Learn the Spring basics — Spring Boot builds on many other Spring projects

Spring boot项目建立在许多其他的spring项目之上

Spring boot features:

1. Core Featrues:spring应用，外部的配置，profiles,日志
2. Web 应用：mvc 内嵌容器（即内嵌tomcat等容器servlet容器）
3. 数据库操作：支持sql 和no sql
4. 消息系统：jms
5. 测试：boot应用 utils
6. 扩展：自动配置 @Conditions

easy to create stand-alone（独立的）, production-grade（工业品味的） Spring based Applications

spring boot可过构建jar应用 war应用，spring boot团队还提供了一个command line工具可以运行spring 的脚本

spring boot的目标

1. 快 而简洁
2. 提供了项目所需的通用配置，让使用者更加的关注代码而不是那些配置
3. 几乎无需xml的配置

系统需求

Spring 1.5.7要求

1. Java 7+ 推荐使用java 8
2. Spring fream work 4.3.11+
3. Maven 3.2+ gradle 2.9+

内嵌的servlet容器

1. Tomcat 7 8
2. Jetty 8 9.2 9.3
3. Undertow 1.3

Servlet 版本 3.0+

典型的pom配置

1. 项目需要继承自org.springframework.boot，所以需要parent标签
2. 根据项目的需求增加相应的starter依赖
3. 另外，maven平台提供了一个构建插件来构建jar应用

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://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>1.5.7.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>

安装spring boot command line

下载地址

* [spring-boot-cli-1.5.7.RELEASE-bin.zip](http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.5.7.RELEASE/spring-boot-cli-1.5.7.RELEASE-bin.zip)
* [spring-boot-cli-1.5.7.RELEASE-bin.tar.gz](http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.5.7.RELEASE/spring-boot-cli-1.5.7.RELEASE-bin.tar.gz)

添加环境变量

安装后spring –version 验证

可以参照[BASH](https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29" \t "_top)的配置，从而使得bash环境有自动补全功能，这样可以编辑文本并由spring boot cli来运行，可以运行groovy和java版本

编辑app.groovy

*@RestController*

**class** ThisWillActuallyRun {

*@RequestMapping("/")*

String home() {

"Hello World!"

}

}

运行：spring run app.groovy

第一次运行可能会很慢因为它要下载很多的依赖，之后的运行会很快

部署spring boot应用

最基本的pom

只需要指定parent 标签就可以

然后添加各种依赖 和 功能插件

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://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>1.5.7.RELEASE</version>

</parent>

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

</project>

添加依赖

spring-boot-starter-parent 的设置可以为我们引入很多的默认依赖和配置，它提供的[dependency-management](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-dependency-management)使得我们也不需要不需指明依赖的版本，这些版本号可以省略

The mvn dependency:tree command prints a tree representation of your project dependencies

spring-boot-starter-web的依赖为我们添加了很多的依赖，包括tomcat服务器

编写简单的代码

Exampl.java

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

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

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

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

*@RestController //表明是一个rest风格的controller*

*@EnableAutoConfiguration //自动配置*

**public** **class** Example {

*@RequestMapping("/")*

String home() {

**return** "Hello World!";

}

**public** **static** **void** main(String[] args) **throws** Exception {//spring boot 应用程序主函数

SpringApplication.run(Example.**class**, args);

}

}

EnableAutoConfiguration 注解

This annotation tells Spring Boot to “guess” how you will want to configure Spring, based on the jar dependencies that you have added

该自动配置注解会根据我们的以来来自动设置默认的配置，如web的8080端口等

自动配置（其实就是默认的配置）是与starter的类型相关的，但不是强相关

Main主方法

Java虚拟机调用main方法，main方法委托给springApplication，该类的run方法然后会启动相关的自动配置和服务。

mvn spring-boot:run from the root project directory to start the application

创建一个jar应用程序

为了创建一个可执行的应用，加入以下插件依赖

To create an executable jar we need to add the spring-boot-maven-plugin to our pom.xml. Insert the following lines just below the dependencies section:

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

编辑完pom后，在命令行运行mvn package命令会将应用打包成jar文件

运行jar程序，在命令行执行命令 java –jar xxx.jar

### 使用spring boot

#### 构建系统

Maven 或者 gradle

### 发布系统

Each release of Spring Boot is associated with a base version of the Spring Framework so we **highly** recommend you to not specify its version on your own

每一个spring boot 都是以一个基础的spring frame work 为基础的，所以强烈建议不要在spring boot 的依赖上加上版本信息。

### Maven

如果项目继承自spring-boot-starter-parent，则项目默认拥有了以下的基本配置：

1. Java 1.6+
2. Utf-8 源代码设置
3. [Dependency Management section](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-dependency-management) 使得对于通用的依赖不需要指定版本信息
4. 资源过滤
5. 插件配置，这包括 [exec plugin](http://www.mojohaus.org/exec-maven-plugin/), [surefire](https://maven.apache.org/surefire/maven-surefire-plugin/), [Git commit ID](https://github.com/ktoso/maven-git-commit-id-plugin), [shade](https://maven.apache.org/plugins/maven-shade-plugin/)
6. 自动管理配置文件的不同的profile（如pro test dev）
7. 可以设置resource.delimiter

### 继承自 starter parent

配置文件增加以下几行即可：

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

<parent>

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

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

<version>1.5.7.RELEASE</version>

</parent>

正如上面说的继承自这个starter-parent会给我们带来很多的好处，如果我们仍然依赖于其他的starter的时候，只需要引入相应的依赖而不需要指明特定的版本。

You should only need to specify the Spring Boot version number on this dependency. If you import additional starters, you can safely omit the version number

但是我们仍然可以指定某些starter的版本，通过类似如下这样的配置：

<properties>

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

</properties>

### 不使用parent pom

如果由于某些原因我们没有使用继承自spring-boot-starter-parent的配置，例如我们需要自己的配置或者我们希望继承自其他的项目，这时为了使用spring-boot-starter-parent为我们带来的便利，可以使用如下的配置：

<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>1.5.7.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

### 修改java的版本

只需要以下的配置

<properties>

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

</properties>

### 使用spring boot maven plugin

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

这个插件能够将项目打包成可执行的jar文件

### Starters

Starter 为我们做了很多的工作，也为我们引入了很多的默认依赖

The starters contain a lot of the dependencies that you need to get a project up and running quickly and with a consistent, supported set of managed transitive dependencies.

所有的starter的名字都有类似的模式spring-boot-starter-\*

### 使代码结构化

1. 避免使用默认的包

当定义一个没有指定package的java文件的时候默认放入了默认的包内default package，强烈建议为每个java文件加上包定义，避免使用默认的包

1. 放置main application class

强烈建议将main类放在根包上，@EnableAutoConfiguration注解经常被放在置main class上，这个注解可以为特定的元素配置一个基本的搜索包；另外放在根包下，可以使用@ComponentScan注解而不需要配置basePackage属性，因为默认扫描main class 所在的包即子包，而main 在根包下说明可以扫描到所有的文件。也可以使用@SpringBootApplication注解

典型的结构如下：

com

+- example

+- myproject

+- Application.java

|

+- domain

| +- Customer.java

| +- CustomerRepository.java

|

+- service

| +- CustomerService.java

|

+- web

+- CustomerController.java

### Configuration classes

强烈建议在main 类加上@Configuration注解

### 引入其他的注解类

@Import可以用来引入其他的配置类，也可以使用@ComponentScan自动扫描被@Configuration修饰的所有的配置类

### 导入xml配置

如果仍然使用基于xml的配置，强烈建议使用@Configuration注解在主类上，并且使用@ImportResource注解来加载xml的配置文件

## 自动配置

Spring boot能够根据依赖自动配置应用（如web 自动配置端口为8080，mongodb自动配置地址为local,端口号为27017等）

通过@EnableAutoConfiguration注解开启自动配置

### 逐步替代自动配置

自动配置是非侵入性的，在任何时候都可以定义自己的配置以替代默认的配置。

通过调试可以查看默认的自动配置项

### 禁用特定的自动配置

可以使用@EnableAutoConfiguration的exclude属性来停止特定配置类的使用，如*@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})*

