目錄

Spring-Boot-Reference-Guide

Spring Boot Reference Guide中文翻译 -《Spring Boot参考指南》

说明：本文档翻译的版本：1.4.1.RELEASE。 翻译完毕会出教程，敬请期待！

最新版本地址：2.x。

如感兴趣，可以star或fork该仓库！

Github：https://github.com/qibaoguang/

GitBook : Spring Boot参考指南

整合示例：程序猿DD-Spring Boot教程

Email：qibaoguang@gmail.com

从这里开始

交流群：

- spring boot最佳实践2 ： 460560346
- spring boot最佳实践（已满） ：445015546

注 1.3版本查看本仓库的release。

Spring Boot文档

本节对Spring Boot参考文档做了一个简单概述。你可以参考本节，从头到尾依次阅读该文档，也可以跳过不感兴趣的章节。

1. 关于本文档

Spring Boot参考指南有html，pdf和epub等形式的文档，你可以从docs.spring.io/spring-boot/docs/current/reference获取到最新版本。

2. 获取帮助

使用Spring Boot遇到麻烦，我们很乐意帮忙！

- 尝试How-to's－它们为多数常见问题提供解决方案。
- 学习Spring基础知识－Spring Boot是在很多其他Spring项目上构建的，查看spring.io站点可以获取丰富的参考文档。如果你刚开始使用Spring，可以尝试这些指导中的一个。
- 提问题－我们时刻监控着stackoverflow.com上标记为spring-boot的问题。
- 在github.com/spring-projects/spring-boot/issues上报告Spring Boot的bug。

注：Spring Boot的一切都是开源的，包括文档！如果你发现文档有问题，或只是想提高它们的质量，请参与进来！

3. 第一步

如果你想对Spring Boot或Spring有个整体认识，可以从这里开始！

- 从零开始：概述 | 要求 | 安装
- 教程：第一部分 | 第二部分
- 运行示例：第一部分 | 第二部分

4. 使用Spring Boot

准备好使用Spring Boot了？我们已经为你铺好道路.

- 构建系统：Maven | Gradle | Ant | Starters
- 最佳实践：代码结构 | @Configuration | @EnableAutoConfiguration | Beans和依赖注入
- 运行代码：IDE | Packaged | Maven | Gradle
- 应用打包：产品级jars
- Spring Boot命令行：使用CLI

5. 了解Spring Boot特性

想要了解更多Spring Boot核心特性的详情？这就是为你准备的！

- 核心特性：SpringApplication | 外部化配置 | Profiles | 日志
- Web应用：MVC | 内嵌容器
- 使用数据：SQL | NO-SQL
- 消息：概述 | JMS
- 测试：概述 | Boot应用 | 工具
- 扩展：Auto-configuration | @Conditions

6. 迁移到生产环境

当你准备将Spring Boot应用发布到生产环境时，我们提供了一些你可能喜欢的技巧！

- 管理端点：概述 | 自定义
- 连接选项：HTTP | JMX | SSH
- 监控：指标 | 审计 | 追踪 | 进程

7. 高级主题

最后，我们为高级用户准备了一些主题。

- 部署Spring Boot应用：云部署 | 操作系统服务
- 构建工具插件：Maven | Gradle
- 附录：应用属性 | Auto-configuration类 | 可执行Jars

入门指南

如果你想从大体上了解Spring Boot或Spring，本章节正是你所需要的！本节中，我们会回答基本的"what?"，"how?"和"why?"等问题，并通过一些安装指南简单介绍下Spring Boot。然后我们会构建第一个Spring Boot应用，并讨论一些需要遵循的核心原则。

8. Spring Boot介绍

Spring Boot简化了基于Spring的应用开发，你只需要"run"就能创建一个独立的，产品级别的Spring应用。 我们为Spring平台及第三方库提供开箱即用的设置，这样你就可以有条不紊地开始。多数Spring Boot应用只需要很少的Spring配置。

你可以使用Spring Boot创建Java应用，并使用 `java -jar` 启动它或采用传统的war部署方式。我们也提供了一个运行"spring脚本"的命令行工具。

我们主要的目标是：

- 为所有Spring开发提供一个从根本上更快，且随处可得的入门体验。
- 开箱即用，但通过不采用默认设置可以快速摆脱这种方式。
- 提供一系列大型项目常用的非功能性特征，比如：内嵌服务器，安全，指标，健康检测，外部化配置。
- 绝对没有代码生成，也不需要XML配置。

9. 系统要求

默认情况下，Spring Boot 1.4.0.BUILD-SNAPSHOT 需要Java7环境，Spring框架4.3.2.BUILD-SNAPSHOT或以上版本。你可以在Java6下使用Spring Boot，不过需要添加额外配置。具体参考Section 82.11, “How to use Java 6” 。明确提供构建支持的有Maven（3.2+）和Gradle（1.12+）。

注：尽管你可以在Java6或Java7环境下使用Spring Boot，通常建议尽可能使用Java8。

9.1. Servlet容器

下列内嵌容器支持开箱即用（out of the box）：

名称	**Servlet**版本	**Java**版本
Tomcat 8	3.1	Java 7+
Tomcat 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应用部署到任何兼容Servlet 3.0+的容器。

10. Spring Boot安装

Spring Boot可以跟经典的Java开发工具（Eclipse，IntelliJ等）一起使用或安装成一个命令行工具。不管怎样，你都需要安装Java SDK v1.6 或更高版本。在开始之前，你需要检查下当前安装的Java版本：

```
$ java -version
```

如果你是一个Java新手，或只是想体验一下Spring Boot，你可能想先尝试Spring Boot CLI，否则继续阅读“经典”地安装指南。

注：尽管Spring Boot兼容Java 1.6，如果可能的话，你应该考虑使用Java最新版本。

10.1. 为Java开发者准备的安装指南

对于java开发者来说，使用Spring Boot就跟使用其他Java库一样，只需要在你的classpath下引入适当的 `spring-boot-*.jar` 文件。Spring Boot不需要集成任何特殊的工具，所以你可以使用任何IDE或文本编辑器；同时，Spring Boot应用也没有什么特殊之处，你可以像对待其他Java程序那样运行，调试它。

尽管可以拷贝Spring Boot jars，但我们还是建议你使用支持依赖管理的构建工具，比如Maven或Gradle。

10.1.1. Maven安装

Spring Boot兼容Apache Maven 3.2或更高版本。如果本地没有安装Maven，你可以参考maven.apache.org上的指南。

注：在很多操作系统上，可以通过包管理器来安装Maven。OSX Homebrew用户可以尝试 `brew install maven` ，Ubuntu用户可以运行 `sudo apt-get install maven` 。

Spring Boot依赖使用的groupId为 `org.springframework.boot` 。通常，你的Maven POM文件会继承 `spring-boot-starter-parent` 工程，并声明一个或多个“Starter POMs”依赖。此外，Spring Boot提供了一个可选的Maven插件，用于创建可执行jars。

下面是一个典型的pom.xml文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
tp://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http:/
/maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>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>1.4.0.BUILD-SNAPSHOT</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>

    <!-- Add Spring repositories -->
    <!-- (you don't need this if you are using a .RELEASE versio
n) -->
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <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>
</project>
```

注： `spring-boot-starter-parent` 是使用Spring Boot的一种不错的方式，但它并不总是最合适的。有时你可能需要继承一个不同的父 POM，或只是不喜欢我们的默认配置，那你可以使用import作用域这种替代方案，具体查看Section 13.2.2, “Using Spring Boot without the parent POM”。

10.1.2. Gradle安装

Spring Boot兼容Gradle 1.12或更高版本。如果本地没有安装Gradle，你可以参考www.gradle.org上的指南。

Spring Boot的依赖可通过groupId `org.springframework.boot` 来声明。通常，你的项目将声明一个或多个“Starter POMs”依赖。Spring Boot提供了一个很有用的Gradle插件，可以用来简化依赖声明，创建可执行jars。

注：当你需要构建项目时，Gradle Wrapper提供一种给力的获取Gradle的方式。它是一小段脚本和库，跟你的代码一块提交，用于启动构建进程，具体参考Gradle Wrapper。

下面是一个典型的 `build.gradle` 文件：

```
buildscript {
    repositories {
        jcenter()
        maven { url "http://repo.spring.io/snapshot" }
        maven { url "http://repo.spring.io/milestone" }
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-p
lugin:1.4.0.BUILD-SNAPSHOT")
    }
}

apply plugin: 'java'
apply plugin: 'spring-boot'

jar {
    baseName = 'myproject'
    version =  '0.0.1-SNAPSHOT'
}

repositories {
    jcenter()
    maven { url "http://repo.spring.io/snapshot" }
    maven { url "http://repo.spring.io/milestone" }
}

dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    testCompile("org.springframework.boot:spring-boot-starter-te
st")
}
```

10.2. Spring Boot CLI安装

Spring Boot CLI是一个命令行工具，可用于快速搭建基于Spring的原型。它支持运行Groovy脚本，这也就意味着你可以使用类似Java的语法，但不用写很多的模板代码。

Spring Boot不一定非要配合CLI使用，但它绝对是Spring应用取得进展的最快方式（你咋不飞上天呢？）。

10.2.1. 手动安装

Spring CLI分发包可以从Spring软件仓库下载：

1. spring-boot-cli-1.4.0.BUILD-SNAPSHOT-bin.zip
2. spring-boot-cli-1.4.0.BUILD-SNAPSHOT-bin.tar.gz

不稳定的snapshot分发包也可以获取到。

下载完成后，解压分发包，根据存档里的INSTALL.txt操作指南进行安装。总的来说，在 `.zip` 文件的 `bin/` 目录下会有一个spring脚本（Windows下是 `spring.bat` ），或使用 `java -jar` 运行 `lib/` 目录下的 `.jar` 文件（该脚本会帮你确保classpath被正确设置）。

10.2.2. 使用SDKMAN安装

SDKMAN（软件开发包管理器）可以对各种各样的二进制SDK包进行版本管理，包括Groovy和Spring Boot CLI。可以从sdkman.io下载SDKMAN，并使用以下命令安装Spring Boot：

