Web Services | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-web-services/pom.xml) |
| spring-boot-starter- | Starter for building WebSocket [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-websocket/pom.xml) | |
| websocket | applications using Spring | |
|  | Framework’s WebSocket | |
|  | support | |

In addition to the application starters, the following starters can be used to add [*production ready*](#_bookmark297) features:

*Table 13.2. Spring Boot production starters*

|  |  |
| --- | --- |
| **Name**  spring-boot-starter- actuator | **Description Pom**  Starter for using Spring Boot’s [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-actuator/pom.xml) Actuator which provides  production ready features to help you monitor and manage your application |
| spring-boot-starter- remote-shell | Starter for using the CRaSH [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-remote-shell/pom.xml) remote shell to monitor and |

|  |  |
| --- | --- |
| **Name** | **Description Pom** |
|  | manage your application over SSH. Deprecated since 1.5 |

Finally, Spring Boot also includes some starters that can be used if you want to exclude or swap specific technical facets:

*Table 13.3. Spring Boot technical starters*

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Pom** |
| spring-boot-starter- jetty | Starter for using Jetty as the embedded servlet container. An alternative to [spring-boot-](#_bookmark49) [starter-tomcat](#_bookmark49) | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-jetty/pom.xml) |
| spring-boot-starter- log4j2 | Starter for using Log4j2 for logging. An alternative to | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-log4j2/pom.xml) |
|  | [spring-boot-starter-](#_bookmark48) [logging](#_bookmark48) | |
| spring-boot-starter- logging | Starter for logging using Logback. Default logging starter | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-logging/pom.xml) |
| spring-boot-starter- | Starter for using Tomcat as the | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-tomcat/pom.xml) |
| tomcat | embedded servlet container. | |
|  | Default servlet container starter | |
|  | used by [spring-boot-](#_bookmark47) | |
|  | [starter-web](#_bookmark47) | |
| spring-boot-starter- undertow | Starter for using Undertow as the embedded servlet container. An alternative to | [Pom](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-starters/spring-boot-starter-undertow/pom.xml) |
|  | [spring-boot-starter-](#_bookmark49) [tomcat](#_bookmark49) | |

**Tip**

有关其他社区贡献者的列表，请参阅GitHub上的spring-boot-starters模块中的[README file](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/README.adoc) 。

# 构建你的 code

Spring Boot不需要任何特定的代码布局，但是，有一些最佳实践可以提供帮助。

## 使用 默认的 package

当一个类不包含包声明时，它被认为是在“默认包”中。 通常不鼓励使用“默认软件包”，应该避免使用“默认软件包”。 对于使用@ComponentScan，@EntityScan或@SpringBootApplication注释的Spring Boot应用程序来说，这可能会导致特定的问题，因为每个jar的每个类都将被读取。

**Tip**

我们建议您遵循Java推荐的软件包命名约定，并使用反向域名（例如com.example.project）。

## 加载 main application class

我们通常建议您将主应用程序类放在其他类的根包中。 @EnableAutoConfiguration注释通常放在主类上，它隐式地为某些项目定义了一个基本的“搜索包”。 例如，如果您正在编写JPA应用程序，则@EnableAutoConfiguration注释类的包将用于搜索@Entity项目。

使用根包也允许使用@ComponentScan注释而不需要指定basePackage属性。 如果您的主类位于根包中，也可以使用@SpringBootApplication注释。

这是一个典型的布局：

com

+- example

+- myproject

+- Application.java

|

+- domain

| +- Customer.java

| +- CustomerRepository.java

|

+- service

| +- CustomerService.java

|

+- web

+- CustomerController.java

The Application.java file would declare the main method, along with the basic @Configuration.

**package** com.example.myproject;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.EnableAutoConfiguration;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

@Configuration @EnableAutoConfiguration @ComponentScan

**public class** Application {

**public static void** main(String[] args) { SpringApplication.run(Application.**class**, args);

}

}

# 配置 classes

Spring Boot支持基于Java的配置。 虽然可以使用XML源调用SpringApplication.run（），但我们通常建议您的主要来源是@Configuration类。 通常，定义主要方法的类也是主要@Configuration的一个好选择。

**Tip**

在互联网上已经发布了许多使用XML配置的Spring配置示例。 如果可能，请始终尝试使用基于Java的等效配置。 搜索Enable\*注释可以是一个很好的起点。

## 导入其他配置类

你不需要把所有的@Configuration放到一个类中。 @Import注释可用于导入其他配置类。 或者，您可以使用@ComponentScan自动获取所有Spring组件，包括@Configuration类。.

## 导入 XML 配置

如果您绝对必须使用基于XML的配置，我们建议您仍以@Configuration类开头。 然后您可以使用额外的@ImportResource注解来加载XML配置文件。

# Auto-configuration

Spring Boot自动配置会尝试根据您添加的jar依赖项自动配置您的Spring应用程序。 例如，如果HSQLDB在您的类路径中，并且您没有手动配置任何数据库连接Bean，那么我们将自动配置一个内存数据库。

您需要通过添加@EnableAutoConfiguration或者选择加入自动配置

@SpringBootApplication 或 @Configuration 。.

**Tip**

您应该只添加一个@EnableAutoConfiguration注释。 我们通常建议您将其添加到您的主要@Configuration类。

## 主键替代 auto-configuration

自动配置是非侵入性的，在任何时候您都可以开始定义自己的配置来替换自动配置的特定部分。 例如，如果添加自己的DataSource bean，则默认的嵌入式数据库支持将退出。

如果您需要了解当前正在应用的自动配置，以及为什么使用--debug开关启动您的应用程序。 这将启用选择核心记录器的调试日志，并将自动配置报告记录到控制台。

## 禁用特定的 auto-configuration

如果您发现正在应用您不需要的特定自动配置类，则可以使用@EnableAutoConfiguration的exclude属性来禁用它们。

**import** org.springframework.boot.autoconfigure.\*; **import** org.springframework.boot.autoconfigure.jdbc.\*; **import** org.springframework.context.annotation.\*;

@Configuration @EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class}) **public class** MyConfiguration {

}

如果类不在类路径中，则可以使用注释的excludeName属性，并指定完全限定的名称。 最后，您还可以通过spring.autoconfigure.exclude属性来控制自动配置类的列表。

**Tip**

您可以在注释级别和使用属性中定义排除。

# Spring Beans 和依赖注入

您可以自由使用任何标准的Spring框架技术来定义您的bean及其注入的依赖关系。 为了简单起见，我们经常发现使用@ComponentScan来查找bean，并结合使用@Autowired构造函数注入效果很好。

如果按照上面的建议构建代码（在根包中定位应用程序类），则可以添加@ComponentScan而不带任何参数。 所有的应用程序组件（@Component，@Service，@Repository，@Controller等）都将被自动注册为Spring Bean。

这里是一个示例@Service Bean，它使用构造函数注入来获取所需的RiskAssessor bean。

**package** com.example.service;

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

**import** org.springframework.stereotype.Service;

@Service

**public class** DatabaseAccountService **implements** AccountService {

**private final** RiskAssessor riskAssessor; @Autowired

**public** DatabaseAccountService(RiskAssessor riskAssessor) {

**this**.riskAssessor = riskAssessor;

}

*// ...*

}

而如果一个bean有一个构造函数，你可以省略@Autowired。

@Service

**public class** DatabaseAccountService **implements** AccountService {

**private final** RiskAssessor riskAssessor;

**public** DatabaseAccountService(RiskAssessor riskAssessor) {

**this**.riskAssessor = riskAssessor;

}

*// ...*

}

**Tip**

请注意，如何使用构造函数注入允许riskAssessor字段被标记为final，表明它不能被随后更改。

# 使用 @SpringBootApplication 注解

许多Spring Boot开发人员总是使用@Configuration，@EnableAutoConfiguration和@ComponentScan注解其主类。 由于这些注释经常一起使用（特别是如果您遵循以上最佳实践[best practices](#_bookmark50) ），Spring Boot提供了一种方便的@SpringBootApplication替代方法。

@SpringBootApplication注释等价于使用@Configuration，@EnableAutoConfiguration和@ComponentScan及其默认属性：

**package** com.example.myproject;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication *// same as @Configuration @EnableAutoConfiguration @ComponentScan*

**public class** Application {

**public static void** main(String[] args) { SpringApplication.run(Application.**class**, args);

}

}

**Note**

@SpringBootApplication还提供为@EnableAutoConfiguration 和 @ComponentScan注解

别名定制的属性

# 运行你的应用程序

将应用程序打包为jar并使用嵌入式HTTP服务器的最大优点之一就是您可以像运行其他应用程序一样运行应用程序。 调试Spring Boot应用程序也很容易; 你不需要任何特殊的IDE插件或扩展。

**Note**

本节仅介绍基于jar的打包。如果您选择将应用程序打包为war文件，则应参考您的服务器和IDE文档。

## 从一个 IDE 运行

您可以从IDE运行Spring Boot应用程序作为简单的Java应用程序，但是，首先您需要导入项目。 导入步骤取决于您的IDE和构建系统。 大多数IDE可以直接导入Maven项目，例如Eclipse用户可以从File菜单中选择Import ...Existing Maven Projects。

如果不能直接将项目导入到IDE中，则可以使用构建插件生成IDE元数据。 Maven包含[Eclipse](http://maven.apache.org/plugins/maven-eclipse-plugin/) 和 [IDEA](http://maven.apache.org/plugins/maven-idea-plugin/)的插件; Gradle为各种IDE提供插件[various IDEs](https://docs.gradle.org/2.14.1/userguide/userguide.html).。

**Tip**

如果您不小心运行了两次Web应用程序，则会看到“端口已被使用”错误。 STS用户可以使用“重新启动”按钮而不是“运行”来确保关闭任何现有的实例。

## 打包运行应用程序

如果您使用Spring Boot Maven或Gradle插件创建可执行jar，则可以使用java -jar运行应用程序。 例如：

$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

也可以运行打包的应用程序并启用远程调试支持。 这使您可以将调试器附加到打包的应用程序：

$ java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n \

-jar target/myproject-0.0.1-SNAPSHOT.jar

## 使用 Maven 插件

Spring Boot Maven插件包含一个可用于快速编译和运行应用程序的运行目标。 应用程序以分解形式运行，就像在IDE中一样。

$ mvn spring-boot:run

您可能还想使用有用的操作系统环境变量：

$ export MAVEN\_OPTS=-Xmx1024m -XX:MaxPermSize=128M

## 使用 Gradle 插件

Spring Boot Gradle插件还包含一个bootRun任务，可用于以分解形式运行您的应用程序。 无论何时导入spring-boot-gradle-plugin，都会添加bootRun任务：

$ gradle bootRun

您可能也想使用这个有用的操作系统环境变量：

$ export JAVA\_OPTS=-Xmx1024m -XX:MaxPermSize=128M

## 热部署

由于Spring Boot应用程序只是普通的Java应用程序，所以JVM热插拔应该是开箱即用的。 JVM热交换在某种程度上受限于它可以替换的字节码，为了获得更完整的解决方案，可以使用[JRebel](http://zeroturnaround.com/software/jrebel/) 或[Spring Loaded](https://github.com/spring-projects/spring-loaded) 项目。 spring-boot-devtools模块还包括对快速应用程序重新启动的支持。

有关详细信息，请参阅下面的第20章开发人员工具([Chapter 20, *Developer tools*](#_bookmark67) )部分和热插拔“操作方法”

([Hot swapping “How-to”](#_bookmark537) )。

# 开发者工具

Spring Boot包含一组额外的工具，可以使应用程序开发体验更愉快。 spring-boot-devtools模块可以包含在任何项目中以提供额外的开发时间功能。 要包含devtools支持，只需将模块依赖关系添加到您的构建：

**Maven.**

**<dependencies>**

**<dependency>**

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

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

**<optional>**true**</optional>**

**</dependency>**

**</dependencies>**

**Gradle.**

dependencies {

compile(***"org.springframework.boot:spring-boot-devtools"***)

}

**Note**

运行完整打包的应用程序时，开发人员工具会自动禁用 如果您的应用程序是使用java -jar启动的，或者如果它是使用特殊的类加载器启动的，那么它就被认为是“生产应用程序”。 将依赖关系标记为可选项是一种最佳实践，可以防止devtools通过项目传递到其他模块。 Gradle不支持可选的依赖关系，因此您可能希望在此期间看看[propdeps-plugin](https://github.com/spring-projects/gradle-plugins/tree/master/propdeps-plugin) 插件。

**Tip**

重新打包的档案在默认情况下不包含devtools。 如果您想使用某些远程devtools功能([certain remote devtools](#_bookmark79) [feature](#_bookmark79))，则需要禁用excludeDevtools构建属性以包含它。 该属性支持Maven和Gradle插件。

## 默认属性值

Spring Boot支持的一些库使用缓存来提高性能。 例如，模板引擎([template engines](#_bookmark138) )将缓存已编译的模板，以避免重复解析模板文件。 此外，Spring MVC可以在服务静态资源时将HTTP缓存头添加到响应中。

虽然缓存在生产中非常有益，但在开发过程中可能会产生反作用，使您无法看到应用程序中刚才所做的更改。 由于这个原因，spring-boot-devtools默认会禁用这些缓存选项。

缓存选项通常由您的application.properties文件中的设置进行配置。 例如，Thymeleaf提供了spring.thymeleaf.cache属性。 而不需要手动设置这些属性，spring-boot-devtools模块将自动应用合理的开发时间配置。

**Tip**

有关应用的属性的完整列表，请参阅[DevToolsPropertyDefaultsPostProcessor](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/env/DevToolsPropertyDefaultsPostProcessor.java)。

## 自动重启

使用spring-boot-devtools的应用程序将在类路径上的文件发生更改时自动重启。 在IDE中工作时，这是一个非常有用的功能，因为它为代码更改提供了一个非常快速的反馈循环。 默认情况下，将监视指向文件夹的类路径中的任何条目以进行更改。 请注意，某些资源（如静态资产和视图模板）不需要重新启动应用程序(s [do not need to restart](#_bookmark71) [the application](#_bookmark71))。

|  |
| --- |
| **触发重启**  **由于DevTools监视类路径资源，触发重启的唯一方法是更新类路径。 导致类路径更新的方式取决于您使用的IDE。 在Eclipse中，保存修改后的文件将导致类路径更新并触发重启。 在IntelliJ IDEA中，构建项目（Build - > Make Project）将具有相同的效果。** |

**Note**

您也可以通过受支持的构建插件（即Maven和Gradle）启动您的应用程序，只要启用了分叉功能，因为DevTools需要隔离的应用程序类加载器才能正常运行。 当Gradle和Maven在类路径中检测到DevTools时，默认会这样做。

**Tip**

与LiveReload一起使用时，自动重新启动的效果非常好。 详情请参阅下文([See below](#_bookmark77) )。 如果使用JRebel，自动重新启动将被禁用，以支持动态类重新加载。 其他devtools功能（如LiveReload和属性覆盖）仍然可以使用。

**Note**

DevTools依靠应用程序上下文的关闭挂起在重新启动期间关闭它。 如果您禁用了关闭挂起（SpringApplication.setRegisterShutdownHook（false）），它将无法正常工作。

**Note**

当确定类路径上的条目在更改时会触发重新启动时，DevTools会自动忽略名为spring-boot，spring-boot-devtools，spring-boot-autoconfigure，spring-boot-actuator和spring-boot-starter的项目。

**Note**

DevTools需要自定义ApplicationContext使用的ResourceLoader：如果你的应用程序已经提供了一个，它将被打包。 不支持直接覆盖ApplicationContext上的getResource方法。

|  |
| --- |
| 重新启动vs重新加载  Spring Boot提供的重启技术通过使用两个类加载器来工作。 不改变的类（例如来自第三方jar的类）被加载到基类加载器中。 您正在开发的类将加载到重启类加载器中。 当应用程序重新启动时，重启classloader被丢弃，并创建一个新的。 这种方法意味着应用程序重新启动通常比“冷启动”快得多，因为基类加载器已经可用并且已经被填充了。  如果您发现重新启动对于您的应用程序来说不够快，或者遇到类加载问题，则可以考虑从ZeroTurnaround中重新加载技术，例如[JRebel](http://zeroturnaround.com/software/jrebel/) 。 这些工作通过重写类，因为他们被加载，使他们更容易重新加载。 [Spring](https://github.com/spring-projects/spring-loaded) [Loaded](https://github.com/spring-projects/spring-loaded)提供了另一种选择，但是它不支持许多框架，并且没有商业支持。 |

### 排除资源

某些资源不一定需要在更改时触发重新启动。 例如，Thymeleaf模板可以就地编辑。 默认情况下，更改/ META-INF / maven，/ META-INF / resources，/ resources，/ static，/ public或/ templates中的资源不会触发重新启动，但会触发实时重新加载([live reload](#_bookmark77))。 如果你想自定义这些排除，你可以使用spring.devtools.restart.exclude属性。 例如，要仅排除/ static和/ public，您可以设置以下内容：

spring.devtools.restart.exclude=static/\*\*,public/\*\*

**Tip**

如果你想保持这些默认值，并添加额外的排除，使用

spring.devtools.restart.additional-exclude属性。

### 监听额外的路径

您可能希望在更改不在类路径中的文件时重新启动或重新加载应用程序。 为此，请使用spring.devtools.restart.additional- paths属性来配置其他路径以监视更改。 您可以使用上述的spring.devtools.restart.exclude属性来控制额外路径[described above](#_bookmark71)下的更改是否会触发完全重新启动或仅实时重新加载[live reload](#_bookmark77).。

### 禁用重启

如果您不想使用重新启动功能，则可以使用spring.devtools.restart.enabled属性将其禁用。 在大多数情况下，你可以在你的application.properties中设置它（这将仍然初始化重启类加载器，但它不会监视文件的变化）。

如果您需要完全禁用重新启动支持，例如，因为它不适用于特定的库，则需要在调用SpringApplication.run（...）之前设置System属性。 例如

**public static void** main(String[] args) { System.setProperty(***"spring.devtools.restart.enabled"***, ***"false"***); SpringApplication.run(MyApp.**class**, args);

}

### 使用一个触发文件

如果您使用连续编译已更改文件的IDE，则可能只希望在特定时间触发重新启动。 要做到这一点，你可以使用“触发文件”，这是一个特殊的文件，当你想要实际触发重新启动检查时必须修改。 只更改文件会触发检查，只有Devtools检测到必须执行某些操作时才会重新启动。 触发文件可以手动更新，也可以通过IDE插件进行更新。

要使用触发器文件，请使用spring.devtools.restart.trigger-file属性。

**Tip**

您可能需要将spring.devtools.restart.trigger-file设置为全局设置([global setting](#_bookmark78) )，以便所有项目的行为方式相同。

### 自定义重启类 classloader

如上面的[Restart vs Reload](#_bookmark70)部分所述，重新启动功能是通过使用两个类加载器来实现的。 对于大多数应用程序来说，这种方法运行良好，但有时会导致类加载问题。

默认情况下，IDE中的任何打开的项目都将使用“重新启动”类加载器以及任何常规的加载器加载

.jar文件将使用“base”类加载器加载。 如果您使用多模块项目，而不是将每个模块导入到IDE中，

则可能需要自定义项目。 要做到这一点，你可以创建一个META-INF / spring-devtools.properties文件。

spring-devtools.properties文件可以包含restart.exclude。 和restart.include。 前缀属性。 include元素是应该被拉入到“重启”类加载器中的项目，排除元素是应该被下推到“基本”类加载器中的项目。 该属性的值是一个将应用于类路径的正则表达式模式。

例如:

**restart.exclude.companycommonlibs**=/mycorp-common-[\\w-]+\.jar **restart.include.projectcommon**=/mycorp-myproj-[\\w-]+\.jar

**Note**

所有的配置钥匙必须是唯一的。 只要属性以restart.include开头。 要么

restart.exclude。 这将被考虑。

**Tip**

所有类路径中的META-INF / spring-devtools.properties都将被加载。 您可以将文件打包到项目中，也可以打包到项目使用的库中。

### 已知限制

对于使用标准ObjectInputStream进行反序列化的对象，重新启动功能无法正常工作。 如果您需要反序列化数据，则可能需要将Spring的ConfigurableObjectInputStream与Thread.currentThread（）。getContextClassLoader（）一起使用。

不幸的是，有些第三方库反序列化，而不考虑上下文类加载器。 如果您发现这样的问题，您需要向原作者请求修复。

## 实时重新加载

spring-boot-devtools模块包含一个嵌入式LiveReload服务器，当资源发生变化时，可用于触发浏览器刷新。 LiveReload浏览器扩展可从[livereload.com](http://livereload.com/extensions/)的Chrome，Firefox和Safari免费获得。

如果您不想在应用程序运行时启动LiveReload服务器，则可以设置

spring.devtools.livereload.enabled属性为false。

**Note**

一次只能运行一个LiveReload服务器。 在开始您的应用程序之前，请确保没有其他LiveReload服务器正在运行。 如果您从IDE启动多个应用程序，则只有第一个应用程序支持LiveReload。

## 全局设置

您可以通过将一个名为.spring-boot- devtools.properties的文件添加到您的$ HOME文件夹来配置全局devtools设置（请注意文件名以“.”开头）。 添加到此文件的任何属性都将应用于使用devtools的计算机上的所有Spring Boot应用程序。 例如，要将重新启动配置为始终使用触发器文件([trigger file](#_bookmark74))，可以添加以下内容：

**~/.spring-boot-devtools.properties.**

**spring.devtools.reload.trigger-file**=.reloadtrigger

## 远程应用程序

Spring Boot开发人员工具不仅限于本地开发。 您还可以在远程运行应用程序时使用多个功能。 远程支持是可选的，为了启用它，您需要确保devtools包含在重新打包的归档中：

**<build>**

**<plugins>**

**<plugin>**

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

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

**<configuration>**

**<excludeDevtools>**false**</excludeDevtools>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

然后你需要设置一个spring.devtools.remote.secret属性，例如

**spring.devtools.remote.secret**=mysecret

**Warning**

在远程应用程序上启用spring-boot-devtools存在安全风险。 您不应该在生产部署上启用支持。

远程devtools支持分两部分提供; 有一个接受连接的服务器端点以及您在IDE中运行的客户端应用程序。 当设置了spring.devtools.remote.secret属性时，服务器组件会自动启用。 客户端组件必须手动启动。

### 运行远程客户端应用程序

远程客户端应用程序旨在从您的IDE中运行。 您需要使用与您要连接的远程项目相同的类路径来运行org.springframework.boot.devtools.RemoteSpringApplication。 传递给应用程序的非选项参数应该是您要连接到的远程URL。

例如，如果您使用的是Eclipse或STS，并且您已经将一个名为my-app的项目部署到Cloud Foundry，则可以执行以下操作：

* 从“运行”菜单中选择“运行配置...”。
* 创建一个新的Java应用程序“启动配置”。
* 浏览我的应用程序项目。
* 使用org.springframework.boot.devtools.RemoteSpringApplication作为主类。
* 将https://myapp.cfapps.io添加到程序参数（或任何远程URL）。

正在运行的远程客户端将如下所示：

Started RemoteSpringApplication in 0.74 seconds (JVM running for 1.105)

:

main] o.s.b.devtools.RemoteSpringApplication

:

main] o.s.b.d.a.OptionalLiveReloadServer

2015-06-10 18:25:07.074 INFO 14938 --- [

LiveReload server is running on port 35729 2015-06-10 18:25:07.130 INFO 14938 --- [

Starting RemoteSpringApplication on pwmbp with PID 14938 (/Users/pwebb/projects/spring-boot/code/ spring-boot-devtools/target/classes started by pwebb in /Users/pwebb/projects/spring-boot/code/spring- boot-samples/spring-boot-sample-devtools)

2015-06-10 18:25:06.671 INFO 14938 --- [ main] s.c.a.AnnotationConfigApplicationContext : Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@2a17b7b6: startup date [Wed Jun 10 18:25:06 PDT 2015]; root of context hierarchy

2015-06-10 18:25:07.043 WARN 14938 --- [ main] o.s.b.d.r.c.RemoteClientConfiguration : The

connection to http://localhost:8080 is insecure. You should use a URL starting with 'https://'.

main] o.s.b.devtools.RemoteSpringApplication :

2015-06-10 18:25:06.632 INFO 14938 --- [

' | | . |\_| |\_|\_| |\_\ , | |\_|\_\ |\_|\_|\_\ /\ \ |/ / / /

=========|\_|==============| /===================================/\_/\_/\_/

:: Spring Boot Remote :: 1.5.8.RELEASE

/ -\_) ' \/ \_ \ \_/ -\_) ) ) ) )

| \_ \ \_ | |\_ \ \ \ \

\\/ )| |\_)| | | | | || (\_| []::::::[]

\_ \_

\ \ \ \

\_

/\\ / '\_ \_ \_(\_)\_ \_

( ( )\ | '\_ | '\_| | '\_ \/ \_` |

\_

.

**Note**

由于远程客户端使用与实际应用程序相同的类路径，因此可以直接读取应用程序属性。 这是如何读取spring.devtools.remote.secret属性并将其传递给服务器进行身份验证。

**Tip**

总是建议使用https：//作为连接协议，以便流量被加密，密码不能被拦截。

**Tip**

如果您需要使用代理来访问远程应用程序，请配置spring.devtools.remote.proxy.host和spring.devtools.remote.proxy.port属性。

### 远程更新

远程客户端将以与本地重启相同的方式监视您的应用程序类路径的更改。 任何更新的资源将被推送到远程应用程序，并（如果需要）触发重新启动。 如果您正在迭代使用您本地没有的云服务的功能，这可能会非常有帮助。 通常远程更新和重新启动比完整的重建和部署周期快得多。

**Note**

只有远程客户端正在运行时才监视文件。 如果在启动远程客户端之前更改文件，则不会将其推送到远程服务器。

### 远程调试隧道

Java远程调试在诊断远程应用程序的问题时非常有用。 不幸的是，当您的应用程序部署在数据中心之外时，并不总是可以启用远程调试。 如果您使用基于容器的技术（例如Docker），则远程调试也可能会非常棘手。

为了帮助解决这些限制，devtools支持通过HTTP隧道传输远程调试流量。 远程客户端在端口8000上提供本地服务器，您可以将其附加到远程调试器。 建立连接后，调试流量将通过HTTP发送到远程应用程序。 如果你想使用不同的端口，你可以使用spring.devtools.remote.debug.local-port属性。

您需要确保您的远程应用程序在启用远程调试的情况下启动。 这通常可以通过配置JAVA\_OPTS来实现。 例如，使用Cloud Foundry，您可以将以下内容添加到您的manifest.yml中：

*---*

**env**:

**JAVA\_OPTS**: ***"-Xdebug -Xrunjdwp:server=y,transport=dt\_socket,suspend=n"***

**Tip**

请注意，您不需要将address = NNNN选项传递给-Xrunjdwp。 如果省略，Java将简单地选取一个随机空闲端口。

**Note**

通过Internet调试远程服务可能会很慢，您可能需要增加IDE中的超时。 例如，在Eclipse中，您可以选择Java从首选项调试...并将调试器超时（ms）更改为更合适的值（在大多数情况下，60000可以正常工作）。

**Warning**

在IntelliJ IDEA中使用远程调试通道时，必须将所有断点配置为挂起线程而不是VM。 默认情况下，IntelliJ IDEA中的断点会挂起整个虚拟机，而不是挂起命中断点的线程。 这具有暂停管理远程调试通道的线程的不良副作用，导致您的调试会话冻结。 在IntelliJ IDEA中使用远程调试通道时，应将所有断点配置为挂起线程而不是VM。 请参阅[IDEA-165769](https://youtrack.jetbrains.com/issue/IDEA-165769) 了解更多详情。

# 打包生产环境中的应用程序

可执行的jar可用于生产部署。 由于它们是独立的，因此它们也非常适合基于云的部署。

对于额外的“生产就绪”功能，如健康，审计和度量REST或JMX端点; 考虑加入spring-boot-actuator执行器。 有关详细信息，请参见[*Part V, “Spring Boot Actuator: Production-*](#_bookmark297)[*ready features”*](#_bookmark297) for details。

# 接下来读什么

您现在应该对如何使用Spring Boot以及您应该遵循的一些最佳实践有很好的理解。 您现在可以继续深入了解特定的[*Spring Boot features*](#_bookmark85) ，或者可以跳过并阅读Spring Boot的“[production ready](#_bookmark297)”部分。

**Part IV. Spring Boot 特性**

本节将深入探讨Spring Boot的细节。 在这里，您可以了解要使用和定制的关键功能。 如果还没有，可能需要阅读[*Part II, “Getting started”*](#_bookmark8) *和* [*Part III, “Using Spring Boot”*](#_bookmark34) ，以便您了解基础知识。

# SpringApplication

SpringApplication类提供了一个方便的方法来引导将从main（）方法启动的Spring应用程序。 在很多情况下，你可以委托给静态的SpringApplication.run方法：

**public static void** main(String[] args) { SpringApplication.run(MySpringConfiguration.**class**, args);

}

当您的应用程序启动时，您应该看到类似于以下内容：

Started SampleApplication in 2.992 seconds (JVM running for 3.658)

:

main] o.s.b.s.app.SampleApplication

main] .t.TomcatEmbeddedServletContainerFactory :

2014-03-04 13:09:54.912 INFO 41370 --- [

Server initialized with port: 8080

2014-03-04 13:09:56.501 INFO 41370 --- [

Starting SampleApplication v0.1.0 on mycomputer with PID 56603 (/apps/myapp.jar started by pwebb) 2013-07-31 00:08:16.166 INFO 56603 --- [ main] ationConfigEmbeddedWebApplicationContext :

Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@6e5a8246:

startup date [Wed Jul 31 00:08:16 PDT 2013]; root of context hierarchy

:

main] o.s.b.s.app.SampleApplication

/\\ / '\_ \_ \_(\_)\_ \_ \ \ \ \ ( ( )\ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ )| |\_)| | | | | || (\_| | ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | / / / /

=========|\_|==============| /=/\_/\_/\_/

:: Spring Boot :: v1.5.8.RELEASE

2013-07-31 00:08:16.117 INFO 56603 --- [

\_ \_

\_

.

默认情况下会显示INFO日志消息，包括一些相关的启动详细信息，例如启动应用程序的用户。

## 启动失败

如果您的应用程序无法启动，已注册的FailureAnalyzers将有机会提供专门的错误消息和具体操作来解决问题。 例如，如果您在端口8080上启动Web应用程序，并且该端口已被使用，则应该看到类似于以下内容的内容：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* APPLICATION FAILED TO START

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Description:

Embedded servlet container failed to start. Port 8080 was already in use. Action:

Identify and stop the process that's listening on port 8080 or configure this application to listen on another port.

**Note**

Spring Boot提供了大量的FailureAnalyzer实现，你可以很容易地添加你自己的([add your own](#_bookmark440))。

如果没有故障分析仪能够处理异常，您仍然可以显示完整的自动配置报告，以更好地了解出了什么问题。 所以你需要启用调试财产或启用调试日志记录

org.springframework.boot.autoconfigure.logging.AutoConfigurationReportLoggingInitialize

例如，如果使用java -jar运行应用程序，则可以按如下所示启用调试属性：

$ java -jar myproject-0.0.1-SNAPSHOT.jar --debug

## 自定义 the Banner

启动时打印的Banner可以通过将banner.txt文件添加到类路径中或通过将banner.location设置为该文件的位置来更改。 如果文件有一个不寻常的编码，你可以设置banner.charset（默认是UTF-8）。 除了文本文件之外，还可以将banner.gif，banner.jpg或banner.png图像文件添加到您的类路径中，或设置banner.image.location属性。 图像将被转换成ASCII艺术表现形式并打印在任何文字横幅上方。

在您的banner.txt文件中，您可以使用以下任何占位符：

*Table 23.1. Banner variables*

|  |  |
| --- | --- |
| **Variable** | **Description** |
| ${application.version} | The version number of your application as declared in MANIFEST.MF. For example Implementation-Version: 1.0 is printed as 1.0. |
| ${application.formatted-version} | The version number of your application as declared in MANIFEST.MF formatted for display (surrounded with brackets and prefixed with v). For example (v1.0). |
| ${spring-boot.version}  ${spring-boot.formatted-version} | The Spring Boot version that you are using. For example 1.5.8.RELEASE.  The Spring Boot version that you are using formatted for display (surrounded with brackets and prefixed with v). For example (v1.5.8.RELEASE). |
| ${Ansi.NAME} (or ${AnsiColor.NAME},  ${AnsiBackground.NAME},  ${AnsiStyle.NAME}) | Where NAME is the name of an ANSI escape code. See [AnsiPropertySource](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/ansi/AnsiPropertySource.java) for details. |
| ${application.title} | The title of your application as declared in MANIFEST.MF. For example  Implementation-Title: MyApp is printed  as MyApp. |

**Tip**

如果要以编程方式生成Banner，则可以使用SpringApplication.setBanner（...）方法。 使用org.springframework.boot.Banner接口并实现自己的printBanner（）方法。

您还可以使用spring.main.banner-mode属性来确定banner是否必须使用已配置的logger（log）或根本不打印（关闭）打印在System.out（控制台）上。

打印的banner将以springBootBanner名称注册为单例bean。

**Note**

YAML映射为false，因此如果要禁用应用程序中的banner，请确保添加引号。

**spring**:

**main**:

**banner-mode**: ***"off"***

## 自定义 SpringApplication

如果SpringApplication默认不符合您的喜好，您可以创建一个本地实例并对其进行自定义。 例如，要关闭横幅，你会写：

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

SpringApplication app = **new** SpringApplication(MySpringConfiguration.**class**); app.setBannerMode(Banner.Mode.OFF);

app.run(args);

}

**Note**

传递给SpringApplication的构造函数参数是spring bean的配置源。 在大多数情况下，这些将是对@Configuration类的引用，但也可能是对XML配置或应扫描的包的引用。

也可以使用application.properties文件来配置SpringApplication。 有关详细信息，请参阅第[*Chapter 24, Externalized Configuration*](#_bookmark97) 。

有关配置选项的完整列表，请参阅[SpringApplication Javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/SpringApplication.html)。

## Fluent builder API

如果您需要构建一个ApplicationContext层次结构（具有父/子关系的多个上下文），或者如果您只想使用“fluent”构建器API，则可以使用SpringApplicationBuilder。

SpringApplicationBuilder允许你链接多个方法调用，包括parent 和 child的方法，允许你创建一个层

次结构。

**new** SpringApplicationBuilder()

.sources(Parent.**class**)

.child(Application.**class**)

.bannerMode(Banner.Mode.OFF)

.run(args);

For example:

**Note**

创建ApplicationContext层次结构时有一些限制，例如 Web组件必须包含在child context，并且相同的环境将用于parent and child contexts。 有关完整的细节，请参阅[SpringApplicationBuilder Javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/builder/SpringApplicationBuilder.html) 。

## Application events and listeners

除了通常的Spring框架事件，比如[ContextRefreshedEvent](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/context/event/ContextRefreshedEvent.html)，SpringApplication发送一些额外的应用

程序事件。

**Note**

有些事件实际上是在创建ApplicationContext之前触发的，因此您不能将这些监听器注册为@Bean。 您可以通过SpringApplication.addListeners（...）或SpringApplicationBuilder.listeners（...）方法注册它们。

如果您希望自动注册这些监听器而不管创建应用程序的方式如何，您可以将META-INF / spring.factories文件添加到您的项目中，并使用org.springframework.context.ApplicationListener项引用您的监听器。

org.springframework.context.ApplicationListener=com.example.project.MyListener

应用程序事件按照以下顺序发送，就像您的应用程序运行一样：

1. ApplicationStartingEvent在运行开始时发送，但在除监听器和初始化程序的注册之外的任何处理之前发送。
2. 当在上下文中使用的环境是已知的，但在上下文创建之前，发送plicationEnvironmentPreparedEvent。
3. 一个ApplicationPreparedEvent只在刷新开始之前发送，但是在bean定义被载入之后。
4. 刷新后发送ApplicationReadyEvent，并处理任何相关的回调以指示应用程序已准备好为请求提供服务。
5. 如果启动时出现异常，则发送ApplicationFailedEvent。

**Tip**

您通常不需要使用应用程序事件，但可以方便地知道它们存在。 在内部，Spring Boot使用事件来处理各种任务。

## Web environment

SpringApplication将会尝试为你创建正确类型的ApplicationContext。 默认情况下，将使用AnnotationConfigApplicationContext或AnnotationConfigEmbeddedWebApplicationContext，具体取决于您是否正在开发Web应用程序。

用于确定“Web环境”的算法相当简单（基于少数类的存在）。 如果您需要覆盖默认值，则可以使用setWebEnvironment（boolean webEnvironment）。

也可以通过调用setApplicationContextClass（...）来完全控制ApplicationContext类型。

**Tip**

使用SpringApplication时，通常需要调用setWebEnvironment（false）在JUnit测试中。

## 访问应用程序参数

如果您需要访问传递给SpringApplication.run的应用程序参数（...）你可以注入一个org.springframework.boot.ApplicationArguments bean。 ApplicationArguments接口提供对原始String []参数以及解析的选项和非选项参数的访问：

**import** org.springframework.boot.\*

**import** org.springframework.beans.factory.annotation.\*

**import** org.springframework.stereotype.\*

@Component

**public class** MyBean {

@Autowired

**public** MyBean(ApplicationArguments args) { **boolean** debug = args.containsOption(***"debug"***); List<String> files = args.getNonOptionArgs();

*// if run with "--debug logfile.txt" debug=true, files=["logfile.txt"]*

}

}

**Tip**

Spring Boot也将在Spring环境中注册一个CommandLinePropertySource。 这使您可以使用@Value注释来注入单个应用程序参数。

## 使用 ApplicationRunner 或 CommandLineRunner

如果你需要在SpringApplication启动后运行一些特定的代码，你可以实现ApplicationRunner或者CommandLineRunner接口。 两个接口都以相同的方式工作，并提供了一个将在SpringApplication.run（...）完成之前调用的单个运行方法。

CommandLineRunner接口作为一个简单的字符串数组提供对应用程序参数的访问，而ApplicationRunner使用上面讨论的ApplicationArguments接口。

**import** org.springframework.boot.\*

**import** org.springframework.stereotype.\*

@Component

**public class** MyBean **implements** CommandLineRunner {

**public void** run(String... args) {

*// Do something...*

}

}

还可以实现org.springframework.core.Ordered接口或使用org.springframework.core.annotation.Order注释，如果定义了多个CommandLineRunner或ApplicationRunner bean，则必须按特定顺序调用它们。

## 退出应用程序

每个SpringApplication都会向JVM注册一个shutdown hook，以确保ApplicationContext在退出时正常关闭。 所有标准的Spring生命周期回调（比如DisposableBean接口或者@PreDestroy注解）都可以使用。

另外，如果在调用SpringApplication.exit（）时想要返回特定的退出代码，那么，bean可以实现org.springframework.boot.ExitCodeGenerator接口来满足这个要求。 这个退出代码然后可以传递给System.exit（）以返回它作为状态代码。

@SpringBootApplication

**public class** ExitCodeApplication {

@Bean

**public** ExitCodeGenerator exitCodeGenerator() {

**return new** ExitCodeGenerator() { @Override

**public int** getExitCode() {

**return** 42;

}

};

}

**public static void** main(String[] args) { System.exit(SpringApplication

.exit(SpringApplication.run(ExitCodeApplication.**class**, args)));

}

}

另外，ExitCodeGenerator接口可能由异常实现。 遇到这样的异常时，Spring Boot将返回由实现的getExitCode（）方法提供的退出代码。

## Admin 功能

通过指定spring.application.admin.enabled属性，可以为应用程序启用与管理相关的功能。 这暴露了平台MBeanServer上的[SpringApplicationAdminMXBean](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/admin/SpringApplicationAdminMXBean.java) 。 您可以使用此功能远程管理您的Spring Boot应用程序。 这对于任何服务包装实现也是有用的。

**Tip**

如果您想知道应用程序在哪个HTTP端口上运行，请使用key获取该属性local.server.port。

**Note**

启用此功能时要小心，因为MBean公开了关闭应用程序的方法。

# 外部化配置

Spring Boot允许您将配置外部化，以便在不同的环境中使用相同的应用程序代码。 您可以使用属性文件，YAML文件，environment variables和command-line来外部化配置。 属性值可以使用@Value注解直接注入到bean中，可以通过Spring的Environment抽象来访问，也可以通过@ConfigurationProperties绑定到结构化对象([bound to](#_bookmark109) [structured objects](#_bookmark109) )。

Spring Boot使用了一个非常特殊的PropertySource命令，该命令旨在允许合理的重写值。 属性按以下顺序考虑：

1. 在您的主目录上开发[Devtools global settings properties](#_bookmark78)（当devtools处于活动状态时〜/ .spring-boot- devtools.properties）。
2. 测试中的[@TestPropertySource](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/test/context/TestPropertySource.html) 注释.
3. 测试时[@[SpringBootTest#properties](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/test/context/SpringBootTest.html)](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/test/context/TestPropertySource.html) 注释属性.
4. 命令行参数
5. 来自SPRING\_APPLICATION\_JSON的属性（嵌入在环境变量或系统属性中的内联JSON）
6. ServletConfig 初始化参数.
7. ServletContext 初始化参数.
8. 来自 java:comp/env JNDI 属性.
9. Java System properties (System.getProperties()).
10. OS 环境变量.
11. RandomValuePropertySource只具有随机的属性\*。
12. [Profile-specific application properties](#_bookmark101) outside of your packaged jar (application-

{profile}.properties and YAML variants)

1. [Profile-specific application properties](#_bookmark101) packaged inside your jar (application-

{profile}.properties and YAML variants)

1. Application properties outside of your packaged jar (application.properties and YAML variants).
2. Application properties packaged inside your jar (application.properties and YAML variants).
3. [@PropertySource](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/context/annotation/PropertySource.html) annotations on your @Configuration classes.

17.Default properties (specified using SpringApplication.setDefaultProperties).

为了提供一个具体的例子，假设你开发了一个使用name属性的@Component：

**import** org.springframework.stereotype.\*

**import** org.springframework.beans.factory.annotation.\*

@Component

**public class** MyBean {

@Value("${name}")

**private** String name;

*// ...*

}

在您的应用程序类路径中（例如，在您的jar中），您可以拥有一个application.properties，为名称提供一个合理的默认属性值。 在新环境中运行时，可以在jar外部提供一个application.properties来覆盖名称; 对于一次性测试，您可以使用特定的命令行开关（例如java -jar app.jar - name =“Spring”）启动。

**Tip**

可以使用环境变量在命令行上提供SPRING\_APPLICATION\_JSON属性。 例如在一个UN \* X shell中：

$ SPRING\_APPLICATION\_JSON='{"foo":{"bar":"spam"}}' java -jar myapp.jar

在这个例子中，您将在Spring环境中以foo.bar = spam结尾。 您还可以在System变量中提供JSON作为spring.application.json：

$ java -Dspring.application.json='{"foo":"bar"}' -jar myapp.jar

或命令行参数：

$ java -jar myapp.jar --spring.application.json='{"foo":"bar"}'

或者作为JNDI变量java：comp / env / spring.application.json。

## 配置 random values

RandomValuePropertySource用于注入随机值（例如，注入秘密或测试用例）。 它可以产生integers，longs，uuids或strings，例如

**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]}

random.int \*语法是OPEN value（，max）CLOSE其中OPEN，CLOSE是任何字符和值，max是整数。 如果提供最大值，则值是最小值，最大值是最大值（不包括）。

## 访问 command line properties

默认情况下，SpringApplication将转换任何命令行选项参数（以' -- '开头，例如 --server.port = 9000）添加到一个属性并将其添加到Spring环境。 如上所述，命令行属性总是优先于其他属性源。

如果您不希望将命令行属性添加到环境中，则可以使用禁用它们SpringApplication.setAddCommandLineProperties（false）。

## 应用程序 property files

SpringApplication将从以下位置的application.properties文件中加载属性，并将它们添加到Spring环境中：

1. 当前目录的A / config子目录.
2. 当前目录
3. 一个 classpath /config 包
4. classpath root

该列表按优先顺序排列（在列表中较高的位置定义的属性将覆盖在较低位置定义的属性）。

**Note**

您也可以使用[use YAML ('.yml') files](#_bookmark103) 文件替代“.properties”

如果您不喜欢application.properties作为配置文件名，则可以通过指定spring.config.name环境属性来切换到另一个。 您还可以使用spring.config.location环境属性（以逗号分隔的目录位置列表或文件路径）引用显式位置。

$ java -jar myproject.jar --spring.config.name=myproject

或

$ java -jar myproject.jar --spring.config.location=classpath:/default.properties,classpath:/ override.properties

**Warning**

spring.config.name和spring.config.location很早就用来确定哪些文件必须被加载，因此必须将其定义为环境属性（通常是OS env，系统属性或命令行参数）。

如果spring.config.location包含目录（而不是文件），它们应该以/结尾(并且会在加载之前附加从spring.config.name生成的名称，包括配置文件特定的文件名。) 在spring.config.location中指定的文件按原样使用，不支持特定于配置文件的变体，并且将被任何特定于配置文件的特性覆盖。

配置位置按相反顺序搜索。 默认情况下，配置的位置是classpath：/，classpath：/ config /，

file：. / file：./ config /。 结果搜索顺序是：

1. file:./config/
2. file:./
3. classpath:/config/
4. classpath:/

当配置自定义配置位置时，除了默认位置之外，还会使用它们。 在默认位置之前搜索自定义位置。 例如，如果配置了自定义位置classpath：/ custom-config /，file：./ custom-config /，则搜索顺序变为：

1. file:./custom-config/
2. classpath:custom-config/
3. file:./config/
4. file:./
5. classpath:/config/
6. classpath:/

此搜索顺序允许您在一个配置文件中指定默认值，然后在另一个配置文件中选择性地覆盖这些值。 您可以在默认位置之一的application.properties（或其他选择任何使用spring.config.name命名的）中为您的应用程序提供默认值。 这些默认值可以在运行时被置于其中一个自定义位置的不同文件覆盖。

**Note**

如果使用环境变量而非系统属性，则大多数操作系统不允许使用句点分隔的键名(period-separated key names)，但可以使用下划线（例如SPRING\_CONFIG\_NAME，而不是spring.config.name）。

**Note**

如果您正在容器中运行，那么可以使用JNDI属性（在java：comp / env中）或servlet上下文初始化参数，而不是使用环境变量或系统属性。

## Profile-specific properties

除了application.properties文件外，还可以使用命名约定application- {profile} .properties来定义特定于配置文件的属性。 如果没有设置活动配置文件（即，如果没有显式激活配置文件，则加载来自application-default.properties的属性），则环境具有一组默认配置文件（默认情况下为[缺省值]）。

Profile-specific properties 从标准application.properties的相同位置加载，配置文件特定的文件总是覆盖非特定的文件，而不管配置文件特定的文件是在打包的jar内部还是外部。

如果指定了多个配置文件，则应用最后一个赢取策略(last wins strategy applies)。 例如，由spring.profiles.active属性指定的配置文件会在通过SpringApplication API配置的配置文件之后添加，因此优先。

**Note**

如果您在spring.config.location中指定了任何文件，则不会考虑这些文件的特定于配置文件的变体。 如果您还想使用配置文件特定的属性，请使用spring.config.location中的目录。

## 属性中的占位符

application.properties中的值在使用时会通过现有的Environment进行过滤，因此您可以返回到以前定义的值（例如，从System properties）。

**app.name**=MyApp

**app.description**=${app.name} is a Spring Boot application

**Tip**

您也可以使用这种技术来创建现有Spring Boot属性的“short’”变体。 有关详细信息，请参见[*Section 72.4, “Use ‘short’ command line arguments”*](#_bookmark451) 。

## 使用 YAML 代替 Properties

[YAML](http://yaml.org/)是JSON的超集，因此是用于指定分层配置数据的非常方便的格式。 SpringApplication类将自动支持YAML作为属性的替代方法，只要在classpath中有[SnakeYAML](http://www.snakeyaml.org/) 库。

**Note**

如果您使用“Starters”，SnakeYAML将通过spring-boot-starter自动提供。

### 加载 YAML

Spring框架提供了两个方便的类，可以用来加载YAML文档。 YamlPropertiesFactoryBean将作为属性加载YAML，YamlMapFactoryBean将作为Map加载YAML。

例如，下面的YAML文件：

**environments**: **dev**:

**url**[: http://dev.bar.com](http://dev.bar.com/)

**name**: Developer Setup

**prod**:

**url**[: http://foo.bar.com](http://foo.bar.com/)

**name**: My Cool App

将被转化为这些属性：

**environments.dev.url**[=http://dev.bar.com](http://dev.bar.com/) **environments.dev.name**=Developer Setup **environments.prod.url**[=http://foo.bar.com](http://foo.bar.com/) **environments.prod.name**=My Cool App

YAML列表被表示为具有[index]解引用的属性键，例如这个YAML：

**my**:

**servers**:

* dev.bar.com
* foo.bar.com

将被转化为这些属性：

**my.servers[0]**=dev.bar.com **my.servers[1]**=foo.bar.com

要使用Spring DataBinder实用程序（这是@ConfigurationProperties所做的）绑定属性，需要在java.util.List（或Set）类型的目标Bean中拥有一个属性，并且您需要提供setter，或者 用一个可变的值初始化它，

例如 这将绑定到上面的属性

@ConfigurationProperties(prefix="my")

**public class** Config {

**private** List<String> servers = **new** ArrayList<String>();

**public** List<String> getServers() {

**return this**.servers;

}

}

**Note**

当按照您所期望的方式重写将无法正常工作时，需要额外的注意。 在上面的例子中，当my.servers被重新定义在几个地方时，单个元素是被覆盖的目标，而不是列表。 为了确保具有更高优先级的PropertySource可以覆盖列表，您需要将其定义为单个属性：

**my**:

**servers**: dev.bar.com,foo.bar.com

### 在Spring环境中将YAML作为属性公开

YamlPropertySourceLoader类可以用来在Spring环境中将YAML作为一个PropertySource公开。 这使您可以使用熟悉的@Value注释和占位符语法来访问YAML属性。

### Multi-profile YAML 文件

您可以使用spring.profiles在单个文件中指定多个特定于配置文件的YAML文档键来指示文件何时适用。

例如：

**server**:

**address**: 192.168.1.100

*---*

**spring**:

**profiles**: development

**server**:

**address**: 127.0.0.1

*---*

**spring**:

**profiles**: production

**server**:

**address**: 192.168.1.120

在上面的示例中，如果开发配置文件处于活动状态，则server.address属性将为127.0.0.1。 如果开发和生产配置文件未启用，则属性的值将是192.168.1.100。

如果在应用程序上下文启动时没有显式激活，则默认配置文件被激活。 所以在这个YAML中，我们设置了security.user.password的一个值，它只在“default”配置文件中可用：

**server**: **port**: 8000

*---*

**spring**:

**profiles**: default

**security**: **user**:

**password**: weak

而在这个例子中，密码总是被设置，因为它没有附加到任何配置文件，并且必须根据需要在所有其他配置文件中明确地重置:

**server**: **port**: 8000

**security**: **user**:

**password**: weak

使用“spring.profiles”元素指定的Spring配置文件可以有选择地使用！ 字符。 如果为单个文档指定了否定配置文件和非non-negated配置文件，则至少有一个非non-negated配置文件必须匹配，且non-negated配置文件可能不匹配。

### YAML 缺点

YAML文件不能通过@PropertySource注解加载。 所以在需要以这种方式加载值的情况下，您需要使用一个属性文件。

### Merging YAML lists

正如我们上面看到的([we have seen above](#_bookmark104))，任何YAML内容最终都会转换为属性。 当通过配置文件覆盖“列表”属性时，该过程可能不直观。

例如，假设名称和描述属性默认为空的MyPojo对象。 让我们从FooProperties公开一个MyPojo列表：

@ConfigurationProperties("foo")

**public class** FooProperties {

**private final** List<MyPojo> list = **new** ArrayList<>();

**public** List<MyPojo> getList() {

**return this**.list;

}

}

考虑以下配置：

**foo**:

**list**:

**- name**: my name

**description**: my description

*---*

**spring**: **profiles**: dev

**foo**:

**list**:

**- name**: my another name

如果开发者配置文件不活动，FooProperties.list将包含一个如上定义的MyPojo条目。 如果启用配置文件启用，但列表仍将只包含一个条目（名称“我的另一个名称”和说明空）。 此配置不会将第二个MyPojo实例添加到列表中，并且不会合并这些项目。

当在多个配置文件中指定一个集合时，将使用具有最高优先级的集合（并且仅使用该集合）：

**foo**:

**list**:

* **name**: my name

**description**: my description

* **name**: another name

**description**: another description

*---*

**spring**: **profiles**: dev

**foo**:

**list**:

**- name**: my another name

在上面的例子中，考虑到dev配置文件是活动的，FooProperties.list将包含一个MyPojo条目

（名称为“我的另一个名称”，说明为空）。

## Type-safe 配置文件

使用@Value（“$ {property}”）注释来注入配置属性有时会非常麻烦，特别是如果您使用多个属性或者您的数据是分层的。 Spring Boot提供了另一种处理属性的方法，允许强类型的bean管理和验证应用程序的配置。

**package** com.example;

**import** java.net.InetAddress; **import** java.util.ArrayList; **import** java.util.Collections; **import** java.util.List;

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

@ConfigurationProperties("foo")

**public class** FooProperties {

**private boolean** enabled;

**private** InetAddress remoteAddress;

**private final** Security security = **new** Security();

**public boolean** isEnabled() { ... }

**public void** setEnabled(**boolean** enabled) { ... }

**public** InetAddress getRemoteAddress() { ... }

**public void** setRemoteAddress(InetAddress remoteAddress) { ... }

**public** Security getSecurity() { ... }

**public static class** Security { **private** String username; **private** String password;

**private** List<String> roles = **new** ArrayList<>(Collections.singleton(***"USER"***));

**public** String getUsername() { ... }

**public void** setUsername(String username) { ... }

**public** String getPassword() { ... }

**public void** setPassword(String password) { ... }

**public** List<String> getRoles() { ... }

**public void** setRoles(List<String> roles) { ... }

}

}

上面的 POJO 定义了以下属性:

* foo.enabled, 默认为 false
* foo.remote-address, 与可以从String强制类型
* foo.security.username, 具有名称由属性名称确定的嵌套安全性。特别是返回类型没有在那里使用，可能是SecurityProperties
* foo.security.password
* foo.security.roles, 与String的集合

**Note**

getters和setter通常是强制的，因为绑定是通过标准的Java Beans属性描述符来完成的，就像在Spring MVC中一样。 有些情况下，可能会忽略setter:

* + Maps, 只要它们被初始化，就需要一个getter，但不一定是setter，因为它们可以被绑定器binder改变。
  + 集合和数组可以通过索引（通常使用YAML）或使用单个逗号分隔值（属性）来访问。 在后一种情况下，setter是强制性的。 我们建议始终为这种类型添加一个setter。 如果初始化一个集合，确保它不是不可变的（如上例）
  + 如果初始化嵌套的POJO属性（如上例中的Security域），则不需要setter。 如果您希望binder使用其默认构造函数即时创建实例，则需要一个setter。

有些人使用Project Lombok来自动添加getter和setter。 确保Lombok不会为这种类型生成任何特定的构造函数，因为它将被容器自动使用来实例化对象。

**Tip**

另外请参考 [differences between @Value and @ConfigurationProperties](#_bookmark114) 的区别.

您还需要列出要在@EnableConfigurationProperties中注册的属性类

注解：

@Configuration

@EnableConfigurationProperties(FooProperties.class)

**public class** MyConfiguration {

}

**Note**

当以这种方式注册@ConfigurationProperties bean时，该bean将具有常规名称：<prefix> - <fqn>，其中<prefix>是@ConfigurationProperties注释中指定的环境键前缀，<fqn>是 bean。 如果注释不提供任何前缀，则只使用bean的完全限定名称。

上例中的bean名称将是 foo-com.example.FooProperties.

即使上面的配置为FooProperties创建了一个常规的bean，我们也建议@ConfigurationProperties只处理环境，特别是不要从上下文中注入其他bean。 话虽如此，@EnableConfigurationProperties注释也会自动应用到您的项目中，以便任何现有的使用@ConfigurationProperties注释的bean都将从环境配置。 您可以通过确保FooProperties已经是一个bean来快速配置MyConfiguration：

@Component @ConfigurationProperties(prefix="foo") **public class** FooProperties {

*// ... see above*

}

SpringApplication外部YAML配置的这种配置特别适合：

*# application.yml*

**foo**:

**remote-address**: 192.168.1.1

**security**:

**username**: foo

**roles**:

* USER
* ADMIN

*# additional configuration as required*

要使用@ConfigurationProperties bean，您可以像其他任何bean一样注入它们。

@Service

**public class** MyService {

**private final** FooProperties properties; @Autowired

**public** MyService(FooProperties properties) {

**this**.properties = properties;

}

*//...*

@PostConstruct

**public void** openConnection() {

Server server = **new** Server(**this**.properties.getRemoteAddress());

*// ...*

}

}

**Tip**

使用@ConfigurationProperties还允许您生成元数据文件，IDE可以使用这些元数据文件为自己的密钥提供自动完成功能，详情请参阅附录B[Appendix B, *Configuration meta-*](#_bookmark571)[*data*](#_bookmark571) ，配置元数据附录。

### Third-party 配置

除了使用@ConfigurationProperties注解一个类，你也可以在public @Bean方法上使用它。 如果要将属性绑定到不受控制的第三方组件，这可能特别有用。

要从Environment属性配置一个bean，添加@ConfigurationProperties到它的bean注册：

@ConfigurationProperties(prefix = "bar") @Bean

**public** BarComponent barComponent() {

...

}

使用bar前缀定义的任何属性都将以类似于上面的FooProperties示例的方式映射到该BarComponent bean上。

### Relaxed binding

Spring Boot使用一些宽松的规则来将环境属性绑定到@ConfigurationProperties bean，所以不需要在Environment属性名称和bean属性名称之间完全匹配。 其中有用的常见示例包括虚线分隔（例如，上下文路径绑定到contextPath）和大写（例如PORT绑定到端口）环境属性。

例如，给定以下@ConfigurationProperties类：

@ConfigurationProperties(prefix="person")

**public class** OwnerProperties {

**private** String firstName;

**public** String getFirstName() {

**return this**.firstName;

}

**public void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

}

以下属性名称都可以使用：

*Table 24.1. relaxed binding*

|  |
| --- |
| **Property Note** |
| person.firstNameStandard camel case syntax. |

|  |
| --- |
| **Property Note** |
| person.first- Dashed notation, recommended for use in .properties and .yml files.  name  person.first\_namUenderscore notation, alternative format for use in .properties and .yml  files. |
| PERSON\_FIRST\_NAMUEpper case format. Recommended when using a system environment variables. |

### Properties 转换

当它绑定到@ConfigurationProperties bean时，Spring将试图强制外部应用程序属性为正确的类型。 如果您需要自定义类型转换，您可以提供一个ConversionService bean（带有bean id conversionService）或自定义属性编辑器（通过CustomEditorConfigurer bean）或自定义Converters（注释为@ConfigurationPropertiesBinding的bean定义）。

**Note**

由于此bean在应用程序生命周期中很早被请求，因此请确保限制ConversionService所使用的依赖项。 通常情况下，您需要的任何依赖项可能在创建时未完全初始化。 如果不需要配置密钥强制转换，并且只依赖使用@ConfigurationPropertiesBinding限定的自定义转换器，则可能需要重命名自定义的ConversionService。

### @ConfigurationProperties 验证

当Spring的@Validated批注注释时，Spring Boot将尝试验证@ConfigurationProperties类。 您可以直接在您的配置类上使用JSR-303的javax.validation约束条件注释。 只要确保符合JSR-303的实现在您的类路径上，然后将约束注释添加到您的字段中：

@ConfigurationProperties(prefix="foo") @Validated

**public class** FooProperties {

@NotNull

**private** InetAddress remoteAddress;

*// ... getters and setters*

}

为了验证嵌套属性的值，您必须将相关字段注释为@Valid来触发其验证。 例如，建立在上面的FooProperties示例上：

@ConfigurationProperties(prefix="connection") @Validated

**public class** FooProperties {

@NotNull

**private** InetAddress remoteAddress;

@Valid

**private final** Security security = **new** Security();

*// ... getters and setters*

**public static class** Security { @NotEmpty

**public** String username;

*// ... getters and setters*

}

}

你也可以通过创建一个名为configurationPropertiesValidator的bean定义来添加一个自定义的Spring Validator。 应该声明@Bean方法是静态的。 配置属性验证器在应用程序生命周期的早期就被创建，并将@Bean方法声明为静态，这样就可以在不需要实例化@Configuration类的情况下创建bean。 这避免了可能由于实例化提前而导致的任何问题。 有一个属性验证示例[property validation sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-property-validation) ，所以你可以看到如何设置的东西。

**Tip**

spring-boot-actuator模块包含一个暴露所有@ConfigurationProperties bean的端点。 只需将您的Web浏览器指向/ configprops或使用等效的JMX端点即可。 请参阅生产准备功能[*Production ready features*](#_bookmark299)。 部分的细节。

### @ConfigurationProperties vs. @Value

@Value是核心容器功能，它不提供与类型安全配置属性相同的功能。 下表总结了@ConfigurationProperties和@Value支持的功能：

|  |  |
| --- | --- |
| **Feature @ConfigurationProp** | **@Value** |
| [Relaxed binding](#_bookmark111) Yes  [Meta-data support](#_bookmark571) Yes | No  No |
| SpEL evaluation No | Yes |

如果您为自己的组件定义了一组配置关键字，我们建议您将它们分组到POJO注释的@ConfigurationProperties中。 另请注意，由于@Value不支持放宽绑定，因此如果您需要使用环境变量提供值，则不是一个好选择。

最后，虽然可以在@Value中编写SpEL表达式，但是这些表达式不会从应用程序属性文件中处理[Application property files](#_bookmark100)。

# Profiles

Spring Profiles提供了一种分离部分应用程序配置的方法，并使其仅在特定环境中可用。 任何@Component或@Configuration都可以使用@Profile标签来限制何时加载:

在普通的Spring方法中，可以使用spring.profiles.active环境属性来指定哪些配置文件处于活动状态。 你可以用任何常用的方式来指定属性，例如你可以在你的application.properties中包含它：

**spring.profiles.active**=dev,hsqldb

或使用 switch --spring.profiles.active = dev，hsqldb在命令行中指定。

## 添加激活 profiles

spring.profiles.active属性遵循与其他属性相同的排序规则，最高的PropertySource将获胜。 这意味着您可以在application.properties中指定活动配置文件，然后使用命令行开关替换它们。

有时，将特定于配置文件的属性添加到活动配置文件而不是替换它们会很有用。 spring.profiles.include属性可以用来无条件添加活动配置文件。 SpringApplication入口点还有一个用于设置其他配置文件的Java API（即，在由spring.profiles.active属性激活的那些配置文件之上）：请参阅setAdditionalProfiles（）方法。

例如，当具有以下属性的应用程序使用switch - spring.profiles.active = prod运行时，proddb和prodmq配置文件也将被激活：

*---*

**my.property**: fromyamlfile

*---*

**spring.profiles**: prod

**spring.profiles.include**:

* proddb
* prodmq

**Note**

请记住，可以在YAML文档中定义spring.profiles属性，以确定何时将此特定文档包含在配置中。 有关更多详细信息，请参见[Section 72.7,](#_bookmark454) [“Change configuration depending on the environment”](#_bookmark454) 。

## 变成方式配置 profiles

您可以通过在应用程序运行之前调用SpringApplication.setAdditionalProfiles（...）以编程方式设置活动配置文件。 使用Spring的ConfigurableEnvironment接口也可以激活配置文件。

## Profile-specific configuration files

将application.properties（或application.yml）和通过@ConfigurationProperties引用的文件的特定于配置文件的变体(Profile-specific variants)视为加载文件。 有关详细信息，请参见[*Section 24.4,*](#_bookmark101)[*“Profile-specific properties”*](#_bookmark101) 。

# Logging

Spring Boot使用[Commons Logging](http://commons.apache.org/logging) 进行所有内部日志记录，但是将底层日志实现保留为打开状态。 为[Java Util Logging](http://docs.oracle.com/javase/7/docs/api/java/util/logging/package-summary.html), [Log4J2](http://logging.apache.org/log4j/2.x/) 和 [Logback](http://logback.qos.ch/)提供了默认配置。 在每种情况下，loggers都预先配置为使用控制台输出，可选的文件输出也可用。

默认情况下，如果使用“Starter”，Logback将用于日志记录。 还包括适当的Logback路由，以确保使用Java Util日志记录，Commons Logging，Log4J或SLF4J的相关库都能正常工作。

**Tip**

Java有很多可用的日志框架。 不要担心，如果上面的列表似乎令人困惑。 一般来说，你不需要改变你的日志依赖性，Spring Boot的默认设置就可以正常工作。

## Log 格式化

Spring Boot的默认日志输出如下所示：

:

:

:

:

2014-03-05 10:57:51.253 INFO 45469 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] Initializing Spring embedded WebApplicationContext

2014-03-05 10:57:51.253 INFO 45469 --- [ost-startStop-1] o.s.web.context.ContextLoader Root WebApplicationContext: initialization completed in 1358 ms

2014-03-05 10:57:51.698 INFO 45469 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean Mapping servlet: 'dispatcherServlet' to [/]

2014-03-05 10:57:51.702 INFO 45469 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean

Mapping filter: 'hiddenHttpMethodFilter' to: [/\*]

main] org.apache.catalina.core.StandardEngine :

2014-03-05 10:57:51.112 INFO 45469 --- [

Starting Servlet Engine: Apache Tomcat/7.0.52

以下项目被输出：

* 日期和时间 - 精确到毫秒，易于排序。
* 日志级别 - 错误，警告，信息，调试或跟踪。
* Process ID.
* A ---分隔符来区分实际日志消息的开始。
* 线程名称 - 括在方括号中（可能会截断控制台输出）。
* Logger name — 这通常是源类名称（通常缩写）。
* log message.

**Note**

Logback没有FATAL级别（它被映射到ERROR）

## Console 输出

默认日志配置会在写入消息时将消息回送到控制台。 默认情况下会记录ERROR，WARN和INFO级别的消息。 您也可以通过使用--debug标志启动应用程序来启用“调试”模式。

$ java -jar myapp.jar --debug

**Note**

你也可以在你的application.properties中指定debug = true。

当启用调试模式时，会选择一些核心loggers（嵌入式容器，Hibernate和Spring Boot）来输出更多的信息。 启用调试模式不会将您的应用程序配置为使用DEBUG级别记录所有消息。

或者，您可以通过使用trace标志（或application.properties中的trace = true）启动应用程序来启用“跟踪”模式。 这将启用对核心记录器（嵌入式容器，Hibernate模式生成和整个Spring产品组合）选择的跟踪记录。

### Color-coded 输出

如果您的终端支持ANSI，则会使用彩色输出来提高可读性。 你可以设置spring.output.ansi.enabled为

[supported value](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/ansi/AnsiOutput.Enabled.html) 来覆盖自动检测。

颜色编码使用％clr转换字进行配置。 在最简单的形式中，转换器将根据日志级别为输出着色，例如：

%clr(%5p)

The mapping of log level to a color is as follows:

|  |  |
| --- | --- |
| **Level** | **Color** |
| FATAL | Red |
| ERROR | Red |
| WARN  INFO | Yellow  Green |
| DEBUG | Green |
| TRACE | Green |

或者，您可以通过提供转换选项来指定应使用的颜色或样式。 例如，要使文本变成黄色：

%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){yellow}

支持以下颜色和样式：

* blue
* cyan
* faint
* green
* magenta
* red
* yellow

## File output

默认情况下，Spring Boot将只能输出到控制台，不会写入日志文件。 如果除了控制台输出之外还想写日志文件，则需要设置logging.file或logging.path属性（例如，在application.properties中）。

下表显示了logging。\*属性如何一起使用：

*Table 26.1. Logging properties*

|  |  |  |
| --- | --- | --- |
| **logging.filleogging.path**  *(none) (none)* | **Example** | **Description**  Console only logging. |
| Specific file *(none)* | my.log | Writes to the specified log file. Names can be an exact location or relative to the current directory. |
| *(none)* Specific directory | /var/log | Writes spring.log to the specified directory. Names can be an exact location or relative to the current directory. |

日志文件在达到10 MB时会循环，并且与控制台输出一样，默认情况下会记录ERROR，WARN和INFO级别的消息。

**Note**

日志记录系统在应用程序生命周期的早期初始化，因此在通过@PropertySource注解加载的属性文件中不会找到这样的日志记录属性。

**Tip**

日志记录属性独立于实际的日志记录基础结构。 因此，特定的配置keys（例如Logback的logback.configurationFile）不受Spring Boot的管理。

## Log 级别

所有支持的日志记录系统都可以使用'logging.level.\* = LEVEL'在Spring Environment中设置日志级别（例如在application.properties中），其中LEVEL是TRACE，DEBUG，INFO，WARN，ERROR， FATAL，OFF之一 。 根记录器可以使用logging.level.root进行配置。 示例application.properties：

**logging.level.root**=WARN **logging.level.org.springframework.web**=DEBUG **logging.level.org.hibernate**=ERROR

**Note**

默认情况下，Spring Boot会重新映射Thymeleaf INFO消息，以便在DEBUG级别进行记录。 这有助于减少标准日志输出中的noise。 有关如何在自己的配置中应用重新映射的详细信息，请参阅[LevelRemappingAppender](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/logging/logback/LevelRemappingAppender.java)。

## 自定义日志配置

可以通过在类路径中包含适当的库来激活各种日志记录系统，并且可以通过在类路径的根目录中或在Spring环境属性logging.config中指定的位置提供合适的配置文件来进一步进行定制。

您可以使用org.springframework.boot.logging.LoggingSystem系统属性强制Spring Boot使用特定的日志记录系统。 该值应该是LoggingSystem实现的完全限定(fully-qualified)的类名称。 您还可以完全禁用Spring Boot的日志记录配置，使用值为none。

**Note**

由于在创建ApplicationContext之前对日志进行初始化，因此无法在Spring @Configuration文件中控制来自@PropertySources的日志记录。 系统属性和传统的Spring Boot外部配置文件工作得很好。）

根据您的日志记录系统，将加载以下文件：

|  |  |
| --- | --- |
| **Logging System** | **Customization** |
| Logback | logback-spring.xml, logback- spring.groovy, logback.xml or logback.groovy |
| Log4j2  JDK (Java Util Logging) | log4j2-spring.xml or log4j2.xml  logging.properties |

**Note**

如果可能，我们建议您使用-spring变体进行日志记录配置（例如，logback-spring.xml而不是logback.xml）。 如果使用标准配置位置，Spring将无法完全控制日志初始化。

**Warning**

Java Util Logging存在已知的类加载问题，从“executable jar”运行时会导致问题。 我们建议您尽可能避免。

为了帮助定制，一些其他的属性从Spring环境中传递

系统属性：

|  |  |
| --- | --- |
| **Spring Environment**  logging.exception- conversion-word | **System Property Comments**  LOG\_EXCEPTION\_CONVERSION\_WTOhReDconversion word that’s  used when logging exceptions. |
| logging.file | LOG\_FILE Used in default log configuration if defined. |

|  |  |  |
| --- | --- | --- |
| **Spring Environment** | **System Property** | **Comments** |
| logging.path  logging.pattern.console | LOG\_PATH  CONSOLE\_LOG\_PATTERN | Used in default log configuration if defined.  The log pattern to use on the console (stdout). (Only supported with the default logback setup.) |
| logging.pattern.file  logging.pattern.level | FILE\_LOG\_PATTERN  LOG\_LEVEL\_PATTERN | The log pattern to use in a file (if LOG\_FILE enabled). (Only supported with the default logback setup.)  The format to use to render the log level (default %5p).  (Only supported with the default  logback setup.) |
| PID | PID | The current process ID (discovered if possible and when not already defined as an OS environment variable). |

所有支持的日志系统在解析其配置文件时都可以查询系统属性。 有关示例，请参阅spring-boot.jar中的默认配置。

**Tip**

如果您想在日志记录属性中使用占位符，则应该使用Spring Boot的语法，而不是基础框架的语法。 值得注意的是，如果您使用的是Logback，则应该使用：作为属性名称和其默认值之间的分隔符，而不是： -

**Tip**

您可以通过仅覆盖LOG\_LEVEL\_PATTERN（或使用Logback的logging.pattern.level）来向日志行添加MDC和其他临时内容。 例如，如果使用logging.pattern.level = user：％X {user}％5p，则默认日志格式将包含“user”的MDC条目（如果存在）。

2015-09-30 12:30:04.031 user:juergen INFO 22174 --- [ nio-8080-exec-0] demo.Controller Handling authenticated request

## Logback 扩展

Spring Boot包含许多对Logback的扩展，可以帮助进行高级配置。 您可以在您的logback-spring.xml配置文件中使用这些扩展。

**Note**

您不能在标准logback.xml配置文件中使用扩展，因为它太早加载。 您需要使用logback-spring.xml或定义logging.config属性。

**Warning**

这些扩展名不能用于Logback的配置扫描([configuration scanning](http://logback.qos.ch/manual/configuration.html#autoScan))。 如果您尝试这样做，则更改配置文件将导致类似于以下记录之一的错误：

ERROR in ch.qos.logback.core.joran.spi.Interpreter@4:71 - no applicable action for [springProperty], current ElementPath is [[configuration][springProperty]]

ERROR in ch.qos.logback.core.joran.spi.Interpreter@4:71 - no applicable action for [springProfile], current ElementPath is [[configuration][springProfile]]

### Profile-specific configuration

<springProfile>标签允许您根据活动的Spring配置文件选择性地包含或排除配置的各个部分。 配置文件部分在<configuration>元素的任何地方都受支持。 使用name属性来指定哪个配置文件接受配置。 可以使用逗号分隔列表指定多个配置文件。

**<springProfile name**=**"staging">**

*<!-- configuration to be enabled when the "staging" profile is active -->*

**</springProfile>**

**<springProfile name**=**"dev, staging">**

*<!-- configuration to be enabled when the "dev" or "staging" profiles are active -->*

**</springProfile>**

**<springProfile name**=**"!production">**

*<!-- configuration to be enabled when the "production" profile is not active -->*

**</springProfile>**

### Environment properties

<springProperty>标签允许您在Spring环境中使用Spring环境的属性。 如果要在您的logback配置中访问您的application.properties文件中的值，这可能很有用。 该标签的工作方式与Logback的标准<property>标签类似，但不是指定直接值，而是指定属性的来源（来自环境）。 如果需要将属性存储在本地范围以外的地方，可以使用scope属性。 如果您在环境中未设置属性的情况下需要回退值，则可以使用defaultValue属性。

**<springProperty scope**=**"context" name**=**"fluentHost" source**=**"myapp.fluentd.host" defaultValue**=**"localhost"/>**

**<appender name**=**"FLUENT" class**=**"ch.qos.logback.more.appenders.DataFluentAppender">**

**<remoteHost>**${fluentHost}**</remoteHost>**

...

**</appender>**

**Tip**

RelaxedPropertyResolver用于访问环境属性。 如果以虚线符号（my-property-name）指定源，则会尝试所有宽松的变体(relaxed variations)（myPropertyName，MY\_PROPERTY\_NAME等）。

# 开发 web 应用程序

Spring Boot非常适合Web应用程序的开发。 您可以使用嵌入式Tomcat，Jetty或Undertow轻松创建自包含的HTTP服务器。 大多数Web应用程序将使用spring-boot-starter-web模块来快速启动和运行。

如果您还没有开发Spring Boot Web应用程序，则可以按照“Hello World！”进行操作。 Getting started。

## ‘Spring Web MVC framework’

Spring Web MVC框架（通常简称为“Spring MVC”）是一个丰富的“模型视图控制器”Web框架。 Spring MVC允许您创建特殊的@Controller或@RestController bean来处理传入的HTTP请求。 您的控制器中的方法使用@RequestMapping注释映射到HTTP。

这里是一个典型的例子@RestController来提供JSON数据：

@RestController @RequestMapping(value="/users") **public class** MyRestController {

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

**public** User getUser(@PathVariable Long user) {

*// ...*

}

@RequestMapping(value="/{user}/customers", method=RequestMethod.GET) List<Customer> getUserCustomers(@PathVariable Long user) {

*// ...*

}

@RequestMapping(value="/{user}", method=RequestMethod.DELETE)

**public** User deleteUser(@PathVariable Long user) {

*// ...*

}

}

Spring MVC是核心Spring框架的一部分，详细信息见参考[reference](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle#mvc) [documentation](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle#mvc)。 在[spring.io/guides](http://spring.io/guides) 中也有几个指南，涵盖了Spring MVC。

### Spring MVC auto-configuration

Spring Boot为Spring MVC提供了自动配置，可以与大多数应用程序配合使用。 自动配置在Spring的默认设置之上添加以下功能：

* 包含ContentNegotiatingViewResolver和BeanNameViewResolver bean。
* 支持提供静态资源，包括对WebJars的支持（见下文）。
* 自动注册Converter，GenericConverter，Formatter bean。
* 支持HttpMessageConverters（见下文）。
* 自动注册MessageCodesResolver（见下文）。
* 静态index.html支持。
* 自定义Favicon支持（见下文）。
* 自动使用可配置的WebBindingInitializer bean（见下文）。

如果你想保持Spring Boot MVC的功能，并且你只是想添加额外的[MVC](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle#mvc) [configuration](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle#mvc) （拦截器，格式化器，视图控制器等），你可以添加你自己的@Configuration类型的WebMvcConfigurerAdapter，但是没有@EnableWebMvc。 如果您希望提供RequestMappingHandlerMapping，RequestMappingHandlerAdapter或ExceptionHandlerExceptionResolver的自定义实例，则可以声明提供此类组件的WebMvcRegistrationsAdapter实例。

如果你想完全控制Spring MVC，你可以添加你自己的用@EnableWebMvc注解的@Configuration。

### HttpMessageConverters

Spring MVC使用HttpMessageConverter接口来转换HTTP请求和响应。 明智的默认是包括在盒子，例如对象可以自动转换为JSON（使用Jackson）或XML（使用Jackson XML扩展如果可用，否则使用JAXB）。 字符串默认使用UTF-8编码。

如果你需要添加或定制转换器，你可以使用Spring Boot的HttpMessageConverters

类：

**import** org.springframework.boot.autoconfigure.web.HttpMessageConverters;

**import** org.springframework.context.annotation.\*;

**import** org.springframework.http.converter.\*;

@Configuration

**public class** MyConfiguration {

@Bean

**public** HttpMessageConverters customConverters() { HttpMessageConverter<?> additional = ...

HttpMessageConverter<?> another = ...

