



Spring Boot

EN ACCIÓN

Paredes de Craig

FORDENADO POR Andrew Glover



Dotación

Spring Boot en acción

PAREDES CRAIG



MANN I NG

Isla refugio

Para obtener información en línea y solicitar este y otros libros de Manning, visite www.manning.com. El editor ofrece descuentos en este libro cuando se pide en cantidad. Para obtener más información, póngase en contacto


Departamento de ventas especiales
Publicaciones Manning Co.
20 Baldwin Road
Apartado de correos 761
Shelter Island, NY 11964 Correo electrónico:
orders@manning.com

© 2016 por Manning Publications Co. Todos los derechos reservados.

Ninguna parte de esta publicación puede ser reproducida, almacenada en un sistema de recuperación o transmitida, en cualquier forma o por medio electrónico, mecánico, fotocopiado o de otro modo, sin el permiso previo por escrito del editor.

Muchas de las designaciones utilizadas por los fabricantes y vendedores para distinguir sus productos se reclaman como marcas comerciales. Cuando esas designaciones aparecen en el libro, y Manning Publications tenía conocimiento de un reclamo de marca registrada, las designaciones se han impreso en mayúsculas iniciales o en mayúsculas.

☺ Reconociendo la importancia de preservar lo que se ha escrito, es política de Manning que los libros que publicamos se impriman en papel libre de ácido, y hacemos todo lo posible para lograrlo. Reconociendo también nuestra responsabilidad de conservar los recursos de nuestro planeta, los libros de Manning están impresos en papel que es al menos un 15 por ciento reciclado y procesado sin el uso de cloro elemental.

 Publicaciones Manning Co.
20 Baldwin Road
Apartado de correos 761
Shelter Island, NY 11964

Editor de desarrollo:	Cynthia Kane
Editor de desarrollo técnico:	Robert Casazza
Editor de copia:	Andy Carroll
Corrector de pruebas:	Corbin Collins
Corrector técnico:	John Guthrie
Tipógrafo:	Gordan Salinovic
Diseñador de la portada:	Marija Tudor

ISBN 9781617292545

Impreso en los Estados Unidos de América.

1 2 3 4 5 6 7 8 9 10 - EBM - 20 19 18 17 16 15

contenido

prólogo vii
prefacio ix
sobre este libro xii

1 Muelle de arranque 1

agradecimientos xv
1.1 Spring reiniciado 2

Echando un vistazo a la primavera 2 . *Examen de los elementos esenciales de Spring Boot 4*
.Lo que Spring Boot no es 7

1.2 Empezando con Spring Boot 8

Instalación de Spring Boot CLI 8 . *Inicializando un proyecto Spring Boot con Spring Initializr 12*

2 Desarrollando su primera aplicación Spring Boot 23

1.3 Resumen 22
2.1 Poner a trabajar Spring Boot 24

Examinar un proyecto Spring Boot recién inicializado 26 . *Disecionando un proyecto Spring Boot build 30*

2.2 Uso de dependencias de inicio 33

Especificación de dependencias basadas en facetas 34 . *Anulación de las dependencias transitivas de arranque 35*

2.3 Uso de la configuración automática 37

Centrándose en la funcionalidad de la aplicación 37 .Ejecución de la aplicación 43 .¿Lo que acaba de suceder? 45

3 Personalización de la configuración 49

2.4 Resumen 48

3.1 Anulación de la configuración automática de Spring Boot 50

Asegurar la aplicación 50 .Creación de una configuración de seguridad personalizada 51 .Echando otro vistazo bajo las sábanas de la configuración automática 55

3.2 Configuración externa con propiedades 57

Autoconfiguración de ajuste fino 58 .Configuración externa de beans de aplicación 64 .Configuración con perfiles 69

3.3 Personalizar las páginas de error de la aplicación 71

4 Prueba con Spring Boot 76

3.4 Resumen 74

4.1 Configuración automática de pruebas de integración 77

4.2 Prueba de aplicaciones web 79

Mocking Spring MVC 80 .Prueba de seguridad web 83

4.3 Probar una aplicación en ejecución 86

Iniciar el servidor en un puerto aleatorio 87 .Prueba de páginas HTML con Selenium 88

5 Ponerse genial con Spring Boot CLI 92

4.4 Resumen 90

5.1 Desarrollo de una aplicación CLI Spring Boot 93

Configuración del proyecto CLI 93 .Eliminando el ruido de código con Groovy 94 .¿Lo que acaba de suceder? 98

5.2 Tomando dependencias 100

Anulación de las versiones de dependencia predeterminadas 101 .Agregar repositorios de dependencia 102

5.3 Ejecución de pruebas con CLI 102

5.4 Creación de un artefacto desplegable 105

5.5 Resumen 106

6 *Aplicación de Grails en Spring Boot 107*

6.1 Uso de GORM para la persistencia de datos 108

6.2 Definición de vistas con Groovy Server Pages 113

6.3 Mezcla de bota de resorte con rejillas 3115

Creación de un nuevo proyecto de Grails 116 . Definición del dominio 118 Escritura de un controlador Grails 119 . Creando la vista 120

7 *Echando un vistazo al interior con el actuador 124*

6.4 Resumen 123

7.1 Exploración de los puntos finales del actuador 125

Ver detalles de configuración 126 . Tocar métricas de tiempo de ejecución 133 Cerrar la aplicación 139 . Obtención de información de la aplicación 140

7.2 Conexión a la carcasa remota del actuador 141

Visualización del informe de configuración automática 142 . Listado de beans de aplicación 143 Observación de métricas de aplicaciones 144 . Invocar puntos finales del actuador 145

7.3 Supervisión de su aplicación con JMX 146

7.4 Personalización del actuador 148

Cambiar los ID de los puntos finales 148 . Habilitación y deshabilitación de puntos finales 149 Adición de indicadores y métricas personalizadas 149 . Creación de un repositorio de seguimiento personalizado 153 . Conectar indicadores de salud personalizados 155

7.5 Asegurar puntos finales del actuador 156

8 *Implementación de aplicaciones Spring Boot 160*

7.6 Resumen 159

8.1 Opciones de implementación de pesaje 161

8.2 Implementación en un servidor de aplicaciones 162

Construyendo un archivo WAR 162 . Creación de un perfil de producción 164 Habilitación de la migración de la base de datos 168

8.3 Empujando a la nube 173

Implementación en Cloud Foundry 173 . Implementación en Heroku 177

8.4 Resumen 180

Apéndice A Herramientas para desarrolladores de Spring Boot 181

apéndice B Iniciadores de Spring Boot 188

Apéndice C Propiedades de configuración 195

apéndice D Dependencias de Spring Boot 232

índice 243

prefacio

En la primavera de 2014, el equipo de ingeniería de entrega de Netflix se propuso lograr un objetivo elevado: permitir la entrega continua global de extremo a extremo a través de una plataforma de software que facilita tanto la extensibilidad como la resiliencia. Mi equipo había construido previamente dos aplicaciones diferentes que intentaban abordar las necesidades de entrega e implementación de Netflix, pero ambas estaban comenzando a mostrar signos reveladores de monolito y ninguna cumplía los objetivos de flexibilidad y resistencia. Además, el efecto más frustrante de estas aplicaciones monolíticas fue, en última instancia, que no pudimos seguir el ritmo de la innovación de nuestro socio. Los usuarios habían comenzado a moverse con nuestras herramientas en lugar de con ellas.