```
$ sdk install springboot
$ spring --version
Spring Boot v1.4.0.BUILD-SNAPSHOT
```

如果你正在为CLI开发新的特性，并想轻松获取刚构建的版本，可以使用以下命令：

```
$ sdk install springboot dev /path/to/spring-boot/spring-boot-cl
i/target/spring-boot-cli-1.4.0.BUILD-SNAPSHOT-bin/spring-1.4.0.B
UILD-SNAPSHOT/
$ sdk default springboot dev
$ spring --version
Spring CLI v1.4.0.BUILD-SNAPSHOT
```

这将会安装一个名叫dev的本地spring实例，它指向你的目标构建位置，所以每次你重新构建Spring Boot，spring都会更新为最新的。

你可以通过以下命令来验证：

```
$ sdk ls springboot

================================================================================
Available Springboot Versions
================================================================================
> + dev
* 1.4.0.BUILD-SNAPSHOT

================================================================================
+ - local version
* - installed
> - currently in use
================================================================================
```

10.2.3. 使用OSX Homebrew进行安装

如果你的环境是Mac，并使用Homebrew，想要安装Spring Boot CLI只需以下操作：

```
$ brew tap pivotal/tap
$ brew install springboot
```

Homebrew将把spring安装到 `/usr/local/bin` 下。

注：如果该方案不可用，可能是因为你的brew版本太老了。你只需执行 `brew update` 并重试即可。

10.2.4. 使用MacPorts进行安装

如果你的环境是Mac，并使用MacPorts，想要安装Spring Boot CLI只需以下操作：

```
$ sudo port install spring-boot-cli
```

10.2.5. 命令行实现

Spring Boot CLI启动脚本为BASH和zsh shells提供完整的命令行实现。你可以在任何shell中source脚本（名称也是spring），或将它放到用户或系统范围内的bash初始化脚本里。在Debian系统中，系统级的脚本位于 `/shell-completion/bash` 下，当新的shell启动时该目录下的所有脚本都会被执行。如果想要手动运行脚本，假如你已经安装了SDKMAN，可以使用以下命令：

```
$ . ~/.sdkman/candidates/springboot/current/shell-completion/bash/spring
$ spring <HIT TAB HERE>
  grab  help  jar  run  test  version
```

注：如果你使用Homebrew或MacPorts安装Spring Boot CLI，命令行实现脚本会自动注册到你的shell。

10.2.6. Spring CLI示例快速入门

下面是一个相当简单的web应用，你可以用它测试Spring CLI安装是否成功。创建一个名叫 `app.groovy` 的文件：

```
@RestController
class ThisWillActuallyRun {

    @RequestMapping("/")
    String home() {
        "Hello World!"
    }

}
```

然后只需在shell中运行以下命令：

```
$ spring run app.groovy
```

注：首次运行该应用将会花费一些时间，因为需要下载依赖，后续运行将会快很多。

使用你最喜欢的浏览器打开localhost:8080，然后就可以看到如下输出：

```
Hello World!
```

10.3. 版本升级

如果你正在升级Spring Boot的早期发布版本，那最好查看下project wiki上的"release notes"，你会发现每次发布对应的升级指南和一个"new and noteworthy"特性列表。

想要升级一个已安装的CLI，你需要使用合适的包管理命令，例如 `brew upgrade` ；如果是手动安装CLI，按照standard instructions操作并记得更新你的PATH环境变量以移除任何老的引用。

11. 开发你的第一个Spring Boot应用

我们将使用Java开发一个简单的"Hello World" web应用，以此强调下Spring Boot的一些关键特性。项目采用Maven进行构建，因为大多数IDEs都支持它。

注：spring.io网站包含很多Spring Boot"入门"指南，如果你正在找特定问题的解决方案，可以先去那瞅瞅。你也可以简化下面的步骤，直接从start.spring.io的依赖搜索器选中 `web` starter，这会自动生成一个新的项目结构，然后你就可以happy的敲代码了。具体详情参考文档。

在开始前，你需要打开终端检查下安装的Java和Maven版本是否可用：

```
$ 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; 20
14-08-11T13:58:10-07:00)
Maven home: /Users/user/tools/apache-maven-3.1.1
Java version: 1.7.0_51, vendor: Oracle Corporation
```

注：该示例需要创建单独的文件夹，后续的操作建立在你已创建一个合适的文件夹，并且它是你的“当前目录”。

11.1. 创建POM

让我们以创建一个Maven `pom.xml` 文件作为开始吧，因为 `pom.xml` 是构建项目的处方！打开你最喜欢的文本编辑器，并添加以下内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
tp://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http:/
/maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>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>1.4.1.BUILD-SNAPSHOT</version>
    </parent>

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

    <!-- (you don't need this if you are using a .RELEASE versio
n) -->
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <url>http://repo.spring.io/snapshot</url>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <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>
</project>
```

这样一个可工作的构建就完成了，你可以通过运行 `mvn package` 测试它（暂时忽略"jar将是空的-没有包含任何内容！"的警告）。

注：此刻，你可以将该项目导入到IDE中（大多数现代的Java IDE都包含对Maven的内建支持）。简单起见，我们将继续使用普通的文本编辑器完成该示例。

11.2. 添加classpath依赖

Spring Boot提供很多"Starter POMs"，用来简化添加jars到classpath的操作。示例程序中已经在POM的partent节点使用了 `spring-boot-starter-parent` ，它是一个特殊的starter，提供了有用的Maven默认设置。同时，它也提供一个 `dependency-management` 节点，这样对于期望（"blessed"）的依赖就可以省略version标记了。

其他"Starter POMs"只简单提供开发特定类型应用所需的依赖。由于正在开发web应用，我们将添加 `spring-boot-starter-web` 依赖-但在此之前，让我们先看下目前的依赖：

```
$ mvn dependency:tree
[INFO] com.example:myproject:jar:0.0.1-SNAPSHOT
```

`mvn dependency:tree` 命令可以将项目依赖以树形方式展现出来，你可以看到 `spring-boot-starter-parent` 本身并没有提供依赖。编辑 `pom.xml` ，并在parent节点下添加 `spring-boot-starter-web` 依赖：

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

如果再次运行 `mvn dependency:tree` ，你将看到现在多了一些其他依赖，包括Tomcat web服务器和Spring Boot自身。

11.3. 编写代码

为了完成应用程序，我们需要创建一个单独的Java文件。Maven默认会编译 `src/main/java` 下的源码，所以你需要创建那样的文件结构，并添加一个名为 `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);
    }

}
```

尽管代码不多，但已经发生了很多事情，让我们分步探讨重要的部分吧！

11.3.1. @RestController和@RequestMapping注解

Example类上使用的第一个注解是 `@RestController` ，这被称为构造型（stereotype）注解。它为阅读代码的人提供暗示（这是一个支持REST的控制器），对于Spring，该类扮演了一个特殊角色。在本示例中，我们的类是一个web `@Controller` ，所以当web请求进来时，Spring会考虑是否使用它来处理。

`@RequestMapping` 注解提供路由信息，它告诉Spring任何来自"/"路径的HTTP请求都应该被映射到 `home` 方法。 `@RestController` 注解告诉Spring以字符串的形式渲染结果，并直接返回给调用者。

注： `@RestController` 和 `@RequestMapping` 是Spring MVC中的注解（它们不是Spring Boot的特定部分），具体参考Spring文档的MVC章节。

11.3.2. @EnableAutoConfiguration注解

第二个类级别的注解是 `@EnableAutoConfiguration` ，这个注解告诉Spring Boot根据添加的jar依赖猜测你想如何配置Spring。由于 `spring-boot-starter-web` 添加了Tomcat和Spring MVC，所以auto-configuration将假定你正在开发一个web应用，并对Spring进行相应地设置。

Starters和**Auto-Configuration**：Auto-configuration设计成可以跟"Starters"一起很好的使用，但这两个概念没有直接的联系。你可以自由地挑选starters以外的jar依赖，Spring Boot仍会尽最大努力去自动配置你的应用。

11.3.3. main方法

应用程序的最后部分是main方法，这是一个标准的方法，它遵循Java对于一个应用程序入口点的约定。我们的main方法通过调用 `run` ，将业务委托给了Spring Boot的SpringApplication类。SpringApplication将引导我们的应用，启动Spring，相应地启动被自动配置的Tomcat web服务器。我们需要将 `Example.class` 作为参数传递给 `run` 方法，以此告诉SpringApplication谁是主要的Spring组件，并传递args数组以暴露所有的命令行参数。

11.4. 运行示例

到此，示例应用可以工作了。由于使用了 `spring-boot-starter-parent` POM，这样我们就有了一个非常有用的run目标来启动程序。在项目根目录下输入 `mvn spring-boot:run` 启动应用：

```
$ mvn spring-boot:run

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::  (v1.4.1.BUILD-SNAPSHOT)
....... . . .
....... . . . (log output here)
....... . . .
........ Started Example in 2.222 seconds (JVM running for 6.514
)
```

如果使用浏览器打开localhost:8080，你应该可以看到如下输出：