**return new** HttpMessageConverters(additional, another);

}

}

存在于上下文中的任何HttpMessageConverter bean将被添加到转换器列表中。 您也可以通过这种方式覆盖默认转换器。

### 自定义JSON序列化器和反序列化器

如果您使用Jackson来序列化和反序列化JSON数据，则可能需要编写自己的JsonSerializer和JsonDeserializer类。 自定义序列化器通常通过[registered with](http://wiki.fasterxml.com/JacksonHowToCustomDeserializers) [Jackson via a Module](http://wiki.fasterxml.com/JacksonHowToCustomDeserializers)，但Spring Boot提供了一个替代的@JsonComponent注释，这使得更容易直接注册Spring Bean。

您可以直接在JsonSerializer或JsonDeserializer实现上使用@JsonComponent。 您也可以在包含序列化器/反序列化器的类中使用它作为内部类。 例如：

**import** java.io.\*;

**import** com.fasterxml.jackson.core.\*;

**import** com.fasterxml.jackson.databind.\*;

**import** org.springframework.boot.jackson.\*;

@JsonComponent

**public class** Example {

**public static class** Serializer **extends** JsonSerializer<SomeObject> {

*// ...*

}

**public static class** Deserializer **extends** JsonDeserializer<SomeObject> {

*// ...*

}

}

ApplicationContext中的所有@JsonComponent bean将自动注册到Jackson，并且由于@JsonComponent使用@Component进行元注释，所以通常的组件扫描规则适用。

Spring Boot还提供了[JsonObjectSerializer](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jackson/JsonObjectSerializer.java) 和[JsonObjectDeserializer](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jackson/JsonObjectDeserializer.java) 基类，在序列化对象时为标准Jackson版本提供了有用的替代方法。 有关详细信息，请参阅Javadoc。

### MessageCodesResolver

Spring MVC有一个策略来生成错误代码，用于从绑定错误中呈现错误消息：MessageCodesResolver。 如果您设置spring.mvc.message-codes-resolver.format属性PREFIX\_ERROR\_CODE或POSTFIX\_ERROR\_CODE（请参阅DefaultMessageCodesResolver.Format中的枚举），Spring Boot将为您创建一个。

### 静态内容

默认情况下，Spring Boot将从类路径中的/ static（或/ public或/ resources或/ META-INF / resources）目录或从ServletContext的根目录中提供静态内容。 它使用Spring MVC中的ResourceHttpRequestHandler，因此可以通过添加自己的WebMvcConfigurerAdapter并重写addResourceHandlers方法来修改该行为。

在stand-alone Web应用程序中，容器中的默认servlet也被启用，并作为后备，如果Spring决定不处理，则从ServletContext的根目录提供内容。 大多数情况下，这是不会发生的（除非你修改默认的MVC配置），因为Spring总是能够通过DispatcherServlet处理请求。

默认情况下，资源映射到/ \*\*，但可以通过spring.mvc.static-path- pattern调整。 例如，将所有资源重定位到/ resources / \*\*可以实现如下：

**spring.mvc.static-path-pattern**=/resources/\*\*

您也可以使用spring.resources.static-locations（使用目录位置列表替换缺省值）来自定义静态资源位置。 如果您这样做，默认的欢迎页面检测将切换到您的自定义位置。 因此，如果您的任何位置在启动时都有index.html，它将成为应用程序的主页。

除了上面的“标准”静态资源位置之外，Webjars内容还有一个特殊情况。 如果以Webjars格式打包，那么在/ webjars / \*\*中具有路径的任何资源都将从jar文件提供。

**Tip**

如果您的应用程序将被打包为jar，请不要使用src / main / webapp目录。 虽然这个目录是一个通用的标准，但是它只能用于war包装，如果你生成一个jar包，它会被大多数的构建工具默默地忽略。

Spring Boot还支持Spring MVC提供的高级资源处理功能，允许使用例如缓存清除静态资源或使用Webjars的版本不可知的URL。

要为Webjars使用版本不可知的URL，只需添加webjars-locator依赖项即可。 然后声明你的Webjar，例如jQuery，作为“/webjars/jquery/dist/jquery.min.js”，导致“/webjars/jquery/xyz/dist/jquery.min.js”其中xyz是Webjar版本。

**Note**

如果您使用的是JBoss，则需要声明webjars-locator-jboss-vfs依赖项，而不是webjars-locator; 否则所有的Webjar将解析为404。

要使用缓存清除，以下配置将为所有静态资源配置缓存清除解决方案，从而在URL中有效添加内容哈希，如<link href =“/ css / spring-2a2d595e6ed9a0b24f027f2b63b134d6.css”/>

**spring.resources.chain.strategy.content.enabled**=true **spring.resources.chain.strategy.content.paths**=/\*\*

**Note**

在模板中，资源链接在运行时被重写，这要归功于为Thymeleaf和FreeMarker自动配置的ResourceUrlEncodingFilter。 使用JSP时，您应该手动声明此过滤器。 其他模板引擎现在不会自动支持，但可以使用自定义模板宏/助手和[ResourceUrlProvider](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/web/servlet/resource/ResourceUrlProvider.html)的使用。

例如，在使用JavaScript模块加载程序动态加载资源时，重命名文件不是一个选项。 这就是为什么其他策略也支持并可以结合的原因。 “fixed”策略将在URL中添加静态版本字符串，而不更改文件名称：

**spring.resources.chain.strategy.content.enabled**=true **spring.resources.chain.strategy.content.paths**=/\*\* **spring.resources.chain.strategy.fixed.enabled**=true **spring.resources.chain.strategy.fixed.paths**=/js/lib/ **spring.resources.chain.strategy.fixed.version**=v12

使用此配置，位于“/ js / lib /”下的JavaScript模块将使用固定的版本控制策略“/v12/js/lib/mymodule.js”，而其他资源仍将使用内容<link href =“/ css /spring-2a2d595e6ed9a0b24f027f2b63b134d6.css“/>。

有关更多受支持的选项，请参阅[ResourceProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ResourceProperties.java) 。

**Tip**

这个特性已经在专门的[blog post](https://spring.io/blog/2014/07/24/spring-framework-4-1-handling-static-web-resources) 和Spring框架的参考文档([reference documentation](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#mvc-config-static-resources))中有详细的介绍。

### Custom Favicon

Spring Boot会在配置的静态内容位置和类路径的根目录（按此顺序）中查找favicon.ico。 如果这样的文件存在，它会自动用作应用程序的图标。

### ConfigurableWebBindingInitializer

Spring MVC使用WebBindingInitializer为特定请求初始化WebDataBinder。 如果您创建自己的ConfigurableWebBindingInitializer @Bean，则Spring Boot将自动配置Spring MVC以使用它。

### Template engines

除了REST Web服务，您还可以使用Spring MVC来提供动态HTML内容。 Spring MVC支持各种模板技术，包括Thymeleaf，FreeMarker和JSP。 许多其他模板引擎也提供了自己的Spring MVC集成。

Spring Boot包含以下模板引擎的自动配置支持：

* [FreeMarker](http://freemarker.org/docs/)
* [Groovy](http://docs.groovy-lang.org/docs/next/html/documentation/template-engines.html#_the_markuptemplateengine)
* [Thymeleaf](http://www.thymeleaf.org/)
* [Mustache](http://mustache.github.io/)

**Tip**

如果可能的话，JSP应该被避免，在嵌入的servlet容器中使用它们有几个已知的限制([known limitations](#_bookmark155) )。

当您使用默认配置的模板引擎之一时，您的模板将从 src/main/resources/templates 中自动获取。

**Tip**

IntelliJ IDEA根据你运行应用程序的方式不同地命令类路径。 通过主要方法在IDE中运行应用程序将导致与使用Maven或Gradle或从其打包的jar运行应用程序时不同的顺序。 这可能会导致Spring Boot无法在类路径中找到模板。 如果您受到这个问题的影响，您可以在IDE中重新排序类路径，以便首先放置模块的类和资源。 或者，您可以配置模板前缀以搜索类路径中的每个模板目录：classpath \*：/ templates /。

### Error Handling

Spring Boot默认提供了一个/error，以合理的方式处理所有的错误，并且在servlet容器中被注册为一个“global”错误页面。 对于机器客户端，它将产生一个JSON响应，包含错误的详细信息，HTTP状态和异常消息。 对于浏览器客户端，有一个'whitelabel'错误视图，它以HTML格式呈现相同的数据（自定义它只是添加一个解决'error'的视图）。 要完全替换默认行为，您可以实现ErrorController并注册该类型的bean定义，或者只需添加ErrorAttributes类型的bean以使用现有机制，但替换内容即可。

**Tip**

BasicErrorController可以用作自定义ErrorController的基类。 如果要为新的内容类型添加处理程序（默认情况下是专门处理text/html 并提供所有其他内容的回退），此功能特别有用。 为此，只需扩展BasicErrorController并添加一个带有@RequestMapping属性的公共方法，并创建一个新类型的bean。

您还可以定义@ControllerAdvice来定制JSON文档以返回特定的控制器和/或异常类型。

@ControllerAdvice(basePackageClasses = FooController.class)

**public class** FooControllerAdvice **extends** ResponseEntityExceptionHandler {

@ExceptionHandler(YourException.class) @ResponseBody

ResponseEntity<?> handleControllerException(HttpServletRequest request, Throwable ex) { HttpStatus status = getStatus(request);

**return new** ResponseEntity<>(**new** CustomErrorType(status.value(), ex.getMessage()), status);

}

**private** HttpStatus getStatus(HttpServletRequest request) {

Integer statusCode = (Integer) request.getAttribute(***"javax.servlet.error.status\_code"***);

**if** (statusCode == null) {

**return** HttpStatus.INTERNAL\_SERVER\_ERROR;

}

**return** HttpStatus.valueOf(statusCode);

}

}

在上面的例子中，如果由与FooController相同的包中定义的控制器抛出YourException，则将使用CustomErrorType POJO的json表示而不是ErrorAttributes表示形式。

#### 自定义错误页面

如果要为给定的状态代码显示自定义的HTML错误页面，请将文件添加到 /error 文件夹。 错误页面可以是静态HTML（即添加在任何静态资源文件夹下），也可以使用模板构建。 该文件的名称应该是确切的状态码或一系列掩码。

例如，要将404映射到静态HTML文件，您的文件夹结构如下所示：

src/

+- main/

+- java/

| + <source code>

+- resources/

+- public/

+- error/

| +- 404.html

+- <other public assets>

要使用FreeMarker模板来映射所有的5xx错误，你会有这样的结构：

src/

+- main/

+- java/

| + <source code>

+- resources/

+- templates/

+- error/

| +- 5xx.ftl

+- <other templates>

对于更复杂的映射，您还可以添加实现ErrorViewResolver的Bean接口。

**public class** MyErrorViewResolver **implements** ErrorViewResolver {

@Override

**public** ModelAndView resolveErrorView(HttpServletRequest request, HttpStatus status, Map<String, Object> model) {

*// Use the request or status to optionally return a ModelAndView*

**return** ...

}

}

您还可以使用常规的Spring MVC功能，如[@ExceptionHandler methods](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#mvc-exceptionhandlers)和

[@ControllerAdvice](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#mvc-ann-controller-advice)。 ErrorController将会接收任何未处理的异常。

#### 映射Spring MVC 之外的页面

对于不使用Spring MVC的应用程序，可以使用ErrorPageRegistrar接口直接注册ErrorPages。 这个抽象直接与底层的嵌入式servlet容器一起工作，即使你没有Spring MVC DispatcherServlet也可以工作。

@Bean

**public** ErrorPageRegistrar errorPageRegistrar(){

**return new** MyErrorPageRegistrar();

}

*// ...*

**private static class** MyErrorPageRegistrar **implements** ErrorPageRegistrar { @Override

**public void** registerErrorPages(ErrorPageRegistry registry) { registry.addErrorPages(**new** ErrorPage(HttpStatus.BAD\_REQUEST, ***"/400"***));

}

}

注： 如果你注册了一个错误页面，并且这个错误页面将被一个过滤器处理的路径（例如，像一些非Spring web框架，比如Jersey和Wicket），那么这个过滤器必须被明确地注册为一个错误调度器。

@Bean

**public** FilterRegistrationBean myFilter() {

FilterRegistrationBean registration = **new** FilterRegistrationBean(); registration.setFilter(**new** MyFilter());

...

registration.setDispatcherTypes(EnumSet.allOf(DispatcherType.**class**));

**return** registration;

}

（默认FilterRegistrationBean不包括ERROR调度程序类型）。

#### WebSphere Application Server上的错误处理

在部署到servlet容器时，Spring Boot使用其错误页面过滤器将具有错误状态的请求转发到适当的错误页面。 如果响应尚未提交，则只能将请求转发到正确的错误页面。 默认情况下，WebSphere Application Server 8.0和更高版本在成功完成servlet的服务方法后提交响应。 您应该通过将com.ibm.ws.webcontainer.invokeFlushAfterService设置为false来禁用此行为

### Spring HATEOAS

如果您正在开发使用超媒体(hypermedia)的RESTful API，Spring Boot为Spring HATEOAS提供了自动配置，可以与大多数应用程序配合使用。 自动配置取代了使用@EnableHypermediaSupport并注册大量bean的需求，以便于构建基于超媒体(hypermedia)的应用程序，包括LinkDiscoverers（用于客户端支持）和ObjectMapper，用于将响应正确地编组为所需的表示形式。 ObjectMapper将根据spring.jackson.\*属性或Jackson2ObjectMapperBuilder bean（如果存在）进行自定义。

你可以使用@EnableHypermediaSupport来控制Spring HATEOAS的配置。 请注意，这将禁用上述的ObjectMapper定制。

### CORS support

跨源资源共享([Cross-origin resource sharing](http://en.wikipedia.org/wiki/Cross-origin_resource_sharing) (CORS))是大多数浏览器([most browsers](http://caniuse.com/#feat%3Dcors) )实现的[W3C specification](http://www.w3.org/TR/cors/) 规范，允许您以灵活的方式指定什么样的跨域请求被授权，而不是使用像IFRAME或JSONP那样的不太安全和功能较弱的方法。

从4.2版开始，Spring MVC支持CORS([supports CORS](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#cors))开箱即用。 使用[controller method CORS](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#_controller_method_cors_configuration) [configuration](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#_controller_method_cors_configuration) 在Spring Boot应用程序中使用[@CrossOrigin](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html) 标注的CORS配置不需要任何特定的配置。 [Global CORS configuration](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#_global_cors_configuration) 可以通过使用定制的addCorsMappings（CorsRegistry）方法注册WebMvcConfigurer bean来定义：

@Configuration

**public class** MyConfiguration {

@Bean

**public** WebMvcConfigurer corsConfigurer() {

**return new** WebMvcConfigurerAdapter() { @Override

**public void** addCorsMappings(CorsRegistry registry) { registry.addMapping(***"/api/\*\*"***);

}

};

}

}

## JAX-RS and Jersey

如果您更喜欢REST endpoints的JAX-RS编程模型，则可以使用其中一个可用的实现，而不是Spring MVC。 Jersey 1.x和Apache CXF在开箱即用的情况下工作得非常好，如果您只是在您的应用程序上下文中将其Servlet或Filter注册为@Bean。 Jersey 2.x有一些原生的Spring支持，所以我们还在Spring Boot中为它提供了自动配置支持。

要开始使用Jersey 2.x，只需将spring-boot-starter-jersey作为依赖包含在内，然后您需要一个ResourceConfig类型的@Bean，其中注册了所有的端点(endpoints)：

@Component

**public class** JerseyConfig **extends** ResourceConfig {

**public** JerseyConfig() { register(Endpoint.**class**);

}

}

**Warning**

Jersey’s对扫描可执行档案的支持是相当有限的。 例如，运行可执行的war文件时，它不能扫描WEB-INF/classes中的包中的端点(endpoints)。 为了避免这种限制，不应该使用软件包方法，并且应该使用如上所示的注册方法单独注册端点(endpoints)。

您还可以注册任意数量的实现ResourceConfigCustomizer的bean以进行更高级的自定义。

所有注册的端点(endpoints)应该是具有HTTP资源注释（@GET等）的@Components，例如

@Component @Path("/hello")

**public class** Endpoint {

@GET

**public** String message() {

**return *"Hello"***;

}

}

由于Endpoint是一个Spring @Component，因此它的生命周期由Spring管理，您可以使用@Autowired依赖关系，并使用@Value注入外部配置。 Jersey servlet将被默认注册并映射到/ \*。 您可以通过将@ApplicationPath添加到您的ResourceConfig来更改映射。

默认情况下，Jersey将被设置为一个名为jerseyServletRegistration的ServletRegistrationBean类型的@Bean中的Servlet。 默认情况下，servlet会被懒惰地初始化(initialized lazily)，但是你可以通过spring.jersey.servlet.load-on-startup来定制它。你可以通过创建一个你自己的名字来禁用或者重载这个bean。 你也可以通过设置spring.jersey.type = filter来使用Filter而不是Servlet（在这种情况下，替换或覆盖的是@Bean）是jerseyFilterRegistration。 这个servlet有一个@Order，你可以使用spring.jersey.filter.order来设置。 Servlet和Filter注册都可以通过使用spring.jersey.init.\*来指定init参数来指定属性映射。

有一个[Jersey sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-jersey) ，所以你可以看到如何设置的东西。 还有一个[Jersey 1.x sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-jersey1)。 请注意，在Jersey 1.x示例中，spring-boot maven插件已被配置为解压缩某些Jersey jar(unpack some Jersey jars)，以便JAX-RS可以扫描它们（因为示例要求在其Filter注册中扫描它们）。 如果您的任何JAX-RS资源打包为嵌套jar，则可能需要执行相同的操作。

## 嵌入式servlet容器支持

Spring Boot包括对嵌入式Tomcat，Jetty和Undertow服务器的支持。 大多数开发人员只需使用适当的“Starter”来获取完全配置的实例。 默认情况下，嵌入式服务器将侦听端口8080上的HTTP请求。

**Warning**

如果您选择在CentOS上使用Tomcat，请注意，默认情况下会使用临时目录来存储已编译的JSP，文件上载等。当您的应用程序运行时，tmpwatch可能会删除此目录导致失败。 为了避免这种情况，你可能想要自定义你的tmpwatch配置，以便tomcat.\*目录不被删除或配置server.tomcat.basedir，以便嵌入式Tomcat使用不同的位置。

### Servlets, Filters, and listeners

当使用嵌入式servlet容器时，可以通过使用Spring bean或通过扫描Servlet组件来注册Servlet，Filters和Servlet规范中的所有监听器（例如HttpSessionListener）。

#### Registering Servlets, Filters, and listeners as Spring beans

任何作为Spring bean的Servlet，Filter或Servlet \* Listener实例都将在嵌入容器中注册。 如果要在配置期间从application.properties中引用值，这可能会特别方便。

默认情况下，如果上下文(context)只包含一个Servlet，它将被映射到/。 在多个Servlet bean的情况下，bean名称将被用作路径前缀。 过滤器将映射到/ \*。

### Servlet Context Initialization

嵌入式servlet容器不会直接执行Servlet 3.0+ javax.servlet.ServletContainerInitializer接口或Spring的org.springframework.web.WebApplicationInitializer接口。 这是一个有意的设计决策，旨在降低设计在战争中运行的第三方库将破坏Spring Boot应用程序的风险。

如果您需要在Spring Boot应用程序中执行servlet上下文初始化，则应该注册一个实现了org.springframework.boot.context.embedded.ServletContextInitializer接口的bean。 单一的onStartup方法提供了对ServletContext的访问，并且如果需要的话可以很容易地用作现有WebApplicationInitializer的适配器。

#### Scanning for Servlets, Filters, and listeners

当使用嵌入式容器时，自动注册@WebServlet，@WebFilter和

可以使用@ServletComponentScan启用@WebListener注释类。

**Tip**

@ServletComponentScan在独立的容器中将不起作用，容器的内置发现机制将被使用。

### The EmbeddedWebApplicationContext

Spring Boot使用了一种新型的ApplicationContext来实现对嵌入式servlet容器的支持。 EmbeddedWebApplicationContext是一种特殊类型的WebApplicationContext，通过搜索单个EmbeddedServletContainerFactory bean来引导自身。 通常，TomcatEmbeddedServletContainerFactory，JettyEmbeddedServletContainerFactory或UndertowEmbeddedServletContainerFactory将被自动配置。

**Note**

您通常不需要知道这些实现类。 大多数应用程序将被自动配置，并且将代表您创建相应的ApplicationContext和EmbeddedServletContainerFactory。

### Customizing embedded servlet containers

通用的servlet容器设置可以使用Spring Environment属性进行配置。 通常你会在你的application.properties文件中定义属性。

通用服务器设置包括:

* 网络设置：传入HTTP请求的侦听端口（server.port），绑定到server.address的接口地址等
* 会话(Session)设置：会话是否持续（server.session.persistence），会话超时（server.session.timeout），会话数据的位置（server.session.store-dir）和会话cookie配置（server.session.cookie）。
* 错误管理：错误页面的位置（server.error.path）等
* [SSL](#_bookmark464)
* [HTTP compression](#_bookmark486)

Spring Boot尽可能地尝试暴露常见的设置，但这并不总是可能的。 对于这些情况，专用名称空间提供了特定于服务器的定制（请参阅server.tomcat和server.undertow）。 例如，可以使用嵌入式servlet容器的特定功能来配置访问日志([access logs](#_bookmark465) )。

**Tip**

参阅 [ServerProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java) 类以获取完整列表

#### 程序化定制 (Programmatic customization)

如果您需要以编程方式配置嵌入式Servlet容器，则可以注册一个实现了EmbeddedServletContainerCustomizer接口的Spring bean。 EmbeddedServletContainerCustomizer提供对ConfigurableEmbeddedServletContainer的访问，其中包含大量定制设置方法。

**import** org.springframework.boot.context.embedded.\*;

**import** org.springframework.stereotype.Component;

@Component

**public class** CustomizationBean **implements** EmbeddedServletContainerCustomizer {

@Override

**public void** customize(ConfigurableEmbeddedServletContainer container) { container.setPort(9000);

}

}

#### 直接自定义 ConfigurableEmbeddedServletContainer

如果上述自定义技术太有限，则可以自己注册TomcatEmbeddedServletContainerFactory，JettyEmbeddedServletContainerFactory或UndertowEmbeddedServletContainerFactory bean。

@Bean

**public** EmbeddedServletContainerFactory servletContainer() { TomcatEmbeddedServletContainerFactory factory = **new** TomcatEmbeddedServletContainerFactory(); factory.setPort(9000);

factory.setSessionTimeout(10, TimeUnit.MINUTES);

factory.addErrorPages(**new** ErrorPage(HttpStatus.NOT\_FOUND, ***"/notfound.html"***));

**return** factory;

}

安装程序提供了许多配置选项。 还提供了几个受保护的方法“hooks”，如果你需要做更奇特的事情。 有关详细信息，请参阅源代码文档。

### JSP 限制

运行使用嵌入式servlet容器的Spring Boot应用程序（并打包为可执行文件）时，JSP支持有一些限制。

* 使用Tomcat时，如果使用war包，即可执行的war会运行，并且可以部署到标准容器（不限于Tomcat），也可以部署到Tomcat。 由于Tomcat中的硬编码文件模式，可执行jar不起作用。
* 使用Jetty时，如果使用war打包，即可执行的war可以工作，并且也可以部署到任何标准容器，则应该可以工作。[error handling](#_bookmark139), [custom error pages](#_bookmark140)
* Undertow 不支持 JSPs.
* 创建自定义error.jsp页面不会覆盖错误处理([error handling](#_bookmark139))的默认视图，而应该使用自定义错误页面( [custom error pages](#_bookmark140) )。

# Security

如果Spring Security位于类路径上，那么默认情况下，Web应用程序将在所有HTTP端点上进行“basic”身份验证。 要向Web应用程序添加方法级别的安全性，您还可以添加所需设置的@EnableGlobalMethodSecurity。 更多的信息可以在[Spring Security Reference](http://docs.spring.io/spring-security/site/docs/4.2.3.RELEASE/reference/htmlsingle#jc-method).中找到。

默认的AuthenticationManager有一个用户（'user'用户名和随机密码，当应用程序启动时以INFO级别打印）

Using default security password: 78fa095d-3f4c-48b1-ad50-e24c31d5cf35

**Note**

如果您对日志记录配置进行了微调，请确保将org.springframework.boot.autoconfigure.security类别设置为记录INFO消息，否则将不会打印默认密码。

您可以通过提供一个security.user.password来更改密码。 这个和其他有用的属性通过[SecurityProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/SecurityProperties.java) （属性前缀“security”）来外化。

默认的安全配置是在SecurityAutoConfiguration和从那里导入的类中实现的（SpringBootWebSecurityConfiguration用于web安全性，AuthenticationManagerConfiguration用于在非web应用中也是相关的认证配置）。 要完全关闭默认的Web应用程序安全配置，您可以使用@EnableWebSecurity添加一个bean（这不会禁用认证管理器配置或Actuator的安全性）。 要定制它，通常使用WebSecurityConfigurerAdapter类型的外部属性和bean（例如添加基于表单的登录）。

**Note**

如果添加@EnableWebSecurity并禁用Actuator安全性，则将为整个应用程序获取默认的基于表单的登录，除非添加自定义的WebSecurityConfigurerAdapter。

要关闭身份验证管理器配置(authentication manager configuration)，您可以添加一个AuthenticationManager类型的bean，或者通过将AuthenticationManagerBuilder自动装配到您的一个@Configuration类中的方法来配置全局AuthenticationManager。 在[Spring Boot samples](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/) 中有几个安全的应用程序来帮助您开始使用常见用例。

您在Web应用程序中使用的基本功能是：

▪ 一个好的AuthenticationManager bean在内存中存储和单个用户（见SecurityProperties.User作为用户的属性）。

* 常见静态资源位置 (/css/\*\*, /js/\*\*, /images/\*\*, / webjars/\*\* and \*\*/favicon.ico).
* 所有其他端点的HTTP基本安全性。
* 发布到Spring的ApplicationEventPublisher的安全事件（成功和失败的身份验证和访问被拒绝）。
* Spring Security提供的常见底层特性（HSTS，XSS，CSRF，caching）默认是开启的。

以上所有功能都可以使用外部属性（security.\*）来打开和关闭或修改。 要覆盖访问规则而不更改任何其他自动配置的功能，请使用@Order（SecurityProperties.ACCESS\_OVERRIDE\_ORDER）添加一个类型为WebSecurityConfigurerAdapter的@Bean，并对其进行配置以满足您的需求。

**Note**

默认情况下，WebSecurityConfigurerAdapter将匹配任何路径。 如果您不想完全覆盖Spring Boot的自动配置的访问规则，则您的适配器必须显式配置您想覆盖的路径。

## OAuth2

如果在类路径上有spring-security-oauth2，则可以利用一些自动配置来轻松设置授权或资源服务器。 有关完整的详细信息，请参阅[Security OAuth 2 Developers Guide](http://projects.spring.io/spring-security-oauth/docs/oauth2.html)。

### 授权服务器

要创建授权服务器并授予访问令牌(tokens)，您需要使用@EnableAuthorizationServer并提供security.oauth2.client.client-id和security.oauth2.client.client-secret]属性。 客户端将在内存资源库中注册。

完成之后，您将能够使用客户端凭据创建访问令牌(token)，例如：

$ curl client:secret@localhost:8080/oauth/token -d grant\_type=password -d username=user -d password=pwd

/ token端点的基本认证凭证是client-id和client-secret。 用户凭证是普通的Spring Security用户详细信息（在Spring Boot中默认为“user”和一个随机密码）。

### 资源服务器

要使用访问令牌(token)，您需要一个资源服务器（可以与授权服务器相同）。 创建资源服务器很简单，只需添加@EnableResourceServer并提供一些配置，以允许服务器解码访问令牌(token)。 如果您的应用程序也是一个授权服务器，它已经知道如何解码令牌(token)，所以没有别的办法。 如果你的应用是一个独立的服务，那么你需要给它更多的配置，以下选项之一：

* security.oauth2.resource.user-info-uri使用/ me resource（例如Pivotal Web Services（PWS）上的https：// uaa.run.pivotal.io/userinfo）

▪ security.oauth2.resource.token-info-uri使用令牌(token)解码端点（例如，

PWS上的https://uaa.run.pivotal.io/check\_token）。

如果你同时指定了user-info-uri和token-info-uri，那么你可以设置一个标志来说明一个是比另一个优先的（prefer-token-info = true是默认值）。

如果令牌(token)是JWT，或者（而不是user-info-uri或token-info-uri），则可以配置security.oauth2.resource.jwt.key-value以在本地对其进行解码（其中密钥是验证密钥）。 验证密钥值是对称密钥或PEM编码的RSA公钥。 如果你没有密钥并且它是公开的，你可以提供一个可以下载的URI（作为带有“value”字段的JSON对象）和security.oauth2.resource.jwt.key-uri。 例如。 在PWS上：

$ curl https://uaa.run.pivotal.io/token\_key

{"alg":"SHA256withRSA","value":"-----BEGIN PUBLIC KEY-----\nMIIBI...\n-----END PUBLIC KEY-----\n"}

此外，如果您的授权服务器具有返回一组JSON Web密钥（JWK）的端点，则可以配置security.oauth2.resource.jwk.key-set-uri。 例如。 在PWS上：

$ curl https://uaa.run.pivotal.io/token\_keys

{"keys":[{"kid":"key-1","alg":"RS256","value":"-----BEGIN PUBLIC KEY-----\nMIIBI...\n-----END PUBLIC KEY-----\n"]}

**Note**

配置JWT和JWK属性都会导致错误。 应该只配置security.oauth2.resource.jwt.key-uri（或security.oauth2.resource.jwt.key-value）和security.oauth2.resource.jwk.key-set-uri中的一个。

**Warning**

如果使用security.oauth2.resource.jwt.key-URI或

`security.oauth2.resource.jwk.key-set-uri'，授权服务器需要在应用程序启动时运行。 如果找不到密钥，它会记录警告，并告诉您如何解决问题。

OAuth2资源受过滤器链保护，顺序为security.oauth2.resource.filter- order，缺省情况下，过滤器默认保护执行器端点（因此执行器端点将保持HTTP Basic，除非您更改顺序）。

## Token 类型的用户信息

Google和某些其他第三方身份提供商对标头中发送给用户信息端点的标记类型名称更为严格。 默认是适合大多数提供者的“Bearer”，并且符合规范，但是如果你需要改变它，你可以设置security.oauth2.resource.token-type。

## 自定义用户信息 RestTemplate

如果您有user-info-uri，资源服务器功能在内部使用OAuth2RestTemplate来获取用于验证的用户详细信息。

这是作为UserInfoRestTemplateFactory类型的@Bean提供的。 对于大多数提供者来说，默认值应该是可

以的，但是偶尔你可能需要添加额外的拦截器，或者改变请求认证者（这是令牌被附加到传出请求的方式）。

要添加一个自定义，只需创建一个UserInfoRestTemplateCustomizer类型的bean - 它有一个方法，在bean被

创建之后但在被初始化之前被调用。 正在定制的其余模板这里只是用来进行身份验证。 或者，你可以定义你

自己的UserInfoRestTemplateFactory @Bean来完全控制。

**Tip**

要在YAML中设置RSA键值，可以使用“管道”连续标记将其分割成多行（“|”）并记住缩进键值（这是标准的YAML语言功能）。 例：

**security**:

**oauth2**:

**resource**:

**jwt**:

**keyValue**: |

-----BEGIN PUBLIC KEY----- MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKC...

-----END PUBLIC KEY-----

### Client

要将您的Web应用程序转换为OAuth2客户端，您可以简单地添加@EnableOAuth2Client，Spring Boot将创建一个OAuth2ClientContext和OAuth2ProtectedResourceDetails，这些是创建OAuth2RestOperations所必需的。 Spring Boot不会自动创建这样的bean，但您可以轻松创建自己的：

@Bean

**public** OAuth2RestTemplate oauth2RestTemplate(OAuth2ClientContext oauth2ClientContext, OAuth2ProtectedResourceDetails details) {

**return new** OAuth2RestTemplate(details, oauth2ClientContext);

}

**Note**

您可能需要添加限定符，并将您的配置作为多个RestTemplate进行检查

可能会在您的应用程序中定义。

此配置使用security.oauth2.client.\*作为凭证（与您在Authorization Server中使用的相同），但此外还需要了解授权服务器中的授权和令牌URI。 例如：

**application.yml.**

**security**:

**oauth2**:

**client**:

**clientId**: bd1c0a783ccdd1c9b9e4

**clientSecret**: 1a9030fbca47a5b2c28e92f19050bb77824b5ad1 **accessTokenUri**: https://github.com/login/oauth/access\_token **userAuthorizationUri**: https://github.com/login/oauth/authorize **clientAuthenticationScheme**: form

当您尝试使用OAuth2RestTemplate时，具有此配置的应用程序将重定向到Github进行授权。 如果您已经登录到Github，您甚至不会注意到它已经通过身份验证。 如果您的应用程序在端口8080上运行（在Github或其他提供程序中注册您自己的客户端应用程序以获得更大的灵活性），这些特定的凭据将仅适用。

限制客户端在获取您可以设置的访问令牌时所要求的范围

security.oauth2.client.scope（逗号分隔或YAML中的数组）。 默认情况下，作用域为空，授权服务器决定默认设置应该是什么，通常取决于它所保存的客户端注册中的设置。

**Note**

还有一个curity.oauth2.client.client-authentication-scheme默认为“header”的设置（但如果像Github一样，你可能需要将它设置为“form”，你的OAuth2提供者不喜欢头认证）。 实际上，security.oauth2.client.\*属性绑定到AuthorizationCodeResourceDetails的一个实例，因此可以指定它的所有属性。

**Tip**

在非web应用程序中，您仍然可以创建OAuth2RestOperations，并且它仍然连接到security.oauth2.client.\*配置。 在这种情况下，它是一个“客户端凭据令牌授予(client credentials token grant)”，如果您使用它（并且不需要使用@EnableOAuth2Client或@EnableOAuth2Sso），您将会询问您。 要防止定义基础结构，只需从您的配置中删除security.oauth2.client.client-id（或将其设置为空字符串）。

### 单点登录

可以使用OAuth2客户端从提供者(provider)获取用户详细信息（如果这些功能可用），然后将其转换为Spring Security的身份验证令牌。 上面的资源服务器通过user-info-uri属性支持这一点。这是基于OAuth2的单点登录（SSO）协议的基础，Spring Boot通过提供注释@EnableOAuth2Sso使参与变得容易。 上面的Github客户端可以使用Github / user / endpoint保护所有的资源并进行身份验证，通过添加注释并声明在哪里可以找到端点（除了上面已经列出的security.oauth2.client.\*配置外）：

**application.yml.**

**security**:

**oauth2**:

...

**resource**:

**userInfoUri**: https://api.github.com/user

**preferTokenInfo**: **false**

由于默认情况下所有路径都是安全的，因此没有“home”页面，您可以向未经身份验证的用户显示并请他们登录（通过访问/ login路径或security.oauth2.sso.login-path指定的路径）。

要定制访问规则或路径来保护，所以你可以添加一个“home”页面，@EnableOAuth2Sso可以被添加到一个WebSecurityConfigurerAdapter中，并且这个注释将使它被装饰和增强，并带有必要的部分来获得/ login 路径工作。 例如，在这里，我们只允许对“/”的主页进行未经验证的访问，并保留其他所有的默认值：

@Configuration

**static class** WebSecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Override

**public void** init(WebSecurity web) { web.ignoring().antMatchers(***"/"***);

}

@Override

**protected void** configure(HttpSecurity http) **throws** Exception { http.antMatcher(***"/\*\*"***).authorizeRequests().anyRequest().authenticated();

}

}

## Actuator Security

如果Actuator也在使用中，你会发现：

* 即使应用程序端点不安全，管理端点也是安全的。
* 安全事件转换为AuditEvent实例并发布到AuditEventRepository。
* 默认用户将具有ACTUATOR角色以及USER角色。

执行器(Acturator)安全功能可以使用外部属性（management.security.\*）进行修改。 要覆盖应用程序访问规则，请添加WebSecurityConfigurerAdapter类型的@Bean，如果您不想覆盖执行程序访问规则，请使用@Order（SecurityProperties.ACCESS\_OVERRIDE\_ORDER），如果您要覆盖执行程序访问规则，请使用@Order（ManagementServerProperties.ACCESS\_OVERRIDE\_ORDER） 执行器访问规则。

# 使用 SQL 数据库

Spring框架为使用SQL数据库提供了广泛的支持。 从使用JdbcTemplate的直接JDBC访问到完成诸如Hibernate的“对象关系映射”技术。 Spring Data提供了额外的功能级别，直接从接口创建Repository实现，并使用约定从方法名称中生成查询。

## 配置数据源(DataSource)

Java的javax.sql.DataSource接口提供了使用数据库连接的标准方法。 传统上，DataSource使用URL和一些凭据来建立数据库连接。

**Tip**

检查更多高级示例的“[the ‘How-to’ section](#_bookmark505) ”部分，通常要完全控制DataSource的配置。

### 支持嵌入式数据源(Database)

使用内存嵌入式数据库开发应用程序通常很方便。 显然，内存数据库不提供持久性存储; 您将需要在应用程序启动时填充数据库，并准备在应用程序结束时丢弃数据。

**Tip**

“操作方法”部分包含有关如何初始化数据库的一节[*section on how to initialize a database*](#_bookmark517)

Spring Boot可以自动配置嵌入式[H2](http://www.h2database.com/), [HSQL](http://hsqldb.org/) 和 [Derby](http://db.apache.org/derby/) 数据库。 您不需要提供任何连接URL，只需将构建依赖项包含到要使用的嵌入式数据库中即可。

**Note**

如果您在测试中使用此功能，则可能会注意到无论您使用的是多少个应用程序上下文，整个测试套件都会重复使用相同的数据库。 如果你想确保每个上下文有一个单独的嵌入式数据库，你应该设置spring.datasource.generate-unique-name为true。

例如，典型的POM依赖关系是：

**<dependency>**

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

**<artifactId>**spring-boot-starter-data-jpa**</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>**org.hsqldb**</groupId>**

**<artifactId>**hsqldb**</artifactId>**

**<scope>**runtime**</scope>**

**</dependency>**

**Note**

您需要依靠spring-jdbc来自动配置嵌入式数据库。 在这个例子中，它是通过spring-boot-starter-data-jpa传递的。

**Tip**

如果出于某种原因，您要为嵌入式数据库配置连接URL，则应注意确保数据库的自动关闭被禁用。 如果您使用H2，则应使用DB\_CLOSE\_ON\_EXIT = FALSE来执行此操作。 如果你正在使用HSQLDB，你应该确保不使用shutdown = true。 禁用数据库的自动关闭允许Spring Boot控制何时关闭数据库，从而确保在不再需要访问数据库时发生。

### 连接到生产数据库

生产数据库连接也可以使用池化数据源自动配置。 以下是选择特定实现的算法：

* 我们更喜欢Tomcat将数据源集成到DataSource的性能和并发性，所以如果可用的话，我们总是选择它。
* 否则，如果HikariCP可用，我们将使用它。
* 如果Tomcat池数据源和HikariCP都不可用，并且Commons DBCP可用，我们将使用它，但我们不建议在生产中使用它，并且不支持它。
* 最后，如果Commons DBCP2可用，我们将使用它。

如果使用spring-boot-starter-jdbc或spring-boot-starter-data-jpa'starters'，则会自动获得对tomcat-jdbc的依赖关系。

**Note**

您可以完全绕过该算法，并通过spring.datasource.type属性指定要使用的连接池。 如果您在Tomcat容器中运行应用程序，这一点尤其重要，因为默认情况下提供了tomcat-jdbc。

**Tip**

其他连接池可以始终手动配置。 如果你定义你自己的数据源bean，自动配置将不会生效。

DataSource配置由spring.datasource.\*中的外部配置属性控制。 例如，您可以在application.properties中声明以下部分：

**spring.datasource.url**=jdbc:mysql://localhost/test **spring.datasource.username**=dbuser **spring.datasource.password**=dbpass **spring.datasource.driver-class-name**=com.mysql.jdbc.Driver

**Note**

您至少应该使用spring.datasource.url属性指定url，否则Spring Boot将尝试自动配置嵌入式数据库。

**Tip**

您通常不需要指定驱动程序类名称，因为Spring引导可以从url中为大多数数据库推断它。

**Note**

对于要创建的池数据源，我们需要能够验证有效的驱动程序类是否可用，所以我们在做任何事之前都要检查它。即 如果你设置spring.datasource.driver-class-name = com.mysql.jdbc.Driver，那么这个类必须是可加载的。

有关更多受支持的选项，请参阅[DataSourceProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceProperties.java) 。 这些是标准选项，不管实际的实施情况如何。 还可以使用各自的前缀（spring.datasource.tomcat.\*，spring.datasource.hikari.\*和spring.datasource.dbcp2.\*）来微调特定于实现的设置。 请参阅您正在使用的连接池实现的文档以获取更多详细信息。

例如，如果您正在使用Tomcat连接池，则可以自定义许多其他设置：

*# Number of ms to wait before throwing an exception if no connection is available.*

**spring.datasource.tomcat.max-wait**=10000

*# Maximum number of active connections that can be allocated from this pool at the same time.*

**spring.datasource.tomcat.max-active**=50

*# Validate the connection before borrowing it from the pool.*

**spring.datasource.tomcat.test-on-borrow**=true

### 连接到一个JNDI数据源

如果您将Spring Boot应用程序部署到应用程序服务器，则可能需要使用应用程序服务器内置功能配置和管理数据源，并使用JNDI访问它。

spring.datasource.jndi-name属性可以用来替代spring.datasource.url，spring.datasource.username和spring.datasource.password属性来访问特定JNDI位置的数据源。 例如，application.properties中的以下部分显示了如何访问JBoss AS定义的数据源：

**spring.datasource.jndi-name**=java:jboss/datasources/customers

## Using JdbcTemplate

Spring’s JdbcTemplate and NamedParameterJdbcTemplate classes are auto-configured and you can @Autowire them directly into your own beans:

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

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.stereotype.Component;

@Component

**public class** MyBean {

**private final** JdbcTemplate jdbcTemplate; @Autowired

**public** MyBean(JdbcTemplate jdbcTemplate) {

**this**.jdbcTemplate = jdbcTemplate;

}

*// ...*

}

## JPA 和 ‘Spring Data’

Java持久性API是一种标准技术，允许您将对象映射到关系数据库。 spring-boot-starter-data-jpa POM提供了一个快速开始的方法。 它提供了以下关键依赖关系：

* Hibernate - 最流行的JPA实现之一。
* Spring Data JPA - 使实现基于JPA的存储库变得很容易。
* Spring ORM - Spring框架的核心ORM支持。

**Tip**

这里我们不会涉及太多的JPA或Spring Data的细节。 您可以按照[spring.io](http://spring.io/)中的'使用JPA访问数据([‘Accessing](http://spring.io/guides/gs/accessing-data-jpa/) [Data with JPA’](http://spring.io/guides/gs/accessing-data-jpa/))'指导并阅读[Spring Data JPA](http://projects.spring.io/spring-data-jpa/)和[Hibernate](http://hibernate.org/orm/documentation/)参考文档。

**Note**

默认情况下，Spring Boot使用Hibernate 5.0.x. 但是，如果您愿意，也可以使用4.3.x或5.2.x。 请参阅[Hibernate 4](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-hibernate4) 和 [Hibernate 5.2](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-hibernate52) 示例以了解如何执行此操作。

### 实体类(Entity Classes)

传统上，JPA的“实体”类是在persistence.xml文件中指定的。 使用Spring Boot这个文件是不必要的，而是使用“实体扫描”。 默认情况下，将会搜索主配置类下的所有包（使用@EnableAutoConfiguration或@SpringBootApplication注解的包）。

任何用@Entity，@Embeddable或@MappedSuperclass注解的类都将被考虑。 典型的实体类看起来像这样：

**package** com.example.myapp.domain;

**import** java.io.Serializable;

**import** javax.persistence.\*;

@Entity

**public class** City **implements** Serializable {

@Id @GeneratedValue **private** Long id;

@Column(nullable = false)

**private** String name;

@Column(nullable = false)

**private** String state;

*// ... additional members, often include @OneToMany mappings*

**protected** City() {

*// no-args constructor required by JPA spec*

*// this one is protected since it shouldn't be used directly*

}

**public** City(String name, String state) {

**this**.name = name;

**this**.country = country;

}

**public** String getName() {

**return this**.name;

}

**public** String getState() {

**return this**.state;

}

*// ... etc*

}

**Tip**

您可以使用@EntityScan批注自定义实体扫描位置。 [*Section 77.4, “Separate @Entity definitions from Spring configuration”*](#_bookmark508) 。

### Spring Data JPA Repositories

Spring Data JPA存储库是您可以定义以访问数据的接口。 JPA查询是从您的方法名称自动创建的。 例如，CityRepository接口可能会声明findAllByState（String state）方法来查找给定状态下的所有城市。

对于更复杂的查询，您可以使用Spring Data Query注释来注释您的方法。

Spring数据存储库通常从 [Repository](http://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/Repository.html) 或 [CrudRepository](http://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html) 接口扩展而来。 如果使用自动配置，则将从包含主配置类（使用@EnableAutoConfiguration或@SpringBootApplication注释的那个）的包中搜索存储库。

这是一个典型的Spring数据库：

**package** com.example.myapp.domain;

**import** org.springframework.data.domain.\*;

**import** org.springframework.data.repository.\*;

**public interface** CityRepository **extends** Repository<City, Long> { Page<City> findAll(Pageable pageable);

City findByNameAndCountryAllIgnoringCase(String name, String country);

}

**Tip**

我们几乎没有触及Spring Data JPA的表面。 有关完整的详细信息，请查阅[reference documentation](http://projects.spring.io/spring-data-jpa/)。

### 创建和删除 JPA databases

默认情况下，只有在使用嵌入式数据库（H2，HSQL或Derby）时才会自动创建JPA数据库。 您可以使用spring.jpa.\*属性显式配置JPA设置。 例如，要创建和删除表，可以将以下内容添加到application.properties中。

spring.jpa.hibernate.ddl-auto=create-drop

**Note**

Hibernate自己的内部属性名称（如果你碰巧记得更好）是hibernate.hbm2ddl.auto。 您可以使用spring.jpa.properties.\*（在将前缀添加到实体管理器之前剥离前缀）将其与其他Hibernate本机属性一起设置。

Example:

spring.jpa.properties.hibernate.globally\_quoted\_identifiers=true

将hibernate.globally\_quoted\_identifiers传递给Hibernate实体管理器。

默认情况下，DDL执行（或验证）被推迟到ApplicationContext已经启动。 还有一个spring.jpa.generate-ddl标志，但是如果Hibernate自动配置处于活动状态，则不会使用该标志，因为ddl-auto设置更加精细。

### Open EntityManager in View

如果你正在运行一个web应用程序，Spring Boot将默认注册[OpenEntityManagerInViewInterceptor](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/orm/jpa/support/OpenEntityManagerInViewInterceptor.html) 来应用“在视图中打开EntityManager”模式，即允许在web视图中延迟加载。 如果你不想要这个行为，你应该在你的application.properties中将spring.jpa.open-in-view设置为false。

## 使用 H2’s web console

[H2 database](http://www.h2database.com/) 提供了一个基于浏览器的控制台([browser-based console](http://www.h2database.com/html/quickstart.html#h2_console) )，Spring Boot可以为你自动配置。 当满足以下条件时，控制台将自动配置：

* 您正在开发一个Web应用程序
* com.h2database：h2位于 classpath
* 您正在使用Spring Boot的开发人员工具 [Spring Boot’s developer tools](#_bookmark67)

**Tip**

如果你不使用Spring Boot的开发工具，但仍然想使用H2的控制台，那么你可以通过配置值为true的spring.h2.console.enabled属性来实现。 H2控制台仅用于开发过程中，因此应注意确保在生产环境中将spring.h2.console.enabled设置为true。

### 改变 H2 console’s path

默认情况下，控制台将在/ h2-console上可用。 您可以使用spring.h2.console.path属性自定义控制台的路径。

### Securing the H2 console

当Spring Security在类路径上并且启用了基本身份验证时，H2控制台将使用基本身份验证自动进行保护。 以下属性可用于自定义安全性配置：

* security.user.role
* security.basic.authorize-mode
* security.basic.enabled

## 使用 jOOQ

Java面向对象查询([jOOQ](http://www.jooq.org/))是[Data Geekery](http://www.datageekery.com/) 的一个流行产品，它可以从您的数据库中生成Java代码，并允许您通过流畅的API构建类型安全的SQL查询。 商业和开源版本都可以和Spring Boot一起使用。

### (代码生成)Code Generation

为了使用jOOQ类型安全的查询，您需要从数据库模式生成Java类。 您可以按照jOOQ用户手册([jOOQ user manual](http://www.jooq.org/doc/3.6/manual-single-page/#jooq-in-7-steps-step3))中的说明进行操作。 如果您使用的是jooq-codegen-maven插件（也可以使用spring-boot-starter-parent“parent POM”），则可以安全地省略插件的<version>标签。 您也可以使用Spring Boot定义的版本变量（例如h2.version）来声明插件的数据库依赖性。 这是一个例子：

**<plugin>**

**<groupId>**org.jooq**</groupId>**

**<artifactId>**jooq-codegen-maven**</artifactId>**

**<executions>**

...

**</executions>**

**<dependencies>**

**<dependency>**

**<groupId>**com.h2database**</groupId>**

**<artifactId>**h2**</artifactId>**

**<version>**${h2.version}**</version>**

**</dependency>**

**</dependencies>**

**<configuration>**

**<jdbc>**

**<driver>**org.h2.Driver**</driver>**

**<url>**jdbc:h2:~/yourdatabase**</url>**

**</jdbc>**

**<generator>**

...

**</generator>**

**</configuration>**

**</plugin>**

### 使用 DSLContext

jOOQ提供的流畅API是通过org.jooq.DSLContext接口启动的。 Spring Boot会自动配置一个DSLContext作为Spring Bean并将其连接到您的应用程序DataSource。 要使用DSLContext，你可以@Autowire它：

@Component

**public class** JooqExample **implements** CommandLineRunner {

**private final** DSLContext create;

@Autowired

**public** JooqExample(DSLContext dslContext) {

**this**.create = dslContext;

}

}

**Tip**

jOOQ手册倾向于使用一个名为create的变量来保存DSLContext，我们在这个例子中也做了同样的事情。

然后，您可以使用DSLContext来构建您的查询：

**public** List<GregorianCalendar> authorsBornAfter1980() {

**return this**.create.selectFrom(AUTHOR)

.where(AUTHOR.DATE\_OF\_BIRTH.greaterThan(**new** GregorianCalendar(1980, 0, 1)))

.fetch(AUTHOR.DATE\_OF\_BIRTH);

}

### 自定义JOOQ(Customizing Jooq)

你可以通过设置spring.jooq.sql-dialect来自定义jOOQ使用的SQL方言

application.properties。 例如，要指定Postgres，您可以添加：

**spring.jooq.sql-dialect**=Postgres

更高级的自定义可以通过定义自己的@Bean定义来实现，这将在创建jOOQ配置时使用。 您可以为以下jOOQ类型定义bean：

* ConnectionProvider
* TransactionProvider
* RecordMapperProvider
* RecordListenerProvider
* ExecuteListenerProvider
* VisitListenerProvider

如果要完全控制jOOQ配置，也可以创建自己的org.jooq.Configuration @Bean。

# 使用 NoSQL 技术

Spring Data提供的其他项目可以帮助您访问各种NoSQL技术，包括[MongoDB](http://projects.spring.io/spring-data-mongodb/), [Neo4J](http://projects.spring.io/spring-data-neo4j/), [Elasticsearch](https://github.com/spring-projects/spring-data-elasticsearch/), [Solr](http://projects.spring.io/spring-data-solr/), [Redis](http://projects.spring.io/spring-data-redis/), [Gemfire](http://projects.spring.io/spring-data-gemfire/), [Cassandra](http://projects.spring.io/spring-data-cassandra/), [Couchbase](http://projects.spring.io/spring-data-couchbase/) 和 [LDAP](http://projects.spring.io/spring-data-ldap/)。 Spring Boot提供Redis，MongoDB，Neo4j，Elasticsearch，Solr Cassandra，Couchbase和LDAP的自动配置; 你可以使用其他项目，但是你需要自己配置它们。 请参阅[projects.spring.io/spring-data](http://projects.spring.io/spring-data).上的相应参考文档。

## Redis

[Redis](http://redis.io/) 是一个缓存，消息代理和功能丰富的键值存储。 Spring Boot为Jedis客户端库和[Spring Data Redis](https://github.com/spring-projects/spring-data-redis)提供的抽象类提供基本的自动配置。 有一个spring-boot-starter-data-redis“Starter”，用于方便地收集依赖关系。

### 连接到 Redis

您可以像其他Spring Bean一样注入自动配置的RedisConnectionFactory，StringRedisTemplate或者vanilla RedisTemplate实例。 默认情况下，实例将尝试使用localhost连接到Redis服务器：6379：

@Component

**public class** MyBean {

**private** StringRedisTemplate template; @Autowired

**public** MyBean(StringRedisTemplate template) {

**this**.template = template;

}

*// ...*

}

如果你添加了自己的@Bean自动配置的类型，它将会替换默认的（除了RedisTemplate的情况，排除是基于bean名称'redisTemplate'而不是它的类型的）。 如果commons-pool2在类路径上，默认情况下你会得到一个池连接工厂。

## MongoDB

[MongoDB](http://www.mongodb.com/)是一个开源NoSQL文档数据库，它使用类似JSON的模式而不是传统的基于表格的关系数据。 Spring Boot为使用MongoDB提供了一些便利，包括spring-boot-starter-data-mongodb“Starter”。

### 连接到 MongoDB database

您可以注入一个自动配置的org.springframework.data.mongodb.MongoDbFactory来访问Mongo数据库。 默认情况下，实例将尝试使用URL mongodb：// localhost / test连接到MongoDB服务器：

**import** org.springframework.data.mongodb.MongoDbFactory;

**import** com.mongodb.DB;

@Component

**public class** MyBean {

**private final** MongoDbFactory mongo; @Autowired

**public** MyBean(MongoDbFactory mongo) {

**this**.mongo = mongo;

}

*// ...*

**public void** example() {

DB db = mongo.getDb();

*// ...*

}

}

您可以设置spring.data.mongodb.uri属性来更改URL并配置其他设置，如副本集：

**spring.data.mongodb.uri**=mongodb://user:secret@mongo1.example.com:12345,mongo2.example.com:23456/test

另外，只要你使用Mongo 2.x，指定一个主机/端口。 例如，您可以在application.properties中声明以下内容：

**spring.data.mongodb.host**=mongoserver **spring.data.mongodb.port**=27017

**Note**

如果您使用的是Mongo 3.0 Java驱动程序，则不支持spring.data.mongodb.host和spring.data.mongodb.port。 在这种情况下，应该使用spring.data.mongodb.uri来提供所有的配置。

**Tip**

如果未指定spring.data.mongodb.port，则使用默认值27017。 你可以简单地从上面的示例中删除这一行。

**Tip**

如果你不使用Spring Data Mongo，你可以注入com.mongodb.Mongo bean而不是使用MongoDbFactory。

如果要完全控制建立MongoDB连接，还可以声明自己的MongoDbFactory或Mongo Bean。

### MongoTemplate

Spring Data Mongo提供了一个[MongoTemplate](http://docs.spring.io/spring-data/mongodb/docs/current/api/org/springframework/data/mongodb/core/MongoTemplate.html) 类，它的设计与Spring非常类似

JdbcTemplate的。 和JdbcTemplate一样，Spring Boot会自动配置一个bean来简单地注入：

**import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.data.mongodb.core.MongoTemplate; **import** org.springframework.stereotype.Component;

@Component

**public class** MyBean {

**private final** MongoTemplate mongoTemplate;

@Autowired

**public** MyBean(MongoTemplate mongoTemplate) {

**this**.mongoTemplate = mongoTemplate;

}

*// ...*

}

有关完整的详细信息，请参阅MongoOperations Javadoc。

### Spring Data MongoDB repositories

Spring Data包含对MongoDB的存储库支持。 和前面讨论的JPA库一样，基本的原则是查询是基于方法名自动为你构建的。

实际上，Spring Data JPA和Spring Data MongoDB共享相同的通用基础结构; 所以你可以从之前的JPA例子中，假设City现在是一个Mongo数据类而不是JPA @Entity，它将以相同的方式工作。

**package** com.example.myapp.domain;

**import** org.springframework.data.domain.\*;

**import** org.springframework.data.repository.\*;

**public interface** CityRepository **extends** Repository<City, Long> { Page<City> findAll(Pageable pageable);

City findByNameAndCountryAllIgnoringCase(String name, String country);

}

**Tip**

有关Spring Data MongoDB的完整详细信息，包括丰富的对象映射技术，请参阅[reference documentation](http://projects.spring.io/spring-data-mongodb/)。

### 嵌入式(Embedded) Mongo

Spring Boot为[Embedded Mongo](https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo)提供自动配置。 要在Spring Boot应用程序中使用它，请添加对de.flapdoodle.embed：de.flapdoodle.embed.mongo的依赖关系。

Mongo将监听的端口可以使用spring.data.mongodb.port属性进行配置。 要使用随机分配的空闲端口，请使用零值。 由MongoAutoConfiguration创建的MongoClient将被自动配置为使用随机分配的端口。

**Note**

如果您不配置自定义端口，默认情况下嵌入式支持将使用随机端口（而不是27017）。

如果在类路径中有SLF4J，Mongo产生的输出将自动路由到名为org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongo的记录器。

## Neo4j

[Neo4j](http://neo4j.com/)是一个开放源代码的NoSQL图形数据库，它使用由一级关系相关的丰富的节点数据模型，比传统的rdbms方法更适合于连接大数据。 Spring Boot为使用Neo4j提供了一些便利，包括spring-boot-starter-data-neo4j“Starter”。

### 连接到 Neo4j database

你可以注入一个自动配置的Neo4jSession，Session或者Neo4jOperations实例，就像其他的Spring Bean一样。 默认情况下，实例将尝试使用localhost：7474连接到Neo4j服务器：

@Component

**public class** MyBean {

**private final** Neo4jTemplate neo4jTemplate; @Autowired

**public** MyBean(Neo4jTemplate neo4jTemplate) {

**this**.neo4jTemplate = neo4jTemplate;

}

*// ...*

}

您可以通过添加您自己的org.neo4j.ogm.config.Configuration @Bean来完全控制配置。 另外，添加一个类型为Neo4jOperations的@Bean会禁用自动配置。

您可以通过spring.data.neo4j.\* 属性配置用户和证书使用：

**spring.data.neo4j.uri**=http://my-server:7474 **spring.data.neo4j.username**=neo4j **spring.data.neo4j.password**=secret

### 使用嵌入式模式( embedded mode)

如果将org.neo4j：neo4j-ogm-embedded-driver添加到应用程序的依赖项中，Spring Boot将自动配置Neo4j的进程内嵌入实例，在应用程序关闭时不会保留任何数据。 您可以使用spring.data.neo4j.embedded.enabled = false显式禁用该模式。 您也可以为嵌入模式启用持久性：

spring.data.neo4j.uri=file://var/tmp/graph.db

**Note**

Neo4j OGM嵌入式驱动程序不提供Neo4j内核。 用户需要手动提供这种依赖关系，请参阅文档以获取更多详细信息[the documentation](http://neo4j.com/docs/ogm-manual/current/reference/#reference%3Agetting-started) 。

### Neo4jSession

默认情况下，如果您正在运行Web应用程序，则会话将被绑定到线程以进行整个请求处理（即“在会话中打开会话”模式）。 如果您不想要这种行为，请将以下内容添加到application.properties中：

spring.data.neo4j.open-in-view=false

### Spring Data Neo4j repositories

Spring数据包括Neo4j的存储库支持。

实际上，Spring Data JPA和Spring Data Neo4j共享相同的通用基础结构; 所以你可以从之前的JPA例子中，假设City现在是一个Neo4j OGM @NodeEntity而不是JPA @Entity，它将以相同的方式工作。

**Tip**

您可以使用@EntityScan注解自定义实体扫描位置。

要启用存储库支持（并可选择支持@Transactional），请将以下两个注释添加到Spring配置中：

@EnableNeo4jRepositories(basePackages = "com.example.myapp.repository") @EnableTransactionManagement

### Repository example

**package** com.example.myapp.domain;

**import** org.springframework.data.domain.\*;

**import** org.springframework.data.repository.\*;

**public interface** CityRepository **extends** GraphRepository<City> { Page<City> findAll(Pageable pageable);

City findByNameAndCountry(String name, String country);

}

**Tip**

有关Spring Data Neo4j的完整详细信息，包括丰富的对象映射技术，请参阅[reference documentation](http://projects.spring.io/spring-data-neo4j/)。

## Gemfire

[Spring Data Gemfire](https://github.com/spring-projects/spring-data-gemfire) 为访问[Pivotal Gemfire](http://pivotal.io/big-data/pivotal-gemfire#details) 数据管理平台提供了方便Spring的工具。 有一个spring-boot-starter-data-gemfire“Starter”以方便的方式收集依赖关系。 目前没有Gemfire的自动配置支持，但是您可以使用[single annotation (@EnableGemfireRepositories)](https://github.com/spring-projects/spring-data-gemfire/blob/master/src/main/java/org/springframework/data/gemfire/repository/config/EnableGemfireRepositories.java)来启用Spring Data Repositories。

## Solr

[Apache Solr](http://lucene.apache.org/solr/) 是一个搜索引擎。 Spring Boot为Solr 5客户端库和[Spring Data Solr](https://github.com/spring-projects/spring-data-solr)提供的抽象类提供基本的自动配置。 有一个spring-boot-starter-data-solr'Starter'用于以一种方便的方式收集依赖关系。

### 连接到 Solr

您可以像其他任何Spring bean一样注入自动配置的SolrClient实例。 默认情况下，实例将尝试使用localhost:8983/solr:

@Component

**public class** MyBean {

**private** SolrClient solr; @Autowired

**public** MyBean(SolrClient solr) {

**this**.solr = solr;

}

*// ...*

}

如果你添加一个你自己的SolrClient类型的@Bean，它将会替换默认值。

### Spring Data Solr repositories

Spring Data包含Apache Solr的存储库支持。 和前面讨论的JPA库一样，基本的原则是查询是基于方法名自动为你构建的。

实际上，Spring Data JPA和Spring Data Solr共享相同的通用基础结构; 所以你可以从之前的JPA例子中，假设City现在是@SolrDocument类而不是JPA @Entity，它将以相同的方式工作。

**Tip**

有关Spring Data Solr的完整详细信息，请参阅其参考文档[reference documentation](http://projects.spring.io/spring-data-solr/)。

## Elasticsearch

[Elasticsearch](http://www.elasticsearch.org/) 是一个开源的，分布式的实时搜索和分析引擎。 Spring Boot为弹性数据Elasticsearch提供的Elasticsearch和抽象类提供了基本的自动配置[Spring Data](https://github.com/spring-projects/spring-data-elasticsearch) [Elasticsearch](https://github.com/spring-projects/spring-data-elasticsearch)。 有一个spring-boot-starter-data-elasticsearch“Starter”用于以一种方便的方式收集依赖关系。 Spring Boot也支持[Jest](https://github.com/searchbox-io/Jest)。

### 使用Jest连接到 Elasticsearch

如果你在类路径上有Jest，你可以注入一个自动配置的JestClient目标

默认为 localhost:9200。 您可以进一步调整客户端的配置方式：

**spring.elasticsearch.jest.uris**=http://search.example.com:9200 **spring.elasticsearch.jest.read-timeout**=10000 **spring.elasticsearch.jest.username**=user **spring.elasticsearch.jest.password**=secret

您还可以注册任意数量的实现HttpClientConfigBuilderCustomizer的bean，以实现更高级的自定义。 下面的例子调整额外的HTTP设置：

**static class** HttpSettingsCustomizer **implements** HttpClientConfigBuilderCustomizer {

@Override

**public void** customize(HttpClientConfig.Builder builder) { builder.maxTotalConnection(100).defaultMaxTotalConnectionPerRoute(5);

}

}

要完全控制注册，请定义一个JestClient bean。

### 使用 Spring Data 连接到 Elasticsearch

您可以注入自动配置的ElasticsearchTemplate或Elasticsearch Client实例，就像您使用其他Spring Bean一样。 默认情况下，实例将嵌入一个本地内存服务器（Elasticsearch术语中的一个节点），并使用当前工作目录作为服务器的主目录。 在这个设置中，首先要做的是告诉Elasticsearch在哪里存储文件：

**spring.data.elasticsearch.properties.path.home**=/foo/bar

或者，您可以通过设置切换到远程服务器（即TransportClient）

将spring.data.elasticsearch.cluster-nodes复制到冒号分隔(comma-separated)的“host：port”列表中。

**spring.data.elasticsearch.cluster-nodes**=localhost:9300

@Component

**public class** MyBean {

**private** ElasticsearchTemplate template; @Autowired

**public** MyBean(ElasticsearchTemplate template) {

**this**.template = template;

}

*// ...*

}

如果你添加一个你自己的ElasticsearchTemplate类型的@Bean，它将取代默认值。

### Spring Data Elasticsearch repositories

Spring Data包含Elasticsearch的存储库支持。 和前面讨论的JPA库一样，基本的原则是查询是基于方法名自动为你构建的。

实际上，Spring Data JPA和Spring Data Elasticsearch共享相同的通用基础结构; 所以你可以从之前的JPA例子中，假设City现在是Elasticsearch @Document类而不是JPA @Entity，它将以相同的方式工作。

**Tip**

有关Spring Data Elasticsearch的完整详细信息，请参阅其参考文档[reference documentation](http://docs.spring.io/spring-data/elasticsearch/docs/)。

## Cassandra

[Cassandra](http://cassandra.apache.org/)是一个开源的分布式数据库管理系统，用于处理大量商品服务器上的大量数据。 Spring Boot提供了Cassandra的自动配置和[Spring Data Cassandra](https://github.com/spring-projects/spring-data-cassandra)提供的抽象。 有一个spring-boot-starter-data-cassandra“Starter”用于以一种方便的方式收集依赖关系

### 连接到 Cassandra

您可以像使用其他Spring Bean一样注入自动配置的CassandraTemplate或Cassandra Session实例。 spring.data.cassandra.\* 属性可用于定制连接。 一般来说，您将提供keyspace-name 和 contact-points属性：

**spring.data.cassandra.keyspace-name**=mykeyspace **spring.data.cassandra.contact-points**=cassandrahost1,cassandrahost2

@Component

**public class** MyBean {

**private** CassandraTemplate template; @Autowired

**public** MyBean(CassandraTemplate template) {

**this**.template = template;

}

*// ...*

}

如果你添加一个类型为CassandraTemplate的@Bean，它将会替换默认的。

### Spring Data Cassandra repositories

Spring Data包含对Cassandra的基本存储库支持。 目前这比前面讨论的JPA库更有限，并且需要使用@Query注释finder方法。

**Tip**

有关Spring Data Cassandra的完整详细信息，请参阅其参考文档[reference documentation](http://docs.spring.io/spring-data/cassandra/docs/)。

## Couchbase

[Couchbase](http://www.couchbase.com/)是一个开源的分布式多模型NoSQL面向文档的数据库，针对交互式应用进行了优化。 Spring Boot提供了Couchbase的自动配置和[Spring Data Couchbase](https://github.com/spring-projects/spring-data-couchbase)提供的抽象。 有一个spring-boot-starter-data-couchbase“Starter”，用于以一种方便的方式收集依赖关系

### 连接到 Couchbase

通过添加Couchbase SDK和一些配置，您可以非常轻松地获取Bucket和Cluster。 spring.couchbase.\*属性可用于定制连接。 一般你会提供引导主机，存储bucket名称和密码：

**spring.couchbase.bootstrap-hosts**=my-host-1,192.168.1.123 **spring.couchbase.bucket.name**=my-bucket **spring.couchbase.bucket.password**=secret

**Tip**

您至少需要提供引导主机，在这种情况下，存储区名称是默认的，密码是空字符串。 或者，你可以定义你自己的org.springframework.data.couchbase.config.CouchbaseConfigurer @Bean控制整个配置。

还可以自定义一些CouchbaseEnvironment设置。 例如，以下配置更改用于打开新Bucket并启用SSL支持的超时：

**spring.couchbase.env.timeouts.connect**=3000 **spring.couchbase.env.ssl.key-store**=/location/of/keystore.jks **spring.couchbase.env.ssl.key-store-password**=secret

检查spring.couchbase.env.\*属性以获取更多详细信息。

### Spring Data Couchbase repositories

Spring Data包含Couchbase的存储库支持。 有关Spring Data Couchbase的完整详细信息，请参阅其参考文档[reference documentation](http://docs.spring.io/spring-data/couchbase/docs/current/reference/html/)。

只要默认的CouchbaseConfigurer可用（如上所述启用了couchbase支持时发生的情况），您可以像使用其他Spring Bean一样注入一个自动配置的CouchbaseTemplate实例。

@Component

**public class** MyBean {

**private final** CouchbaseTemplate template; @Autowired

**public** MyBean(CouchbaseTemplate template) {

**this**.template = template;

}

*// ...*

}

您可以在自己的配置中定义几个bean来覆盖自动配置提供的bean：

* 名为couchbaseTemplate的CouchbaseTemplate @Bean
* 名为couchbaseIndexManager的IndexManager @ Bean
* 名为couchbaseCustomConversions的CustomConversions @Bean

为了避免在自己的配置中对这些名称进行硬编码(hard-coding)，可以重用Spring Data Couchbase提供的BeanNames。 例如，您可以自定义转换器使用如下：

@Configuration

**public class** SomeConfiguration {

@Bean(BeanNames.COUCHBASE\_CUSTOM\_CONVERSIONS)

**public** CustomConversions myCustomConversions() {

**return new** CustomConversions(...);

}

*// ...*

}

**Tip**

如果您想完全绕过Spring Data Couchbase的自动配置，请提供您自己的org.springframework.data.couchbase.config.AbstractCouchbaseDataConfiguration实现。

## LDAP

[LDAP](https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol) （轻量级目录访问协议）是一种开放的，厂商中立的行业标准应用协议，用于通过IP网络访问和维护分布式目录信息服务。 Spring Boot为任何兼容的LDAP服务器提供自动配置，并支持来自[UnboundID](https://www.ldap.com/unboundid-ldap-sdk-for-java)的嵌入式内存LDAP服务器

[Spring Data LDAP](https://github.com/spring-projects/spring-data-ldap)提供LDAP抽象。 有一个spring-boot-starter-data-ldap“Starter”，用于方便地收集依赖关系。

### 连接到 LDAP server

要连接到LDAP服务器，请确保您声明了对spring-boot-starter-data-ldap“Starter”或spring-ldap-core的依赖关系，然后在application.properties中声明您的服务器的URL：

**spring.ldap.urls**=ldap://myserver:1235 **spring.ldap.username**=admin **spring.ldap.password**=secret

如果你需要自定义连接设置，你可以使用spring.ldap.base和spring.ldap.base-environment属性。

### Spring Data LDAP repositories

Spring Data包含LDAP的存储库支持。 有关Spring Data LDAP的完整详细信息，请参阅其参考文档。

Spring Data includes repository support for LDAP. For complete details of Spring Data LDAP, refer to their [reference documentation](http://docs.spring.io/spring-data/ldap/docs/1.0.x/reference/html/).