Se hizo evidente que si queríamos proporcionar valor real a la empresa e innovar rápidamente, necesitábamos dividir los monolitos en servicios pequeños e independientes que pudieran lanzarse a voluntad. Adoptar una arquitectura de microservicio nos dio la esperanza de que también podríamos abordar los objetivos gemelos de flexibilidad y resistencia. Pero necesitábamos hacerlo sobre una base creíble en la que pudiéramos contar con simultaneidad real, monitoreo legítimo, descubrimiento de servicios confiable y fácil y un excelente rendimiento en tiempo de ejecución.

Con la JVM como base fundamental, buscamos un marco que nos diera una velocidad rápida y una operacionalización firme desde el primer momento. Nos concentramos en Spring Boot.

Spring Boot facilita la creación de servicios listos para producción y con tecnología Spring sin mucho código. De hecho, el hecho de que una simple aplicación Spring Boot Hello World pueda caber en un tweet es una desviación radical de lo que requería la misma funcionalidad en la JVM hace solo unos pocos años. Las características no funcionales listas para usar, como la seguridad, las métricas, las comprobaciones de estado, los servidores integrados y la configuración externa, hicieron de Boot una opción fácil para nosotros.

Sin embargo, cuando nos embarcamos en nuestro viaje de Spring Boot, era difícil conseguir documentación sólida. Confiar en el código fuente no es la forma más divertida de descubrir cómo aprovechar adecuadamente las características de un marco.

No es sorprendente ver al autor del venerable libro de Manning, *Primavera en acción*. Asuma el desafío de resumir de manera concisa los aspectos centrales de trabajar con Spring Boot en otro libro convincente. ¡Tampoco es sorprendente que Craig y el equipo de Manning hayan hecho otro trabajo tremendamente maravilloso! *Spring Boot en acción* es un libro de fácil lectura, como ahora esperamos de Craig y Manning.

Desde la introducción que llama la atención del capítulo 1 a Boot y la ahora legendaria aplicación tweetableBoot de caracteres noventa hasta un análisis en profundidad del Actuador de Boot en el capítulo 7, que habilita una serie de características operativas mágicas automáticas necesarias para cualquier aplicación de producción, *Spring Boot en acción* no deja piedra sin remover. De hecho, para mí, la inmersión profunda del capítulo 7 en el Actuador respondió algunas de las preguntas persistentes que tenía en la parte posterior de mi cabeza desde que tomé Boot hace más de un año. El examen minucioso del Capítulo 8 de las opciones de implementación me abrió los ojos a la simplicidad de Cloud Foundry para las implementaciones en la nube. Uno de mis capítulos favoritos es el capítulo 4, donde Craig explora las muchas opciones poderosas para probar fácilmente una aplicación de arranque. Desde el principio, me sorprendieron gratamente algunas de las funciones de prueba de Spring, y Boot las aprovecha muy bien.

Como he dicho públicamente antes, Spring Boot es el tipo de marco que la comunidad Java ha estado buscando durante más de una década. Sus funciones de desarrollo fáciles de usar y su operatividad lista para usar hacen que el desarrollo de Java sea divertido nuevamente. Me complace informar que Spring y Spring Boot son la base de la nueva plataforma de entrega continua de Netflix. Además, otros equipos de Netflix están siguiendo el mismo camino porque ellos también ven los innumerables beneficios de Boot.

Es con entusiasmo y pasión a partes iguales que apoyo absolutamente el libro de Craig como la documentación de Spring Boot fácil de digerir y divertida de leer que la comunidad de Java ha estado esperando desde que Boot tomó a la comunidad por asalto. El estilo de escritura accesible de Craig y el análisis amplio de las funciones y funciones centrales de Boot seguramente dejarán a los lectores con una sólida comprensión de Boot (junto con una alegre sensación de asombro por él).

Sigan con el gran trabajo Craig, Manning Publications y todos los brillantes desarrolladores que han hecho de Spring Boot lo que es hoy. Cada uno de ustedes ha asegurado un futuro brillante para la JVM.

A NDREW GRAMO AMANTE

METRO ANAGER, D ELIVERY MI INGENIERÍA EN NORTE ETLFLIX

prefacio

En la Feria Mundial de Nueva York de 1964, Walt Disney presentó tres atracciones innovadoras: "es un mundo pequeño", "Grandes momentos con el Sr. Lincoln" y el "Carrusel del progreso". Desde entonces, estas tres atracciones se han trasladado a Disneyland y Walt Disney World, y todavía puedes verlas hoy.

Mi favorito de estos es el carrusel del progreso. Supuestamente, también era uno de los favoritos de Walt Disney. Es parte de un espectáculo y parte de un espectáculo en el que el área de asientos gira alrededor de un área central con cuatro escenarios. Cada etapa cuenta la historia de una familia en diferentes períodos del siglo XX (principios de 1900, 1920, 1940 y épocas recientes), destacando los avances tecnológicos en ese período de tiempo. La historia de la innovación se cuenta desde una lavadora de manivela hasta iluminación eléctrica y radio, lavavajillas automáticos y televisión, computadoras y electrodomésticos activados por voz.

En cada acto, el padre (que también es el narrador del programa) habla de los últimos inventos y dice "No puede ser mejor", solo para descubrir que, de hecho, mejora en el siguiente acto como la tecnología avanza.

Aunque Spring no tiene una historia tan larga como la que se muestra en el Carrusel del Progreso, siento lo mismo acerca de Spring como lo sentía "Progress Dad" acerca del siglo XX. Todas y cada una de las aplicaciones de Spring parecen mejorar mucho la vida de los desarrolladores. Con solo ver cómo se declaran y conectan los componentes de Spring, podemos ver la siguiente progresión a lo largo de la historia de Spring:

- Cuando Spring 1.0 entró en escena, cambió por completo la forma en que desarrollamos aplicaciones Java empresariales. La inyección de dependencia de Spring y las transacciones declarativas significaron que no hubo más acoplamiento estrecho de componentes y no más EJB de peso pesado. No podría ser mejor.
- Con Spring 2.0 podríamos usar espacios de nombres XML personalizados para la configuración, haciendo que Spring sea aún más fácil de usar con archivos de configuración más pequeños y fáciles de entender. No podría ser mejor.
- Spring 2.5 nos dio un modelo de inyección de dependencia orientado a anotaciones mucho más elegante con `@ Componente` y `@ Autowired` anotaciones, así como un modelo de programación Spring MVC orientado a anotaciones. No más declaraciones explícitas de los componentes de la aplicación y no más subclases de una de varias clases de controlador base. No podría ser mejor.
- Luego, con Spring 3.0 nos dieron una nueva alternativa de configuración basada en Java a XML que se mejoró aún más en Spring 3.1 con una variedad de `@ Habilitar-` anotaciones prefijadas. Por primera vez, resulta realista escribir una aplicación Spring completa sin ninguna configuración XML. No podría ser mejor. Spring 4.0 desató el soporte para la configuración condicional, donde las decisiones en tiempo de ejecución
- determinarían qué configuración se usaría y cuál se ignoraría en función de la ruta de clase, el entorno y otros factores de la aplicación. Ya no necesitábamos escribir scripts para tomar esas decisiones en el momento de la compilación y elegir qué configuración debería incluirse en la implementación. ¿Cómo podría mejorar?