```
Hello World!
```

点击 `ctrl-c` 温雅地关闭应用程序。

11.5. 创建可执行jar

让我们通过创建一个完全自包含，并可以在生产环境运行的可执行jar来结束示例吧！可执行jars（有时被称为胖jars "fat jars"）是包含编译后的类及代码运行所需依赖jar的存档。

可执行**jars**和**Java**：Java没有提供任何标准方式，用于加载内嵌jar文件（即jar文件中还包含jar文件），这对分发自包含应用来说是个问题。为了解决该问题，很多开发者采用"共享的"jars。共享的jar只是简单地将所有jars的类打包进一个单独的存档，这种方式存在的问题是，很难区分应用程序中使用了哪些库。在多个jars中如果存在相同的文件名（但内容不一样）也会是一个问题。Spring Boot采取一个不同的方式，允许你直接的直接内嵌jars。

为了创建可执行的jar，我们需要将 `spring-boot-maven-plugin` 添加到 `pom.xml` 中，在dependencies节点后面插入以下内容：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

注：`spring-boot-starter-parent` POM包含绑定到repackage目标的 `<executions>` 配置。如果不使用parent POM，你需要自己声明该配置，具体参考插件文档。

保存 `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-exampl
e/target/myproject-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:1.4.1.BUILD-SNAPSHOT:repacka
ge (default) @ myproject ---
[INFO] ------------------------------------------------------------
---------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
---------------
```

如果查看target目录，你应该可以看到 `myproject-0.0.1-SNAPSHOT.jar` ，该文件大概有10Mb。想查看内部结构，可以运行 `jar tvf` ：

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

在该目录下，你应该还能看到一个很小的名为 `myproject-0.0.1-SNAPSHOT.jar.original` 的文件，这是在Spring Boot重新打包前，Maven创建的原始jar文件。

可以使用 `java -jar` 命令运行该应用程序：

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


  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::  (v1.3.0.BUILD-SNAPSHOT)
....... . . .
....... . . . (log output here)
....... . . .
........ Started Example in 2.536 seconds (JVM running for 2.864
)
```

如上所述，点击 `ctrl-c` 可以温雅地退出应用。

12. 接下来阅读什么

希望本章节已为你提供一些Spring Boot的基础部分，并帮你找到开发自己应用的方式。如果你是任务驱动型的开发者，那可以直接跳到spring.io，check out一些入门指南，以解决特定的"使用Spring如何做"的问题；我们也有Spring Boot相关的How-to参考文档。

Spring Boot仓库有大量可以运行的示例，这些示例代码是彼此独立的(运行或使用示例的时候不需要构建其他示例)。

否则，下一步就是阅读 III、使用Spring Boot，如果没耐心，可以跳过该章节，直接阅读 IV、Spring Boot特性。

使用Spring Boot

本章节将详细介绍如何使用Spring Boot，不仅覆盖构建系统，自动配置，如何运行应用等主题，还包括一些Spring Boot的最佳实践。尽管Spring Boot本身没有什么特别的（跟其他一样，它只是另一个你可以使用的库），但仍有一些建议，如果遵循的话将会事半功倍。

如果你刚接触Spring Boot，那最好先阅读上一章节的Getting Started指南。

13. 构建系统

强烈建议你选择一个支持依赖管理，能消费发布到"Maven中央仓库"的artifacts的构建系统，比如Maven或Gradle。使用其他构建系统也是可以的，比如Ant，但它们可能得不到很好的支持。

14. 组织你的代码

Spring Boot不要求使用任何特殊的代码结构，不过，遵循以下的一些最佳实践还是挺有帮助的。

14.1. 使用"default"包

当类没有声明 `package` 时，它被认为处于 `default package` 下。通常不推荐使用 `default package` ，因为对于使用 `@ComponentScan` ， `@EntityScan` 或 `@SpringBootApplication` 注解的Spring Boot应用来说，它会扫描每个jar中的类，这会造成一定的问题。

注 我们建议你遵循Java推荐的包命名规范，使用一个反转的域名（例如 `com.example.project` ）。

14.2. 放置应用的main类

通常建议将应用的main类放到其他类所在包的顶层(root package)，并将 `@EnableAutoConfiguration` 注解到你的main类上，这样就隐式地定义了一个基础的包搜索路径（search package），以搜索某些特定的注解实体（比如@Service，@Component等）。例如，如果你正在编写一个JPA应用，Spring将搜索 `@EnableAutoConfiguration` 注解的类所在包下的 `@Entity` 实体。

采用root package方式，你就可以使用 `@ComponentScan` 注解而不需要指定 `basePackage` 属性，也可以使用 `@SpringBootApplication` 注解，只要将main类放到root package中。

下面是一个典型的结构：

```
com
 +- example
     +- myproject
         +- Application.java
         |
         +- domain
         |   +- Customer.java
         |   +- CustomerRepository.java
         |
         +- service
         |   +- CustomerService.java
         |
         +- web
             +- CustomerController.java
```

`Application.java` 将声明 `main` 方法，还有基本的 `@Configuration`。

```
package com.example.myproject;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfigur
ation;
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);
    }

}
```

15. 配置类

Spring Boot提倡基于Java的配置。尽管你可以使用XML源调用 `SpringApplication.run()` ，不过还是建议你使用 `@Configuration` 类作为主要配置源。通常定义了 `main` 方法的类也是使用 `@Configuration` 注解的一个很好的替补。

注：虽然网络上有很多使用XML配置的Spring示例，但你应该尽可能的使用基于Java的配置，搜索查看 `enable*` 注解就是一个好的开端。

15.1. 导入其他配置类

你不需要将所有的 `@Configuration` 放进一个单独的类， `@Import` 注解可以用来导入其他配置类。另外，你也可以使用 `@ComponentScan` 注解自动收集所有Spring组件，包括 `@Configuration` 类。

15.2. 导入XML配置

如果必须使用XML配置，建议你仍旧从一个 `@Configuration` 类开始，然后使用 `@ImportResource` 注解加载XML配置文件。

16. 自动配置

Spring Boot自动配置（auto-configuration）尝试根据添加的jar依赖自动配置你的Spring应用。例如，如果classpath下存在 `HSQLDB` ，并且你没有手动配置任何数据库连接的beans，那么Spring Boot将自动配置一个内存型（in-memory）数据库。

实现自动配置有两种可选方式，分别是
将 `@EnableAutoConfiguration` 或 `@SpringBootApplication` 注解到 `@Configuration` 类上。

注：你应该只添加一个 `@EnableAutoConfiguration` 注解，通常建议将它添加到主配置类（primary `@Configuration` ）上。

16.1. 逐步替换自动配置

自动配置（Auto-configuration）是非侵入性的，任何时候你都可以定义自己的配置类来替换自动配置的特定部分。例如，如果你添加自己的 `DataSource` bean，默认的内嵌数据库支持将不被考虑。

如果需要查看当前应用启动了哪些自动配置项，你可以在运行应用时打开 `--debug` 开关，这将为核心日志开启debug日志级别，并将自动配置相关的日志输出到控制台。

16.2. 禁用特定的自动配置项

如果发现启用了不想要的自动配置项，你可以使用 `@EnableAutoConfiguration` 注解的exclude属性禁用它们：

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

@Configuration
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.cl
ass})
public class MyConfiguration {
}
```

如果该类不在classpath中，你可以使用该注解的excludeName属性，并指定全限定名来达到相同效果。最后，你可以通过 `spring.autoconfigure.exclude` 属性exclude多个自动配置项（一个自动配置项集合）。

注 通过注解级别或exclude属性都可以定义排除项。

17. Spring Beans和依赖注入

你可以自由地使用任何标准的Spring框架技术去定义beans和它们注入的依赖。简单起见，我们经常使用 `@ComponentScan` 注解搜索beans，并结合 `@Autowired` 构造器注入。

如果遵循以上的建议组织代码结构（将应用的main类放到包的最上层，即root package），那么你就可以添加 `@ComponentScan` 注解而不需要任何参数，所有应用组件（ `@Component` , `@Service` , `@Repository` , `@Controller` 等）都会自动注册成Spring Beans。

下面是一个 `@Service` Bean的示例，它使用构建器注入获取一个需要的 `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;
    }

    // ...
}
```

注 注意使用构建器注入允许 `riskAssessor` 字段被标记为 `final` ，这意味着 `riskAssessor` 后续是不能改变的。

18. 使用@SpringBootApplication注解

很多Spring Boot开发者经常使用 `@Configuration` ， `@EnableAutoConfiguration` ， `@ComponentScan` 注解他们的main类，由于这些注解如此频繁地一块使用（特别是遵循以上最佳实践的时候），Spring Boot就提供了一个方便的 `@SpringBootApplication` 注解作为代替。

`@SpringBootApplication` 注解等价于以默认属性使用 `@Configuration` ， `@EnableAutoConfiguration` 和 `@ComponentScan` ：

```
package com.example.myproject;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplicat
ion;

@SpringBootApplication // same as @Configuration @EnableAutoConf
iguration @ComponentScan
public class Application {

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

}
```

注 `@SpringBootApplication` 注解也提供了用于自定义 `@EnableAutoConfiguration` 和 `@ComponentScan` 属性的别名（aliases）。

19. 运行应用程序

将应用打包成jar，并使用内嵌HTTP服务器的一个最大好处是，你可以像其他方式那样运行你的应用程序。调试Spring Boot应用也很简单，你都不需要任何特殊IDE插件或扩展！

注：本章节只覆盖基于jar的打包，如果选择将应用打包成war文件，你最好参考相关的服务器和IDE文档。

19.1. 从IDE中运行

你可以从IDE中运行Spring Boot应用，就像一个简单的Java应用，但首先需要导入项目。导入步骤取决于你的IDE和构建系统，大多数IDEs能够直接导入Maven项目，例如Eclipse用户可以选择 `File` 菜单的 `Import…` --> `Existing Maven Projects` 。

如果不能直接将项目导入IDE，你可以使用构建系统生成IDE的元数据。Maven有针对Eclipse和IDEA的插件；Gradle为各种IDEs提供插件。

注 如果意外地多次运行一个web应用，你将看到一个"端口已被占用"的错误。STS用户可以使用 `Relaunch` 而不是 `Run` 按钮，以确保任何存在的实例是关闭的。

19.2. 作为一个打包后的应用运行

如果使用Spring Boot Maven或Gradle插件创建一个可执行jar，你可以使用 `java -jar` 运行应用。例如：

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

Spring Boot支持以远程调试模式运行一个打包的应用，下面的命令可以为应用关联一个调试器：