您也可以像使用其他Spring Bean一样注入自动配置的LdapTemplate实例。

@Component

**public class** MyBean {

**private final** LdapTemplate template; @Autowired

**public** MyBean(LdapTemplate template) {

**this**.template = template;

}

*// ...*

}

### 嵌入式 in-memory LDAP server

出于测试目的，Spring Boot支持从[UnboundID](https://www.ldap.com/unboundid-ldap-sdk-for-java)自动配置内存中的LDAP服务器。 要配置服务器，请向com.unboundid：unboundid-ldapsdk添加一个依赖项，并声明一个base-dn属性：

**spring.ldap.embedded.base-dn**=dc=spring,dc=io

默认情况下，服务器将在随机端口上启动，并触发常规的LDAP支持（不需要指定spring.ldap.urls属性）。

如果你的类路径中有一个schema.ldif文件，它将被用来初始化服务器。 如果要从其他资源加载初始化脚本，也可以使用spring.ldap.embedded.ldif属性。

默认情况下，将使用标准架构来验证LDIF文件，您可以使用spring.ldap.embedded.validation.enabled属性完全关闭验证。 如果您有自定义属性，则可以使用spring.ldap.embedded.validation.schema来定义自定义属性类型或对象类。

# Caching

Spring框架提供了对应用程序透明地添加缓存的支持。 其核心是抽象将缓存应用于方法，从而根据缓存中可用的信息减少执行次数。 缓存逻辑是透明应用的，对调用者没有任何干扰。 只要通过@EnableCaching批注启用缓存支持，Spring Boot就会自动配置缓存基础结构。

**Note**

查看Spring Framework参考的相关部分[relevant section](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#cache) 以获取更多详细信息。

简而言之，将缓存添加到服务的操作中，就像将相关的注释添加到其方法一样简单：

**import** org.springframework.cache.annotation.Cacheable

**import** org.springframework.stereotype.Component;

@Component

**public class** MathService {

@Cacheable("piDecimals")

**public int** computePiDecimal(**int** i) {

*// ...*

}

}

这个例子演示了如何在一个潜在的代价高昂的操作上使用缓存。 在调用computePiDecimal之前，抽象将在匹配 i 参数的piDecimals缓存中查找条目。 如果找到一个条目，则缓存中的内容立即返回给调用者，并且该方法不被调用。 否则，在返回值之前调用该方法并更新缓存。

**Note**

您也可以透明地使用标准的JSR-107（JCache）注释（例如@CacheResult）。 我们强烈建议您不要混淆和匹配。

如果你不添加任何特定的缓存库，Spring Boot会自动配([Simple provider](#_bookmark226) )一个在内存中使用并发映射的简单提供程序。 当需要缓存的时候（例如上面的例子中的piDecimals），这个提供者将会为你创建它。 简单的提供者并不是真正被推荐用于生产用途，但是对于入门和确保理解这些特性来说是非常好的。 当您决定使用缓存提供程序时，请确保阅读其文档以了解如何配置应用程序使用的缓存。 实际上，所有提供程序都要求您显式配置在应用程序中使用的每个缓存。 有些提供了一种自定义由spring.cache.cache-names属性定义的默认缓存的方法。

**Tip**

透明地[update](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#cache-annotations-put) 或 [evict](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#cache-annotations-evict) 缓存中的数据也是可能的。

**Note**

如果您正在使用非基于接口的Bean的缓存基础结构，请确保启用@EnableCaching的proxyTargetClass属性。

## Supported cache providers

缓存抽象不提供实际的存储，并依赖于由org.springframework.cache.Cache和org.springframework.cache.CacheManager接口实现的抽象。

如果您尚未定义类型为CacheManager的Bean或名为cacheResolver的CacheResolver

（请参阅CachingConfigurer），Spring Boot尝试检测以下提供程序（按此顺序）：

* [Generic](#_bookmark217)
* [JCache (JSR-107)](#_bookmark218) (EhCache 3, Hazelcast, Infinispan, etc)
* [EhCache 2.x](#_bookmark219)
* [Hazelcast](#_bookmark220)
* [Infinispan](#_bookmark221)
* [Couchbase](#_bookmark222)
* [Redis](#_bookmark223)
* [Caffeine](#_bookmark224)
* [Guava](#_bookmark225) (deprecated)
* [Simple](#_bookmark226)

**Tip**

也可以通过spring.cache.type属性来强制缓存提供者使用。 如果您需要在特定环境（例如测试）中完全禁用缓存[disable caching altogether](#_bookmark227) ，请使用此属性。

**Tip**

使用spring-boot-starter-cache“Starter”快速添加基本的缓存依赖关系。 初学者引入了spring-context-support：如果你手动添加依赖项，你必须包含spring-context-support才能使用JCache，EhCache 2.x或者Guava支持。

如果CacheManager是由Spring Boot自动配置的，那么可以通过公开一个实现CacheManagerCustomizer接口的bean来完全初始化它的配置。 以下设置一个标志来表示空值应该传递给底层的地图(map)。

@Bean

**public** CacheManagerCustomizer<ConcurrentMapCacheManager> cacheManagerCustomizer() {

**return new** CacheManagerCustomizer<ConcurrentMapCacheManager>() { @Override

**public void** customize(ConcurrentMapCacheManager cacheManager) { cacheManager.setAllowNullValues(false);

}

};

}

**Note**

在上面的例子中，需要一个自动配置的ConcurrentMapCacheManager。 如果不是这种情况（要么提供自己的配置，要么自动配置不同的缓存提供程序），定制程序根本不会被调用。 您可以拥有任意数量的定制器，您也可以像往常一样使用@Order 或 Ordered来定购它们。

### Generic

如果上下文定义了至少一个org.springframework.cache.Cache，则使用通用cache bean。

包含所有类型的Bean的CacheManager被创建。

### JCache (JSR-107)

通过类路径上的javax.cache.spi.CachingProvider（即符合JSR-107的高速缓存库）和由spring-boot-starter-cache“Starter”提供的JCacheCacheManager来引导JCache。 在那里有各种兼容的库，Spring Boot提供Ehcache 3，Hazelcast和Infinispan的依赖管理。 任何其他兼容的库也可以添加。

可能会出现多个供应商存在的情况，在这种情况下，供应商必须明确指定。 即使JSR-107标准没有强制规定配置文件的位置，Spring Boot也会尽力满足实现细节。

*# Only necessary if more than one provider is present* **spring.cache.jcache.provider**=com.acme.MyCachingProvider **spring.cache.jcache.config**=classpath:acme.xml

**Note**

由于缓存库可能提供本地实现和JSR-107支持，因此Spring Boot将更喜欢JSR-107支持，因此如果切换到不同的JSR-107实现，则可以使用相同的功能。

**Tip**

Spring Boot对Hazelcast有广泛的支持[general support for Hazelcast](#_bookmark254)。 如果有一个HazelcastInstance可用，那么它将自动重用于CacheManager，除非指定了spring.cache.jcache.config属性。

有几种方法可以自定义底层的javax.cache.cacheManager：

* 可以通过spring.cache.cache-names属性在启动时创建缓存。 如果自定义定义了javax.cache.configuration.Configuration bean，用它来定制它们。
* org.springframework.boot.autoconfigure.cache.JCacheManagerCustomizer bean将通过CacheManager的引用进行调用，以实现完全自定义。

**Tip**

如果定义了一个标准的javax.cache.CacheManager bean，它将自动包装在抽象所期望的org.springframework.cache.CacheManager实现中。 没有进一步的定制应用于此。

### EhCache 2.x

如果可以在类路径的根目录中找到名为ehcache.xml的文件，则使用EhCache 2.x。 如果EhCache 2.x，由spring-boot-starter-cache“Starter”提供的EhCacheCacheManager和这样的文件存在，它将被用来引导缓存管理器。 备用配置文件也可以使用：

**spring.cache.ehcache.config**=classpath:config/another-config.xml

### Hazelcast

[general support for Hazelcast](#_bookmark254)有广泛的支持。 如果HazelcastInstance已被自动配置，它会自动包装在CacheManager中。

### Infinispan

Infinispan没有默认配置文件位置，因此必须明确指定（或使用默认引导程序）。

**spring.cache.infinispan.config**=infinispan.xml

可以通过spring.cache.cache-names属性在启动时创建缓存。 如果自定义

定义了ConfigurationBuilder bean，用它来定制它们。

**Note**

Infinispan在Spring Boot中的支持仅限于嵌入式模式，非常基础。 如果您需要更多的选项，则应该使用官方的Infinispan Spring Boot启动器，请查阅文档the documentation以获取更多详细信息。

### Couchbase

如果Couchbase Java客户端和couchbase-spring-cache实现可用，并且Couchbase已配置[configured](#_bookmark208)，则将自动配置CouchbaseCacheManager。 也可以使用spring.cache.cache-names属性在启动时创建额外的缓存。 这些将在自动配置的Bucket上运行。 您还可以使用定制器在另一个存储Bucket上创建更多的高速缓存：假设您需要“主”存储Bucket（foo和bar）上的两个高速缓存以及一个定制时间在另一个存储Bucket上存活2秒的高速缓存。 首先，您可以简单地通过配置创建两个第一个缓存：

**spring.cache.cache-names**=foo,bar

然后定义这个额外的@Configuration来配置额外的Bucket和biz缓存：

@Configuration

**public class** CouchbaseCacheConfiguration {

**private final** Cluster cluster;

**public** CouchbaseCacheConfiguration(Cluster cluster) {

**this**.cluster = cluster;

}

@Bean

**public** Bucket anotherBucket() {

**return this**.cluster.openBucket(***"another"***, ***"secret"***);

}

@Bean

**public** CacheManagerCustomizer<CouchbaseCacheManager> cacheManagerCustomizer() {

**return** c -> {

c.prepareCache(***"biz"***, CacheBuilder.newInstance(anotherBucket())

.withExpiration(2));

};

}

}

此示例配置重用通过自动配置创建的群集。

### Redis

如果Redis可用且已配置，则RedisCacheManager将自动配置。 也可以使用spring.cache.cache-names属性在启动时创建额外的缓存。

**Note**

默认情况下，添加一个关键字前缀，以防止如果两个单独的缓存使用相同的密钥，Redis将有重叠的键，并可能返回无效的值。 如果您创建自己的RedisCacheManager，我们强烈建议您保持启用此设置。

### Caffeine

Caffeine是Guava缓存的Java 8重写，将取代Spring Boot 2.0中的Guava支持。 如果存在Caffeine，CaffeineCacheManager（由spring-boot-starter-cache'Starter'提供）将自动配置。 可以使用spring.cache.cachename属性在启动时创建缓存，并按以下顺序（按此顺序）进行自定义：

1. 由spring.cache.caafeine.spec定义的缓存规范
2. 定义了一个com.github.benmanes.caffeine.cache.CaffeineSpec bean
3. 定义了com.github.benmanes.cafeine.cache.Caffeine bean

例如，以下配置会创建一个最大大小为500的foo和bar高速缓存，并且生存时间为10分钟

**spring.cache.cache-names**=foo,bar **spring.cache.caffeine.spec**=maximumSize=500,expireAfterAccess=600s

另外，如果定义了一个com.github.benmanes.caffeine.cache.CacheLoader bean，它会自动关联到CaffeineCacheManager。 由于CacheLoader将与缓存管理器管理的所有缓存关联，因此必须将其定义为CacheLoader <Object，Object>。 任何其他泛型类型将被自动配置忽略。

### Guava (deprecated)

如果存在Guava，GuavaCacheManager将自动配置。 可以使用spring.cache.cache-names属性在启动时创建缓存，并按以下顺序（按此顺序）进行自定义：

1. 由spring.cache.guava.spec定义的缓存规范
2. 定义了com.google.common.cache.CacheBuilderSpec bean
3. 定义com.google.common.cache.CacheBuilder bean

例如，以下配置会创建一个最大大小为500的foo和bar高速缓存，并且生存时间为10分钟

**spring.cache.cache-names**=foo,bar **spring.cache.guava.spec**=maximumSize=500,expireAfterAccess=600s

另外，如果定义了com.google.common.cache.CacheLoader bean，它将自动关联到GuavaCacheManager。 由于CacheLoader将与缓存管理器管理的所有缓存关联，因此必须将其定义为CacheLoader <Object，Object>。 任何其他泛型类型将被自动配置忽略。

### Simple

如果找不到任何其他提供者，则配置使用ConcurrentHashMap作为缓存存储的简单实现。 如果应用程序中没有缓存库，则这是默认值。 高速缓存默认是即时创建的，但您可以使用cache-names属性来限制可用高速缓存的列表。 例如，如果你只想要foo和bar高速缓存：

**spring.cache.cache-names**=foo,bar

如果你这样做，并且你的应用程序使用未列出的缓存，那么在需要缓存时运行时会失败，但在启动时不会运行。 这与“real”缓存提供程序在使用未声明的缓存时的行为方式类似。

### None

当@EnableCaching出现在您的配置中时，也会有一个合适的缓存配置。 如果您需要在某些环境中完全禁用缓存，则强制缓存类型为none，以使用无操作的实现：

**spring.cache.type**=none

# Messaging

Spring框架为消息系统集成提供了广泛的支持：从使用JmsTemplate的JMS API的简化使用到异步接收消息的完整基础结构。 Spring AMQP为“高级消息队列协议”提供了类似的功能，Spring Boot也为RabbitTemplate和RabbitMQ提供了自动配置选项。 在Spring WebSocket中也支持STOMP消息本身，Spring Boot通过启动器和少量的自动配置支持。 Spring Boot也支持Apache Kafka。

## JMS

javax.jms.ConnectionFactory接口提供了一个创建javax.jms.Connection的标准方法，用于与JMS代理进行交互。 尽管Spring需要一个ConnectionFactory来处理JMS，但通常不需要直接使用它，而是可以依赖更高级别的消息抽象（请参阅Spring Framework参考文档[relevant section](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#jms) 的相关章节以获取详细信息）。 Spring Boot还会自动配置发送和接收消息的必要基础结构。

### ActiveMQ support

当Spring Boot检测到ActiveMQ在类路径中可用时，也可以配置ConnectionFactory。 如果代理存在，则会自动启动并配置嵌入式代理（只要未通过配置指定代理URL）。

**Note**

如果您使用的是spring-boot-starter-activemq，则提供必要的依赖关系来连接或嵌入ActiveMQ实例，以及Spring基础结构以与JMS集成。

ActiveMQ配置由spring.activemq.\*中的外部配置属性控制。 例如，您可以在application.properties中声明以下部分：

**spring.activemq.broker-url**=tcp://192.168.1.210:9876 **spring.activemq.user**=admin **spring.activemq.password**=secret

您还可以通过向org.apache.activemq：activemq- pool添加依赖项来合并JMS资源，并相应地配置PooledConnectionFactory：

**spring.activemq.pool.enabled**=true **spring.activemq.pool.max-connections**=50

**Tip**

请参阅[ActiveMQProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQProperties.java) 以获取更多受支持的选项。 您还可以注册任意数量的实现ActiveMQConnectionFactoryCustomizer的bean以进行更高级的自定义。

默认情况下，如果ActiveMQ不存在，ActiveMQ将创建一个目标，因此目标将根据其提供的名称进行解析。

### Artemis support

当Spring Boot检测到Artemis在类路径中可用时，它可以自动配置一个ConnectionFactory。 如果代理存在，则会自动启动并配置嵌入式代理（除非已明确设置mode属性）。 支持的模式有：嵌入式（明确规定嵌入式代理是必需的，如果代理在类路径中不可用，则会导致错误），本地代理使用netty传输协议连接到代理。 当配置后者时，Spring Boot会使用默认设置配置一个连接到本地机器上运行的代理的ConnectionFactory。

**Note**

如果您使用的是spring-boot-starter-artemis，则必须提供连接到现有Artemis实例的依赖关系，以及Spring基础结构以与JMS集成。 将org.apache.activemq：artemis-jms-server添加到您的应用程序允许您使用嵌入模式。

Artemis配置由spring.artemis.\*中的外部配置属性控制。 例如，您可以在application.properties中声明以下部分：

**spring.artemis.mode**=native **spring.artemis.host**=192.168.1.210 **spring.artemis.port**=9876 **spring.artemis.user**=admin **spring.artemis.password**=secret

在嵌入代理时，可以选择是否要启用持久性，以及应该使其可用的目标列表。 这些可以被指定为冒号号分隔的列表，以使用默认选项创建它们; 也可以分别为高级队列和主题配置定义org.apache.activemq.artemis.jms.server.config.JMSQueueConfiguration

或org.apache.activemq.artemis.jms.server.config.TopicConfiguration类型的bean。

请参阅[ArtemisProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/artemis/ArtemisProperties.java) 获取更多支持的选项。

根本不涉及任何JNDI查找，并且使用Artemis配置中的“name”属性或通过配置提供的名称来解析目的地的名称。

### 使用 JNDI ConnectionFactory

如果您正在应用程序服务器中运行应用程序，Spring Boot将尝试使用JNDI查找JMS ConnectionFactory。 默认情况下，将检查位置java：/ JmsXA和java：/ XAConnectionFactory。 如果您需要指定替代位置，则可以使用spring.jms.jndi-name属性：

**spring.jms.jndi-name**=java:/MyConnectionFactory

### Sending a message

Spring的JmsTemplate是自动配置的，你可以直接将它自动装入到你自己的bean中：

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

**import** org.springframework.jms.core.JmsTemplate;

**import** org.springframework.stereotype.Component;

@Component

**public class** MyBean {

**private final** JmsTemplate jmsTemplate; @Autowired

**public** MyBean(JmsTemplate jmsTemplate) {

**this**.jmsTemplate = jmsTemplate;

}

*// ...*

}

**Note**

[JmsMessagingTemplate](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/jms/core/JmsMessagingTemplate.html) 可以以类似的方式注入。 如果定义了DestinationResolver或MessageConverter Bean，它们将自动关联到自动配置的JmsTemplate。

### Receiving a message

当存在JMS基础结构时，可以使用@JmsListener对任何bean进行注释以创建侦听器端点。 如果没有定义JmsListenerContainerFactory，则会自动配置一个默认的JmsListenerContainerFactory。 如果定义了DestinationResolver或MessageConverter Bean，它们将自动关联到默认工厂。

默认工厂默认是事务性的。 如果您正在JtaTransactionManager存在的基础架构中运行，则默认情况下它将关联到侦听器容器。 如果不是，sessionTransacted标志将被启用。 在后一种情况下，您可以通过在侦听器方法（或其代理）上添加@Transactional来将本地数据存储事务与传入消息的处理关联起来。 这将确保一旦本地事务完成，传入消息就被确认。 这还包括发送已在相同的JMS会话上执行的响应消息。

以下组件在someQueue目标上创建侦听器端点：

@Component

**public class** MyBean {

@JmsListener(destination = "someQueue")

**public void** processMessage(String content) {

*// ...*

}

}

**Tip**

查看[the Javadoc of @EnableJms](http://docs.spring.io/spring/docs/4.3.12.RELEASE/javadoc-api/org/springframework/jms/annotation/EnableJms.html) 获取更多细节。

如果您需要创建更多的JmsListenerContainerFactory实例，或者您想要覆盖默认值，Spring Boot会提供一个DefaultJmsListenerContainerFactoryConfigurer，您可以使用它来初始化DefaultJmsListenerContainerFactory，其设置与自动配置的设置相同。

例如，下面公开了使用特定MessageConverter的另一个工厂：

@Configuration

**static class** JmsConfiguration {

@Bean

**public** DefaultJmsListenerContainerFactory myFactory( DefaultJmsListenerContainerFactoryConfigurer configurer) {

DefaultJmsListenerContainerFactory factory =

**new** DefaultJmsListenerContainerFactory(); configurer.configure(factory, connectionFactory()); factory.setMessageConverter(myMessageConverter()); **return** factory;

}

}

那么你可以在任何@ JmsListener-annotatated的方法中使用，如下所示：

@Component

**public class** MyBean {

@JmsListener(destination = ***"someQueue"***, **containerFactory="myFactory"**) **public void** processMessage(String content) {

*// ...*

}

}

## AMQP

高级消息队列协议（AMQP）是面向消息中间件的平台中立的有线协议。 Spring AMQP项目将核心Spring概念应用于基于AMQP的消息传递解决方案的开发。 Spring Boot通过RabbitMQ为AMQP提供了一些便利，包括spring-boot-starter-amqp'Starter'。

### RabbitMQ support

RabbitMQ是一个基于AMQP协议的轻量级，可靠，可扩展和可移植的消息代理。 Spring使用RabbitMQ使用AMQP协议进行通信。

RabbitMQ配置由spring.rabbitmq.\* 中的外部配置属性控制。 例如，您可以在application.properties中声明以下部分：

**spring.rabbitmq.host**=localhost **spring.rabbitmq.port**=5672 **spring.rabbitmq.username**=admin **spring.rabbitmq.password**=secret

请参阅[RabbitProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/amqp/RabbitProperties.java) 获取更多支持的选项。

**Tip**

检查了解[Understanding AMQP, the protocol used by RabbitMQ](http://spring.io/blog/2010/06/14/understanding-amqp-the-protocol-used-by-rabbitmq/) 使用的协议了解更多细节。

### Sending a message

Spring的AmqpTemplate和AmqpAdmin是自动配置的，你可以直接将它们自动装入自己的bean中：

**import** org.springframework.amqp.core.AmqpAdmin;

**import** org.springframework.amqp.core.AmqpTemplate;

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

**import** org.springframework.stereotype.Component;

@Component

**public class** MyBean {

**private final** AmqpAdmin amqpAdmin;

**private final** AmqpTemplate amqpTemplate;

@Autowired

**public** MyBean(AmqpAdmin amqpAdmin, AmqpTemplate amqpTemplate) {

**this**.amqpAdmin = amqpAdmin;

**this**.amqpTemplate = amqpTemplate;

}

*// ...*

}

**Note**

[RabbitMessagingTemplate](http://docs.spring.io/spring-amqp/docs/current/api/org/springframework/amqp/rabbit/core/RabbitMessagingTemplate.html) 可以以类似的方式注入。 如果是一个MessageConverter

bean被定义，它会自动关联到自动配置的AmqpTemplate。

任何定义为bean的org.springframework.amqp.core.Queue将自动用于在RabbitMQ实例上声明相应的队列（如有必要）。

您可以在AmqpTemplate上启用重试以重试操作，例如在代理连接丢失的情况下。 重试是默认禁用的。

### Receiving a message

当存在Rabbit基础结构时，可以使用@RabbitListener对任何bean进行注释以创建一个侦听器端点。 如果没有定义RabbitListenerContainerFactory，则会自动配置一个默认的RabbitListenerContainerFactory。 如果定义了MessageConverter或MessageRecoverer bean，它们将自动关联到默认工厂。

以下组件在someQueue队列上创建一个侦听器端点：

@Component

**public class** MyBean {

@RabbitListener(queues = "someQueue")

**public void** processMessage(String content) {

*// ...*

}

}

**Tip**

查看the Javadoc of @EnableRabbit获取更多细节。

如果您需要创建更多的RabbitListenerContainerFactory实例，或者您想要覆盖默认值，Spring Boot会提供一个SimpleRabbitListenerContainerFactoryConfigurer，您可以使用它来初始化一个与自动配置的设置相同的SimpleRabbitListenerContainerFactory。

例如，下面公开了使用特定MessageConverter的另一个工厂：

@Configuration

**static class** RabbitConfiguration {

@Bean

**public** SimpleRabbitListenerContainerFactory myFactory( SimpleRabbitListenerContainerFactoryConfigurer configurer) {

SimpleRabbitListenerContainerFactory factory =

**new** SimpleRabbitListenerContainerFactory(); configurer.configure(factory, connectionFactory); factory.setMessageConverter(myMessageConverter()); **return** factory;

}

}

那么你可以在任何@ RabbitListener注释的方法中使用，如下所示：

@Component

**public class** MyBean {

@RabbitListener(queues = ***"someQueue"***, **containerFactory="myFactory"**) **public void** processMessage(String content) {

*// ...*

}

}

您可以启用重试来处理侦听器引发异常的情况。 默认使用RejectAndDontRequeueRecoverer，但是你可以定义一个你自己的MessageRecoverer。 当重试耗尽时，如果代理配置如此，则该消息将被拒绝并丢弃或路由到死信交换。 重试是默认禁用的。

**Important**

如果重试未启用，并且侦听器引发异常，默认情况下，传递将无限期地重试。 您可以通过两种方式修改此行为; 将defaultRequeueRejected属性设置为false，并尝试零重新发送; 或者抛出一个AmqpRejectAndDontRequeueException异常来表明消息应该被拒绝。 这是启用重试并达到最大传送次数时使用的机制。

## Apache Kafka Support

通过提供spring-kafka项目的自动配置来支持[Apache Kafka](http://kafka.apache.org/) 。

Kafka配置由spring.kafka.\* 中的外部配置属性控制。 例如，您可以在application.properties中声明以下部分：

**spring.kafka.bootstrap-servers**=localhost:9092 **spring.kafka.consumer.group-id**=myGroup

请参阅[KafkaProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/kafka/KafkaProperties.java) 获取更多支持的选项。

### Sending a Message

Spring的KafkaTemplate是自动配置的，你可以直接在你自己的bean中自动装配它们：

@Component

**public class** MyBean {

**private final** KafkaTemplate kafkaTemplate; @Autowired

**public** MyBean(KafkaTemplate kafkaTemplate) {

**this**.kafkaTemplate = kafkaTemplate;

}

*// ...*

}

### Receiving a Message

当存在Apache Kafka基础结构时，任何bean都可以使用@KafkaListener进行注释以创建侦听器端点。 如果没有定义KafkaListenerContainerFactory，则使用spring.kafka.listener.\* 中定义的键自动配置默认的一个。

以下组件在someTopic主题上创建侦听器端点：

@Component

**public class** MyBean {

@KafkaListener(topics = "someTopic")

**public void** processMessage(String content) {

*// ...*

}

}

### Additional Kafka Properties

[Appendix A, *Common application*](#_bookmark569)[*properties*](#_bookmark569)中显示了自动配置支持的属性。 请注意，这些属性（连字符或camelCase）大部分直接映射到Apache Kafka dotted属性，有关详细信息，请参阅Apache Kafka文档。

前面几个属性适用于生产者和消费者，但是如果你想为每个生产者和消费者使用不同的值，可以在生产者或消费者层面指定。 Apache Kafka指定重要属性：HIGH，MEDIUM和LOW。 Spring Boot自动配置支持所有HIGH重要性属性，一些选定的MEDIUM和LOW，以及任何没有默认值的属性。

Kafka支持的属性只有一部分可以通过KafkaProperties类获得。 如果您希望使用不直接支持的其他属性来配置生产者或消费者，请使用以下命令：

**spring.kafka.properties.foo.bar**=baz

这将常见的foo.bar Kafka属性设置为baz。

这些属性将由消费者和生产者工厂bean共享。 如果您希望使用不同的属性自定义这些组件，例如为每个属性使用不同的度量标准读取器，则可以覆盖这些bean定义，如下所示：

@Configuration

**public static class** CustomKafkaBeans {

**/\*\***

* **Customized ProducerFactory bean.**
* **@param properties the kafka properties.**

**\* @return the bean.**

**\*/**

@Bean

**public** ProducerFactory<?, ?> kafkaProducerFactory(KafkaProperties properties) { Map<String, Object> producerProperties = properties.buildProducerProperties(); producerProperties.put(CommonClientConfigs.METRIC\_REPORTER\_CLASSES\_CONFIG,

MyProducerMetricsReporter.**class**);

**return new** DefaultKafkaProducerFactory<Object, Object>(producerProperties);

}

**/\*\***

* **Customized ConsumerFactory bean.**
* **@param properties the kafka properties.**
* **@return the bean.**

**\*/**

@Bean

**public** ConsumerFactory<?, ?> kafkaConsumerFactory(KafkaProperties properties) { Map<String, Object> consumerProperties = properties.buildConsumerProperties(); consumerProperties.put(CommonClientConfigs.METRIC\_REPORTER\_CLASSES\_CONFIG,

MyConsumerMetricsReporter.**class**);

**return new** DefaultKafkaConsumerFactory<Object, Object>(consumerProperties);

}

}

# 调用 REST services

如果您需要从应用程序调用远程REST服务，则可以使用Spring Framework的RestTemplate类。 由于RestTemplate实例经常需要在使用之前进行定制，因此Spring Boot不提供任何自动配置的RestTemplate bean。 但是，它会自动配置一个RestTemplateBuilder，可以在需要时使用RestTemplateBuilder创建RestTemplate实例。 自动配置的RestTemplateBuilder将确保合理的HttpMessageConverters应用于RestTemplate实例。

这是一个典型的例子：

@Service

**public class** MyBean {

**private final** RestTemplate restTemplate;

**public** MyBean(RestTemplateBuilder restTemplateBuilder) {

**this**.restTemplate = restTemplateBuilder.build();

}

**public** Details someRestCall(String name) {

**return this**.restTemplate.getForObject(***"/{name}/details"***, Details.**class**, name);

}

}

**Tip**

RestTemplateBuilder包含许多有用的方法，可以用来快速配置RestTemplate。 例如，要添加BASIC认证支持，您可以使用builder.basicAuthorization（“user”，“password”）。build（）。

## 自定义 RestTemplate

RestTemplate自定义有三种主要方法，具体取决于您希望自定义应用的范围。

为了尽可能缩小任何自定义的范围，请注入自动配置的RestTemplateBuilder，然后根据需要调用其方法。 每个方法调用都会返回一个新的RestTemplateBuilder实例，所以自定义只会影响构建器的这种使用。

要进行应用程序范围application-wide的添加式定制，可以使用RestTemplateCustomizer bean。 所有这些bean都会自动注册到自动配置的RestTemplateBuilder中，并将应用于任何使用它构建的模板。

下面是一个定制器的例子，它为192.168.0.5以外的所有主机配置代理的使用：

**static class** ProxyCustomizer **implements** RestTemplateCustomizer {

@Override

**public void** customize(RestTemplate restTemplate) { HttpHost proxy = **new** HttpHost(***"proxy.example.com"***); HttpClient httpClient = HttpClientBuilder.create()

.setRoutePlanner(**new** DefaultProxyRoutePlanner(proxy) {

@Override

**public** HttpHost determineProxy(HttpHost target, HttpRequest request, HttpContext context)

**throws** HttpException {

**if** (target.getHostName().equals(***"192.168.0.5"***)) {

**return** null;

}

**return super**.determineProxy(target, request, context);

}

}).build(); restTemplate.setRequestFactory(

**new** HttpComponentsClientHttpRequestFactory(httpClient));

}

}

最后，最极端的（也是很少使用的）选项是创建你自己的RestTemplateBuilder bean。 这将关闭RestTemplateBuilder的自动配置，并阻止使用RestTemplateCustomizer bean。

# Validation

只要JSR-303实现（例如Hibernate验证器）位于类路径上，Bean Validation 1.1支持的方法验证功能就会自动启用。 这允许bean方法在其参数和/或返回值上使用javax.validation约束进行注释。 具有这种注释方法的目标类需要在类型级别使用@Validated注解进行注释，以便为其内联约束注释搜索它们的方法。

例如，下面的服务触发第一个参数的验证，确保它的大小在8到10之间

@Service @Validated

**public class** MyBean {

**public** Archive findByCodeAndAuthor(@Size(min = 8, max = 10) String code, Author author) {

...

}

}

# Sending email

Spring框架为使用JavaMailSender发送电子邮件提供了一个简单的抽象

界面和Spring Boot提供自动配置以及启动模块。

**Tip**

检查参考文档[reference documentation](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#mail)以获得如何使用的详细说明

JavaMailSender。

如果spring.mail.host和相关库（由spring-boot-starter-mail定义）可用，则创建默认的JavaMailSender（如果不存在）。 发件人可以通过spring.mail命名空间中的配置项进一步自定义，请参阅[MailProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailProperties.java) 以获取更多详细信息。

特别是，某些默认的超时值是无限的，您可能想要更改该值以避免线程阻塞无响应的邮件服务器：

**spring.mail.properties.mail.smtp.connectiontimeout**=5000 **spring.mail.properties.mail.smtp.timeout**=3000 **spring.mail.properties.mail.smtp.writetimeout**=5000

# 分布式事务 JTA

Spring Boot使用[Atomikos](http://www.atomikos.com/) 或 [Bitronix](https://github.com/bitronix/btm)嵌入式事务管理器支持跨多个XA资源的分布式JTA事务。 在部署到合适的Java EE应用服务器时，也支持JTA事务。

当检测到JTA环境时，Spring的JtaTransactionManager将被用来管理事务。 自动配置的JMS，DataSource和JPA bean将被升级以支持XA事务。 你可以使用标准的Spring成语(idioms)例如@Transactional来参与分布式事务。 如果您处于JTA环境中并且仍想使用本地事务，则可以将spring.jta.enabled属性设置为false以禁用JTA自动配置。

## 使用 Atomikos transaction manager

Atomikos是一个流行的开源事务管理器，可以嵌入到Spring Boot应用程序中。 您可以使用spring-boot-starter-jta-atomikos启动器来引入相应的Atomikos库。 Spring Boot会自动配置Atomikos，并确保适当的依赖设置被应用到你的Spring bean，以便正确的启动和关闭顺序。

默认情况下，Atomikos事务日志将写入应用程序主目录（应用程序jar文件所在的目录）中的事务日志目录。 您可以通过在application.properties文件中设置spring.jta.log-dir属性来自定义此目录。 启动spring.jta.atomikos.properties的属性也可以用来定制Atomikos UserTransactionServiceImp。 有关完整的详细信息，请参阅[AtomikosProperties Javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/jta/atomikos/AtomikosProperties.html) 。

**Note**

为了确保多个事务管理器可以安全地协调相同的资源管理器，每个Atomikos实例必须配置一个唯一的ID。 默认情况下，这个ID是运行Atomikos的机器的IP地址。 为确保生产中的唯一性，您应该为应用程序的每个实例配置不同值的spring.jta.transaction-manager-id属性。

## 使用 Bitronix transaction manager

Bitronix是流行的开源JTA事务管理器实现。 您可以使用spring-boot-starter-jta-bitronix启动器将相应的Bitronix依赖项添加到您的项目中。 和Atomikos一样，Spring Boot会自动配置Bitronix，并对bean进行后处理，以确保启动和关闭顺序是正确的。

默认情况下，Bitronix事务日志文件（part1.btm和part2.btm）将写入应用程序主目录中的事务日志目录。 您可以使用spring.jta.log-dir属性来自定义此目录。 属性起始spring.jta.bitronix.properties也绑定到bitronix.tm.Configuration bean，允许完全自定义。 有关详细信息，请参阅[Bitronix documentation](https://github.com/bitronix/btm/wiki/Transaction-manager-configuration) 。

**Note**

为了确保多个事务管理器可以安全地协调相同的资源管理器，每个Bitronix实例必须配置一个唯一的ID。 默认情况下，这个ID是IP地址Bitronix正在运行的机器。 为确保生产中的唯一性，您应该为应用程序的每个实例配置不同值的spring.jta.transaction-manager-id属性。

## 使用 Narayana transaction manager

Narayana是JBoss支持的流行的开源JTA事务管理器实现。 您可以使用spring-boot-starter-jta-narayana启动器将相应的Narayana依赖项添加到您的项目中。 与Atomikos和Bitronix一样，Spring Boot会自动配置Narayana并后处理您的bean，以确保启动和关闭顺序是正确的。

默认情况下，Narayana事务日志将写入应用程序主目录（应用程序jar文件所在的目录）中的事务日志目录。 您可以通过在application.properties文件中设置spring.jta.log-dir属性来自定义此目录。 启动spring.jta.narayana.properties的属性也可以用来自定义Narayana配置。 有关完整的详细信息，请参阅[NarayanaProperties Javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/jta/narayana/NarayanaProperties.html) 。

**Note**

为确保多个事务管理器可以安全地协调相同的资源管理器，必须为每个Narayana实例配置一个唯一的ID。 默认情况下，这个ID设置为1.为了确保生产中的唯一性，您应该为应用程序的每个实例配置不同值的spring.jta.transaction-manager-id属性。

## 使用 Java EE managed transaction manager

如果您将Spring Boot应用程序打包为war或ear文件并将其部署到Java EE应用程序服务器，则可以使用应用程序服务器内置的事务管理器。 Spring Boot将尝试通过查看常见的JNDI位置（java：comp / UserTransaction，java：comp / TransactionManager等）来自动配置事务管理器。 如果您正在使用应用程序服务器提供的事务服务，则通常还需要确保所有资源都由服务器管理并通过JNDI公开。 Spring Boot将尝试通过在JNDI路径java：/ JmsXA或java：/ XAConnectionFactory中查找ConnectionFactory来自动配置JMS，并且可以使用[spring.datasource.jndi-name property](#_bookmark169)来配置您的DataSource。

## 混合 XA 和 non-XA JMS connections

当使用JTA时，主要的JMS ConnectionFactory bean将被XA感知并参与分布式事务。 在某些情况下，您可能需要使用非XA ConnectionFactory来处理某些JMS消息。 例如，您的JMS处理逻辑可能需要比XA超时更长的时间。

如果要使用非XA ConnectionFactory，则可以注入nonXaJmsConnectionFactory Bean，而不是@Primary jmsConnectionFactory Bean。 为了保持一致性，还使用bean别名xaJmsConnectionFactory提供了jmsConnectionFactory bean。

For example:

*// Inject the primary (XA aware) ConnectionFactory*

@Autowired

**private** ConnectionFactory defaultConnectionFactory;

*// Inject the XA aware ConnectionFactory (uses the alias and injects the same as above)*

@Autowired @Qualifier("xaJmsConnectionFactory")

**private** ConnectionFactory xaConnectionFactory;

*// Inject the non-XA aware ConnectionFactory* @Autowired @Qualifier("nonXaJmsConnectionFactory")

**private** ConnectionFactory nonXaConnectionFactory;

## 支持另一种嵌入式 transaction manager

[XAConnectionFactoryWrapper](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jta/XAConnectionFactoryWrapper.java) 和 [XADataSourceWrapper](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jta/XADataSourceWrapper.java) 接口可用于支持替代嵌入式事务管理器。 这些接口负责包装XAConnectionFactory和XADataSource bean，并将它们公开为常规的ConnectionFactory和DataSource bean，它们将透明地注册到分布式事务中。 只要您的ApplicationContext中注册了JtaTransactionManager bean和适当的XA包装Bean，DataSource和JMS自动配置将使用JTA变体。

[BitronixXAConnectionFactoryWrapper](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jta/bitronix/BitronixXAConnectionFactoryWrapper.java) 和 [BitronixXADataSourceWrapper](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jta/bitronix/BitronixXADataSourceWrapper.java) 提供了如何编写XA包装的很好的例子。

# Hazelcast

如果Hazelcast在类路径中，Spring Boot将自动配置一个可以在应用程序中注入的HazelcastInstance。 只有在找到配置的情况下才会创建HazelcastInstance。

你可以定义一个com.hazelcast.config.Config bean，我们将使用它。 如果您的配置定义了一个实例名称，我们将尝试找到一个现有的实例，而不是创建一个新的实例。

您还可以通过配置指定hazelcast.xml配置文件：

**spring.hazelcast.config**=classpath:config/my-hazelcast.xml

否则，Spring Boot将尝试从默认位置（即工作目录中的hazelcast.xml或类路径的根目录）中查找Hazelcast配置。 我们还检查是否设置了hazelcast.config系统属性。 查看[Hazelcast documentation](http://docs.hazelcast.org/docs/latest/manual/html-single/) 了解更多详情。

**Note**

Spring Boot也对[explicit caching support for Hazelcast](#_bookmark220)。 如果启用了缓存，则HazelcastInstance会自动包装在CacheManager实现中。

# Spring 集成(Integration)

Spring Boot为Spring Integration提供了一些便利，包括spring-boot-starter-integration“Starter”。 Spring Integration提供了对消息传递和其他传输（如HTTP，TCP等）的抽象。如果Spring Integration在类路径中可用，它将通过@EnableIntegration注释进行初始化。 如果'spring-integration-jmx'也在类路径上，消息处理统计将通过JMX发布。 有关更多详细信息，请参阅[IntegrationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/integration/IntegrationAutoConfiguration.java) 类。

# Spring Session

Spring Boot为广泛的存储提供Spring Session自动配置：

* JDBC
* MongoDB
* Redis
* Hazelcast
* HashMap

如果Spring Session可用，则必须选择您希望用来存储会话的[StoreType](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/StoreType.java) 。 例如，要将JDBC用作后端存储，您可以按如下方式配置您的应用程序：

**spring.session.store-type**=jdbc

**Tip**

您可以通过将store-type设置为none来禁用Spring Session。

每个store都有特定的附加设置。 例如，可以为jdbc存储自定义表的名称：

**spring.session.jdbc.table-name**=SESSIONS

# 监控 和 管理 JMX

Java管理扩展（JMX）提供了一个标准机制来监视和管理应用程序。 默认情况下，Spring Boot将创建一个bean id为“mbeanServer”的MBeanServer，并公开所有使用Spring JMX注释（@ManagedResource，@ManagedAttribute，@ManagedOperation）注释的bean。

有关更多详细信息，请参阅[JmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jmx/JmxAutoConfiguration.java) 类。

# Testing

Spring Boot提供了许多实用程序和注解来帮助您测试应用程序。 测试支持由两个模块提供; spring-boot-test包含核心项目，而spring-boot-test-autoconfigure支持测试的自动配置。

大多数开发人员将只使用spring-boot-starter-test“Starter”，它可以导入Spring Boot测试模块以及JUnit，AssertJ，Hamcrest和其他一些有用的库。

## Test scope dependencies

如果你使用spring-boot-starter-test'Starter'（在测试范围内），你会发现以下提供的库：

* [JUnit](http://junit.org/) — 单元测试Java应用程序的事实标准。
* [Spring Test](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#integration-testing) & Spring Boot测试 - Spring Boot应用程序的实用程序和集成测试支持。
* [AssertJ](http://joel-costigliola.github.io/assertj/) —流利的断言库( A fluent assertion library)
* [Hamcrest](http://hamcrest.org/JavaHamcrest/) — 匹配器(matcher)对象库(也称为 constraints或predicates).
* [Mockito](http://mockito.org/) — A Java mocking framework.
* [JSONassert](https://github.com/skyscreamer/JSONassert) — JSON断言库(assertion library).
* [JsonPath](https://github.com/jayway/JsonPath) — 适用于JSON的Xpath(XPath for JSON)

**Note**

默认情况下，Spring Boot使用Mockito 1.x. 但是，如果你愿意，也可以使用2.x。

这些是我们通常在编写测试时觉得有用的常用库。 如果这些不符合您的需求，您可以自由添加您自己的附加测试依赖项。

## Testing Spring applications

依赖注入的一个主要优点是它应该使你的代码更容易进行单元测试。 你甚至可以简单地使用new运算符实例化对象，甚至不涉及Spring。 您也可以使用模拟对象而不是真正的依赖关系。

通常你需要超越“单元测试”，开始“集成测试”（实际上涉及到Spring ApplicationContext）。 能够在不需要部署应用程序或不需要连接到其他基础架构的情况下执行集成测试就很有用。

Spring Framework包含一个专门用于这种集成测试的测试模块。 您可以直接向org.springframework声明一个依赖项：spring-test或者使用spring-boot-starter-test“Starter”来传递它。

如果你还没有使用过spring-test模块，那么你应该先阅读Spring框架参考文档的相关部分。

## Testing Spring Boot applications

一个Spring Boot应用程序只是一个Spring ApplicationContext，所以没有什么特别的事情要做，以超越你通常使用vanilla Spring上下文进行测试。 但要注意的一点是，Spring Boot的外部属性，日志记录和其他功能只有在使用SpringApplication创建时才会默认安装在上下文中。

Spring Boot提供了一个@SpringBootTest注解，当您需要Spring Boot功能时，它可以用作标准spring-test @ContextConfiguration注释的替代方法。 注解的工作原理是通过SpringApplication在测试中创建ApplicationContext。

您可以使用@SpringBootTest的webEnvironment属性来进一步优化测试的运行方式：

* MOCK - 加载一个WebApplicationContext并提供一个模拟servlet环境。 嵌入式servlet容器在使用此注释时不会启动。 如果servlet API不在你的类路径上，这个模式将透明地回退到创建一个普通的非Web ApplicationContext。 可以与@AutoConfigureMockMvc结合使用，用于基于MockMvc的应用程序测试。
* RANDOM\_PORT - 加载一个EmbeddedWebApplicationContext并提供一个真正的servlet环境。 嵌入式servlet容器启动并在随机端口上侦听。
* DEFINED\_PORT - 加载EmbeddedWebApplicationContext并提供一个真正的servlet环境。 嵌入式servlet容器启动并监听定义的端口（即从application.properties或默认端口8080）。
* NONE — 使用SpringApplication加载ApplicationContext，但不提供人户servlet环境(mock或其它)

**Note**

如果你的测试是@Transactional，默认情况下它将在每个测试方法结束时回滚事务。 然而，由于使用RANDOM\_PORT或DEFINED\_PORT这种安排隐式地提供了一个真正的servlet环境，所以HTTP客户机和服务器将在不同的线程中运行，从而分离事务。 在这种情况下，在服务器上启动的任何事务都不会回滚。

**Note**

除了@SpringBootTest之外，还提供了许多其他注释来测试应用程序的更具体的切片。 详情请参阅下文。

**Tip**

不要忘记还要在测试中添加@RunWith（SpringRunner.class），否则注释将被忽略。

### Detecting test configuration

如果您熟悉Spring测试框架，则可以使用@ContextConfiguration（classes = ...）来指定要加载哪个Spring @Configuration。 或者，您可能经常在测试中使用嵌套的@Configuration类。

在测试Spring Boot应用程序时，这通常不是必需的。 Spring Boot的@ \*Test注释会自动搜索您的主要配置，只要您没有明确定义一个配置。

搜索算法从包含测试的包开始工作，直到找到@SpringBootApplication或@SpringBootConfiguration注释类。 只要你以合理的方式[structured your code](#_bookmark50) ，你的主要配置通常是可以找到的。

如果你想定制主配置，你可以使用一个嵌套的@TestConfiguration类。 与嵌套的@Configuration类不同，它将被用来代替应用程序的主要配置，除了应用程序的主要配置之外，还将使用一个嵌套的@TestConfiguration类。

**Note**

Spring的测试框架将缓存测试之间的应用程序上下文。 因此，只要您的测试共享相同的配置（不管它如何被发现），加载上下文的潜在的耗时过程将只发生一次。

### Excluding test configuration

如果您的应用程序使用组件扫描，例如，如果您使用@SpringBootApplication或@ComponentScan，则可能会发现仅为特定测试创建的top-level配置类意外地被加载。

[have seen above](#_bookmark262)，@ TestConfiguration可以在测试的内部类上使用来定制主配置。 当放置在顶级类上时，@ TestConfiguration指示不应该通过扫描来拾取src / test / java中的类。 然后可以在需要的地方明确地导入该类：

@RunWith(SpringRunner.class) @SpringBootTest @Import(MyTestsConfiguration.class) **public class** MyTests {

@Test

**public void** exampleTest() {

...

}

}

**Note**

如果直接使用@ComponentScan（即不通过@SpringBootApplication），则需要使用TypeExcludeFilter注册。 有关详细信息，请参阅[the Javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/context/TypeExcludeFilter.html) 。

### Working with random ports

如果您需要启动完整的运行服务器进行测试，我们建议您使用随机端口。 如果您使用@SpringBootTest（webEnvironment = WebEnvironment.RANDOM\_PORT），则每次运行测试时都会随机选取一个可用端口。

@LocalServerPort注解可以用来注入用于测试的实际端口([inject the actual port used](#_bookmark463) )。 为了方便起见，需要对已启动的服务器进行REST调用的测试还可以@Autowire一个TestRestTemplate，它将解析到正在运行的服务器的相关链接。

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

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

**import** org.springframework.boot.test.context.SpringBootTest;

**import** org.springframework.boot.test.context.SpringBootTest.WebEnvironment;

**import** org.springframework.boot.test.web.client.TestRestTemplate;

**import** org.springframework.test.context.junit4.SpringRunner;

**import static** org.assertj.core.api.Assertions.assertThat; @RunWith(SpringRunner.class)

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT)

**public class** RandomPortExampleTests {

@Autowired

**private** TestRestTemplate restTemplate;

@Test

**public void** exampleTest() {

String body = **this**.restTemplate.getForObject(***"/"***, String.**class**); assertThat(body).isEqualTo(***"Hello World"***);

}

}

### Mocking and spying beans

运行测试时有时需要在应用程序上下文中模拟某些组件。 例如，您可能对开发期间不可用的某个远程服务有一个正面(facade)。 当你想模拟在真实环境中很难触发的故障时，模拟也是有用的。

**Note**

只要您的测试使用Spring Boot的测试注释（即@SpringBootTest），该功能就会自动启用。 要以不同的安排使用此功能，需要明确添加侦听器：

@TestExecutionListeners(MockitoTestExecutionListener.**class**)

下面是一个典型的例子，我们用一个模拟实现替换现有的RemoteService bean：

**import** org.junit.\*;

**import** org.junit.runner.\*;

**import** org.springframework.beans.factory.annotation.\*;

**import** org.springframework.boot.test.context.\*; **import** org.springframework.boot.test.mock.mockito.\*; **import** org.springframework.test.context.junit4.\*;

**import static** org.assertj.core.api.Assertions.\*;

**import static** org.mockito.BDDMockito.\*;

@RunWith(SpringRunner.class) @SpringBootTest

**public class** MyTests {

@MockBean

**private** RemoteService remoteService;

@Autowired

**private** Reverser reverser;

@Test

**public void** exampleTest() {

*// RemoteService has been injected into the reverser bean* given(**this**.remoteService.someCall()).willReturn(***"mock"***); String reverse = reverser.reverseSomeCall(); assertThat(reverse).isEqualTo(***"kcom"***);

}

}

另外你也可以使用@SpyBean来包装任何现有的Mockito spy。 请参阅Javadoc了解详细信息。

### Auto-configured tests

Spring Boot的自动配置系统适用于应用程序，但有时可能对测试有点太多。 只加载测试应用程序的“slice”所需的配置部分通常很有帮助。 例如，您可能想要测试Spring MVC控制器是否正确映射了URL，并且您不希望在这些测试中涉及数据库调用; 或者您可能想要测试JPA实体，并且在这些测试运行时您对web层不感兴趣。

spring-boot-test-autoconfigure模块包含许多注释，可用于自动配置这些“切片”。 它们中的每一个都以类似的方式工作，提供了一个@ ...Test注释，用于加载ApplicationContext和一个或多个@ AutoConfigure ...注释，可用于自定义自动配置设置。

**Note**

每个切片加载一组非常有限的自动配置类。 如果您需要排除其中的一个，大多数@ ...Test注释提供了一个excludeAutoConfiguration属性。 或者，您可以使用@ ImportAutoConfiguration＃exclude。

**Tip**

也可以将@ AutoConfigure ...注释与标准的@SpringBootTest注解一起使用。 如果您对“slice”应用程序不感兴趣，但想要一些自动配置的测试bean，则可以使用此组合。

### Auto-configured JSON tests

要测试对象JSON序列化和反序列化是否按预期工作，可以使用@JsonTest注释。 @JsonTest将自动配置Jackson ObjectMapper，任何@JsonComponent bean和任何Jackson模块。 它也配置Gson，如果你碰巧使用，而不是，或Jackson。 如果您需要配置自动配置的元素，则可以使用@AutoConfigureJsonTesters注释。

Spring Boot包含基于AssertJ的帮助程序，它们与JSONassert和JsonPath库一起工作来检查JSON是否

如预期的那样。 JacksonTester，GsonTester和BasicJsonTester类可以分别用于Jackson，Gson和Strings。 测试类上的任何助手字段都可以使用@JsonTest时@Autowired。

**import** org.junit.\*;

**import** org.junit.runner.\*;

**import** org.springframework.beans.factory.annotation.\*; **import** org.springframework.boot.test.autoconfigure.json.\*; **import** org.springframework.boot.test.context.\*;

**import** org.springframework.boot.test.json.\*;

**import** org.springframework.test.context.junit4.\*;

**import static** org.assertj.core.api.Assertions.\*; @RunWith(SpringRunner.class)

@JsonTest

**public class** MyJsonTests {

@Autowired

**private** JacksonTester<VehicleDetails> json;

@Test

**public void** testSerialize() **throws** Exception {

VehicleDetails details = **new** VehicleDetails(***"Honda"***, ***"Civic"***);

*// Assert against a `.json` file in the same package as the test*

assertThat(**this**.json.write(details)).isEqualToJson(***"expected.json"***);

*// Or use JSON path based assertions* assertThat(**this**.json.write(details)).hasJsonPathStringValue(***"@.make"***); assertThat(**this**.json.write(details)).extractingJsonPathStringValue(***"@.make"***)

.isEqualTo(***"Honda"***);

}

@Test

**public void** testDeserialize() **throws** Exception {

String content = ***"{\"make\":\"Ford\",\"model\":\"Focus\"}"***; assertThat(**this**.json.parse(content))

.isEqualTo(**new** VehicleDetails(***"Ford"***, ***"Focus"***)); assertThat(**this**.json.parseObject(content).getMake()).isEqualTo(***"Ford"***);

}

}

**Note**

JSON帮助程序类也可以直接在标准的单元测试中使用。 只需调用initFields

如果您没有使用@JsonTest，则在@Before方法中使用助手的方法。

@JsonTest启用的自动配置列表可以在[found in the appendix](#_bookmark592)中找到。

### Auto-configured Spring MVC tests

要测试Spring MVC控制器是否按预期工作，可以使用@WebMvcTest注释。 @WebMvcTest将自动配置Spring MVC基础架构，并将扫描的bean限制为@Controller，@ControllerAdvice，@JsonComponent，Filter，WebMvcConfigurer和HandlerMethodArgumentResolver。 使用此批注时，不会扫描常规的@Component bean。

通常情况下，@ WebMvcTest将被限制在单个控制器中，并与@MockBean结合使用，为所需的合作者提供模拟实现。

@WebMvcTest也自动配置MockMvc。 模拟MVC提供了一种快速测试MVC控制器的强大方法，无需启动完整的HTTP服务器。

**Tip**

您还可以通过使用@AutoConfigureMockMvc注释来在non-@WebMvcTest（例如SpringBootTest）中自动配置MockMvc。

**import** org.junit.\*;

**import** org.junit.runner.\*;

**import** org.springframework.beans.factory.annotation.\*;

**import** org.springframework.boot.test.autoconfigure.web.servlet.\*;

**import** org.springframework.boot.test.mock.mockito.\*;

**import static** org.assertj.core.api.Assertions.\*;

**import static** org.mockito.BDDMockito.\*;

**import static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@RunWith(SpringRunner.class) @WebMvcTest(UserVehicleController.class) **public class** MyControllerTests {

@Autowired

**private** MockMvc mvc;

@MockBean

**private** UserVehicleService userVehicleService;

@Test

**public void** testExample() **throws** Exception { given(**this**.userVehicleService.getVehicleDetails(***"sboot"***))

.willReturn(**new** VehicleDetails(***"Honda"***, ***"Civic"***));

**this**.mvc.perform(get(***"/sboot/vehicle"***).accept(MediaType.TEXT\_PLAIN))

.andExpect(status().isOk()).andExpect(content().string(***"Honda Civic"***));

}

}

**Tip**

如果需要配置自动配置的元素（例如，应应用servlet过滤器），则可以使用@AutoConfigureMockMvc注释中的属性。

如果您使用HtmlUnit或Selenium，自动配置也将提供WebClient bean和/或

WebDriver bean。 这里是一个使用HtmlUnit的例子：

**import** com.gargoylesoftware.htmlunit.\*;

**import** org.junit.\*;

**import** org.junit.runner.\*;

**import** org.springframework.beans.factory.annotation.\*;

**import** org.springframework.boot.test.autoconfigure.web.servlet.\*;

**import** org.springframework.boot.test.mock.mockito.\*;

**import static** org.assertj.core.api.Assertions.\*;

**import static** org.mockito.BDDMockito.\*;

@RunWith(SpringRunner.class) @WebMvcTest(UserVehicleController.class) **public class** MyHtmlUnitTests {

@Autowired

**private** WebClient webClient;

@MockBean

**private** UserVehicleService userVehicleService;

@Test

**public void** testExample() **throws** Exception { given(**this**.userVehicleService.getVehicleDetails(***"sboot"***))

.willReturn(**new** VehicleDetails(***"Honda"***, ***"Civic"***)); HtmlPage page = **this**.webClient.getPage(***"/sboot/vehicle.html"***);

assertThat(page.getBody().getTextContent()).isEqualTo(***"Honda Civic"***);

}

}

**Note**

默认情况下，Spring Boot会把WebDriver bean放在一个特殊的“scope”中，以确保在每次测试后驱动程序都退出，并注入新的实例。 如果你不想要这个行为，你可以在你的WebDriver @Bean定义中添加@Scope（“singleton”）。

可以在附录中找到由@WebMvcTest启用的自动配置列表([found in the appendix](#_bookmark592))。

### Auto-configured Data JPA tests

如果你想测试JPA应用程序，可以使用@DataJpaTest。 默认情况下，它将配置内存嵌入式数据库，扫描@Entity类并配置Spring Data JPA存储库。 常规的@Component bean将不会被加载到ApplicationContext中。

数据JPA测试是默认情况下每个测试结束时的事务性和回滚性，有关更多详细信息，请参阅[relevant section](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle#testcontext-tx-enabling-transactions)参考资料中的相关章节。 如果这不是你想要的，你可以按照以下步骤禁用一个测试或整个类的事务管理：

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

**import** org.springframework.test.context.junit4.SpringRunner; **import** org.springframework.transaction.annotation.Propagation; **import** org.springframework.transaction.annotation.Transactional;

@RunWith(SpringRunner.class) @DataJpaTest

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

**public class** ExampleNonTransactionalTests {

}

数据JPA测试也可能注入一个[TestEntityManager](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-test-autoconfigure/src/main/java/org/springframework/boot/test/autoconfigure/orm/jpa/TestEntityManager.java) bean，它提供了专门为测试而设计的标准JPA EntityManager的替代方案。 如果你想在@DataJpaTests之外使用TestEntityManager，你也可以使用@AutoConfigureTestEntityManager注解。 如果您需要，JdbcTemplate也可用。

**import** org.junit.\*;

**import** org.junit.runner.\*;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.\*;

**import static** org.assertj.core.api.Assertions.\*; @RunWith(SpringRunner.class)

@DataJpaTest

**public class** ExampleRepositoryTests {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

**private** UserRepository repository;

@Test

**public void** testExample() **throws** Exception { **this**.entityManager.persist(**new** User(***"sboot"***, ***"1234"***)); User user = **this**.repository.findByUsername(***"sboot"***); assertThat(user.getUsername()).isEqualTo(***"sboot"***); assertThat(user.getVin()).isEqualTo(***"1234"***);

}

}

In-memory的嵌入式数据库通常可以很好地进行测试，因为它们速度快，不需要任何开发者安装。 但是，如果您希望针对真实数据库运行测试，则可以使用@AutoConfigureTestDatabase注解：

@RunWith(SpringRunner.class) @DataJpaTest

@AutoConfigureTestDatabase(replace=Replace.NONE)

**public class** ExampleRepositoryTests {

*// ...*

}

可以在附录中找到由@DataJpaTest启用的自动配置列表([found in the appendix](#_bookmark592))。

### Auto-configured JDBC tests

@JdbcTest类似于@DataJpaTest，但对于纯jdbc相关的测试。 默认情况下，它也将配置一个内存嵌入式数据库和一个JdbcTemplate。 常规的@Component bean将不会被加载到ApplicationContext中。

JDBC测试在每个测试结束时都是事务性的和回滚的，默认情况下，请参阅Spring Reference Documentation([relevant section](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle#testcontext-tx-enabling-transactions))中的相关部分以获取更多细节。 如果这不是你想要的，你可以按照以下步骤禁用一个测试或整个类的事务管理：

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.springframework.boot.test.autoconfigure.jdbc.JdbcTest; **import** org.springframework.test.context.junit4.SpringRunner; **import** org.springframework.transaction.annotation.Propagation; **import** org.springframework.transaction.annotation.Transactional;

@RunWith(SpringRunner.class) @JdbcTest

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

**public class** ExampleNonTransactionalTests {

}

如果你喜欢你的测试运行在真实的数据库上，你可以使用@AutoConfigureTestDatabase

注解的方式与DataJpaTest相同。

@JdbcTest启用的自动配置列表可以在附录([found in the appendix](#_bookmark592))中找到。

### Auto-configured Data MongoDB tests

如果要测试MongoDB应用程序，可以使用@DataMongoTest。 默认情况下，它将配置内存中嵌入的MongoDB（如果可用），配置MongoTemplate，扫描@Document类并配置Spring Data MongoDB存储库。 常规的@Component bean将不会被加载到ApplicationContext中：

**import** org.junit.runner.RunWith;

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

**import** org.springframework.boot.test.autoconfigure.data.mongo.DataMongoTest;

**import** org.springframework.data.mongodb.core.MongoTemplate;

**import** org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class) @DataMongoTest

**public class** ExampleDataMongoTests {

@Autowired

**private** MongoTemplate mongoTemplate;

*//*

}

内存中的嵌入式MongoDB通常适用于测试，因为它速度快，不需要任何开发人员的安装。 但是，如果您希望针对真正的MongoDB服务器运行测试，则应该排除嵌入式MongoDB自动配置：

**import** org.junit.runner.RunWith;

**import** org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongoAutoConfiguration;

**import** org.springframework.boot.test.autoconfigure.data.mongo.DataMongoTest;

**import** org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@DataMongoTest(excludeAutoConfiguration = EmbeddedMongoAutoConfiguration.class)

**public class** ExampleDataMongoNonEmbeddedTests {

}

可以在附录([found in the appendix](#_bookmark592))中找到@DataMongoTest启用的自动配置列表。

### Auto-configured REST clients

如果您想测试REST客户端，则可以使用@RestClientTest注解。 默认情况下，它会自动配置Jackson和GSON支持，配置RestTemplateBuilder并添加对MockRestServiceServer的支持。 应该使用@RestClientTest的value或components属性指定要测试的特定bean：

@RunWith(SpringRunner.class) @RestClientTest(RemoteVehicleDetailsService.class) **public class** ExampleRestClientTest {

@Autowired

**private** RemoteVehicleDetailsService service;

@Autowired

**private** MockRestServiceServer server;

@Test

**public void** getVehicleDetailsWhenResultIsSuccessShouldReturnDetails()

**throws** Exception {

**this**.server.expect(requestTo(***"/greet/details"***))

.andRespond(withSuccess(***"hello"***, MediaType.TEXT\_PLAIN)); String greeting = **this**.service.callRestService(); assertThat(greeting).isEqualTo(***"hello"***);

}

}

可以在附录([found in the appendix](#_bookmark592))中找到由@RestClientTest启用的自动配置列表。

### Auto-configured Spring REST Docs tests

如果您想在测试中使用Spring REST Docs，可以使用@AutoConfigureRestDocs注释。 它会自动配置MockMvc来使用Spring REST Docs，并删除对Spring REST Docs的JUnit规则的需求。

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

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

**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

**import** org.springframework.http.MediaType;

**import** org.springframework.test.context.junit4.SpringRunner;

**import** org.springframework.test.web.servlet.MockMvc;

**import static** org.springframework.restdocs.mockmvc.MockMvcRestDocumentation.document; **import static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get; **import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@RunWith(SpringRunner.class) @WebMvcTest(UserController.class) @AutoConfigureRestDocs("target/generated-snippets") **public class** UserDocumentationTests {

@Autowired

**private** MockMvc mvc;

@Test

**public void** listUsers() **throws** Exception {

**this**.mvc.perform(get(***"/users"***).accept(MediaType.TEXT\_PLAIN))

.andExpect(status().isOk())

.andDo(document(***"list-users"***));

}

}

除了配置输出目录之外，@AutoConfigureRestDocs还可以配置将显示在任何记录的URI中的host, scheme, 和 port。 如果您需要更多地控制Spring REST Docs的配置，可以使用RestDocsMockMvcConfigurationCustomizer bean：

@TestConfiguration

**static class** CustomizationConfiguration

**implements** RestDocsMockMvcConfigurationCustomizer {

@Override

**public void** customize(MockMvcRestDocumentationConfigurer configurer) { configurer.snippets().withTemplateFormat(TemplateFormats.markdown());

}

}

如果要使用Spring REST Docs对参数化输出目录的支持，可以创建一个RestDocumentationResultHandler bean。 自动配置将使用此结果处理程序调用alwaysDo，从而导致每个MockMvc调用自动生成默认片段：

@TestConfiguration

**static class** ResultHandlerConfiguration {

@Bean

**public** RestDocumentationResultHandler restDocumentation() {

**return** MockMvcRestDocumentation.document(***"{method-name}"***);

}

}

### Using Spock to test Spring Boot applications

如果您希望使用Spock来测试Spring Boot应用程序，则应该在Spock的spock-spring模块中添加对应用程序构建的依赖关系。 Spock-Spring将Spring的测试框架集成到Spock中。 具体如何使用Spock来测试Spring Boot应用程序取决于您正在使用的Spock版本。

**Note**

Spring Boot为Spock 1.0提供依赖管理。 如果你想使用Spock 1.1，你应该重写build.gradle中的[override the spock.version property](#_bookmark550) 或者pom.xml文件中的spock.version属性。

当使用Spock 1.1时，上述([described above](#_bookmark261) )注释只能用于注释使用@SpringBootTest进行规范，

以满足您的测试需求。

使用Spock 1.0时，@ SpringBootTest将不适用于Web项目。 你需要使用@SpringApplicationConfiguration和@WebIntegrationTest（randomPort = true）。 无法使用@SpringBootTest意味着您也失去了自动配置的TestRestTemplate bean。 您可以使用以下配置自行创建等价的bean：

@Configuration

**static class** TestRestTemplateConfiguration {

@Bean

**public** TestRestTemplate testRestTemplate( ObjectProvider<RestTemplateBuilder> builderProvider, Environment environment) {

RestTemplateBuilder builder = builderProvider.getIfAvailable(); TestRestTemplate template = builder == null ? **new** TestRestTemplate()

: **new** TestRestTemplate(builder.build()); template.setUriTemplateHandler(**new** LocalHostUriTemplateHandler(environment)); **return** template;

}

}

## Test utilities

一些测试实用程序类被封装为spring-boot的一部分，在测试应用程序时通常很有用。

### ConfigFileApplicationContextInitializer

ConfigFileApplicationContextInitializer是一个ApplicationContextInitializer，它可以应用于你的测试来加载Spring Boot的application.properties文件。 当你不需要@SpringBootTest提供的完整功能时，可以使用它。

@ContextConfiguration(classes = Config.**class**,

initializers = ConfigFileApplicationContextInitializer.**class**)

**Note**

单独使用ConfigFileApplicationContextInitializer不会提供对@Value（“$ {...}”）注入的支持。 唯一的工作是确保application.properties文件被加载到Spring的环境中。 对于@Value支持，您需要另外配置

### EnvironmentTestUtils

EnvironmentTestUtils允许您快速添加属性到ConfigurableEnvironment或

ConfigurableApplicationContext。 只需使用key = value字符串调用它即可：

EnvironmentTestUtils.addEnvironment(env, ***"org=Spring"***, ***"name=Boot"***);

### OutputCapture

OutputCapture是一个JUnit规则，可以用来捕获System.out和System.err输出。 只需将捕获声明为@Rule，然后使用toString（）进行断言：

**import** org.junit.Rule;

**import** org.junit.Test;

**import** org.springframework.boot.test.rule.OutputCapture;

**import static** org.hamcrest.Matchers.\*;

**import static** org.junit.Assert.\*;

**public class** MyTest { @Rule

**public** OutputCapture capture = **new** OutputCapture();

@Test

**public void** testName() **throws** Exception { System.out.println(***"Hello World!"***); assertThat(capture.toString(), containsString(***"World"***));

}

}

### TestRestTemplate

TestRestTemplate是Spring RestTemplate的一个方便选择，在集成测试中很有用。 您可以获得一个vanilla template或一个发送基本HTTP认证（使用用户名和密码）的模板。 无论哪种情况，模板都将以适合测试的方式运行，不会在服务器端错误上抛出异常。 建议使用Apache HTTP Client（版本4.3.2或更高版本），但不是强制性的，如果你的类路径上有这个，TestRestTemplate将通过适当地配置客户端进行响应。 如果您使用Apache的HTTP客户端，则会启用一些额外的测试友好功能：

* 重定向不会被遵循（所以你可以断言响应位置）
* Cookie将被忽略（所以模板是无状态的）

TestRestTemplate可以在集成测试中直接实例化：

**public class** MyTest {

**private** TestRestTemplate template = **new** TestRestTemplate(); @Test

**public void** testRequest() **throws** Exception {

HttpHeaders headers = template.getForEntity([***"http://myhost.com/example"***](http://myhost.com/example), String.**class**).getHeaders();

assertThat(headers.getLocation().toString(), containsString(***"myotherhost"***));

}

}

另外，如果您使用WebEnvironment.RANDOM\_PORT或WebEnvironment.DEFINED\_PORT使用@SpringBootTest注释，则只需注入完全配置的TestRestTemplate并开始使用它。 如有必要，可以通过RestTemplateBuilder bean应用其他自定义设置。 任何不指定主机和端口的URL都将自动连接到嵌入式服务器：

@RunWith(SpringRunner.class) @SpringBootTest

**public class** MyTest {

@Autowired

**private** TestRestTemplate template;

@Test

**public void** testRequest() **throws** Exception {

HttpHeaders headers = template.getForEntity(***"/example"***, String.**class**).getHeaders(); assertThat(headers.getLocation().toString(), containsString(***"myotherhost"***));

}

@TestConfiguration

**static class** Config {

@Bean

**public** RestTemplateBuilder restTemplateBuilder() {

**return new** RestTemplateBuilder()

.additionalMessageConverters(...)

.customizers(...);

}

}

}

# WebSockets

Spring Boot为嵌入式Tomcat（8和7），Jetty 9和Undertow提供WebSockets自动配置。 如果您将war文件部署到独立容器，Spring Boot会假定该容器将负责配置其WebSocket支持。

Spring Framework提供了丰富的WebSocket支持[rich WebSocket support](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#websocket) ，可以通过spring-boot-starter-websocket模块轻松访问。

# Web Services

Spring Boot提供Web服务自动配置，以便所有需要定义您的端点。

Spring Web Services功能([Spring Web Services features](http://docs.spring.io/spring-ws/docs/2.4.0.RELEASE/reference/htmlsingle))可以通过

spring-boot-starter-webservices模块轻松访问。

# 创建自己的 auto-configuration

如果您在开发共享库的公司工作，或者如果您在开源或商业库上工作，则可能需要开发自己的自动配置。 自动配置类可以捆绑在外部的jars里，并且仍然可以被Spring Boot找到。

自动配置可以与提供自动配(auto-configuration)）代码的“启动器”以及您将使用的典型库相关联。 我们将首先介绍您需要知道的构建自己的自动配置，然后我们将继续执行创建自定义启动器所需的典型步骤([typical steps required to create a custom starter](#_bookmark292))。

**Tip**

[demo project](https://github.com/snicoll-demos/spring-boot-master-auto-configuration) 可用于展示如何逐步创建启动器。

## 了解 auto-configured beans

在引擎盖下，自动配置是通过标准的@Configuration类来实现的。 额外的@Conditional注解用于约束何时应用自动配置。 通常，自动配置类使用@ConditionalOnClass和@ConditionalOnMissingBean注释。 这确保只有在找到相关的类并且没有声明自己的@Configuration时，自动配置才会适用。

您可以浏览f [spring-boot-autoconfigure](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure) 的源代码来查看@Configuration

我们提供的类（参见[META-INF/spring.factories](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/resources/META-INF/spring.factories) 文件）。

## 加载 auto-configuration (候选)candidates

Spring Boot检查发布的jar文件中是否存在META-INF / spring.factories文件。 该文件应在EnableAutoConfiguration密钥下列出您的配置类。

org.springframework.boot.autoconfigure.EnableAutoConfiguration=\ com.mycorp.libx.autoconfigure.LibXAutoConfiguration,\ com.mycorp.libx.autoconfigure.LibXWebAutoConfiguration

如果需要按特定顺序应用配置，则可以使用[@AutoConfigureAfter](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/AutoConfigureAfter.java) 或 [@AutoConfigureBefore](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/AutoConfigureBefore.java) 注释。 例如，如果您提供特定于Web的配置，则可能需要在WebMvcAutoConfiguration之后应用您的类。

如果您想要订购某些不应彼此直接了解的自动配置，则还可以使用@AutoconfigureOrder。 该注释具有与常规@Order注释相同的语义，但为自动配置类提供了专用的顺序。

**Note**

自动配置只能以这种方式加载。 确保它们在特定的包装空间中定义，并且它们绝对不是特别针对组件扫描的目标。

## 条件 annotations

您几乎总是希望在自动配置类中包含一个或多个@Conditional注释。 @ConditionalOnMissingBean是一个常用的例子，如果开发人员不满意你的默认值，开发人员可以“覆盖”自动配置。

Spring Boot包含许多@Conditional注释，您可以在自己的代码中重复使用批注@Configuration类或各个@Bean方法。

### Class 条件(conditions)

@ConditionalOnClass和@ConditionalOnMissingClass注释允许根据是否存在特定的类来包含配置。 由于使用[ASM](http://asm.ow2.org/) 解析注释元数据的事实，您实际上可以使用value属性来引用真实类，即使该类可能实际上并不出现在正在运行的应用程序类路径中。 如果您希望使用字符串值指定类名称，也可以使用name属性。

**Tip**

如果您使用@ConditionalOnClass或@ConditionalOnMissingClass作为元注释的一部分来组成自己的注释，那么您必须使用名称作为引用类，在这种情况下不处理。

### Bean 条件(conditions)

@ConditionalOnBean和@ConditionalOnMissingBean注释允许基于特定bean的存在或不存在来包含bean。 您可以使用value属性来按类型指定bean，或者使用name来指定bean的名称。 搜索属性允许您限制搜索bean时应考虑的ApplicationContext层次结构。

放置在@Bean方法上时，目标类型默认为方法的返回类型，例如：

@Configuration

**public class** MyAutoConfiguration {

@Bean @ConditionalOnMissingBean

**public** MyService myService() { ... }

}

在上面的例子中，如果在ApplicationContext中没有包含MyService类型的bean，myService bean将被创建。

**Tip**

您需要非常小心添加bean定义的顺序，因为这些条件是基于到目前为止处理的内容进行评估的。 出于这个原因，我们建议在自动配置类上只使用@ConditionalOnBean和@ConditionalOnMissingBean注释（因为在添加任何用户定义的bean定义之后，这些注释会被保证加载）。

**Note**

@ConditionalOnBean和@ConditionalOnMissingBean不会阻止创建@Configuration类。 在类级使用这些条件相当于用注释标记每个包含的@Bean方法。

### 属性条件(Property conditions)

@ConditionalOnProperty注解允许基于Spring环境属性包含配置。 使用前缀和名称属性来指定应该检查的属性。 默认情况下，任何存在且不等于false的属性都将被匹配。 您还可以使用havingValue和matchIfMissing属性创建更高级的检查。

### 资源条件(Resource conditions)

@ConditionalOnResource批注允许配置仅在特定资源存在时才包含。 可以使用常规的Spring约定来指定资源，例如file：/home/user/test.dat。

### Web application conditions

@ConditionalOnWebApplication和@ConditionalOnNotWebApplication批注允许包含配置，具体取决于应用程序是否是“Web应用程序”。 Web应用程序是使用Spring WebApplicationContext的任何应用程序，定义会话范围(scope)或具有StandardServletEnvironment。

### SpEL expression conditions

@ConditionalOnExpression批注允许根据[SpEL expression](http://docs.spring.io/spring/docs/4.3.12.RELEASE/spring-framework-reference/htmlsingle/#expressions)的结果包含配置。

## 创建自己的 starter

一个完整的Spring Boot启动器可能包含以下组件：

* 包含自动配置代码的自动配置模块。
* 启动器模块提供对自动配置模块的依赖关系，以及库和任何通常有用的附加依赖项。 简而言之，添加启动器应该足以开始使用该库。

**Tip**

如果您不需要将这两个问题分开，您可以将自动配置代码和依赖管理结合在一个模块中。

### Naming

请确保为您的初学者提供适当的名称空间(namespace)。 即使使用不同的Maven组标识，也不要使用spring-boot启动模块名称。 我们可能会为您将来自动配置的东西提供官方支持。

这是一个经验法则。 假设您正在为“acme”创建一个启动器(starter)，命名自动配置模块acme-spring-boot-autoconfigure和启动器acme-spring-boot-starter。 如果只有一个模块组合这两个模块，请使用acme-spring-boot-starter。

此外，如果您的初学者提供配置密钥(configuration keys)，请为其使用适当的名称空间。 特别是，不要把你的密钥包含在Spring Boot使用的命名空间中（例如server, management, spring等）。 这些都是“ours”，我们可能会在未来改善/修改它们，这样可能会破坏你的东西。

确保[trigger meta-data generation](#_bookmark586)，以便您的密钥(keys)也可以使用IDE帮助。 您可能需要查看生成的元数据（META-INF / spring-configuration-metadata.json），以确保您的密钥(keys)被正确记录。

### 自动配置模块 (Autoconfigure module)

自动配置模块包含开始使用库所需的一切。 它还可能包含配置键(keys)定义（@ConfigurationProperties）和任何回调接口，可用于进一步定制组件的初始化方式。

**Tip**

您应该将库的依赖项标记为可选，以便您可以更轻松地将自动配置模块包含在项目中。 如果你这样做，库将不会被提供，Spring Boot默认会退出。

### Starter module

首先是一个空jar，真的。 它唯一的目的是提供必要的依赖来与library合作。 把它看作是对开始所需要的东西的一种自以为是的观点。

不要对添加起动器的项目做出假设。 如果您自动配置的库通常需要其他启动程序，请提及它们。 如果可选依赖项的数量很高，那么提供一组适当的默认依赖关系可能会很困难，因为您应该避免为库的典型用法带来不必要的依赖关系。

# What to read next

如果您想了解更多关于本节讨论的任何类的信息，可以查看[Spring](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api) [Boot API documentation](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api) ，也可以直接浏览[source code directly](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE)。 如果您有具体问题，请参阅操作指南部分[how-to](#_bookmark438) 。

如果您对Spring Boot的核心功能感到满意，可以继续阅读有关生产就绪功能[production-](#_bookmark297) [ready features](#_bookmark297)。

**Part V. Spring Boot Actuator: Production-ready features**

Spring Boot包含许多附加功能，可帮助您在应用程序投入生产时监视和管理应用程序。 您可以选择使用HTTP端点，JMX甚至远程shell（SSH或Telnet）来管理和监视您的应用程序。 审计，健康和指标收集可以自动应用于您的应用程序。

执行器HTTP端点仅适用于基于Spring MVC的应用程序。 尤其是，除非启用Spring MVC，否则它将不能与Jersey一起工作[unless you enable Spring MVC as well.](#_bookmark532)。

# 启用 production-ready features

[spring-boot-actuator](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator) 模块提供了Spring Boot的所有生产就绪功能。 启用这些功能的最简单方法是在spring-boot-starter-actuator ‘Starter’上添加一个依赖项。

|  |
| --- |
| **Definition of Actuator**  致动器(actuator)是制造术语，指的是用于移动或控制某物的机械装置。 执行器可以从一个小的变化中产生大量的运动。 |

要将执行器添加到基于Maven的项目中，请添加以下“Starter”依赖项：

**<dependencies>**

**<dependency>**

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

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

**</dependency>**

**</dependencies>**

对于, 使用一下声明:

dependencies {

compile(***"org.springframework.boot:spring-boot-starter-actuator"***)

}

# Endpoints(端点)

执行器端点(Actuator endpoints)允许您监视和与您的应用程序进行交互。 Spring Boot包含许多内置的端点，您也可以添加自己的端点。 例如，健康端点提供基本的应用程序健康信息。

端点的暴露方式取决于您选择的技术类型。 大多数应用程序选择HTTP监视，其中端点的ID映射到一个URL。 例如，默认情况下，运行状况端点将映射到/ health。

以下技术不可知的端点可用：

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Sensitive Default** |
| actuator  auditevents | Provides a hypermedia-based “discovery page” for the other endpoints. Requires Spring HATEOAS to be on the classpath.  Exposes audit events information for the current application. | true  true |
| autoconfig | Displays an auto-configuration report showing all auto- configuration candidates and the reason why they ‘were’ or ‘were not’ applied. | true |
| beans | Displays a complete list of all the Spring beans in your application. | true |
| configprops | Displays a collated list of all @ConfigurationProperties. | true |
| dump env | Performs a thread dump.  Exposes properties from Spring’s  ConfigurableEnvironment. | true true |
| flyway | Shows any Flyway database migrations that have been applied. | true |
| health | Shows application health information (when the application is secure, a simple ‘status’ when accessed over an unauthenticated connection or full message details when authenticated). | false |
| info loggers | Displays arbitrary application info.  Shows and modifies the configuration of loggers in the application. | false true |
| liquibase | Shows any Liquibase database migrations that have been applied. | true |
| metrics | Shows ‘metrics’ information for the current application. | true |
| mappings | Displays a collated list of all @RequestMapping paths. | true |

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Sensitive Default** |
| shutdown | Allows the application to be gracefully shutdown (not enabled | true |
|  | by default). |  |
| trace | Displays trace information (by default the last 100 HTTP | true |
|  | requests). |  |

如果您使用Spring MVC，还可以使用以下附加端点：

|  |  |  |
| --- | --- | --- |
| **ID**  docs | **Description**  Displays documentation, including example requests and responses, for the Actuator’s endpoints. Requires spring- boot-actuator-docs to be on the classpath. | **Sensitive Default**  false |
| heapdump | Returns a GZip compressed hprof heap dump file. | true |
| jolokia | Exposes JMX beans over HTTP (when Jolokia is on the classpath). | true |
| logfile | Returns the contents of the logfile (if logging.file or logging.path properties have been set). Supports the use of the HTTP Range header to retrieve part of the log file’s content. | true |

**Note**

根据端点的暴露方式，敏感属性(sensitive property)可以用作安全提示。 例如，sensitive property在通过HTTP访问时将需要用户名/密码（或者如果未启用Web安全性，则只需将其禁用）。

## 自定义 endpoints

端点可以使用Spring属性进行自定义。 您可以更改端点是否已启用，是否敏感，甚至是否为其ID。

例如，这里是一个application.properties，它改变了bean的敏感度和id

端点，并启用关机。

**endpoints.beans.id**=springbeans **endpoints.beans.sensitive**=false **endpoints.shutdown.enabled**=true

**Note**

前缀#endpoints +. +name“用于唯一标识正在配置的端点。

默认情况下，除关闭以外的所有端点均已启用。 如果您希望专门“opt-in”端点启用，则可以使用endpoints.enabled属性。 例如，以下将禁用除info以外的所有端点：

**endpoints.enabled**=false **endpoints.info.enabled**=true

同样，您也可以选择全局设置所有端点的“sensitive”标志。 默认情况下，敏感标志取决于端点的类型（请参阅上表）。 例如，要将所有端点标记为除info之外的其他端点：

**endpoints.sensitive**=true **endpoints.info.sensitive**=false

## Hypermedia for actuator MVC endpoints

如果endpoints.hypermedia.enabled设置为true，并且[Spring HATEOAS](http://projects.spring.io/spring-hateoas) 位于类路径中（例如，通过spring-boot-starter-hateoas或者使用[Spring Data REST](http://projects.spring.io/spring-data-rest)），那么来自Actuator的HTTP端点将通过超媒体链接进行增强， 并添加一个“discovery page”链接到所有的端点。 默认情况下，“discovery page”在/执行器上可用。 它被实现为一个端点，允许使用属性来配置它的路径（endpoints.actuator.path），以及是否启用（endpoints.actuator.enabled）。

当定制管理上下文路径被配置时，“discovery page”将自动从/执行器移动到管理上下文的根。 例如，如果管理上下文路径是/ management，则发现页面将从/ management可用。

如果[HAL Browser](https://github.com/mikekelly/hal-browser)通过其webjar（org.webjars：hal-browser）或通过spring-data-rest-hal-browser在类路径上，那么以HAL浏览器形式的HTML“discovery page” 还提供了。

## 跨域支持(CORS support)

跨源资源共享（[Cross-origin resource sharing](http://en.wikipedia.org/wiki/Cross-origin_resource_sharing) CORS）是[W3C specification](http://www.w3.org/TR/cors/) 规范，允许您以灵活的方式指定哪种跨域请求被授权。 执行器的MVC端点可以配置为支持这种情况。

CORS支持在默认情况下是禁用的，只有在设置了endpoints.cors.allowed- origins属性后才能启用。 下面的配置允许来自example.com域的GET和POST调用：

**endpoints.cors.allowed-origins**[=http://example.com](http://example.com/) **endpoints.cors.allowed-methods**=GET,POST

**Tip**

检查[EndpointCorsProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/EndpointCorsProperties.java) 以获取完整的选项列表。

## 添加自定义 endpoints

如果添加了Endpoint类型的@Bean，那么它将自动通过JMX和HTTP（如果有可用的服务器）公开。 可以通过创建MvcEndpoint类型的bean来进一步定制HTTP端点。 您的MvcEndpoint不是@Controller，但可以使用@RequestMapping（和@ Managed \*）来公开资源。

**Tip**

如果你正在做一个库特性，可以考虑在@ org.springframework.boot.actuate.autoconfigure.ManagementContextConfiguration项下添加一个用@ManagementContextConfiguration注解的配置类到/META-INF/spring.factories。

如果你这样做，那么如果你的用户要求一个单独的管理端口或地址，端点将会移动到所有其他MVC端点的子上下文中。 如果要将静态资源（例如）添加到管理端点，则以这种方式声明的配置可以是WebConfigurerAdapter。

## 健康信息(Health information)

健康信息可用于检查正在运行的应用程序的状态。 如果生产系统停机，它通常被监控软件用来提醒某人。 运行状况端点公开的默认信息取决于访问方式。 对于安全应用程序中的未经身份验证的连接，将返回一个简单的“status”消息，对于已验证的连接，还会显示其他详细信息（有关HTTP详细信息，请参见[Section 48.7, “HTTP health endpoint format and access restrictions”](#_bookmark321) ）。

运行状况信息从ApplicationContext中定义的所有[HealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/HealthIndicator.java) beans中收集。 Spring Boot包含许多自动配置的HealthIndicators，您也可以编写自己的。 缺省情况下，最终系统状态由HealthAggregator派生，HealthAggregator根据状态的有序列表对每个HealthIndicator的状态进行排序。 排序列表中的第一个状态用作总体健康状态。 如果没有HealthIndicator返回HealthAggregator已知的状态，则使用UNKNOWN状态。

## (安全与健康指标)Security with HealthIndicators

由HealthIndicators返回的信息本质上通常有些敏感。 例如，您可能不想将数据库服务器的详细信息发布到世界上。 出于这个原因，默认情况下，只有健康状态暴露在未经身份验证的HTTP连接上。 如果您对完整的健康信息总是感到高兴，您可以将endpoints.health.sensitive设置为false。

还可以缓存健康响应以防止“拒绝服务”攻击。 如果要更改1000毫秒的默认缓存周期，请使用endpoints.health.time-to-live属性。

### Auto-configured HealthIndicators

以下HealthIndicators在适当情况下由Spring Boot自动配置：

| **Name** | **Description** |
| --- | --- |
| [CassandraHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/CassandraHealthIndicator.java) | Checks that a Cassandra database is up. |
| [DiskSpaceHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/DiskSpaceHealthIndicator.java) | Checks for low disk space. |
| [DataSourceHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/DataSourceHealthIndicator.java) | Checks that a connection to DataSource can be obtained. |
| [ElasticsearchHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/ElasticsearchHealthIndicator.java) | Checks that an Elasticsearch cluster is up. |
| [JmsHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/JmsHealthIndicator.java) | Checks that a JMS broker is up. |
| [MailHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/MailHealthIndicator.java) | Checks that a mail server is up. |
| [MongoHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/MongoHealthIndicator.java) | Checks that a Mongo database is up. |
| [RabbitHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/RabbitHealthIndicator.java) | Checks that a Rabbit server is up. |
| [RedisHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/RedisHealthIndicator.java) | Checks that a Redis server is up. |
| [SolrHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/SolrHealthIndicator.java) | Checks that a Solr server is up. |

**Tip**

可以使用management.health.defaults.enabled属性全部禁用它们。

### 编写自定义 HealthIndicators

要提供自定义健康信息，您可以注册实现HealthIndicator接口的Spring bean。 您需要提供health（）方法的实现并返回健康响应。 健康响应应该包括一个状态，并可以选择包含额外的细节来显示。