Y luego vino Spring Boot. Aunque con cada lanzamiento de Spring pensamos que no podría mejorar, Spring Boot demostró que todavía queda mucha magia en Spring. De hecho, creo que Spring Boot es lo más importante y emocionante que ha sucedido en el desarrollo de Java en mucho tiempo.

Sobre la base de los avances anteriores en Spring Framework, Spring Boot permite la configuración automática, lo que hace posible que Spring detecte de manera inteligente qué tipo de aplicación está creando y configure automáticamente los componentes necesarios para satisfacer las necesidades de la aplicación. No es necesario escribir una configuración explícita para escenarios de configuración comunes; Spring se ocupará de ello por ti.

Las dependencias de inicio de Spring Boot hacen que sea aún más fácil seleccionar qué bibliotecas de tiempo de compilación y tiempo de ejecución incluir en las compilaciones de su aplicación al agregar las dependencias comúnmente necesarias. Los arrancadores de Spring Boot no solo mantienen la sección de dependencias de sus especificaciones de compilación más corta, sino que evitan que tenga que pensar demasiado en las bibliotecas y versiones específicas que necesita.

La interfaz de línea de comandos de Spring Boot ofrece una opción atractiva para desarrollar aplicaciones Spring en Groovy con un mínimo de ruido o ceremonia común en las aplicaciones Java. Con Spring Boot CLI, no hay necesidad de métodos de acceso, modificadores de acceso como público o privado, punto y coma, o el regreso palabra clave. En muchos casos, incluso puede eliminar importar declaraciones. Y debido a que ejecuta la aplicación como scripts desde la línea de comandos, no necesita una especificación de compilación.

El actuador de Spring Boot le brinda información sobre el funcionamiento interno de una aplicación en ejecución. Puede ver exactamente qué beans están en el contexto de la aplicación Spring, cómo se asignan los controladores Spring MVC a las rutas, las propiedades de configuración disponibles para su aplicación y mucho más.

Con todas estas maravillosas características habilitadas por Spring Boot, ¡ciertamente no puede ser mejor!

En este libro, verá cómo Spring Boot ha hecho que Spring sea aún mejor que antes. Veremos la configuración automática, los arrancadores de Spring Boot, el Spring Boot CLI y el actuador. Y jugaremos con la última versión de Grails, que se basa en Spring Boot. Cuando terminemos, probablemente pensarás que Spring no podría mejorar.

Si hemos aprendido algo del carrusel del progreso de Walt Disney, es que cuando pensamos que las cosas no pueden mejorar, es inevitable que mejoren. Los avances que ofrece Spring Boot ya se están aprovechando para permitir avances aún mayores. Es difícil imaginar que Spring mejore de lo que es ahora, pero ciertamente lo hará. Con Spring, siempre hay un gran mañana hermoso.

sobre este libro

Spring Boot tiene como objetivo simplificar el desarrollo de Spring. Como tal, el alcance de Spring Boot se extiende para tocar todo lo que toca Spring. Sería imposible escribir un libro que cubra todas las formas en que se puede usar Spring Boot, ya que hacerlo implicaría cubrir cada una de las tecnologías que admite Spring. En lugar de, *Spring Boot en acción*

tiene como objetivo resumir Spring Boot en cuatro temas principales: configuración automática, dependencias de arranque, la interfaz de línea de comandos y el actuador. En el camino, tocaremos algunas características de Spring según sea necesario, pero el enfoque estará principalmente en Spring Boot.

Spring Boot en acción es para todos los desarrolladores de Java. Aunque algunos antecedentes en Spring podrían considerarse un requisito previo, Spring Boot tiene una forma de hacer que Spring sea más accesible incluso para aquellos que son nuevos en Spring. Sin embargo, dado que este libro se centrará en Spring Boot y no profundizará en Spring en sí, puede resultarle útil combinarlo con otros materiales de Spring, como *Primavera en acción, cuarta edición* (Manning, 2014).

Mapa vial

Spring Boot en acción está dividido en siete capítulos:

- En el capítulo 1, se le dará una descripción general de Spring Boot, incluidos los aspectos básicos de la configuración automática, las dependencias de inicio, la interfaz de línea de comandos y el actuador.
- El Capítulo 2 profundiza en Spring Boot, centrándose en la configuración automática y las dependencias de inicio. En este capítulo, creará una aplicación Spring completa con muy poca configuración explícita.

- El capítulo 3 comienza donde termina el capítulo 2, mostrando cómo puede influir en la configuración automática estableciendo las propiedades de la aplicación o anulando completamente la configuración automática cuando no satisface sus necesidades.
- En el capítulo 4 veremos cómo escribir pruebas de integración automatizadas para aplicaciones Spring Boot.
- En el capítulo 5, verá cómo Spring Boot CLI ofrece una alternativa convincente al desarrollo convencional de Java al permitirle escribir aplicaciones completas como un conjunto de scripts Groovy que se ejecutan desde la línea de comandos.
- Mientras estamos en el tema de Groovy, el capítulo 6 echa un vistazo a Grails 3, la última versión del marco de Grails, que ahora se basa en Spring Boot.
- En el capítulo 7, verá cómo aprovechar el actuador de Spring Boot para excavar dentro de una aplicación en ejecución y ver qué la hace funcionar. Verá cómo usar los puntos finales web de Actuador, así como un shell remoto y MBeans JMX para echar un vistazo a las partes internas de una aplicación.
- El Capítulo 8 concluye discutiendo varias opciones para implementar su aplicación Spring Boot, incluida la implementación del servidor de aplicaciones tradicional y la implementación en la nube.

Convenciones de código y descargas

Hay muchos ejemplos de código a lo largo de este libro. Estos ejemplos siempre aparecerán en un fuente de código de ancho fijo como esta. Cualquier nombre de clase, nombre de método o fragmento XML dentro del texto normal del libro aparecerá en fuente de código también. Muchas de las clases y paquetes de Spring tienen nombres excepcionalmente largos (pero expresivos). Debido a esto, los marcadores de continuación de línea (`\>`) puede incluirse cuando sea necesario. No todos los ejemplos de código de este libro estarán completos. A menudo, solo muestro uno o dos métodos de una clase para centrarme en un tema en particular.

El código fuente completo de las aplicaciones que se encuentran en el libro se puede descargar desde el sitio web del editor en www.manning.com/books/spring-boot-in-action.