```
$ java -Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=80
00,suspend=n \
       -jar target/myproject-0.0.1-SNAPSHOT.jar
```

19.3. 使用Maven插件运行

Spring Boot Maven插件包含一个 `run` 目标，可用来快速编译和运行应用程序，并且跟在IDE运行一样支持热加载。

```
$ mvn spring-boot:run
```

你可以使用一些有用的操作系统环境变量：

```
$ export MAVEN_OPTS=-Xmx1024m -XX:MaxPermSize=128M
```

19.4. 使用Gradle插件运行

Spring Boot Gradle插件也包含一个 `bootRun` 任务，可用来运行你的应用程序。无论你何时import `spring-boot-gradle-plugin` ， `bootRun` 任务总会被添加进去。

```
$ gradle bootRun
```

你可能想使用一些有用的操作系统环境变量：

```
$ export JAVA_OPTS=-Xmx1024m -XX:MaxPermSize=128M
```

19.5. 热交换

由于Spring Boot应用只是普通的Java应用，所以JVM热交换（hot-swapping）也能开箱即用。不过JVM热交换能替换的字节码有限制，想要更彻底的解决方案可以使用Spring Loaded项目或JRebel。`spring-boot-devtools`模块也支持应用快速重启(restart)。

详情参考下面的Chapter 20, Developer tools和“How-to”章节。

Spring Boot特性

本章节将深入详细的介绍Spring Boot，通过阅读本节你可以了解到需要使用和定制的核心特性。如果没做好准备，你可以先阅读Part II. Getting started和Part III, “Using Spring Boot” 章节，以对Spring Boot有个良好的基本认识。

Spring Boot执行器：Production-ready特性

Spring Boot包含很多其他特性，可用来帮你监控和管理发布到生产环境的应用。你可以选择使用HTTP端点，JMX，甚至通过远程shell（SSH或Telnet）来管理和监控应用。审计（Auditing），健康（health）和数据采集（metrics gathering）会自动应用到你的应用。

Actuator HTTP端点只能用在基于Spring MVC的应用，特别地，它不能跟Jersey一块使用，除非你也启用Spring MVC。

Spring Boot CLI

Spring Boot CLI是一个命令行工具，如果想使用Spring进行快速开发可以使用它。它允许你运行Groovy脚本，这意味着你可以使用熟悉的类Java语法，并且没有那么多的模板代码。你可以通过Spring Boot CLI启动新项目，或为它编写命令。

构建工具插件

Spring Boot为Maven和Gradle提供构建工具插件，该插件提供各种各样的特性，包括打包可执行jars。本章节提供关于插件的更多详情及用于扩展一个不支持的构建系统所需的帮助信息。如果你是刚刚开始，那可能需要先阅读Part III, “Using Spring Boot”章节的“Chapter 13, Build systems”。

How-to指南

本章节将回答一些常见的"我该怎么做"类型的问题，这些问题在我们使用Spring Boot时经常遇到。这虽然不是一个详尽的列表，但它覆盖了很多方面。

如果遇到一个特殊的我们没有覆盖的问题，你可以查看stackoverflow.com，看是否已经有人给出了答案；这也是一个很好的提新问题的地方（请使用 `spring-boot` 标签）。

我们也乐意扩展本章节；如果想添加一个'how-to'，你可以给我们发一个pull请求。

73.3. 创建其他的可执行JAR

如果你想将自己的项目以library jar的形式被其他项目依赖，并且需要它是一个可执行版本（例如demo），你需要使用略微不同的方式来配置该构建。

对于Maven来说，正常的JAR插件和Spring Boot插件都有一个'classifier'，你可以添加它来创建另外的JAR。示例如下（使用Spring Boot Starter Parent管理插件版本，其他配置采用默认设置）：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <classifier>exec</classifier>
            </configuration>
        </plugin>
    </plugins>
</build>
```

上述配置会产生两个jars，默认的一个和使用带有classifier 'exec'的Boot插件构建的可执行的一个。

对于Gradle用户来说，步骤类似。示例如下：

```
bootRepackage  {
    classifier = 'exec'
}
```

73.9.2. JTA API兼容性

Java事务API自身并不要求Java 7，而是官方的API jar包含的已构建类要求Java 7。如果你正在使用JTA，那么你需要使用能够在Java 6工作的构建版本替换官方的JTA 1.2 API jar。为了完成该操作，你需要排除任何对 `javax.transaction:javax.transaction-api` 的传递依赖，并使用 `org.jboss.spec.javax.transaction:jboss-transaction-api_1.2_spec:1.0.0.Final` 依赖替换它们。

X.附录

附录A. 常见应用属性

你可以在 `application.properties/application.yml` 文件内部或通过命令行开关来指定各种属性。本章节提供了一个常见Spring Boot属性的列表及使用这些属性的底层类的引用。