**import** org.springframework.boot.actuate.health.Health;

**import** org.springframework.boot.actuate.health.HealthIndicator;

**import** org.springframework.stereotype.Component;

@Component

**public class** MyHealthIndicator **implements** HealthIndicator {

@Override

**public** Health health() {

**int** errorCode = check(); *// perform some specific health check*

**if** (errorCode != 0) {

**return** Health.down().withDetail(***"Error Code"***, errorCode).build();

}

**return** Health.up().build();

}

}

**Note**

给定的HealthIndicator的标识符是没有HealthIndicator后缀的Bean的名称（如果存在）。 在上面的示例中，健康信息将在名为my的条目中可用。

除了Spring Boot的预定义[Status](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/Status.java) 类型之外，运行状况还可以返回代表新系统状态的自定义状态。 在这种情况下，还需要提供[HealthAggregator](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/HealthAggregator.java) 接口的自定义实现，或者必须使用management.health.status.order配置属性配置默认实现。

例如，假设您的一个HealthIndicator实现中正在使用代码为FATAL的新状态。 要配置严重性顺序，请将以下内容添加到应用程序属性中：

**management.health.status.order**=FATAL, DOWN, OUT\_OF\_SERVICE, UNKNOWN, UP

响应中的HTTP状态代码反映整体健康状况（例如，UP映射到200，OUT\_OF\_SERVICE或从DOWN到503）。 如果您通过HTTP访问运行状况端点，则可能还需要使用HealthMvcEndpoint注册自定义状态映射。 例如，以下映射FATAL到503（服务不可用）：

**endpoints.health.mapping.FATAL**=503

内置状态的默认状态映射如下：

|  |  |
| --- | --- |
| **Status**  DOWN | **Mapping**  SERVICE\_UNAVAILABLE (503) |
| OUT\_OF\_SERVICE | SERVICE\_UNAVAILABLE (503) |
| UP  UNKNOWN | No mapping by default, so http status is 200  No mapping by default, so http status is 200 |

## Application information

应用程序信息公开从ApplicationContext中定义的所有[InfoContributor](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/InfoContributor.java) beans收集的各种信息。 Spring Boot包含许多自动配置的InfoContributor，您也可以编写自己的。

### Auto-configured InfoContributors

以下InfoContributor可在适当情况下由Spring Boot自动配置：

| **Name** | **Description** |
| --- | --- |
| [EnvironmentInfoContributor](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/EnvironmentInfoContributor.java) | Expose any key from the Environmentunder the info key. |
| [GitInfoContributor](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/GitInfoContributor.java) | Expose git information if a git.properties file is available. |
| [BuildInfoContributor](https://github.com/spring-projects/spring-boot/tree/v1.5.9.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/BuildInfoContributor.java) | Expose build information if a META-INF/build-info.properties file is available. |

**Tip**

可以使用management.info.defaults.enabled属性来禁用它们。

### 自定义 application info information

您可以通过设置info.\* Spring属性来自定义信息端点公开的数据。 信息键下的所有环境属性将被自动公开。 例如，您可以将以下内容添加到您的application.properties中：

**info.app.encoding**=UTF-8 **info.app.java.source**=1.8 **info.app.java.target**=1.8

**Tip**

而不是硬编码(hardcoding)这些值，你也可以在构建时展开信息属性([expand info properties at build time](#_bookmark446))。

假设你正在使用Maven，你可以重写上面的例子，如下所示：

[**info.app.encoding**=@project.build.sourceEncoding@](mailto:info.app.encoding%3D@project.build.sourceEncoding@) [**info.app.java.source**=@java.version@](mailto:info.app.java.source%3D@java.version@) [**info.app.java.target**=@java.version@](mailto:info.app.java.target%3D@java.version@)

### Git commit information

info端点的另一个有用的功能是在构建项目时发布有关git源代码库状态的信息。 如果一个GitProperties bean可用，那么git.branch，git.commit.id和git.commit.time属性将被公开。

**Tip**

如果git.properties文件在类路径的根目录中可用，则GitProperties bean将自动配置。 有关更多详细信息，请参阅生成git信息([Generate git information](#_bookmark549) )。

如果要显示完整的git信息（即git.properties的完整内容），请使用

management.info.git.mode属性：

**management.info.git.mode**=full

### Build information

如果BuildProperties bean可用，info端点还可以发布有关您的构建的信息。 如果META-INF / build-info.properties文件在类路径中可用，则会发生这种情况。

**Tip**

Maven和Gradle插件都可以生成该文件，请参阅生成构建信息([Generate build information](#_bookmark548) )以获取更多详细信息。

### 编写自定义 InfoContributors

为了提供定制的应用程序信息，你可以注册实现了的Spring bean

[InfoContributor](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/InfoContributor.java) 接口。

下面的例子提供了一个具有单个值的示例条目：

**import** java.util.Collections;

**import** org.springframework.boot.actuate.info.Info;

**import** org.springframework.boot.actuate.info.InfoContributor;

**import** org.springframework.stereotype.Component;

@Component

**public class** ExampleInfoContributor **implements** InfoContributor {

@Override

**public void** contribute(Info.Builder builder) { builder.withDetail(***"example"***,

Collections.singletonMap(***"key"***, ***"value"***));

}

}

如果您点击信息端点，您应该会看到一个包含以下附加条目的响应：

**{**

***"example"***: **{**

***"key"*** : ***"value"***

**}**

**}**

# Monitoring and management over HTTP

如果您正在开发Spring MVC应用程序，则Spring Boot Actuator将自动配置所有启用的端点以通过HTTP进行公开。 默认约定是使用端点的id作为URL路径。 例如，暴露 /health。

## Accessing sensitive endpoints

默认情况下，所有sensitive的HTTP端点都是安全的，只有具有ACTUATOR角色的用户可以访问它们。 安全性是使用标准的HttpServletRequest.isUserInRole方法强制执行的。

**Tip**

如果您需要与ACTUATOR不同的东西，请使用management.security.roles属性。

如果您在防火墙后面部署应用程序，则可能希望无需验证即可访问您的所有执行器端点。 您可以通过更改management.security.enabled属性来完成此操作：

**application.properties.**

**management.security.enabled**=false

**Note**

默认情况下，执行器端点暴露在提供常规HTTP通信的相同端口上。 注意不要在更改management.security.enabled属性时意外暴露敏感信息。

如果您公开部署应用程序，则可能需要添加“Spring Security”来处理用户身份验证。 当添加“Spring Security”时，默认情况下，“basic”身份验证将与用户名用户和生成的密码（应用程序启动时打印在控制台上）一起使用。

**Tip**

生成的密码在应用程序启动时被记录。 搜索“Using default security password”。

您可以使用Spring属性来更改用户名和密码，并更改访问端点所需的安全角色。 例如，您可以在application.properties中设置以下内容：

**security.user.name**=admin **security.user.password**=secret **management.security.roles**=SUPERUSER

如果您的应用程序具有自定义安全配置，并且您希望所有执行器端点无需身份验证即可访问，则需要在安全配置中明确配置该端点。 除此之外，您需要将management.security.enabled属性更改为false。

如果您的自定义安全配置保护您的执行器端点，则还需要确保经过身份验证的用户具有在management.security.roles下指定的角色。

**Tip**

如果您没有向未经身份验证的用户公开基本健康信息的用例，并且已经使用自定义安全保护了执行程序端点，则可以将management.security.enabled设置为false。 这将通知Spring Boot跳过额外的角色检查。

如果您没有向未经身份验证的用户公开基本健康信息的用例，并且已经使用自定义安全保护了执行程序端点，则可以将management.security.enabled设置为false。 这将通知Spring Boot跳过额外的角色检查。

## Customizing the management endpoint paths

有时将所有管理端点分组在单一路径下是有用的。 例如，您的应用程序可能已经使用/info作为其他目的。 您可以使用management.context- path属性为管理端点设置前缀：

**management.context-path**=/manage

上面的application.properties示例会将终端从/{id} to /manage/{id}（例如/manage/info）。

**Note**

除非已将管理端口配置为[expose endpoints using a different HTTP](#_bookmark317) [port](#_bookmark317)，否则management.context-path与server.context-path相关。

您还可以更改端点的“id”（使用endpoints.{name}.id），然后更改MVC端点的默认资源路径。 法定端点ID仅由字母数字字符组成（因为它们可能会暴露在许多地方，包括禁止使用特殊字符的JMX对象名称）。 MVC路径可以通过配置endpoints.{name}.path单独更改，并且对这些值没有验证（因此您可以使用URL路径中合法的任何内容）。 例如，要将/health端点的位置更改为/ping/me，可以设置endpoints.health.path=/ping/me。

**Note**

即使端点路径是单独配置的，它仍然相对于management.context- path。

**Tip**

如果您提供自定义的MvcEndpoint，请记住包含一个可设置的路径属性，如果您希望代码的行为与标准MVC端点类似，则默认为/{id}。 （看看HealthMvcEndpoint，看看你可能会这样做）。如果你的自定义端点是一个端点（而不是MvcEndpoint），那么Spring Boot将为你处理路径。

## Customizing the management server port

使用默认的HTTP端口公开管理端点是基于云的部署的明智选择。 但是，如果您的应用程序在您自己的数据中心内运行，则可能希望使用不同的HTTP端口公开端点。

可以使用management.port属性来更改HTTP端口。

**management.port**=8081

由于您的管理端口通常受防火墙保护，并且不向公众公开，即使您的主应用程序是安全的，您也可能不需要管理端点上的安全性。 在这种情况下，您将在类路径上具有Spring Security，并且可以像这样禁用管理安全性：

**management.security.enabled**=false

（如果你在类路径上没有Spring Security，那么就不需要像这样明确地禁用管理安全性，甚至可能会破坏应用程序。）

## Configuring management-specific SSL

配置为使用自定义端口时，管理服务器也可以使用各种management.ssl.\*属性配置自己的SSL。 例如，这允许管理服务器通过HTTP可用，而main application使用HTTPS：

**server.port**=8443 **server.ssl.enabled**=true

**server.ssl.key-store**=classpath:store.jks **server.ssl.key-password**=secret **management.port**=8080 **management.ssl.enabled**=false

或者，主服务器和管理服务器都可以使用SSL，但使用不同的密钥存储区：

**server.port**=8443 **server.ssl.enabled**=true

**server.ssl.key-store**=classpath:main.jks **server.ssl.key-password**=secret **management.port**=8080 **management.ssl.enabled**=true

**management.ssl.key-store**=classpath:management.jks **management.ssl.key-password**=secret

## Customizing the management server address

您可以通过设置management.address属性来自定义管理端点可用的地址。 如果您只想在内部网络或运行网络上进行监听，或者仅监听来localhost的连接，这会非常有用。

**Note**

如果端口与主服务器端口不同，则只能监听其他地址。

这是一个不允许远程管理连接的示例application.properties：

**management.port**=8081 **management.address**=127.0.0.1

## Disabling HTTP endpoints

如果您不想通过HTTP公开端点，则可以将管理端口设置为-1：

**management.port**=-1

## HTTP health endpoint format and access restrictions

健康端点公开的信息取决于是否匿名访问，以及封闭应用程序是否安全。 默认情况下，当在安全应用程序中匿名访问时，隐藏有关服务器健康状况的任何细节，端点将简单指示服务器是启动还是关闭。 此外，响应被缓存了一个可配置的时间段，以防止端点被用于拒绝服务攻击。 endpoints.health.time-to-live属性用于配置以毫秒为单位的缓存周期。 它默认为1000，即一秒。

汇总的HTTP响应示例（匿名请求的缺省值）：

$ curl -i localhost:8080/health HTTP/1.1 200

X-Application-Context: application

Content-Type: application/vnd.spring-boot.actuator.v1+json;charset=UTF-8 Content-Length: 15

{"status":"UP"}

汇总状态为“DOWN”的HTTP响应示例（请注意503状态码）：

$ curl -i localhost:8080/health HTTP/1.1 503

X-Application-Context: application

Content-Type: application/vnd.spring-boot.actuator.v1+json;charset=UTF-8 Content-Length: 17

{"status":"DOWN"}

示例详细的HTTP响应：

$ curl -i localhost:8080/health HTTP/1.1 200 OK

X-Application-Context: application

Content-Type: application/vnd.spring-boot.actuator.v1+json;charset=UTF-8 Content-Length: 221

{

"status" : "UP", "diskSpace" : {

"status" : "UP", "total" : 63251804160,

"free" : 31316164608,

"threshold" : 10485760

},

"db" : {

"status" : "UP",

"database" : "H2", "hello" : 1

}

}

可以增强上述限制，从而仅允许经认证的用户完全访问安全应用程序中的健康端点。 为此，请将endpoints.health.sensitive设置为true。 以下是行为总结（缺省敏感标志值“false”以粗体显示）：

|  |  |  |
| --- | --- | --- |
| **management.securit** | **ye.nednpaobilnetds.health.seUnsniauttihveenticated** | **Authenticated (with right role)** |
| false | \* Full content | Full content |

|  |  |  |
| --- | --- | --- |
| **management.securit** | **ye.nednpaobilnetds.health.seUnsniauttihveenticated** | **Authenticated (with right role)** |
| true  true | **false** Status only  true No content | Full content  Full content |

# Monitoring and management over JMX

Java管理扩展（Java Management Extensions-JMX）提供了一个标准机制来监视和管理应用程序。 默认情况下，Spring Boot会在org.springframework.boot域下(domain)将管理端点公开为JMX MBean。

## Customizing MBean names

MBean的名称通常由端点的ID生成。 例如健康端点显示为：

org.springframework.boot/Endpoint/healthEndpoint。

如果您的应用程序包含多个Spring ApplicationContext，您可能会发现名称冲突。 要解决此问题，可以将endpoints.jmx.unique-names属性设置为true，以便MBean名称始终是唯一的。

您还可以自定义公开端点的JMX域。 这是一个例子

application.properties：

**endpoints.jmx.domain**=myapp **endpoints.jmx.unique-names**=true

## Disabling JMX endpoints

如果您不想通过JMX公开端点，则可以将endpoints.jmx.enabled属性设置为false：

**endpoints.jmx.enabled**=false

## Using Jolokia for JMX over HTTP

Jolokia是一个JMX-HTTP桥梁，它提供了访问JMX bean的另一种方法。 要使用Jolokia，只需将依赖关系包含到org.jolokia：jolokia-core。 例如，使用Maven你可以添加以下内容：

**<dependency>**

**<groupId>**org.jolokia**</groupId>**

**<artifactId>**jolokia-core**</artifactId>**

**</dependency>**

Jolokia然后可以访问您的管理HTTP服务器上的/jolokia。

### Customizing Jolokia

Jolokia有许多你通常使用servlet参数进行配置的设置。 使用Spring Boot，你可以使用你的application.properties，只需在jolokia.config前加参数。

**jolokia.config.debug**=true

### Disabling Jolokia

如果您使用的是Jolokia，但您不希望Spring Boot进行配置，只需设置

endpoints.jolokia.enabled属性为false：

**endpoints.jolokia.enabled**=false

# Monitoring and management using a remote shell (deprecated弃用)

Spring Boot支持一个名为“CRaSH”的集成Java外壳。 您可以使用CRaSH ssh或telnet到您正在运行的应用程序。 要启用远程shell支持，请将以下依赖项添加到您的项目中：

**<dependency>**

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

**<artifactId>**spring-boot-starter-remote-shell**</artifactId>**

**</dependency>**

**Note**

远程shell已被弃用，并将在Spring Boot 2.0中被删除。

**Tip**

如果你还想启用telnet访问，你还需要依赖 org.crsh：crsh.shell.telnet。

**Note**

CRaSH需要在运行中编译命令时使用JDK运行。 如果基本帮助命令失败，那么您可能正在运行JRE。

## Connecting to the remote shell

默认情况下，远程shell将侦听端口2000上的连接。默认用户是user，默认密码将随机生成并显示在日志输出中。 如果您的应用程序使用Spring Security，则默认情况下，shell将使用相同的配置([the same configuration](#_bookmark156) )。 如果没有，将应用一个简单的身份验证，你应该看到这样的消息：

Using default password for shell access: ec03e16c-4cf4-49ee-b745-7c8255c1dd7e

Linux和OSX用户可以使用ssh连接到远程shell，Windows用户可以下载并安装[PuTTY](http://www.putty.org/)。

/\\ / '\_ \_ \_(\_)\_ \_ \ \ \ \ ( ( )\ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ )| |\_)| | | | | || (\_| | ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | / / / /

=========|\_|==============| /=/\_/\_/\_/

:: Spring Boot :: (v1.5.8.RELEASE) on myhost

\_ \_

$ ssh -p 2000 user@localhost

user@localhost's password:

. \_

键入帮助以获取命令列表。 Spring Boot提供了metrics, beans, autoconfig and endpoint

commands

### Remote shell credentials(证书)

您可以使用management.shell.auth.simple.user.name和management.shell.auth.simple.user.password属性来配置自定义连接凭证。 也可以使用“Spring Security”AuthenticationManager来处理登录职责。 有关完整的详细信息，请参阅[CrshAutoConfiguration](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/CrshAutoConfiguration.html) 和 [ShellProperties](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/ShellProperties.html) 。

## Extending the remote shell

远程shell可以用很多有趣的方式进行扩展。

### Remote shell commands

您可以使用Groovy编写额外的shell命令（有关详细信息，请参阅CRaSH文档）。 由于CRaSH的Java编译器的限制，不支持用Java编写的命令。 默认情况下，Spring Boot将在以下位置搜索命令：

* classpath\*:/commands/\*\*
* classpath\*:/crash/commands/\*\*

**Tip**

您可以通过设置shell.command-path-patterns属性来更改搜索路径。

**Note**

如果您使用的是可执行文件，shell命令所依赖的任何类都必须封装在嵌套的jar文件中，而不是直接放在可执行文件jar或war中。

这是一个简单的“hello”命令，可以从 src/main/resources/commands/ hello.groovy中加载：

**package** commands

**import** org.crsh.cli.Command

**import** org.crsh.cli.Usage

**import** org.crsh.command.InvocationContext

**class** hello {

@Usage("Say Hello") @Command

def main(InvocationContext context) {

**return *"Hello"***

}

}

Spring Boot为InvocationContext添加了一些可以从命令访问的附加属性：

|  |
| --- |
| **Attribute Name Description** |
| spring.boot.version The version of Spring Boot  spring.version The version of the core Spring Framework |

|  |
| --- |
| **Attribute Name Description** |
| spring.beanfactory Access to the Spring BeanFactory  spring.environment Access to the Spring Environment |

### Remote shell plugins

除了新命令之外，还可以扩展其他CRaSH shell功能。 所有扩展了org.crsh.plugin.CRaSHPlugin的Spring Bean都会自动注册到shell中。

有关更多信息，请参阅[CRaSH reference documentation](http://www.crashub.org/)。

# Loggers

Spring Boot Actuator能够在运行时查看和配置应用程序的日志级别。 您可以查看整个列表或单个记录器的配置，该配置由显式配置的日志记录级别以及记录框架给出的有效日志记录级别组成。 这些级别可以是：

* TRACE
* DEBUG
* INFO
* WARN
* ERROR
* FATAL
* OFF
* null

用null表示没有明确的配置。

## 配置 Logger

为了配置一个给定的记录器，可以将一个部分实体发布到资源的URI：

**{**

***"configuredLevel"***: ***"DEBUG"***

**}**

# Metrics(度量)

Spring Boot Actuator包含一个带‘gauge’ 和 ‘counter’支持的度量服务。 “‘gauge”记录一个单一的值; 和“counter”记录增量（增量或减量）。 Spring Boot Actuator还提供了一个[PublicMetrics](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/endpoint/PublicMetrics.java) 接口，您可以使用该接口公开您通过这两种机制之一无法记录的度量标准。 以[SystemPublicMetrics](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/endpoint/SystemPublicMetrics.java) 为例。

所有HTTP请求的度量(Metrics)标准都会自动记录下来，所以如果你点击度量(Metrics)标准端点，你应该看到类似这样的响应：

**{**

***"counter.status.200.root"***: 20**,**

***"counter.status.200.metrics"***: 3**,**

***"counter.status.200.star-star"***: 5**,**

***"counter.status.401.root"***: 4**,**

***"gauge.response.star-star"***: 6**,**

***"gauge.response.root"***: 2**,**

***"gauge.response.metrics"***: 3**,**

***"classes"***: 5808**,**

***"classes.loaded"***: 5808**,**

***"classes.unloaded"***: 0**,**

***"heap"***: 3728384**,**

***"heap.committed"***: 986624**,**

***"heap.init"***: 262144**,**

***"heap.used"***: 52765**,**

***"nonheap"***: 0**,**

***"nonheap.committed"***: 77568**,**

***"nonheap.init"***: 2496**,**

***"nonheap.used"***: 75826**,**

***"mem"***: 986624**,**

***"mem.free"***: 933858**,**

***"processors"***: 8**,**

***"threads"***: 15**,**

***"threads.daemon"***: 11**,**

***"threads.peak"***: 15**,**

***"threads.totalStarted"***: 42**,**

***"uptime"***: 494836**,**

***"instance.uptime"***: 489782**,**

***"datasource.primary.active"***: 5**,**

***"datasource.primary.usage"***: 0.25

**}**

在这里，我们可以看到基本的内存，堆，类加载，处理器和线程池信息以及一些HTTP指标。 在这种情况下，根（'/'）和/metrics URL分别返回了20次和200次HTTP 200响应。 它也似乎是根URL返回HTTP 401（未经授权）4次。 双星号（star-star）来自与Spring MVC匹配的请求/ \*\*（通常是静态资源）。

仪表(gauge)显示请求的最后响应时间。 所以最后的根请求需要2ms才能做出响应，而最后的/指标需要3ms。

**Note**

在这个例子中，我们实际上是使用/ metrics URL通过HTTP访问端点，这就解释了为什么度量(metrics)出现在响应中。

## System metrics

Spring Boot公开以下系统度量(metrics)标准：

* The total system memory in KB (mem)
* The amount of free memory in KB (mem.free)
* The number of processors (processors)
* The system uptime in milliseconds (uptime)
* The application context uptime in milliseconds (instance.uptime)
* The average system load (systemload.average)
* Heap information in KB (heap, heap.committed, heap.init, heap.used)
* Thread information (threads, thread.peak, thread.daemon)
* Class load information (classes, classes.loaded, classes.unloaded)
* Garbage collection information (gc.xxx.count, gc.xxx.time)

## DataSource metrics

针对应用程序中定义的每个受支持的DataSource公开以下度量标准：

* The number of active connections (datasource.xxx.active)
* The current usage of the connection pool (datasource.xxx.usage).

所有数据源度量标准共享数据源(datasource)。 字首。 前缀进一步限定为每个数据源：

* If the data source is the primary data source (that is either the only available data source or the one flagged @Primary amongst the existing ones), the prefix is datasource.primary.
* If the data source bean name ends with DataSource, the prefix is the name of the bean without

DataSource (i.e. datasource.batch for batchDataSource).

* In all other cases, the name of the bean is used.

可以通过注册具有自定义版本的DataSourcePublicMetrics的bean来覆盖部分或全部这些默认值。 默认情况下，Spring Boot为所有支持的数据源提供元数据; 你可以添加额外的DataSourcePoolMetadataProvider bean，如果你最喜欢的数据源不支持开箱即用。 有关示例，请参阅DataSourcePoolMetadataProvidersConfiguration。

## Cache metrics

针对您的应用程序中定义的每个受支持的缓存，公开以下指标：

* The current size of the cache (cache.xxx.size)
* Hit ratio (cache.xxx.hit.ratio)
* Miss ratio (cache.xxx.miss.ratio)

**Note**

高速缓存提供程序不会以一致的方式显示命中/错过率。 虽然有些揭露了合计值（即自从上一次

统计数据被清除后的命中率），其他值显示时间值（即最后一秒的命中率）。 查看你的缓存提

供者文档以获取更多细节。

如果两个不同的缓存管理器碰巧定义了相同的缓存，则缓存的名称前面加上CacheManager bean的名称。

通过注册具有自定义版本的CachePublicMetrics的bean，可以覆盖部分或全部这些默认值。 默认情况下，Spring Boot为EhCache，Hazelcast，Infinispan，JCache和Guava提供缓存统计信息。 如果您最喜爱的缓存库不支持开箱即用，您可以添加额外的CacheStatisticsProvider bean。 有关示例，请参阅CacheStatisticsAutoConfiguration。

## Tomcat session metrics

如果您使用Tomcat作为嵌入式servlet容器，session metrics将自动公开。 httpsessions.active和httpsessions.max键提供活动会话数和最大会话数。

## Recording your own metrics

要记录自己的指标，请在您的bean中注入一个[CounterService](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/metrics/CounterService.java) and/or [GaugeService](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/metrics/GaugeService.java) 。 CounterService公开增量，减量和重置方法; GaugeService提供了一个提交方法。

下面是一个简单的例子，它计算一个方法被调用的次数：

**import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.boot.actuate.metrics.CounterService; **import** org.springframework.stereotype.Service;

@Service

**public class** MyService {

**private final** CounterService counterService; @Autowired

**public** MyService(CounterService counterService) {

**this**.counterService = counterService;

}

**public void** exampleMethod() {

**this**.counterService.increment(***"services.system.myservice.invoked"***);

}

}

**Tip**

您可以使用任何字符串作为度量标准名称，但是您应该遵循所选store/ graphing技术的指导原则。 有关Graphite的一些很好的指导可以在[Matt Aimonetti’s Blog](http://matt.aimonetti.net/posts/2013/06/26/practical-guide-to-graphite-monitoring/)上找到。

## Adding your own public metrics

要添加每次调用度量标准端点时计算的附加度量标准，只需注册其他的PublicMetrics实现bean。 默认情况下，所有这些bean都由端点收集。 您可以通过定义您自己的MetricsEndpoint来轻松地更改这一点。

## Special features with Java 8

Spring Boot提供的GaugeService和CounterService的默认实现取决于您使用的Java版本。 使用Java 8（或更高的版本），实现切换到为快速写入而优化的高性能版本，由原子内存缓冲区支持，而不是由一成不变但相对昂贵的Metric <？>类型（计数器大约快5倍 测量的速度大约是基于存储库的实现速度的两倍）。 即使对于Java 7（它们具有某些Java 8并发库的反向端口），Dropwizard度量服务（见下文）也非常有效，但它们不记录度量值的时间戳。 如果衡量指标收集的性能是一个问题，那么总是建议使用高性能选项之一，而且也只是偶尔读取度量标准，以便写入在本地进行缓冲，并且只在需要时才读取。

**Note**

如果您使用的是Java 8或者如果使用的是Dropwizard指标(metrics)，则默认不使用旧的MetricRepository及其InMemoryMetricRepository实现。

## Metric writers, exporters and aggregation

Spring Boot提供了一个名为Exporter的标记接口(marker interface)的实现，它可以将度量读数从内存缓冲区复制到可以分析和显示的地方。 事实上，如果您提供实现MetricWriter接口的@Bean（或简单用例的GaugeWriter）并将其标记为@ExportMetricWriter，那么它将自动连接到一个导出器，并每5秒钟提供一次度量更新（通过spring.metrics.export.delay-millis）。 另外，您定义并标记为@ExportMetricReader的任何MetricReader将由默认导出器导出其值。

**Note**

此功能在您的应用程序中启用计划（@EnableScheduling），如果您运行集成测试，则您的计划任务将启动，这可能会造成问题。 您可以通过将spring.metrics.export.enabled设置为false来禁用此行为。

默认导出器是一个MetricCopyExporter，通过不复制自上次调用以来未更改过的值（可以使用标志spring.metrics.export.send-latest关闭优化）来尝试优化自身。 另请注意，Dropwizard MetricRegistry不支持时间戳，因此如果使用Dropwizard度量标准（所有度量标准都将在每个标记上复制），优化将不可用。

导出触发器(trigger)的默认值（delay-millis，include，excluded和latest-latest）可以设置为spring.metrics.export.\*。特定MetricWriters的单个值可以设置为spring.metrics.export.triggers.<name>.\*其中<name>是一个bean名称（或匹配bean名称的模式）。

**Warning**

如果关闭默认的MetricRepository（例如使用Dropwizard度量标准），则会自动导出度量标准。 你可以得到相同的功能，声明一个你自己的MetricReader类型的bean，并声明它是@ExportMetricReader。

### Example: Export to Redis

如果您提供了RedisMetricRepository类型的@Bean并将其标记为@ExportMetricWriter，则度量标准将导出到Redis缓存以进行聚合。 RedisMetricRepository有两个重要的参数来配置它：prefix和key（传递给它的构造函数）。 最好使用对应用程序实例唯一的前缀（例如，使用随机值，也可以使用应用程序的逻辑名称，以便与同一应用程序的其他实例关联）。 “key”用于保留所有度量名称的全局索引，因此它应该是唯一的“globally”，无论这对您的系统意味着什么（例如，如果具有不同的键，同一系统的两个实例可以共享一个Redis缓存）。

Example:

@Bean @ExportMetricWriter

MetricWriter metricWriter(MetricExportProperties export) {

**return new** RedisMetricRepository(connectionFactory, export.getRedis().getPrefix(), export.getRedis().getKey());

}

**application.properties.**

**spring.metrics.export.redis.prefix**: metrics.mysystem.${spring.application.name:application}.

${random.value:0000}

**spring.metrics.export.redis.key**: keys.metrics.mysystem

前缀由最后的应用程序名称和ID构成，所以稍后可以很容易地用来标识具有相同逻辑名称的一组进程。

**Note**

设置密钥(key)和前缀(prefix)非常重要。 密钥用于所有存储库操作，并可由多个存储库共享。 如果多个存储库共享一个密钥（就像在需要聚合的情况下那样），那么通常有一个只读的“master”存储库，它有一个简短但可识别的前缀（如“metrics.mysystem”）， 和许多只有以前缀开头的前缀的只写存储库（例如上面的例子中的metrics.mysystem。\*）。 从这样的“主”存储库中读取所有密钥是有效的，但是读取具有较长前缀的子集（例如，使用其中一个写入存储库）效率低下。

**Tip**

上面的例子使用MetricExportProperties来注入和提取密钥和前缀。 这是作为一个方便的spring boot提供给你，配置合理的默认值。 只要遵循建议，没有什么可以阻止你使用自己的values。

### Example: Export to Open TSDB

如果您提供OpenTsdbGaugeWriter类型的@Bean并将其标记为@ExportMetricWriter度量标准将导出到[Open TSDB](http://opentsdb.net/) 进行聚合。 OpenTsdbGaugeWriter具有您需要设置为打开TSDB“/ put”端点的url属性，例如，本地localhost:4242/api/put）。 它还有一个命名策略，您可以自定义或配置这些策略，以使指标与您在服务器上需要的数据结构相匹配。 默认情况下，它只是将度量标准名称作为Open TSDB度量标准名称进行传递，并将标记“domain”（值为“org.springframework.metrics”）和“process”（其值等于命名的对象哈希值 战略）。 因此，在运行应用程序并生成一些指标之后，您可以检查TSD UI中的指标（默认为localhost:4242）。

Example:

curl localhost:4242/api/query?start=1h-ago&m=max:counter.status.200.root [

{

"metric": "counter.status.200.root", "tags": {

"domain": "org.springframework.metrics", "process": "b968a76"

},

"aggregateTags": [], "dps": {

"1430492872": 2,

"1430492875": 6

}

}

]

### Example: Export to Statsd

要将度量值导出到Statsd，首先确保已经添加了com.timgroup:java-statsd- client作为项目的依赖项（Spring Boot为其提供依赖项管理）。 然后添加一个spring.metrics.export.statsd.host值到你的application.properties文件。 连接将打开到端口8125，除非提供了spring.metrics.export.statsd.port覆盖。 如果你想要一个自定义的前缀，你可以使用spring.metrics.export.statsd.prefix。

或者，您可以提供StatsdMetricWriter类型的@Bean并将其标记为@ExportMetricWriter：

@Value("${spring.application.name:application}.${random.value:0000}")

**private** String prefix = ***"metrics"***;

@Bean @ExportMetricWriter

MetricWriter metricWriter() {

**return new** StatsdMetricWriter(prefix, ***"localhost"***, 8125);

}

### Example: Export to JMX

如果您提供了标记为@ExportMetricWriter的JmxMetricWriter类型的@Bean，则将度量标准作为MBean导出到本地服务器（只要打开，MBeanExporter就由Spring Boot JMX自动配置提供）。 然后，可以使用任何了解JMX的工具（例如JConsole或JVisualVM）来inspected, graphed, alerted等。

Example:

@Bean @ExportMetricWriter

MetricWriter metricWriter(MBeanExporter exporter) {

**return new** JmxMetricWriter(exporter);

}

每个度量都作为单独的MBean导出。 ObjectNames的格式由ObjectNamingStrategy提供，可以注入到JmxMetricWriter中（默认情况下，分割度量(period-separated)标准名称，并以JVisualVM或JConsole中的度量值组方式很好地标记前两个以句点分隔的部分）。

## Aggregating metrics from multiple sources

有一个AggregateMetricReader可用于合并来自不同物理来源的指标。 相同逻辑度量的来源只需要用句点分隔的前缀来发布它们，阅读器就会聚合（通过截断度量名称和删除前缀）。 计数器被汇总，其他所有数据（例如量表）都将采用最新的值。

如果将多个应用程序实例提供给中央(central)（例如Redis）存储库并且要显示结果，这非常有用。 特别建议与MetricReaderPublicMetrics配合使用，以便将结果连接到“/ metrics”端点。

Example:

@Autowired

**private** MetricExportProperties export;

@Bean

**public** PublicMetrics metricsAggregate() {

**return new** MetricReaderPublicMetrics(aggregatesMetricReader());

}

**private** MetricReader globalMetricsForAggregation() {

**return new** RedisMetricRepository(**this**.connectionFactory,

**this**.export.getRedis().getAggregatePrefix(), **this**.export.getRedis().getKey());

}

**private** MetricReader aggregatesMetricReader() { AggregateMetricReader repository = **new** AggregateMetricReader(

globalMetricsForAggregation());

**return** repository;

}

**Note**

上面的例子使用MetricExportProperties来注入和提取key和prefix。 这是作为一个方便的spring boot提供给你，默认值将是明智的。 它们在MetricExportAutoConfiguration中设置。

**Note**

上面的MetricReaders不是@Beans，并没有标记为@ExportMetricReader，因为它们只是从其他存储库收集和分析数据，而不想导出它们的值。

## Dropwizard Metrics

在向io.dropwizard.metrics:metrics-core库声明依赖项时，将创建一个默认的MetricRegistry Spring bean; 如果您需要自定义，也可以注册您自己的@Bean实例。[Dropwizard ‘Metrics’ library](https://dropwizard.github.io/metrics/)的用户会发现Spring Boot指标会自动发布到com.codahale.metrics.MetricRegistry。 MetricRegistry的度量标准也通过/metrics端点自动公开

当使用Dropwizard指标时，默认的CounterService和GaugeService被一个DropwizardMetricServices替代，这是一个围绕着MetricRegistry的包装（所以你可以@Autowired这些服务之一，并像平常一样使用它）。 您还可以通过在适当的类型前添加指标名称（例如: timer.\*, histogram.\* for gauges, and meter.\* for counters），来创建“special”Dropwizard度量。

## Message channel integration

如果存在称为metricsChannel的MessageChannel bean，则将创建一个MetricWriter，将度量标准写入该通道。 发送到通道的每个消息都将包含Delta或Metric有效内容，并具有一个metricName header。 writer自动连接到导出器（所有writer），所有metric values将出现在通道上，并且可以由用户进行额外的分析或操作（由您来提供通道和您需要的任何用户）。

# Auditing(审计)

Spring Boot Actuator有一个灵活的审计(audit)框架，一旦Spring Security发挥作用就会发布事件（默认情况下为' authentication success '，' failure'和' access denied '异常）。 这对于报告(reporting)是非常有用的，并且还可以基于认证失败实施锁定策略。 要定制发布的安全事件，您可以提供自己的AbstractAuthenticationAuditListener和AbstractAuthorizationAuditListener实现。

您也可以选择将审计(audit)服务用于您自己的业务事件。 为此，您可以将现有的AuditEventRepository注入到自己的组件中，直接使用它，也可以通过Spring ApplicationEventPublisher（使用ApplicationEventPublisherAware）发布AuditApplicationEvent。

# Tracing(跟踪)

所有HTTP请求都会自动启用跟踪。 您可以查看跟踪端点并获取有关最近100个请求的基本信息：

[**{**

***"timestamp"***: 1394343677415**,**

***"info"***: **{**

***"method"***: ***"GET"*,**

***"path"***: ***"/trace"*, *"headers"***: **{**

***"request"***: **{**

***"Accept"***: ***"text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8"*, *"Connection"***: ***"keep-alive"*,**

***"Accept-Encoding"***: ***"gzip, deflate"*,**

***"User-Agent"***: ***"Mozilla/5.0 Gecko/Firefox"*, *"Accept-Language"***: ***"en-US,en;q=0.5"*,**

***"Cookie"***: ***"\_ga=GA1.1.827067509.1390890128; ..."***

***"Authorization"***: ***"Basic ..."*, *"Host"***: ***"localhost:8080"***

**},**

***"response"***: **{**

***"Strict-Transport-Security"***: ***"max-age=31536000 ; includeSubDomains"*, *"X-Application-Context"***: ***"application:8080"*,**

***"Content-Type"***: ***"application/json;charset=UTF-8"*, *"status"***: ***"200"***

**}**

**}**

**}**

},**{**

***"timestamp"***: 1394343684465**,**

...

**}]**

跟踪默认包含在跟踪中：

|  |  |
| --- | --- |
| **Name**  Request Headers | **Description**  Headers from the request. |
| Response Headers | Headers from the response. |
| Cookies | Cookie from request headers and Set-Cookie from response headers. |
| Errors | The error attributes (if any). |
| Time Taken | The time taken to service the request in milliseconds. |

## Custom tracing

如果您需要跟踪其他事件，则可以将[TraceRepository](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/trace/TraceRepository.java) 注入Spring bean。该add方法接受一个将被转换为

JSON并记录的Map结构。

默认情况下，将使用InMemoryTraceRepository来存储最近的100个事件。 如果需要扩展容量，可以定义自己的InMemoryTraceRepository bean实例。 您也可以根据需要创建自己的TraceRepository实现。

# Process monitoring

在Spring Boot Actuator中，您可以找到几个类来创建对进程监视有用的文件：

* ApplicationPidFileWriter creates a file containing the application PID (by default in the application directory with the file name application.pid).
* EmbeddedServerPortFileWriter creates a file (or files) containing the ports of the embedded server (by default in the application directory with the file name application.port).

这些编写器(writers)默认情况下不会被激活，但是您可以通过下面介绍的方式启用它们。

## Extend configuration

在META-INF/spring.factories文件中，您可以激活写入PID文件的侦听器。 例：

org.springframework.context.ApplicationListener=\ org.springframework.boot.system.ApplicationPidFileWriter,\ org.springframework.boot.actuate.system.EmbeddedServerPortFileWriter

## Programmatically

您还可以通过调用SpringApplication.addListeners（...）方法并传递相应的Writer对象来激活侦听器。 此方法还允许您通过Writer构造函数自定义文件名和路径。

# Cloud Foundry support

Spring Boot的执行器(actuator)模块包含额外的支持，当您部署到兼容的Cloud Foundry实例时，该支持会被激活。/cloudfoundryapplication路径为所有NamedMvcEndpoint bean提供了一个替代安全路由。

扩展支持允许Cloud Foundry管理UIs（例如，可用于查看已部署应用程序的Web应用程序）通过Spring Boot执行程序信息进行扩充。 例如，应用程序状态页面可能包含完整的健康信息，而不是典型的“running”或“stopped”状态。

**Note**

/cloudfoundryapplication路径不能直接被普通用户访问。 为了使用端点，必须将有效的UAA令牌(token)与请求一起传递。

## Disabling extended Cloud Foundry actuator support

如果要完全禁用/cloudfoundryapplication端点，则可以将以下内容添加到application.properties文件中：

**application.properties.**

**management.cloudfoundry.enabled**=false

## Cloud Foundry self signed certificates

默认情况下，/ cloudfoundryapplication端点的安全验证会对各种Cloud Foundry服务进行SSL调用。 如果您的Cloud Foundry UAA或云控制器服务使用自签名证书(self-signed certificates)，则需要设置以下属性：

**application.properties.**

**management.cloudfoundry.skip-ssl-validation**=true

## Custom security configuration

如果您定义自定义安全配置(custom security configuration)，并且希望扩展Cloud Foundry执行器(actuator)支持，则应确保/cloudfoundryapplication/\*\*路径已打开。 如果没有直接的开放路线，您的Cloud Foundry应用程序管理器将无法获取端点数据。

对于Spring Security，通常会包含像mvcMatchers（"/ cloudfoundryapplication/\*\*"）。permitAll（）在您的配置中：

@Override

**protected void** configure(HttpSecurity http) **throws** Exception { http

.authorizeRequests()

.mvcMatchers(***"/cloudfoundryapplication/\*\*"***)

.permitAll()

.mvcMatchers(***"/mypath"***)

.hasAnyRole(***"SUPERUSER"***)

.anyRequest()

.authenticated().and()

.httpBasic();

}

# What to read next

如果您想了解本章讨论的一些概念，可以看一下执行器(actuator)的[sample applications](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples)。 您也可能想阅读[Graphite](http://graphite.wikidot.com/)等图形工具。

否则，您可以继续阅读关于[‘deployment options’](#_bookmark363) 的内容，或者跳过一些关于Spring Boot [*build tool plugins*](#_bookmark407)的深入信息。

**Part VI. Deploying Spring Boot applications**

Spring Boot的灵活的打包选项在部署应用程序时提供了大量的选择。 您可以轻松地将Spring Boot应用程序部署到各种云平台，容器映像（例如Docker）或虚拟/真实机器(virtual/real machines)。

本节介绍一些更常见的部署方案。

# Deploying to the cloud

Spring Boot的可执行程序jar是为大多数流行的PaaS（平台即服务）提供商准备的。 这些提供者往往要求你“platform-as-a-service”; 他们管理应用程序进程（特别是不是Java应用程序），因此他们需要一些中间层，使您的应用程序适应云对正在运行的进程的概念。

两家受欢迎的云提供商Heroku和Cloud Foundry采用“buildpack”方法。 buildpack将你部署的代码封装在启动应用程序所需的任何东西上：它可能是一个JDK和一个java调用，它可能是一个嵌入式Web服务器，或者它可能是一个成熟的应用程序服务器。 一个buildpack是可插入的，但理想情况下，你应该能够尽可能少的自定义。 这减少了不受您控制的功能的占用空间。 它最大限度地减少了开发和生产环境之间的差异。

理想情况下，您的应用程序（如Spring Boot可执行文件jar）具有运行所需的所有内容。

在本节中，我们将看看如何使我们在“Getting Started”一节中开发并在云中运行的简单应用程序([simple application that we developed](#_bookmark24) )成为可能。

## Cloud Foundry

如果没有指定其他buildpack，Cloud Foundry将提供默认构建包。 Cloud Foundry [Java buildpack](https://github.com/cloudfoundry/java-buildpack) 对Spring应用程序（包括Spring Boot）提供了很好的支持。 您可以部署独立的可执行jar应用程序以及传统的.war打包的应用程序。

构建完应用程序（例如，使用mvn clean package）并安装[installed the cf](http://docs.cloudfoundry.org/devguide/installcf/install-go-cli.html) [command line tool](http://docs.cloudfoundry.org/devguide/installcf/install-go-cli.html)后，只需使用cf push命令部署应用程序，如下所示，将路径替换为已编译的.jar文件。 推送应用程序之前，请确保已使用[logged in with your cf command line client](http://docs.cloudfoundry.org/devguide/installcf/whats-new-v6.html#login) 登录。

$ cf push acloudyspringtime -p target/demo-0.0.1-SNAPSHOT.jar

请参阅[cf push documentation](http://docs.cloudfoundry.org/devguide/installcf/whats-new-v6.html#push) 以获取更多选项。 如果在同一个目录下有一个Cloud Foundry [manifest.yml](http://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html) 文件，将会被查询。

**Note**

在这里，我们用acloudyspringtime代替你给cf的任何值作为你的应用程序的名字。

在这一点上，cf将开始上传您的应用程序：

Uploading acloudyspringtime... **OK**

Preparing to start acloudyspringtime... **OK**

-----> Downloaded app package (**8.9M**)

-----> Java Buildpack source: system

-----> Downloading Open JDK 1.7.0\_51 from .../x86\_64/openjdk-1.7.0\_51.tar.gz (**1.8s**) Expanding Open JDK to .java-buildpack/open\_jdk (**1.2s**)

-----> Downloading Spring Auto Reconfiguration from 0.8.7 .../auto-reconfiguration-0.8.7.jar (**0.1s**)

-----> Uploading droplet (**44M**)

Checking status of app 'acloudyspringtime'...

0 of 1 instances running (1 starting)

...

0 of 1 instances running (1 down)

...

0 of 1 instances running (1 starting)

...

1 of 1 instances running (1 running)

App started

恭喜！ 应用程序现在 live！

然后验证部署的应用程序的状态很容易：

$ cf apps

Getting applications in ... OK

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name | requested state | instances | memory | disk | urls |
| ... |  |  |  |  |  |
| acloudyspringtime | started | 1/1 | 512M | 1G | acloudyspringtime.cfapps.io |
| ... |  |  |  |  |  |

一旦Cloud Foundry确认已经部署了您的应用程序，您应该能够按照给定的URI（在这种情况下为http://acloudyspringtime.cfapps.io/）来访问应用程序。

### Binding to services

默认情况下，有关正在运行的应用程序的元数据以及服务连接信息将作为环境变量（例如：$ VCAP\_SERVICES）公开给应用程序。 这种架构决定是由于Cloud Foundry的多语言（任何语言和平台都可以作为buildpack支持）性质; 过程范围的环境变量是语言不可知的。

境变量并不总是最简单的API，所以Spring Boot会自动提取它们，并将数据变成可以通过Spring的环境抽象访问的属性：

@Component

**class** MyBean **implements** EnvironmentAware {

**private** String instanceId; @Override

**public void** setEnvironment(Environment environment) {

**this**.instanceId = environment.getProperty(***"vcap.application.instance\_id"***);

}

*// ...*

}

所有Cloud Foundry属性都以vcap作为前缀。 您可以使用vcap属性来访问应用程序信息（如应用程序的公用URL）和服务信息（如数据库凭证）。 有关完整的详细信息，请参阅CloudFoundryVcapEnvironmentPostProcessor Javadoc。

**Tip**

[Spring Cloud Connectors](http://cloud.spring.io/spring-cloud-connectors/) 项目更适合配置数据源等任务。 Spring Boot包括自动配置支持和spring-boot-starter-cloud- connectors starter。

## Heroku

Heroku是另一个流行的PaaS平台。 要定制Heroku构建，您需要提供一个Procfile，它提供了部署应用程序所需的咒语(incantation)。 Heroku分配一个端口供Java应用程序使用，然后确保到外部URI的路由工作。

您必须配置您的应用程序以侦听正确的端口。 以下是我们的入门REST应用程序的Procfile：

web: java -Dserver.port=$PORT -jar target/demo-0.0.1-SNAPSHOT.jar

Spring Boot makes -D arguments 成为可从Spring Environment实例访问的属性。 将server.port配置属性提供给嵌入式Tomcat，Jetty或Undertow实例，然后在启动时使用它。 $ PORT环境变量由Heroku PaaS分配给我们。

Heroku默认使用Java 1.8。 只要您的Maven或Gradle构建版本设置为使用相同的版本（Maven用户可以使用java.version属性），这就好了。 如果您想使用JDK 1.7，请在您的pom.xml和Procfile附近创建一个名为system.properties的新文件。 在这个文件中添加以下内容：

java.runtime.version=1.7

这应该是你需要的一切。 Heroku部署最常见的工作流程是git将代码推送到生产环境。

$ git push heroku master

Initializing repository, **done**. Counting objects: 95, **done**.

Delta compression using up to 8 threads. Compressing objects: 100% (78/78), **done**.

Writing objects: 100% (95/95), 8.66 MiB | 606.00 KiB/s, **done**. Total 95 (delta 31), reused 0 (delta 0)

-----> Java app detected

-----> Installing OpenJDK 1.8... **done**

-----> Installing Maven 3.3.1... **done**

-----> Installing settings.xml... **done**

-----> Executing: mvn -B -DskipTests=true clean install

[INFO] Scanning for projects... [Downloading: http://repo.spring.io/...](http://repo.spring.io/)

[Downloaded: http://repo.spring.io/...](http://repo.spring.io/) (818 B at 1.8 KB/sec)

....

[Downloaded: http://s3pository.heroku.com/jvm/...](http://s3pository.heroku.com/jvm/) (152 KB at 595.3 KB/sec) [INFO] Installing /tmp/build\_0c35a5d2-a067-4abc-a232-14b1fb7a8229/target/... [INFO] Installing /tmp/build\_0c35a5d2-a067-4abc-a232-14b1fb7a8229/pom.xml ... [INFO] ------------------------------------------------------------------------ [INFO] **BUILD SUCCESS**

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 59.358s

[INFO] Finished at: Fri Mar 07 07:28:25 UTC 2014 [INFO] Final Memory: 20M/493M

[INFO] ------------------------------------------------------------------------

-----> Discovering process types Procfile declares types -> **web**

-----> Compressing... **done**, 70.4MB

-----> Launching... **done**, v6

<http://agile-sierra-1405.herokuapp.com/>**deployed to Heroku**

To git@heroku.com:agile-sierra-1405.git

\* [new branch] master -> master

你的应用程序现在应该在Heroku上运行.

## OpenShift

[OpenShift](https://www.openshift.com/) 是RedHat公共（和企业）PaaS解决方案。 像Heroku一样，它通过运行由git提交触发的脚本来工作，所以只要Java运行时可用，您就可以用任何方式编写启动Spring Boot应用程序的脚本（这是一个标准功能，您可以在OpenShift）。 要做到这一点，你可以使用[DIY Cartridge](https://www.openshift.com/developers/do-it-yourself) 和钩子在你的仓库下.openshift/ action\_hooks:

基本模式是：

1. 确保Java和您的构建工具远程安装，例如 使用pre\_build钩子（默认安装Java和Maven，Gradle不是）
2. 使用构建钩子来构建您的jar（使用Maven或Gradle），例如

#!/bin/bash

cd $OPENSHIFT\_REPO\_DIR

mvn package -s .openshift/settings.xml -DskipTests=true

1. 添加一个调用java -jar的开始钩子...

#!/bin/bash

cd $OPENSHIFT\_REPO\_DIR

nohup java -jar target/\*.jar --server.port=${OPENSHIFT\_DIY\_PORT} --server.address=${OPENSHIFT\_DIY\_IP} &

1. 使用一个停止钩（因为开始应该干净地返回），例如

#!/bin/bash

source $OPENSHIFT\_CARTRIDGE\_SDK\_BASH

PID=$(ps -ef | grep java.\*\.jar | grep -v grep | awk '{ print $2 }') if [ -z "$PID" ]

then

client\_result "Application is already stopped"

else

kill $PID

fi

1. 在你的平台上嵌入平台提供的环境变量的服务绑定 application.properties, 例如：

spring.datasource.url: jdbc:mysql://${OPENSHIFT\_MYSQL\_DB\_HOST}:${OPENSHIFT\_MYSQL\_DB\_PORT}/

${OPENSHIFT\_APP\_NAME}

spring.datasource.username: ${OPENSHIFT\_MYSQL\_DB\_USERNAME} spring.datasource.password: ${OPENSHIFT\_MYSQL\_DB\_PASSWORD}

在他们的网站上有一个在[running Gradle in OpenShift](https://www.openshift.com/blogs/run-gradle-builds-on-openshift) ，可以让你开始使用Gradle构建来运行应用程序。

## Amazon Web Services (AWS)

亚马逊网络服务提供了多种方式来安装基于Spring Boot的应用程序，既可以是传统的Web应用程序（war），也可以是带有嵌入式Web服务器的可执行jar文件。 选项包括：

* AWS Elastic Beanstalk
* AWS Code Deploy
* AWS OPS Works
* AWS Cloud Formation
* AWS Container Registry

每个模块都有不同的功能和定价模式，下面我们将只描述最简单的选项：AWS Elastic Beanstalk。

### AWS Elastic Beanstalk

正如[Elastic Beanstalk Java guide](http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_Java.html)指南中所述，部署Java应用程序有两个主要选项; 您可以使用“Tomcat平台”或“Java SE平台”。

#### Using the Tomcat platform

这个选项适用于产生war文件的Spring Boot项目。 没有任何特殊的配置要求，只需按照官方指南。

#### Using the Java SE platform

此选项适用于生成jar文件并运行嵌入式Web容器的Spring Boot项目。 Elastic Beanstalk环境在端口80上运行nginx实例，以代理在端口5000上运行的实际应用程序。要配置它，请将以下内容添加到application.properties中：

server.port=5000

#### Best practices

**上传二进制文件而不是源文件**

默认情况下，Elastic Beanstalk会上传源代码并在AWS中编译它们。 要上传二进制文件，请将以下内容添加到.elasticbeanstalk / config.yml文件中：

deploy:

artifact: target/demo-0.0.1-SNAPSHOT.jar

**通过设置环境类型来降低成本**

默认情况下，Elastic Beanstalk环境是负载平衡的。 负载平衡器具有成本透视图，为了避免这种情况，请按照Amazon文档([in the Amazon documentation](http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-create-wizard.html#environments-create-wizard-capacity))中的说明将环境类型设置为“单实例”。 使用CLI也可以使用以下命令创建单实例环境：

eb create -s

### Summary

这是获得AWS最简单的方法之一，但还有更多的事情要做，例如：如何将Elastic Beanstalk集成到任何CI / CD工具，使用Elastic Beanstalk maven插件而不是CLI等等。 [blog](https://exampledriven.wordpress.com/2017/01/09/spring-boot-aws-elastic-beanstalk-example/) 更详细地介绍了这些话题。

## Boxfuse and Amazon Web Services

[Boxfuse](https://boxfuse.com/) 通过将Spring Boot可执行程序jar或war转换为可以在VirtualBox或AWS上以未改变的方式部署的最小VM映像来实现。 Boxfuse与Spring进行了深度整合启动并将使用Spring Boot配置文件中的信息来自动配置端口和运行状况检查URL。 Boxfuse利用这些信息来生成它所生成的图像以及它提供的所有资源（实例，安全组，弹性负载均衡器等）。

一旦创建了[Boxfuse account](https://console.boxfuse.com/)，将其连接到您的AWS账户，并安装了最新版本的Boxfuse客户端，您可以按如下方式将Spring Boot应用程序部署到AWS（确保应用程序是由Maven或Gradle首先使用， 例如，mvn clean package）：

$ boxfuse run myapp-1.0.jar -env=prod

请参阅[boxfuse run documentation](https://boxfuse.com/docs/commandline/run.html) 以获取更多选项。 如果当前目录中存在[boxfuse.com/docs/commandline/](https://boxfuse.com/docs/commandline/#configuration) [#configuration](https://boxfuse.com/docs/commandline/#configuration)  [boxfuse.conf]文件，将会查询该文件。

**Tip**

默认情况下，Boxfuse将在启动时激活名为boxfuse的Spring配置文件，如果您的可执行jar或war包含[boxfuse.com/docs/payloads/springboot.html#configuration](https://boxfuse.com/docs/payloads/springboot.html#configuration) [application-boxfuse.properties]文件，则Boxfuse将基于其配置 它包含的属性。

此时boxfuse将为您的应用程序创建一个映像，将其上传，然后在AWS上配置并启动必要的资源：

Fusing Image for myapp-1.0.jar ...

Image fused in 00:06.838s (53937 K) -> axelfontaine/myapp:1.0 Creating axelfontaine/myapp ...

Pushing axelfontaine/myapp:1.0 ... Verifying axelfontaine/myapp:1.0 ... Creating Elastic IP ...

Mapping myapp-axelfontaine.boxfuse.io to 52.28.233.167 ...

Waiting for AWS to create an AMI for axelfontaine/myapp:1.0 in eu-central-1 (this may take up to 50 seconds) ...

AMI created in 00:23.557s -> ami-d23f38cf

Creating security group boxfuse-sg\_axelfontaine/myapp:1.0 ...

Launching t2.micro instance of axelfontaine/myapp:1.0 (ami-d23f38cf) in eu-central-1 ... Instance launched in 00:30.306s -> i-92ef9f53

[Waiting for AWS to boot Instance i-92ef9f53 and Payload to start at http://52.28.235.61/](http://52.28.235.61/) ... Payload started in 00:29.266s -> <http://52.28.235.61/>

Remapping Elastic IP 52.28.233.167 to i-92ef9f53 ...

Waiting 15s for AWS to complete Elastic IP Zero Downtime transition ...

[Deployment completed successfully. axelfontaine/myapp:1.0 is up and running at http://myapp-](http://myapp-/) axelfontaine.boxfuse.io/

您的应用程序现在应该在AWS上运行。

有一个关于在[deploying Spring Boot apps on EC2](https://boxfuse.com/blog/spring-boot-ec2.html) 的博客，以及他们网站上Boxfuse Spring Boot集成的文档- [documentation for the Boxfuse Spring](https://boxfuse.com/docs/payloads/springboot.html) [Boot integration](https://boxfuse.com/docs/payloads/springboot.html) ，这将使您开始使用Maven构建来运行应用程序。

## Google Cloud

Google云有几个可用于启动Spring Boot应用程序的选项。 最容易入门的可能是App Engine，但是您也可以找到在Container Engine的容器中或在使用Compute Engine的虚拟机上运行Spring Boot的方法。

要在App Engine中运行，您可以先在用户界面中创建一个项目，为您设置唯一的标识符以及HTTP路由。 将Java应用程序添加到项目中并将其保留为空，然后使用[Google Cloud](https://cloud.google.com/sdk/downloads) [SDK](https://cloud.google.com/sdk/downloads) 将Spring Boot应用程序从命令行或CI构建推入该插槽(slot)。

App Engine需要您创建一个app.yaml文件来描述您的应用程序需要的资源。 通常你把它放在src / main / appengine中，看起来像这样：

**service**: default

**runtime**: java

**env**: flex

**runtime\_config**: **jdk**: openjdk8

**handlers**:

**- url**: /.\*

**script**: this field is required**,** but ignored

**manual\_scaling**: **instances**: 1

**health\_check**: **enable\_health\_check**: False

**env\_variables**:

**ENCRYPT\_KEY**: your\_encryption\_key\_here

例如，您可以通过将Maven插件添加到构建配置来部署应用程序：

**<plugin>**

**<groupId>**com.google.cloud.tools**</groupId>**

**<artifactId>**appengine-maven-plugin**</artifactId>**

**<version>**1.3.0**</version>**

**<configuration>**

**<project>**myproject**</project>**

**</configuration>**

**</plugin>**

然后使用mvn appengine:deploy进行部署（如果您需要首先进行身份验证，则构建将失败）。

**Note**

Google App Engine Classic绑定到Servlet 2.5 API，所以不能在没有修改的情况下部署Spring应用程序。 请参阅[Servlet 2.5 section](#_bookmark567) 。

# Installing Spring Boot applications

除了使用java -jar运行Spring Boot应用程序之外，还可以为Unix系统创建完全可执行的应用程序。 完全可执行的jar可以像任何其他可执行的二进制文件一样执行，也可以使用[registered with init.d or systemd](#_bookmark379)。 这使得在普通生产环境中安装和管理Spring Boot应用程序变得非常简单。

**Warning**

完全可执行的jar可以通过在文件的前面嵌入一个额外的脚本来工作。 目前，一些工具不接受这种格式，所以你可能无法一直使用这种技术。 例如，jar -xf可能默默无法提取已经被完全执行的jar或war。 建议如果您打算直接执行它，而不是使用java -jar或将其部署到servlet容器，那么您只能使jar或war完全可执行。

要用Maven创建一个‘fully executable”的jar，使用下面的插件配置：

**<plugin>**

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

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

**<configuration>**

**<executable>**true**</executable>**

**</configuration>**

**</plugin>**

使用Gradle，等效的配置是：

springBoot {

executable = true

}

然后，您可以通过输入./my-application.jar（其中my-application是您的artifact的名称）运行您的应用程序。 包含该jar的目录将被用作应用程序的工作目录。

## 支持的操作系统(Supported operating systems)

默认脚本支持大多数Linux发行版，并在CentOS和Ubuntu上进行测试。 其他平台，比如OS X和FreeBSD，将需要使用定制的embeddedLaunchScript。

## Unix/Linux services

Spring Boot应用程序可以使用init.d或systemd轻松启动为Unix / Linux服务。

### 作为init.d服务安装(Installation as an init.d service (System V))

如果您已经配置了Spring Boot的Maven或Gradle插件来生成一个[fully executable jar](#_bookmark377)，并且您没有使用自定义embeddedLaunchScript，那么您的应用程序可以用作init.d服务。 将jar简单地链接到init.d，以支持标准的启动，停止，重新启动和状态命令。

该脚本支持以下功能：

* 以拥有该jar文件的用户身份启动服务
* 使用/var/run/<appname>/<appname>.pid跟踪应用程序的PID
* Writes console logs to /var/log/<appname>.log

假设您在/ var / myapp中安装了Spring Boot应用程序，要将Spring Boot应用程序安装为init.d服务，只需创建一个符号链接即可：

$ sudo ln -s /var/myapp/myapp.jar /etc/init.d/myapp

一旦安装，您可以按照通常的方式启动和停止服务。 例如，在基于Debian的系统上：

$ service myapp start

**Tip**

如果您的应用程序无法启动，请检查写入/var/log/<appname>.log的日志文件是否有错误。

您还可以将应用程序标记为使用标准操作系统工具自动启动。 例如，在Debian上：

$ update-rc.d myapp defaults <priority>

#### Securing an init.d service

**Note**

以下是关于如何保护作为init.d服务运行的Spring Boot应用程序的一组准则。 这并不是为了强化应用程序及其运行环境而应该做的所有事情的详尽清单。

当以root用户身份执行时，就像使用root启动init.d服务一样，默认的可执行脚本将以拥有jar文件的用户身份运行应用程序。 你永远不应该以root运行一个Spring Boot应用程序，所以你的应用程序的jar文件不应该被root所有。 相反，创建一个特定的用户来运行你的应用程序，并使用chown使其成为jar文件的所有者。 例如：

$ chown bootapp:bootapp your-app.jar

在这种情况下，默认的可执行脚本将作为bootapp用户运行应用程序。

**Tip**

为了减少应用程序用户帐户被攻破的可能性，您应该考虑防止它使用登录shell。 例如，将帐户的外壳设置为/ usr / sbin / nologin。

您还应该采取措施来防止修改应用程序的jar文件。 首先，配置它的权限，使其不能被写入，只能被其所有者读取或执行：

$ chmod 500 your-app.jar

其次，如果您的应用程序或运行该应用程序的帐户受到威胁，您还应该采取措施来限制损害。 如果攻击者获得访问权限，他们可以将jar文件写入并更改其内容。 防止这种情况的一种方法是使用chattr使其不可变：

$ sudo chattr +i your-app.jar

这将阻止任何用户（包括root）修改jar。

如果使用root来控制应用程序的服务，并使用[use a .conf file](#_bookmark386) 文件来定制其启动，那么.conf文件将被

root用户读取和评估。 它应该得到相应的保证。 使用chmod，以便文件只能被所有者读取，并使用

chown来使root拥有者：

$ chmod 400 your-app.conf

$ sudo chown root:root your-app.conf

### Installation as a systemd service

Systemd是System V init系统的继承者，现在已经被许多现代Linux发行版所使用。 尽管可以继续使用systemd的init.d脚本，但也可以使用systemd“‘service”脚本启动Spring Boot应用程序。

假设您在/ var / myapp中安装了Spring Boot应用程序，要将Spring Boot应用程序安装为systemd服务，请使用以下示例创建一个名为myapp.service的脚本，并将其放在/ etc / systemd / system目录中：

[Unit] Description=myapp After=syslog.target

[Service] User=myapp

ExecStart=/var/myapp/myapp.jar SuccessExitStatus=143

[Install]

WantedBy=multi-user.target

**Tip**

请记住更改应用程序的说明，用户和执行开始字段。

**Tip**

请注意，ExecStart字段不会声明脚本动作命令，这意味着运行命令是默认使用的。

请注意，与以init.d服务运行不同的是，运行应用程序，PID文件和控制台日志文件的用户由systemd本身管理，因此必须使用“service”脚本中的相应字段进行配置。 有关更多详细信息，请参阅服务单位配置手册页[service unit configuration man page](http://www.freedesktop.org/software/systemd/man/systemd.service.html) 。

要将应用程序标记为在系统引导时自动启动，请使用以下命令：

$ systemctl enable myapp.service

有关更多详细信息，请参阅man systemctl。

### 自定义 startup script

Maven或Gradle插件编写的默认嵌入式启动脚本可以通过多种方式进行自定义。 对于大多数人来说，使用默认脚本以及一些自定义是通常就够了。 如果您发现无法自定义您需要的内容，则可以始终使用embeddedLaunchScript选项来完全编写自己的文件。

#### Customizing script when it’s written

在写入jar文件时，定制启动脚本的元素通常是有意义的。 例如，init.d脚本可以提供一个“description”，因为你知道这个前面（而且不会改变），所以当你生成jar文件时，你也可以提供它。

要定制书写元素，请使用Spring Boot Maven或Gradle插件的embeddedLaunchScriptProperties选项。

默认脚本支持以下属性替换：

| **Name** | **Description** |
| --- | --- |
| mode | The script mode. Defaults to auto. |
| initInfoProvides | The Provides section of “INIT INFO”. Defaults to spring-boot-application for Gradle and to ${project.artifactId} for Maven. |
| initInfoRequiredStart | The Required-Start section of “INIT INFO”. Defaults to $remote\_fs $syslog $network. |
| initInfoRequiredStop | The Required-Stop section of “INIT INFO”. Defaults to $remote\_fs $syslog $network. |
| initInfoDefaultStart | The Default-Start section of “INIT INFO”. Defaults to 2 3 4 5. |
| initInfoDefaultStop | The Default-Stop section of “INIT INFO”. Defaults to 0 1 6. |
| initInfoShortDescription | The Short-Description section of “INIT INFO”. Defaults to Spring Boot Application for Gradle and to ${project.name} for Maven. |
| initInfoDescription | The Description section of “INIT INFO”. Defaults to Spring Boot Application for Gradle and to ${project.description}(falling back to ${project.name}) for Maven. |
| initInfoChkconfig | The chkconfig section of “INIT INFO”. Defaults to 2345 99 01. |
| confFolder | The default value for CONF\_FOLDER. Defaults to the folder containing the jar. |
| logFolder | The default value for LOG\_FOLDER. Only valid for an init.d service. |
| logFilename | The default value for LOG\_FILENAME. Only valid for an init.d service. |
| pidFolder | The default value for PID\_FOLDER. Only valid for an init.d service. |
| pidFilename | The default value for the name of the pid file in PID\_FOLDER. Only valid for an init.d service. |
| useStartStopDaemon | If the start-stop-daemon command, when it’s available, should be used to control the process. Defaults to true. |
| stopWaitTime | The default value for STOP\_WAIT\_TIME. Only valid for an init.d service. Defaults to 60 seconds. |

#### Customizing script when it runs

对于写入jar之后需要定制的脚本项目，可以使用环境变量或配置文件[config file](#_bookmark386)

For items of the script that need to be customized *after* the jar has been written you can use environment variables or a [config file](#_bookmark386).

默认脚本支持以下环境属性：

| **Variable** | **Description** |
| --- | --- |
| MODE | The “mode” of operation. The default depends on the way the jar was built, but will usually be auto *(meaning it tries to guess if it is an init script by checking if it is a symlink in a directory called init.d)*. You can explicitly set it to service so that the stop|start|status|restart commands work, or to run if you just want to run the script in the foreground. |
| USE\_START\_STOP\_DAEMON | If the start-stop-daemon command, when it’s available, should be used to control the process. Defaults to true. |
| PID\_FOLDER | The root name of the pid folder (/var/run by default). |
| LOG\_FOLDER | The name of the folder to put log files in (/var/log by default). |
| CONF\_FOLDER | The name of the folder to read .conf files from (same folder as jar-file by default). |
| LOG\_FILENAME | The name of the log file in the LOG\_FOLDER (<appname>.log by default). |
| APP\_NAME | The name of the app. If the jar is run from a symlink the script guesses the app name, but if it is not a symlink, or you want to explicitly set the app name this can be useful. |
| RUN\_ARGS | The arguments to pass to the program (the Spring Boot app). |
| JAVA\_HOME | The location of the java executable is discovered by using the PATH by default, but you can set it explicitly if there is an executable file at $JAVA\_HOME/bin/java. |
| JAVA\_OPTS | Options that are passed to the JVM when it is launched. |
| JARFILE | The explicit location of the jar file, in case the script is being used to launch a jar that it is not actually embedded in. |
| DEBUG | if not empty will set the -x flag on the shell process, making it easy to see the logic in the script. |
| STOP\_WAIT\_TIME | The time in seconds to wait when stopping the application before forcing a shutdown (60 by default). |

**Note**

PID\_FOLDER，LOG\_FOLDER和LOG\_FILENAME变量仅对init.d服务有效。 在systemd中，使用“service”脚本进行等效的自定义。 检查服务单位配置手册页[service unit configuration man page](http://www.freedesktop.org/software/systemd/man/systemd.service.html) 获取更多详细信息。

除了JARFILE和APP\_NAME之外，上述设置可以使用.conf文件进行配置。 该文件预计在jar文件旁边，并且具有相同的名称，但后缀为.conf而不是.jar。 例如，名为/var/myapp/myapp.jar的jar将使用名为/ var / myapp / myapp.conf的配置文件。

**myapp.conf.**

JAVA\_OPTS=-Xmx1024M

LOG\_FOLDER=/custom/log/folder

**Tip**

你可以使用CONF\_FOLDER环境变量来定制配置文件的位置，如果你不喜欢它居住在jar旁边的话。

要了解有关保护此文件的适当方法，请参阅保护init.d服务的指导原则[the guidelines for securing an init.d service](#_bookmark381)。

## Microsoft Windows services

Spring Boot应用程序可以使用[winsw](https://github.com/kohsuke/winsw)作为Windows服务启动。

分别维护Spring Boot核心的示例逐步介绍[maintained separately](https://github.com/snicoll-scratches/spring-boot-daemon) 了如何为Spring Boot应用程序创建Windows服务。

# What to read next

查看[Cloud Foundry](http://www.cloudfoundry.com/), [Heroku](https://www.heroku.com/), [OpenShift](https://www.openshift.com/) 和 [Boxfuse](https://boxfuse.com/) 网站，了解有关PaaS可提供的各种功能的更多信息。 这些只是四个最受欢迎的Java PaaS提供商，因为Spring Boot非常适合基于云的部署，所以您可以自由地考虑其他提供商。

下一节继续介绍[*Spring Boot CLI*](#_bookmark389); 或者你可以跳转到阅读有关[*build*](#_bookmark407)[*tool plugins*](#_bookmark407)。

**Part VII. Spring Boot CLI**

Spring Boot CLI是一个命令行工具，如果您想用Spring快速开发，可以使用它。 它允许你运行Groovy脚本，这意味着你有一个熟悉的类Java语法，没有太多的样板代码。 你也可以引导一个新的项目，或者编写你自己的命令。

# Installing the CLI

Spring Boot CLI可以手动安装; 使用SDKMAN！ （SDK管理器），或者如果您是OSX用户，则使用Homebrew或MacPorts。 有关全面的安装说明，请参见[*Section 10.2, “Installing the Spring Boot CLI”*](#_bookmark16) 。

# 使用 CLI(command-line interface)

一旦你安装了CLI，你可以通过输入spring来运行它。 如果您运行没有任何参数的spring，将显示一个简单的帮助屏幕：

$ spring

usage: spring [--help] [--version]

<command> [<args>] Available commands are:

run [options] <files> [--] [args] Run a spring groovy script

*... more command help is shown here*

您可以使用帮助获得有关任何受支持命令的更多详细信息。 例如：

$ spring help run

spring run - Run a spring groovy script

usage: spring run [options] <files> [--] [args] Option Description

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

--autoconfigure [Boolean] Add autoconfigure compiler

transformations (default: true)

--classpath, -cp Additional classpath entries

-e, --edit Open the file with the default system editor

--no-guess-dependencies Do not attempt to guess dependencies

--no-guess-imports Do not attempt to guess imports

-q, --quiet Quiet logging

-v, --verbose Verbose logging of dependency resolution

--watch Watch the specified file for changes

version命令提供了一个快速的方法来检查你使用的是哪个版本的Spring Boot。

$ spring version

Spring CLI v1.5.8.RELEASE

## 使用CLI Running applications

您可以使用run命令编译和运行Groovy源代码。 Spring Boot CLI是完全自包含(self-contained)的，因此您不需要任何外部Groovy安装。

下面是一个用Groovy编写的“hello world”Web应用程序示例：

**hello.groovy.**

@RestController

**class** WebApplication {

@RequestMapping("/") String home() {

***"Hello World!"***

}

}

编译并运行应用程序类型：

$ spring run hello.groovy

要将命令行参数传递给应用程序，需要使用 -- 将它们从“spring”命令参数中分离出来。

$ spring run hello.groovy -- --server.port=9000

要设置JVM命令行参数，可以使用JAVA\_OPTS环境变量，例如

$ JAVA\_OPTS=-Xmx1024m spring run hello.groovy

### Deduced “grab” dependencies

标准Groovy包含一个@Grab注释，它允许您声明第三方库的依赖关系。 这个有用的技术允许Groovy像Maven或Gradle那样下载jar，但不需要使用构建工具。

Spring Boot进一步扩展了这一技术，并将尝试根据您的代码推断哪些库“grab”。 例如，因为上面的WebApplication代码使用了@RestController注解，所以“Tomcat”和“Spring MVC”将被抓取。

一下仙姑用作 “grab hints”:

|  |  |
| --- | --- |
| **Items** | **Grabs** |
| JdbcTemplate, NamedParameterJdbcTemplate, DataSource | JDBC Application. |
| @EnableJms | JMS Application. |
| @EnableCaching | Caching abstraction. |
| @Test  @EnableRabbit | JUnit.  RabbitMQ. |
| @EnableReactor | Project Reactor. |
| extends Specification | Spock test. |
| @EnableBatchProcessing | Spring Batch. |
| @MessageEndpoint @EnableIntegrationPatterns | Spring Integration. |
| @EnableDeviceResolver  @Controller @RestController @EnableWebMvc | Spring Mobile.  Spring MVC + Embedded Tomcat. |
| @EnableWebSecurity  @EnableTransactionManagement | Spring Security.  Spring Transaction Management. |

**Tip**

请参阅Spring Boot CLI源代码中[CompilerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-cli/src/main/java/org/springframework/boot/cli/compiler/CompilerAutoConfiguration.java) 的子类，以了解如何应用自定义。

### Deduced “grab” coordinates

Spring Boot通过允许你指定一个没有组或者版本的依赖，例如@Grab（'freemarker'）来扩展Groovy的标准@Grab支持。 这将会参考Spring Boot的默认依赖元数据来推断artifact’s的组和版本。 请注意，默认元数据与您所使用的CLI版本相关联 - 只有当您移至CLI的新版本时，才会更改默认元数据，从而使您可以控制何时版本的依赖关系可能会发生变化。 显示默认元数据中包含的依赖关系及其版本的表可以在附录([appendix](#_bookmark607))中找到。

### Default import statements

为了减少Groovy代码的大小，会自动包含几个import语句。 注意上面的例子是如何引用@Component，@RestController和@RequestMapping而不需要使用fully-qualified的名称或import语句。

**Tip**

许多Spring注解将不使用import语句。 尝试运行您的应用程序以查看添加导入之前的失败。

### Automatic main method

与等效的Java应用程序不同，您不需要在Groovy脚本中包含公共静态void main（String [] args）方法。 SpringApplication是自动创建的，编译代码充当源代码。

### Custom dependency management

默认情况下，CLI在解析@Grab依赖项时使用在spring-boot-dependencies中声明的依赖项管理。 可以使用@DependencyManagementBom注释来配置其他依赖项管理，这将重写默认的依赖项管理。 注解的值应该指定一个或多个Maven BOM的坐标（groupId：artifactId：version）。

例如，下面的声明：

@DependencyManagementBom(***"com.example.custom-bom:1.0.0"***)

将在com / example / custom-versions / 1.0.0 /下的Maven仓库中找到custom-bom-1.0.0.pom。

当指定多个BOM时，按照它们声明的顺序应用它们。 例如：

@DependencyManagementBom([***"com.example.custom-bom:1.0.0"***,

***"com.example.another-bom:1.0.0"***])

表示另一个bom中的依赖关系管理将覆盖custom-bom中的依赖关系管理。

您可以在任何可以使用@Grab的地方使用@DependencyManagementBom，但是，为了确保依赖性管理的顺序一致，您最多只能在应用程序中使用@DependencyManagementBom。 一个有用的依赖管理源（Spring Boot的依赖管理的超集）就是Spring IO Platform，@DependencyManagementBom（ [Spring IO Platform](http://platform.spring.io/)：平台BOM：1.1.2.RELEASE'）。

## Testing your code

test命令允许您为应用程序编译和运行测试。 典型用法如下所示：

$ spring test app.groovy tests.groovy Total: 1, Success: 1, : Failures: 0 Passed? true

在这个例子中，tests.groovy包含JUnit @Test方法或Spock规范类。 所有常见的框架注释和静态方法都应该可用，而不必导入它们。

这里是我们上面使用的tests.groovy文件（用JUnit测试）：

**class** ApplicationTests {

@Test

**void** homeSaysHello() {

assertEquals(***"Hello World!"***, **new** WebApplication().home())

}

}

**Tip**

如果您有多个测试源文件，您可能更愿意将它们组织到一个测试目录中。

## Applications with multiple source files

所有接受文件输入的命令都可以使用“shell globbing”。 这使您可以轻松地从单个目录中使用多个文件，例如:

$ spring run \*.groovy

如果您想从主应用程序代码中分离出您的“test” 或 “spec”代码，该技术也可能非常有用：

$ spring test app/\*.groovy test/\*.groovy

## Packaging your application

您可以使用jar命令将您的应用程序打包成一个自包含的可执行jar文件。 例如：

$ spring jar my-app.jar \*.groovy

生成的jar将包含通过编译应用程序和所有应用程序的依赖关系生成的类，以便可以使用java -jar运行。 jar文件还将包含来自应用程序类路径的条目。 您可以使用--include和--exclude（两者都是逗号分隔的，并且都接受值“+”和“ - ”的前缀来表示它们应该从默认值中被删除）向jar中添加显式路径。 默认包括是

public/\*\*, resources/\*\*, static/\*\*, templates/\*\*, META-INF/\*\*, \*

并且默认excludes是

.\*, repository/\*\*, build/\*\*, target/\*\*, \*\*/\*.jar, \*\*/\*.groovy

有关更多信息，请参阅spring帮助jar的输出。

## Initialize a new project

init命令允许您在不离开shell的情况下使用[start.spring.io](https://start.spring.io/) 创建一个新项目。 例如：

$ spring init --dependencies=web,data-jpa my-project Using service at https://start.spring.io

Project extracted to '/Users/developer/example/my-project'

这将使用spring-boot-starter-web和spring-boot-starter-data-jpa创建一个基于Maven的项目的my-project目录。 您可以使用 -- list标志列出服务的功能

$ spring init --list

=======================================

Capabilities of https://start.spring.io

=======================================

Available dependencies:

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

actuator - Actuator: Production ready features to help you monitor and manage your application

...

web - Web: Support for full-stack web development, including Tomcat and spring-webmvc websocket - Websocket: Support for WebSocket development

ws - WS: Support for Spring Web Services

Available project types:

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

gradle-build - Gradle Config [format:build, build:gradle] gradle-project - Gradle Project [format:project, build:gradle] maven-build - Maven POM [format:build, build:maven]

maven-project - Maven Project [format:project, build:maven] (default)

...

init命令支持很多选项，请查看帮助输出以获取更多详细信息。 例如，以下命令使用Java 8和war包创建一个gradle项目：

$ spring init --build=gradle --java-version=1.8 --dependencies=websocket --packaging=war sample-app.zip Using service at https://start.spring.io

Content saved to 'sample-app.zip'

## Using the embedded shell

Spring Boot包含BASH和zsh shell的命令行完成脚本。 如果你没有使用这两个shell（也许你是一个Windows用户），那么你可以使用shell命令启动一个集成的shell。

$ spring shell

**Spring Boot** (v1.5.8.RELEASE)

Hit TAB to complete. Type \'help' and hit RETURN for help, and \'exit' to quit.