Autor en línea

La compra de *Spring Boot en acción* incluye acceso gratuito a un foro web privado dirigido por Manning Publications, donde puede hacer comentarios sobre el libro, hacer preguntas técnicas y recibir ayuda del autor y de otros usuarios. Para acceder al foro y suscribirse a él, dirija su navegador web a www.manning.com/books/spring-boot-in-action. Esta página proporciona información sobre cómo ingresar al foro una vez que esté registrado, qué tipo de ayuda está disponible y las reglas de conducta en el foro.

El compromiso de Manning con nuestros lectores es proporcionar un lugar donde pueda tener lugar un diálogo significativo entre los lectores individuales y entre los lectores y el autor. No es un compromiso con una cantidad específica de participación por parte del autor cuya contribución al foro sigue siendo voluntaria (y no remunerada). Le sugerimos que intente hacerle al autor algunas preguntas desafiantes para que no se pierda su interés.

El foro Author Online y los archivos de discusiones anteriores serán accesibles desde el sitio web del editor siempre que el libro esté impreso.

Acerca de la ilustración de la portada

La figura de la portada de *Spring Boot en acción* se titula "Hábito de un tártaro en Kasan", que es la capital de la República de Tartaristán en Rusia. La ilustración está tomada de Thomas Jefferys *Una colección de vestidos de diferentes naciones, antiguas y modernas* (cuatro volúmenes), Londres, publicado entre 1757 y 1772. La portada indica que se trata de grabados en cobre pintado a mano, realizados con goma arábica. Thomas Jefferys (1719-1771) fue llamado "Geógrafo del rey Jorge III". Fue un cartógrafo inglés que fue el principal proveedor de mapas de su época. Grabó e imprimió mapas para el gobierno y otros organismos oficiales y produjo una amplia gama de mapas y atlas comerciales, especialmente de América del Norte. Su trabajo como cartógrafo despertó un interés en las costumbres de vestimenta local de las tierras que inspeccionó y cartografió, que se muestran brillantemente en esta colección.

La fascinación por las tierras lejanas y los viajes por placer eran fenómenos relativamente nuevos a finales del siglo XVIII, y colecciones como ésta se hicieron populares, presentando tanto al turista como al viajero de sillón a los habitantes de otros países. La diversidad de dibujos en los volúmenes de Jefferys habla vívidamente de la singularidad e individualidad de las naciones del mundo hace unos 200 años. Los códigos de vestimenta han cambiado desde entonces, y la diversidad por región y país, tan rica en ese momento, se ha desvanecido. En la actualidad, a menudo es difícil distinguir al habitante de un continente de otro. Quizás, tratando de verlo con optimismo, hemos cambiado una diversidad cultural y visual por una vida personal más variada. O una vida intelectual y técnica más variada e interesante.

En un momento en el que es difícil diferenciar un libro de computadora de otro, Manning celebra la inventiva y la iniciativa del negocio de las computadoras con portadas de libros basadas en la rica diversidad de la vida regional de hace dos siglos, revivida por las imágenes de Jeffreys.

expresiones de gratitud

Este libro mostrará cómo Spring Boot puede lidiar automáticamente con las cosas detrás de escena que entran en una aplicación, lo que le permite concentrarse en las tareas que hacen que su aplicación sea única. En muchos sentidos, esto es análogo a lo que se hizo para hacer realidad este libro. Había tantas otras personas que se ocupaban de hacer que las cosas sucedieran y yo era libre de concentrarme en escribir el contenido del libro. Por encargarme del trabajo detrás de escena en Manning, me gustaría agradecer a Cynthia Kane, Robert Casazza, Andy Carroll, Corbin Collins, Kevin Sullivan, Mary Piergies, Janet Vail, Ozren Harlovic y Candace Gillhoolley.

Las pruebas de escritura le ayudan a saber si su software está cumpliendo sus objetivos. Del mismo modo, aquellos que revisaron *Spring Boot en acción* mientras aún se estaba escribiendo me dio la retroalimentación que necesitaba para asegurarme de que el libro se mantuviera en el objetivo. Por esto, mi agradecimiento para Aykut Acikel, Bachir Chihani, Eric Kramer, Francesco Persico, Furkan Kamaci, Gregor Zurowski, Mario Arias, Michael A. Angelo, Mykel Alvis, Norbert Kuchenmeister, Phil Whiles, Raphael Villela, Sam Kreter, Travis Nelson, Wilfredo R. Ronsini Jr. y William Fly. Un agradecimiento especial a John Guthrie por una revisión técnica final poco antes de que el manuscrito entrara en producción. Y un agradecimiento muy especial a Andrew Glover por contribuir con el prólogo de mi libro.

Por supuesto, este libro no sería posible ni siquiera necesario sin el increíble trabajo realizado por los talentosos miembros del equipo de Spring. Es asombroso lo que haces, y estoy muy emocionado de ser parte de un equipo que está cambiando la forma en que se desarrolla el software.

Muchas gracias a todos los involucrados en la gira No Fluff / Just Stuff, ya sean mis compañeros presentadores o aquellos que se acercan para escucharnos hablar. Las conversaciones que hemos tenido han contribuido de alguna manera a cómo se formó este libro.

Un libro como este no sería posible sin un alfabeto para componer en palabras. Así que, al igual que en mi libro anterior, me gustaría aprovechar esta oportunidad para agradecer a los fenicios por la invención del primer alfabeto.

Por último, pero ciertamente no menos importante ... mi amor, devoción y gracias van para mi hermosa esposa Raymie y mis increíbles hijas, Maisy y Madi. Una vez más, has tolerado otro proyecto de escritura. Ahora que está hecho, deberíamos ir a Disney World. ¿Qué dices?

1

Primavera de arranque

Este capítulo cubre

- Cómo Spring Boot simplifica el desarrollo de aplicaciones Spring
- Las características esenciales de Spring Boot
- Configurar un espacio de trabajo de Spring Boot

Spring Framework ha existido durante más de una década y ha encontrado un lugar como el marco estándar de facto para desarrollar aplicaciones Java. Con una historia tan larga e histórica, algunos podrían pensar que Spring se ha asentado, descansando en sus laureles, y no está haciendo nada nuevo o emocionante. Algunos incluso podrían decir que Spring es un legado y que es hora de buscar innovación en otro lado.

Algunos estarían equivocados.

Hay muchas cosas nuevas e interesantes que están ocurriendo en el ecosistema de Spring, incluido el trabajo en las áreas de computación en la nube, big data, persistencia de datos sin esquema, programación reactiva y desarrollo de aplicaciones del lado del cliente.

Quizás la novedad más emocionante, llamativa y revolucionaria que ha llegado a Spring en el último año es Spring Boot. Spring Boot ofrece un nuevo paradigma para desarrollar aplicaciones Spring con mínima fricción. Con Spring Boot, podrá desarrollar aplicaciones Spring con más agilidad y podrá

concéntrese en abordar las necesidades de funcionalidad de su aplicación con una idea mínima (o posiblemente nula) de configurar Spring. De hecho, una de las cosas principales que hace Spring Boot es sacar a Spring de su camino para que pueda hacer las cosas.