在pom中的property添加spring.autoconfigure.exclude也可以排除掉自动配置类

## Spring的bean和依赖注入

@ComponentScan注解会扫描需要被spring管理的beans,结合@Autowire注入依赖

如果将main类放在了root package下，使用@ComponentScan不需要添加任何的属性，所有的带有@Component, @Service, @Repository, @Controller 注解的类都可以被自动注册为spring 的beans.

一个实例：

**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;

}

*// ...*

}

## 使用@SpringBootApplication注解

Springbootapplication 等价于Configuration enableautoConfiguration 和componentScan

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes

## 运行应用

打包成jar 运行

### 作为一个打包应用运行

使用java –jar命令来运行

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

也可以启用远程调试：

It is also possible to run a packaged application with remote debugging support enabled. This allows you to attach a debugger to your packaged application:

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

-jar target/myproject-0.0.1-SNAPSHOT.jar

### Maven插件

使用maven插件打包并运行应用

$ mvn spring-boot:run

## 开发者工具

Spring-boot-devtools可以添加进项目以增加开发的效率

The spring-boot-devtools module can be included in any project to provide additional development-time features

Maven实例：

<dependencies>

<dependency>

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

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

<optional>true</optional>

</dependency>

</dependencies>

在生产环境中，开发者工具默认是被禁用的

Developer tools are automatically disabled when running a fully packaged application

### 开发工具的默认属性

Spring boot 支持的很多类库使用了cache， 如模板引擎 处理静态资源的http header信息

Spring 开发者工具默认禁用了这些缓存特性，因为它们不利于开发

缓存选项通常需要在application.properties中配置

Rather than needing to set these properties manually, the spring-boot-devtools module will automatically apply sensible development-time configuration.

Spring-boot-devtools模块会自动应用开发阶段配置

For a complete list of the properties that are applied see [DevToolsPropertyDefaultsPostProcessor](https://github.com/spring-projects/spring-boot/tree/v1.5.7.RELEASE/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/env/DevToolsPropertyDefaultsPostProcessor.java" \t "_top).

### 自动重启

配置了spring-boot-devtools后，当类文件改变的之后会自动重启应用。

Applications that use spring-boot-devtools will automatically restart whenever files on the classpath change

如果要禁止使用自动重启功能，可以这样配置SpringApplication.setRegisterShutdownHook(false)

两套classloader

This approach means that application restarts are typically much faster than “cold starts” since the *base* classloader is already available and populated

### 排除自动重启资源

By default changing resources in /META-INF/maven, /META-INF/resources, /resources, /static, /public or /templates will not trigger a restart but will trigger a [live reload](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-devtools-livereload)

这些资源的改变不会导致应用重启但会导致重载

spring.devtools.restart.exclude属性pom配置可以设定排除重启的规则，如

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

除了static和public的改变不会导致重启外其他的改变都会导致应用的重启

如果想要添加排除在外的重启项规则可在pom中配置

spring.devtools.restart.additional-exclude属性

### 监控其他的路径

spring.devtools.restart.additional-paths属性pom可以配置请他的路径(一般为非classloader路径)，这样当这些文件改变时也会导致应用重启。相反，spring.devtools.restart.exclude属性会排除掉导致重启的路径。

### 禁用应用重启

通过在pom中设置spring.devtools.restart.enabled属性来禁用应用重启的特性。这个属性也可以在application.properties中配置

如果要完全禁用应用重启特性，可以这样

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

System.setProperty("spring.devtools.restart.enabled", "false");