从嵌入式shell中，您可以直接运行其他命令：

$ version

Spring CLI v1.5.8.RELEASE

嵌入式shell支持ANSI颜色输出以及制表符完成。 如果你需要运行一个本地命令，你可以使用！ 字首。 点击ctrl-c将退出嵌入式shell。

## Adding extensions to the CLI

您可以使用install命令向CLI添加扩展。 该命令在格式组中使用一组或多组artifact坐标：group:artifact:version。 例如：

$ spring install com.example:spring-boot-cli-extension:1.0.0.RELEASE

除了安装由您提供的坐标标识的artifacts'之外，所有artifacts'的依赖关系也将被安装。

要卸载依赖项，请使用uninstall命令。 与安装命令一样，它在格式组中使用一组或多组artifact坐标：group:artifact:version。 例如：

$ spring uninstall com.example:spring-boot-cli-extension:1.0.0.RELEASE

它将卸载由您提供的坐标标识的artifacts及其相关性。 要卸载所有其他依赖关系，可以使用--all选项。 例如：

$ spring uninstall --all

# Developing application with the Groovy beans DSL

Spring Framework 4.0对beans（borrowed from [Grails](http://grails.org/)）“DSL”具有本地支持，并且可以使用相同的格式将Groovy应用程序脚本中的bean定义嵌入到其中。 有时候，这是包含中间件声明等外部功能的好方法。 例如：

@Configuration

**class** Application **implements** CommandLineRunner {

@Autowired SharedService service

@Override

**void** run(String... args) { println service.message

}

}

**import** my.company.SharedService beans {

service(SharedService) { message = ***"Hello World"***

}

}

只要他们停留在top level，就可以将类声明与beans {}混合在同一个文件中，或者如果您愿意，可以将beans DSL放在单独的文件中。

# Configuring the CLI with settings.xml

Spring Boot CLI使用Maven的依赖关系解析引擎Aether来解决依赖关系。 CLI使用~/.m2/settings.xml中的Maven配置来配置Aether。 CLI遵循以下配置设置：

* Offline
* Mirrors
* Servers
* Proxies
* Profiles
* Activation
* Repositories
* Active profiles

有关更多信息，请参阅[Maven’s settings documentation](https://maven.apache.org/settings.html) 。

# What to read next

在GitHub仓库中有一些[sample groovy scripts](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-cli/samples) ，可以用来试用Spring Boot CLI。 整个[source code](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-cli/src/main/java/org/springframework/boot/cli)中还有大量的Javadoc。

如果你发现你达到了CLI工具的极限，你可能会想要把你的应用程序转换成完整的Gradle或者Maven构建的“groovy project”。 下一节将介绍Spring Boot的[*Build*](#_bookmark407)[*tool plugins*](#_bookmark407) ，您可以使用Gradle或Maven。

**Part VIII. Build tool plugins**

Spring Boot provides build tool plugins for Maven and Gradle. The plugins offer a variety of features, including the packaging of executable jars. This section provides more details on both plugins, as well as some help should you need to extend an unsupported build system. If you are just getting started, you might want to read “[Chapter 13, *Build systems*](#_bookmark35)” from the [Part III, “Using Spring Boot”](#_bookmark34) section first.

# Spring Boot Maven plugin

[Spring Boot Maven Plugin](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/)在Maven中提供了Spring Boot支持，允许你打包可执行的jar或war档案并运行一个应用程序“in-place”。 要使用它，你必须使用Maven 3.2（或更高版本）。

**Note**

有关完整的插件文档，请参阅[Spring Boot Maven Plugin Site](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/) 站点。

## Including the plugin

要使用Spring Boot Maven Plugin，只需在pom.xml的插件部分包含适当的XML

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

**<project xmlns**=[**"http://maven.apache.org/POM/4.0.0"**](http://maven.apache.org/POM/4.0.0) **xmlns:xsi**=[**"http://www.w3.org/2001/XMLSchema-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xsi:schemaLocation**=[**"http://maven.apache.org/POM/4.0.0**](http://maven.apache.org/POM/4.0.0)[**http://maven.apache.org/xsd/maven-4.0.0.xsd"**](http://maven.apache.org/xsd/maven-4.0.0.xsd)**>**

**<modelVersion>**4.0.0**</modelVersion>**

*<!-- ... -->*

**<build>**

**<plugins>**

**<plugin>**

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

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

**<version>**1.5.8.RELEASE**</version>**

**<executions>**

**<execution>**

**<goals>**

**<goal>**repackage**</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

**</plugins>**

**</build>**

**</project>**

这个配置将重新包装一个在Maven生命周期的包装阶段生成的jar或war。 以下示例显示目标目录中的重新打包的jar以及原始jar：

$ mvn package

$ ls target/\*.jar

target/myproject-1.0.0.jar target/myproject-1.0.0.jar.original

如果你不包括上面的<execution />配置，你可以自己运行插件（但是只有在使用包目标的时候）。 例如：

$ mvn package spring-boot:repackage

$ ls target/\*.jar

target/myproject-1.0.0.jar target/myproject-1.0.0.jar.original

如果您正在使用里程碑(milestone)或快照(snapshot)版本，则还需要添加适当的pluginRepository元素：

**<pluginRepositories>**

**<pluginRepository>**

**<id>**spring-snapshots**</id>**

**<url>**<http://repo.spring.io/snapshot>**</url>**

**</pluginRepository>**

**<pluginRepository>**

**<id>**spring-milestones**</id>**

**<url>**<http://repo.spring.io/milestone>**</url>**

**</pluginRepository>**

**</pluginRepositories>**

## Packaging executable jar and war files

一旦spring-boot-maven-plugin包含在你的pom.xml文件中，它就会自动尝试使用spring-boot：repackage目标来重写归档文件。 你应该配置你的项目使用通常的包装元素来构建一个jar或war（适当的时候）：

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

**<project xmlns**=[**"http://maven.apache.org/POM/4.0.0"**](http://maven.apache.org/POM/4.0.0) **xmlns:xsi**=[**"http://www.w3.org/2001/XMLSchema-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xsi:schemaLocation**=[**"http://maven.apache.org/POM/4.0.0**](http://maven.apache.org/POM/4.0.0)[**http://maven.apache.org/xsd/maven-4.0.0.xsd"**](http://maven.apache.org/xsd/maven-4.0.0.xsd)**>**

*<!-- ... -->*

**<packaging>**jar**</packaging>**

*<!-- ... -->*

**</project>**

在打包(package)阶段，您的现有存档将被Spring Boot增强。 要启动的主类可以使用配置选项指定，也可以通过以常用方式向清单中添加Main-Class属性来指定。 如果你不指定一个主类，插件将使用public static void main（String [] args）方法搜索一个类。

要构建和运行项目工件，可以键入以下内容：

$ mvn package

$ java -jar target/mymodule-0.0.1-SNAPSHOT.jar

要构建可执行文件并将其部署到外部容器中，需要将嵌入容器依赖项标记为“provided”，例如：

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

**<project xmlns**=[**"http://maven.apache.org/POM/4.0.0"**](http://maven.apache.org/POM/4.0.0) **xmlns:xsi**=[**"http://www.w3.org/2001/XMLSchema-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xsi:schemaLocation**=[**"http://maven.apache.org/POM/4.0.0**](http://maven.apache.org/POM/4.0.0)[**http://maven.apache.org/xsd/maven-4.0.0.xsd"**](http://maven.apache.org/xsd/maven-4.0.0.xsd)**>**

*<!-- ... -->*

**<packaging>**war**</packaging>**

*<!-- ... -->*

**<dependencies>**

**<dependency>**

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

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

**</dependency>**

**<dependency>**

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

**<artifactId>**spring-boot-starter-tomcat**</artifactId>**

**<scope>**provided**</scope>**

**</dependency>**

*<!-- ... -->*

**</dependencies>**

**</project>**

**Tip**

有关如何创建可部署的war文件的更多详细信息，请参阅““[Section 85.1, “Create a deployable war file”](#_bookmark563)”部分。

高级配置选项和示例可在插件信息页面([plugin info page](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/))中找到。

# Spring Boot Gradle plugin

Spring Boot Gradle插件在Gradle中提供Spring Boot支持，允许您打包可执行的jar或war档案，运行Spring Boot应用程序并使用spring-boot-dependencies提供的依赖管理。

## Including the plugin

要使用Spring Boot Gradle Plugin，请使用plugins块进行配置：

plugins {

id ***'org.springframework.boot'*** version ***'1.5.8.RELEASE'***

}

## Gradle dependency management

spring-boot插件会自动应用[Dependency Management Plugin](https://github.com/spring-gradle-plugins/dependency-management-plugin/) ，并将其配置为导入spring-boot-starter-parent bom。 这为Maven用户提供了一个类似的依赖管理经验。 例如，它允许您在声明bom中管理的依赖项时省略版本号。 为了使用这个功能，只需要以通常的方式声明依赖关系，但是保留版本号为空：

dependencies {

compile(***"org.springframework.boot:spring-boot-starter-web"***) compile(***"org.thymeleaf:thymeleaf-spring4"***) compile(***"nz.net.ultraq.thymeleaf:thymeleaf-layout-dialect"***)

}

**Note**

您声明的spring-boot gradle插件的版本决定了导入的spring-boot-starter-parent bom的版本（这确保了版本总是可重复的）。 您应该始终将spring-boot gradle插件的版本设置为您希望使用的实际Spring Boot版本。 所提供版本的细节可以在[appendix](#_bookmark607)中找到。

要了解有关Dependency Management插件功能的更多信息，请参阅其[documentation](https://github.com/spring-gradle-plugins/dependency-management-plugin/blob/master/README.md)。

## Packaging executable jar and war files

一旦spring-boot插件已经应用到您的项目，它会自动尝试重写档案，使他们可执行使用[bootRepackage task](#_bookmark417)。 您应该配置您的项目以通常的方式建立一个jar或war（如适用）。

要启动的主类可以使用配置选项指定，也可以通过向清单添加Main-Class属性来指定。 如果你不指定一个主类，插件将使用public static void main（String [] args）方法搜索一个类。

**Tip**

检查[Section 67.6, “Repackage configuration”](#_bookmark417) ”，查看配置选项的完整列表。

要构建和运行项目工件，可以键入以下内容：

$ gradle build

$ java -jar build/libs/mymodule-0.0.1-SNAPSHOT.jar

要构建一个既可执行又可部署到外部容器的war文件，您需要将嵌入的容器依赖项标记为属于war插件的providedRuntime配置，例如：

...

apply plugin: ***'war'***

war {

baseName = ***'myapp'***

version = ***'0.5.0'***

}

repositories { jcenter()

maven { url [***"http://repo.spring.io/libs-snapshot"***](http://repo.spring.io/libs-snapshot) }

}

dependencies {

compile(***"org.springframework.boot:spring-boot-starter-web"***) providedRuntime(***"org.springframework.boot:spring-boot-starter-tomcat"***)

...

}

**Tip**

有关如何创建可部署的war文件的更多详细信息，请参阅“Section 85.1, “Create a deployable war file”部分。

## Running a project in-place

要在不建立jar的情况下运行一个项目，你可以使用“bootRun”任务：

$ gradle bootRun

如果[devtools](#_bookmark67) 已经添加到您的项目中，它会自动监控您的应用程序的变化。 或者，您也可以运行应用程序，以便您的静态类路径资源（即默认情况下在src/main/ resources中）可在实时应用程序中重新加载，这在开发时可能会有所帮助。

bootRun {

addResources = true

}

使静态类路径资源可重新加载意味着bootRun不使用processResources任务的输出，即，当使用bootRun调用时，应用程序将以未处理的形式使用资源。

## Spring Boot plugin configuration

gradle插件会自动扩展您的构建脚本DSL，并使用springBoot元素来进行Boot插件的全局配置。 像使用其他Gradle扩展一样设置适当的属性（请参阅下面的配置选项列表）：

springBoot {

backupSource = false

}

## Repackage configuration

该插件添加一个bootRepackage任务，您也可以直接配置，例如：

bootRepackage {

mainClass = ***'demo.Application'***

}

以下配置选项可用：

|  |  |
| --- | --- |
| **Name** | **Description** |
| enabled  mainClass | Boolean flag to switch the repackager off (sometimes useful if you want the other Boot features but not this one)  The main class that should be run. If not specified, and you have applied the application plugin, the mainClassName project property will be used. If the application plugin has not been applied or no mainClassName has been specified, the archive will be searched for a suitable class. "Suitable" means a unique class with a well-formed main() method (if more than one is found the build will fail). If you have applied the application plugin, the main class can also be specified via its "run" task (main property) and/or its "startScripts" task (mainClassName property) as an alternative to using the "springBoot" configuration. |
| classifier | A file name segment (before the extension) to add to the archive, so that the original is preserved in its original location. Defaults to null in which case the archive is repackaged in place. The default is convenient for many purposes, but if you want to use  the original jar as a dependency in another project you must use a classifier to define the executable archive. |
| withJarTask | The name or value of the Jar task (defaults to all tasks of type  Jar) which is used to locate the archive to repackage. |
| customConfiguration  executable | The name of the custom configuration which is used to populate the nested lib directory (without specifying this you get all compile and runtime dependencies).  Boolean flag to indicate if jar files are fully executable on Unix like operating systems. Defaults to false. |
| embeddedLaunchScript  embeddedLaunchScriptProp | The embedded launch script to prepend to the front of the jar if it is fully executable. If not specified the 'Spring Boot' default script will be used.  erAtdideitisonal properties that to be expanded in the launch script. The default script supports a mode property which can contain the values auto, service or run. |
| excludeDevtools | Boolean flag to indicate if the devtools jar should be excluded from the repackaged archives. Defaults to true. |

## Repackage with custom Gradle configuration

有时可能不包装从编译，运行时和提供的范围解决的默认依赖关系更合适。 如果创建的可执行jar文件打算按原样运行，则需要将所有依赖项嵌套在其中; 但是，如果计划要分解jar文件并手动运行主类，则可能已经通过CLASSPATH提供了一些库。 在这种情况下，您可以使用不同的依赖关系重新包装您的jar。

使用自定义配置将自动禁止从编译，运行时和提供的作用域进行依赖关系解析。 自定义配置可以全局定义（在springBoot部分内）或每个任务。

task clientJar(type: Jar) { appendix = ***'client'***

from sourceSets.main.output exclude(***'\*\*/\*Something\*'***)

}

task clientBoot(type: BootRepackage, dependsOn: clientJar) { withJarTask = clientJar

customConfiguration = ***"mycustomconfiguration"***

}

在上面的例子中，我们创建了一个新的clientJar Jar任务来打包来自编译源的定制文件集。 然后，我们创建了一个新的clientBoot BootRepackage任务，并指示它只处理clientJar任务和mycustomconfiguration。

configurations {

mycustomconfiguration.exclude group: ***'log4j'***

}

dependencies {

mycustomconfiguration configurations.runtime

}

BootRepackage中提到的配置是普通的[Gradle configuration](https://docs.gradle.org/2.14.1/dsl/org.gradle.api.artifacts.Configuration.html)。 在上面的例子中，我们创建了一个名为mycustomconfiguration的新配置，指示它从运行时派生并排除log4j组。 如果执行clientBoot任务，则重新打包的引导jar将具有运行时的所有依赖关系，但不包含log4j jar。

### Configuration options

以下配置选项可用：

|  |  |
| --- | --- |
| **Name**  mainClass | **Description**  The main class that should be run by the executable archive. |
| providedConfiguration  backupSource | The name of the provided configuration (defaults to  providedRuntime).  If the original source archive should be backed-up before being repackaged (defaults to true). |
| customConfiguration layout | The name of the custom configuration.  The type of archive, corresponding to how the dependencies are laid out inside (defaults to a guess based on the archive type).  See [available layouts for more details](#_bookmark420). |

|  |  |
| --- | --- |
| **Name** | **Description** |
| layoutFactory  requiresUnpack | A layout factory that can be used if a custom layout is required. Alternative layouts can be provided by 3rd parties. Layout factories are only used when layout is not specified.  A list of dependencies (in the form “groupId:artifactId” that must be unpacked from fat jars in order to run. Items are still packaged into the fat jar, but they will be automatically unpacked when it runs. |

### Available layouts

布局属性配置归档的格式以及是否包含引导加载程序。 以下布局可用：

|  |  |  |
| --- | --- | --- |
| **Name**  JAR | **Description**  Regular executable [JAR layout](#_bookmark595). | **Executable**  Yes |
| WAR | Executable [WAR layout](#_bookmark596). provided dependencies are placed in WEB-INF/lib-provided to avoid any clash when the war is deployed in a servlet container. | Yes |
| ZIP (alias to DIR) MODULE | Similar to JAR layout, using [PropertiesLauncher](#_bookmark602).  Bundle dependencies (excluding those with provided  scope) and project resources. | Yes No |
| NONE | Bundle all dependencies and project resources. | No |

### Using a custom layout

如果您对如何在重新打包的jar内部安排依赖项和加载器类有自定义要求，则可以使用自定义布局。 任何定义一个或多个LayoutFactory实现的库都可以添加到构建脚本依赖项中，然后布局工厂在springBoot配置中可用。 例如：

buildscript { repositories {

mavenCentral()

}

dependencies {

classpath(***"org.springframework.boot:spring-boot-gradle-plugin:1.5.8.RELEASE"***) classpath(***"com.example:custom-layout:1.0.0"***)

}

}

springBoot {

layoutFactory = **new** com.example.CustomLayoutFactory()

}

**Note**

如果构建类路径上只有一个自定义的LayoutFactory，并且在META-INF / spring.factories中列出，则不必在springBoot配置中显式设置它。 布局工厂仅在未指定明确的布局时使用。

## Understanding how the Gradle plugin works

将Spring引导应用于Gradle项目时，会自动创建一个名为bootRepackage的默认任务。 bootRepackage任务依赖于Gradle汇编任务，执行时会尝试查找所有限定符为空的jar工件（即测试和源代码jar被自动跳过）。

由于bootRepackage发现所有创建的jar工件，Gradle任务执行顺序非常重要。 大多数项目只创建一个jar文件，所以通常这不是问题; 但是，如果您打算创建一个更复杂的项目设置，使用自定义的Jar和BootRepackage任务，则可以考虑进行一些调整。

如果你只是从你的项目中创建自定义的jar文件，你可以简单的关闭默认的jar和

bootRepackage任务：

jar.enabled = false bootRepackage.enabled = false

另一个选项是指示默认的bootRepackage任务只能使用默认的jar任务。

bootRepackage.withJarTask = jar

如果您有一个默认的项目设置，在其中创建并重新打包主jar文件，并且您还想创建其他自定义jar，则可以将自定义重新打包任务组合在一起，并使用dependsOn，以便bootJars任务将在缺省 bootRepackage任务被执行：

task bootJars

bootJars.dependsOn = [clientBoot1,clientBoot2,clientBoot3] build.dependsOn(bootJars)

以上所有的调整通常用于避免已经创建的启动jar重新打包的情况。 重新包装现有的启动jar不会破坏任何东西，但是您可能会发现它包含不必要的依赖关系。

## Publishing artifacts to a Maven repository using Gradle

如果你declaring dependencies without versions，并且你想发布工件到一个Maven仓库，你将需要配置Maven出版物，详细介绍Spring Boot的依赖管理。 这可以通过将其配置为发布从spring-boot-starter-parent继承的poms或从spring-boot-dependencies导入依赖关系管理来实现。 这个配置的确切细节取决于你如何使用Gradle以及你如何发布工件。

### Configuring Gradle to produce a pom that inherits dependency management

以下是配置Gradle生成从spring-boot-starter-parent继承的pom的示例。 请参阅[Gradle User Guide](https://docs.gradle.org/2.14.1/userguide/userguide.html) 了解更多信息。

uploadArchives { repositories {

mavenDeployer { pom {

project {

parent {

groupId ***"org.springframework.boot"***

artifactId ***"spring-boot-starter-parent"***

version ***"1.5.8.RELEASE"***

}

}

}

}

}

}

### Configuring Gradle to produce a pom that imports dependency management

以下是配置Gradle生成导入由spring-boot-dependencies提供的依赖关系管理的pom的示例。 请参阅[Gradle User Guide](http://www.gradle.org/docs/current/userguide/userguide.html) 了解更多信息。

uploadArchives { repositories {

mavenDeployer { pom {

project {

dependencyManagement { dependencies {

dependency {

groupId ***"org.springframework.boot"*** artifactId ***"spring-boot-dependencies"*** version ***"1.5.8.RELEASE"***

type ***"pom"***

scope ***"import"***

}

}

}

}

}

}

}

}

# Spring Boot AntLib module

Spring Boot AntLib模块为Apache Ant提供了基本的Spring Boot支持。 您可以使用该模块来创建可执行的jar。 要使用这个模块，你需要在你的build.xml中声明一个额外的spring-boot命名空间：

**<project xmlns:ivy**=**"antlib:org.apache.ivy.ant" xmlns:spring-boot**=**"antlib:org.springframework.boot.ant" name**=**"myapp" default**=**"build">**

...

**</project>**

您需要记住使用-lib选项启动Ant，例如：

$ ant -lib <folder containing spring-boot-antlib-1.5.8.RELEASE.jar>

**Tip**

“使用Spring Boot”部分包含一个更完整的使用Apache Ant的例子

[using Apache Ant withspring-boot-antlib](#_bookmark43)

## Spring Boot Ant tasks

一旦声明了spring-boot-antlib名称空间，就可以使用下列附加任务。

### spring-boot:exejar

exejar任务可以用来创建一个Spring Boot可执行jar。 该任务支持以下属性：

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Required** |
| destfile | The destination jar file to create | Yes |
| classes start-class | The root directory of Java class files The main application class to run | Yes  No *(default is first class found declaring a main method)* |

以下嵌套元素可以与任务一起使用：

|  |
| --- |
| **Element Description** |
| resources One or more [Resource Collections](http://ant.apache.org/manual/Types/resources.html#collection) describing a set of [Resources](http://ant.apache.org/manual/Types/resources.html) that should be added to the content of the created jar file.  lib One or more [Resource Collections](http://ant.apache.org/manual/Types/resources.html#collection) that should be added to the set of jar libraries that make up the runtime dependency classpath of the application. |

### Examples

**Specify start-class.**

**<spring-boot:exejar destfile**=**"target/my-application.jar" classes**=**"target/classes" start-class**=**"com.foo.MyApplication">**

**<resources>**

**<fileset dir**=**"src/main/resources" />**

**</resources>**

**<lib>**

**<fileset dir**=**"lib" />**

**</lib>**

**</spring-boot:exejar>**

**Detect(检测) start-class.**

**<exejar destfile**=**"target/my-application.jar" classes**=**"target/classes">**

**<lib>**

**<fileset dir**=**"lib" />**

**</lib>**

**</exejar>**

## spring-boot:findmainclass

findmainclass任务由exejar在内部使用来定位一个声明main的类。 如果需要，您也可以直接在您的构建中使用此任务。 支持以下属性

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Required** |
| classesroot | The root directory of Java class files | Yes *(unless mainclass is specified)* |
| mainclass | Can be used to short-circuit the main | No |
|  | class search |  |
| property | The Ant property that should be set | No *(result will be logged if unspecified)* |
|  | with the result |  |

### Examples

**Find and log.**

**<findmainclass classesroot**=**"target/classes" />**

**Find and set.**

**<findmainclass classesroot**=**"target/classes" property**=**"main-class" />**

**Override and set.**

**<findmainclass mainclass**=**"com.foo.MainClass" property**=**"main-class" />**

# Supporting other build systems

如果你想使用Maven，Gradle或Ant以外的构建工具，你可能需要开发自己的插件。 可执行文件需要遵循特定的格式，某些条目需要以非压缩格式写入（详见附录中[*executable jar format*](#_bookmark593) ）。

Spring Boot Maven和Gradle插件都使用spring-boot-loader-tools来生成jar。 如果需要，您也可以自由直接使用这个库。

## Repackaging archives

要重新打包现有的归档(archive)文件，使其成为独立的可执行归档文件，请使用org.springframework.boot.loader.tools.Repackager。 Repackager类采用引用现有jar或war归档的单个构造函数参数。 使用两种可用的repackage()方法之一来替换原始文件或写入新的目标。 重新打包程序运行之前，还可以配置各种设置。

## Nested(嵌套) libraries

重新打包存档(archive)时，可以使用org.springframework.boot.loader.tools.Libraries接口包含对依赖项文件的引用。 我们在这里不提供任何具体的库实现，因为它们通常构建系统特定的。

如果您的存档(archive)已包含库，则可以使用Libraries.NONE。

## Finding a main class

如果不使用Repackager.setMainClass（）来指定主类，则重新打包程序将使用[ASM](http://asm.ow2.org/)读取类文件，并尝试使用公共静态void main（String [] args）方法找到合适的类。 如果找到多个候选人(candidate)，则会引发异常。

## Example repackage implementation

这里是一个典型的重新包装(repackage)的例子：

Repackager repackager = **new** Repackager(sourceJarFile); repackager.setBackupSource(false); repackager.repackage(**new** Libraries() {

@Override

**public void** doWithLibraries(LibraryCallback callback) **throws** IOException {

*// Build system specific implementation, callback for each dependency*

*// callback.library(new Library(nestedFile, LibraryScope.COMPILE));*

}

});

# What to read next

如果您对构建工具插件的工作方式感兴趣，可以查看GitHub上的[spring-boot-tools](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-tools) 模块。 附录中介绍了有关[executable jar format](#_bookmark593) 的更多技术细节。

如果你有特定的构建相关的问题，你可以看看“[how-to](#_bookmark438)”的指南。

**Part IX. ‘How-to’ guides**

This section provides answers to some common ‘how do I do that…’ type of questions that often arise when using Spring Boot. This is by no means an exhaustive list, but it does cover quite a lot.

If you are having a specific problem that we don’t cover here, you might want to check out [stackoverflow.com](http://stackoverflow.com/tags/spring-boot) to see if someone has already provided an answer; this is also a great place to ask new questions (please use the spring-boot tag).

We’re also more than happy to extend this section; If you want to add a ‘how-to’ you can send us a [pull request](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE).

# Spring Boot application

## 创建你自己的 FailureAnalyzer

[FailureAnalyzer](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/diagnostics/FailureAnalyzer.html) 是在启动时拦截异常并将其转换为可读的消息并包装到[FailureAnalysis](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/diagnostics/FailureAnalysis.html)中的好方法。 Spring Boot为应用程序上下文相关异常，JSR-303验证等提供了这样的分析器。 创建你自己的实际上是非常容易的。

AbstractFailureAnalyzer是FailureAnalyzer的一个方便的扩展，它检查异常中是否存在指定的异常类型。 你可以扩展它，这样你的实现只有在实际存在时才有机会处理这个异常。 如果由于某种原因你不能处理异常，则返回null给另一个实现一个处理异常的机会。

FailureAnalyzer实现将在META-INF / spring.factories中注册：以下注册ProjectConstraintViolationFailureAnalyzer：

**org.springframework.boot.diagnostics.FailureAnalyzer**=\ com.example.ProjectConstraintViolationFailureAnalyzer

## auto-configuration 疑难解答(Troubleshoot auto-configuration)

Spring Boot自动配置尽最大努力去做“do the right thing”，但有时候事情会失败，很难说清原因。

在任何Spring Boot ApplicationContext中都有一个非常有用的ConditionEvaluationReport。 你会看到它，如果你启用DEBUG日志输出。 如果您使用spring-boot-actuator，那么还有一个自动配置端点以JSON呈现报表。 使用它来调试应用程序，看看Spring Boot在运行时添加了哪些功能（哪些功能没有添加）。

通过查看源代码和Javadoc，可以回答更多的问题。 一些经验法则：

* 查找称为\* AutoConfiguration的类，并阅读它们的源代码，特别是@ Conditional \*注释，以找出它们启用的功能以及何时启用的功能。 Add --debug到命令行或系统属性-Ddebug获取在您的应用程序中所做的所有自动配置决策的控制台日志。 在一个正在运行的Actuator应用程序中查看autoconfig端点（'/ autoconfig'或JMX等价物）以获取相同的信息。
* 查找@ConfigurationProperties类（例如[ServerProperties](https://github.com/spring-projects/spring-boot/blob/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java)）并从那里读取可用的外部配置选项。 @ConfigurationProperties有一个名称属性作为外部属性的前缀，因此ServerProperties有prefix =“server”，它的配置属性是server.port，server.address等。在一个正在运行的Actuator应用程序中查看configprops端点。
* 查找使用RelaxedPropertyResolver将配置值明确地从环境中提取出来。它通常与前缀一起使用
* 查找直接绑定到环境的@Value注解。这不如RelaxedPropertyResolver方法灵活，但是允许一些放松的绑定，特别是OS环境变量(所以CAPITALS\_AND\_UNDERSCORES是period.separated的同义词)。
* 查找@ConditionalOnExpression注释，用于响应SpEL表达式开启和关闭功能，通常使用从环境中解析的占位符进行评估。

## 在开始之前自定义Environment或Application

SpringApplication具有ApplicationListeners和ApplicationContextInitializers，用于将自定义应用于上下文或环境。 Spring Boot从META-INF / spring.factories中加载了许多这样的自定义内部使用。 有多种方法可以注册更多的：

* 在运行之前，通过调用SpringApplication上的addListeners和addInitializers方法为每个应用程序编程。
* 声明的每个应用程序通过设置context.initializer.classes或context.listener.classes
* 通过添加META-INF / spring.factories并打包应用程序都用作库的jar文件，为所有应用程序声明性地进行声明。

SpringApplication将一些特殊的ApplicationEvents发送到监听器（甚至在创建上下文之前），然后为ApplicationContext发布的事件注册监听器。 有关完整列表，请参见[*Section 23.5, “Application events and listeners”*](#_bookmark91) 应用程序事件和侦听器”。

在使用EnvironmentPostProcessor刷新应用程序上下文之前，也可以定制环境。 每个实现应该在META-INF / spring.factories中注册：

**org.springframework.boot.env.EnvironmentPostProcessor**=com.example.YourEnvironmentPostProcessor

该实现可以加载任意文件并将其添加到环境中。 例如，这个例子从类路径中加载一个YAML配置文件：

**public class** EnvironmentPostProcessorExample **implements** EnvironmentPostProcessor {

**private final** YamlPropertySourceLoader loader = **new** YamlPropertySourceLoader(); @Override

**public void** postProcessEnvironment(ConfigurableEnvironment environment, SpringApplication application) {

Resource path = **new** ClassPathResource(***"com/example/myapp/config.yml"***); PropertySource<?> propertySource = loadYaml(path); environment.getPropertySources().addLast(propertySource);

}

**private** PropertySource<?> loadYaml(Resource path) {

**if** (!path.exists()) {

**throw new** IllegalArgumentException(***"Resource "*** + path + ***" does not exist"***);

}

**try** {

**return this**.loader.load(***"custom-resource"***, path, null);

}

**catch** (IOException ex) {

**throw new** IllegalStateException(

***"Failed to load yaml configuration from "*** + path, ex);

}

}

}

**Tip**

环境已经准备好了Spring Boot默认加载的所有常用属性资源。 因此可以从环境中获取文件的位置。 此示例在列表的末尾添加自定义资源属性源，以便在任何常用其他位置定义的键优先。 自定义实现可能显然定义了另一个订(order)。

**Note**

在@SpringBootApplication上使用@PropertySource看起来既方便又简单，可以在环境中加载自定义资源，我们不建议这样做，因为在BootContext刷新之前，Spring Boot会准备环境。 任何通过@PropertySource定义的键都会被加载太迟而不能对自动配置产生任何影响。

## 构建一个ApplicationContext层次结构(添加parent或root context)

您可以使用ApplicationBuilder类创建parent/child ApplicationContext层次结构。 有关更多信息，请参见[*Section 23.4, “Fluent builder API”*](#_bookmark90)。

## 创建一个非Web应用程序

并非所有的Spring应用程序都必须是Web应用程序（或Web服务）。 如果你想在主要方法中执行一些代码，而且还要引导一个Spring应用程序来设置要使用的基础结构，那么使用Spring Boot的SpringApplication功能很容易。 SpringApplication根据是否认为需要Web应用程序来更改它的ApplicationContext类。 您可以做的第一件事就是将servlet API依赖关系从类路径中移除。 如果你不能这样做（例如，你从同一个代码库运行2个应用程序），那么你可以显式地在你的SpringApplication实例上调用setWebEnvironment（false），或者设置applicationContextClass属性（通过Java API或者外部属性）。 要作为业务逻辑运行的应用程序代码可以作为CommandLineRunner实现，并以@Bean定义的形式放入上下文中。

# Properties(属性) & configuration(配置)

## 在构建(build)是自动扩展属性(property)

不是硬编码(hardcoding)在项目的构建配置中指定的一些属性，而是使用现有的构建配置来自动扩展它们。 这在Maven和Gradle中都是可能的。

### 使用Maven自动扩展属性(property)

您可以使用资源过滤自动扩展Maven项目的属性。 如果使用spring-boot-starter-parent，则可以通过@ .. @占位符引用Maven的“项目属性”。

[**app.encoding**=@project.build.sourceEncoding@](mailto:app.encoding%3D@project.build.sourceEncoding@) [**app.java.version**=@java.version@](mailto:app.java.version%3D@java.version@)

**Note**

只有生产配置以这种方式进行过滤（即，不在src / test / resources上应用过滤）。

**Tip**

如果启用addResources标志(flag)，则spring-boot：run可以直接将src / main / resources添加到类路径（用于热重载）。 这规避了资源过滤和这个功能。 您可以使用exec：java目标，也可以自定义插件的配置，请参阅插件使用页面([plugin usage page](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/usage.html) )了解更多详细信息。

如果你不使用starter parent，你需要在你的pom.xml中（在<build />元素中）：

并且 (inside <plugins/>):

**<plugin>**

**<groupId>**org.apache.maven.plugins**</groupId>**

**<artifactId>**maven-resources-plugin**</artifactId>**

**<version>**2.7**</version>**

**<configuration>**

**<delimiters>**

**<delimiter>**@**</delimiter>**

**</delimiters>**

**<useDefaultDelimiters>**false**</useDefaultDelimiters>**

**</configuration>**

**</plugin>**

**Note**

如果在配置中使用标准的Spring占位符（例如$ {foo}），则useDefaultDelimiters属性非常重要。 如果这个属性没有被设置为false，这些可能会被扩展。

### 使用Gradle自动扩展属性(property)

您可以通过配置Java插件来自动扩展Gradle项目中的属性processResources任务(task)这样做：

processResources { expand(project.properties)

}

然后，您可以通过占位符引用您的Gradle项目的属性，例如

**app.name**=${name} **app.description**=${description}

**Note**

Gradle的expand方法使用Groovy的SimpleTemplateEngine来转换$ {..}标记。 $ {..}样式与Spring自己的属性占位符机制冲突。 要使用Spring属性占位符和自动扩展，Spring属性占位符需要像\ $ {..}一样被转义。

## 外部化SpringApplication的配置

SpringApplication具有bean属性（主要是setter），因此您可以在创建应用程序时使用其Java API来修改其行为。 或者你可以使用spring.main.\*中的属性外化配置。 例如。 在你可能有的application.properties中。

**spring.main.web-environment**=false **spring.main.banner-mode**=off

然后Spring Boot banner将不会在启动时被打印，而应用程序将不会是一个Web应用程序。

**Note**

上面的例子也演示了如何灵活绑定允许在属性名称中使用下划线（\_）以及破折号（ - ）。

在外部配置中定义的属性会覆盖通过Java API指定的值，但是用于创建ApplicationContext的资源的显着例外。 我们来考虑这个应用程序

**new** SpringApplicationBuilder()

.bannerMode(Banner.Mode.OFF)

.sources(demo.MyApp.**class**)

.run(args);

与以下配置一起使用：

**spring.main.sources**=com.acme.Config,com.acme.ExtraConfig **spring.main.banner-mode**=console

实际应用程序现在将显示banner（由配置重写）并使用ApplicationContext的三个源（按此顺序）：demo.MyApp，com.acme.Config，com.acme.ExtraConfig。

## 更改应用程序的扩展属性的位置

默认情况下，来自不同来源的属性会按照定义的顺序添加到Spring环境中（具体顺序请参见“[*Chapter 24, Externalized Configuration*](#_bookmark97) ”）。

一个很好的方法来增加和修改这个是添加@PropertySource注释到你的应用程序源。 传递给SpringApplication静态方便方法的类以及使用setSources（）方法添加的类将被检查以确定它们是否具有@PropertySources，如果它们的确如此，那么这些属性将尽早添加到Environment中，以便在ApplicationContext生命周期的所有阶段中使用。 以这种方式添加的属性的优先级低于使用默认位置（例如，application.properties），系统属性，环境变量或命令行添加的属性的优先级。

您还可以提供系统属性（或环境变量）来更改行为：

* spring.config.name（SPRING\_CONFIG\_NAME），默认为应用程序作为文件名的根。
* spring.config.location（SPRING\_CONFIG\_LOCATION）是要加载的文件（例如类路径资源或URL）。 为此文档设置了一个单独的环境属性源，可以通过系统属性，环境变量或命令行覆盖它。

不管你在环境中设置了什么，Spring Boot将总是加载application.properties，如上所述。 如果使用YAML，默认情况下，具有“.yml”扩展名的文件也会添加到列表中。

Spring Boot会记录在DEBUG级别加载的配置文件以及在TRACE级别未找到的候选项。

有关更多详细信息，请参阅[ConfigFileApplicationListener](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/context/config/ConfigFileApplicationListener.java) 。

## 使用’short’命令行参数( ‘short’ command line arguments)

有些人喜欢使用（例如）--port = 9000而不是--server.port = 9000来在命令行上设置配置属性。 您可以通过在application.properties中使用占位符来轻松地启用此功能。

**server.port**=${port:8080}

**Tip**

如果从spring-boot-starter-parent POM继承，maven-resources-plugins的默认过滤器标记已从$ {\*}更改为@（即@ maven.token @而不是$ {maven.token }）来防止与Spring风格的占位符冲突。 如果您已经直接为application.properties启用了maven过滤，那么您可能还需要将默认的过滤令牌更改为使用[other delimiters](http://maven.apache.org/plugins/maven-resources-plugin/resources-mojo.html#delimiters)。

**Note**

在这种特定情况下，端口绑定将在Heroku和Cloud Foundry等PaaS环境中工作，因为在这两个平台中，PORT环境变量是自动设置的，Spring可以绑定到大写的环境属性同义词。

## 使用YAML作为外部属性( YAML for external properties)

YAML是JSON的一个超集，因此是以分层格式存储外部属性的非常方便的语法。 例如。

**spring**:

**application**:

**name**: cruncher

**datasource**:

**driverClassName**: com.mysql.jdbc.Driver

**url**: jdbc:mysql://localhost/test

**server**:

**port**: 9000

创建一个名为application.yml的文件，并将其粘贴到你的类路径的根目录下，然后将snakeyaml添加到你的依赖项中（Maven坐标org.yaml：snakeyaml，如果你使用spring-boot-starter，就已经包含了）。 一个YAML文件被解析为一个Java Map <String，Object>（就像一个JSON对象），Spring Boot展平地图，这样它就是一层深度的，并且有周期分隔的键，就像人们习惯的 Java中的属性文件。

上面的示例YAML对应于一个application.properties文件

**spring.application.name**=cruncher **spring.datasource.driverClassName**=com.mysql.jdbc.Driver **spring.datasource.url**=jdbc:mysql://localhost/test **server.port**=9000

有关YAML的更多信息，请参见[*Section 24.6, “Using YAML instead of Properties”*](#_bookmark103) 使用YAML而不是 .propertyes文件”。

## 设置活动的 Spring profiles

Spring环境有一个API，但通常你可以设置一个System属性（spring.profiles.active）或一个OS环境变量（SPRING\_PROFILES\_ACTIVE）。 例如。 用-D参数启动你的应用程序（记住把它放在主类或jar文件之前）：

$ java -jar -Dspring.profiles.active=production demo-0.0.1-SNAPSHOT.jar

在Spring Boot中，您还可以在application.properties中设置活动配置文件，例如

**spring.profiles.active**=production

以这种方式设置的值被系统属性或环境变量设置取代，而不是由SpringApplicationBuilder.profiles（）方法取代。 因此后面的Java API可以用来增加配置文件而不改变默认值。

有关更多信息，请参见[*Chapter 25, Profiles*](#_bookmark115) “ Spring Boot功能”一节中的配置文件。

## 根据环境更改配置

YAML文件实际上是由---行分隔的一系列文档，并且每个文档被分别解析为一个拼合的地图(map)。

如果一个YAML文档包含一个spring.profiles键，那么这个profiles profile值（以逗号分隔的配置文件列表）被送入到Spring Environment.acceptsProfiles（）中，如果这些配置文件中的任何一个是激活的，那么这个文档被包含在最后的合并 否则不）。

Example:

**server**:

**port**: 9000

*---*

**spring**:

**profiles**: development

**server**:

**port**: 9001

*---*

**spring**:

**profiles**: production

**server**:

**port**: 0

在这个例子中，默认端口是9000，但是如果Spring配置文件'development'被激活，那么端口是9001，如果'production'被激活，那么它是0。

YAML文档按照遇到的顺序合并（所以后面的值会覆盖前面的值）。

要对属性文件执行相同的操作，可以使用application - $ {profile} .properties指定特定于配置文件的值。

## 发现外部属性的内置选项(Discover built-in options for external properties)

Spring Boot将application.properties（或.yml）（以及其他位置）的外部属性绑定到运行时的应用程序中。 在单个位置没有（也不可能是）所有受支持属性的详尽列表，因为贡献可能来自类路径上的其他jar文件。

具有执行器(Actuator)功能的正在运行的应用程序具有configprops端点，该端点显示通过@ConfigurationProperties可用的所有绑定和可绑定属性。

附录中包含一个[application.properties](#_bookmark569) 示例，其中包含Spring Boot支持的最常用属性列表。 最终列表来自搜索@ConfigurationProperties和@Value注释的源代码，以及偶尔使用RelaxedPropertyResolver。

# 嵌入式servlet容器

## 将Servlet,Filter或Listener添加到应用程序

有两种方法可以将Servlet，Filter，ServletContextListener和Servlet规范支持的其他侦听器添加到您的应用程序中。 您可以为它们提供Spring bean，也可以启用对Servlet组件的扫描。

### 使用Spring bean添加一个Servlet,Filter或Listener

要添加Servlet，Filter或Servlet \* Listener，请为其提供@Bean定义。 当你想要注入配置或者依赖时，这可能非常有用。 但是，您必须非常小心，不要导致太多其他bean的急切初始化，因为它们必须在应用程序生命周期的早期安装在容器中（例如，让它们依赖于您的DataSource或 JPA配置）。 您可以通过在第一次使用而不是初始化时懒惰地初始化它们来解决这些限制。

在Filters和Servlet的情况下，您还可以通过添加FilterRegistrationBean或ServletRegistrationBean（而不是底层组件）来添加映射和初始化参数。

**Note**

如果在过滤器注册中没有指定dispatcherType，它将匹配FORWARD，INCLUDE和

请求。 如果已启用异步，则它也将匹配ASYNC。

如果要迁移web.xml中没有调度程序元素的过滤器，则需要自行指定dispatcherType：

@Bean

**public** FilterRegistrationBean myFilterRegistration() { FilterRegistrationBean registration = **new** FilterRegistrationBean(); registration.setDispatcherTypes(DispatcherType.REQUEST);

....

**return** registration;

}

#### 禁用Servlet或Filter的注册

如上所述([described above](#_bookmark458))，任何Servlet或Filter bean将自动注册到servlet容器。 要禁用特定Filter或Servlet bean的注册，请为其创建一个注册Bean并将其标记为禁用。 例如：

@Bean

**public** FilterRegistrationBean registration(MyFilter filter) { FilterRegistrationBean registration = **new** FilterRegistrationBean(filter); registration.setEnabled(false);

**return** registration;

}

### 使用classpath scanning添加Servlet,Filters,和Listeners

@WebServlet，@WebFilter和@WebListener带注释的类可以通过注释一个@Configuration类自动注册到一个嵌入式的servlet容器

@ServletComponentScan并指定包含您要注册的组件的包。 默认情况下，@ServletComponentScan将从注释类的包中扫描。

## 更改 HTTP 端口

在独立应用程序中，主HTTP端口默认为8080，但可以使用server.port（例如在application.properties或System属性中）进行设置。 由于环境值的松散绑定，您还可以使用SERVER\_PORT（例如作为OS环境变量）。

要完全关闭HTTP端点，但仍要创建WebApplicationContext，请使用

server.port = -1（这有时用于测试）。

有关更多详细信息，请参阅“[*the section called “Customizing embedded servlet containers”*](#_bookmark152) 一节或[ServerProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java) 源代码。

## 使用随机未分配的HTTP端口

要扫描一个空闲端口（使用OS本机来防止冲突），请使用server.port = 0。

## 在运行时发现HTTP端口

您可以从日志输出中访问运行服务器的端口，也可以通过EmbeddedServletContainer从EmbeddedWebApplicationContext访问端口。 最好的办法是确保它已经初始化了，就是添加一个ApplicationListener类型的@Bean，并且在容器发布的时候把容器拉出来。

使用@SpringBootTest（webEnvironment = WebEnvironment.RANDOM\_PORT）的测试也可以使用@LocalServerPort注释将实际端口注入到字段中。 例如：

@RunWith(SpringJUnit4ClassRunner.class) @SpringBootTest(webEnvironment=WebEnvironment.RANDOM\_PORT) **public class** MyWebIntegrationTests {

@Autowired EmbeddedWebApplicationContext server;

@LocalServerPort

**int** port;

*// ...*

}

**Note**

@LocalServerPort是@Value的meta-annotation（“$ {local.server.port}”）。 不要试图在普通应用程序中注入端口。 正如我们刚刚看到的，只有在容器初始化后才会设置该值。 与测试相反，应用程序代码回调被提前处理（即在值实际可用之前）。

## Configure SSL

SSL可以通过设置不同的server.ssl.\*属性来声明性地配置，通常在

application.properties或application.yml。 例如：

**server.port**=8443

**server.ssl.key-store**=classpath:keystore.jks **server.ssl.key-store-password**=secret **server.ssl.key-password**=another-secret

请参阅[Ssl](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/context/embedded/Ssl.java) 了解所有支持的属性的详细信息。

使用类似上例的配置意味着应用程序将不再支持8080端口上的普通HTTP连接器。Spring Boot不支持通过application.properties配置HTTP连接器和HTTPS连接器。 如果你想有两个，那么你需要以编程方式配置其中之一。 建议使用application.properties来配置HTTPS，因为HTTP连接器更容易以编程方式进行配置。 有关示例，请参阅[spring-boot-sample-tomcat-multi-connectors](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-tomcat-multi-connectors) 示例项目。

## 配置访问 Logging

访问日志可以通过各自的命名空间为Tomcat和Undertow配置。 例如，以下日志以自定义模式([custom pattern](https://tomcat.apache.org/tomcat-8.0-doc/config/valve.html#Access_Logging))访问Tomcat。

**server.tomcat.basedir**=my-tomcat **server.tomcat.accesslog.enabled**=true **server.tomcat.accesslog.pattern**=%t %a "%r" %s (%D ms)

**Note**

日志的默认位置是相对于tomcat基目录的日志目录，默认情况下该目录是临时目录，所以您可能需要修复Tomcat的基础目录或使用日志的绝对路径。 在上面的示例中，日志将在my-tomcat / logs中相对于应用程序的工作目录可用。

访问日志可以用类似的方式进行配置

**server.undertow.accesslog.enabled**=true **server.undertow.accesslog.pattern**=%t %a "%r" %s (%D ms)

日志存储在相对于应用程序的工作目录的日志目录中。 这可以通过server.undertow.accesslog.directory进行自定义。

## Use behind a front-end proxy server

您的应用程序可能需要发送302重定向或使用绝对链接呈现内容。 在代理之后运行时，调用者需要链接到代理，而不是托管应用的机器的物理地址。 通常情况下，这种情况是通过与代理签订合同来处理的，代理将添加标题来告诉后端如何构建到自己的链接。

如果代理添加了常规的 X-Forwarded-For 和 X-Forwarded-Proto headers(大多数情况下是这样的)。只要server.use-forward-headers在你的应用中设置为true，那么就可以正确地呈现绝对链接。

**Note**

如果您的应用程序在Cloud Foundry或Heroku中运行，server.use-forward-headers

属性默认为true，如果没有指定。 在所有其他情况下，它默认为false。

### 自定义Tomcat的代理配置

如果您正在使用Tomcat，则可以另外配置用于携带“forwarded”信息的标头的名称：

server.tomcat.remote-ip-header=x-your-remote-ip-header server.tomcat.protocol-header=x-your-protocol-header

server.tomcat.internal-proxies=192\\.168\\.\\d{1,3}\\.\\d{1,3}

Tomcat也配置了一个默认的正则表达式来匹配要被信任的内部代理。 默认情况下，10/8，192.168 / 16,169.254 / 16和127/8中的IP地址是可信的。 您可以通过向application.properties添加条目来自定义阀(valve’s)的配置。

**Note**

只有在使用属性文件进行配置时才需要双反斜杠。 如果您使用的是YAML，则单个反斜杠就足够了，并且与上面显示的相同的值将是192 \ .168 \。\ d {1,3} \。\ d {1,3}。

**Note**

您可以通过将内部代理(internal-proxies)设置为空来信任所有代理（但不要在生产中这样做）。

您可以通过切换自动关闭（即set server.use-forward-headers = false）并在TomcatEmbeddedServletContainerFactory bean中添加一个新的(valve)实例来完全控制Tomcat RemoteIpValve的配置。

## 配置 Tomcat

一般来说，您可以按照[*Section 72.8, “Discover built-in options for external properties”*](#_bookmark455)中有关@ConfigurationProperties（ServerProperties是此处的主要内容）的建议，还可以查看EmbeddedServletContainerCustomizer和各种可以添加到一个Tomcat的\* Customizer 那些。 Tomcat API非常丰富，所以一旦你有权访问TomcatEmbeddedServletContainerFactory，你可以通过多种方式修改它。 或者核选项是添加你自己的TomcatEmbeddedServletContainerFactory。

## Enable Multiple Connectors with Tomcat

将一个org.apache.catalina.connector.Connector添加到TomcatEmbeddedServletContainerFactory中，该TomcatEmbeddedServletContainerFactory可以允许多个连接器， HTTP和HTTPS连接器：

@Bean

**public** EmbeddedServletContainerFactory servletContainer() { TomcatEmbeddedServletContainerFactory tomcat = **new** TomcatEmbeddedServletContainerFactory(); tomcat.addAdditionalTomcatConnectors(createSslConnector());

**return** tomcat;

}

**private** Connector createSslConnector() {

Connector connector = **new** Connector(***"org.apache.coyote.http11.Http11NioProtocol"***); Http11NioProtocol protocol = (Http11NioProtocol) connector.getProtocolHandler();

**try** {

File keystore = **new** ClassPathResource(***"keystore"***).getFile(); File truststore = **new** ClassPathResource(***"keystore"***).getFile();

connector.setSecure(true); connector.setPort(8443); protocol.setSSLEnabled(true);

protocol.setKeystoreFile(keystore.getAbsolutePath()); protocol.setKeystorePass(***"changeit"***); protocol.setTruststoreFile(truststore.getAbsolutePath()); protocol.setTruststorePass(***"changeit"***); protocol.setKeyAlias(***"apitester"***);

**return** connector;

}

**catch** (IOException ex) {

**throw new** IllegalStateException(***"can't access keystore: ["*** + ***"keystore"***

+ ***"] or truststore: ["*** + ***"keystore"*** + ***"]"***, ex);

}

}

## 使用 Tomcat’s LegacyCookieProcessor

Spring Boot使用的嵌入式Tomcat不支持开箱即用的Cookie格式的“Version0”，您可能会看到以下错误：

java.lang.IllegalArgumentException: An invalid character [32] was present in the Cookie value

如果可能的话，你应该考虑更新你的代码，只存储符合以后Cookie规范的值。 但是，如果您无法更改写入cookie的方式，则可以将Tomcat配置为使用LegacyCookieProcessor。 要切换到LegacyCookieProcessor，请使用添加了TomcatContextCustomizer的EmbeddedServletContainerCustomizer bean：

@Bean

**public** EmbeddedServletContainerCustomizer cookieProcessorCustomizer() {

**return new** EmbeddedServletContainerCustomizer() {

@Override

**public void** customize(ConfigurableEmbeddedServletContainer container) {

**if** (container **instanceof** TomcatEmbeddedServletContainerFactory) { ((TomcatEmbeddedServletContainerFactory) container)

.addContextCustomizers(**new** TomcatContextCustomizer() {

@Override

**public void** customize(Context context) { context.setCookieProcessor(**new** LegacyCookieProcessor());

}

});

}

}

};

}

## 使用 Jetty 代替 Tomcat

Spring Boot启动器(starters)（特别是spring-boot-starter-web）默认使用Tomcat作为嵌入式容器。 您需要排除这些依赖关系，并包括Jetty一个。 Spring Boot提供了Tomcat和Jetty依赖关系作为独立的初始者捆绑在一起，以使这个过程尽可能简单。

Maven 文件示例:

**<dependency>**

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

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

**<exclusions>**

**<exclusion>**

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

**<artifactId>**spring-boot-starter-tomcat**</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**<dependency>**

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

**<artifactId>**spring-boot-starter-jetty**</artifactId>**

**</dependency>**

Gradle 文件示例:

configurations {

compile.exclude module: ***"spring-boot-starter-tomcat"***

}

dependencies {

compile(***"org.springframework.boot:spring-boot-starter-web:1.5.8.RELEASE"***) compile(***"org.springframework.boot:spring-boot-starter-jetty:1.5.8.RELEASE"***)

*// ...*

}

## 配置 Jetty

通常，您可以按照[*Section 72.8, “Discover built-in options for external properties”*](#_bookmark455)中有关@ConfigurationProperties（ServerProperties是此处的主要内容）的建议，还可以查看EmbeddedServletContainerCustomizer。 Jetty API非常丰富，所以一旦你有权访问JettyEmbeddedServletContainerFactory，你可以通过多种方式对其进行修改。 或者核选项是添加你自己的JettyEmbeddedServletContainerFactory。

## 使用 Undertow 代替 Tomcat

使用Undertow而不是Tomcat与使用[using Jetty instead of Tomcat](#_bookmark471)非常相似。 您需要排除Tomcat依赖关系，并包含Undertow启动器。

Maven 文件示例:

**<dependency>**

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

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

**<exclusions>**

**<exclusion>**

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

**<artifactId>**spring-boot-starter-tomcat**</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**<dependency>**

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

**<artifactId>**spring-boot-starter-undertow**</artifactId>**

**</dependency>**

Gradle 文件示例:

configurations {

compile.exclude module: ***"spring-boot-starter-tomcat"***

}

dependencies {

compile(***"org.springframework.boot:spring-boot-starter-web:1.5.8.RELEASE"***) compile(***"org.springframework.boot:spring-boot-starter-undertow:1.5.8.RELEASE"***)

*// ...*

}

## 配置 Undertow

通常，您可以按照[*Section 72.8, “Discover built-in options for external properties”*](#_bookmark455)中有关@ConfigurationProperties（ServerProperties和ServerProperties.Undertow是此处的主要内容）的建议，还可以查看EmbeddedServletContainerCustomizer。 一旦访问UndertowEmbeddedServletContainerFactory，您可以使用UndertowBuilderCustomizer修改Undertow的配置以满足您的需求。 或者核选项是添加你自己的UndertowEmbeddedServletContainerFactory。

## Enable Multiple Listeners with Undertow

Add an UndertowBuilderCustomizer to the UndertowEmbeddedServletContainerFactory

and add a listener to the Builder:

@Bean

**public** UndertowEmbeddedServletContainerFactory embeddedServletContainerFactory() { UndertowEmbeddedServletContainerFactory factory = **new** UndertowEmbeddedServletContainerFactory(); factory.addBuilderCustomizers(**new** UndertowBuilderCustomizer() {

@Override

**public void** customize(Builder builder) { builder.addHttpListener(8080, ***"0.0.0.0"***);

}

});

**return** factory;

}

## 使用 Tomcat 7.x or 8.0

Tomcat 7＆8.0使用Spring Boot，但是默认是使用Tomcat 8.5。 如果你不能使用Tomcat

8.5（例如，因为您正在使用Java 1.6），您将需要更改您的类路径以引用不同的版本。

### Use Tomcat 7.x or 8.0 with Maven

如果您正在使用starters和parent，则可以更改Tomcat版本属性并另外导入tomcat-juli。 例如。 对于简单的Web应用程序或服务：

**<properties>**

**<tomcat.version>**7.0.59**</tomcat.version>**

**</properties>**

**<dependencies>**

...

**<dependency>**

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

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

**</dependency>**

**<dependency>**

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

**<artifactId>**tomcat-juli**</artifactId>**

**<version>**${tomcat.version}**</version>**

**</dependency>**

...

**</dependencies>**

### 在Gradle中使用 Tomcat 7.x 或 8.0

使用Gradle，您可以通过设置tomcat.version属性来更改Tomcat版本，然后另外包含tomcat-juli：

ext[***'tomcat.version'***] = ***'7.0.59'***

dependencies {

compile ***'org.springframework.boot:spring-boot-starter-web'***

compile group:***'org.apache.tomcat'***, name:***'tomcat-juli'***, version:property(***'tomcat.version'***)

}

## 使用 Jetty 9.2

Jetty 9.2和Spring Boot一起工作，但默认是使用Jetty 9.3。 如果你不能使用Jetty 9.3（例如，因为你正在使用Java 7），你将需要将你的类路径改为引用Jetty 9.2。

### 在Maven中使用 Jetty 9.2

如果您正在使用starters 和 parent，您可以添加Jetty启动器并覆盖

jetty.version属性：

**<properties>**

**<jetty.version>**9.2.17.v20160517**</jetty.version>**

**</properties>**

**<dependencies>**

**<dependency>**

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

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

**<exclusions>**

**<exclusion>**

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

**<artifactId>**spring-boot-starter-tomcat**</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**<dependency>**

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

**<artifactId>**spring-boot-starter-jetty**</artifactId>**

**</dependency>**

**</dependencies>**

### Use Jetty 9.2 with Gradle

您可以设置jetty.version属性。 例如，对于简单的web应用程序或服务：

ext[***'jetty.version'***] = ***'9.2.17.v20160517'***

dependencies {

compile (***'org.springframework.boot:spring-boot-starter-web'***) {

exclude group: ***'org.springframework.boot'***, module: ***'spring-boot-starter-tomcat'***

}

compile (***'org.springframework.boot:spring-boot-starter-jetty'***)

}

## 使用 Jetty 8

Jetty 8与Spring Boot一起工作，但默认是使用Jetty 9.3。 如果您不能使用Jetty 9.3（例如，因为您使用的是Java 1.6），则需要将您的类路径更改为引用Jetty 8.您还需要排除Jetty的与WebSocket相关的依赖项。

### 在Maven中使用Jetty 8

如果您使用的是starters 和 parent，只需添加Jetty starter入门者所需的WebSocket排除项，并更改版本属性（例如， 对于简单的Web应用程序或服务：

**<properties>**

**<jetty.version>**8.1.15.v20140411**</jetty.version>**

**<jetty-jsp.version>**2.2.0.v201112011158**</jetty-jsp.version>**

**</properties>**

**<dependencies>**

**<dependency>**

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

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

**<exclusions>**

**<exclusion>**

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

**<artifactId>**spring-boot-starter-tomcat**</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**<dependency>**

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

**<artifactId>**spring-boot-starter-jetty**</artifactId>**

**<exclusions>**

**<exclusion>**

**<groupId>**org.eclipse.jetty.websocket**</groupId>**

**<artifactId>**\***</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**</dependencies>**

### 在Gradle中使用Jetty 8

您可以设置jetty.version属性并排除WebSocket依赖项，例如 对于简单的Web应用程序或服务：

ext[***'jetty.version'***] = ***'8.1.15.v20140411'***

dependencies {

compile (***'org.springframework.boot:spring-boot-starter-web'***) {

exclude group: ***'org.springframework.boot'***, module: ***'spring-boot-starter-tomcat'***

}

compile (***'org.springframework.boot:spring-boot-starter-jetty'***) { exclude group: ***'org.eclipse.jetty.websocket'***

}

}

## 使用 @ServerEndpoint

如果要在使用嵌入容器的Spring Boot应用程序中使用@ServerEndpoint，则必须声明一个ServerEndpointExporter @Bean：

@Bean

**public** ServerEndpointExporter serverEndpointExporter() {

**return new** ServerEndpointExporter();

}

这个bean将注册任何带有底层WebSocket容器的@ServerEndpoint注释的bean。 部署到独立的servlet容器时，此角色由servlet容器初始值设定项执行，并且不需要ServerEndpointExporter bean。

## 启用 HTTP response compression

Jetty，Tomcat和Undertow支持HTTP响应压缩。 它可以通过启用application.properties：

**server.compression.enabled**=true

默认情况下，响应的长度必须至少为2048字节才能执行压缩。 这可以使用server.compression.min-response-size属性进行配置。

默认情况下，响应只有在内容类型为以下内容时才会被压缩：

* text/html
* text/xml
* text/plain
* text/css

这可以使用server.compression.mime-types属性进行配置。

# Spring MVC

## 编写一个 JSON REST 服务

只要Jackson2在类路径上，Spring Boot应用程序中的任何Spring @RestController默认都会呈现JSON响应。 例如：

@RestController

**public class** MyController {

@RequestMapping("/thing")

**public** MyThing thing() {

**return new** MyThing();

}

}

只要MyThing可以被Jackson2序列化（例如普通的POJO或者Groovy对象），那么默认情况下localhost:8080/thing将为其提供JSON表示。 有时在浏览器中，您可能会看到XML响应，因为浏览器倾向于发送喜欢XML的接受标头。

## 编写一个 XML REST 服务

如果你在类路径上有Jackson XML扩展（jackson-dataformat-xml），它将被用来呈现XML响应，和我们用于JSON的例子一样。 要使用它，请将以下依赖项添加到您的项目中：

**<dependency>**

**<groupId>**com.fasterxml.jackson.dataformat**</groupId>**

**<artifactId>**jackson-dataformat-xml**</artifactId>**

**</dependency>**

您可能还想添加对Woodstox的依赖。 它比JDK提供的默认StAX实现更快，并且还增加了漂亮的打印支持和改进的命名空间处理：

**<dependency>**

**<groupId>**org.codehaus.woodstox**</groupId>**

**<artifactId>**woodstox-core-asl**</artifactId>**

**</dependency>**

如果Jackson的XML扩展不可用，那么将使用JAXB（在JDK中默认提供），并附加要求将MyThing注释为@XmlRootElement：

@XmlRootElement

**public class** MyThing {

**private** String name;

*// .. getters and setters*

}

要让服务器呈现XML而不是JSON，您可能需要发送Accept：text/xml 头（或使用浏览器）。

## 自定义 Jackson ObjectMapper

Spring MVC（客户端和服务器端）使用HttpMessageConverters来协商HTTP交换中的内容转换。 如果Jackson在类路径中，则已经获得由Jackson2ObjectMapperBuilder提供的默认转换器，其中的一个实例将为您自动配置。

默认情况下创建的ObjectMapper（或用于Jackson XML转换器的XmlMapper）实例具有以下自定义属性：

* MapperFeature.DEFAULT\_VIEW\_INCLUSION已禁用
* DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES已禁用

Spring Boot 还具有一些功能，可以更轻松地自定义此行为。

You can configure the ObjectMapper and XmlMapper instances using the environment. Jackson provides an extensive suite of simple on/off features that can be used to configure various aspects of its processing. These features are described in six enums in Jackson which map onto properties in the environment:

|  |  |
| --- | --- |
| **Jackson enum** | **Environment property** |
| com.fasterxml.jackson.  databind.DeserializationFeature | spring.jackson.deserialization.  <feature\_name>=true|false |
| com.fasterxml.jackson.  core.JsonGenerator.Feature | spring.jackson.generator.  <feature\_name>=true|false |
| com.fasterxml.jackson.  databind.MapperFeature | spring.jackson.mapper.  <feature\_name>=true|false |
| com.fasterxml.jackson.core.  JsonParser.Feature | spring.jackson.parser.  <feature\_name>=true|false |
| com.fasterxml.jackson.  databind.SerializationFeature | spring.jackson.serialization.  <feature\_name>=true|false |
| com.fasterxml.jackson  .annotation.JsonInclude.Include | spring.jackson.default-property-inclusion=always|non\_null|  non\_absent|non\_default|non\_empty |

例如，要启用漂亮的打印，请设置spring.jackson.serialization.indent\_output = true。 请注意，由于使用[relaxed](#_bookmark111) [binding](#_bookmark111)，indent\_output的情况不必与相应的枚举常量INDENT\_OUTPUT的情况相匹配。

此基于环境的配置应用于自动配置的Jackson2ObjectMapperBuilder bean，并将应用于使用构建器创建的任何映射器，包括自动配置(auto-configured)的ObjectMapper bean。

任何com.fasterxml.jackson.databind.Module类型的bean都将自动注册到自动配置的Jackson2ObjectMapperBuilder中，并应用于它创建的任何ObjectMapper实例。 这为在向应用程序添加新功能时提供了一种全局机制来提供自定义模块。

如果要完全替换默认的ObjectMapper，则可以定义该类型的@Bean并将其标记为@Primary，或者，如果您更喜欢基于构建器的方法，请定义一个Jackson2ObjectMapperBuilder @Bean。 请注意，在任何情况下，这将禁用ObjectMapper的所有自动配置。

如果您提供了MappingJackson2HttpMessageConverter类型的@Beans，那么它们将替换MVC配置中的默认值。 此外，提供了一个便捷的Bean类型HttpMessageConverters（总是可用，如果您使用默认的MVC配置），有一些有用的方法来访问默认和用户增强的消息转换器(user-enhanced message converters)。

另外请参考 [*Section 74.4, “Customize the @ResponseBody rendering”*](#_bookmark491)部分和

[WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration.java) 源码查看更多详细信息。

## 自定义 @ResponseBody rendering

Spring使用HttpMessageConverters呈现@ResponseBody（或来自@RestController的响应）。 您可以通过在Spring Boot context中简单添加该类型的bean来提供额外的转换器。 如果您添加的bean是默认包含的类型（如用于JSON转换的MappingJackson2HttpMessageConverter），那么它将替换默认值。 提供了一个类型为HttpMessageConverters的便利bean（如果使用默认的MVC配置，总是可用的），它有一些有用的方法来访问默认的和用户增强的消息转换器（例如，如果您想手动将它们注入自定义的RestTemplate）。

和正常的MVC用法一样，您提供的任何WebMvcConfigurerAdapter bean都可以通过重写configureMessageConverters方法来提供转换器，但与普通MVC不同的是，您只能提供所需的其他转换器（因为Spring Boot使用相同的机制来提供其默认值）。 最后，如果您通过提供自己的@EnableWebMvc配置来选择退出Spring Boot默认MVC配置，那么您可以完全控制并使用来自WebMvcConfigurationSupport的getMessageConverters手动执行所有操作。

查看 [WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration.java) 源码获取更多信息。

## Handling Multipart File Uploads

Spring Boot包含Servlet 3 javax.servlet.http.Part API来支持上传文件。 默认情况下，Spring Boot在每个文件中配置最大文件为1MB，单个请求中最大为10MB的文件数据。 您可以通过使用MultipartProperties类中公开的属性来覆盖这些值以及存储中间数据的位置（例如，到 /tmp目录）和将数据刷新到磁盘的阈值。 例如，如果要指定文件不受限制，请将spring.http.multipart.max-file-size属性设置为-1。

当您想在Spring MVC控制器处理程序方法中将Multipart编码的文件数据作为MultipartFile类型的@ RequestParam注释参数接收时，multipart support很有帮助。

查看 [MultipartAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/MultipartAutoConfiguration.java) 源码获取更多信息。

## 关闭 Spring MVC DispatcherServlet

Spring Boot希望从应用程序的根目录下载所有内容。 如果您宁愿将您自己的servlet映射到该URL，则可以这样做，但是当然可能会失去其他一些Boot MVC功能。 要添加自己的servlet并将其映射到根资源，只需声明一个类型为Servlet的@Bean，并为其指定一个特殊的bean名称dispatcherServlet（如果要关闭它，也可以使用该名称创建一个不同类型的bean） 不能取代它）。

## 关闭默认 MVC 配置

完全控制MVC配置的最简单的方法是使用@EnableWebMvc注解提供自己的@Configuration。这将使所有的MVC配置在你的hands。

## 自定义 ViewResolvers

ViewResolver是Spring MVC的核心组件，将@Controller中的视图名称转换为实际的View实现。 请注意，ViewResolvers主要用于UI应用程序，而不是REST风格的服务（View不用于呈现@ResponseBody）。 有很多ViewResolver的实现可供选择，而Spring本身并不认为你应该使用哪一个。 另一方面，Spring Boot根据在类路径和应用程序上下文(application context)中找到的内容，为您安装一个或两个。 DispatcherServlet使用它在应用程序上下文中找到的所有解析器，依次尝试每个解析器，直到获得结果，所以如果要添加自己的解析器，则必须知道该解析器添加的顺序和位置。

WebMvcAutoConfiguration将以下ViewResolver添加到您的上下文中：

* 具有bean id“defaultViewResolver”的InternalResourceViewResolver。 这个查找可以使用DefaultServlet呈现的物理资源（例如，如果使用的是静态资源和JSP页面）。 它将前缀和后缀应用于视图名称，然后在servlet上下文中查找具有该路径的物理资源（默认值均为空，但可通过spring.mvc.view.prefix和spring.mvc.view.suffix访问外部配置）。 它可以通过提供相同类型的bean来重写。[blog about ContentNegotiatingViewResolver](https://spring.io/blog/2013/06/03/content-negotiation-using-views)
* 一个带有“beanNameViewResolver”的BeanNameViewResolver。 这是视图解析器链中的一个有用的成员，并且将拾取与正在解析的视图名称相同的所有bean。 不应该有必要重写或替换它。
* 如果实际上有View类型的bean，则仅添加id为“viewResolver”的ContentNegotiatingViewResolver。 这是一个“主”解析器，委托给所有其他人，试图找到与客户端发送的“接受”HTTP头匹配。 有一个关于[blog about ContentNegotiatingViewResolver](https://spring.io/blog/2013/06/03/content-negotiation-using-views) ，你可能想了解更多，也可以看看源代码的细节。 您可以通过定义一个名为“viewResolver”的bean来关闭自动配置的ContentNegotiatingViewResolver。
* 如果您使用Thymeleaf，您也将拥有一个带有“thymeleafViewResolver”的ThymeleafViewResolver。 它通过用前缀和后缀来包围视图名称来查找资源（外部化为spring.thymeleaf.prefix和spring.thymeleaf.suffix，分别为默认的'classpath：/ templates /'和'.html'）。 它可以通过提供一个相同名字的bean来覆盖。
* 如果您使用FreeMarker，您也将拥有一个ID为“freeMarkerViewResolver”的FreeMarkerViewResolver。 它通过使用前缀和后缀包围视图名称来查找加载器路径中的资源（外化为spring.freemarker.templateLoaderPath，默认为“classpath：/ templates /”）（外化为spring.freemarker.prefix和spring.freemarker.suffix ，分别为空和'.ftl'的默认值）。 它可以通过提供一个相同名字的bean来覆盖。
* 如果您使用Groovy模板（实际上，如果groovy-templates位于您的类路径中），那么您也将拥有一个带有“groovyMarkupViewResolver”标识的GroovyMarkupViewResolver。 它通过用前缀和后缀包围视图名称来查找加载器路径中的资源（外化为spring.groovy.template.prefix和spring.groovy.template.suffix，默认为'classpath：/ templates /'和'.tpl' 分别）。 它可以通过提供一个相同名字的bean来覆盖。

查看[WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration.java),[ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java), [FreeMarkerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java) 和[GroovyTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.java)

## 使用 Thymeleaf 3

默认情况下，spring-boot-starter-thymeleaf使用Thymeleaf 2.1。 如果您使用的是spring-boot-starter-parent，则可以通过覆盖thymeleaf.version和thymeleaf-layout-dialect.version属性来使用Thymeleaf 3，例如：

**<properties>**

**<thymeleaf.version>**3.0.2.RELEASE**</thymeleaf.version>**

**<thymeleaf-layout-dialect.version>**2.1.1**</thymeleaf-layout-dialect.version>**

**</properties>**

**Note**

如果您自己正在管理依赖关系，请查看与这两个版本相关的工件列表的spring-boot-dependencies。

要避免出现关于不推荐使用HTML 5模板模式和正在使用HTML模板模式的警告消息，您可能还需要将spring.thymeleaf.mode明确配置为HTML，例如：

**spring.thymeleaf.mode**: HTML

请参阅[Thymeleaf 3 sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-web-thymeleaf3) 样本以查看此操作。

如果您正在使用任何其他自动配置的Thymeleaf Extras（Spring Security，Data Attribute或Java 8 Time），则还应该将其每个版本覆盖为与Thymeleaf 3.0兼容的版本。

# HTTP clients

## 使用代理配置 RestTemplate

如第[Section 33.1, “RestTemplate customization”](#_bookmark244)中所述，RestTemplateCustomizer可以与RestTemplateBuilder配合使用来构建定制的RestTemplate。 这是创建配置为使用代理的RestTemplate的推荐方法。

代理配置的确切细节取决于正在使用的底层客户端请求工厂。 下面是一个使用HttpClient配置HttpComponentsClientRequesfor all hosts except 192.168.0.5.

**static class** ProxyCustomizer **implements** RestTemplateCustomizer {

@Override

**public void** customize(RestTemplate restTemplate) { HttpHost proxy = **new** HttpHost(***"proxy.example.com"***); HttpClient httpClient = HttpClientBuilder.create()

.setRoutePlanner(**new** DefaultProxyRoutePlanner(proxy) {

@Override

**public** HttpHost determineProxy(HttpHost target, HttpRequest request, HttpContext context)

**throws** HttpException {

**if** (target.getHostName().equals(***"192.168.0.5"***)) {

**return** null;

}

**return super**.determineProxy(target, request, context);

}

}).build(); restTemplate.setRequestFactory(

**new** HttpComponentsClientHttpRequestFactory(httpClient));

}

}

# Logging

除了Commons Logging API，Spring Boot没有强制的日志依赖性，其中有许多实现可供选择。 要使用[Logback](http://logback.qos.ch/) ，你需要在类路径中包含它和jcl-over-slf4j（它实现Commons Logging API）。 最简单的方法是通过所有依赖于spring-boot-starter-logging的初学者。 对于Web应用程序，您只需要spring-boot-starter-web，因为它依赖于日志启动器。 例如，使用Maven：

**<dependency>**

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

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

**</dependency>**

Spring Boot有一个LoggingSystem抽象，它试图根据类路径的内容来配置日志记录。 如果Logback可用，则是首选。

如果你需要做的唯一的改变就是设置各种记录器的级别，那么你可以在application.properties中使用“logging.level”前缀来完成这个操作。

**logging.level.org.springframework.web**=DEBUG **logging.level.org.hibernate**=ERROR

您还可以使用“logging.file”来设置要登录的文件的位置（除了控制台之外）。

要配置日志记录系统的更精细的设置，您需要使用LoggingSystem支持的本机配置格式。 默认情况下，Spring Boot会从系统的默认位置（例如，类路径：logback.xml）中获取本机配置，但是您可以使用“logging.config”属性来设置配置文件的位置。

## 配置 Logback 进行日志记录

如果你把一个logback.xml放到你的类路径的根目录下，它将会从那里（或者是使用logback-spring.xml来利用Boot提供的模板特性）。 Spring Boot提供了一个默认的基本配置，如果您只想设置级别，则可以包含这个配置。

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

**<configuration>**

**<include resource**=**"org/springframework/boot/logging/logback/base.xml"/>**

**<logger name**=**"org.springframework.web" level**=**"DEBUG"/>**

**</configuration>**

如果您查看spring-boot jar中的base.xml，您将看到它使用LoggingSystem为您创建的一些有用的System属性。 这些是：

* ${PID} the current process ID.
* ${LOG\_FILE} if logging.file was set in Boot’s external configuration.
* ${LOG\_PATH} if logging.path was set (representing a directory for log files to live in).
* ${LOG\_EXCEPTION\_CONVERSION\_WORD} if logging.exception-conversion-word was set in Boot’s external configuration.