A lo largo de los capítulos de este libro, exploraremos varias facetas del desarrollo de Spring Boot. Pero primero, echemos un vistazo de alto nivel a lo que Spring Boot tiene para ofrecer.

1.1 *Spring reiniciado*

Spring comenzó como una alternativa ligera a Java Enterprise Edition (JEE, o J2EE como se conocía en ese momento). En lugar de desarrollar componentes como Enterprise JavaBeans (EJB) pesados, Spring ofreció un enfoque más simple para el desarrollo de Java empresarial, utilizando la inyección de dependencias y la programación orientada a aspectos para lograr las capacidades de EJB con objetos Java antiguos (POJO).

Pero mientras Spring era liviano en términos de código de componentes, era pesado en términos de configuración. Inicialmente, Spring se configuró con XML (y mucho). Spring 2.5 introdujo el escaneo de componentes basado en anotaciones, que eliminó una gran cantidad de configuración XML explícita para los propios componentes de una aplicación. Y Spring 3.0 introdujo una configuración basada en Java como una opción de tipo seguro y refactorizable a XML.

Aun así, no hubo escapatoria de la configuración. Habilitar ciertas funciones de Spring, como la gestión de transacciones y Spring MVC, requería una configuración explícita, ya sea en XML o Java. Habilitar funciones de bibliotecas de terceros, como las vistas web basadas en Thymeleaf, requería una configuración explícita. Configurar servlets y filtros (como Spring's `DispatcherServlet`) requiere configuración explícita en `web.xml` o en un inicializador de servlet. La configuración reducida de escaneo de componentes y la configuración de Java lo hicieron menos incómodo, pero Spring aún requería mucha configuración.

Toda esa configuración representa fricción en el desarrollo. Cualquier tiempo dedicado a escribir la configuración es tiempo que no se dedica a escribir la lógica de la aplicación. El cambio mental necesario para pensar en la configuración de una función de Spring distrae la atención de la solución del problema empresarial. Como cualquier marco, Spring hace mucho por ti, pero exige que hagas mucho por él a cambio.

Además, la gestión de la dependencia del proyecto es una tarea ingrata. Decidir qué bibliotecas deben formar parte de la construcción del proyecto es bastante complicado. Pero es aún más difícil saber qué versiones de esas bibliotecas funcionarán bien con otras.

Por importante que sea, la gestión de la dependencia es otra forma de fricción. Cuando agrega dependencias a su compilación, no está escribiendo código de aplicación. Cualquier incompatibilidad que provenga de seleccionar las versiones incorrectas de esas dependencias puede ser un verdadero asesino de la productividad.

Spring Boot ha cambiado todo eso.

1.1.1 *Una nueva mirada a Spring*

Suponga que tiene la tarea de desarrollar una aplicación web Hello World muy simple con Spring. ¿Qué necesitarías hacer? Puedo pensar en un puñado de cosas que necesitarías como mínimo:

- Una estructura de proyecto, completa con un archivo de compilación de Maven o Gradle que incluye las dependencias necesarias. Como mínimo, necesitará Spring MVC y la API de Servlet expresados como dependencias.
- Archivo Aweb.xml (o un WebApplicationInitializer implementación) que declara Muelles DispatcherServlet.
- Una configuración de Spring que habilita Spring MVC.
- Una clase de controlador que responderá a las solicitudes HTTP con "Hello World".
- Un servidor de aplicaciones web, como Tomcat, para implementar la aplicación.

Lo más sorprendente de esta lista es que solo un elemento es específico para desarrollar la funcionalidad Hello World: el controlador. El resto es un texto estándar genérico que necesitaría para cualquier aplicación web desarrollada con Spring. Pero si todas las aplicaciones web de Spring lo necesitan, ¿por qué debería proporcionarlo?

Suponga por un momento que el controlador es todo lo que necesita. Como resultado, la clase de controlador basada en Groovy que se muestra en el listado 1.1 es una aplicación Spring completa (aunque simple).

Listado 1.1 Una aplicación Spring completa basada en Groovy

```
@RestController
class HelloController {

    @RequestMapping("/")
    def hola () {
        volver "Hola mundo"
    }

}
```

No hay configuración. Sin web.xml. Sin especificación de construcción. Ni siquiera un servidor de aplicaciones. Esta es la aplicación completa. Spring Boot se encargará de la logística de ejecución de la aplicación. Solo necesitas traer el código de la aplicación.

Suponiendo que tiene instalada la interfaz de línea de comandos (CLI) de Spring Boot, puede ejecutar HelloController en la línea de comando así:

```
$ spring run HelloController.groovy
```

También puede haber notado que ni siquiera era necesario compilar el código. Spring Boot CLI pudo ejecutarlo desde su forma no compilada.

Elegí escribir este controlador de ejemplo en Groovy porque la simplicidad del lenguaje Groovy se presenta bien junto con la simplicidad de Spring Boot. Pero Spring Boot no requiere que uses Groovy. De hecho, gran parte del código que escribiremos en este libro estará en Java. Pero habrá algo de Groovy aquí y allá, cuando corresponda.

No dude en consultar la sección 1.21 para ver cómo instalar Spring Boot CLI, para que pueda probar esta pequeña aplicación web. Pero por ahora, veremos las piezas clave de Spring Boot para ver cómo cambia el desarrollo de la aplicación Spring.

1.1.2 *Examinando lo esencial de Spring Boot*

Spring Boot aporta una gran cantidad de magia al desarrollo de aplicaciones Spring. Pero hay cuatro trucos básicos que realiza:

- *Configuración automática*—Spring Boot puede proporcionar automáticamente la configuración para la funcionalidad de la aplicación común a muchas aplicaciones de Spring.
- *Dependencias de arranque*—Dile a Spring Boot qué tipo de funcionalidad necesitas y se asegurará de que las bibliotecas necesarias se agreguen a la compilación.
- *La interfaz de línea de comandos*—Esta característica opcional de Spring Boot le permite escribir aplicaciones completas con solo el código de la aplicación, pero sin necesidad de construir un proyecto tradicional.
- *El actuador*—Le da una idea de lo que sucede dentro de una aplicación Spring Boot en ejecución.

Cada una de estas características sirve para simplificar el desarrollo de aplicaciones Spring en su propio camino. Veremos cómo emplearlos al máximo a lo largo de este libro. Pero por ahora, echemos un vistazo rápido a lo que ofrece cada uno.