SpringApplication.run(MyApp.**class**, args);

}

使用触发文件

spring.devtools.restart.trigger-file属性可以指定一个触发文件

改变这个文件会触发一次检查，并不一定会导致应用重启

Changing the file only triggers the check and the restart will only occur if Devtools has detected it has to do something

## 热加载LiveReload

spring-boot-devtools模块包含了一个热加载服务器，这个服务器会在资源被改变时触发浏览器刷新

如果不需要热加载特性，可以在pom中设置spring.devtools.livereload.enabled属性

### 全局设置

可以通过增加一个配置文件.spring-boot-devtools.properties来设置devtools的全局设置，如

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

### 远程应用

需要开启devtools

<build>

<plugins>

<plugin>

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

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

<configuration>

<excludeDevtools>false</excludeDevtools>

</configuration>

</plugin>

</plugins>

</build>

然后设置属性spring.devtools.remote.secret，如spring.devtools.remote.secret=mysecret

远程调试server和client，只有设置了spring.devtools.remote.secret属性才会自动启动server

Remote devtools support is provided in two parts; there is a server side endpoint that accepts connections, and a client application that you run in your IDE. The server component is automatically enabled when the spring.devtools.remote.secret property is set. The client component must be launched manually

### 启动远程客户端应用

需要启动org.springframework.boot.devtools.RemoteSpringApplication

应用(这个应用使用与springbootapplication完全一样的classpath这样才能将远程应用镜像到本地环境进行调试，因为远程应用于本地的是完全一样的)，启用需要传递remote url作为参数

启动步骤：假设有一个名为my-app的项目，部署在cloud foundy

* Select Run Configurations…​ from the Run menu.
* Create a new Java Application “launch configuration”.
* Browse for the my-app project.
* Use org.springframework.boot.devtools.RemoteSpringApplication as the main class.

Add https://myapp.cfapps.io to the Program arguments (or whatever your remote URL is)

Because the remote client is using the same classpath as the real application it can directly read application properties. This is how the spring.devtools.remote.secret property is read and passed to the server for authentication.

### 远程更新

远程应用会在本地classpath修改后自动跟新，就像之前的application restart一样

The remote client will monitor your application classpath for changes in the same way as the [local restart](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-devtools-restart). Any updated resource will be pushed to the remote application and *(if required)* trigger a restart

而且根据前面，重启应用会很快

### 远程调试通道

为了解决远程调试问题，spring boot的devtools提供了基于http的远程调试特性。

远程client会启动一个本地服务监听8000端口，这样就能够附加上一个远程调试器。可以通过spring.devtools.remote.debug.local-port来设置一个不同的端口

调试前需要确保远程应用开启了远程调试，这个需要配置JAVA\_OPS

示例：

*---*

env:

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

在使用idea调试时，需要确保断点打到了执行线程而非vm上。

When using the remote debug tunnel with IntelliJ IDEA, all breakpoints must be configured to suspend the thread rather than the VM. By default, breakpoints in IntelliJ IDEA suspend the entire VM rather than only suspending the thread that hit the breakpoint. This has the unwanted side-effect of suspending the thread that manages the remote debug tunnel, causing your debugging session to freeze. When using the remote debug tunnel with IntelliJ IDEA, all breakpoints should be configured to suspend the thread rather than the VM. Please see [IDEA-165769](https://youtrack.jetbrains.com/issue/IDEA-165769) for further details.

# Spring boot 特性

### Spring application

开启debug会输出所有的启动信息

运行应用并开启debug命令：

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

### 自定义banner

The SpringApplication.setBanner(…​) method can be used if you want to generate a banner programmatically. Use the org.springframework.boot.Banner interface and implement your own printBanner() method.

### 自定义springapplication

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

SpringApplication app = **new** SpringApplication(MySpringConfiguration.**class**);

app.setBannerMode(Banner.Mode.OFF);

app.run(args);

}