Spring Boot还使用自定义的Logback转换器在控制台（但不是日志文件）上提供了一些漂亮的ANSI彩色终端输出。 有关详细信息，请参阅默认的base.xml配置。

如果Groovy位于类路径中，则应该可以使用logback.groovy配置Logback（如果存在，将会给予优先权）。

### 只配置文件输出的logback(Configure logback for file only output)

如果要禁用控制台日志并仅将输出写入文件，则需要导入file-appender.xml而不是console-appender.xml的自定义logback- spring.xml：

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

**<configuration>**

**<include resource**=**"org/springframework/boot/logging/logback/defaults.xml" />**

**<property name**=**"LOG\_FILE" value**=**"${LOG\_FILE:-${LOG\_PATH:-${LOG\_TEMP:-${java.io.tmpdir:-/ tmp}}/}spring.log}"/>**

**<include resource**=**"org/springframework/boot/logging/logback/file-appender.xml" />**

**<root level**=**"INFO">**

**<appender-ref ref**=**"FILE" />**

**</root>**

**</configuration>**

您还需要将logging.file添加到您的application.properties中：

**logging.file**=myapplication.log

## 配置 Log4j 进行日志输出

如果在类路径上，Spring Boot支持[Log4j 2](http://logging.apache.org/log4j/2.x) 的日志配置。 如果您使用starters来组装依赖关系，那意味着您必须排除Logback，然后包含log4j 2。 如果你不使用starters，那么除了Log4j 2之外，你还需要提供jcl-over-slf4j（至少）。

最简单的路径可能是通过初学者，尽管需要一些排除，例如， 在Maven：

**<dependency>**

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

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

**</dependency>**

**<dependency>**

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

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

**<exclusions>**

**<exclusion>**

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

**<artifactId>**spring-boot-starter-logging**</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**<dependency>**

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

**<artifactId>**spring-boot-starter-log4j2**</artifactId>**

**</dependency>**

**Note**

Log4j入门者的使用收集了常见日志记录需求的依赖关系（例如，包括让Tomcat使用java.util.logging，但使用Log4j 2配置输出）。 有关更多详细信息，请参阅执行器Log4j 2样本，并查看它的实际使用情况。

### 使用 YAML 或 JSON 来配置 Log4j 2

除了默认的XML配置格式之外，Log4j 2还支持YAML和JSON配置文件。 要将Log4j 2配置为使用备用配置文件格式，请将相应的依赖关系添加到类路径中，并将您的配置文件命名为与您选择的文件格式相匹配：

| **Format** | **Dependencies** | **File names** |
| --- | --- | --- |
| YAML | com.fasterxml.jackson.core:jackson-databindcom.fasterxml.jackson.dataformat:jackson-dataformat-yaml | log4j2.  yamllog4j2.yml |
| JSON | com.fasterxml.jackson.core:jackson-databind | log4j2.  jsonlog4j2.jsn |

# Data Access

## 配置自定义 DataSource

要配置您自己的DataSource，请在您的配置中定义该类型的@Bean。 Spring Boot会在需要的地方重复使用你的DataSource，包括数据库初始化。 如果您需要外部化某些设置，则可以轻松地将DataSource绑定到环境（请参阅“[the section](#_bookmark110) [called “Third-party configuration”](#_bookmark110)一节）。

@Bean @ConfigurationProperties(prefix="app.datasource") **public** DataSource dataSource() {

**return new** FancyDataSource();

}

**app.datasource.url**=jdbc:h2:mem:mydb **app.datasource.username**=sa **app.datasource.pool-size**=30

假设您的FancyDataSource具有常规的JavaBean属性（包括url，用户名和池大小），那么在DataSource可用于其他组件之前，将自动绑定这些设置。 常规的数据库初始化([database initialization](#_bookmark520) )也会发生（所以spring.datasource.\*的相关子集仍然可以与您的自定义配置一起使用）。

如果您正在配置自定义的JNDI数据源，则可以应用相同的原则：

@Bean(destroyMethod="") @ConfigurationProperties(prefix="app.datasource") **public** DataSource dataSource() **throws** Exception {

JndiDataSourceLookup dataSourceLookup = **new** JndiDataSourceLookup();

**return** dataSourceLookup.getDataSource(***"java:comp/env/jdbc/YourDS"***);

}

Spring Boot还提供了一个实用程序构建器类DataSourceBuilder，可用于创建其中一个标准数据源（如果它在类路径上）。 构建器可以根据类路径上的可用内容来检测要使用的类。 它也会根据JDBC URL自动检测驱动程序。

@Bean @ConfigurationProperties("app.datasource") **public** DataSource dataSource() {

**return** DataSourceBuilder.create().build();

}

要使用该DataSource运行应用程序，所需要的只是连接信息; 还可以提供特定于池的设置，请检查将在运行时使用的实现以获取更多详细信息。

**app.datasource.url**=jdbc:mysql://localhost/test **app.datasource.username**=dbuser **app.datasource.password**=dbpass **app.datasource.pool-size**=30

但有一个问题。 由于连接池的实际类型未公开，所以在您的自定义DataSource的元数据中不会生成任何键，并且您的IDE中没有完成（DataSource接口不公开任何属性）。 另外，如果你碰巧在类路径上只有Hikari，这个基本设置将不起作用，因为Hikari没有url参数（但是是一个jdbcUrl参数）。 你将不得不重写你的配置如下：

**app.datasource.jdbc-url**=jdbc:mysql://localhost/test **app.datasource.username**=dbuser **app.datasource.password**=dbpass **app.datasource.maximum-pool-size**=30

你可以通过强制连接池来解决这个问题，并返回一个专门的实现，而不是DataSource。 您将无法在运行时更改实现，但选项列表将是显式的。

@Bean @ConfigurationProperties("app.datasource") **public** HikariDataSource dataSource() {

**return** (HikariDataSource) DataSourceBuilder.create()

.type(HikariDataSource.**class**).build();

}

你甚至可以进一步利用DataSourceProperties为你做的事情，即提供一个默认的嵌入式数据库，如果没有提供一个合理的用户名和密码的URL。 你可以很容易地从任何DataSourceProperties的状态初始化一个DataSourceBuilder，所以你可以注入Spring Boot自动创建的一个。 但是，这会将您的配置拆分为两个名称空间：url，用户名，密码，spring.datasource的类型和驱动程序，以及其他自定义名称空间（app.datasource）。 为了避免这种情况，您可以在自定义名称空间上重新定义自定义的DataSourceProperties：

@Bean @Primary

@ConfigurationProperties("app.datasource")

**public** DataSourceProperties dataSourceProperties() {

**return new** DataSourceProperties();

}

@Bean @ConfigurationProperties("app.datasource")

**public** HikariDataSource dataSource(DataSourceProperties properties) {

**return** (HikariDataSource) properties.initializeDataSourceBuilder()

.type(HikariDataSource.**class**).build();

}

除了选择了一个专用的连接池（在代码中）并且它的设置暴露在相同的命名空间中之外，这个设置使您与默认情况下的Spring Boot成为一对。 由于DataSourceProperties正在为您处理url / jdbcUrl翻译，因此可以像这样配置它：

**app.datasource.url**=jdbc:mysql://localhost/test **app.datasource.username**=dbuser **app.datasource.password**=dbpass **app.datasource.maximum-pool-size**=30

**Note**

由于您的自定义配置选择Hikari，app.datasource.type将不起作用。 在实践中，构建器将被初始化为可能在那里设置的任何值，然后被对.type（）的调用覆盖。

请参阅[*Section 29.1, “Configure a DataSource”*](#_bookmark166)一节中的第29.1节“配置数据源”和

[DataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.java) 类获取更多细节。

## 配置两个 DataSources

如果您需要配置多个数据源，则可以应用上一节中所述的相同技巧。 但是，您必须将其中一个DataSource @Primary标记为各种自动配置，希望能够逐个获取。

如果您创建自己的数据源，则自动配置将退出。 在下面的示例中，我们提供了与自动配置在主数据源上提供的完全相同的功能集：

@Bean @Primary

@ConfigurationProperties("app.datasource.foo")

**public** DataSourceProperties fooDataSourceProperties() {

**return new** DataSourceProperties();

}

@Bean @Primary

@ConfigurationProperties("app.datasource.foo")

**public** DataSource fooDataSource() {

**return** fooDataSourceProperties().initializeDataSourceBuilder().build();

}

@Bean @ConfigurationProperties("app.datasource.bar") **public** BasicDataSource barDataSource() {

**return** (BasicDataSource) DataSourceBuilder.create()

.type(BasicDataSource.**class**).build();

}

**Tip**

必须将fooDataSourceProperties标记为@Primary，以便数据库初始化程序功能使用您的副本（如果使用的话）。

这两个数据源也都是用于高级自定义的。 例如，你可以配置它们如下：

**app.datasource.foo.type**=com.zaxxer.hikari.HikariDataSource **app.datasource.foo.maximum-pool-size**=30

**app.datasource.bar.url**=jdbc:mysql://localhost/test **app.datasource.bar.username**=dbuser **app.datasource.bar.password**=dbpass **app.datasource.bar.max-total**=30

当然，您也可以将相同的概念应用于辅助数据源：

@Bean @Primary

@ConfigurationProperties("app.datasource.foo")

**public** DataSourceProperties fooDataSourceProperties() {

**return new** DataSourceProperties();

}

@Bean @Primary

@ConfigurationProperties("app.datasource.foo")

**public** DataSource fooDataSource() {

**return** fooDataSourceProperties().initializeDataSourceBuilder().build();

}

@Bean @ConfigurationProperties("app.datasource.bar")

**public** DataSourceProperties barDataSourceProperties() {

**return new** DataSourceProperties();

}

@Bean @ConfigurationProperties("app.datasource.bar") **public** DataSource barDataSource() {

**return** barDataSourceProperties().initializeDataSourceBuilder().build();

}

这个最后的例子在自定义命名空间上配置了两个数据源，它们与Spring Boot在自动配置中所做的相同。

## 使用 Spring Data repositories

Spring Data可以为您创建各种风格的@Repository接口。 只要这些@Repositories包含在@EnableAutoConfiguration类的相同包（或子包）中，Spring Boot将为您处理所有这些。

对于很多应用程序来说，所有你需要的就是把正确的Spring Data依赖关系放到你的类路径中（JPA有一个spring-boot-starter-data-jpa和一个Mongodb的spring-boot-starter-data-mongodb），创建一些 版本库接口来处理你的@Entity对象。 示例在[JPA sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-data-jpa) 或[Mongodb sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-data-mongodb)中。

Spring Boot尝试根据找到的@EnableAutoConfiguration来猜测@Repository定义的位置。 为了获得更多的控制，使用@EnableJpaRepositories注解（从Spring Data JPA）。

## 将@Entity从Spring配置中分离出来

Spring Boot尝试根据找到的@EnableAutoConfiguration来猜测您的@Entity定义的位置。 为了获得更多控制权，您可以使用@EntityScan注释，例如

@Configuration @EnableAutoConfiguration

@EntityScan(basePackageClasses=City.class)

**public class** Application {

*//...*

}

## 配置 JPA properties

Spring Data JPA已经提供了一些与供应商无关的配置选项（例如用于SQL日志记录），Spring Boot公开了这些，还有一些作为外部配置属性的hibernate。 其中一些是根据上下文自动检测，所以你不应该设置它们。

spring.jpa.hibernate.ddl-auto是一种特殊情况，因为它具有不同的默认值，具体取决于您是使用嵌入式数据库（创建 - 删除）还是不使用（none）。 使用的方言也是基于当前的DataSource自动检测的，但是如果你想明确地使用spring.jpa.database，你可以自己设置spring.jpa.database，并绕过启动时的检查。

**Note**

指定一个数据库将导致定义良好的Hibernate方言的配置。 几个数据库有不止一个方言，这可能不适合你的需要。 在这种情况下，您可以将spring.jpa.database设置为默认值，让Hibernate找出事物，或使用spring.jpa.database-platform属性设置方言。

最常见的选项是：

spring.jpa.hibernate.naming.physical-strategy=com.example.MyPhysicalNamingStrategy spring.jpa.show-sql=true

另外，在创建本地EntityManagerFactory时，spring.jpa.properties.\*中的所有属性都作为普通的JPA属性（剥离了前缀）传递。

## 配置 Hibernate Naming Strategy

Spring Boot提供了一致的命名策略，无论您使用的是何种Hibernate代理。 如果你使用的是Hibernate 4，你可以使用spring.jpa.hibernate.naming.strategy来定制它; Hibernate 5定义了一个Physical和Implicit的命名策略。

Spring Boot默认配置SpringPhysicalNamingStrategy。 这个实现提供了与Hibernate 4相同的表结构：所有的点都被下划线替代，而且驼峰也被下划线替换。 默认情况下，所有表名都以小写形式生成，但如果您的模式需要，则可以覆盖该标志。

具体来说，TelephoneNumber实体将被映射到telephone\_number表。

如果您宁愿使用Hibernate 5的默认值，请设置以下属性：

spring.jpa.hibernate.naming.physical- strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

有关更多详细信息，请参阅[HibernateJpaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.java) 和[JpaBaseConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/JpaBaseConfiguration.java) 。

## 使用一个自定义的 EntityManagerFactory

要完全控制EntityManagerFactory的配置，需要添加一个名为“entityManagerFactory”的@Bean。 Spring Boot自动配置基于该类型的bean的存在关闭其实体管理器。

## 使用两个 EntityManagers

即使默认的EntityManagerFactory工作正常，您将需要定义一个新的，否则该类型的第二个bean的存在将关闭默认。 为了简化操作，您可以使用Spring Boot提供的方便的EntityManagerBuilder，或者如果您愿意，可以直接使用Spring ORM中的LocalContainerEntityManagerFactoryBean。

Example:

*// add two data sources configured as above*

@Bean

**public** LocalContainerEntityManagerFactoryBean customerEntityManagerFactory(

EntityManagerFactoryBuilder builder) {

**return** builder

.dataSource(customerDataSource())

.packages(Customer.**class**)

.persistenceUnit(***"customers"***)

.build();

}

@Bean

**public** LocalContainerEntityManagerFactoryBean orderEntityManagerFactory( EntityManagerFactoryBuilder builder) {

**return** builder

.dataSource(orderDataSource())

.packages(Order.**class**)

.persistenceUnit(***"orders"***)

.build();

}

上面的配置几乎可以自行完成。 要完成图片，您还需要为两个EntityManagers配置TransactionManagers。 其中一个可以被Spring Boot中默认的JpaTransactionManager接收，如果你把它标记为@Primary。 另一个必须明确地注入一个新的实例。 或者你也可以使用跨越两者的JTA事务管理器。

如果您使用Spring Data，则需要相应地配置@EnableJpaRepositories：

@Configuration

@EnableJpaRepositories(basePackageClasses = Customer.class, entityManagerFactoryRef = "customerEntityManagerFactory")

**public class** CustomerConfiguration {

...

}

@Configuration

@EnableJpaRepositories(basePackageClasses = Order.class, entityManagerFactoryRef = "orderEntityManagerFactory")

**public class** OrderConfiguration {

...

}

## 使用传统的 traditional persistence.xml

Spring不需要使用XML来配置JPA提供程序，Spring Boot假定您想要利用该功能。 如果你喜欢使用persistence.xml，那么你需要定义自己的@Bean类型LocalEntityManagerFactoryBean（id为'entityManagerFactory'，并在那里设置持久化单元名称。

有关默认设置，请参阅[JpaBaseConfiguration](https://github.com/spring-projects/spring-boot/blob/master/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/JpaBaseConfiguration.java) 。

## 使用 Spring Data JPA 和 Mongo repositories

Spring Data JPA和Spring Data Mongo可以自动为您创建Repository实现。 如果它们都出现在类路径中，那么可能需要做一些额外的配置来告诉Spring Boot您为哪一个（或两者）创建存储库。 最明显的方法是使用标准的Spring Data @ Enable \* Repositories，并告诉它你的Repository接口的位置（其中'\*'是'Jpa'或'Mongo'或两者）。

还有一些标志spring.data.\* .repositories.enabled，您可以使用它们在外部配置中打开和关闭自动配置

的存储库。 例如，如果您想要关闭Mongo软件仓库并仍然使用自动配置的MongoTemplate，这很有用。

其他自动配置的Spring Data存储库类型（Elasticsearch，Solr）也存在同样的障碍和相同的功能。 只需分别更改注释和标志的名称。

## 将Spring数据存储库公开为REST端点

Spring Data REST只要为应用程序启用了Spring MVC，就可以将Repository实现公开为REST端点。

Spring Boot公开了一些自定义[RepositoryRestConfiguration](http://docs.spring.io/spring-data/rest/docs/current/api/org/springframework/data/rest/core/config/RepositoryRestConfiguration.html)的spring.data.rest命名空间的有用属性。 如果你需要提供额外的定制，你应该使用[RepositoryRestConfigurer](http://docs.spring.io/spring-data/rest/docs/current/api/org/springframework/data/rest/webmvc/config/RepositoryRestConfigurer.html) bean。

**Note**

如果你没有在你自定义的RepositoryRestConfigurer中指定任何命令，它将在一个Spring Boot在内部使用后运行。 如果您需要指定订单(order)，请确保它大于0。

## 配置由JPA使用的组件

如果你想配置一个将被JPA使用的组件，那么你需要确保组件在JPA之前被初始化。 如果组件是自动配置的，Spring Boot将为您处理这个问题。 例如，当Flyway自动配置时，Hibernate被配置为依赖于Flyway，这样后者有机会在Hibernate尝试使用它之前初始化数据库。

如果您自己配置组件，则可以使用EntityManagerFactoryDependsOnPostProcessor子类作为设置必需依赖项的便捷方式。 例如，如果您使用Elasticsearch的Hibernate Search作为其索引管理器，那么任何EntityManagerFactory bean都必须配置为依赖于elasticsearchClient bean：

**/\*\***

* **{@link EntityManagerFactoryDependsOnPostProcessor} that ensures that**
* **{@link EntityManagerFactory} beans depend on the {@code elasticsearchClient} bean.**

**\*/**

@Configuration

**static class** ElasticsearchJpaDependencyConfiguration

**extends** EntityManagerFactoryDependsOnPostProcessor {

ElasticsearchJpaDependencyConfiguration() {

**super**(***"elasticsearchClient"***);

}

}

# Database 初始化(initialization)

SQL数据库可以用不同的方式初始化，这取决于你的堆栈是什么。 当然，只要数据库是一个独立的进程，你也可以手动完成。

## 使用JPA初始化数据库

JPA具有用于生成DDL的功能，可以将这些功能设置为在启动时针对数据库运行。 这是通过两个外部属性来控制的：

* spring.jpa.generate-ddl (boolean) switches the feature on and off and is vendor independent.
* spring.jpa.hibernate.ddl-auto (enum) is a Hibernate feature that controls the behavior in a more fine-grained way. See below for more detail.

## 使用Hibernate初始化一个数据

您可以显式设置spring.jpa.hibernate.ddl-auto，标准Hibernate属性值为none，验证，更新，创建，创建和删除。 Spring Boot根据是否认为你的数据库是嵌入的（默认的创建 - 删除）或不是（默认没有）为你选择一个默认值。 通过查看连接类型来检测嵌入式数据库：hsqldb，h2和derby被嵌入，其余不是。 从内存切换到“真实”数据库时请小心，不要假定新平台中存在表和数据。 您必须显式设置ddl-auto，或使用其他机制之一来初始化数据库。

**Note**

您可以通过启用org.hibernate.SQL记录器来输出模式创建。 如果启用[debug mode](#_bookmark121)，这将自动完成。

另外，如果Hibernate从头开始创建模式（即，如果ddl-auto属性设置为create或create-drop），那么在启动时将在启动时执行类路径根目录中名为import.sql的文件。 这对于演示和测试是非常有用的，如果你非常小心的话，可能不是你想要在生产中的类路径上。 这是一个Hibernate功能（与Spring无关）。

## 初始化 database

Spring Boot可以自动创建数据源的模式（DDL脚本）并对其进行初始化（DML脚本）：它分别从标准根类路径位置schema.sql和data.sql中加载SQL。 另外Spring Boot将会处理schema-$ {platform} .sql和data - $ {platform} .sql文件（如果存在），其中platform是spring.datasource.platform的值。 这使您可以根据需要切换到数据库特定的脚本，例如 您可以选择将其设置为数据库的供应商名称（hsqldb，h2，oracle，mysql，postgresql等）。

Spring Boot默认启用Spring JDBC初始化程序的fail-fast功能，所以如果脚本导致异常，应用程序将无法启动。 你可以使用spring.datasource.continue-on-error来调整它。

**Note**

在基于JPA的应用程序中，可以选择让Hibernate创建模式或使用schema.sql，但不能同时使用两者。 如果你选择了后者，请确保禁用spring.jpa.hibernate.ddl-auto。

您也可以通过将spring.datasource.initialize设置为false来禁用初始化。

## 初始化 Spring Batch database

如果您使用的是Spring Batch，那么它将为大多数流行的数据库平台预先打包SQL初始化脚本。 Spring Boot会检测你的数据库类型，并默认执行这些脚本，在这种情况下，会将失败的快速设置切换为假（错误记录，但不阻止应用程序启动）。 这是因为脚本被认为是可靠的，通常不包含错误，所以错误是可以忽略的，忽略它们使得脚本是幂等的。 您可以使用spring.batch.initializer.enabled = false显式地关闭初始化。

## 使用 higher-level database 迁移(migration)工具

Spring Boot支持两种更高级的迁移工具：[Flyway](http://flywaydb.org/) 和 [Liquibase](http://www.liquibase.org/)。

### 再启动时执行Flyway数据库迁移

要在启动时自动运行Flyway数据库迁移，请添加org.flywaydb：flyway-core到你的类路径。

迁移是V <VERSION> <NAME> .sql格式的脚本（带有<VERSION>下划线分隔的版本，例如'1'或'2\_1'）。 默认情况下，它们位于文件夹classpath：db / migration中，但是您可以使用flyway.locations来修改它。 您还可以添加特殊的{vendor}占位符来使用供应商特定的脚本。 假设如下：

**flyway.locations**=db/migration/{vendor}

这个配置不是使用db / migration，而是根据数据库的类型（即db / migration / mysql for MySQL）来设置文件夹的使用。 受支持的数据库列表在[DatabaseDriver](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jdbc/DatabaseDriver.java)中可用。

另请参阅flyway-core的Flyway类以了解可用设置（如模式等）的详细信息。另外，Spring Boot在[FlywayProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayProperties.java) 中提供了一组属性，可用于禁用迁移或关闭位置检查。 Spring Boot将调用Flyway.migrate（）来执行数据库迁移。 如果你想要更多的控制，提供一个实现了[FlywayMigrationStrategy](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayMigrationStrategy.java)的@Bean。

**Tip**

如果您想使用[Flyway callbacks](http://flywaydb.org/documentation/callbacks.html)，那么这些脚本也应该存在于classpath：db / migration文件夹中。

默认情况下，Flyway将在你的上下文中自动装载（@Primary）数据源，并将其用于迁移。 如果你喜欢使用不同的数据源，你可以创建一个@Bean作为@FlywayDataSource - 如果你这样做，记得创建另一个数据源，并将其标记为@Primary，如果你想要两个数据源。 或者你可以通过设置flyway来使用Flyway的本地DataSource。 [url，user，password]在外部属性中。

有一个[Flyway sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-flyway) ，所以你可以看到如何设置的东西。

您也可以使用Flyway为特定场景提供数据。 例如，您可以在src / test / resources中放置特定于测试的迁移，并且只有在您的应用程序开始测试时才会运行它们。 如果您希望更复杂一些，则可以使用特定于配置文件的配置来自定义flyway.locations，以便某些特定配置文件处于活动状态时才能运行某些特定配置。 例如，在application-dev.properties中，您可以将flyway.locations设置为classpath：/ db / migration，classpath：/ dev / db / migration，dev / db / migration中的迁移将只在dev配置文件处于活动状态时运行。

### 在启动时执行Liquibase数据库迁移

要在启动时自动运行Liquibase数据库迁移，添加org.liquibase：liquibase-core到你的类路径。

主(master)更改日志默认从db / changelog / db.changelog-master.yaml中读取，但可以使用liquibase.change-log进行设置。 除了YAML，Liquibase还支持JSON，XML和SQL更改日志格式。

默认情况下，Liquibase会在你的上下文中自动装载（@Primary）数据源，并用它来进行迁移。 如果你喜欢使用不同的数据源，你可以创建一个@Bean作为@LiquibaseDataSource - 如果你这样做的话，记得创建另一个数据源，如果你想要两个数据源，就把它标记为@Primary。 或者你可以通过设置liquibase来使用Liquibase的本地DataSource。 [url，user，password]在外部属性中。

有关上下文，默认模式等可用设置的详细信息，请参阅[LiquibaseProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/liquibase/LiquibaseProperties.java) 。

有一个[Liquibase sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-liquibase) ，所以你可以看到如何设置。

# Messaging

## 禁用 transacted JMS session

如果您的JMS代理不支持事务会话，则必须完全禁用事务支持。 如果您创建自己的JmsListenerContainerFactory，则无需执行任何操作，因为默认情况下不会进行事务处理。 如果您想使用DefaultJmsListenerContainerFactoryConfigurer来重用Spring Boot的默认设置，可以按如下所示禁用事务性会话：

@Bean

**public** DefaultJmsListenerContainerFactory jmsListenerContainerFactory( ConnectionFactory connectionFactory, DefaultJmsListenerContainerFactoryConfigurer configurer) {

DefaultJmsListenerContainerFactory listenerFactory =

**new** DefaultJmsListenerContainerFactory(); configurer.configure(listenerFactory, connectionFactory); listenerFactory.setTransactionManager(null); listenerFactory.setSessionTransacted(false);

**return** listenerFactory;

}

这将覆盖默认的工厂，这应该应用于您的应用程序定义的任何其他工厂（如果有的话）。

# Batch applications

**Note**

默认情况下，批处理应用程序需要一个DataSource来存储作业详细信息。 如果你想偏离这一点，你需要实现BatchConfigurer，请参阅[The Javadoc of](http://docs.spring.io/spring-batch/apidocs/org/springframework/batch/core/configuration/annotation/EnableBatchProcessing.html) [@EnableBatchProcessing](http://docs.spring.io/spring-batch/apidocs/org/springframework/batch/core/configuration/annotation/EnableBatchProcessing.html) 获取更多细节。

## 启动时执行Spring Batch jobs

Spring Batch自动配置是通过在你的上下文中添加@EnableBatchProcessing（从Spring Batch）来实现的。

默认情况下，它会在启动时执行应用程序上下文中的所有作业（有关详细信息，请参阅[JobLauncherCommandLineRunner](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/JobLauncherCommandLineRunner.java)）。 您可以通过指定spring.batch.job.names（逗号分隔(comma-separated)的作业名称模式）来缩小特定作业或作业的范围。

如果应用程序上下文包含JobRegistry，则在注册表中查找spring.batch.job.names中的作业，而不是从上下文自动装入。 这是一个常见的模式，其中有更复杂的系统，其中多个作业在子环境中定义并集中注册。

有关更多详细信息，请参阅[BatchAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/BatchAutoConfiguration.java) 和[@EnableBatchProcessing](https://github.com/spring-projects/spring-batch/blob/master/spring-batch-core/src/main/java/org/springframework/batch/core/configuration/annotation/EnableBatchProcessing.java) 。

# Actuator(执行器)

## 更改执行端点的HTTP端口(port)或地址(address)

在独立应用程序中，Actuator HTTP端口默认与主HTTP端口相同。 为了让应用程序在不同的端口上侦听，设置了外部属性management.port。 要监听完全不同的网络地址（例如，如果您有一个内部网络用于管理，而另一个用于外部用户应用程序），您还可以将management.address设置为服务器能够绑定到的有效IP地址。

有关更多详细信息，请参阅[ManagementServerProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/ManagementServerProperties.java) 源代码和“[*Section 48.3,*](#_bookmark317)[*“Customizing the management server port”*](#_bookmark317) 。

## 自定义 ‘whitelabel’ error page

如果您遇到服务器错误（使用JSON和其他media types的计算机客户端应该看到具有正确错误代码的合理响应），Spring Boot将安装一个“浏览器客户端”错误页面。

**Note**

设置server.error.whitelabel.enabled = false将关闭的默认错误页面恢复为您正在使用的servlet容器的默认值。 请注意，Spring Boot仍然会尝试解决错误视图，所以你可能会添加你自己的错误页面，而不是完全禁用它。

用你自己的覆盖错误页取决于你正在使用的模板技术。 例如，如果您使用的是Thymeleaf，则可以添加一个error.html模板，如果您使用的是FreeMarker，则可以添加一个error.ftl模板。 一般来说，你需要的是一个用错误名称解析的视图，和/或处理/错误路径的@Controller。 除非你替换了一些默认配置，否则你应该在你的ApplicationContext中找到一个BeanNameViewResolver，所以带有id错误的@Bean将是一个简单的方法。 查看[ErrorMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration.java) 以获取更多选项。

有关如何在servlet容器中注册处理程序的详细信息，另请参阅错误处理([Error Handling](#_bookmark139) )一节。

## Actuator 和 Jersey

执行器(Actuator)HTTP端点仅适用于基于Spring MVC的应用程序。 如果您想使用Jersey并仍然使用执行器，则需要启用Spring MVC（例如，取决于spring-boot-starter-web）。 默认情况下，Jersey和Spring MVC调度程序servlet都映射到相同的路径（/）。 您将需要更改其中之一的路径（通过为Spring MVC配置server.servlet-path或为Jersey配置spring.jersey.application-path）。 例如，如果将server.servlet-path = / system添加到application.properties中，则执行器HTTP端点将在/ system下可用。

# Security

## 关闭 Spring Boot security configuration

如果您在应用程序中的任何位置使用@EnableWebSecurity定义@Configuration，它将关闭Spring Boot中的默认Web应用程序安全设置（但启用Actuator的安全性）。 要调整默认值，请尝试设置security.\*（请参阅[SecurityProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/SecurityProperties.java) 以获取可用设置的详细信息）以及[Common application properties](#_bookmark570)的SECURITY部分。

## 更改 AuthenticationManager 并添加用户账户

如果你提供了一个类型为AuthenticationManager的@Bean，那么将不会创建默认的，所以你可以使用Spring Security的完整功能集（例如[various authentication options](http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#jc-authentication)）。

Spring Security还提供了一个方便的AuthenticationManagerBuilder，它可以用来构建一个带有公共选项的AuthenticationManager。 在webapp中使用这种方法的推荐方法是将其注入WebSecurityConfigurerAdapter中的void方法，

@Configuration

**public class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

**public void** configureGlobal(AuthenticationManagerBuilder auth) **throws** Exception { auth.inMemoryAuthentication()

.withUser(***"barry"***).password(***"password"***).roles(***"USER"***); *// ... etc.*

}

*// ... other stuff for application security*

}

如果将其放在嵌套类或独立类中（即不会混入大量可能影响实例化顺序的其他@Beans），您将获得最佳结果。 [secure web](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-web-secure) [sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-web-secure) 是一个有用的模板。

如果遇到实例化问题（例如，使用JDBC或JPA作为用户详细信息存储），可能需要将AuthenticationManagerBuilder回调提取到GlobalAuthenticationConfigurerAdapter（在init（）方法中，以便在其他地方需要身份验证管理器之前进行）。

@Configuration

**public class** AuthenticationManagerConfiguration **extends**

GlobalAuthenticationConfigurerAdapter {

@Override

**public void** init(AuthenticationManagerBuilder auth) { auth.inMemoryAuthentication() *// ... etc.*

}

}

## 在代理服务器上运行时启用HTTPS

确保您的所有主要端点只能通过HTTPS使用，这对于任何应用程序来说都是一件非常重要的事情。 如果您使用Tomcat作为servlet容器，那么Spring Boot会在检测到某些环境设置时自动添加Tomcat自己的RemoteIpValve，并且您应该能够依赖HttpServletRequest来报告它是否安全（即使是代理的下游 服务器处理真正的SSL终止）。 标准行为是由某些请求头(headers)（x-forwarded-for和x-forwarded-proto）的存在或不存在决定的，它们的名字是常规的，所以它应该与大多数前端代理一起工作。 您可以通过在application.properties中添加一些条目来打开阀门。

**server.tomcat.remote-ip-header**=x-forwarded-for **server.tomcat.protocol-header**=x-forwarded-proto

（任何一个属性的存在都会打开阀门，或者您可以添加RemoteIpValve通过添加一个TomcatEmbeddedServletContainerFactory bean）。

Spring Security也可以被配置为需要所有（或某些请求）的安全通道。 要在Spring Boot应用程序中将其切换，只需在application.properties中将security.require\_ssl设置为true即可。

# Hot swapping

## Reload static content

有几个热重载的选项。 推荐的方法是使用[spring-boot-](#_bookmark67) [devtools](#_bookmark67) ，因为它提供了额外的开发时间功能，如支持快速应用程序重启和LiveReload以及合理的开发时配置（例如模板缓存）。 Devtools通过监视类路径的变化来工作。 这意味着静态资源的变化必须“build”，以使变更生效。 默认情况下，当您保存更改时，这会在Eclipse中自动发生。 在IntelliJ IDEA中，Make Project将触发必要的构建。 由于默认的重新启动([default restart](#_bookmark71) [exclusions](#_bookmark71))排除，对静态资源的更改不会触发应用程序的重新启动。 但是，他们会触发一个实时的重新加载。

另外，在IDE中运行（特别是在调试时）是一种很好的开发方式（所有的现代IDE都允许重新加载静态资源，通常也可以热切换Java类的变化）。

最后，可以配置[Maven and Gradle plugins](#_bookmark407) （请参阅addResources属性）以支持从命令行直接从源重新加载静态文件。 如果您使用更高级别的工具编写代码，则可以将其用于外部css / js编译器进程。

## 重新加载模板而不重新启动容器

Spring Boot支持的大部分模板技术都包含禁用缓存的配置选项（请参阅下面的详细信息）。 如果您使用的是spring-boot-devtools模块，则在开发时这些属性将自动为您配置[automatically configured](#_bookmark68) 。

### Thymeleaf templates

如果您使用的是Thymeleaf，则将spring.thymeleaf.cache设置为false。 看到

[ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java) 其他Thymeleaf自定义选项。

### FreeMarker templates

如果您正在使用FreeMarker，请将spring.freemarker.cache设置为false。 看到

[FreeMarkerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java) 适用于其他FreeMarker自定义选项。

### Groovy templates

如果您正在使用Groovy模板，请将spring.groovy.template.cache设置为false。 看到

[GroovyTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.java) 用于其他Groovy自定义选项。

## application 快速重启

spring-boot-devtools模块包含对自动应用程序重新启动的支持。 虽然速度不及[JRebel](http://zeroturnaround.com/software/jrebel/) 或 [Spring Loaded](https://github.com/spring-projects/spring-loaded) 等技术，但通常比“冷启动”要快得多。 在调查下面讨论的一些更复杂的重新加载选项之前，您应该尝试一下。

有关更多详细信息，请参阅[Chapter 20, *Developer tools*](#_bookmark67) 部分。

## 重新加载Java类而不重新启动容器

现代的IDE（Eclipse，IDEA等）都支持字节码的热插拔，所以如果你做了一个不影响类或方法签名的改变，它应该干净地重新加载而没有副作用。

[Spring Loaded](https://github.com/spring-projects/spring-loaded) 进一步发展，它可以通过方法签名中的更改来重新加载类定义。 通过一些自定义，它可以强制一个ApplicationContext自己刷新（但是没有一个通用的机制来确保对于正在运行的应用程序来说是安全的，所以它可能只是一个开发时间技巧）。

### 用 Maven 配置 Spring Loaded

要使用Spring加载的Maven命令行，只需将其作为Spring Boot插件声明的依赖项添加即可。

**<plugin>**

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

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

**<dependencies>**

**<dependency>**

**<groupId>**org.springframework**</groupId>**

**<artifactId>**springloaded**</artifactId>**

**<version>**1.2.6.RELEASE**</version>**

**</dependency>**

**</dependencies>**

**</plugin>**

对于Eclipse和IntelliJ IDEA，只要他们的构建配置与Maven默认设置保持一致（Eclipse m2e可以实现这一功能），这通常可以很好地工作。

### 使用Gradle和IntelliJ IDEA配置Spring Loaded

如果你想使用Spring Loaded结合Gradle和IntelliJ IDEA，你需要跳过一些箍环。 默认情况下，IntelliJ IDEA会将类编译到与Gradle不同的位置，从而导致Spring Loaded监视失败。

要正确配置IntelliJ IDEA，您可以使用Gradle插件：

buildscript {

repositories { jcenter() } dependencies {

classpath ***"org.springframework.boot:spring-boot-gradle-plugin:1.5.8.RELEASE"***

classpath ***'org.springframework:springloaded:1.2.6.RELEASE'***

}

}

apply plugin: ***'idea'***

idea {

module {

inheritOutputDirs = false

outputDir = file(***"$buildDir/classes/main/"***)

}

}

*// ...*

**Note**

必须将IntelliJ IDEA配置为使用与命令行Gradle任务相同的Java版本，并且必须将springloaded作为构建脚本依赖项包含在内。

您还可以在IntelliJ IDEA中另外启用“automatically compile”，以便在保存文件时自动编译您的代码。

# Build

## 生成 build information

Maven和Gradle插件都允许生成包含项目坐标，名称和版本的构建信息。 该插件也可以配置为通过配置添加其他属性。 当这个文件出现时，Spring Boot会自动配置一个BuildProperties bean。

要使用Maven生成构建信息，请为build-info goal添加一个执行：

**<build>**

**<plugins>**

**<plugin>**

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

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

**<version>**1.5.8.RELEASE**</version>**

**<executions>**

**<execution>**

**<goals>**

**<goal>**build-info**</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

**</plugins>**

**</build>**

**Tip**

查看[Spring Boot Maven Plugin documentation](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/) 以获取更多详细信息。

和 Gradle 一样:

springBoot {

buildInfo()

}

可以使用DSL添加其他属性：

springBoot {

buildInfo {

additionalProperties = [

***'foo'***: ***'bar'***

]

}

}

## 生成 git information

Maven和Gradle都允许在构建项目时生成一个git.properties文件，其中包含有关git源代码库状态的信息。

对于Maven用户来说，spring-boot-starter-parent POM包含一个预先配置的插件来生成一个git.properties文件。 只需将以下声明添加到您的POM：

**<build>**

**<plugins>**

**<plugin>**

**<groupId>**pl.project13.maven**</groupId>**

**<artifactId>**git-commit-id-plugin**</artifactId>**

**</plugin>**

**</plugins>**

**</build>**

Gradle用户可以使用gradle-git-properties插件获得相同的结果

plugins {

id ***"com.gorylenko.gradle-git-properties"*** version ***"1.4.17"***

}

**Tip**

预计git.properties中的提交时间将与格式yyyy-MM-dd'T'HH：mm：ssZ匹配。 这是上面列出的两个插件的默认格式。 使用这种格式可以将时间解析成日期和格式，并在序列化为JSON时由Jackson的日期序列化配置设置进行控制。

## 自定义 dependency versions

如果您使用直接或间接从spring-boot-dependencies（例如spring-boot-starter-parent）继承的Maven构建，但是您希望覆盖特定的第三方依赖关系，则可以添加适当的<properties>元素。 浏览[spring-boot-dependencies](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-dependencies/pom.xml)  POM以获取完整的属性列表。 例如，要选择一个不同的slf4j版本，您可以添加以下内容：

**<properties>**

**<slf4j.version>**1.7.5**<slf4j.version>**

**</properties>**

**Note**

这只有在你的Maven项目（直接或间接）从spring-boot-dependencies继承时才有效。 如果你在你自己的dependencyManagement部分中用<scope> import </ scope>添加了spring-boot-dependencies，那么你必须自己重新定义这个artifact，而不是重写属性。

**Warning**

每个Spring Boot版本都是针对特定的第三方依赖关系进行设计和测试的。 覆盖版本可能会导致兼容性问题。

要覆盖Gradle中的依赖版本，您可以指定一个版本，如下所示：

ext[***'slf4j.version'***] = ***'1.7.5'***

有关更多信息，请参阅[Gradle Dependency Management Plugin documentation](https://github.com/spring-gradle-plugins/dependency-management-plugin)。

## 使用Maven创建一个可执行的 JAR

spring-boot-maven-plugin可以用来创建一个可执行的“fat”JAR。 如果您使用的是spring-boot-starter-parent POM，则可以简单地声明该插件，并将您的jar包重新打包：

**<build>**

**<plugins>**

**<plugin>**

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

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

**</plugin>**

**</plugins>**

**</build>**

如果你不使用父POM，你仍然可以使用这个插件，但是你必须另外添加一个<executions>部分：

**<build>**

**<plugins>**

**<plugin>**

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

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

**<version>**1.5.8.RELEASE**</version>**

**<executions>**

**<execution>**

**<goals>**

**<goal>**repackage**</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

**</plugins>**

**</build>**

有关完整的使用详细信息，请参阅[plugin documentation](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/usage.html) 。

## 使用 Spring Boot application 作为依赖项

像war文件一样，Spring Boot应用程序并不打算用作依赖项。 如果您的应用程序包含要与其他项目共享的类，则推荐的方法是将该代码移动到单独的模块中。 这个单独的模块可以依靠你的应用程序和其他项目。

如果你不能按照上面的建议重新安排你的代码，那么Spring Boot的Maven和Gradle插件必须被配置为产生一个独立的工件，它适合作为一个依赖项。 可执行文件不能用作依赖项，因为[executable jar format](#_bookmark595) 在BOOT-INF / classes中打包应用程序类。 这意味着当可执行jar被用作依赖时，它们不能被找到。

为了生成两个artifacts，一个可以用作依赖关系，一个可执行工件，必须指定一个分类器(classifier)。 此分类器应用于可执行档案(executable archive)的名称，保留用作依赖项的默认档案。

要在Maven中配置exec分类器，可以使用以下配置：

**<build>**

**<plugins>**

**<plugin>**

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

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

**<configuration>**

**<classifier>**exec**</classifier>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

在使用Gradle时，可以使用以下配置：

bootRepackage { classifier = ***'exec'***

}

## 当一个可执行的jar运行时，提取特定的库(Extract specific libraries)

可执行jar中的大多数嵌套库不需要解压缩以便运行，但某些库可能会遇到问题。 例如，JRuby包含自己的嵌套jar支持，它假设jruby-complete.jar总是直接作为一个文件直接使用。

为了处理任何有问题的库，可以标记特定的嵌套jar应该在可执行jar第一次运行时自动解压到“temp文件夹”。

例如，为了说明JRuby应该被标记为使用Maven Plugin解压缩，您可以添加以下配置：

**<build>**

**<plugins>**

**<plugin>**

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

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

**<configuration>**

**<requiresUnpack>**

**<dependency>**

**<groupId>**org.jruby**</groupId>**

**<artifactId>**jruby-complete**</artifactId>**

**</dependency>**

**</requiresUnpack>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

和Gradle一样：

springBoot {

requiresUnpack = [***'org.jruby:jruby-complete'***]

}

## 使用exclusions 创建 non-executable JAR

通常，如果您将可执行文件和不可执行的jar作为构建产品，则可执行文件版本将具有库jar中不需要的其他配置文件。 例如。 application.yml配置文件可能会从不可执行的JAR中排除。

下面是在Maven中如何做到这一点：

**<build>**

**<plugins>**

**<plugin>**

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

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

**<configuration>**

**<classifier>**exec**</classifier>**

**</configuration>**

**</plugin>**

**<plugin>**

**<artifactId>**maven-jar-plugin**</artifactId>**

**<executions>**

**<execution>**

**<id>**exec**</id>**

**<phase>**package**</phase>**

**<goals>**

**<goal>**jar**</goal>**

**</goals>**

**<configuration>**

**<classifier>**exec**</classifier>**

**</configuration>**

**</execution>**

**<execution>**

**<phase>**package**</phase>**

**<goals>**

**<goal>**jar**</goal>**

**</goals>**

**<configuration>**

*<!-- Need this to ensure application.yml is excluded -->*

**<forceCreation>**true**</forceCreation>**

**<excludes>**

**<exclude>**application.yml**</exclude>**

**</excludes>**

**</configuration>**

**</execution>**

**</executions>**

**</plugin>**

**</plugins>**

**</build>**

在Gradle中，您可以使用标准的任务DSL功能创建一个新的JAR归档(archive)文件，然后使用

bootRepackage任务取决于使用其withJarTask属性的那个：

jar {

baseName = ***'spring-boot-sample-profile'***

version = ***'0.0.0'***

excludes = [***'\*\*/application.yml'***]

}

task(***'execJar'***, type:Jar, dependsOn: ***'jar'***) { baseName = ***'spring-boot-sample-profile'*** version = ***'0.0.0'***

classifier = ***'exec'***

from sourceSets.main.output

}

bootRepackage {

withJarTask = tasks[***'execJar'***]

}

## 远程调试使用Maven启动的Spring Boot应用程序

要将一个远程调试器附加到一个使用Maven开始的Spring Boot应用程序中，您可以使用

jvmArguments属性的[maven plugin](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/)。

查看 [this example](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/maven-plugin/examples/run-debug.html) 获取更多详细信息

## 远程调试使用Gradle启动的Spring Boot application

要将远程调试器附加到以Gradle启动的Spring Boot应用程序，可以使用jvmArgs

bootRun任务的属性或--debug-jvm命令行选项。

build.gradle:

bootRun {

jvmArgs ***"-agentlib:jdwp=transport=dt\_socket,server=y,suspend=y,address=5005"***

}

Command line:

$ gradle bootRun --debug-jvm

查看 [Gradle Application Plugin](http://www.gradle.org/docs/current/userguide/application_plugin.html) 获取更多详细信息.

## 在不使用spring-boot-antlib的情况下从Ant构建可执行文件

要用Ant构建，您需要获取依赖关系，编译然后创建jar或war归档文件。 要使其可执行，您可以使用spring-boot-antlib模块，也可以按照以下说明操作：

1. 如果您正在构建jar，请将应用程序的类和资源打包到嵌套的BOOT-INF / classes目录中。 如果你正在建立war，像往常一样将应用程序的类打包在一个嵌套的WEB-INF / classes目录中。
2. 将运行时依赖关系添加到用于jar或WEB-INF / lib的嵌套BOOT-INF / lib目录中以进行war。 切记不要压缩存档中的条目。
3. 将提供的（嵌入容器）依赖关系添加到为war提供的jar或WEB-INF / lib的嵌套BOOT-INF / lib目录中。 切记不要压缩存档中的条目。
4. 在档案的根目录添加spring-boot-loader类（所以Main-Class可用）。
5. 使用适当的发射器(launcher)，例如 JarLauncher用于jar文件，作为清单中的Main-Class属性，并指定其作为清单条目所需的其他属性，主要是Start-Class。

Example:

**<target name**=**"build" depends**=**"compile">**

**<jar destfile**=**"target/${ant.project.name}-${spring-boot.version}.jar" compress**=**"false">**

**<mappedresources>**

**<fileset dir**=**"target/classes" />**

**<globmapper from**=**"\*" to**=**"BOOT-INF/classes/\*"/>**

**</mappedresources>**

**<mappedresources>**

**<fileset dir**=**"src/main/resources" erroronmissingdir**=**"false"/>**

**<globmapper from**=**"\*" to**=**"BOOT-INF/classes/\*"/>**

**</mappedresources>**

**<mappedresources>**

**<fileset dir**=**"${lib.dir}/runtime" />**

**<globmapper from**=**"\*" to**=**"BOOT-INF/lib/\*"/>**

**</mappedresources>**

**<zipfileset src**=**"${lib.dir}/loader/spring-boot-loader-jar-${spring-boot.version}.jar" />**

**<manifest>**

**<attribute name**=**"Main-Class" value**=**"org.springframework.boot.loader.JarLauncher" />**

**<attribute name**=**"Start-Class" value**=**"${start-class}" />**

**</manifest>**

**</jar>**

**</target>**

[Ant Sample](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-ant) 具有一个build.xml，其中包含一个manual task，如果您运行它，该任务应该可以正常工作

$ ant -lib <folder containing ivy-2.2.jar> clean manual

之后你可以运行应用程序

$ java -jar target/\*.jar

## 如何使用 Java 6

如果你想在Java 6中使用Spring Boot，那么你需要做少量的配置修改。 确切的更改取决于您的应用程序的功能。

### 嵌入式 servlet container 兼容性

如果您正在使用Boot的嵌入式Servlet容器之一，则必须使用Java 6兼容的容器。 Tomcat 7和Jetty 8都与Java 6兼容。 有关详细信息，请参见[Section 73.16, “Use Tomcat 7.x or](#_bookmark476) [8.0”](#_bookmark476) and [Section 73.18, “Use Jetty 8”](#_bookmark482) 。

### Jackson

Jackson 2.7及更高版本需要Java 7.如果你想使用Jackson与Java 6，你将不得不降级到Jackson 2.6。

Spring Boot使用从Jackson 2.7引入的Jackson BOM，因此您不能只重写jackson.version属性。 为了使用Jackson 2.6，必须在构建的dependencyManagement部分中定义各个模块，请查看[this example](https://github.com/spring-projects/spring-boot/blob/0ffc7dc13f6de82c199a6d503354a88c7aaec2d9/spring-boot-dependencies/pom.xml#L523-L597) 以获取更多详细信息。

### JTA API 兼容性

虽然Java Transaction API本身不需要Java 7，但是官方的API jar包含了已经被构建为需要Java 7的类。如果您使用的是JTA，那么您将需要将官方的JTA 1.2 API jar替换为已经构建的jar 使用Java 6.为此，请排除javax.transaction上的任何传递依赖项：javax.transaction-api并将其替换为org.jboss.spec.javax.transaction上的依赖项：jboss-transaction-api\_1.2\_spec：1.0.0.Final

# Traditional deployment

## Create a deployable war file

生成可部署的war文件的第一步是提供一个SpringBootServletInitializer子类并覆盖其配置方法。 这使得Spring框架的Servlet 3.0支持，并允许你配置你的应用程序，当它由servlet容器启动。 通常，您更新您的应用程序的主类来扩展SpringBootServletInitializer：

@SpringBootApplication

**public class** Application **extends** SpringBootServletInitializer {

@Override

**protected** SpringApplicationBuilder configure(SpringApplicationBuilder application) {

**return** application.sources(Application.**class**);

}

**public static void** main(String[] args) **throws** Exception { SpringApplication.run(Application.**class**, args);

}

}

下一步是更新您的构建配置，以便您的项目生成一个war文件，而不是一个jar文件。 如果您使用的是Maven，并使用spring-boot-starter-parent（为您配置Maven的war插件），您只需修改pom.xml以将打包更改为war：

**<packaging>**war**</packaging>**

如果您使用的是Gradle，则需要修改build.gradle以将war插件应用到项目中：

apply plugin: ***'war'***

这个过程的最后一步是确保嵌入的servlet容器不会干扰要部署war文件的servlet容器。 为此，您需要标记所提供的嵌入式servlet容器依赖项。

如果你使用 Maven:

**<dependencies>**

*<!-- … -->*

**<dependency>**

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

**<artifactId>**spring-boot-starter-tomcat**</artifactId>**

**<scope>**provided**</scope>**

**</dependency>**

*<!-- … -->*

**</dependencies>**

如果你是用 Gradle:

dependencies {

*// …*

providedRuntime ***'org.springframework.boot:spring-boot-starter-tomcat'***

*// …*

}

**Note**

如果您使用的是支持仅编译依赖项（2.12或更高版本）的Gradle版本，则应该继续使用providedRuntime。 在其他限制中，compileOnly依赖不在测试类路径上，所以任何基于Web的集成测试都将失败。

如果您使用的是[Spring Boot build tools](#_bookmark407)，则按照提供的方式标记嵌入的Servlet容器依赖项将生成一个可执行的war文件，其中提供的依赖项打包在lib提供的目录中。 这意味着，除了可以部署到servlet容器之外，还可以在命令行上使用java -jar运行应用程序。

**Tip**

看看Spring Boot的示例应用程序，以获得上述配置的基于[Maven-based example](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-samples/spring-boot-sample-traditional/pom.xml)。

## Create a deployable war file for older servlet containers

较老的Servlet容器不支持Servlet 3.0中使用的ServletContextInitializer引导进程。 您仍然可以在这些容器中使用Spring和Spring Boot，但是您将需要将web.xml添加到您的应用程序，并将其配置为通过DispatcherServlet加载ApplicationContext。

## Convert an existing application to Spring Boot

对于非web应用程序来说，应该很简单（丢弃创建ApplicationContext的代码并将其替换为对SpringApplication或SpringApplicationBuilder的调用）。 Spring MVC Web应用程序通常可以首先创建可部署的战争应用程序，然后将其稍后迁移到可执行的战争和/或jar。 有用的阅读材料在“将jar转换为war的[Getting Started Guide on Converting a jar to a war](http://spring.io/guides/gs/convert-jar-to-war/)”中。

通过扩展SpringBootServletInitializer来创建可部署的war（例如在一个名为应用程序），

并添加Spring Boot @SpringBootApplication注释。 例：

@SpringBootApplication

**public class** Application **extends** SpringBootServletInitializer {

@Override

**protected** SpringApplicationBuilder configure(SpringApplicationBuilder application) {

*// Customize the application or call application.sources(...) to add sources*

*// Since our example is itself a @Configuration class (via @SpringBootApplication)*

*// we actually don't need to override this method.*

**return** application;

}

}

请记住，无论你放在源代码中，只是一个Spring ApplicationContext，通常任何已经工作的东西都应该在这里工作。 可能稍后会删除一些bean，并让Spring Boot为它们提供自己的默认值，但应该可以先做一些工作。

可以将静态资源移动到类路径根中的/public (or /static or /resources or /META-INF/resources)。 messages.properties一样（Spring Boot会在类路径的根目录中自动检测到这个）。

Spring DispatcherServlet和Spring Security的Vanilla使用不需要进一步的修改。 如果您的应用程序中有其他功能，例如使用其他servlet或过滤器，那么您可能需要在Application上下文中添加一些配置，从web.xml中替换这些元素，如下所示：

* Servlet或ServletRegistrationBean类型的@Bean将该bean安装在容器中，就好像它是web.xml中的<servlet />和<servlet-mapping />一样。
* 类型为Filter或FilterRegistrationBean的@Bean的行为类似(如<filter/和<filter-mapping>)
* XML文件中的ApplicationContext可以通过应用程序中的@ImportResource添加。 或者注释配置大量使用的简单情况已经可以作为@Bean定义在几行中重新创建。

一旦war正在进行，我们通过向我们的应用程序添加主要方法来使其可执行。

**public static void** main(String[] args) { SpringApplication.run(Application.**class**, args);

}

**Note**

如果您打算将应用程序作为战争或可执行应用程序启动，则需要在SpringBootServletInitializer回调和主方法都可用的方法中共享构建器的自定义设置，如下所示：

@SpringBootApplication

**public class** Application **extends** SpringBootServletInitializer {

@Override

**protected** SpringApplicationBuilder configure(SpringApplicationBuilder builder) {

**return** configureApplication(builder);

}

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

configureApplication(**new** SpringApplicationBuilder()).run(args);

}

**private static** SpringApplicationBuilder configureApplication(SpringApplicationBuilder builder) {

**return** builder.sources(Application.**class**).bannerMode(Banner.Mode.OFF);

}

}

应用程序可以分为多个类别：

* Servlet 3.0+ applications with no web.xml.
* Applications with a web.xml.
* Applications with a context hierarchy.
* Applications without a context hierarchy.

所有这些都应该适合翻译，但每个可能需要稍微不同的技巧。

如果Servlet 3.0+应用程序已经使用Spring Servlet 3.0+初始化程序支持类，那么它们可能会非常容易转换。 通常来自现有的WebApplicationInitializer的所有代码可以移动到SpringBootServletInitializer中。 如果您的现有应用程序有多个ApplicationContext（例如，如果它使用stractDispatcherServletInitializer），那么您可能能够将所有上下文源压缩到一个SpringApplication中。 您可能遇到的主要难题是如果这不起作用，您需要维护上下文层次结构。 请参阅构[entry on building a hierarchy](#_bookmark443) 。 包含Web特定功能的现有父上下文通常需要分解，以便所有ServletContextAware组件都在子上下文中。

不是Spring应用程序的应用程序可能会转换成Spring Boot应用程序，上面的指导可能会有帮助，但是您的里程可能会有所不同。

## Deploying a WAR to WebLogic

要将Spring Boot应用程序部署到WebLogic，您必须确保您的Servlet初始化程序直接实现WebApplicationInitializer（即使从已经实现它的基类中进行扩展）。

一个典型的WebLogic初始化器就是这样的：

**import** org.springframework.boot.autoconfigure.SpringBootApplication; **import** org.springframework.boot.context.web.SpringBootServletInitializer; **import** org.springframework.web.WebApplicationInitializer;

@SpringBootApplication

**public class** MyApplication **extends** SpringBootServletInitializer **implements** WebApplicationInitializer {

}

如果使用logback，则还需要告知WebLogic更喜欢打包的版本，而不是预先安装在服务器上的版本。 您可以通过添加具有以下内容的WEB-INF / weblogic.xml文件来执行此操作：

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

**<wls:weblogic-web-app xmlns:wls**=[**"http://xmlns.oracle.com/weblogic/weblogic-web-app"**](http://xmlns.oracle.com/weblogic/weblogic-web-app) **xmlns:xsi**=[**"http://www.w3.org/2001/XMLSchema-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xsi:schemaLocation**=[**"http://java.sun.com/xml/ns/javaee**](http://java.sun.com/xml/ns/javaee)

[**http://java.sun.com/xml/ns/javaee/ejb-jar\_3\_0.xsd**](http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd)[**http://xmlns.oracle.com/weblogic/weblogic-web-app**](http://xmlns.oracle.com/weblogic/weblogic-web-app)[**http://xmlns.oracle.com/weblogic/weblogic-web-app/1.4/weblogic-web-app.xsd"**](http://xmlns.oracle.com/weblogic/weblogic-web-app/1.4/weblogic-web-app.xsd)**>**

**<wls:container-descriptor>**

**<wls:prefer-application-packages>**

**<wls:package-name>**org.slf4j**</wls:package-name>**

**</wls:prefer-application-packages>**

**</wls:container-descriptor>**

**</wls:weblogic-web-app>**

## Deploying a WAR in an Old (Servlet 2.5) Container

Spring Boot使用Servlet 3.0 API来初始化ServletContext（注册Servlets等），所以不能在Servlet 2.5容器中使用相同的应用程序。 然而，使用一些特殊的工具可以在一个较老的容器上运行Spring Boot应用程序。 如果您将org.springframework.boot：spring-boot-legacy作为依赖项（独立于Spring Boot的核心并且目前在1.0.2.RELEASE上提供[maintained separately](https://github.com/scratches/spring-boot-legacy) ），那么您只需要创建一个web.xml并 声明一个上下文监听器来创建应用程序上下文以及你的过滤器和servlet。 上下文监听器是Spring Boot的一个特殊用途，但其余部分对于Servlet 2.5中的Spring应用程序来说是正常的。 例：

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

**<web-app version**=**"2.5" xmlns**=[**"http://java.sun.com/xml/ns/javaee"**](http://java.sun.com/xml/ns/javaee) **xmlns:xsi**=[**"http://www.w3.org/2001/XMLSchema-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xsi:schemaLocation**=[**"http://java.sun.com/xml/ns/javaee**](http://java.sun.com/xml/ns/javaee)[**http://java.sun.com/xml/ns/javaee/web-**](http://java.sun.com/xml/ns/javaee/web-)

**app\_2\_5.xsd">**

**<context-param>**

**<param-name>**contextConfigLocation**</param-name>**

**<param-value>**demo.Application**</param-value>**

**</context-param>**

**<listener>**

**<listener-class>**org.springframework.boot.legacy.context.web.SpringBootContextLoaderListener**</ listener-class>**

**</listener>**

**<filter>**

**<filter-name>**metricsFilter**</filter-name>**

**<filter-class>**org.springframework.web.filter.DelegatingFilterProxy**</filter-class>**

**</filter>**

**<filter-mapping>**

**<filter-name>**metricsFilter**</filter-name>**

**<url-pattern>**/\***</url-pattern>**

**</filter-mapping>**

**<servlet>**

**<servlet-name>**appServlet**</servlet-name>**

**<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**

**<init-param>**

**<param-name>**contextAttribute**</param-name>**

**<param-value>**org.springframework.web.context.WebApplicationContext.ROOT**</param-value>**

**</init-param>**

**<load-on-startup>**1**</load-on-startup>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**appServlet**</servlet-name>**

**<url-pattern>**/**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

在这个例子中，我们使用一个应用程序上下文（由上下文监听器创建的上下文），并使用init参数将其附加到DispatcherServlet。 这在Spring Boot应用程序中是正常的（通常你只有一个应用程序上下文）。

**Part X. 附录**

**附录 A. Common application properties**

可以在application.properties/application.yml文件中指定各种属性，或者作为命令行开关。 本节提供了常用Spring Boot属性的列表以及对使用它们的基础类的引用。

**Note**

属性贡献可以来自你的类路径上的其他jar文件，所以你不应该认为这是一个详尽的列表。 定义自己的属性也是完全合法的。

**Warning**

此示例文件仅作为指导。 不要将整个内容复制/粘贴到应用程序中; 而只选择你需要的属性。

*# =================================================================== # COMMON SPRING BOOT PROPERTIES*

*#*

*# This sample file is provided as a guideline. Do NOT copy it in its # entirety to your own application. ^^^*

*# ===================================================================*

*# ---------------------------------------- # CORE PROPERTIES*

*# ----------------------------------------*

*# BANNER*

**banner.charset**=UTF-8 *# Banner file encoding.*

**banner.location**=classpath:banner.txt *# Banner file location.* **banner.image.location**=classpath:banner.gif *# Banner image file location (jpg/png can also be used).* **banner.image.width**= *# Width of the banner image in chars (default 76)*

**banner.image.height**= *# Height of the banner image in chars (default based on image height)*

**banner.image.margin**= *# Left hand image margin in chars (default 2)*

**banner.image.invert**= *# If images should be inverted for dark terminal themes (default false)*

*# LOGGING*

**logging.config**= *# Location of the logging configuration file. For instance `classpath:logback.xml` for Logback*

**logging.exception-conversion-word**=%wEx *# Conversion word used when logging exceptions.*

**logging.file**= *# Log file name. For instance `myapp.log`*

**logging.level.\***= *# Log levels severity mapping. For instance `logging.level.org.springframework=DEBUG`*

**logging.path**= *# Location of the log file. For instance `/var/log`*

**logging.pattern.console**= *# Appender pattern for output to the console. Only supported with the default logback setup.*

**logging.pattern.file**= *# Appender pattern for output to the file. Only supported with the default logback setup.*

**logging.pattern.level**= *# Appender pattern for log level (default %5p). Only supported with the default logback setup.*

**logging.register-shutdown-hook**=false *# Register a shutdown hook for the logging system when it is initialized.*

*# AOP*

**spring.aop.auto**=true *# Add @EnableAspectJAutoProxy.*

**spring.aop.proxy-target-class**= *# Whether subclass-based (CGLIB) proxies are to be created (true) as opposed to standard Java interface-based proxies (false). Defaults to "true" when using Spring Transaction Management, otherwise "false".*

*# IDENTITY (*[ContextIdApplicationContextInitializer](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/context/ContextIdApplicationContextInitializer.java))

**spring.application.index**= *# Application index.*

**spring.application.name**= *# Application name.*

*# ADMIN (*[SpringApplicationAdminJmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/admin/SpringApplicationAdminJmxAutoConfiguration.java))

**spring.application.admin.enabled**=false *# Enable admin features for the application.*

**spring.application.admin.jmx-name**=org.springframework.boot:type=Admin,name=SpringApplication *# JMX name of the application admin MBean.*

*# AUTO-CONFIGURATION*

**spring.autoconfigure.exclude**= *# Auto-configuration classes to exclude.*

*# SPRING CORE*

**spring.beaninfo.ignore**=true *# Skip search of BeanInfo classes.*

*# SPRING CACHE (*[CacheProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cache/CacheProperties.java))

**spring.cache.cache-names**= *# Comma-separated list of cache names to create if supported by the underlying cache manager.*

**spring.cache.caffeine.spec**= *# The spec to use to create caches. Check CaffeineSpec for more details on the spec format.*

**spring.cache.couchbase.expiration**=0 *# Entry expiration in milliseconds. By default the entries never expire.*

**spring.cache.ehcache.config**= *# The location of the configuration file to use to initialize EhCache.*

**spring.cache.guava.spec**= *# The spec to use to create caches. Check CacheBuilderSpec for more details on the spec format.*

**spring.cache.infinispan.config**= *# The location of the configuration file to use to initialize Infinispan.*

**spring.cache.jcache.config**= *# The location of the configuration file to use to initialize the cache manager.*

**spring.cache.jcache.provider**= *# Fully qualified name of the CachingProvider implementation to use to retrieve the JSR-107 compliant cache manager. Only needed if more than one JSR-107 implementation is available on the classpath.*

**spring.cache.type**= *# Cache type, auto-detected according to the environment by default.*

*# SPRING CONFIG - using environment property only (*[ConfigFileApplicationListener](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/context/config/ConfigFileApplicationListener.java))

**spring.config.location**= *# Config file locations.*

**spring.config.name**=application *# Config file name.*

*# HAZELCAST (*[HazelcastProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hazelcast/HazelcastProperties.java))

**spring.hazelcast.config**= *# The location of the configuration file to use to initialize Hazelcast.*

*# PROJECT INFORMATION (*[ProjectInfoProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/info/ProjectInfoProperties.java))

**spring.info.build.location**=classpath:META-INF/build-info.properties *# Location of the generated build- info.properties file.*

**spring.info.git.location**=classpath:git.properties *# Location of the generated git.properties file.*

*# JMX*

**spring.jmx.default-domain**= *# JMX domain name.* **spring.jmx.enabled**=true *# Expose management beans to the JMX domain.* **spring.jmx.server**=mbeanServer *# MBeanServer bean name.*

*# Email (*[MailProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailProperties.java))

**spring.mail.default-encoding**=UTF-8 *# Default MimeMessage encoding.*

**spring.mail.host**= *# SMTP server host. For instance `smtp.example.com`*

**spring.mail.jndi-name**= *# Session JNDI name. When set, takes precedence to others mail settings.*

**spring.mail.password**= *# Login password of the SMTP server.*

**spring.mail.port**= *# SMTP server port.*

**spring.mail.properties.\***= *# Additional JavaMail session properties.*

**spring.mail.protocol**=smtp *# Protocol used by the SMTP server.*

**spring.mail.test-connection**=false *# Test that the mail server is available on startup.*

**spring.mail.username**= *# Login user of the SMTP server.*

*# APPLICATION SETTINGS (*[SpringApplication](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/SpringApplication.java))

**spring.main.banner-mode**=console *# Mode used to display the banner when the application runs.*

**spring.main.sources**= *# Sources (class name, package name or XML resource location) to include in the ApplicationContext.*

**spring.main.web-environment**= *# Run the application in a web environment (auto-detected by default).*

*# FILE ENCODING (*[FileEncodingApplicationListener](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/context/FileEncodingApplicationListener.java))

**spring.mandatory-file-encoding**= *# Expected character encoding the application must use. # INTERNATIONALIZATION (*[MessageSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/MessageSourceAutoConfiguration.java))

**spring.messages.always-use-message-format**=false *# Set whether to always apply the MessageFormat rules, parsing even messages without arguments.*

**spring.messages.basename**=messages *# Comma-separated list of basenames, each following the ResourceBundle convention.*

**spring.messages.cache-seconds**=-1 *# Loaded resource bundle files cache expiration, in seconds. When set to -1, bundles are cached forever.*

**spring.messages.encoding**=UTF-8 *# Message bundles encoding.*

**spring.messages.fallback-to-system-locale**=true *# Set whether to fall back to the system Locale if no files for a specific Locale have been found.*

*# OUTPUT*

**spring.output.ansi.enabled**=detect *# Configure the ANSI output.*

*# PID FILE (*[ApplicationPidFileWriter](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/system/ApplicationPidFileWriter.java))

**spring.pid.fail-on-write-error**= *# Fail if ApplicationPidFileWriter is used but it cannot write the PID file.*

**spring.pid.file**= *# Location of the PID file to write (if ApplicationPidFileWriter is used).*

*# PROFILES*

**spring.profiles.active**= *# Comma-separated list (or list if using YAML) of* [active profiles](#_bookmark453).

**spring.profiles.include**= *# Unconditionally activate the specified comma separated profiles (or list of profiles if using YAML).*

*# SENDGRID (*[SendGridAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/sendgrid/SendGridAutoConfiguration.java))

**spring.sendgrid.api-key**= *# SendGrid api key (alternative to username/password).* **spring.sendgrid.username**= *# SendGrid account username.* **spring.sendgrid.password**= *# SendGrid account password.* **spring.sendgrid.proxy.host**= *# SendGrid proxy host.*

**spring.sendgrid.proxy.port**= *# SendGrid proxy port.*

*# ---------------------------------------- # WEB PROPERTIES*

*# ----------------------------------------*

*# EMBEDDED SERVER CONFIGURATION (*[ServerProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java))

**server.address**= *# Network address to which the server should bind to.* **server.compression.enabled**=false *# If response compression is enabled.* **server.compression.excluded-user-agents**= *# List of user-agents to exclude from compression.* **server.compression.mime-types**=text/html,text/xml,text/plain,text/css,text/javascript,application/ javascript *# Comma-separated list of MIME types that should be compressed.*

**server.compression.min-response-size**=2048 *# Minimum response size that is required for compression to be performed.*

**server.connection-timeout**= *# Time in milliseconds that connectors will wait for another HTTP request before closing the connection. When not set, the connector's container-specific default will be used. Use a value of -1 to indicate no (i.e. infinite) timeout.*

**server.context-parameters.\***= *# Servlet context init parameters. For instance `server.context- parameters.a=alpha`*

**server.context-path**= *# Context path of the application.*

**server.display-name**=application *# Display name of the application.*

**server.max-http-header-size**=0 *# Maximum size in bytes of the HTTP message header.* **server.error.include-stacktrace**=never *# When to include a "stacktrace" attribute.* **server.error.path**=/error *# Path of the error controller.*

**server.error.whitelabel.enabled**=true *# Enable the default error page displayed in browsers in case of a server error.*

**server.jetty.acceptors**= *# Number of acceptor threads to use.*

**server.jetty.max-http-post-size**=0 *# Maximum size in bytes of the HTTP post or put content.*

**server.jetty.selectors**= *# Number of selector threads to use.*

**server.jsp-servlet.class-name**=org.apache.jasper.servlet.JspServlet *# The class name of the JSP servlet.*

**server.jsp-servlet.init-parameters.\***= *# Init parameters used to configure the JSP servlet* **server.jsp-servlet.registered**=true *# Whether or not the JSP servlet is registered* **server.port**=8080 *# Server HTTP port.*

**server.server-header**= *# Value to use for the Server response header (no header is sent if empty)*

**server.servlet-path**=/ *# Path of the main dispatcher servlet.*

**server.use-forward-headers**= *# If X-Forwarded-\* headers should be applied to the HttpRequest.* **server.session.cookie.comment**= *# Comment for the session cookie.* **server.session.cookie.domain**= *# Domain for the session cookie.*

**server.session.cookie.http-only**= *# "HttpOnly" flag for the session cookie.* **server.session.cookie.max-age**= *# Maximum age of the session cookie in seconds.* **server.session.cookie.name**= *# Session cookie name.*

**server.session.cookie.path**= *# Path of the session cookie.*

**server.session.cookie.secure**= *# "Secure" flag for the session cookie.* **server.session.persistent**=false *# Persist session data between restarts.* **server.session.store-dir**= *# Directory used to store session data.* **server.session.timeout**= *# Session timeout in seconds.*

**server.session.tracking-modes**= *# Session tracking modes (one or more of the following: "cookie", "url", "ssl").*

**server.ssl.ciphers**= *# Supported SSL ciphers.*

**server.ssl.client-auth**= *# Whether client authentication is wanted ("want") or needed ("need"). Requires a trust store.*

**server.ssl.enabled**= *# Enable SSL support.*

**server.ssl.enabled-protocols**= *# Enabled SSL protocols.*

**server.ssl.key-alias**= *# Alias that identifies the key in the key store.*

**server.ssl.key-password**= *# Password used to access the key in the key store.*

**server.ssl.key-store**= *# Path to the key store that holds the SSL certificate (typically a jks file).*

**server.ssl.key-store-password**= *# Password used to access the key store.* **server.ssl.key-store-provider**= *# Provider for the key store.* **server.ssl.key-store-type**= *# Type of the key store.* **server.ssl.protocol**=TLS *# SSL protocol to use.*

**server.ssl.trust-store**= *# Trust store that holds SSL certificates.* **server.ssl.trust-store-password**= *# Password used to access the trust store.* **server.ssl.trust-store-provider**= *# Provider for the trust store.* **server.ssl.trust-store-type**= *# Type of the trust store.*

**server.tomcat.accept-count**= *# Maximum queue length for incoming connection requests when all possible request processing threads are in use.*

**server.tomcat.accesslog.buffered**=true *# Buffer output such that it is only flushed periodically.*

**server.tomcat.accesslog.directory**=logs *# Directory in which log files are created. Can be relative to the tomcat base dir or absolute.*

**server.tomcat.accesslog.enabled**=false *# Enable access log.*

**server.tomcat.accesslog.file-date-format**=.yyyy-MM-dd *# Date format to place in log file name.* **server.tomcat.accesslog.pattern**=common *# Format pattern for access logs.* **server.tomcat.accesslog.prefix**=access\_log *# Log file name prefix.*

**server.tomcat.accesslog.rename-on-rotate**=false *# Defer inclusion of the date stamp in the file name until rotate time.*

**server.tomcat.accesslog.request-attributes-enabled**=false *# Set request attributes for IP address, Hostname, protocol and port used for the request.*

**server.tomcat.accesslog.rotate**=true *# Enable access log rotation.*

**server.tomcat.accesslog.suffix**=.log *# Log file name suffix.*

**server.tomcat.additional-tld-skip-patterns**= *# Comma-separated list of additional patterns that match jars to ignore for TLD scanning.*

**server.tomcat.background-processor-delay**=30 *# Delay in seconds between the invocation of backgroundProcess methods.*

**server.tomcat.basedir**= *# Tomcat base directory. If not specified a temporary directory will be used.*

**server.tomcat.internal-proxies**=10\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}|\\ 192\\.168\\.\\d{1,3}\\.\\d{1,3}|\\

169\\.254\\.\\d{1,3}\\.\\d{1,3}|\\

127\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}|\\

172\\.1[6-9]{1}\\.\\d{1,3}\\.\\d{1,3}|\\

172\\.2[0-9]{1}\\.\\d{1,3}\\.\\d{1,3}|\\

172\\.3[0-1]{1}\\.\\d{1,3}\\.\\d{1,3} *# regular expression matching trusted IP addresses.*

**server.tomcat.max-connections**= *# Maximum number of connections that the server will accept and process at any given time.*

**server.tomcat.max-http-post-size**=0 *# Maximum size in bytes of the HTTP post content.*

**server.tomcat.max-threads**=0 *# Maximum amount of worker threads.*

**server.tomcat.min-spare-threads**=0 *# Minimum amount of worker threads.*

**server.tomcat.port-header**=X-Forwarded-Port *# Name of the HTTP header used to override the original port value.*

**server.tomcat.protocol-header**= *# Header that holds the incoming protocol, usually named "X-Forwarded- Proto".*

**server.tomcat.protocol-header-https-value**=https *# Value of the protocol header that indicates that the incoming request uses SSL.*

**server.tomcat.redirect-context-root**= *# Whether requests to the context root should be redirected by appending a / to the path.*

**server.tomcat.remote-ip-header**= *# Name of the http header from which the remote ip is extracted. For instance `X-FORWARDED-FOR`*

**server.tomcat.uri-encoding**=UTF-8 *# Character encoding to use to decode the URI.* **server.undertow.accesslog.dir**= *# Undertow access log directory.* **server.undertow.accesslog.enabled**=false *# Enable access log.* **server.undertow.accesslog.pattern**=common *# Format pattern for access logs.* **server.undertow.accesslog.prefix**=access\_log. *# Log file name prefix.* **server.undertow.accesslog.rotate**=true *# Enable access log rotation.* **server.undertow.accesslog.suffix**=log *# Log file name suffix.*

**server.undertow.buffer-size**= *# Size of each buffer in bytes.* **server.undertow.direct-buffers**= *# Allocate buffers outside the Java heap.* **server.undertow.io-threads**= *# Number of I/O threads to create for the worker.*

**server.undertow.max-http-post-size**=0 *# Maximum size in bytes of the HTTP post content.*

**server.undertow.worker-threads**= *# Number of worker threads.*

*# FREEMARKER (*[FreeMarkerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java))

**spring.freemarker.allow-request-override**=false *# Set whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name.*

**spring.freemarker.allow-session-override**=false *# Set whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name.*

**spring.freemarker.cache**=false *# Enable template caching.*

**spring.freemarker.charset**=UTF-8 *# Template encoding.*

**spring.freemarker.check-template-location**=true *# Check that the templates location exists.* **spring.freemarker.content-type**=text/html *# Content-Type value.* **spring.freemarker.enabled**=true *# Enable MVC view resolution for this technology.*

**spring.freemarker.expose-request-attributes**=false *# Set whether all request attributes should be added to the model prior to merging with the template.*

**spring.freemarker.expose-session-attributes**=false *# Set whether all HttpSession attributes should be added to the model prior to merging with the template.*

**spring.freemarker.expose-spring-macro-helpers**=true *# Set whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext".*

**spring.freemarker.prefer-file-system-access**=true *# Prefer file system access for template loading. File system access enables hot detection of template changes.*

**spring.freemarker.prefix**= *# Prefix that gets prepended to view names when building a URL.* **spring.freemarker.request-context-attribute**= *# Name of the RequestContext attribute for all views.* **spring.freemarker.settings.\***= *# Well-known FreeMarker keys which will be passed to FreeMarker's*

*Configuration.*

**spring.freemarker.suffix**=.ftl *# Suffix that gets appended to view names when building a URL.* **spring.freemarker.template-loader-path**=classpath:/templates/ *# Comma-separated list of template paths.* **spring.freemarker.view-names**= *# White list of view names that can be resolved.*

*# GROOVY TEMPLATES (*[GroovyTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.java))

**spring.groovy.template.allow-request-override**=false *# Set whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name.*

**spring.groovy.template.allow-session-override**=false *# Set whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name.*

**spring.groovy.template.cache**= *# Enable template caching.*

**spring.groovy.template.charset**=UTF-8 *# Template encoding.*

**spring.groovy.template.check-template-location**=true *# Check that the templates location exists.* **spring.groovy.template.configuration.\***= *# See GroovyMarkupConfigurer* **spring.groovy.template.content-type**=test/html *# Content-Type value.* **spring.groovy.template.enabled**=true *# Enable MVC view resolution for this technology.*

**spring.groovy.template.expose-request-attributes**=false *# Set whether all request attributes should be added to the model prior to merging with the template.*

**spring.groovy.template.expose-session-attributes**=false *# Set whether all HttpSession attributes should be added to the model prior to merging with the template.*

**spring.groovy.template.expose-spring-macro-helpers**=true *# Set whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext".*

**spring.groovy.template.prefix**= *# Prefix that gets prepended to view names when building a URL.* **spring.groovy.template.request-context-attribute**= *# Name of the RequestContext attribute for all views.* **spring.groovy.template.resource-loader-path**=classpath:/templates/ *# Template path.* **spring.groovy.template.suffix**=.tpl *# Suffix that gets appended to view names when building a URL.* **spring.groovy.template.view-names**= *# White list of view names that can be resolved.*

*# SPRING HATEOAS (*[HateoasProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hateoas/HateoasProperties.java))

**spring.hateoas.use-hal-as-default-json-media-type**=true *# Specify if application/hal+json responses should be sent to requests that accept application/json.*

*# HTTP message conversion*

**spring.http.converters.preferred-json-mapper**=jackson *# Preferred JSON mapper to use for HTTP message conversion. Set to "gson" to force the use of Gson when both it and Jackson are on the classpath.*

*# HTTP encoding (*[HttpEncodingProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/HttpEncodingProperties.java))

**spring.http.encoding.charset**=UTF-8 *# Charset of HTTP requests and responses. Added to the "Content-Type" header if not set explicitly.*

**spring.http.encoding.enabled**=true *# Enable http encoding support.*

**spring.http.encoding.force**= *# Force the encoding to the configured charset on HTTP requests and responses.*

**spring.http.encoding.force-request**= *# Force the encoding to the configured charset on HTTP requests. Defaults to true when "force" has not been specified.*

**spring.http.encoding.force-response**= *# Force the encoding to the configured charset on HTTP responses.*

**spring.http.encoding.mapping**= *# Locale to Encoding mapping.*

*# MULTIPART (*[MultipartProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/MultipartProperties.java))

**spring.http.multipart.enabled**=true *# Enable support of multi-part uploads.*

**spring.http.multipart.file-size-threshold**=0 *# Threshold after which files will be written to disk. Values can use the suffixed "MB" or "KB" to indicate a Megabyte or Kilobyte size.*

**spring.http.multipart.location**= *# Intermediate location of uploaded files.*

**spring.http.multipart.max-file-size**=1MB *# Max file size. Values can use the suffixed "MB" or "KB" to indicate a Megabyte or Kilobyte size.*

**spring.http.multipart.max-request-size**=10MB *# Max request size. Values can use the suffixed "MB" or "KB" to indicate a Megabyte or Kilobyte size.*

**spring.http.multipart.resolve-lazily**=false *# Whether to resolve the multipart request lazily at the time of file or parameter access.*

*# JACKSON (*[JacksonProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jackson/JacksonProperties.java))

**spring.jackson.date-format**= *# Date format string or a fully-qualified date format class name. For instance `yyyy-MM-dd HH:mm:ss`.*