AUTO-CONFIGURACIÓN

En el código fuente de cualquier aplicación Spring, encontrará la configuración de Java o la configuración XML (o ambas) que habilita ciertas características y funcionalidades de soporte para la aplicación. Por ejemplo, si alguna vez ha escrito una aplicación que accede a una base de datos relacional con JDBC, probablemente haya configurado Spring's `JdbcTemplate` como un bean en el contexto de la aplicación Spring. Apuesto a que la configuración se parecía mucho a esto:

```
@Frijol
public JdbcTemplate jdbcTemplate (DataSource dataSource) {
    return new JdbcTemplate (dataSource);
}
```

Esta declaración de `frijol` muy simple crea una instancia de `JdbcTemplate`, inyectándolo con su única dependencia, un Fuente de datos. Por supuesto, eso significa que también deberá configurar un Fuente de datos bean para que se cumpla la dependencia. Para completar este escenario de configuración, suponga que debe configurar una base de datos H2 incorporada como el Fuente de datos `frijol`:

```
@Frijol
fuente de datos pública fuente de datos () {
    devolver nuevo EmbeddedDatabaseBuilder ()
        . setType (EmbeddedDatabaseType.H2)
        . addScripts ('esquema.sql', 'datos.sql')
        . construir();
}
```

Este método de configuración de bean crea una base de datos incorporada, especificando dos scripts SQL para ejecutar en la base de datos incorporada. Los `construir()` el método devuelve un Fuente de datos que hace referencia a la base de datos incrustada.

Ninguno de estos dos métodos de configuración de frijoles es terriblemente complejo o extenso. Pero representan solo una fracción de la configuración en una aplicación Spring típica. Además, hay innumerables aplicaciones de Spring que tendrán exactamente estos mismos métodos. Cualquier aplicación que necesite una base de datos integrada y JdbcTemplate necesitará esos métodos. En resumen, es una configuración estándar.

Si es tan común, ¿por qué debería escribirlo?

Spring Boot puede configurar automáticamente estos escenarios de configuración comunes. Si Spring Boot detecta que tiene la biblioteca de la base de datos H2 en la ruta de clase de su aplicación, configurará automáticamente una base de datos H2 incorporada. Si JdbcTemplate está en la ruta de clases, entonces también configurará un JdbcTemplate frijol para ti. No es necesario que se preocupe por configurar esos beans. Estarán configurados para usted, listos para inyectarse en cualquiera de los beans que escriba.

Hay mucho más en la configuración automática de Spring Boot que las bases de datos integradas y JdbcTemplate. Hay varias docenas de formas en que Spring Boot puede quitarle la carga de la configuración, incluida la configuración automática para la API de persistencia de Java (JPA), las plantillas de Thymeleaf, la seguridad y Spring MVC. Nos sumergiremos en la configuración automática a partir del capítulo 2.

DEPENDENCIAS DEL ARRANQUE

Puede ser un desafío agregar dependencias a la construcción de un proyecto. ¿Qué biblioteca necesitas? ¿Cuáles son su grupo y artefacto? ¿Qué versión necesitas? ¿Esa versión funcionará bien con otras dependencias en el mismo proyecto?

Spring Boot ofrece ayuda con la gestión de la dependencia del proyecto mediante dependencias de inicio. Las dependencias iniciales son realmente dependencias especiales de Maven (y Gradle) que aprovechan la resolución de dependencia transitiva para agregar bibliotecas de uso común en un puñado de dependencias definidas por funciones.

Por ejemplo, suponga que va a crear una API REST con Spring MVC que funciona con representaciones de recursos JSON. Además, desea aplicar la validación declarativa según la especificación JSR-303 y servir la aplicación mediante un servidor Tomcat integrado. Para lograr todo esto, necesitará (como mínimo) las siguientes ocho dependencias en su compilación de Maven o Gradle:

- org.springframework: spring-core
- org.springframework: spring-web
- org.springframework: spring-webmvc
- com.fasterxml.jackson.core: jackson-databind
- org.hibernate: hibernate-validator
- org.apache.tomcat.embed: tomcat-embed-core
- org.apache.tomcat.embed: tomcat-embed-el
- org.apache.tomcat.embed: tomcat-embed-logging-juli

Por otro lado, si aprovechara las dependencias del iniciador Spring Boot, simplemente podría agregar el iniciador "web" Spring Boot (org.springframework

. arranque: spring-boot-starter-web) como una dependencia de compilación. Esta dependencia única

atraerá transitivamente todas esas otras dependencias para que no tenga que pedir las todas.

Pero hay algo más sutil acerca de las dependencias de inicio que simplemente reducir el recuento de dependencias de compilación. Tenga en cuenta que al agregar el iniciador "web" a su compilación, está especificando un tipo de funcionalidad que su aplicación necesita. Su aplicación es una aplicación web, por lo que agrega el iniciador "web". Del mismo modo, si su aplicación utilizará la persistencia JPA, puede agregar el iniciador "jpa". Si necesita seguridad, puede agregar el iniciador de "seguridad". En resumen, ya no necesita pensar en qué bibliotecas necesitará para admitir determinadas funciones; simplemente solicita esa funcionalidad a través de la dependencia de arranque pertinente.

También tenga en cuenta que las dependencias de arranque de Spring Boot le liberan de preocuparse por las versiones de estas bibliotecas que necesita. Las versiones de las bibliotecas que extraen los principiantes se han probado juntas para que pueda estar seguro de que no habrá incompatibilidades entre ellas.

Junto con la configuración automática, comenzaremos a usar las dependencias de inicio de inmediato, comenzando en el capítulo 2.

LA INTERFAZ DE LÍNEA DE COMANDOS (CLI)

Además de la configuración automática y las dependencias de inicio, Spring Boot también ofrece una nueva forma intrigante de escribir rápidamente aplicaciones Spring. Como vio anteriormente en la sección 1.1, Spring Boot CLI hace posible escribir aplicaciones haciendo más que escribir el código de la aplicación.

La CLI de Spring Boot aprovecha las dependencias de inicio y la configuración automática para permitirle concentrarse en escribir código. No solo eso, ¿te diste cuenta de que no hay importar líneas en el listado 1.1? ¿Cómo supo la CLI qué paquetes RequestMapping y RestController ¿viene de? De hecho, ¿cómo terminaron esas clases en el classpath?

La respuesta corta es que la CLI detectó que se están utilizando esos tipos y sabe qué dependencias de inicio agregar a la ruta de clases para que funcione. Una vez que esas dependencias están en la ruta de clases, se activa una serie de configuración automática que garantiza que DispatcherServlet y Spring MVC están habilitados para que el controlador pueda responder a las solicitudes HTTP.

La CLI de Spring Boot es una pieza opcional del poder de Spring Boot. Aunque proporciona un enorme poder y simplicidad para el desarrollo de Spring, también presenta un modelo de desarrollo bastante poco convencional. Si este modelo de desarrollo es demasiado extremo para su gusto, no hay problema. Aún puede aprovechar todo lo demás que Spring Boot tiene para ofrecer incluso si no usa la CLI. Pero si le gusta lo que ofrece la CLI, definitivamente querrá ver el capítulo 5 donde profundizaremos en la CLI de Spring Boot.

EL ACTUADOR

La última pieza del rompecabezas Spring Boot es el actuador. Mientras que las otras partes de Spring Boot simplifican el desarrollo de Spring, el Actuador ofrece en cambio la capacidad de inspeccionar las partes internas de su aplicación en tiempo de ejecución. Con el actuador instalado, puede inspeccionar el funcionamiento interno de su aplicación, incluidos detalles como

- Qué beans se han configurado en el contexto de la aplicación Spring Qué decisiones se tomaron
- mediante la configuración automática de Spring Boot
- Qué variables de entorno, propiedades del sistema, propiedades de configuración y argumentos de la línea de comandos están disponibles para su aplicación
- El estado actual de los subprocesos en su aplicación y el soporte de su aplicación.
-
- Varias métricas relacionadas con el uso de memoria, recolección de basura, solicitudes web y uso de fuentes de datos

El actuador expone esta información de dos formas: a través de puntos finales web o mediante una interfaz de shell. En el último caso, puede abrir un shell seguro (SSH) en su aplicación y emitir comandos para inspeccionar su aplicación mientras se ejecuta.