注：属性可以来自classpath下的其他jar文件中，所以你不应该把它当成详尽的列表。定义你自己的属性也是相当合法的。

注：示例文件只是一个指导。不要拷贝/粘贴整个内容到你的应用，而是只提取你需要的属性。

```
# ===================================================================
=====
# 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
# ----------------------------------------

# SPRING CONFIG (ConfigFileApplicationListener)
spring.config.name= # config file name (default to 'application'
)
spring.config.location= # location of config file

# PROFILES
spring.profiles.active= # comma list of active profiles
spring.profiles.include= # unconditionally activate the specifie
d comma separated profiles

# APPLICATION SETTINGS (SpringApplication)
spring.main.sources=
spring.main.web-environment= # detect by default
```

```
spring.main.show-banner=true
spring.main....= # see class for all properties

# LOGGING
logging.path=/var/logs
logging.file=myapp.log
logging.config= # location of config file (default classpath:log
back.xml for logback)
logging.level.*= # levels for loggers, e.g. "logging.level.org.s
pringframework=DEBUG" (TRACE, DEBUG, INFO, WARN, ERROR, FATAL, O
FF)

# IDENTITY (ContextIdApplicationContextInitializer)
spring.application.name=
spring.application.index=

# EMBEDDED SERVER CONFIGURATION (ServerProperties)
server.port=8080
server.address= # bind to a specific NIC
server.session-timeout= # session timeout in seconds
server.context-parameters.*= # Servlet context init parameters,
e.g. server.context-parameters.a=alpha
server.context-path= # the context path, defaults to '/'
server.servlet-path= # the servlet path, defaults to '/'
server.ssl.enabled=true # if SSL support is enabled
server.ssl.client-auth= # want or need
server.ssl.key-alias=
server.ssl.ciphers= # supported SSL ciphers
server.ssl.key-password=
server.ssl.key-store=
server.ssl.key-store-password=
server.ssl.key-store-provider=
server.ssl.key-store-type=
server.ssl.protocol=TLS
server.ssl.trust-store=
server.ssl.trust-store-password=
server.ssl.trust-store-provider=
server.ssl.trust-store-type=
server.tomcat.access-log-pattern= # log pattern of the access lo
g
```

```
server.tomcat.access-log-enabled=false # is access logging enabl
ed
server.tomcat.compression=off # is compression enabled (off, on,
 or an integer content length limit)
server.tomcat.compressable-mime-types=text/html,text/xml,text/pl
ain # comma-separated list of mime types that Tomcat will compre
ss
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} # regular expressio
n matching trusted IP addresses
server.tomcat.protocol-header=x-forwarded-proto # front end prox
y forward header
server.tomcat.port-header= # front end proxy port header
server.tomcat.remote-ip-header=x-forwarded-for
server.tomcat.basedir=/tmp # base dir (usually not needed, defau
lts to tmp)
server.tomcat.background-processor-delay=30; # in seconds
server.tomcat.max-http-header-size= # maximum size in bytes of t
he HTTP message header
server.tomcat.max-threads = 0 # number of threads in protocol ha
ndler
server.tomcat.uri-encoding = UTF-8 # character encoding to use f
or URL decoding

# SPRING MVC (WebMvcProperties)
spring.mvc.locale= # set fixed locale, e.g. en_UK
spring.mvc.date-format= # set fixed date format, e.g. dd/MM/yyyy
spring.mvc.favicon.enabled=true
spring.mvc.message-codes-resolver-format= # PREFIX_ERROR_CODE /
POSTFIX_ERROR_CODE
spring.mvc.ignore-default-model-on-redirect=true # If the the co
ntent of the "default" model should be ignored redirects
spring.view.prefix= # MVC view prefix
spring.view.suffix= # ... and suffix

# SPRING RESOURCES HANDLING (ResourceProperties)
spring.resources.cache-period= # cache timeouts in headers sent
```

```
to browser
spring.resources.add-mappings=true # if default mappings should
be added

# MULTIPART (MultipartProperties)
multipart.enabled=true
multipart.file-size-threshold=0 # Threshold after which files wi
ll be written to disk.
multipart.location= # Intermediate location of uploaded files.
multipart.max-file-size=1Mb # Max file size.
multipart.max-request-size=10Mb # Max request size.

# SPRING HATEOAS (HateoasProperties)
spring.hateoas.apply-to-primary-object-mapper=true # if the prim
ary mapper should also be configured

# HTTP encoding (HttpEncodingProperties)
spring.http.encoding.charset=UTF-8 # the encoding of HTTP reques
ts/responses
spring.http.encoding.enabled=true # enable http encoding support
spring.http.encoding.force=true # force the configured encoding

# HTTP message conversion
spring.http.converters.preferred-json-mapper= # the preferred JS
ON mapper to use for HTTP message conversion. Set to "gson" to f
orce the use of Gson when both it and Jackson are on the classpa
th.

# HTTP response compression (GzipFilterProperties)
spring.http.gzip.buffer-size= # size of the output buffer in byt
es
spring.http.gzip.deflate-compression-level= # the level used for
 deflate compression (0-9)
spring.http.gzip.deflate-no-wrap= # noWrap setting for deflate c
ompression (true or false)
spring.http.gzip.enabled=true # enable gzip filter support
spring.http.gzip.excluded-agents= # comma-separated list of user
 agents to exclude from compression
spring.http.gzip.excluded-agent-patterns= # comma-separated list
 of regular expression patterns to control user agents excluded
```

```
from compression
spring.http.gzip.excluded-paths= # comma-separated list of paths
 to exclude from compression
spring.http.gzip.excluded-path-patterns= # comma-separated list
of regular expression patterns to control the paths that are exc
luded from compression
spring.http.gzip.methods= # comma-separated list of HTTP methods
 for which compression is enabled
spring.http.gzip.mime-types= # comma-separated list of MIME type
s which should be compressed
spring.http.gzip.min-gzip-size= # minimum content length require
d for compression to occur
spring.http.gzip.vary= # Vary header to be sent on responses tha
t may be compressed

# JACKSON (JacksonProperties)
spring.jackson.date-format= # Date format string (e.g. yyyy-MM-d
d HH:mm:ss), or a fully-qualified date format class name (e.g. c
om.fasterxml.jackson.databind.util.ISO8601DateFormat)
spring.jackson.property-naming-strategy= # One of the constants
on Jackson's PropertyNamingStrategy (e.g. CAMEL_CASE_TO_LOWER_CA
SE_WITH_UNDERSCORES) or the fully-qualified class name of a Prop
ertyNamingStrategy subclass
spring.jackson.deserialization.*= # see Jackson's Deserializatio
nFeature
spring.jackson.generator.*= # see Jackson's JsonGenerator.Featur
e
spring.jackson.mapper.*= # see Jackson's MapperFeature
spring.jackson.parser.*= # see Jackson's JsonParser.Feature
spring.jackson.serialization.*= # see Jackson's SerializationFea
ture
spring.jackson.serialization-inclusion= # Controls the inclusion
 of properties during serialization (see Jackson's JsonInclude.I
nclude)

# THYMELEAF (ThymeleafAutoConfiguration)
spring.thymeleaf.check-template-location=true
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.excluded-view-names= # comma-separated list of
view names that should be excluded from resolution
```

```
spring.thymeleaf.view-names= # comma-separated list of view name
s that can be resolved
spring.thymeleaf.suffix=.html
spring.thymeleaf.mode=HTML5
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.content-type=text/html # ;charset=<encoding> is
 added
spring.thymeleaf.cache=true # set to false for hot refresh

# FREEMARKER (FreeMarkerAutoConfiguration)
spring.freemarker.allow-request-override=false
spring.freemarker.cache=true
spring.freemarker.check-template-location=true
spring.freemarker.charset=UTF-8
spring.freemarker.content-type=text/html
spring.freemarker.expose-request-attributes=false
spring.freemarker.expose-session-attributes=false
spring.freemarker.expose-spring-macro-helpers=false
spring.freemarker.prefix=
spring.freemarker.request-context-attribute=
spring.freemarker.settings.*=
spring.freemarker.suffix=.ftl
spring.freemarker.template-loader-path=classpath:/templates/ # c
omma-separated list
spring.freemarker.view-names= # whitelist of view names that can
 be resolved

# GROOVY TEMPLATES (GroovyTemplateAutoConfiguration)
spring.groovy.template.cache=true
spring.groovy.template.charset=UTF-8
spring.groovy.template.configuration.*= # See Groovy's TemplateC
onfiguration
spring.groovy.template.content-type=text/html
spring.groovy.template.prefix=classpath:/templates/
spring.groovy.template.suffix=.tpl
spring.groovy.template.view-names= # whitelist of view names tha
t can be resolved

# VELOCITY TEMPLATES (VelocityAutoConfiguration)
spring.velocity.allow-request-override=false
```

```
spring.velocity.cache=true
spring.velocity.check-template-location=true
spring.velocity.charset=UTF-8
spring.velocity.content-type=text/html
spring.velocity.date-tool-attribute=
spring.velocity.expose-request-attributes=false
spring.velocity.expose-session-attributes=false
spring.velocity.expose-spring-macro-helpers=false
spring.velocity.number-tool-attribute=
spring.velocity.prefer-file-system-access=true # prefer file sys
tem access for template loading
spring.velocity.prefix=
spring.velocity.properties.*=
spring.velocity.request-context-attribute=
spring.velocity.resource-loader-path=classpath:/templates/
spring.velocity.suffix=.vm
spring.velocity.toolbox-config-location= # velocity Toolbox conf
ig location, for example "/WEB-INF/toolbox.xml"
spring.velocity.view-names= # whitelist of view names that can b
e resolved

# JERSEY (JerseyProperties)
spring.jersey.type=servlet # servlet or filter
spring.jersey.init= # init params
spring.jersey.filter.order=

# INTERNATIONALIZATION (MessageSourceAutoConfiguration)
spring.messages.basename=messages
spring.messages.cache-seconds=-1
spring.messages.encoding=UTF-8



# SECURITY (SecurityProperties)
security.user.name=user # login username
security.user.password= # login password
security.user.role=USER # role assigned to the user
security.require-ssl=false # advanced settings ...
security.enable-csrf=false
security.basic.enabled=true
security.basic.realm=Spring
```

```
security.basic.path= # /**
security.basic.authorize-mode= # ROLE, AUTHENTICATED, NONE
security.filter-order=0
security.headers.xss=false
security.headers.cache=false
security.headers.frame=false
security.headers.content-type=false
security.headers.hsts=all # none / domain / all
security.sessions=stateless # always / never / if_required / sta
teless
security.ignored= # Comma-separated list of paths to exclude fro
m the default secured paths

# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties
)
spring.datasource.name= # name of the data source
spring.datasource.initialize=true # populate using data.sql
spring.datasource.schema= # a schema (DDL) script resource refer
ence
spring.datasource.data= # a data (DML) script resource reference
spring.datasource.sql-script-encoding= # a charset for reading S
QL scripts
spring.datasource.platform= # the platform to use in the schema
resource (schema-${platform}.sql)
spring.datasource.continue-on-error=false # continue even if can
't be initialized
spring.datasource.separator=; # statement separator in SQL initi
alization scripts
spring.datasource.driver-class-name= # JDBC Settings...
spring.datasource.url=
spring.datasource.username=
spring.datasource.password=
spring.datasource.jndi-name= # For JNDI lookup (class, url, user
name & password are ignored when set)
spring.datasource.max-active=100 # Advanced configuration...
spring.datasource.max-idle=8
spring.datasource.min-idle=8
spring.datasource.initial-size=10
spring.datasource.validation-query=
spring.datasource.test-on-borrow=false
```

```
spring.datasource.test-on-return=false
spring.datasource.test-while-idle=
spring.datasource.time-between-eviction-runs-millis=
spring.datasource.min-evictable-idle-time-millis=
spring.datasource.max-wait=
spring.datasource.jmx-enabled=false # Export JMX MBeans (if supp
orted)

# DAO (PersistenceExceptionTranslationAutoConfiguration)
spring.dao.exceptiontranslation.enabled=true

# MONGODB (MongoProperties)
spring.data.mongodb.host= # the db host
spring.data.mongodb.port=27017 # the connection port (defaults t
o 27107)
spring.data.mongodb.uri=mongodb://localhost/test # connection UR
L
spring.data.mongodb.database=
spring.data.mongodb.authentication-database=
spring.data.mongodb.grid-fs-database=
spring.data.mongodb.username=
spring.data.mongodb.password=
spring.data.mongodb.repositories.enabled=true # if spring data r
epository support is enabled

# JPA (JpaBaseConfiguration, HibernateJpaAutoConfiguration)
spring.jpa.properties.*= # properties to set on the JPA connecti
on
spring.jpa.open-in-view=true
spring.jpa.show-sql=true
spring.jpa.database-platform=
spring.jpa.database=
spring.jpa.generate-ddl=false # ignored by Hibernate, might be u
seful for other vendors
spring.jpa.hibernate.naming-strategy= # naming classname
spring.jpa.hibernate.ddl-auto= # defaults to create-drop for emb
edded dbs
spring.data.jpa.repositories.enabled=true # if spring data repos
itory support is enabled
```

```
# JTA (JtaAutoConfiguration)
spring.jta.log-dir= # transaction log dir
spring.jta.*= # technology specific configuration

# ATOMIKOS
spring.jta.atomikos.connectionfactory.borrow-connection-timeout=
30 # Timeout, in seconds, for borrowing connections from the poo
l
spring.jta.atomikos.connectionfactory.ignore-session-transacted-
flag=true # Whether or not to ignore the transacted flag when cr
eating session
spring.jta.atomikos.connectionfactory.local-transaction-mode=fal
se # 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 thr
ead
spring.jta.atomikos.connectionfactory.max-idle-time=60 # The tim
e, in seconds, after which connections are cleaned up from the p
ool
spring.jta.atomikos.connectionfactory.max-lifetime=0 # The time,
 in seconds, that a connection can be pooled for before being de
stroyed. 0 denotes no limit.
spring.jta.atomikos.connectionfactory.max-pool-size=1 # The maxi
mum size of the pool
spring.jta.atomikos.connectionfactory.min-pool-size=1 # The mini
mum size of the pool
spring.jta.atomikos.connectionfactory.reap-timeout=0 # The reap
timeout, in seconds, for borrowed connections. 0 denotes no limi
t.
spring.jta.atomikos.connectionfactory.unique-resource-name=jmsCo
nnectionFactory # The unique name used to identify the resource
during recovery
spring.jta.atomikos.datasource.borrow-connection-timeout=30 # Ti
meout, in seconds, for borrowing connections from the pool
spring.jta.atomikos.datasource.default-isolation-level= # Defaul
t isolation level of connections provided by the pool
spring.jta.atomikos.datasource.login-timeout= # Timeout, in seco
nds, for establishing a database connection
spring.jta.atomikos.datasource.maintenance-interval=60 # The tim