**spring.jackson.default-property-inclusion**= *# Controls the inclusion of properties during serialization.*

**spring.jackson.deserialization.\***= *# Jackson on/off features that affect the way Java objects are deserialized.*

**spring.jackson.generator.\***= *# Jackson on/off features for generators.*

**spring.jackson.joda-date-time-format**= *# Joda date time format string. If not configured, "date-format" will be used as a fallback if it is configured with a format string.*

**spring.jackson.locale**= *# Locale used for formatting.* **spring.jackson.mapper.\***= *# Jackson general purpose on/off features.* **spring.jackson.parser.\***= *# Jackson on/off features for parsers.*

**spring.jackson.property-naming-strategy**= *# One of the constants on Jackson's PropertyNamingStrategy. Can also be a fully-qualified class name of a PropertyNamingStrategy subclass.*

**spring.jackson.serialization.\***= *# Jackson on/off features that affect the way Java objects are serialized.*

**spring.jackson.time-zone**= *# Time zone used when formatting dates. For instance `America/Los\_Angeles`*

*# JERSEY (*[JerseyProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jersey/JerseyProperties.java))

**spring.jersey.application-path**= *# Path that serves as the base URI for the application. Overrides the value of "@ApplicationPath" if specified.*

**spring.jersey.filter.order**=0 *# Jersey filter chain order.*

**spring.jersey.init.\***= *# Init parameters to pass to Jersey via the servlet or filter.* **spring.jersey.servlet.load-on-startup**=-1 *# Load on startup priority of the Jersey servlet.* **spring.jersey.type**=servlet *# Jersey integration type.*

*# SPRING LDAP (*[LdapProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/LdapProperties.java))

**spring.ldap.urls**= *# LDAP URLs of the server.*

**spring.ldap.base**= *# Base suffix from which all operations should originate.*

**spring.ldap.username**= *# Login user of the server.* **spring.ldap.password**= *# Login password of the server.* **spring.ldap.base-environment.\***= *# LDAP specification settings.*

*# EMBEDDED LDAP (*[EmbeddedLdapProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/embedded/EmbeddedLdapProperties.java))

**spring.ldap.embedded.base-dn**= *# The base DN* **spring.ldap.embedded.credential.username**= *# Embedded LDAP username.* **spring.ldap.embedded.credential.password**= *# Embedded LDAP password.*

**spring.ldap.embedded.ldif**=classpath:schema.ldif *# Schema (LDIF) script resource reference.* **spring.ldap.embedded.port**= *# Embedded LDAP port.* **spring.ldap.embedded.validation.enabled**=true *# Enable LDAP schema validation.* **spring.ldap.embedded.validation.schema**= *# Path to the custom schema.*

*# SPRING MOBILE DEVICE VIEWS (*[DeviceDelegatingViewResolverAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mobile/DeviceDelegatingViewResolverAutoConfiguration.java))

**spring.mobile.devicedelegatingviewresolver.enable-fallback**=false *# Enable support for fallback resolution.*

**spring.mobile.devicedelegatingviewresolver.enabled**=false *# Enable device view resolver.*

**spring.mobile.devicedelegatingviewresolver.mobile-prefix**=mobile/ *# Prefix that gets prepended to view names for mobile devices.*

**spring.mobile.devicedelegatingviewresolver.mobile-suffix**= *# Suffix that gets appended to view names for mobile devices.*

**spring.mobile.devicedelegatingviewresolver.normal-prefix**= *# Prefix that gets prepended to view names for normal devices.*

**spring.mobile.devicedelegatingviewresolver.normal-suffix**= *# Suffix that gets appended to view names for normal devices.*

**spring.mobile.devicedelegatingviewresolver.tablet-prefix**=tablet/ *# Prefix that gets prepended to view names for tablet devices.*

**spring.mobile.devicedelegatingviewresolver.tablet-suffix**= *# Suffix that gets appended to view names for tablet devices.*

*# SPRING MOBILE SITE PREFERENCE (*[SitePreferenceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mobile/SitePreferenceAutoConfiguration.java))

**spring.mobile.sitepreference.enabled**=true *# Enable SitePreferenceHandler.*

*# MUSTACHE TEMPLATES (*[MustacheAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mustache/MustacheAutoConfiguration.java))

**spring.mustache.allow-request-override**= *# Set whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name.*

**spring.mustache.allow-session-override**= *# Set whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name.*

**spring.mustache.cache**= *# Enable template caching.*

**spring.mustache.charset**= *# Template encoding.*

**spring.mustache.check-template-location**= *# Check that the templates location exists.*

**spring.mustache.content-type**= *# Content-Type value.*

**spring.mustache.enabled**= *# Enable MVC view resolution for this technology.*

**spring.mustache.expose-request-attributes**= *# Set whether all request attributes should be added to the model prior to merging with the template.*

**spring.mustache.expose-session-attributes**= *# Set whether all HttpSession attributes should be added to the model prior to merging with the template.*

**spring.mustache.expose-spring-macro-helpers**= *# Set whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext".*

**spring.mustache.prefix**=classpath:/templates/ *# Prefix to apply to template names.* **spring.mustache.request-context-attribute**= *# Name of the RequestContext attribute for all views.* **spring.mustache.suffix**=.html *# Suffix to apply to template names.*

**spring.mustache.view-names**= *# White list of view names that can be resolved.*

*# SPRING MVC (*[WebMvcProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/WebMvcProperties.java))

**spring.mvc.async.request-timeout**= *# Amount of time (in milliseconds) before asynchronous request handling times out.*

**spring.mvc.date-format**= *# Date format to use. For instance `dd/MM/yyyy`.*

**spring.mvc.dispatch-trace-request**=false *# Dispatch TRACE requests to the FrameworkServlet doService method.*

**spring.mvc.dispatch-options-request**=true *# Dispatch OPTIONS requests to the FrameworkServlet doService method.*

**spring.mvc.favicon.enabled**=true *# Enable resolution of favicon.ico.* **spring.mvc.formcontent.putfilter.enabled**=true *# Enable Spring's HttpPutFormContentFilter.* **spring.mvc.ignore-default-model-on-redirect**=true *# If the content of the "default" model should be*

*ignored during redirect scenarios.*

**spring.mvc.locale**= *# Locale to use. By default, this locale is overridden by the "Accept-Language" header.*

**spring.mvc.locale-resolver**=accept-header *# Define how the locale should be resolved.*

**spring.mvc.log-resolved-exception**=false *# Enable warn logging of exceptions resolved by a "HandlerExceptionResolver".*

**spring.mvc.media-types.\***= *# Maps file extensions to media types for content negotiation.*

**spring.mvc.message-codes-resolver-format**= *# Formatting strategy for message codes. For instance*

*`PREFIX\_ERROR\_CODE`.*

**spring.mvc.servlet.load-on-startup**=-1 *# Load on startup priority of the Spring Web Services servlet.*

**spring.mvc.static-path-pattern**=/\*\* *# Path pattern used for static resources.*

**spring.mvc.throw-exception-if-no-handler-found**=false *# If a "NoHandlerFoundException" should be thrown if no Handler was found to process a request.*

**spring.mvc.view.prefix**= *# Spring MVC view prefix.*

**spring.mvc.view.suffix**= *# Spring MVC view suffix.*

*# SPRING RESOURCES HANDLING (*[ResourceProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ResourceProperties.java))

**spring.resources.add-mappings**=true *# Enable default resource handling.*

**spring.resources.cache-period**= *# Cache period for the resources served by the resource handler, in seconds.*

**spring.resources.chain.cache**=true *# Enable caching in the Resource chain.*

**spring.resources.chain.enabled**= *# Enable the Spring Resource Handling chain. Disabled by default unless at least one strategy has been enabled.*

**spring.resources.chain.gzipped**=false *# Enable resolution of already gzipped resources.* **spring.resources.chain.html-application-cache**=false *# Enable HTML5 application cache manifest rewriting.* **spring.resources.chain.strategy.content.enabled**=false *# Enable the content Version Strategy.* **spring.resources.chain.strategy.content.paths**=/\*\* *# Comma-separated list of patterns to apply to the*

*Version Strategy.*

**spring.resources.chain.strategy.fixed.enabled**=false *# Enable the fixed Version Strategy.*

**spring.resources.chain.strategy.fixed.paths**=/\*\* *# Comma-separated list of patterns to apply to the Version Strategy.*

**spring.resources.chain.strategy.fixed.version**= *# Version string to use for the Version Strategy.*

**spring.resources.static-locations**=classpath:/META-INF/resources/,classpath:/resources/,classpath:/ static/,classpath:/public/ *# Locations of static resources.*

*# SPRING SESSION (*[SessionProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/SessionProperties.java))

**spring.session.hazelcast.flush-mode**=on-save *# Sessions flush mode.*

**spring.session.hazelcast.map-name**=spring:session:sessions *# Name of the map used to store sessions.*

**spring.session.jdbc.initializer.enabled**= *# Create the required session tables on startup if necessary. Enabled automatically if the default table name is set or a custom schema is configured.*

[**spring.session.jdbc.schema**=classpath:org/springframework/session/jdbc/schema-@@platform@@.sql](mailto:org/springframework/session/jdbc/schema-@@platform@@.sql) *# Path to the SQL file to use to initialize the database schema.*

**spring.session.jdbc.table-name**=SPRING\_SESSION *# Name of database table used to store sessions.* **spring.session.mongo.collection-name**=sessions *# Collection name used to store sessions.* **spring.session.redis.flush-mode**=on-save *# Sessions flush mode.*

**spring.session.redis.namespace**= *# Namespace for keys used to store sessions.*

**spring.session.store-type**= *# Session store type.*

*# SPRING SOCIAL (*[SocialWebAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/SocialWebAutoConfiguration.java))

**spring.social.auto-connection-views**=false *# Enable the connection status view for supported providers.*

*# SPRING SOCIAL FACEBOOK (*[FacebookAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/FacebookAutoConfiguration.java)) **spring.social.facebook.app-id**= *# your application's Facebook App ID* **spring.social.facebook.app-secret**= *# your application's Facebook App Secret*

*# SPRING SOCIAL LINKEDIN (*[LinkedInAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/LinkedInAutoConfiguration.java)) **spring.social.linkedin.app-id**= *# your application's LinkedIn App ID* **spring.social.linkedin.app-secret**= *# your application's LinkedIn App Secret*

*# SPRING SOCIAL TWITTER (*[TwitterAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/TwitterAutoConfiguration.java)) **spring.social.twitter.app-id**= *# your application's Twitter App ID* **spring.social.twitter.app-secret**= *# your application's Twitter App Secret*

*# THYMELEAF (*[ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java))

**spring.thymeleaf.cache**=true *# Enable template caching.*

**spring.thymeleaf.check-template**=true *# Check that the template exists before rendering it.* **spring.thymeleaf.check-template-location**=true *# Check that the templates location exists.* **spring.thymeleaf.content-type**=text/html *# Content-Type value.* **spring.thymeleaf.enabled**=true *# Enable MVC Thymeleaf view resolution.* **spring.thymeleaf.encoding**=UTF-8 *# Template encoding.*

**spring.thymeleaf.excluded-view-names**= *# Comma-separated list of view names that should be excluded from resolution.*

**spring.thymeleaf.mode**=HTML5 *# Template mode to be applied to templates. See also StandardTemplateModeHandlers.*

**spring.thymeleaf.prefix**=classpath:/templates/ *# Prefix that gets prepended to view names when building a URL.*

**spring.thymeleaf.suffix**=.html *# Suffix that gets appended to view names when building a URL.* **spring.thymeleaf.template-resolver-order**= *# Order of the template resolver in the chain.* **spring.thymeleaf.view-names**= *# Comma-separated list of view names that can be resolved.*

*# SPRING WEB SERVICES (*[WebServicesProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/webservices/WebServicesProperties.java))

**spring.webservices.path**=/services *# Path that serves as the base URI for the services.* **spring.webservices.servlet.init**= *# Servlet init parameters to pass to Spring Web Services.* **spring.webservices.servlet.load-on-startup**=-1 *# Load on startup priority of the Spring Web Services*

*servlet.*

*# ---------------------------------------- # SECURITY PROPERTIES*

*# ----------------------------------------*

*# SECURITY (*[SecurityProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/SecurityProperties.java))

**security.basic.authorize-mode**=role *# Security authorize mode to apply.* **security.basic.enabled**=true *# Enable basic authentication.* **security.basic.path**=/\*\* *# Comma-separated list of paths to secure.* **security.basic.realm**=Spring *# HTTP basic realm name.*

**security.enable-csrf**=false *# Enable Cross Site Request Forgery support.*

**security.filter-order**=0 *# Security filter chain order.*

**security.filter-dispatcher-types**=ASYNC, FORWARD, INCLUDE, REQUEST *# Security filter chain dispatcher types.*

**security.headers.cache**=true *# Enable cache control HTTP headers.* **security.headers.content-security-policy**= *# Value for content security policy header.* **security.headers.content-security-policy-mode**=default *# Content security policy mode.*

**security.headers.content-type**=true *# Enable "X-Content-Type-Options" header.*

**security.headers.frame**=true *# Enable "X-Frame-Options" header.*

**security.headers.hsts**=all *# HTTP Strict Transport Security (HSTS) mode (none, domain, all).*

**security.headers.xss**=true *# Enable cross site scripting (XSS) protection.*

**security.ignored**= *# Comma-separated list of paths to exclude from the default secured paths.* **security.require-ssl**=false *# Enable secure channel for all requests.* **security.sessions**=stateless *# Session creation policy (always, never, if\_required, stateless).* **security.user.name**=user *# Default user name.*

**security.user.password**= *# Password for the default user name. A random password is logged on startup by default.*

**security.user.role**=USER *# Granted roles for the default user name.*

*# SECURITY OAUTH2 CLIENT (*[OAuth2ClientProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/OAuth2ClientProperties.java))

**security.oauth2.client.client-id**= *# OAuth2 client id.*

**security.oauth2.client.client-secret**= *# OAuth2 client secret. A random secret is generated by default*

*# SECURITY OAUTH2 RESOURCES (*[ResourceServerProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/resource/ResourceServerProperties.java))

**security.oauth2.resource.filter-order**= *# The order of the filter chain used to authenticate tokens.*

**security.oauth2.resource.id**= *# Identifier of the resource.*

**security.oauth2.resource.jwt.key-uri**= *# The URI of the JWT token. Can be set if the value is not available and the key is public.*

**security.oauth2.resource.jwt.key-value**= *# The verification key of the JWT token. Can either be a symmetric secret or PEM-encoded RSA public key.*

**security.oauth2.resource.jwk.key-set-uri**= *# The URI for getting the set of keys that can be used to validate the token.*

**security.oauth2.resource.prefer-token-info**=true *# Use the token info, can be set to false to use the user info.*

**security.oauth2.resource.service-id**=resource *#*

**security.oauth2.resource.token-info-uri**= *# URI of the token decoding endpoint.* **security.oauth2.resource.token-type**= *# The token type to send when using the userInfoUri.* **security.oauth2.resource.user-info-uri**= *# URI of the user endpoint.*

*# SECURITY OAUTH2 SSO (*[OAuth2SsoProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/client/OAuth2SsoProperties.java))

**security.oauth2.sso.filter-order**= *# Filter order to apply if not providing an explicit WebSecurityConfigurerAdapter*

**security.oauth2.sso.login-path**=/login *# Path to the login page, i.e. the one that triggers the redirect to the OAuth2 Authorization Server*

*# ---------------------------------------- # DATA PROPERTIES*

*# ----------------------------------------*

*# FLYWAY (*[FlywayProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayProperties.java))

**flyway.baseline-description**= *#*

**flyway.baseline-version**=1 *# version to start migration*

**flyway.baseline-on-migrate**= *#*

**flyway.check-location**=false *# Check that migration scripts location exists.*

**flyway.clean-on-validation-error**= *#* **flyway.enabled**=true *# Enable flyway.* **flyway.encoding**= *#*

**flyway.ignore-failed-future-migration**= *#*

**flyway.init-sqls**= *# SQL statements to execute to initialize a connection immediately after obtaining it.*

**flyway.locations**=classpath:db/migration *# locations of migrations scripts*

**flyway.out-of-order**= *#*

**flyway.password**= *# JDBC password if you want Flyway to create its own DataSource*

**flyway.placeholder-prefix**= *#* **flyway.placeholder-replacement**= *#* **flyway.placeholder-suffix**= *#* **flyway.placeholders.\***= *#* **flyway.schemas**= *# schemas to update* **flyway.sql-migration-prefix**=V *#* **flyway.sql-migration-separator**= *#* **flyway.sql-migration-suffix**=.sql *#* **flyway.table**= *#*

**flyway.url**= *# JDBC url of the database to migrate. If not set, the primary configured data source is used.*

**flyway.user**= *# Login user of the database to migrate.*

**flyway.validate-on-migrate**= *#*

*# LIQUIBASE (*[LiquibaseProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/liquibase/LiquibaseProperties.java))

**liquibase.change-log**=classpath:/db/changelog/db.changelog-master.yaml *# Change log configuration path.*

**liquibase.check-change-log-location**=true *# Check the change log location exists.* **liquibase.contexts**= *# Comma-separated list of runtime contexts to use.* **liquibase.default-schema**= *# Default database schema.*

**liquibase.drop-first**=false *# Drop the database schema first.* **liquibase.enabled**=true *# Enable liquibase support.* **liquibase.labels**= *# Comma-separated list of runtime labels to use.* **liquibase.parameters.\***= *# Change log parameters.* **liquibase.password**= *# Login password of the database to migrate.*

**liquibase.rollback-file**= *# File to which rollback SQL will be written when an update is performed.*

**liquibase.url**= *# JDBC url of the database to migrate. If not set, the primary configured data source is used.*

**liquibase.user**= *# Login user of the database to migrate.*

*# COUCHBASE (*[CouchbaseProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/couchbase/CouchbaseProperties.java))

**spring.couchbase.bootstrap-hosts**= *# Couchbase nodes (host or IP address) to bootstrap from.* **spring.couchbase.bucket.name**=default *# Name of the bucket to connect to.* **spring.couchbase.bucket.password**= *# Password of the bucket.*

**spring.couchbase.env.endpoints.key-value**=1 *# Number of sockets per node against the Key/value service.* **spring.couchbase.env.endpoints.query**=1 *# Number of sockets per node against the Query (N1QL) service.* **spring.couchbase.env.endpoints.view**=1 *# Number of sockets per node against the view service.* **spring.couchbase.env.ssl.enabled**= *# Enable SSL support. Enabled automatically if a "keyStore" is*

*provided unless specified otherwise.*

**spring.couchbase.env.ssl.key-store**= *# Path to the JVM key store that holds the certificates.* **spring.couchbase.env.ssl.key-store-password**= *# Password used to access the key store.* **spring.couchbase.env.timeouts.connect**=5000 *# Bucket connections timeout in milliseconds.* **spring.couchbase.env.timeouts.key-value**=2500 *# Blocking operations performed on a specific key timeout*

*in milliseconds.*

**spring.couchbase.env.timeouts.query**=7500 *# N1QL query operations timeout in milliseconds.* **spring.couchbase.env.timeouts.socket-connect**=1000 *# Socket connect connections timeout in milliseconds.* **spring.couchbase.env.timeouts.view**=7500 *# Regular and geospatial view operations timeout in*

*milliseconds.*

*# DAO (*[PersistenceExceptionTranslationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/dao/PersistenceExceptionTranslationAutoConfiguration.java))

**spring.dao.exceptiontranslation.enabled**=true *# Enable the PersistenceExceptionTranslationPostProcessor.*

*# CASSANDRA (*[CassandraProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cassandra/CassandraProperties.java))

**spring.data.cassandra.cluster-name**= *# Name of the Cassandra cluster.* **spring.data.cassandra.compression**=none *# Compression supported by the Cassandra binary protocol.* **spring.data.cassandra.connect-timeout-millis**= *# Socket option: connection time out.* **spring.data.cassandra.consistency-level**= *# Queries consistency level.* **spring.data.cassandra.contact-points**=localhost *# Comma-separated list of cluster node addresses.* **spring.data.cassandra.fetch-size**= *# Queries default fetch size.*

**spring.data.cassandra.keyspace-name**= *# Keyspace name to use.* **spring.data.cassandra.load-balancing-policy**= *# Class name of the load balancing policy.* **spring.data.cassandra.port**= *# Port of the Cassandra server.* **spring.data.cassandra.password**= *# Login password of the server.* **spring.data.cassandra.read-timeout-millis**= *# Socket option: read time out.* **spring.data.cassandra.reconnection-policy**= *# Reconnection policy class.* **spring.data.cassandra.repositories.enabled**= *# Enable Cassandra repositories.* **spring.data.cassandra.retry-policy**= *# Class name of the retry policy.* **spring.data.cassandra.serial-consistency-level**= *# Queries serial consistency level.* **spring.data.cassandra.schema-action**=none *# Schema action to take at startup.* **spring.data.cassandra.ssl**=false *# Enable SSL support.*

**spring.data.cassandra.username**= *# Login user of the server.*

*# DATA COUCHBASE (*[CouchbaseDataProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseDataProperties.java))

**spring.data.couchbase.auto-index**=false *# Automatically create views and indexes.*

**spring.data.couchbase.consistency**=read-your-own-writes *# Consistency to apply by default on generated queries.*

**spring.data.couchbase.repositories.enabled**=true *# Enable Couchbase repositories.*

*# ELASTICSEARCH (*[ElasticsearchProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchProperties.java))

**spring.data.elasticsearch.cluster-name**=elasticsearch *# Elasticsearch cluster name.*

**spring.data.elasticsearch.cluster-nodes**= *# Comma-separated list of cluster node addresses. If not specified, starts a client node.*

**spring.data.elasticsearch.properties.\***= *# Additional properties used to configure the client.*

**spring.data.elasticsearch.repositories.enabled**=true *# Enable Elasticsearch repositories.*

*# DATA LDAP*

**spring.data.ldap.repositories.enabled**=true *# Enable LDAP repositories.*

*# MONGODB (*[MongoProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/MongoProperties.java))

**spring.data.mongodb.authentication-database**= *# Authentication database name.*

**spring.data.mongodb.database**=test *# Database name.*

**spring.data.mongodb.field-naming-strategy**= *# Fully qualified name of the FieldNamingStrategy to use.*

**spring.data.mongodb.grid-fs-database**= *# GridFS database name.* **spring.data.mongodb.host**=localhost *# Mongo server host. Cannot be set with uri.* **spring.data.mongodb.password**= *# Login password of the mongo server. Cannot be set with uri.* **spring.data.mongodb.port**=27017 *# Mongo server port. Cannot be set with uri.* **spring.data.mongodb.repositories.enabled**=true *# Enable Mongo repositories.*

**spring.data.mongodb.uri**=mongodb://localhost/test *# Mongo database URI. Cannot be set with host, port and credentials.*

**spring.data.mongodb.username**= *# Login user of the mongo server. Cannot be set with uri.*

*# DATA REDIS*

**spring.data.redis.repositories.enabled**=true *# Enable Redis repositories.*

*# NEO4J (*[Neo4jProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/neo4j/Neo4jProperties.java))

**spring.data.neo4j.compiler**= *# Compiler to use.*

**spring.data.neo4j.embedded.enabled**=true *# Enable embedded mode if the embedded driver is available.*

**spring.data.neo4j.open-in-view**=false *# Register OpenSessionInViewInterceptor. Binds a Neo4j Session to the thread for the entire processing of the request.*

**spring.data.neo4j.password**= *# Login password of the server.* **spring.data.neo4j.repositories.enabled**=true *# Enable Neo4j repositories.* **spring.data.neo4j.uri**= *# URI used by the driver. Auto-detected by default.* **spring.data.neo4j.username**= *# Login user of the server.*

*# DATA REST (*[RepositoryRestProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/rest/RepositoryRestProperties.java))

**spring.data.rest.base-path**= *# Base path to be used by Spring Data REST to expose repository resources.*

**spring.data.rest.default-page-size**= *# Default size of pages.*

**spring.data.rest.detection-strategy**=default *# Strategy to use to determine which repositories get exposed.*

**spring.data.rest.enable-enum-translation**= *# Enable enum value translation via the Spring Data REST default resource bundle.*

**spring.data.rest.limit-param-name**= *# Name of the URL query string parameter that indicates how many results to return at once.*

**spring.data.rest.max-page-size**= *# Maximum size of pages.*

**spring.data.rest.page-param-name**= *# Name of the URL query string parameter that indicates what page to return.*

**spring.data.rest.return-body-on-create**= *# Return a response body after creating an entity.* **spring.data.rest.return-body-on-update**= *# Return a response body after updating an entity.* **spring.data.rest.sort-param-name**= *# Name of the URL query string parameter that indicates what direction*

*to sort results.*

*# SOLR (*[SolrProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/solr/SolrProperties.java))

**spring.data.solr.host**=http://127.0.0.1:8983/solr *# Solr host. Ignored if "zk-host" is set.* **spring.data.solr.repositories.enabled**=true *# Enable Solr repositories.* **spring.data.solr.zk-host**= *# ZooKeeper host address in the form HOST:PORT.*

*# DATASOURCE (*[DataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.java) & [DataSourceProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceProperties.java))

**spring.datasource.continue-on-error**=false *# Do not stop if an error occurs while initializing the database.*

**spring.datasource.data**= *# Data (DML) script resource references.*

**spring.datasource.data-username**= *# User of the database to execute DML scripts (if different).* **spring.datasource.data-password**= *# Password of the database to execute DML scripts (if different).* **spring.datasource.dbcp2.\***= *# Commons DBCP2 specific settings*

**spring.datasource.driver-class-name**= *# Fully qualified name of the JDBC driver. Auto-detected based on the URL by default.*

**spring.datasource.generate-unique-name**=false *# Generate a random datasource name.* **spring.datasource.hikari.\***= *# Hikari specific settings* **spring.datasource.initialize**=true *# Populate the database using 'data.sql'.*

**spring.datasource.jmx-enabled**=false *# Enable JMX support (if provided by the underlying pool).*

**spring.datasource.jndi-name**= *# JNDI location of the datasource. Class, url, username & password are ignored when set.*

**spring.datasource.name**=testdb *# Name of the datasource.*

**spring.datasource.password**= *# Login password of the database.*

**spring.datasource.platform**=all *# Platform to use in the DDL or DML scripts (e.g. schema-${platform}.sql or data-${platform}.sql).*

**spring.datasource.schema**= *# Schema (DDL) script resource references.*

**spring.datasource.schema-username**= *# User of the database to execute DDL scripts (if different).*

**spring.datasource.schema-password**= *# Password of the database to execute DDL scripts (if different).*

**spring.datasource.separator**=; *# Statement separator in SQL initialization scripts.* **spring.datasource.sql-script-encoding**= *# SQL scripts encoding.* **spring.datasource.tomcat.\***= *# Tomcat datasource specific settings*

**spring.datasource.type**= *# Fully qualified name of the connection pool implementation to use. By default, it is auto-detected from the classpath.*

**spring.datasource.url**= *# JDBC url of the database.*

**spring.datasource.username**= *# Login user of the database.* **spring.datasource.xa.data-source-class-name**= *# XA datasource fully qualified name.* **spring.datasource.xa.properties**= *# Properties to pass to the XA data source.*

*# JEST (Elasticsearch HTTP client) (*[JestProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/elasticsearch/jest/JestProperties.java))

**spring.elasticsearch.jest.connection-timeout**=3000 *# Connection timeout in milliseconds.*

**spring.elasticsearch.jest.multi-threaded**=true *# Enable connection requests from multiple execution threads.*

**spring.elasticsearch.jest.password**= *# Login password.* **spring.elasticsearch.jest.proxy.host**= *# Proxy host the HTTP client should use.* **spring.elasticsearch.jest.proxy.port**= *# Proxy port the HTTP client should use.* **spring.elasticsearch.jest.read-timeout**=3000 *# Read timeout in milliseconds.*

**spring.elasticsearch.jest.uris**=http://localhost:9200 *# Comma-separated list of the Elasticsearch instances to use.*

**spring.elasticsearch.jest.username**= *# Login user.*

*# H2 Web Console (*[H2ConsoleProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/h2/H2ConsoleProperties.java))

**spring.h2.console.enabled**=false *# Enable the console.*

**spring.h2.console.path**=/h2-console *# Path at which the console will be available.* **spring.h2.console.settings.trace**=false *# Enable trace output.* **spring.h2.console.settings.web-allow-others**=false *# Enable remote access.*

*# JOOQ (*[JooqAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jooq/JooqAutoConfiguration.java))

**spring.jooq.sql-dialect**= *# SQLDialect JOOQ used when communicating with the configured datasource. For instance `POSTGRES`*

*# JPA (*[JpaBaseConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/JpaBaseConfiguration.java), [HibernateJpaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.java))

**spring.data.jpa.repositories.enabled**=true *# Enable JPA repositories.*

**spring.jpa.database**= *# Target database to operate on, auto-detected by default. Can be alternatively set using the "databasePlatform" property.*

**spring.jpa.database-platform**= *# Name of the target database to operate on, auto-detected by default. Can be alternatively set using the "Database" enum.*

**spring.jpa.generate-ddl**=false *# Initialize the schema on startup.*

**spring.jpa.hibernate.ddl-auto**= *# DDL mode. This is actually a shortcut for the "hibernate.hbm2ddl.auto" property. Default to "create-drop" when using an embedded database, "none" otherwise.*

**spring.jpa.hibernate.naming.implicit-strategy**= *# Hibernate 5 implicit naming strategy fully qualified name.*

**spring.jpa.hibernate.naming.physical-strategy**= *# Hibernate 5 physical naming strategy fully qualified name.*

**spring.jpa.hibernate.naming.strategy**= *# Hibernate 4 naming strategy fully qualified name. Not supported with Hibernate 5.*

**spring.jpa.hibernate.use-new-id-generator-mappings**= *# Use Hibernate's newer IdentifierGenerator for AUTO, TABLE and SEQUENCE.*

**spring.jpa.open-in-view**=true *# Register OpenEntityManagerInViewInterceptor. Binds a JPA EntityManager to the thread for the entire processing of the request.*

**spring.jpa.properties.\***= *# Additional native properties to set on the JPA provider.*

**spring.jpa.show-sql**=false *# Enable logging of SQL statements.*

*# JTA (*[JtaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/jta/JtaAutoConfiguration.java)) **spring.jta.enabled**=true *# Enable JTA support.* **spring.jta.log-dir**= *# Transaction logs directory.*

**spring.jta.transaction-manager-id**= *# Transaction manager unique identifier.*

*# ATOMIKOS (*[AtomikosProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jta/atomikos/AtomikosProperties.java))

**spring.jta.atomikos.connectionfactory.borrow-connection-timeout**=30 *# Timeout, in seconds, for borrowing connections from the pool.*

**spring.jta.atomikos.connectionfactory.ignore-session-transacted-flag**=true *# Whether or not to ignore the transacted flag when creating session.*

**spring.jta.atomikos.connectionfactory.local-transaction-mode**=false *# Whether or not local transactions are desired.*

**spring.jta.atomikos.connectionfactory.maintenance-interval**=60 *# The time, in seconds, between runs of the pool's maintenance thread.*

**spring.jta.atomikos.connectionfactory.max-idle-time**=60 *# The time, in seconds, after which connections are cleaned up from the pool.*

**spring.jta.atomikos.connectionfactory.max-lifetime**=0 *# The time, in seconds, that a connection can be pooled for before being destroyed. 0 denotes no limit.*

**spring.jta.atomikos.connectionfactory.max-pool-size**=1 *# The maximum size of the pool.* **spring.jta.atomikos.connectionfactory.min-pool-size**=1 *# The minimum size of the pool.* **spring.jta.atomikos.connectionfactory.reap-timeout**=0 *# The reap timeout, in seconds, for borrowed*

*connections. 0 denotes no limit.*

**spring.jta.atomikos.connectionfactory.unique-resource-name**=jmsConnectionFactory *# The unique name used to identify the resource during recovery.*

**spring.jta.atomikos.datasource.borrow-connection-timeout**=30 *# Timeout, in seconds, for borrowing connections from the pool.*

**spring.jta.atomikos.datasource.default-isolation-level**= *# Default isolation level of connections provided by the pool.*

**spring.jta.atomikos.datasource.login-timeout**= *# Timeout, in seconds, for establishing a database connection.*

**spring.jta.atomikos.datasource.maintenance-interval**=60 *# The time, in seconds, between runs of the pool's maintenance thread.*

**spring.jta.atomikos.datasource.max-idle-time**=60 *# The time, in seconds, after which connections are cleaned up from the pool.*

**spring.jta.atomikos.datasource.max-lifetime**=0 *# The time, in seconds, that a connection can be pooled for before being destroyed. 0 denotes no limit.*

**spring.jta.atomikos.datasource.max-pool-size**=1 *# The maximum size of the pool.* **spring.jta.atomikos.datasource.min-pool-size**=1 *# The minimum size of the pool.* **spring.jta.atomikos.datasource.reap-timeout**=0 *# The reap timeout, in seconds, for borrowed connections.*

*0 denotes no limit.*

**spring.jta.atomikos.datasource.test-query**= *# SQL query or statement used to validate a connection before returning it.*

**spring.jta.atomikos.datasource.unique-resource-name**=dataSource *# The unique name used to identify the resource during recovery.*

**spring.jta.atomikos.properties.checkpoint-interval**=500 *# Interval between checkpoints.* **spring.jta.atomikos.properties.default-jta-timeout**=10000 *# Default timeout for JTA transactions.* **spring.jta.atomikos.properties.enable-logging**=true *# Enable disk logging.* **spring.jta.atomikos.properties.force-shutdown-on-vm-exit**=false *# Specify if a VM shutdown should trigger*

*forced shutdown of the transaction core.*

**spring.jta.atomikos.properties.log-base-dir**= *# Directory in which the log files should be stored.* **spring.jta.atomikos.properties.log-base-name**=tmlog *# Transactions log file base name.* **spring.jta.atomikos.properties.max-actives**=50 *# Maximum number of active transactions.* **spring.jta.atomikos.properties.max-timeout**=300000 *# Maximum timeout (in milliseconds) that can be*

*allowed for transactions.*

**spring.jta.atomikos.properties.serial-jta-transactions**=true *# Specify if sub-transactions should be joined when possible.*

**spring.jta.atomikos.properties.service**= *# Transaction manager implementation that should be started.*

**spring.jta.atomikos.properties.threaded-two-phase-commit**=false *# Use different (and concurrent) threads for two-phase commit on the participating resources.*

**spring.jta.atomikos.properties.transaction-manager-unique-name**= *# Transaction manager's unique name.*

*# BITRONIX*

**spring.jta.bitronix.connectionfactory.acquire-increment**=1 *# Number of connections to create when growing the pool.*

**spring.jta.bitronix.connectionfactory.acquisition-interval**=1 *# Time, in seconds, to wait before trying to acquire a connection again after an invalid connection was acquired.*

**spring.jta.bitronix.connectionfactory.acquisition-timeout**=30 *# Timeout, in seconds, for acquiring connections from the pool.*

**spring.jta.bitronix.connectionfactory.allow-local-transactions**=true *# Whether or not the transaction manager should allow mixing XA and non-XA transactions.*

**spring.jta.bitronix.connectionfactory.apply-transaction-timeout**=false *# Whether or not the transaction timeout should be set on the XAResource when it is enlisted.*

**spring.jta.bitronix.connectionfactory.automatic-enlisting-enabled**=true *# Whether or not resources should be enlisted and delisted automatically.*

**spring.jta.bitronix.connectionfactory.cache-producers-consumers**=true *# Whether or not produces and consumers should be cached.*

**spring.jta.bitronix.connectionfactory.defer-connection-release**=true *# Whether or not the provider can run many transactions on the same connection and supports transaction interleaving.*

**spring.jta.bitronix.connectionfactory.ignore-recovery-failures**=false *# Whether or not recovery failures should be ignored.*

**spring.jta.bitronix.connectionfactory.max-idle-time**=60 *# The time, in seconds, after which connections are cleaned up from the pool.*

**spring.jta.bitronix.connectionfactory.max-pool-size**=10 *# The maximum size of the pool. 0 denotes no limit.*

**spring.jta.bitronix.connectionfactory.min-pool-size**=0 *# The minimum size of the pool.*

**spring.jta.bitronix.connectionfactory.password**= *# The password to use to connect to the JMS provider.*

**spring.jta.bitronix.connectionfactory.share-transaction-connections**=false *# Whether or not connections in the ACCESSIBLE state can be shared within the context of a transaction.*

**spring.jta.bitronix.connectionfactory.test-connections**=true *# Whether or not connections should be tested when acquired from the pool.*

**spring.jta.bitronix.connectionfactory.two-pc-ordering-position**=1 *# The position that this resource should take during two-phase commit (always first is Integer.MIN\_VALUE, always last is Integer.MAX\_VALUE).*

**spring.jta.bitronix.connectionfactory.unique-name**=jmsConnectionFactory *# The unique name used to identify the resource during recovery.*

**spring.jta.bitronix.connectionfactory.use-tm-join**=true Whether or not TMJOIN should be used when starting XAResources.

**spring.jta.bitronix.connectionfactory.user**= *# The user to use to connect to the JMS provider.*

**spring.jta.bitronix.datasource.acquire-increment**=1 *# Number of connections to create when growing the pool.*

**spring.jta.bitronix.datasource.acquisition-interval**=1 *# Time, in seconds, to wait before trying to acquire a connection again after an invalid connection was acquired.*

**spring.jta.bitronix.datasource.acquisition-timeout**=30 *# Timeout, in seconds, for acquiring connections from the pool.*

**spring.jta.bitronix.datasource.allow-local-transactions**=true *# Whether or not the transaction manager should allow mixing XA and non-XA transactions.*

**spring.jta.bitronix.datasource.apply-transaction-timeout**=false *# Whether or not the transaction timeout should be set on the XAResource when it is enlisted.*

**spring.jta.bitronix.datasource.automatic-enlisting-enabled**=true *# Whether or not resources should be enlisted and delisted automatically.*

**spring.jta.bitronix.datasource.cursor-holdability**= *# The default cursor holdability for connections.*

**spring.jta.bitronix.datasource.defer-connection-release**=true *# Whether or not the database can run many transactions on the same connection and supports transaction interleaving.*

**spring.jta.bitronix.datasource.enable-jdbc4-connection-test**= *# Whether or not Connection.isValid() is called when acquiring a connection from the pool.*

**spring.jta.bitronix.datasource.ignore-recovery-failures**=false *# Whether or not recovery failures should be ignored.*

**spring.jta.bitronix.datasource.isolation-level**= *# The default isolation level for connections.* **spring.jta.bitronix.datasource.local-auto-commit**= *# The default auto-commit mode for local transactions.* **spring.jta.bitronix.datasource.login-timeout**= *# Timeout, in seconds, for establishing a database*

*connection.*

**spring.jta.bitronix.datasource.max-idle-time**=60 *# The time, in seconds, after which connections are cleaned up from the pool.*

**spring.jta.bitronix.datasource.max-pool-size**=10 *# The maximum size of the pool. 0 denotes no limit.* **spring.jta.bitronix.datasource.min-pool-size**=0 *# The minimum size of the pool.* **spring.jta.bitronix.datasource.prepared-statement-cache-size**=0 *# The target size of the prepared*

*statement cache. 0 disables the cache.*

**spring.jta.bitronix.datasource.share-transaction-connections**=false *# Whether or not connections in the ACCESSIBLE state can be shared within the context of a transaction.*

**spring.jta.bitronix.datasource.test-query**= *# SQL query or statement used to validate a connection before returning it.*

**spring.jta.bitronix.datasource.two-pc-ordering-position**=1 *# The position that this resource should take during two-phase commit (always first is Integer.MIN\_VALUE, always last is Integer.MAX\_VALUE).*

**spring.jta.bitronix.datasource.unique-name**=dataSource *# The unique name used to identify the resource during recovery.*

**spring.jta.bitronix.datasource.use-tm-join**=true Whether or not TMJOIN should be used when starting XAResources.

**spring.jta.bitronix.properties.allow-multiple-lrc**=false *# Allow multiple LRC resources to be enlisted into the same transaction.*

**spring.jta.bitronix.properties.asynchronous2-pc**=false *# Enable asynchronously execution of two phase commit.*

**spring.jta.bitronix.properties.background-recovery-interval-seconds**=60 *# Interval in seconds at which to run the recovery process in the background.*

**spring.jta.bitronix.properties.current-node-only-recovery**=true *# Recover only the current node.*

**spring.jta.bitronix.properties.debug-zero-resource-transaction**=false *# Log the creation and commit call stacks of transactions executed without a single enlisted resource.*

**spring.jta.bitronix.properties.default-transaction-timeout**=60 *# Default transaction timeout in seconds.* **spring.jta.bitronix.properties.disable-jmx**=false *# Enable JMX support.* **spring.jta.bitronix.properties.exception-analyzer**= *# Set the fully qualified name of the exception*

*analyzer implementation to use.*

**spring.jta.bitronix.properties.filter-log-status**=false *# Enable filtering of logs so that only mandatory logs are written.*

**spring.jta.bitronix.properties.force-batching-enabled**=true *# Set if disk forces are batched.* **spring.jta.bitronix.properties.forced-write-enabled**=true *# Set if logs are forced to disk.* **spring.jta.bitronix.properties.graceful-shutdown-interval**=60 *# Maximum amount of seconds the TM will*

*wait for transactions to get done before aborting them at shutdown time.*

**spring.jta.bitronix.properties.jndi-transaction-synchronization-registry-name**= *# JNDI name of the TransactionSynchronizationRegistry.*

**spring.jta.bitronix.properties.jndi-user-transaction-name**= *# JNDI name of the UserTransaction.*

**spring.jta.bitronix.properties.journal**=disk *# Name of the journal. Can be 'disk', 'null' or a class name.*

**spring.jta.bitronix.properties.log-part1-filename**=btm1.tlog *# Name of the first fragment of the journal.*

**spring.jta.bitronix.properties.log-part2-filename**=btm2.tlog *# Name of the second fragment of the journal.*

**spring.jta.bitronix.properties.max-log-size-in-mb**=2 *# Maximum size in megabytes of the journal fragments.*

**spring.jta.bitronix.properties.resource-configuration-filename**= *# ResourceLoader configuration file name.*

**spring.jta.bitronix.properties.server-id**= *# ASCII ID that must uniquely identify this TM instance. Default to the machine's IP address.*

**spring.jta.bitronix.properties.skip-corrupted-logs**=false *# Skip corrupted transactions log entries.*

**spring.jta.bitronix.properties.warn-about-zero-resource-transaction**=true *# Log a warning for transactions executed without a single enlisted resource.*

*# NARAYANA (*[NarayanaProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot/src/main/java/org/springframework/boot/jta/narayana/NarayanaProperties.java))

**spring.jta.narayana.default-timeout**=60 *# Transaction timeout in seconds.* **spring.jta.narayana.expiry- scanners**=com.arjuna.ats.internal.arjuna.recovery.ExpiredTransactionStatusManagerScanner *# Comma- separated list of expiry scanners.*

**spring.jta.narayana.log-dir**= *# Transaction object store directory.*

**spring.jta.narayana.one-phase-commit**=true *# Enable one phase commit optimisation.*

**spring.jta.narayana.periodic-recovery-period**=120 *# Interval in which periodic recovery scans are performed in seconds.*

**spring.jta.narayana.recovery-backoff-period**=10 *# Back off period between first and second phases of the recovery scan in seconds.*

**spring.jta.narayana.recovery-db-pass**= *# Database password to be used by recovery manager.* **spring.jta.narayana.recovery-db-user**= *# Database username to be used by recovery manager.* **spring.jta.narayana.recovery-jms-pass**= *# JMS password to be used by recovery manager.* **spring.jta.narayana.recovery-jms-user**= *# JMS username to be used by recovery manager.* **spring.jta.narayana.recovery-modules**= *# Comma-separated list of recovery modules.* **spring.jta.narayana.transaction-manager-id**=1 *# Unique transaction manager id.* **spring.jta.narayana.xa-resource-orphan-filters**= *# Comma-separated list of orphan filters.*

*# EMBEDDED MONGODB (*[EmbeddedMongoProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/embedded/EmbeddedMongoProperties.java))

**spring.mongodb.embedded.features**=SYNC\_DELAY *# Comma-separated list of features to enable.* **spring.mongodb.embedded.storage.database-dir**= *# Directory used for data storage.* **spring.mongodb.embedded.storage.oplog-size**= *# Maximum size of the oplog in megabytes.* **spring.mongodb.embedded.storage.repl-set-name**= *# Name of the replica set.* **spring.mongodb.embedded.version**=2.6.10 *# Version of Mongo to use.*

*# REDIS (*[RedisProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisProperties.java))

**spring.redis.cluster.max-redirects**= *# Maximum number of redirects to follow when executing commands across the cluster.*

**spring.redis.cluster.nodes**= *# Comma-separated list of "host:port" pairs to bootstrap from.*

**spring.redis.database**=0 *# Database index used by the connection factory.*

**spring.redis.url**= *# Connection URL, will override host, port and password (user will be ignored), e.g. redis://user:password@example.com:6379*

**spring.redis.host**=localhost *# Redis server host.* **spring.redis.password**= *# Login password of the redis server.* **spring.redis.ssl**=false *# Enable SSL support.*

**spring.redis.pool.max-active**=8 *# Max number of connections that can be allocated by the pool at a given time. Use a negative value for no limit.*

**spring.redis.pool.max-idle**=8 *# Max number of "idle" connections in the pool. Use a negative value to indicate an unlimited number of idle connections.*

**spring.redis.pool.max-wait**=-1 *# Maximum amount of time (in milliseconds) a connection allocation should block before throwing an exception when the pool is exhausted. Use a negative value to block indefinitely.*

**spring.redis.pool.min-idle**=0 *# Target for the minimum number of idle connections to maintain in the pool. This setting only has an effect if it is positive.*

**spring.redis.port**=6379 *# Redis server port.* **spring.redis.sentinel.master**= *# Name of Redis server.* **spring.redis.sentinel.nodes**= *# Comma-separated list of host:port pairs.* **spring.redis.timeout**=0 *# Connection timeout in milliseconds.*

*# TRANSACTION (*[TransactionProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/TransactionProperties.java))

**spring.transaction.default-timeout**= *# Default transaction timeout in seconds.*

**spring.transaction.rollback-on-commit-failure**= *# Perform the rollback on commit failures.*

*# ---------------------------------------- # INTEGRATION PROPERTIES*

*# ----------------------------------------*

*# ACTIVEMQ (*[ActiveMQProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQProperties.java))

**spring.activemq.broker-url**= *# URL of the ActiveMQ broker. Auto-generated by default.*

**spring.activemq.close-timeout**=15000 *# Time to wait, in milliseconds, before considering a close complete.*

**spring.activemq.in-memory**=true *# Specify if the default broker URL should be in memory. Ignored if an explicit broker has been specified.*

**spring.activemq.non-blocking-redelivery**=false *# Do not stop message delivery before re-delivering messages from a rolled back transaction. This implies that message order will not be preserved when this is enabled.*

**spring.activemq.password**= *# Login password of the broker.*

**spring.activemq.send-timeout**=0 *# Time to wait, in milliseconds, on Message sends for a response. Set it to 0 to indicate to wait forever.*

**spring.activemq.user**= *# Login user of the broker.*

**spring.activemq.packages.trust-all**= *# Trust all packages.*

**spring.activemq.packages.trusted**= *# Comma-separated list of specific packages to trust (when not trusting all packages).*

**spring.activemq.pool.block-if-full**=true *# Block when a connection is requested and the pool is full. Set it to false to throw a "JMSException" instead.*

**spring.activemq.pool.block-if-full-timeout**=-1 *# Blocking period, in milliseconds, before throwing an exception if the pool is still full.*

**spring.activemq.pool.create-connection-on-startup**=true *# Create a connection on startup. Can be used to warm-up the pool on startup.*

**spring.activemq.pool.enabled**=false *# Whether a PooledConnectionFactory should be created instead of a regular ConnectionFactory.*

**spring.activemq.pool.expiry-timeout**=0 *# Connection expiration timeout in milliseconds.* **spring.activemq.pool.idle-timeout**=30000 *# Connection idle timeout in milliseconds.* **spring.activemq.pool.max-connections**=1 *# Maximum number of pooled connections.* **spring.activemq.pool.maximum-active-session-per-connection**=500 *# Maximum number of active sessions per*

*connection.*

**spring.activemq.pool.reconnect-on-exception**=true *# Reset the connection when a "JMXException" occurs.*

**spring.activemq.pool.time-between-expiration-check**=-1 *# Time to sleep, in milliseconds, between runs of the idle connection eviction thread. When negative, no idle connection eviction thread runs.*

**spring.activemq.pool.use-anonymous-producers**=true *# Use only one anonymous "MessageProducer" instance. Set it to false to create one "MessageProducer" every time one is required.*

*# ARTEMIS (*[ArtemisProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/artemis/ArtemisProperties.java))

**spring.artemis.embedded.cluster-password**= *# Cluster password. Randomly generated on startup by default.*

**spring.artemis.embedded.data-directory**= *# Journal file directory. Not necessary if persistence is turned off.*

**spring.artemis.embedded.enabled**=true *# Enable embedded mode if the Artemis server APIs are available.*

**spring.artemis.embedded.persistent**=false *# Enable persistent store.* **spring.artemis.embedded.queues**= *# Comma-separated list of queues to create on startup.* **spring.artemis.embedded.server-id**= *# Server id. By default, an auto-incremented counter is used.* **spring.artemis.embedded.topics**= *# Comma-separated list of topics to create on startup.* **spring.artemis.host**=localhost *# Artemis broker host.*

**spring.artemis.mode**= *# Artemis deployment mode, auto-detected by default.* **spring.artemis.password**= *# Login password of the broker.* **spring.artemis.port**=61616 *# Artemis broker port.*

**spring.artemis.user**= *# Login user of the broker.*

*# SPRING BATCH (*[BatchProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/BatchProperties.java))

**spring.batch.initializer.enabled**= *# Create the required batch tables on startup if necessary. Enabled automatically if no custom table prefix is set or if a custom schema is configured.*

**spring.batch.job.enabled**=true *# Execute all Spring Batch jobs in the context on startup.*

**spring.batch.job.names**= *# Comma-separated list of job names to execute on startup (For instance*

*`job1,job2`). By default, all Jobs found in the context are executed.*

[**spring.batch.schema**=classpath:org/springframework/batch/core/schema-@@platform@@.sql](mailto:org/springframework/batch/core/schema-@@platform@@.sql) *# Path to the SQL file to use to initialize the database schema.*

**spring.batch.table-prefix**= *# Table prefix for all the batch meta-data tables.*

*# JMS (*[JmsProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/JmsProperties.java))

**spring.jms.jndi-name**= *# Connection factory JNDI name. When set, takes precedence to others connection factory auto-configurations.*

**spring.jms.listener.acknowledge-mode**= *# Acknowledge mode of the container. By default, the listener is transacted with automatic acknowledgment.*

**spring.jms.listener.auto-startup**=true *# Start the container automatically on startup.* **spring.jms.listener.concurrency**= *# Minimum number of concurrent consumers.* **spring.jms.listener.max-concurrency**= *# Maximum number of concurrent consumers.* **spring.jms.pub-sub-domain**=false *# Specify if the default destination type is topic.*

**spring.jms.template.default-destination**= *# Default destination to use on send/receive operations that do not have a destination parameter.*

**spring.jms.template.delivery-delay**= *# Delivery delay to use for send calls in milliseconds.* **spring.jms.template.delivery-mode**= *# Delivery mode. Enable QoS when set.* **spring.jms.template.priority**= *# Priority of a message when sending. Enable QoS when set.* **spring.jms.template.qos-enabled**= *# Enable explicit QoS when sending a message.* **spring.jms.template.receive-timeout**= *# Timeout to use for receive calls in milliseconds.*

**spring.jms.template.time-to-live**= *# Time-to-live of a message when sending in milliseconds. Enable QoS when set.*

*# APACHE KAFKA (*[KafkaProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/kafka/KafkaProperties.java))

**spring.kafka.bootstrap-servers**= *# Comma-delimited list of host:port pairs to use for establishing the initial connection to the Kafka cluster.*

**spring.kafka.client-id**= *# Id to pass to the server when making requests; used for server-side logging.*

**spring.kafka.consumer.auto-commit-interval**= *# Frequency in milliseconds that the consumer offsets are auto-committed to Kafka if 'enable.auto.commit' true.*

**spring.kafka.consumer.auto-offset-reset**= *# What to do when there is no initial offset in Kafka or if the current offset does not exist any more on the server.*

**spring.kafka.consumer.bootstrap-servers**= *# Comma-delimited list of host:port pairs to use for establishing the initial connection to the Kafka cluster.*

**spring.kafka.consumer.client-id**= *# Id to pass to the server when making requests; used for server-side logging.*

**spring.kafka.consumer.enable-auto-commit**= *# If true the consumer's offset will be periodically committed in the background.*

**spring.kafka.consumer.fetch-max-wait**= *# Maximum amount of time in milliseconds the server will block before answering the fetch request if there isn't sufficient data to immediately satisfy the requirement given by "fetch.min.bytes".*

**spring.kafka.consumer.fetch-min-size**= *# Minimum amount of data the server should return for a fetch request in bytes.*

**spring.kafka.consumer.group-id**= *# Unique string that identifies the consumer group this consumer belongs to.*

**spring.kafka.consumer.heartbeat-interval**= *# Expected time in milliseconds between heartbeats to the consumer coordinator.*

**spring.kafka.consumer.key-deserializer**= *# Deserializer class for keys.*

**spring.kafka.consumer.max-poll-records**= *# Maximum number of records returned in a single call to poll().*

**spring.kafka.consumer.value-deserializer**= *# Deserializer class for values.*

**spring.kafka.listener.ack-count**= *# Number of records between offset commits when ackMode is "COUNT" or "COUNT\_TIME".*

**spring.kafka.listener.ack-mode**= *# Listener AckMode; see the spring-kafka documentation.*

**spring.kafka.listener.ack-time**= *# Time in milliseconds between offset commits when ackMode is "TIME" or "COUNT\_TIME".*

**spring.kafka.listener.concurrency**= *# Number of threads to run in the listener containers.* **spring.kafka.listener.poll-timeout**= *# Timeout in milliseconds to use when polling the consumer.* **spring.kafka.producer.acks**= *# Number of acknowledgments the producer requires the leader to have*

*received before considering a request complete.*

**spring.kafka.producer.batch-size**= *# Number of records to batch before sending.*

**spring.kafka.producer.bootstrap-servers**= *# Comma-delimited list of host:port pairs to use for establishing the initial connection to the Kafka cluster.*

**spring.kafka.producer.buffer-memory**= *# Total bytes of memory the producer can use to buffer records waiting to be sent to the server.*

**spring.kafka.producer.client-id**= *# Id to pass to the server when making requests; used for server-side logging.*

**spring.kafka.producer.compression-type**= *# Compression type for all data generated by the producer.*

**spring.kafka.producer.key-serializer**= *# Serializer class for keys.* **spring.kafka.producer.retries**= *# When greater than zero, enables retrying of failed sends.* **spring.kafka.producer.value-serializer**= *# Serializer class for values.* **spring.kafka.properties.\***= *# Additional properties used to configure the client.* **spring.kafka.ssl.key-password**= *# Password of the private key in the key store file.* **spring.kafka.ssl.keystore-location**= *# Location of the key store file.* **spring.kafka.ssl.keystore-password**= *# Store password for the key store file.* **spring.kafka.ssl.truststore-location**= *# Location of the trust store file.* **spring.kafka.ssl.truststore-password**= *# Store password for the trust store file.* **spring.kafka.template.default-topic**= *# Default topic to which messages will be sent.*

*# RABBIT (*[RabbitProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/amqp/RabbitProperties.java))

**spring.rabbitmq.addresses**= *# Comma-separated list of addresses to which the client should connect.*

**spring.rabbitmq.cache.channel.checkout-timeout**= *# Number of milliseconds to wait to obtain a channel if the cache size has been reached.*

**spring.rabbitmq.cache.channel.size**= *# Number of channels to retain in the cache.* **spring.rabbitmq.cache.connection.mode**=channel *# Connection factory cache mode.* **spring.rabbitmq.cache.connection.size**= *# Number of connections to cache.* **spring.rabbitmq.connection-timeout**= *# Connection timeout, in milliseconds; zero for infinite.* **spring.rabbitmq.dynamic**=true *# Create an AmqpAdmin bean.*

**spring.rabbitmq.host**=localhost *# RabbitMQ host.*

**spring.rabbitmq.listener.simple.acknowledge-mode**= *# Acknowledge mode of container.* **spring.rabbitmq.listener.simple.auto-startup**=true *# Start the container automatically on startup.* **spring.rabbitmq.listener.simple.concurrency**= *# Minimum number of consumers.* **spring.rabbitmq.listener.simple.default-requeue-rejected**= *# Whether or not to requeue delivery failures;*

*default `true`.*

**spring.rabbitmq.listener.simple.idle-event-interval**= *# How often idle container events should be published in milliseconds.*

**spring.rabbitmq.listener.simple.max-concurrency**= *# Maximum number of consumers.*

**spring.rabbitmq.listener.simple.prefetch**= *# Number of messages to be handled in a single request. It should be greater than or equal to the transaction size (if used).*

**spring.rabbitmq.listener.simple.retry.enabled**=false *# Whether or not publishing retries are enabled.*

**spring.rabbitmq.listener.simple.retry.initial-interval**=1000 *# Interval between the first and second attempt to deliver a message.*

**spring.rabbitmq.listener.simple.retry.max-attempts**=3 *# Maximum number of attempts to deliver a message.* **spring.rabbitmq.listener.simple.retry.max-interval**=10000 *# Maximum interval between attempts.* **spring.rabbitmq.listener.simple.retry.multiplier**=1.0 *# A multiplier to apply to the previous delivery*

*retry interval.*

**spring.rabbitmq.listener.simple.retry.stateless**=true *# Whether or not retry is stateless or stateful.*

**spring.rabbitmq.listener.simple.transaction-size**= *# Number of messages to be processed in a transaction. For best results it should be less than or equal to the prefetch count.*

**spring.rabbitmq.password**= *# Login to authenticate against the broker.*

**spring.rabbitmq.port**=5672 *# RabbitMQ port.*

**spring.rabbitmq.publisher-confirms**=false *# Enable publisher confirms.*

**spring.rabbitmq.publisher-returns**=false *# Enable publisher returns.*

**spring.rabbitmq.requested-heartbeat**= *# Requested heartbeat timeout, in seconds; zero for none.*

**spring.rabbitmq.ssl.enabled**=false *# Enable SSL support.*

**spring.rabbitmq.ssl.key-store**= *# Path to the key store that holds the SSL certificate.* **spring.rabbitmq.ssl.key-store-password**= *# Password used to access the key store.* **spring.rabbitmq.ssl.trust-store**= *# Trust store that holds SSL certificates.* **spring.rabbitmq.ssl.trust-store-password**= *# Password used to access the trust store.* **spring.rabbitmq.ssl.algorithm**= *# SSL algorithm to use. By default configure by the rabbit client*

*library.*

**spring.rabbitmq.template.mandatory**=false *# Enable mandatory messages.* **spring.rabbitmq.template.receive-timeout**=0 *# Timeout for `receive()` methods.* **spring.rabbitmq.template.reply-timeout**=5000 *# Timeout for `sendAndReceive()` methods.* **spring.rabbitmq.template.retry.enabled**=false *# Set to true to enable retries in the `RabbitTemplate`.*

**spring.rabbitmq.template.retry.initial-interval**=1000 *# Interval between the first and second attempt to publish a message.*

**spring.rabbitmq.template.retry.max-attempts**=3 *# Maximum number of attempts to publish a message.* **spring.rabbitmq.template.retry.max-interval**=10000 *# Maximum number of attempts to publish a message.* **spring.rabbitmq.template.retry.multiplier**=1.0 *# A multiplier to apply to the previous publishing retry*

*interval.*

**spring.rabbitmq.username**= *# Login user to authenticate to the broker.*

**spring.rabbitmq.virtual-host**= *# Virtual host to use when connecting to the broker.*

*# ---------------------------------------- # ACTUATOR PROPERTIES*

*# ----------------------------------------*

*# ENDPOINTS (*[AbstractEndpoint](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/endpoint/AbstractEndpoint.java) subclasses) **endpoints.enabled**=true *# Enable endpoints.* **endpoints.sensitive**= *# Default endpoint sensitive setting.* **endpoints.actuator.enabled**=true *# Enable the endpoint.* **endpoints.actuator.path**= *# Endpoint URL path.*

**endpoints.actuator.sensitive**=false *# Enable security on the endpoint.* **endpoints.auditevents.enabled**= *# Enable the endpoint.* **endpoints.auditevents.path**= *# Endpoint path.* **endpoints.auditevents.sensitive**=false *# Enable security on the endpoint.* **endpoints.autoconfig.enabled**= *# Enable the endpoint.* **endpoints.autoconfig.id**= *# Endpoint identifier.* **endpoints.autoconfig.path**= *# Endpoint path.*

**endpoints.autoconfig.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.beans.enabled**= *# Enable the endpoint.* **endpoints.beans.id**= *# Endpoint identifier.* **endpoints.beans.path**= *# Endpoint path.*

**endpoints.beans.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.configprops.enabled**= *# Enable the endpoint.*

**endpoints.configprops.id**= *# Endpoint identifier.*

**endpoints.configprops.keys-to-sanitize**=password,secret,key,token,.\*credentials.\*,vcap\_services *# Keys that should be sanitized. Keys can be simple strings that the property ends with or regex expressions.*

**endpoints.configprops.path**= *# Endpoint path.*

**endpoints.configprops.sensitive**= *# Mark if the endpoint exposes sensitive information.* **endpoints.docs.curies.enabled**=false *# Enable the curie generation.* **endpoints.docs.enabled**=true *# Enable actuator docs endpoint.*

**endpoints.docs.path**=/docs *#* **endpoints.docs.sensitive**=false *#* **endpoints.dump.enabled**= *# Enable the endpoint.* **endpoints.dump.id**= *# Endpoint identifier.* **endpoints.dump.path**= *# Endpoint path.*

**endpoints.dump.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.env.enabled**= *# Enable the endpoint.*

**endpoints.env.id**= *# Endpoint identifier.*

**endpoints.env.keys-to-sanitize**=password,secret,key,token,.\*credentials.\*,vcap\_services *# Keys that should be sanitized. Keys can be simple strings that the property ends with or regex expressions.*

**endpoints.env.path**= *# Endpoint path.*

**endpoints.env.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.flyway.enabled**= *# Enable the endpoint.*

**endpoints.flyway.id**= *# Endpoint identifier.*

**endpoints.flyway.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.health.enabled**= *# Enable the endpoint.*

**endpoints.health.id**= *# Endpoint identifier.*

**endpoints.health.mapping.\***= *# Mapping of health statuses to HTTP status codes. By default, registered health statuses map to sensible defaults (i.e. UP maps to 200).*

**endpoints.health.path**= *# Endpoint path.*

**endpoints.health.sensitive**= *# Mark if the endpoint exposes sensitive information.* **endpoints.health.time-to-live**=1000 *# Time to live for cached result, in milliseconds.* **endpoints.heapdump.enabled**= *# Enable the endpoint.*

**endpoints.heapdump.path**= *# Endpoint path.*

**endpoints.heapdump.sensitive**= *# Mark if the endpoint exposes sensitive information.* **endpoints.hypermedia.enabled**=false *# Enable hypermedia support for endpoints.* **endpoints.info.enabled**= *# Enable the endpoint.*

**endpoints.info.id**= *# Endpoint identifier.*

**endpoints.info.path**= *# Endpoint path.*

**endpoints.info.sensitive**= *# Mark if the endpoint exposes sensitive information.* **endpoints.jolokia.enabled**=true *# Enable Jolokia endpoint.* **endpoints.jolokia.path**=/jolokia *# Endpoint URL path.* **endpoints.jolokia.sensitive**=true *# Enable security on the endpoint.* **endpoints.liquibase.enabled**= *# Enable the endpoint.*

**endpoints.liquibase.id**= *# Endpoint identifier.*

**endpoints.liquibase.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.logfile.enabled**=true *# Enable the endpoint.* **endpoints.logfile.external-file**= *# External Logfile to be accessed.* **endpoints.logfile.path**=/logfile *# Endpoint URL path.* **endpoints.logfile.sensitive**=true *# Enable security on the endpoint.* **endpoints.loggers.enabled**=true *# Enable the endpoint.* **endpoints.loggers.id**= *# Endpoint identifier.* **endpoints.loggers.path**=/logfile *# Endpoint path.*

**endpoints.loggers.sensitive**=true *# Mark if the endpoint exposes sensitive information.*

**endpoints.mappings.enabled**= *# Enable the endpoint.* **endpoints.mappings.id**= *# Endpoint identifier.* **endpoints.mappings.path**= *# Endpoint path.*

**endpoints.mappings.sensitive**= *# Mark if the endpoint exposes sensitive information.* **endpoints.metrics.enabled**= *# Enable the endpoint.* **endpoints.metrics.filter.enabled**=true *# Enable the metrics servlet filter.*

**endpoints.metrics.filter.gauge-submissions**=merged *# Http filter gauge submissions (merged, per-http- method)*

**endpoints.metrics.filter.counter-submissions**=merged *# Http filter counter submissions (merged, per-http- method)*

**endpoints.metrics.id**= *# Endpoint identifier.*

**endpoints.metrics.path**= *# Endpoint path.*

**endpoints.metrics.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.shutdown.enabled**= *# Enable the endpoint.*

**endpoints.shutdown.id**= *# Endpoint identifier.*

**endpoints.shutdown.path**= *# Endpoint path.*

**endpoints.shutdown.sensitive**= *# Mark if the endpoint exposes sensitive information.*

**endpoints.trace.enabled**= *# Enable the endpoint.* **endpoints.trace.filter.enabled**=true *# Enable the trace servlet filter.* **endpoints.trace.id**= *# Endpoint identifier.*

**endpoints.trace.path**= *# Endpoint path.*

**endpoints.trace.sensitive**= *# Mark if the endpoint exposes sensitive information.*

*# ENDPOINTS CORS CONFIGURATION (*[EndpointCorsProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/EndpointCorsProperties.java))

**endpoints.cors.allow-credentials**= *# Set whether credentials are supported. When not set, credentials are not supported.*

**endpoints.cors.allowed-headers**= *# Comma-separated list of headers to allow in a request. '\*' allows all headers.*

**endpoints.cors.allowed-methods**=GET *# Comma-separated list of methods to allow. '\*' allows all methods.*

**endpoints.cors.allowed-origins**= *# Comma-separated list of origins to allow. '\*' allows all origins. When not set, CORS support is disabled.*

**endpoints.cors.exposed-headers**= *# Comma-separated list of headers to include in a response.*

**endpoints.cors.max-age**=1800 *# How long, in seconds, the response from a pre-flight request can be cached by clients.*

*# JMX ENDPOINT (*[EndpointMBeanExportProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/EndpointMBeanExportProperties.java))

**endpoints.jmx.domain**= *# JMX domain name. Initialized with the value of 'spring.jmx.default-domain' if set.*

**endpoints.jmx.enabled**=true *# Enable JMX export of all endpoints.*

**endpoints.jmx.static-names**= *# Additional static properties to append to all ObjectNames of MBeans representing Endpoints.*

**endpoints.jmx.unique-names**=false *# Ensure that ObjectNames are modified in case of conflict.*

*# JOLOKIA (*[JolokiaProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/JolokiaProperties.java))

**jolokia.config.\***= *# See Jolokia manual*

*# MANAGEMENT HTTP SERVER (*[ManagementServerProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/ManagementServerProperties.java))

**management.add-application-context-header**=true *# Add the "X-Application-Context" HTTP header in each response.*

**management.address**= *# Network address that the management endpoints should bind to.* **management.context-path**= *# Management endpoint context-path. For instance `/actuator`* **management.cloudfoundry.enabled**= *# Enable extended Cloud Foundry actuator endpoints*

**management.cloudfoundry.skip-ssl-validation**= *# Skip SSL verification for Cloud Foundry actuator endpoint security calls*

**management.port**= *# Management endpoint HTTP port. Uses the same port as the application by default. Configure a different port to use management-specific SSL.*

**management.security.enabled**=true *# Enable security.*

**management.security.roles**=ACTUATOR *# Comma-separated list of roles that can access the management endpoint.*

**management.security.sessions**=stateless *# Session creating policy to use (always, never, if\_required, stateless).*

**management.ssl.ciphers**= *# Supported SSL ciphers. Requires a custom management.port.*

**management.ssl.client-auth**= *# Whether client authentication is wanted ("want") or needed ("need"). Requires a trust store. Requires a custom management.port.*

**management.ssl.enabled**= *# Enable SSL support. Requires a custom management.port.* **management.ssl.enabled-protocols**= *# Enabled SSL protocols. Requires a custom management.port.* **management.ssl.key-alias**= *# Alias that identifies the key in the key store. Requires a custom*

*management.port.*

**management.ssl.key-password**= *# Password used to access the key in the key store. Requires a custom management.port.*

**management.ssl.key-store**= *# Path to the key store that holds the SSL certificate (typically a jks file). Requires a custom management.port.*

**management.ssl.key-store-password**= *# Password used to access the key store. Requires a custom management.port.*

**management.ssl.key-store-provider**= *# Provider for the key store. Requires a custom management.port.* **management.ssl.key-store-type**= *# Type of the key store. Requires a custom management.port.* **management.ssl.protocol**=TLS *# SSL protocol to use. Requires a custom management.port.* **management.ssl.trust-store**= *# Trust store that holds SSL certificates. Requires a custom*

*management.port.*

**management.ssl.trust-store-password**= *# Password used to access the trust store. Requires a custom management.port.*

**management.ssl.trust-store-provider**= *# Provider for the trust store. Requires a custom management.port.*

**management.ssl.trust-store-type**= *# Type of the trust store. Requires a custom management.port. # HEALTH INDICATORS*

**management.health.db.enabled**=true *# Enable database health check.* **management.health.cassandra.enabled**=true *# Enable cassandra health check.* **management.health.couchbase.enabled**=true *# Enable couchbase health check.* **management.health.defaults.enabled**=true *# Enable default health indicators.* **management.health.diskspace.enabled**=true *# Enable disk space health check.* **management.health.diskspace.path**= *# Path used to compute the available disk space.* **management.health.diskspace.threshold**=0 *# Minimum disk space that should be available, in bytes.* **management.health.elasticsearch.enabled**=true *# Enable elasticsearch health check.* **management.health.elasticsearch.indices**= *# Comma-separated index names.* **management.health.elasticsearch.response-timeout**=100 *# The time, in milliseconds, to wait for a response*

*from the cluster.*

**management.health.jms.enabled**=true *# Enable JMS health check.* **management.health.ldap.enabled**=true *# Enable LDAP health check.* **management.health.mail.enabled**=true *# Enable Mail health check.* **management.health.mongo.enabled**=true *# Enable MongoDB health check.* **management.health.rabbit.enabled**=true *# Enable RabbitMQ health check.* **management.health.redis.enabled**=true *# Enable Redis health check.* **management.health.solr.enabled**=true *# Enable Solr health check.*

**management.health.status.order**=DOWN, OUT\_OF\_SERVICE, UP, UNKNOWN *# Comma-separated list of health statuses in order of severity.*

*# INFO CONTRIBUTORS (*[InfoContributorProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/InfoContributorProperties.java)) **management.info.build.enabled**=true *# Enable build info.* **management.info.defaults.enabled**=true *# Enable default info contributors.* **management.info.env.enabled**=true *# Enable environment info.* **management.info.git.enabled**=true *# Enable git info.* **management.info.git.mode**=simple *# Mode to use to expose git information.*

*# REMOTE SHELL (*[ShellProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/ShellProperties.java))

**management.shell.auth.type**=simple *# Authentication type. Auto-detected according to the environment.*

**management.shell.auth.jaas.domain**=my-domain *# JAAS domain.*

**management.shell.auth.key.path**= *# Path to the authentication key. This should point to a valid ".pem" file.*

**management.shell.auth.simple.user.name**=user *# Login user.* **management.shell.auth.simple.user.password**= *# Login password.* **management.shell.auth.spring.roles**=ACTUATOR *# Comma-separated list of required roles to login to the*

*CRaSH console.*

**management.shell.command-path-patterns**=classpath\*:/commands/\*\*,classpath\*:/crash/commands/\*\* *# Patterns to use to look for commands.*

**management.shell.command-refresh-interval**=-1 *# Scan for changes and update the command if necessary (in seconds).*

**management.shell.config-path-patterns**=classpath\*:/crash/\* *# Patterns to use to look for configurations.* **management.shell.disabled-commands**=jpa\*,jdbc\*,jndi\* *# Comma-separated list of commands to disable.* **management.shell.disabled-plugins**= *# Comma-separated list of plugins to disable. Certain plugins are*

*disabled by default based on the environment.*

**management.shell.ssh.auth-timeout** = *# Number of milliseconds after user will be prompted to login again.*

**management.shell.ssh.enabled**=true *# Enable CRaSH SSH support.*

**management.shell.ssh.idle-timeout** = *# Number of milliseconds after which unused connections are closed.*

**management.shell.ssh.key-path**= *# Path to the SSH server key.*

**management.shell.ssh.port**=2000 *# SSH port.*

**management.shell.telnet.enabled**=false *# Enable CRaSH telnet support. Enabled by default if the TelnetPlugin is available.*

**management.shell.telnet.port**=5000 *# Telnet port.*

*# TRACING (*[TraceProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/trace/TraceProperties.java))

**management.trace.include**=request-headers,response-headers,cookies,errors *# Items to be included in the trace.*

*# METRICS EXPORT (*[MetricExportProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/metrics/export/MetricExportProperties.java))

**spring.metrics.export.aggregate.key-pattern**= *# Pattern that tells the aggregator what to do with the keys from the source repository.*

**spring.metrics.export.aggregate.prefix**= *# Prefix for global repository if active.*

**spring.metrics.export.delay-millis**=5000 *# Delay in milliseconds between export ticks. Metrics are exported to external sources on a schedule with this delay.*

**spring.metrics.export.enabled**=true *# Flag to enable metric export (assuming a MetricWriter is available).*

**spring.metrics.export.excludes**= *# List of patterns for metric names to exclude. Applied after the includes.*

**spring.metrics.export.includes**= *# List of patterns for metric names to include.* **spring.metrics.export.redis.key**=keys.spring.metrics *# Key for redis repository export (if active).* **spring.metrics.export.redis.prefix**=spring.metrics *# Prefix for redis repository if active.*

**spring.metrics.export.send-latest**= *# Flag to switch off any available optimizations based on not exporting unchanged metric values.*

**spring.metrics.export.statsd.host**= *# Host of a statsd server to receive exported metrics.* **spring.metrics.export.statsd.port**=8125 *# Port of a statsd server to receive exported metrics.* **spring.metrics.export.statsd.prefix**= *# Prefix for statsd exported metrics.* **spring.metrics.export.triggers.\***= *# Specific trigger properties per MetricWriter bean name.*

*# ---------------------------------------- # DEVTOOLS PROPERTIES*

*# ----------------------------------------*

*# DEVTOOLS (*[DevToolsProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/autoconfigure/DevToolsProperties.java))

**spring.devtools.livereload.enabled**=true *# Enable a livereload.com compatible server.*

**spring.devtools.livereload.port**=35729 *# Server port.*

**spring.devtools.restart.additional-exclude**= *# Additional patterns that should be excluded from triggering a full restart.*

**spring.devtools.restart.additional-paths**= *# Additional paths to watch for changes.*

**spring.devtools.restart.enabled**=true *# Enable automatic restart.*

**spring.devtools.restart.exclude**=META-INF/maven/\*\*,META-INF/resources/\*\*,resources/\*\*,static/\*\*,public/

\*\*,templates/\*\*,\*\*/\*Test.class,\*\*/\*Tests.class,git.properties *# Patterns that should be excluded from triggering a full restart.*

**spring.devtools.restart.poll-interval**=1000 *# Amount of time (in milliseconds) to wait between polling for classpath changes.*

**spring.devtools.restart.quiet-period**=400 *# Amount of quiet time (in milliseconds) required without any classpath changes before a restart is triggered.*

**spring.devtools.restart.trigger-file**= *# Name of a specific file that when changed will trigger the restart check. If not specified any classpath file change will trigger the restart.*

*# REMOTE DEVTOOLS (*[RemoteDevToolsProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/autoconfigure/RemoteDevToolsProperties.java))

**spring.devtools.remote.context-path**=/.~~spring-boot!~ *# Context path used to handle the remote connection.*

**spring.devtools.remote.debug.enabled**=true *# Enable remote debug support.* **spring.devtools.remote.debug.local-port**=8000 *# Local remote debug server port.* **spring.devtools.remote.proxy.host**= *# The host of the proxy to use to connect to the remote application.* **spring.devtools.remote.proxy.port**= *# The port of the proxy to use to connect to the remote application.* **spring.devtools.remote.restart.enabled**=true *# Enable remote restart.*

**spring.devtools.remote.secret**= *# A shared secret required to establish a connection (required to enable remote support).*

**spring.devtools.remote.secret-header-name**=X-AUTH-TOKEN *# HTTP header used to transfer the shared secret.*

*# ---------------------------------------- # TESTING PROPERTIES*

*# ----------------------------------------*

**spring.test.database.replace**=any *# Type of existing DataSource to replace.*

**spring.test.mockmvc.print**=default *# MVC Print option.*

**附录 B. Configuration meta-data**

Spring Boot jars 附带元数据文件，提供所有支持的配置属性的详细信息。 这些文件旨在允许IDE开发人员在用户使用application.properties或application.yml文件时提供上下文帮助和“代码补全”。

大部分元数据文件是在编译时自动生成的，处理所有用@ConfigurationProperties注解的项目。 但是，可以手动编写部分([write part of the meta-data](#_bookmark588) [manually](#_bookmark588) )元数据用于角落案例或更高级的用例。

## Meta-data format

配置元数据文件位于META-INF/spring-configuration- metadata.json下的jar文件中。它们使用简单的JSON格式，其中的项目分为“groups” or “properties”，附加值提示归入“hints”

{***"groups"***: **[**

**{**

***"name"***: ***"server"*,**

***"type"***: ***"org.springframework.boot.autoconfigure.web.ServerProperties"*, *"sourceType"***: ***"org.springframework.boot.autoconfigure.web.ServerProperties"***

**},**

**{**

***"name"***: ***"spring.jpa.hibernate"*,**

***"type"***: ***"org.springframework.boot.autoconfigure.orm.jpa.JpaProperties$Hibernate"*, *"sourceType"***: ***"org.springframework.boot.autoconfigure.orm.jpa.JpaProperties"*, *"sourceMethod"***: ***"getHibernate()"***

**}**

...

],***"properties"***: **[**

**{**

***"name"***: ***"server.port"*, *"type"***: ***"java.lang.Integer"*,**

***"sourceType"***: ***"org.springframework.boot.autoconfigure.web.ServerProperties"***

**},**

**{**

***"name"***: ***"server.servlet-path"*, *"type"***: ***"java.lang.String"*,**

***"sourceType"***: ***"org.springframework.boot.autoconfigure.web.ServerProperties"*, *"defaultValue"***: ***"/"***

**},**

**{**

***"name"***: ***"spring.jpa.hibernate.ddl-auto"*, *"type"***: ***"java.lang.String"*,**

***"description"***: ***"DDL mode. This is actually a shortcut for the \"hibernate.hbm2ddl.auto\" property."*,**

***"sourceType"***: ***"org.springframework.boot.autoconfigure.orm.jpa.JpaProperties$Hibernate"***

**}**

...