Exploraremos las capacidades del actuador en detalle cuando lleguemos al capítulo 7.

1.1.3 *Lo que no es Spring Boot*

Debido a las cosas increíbles que hace Spring Boot, se ha hablado mucho sobre Spring Boot durante el último año. Dependiendo de lo que haya escuchado o leído sobre Spring Boot antes de leer este libro, es posible que tenga algunos conceptos erróneos acerca de Spring Boot que deben aclararse antes de continuar.

Primero, Spring Boot no es un servidor de aplicaciones. Esta idea errónea se deriva del hecho de que es posible crear aplicaciones web como archivos JAR autoejecutables que se pueden ejecutar en la línea de comandos sin implementar aplicaciones en un servidor de aplicaciones Java convencional. Spring Boot logra esto incorporando un contenedor de servlets (Tomcat, Jetty o Undertow) dentro de la aplicación. Pero es el contenedor de servlets integrado el que proporciona la funcionalidad del servidor de aplicaciones, no Spring Boot en sí.

Del mismo modo, Spring Boot no implementa ninguna especificación empresarial de Java como JPA o JMS. Admite varias especificaciones empresariales de Java, pero lo hace configurando automáticamente beans en Spring que admiten esas funciones. Por ejemplo, Spring Boot no implementa JPA, pero es compatible con JPA al configurar automáticamente los beans apropiados para una implementación de JPA (como Hibernate).

Finalmente, Spring Boot no emplea ninguna forma de generación de código para lograr su magia. En cambio, aprovecha las características de configuración condicional de Spring 4, junto con la resolución de dependencia transitiva ofrecida por Maven y Gradle, para configurar automáticamente beans en el contexto de la aplicación Spring.

En resumen, en el fondo, Spring Boot es solo primavera. En el interior, Spring Boot está haciendo el mismo tipo de configuración de bean en Spring que podría hacer por su cuenta si Spring Boot no existiera. Afortunadamente, debido a que Spring Boot existe, no tiene que lidiar con la configuración estándar explícita y puede concentrarse en la lógica que hace que su aplicación sea única.

A estas alturas ya debería tener una idea general de lo que Spring Boot aporta. Ya es hora de que cree su primera aplicación con Spring Boot. Pero lo primero es lo primero. Veamos cómo puedes dar tus primeros pasos con Spring Boot.

1.2 *Empezando con Spring Boot*

En última instancia, un proyecto Spring Boot es solo un proyecto Spring regular que aprovecha los arrancadores y la configuración automática de Spring Boot. Por lo tanto, cualquier técnica o herramienta con la que ya esté familiarizado para crear un proyecto Spring desde cero se aplicará a un proyecto Spring Boot. Sin embargo, hay algunas opciones convenientes disponibles para iniciar su proyecto con Spring Boot.

La forma más rápida de comenzar con Spring Boot es instalar Spring Boot CLI para que pueda comenzar a escribir código, como el del listado 1.1, que se ejecuta a través de la CLI.

1.2.1 *Instalación de la CLI de Spring Boot*

Como comentamos anteriormente, Spring Boot CLI ofrece un enfoque interesante, aunque poco convencional, para desarrollar aplicaciones Spring. Nos sumergiremos en los detalles de lo que ofrece la CLI en el capítulo 5. Pero, por ahora, veamos cómo instalar Spring Boot CLI para que pueda ejecutar el código que vimos en el listado 1.1.

Hay varias formas de instalar Spring Boot CLI:

- De una distribución descargada
- Uso del Groovy Environment Manager con OS X
- Homebrew
- Como puerto usando MacPorts

Bien mire cada opción de instalación. Además, también veremos cómo instalar el soporte para completar el comando Spring Boot CLI, que es útil si está utilizando CLI en los shells BASH o zsh (lo siento, usuarios de Windows). Primero veamos cómo puede instalar Spring Boot CLI manualmente desde una distribución.

INSTALACIÓN MANUAL DE LA CLI SPRING BOOT

Quizás la forma más sencilla de instalar Spring Boot CLI es descargarlo, descomprimirlo y agregar su directorio bin a su ruta. Puede descargar el archivo de distribución desde cualquiera de estas ubicaciones:

- [http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.3.0.RELEASE / spring-boot-cli-1.3.0.RELEASE-bin.zip](http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.3.0.RELEASE/spring-boot-cli-1.3.0.RELEASE-bin.zip)
- [http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.3.0.RELEASE / spring-boot-cli-1.3.0.RELEASE-bin.tar.gz](http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.3.0.RELEASE/spring-boot-cli-1.3.0.RELEASE-bin.tar.gz)

Una vez que haya descargado la distribución, descomprímala en algún lugar de su sistema de archivos. Dentro del archivo descomprimido, encontrará un directorio bin que contiene un script spring.bat (para Windows) y un script spring para Unix. Agregue este directorio bin a la ruta de su sistema y estará listo para usar Spring Boot CLI.

VINCULACIÓN SIMBÓLICA A LA BOTA DE PRIMAVERA Si está utilizando Spring Boot CLI en una máquina Unix, puede ser útil crear un enlace simbólico al archivo descomprimido y agregar el enlace simbólico a su ruta en lugar del directorio real. Esto facilitará la actualización a una versión más nueva de Spring Boot más adelante (o incluso cambiar entre versiones) simplemente reasignando el enlace simbólico al directorio de la nueva versión.

Puede patear un poco los neumáticos en la instalación verificando la versión de la CLI que se instaló:

```
$ spring --version
```

Si todo está funcionando, se le mostrará la versión de Spring Boot CLI que se instaló.

Aunque esta es la instalación manual, es una opción fácil que no requiere que tengas nada adicional instalado. Si es un usuario de Windows, también es la única opción disponible para usted. Pero si está en una máquina Unix y está buscando algo un poco más automatizado, entonces quizás el Software Development Kit Manager pueda ayudar.

INSTALACIÓN CON EL ADMINISTRADOR DEL KIT DE DESARROLLO DE SOFTWARE

El Software Development Kit Manager (SDKMAN; anteriormente conocido como GVM) se puede utilizar para instalar y administrar múltiples versiones de las instalaciones de Spring Boot CLI. Para utilizar SDKMAN, deberá obtener e instalar la herramienta SDKMAN de [http:// sdkman](http://sdkman.io) .io . La forma más sencilla de instalar SDKMAN es en la línea de comandos:

```
$ curl -s get.sdkman.io | intento
```

Siga las instrucciones dadas en el resultado para completar la instalación de SDKMAN. Para mi máquina, tuve que realizar el siguiente comando en la línea de comandos:

```
$ fuente "/Users/habuma/.sdkman/bin/sdkman-init.sh"
```

Tenga en cuenta que este comando será diferente para diferentes usuarios. En mi caso, mi directorio personal está en / Usuarios / habuma, así que esa es la raíz de la ruta del script de shell. Querrá hacer los ajustes necesarios para adaptarse a su situación.