e, in seconds, between runs of the pool's maintenance thread
```

```
spring.jta.atomikos.datasource.max-idle-time=60 # The time, in s
econds, after which connections are cleaned up from the pool
spring.jta.atomikos.datasource.max-lifetime=0 # The time, in sec
onds, that a connection can be pooled for before being destroyed
. 0 denotes no limit.
spring.jta.atomikos.datasource.max-pool-size=1 # The maximum siz
e of the pool
spring.jta.atomikos.datasource.min-pool-size=1 # The minimum siz
e 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 statem
ent 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

# BITRONIX
spring.jta.bitronix.connectionfactory.acquire-increment=1 # Numb
er of connections to create when growing the pool
spring.jta.bitronix.connectionfactory.acquisition-interval=1 # T
ime, in seconds, to wait before trying to acquire a connection a
gain after an invalid connection was acquired
spring.jta.bitronix.connectionfactory.acquisition-timeout=30 # T
imeout, in seconds, for acquiring connections from the pool
spring.jta.bitronix.connectionfactory.allow-local-transactions=t
rue # 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-enable
d=true # Whether or not resources should be enlisted and deliste
d 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=t
rue # Whether or not the provider can run many transactions on t
he same connection and supports transaction interleaving
spring.jta.bitronix.connectionfactory.ignore-recovery-failures=f
alse # Whether or not recovery failures should be ignored
```

```
spring.jta.bitronix.connectionfactory.max-idle-time=60 # The tim
e, in seconds, after which connections are cleaned up from the p
ool
spring.jta.bitronix.connectionfactory.max-pool-size=10 # The max
imum size of the pool. 0 denotes no limit
spring.jta.bitronix.connectionfactory.min-pool-size=0 # The mini
mum size of the pool
spring.jta.bitronix.connectionfactory.password= # The password t
o use to connect to the JMS provider
spring.jta.bitronix.connectionfactory.share-transaction-connecti
ons=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 # Wh
ether or not connections should be tested when acquired from the
 pool
spring.jta.bitronix.connectionfactory.two-pc-ordering-position=1
 # The postion that this resource should take during two-phase c
ommit (always first is Integer.MIN_VALUE, always last is Integer
.MAX_VALUE)
spring.jta.bitronix.connectionfactory.unique-name=jmsConnectionF
actory # The unique name used to identify the resource during re
covery
spring.jta.bitronix.connectionfactory.use-tm-join=true Whether o
r 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 c
onnections 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 af
ter 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 # W
hether 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 XAR
esource when it is enlisted
spring.jta.bitronix.datasource.automatic-enlisting-enabled=true
```

```
# Whether or not resources should be enlisted and delisted autom
atically
spring.jta.bitronix.datasource.cursor-holdability= # The default
 cursor holdability for connections
spring.jta.bitronix.datasource.defer-connection-release=true # W
hether or not the database can run many transactions on the same
 connection and supports transaction interleaving
spring.jta.bitronix.datasource.enable-jdbc4-connection-test # Wh
ether or not Connection.isValid() is called when acquiring a con
nection 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 is
olation level for connections
spring.jta.bitronix.datasource.local-auto-commit # The default a
uto-commit mode for local transactions
spring.jta.bitronix.datasource.login-timeout= # Timeout, in seco
nds, for establishing a database connection
spring.jta.bitronix.datasource.max-idle-time=60 # The time, in s
econds, after which connections are cleaned up from the pool
spring.jta.bitronix.datasource.max-pool-size=10 # The maximum si
ze of the pool. 0 denotes no limit
spring.jta.bitronix.datasource.min-pool-size=0 # The minimum siz
e 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=fal
se #  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 stateme
nt 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_V
ALUE)
spring.jta.bitronix.datasource.unique-name=dataSource # The uniq
ue name used to identify the resource during recovery
spring.jta.bitronix.datasource.use-tm-join=true Whether or not T
MJOIN should be used when starting XAResources
```

```
# SOLR (SolrProperties)
spring.data.solr.host=http://127.0.0.1:8983/solr
spring.data.solr.zk-host=
spring.data.solr.repositories.enabled=true # if spring data repo
sitory support is enabled

# ELASTICSEARCH (ElasticsearchProperties)
spring.data.elasticsearch.cluster-name= # The cluster name (defa
ults to elasticsearch)
spring.data.elasticsearch.cluster-nodes= # The address(es) of th
e server node (comma-separated; if not specified starts a client
 node)
spring.data.elasticsearch.properties.*= # Additional properties
used to configure the client
spring.data.elasticsearch.repositories.enabled=true # if spring
data repository support is enabled

# DATA REST (RepositoryRestConfiguration)
spring.data.rest.base-uri= # base URI against which the exporter
 should calculate its links

# FLYWAY (FlywayProperties)
flyway.*= # Any public property available on the auto-configured
 `Flyway` object
flyway.check-location=false # check that migration scripts locat
ion exists
flyway.locations=classpath:db/migration # locations of migration
s scripts
flyway.schemas= # schemas to update
flyway.init-version= 1 # version to start migration
flyway.init-sqls= # SQL statements to execute to initialize a co
nnection immediately after obtaining it
flyway.sql-migration-prefix=V
flyway.sql-migration-suffix=.sql
flyway.enabled=true
flyway.url= # JDBC url if you want Flyway to create its own Data
Source
flyway.user= # JDBC username if you want Flyway to create its ow
n DataSource
```

```
flyway.password= # JDBC password if you want Flyway to create it
s own DataSource

# LIQUIBASE (LiquibaseProperties)
liquibase.change-log=classpath:/db/changelog/db.changelog-master
.yaml
liquibase.check-change-log-location=true # check the change log
location exists
liquibase.contexts= # runtime contexts to use
liquibase.default-schema= # default database schema to use
liquibase.drop-first=false
liquibase.enabled=true
liquibase.url= # specific JDBC url (if not set the default datas
ource is used)
liquibase.user= # user name for liquibase.url
liquibase.password= # password for liquibase.url

# JMX
spring.jmx.enabled=true # Expose MBeans from Spring

# RABBIT (RabbitProperties)
spring.rabbitmq.host= # connection host
spring.rabbitmq.port= # connection port
spring.rabbitmq.addresses= # connection addresses (e.g. myhost:9
999,otherhost:1111)
spring.rabbitmq.username= # login user
spring.rabbitmq.password= # login password
spring.rabbitmq.virtual-host=
spring.rabbitmq.dynamic=

# REDIS (RedisProperties)
spring.redis.database= # database name
spring.redis.host=localhost # server host
spring.redis.password= # server password
spring.redis.port=6379 # connection port
spring.redis.pool.max-idle=8 # pool settings ...
spring.redis.pool.min-idle=0
spring.redis.pool.max-active=8
spring.redis.pool.max-wait=-1
spring.redis.sentinel.master= # name of Redis server
```

```
spring.redis.sentinel.nodes= # comma-separated list of host:port
 pairs

# ACTIVEMQ (ActiveMQProperties)
spring.activemq.broker-url=tcp://localhost:61616 # connection UR
L
spring.activemq.user=
spring.activemq.password=
spring.activemq.in-memory=true # broker kind to create if no bro
ker-url is specified
spring.activemq.pooled=false

# HornetQ (HornetQProperties)
spring.hornetq.mode= # connection mode (native, embedded)
spring.hornetq.host=localhost # hornetQ host (native mode)
spring.hornetq.port=5445 # hornetQ port (native mode)
spring.hornetq.embedded.enabled=true # if the embedded server is
 enabled (needs hornetq-jms-server.jar)
spring.hornetq.embedded.server-id= # auto-generated id of the em
bedded server (integer)
spring.hornetq.embedded.persistent=false # message persistence
spring.hornetq.embedded.data-directory= # location of data conte
nt (when persistence is enabled)
spring.hornetq.embedded.queues= # comma-separated queues to crea
te on startup
spring.hornetq.embedded.topics= # comma-separated topics to crea
te on startup
spring.hornetq.embedded.cluster-password= # customer password (r
andomly generated by default)

# JMS (JmsProperties)
spring.jms.jndi-name= # JNDI location of a JMS ConnectionFactory
spring.jms.pub-sub-domain= # false for queue (default), true for
 topic

# Email (MailProperties)
spring.mail.host=smtp.acme.org # mail server host
spring.mail.port= # mail server port
spring.mail.username=
spring.mail.password=
```

```
spring.mail.default-encoding=UTF-8 # encoding to use for MimeMes
sages
spring.mail.properties.*= # properties to set on the JavaMail se
ssion

# SPRING BATCH (BatchDatabaseInitializer)
spring.batch.job.names=job1,job2
spring.batch.job.enabled=true
spring.batch.initializer.enabled=true
spring.batch.schema= # batch schema to load

# SPRING CACHE (CacheProperties)
spring.cache.type= # generic, ehcache, hazelcast, jcache, redis,
 guava, simple, none
spring.cache.config= #
spring.cache.cache-names= # cache names to create on startup
spring.cache.jcache.provider= # fully qualified name of the Cach
ingProvider implementation to use
spring.cache.guava.spec= # guava specs

# AOP
spring.aop.auto=
spring.aop.proxy-target-class=

# FILE ENCODING (FileEncodingApplicationListener)
spring.mandatory-file-encoding= # Expected character encoding th
e application must use

# SPRING SOCIAL (SocialWebAutoConfiguration)
spring.social.auto-connection-views=true # Set to true for defau
lt connection views or false if you provide your own

# SPRING SOCIAL FACEBOOK (FacebookAutoConfiguration)
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)
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)
spring.social.twitter.app-id= # your application's Twitter App I
D
spring.social.twitter.app-secret= # your application's Twitter A
pp Secret

# SPRING MOBILE SITE PREFERENCE (SitePreferenceAutoConfiguration
)
spring.mobile.sitepreference.enabled=true # enabled by default

# SPRING MOBILE DEVICE VIEWS (DeviceDelegatingViewResolverAutoCo
nfiguration)
spring.mobile.devicedelegatingviewresolver.enabled=true # disabl
ed by default
spring.mobile.devicedelegatingviewresolver.normal-prefix=
spring.mobile.devicedelegatingviewresolver.normal-suffix=
spring.mobile.devicedelegatingviewresolver.mobile-prefix=mobile/
spring.mobile.devicedelegatingviewresolver.mobile-suffix=
spring.mobile.devicedelegatingviewresolver.tablet-prefix=tablet/
spring.mobile.devicedelegatingviewresolver.tablet-suffix=

# ----------------------------------------
# ACTUATOR PROPERTIES
# ----------------------------------------

# MANAGEMENT HTTP SERVER (ManagementServerProperties)
management.port= # defaults to 'server.port'
management.address= # bind to a specific NIC
management.context-path= # default to '/'
management.add-application-context-header= # default to true
management.security.enabled=true # enable security
management.security.role=ADMIN # role required to access the man
agement endpoint
management.security.sessions=stateless # session creating policy
 to use (always, never, if_required, stateless)
```

```
# PID FILE (ApplicationPidFileWriter)
spring.pidfile= # Location of the PID file to write

# ENDPOINTS (AbstractEndpoint subclasses)
endpoints.autoconfig.id=autoconfig
endpoints.autoconfig.sensitive=true
endpoints.autoconfig.enabled=true
endpoints.beans.id=beans
endpoints.beans.sensitive=true
endpoints.beans.enabled=true
endpoints.configprops.id=configprops
endpoints.configprops.sensitive=true
endpoints.configprops.enabled=true
endpoints.configprops.keys-to-sanitize=password,secret,key # suf
fix or regex
endpoints.dump.id=dump
endpoints.dump.sensitive=true
endpoints.dump.enabled=true
endpoints.env.id=env
endpoints.env.sensitive=true
endpoints.env.enabled=true
endpoints.env.keys-to-sanitize=password,secret,key # suffix or r
egex
endpoints.health.id=health
endpoints.health.sensitive=true
endpoints.health.enabled=true
endpoints.health.mapping.*= # mapping of health statuses to Http
Status codes
endpoints.health.time-to-live=1000
endpoints.info.id=info
endpoints.info.sensitive=false
endpoints.info.enabled=true
endpoints.mappings.enabled=true
endpoints.mappings.id=mappings
endpoints.mappings.sensitive=true
endpoints.metrics.id=metrics
endpoints.metrics.sensitive=true
endpoints.metrics.enabled=true
endpoints.shutdown.id=shutdown
endpoints.shutdown.sensitive=true
```

```
endpoints.shutdown.enabled=false
endpoints.trace.id=trace
endpoints.trace.sensitive=true
endpoints.trace.enabled=true

# HEALTH INDICATORS (previously health.*)
management.health.db.enabled=true
management.health.elasticsearch.enabled=true
management.health.elasticsearch.response-timeout=100 # the time,
 in milliseconds, to wait for a response from the cluster
management.health.diskspace.enabled=true
management.health.diskspace.path=.
management.health.diskspace.threshold=10485760
management.health.mongo.enabled=true
management.health.rabbit.enabled=true
management.health.redis.enabled=true
management.health.solr.enabled=true
management.health.status.order=DOWN, OUT_OF_SERVICE, UNKNOWN, UP

# MVC ONLY ENDPOINTS
endpoints.jolokia.path=jolokia
endpoints.jolokia.sensitive=true
endpoints.jolokia.enabled=true # when using Jolokia

# JMX ENDPOINT (EndpointMBeanExportProperties)
endpoints.jmx.enabled=true
endpoints.jmx.domain= # the JMX domain, defaults to 'org.springb
oot'
endpoints.jmx.unique-names=false
endpoints.jmx.static-names=

# JOLOKIA (JolokiaProperties)
jolokia.config.*= # See Jolokia manual

# REMOTE SHELL
shell.auth=simple # jaas, key, simple, spring
shell.command-refresh-interval=-1
shell.command-path-patterns= # classpath*:/commands/**, classpat
h*:/crash/commands/**
shell.config-path-patterns= # classpath*:/crash/*
```

```
shell.disabled-commands=jpa*,jdbc*,jndi* # comma-separated list
of commands to disable
shell.disabled-plugins=false # don't expose plugins
shell.ssh.enabled= # ssh settings ...
shell.ssh.key-path=
shell.ssh.port=
shell.telnet.enabled= # telnet settings ...
shell.telnet.port=
shell.auth.jaas.domain= # authentication settings ...
shell.auth.key.path=
shell.auth.simple.user.name=
shell.auth.simple.user.password=
shell.auth.spring.roles=

# SENDGRID (SendGridAutoConfiguration)
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

# GIT INFO
spring.git.properties= # resource ref to generated git info prop
erties file
```