],***"hints"***: **[**

**{**

***"name"***: ***"spring.jpa.hibernate.ddl-auto"*, *"values"***: **[**

**{**

***"value"***: ***"none"*,**

***"description"***: ***"Disable DDL handling."***

**},**

**{**

***"value"***: ***"validate"*,**

***"description"***: ***"Validate the schema, make no changes to the database."***

**},**

**{**

***"value"***: ***"update"*,**

***"description"***: ***"Update the schema if necessary."***

**},**

**{**

***"value"***: ***"create"*,**

***"description"***: ***"Create the schema and destroy previous data."***

**},**

**{**

***"value"***: ***"create-drop"*,**

***"description"***: ***"Create and then destroy the schema at the end of the session."***

**}**

**]**

**}**

]**}**

每个“property”是用户用给定值指定的配置项目。 例如，server.port和server.servlet-path可以在application.properties中指定，如下所示：

**server.port**=9090 **server.servlet-path**=/home

“groups”是较高级别的项目，本身并不指定值，而是为属性提供上下文分组。 例如，server.port和server.servlet-path属性是服务器组的一部分。

**Note**

并不要求每一个“property”都有一个“group”，有些properties可能就存在于自己的权限之中。

最后，“hints”是用于帮助用户配置给定属性的附加信息。 当配置spring.jpa.hibernate.ddl-auto属性时，一个工具可以使用它来为none，validate，update，create和create-drop值提供一些自动完成的帮助。

### Group Attributes

包含在groups数组中的JSON对象可以包含以下属性：

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| name type | String String | The full name of the group. This attribute is mandatory.  The class name of the data type of the group. For example, if the group was based on a class annotated with  @ConfigurationProperties the attribute would contain the fully qualified name of that class. If it was based on a @Bean method, it would be the return type of that method. The attribute may be omitted if the type is not known. |
| description | String | A short description of the group that can be displayed to users. May be omitted if no description is available. It is recommended that descriptions are a short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.). |
| sourceType | String | The class name of the source that contributed this group. For example, if the group was based on a @Bean method annotated |

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| sourceMetho | d String | with @ConfigurationProperties this attribute would contain the fully qualified name of the @Configuration class containing the method. The attribute may be omitted if the source type is not known.  The full name of the method (include parenthesis and argument types) that contributed this group. For example, the name of a @ConfigurationProperties annotated @Bean method. May be omitted if the source method is not known. |

### Property Attributes

包含在properties数组中的JSON对象可以包含以下属性：

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| name | String | The full name of the property. Names are in lowercase dashed form (e.g. server.servlet-path). This attribute is mandatory. |
| type | String | The full signature of the data type of the property. For example, java.lang.String but also a full generic type such as java.util.Map<java.util.String,acme.MyEnum>. This attribute can be used to guide the user as to the types of values that they can enter. For consistency, the type of a primitive is specified using its wrapper counterpart, i.e. boolean becomes java.lang.Boolean. Note that this class may be a complex type that gets converted from a String as values are bound. May be omitted if the type is not known. |
| description  sourceType | String  String | A short description of the group that can be displayed to users. May be omitted if no description is available. It is recommended that descriptions are a short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.).  The class name of the source that contributed this property.  For example, if the property was from a class annotated with @ConfigurationProperties this attribute would contain the fully qualified name of that class. May be omitted if the source type is not known. |
| defaultValu | e Object | The default value which will be used if the property is not specified. Can also be an array of value(s) if the type of the property is an array. May be omitted if the default value is not known. |
| deprecation | Deprecation | Specify if the property is deprecated. May be omitted if the field is not deprecated or if that information is not known. See below for more details. |

包含在每个属性元素的deprecation属性中的JSON对象可以包含以下属性：

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| level  reason | String  String | The level of deprecation, can be either warning (default) or error. When a property has a warning deprecation level it should still be bound in the environment. When it has an error deprecation level however, the property is no longer managed and will not be bound.  A short description of the reason why the property was deprecated. May be omitted if no reason is available. It is recommended that descriptions are a short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.). |
| replacement | String | The full name of the property that is *replacing* this deprecated property. May be omitted if there is no replacement for this property. |

**Note**

在Spring Boot 1.3之前，可以使用一个废弃的布尔属性来代替deprecation元素。 这仍被支持，不应再使用。 如果没有理由和替换，则应该设置一个空的弃用对象。

也可以在代码中声明性地指定弃用，方法是将@DeprecatedConfigurationProperty注释添加到暴露不推荐属性的getter中。 例如，假设app.foo.target属性很混乱，并被重命名为app.foo.name

@ConfigurationProperties("app.foo")

**public class** FooProperties {

**private** String name;

**public** String getName() { ... }

**public void** setName(String name) { ... }

@DeprecatedConfigurationProperty(replacement = "app.foo.name") @Deprecated

**public** String getTarget() {

**return** getName();

}

@Deprecated

**public void** setTarget(String target) { setName(target);

}

}

**Note**

由于代码仍在处理属性，因此无法设置级别，因为总是假设警告。

上面的代码确保不推荐使用的属性仍然有效（委托给后台的name属性）。 一旦getTarget和setTarget方法可以从公共API中删除，元数据中的自动弃用提示也将消失。 如果要保留提示，添加具有错误弃用级别的手动元数据可确保用户仍被通知该属性，并且在提供替换时特别有用。

### Hint Attributes

包含在hints数组中的JSON对象可以包含以下属性：

|  |  |  |
| --- | --- | --- |
| **Name**  name | **Type**  String | **Purpose**  The full name of the property that this hint refers to. Names are in lowercase dashed form (e.g. server.servlet-path). If the property refers to a map (e.g. system.contexts) the hint either applies to the *keys* of the map (system.context.keys) or the  values (system.context.values). This attribute is mandatory. |
| values  providers | ValueHint[]  ValueProvider[] | A list of valid values as defined by the ValueHint object (see below). Each entry defines the value and may have a description  A list of providers as defined by the ValueProvider object (see below). Each entry defines the name of the provider and its parameters, if any. |

包含在每个提示元素的values属性中的JSON对象可以包含以下属性：

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| value | Object | A valid value for the element to which the hint refers to. Can also be an array of value(s) if the type of the property is an array. This attribute is mandatory. |
| description | String | A short description of the value that can be displayed to users. May be omitted if no description is available. It is recommended that descriptions are a short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.). |

包含在每个提示元素的提供者属性中的JSON对象可以包含以下属性：

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| name  parameters | String  JSON object | The name of the provider to use to offer additional content assistance for the element to which the hint refers to.  Any additional parameter that the provider supports (check the documentation of the provider for more details). |

### Repeated meta-data items

具有相同名称的“property” and “group”对象在元数据文件内多次出现是完全可以接受的。 例如，您可以将两个单独的类绑定到相同的前缀，每个类都可能提供属性名称的重叠。 虽然这不是一个常见的情况，元数据的消费者应该注意确保他们支持这样的场景。

## Providing manual hints

为了改善用户体验并进一步帮助用户配置给定属性，可以提供以下元数据：

1. 描述一个property的潜在价值清单。
2. 将提供者关联到一个属性上，使得一个工具可以根据项目的上下文发现潜在值的列表。

### Value hint

每个提示的名称属性指的是一个属性的名称。 在上面的例子中，我们为spring.jpa.hibernate.ddl-auto属性提供了5个值：none，validate，update，create和create-drop。 每个值也可以有一个描述。

如果您的属性是Map类型，则可以为keys and the values提供提示（但不能为地图本身提供）。 必须使用特殊的.keys and .values后缀分别表示keys and the values。

让我们假设一个foo.contexts将magic字符串值映射到一个整数：

@ConfigurationProperties("foo")

**public class** FooProperties {

**private** Map<String,Integer> contexts;

*// getters and setters*

}

magic值是foo和bar。 为了为keys提供额外的内容帮助，您可以将以下内容添加到[the manual meta-data of the module](#_bookmark588)中：

{***"hints"***: **[**

**{**

***"name"***: ***"foo.contexts.keys"*, *"values"***: **[**

**{**

***"value"***: ***"foo"***

**},**

**{**

***"value"***: ***"bar"***

**}**

**]**

**}**

]**}**

**Note**

当然，你应该有一个枚举这两个值来代替。 如果您的IDE支持，这是迄今为止最有效的自动完成方法。

### Value provider

提供者是将语义附加到属性的有效方式。 我们在下面的部分定义了官方提供商，您可以使用自己的提示。 但是请记住，你最喜欢的IDE可能会实现其中的一些，或者没有一个。 它最终也可以提供它自己的。

**Note**

由于这是一项新功能，IDE 供应商将不得不赶上这个新功能

有了这个新功能。 下表总结了支持的提供者列表

| **Name** | **Description** |
| --- | --- |
| any | Permit any additional value to be provided. |
| class-reference | Auto-complete the classes available in the project. Usually constrained by a base class that is specified via the target parameter. |
| handle-as | Handle the property as if it was defined by the type defined via the mandatory target parameter. |
| logger-name | Auto-complete valid logger names. Typically, package and class names available in the current project can be auto-completed. |
| spring-bean-reference | Auto-complete the available bean names in the current project. Usually constrained by a base class that is specified via the targetparameter. |
| spring-profile-name | Auto-complete the available Spring profile names in the project. |

**Tip**

对于给定的属性，只能有一个提供者处于活动状态，但是如果可以通过某种方式来管理属性，则可以指定多个提供者。 确保首先放置最强大的提供者，因为IDE必须使用它可以处理的JSON部分中的第一个提供者。 如果不支持给定属性的提供者，则也不提供特别的内容帮助。

#### Any

任何提供者允许提供任何额外的值。 如果支持，则应应用基于属性类型的常规值验证。

如果您有一个值列表，那么通常会使用此提供程序，并且任何额外的值仍将被视为有效。

以下示例提供了on和off作为system.state的自动完成值; 任何其他值也是允许的：

{***"hints"***: **[**

**{**

***"name"***: ***"system.state"*, *"values"***: **[**

**{**

***"value"***: ***"on"***

**},**

**{**

***"value"***: ***"off"***

**}**

]**,**

***"providers"***: **[**

**{**

***"name"***: ***"any"***

**}**

**]**

**}**

]**}**

#### Class reference

类引用提供程序自动完成项目中可用的类。 这个提供者支持这些参数：

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Default value Description** |
| target  concrete | String  (Class)  boolean | *none* The fully qualified name of the class that should be assignable to the chosen value. Typically used to filter out non candidate classes. Note that this information can be provided by the type itself by exposing a class with the appropriate upper bound.  true Specify if only concrete classes are to be considered as valid candidates. |

下面的元数据片段对应于标准的server.jsp-servlet.class-name

属性，该属性定义要使用的JspServlet类名称：

{***"hints"***: **[**

**{**

***"name"***: ***"server.jsp-servlet.class-name"*, *"providers"***: **[**

**{**

***"name"***: ***"class-reference"*, *"parameters"***: **{**

***"target"***: ***"javax.servlet.http.HttpServlet"***

**}**

**}**

**]**

**}**

]**}**

#### Handle As

**handle-as**提供程序允许您将属性的类型替换为更高级的类型。 当属性具有java.lang.String类型时，通常会发生这种情况，因为您不希望配置类依赖可能不在类路径中的类。 这个提供者支持这些参数：

|  |  |  |
| --- | --- | --- |
| **Parameter**  **target** | **Type**  String  (Class) | **Default value Description**  *none* The fully qualified name of the type to consider for the property. This parameter is mandatory. |

可以使用以下类型：

* 任何列出属性可能值的java.lang.Enum（通过所有方法，尝试使用Enum类型定义属性，而不要求IDE自动完成值的进一步提示）。
* java.nio.charset.Charset: auto-completion of charset/encoding values (e.g. UTF-8)
* java.util.Locale: auto-completion of locales (e.g. en\_US)
* org.springframework.util.MimeType: auto-completion of content type values (e.g. text/ plain)
* org.springframework.core.io.Resource: auto-completion of Spring’s Resource abstraction to refer to a file on the filesystem or on the classpath. (e.g. classpath:/foo.properties)

**Note**

如果可以提供多个值，请使用集合或数组类型向IDE教授(teach)。

下面的元数据片段对应于标准liquibase.change-log属性，该属性定义要使用的更改日志的路径。 它实际上是作为org.springframework.core.io.Resource在内部使用的，但是不能被公开，因为我们需要保留原来的String值来传递给Liquibase API。

{***"hints"***: **[**

**{**

***"name"***: ***"liquibase.change-log"*, *"providers"***: **[**

**{**

***"name"***: ***"handle-as"*, *"parameters"***: **{**

***"target"***: ***"org.springframework.core.io.Resource"***

**}**

**}**

**]**

**}**

]**}**

#### Logger name

**logger-name**提供程序会自动完成有效的logger names。 通常，当前项目中可用的包和类名称可以自动完成。 具体的框架可能会有额外的magic logger names，也可以支持。

由于logger name可以是任意名称，实际上，此提供程序应该允许任何值，但可以突出显示项目类路径中不可用的有效包和类名称。

下面的元数据片段对应于标准的logging.level属性，keys是*logger names*和values对应于标准日志级别或任何自定义级别：

{***"hints"***: **[**

**{**

***"name"***: ***"logging.level.keys"*, *"values"***: **[**

**{**

***"value"***: ***"root"*,**

***"description"***: ***"Root logger used to assign the default logging level."***

**}**

]**,**

***"providers"***: **[**

**{**

***"name"***: ***"logger-name"***

**}**

**]**

**},**

**{**

***"name"***: ***"logging.level.values"*, *"values"***: **[**

**{**

***"value"***: ***"trace"***

**},**

**{**

***"value"***: ***"debug"***

**},**

**{**

***"value"***: ***"info"***

**},**

**{**

***"value"***: ***"warn"***

**},**

**{**

***"value"***: ***"error"***

**},**

**{**

***"value"***: ***"fatal"***

**},**

**{**

***"value"***: ***"off"***

**}**

]**,**

***"providers"***: **[**

**{**

***"name"***: ***"any"***

**}**

**]**

**}**

]**}**

#### Spring bean reference

spring-bean-reference提供程序自动完成在当前项目的配置中定义的bean。 这个提供者支持这些参数：

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Default value Description** |
| target | String  (Class) | *none* The fully qualified name of the bean class that should be assignable to the candidate. Typically used to filter out non candidate beans. |

下面的元数据片段对应于定义要使用的MBeanServer bean名称的标准spring.jmx.server属性：

{***"hints"***: **[**

**{**

***"name"***: ***"spring.jmx.server"*, *"providers"***: **[**

**{**

***"name"***: ***"spring-bean-reference"*, *"parameters"***: **{**

***"target"***: ***"javax.management.MBeanServer"***

**}**

**}**

**]**

**}**

]**}**

**Note**

binder不知道元数据，所以如果你提供了这个提示，你仍然需要使用ApplicationContext将bean的名字转换成实际的Bean引用。

#### Spring profile name

spring-profile-name提供程序自动完成在当前项目的配置中定义的Spring配置文件。

下面的元数据片段对应于标准spring.profiles.active属性，该属性定义了要启用的Spring配置文件的名称：

{***"hints"***: **[**

**{**

***"name"***: ***"spring.profiles.active"*, *"providers"***: **[**

**{**

***"name"***: ***"spring-profile-name"***

**}**

**]**

**}**

]**}**

## Generating your own meta-data using the annotation processor

通过使用spring-boot-configuration-processor jar，您可以从使用@ConfigurationProperties注释的项目轻松生成自己的配置元数据文件。 该jar包含一个Java annotation processor，在您的项目被编译时被调用。 要使用处理器，只需将spring-boot-configuration-processor作为可选的依赖项包含在Maven中，例如添加：

**<dependency>**

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

**<artifactId>**spring-boot-configuration-processor**</artifactId>**

**<optional>**true**</optional>**

**</dependency>**

使用Gradle，你可以使用[propdeps-plugin](https://github.com/spring-gradle-plugins/propdeps-plugin) 并指定：

dependencies {

optional ***"org.springframework.boot:spring-boot-configuration-processor"***

}

compileJava.dependsOn(processResources)

**Note**

您需要将compileJava.dependsOn（processResources）添加到您的构建中，以确保在编译代码之前处理资源。 没有这个指令，额外的spring-configuration-metadata.json文件将不会被处理。

处理器将选择用@ConfigurationProperties注解的类和方法。 配置类中的字段值的Javadoc将被用来填充description属性。

**Note**

您应该只使用带有@ConfigurationProperties字段的简单文本Javadoc，因为它们在被添加到JSON之前不会被处理。

属性是通过标准的getter和setter来存在的，对集合类型有特殊的处理（即使只有一个getter也会被检测到）。 annotation processor还支持使用@Data，@Getter和@Setter lombok注解。

**Note**

如果您在项目中使用AspectJ，则需要确保annotation processor只运行一次。 有几种方法可以做到这一点：使用Maven，您可以明确地配置maven-apt- plugin，并且只在这里添加依赖annotation processor。 您也可以让AspectJ插件运行maven-compiler-plugin配置中的所有处理并禁用注解处理：

**<plugin>**

**<groupId>**org.apache.maven.plugins**</groupId>**

**<artifactId>**maven-compiler-plugin**</artifactId>**

**<configuration>**

**<proc>**none**</proc>**

**</configuration>**

**</plugin>**

### Nested properties

annotation processor 将自动将内部类视为嵌套属性。 例如，下面的类：

@ConfigurationProperties(prefix="server")

**public class** ServerProperties { **private** String name; **private** Host host;

*// ... getter and setters*

**private static class** Host { **private** String ip; **private int** port;

*// ... getter and setters*

}

}

将产生server.name，server.host.ip和server.host.port属性的元数据信息。 你可以在一个字段上使用@NestedConfigurationProperty注解来表示一个普通（非内部）的类应该像嵌套一样对待。

**Tip**

这对collections and maps没有任何影响，因为这些类型是自动识别的，并且为它们中的每一个生成单个元数据属性。

### Adding additional meta-data

Spring Boot的配置文件处理非常灵活; 而且通常情况下属性可能存在不绑定到ConfigurationProperties bean。 您可能还需要调整现有key的一些属性。 为了支持这种情况，并允许您提供自定义的“hints”， annotation processor将自动地将来自META-INF / additional-spring-configuration-metadata.json的项目合并到主元数据文件中。

如果您引用自动检测到的属性，则会在指定时重写说明，默认值和弃用信息。 如果在当前模块中没有识别手动属性声明，则将其添加为新的属性。

additional-spring-configuration-metadata.json文件的格式与常规的spring-configuration-metadata.json完全相同。 额外的属性文件是可选的，如果你没有任何额外的属性，只是不要添加它。

**附录 C. Auto-configuration classes**

以下是Spring Boot提供的所有自动配置类的列表，包含文档和源代码的链接。 请记住在应用程序中查看autoconfig报告，了解哪些功能可以打开。 （使用--debug或-Ddebug启动应用程序，或者在Actuator应用程序中使用自动配置端点）。

## From the “spring-boot-autoconfigure” module

以下自动配置类来自spring-boot-autoconfigure模块：

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [ActiveMQAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQAutoConfiguration.java)  [AopAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/aop/AopAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/aop/AopAutoConfiguration.html) |
| [ArtemisAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/artemis/ArtemisAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jms/artemis/ArtemisAutoConfiguration.html) |
| [BatchAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/BatchAutoConfiguration.java)  [CacheAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cache/CacheAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/batch/BatchAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/cache/CacheAutoConfiguration.html) |
| [CassandraAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cassandra/CassandraAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/cassandra/CassandraAutoConfiguration.html) |
| [CassandraDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/cassandra/CassandraDataAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/cassandra/CassandraDataAutoConfiguration.html) |
| [CassandraRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/cassandra/CassandraRepositoriesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/cassandra/CassandraRepositoriesAutoConfiguration.html) |
| [CloudAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cloud/CloudAutoConfiguration.java)  [ConfigurationPropertiesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/ConfigurationPropertiesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/cloud/CloudAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/context/ConfigurationPropertiesAutoConfiguration.html) |
| [CouchbaseAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/couchbase/CouchbaseAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/couchbase/CouchbaseAutoConfiguration.html) |
| [CouchbaseDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseDataAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseDataAutoConfiguration.html) |
| [CouchbaseRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseRepositoriesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseRepositoriesAutoConfiguration.html) |
| [DataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.html) |
| [DataSourceTransactionManagerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceTransactionManagerAutoConfiguration.java)  [DeviceDelegatingViewResolverAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mobile/DeviceDelegatingViewResolverAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/DataSourceTransactionManagerAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mobile/DeviceDelegatingViewResolverAutoConfiguration.html) |
| [DeviceResolverAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mobile/DeviceResolverAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mobile/DeviceResolverAutoConfiguration.html) |
| [DispatcherServletAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/DispatcherServletAutoConfiguration.java)  [ElasticsearchAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/DispatcherServletAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchAutoConfiguration.html) |
| [ElasticsearchDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchDataAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchDataAutoConfiguration.html) |
| [ElasticsearchRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchRepositoriesAutoConfiguration.java)  [EmbeddedLdapAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/embedded/EmbeddedLdapAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchRepositoriesAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/ldap/embedded/EmbeddedLdapAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [EmbeddedMongoAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/embedded/EmbeddedMongoAutoConfiguration.java)  [EmbeddedServletContainerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/EmbeddedServletContainerAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mongo/embedded/EmbeddedMongoAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/EmbeddedServletContainerAutoConfiguration.html) |
| [ErrorMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration.java)  [FacebookAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/FacebookAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/social/FacebookAutoConfiguration.html) |
| [FallbackWebSecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/FallbackWebSecurityAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/security/FallbackWebSecurityAutoConfiguration.html) |
| [FlywayAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayAutoConfiguration.java)  [FreeMarkerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/flyway/FlywayAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.html) |
| [GroovyTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.java)  [GsonAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/gson/GsonAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/gson/GsonAutoConfiguration.html) |
| [H2ConsoleAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/h2/H2ConsoleAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/h2/H2ConsoleAutoConfiguration.html) |
| [HazelcastAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hazelcast/HazelcastAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/hazelcast/HazelcastAutoConfiguration.html) |
| [HazelcastJpaDependencyAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hazelcast/HazelcastJpaDependencyAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/hazelcast/HazelcastJpaDependencyAutoConfiguration.html) |
| [HibernateJpaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.java)  [HttpEncodingAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/HttpEncodingAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/HttpEncodingAutoConfiguration.html) |
| [HttpMessageConvertersAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/HttpMessageConvertersAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/HttpMessageConvertersAutoConfiguration.html) |
| [HypermediaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hateoas/HypermediaAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/hateoas/HypermediaAutoConfiguration.html) |
| [IntegrationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/integration/IntegrationAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/integration/IntegrationAutoConfiguration.html) |
| [JacksonAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jackson/JacksonAutoConfiguration.java)  [JdbcTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/JdbcTemplateAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jackson/JacksonAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/JdbcTemplateAutoConfiguration.html) |
| [JerseyAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jersey/JerseyAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jersey/JerseyAutoConfiguration.html) |
| [JestAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/elasticsearch/jest/JestAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/elasticsearch/jest/JestAutoConfiguration.html) |
| [JmsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/JmsAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jms/JmsAutoConfiguration.html) |
| [JmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jmx/JmxAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jmx/JmxAutoConfiguration.html) |
| [JndiConnectionFactoryAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/JndiConnectionFactoryAutoConfiguration.java)  [JndiDataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/JndiDataSourceAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jms/JndiConnectionFactoryAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/JndiDataSourceAutoConfiguration.html) |
| [JooqAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jooq/JooqAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jooq/JooqAutoConfiguration.html) |
| [JpaRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/jpa/JpaRepositoriesAutoConfiguration.java)  [JtaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/jta/JtaAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/jpa/JpaRepositoriesAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/transaction/jta/JtaAutoConfiguration.html) |
| [KafkaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/kafka/KafkaAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/kafka/KafkaAutoConfiguration.html) |
| [LdapAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/LdapAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/ldap/LdapAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [LdapDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/ldap/LdapDataAutoConfiguration.java)  [LdapRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/ldap/LdapRepositoriesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/ldap/LdapDataAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/ldap/LdapRepositoriesAutoConfiguration.html) |
| [LinkedInAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/LinkedInAutoConfiguration.java)  [LiquibaseAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/liquibase/LiquibaseAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/social/LinkedInAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/liquibase/LiquibaseAutoConfiguration.html) |
| [MailSenderAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailSenderAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mail/MailSenderAutoConfiguration.html) |
| [MailSenderValidatorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailSenderValidatorAutoConfiguration.java)  [MessageSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/MessageSourceAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mail/MailSenderValidatorAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/context/MessageSourceAutoConfiguration.html) |
| [MongoAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/MongoAutoConfiguration.java)  [MongoDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/mongo/MongoDataAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mongo/MongoAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/mongo/MongoDataAutoConfiguration.html) |
| [MongoRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/mongo/MongoRepositoriesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/mongo/MongoRepositoriesAutoConfiguration.html) |
| [MultipartAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/MultipartAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/MultipartAutoConfiguration.html) |
| [MustacheAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mustache/MustacheAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mustache/MustacheAutoConfiguration.html) |
| [Neo4jDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/neo4j/Neo4jDataAutoConfiguration.java)  [Neo4jRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/neo4j/Neo4jRepositoriesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/neo4j/Neo4jDataAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/neo4j/Neo4jRepositoriesAutoConfiguration.html) |
| [OAuth2AutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/OAuth2AutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/security/oauth2/OAuth2AutoConfiguration.html) |
| [PersistenceExceptionTranslationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/dao/PersistenceExceptionTranslationAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/dao/PersistenceExceptionTranslationAutoConfiguration.html) |
| [ProjectInfoAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/info/ProjectInfoAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/info/ProjectInfoAutoConfiguration.html) |
| [PropertyPlaceholderAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/PropertyPlaceholderAutoConfiguration.java)  [RabbitAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/amqp/RabbitAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/context/PropertyPlaceholderAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/amqp/RabbitAutoConfiguration.html) |
| [ReactorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/reactor/ReactorAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/reactor/ReactorAutoConfiguration.html) |
| [RedisAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/redis/RedisAutoConfiguration.html) |
| [RedisRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisRepositoriesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/redis/RedisRepositoriesAutoConfiguration.html) |
| [RepositoryRestMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/rest/RepositoryRestMvcAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/rest/RepositoryRestMvcAutoConfiguration.html) |
| [SecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/SecurityAutoConfiguration.java)  [SecurityFilterAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/SecurityFilterAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/security/SecurityAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/security/SecurityFilterAutoConfiguration.html) |
| [SendGridAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/sendgrid/SendGridAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/sendgrid/SendGridAutoConfiguration.html) |
| [ServerPropertiesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerPropertiesAutoConfiguration.java)  [SessionAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/SessionAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/ServerPropertiesAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/session/SessionAutoConfiguration.html) |
| [SitePreferenceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mobile/SitePreferenceAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/mobile/SitePreferenceAutoConfiguration.html) |
| [SocialWebAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/SocialWebAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/social/SocialWebAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [SolrAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/solr/SolrAutoConfiguration.java)  [SolrRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/solr/SolrRepositoriesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/solr/SolrAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/solr/SolrRepositoriesAutoConfiguration.html) |
| [SpringApplicationAdminJmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/admin/SpringApplicationAdminJmxAutoConfiguration.java)  [SpringDataWebAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/web/SpringDataWebAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/admin/SpringApplicationAdminJmxAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/data/web/SpringDataWebAutoConfiguration.html) |
| [ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.html) |
| [TransactionAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/TransactionAutoConfiguration.java)  [TwitterAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/social/TwitterAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/transaction/TransactionAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/social/TwitterAutoConfiguration.html) |
| [ValidationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/validation/ValidationAutoConfiguration.java)  [WebClientAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/WebClientAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/validation/ValidationAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/WebClientAutoConfiguration.html) |
| [WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration.html) |
| [WebServicesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/webservices/WebServicesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/webservices/WebServicesAutoConfiguration.html) |
| [WebSocketAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/websocket/WebSocketAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/websocket/WebSocketAutoConfiguration.html) |
| [WebSocketMessagingAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/websocket/WebSocketMessagingAutoConfiguration.java)  [XADataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/XADataSourceAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/websocket/WebSocketMessagingAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/XADataSourceAutoConfiguration.html) |

## From the “spring-boot-actuator” module

以下自动配置类来自spring-boot-actuator模块：

|  |  |
| --- | --- |
| **Configuration Class**  [AuditAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/AuditAutoConfiguration.java) | **Links**  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/AuditAutoConfiguration.html) |
| [CacheStatisticsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/CacheStatisticsAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/CacheStatisticsAutoConfiguration.html) |
| [CloudFoundryActuatorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/cloudfoundry/CloudFoundryActuatorAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/cloudfoundry/CloudFoundryActuatorAutoConfiguration.html) |
| [CrshAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/CrshAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/CrshAutoConfiguration.html) |
| [EndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/EndpointAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/EndpointAutoConfiguration.html) |
| [EndpointMBeanExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/EndpointMBeanExportAutoConfiguration.java)  [EndpointWebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/EndpointWebMvcAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/EndpointMBeanExportAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/EndpointWebMvcAutoConfiguration.html) |
| [HealthIndicatorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/HealthIndicatorAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/HealthIndicatorAutoConfiguration.html) |
| [InfoContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/InfoContributorAutoConfiguration.java)  [JolokiaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/JolokiaAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/InfoContributorAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/JolokiaAutoConfiguration.html) |
| [ManagementServerPropertiesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/ManagementServerPropertiesAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/ManagementServerPropertiesAutoConfiguration.html) |
| [ManagementWebSecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/ManagementWebSecurityAutoConfiguration.java)  [MetricExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/MetricExportAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/ManagementWebSecurityAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/MetricExportAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [MetricFilterAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/MetricFilterAutoConfiguration.java)  [MetricRepositoryAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/MetricRepositoryAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/MetricFilterAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/MetricRepositoryAutoConfiguration.html) |
| [MetricsChannelAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/MetricsChannelAutoConfiguration.java)  [MetricsDropwizardAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/MetricsDropwizardAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/MetricsChannelAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/MetricsDropwizardAutoConfiguration.html) |
| [PublicMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/PublicMetricsAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/PublicMetricsAutoConfiguration.html) |
| [TraceRepositoryAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/TraceRepositoryAutoConfiguration.java)  [TraceWebFilterAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v1.5.8.RELEASE/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/autoconfigure/TraceWebFilterAutoConfiguration.java) | [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/TraceRepositoryAutoConfiguration.html)  [javadoc](http://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/api/org/springframework/boot/actuate/autoconfigure/TraceWebFilterAutoConfiguration.html) |

**附录 D. Test auto-configuration annotations**

以下是各种@…Test注解的表格，可用于测试应用程序的切片以及默认导入的auto-configuration：

| **Test slice** | **Imported auto-configuration** |
| --- | --- |
| @DataJpaTest | org.springframework.boot.autoconfigure.cache.CacheAutoConfigurationorg.springframework.boot.autoconfigure.data.jpa.JpaRepositoriesAutoConfigurationorg.springframework.boot.autoconfigure.flyway.FlywayAutoConfigurationorg.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfigurationorg.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfigurationorg.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfigurationorg.springframework.boot.autoconfigure.liquibase.LiquibaseAutoConfigurationorg.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfigurationorg.springframework.boot.autoconfigure.transaction.TransactionAutoConfigurationorg.springframework.boot.test.autoconfigure.jdbc.TestDatabaseAutoConfigurationorg.springframework.boot.test.autoconfigure.orm.jpa.TestEntityManagerAutoConfiguration |
| @DataMongoTest | org.springframework.boot.autoconfigure.cache.CacheAutoConfigurationorg.springframework.boot.autoconfigure.data.mongo.MongoDataAutoConfigurationorg.springframework.boot.autoconfigure.data.mongo.MongoRepositoriesAutoConfigurationorg.springframework.boot.autoconfigure.mongo.MongoAutoConfigurationorg.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongoAutoConfiguration |
| @JdbcTest | org.springframework.boot.autoconfigure.cache.CacheAutoConfigurationorg.springframework.boot.autoconfigure.flyway.FlywayAutoConfigurationorg.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfigurationorg.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfigurationorg.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfigurationorg.springframework.boot.autoconfigure.liquibase.LiquibaseAutoConfigurationorg.springframework.boot.autoconfigure.transaction.TransactionAutoConfigurationorg.springframework.boot.test.autoconfigure.jdbc.TestDatabaseAutoConfiguration |
| @JsonTest | org.springframework.boot.autoconfigure.cache.CacheAutoConfigurationorg.springframework.boot.autoconfigure.gson.GsonAutoConfigurationorg.springframework.boot.autoconfigure.jackson.JacksonAutoConfigurationorg.springframework.boot.test.autoconfigure.json.JsonTestersAutoConfiguration |
| @RestClientTest | org.springframework.boot.autoconfigure.cache.CacheAutoConfigurationorg.springframework.boot.autoconfigure.gson.GsonAutoConfigurationorg.springframework.boot.autoconfigure.jackson.JacksonAutoConfigurationorg.springframework.boot.autoconfigure.web.HttpMessageConvertersAutoConfigurationorg.springframework.boot.autoconfigure.web.WebClientAutoConfigurationorg.springframework.boot.test.autoconfigure.web.client.MockRestServiceServerAutoConfigurationorg.springframework.boot.test.autoconfigure.web.client.WebClientRestTemplateAutoConfiguration |
| @WebMvcTest | org.springframework.boot.autoconfigure.cache.CacheAutoConfigurationorg.springframework.boot.autoconfigure.context.MessageSourceAutoConfigurationorg.springframework.boot.autoconfigure.freemarker.FreeMarkerAutoConfigurationorg.springframework.boot.autoconfigure.groovy.template.GroovyTemplateAutoConfigurationorg.springframework.boot.autoconfigure.gson.GsonAutoConfigurationorg.springframework.boot.autoconfigure.hateoas.HypermediaAutoConfigurationorg.springframework.boot.autoconfigure.jackson.JacksonAutoConfigurationorg.springframework.boot.autoconfigure.mustache.MustacheAutoConfigurationorg.springframework.boot.autoconfigure.thymeleaf.ThymeleafAutoConfigurationorg.springframework.boot.autoconfigure.validation.ValidationAutoConfigurationorg.springframework.boot.autoconfigure.web.ErrorMvcAutoConfigurationorg.springframework.boot.autoconfigure.web.HttpMessageConvertersAutoConfigurationorg.springframework.boot.autoconfigure.web.ServerPropertiesAutoConfigurationorg.springframework.boot.autoconfigure.web.WebMvcAutoConfigurationorg.springframework.boot.test.autoconfigure.web.servlet.MockMvcAutoConfigurationorg.springframework.boot.test.autoconfigure.web.servlet.MockMvcSecurityAutoConfigurationorg.springframework.boot.test.autoconfigure.web.servlet.MockMvcWebClientAutoConfigurationorg.springframework.boot.test.autoconfigure.web.servlet.MockMvcWebDriverAutoConfiguration |

**附录 E. The executable jar format**

Spring-Boot-loader模块允许Spring Boot支持可执行的jar和war文件。 如果您使用的是Maven或Gradle插件，可执行文件会自动生成，您通常不需要知道它们的工作细节。

如果您需要从不同的构建系统创建可执行的JAR，或者您只是对底层技术感兴趣，本节提供一些背景知识。

## Nested JARs

Java不提供任何标准的方法来加载嵌套的jar文件（即jar文件本身包含在jar中）。 如果您想分发一个self-contained的应用程序，您可以从命令行运行而不需要解包，这可能会有问题。

为了解决这个问题，许多开发者使用“shaded” jars。 一个shaded的jar简单地把所有的类从所有的jar包装成一个“uber jar”。 带shaded的jars的问题在于，很难在应用程序中看到实际使用的库。 如果在多个jar中使用相同的文件名（但是具有不同的内容），则也可能是有问题的。 Spring Boot采用了不同的方法，可以直接嵌入jars。

### The executable jar file structure

Spring Boot Loader兼容jar文件应该按照以下方式构建：

example.jar

|

+-META-INF

| +-MANIFEST.MF

+-org

| +-springframework

| +-boot

| +-loader

| +-<spring boot loader classes>

+-BOOT-INF

+-classes

| +-mycompany

| +-project

| +-YourClasses.class

+-lib

+-dependency1.jar

+-dependency2.jar

应用程序类应放置在嵌套的BOOT-INF/classes目录中。 依赖关系应该放在一个嵌套的BOOT-INF/lib目录中。

### The executable war file structure

Spring Boot Loader兼容的war文件应该按以下方式构建：

example.war

|

+-META-INF

| +-MANIFEST.MF

+-org

| +-springframework

| +-boot

| +-loader

| +-<spring boot loader classes>

+-WEB-INF

+-classes

| +-com

| +-mycompany

| +-project

| +-YourClasses.class

+-lib

| +-dependency1.jar

| +-dependency2.jar

+-lib-provided

+-servlet-api.jar

+-dependency3.jar

依赖关系应放置在嵌套的WEB-INF / lib目录中。 运行嵌入式时所需的任何依赖关系，但在部署到传统Web容器时不需要，则应该放在WEB-INF / lib提供的位置。

## Spring Boot’s “JarFile” class

用于支持加载嵌套jar的核心类是org.springframework.boot.loader.jar.JarFile。 它允许您从标准的jar文件或嵌套的子jar数据中加载jar内容。 第一次加载时，每个JarEntry的位置映射到外部jar的物理文件偏移量：

myapp.jar

+-------------------+-------------------------+

| /BOOT-INF/classes | /BOOT-INF/lib/mylib.jar |

|+-----------------+||+-----------+----------+|

|| A.class ||| B.class | C.class ||

|+-----------------+||+-----------+----------+|

+-------------------+-------------------------+

^ ^ ^

0063 3452 3980

上面的例子显示了如何在myapp.jar的0063位置的/ BOOT-INF / classes中找到A.class。嵌套jar中的B.class实际上可以在myapp.jar中找到位置3452，而C.class位于3980。

有了这些信息，我们可以通过简单地寻找外部jar的适当部分来加载特定的嵌套条目。 我们不需要解压档案，也不需要把所有的录入数据读入内存。

### Compatibility with the standard Java “JarFile”

Spring Boot Loader努力保持与现有的代码和库兼容。 org.springframework.boot.loader.jar.JarFile从java.util.jar.JarFile扩展，应该作为一个插入替换。 getURL（）方法将返回一个URL，该URL打开一个与java.net.JarURLConnection兼容的连接，并可以与Java的URLClassLoader一起使用。

## Launching executable jars

org.springframework.boot.loader.Launcher类是一个特殊的引导类，用作一个可执行的jars主入口点。 它是jar文件中的实际Main-Class，用于设置合适的URLClassLoader并最终调用main（）方法。

有3个启动子类（JarLauncher，WarLauncher和PropertiesLauncher）。 他们的目的是从目录中嵌入的jar文件或war文件加载资源（.class文件等）（而不是显式地在类路径上）。 在JarLauncher和WarLauncher的情况下，嵌套的路径是固定的。 JarLauncher在BOOT-INF / lib /中查找，而WarLauncher在WEB-INF / lib /和WEB-INF / lib-provided /中查找，因此如果需要更多，只需在这些位置添加额外的jar。 在默认情况下，PropertiesLauncher会在应用程序存档中的BOOT-INF / lib /中查找，但是您可以通过在loader.properties中设置环境变量LOADER\_PATH或loader.path来添加其他位置（逗号分隔的目录，存档或目录的列表 archives）。

### Launcher manifest

您需要指定一个适当的启动程序作为META-INF / MANIFEST.MF的Main-Class属性。 应该在Start-Class属性中指定要启动的实际类（即，您编写的包含主方法的类）。

例如，以下是可执行jar文件的一个典型MANIFEST.MF：

Main-Class: org.springframework.boot.loader.JarLauncher Start-Class: com.mycompany.project.MyApplication

对于war file，这将是：

Main-Class: org.springframework.boot.loader.WarLauncher Start-Class: com.mycompany.project.MyApplication

**Note**

您不需要在清单文件中指定Class-Path条目，类路径将从嵌套的jar中推导出来。

### Exploded archives

$ unzip -q myapp.jar

$ java org.springframework.boot.loader.JarLauncher

某些PaaS实现可能会选择在运行之前解压缩archives文件。 例如，Cloud Foundry以这种方式运作。 您可以通过简单地启动适当的启动程序运行一个解压缩的archives：

## PropertiesLauncher Features

PropertiesLauncher具有一些可以使用外部属性（系统属性，环境变量，清单条目或loader.properties）启用的特殊功能。

**Note**

PropertiesLauncher支持从loader.properties和（由于历史原因）application.properties加载属性。 我们建议仅使用loader.properties，因为对application.properties的支持已被弃用，将来可能会被删除。

|  |  |
| --- | --- |
| **Key** | **Purpose** |
| loader.path  loader.home | Comma-separated Classpath, e.g. lib,  ${HOME}/app/lib. Earlier entries take precedence, just like a regular -classpath on the javac command line.  Used to resolve relative paths in  loader.path. E.g. loader.path=lib then  ${loader.home}/lib is a classpath location (along with all jar files in that directory). Also used to locate a loader.properties file.  Example /opt/app (defaults to ${user.dir}). |
| loader.args | Default arguments for the main method (space |
|  | separated) |
| loader.main | Name of main class to launch, e.g. |
|  | com.app.Application. |
| loader.config.name | Name of properties file, e.g. launcher (defaults to loader). |
| loader.config.location | Path to properties file, e.g. classpath:loader.properties (defaults to loader.properties). |
| loader.system | Boolean flag to indicate that all properties should be added to System properties (defaults to false) |

当指定为环境变量或清单条目时，应使用以下名称：

|  |  |  |
| --- | --- | --- |
| **Key** | **Manifest entry** | **Environment variable** |
| loader.path  loader.home | Loader-Path  Loader-Home | LOADER\_PATH  LOADER\_HOME |
| loader.args | Loader-Args | LOADER\_ARGS |
| loader.main  loader.config.location | Start-Class  Loader-Config-Location | LOADER\_MAIN  LOADER\_CONFIG\_LOCATION |
| loader.system | Loader-System | LOADER\_SYSTEM |

**Tip**

构建fat jar时，构建插件会自动将Main-Class属性移动到Start-Class。 如果您正在使用它，请使用Main-Class属性指定要启动的类的名称，并省略Start-Class。

* 在loader.properties中搜索 loader.properties,然后在类路径的根目中搜索loader.properties,然后在classpath:/BOOT-INF/classes中。存在的第一个位置被使用
* loader.home is only the directory location of an additional properties file (overriding the default) as long as loader.config.location is not specified.
* loader.path can contain directories (scanned recursively for jar and zip files), archive paths, a directory within an archive that is scanned for jar files (for example, dependencies.jar!/lib), or wildcard patterns (for the default JVM behavior). Archive paths can be relative to loader.home, or anywhere in the file system with a jar:file: prefix.
* loader.path (if empty) defaults to BOOT-INF/lib (meaning a local directory or a nested one if running from an archive). Because of this PropertiesLauncher behaves the same as JarLauncher when no additional configuration is provided.
* loader.path can not be used to configure the location of loader.properties (the classpath used to search for the latter is the JVM classpath when PropertiesLauncher is launched).
* Placeholder replacement is done from System and environment variables plus the properties file itself on all values before use.
* The search order for properties (where it makes sense to look in more than one place) is env vars, system properties, loader.properties, exploded archive manifest, archive manifest.

## Executable jar restrictions

在使用Spring Boot Loader打包的应用程序时，需要考虑许多限制。

### Zip entry compression

必须使用ZipEntry.STORED方法保存嵌套jar的ZipEntry。 这是必需的，以便我们可以直接查找嵌套jar中的单个内容。 嵌套的jar文件本身的内容仍然可以被压缩，外部jar中的任何其他条目也可以被压缩。

### System ClassLoader

启动的应用程序应该在加载类时使用Thread.getContextClassLoader（）（大多数库和框架默认会这样做）。 尝试通过ClassLoader.getSystemClassLoader（）加载嵌套的jar类将失败。 请注意，java.util.Logging总是使用系统类加载器，因此您应该考虑使用不同的日志记录实现。

## Alternative single jar solutions

如果上面的限制意味着你不能使用Spring Boot Loader，可以考虑下面的选择：

* [Maven Shade Plugin](http://maven.apache.org/plugins/maven-shade-plugin/)
* [JarClassLoader](http://www.jdotsoft.com/JarClassLoader.php)
* [OneJar](http://one-jar.sourceforge.net/)

**附录 F. Dependency versions**

The table below provides details of all of the dependency versions that are provided by Spring Boot in its CLI, Maven dependency management and Gradle plugin. When you declare a dependency on one of these artifacts without declaring a version the version that is listed in the table will be used.

| **Group ID** | **Artifact ID** | **Version** |
| --- | --- | --- |
| antlr | antlr | 2.7.7 |
| ch.qos.logback | logback-access | 1.1.11 |
| ch.qos.logback | logback-classic | 1.1.11 |
| ch.qos.logback | logback-core | 1.1.11 |
| com.atomikos | transactions-jdbc | 3.9.3 |
| com.atomikos | transactions-jms | 3.9.3 |
| com.atomikos | transactions-jta | 3.9.3 |
| com.couchbase.client | couchbase-spring-cache | 2.1.0 |
| com.couchbase.client | java-client | 2.3.7 |
| com.datastax.cassandra | cassandra-driver-core | 3.1.4 |
| com.datastax.cassandra | cassandra-driver-mapping | 3.1.4 |
| com.fasterxml | classmate | 1.3.4 |
| com.fasterxml.jackson.core | jackson-annotations | 2.8.0 |
| com.fasterxml.jackson.core | jackson-core | 2.8.10 |
| com.fasterxml.jackson.core | jackson-databind | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-avro | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-cbor | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-csv | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-ion | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-properties | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-protobuf | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-smile | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-xml | 2.8.10 |
| com.fasterxml.jackson.dataformat | jackson-dataformat-yaml | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-guava | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-hibernate3 | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-hibernate4 | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-hibernate5 | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-hppc | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-jaxrs | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-jdk8 | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-joda | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-json-org | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-jsr310 | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-jsr353 | 2.8.10 |
| com.fasterxml.jackson.datatype | jackson-datatype-pcollections | 2.8.10 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-base | 2.8.10 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-cbor-provider | 2.8.10 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-json-provider | 2.8.10 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-smile-provider | 2.8.10 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-xml-provider | 2.8.10 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-yaml-provider | 2.8.10 |
| com.fasterxml.jackson.jr | jackson-jr-all | 2.8.10 |
| com.fasterxml.jackson.jr | jackson-jr-objects | 2.8.10 |
| com.fasterxml.jackson.jr | jackson-jr-retrofit2 | 2.8.10 |
| com.fasterxml.jackson.jr | jackson-jr-stree | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-afterburner | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-guice | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-jaxb-annotations | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-jsonSchema | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-kotlin | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-mrbean | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-osgi | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-parameter-names | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-paranamer | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-scala\_2.10 | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-scala\_2.11 | 2.8.10 |
| com.fasterxml.jackson.module | jackson-module-scala\_2.12 | 2.8.10 |
| com.gemstone.gemfire | gemfire | 8.2.8 |
| com.github.ben-manes.caffeine | caffeine | 2.3.5 |
| com.github.mxab.thymeleaf.extras | thymeleaf-extras-data-attribute | 1.3 |
| com.google.appengine | appengine-api-1.0-sdk | 1.9.60 |
| com.google.code.gson | gson | 2.8.2 |
| com.googlecode.json-simple | json-simple | 1.1.1 |
| com.h2database | h2 | 1.4.196 |
| com.hazelcast | hazelcast | 3.7.8 |
| com.hazelcast | hazelcast-client | 3.7.8 |
| com.hazelcast | hazelcast-hibernate4 | 3.7.1 |
| com.hazelcast | hazelcast-hibernate5 | 1.1.3 |
| com.hazelcast | hazelcast-spring | 3.7.8 |
| com.jayway.jsonpath | json-path | 2.2.0 |
| com.jayway.jsonpath | json-path-assert | 2.2.0 |
| com.microsoft.sqlserver | mssql-jdbc | 6.1.0.jre7 |
| com.querydsl | querydsl-apt | 4.1.4 |
| com.querydsl | querydsl-collections | 4.1.4 |
| com.querydsl | querydsl-core | 4.1.4 |
| com.querydsl | querydsl-jpa | 4.1.4 |
| com.querydsl | querydsl-mongodb | 4.1.4 |
| com.samskivert | jmustache | 1.13 |
| com.sendgrid | sendgrid-java | 2.2.2 |
| com.sun.mail | javax.mail | 1.5.6 |
| com.timgroup | java-statsd-client | 3.1.0 |
| com.unboundid | unboundid-ldapsdk | 3.2.1 |
| com.zaxxer | HikariCP | 2.5.1 |
| com.zaxxer | HikariCP-java6 | 2.3.13 |
| com.zaxxer | HikariCP-java7 | 2.4.13 |
| commons-beanutils | commons-beanutils | 1.9.3 |
| commons-codec | commons-codec | 1.10 |
| commons-collections | commons-collections | 3.2.2 |
| commons-dbcp | commons-dbcp | 1.4 |
| commons-digester | commons-digester | 2.1 |
| commons-pool | commons-pool | 1.6 |
| de.flapdoodle.embed | de.flapdoodle.embed.mongo | 1.50.5 |
| dom4j | dom4j | 1.6.1 |
| io.dropwizard.metrics | metrics-annotation | 3.1.5 |
| io.dropwizard.metrics | metrics-core | 3.1.5 |
| io.dropwizard.metrics | metrics-ehcache | 3.1.5 |
| io.dropwizard.metrics | metrics-ganglia | 3.1.5 |
| io.dropwizard.metrics | metrics-graphite | 3.1.5 |
| io.dropwizard.metrics | metrics-healthchecks | 3.1.5 |
| io.dropwizard.metrics | metrics-httpasyncclient | 3.1.5 |
| io.dropwizard.metrics | metrics-jdbi | 3.1.5 |
| io.dropwizard.metrics | metrics-jersey | 3.1.5 |
| io.dropwizard.metrics | metrics-jersey2 | 3.1.5 |
| io.dropwizard.metrics | metrics-jetty8 | 3.1.5 |
| io.dropwizard.metrics | metrics-jetty9 | 3.1.5 |
| io.dropwizard.metrics | metrics-jetty9-legacy | 3.1.5 |
| io.dropwizard.metrics | metrics-json | 3.1.5 |
| io.dropwizard.metrics | metrics-jvm | 3.1.5 |
| io.dropwizard.metrics | metrics-log4j | 3.1.5 |
| io.dropwizard.metrics | metrics-log4j2 | 3.1.5 |
| io.dropwizard.metrics | metrics-logback | 3.1.5 |
| io.dropwizard.metrics | metrics-servlet | 3.1.5 |
| io.dropwizard.metrics | metrics-servlets | 3.1.5 |
| io.projectreactor | reactor-bus | 2.0.8.RELEASE |
| io.projectreactor | reactor-core | 2.0.8.RELEASE |
| io.projectreactor | reactor-groovy | 2.0.8.RELEASE |
| io.projectreactor | reactor-groovy-extensions | 2.0.8.RELEASE |
| io.projectreactor | reactor-logback | 2.0.8.RELEASE |
| io.projectreactor | reactor-net | 2.0.8.RELEASE |
| io.projectreactor | reactor-stream | 2.0.8.RELEASE |
| io.projectreactor.spring | reactor-spring-context | 2.0.7.RELEASE |
| io.projectreactor.spring | reactor-spring-core | 2.0.7.RELEASE |
| io.projectreactor.spring | reactor-spring-messaging | 2.0.7.RELEASE |
| io.projectreactor.spring | reactor-spring-webmvc | 2.0.7.RELEASE |
| io.searchbox | jest | 2.0.4 |
| io.undertow | undertow-core | 1.4.22.Final |
| io.undertow | undertow-servlet | 1.4.22.Final |
| io.undertow | undertow-websockets-jsr | 1.4.22.Final |
| javax.cache | cache-api | 1.0.0 |
| javax.jms | jms-api | 1.1-rev-1 |
| javax.mail | javax.mail-api | 1.5.6 |
| javax.servlet | javax.servlet-api | 3.1.0 |
| javax.servlet | jstl | 1.2 |
| javax.transaction | javax.transaction-api | 1.2 |
| javax.validation | validation-api | 1.1.0.Final |
| jaxen | jaxen | 1.1.6 |
| joda-time | joda-time | 2.9.9 |
| junit | junit | 4.12 |
| mysql | mysql-connector-java | 5.1.45 |
| net.java.dev.jna | jna | 4.2.2 |
| net.java.dev.jna | jna-platform | 4.2.2 |
| net.sf.ehcache | ehcache | 2.10.4 |
| net.sourceforge.htmlunit | htmlunit | 2.21 |
| net.sourceforge.jtds | jtds | 1.3.1 |
| net.sourceforge.nekohtml | nekohtml | 1.9.22 |
| nz.net.ultraq.thymeleaf | thymeleaf-layout-dialect | 1.4.0 |
| org.apache.activemq | activemq-amqp | 5.14.5 |
| org.apache.activemq | activemq-blueprint | 5.14.5 |
| org.apache.activemq | activemq-broker | 5.14.5 |
| org.apache.activemq | activemq-camel | 5.14.5 |
| org.apache.activemq | activemq-client | 5.14.5 |
| org.apache.activemq | activemq-console | 5.14.5 |
| org.apache.activemq | activemq-http | 5.14.5 |
| org.apache.activemq | activemq-jaas | 5.14.5 |
| org.apache.activemq | activemq-jdbc-store | 5.14.5 |
| org.apache.activemq | activemq-jms-pool | 5.14.5 |
| org.apache.activemq | activemq-kahadb-store | 5.14.5 |
| org.apache.activemq | activemq-karaf | 5.14.5 |
| org.apache.activemq | activemq-leveldb-store | 5.14.5 |
| org.apache.activemq | activemq-log4j-appender | 5.14.5 |
| org.apache.activemq | activemq-mqtt | 5.14.5 |
| org.apache.activemq | activemq-openwire-generator | 5.14.5 |
| org.apache.activemq | activemq-openwire-legacy | 5.14.5 |
| org.apache.activemq | activemq-osgi | 5.14.5 |
| org.apache.activemq | activemq-partition | 5.14.5 |
| org.apache.activemq | activemq-pool | 5.14.5 |
| org.apache.activemq | activemq-ra | 5.14.5 |
| org.apache.activemq | activemq-run | 5.14.5 |
| org.apache.activemq | activemq-runtime-config | 5.14.5 |
| org.apache.activemq | activemq-shiro | 5.14.5 |
| org.apache.activemq | activemq-spring | 5.14.5 |
| org.apache.activemq | activemq-stomp | 5.14.5 |
| org.apache.activemq | activemq-web | 5.14.5 |
| org.apache.activemq | artemis-amqp-protocol | 1.5.5 |
| org.apache.activemq | artemis-commons | 1.5.5 |
| org.apache.activemq | artemis-core-client | 1.5.5 |
| org.apache.activemq | artemis-jms-client | 1.5.5 |
| org.apache.activemq | artemis-jms-server | 1.5.5 |
| org.apache.activemq | artemis-journal | 1.5.5 |
| org.apache.activemq | artemis-native | 1.5.5 |
| org.apache.activemq | artemis-selector | 1.5.5 |
| org.apache.activemq | artemis-server | 1.5.5 |
| org.apache.activemq | artemis-service-extensions | 1.5.5 |
| org.apache.commons | commons-dbcp2 | 2.1.1 |
| org.apache.commons | commons-pool2 | 2.4.3 |
| org.apache.derby | derby | 10.13.1.1 |
| org.apache.httpcomponents | httpasyncclient | 4.1.3 |
| org.apache.httpcomponents | httpclient | 4.5.5 |
| org.apache.httpcomponents | httpcore | 4.4.9 |
| org.apache.httpcomponents | httpmime | 4.5.5 |
| org.apache.logging.log4j | log4j-1.2-api | 2.7 |
| org.apache.logging.log4j | log4j-api | 2.7 |
| org.apache.logging.log4j | log4j-api-scala\_2.10 | 2.7 |
| org.apache.logging.log4j | log4j-api-scala\_2.11 | 2.7 |
| org.apache.logging.log4j | log4j-core | 2.7 |
| org.apache.logging.log4j | log4j-flume-ng | 2.7 |
| org.apache.logging.log4j | log4j-iostreams | 2.7 |
| org.apache.logging.log4j | log4j-jcl | 2.7 |
| org.apache.logging.log4j | log4j-jmx-gui | 2.7 |
| org.apache.logging.log4j | log4j-jul | 2.7 |
| org.apache.logging.log4j | log4j-liquibase | 2.7 |
| org.apache.logging.log4j | log4j-nosql | 2.7 |
| org.apache.logging.log4j | log4j-slf4j-impl | 2.7 |
| org.apache.logging.log4j | log4j-taglib | 2.7 |
| org.apache.logging.log4j | log4j-web | 2.7 |
| org.apache.solr | solr-analysis-extras | 5.5.5 |
| org.apache.solr | solr-analytics | 5.5.5 |
| org.apache.solr | solr-cell | 5.5.5 |
| org.apache.solr | solr-clustering | 5.5.5 |
| org.apache.solr | solr-core | 5.5.5 |
| org.apache.solr | solr-dataimporthandler | 5.5.5 |
| org.apache.solr | solr-dataimporthandler-extras | 5.5.5 |
| org.apache.solr | solr-langid | 5.5.5 |
| org.apache.solr | solr-map-reduce | 5.5.5 |
| org.apache.solr | solr-morphlines-cell | 5.5.5 |
| org.apache.solr | solr-morphlines-core | 5.5.5 |
| org.apache.solr | solr-solrj | 5.5.5 |
| org.apache.solr | solr-test-framework | 5.5.5 |
| org.apache.solr | solr-uima | 5.5.5 |
| org.apache.solr | solr-velocity | 5.5.5 |
| org.apache.tomcat | tomcat-annotations-api | 8.5.27 |
| org.apache.tomcat | tomcat-jdbc | 8.5.27 |
| org.apache.tomcat | tomcat-jsp-api | 8.5.27 |
| org.apache.tomcat.embed | tomcat-embed-core | 8.5.27 |
| org.apache.tomcat.embed | tomcat-embed-el | 8.5.27 |
| org.apache.tomcat.embed | tomcat-embed-jasper | 8.5.27 |
| org.apache.tomcat.embed | tomcat-embed-websocket | 8.5.27 |
| org.aspectj | aspectjrt | 1.8.13 |
| org.aspectj | aspectjtools | 1.8.13 |
| org.aspectj | aspectjweaver | 1.8.13 |
| org.assertj | assertj-core | 2.6.0 |
| org.codehaus.btm | btm | 2.1.4 |
| org.codehaus.groovy | groovy | 2.4.13 |
| org.codehaus.groovy | groovy-all | 2.4.13 |
| org.codehaus.groovy | groovy-ant | 2.4.13 |
| org.codehaus.groovy | groovy-bsf | 2.4.13 |
| org.codehaus.groovy | groovy-console | 2.4.13 |
| org.codehaus.groovy | groovy-docgenerator | 2.4.13 |
| org.codehaus.groovy | groovy-groovydoc | 2.4.13 |
| org.codehaus.groovy | groovy-groovysh | 2.4.13 |
| org.codehaus.groovy | groovy-jmx | 2.4.13 |
| org.codehaus.groovy | groovy-json | 2.4.13 |
| org.codehaus.groovy | groovy-jsr223 | 2.4.13 |
| org.codehaus.groovy | groovy-nio | 2.4.13 |
| org.codehaus.groovy | groovy-servlet | 2.4.13 |
| org.codehaus.groovy | groovy-sql | 2.4.13 |
| org.codehaus.groovy | groovy-swing | 2.4.13 |
| org.codehaus.groovy | groovy-templates | 2.4.13 |
| org.codehaus.groovy | groovy-test | 2.4.13 |
| org.codehaus.groovy | groovy-testng | 2.4.13 |
| org.codehaus.groovy | groovy-xml | 2.4.13 |
| org.codehaus.janino | janino | 2.7.8 |
| org.crashub | crash.cli | 1.3.2 |
| org.crashub | crash.connectors.ssh | 1.3.2 |
| org.crashub | crash.connectors.telnet | 1.3.2 |
| org.crashub | crash.embed.spring | 1.3.2 |
| org.crashub | crash.plugins.cron | 1.3.2 |
| org.crashub | crash.plugins.mail | 1.3.2 |
| org.crashub | crash.shell | 1.3.2 |
| org.eclipse.jetty | apache-jsp | 9.4.8.v20171121 |
| org.eclipse.jetty | apache-jstl | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-alpn-client | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-alpn-java-client | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-alpn-java-server | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-alpn-server | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-annotations | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-ant | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-client | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-continuation | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-deploy | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-hazelcast | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-http | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-http-spi | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-infinispan | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-io | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-jaas | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-jaspi | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-jmx | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-jndi | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-nosql | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-plus | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-proxy | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-quickstart | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-rewrite | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-runner | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-security | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-server | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-servlet | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-servlets | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-spring | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-start | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-unixsocket | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-util | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-util-ajax | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-webapp | 9.4.8.v20171121 |
| org.eclipse.jetty | jetty-xml | 9.4.8.v20171121 |
| org.eclipse.jetty.cdi | cdi-core | 9.4.8.v20171121 |
| org.eclipse.jetty.cdi | cdi-servlet | 9.4.8.v20171121 |
| org.eclipse.jetty.fcgi | fcgi-client | 9.4.8.v20171121 |
| org.eclipse.jetty.fcgi | fcgi-server | 9.4.8.v20171121 |
| org.eclipse.jetty.gcloud | jetty-gcloud-session-manager | 9.4.8.v20171121 |
| org.eclipse.jetty.http2 | http2-client | 9.4.8.v20171121 |
| org.eclipse.jetty.http2 | http2-common | 9.4.8.v20171121 |
| org.eclipse.jetty.http2 | http2-hpack | 9.4.8.v20171121 |
| org.eclipse.jetty.http2 | http2-http-client-transport | 9.4.8.v20171121 |
| org.eclipse.jetty.http2 | http2-server | 9.4.8.v20171121 |
| org.eclipse.jetty.memcached | jetty-memcached-sessions | 9.4.8.v20171121 |
| org.eclipse.jetty.orbit | javax.servlet.jsp | 2.2.0.v201112011158 |
| org.eclipse.jetty.osgi | jetty-httpservice | 9.4.8.v20171121 |
| org.eclipse.jetty.osgi | jetty-osgi-boot | 9.4.8.v20171121 |
| org.eclipse.jetty.osgi | jetty-osgi-boot-jsp | 9.4.8.v20171121 |
| org.eclipse.jetty.osgi | jetty-osgi-boot-warurl | 9.4.8.v20171121 |
| org.eclipse.jetty.websocket | javax-websocket-client-impl | 9.4.8.v20171121 |
| org.eclipse.jetty.websocket | javax-websocket-server-impl | 9.4.8.v20171121 |
| org.eclipse.jetty.websocket | websocket-api | 9.4.8.v20171121 |
| org.eclipse.jetty.websocket | websocket-client | 9.4.8.v20171121 |
| org.eclipse.jetty.websocket | websocket-common | 9.4.8.v20171121 |
| org.eclipse.jetty.websocket | websocket-server | 9.4.8.v20171121 |
| org.eclipse.jetty.websocket | websocket-servlet | 9.4.8.v20171121 |
| org.ehcache | ehcache | 3.2.3 |
| org.ehcache | ehcache-clustered | 3.2.3 |
| org.ehcache | ehcache-transactions | 3.2.3 |
| org.elasticsearch | elasticsearch | 2.4.6 |
| org.firebirdsql.jdbc | jaybird-jdk16 | 2.2.14 |
| org.firebirdsql.jdbc | jaybird-jdk17 | 2.2.14 |
| org.firebirdsql.jdbc | jaybird-jdk18 | 2.2.14 |
| org.flywaydb | flyway-core | 3.2.1 |
| org.freemarker | freemarker | 2.3.27-incubating |
| org.glassfish | javax.el | 3.0.0 |
| org.glassfish.jersey.bundles.repackaged | jersey-guava | 2.25.1 |
| org.glassfish.jersey.containers | jersey-container-servlet | 2.25.1 |
| org.glassfish.jersey.containers | jersey-container-servlet-core | 2.25.1 |
| org.glassfish.jersey.core | jersey-client | 2.25.1 |
| org.glassfish.jersey.core | jersey-common | 2.25.1 |
| org.glassfish.jersey.core | jersey-server | 2.25.1 |
| org.glassfish.jersey.ext | jersey-bean-validation | 2.25.1 |
| org.glassfish.jersey.ext | jersey-entity-filtering | 2.25.1 |
| org.glassfish.jersey.ext | jersey-spring3 | 2.25.1 |
| org.glassfish.jersey.media | jersey-media-jaxb | 2.25.1 |
| org.glassfish.jersey.media | jersey-media-json-jackson | 2.25.1 |
| org.glassfish.jersey.media | jersey-media-multipart | 2.25.1 |
| org.hamcrest | hamcrest-core | 1.3 |
| org.hamcrest | hamcrest-library | 1.3 |
| org.hibernate | hibernate-core | 5.0.12.Final |
| org.hibernate | hibernate-ehcache | 5.0.12.Final |
| org.hibernate | hibernate-entitymanager | 5.0.12.Final |
| org.hibernate | hibernate-envers | 5.0.12.Final |
| org.hibernate | hibernate-java8 | 5.0.12.Final |
| org.hibernate | hibernate-jpamodelgen | 5.0.12.Final |
| org.hibernate | hibernate-validator | 5.3.6.Final |
| org.hibernate | hibernate-validator-annotation-processor | 5.3.6.Final |
| org.hsqldb | hsqldb | 2.3.5 |
| org.infinispan | infinispan-jcache | 8.2.8.Final |
| org.infinispan | infinispan-spring4-common | 8.2.8.Final |
| org.infinispan | infinispan-spring4-embedded | 8.2.8.Final |
| org.javassist | javassist | 3.21.0-GA |
| org.jboss | jboss-transaction-spi | 7.6.0.Final |
| org.jboss.logging | jboss-logging | 3.3.1.Final |
| org.jboss.narayana.jta | jdbc | 5.5.31.Final |
| org.jboss.narayana.jta | jms | 5.5.31.Final |
| org.jboss.narayana.jta | jta | 5.5.31.Final |
| org.jboss.narayana.jts | narayana-jts-integration | 5.5.31.Final |
| org.jdom | jdom2 | 2.0.6 |
| org.jolokia | jolokia-core | 1.3.7 |
| org.jooq | jooq | 3.9.6 |
| org.jooq | jooq-codegen | 3.9.6 |
| org.jooq | jooq-meta | 3.9.6 |
| org.json | json | 20140107 |
| org.liquibase | liquibase-core | 3.5.3 |
| org.mariadb.jdbc | mariadb-java-client | 1.5.9 |
| org.mockito | mockito-core | 1.10.19 |
| org.mongodb | mongodb-driver | 3.4.3 |
| org.mongodb | mongo-java-driver | 3.4.3 |
| org.mortbay.jasper | apache-el | 8.0.33 |
| org.neo4j | neo4j-ogm-api | 2.1.6 |
| org.neo4j | neo4j-ogm-compiler | 2.1.6 |
| org.neo4j | neo4j-ogm-core | 2.1.6 |
| org.neo4j | neo4j-ogm-http-driver | 2.1.6 |
| org.postgresql | postgresql | 9.4.1212.jre7 |
| org.projectlombok | lombok | 1.16.20 |
| org.seleniumhq.selenium | htmlunit-driver | 2.21 |
| org.seleniumhq.selenium | selenium-api | 2.53.1 |
| org.seleniumhq.selenium | selenium-chrome-driver | 2.53.1 |
| org.seleniumhq.selenium | selenium-edge-driver | 2.53.1 |
| org.seleniumhq.selenium | selenium-firefox-driver | 2.53.1 |
| org.seleniumhq.selenium | selenium-ie-driver | 2.53.1 |
| org.seleniumhq.selenium | selenium-java | 2.53.1 |
| org.seleniumhq.selenium | selenium-remote-driver | 2.53.1 |
| org.seleniumhq.selenium | selenium-safari-driver | 2.53.1 |
| org.seleniumhq.selenium | selenium-support | 2.53.1 |
| org.skyscreamer | jsonassert | 1.4.0 |
| org.slf4j | jcl-over-slf4j | 1.7.25 |
| org.slf4j | jul-to-slf4j | 1.7.25 |
| org.slf4j | log4j-over-slf4j | 1.7.25 |
| org.slf4j | slf4j-api | 1.7.25 |
| org.slf4j | slf4j-ext | 1.7.25 |
| org.slf4j | slf4j-jcl | 1.7.25 |
| org.slf4j | slf4j-jdk14 | 1.7.25 |
| org.slf4j | slf4j-log4j12 | 1.7.25 |
| org.slf4j | slf4j-nop | 1.7.25 |
| org.slf4j | slf4j-simple | 1.7.25 |
| org.spockframework | spock-core | 1.0-groovy-2.4 |
| org.spockframework | spock-spring | 1.0-groovy-2.4 |
| org.springframework | spring-aop | 4.3.14.RELEASE |
| org.springframework | spring-aspects | 4.3.14.RELEASE |
| org.springframework | spring-beans | 4.3.14.RELEASE |
| org.springframework | spring-context | 4.3.14.RELEASE |
| org.springframework | spring-context-support | 4.3.14.RELEASE |
| org.springframework | spring-core | 4.3.14.RELEASE |
| org.springframework | spring-expression | 4.3.14.RELEASE |
| org.springframework | spring-instrument | 4.3.14.RELEASE |
| org.springframework | spring-instrument-tomcat | 4.3.14.RELEASE |
| org.springframework | spring-jdbc | 4.3.14.RELEASE |
| org.springframework | spring-jms | 4.3.14.RELEASE |
| org.springframework | springloaded | 1.2.8.RELEASE |
| org.springframework | spring-messaging | 4.3.14.RELEASE |
| org.springframework | spring-orm | 4.3.14.RELEASE |
| org.springframework | spring-oxm | 4.3.14.RELEASE |
| org.springframework | spring-test | 4.3.14.RELEASE |
| org.springframework | spring-tx | 4.3.14.RELEASE |
| org.springframework | spring-web | 4.3.14.RELEASE |
| org.springframework | spring-webmvc | 4.3.14.RELEASE |
| org.springframework | spring-webmvc-portlet | 4.3.14.RELEASE |
| org.springframework | spring-websocket | 4.3.14.RELEASE |
| org.springframework.amqp | spring-amqp | 1.7.6.RELEASE |
| org.springframework.amqp | spring-rabbit | 1.7.6.RELEASE |
| org.springframework.batch | spring-batch-core | 3.0.8.RELEASE |
| org.springframework.batch | spring-batch-infrastructure | 3.0.8.RELEASE |
| org.springframework.batch | spring-batch-integration | 3.0.8.RELEASE |
| org.springframework.batch | spring-batch-test | 3.0.8.RELEASE |
| org.springframework.boot | spring-boot | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-actuator | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-actuator-docs | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-autoconfigure | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-autoconfigure-processor | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-configuration-metadata | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-configuration-processor | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-devtools | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-loader | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-loader-tools | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-activemq | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-actuator | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-amqp | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-aop | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-artemis | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-batch | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-cache | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-cloud-connectors | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-cassandra | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-couchbase | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-elasticsearch | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-gemfire | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-jpa | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-ldap | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-mongodb | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-neo4j | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-redis | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-rest | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-data-solr | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-freemarker | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-groovy-templates | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-hateoas | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-integration | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-jdbc | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-jersey | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-jetty | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-jooq | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-jta-atomikos | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-jta-bitronix | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-jta-narayana | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-log4j2 | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-logging | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-mail | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-mobile | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-mustache | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-remote-shell | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-security | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-social-facebook | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-social-linkedin | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-social-twitter | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-test | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-thymeleaf | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-tomcat | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-undertow | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-validation | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-web | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-web-services | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-starter-websocket | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-test | 1.5.10.RELEASE |
| org.springframework.boot | spring-boot-test-autoconfigure | 1.5.10.RELEASE |
| org.springframework.cloud | spring-cloud-cloudfoundry-connector | 1.2.5.RELEASE |
| org.springframework.cloud | spring-cloud-core | 1.2.5.RELEASE |
| org.springframework.cloud | spring-cloud-heroku-connector | 1.2.5.RELEASE |
| org.springframework.cloud | spring-cloud-localconfig-connector | 1.2.5.RELEASE |
| org.springframework.cloud | spring-cloud-spring-service-connector | 1.2.5.RELEASE |
| org.springframework.data | spring-cql | 1.5.10.RELEASE |
| org.springframework.data | spring-data-cassandra | 1.5.10.RELEASE |
| org.springframework.data | spring-data-commons | 1.13.10.RELEASE |
| org.springframework.data | spring-data-couchbase | 2.2.10.RELEASE |
| org.springframework.data | spring-data-elasticsearch | 2.1.10.RELEASE |
| org.springframework.data | spring-data-envers | 1.1.10.RELEASE |
| org.springframework.data | spring-data-gemfire | 1.9.10.RELEASE |
| org.springframework.data | spring-data-jpa | 1.11.10.RELEASE |
| org.springframework.data | spring-data-keyvalue | 1.2.10.RELEASE |
| org.springframework.data | spring-data-ldap | 1.0.10.RELEASE |
| org.springframework.data | spring-data-mongodb | 1.10.10.RELEASE |
| org.springframework.data | spring-data-mongodb-cross-store | 1.10.10.RELEASE |
| org.springframework.data | spring-data-mongodb-log4j | 1.10.10.RELEASE |
| org.springframework.data | spring-data-neo4j | 4.2.10.RELEASE |
| org.springframework.data | spring-data-redis | 1.8.10.RELEASE |
| org.springframework.data | spring-data-rest-core | 2.6.10.RELEASE |
| org.springframework.data | spring-data-rest-hal-browser | 2.6.10.RELEASE |
| org.springframework.data | spring-data-rest-webmvc | 2.6.10.RELEASE |
| org.springframework.data | spring-data-solr | 2.1.10.RELEASE |
| org.springframework.hateoas | spring-hateoas | 0.23.0.RELEASE |
| org.springframework.integration | spring-integration-amqp | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-core | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-event | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-feed | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-file | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-ftp | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-gemfire | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-groovy | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-http | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-ip | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-java-dsl | 1.2.3.RELEASE |
| org.springframework.integration | spring-integration-jdbc | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-jms | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-jmx | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-jpa | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-mail | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-mongodb | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-mqtt | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-redis | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-rmi | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-scripting | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-security | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-sftp | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-stomp | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-stream | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-syslog | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-test | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-twitter | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-websocket | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-ws | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-xml | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-xmpp | 4.3.14.RELEASE |
| org.springframework.integration | spring-integration-zookeeper | 4.3.14.RELEASE |
| org.springframework.kafka | spring-kafka | 1.1.7.RELEASE |
| org.springframework.kafka | spring-kafka-test | 1.1.7.RELEASE |
| org.springframework.ldap | spring-ldap-core | 2.3.2.RELEASE |
| org.springframework.ldap | spring-ldap-core-tiger | 2.3.2.RELEASE |
| org.springframework.ldap | spring-ldap-ldif-batch | 2.3.2.RELEASE |
| org.springframework.ldap | spring-ldap-ldif-core | 2.3.2.RELEASE |
| org.springframework.ldap | spring-ldap-odm | 2.3.2.RELEASE |
| org.springframework.ldap | spring-ldap-test | 2.3.2.RELEASE |
| org.springframework.mobile | spring-mobile-device | 1.1.5.RELEASE |
| org.springframework.plugin | spring-plugin-core | 1.2.0.RELEASE |
| org.springframework.plugin | spring-plugin-metadata | 1.2.0.RELEASE |
| org.springframework.restdocs | spring-restdocs-core | 1.1.3.RELEASE |
| org.springframework.restdocs | spring-restdocs-mockmvc | 1.1.3.RELEASE |
| org.springframework.restdocs | spring-restdocs-restassured | 1.1.3.RELEASE |
| org.springframework.retry | spring-retry | 1.2.2.RELEASE |
| org.springframework.security | spring-security-acl | 4.2.4.RELEASE |
| org.springframework.security | spring-security-aspects | 4.2.4.RELEASE |
| org.springframework.security | spring-security-cas | 4.2.4.RELEASE |
| org.springframework.security | spring-security-config | 4.2.4.RELEASE |
| org.springframework.security | spring-security-core | 4.2.4.RELEASE |
| org.springframework.security | spring-security-crypto | 4.2.4.RELEASE |
| org.springframework.security | spring-security-data | 4.2.4.RELEASE |
| org.springframework.security | spring-security-jwt | 1.0.9.RELEASE |
| org.springframework.security | spring-security-ldap | 4.2.4.RELEASE |
| org.springframework.security | spring-security-messaging | 4.2.4.RELEASE |
| org.springframework.security | spring-security-openid | 4.2.4.RELEASE |
| org.springframework.security | spring-security-remoting | 4.2.4.RELEASE |
| org.springframework.security | spring-security-taglibs | 4.2.4.RELEASE |
| org.springframework.security | spring-security-test | 4.2.4.RELEASE |
| org.springframework.security | spring-security-web | 4.2.4.RELEASE |
| org.springframework.security.oauth | spring-security-oauth | 2.0.14.RELEASE |
| org.springframework.security.oauth | spring-security-oauth2 | 2.0.14.RELEASE |
| org.springframework.session | spring-session | 1.3.1.RELEASE |
| org.springframework.session | spring-session-data-gemfire | 1.3.1.RELEASE |
| org.springframework.session | spring-session-data-mongo | 1.3.1.RELEASE |
| org.springframework.session | spring-session-data-redis | 1.3.1.RELEASE |
| org.springframework.session | spring-session-hazelcast | 1.3.1.RELEASE |
| org.springframework.session | spring-session-jdbc | 1.3.1.RELEASE |
| org.springframework.social | spring-social-config | 1.1.4.RELEASE |
| org.springframework.social | spring-social-core | 1.1.4.RELEASE |
| org.springframework.social | spring-social-facebook | 2.0.3.RELEASE |
| org.springframework.social | spring-social-facebook-web | 2.0.3.RELEASE |
| org.springframework.social | spring-social-linkedin | 1.0.2.RELEASE |
| org.springframework.social | spring-social-security | 1.1.4.RELEASE |
| org.springframework.social | spring-social-twitter | 1.1.2.RELEASE |
| org.springframework.social | spring-social-web | 1.1.4.RELEASE |
| org.springframework.ws | spring-ws-core | 2.4.2.RELEASE |
| org.springframework.ws | spring-ws-security | 2.4.2.RELEASE |
| org.springframework.ws | spring-ws-support | 2.4.2.RELEASE |
| org.springframework.ws | spring-ws-test | 2.4.2.RELEASE |
| org.thymeleaf | thymeleaf | 2.1.6.RELEASE |
| org.thymeleaf | thymeleaf-spring4 | 2.1.6.RELEASE |
| org.thymeleaf.extras | thymeleaf-extras-conditionalcomments | 2.1.2.RELEASE |
| org.thymeleaf.extras | thymeleaf-extras-java8time | 2.1.0.RELEASE |
| org.thymeleaf.extras | thymeleaf-extras-springsecurity4 | 2.1.3.RELEASE |
| org.webjars | hal-browser | 9f96c74 |
| org.webjars | webjars-locator | 0.32-1 |
| org.xerial | sqlite-jdbc | 3.15.1 |
| org.yaml | snakeyaml | 1.17 |
| redis.clients | jedis | 2.9.0 |
| wsdl4j | wsdl4j | 1.6.3 |
| xml-apis | xml-apis | 1.4.01 |