附录B. 配置元数据

Spring Boot jars包含元数据文件，它们提供了所有支持的配置属性详情。这些文件设计用于让IDE开发者能够为使用application.properties或application.yml文件的用户提供上下文帮助及代码完成功能。

主要的元数据文件是在编译器通过处理所有被 `@ConfigurationProperties` 注解的节点来自动生成的。

附录B.1. 元数据格式

配置元数据位于jars文件中的 `META-INF/spring-configuration-metadata.json` ，它们使用一个具有"groups"或"properties"分类节点的简单JSON格式：

```
{"groups": [
    {
        "name": "server",
        "type": "org.springframework.boot.autoconfigure.web.Serv
erProperties",
        "sourceType": "org.springframework.boot.autoconfigure.we
b.ServerProperties"
    }
    ...
],"properties": [
    {
        "name": "server.port",
        "type": "java.lang.Integer",
        "sourceType": "org.springframework.boot.autoconfigure.we
b.ServerProperties"
    },
    {
        "name": "server.servlet-path",
        "type": "java.lang.String",
        "sourceType": "org.springframework.boot.autoconfigure.we
b.ServerProperties"
        "defaultValue": "/"
    }
    ...
]}
```

每个"property"是一个配置节点，用户可以使用特定的值指定它。例如， `server.port` 和 `server.servlet-path` 可能在 `application.properties` 中如以下定义：

```
server.port=9090
server.servlet-path=/home
```

"groups"是高级别的节点，它们本身不指定一个值，但为properties提供一个有上下文关联的分组。例如，`server.port`和`server.servlet-path`属性是`server`组的一部分。

注：不需要每个"property"都有一个"group"，一些属性可以以自己的形式存在。

附录B.1.1. Group属性

`groups` 数组包含的JSON对象可以由以下属性组成：

名称	类型	目的
name	String	group的全名，该属性是强制性的
type	String	group数据类型的类名。例如，如果group是基于一个被 `@ConfigurationProperties` 注解的类，该属性将包含该类的全限定名。如果基于一个 `@Bean` 方法，它将是该方法的返回类型。如果该类型未知，则该属性将被忽略
description	String	一个简短的group描述，用于展示给用户。如果没有可用描述，该属性将被忽略。推荐使用一个简短的段落描述，第一行提供一个简洁的总结，最后一行以句号结尾
sourceType	String	贡献该组的来源类名。例如，如果组基于一个被 `@ConfigurationProperties` 注解的 `@Bean` 方法，该属性将包含 `@Configuration` 类的全限定名，该类包含此方法。如果来源类型未知，则该属性将被忽略
sourceMethod	String	贡献该组的方法的全名（包含括号及参数类型）。例如，被 `@ConfigurationProperties` 注解的 `@Bean` 方法名。如果源方法未知，该属性将被忽略

附录B.1.2. Property属性

`properties` 数组中包含的JSON对象可由以下属性构成：

名称	类型	目的
name	String	property的全名，格式为小写虚线分割的形式（比如 `server.servlet-path` ）。该属性是强制性的
type	String	property数据类型的类名。例如 `java.lang.String` 。该属性可以用来指导用户他们可以输入值的类型。为了保持一致，原生类型使用它们的包装类代替，比如 `boolean` 变成了 `java.lang.Boolean` 。注意，这个类可能是个从一个字符串转换而来的复杂类型。如果类型未知则该属性会被忽略
description	String	一个简短的组的描述，用于展示给用户。如果没有描述可用则该属性会被忽略。推荐使用一个简短的段落描述，开头提供一个简洁的总结，最后一行以句号结束
sourceType	String	贡献property的来源类名。例如，如果property来自一个被 `@ConfigurationProperties` 注解的类，该属性将包括该类的全限定名。如果来源类型未知则该属性会被忽略
defaultValue	Object	当property没有定义时使用的默认值。如果property类型是个数组则该属性也可以是个数组。如果默认值未知则该属性会被忽略
deprecated	boolean	指定该property是否过期。如果该字段没有过期或该信息未知则该属性会被忽略

附录B.1.3. 可重复的元数据节点

在同一个元数据文件中出现多次相同名称的"property"和"group"对象是可以接受的。例如，Spring Boot将 `spring.datasource` 属性绑定到Hikari，Tomcat和DBCP类，并且每个都潜在的提供了重复的属性名。这些元数据的消费者需要确保他们支持这样的场景。

附录B.2. 使用注解处理器产生自己的元数据

通过使用 `spring-boot-configuration-processor` jar，你可以从被 `@ConfigurationProperties` 注解的节点轻松的产生自己的配置元数据文件。该jar包含一个在你的项目编译时会被调用的Java注解处理器。想要使用该处理器，你只需简单添加 `spring-boot-configuration-processor` 依赖，例如使用Maven你需要添加：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

使用Gradle时，你可以使用propdeps-plugin并指定：

```
dependencies {
        optional "org.springframework.boot:spring-boot-configura
tion-processor"
    }

    compileJava.dependsOn(processResources)
}
```

注：你需要将 `compileJava.dependsOn(processResources)` 添加到构建中，以确保资源在代码编译之前处理。如果没有该指令，任何 `additional-spring-configuration-metadata.json` 文件都不会被处理。

该处理器会处理被 `@ConfigurationProperties` 注解的类和方法，description属性用于产生配置类字段值的Javadoc说明。

注：你应该使用简单的文本来设置 `@ConfigurationProperties` 字段的Javadoc，因为在没有被添加到JSON之前它们是不被处理的。

属性是通过判断是否存在标准的getters和setters来发现的，对于集合类型有特殊处理（即使只出现一个getter）。该注解处理器也支持使用lombok的 `@Data`，`@Getter` 和 `@Setter` 注解。

附录 B.2.1. 内嵌属性

该注解处理器自动将内部类当做内嵌属性处理。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

    }

}
```

附录 B.2.2. 添加其他的元数据

附录C. 自动配置类

这里有一个Spring Boot提供的所有自动配置类的文档链接和源码列表。也要记着看一下你的应用都开启了哪些自动配置（使用 `--debug` 或 `-Debug` 启动应用，或在一个Actuator应用中使用 `autoconfig` 端点）。

附录 C.1 来自 `spring-boot-autoconfigure` 模块

下面的自动配置类来自 `spring-boot-autoconfigure` 模块：

配置类	链接
ActiveMQAutoConfiguration	javadoc
AopAutoConfiguration	javadoc
BatchAutoConfiguration	javadoc
CacheAutoConfiguration	javadoc
CloudAutoConfiguration	javadoc
DataSourceAutoConfiguration	javadoc
DataSourceTransactionManagerAutoConfiguration	javadoc
DeviceDelegatingViewResolverAutoConfiguration	javadoc
DeviceResolverAutoConfiguration	javadoc
DispatcherServletAutoConfiguration	javadoc

附录C.2 来自 `spring-boot-actuator` 模块

下列的自动配置类来自于 `spring-boot-actuator` 模块：

附录D. 可执行jar格式

`spring-boot-loader` 模块允许Spring Boot对可执行jar和war文件的支持。如果你正在使用Maven或Gradle插件，可执行jar会被自动产生，通常你不需要了解它是如何工作的。

如果你需要从一个不同的构建系统创建可执行jars，或你只是对底层技术好奇，本章节将提供一些背景资料。

附录D.1. 内嵌JARs

Java没有提供任何标准的方式来加载内嵌的jar文件（也就是jar文件自身包含到一个jar中）。如果你正分发一个在不解压缩的情况下可以从命令行运行的自包含应用，那这将是个问题。

为了解决这个问题，很多开发者使用"影子" jars。一个影子jar只是简单的将所有jars的类打包进一个单独的"超级jar"。使用影子jars的问题是它很难分辨在你的应用中实际可以使用的库。在多个jars中存在相同的文件名（内容不同）也是一个问题。Spring Boot另劈蹊径，让你能够直接嵌套jars。

附录D.1.1 可执行jar文件结构

Spring Boot Loader兼容的jar文件应该遵循以下结构：

```
example.jar
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
 |           +-<spring boot loader classes>
 +-com
 |  +-mycompany
 |     + project
 |        +-YouClasses.class
 +-lib
    +-dependency1.jar
    +-dependency2.jar
```

依赖需要放到内部的lib目录下。

附录D.1.2. 可执行war文件结构

Spring Boot Loader兼容的war文件应该遵循以下结构：

```
example.jar
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
 |           +-<spring boot loader classes>
 +-WEB-INF
    +-classes
    |  +-com
    |     +-mycompany
    |        +-project
    |           +-YouClasses.class
    +-lib
    |  +-dependency1.jar
    |  +-dependency2.jar
    +-lib-provided
       +-servlet-api.jar
       +-dependency3.jar
```

依赖需要放到内嵌的 `WEB-INF/lib` 目录下。任何运行时需要但部署到普通web容器不需要的依赖应该放到 `WEB-INF/lib-provided` 目录下。

附录D.2. Spring Boot的"JarFile"类

Spring Boot用于支持加载内嵌jars的核心类是 `org.springframework.boot.loader.jar.JarFile` 。它允许你从一个标准的jar文件或内嵌的子jar数据中加载jar内容。当首次加载的时候，每个JarEntry的位置被映射到一个偏移于外部jar的物理文件：

```
myapp.jar
+---------+---------------------+
|         | /lib/mylib.jar      |
| A.class |+---------+---------+|
|         || B.class | B.class ||
|         |+---------+---------+|
+---------+---------------------+
^          ^          ^
0063       3452       3980
```

上面的示例展示了如何在myapp.jar的0063处找到A.class。来自于内嵌jar的B.class实际可以在myapp.jar的3452处找到，B.class可以在3980处找到（图有问题？）。

有了这些信息，我们就可以通过简单的寻找外部jar的合适部分来加载指定的内嵌实体。我们不需要解压存档，也不需要将所有实体读取到内存中。

附录D.2.1 对标准Java "JarFile"的兼容性

Spring Boot Loader努力保持对已有代码和库的兼容。`org.springframework.boot.loader.jar.JarFile`继承自`java.util.jar.JarFile`，可以作为降级替换。

附录D.3. 启动可执行jars

`org.springframework.boot.loader.Launcher` 类是个特殊的启动类，用于一个可执行jars的主要入口。它实际上就是你jar文件的 `Main-Class` ，并用来设置一个合适的 `URLClassLoader` ，最后调用你的 `main()` 方法。

这里有3个启动器子类，JarLauncher，WarLauncher和PropertiesLauncher。它们的作用是从嵌套的jar或war文件目录中（相对于显示的从classpath）加载资源（.class文件等）。在 `[Jar|War]Launcher` 情况下，嵌套路径是固定的（ `lib/*.jar` 和war的 `lib-provided/*.jar` ），所以如果你需要很多其他jars只需添加到那些位置即可。PropertiesLauncher默认查找你应用存档的 `lib/` 目录，但你可以通过设置环境变量 `LOADER_PATH` 或application.properties中的 `loader.path` 来添加其他的位置（逗号分割的目录或存档列表）。

附录D.3.1 Launcher manifest

你需要指定一个合适的Launcher作为 `META-INF/MANIFEST.MF` 的 `Main-Class` 属性。你实际想要启动的类（也就是你编写的包含main方法的类）需要在 `Start-Class` 属性中定义。

例如，这里有个典型的可执行jar文件的MANIFEST.MF：

```
Main-Class: org.springframework.boot.loader.JarLauncher
Start-Class: com.mycompany.project.MyApplication
```

对于一个war文件，它可能是这样的：

```
Main-Class: org.springframework.boot.loader.WarLauncher
Start-Class: com.mycompany.project.MyApplication
```

注：你不需要在manifest文件中指定 `Class-Path` 实体，classpath会从嵌套的jars中被推导出来。

附录D.3.2. 暴露的存档

一些PaaS实现可能选择在运行前先解压存档。例如，Cloud Foundry就是这样操作的。你可以运行一个解压的存档，只需简单的启动合适的启动器：

```
$ unzip -q myapp.jar
$ java org.springframework.boot.loader.JarLaunche
```

附录D.4. PropertiesLauncher特性

PropertiesLauncher有一些特殊的性质，它们可以通过外部属性来启用（系统属性，环境变量，manifest实体或application.properties）。

Key	作用
loader.path	逗号分割的classpath，比如 `lib:${HOME}/app/lib`
loader.home	其他属性文件的位置，比如/opt/app（默认为 `${user.dir}` ）
loader.args	main方法的默认参数（以空格分割）
loader.main	要启动的main类名称，比如 `com.app.Application`
loader.config.name	属性文件名，比如loader（默认为application）
loader.config.location	属性文件路径，比如 `classpath:loader.properties` （默认为application.properties）
loader.system	布尔标识，表明所有的属性都应该添加到系统属性中（默认为false）

Manifest实体keys通过大写单词首字母及将分隔符从"."改为"-"（比如 `Loader-Path` ）来进行格式化。 `loader.main` 是个特例，它是通过查找manifest的 `Start-Class` ，这样也兼容JarLauncher。

环境变量可以大写字母并且用下划线代替句号。

- `loader.home` 是其他属性文件（覆盖默认）的目录位置，只要没有指定 `loader.config.location` 。
- `loader.path` 可以包含目录（递归地扫描jar和zip文件），存档路径或通配符模式（针对默认的JVM行为）。
- 占位符在使用前会被系统和环境变量加上属性文件本身的值替换掉。

附录D.5. 可执行jar的限制

当使用Spring Boot Loader打包的应用时有一些你需要考虑的限制。

附录D.5.1 Zip实体压缩

对于一个嵌套jar的ZipEntry必须使用 `ZipEntry.STORED` 方法保存。这是需要的，这样我们可以直接查找嵌套jar中的个别内容。嵌套jar的内容本身可以仍旧被压缩，正如外部jar的其他任何实体。

附录D.5.2. 系统ClassLoader

启动的应用在加载类时应该使用 `Thread.getContextClassLoader()`（多数库和框架都默认这样做）。尝试通过 `ClassLoader.getSystemClassLoader()` 加载嵌套的类将失败。请注意 `java.util.Logging` 总是使用系统类加载器，由于这个原因你需要考虑一个不同的日志实现。

附录D.6. 可替代的单一jar解决方案

如果以上限制造成你不能使用Spring Boot Loader，那可以考虑以下的替代方案：

- Maven Shade Plugin
- JarClassLoader
- OneJar

附录E. 依赖版本

下面的表格提供了详尽的依赖版本信息，这些版本由Spring Boot自身的CLI，Maven的依赖管理（dependency management）和Gradle插件提供。当你声明一个对以下artifacts的依赖而没有声明版本时，将使用下面表格中列出的版本。