Una vez que SDKMAN está instalado, puede instalar la CLI de Spring Boot de esta manera:

```
$ sdk instalar springboot $ spring --version
```

Suponiendo que todo vaya bien, se le mostrará la versión actual de Spring Boot.

Si desea actualizar a una versión más reciente de Spring Boot CLI, solo necesita instalarla y comenzar a usarla. Para averiguar qué versiones de Spring Boot CLI están disponibles, use SDKMAN's lista mando:

```
$ sdk list springboot
```

lo lista El comando muestra todas las versiones disponibles, incluidas las que están instaladas y las que están en uso. De esta lista, puede elegir instalar una versión y luego usarla. Por ejemplo, para instalar Spring Boot CLI versión 1.3.0.RELEASE, usaría el Instalar en pc comando, especificando la versión:

```
$ sdk instalar springboot 1.3.0.RELEASE
```

Esto instalará la nueva versión y le preguntará si desea convertirla en la versión predeterminada. Si elige no convertirla en la versión predeterminada o si desea cambiar a una versión diferente, puede usar el usar mando:

```
$ sdk usa springboot 1.3.0.RELEASE
```

Si desea que esa versión sea la predeterminada para todos los shells, use la defecto mando:

```
$ sdk springboot 1.3.0.RELEASE predeterminado
```

Lo bueno de usar SDKMAN para administrar su instalación de Spring Boot CLI es que le permite cambiar fácilmente entre diferentes versiones de Spring Boot. Esto le permitirá probar compilaciones de instantáneas, hitos y versiones candidatas antes de que se publiquen formalmente, pero aún así volver a una versión estable para otros trabajos.

INSTALACIÓN CON HOMEBREW

Si va a desarrollar en una máquina OS X, tiene la opción de usar Homebrew para instalar Spring Boot CLI. Homebrew es un administrador de paquetes para OS X que se utiliza para instalar muchas aplicaciones y herramientas diferentes. La forma más sencilla de instalar Homebrew es ejecutando el script de instalación Ruby:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master / install) "
```

Puede leer más sobre Homebrew (y encontrar otras opciones de instalación) en <http://brew.sh>.

Para instalar Spring Boot CLI usando Homebrew, deberá "tocar" el toque de Pivotal: ¹

```
$ brew tap pivotal / tap
```

Ahora que Homebrew está tocando el grifo de Pivotal, puede instalar SpringBoot CLI así:

```
$ brew instalar springboot
```

Homebrew instalará Spring Boot CLI en /usr/local/bin, y estará listo para funcionar. Puede verificar la instalación comprobando la versión que se instaló:

```
$ spring --version
```

Debería responder mostrándole la versión de Spring Boot que se instaló. También puede intentar ejecutar el código en el listado 1.1.

INSTALACIÓN CON MACPORTS

Otra opción de instalación de Spring Boot CLI para usuarios de OS X es usar MacPorts, otro instalador popular para Mac OS X. Para usar MacPorts para instalar Spring Boot

¹ Tocar es una forma de agregar repositorios adicionales a aquellos desde los que trabaja Homebrew. Pivotal, la empresa detrás de Spring y Spring Boot, ha hecho que Spring Boot CLI esté disponible a través de su grifo.

CLI, primero debe instalar MacPorts, que a su vez requiere que tenga instalado Xcode. Además, los pasos para instalar MacPorts varían según la versión de OS X que esté utilizando. Por tanto, te recomiendo <https://www.macports.org/install.php> para obtener instrucciones sobre la instalación de MacPorts.

Una vez que tenga MacPorts instalado, puede instalar Spring Boot CLI en la línea de comando como esta:

```
$ sudo instalación del puerto spring-boot-cli
```

MacPorts instalará Spring Boot CLI en /opt/local/share/java/spring-boot-cli y colocará un enlace simbólico al binario en /opt/local/bin, que ya debería estar en la ruta del sistema desde la instalación de MacPorts. Puede verificar la instalación comprobando la versión que se instaló:

```
$ spring --version
```

Debería responder mostrándole la versión de Spring Boot que se instaló. También puede intentar ejecutar el código en el listado 1.1.

HABILITAR LA FINALIZACIÓN DE LA LÍNEA DE COMANDOS

La CLI de Spring Boot ofrece un puñado de comandos para ejecutar, empaquetar y probar su aplicación basada en CLI. Además, cada uno de esos comandos tiene varias opciones. Puede resultar difícil recordar todo lo que ofrece la CLI. La finalización de la línea de comandos puede ayudarlo a recordar cómo usar Spring Boot CLI.

Si ha instalado Spring Boot CLI con Homebrew, ya tiene instalada la finalización de la línea de comandos. Pero si instaló Spring Boot manualmente o con SDKMAN, deberá obtener los scripts o instalar los scripts de finalización manualmente. (La finalización de la línea de comandos no es una opción si ha instalado Spring Boot CLI a través de MacPorts).

Los scripts de finalización se encuentran en el directorio de instalación de Spring Boot CLI bajo el subdirectorio de finalización de shell. Hay dos scripts diferentes, uno para BASH y otro para zsh. Para obtener el script de finalización para BASH, puede ingresar lo siguiente en la línea de comando (asumiendo una instalación de SDKMAN):

```
PS ~ / .sdkman / springboot / current / shell-complete / bash / spring
```

Esto le dará la finalización de la CLI de Spring Boot para el shell actual, pero tendrá que obtener este script nuevamente cada vez que inicie un nuevo shell para mantener esa función. Opcionalmente, puede copiar el script a su directorio de script personal o del sistema. La ubicación del directorio de scripts varía para las diferentes instalaciones de Unix, así que consulte la documentación de su sistema (o Google) para más detalles.

Con la finalización de comandos habilitada, debería poder escribir primavera en la línea de comando y luego presione la tecla Tab para que se le ofrezcan opciones sobre qué escribir a continuación. Una vez que haya elegido un comando, escriba - (guión doble) y luego presione Tab nuevamente para que se muestre una lista de opciones para ese comando.

Si está desarrollando en Windows o no está usando BASH o zsh, no puede usar estos scripts de finalización de la línea de comandos. Aun así, puede completar el comando si ejecuta el shell CLI de Spring Boot:

```
$ concha de primavera
```

A diferencia de los scripts de finalización de comandos para BASH y zsh (que operan dentro del shell BASH / zsh), el shell CLI de Spring Boot abre un nuevo shell específico de Spring Boot. Desde este shell, puede ejecutar cualquiera de los comandos de la CLI y completar el comando con la tecla Tab.

La CLI de Spring Boot ofrece una manera fácil de comenzar con Spring Boot y crear prototipos de aplicaciones simples. Como veremos más adelante en el capítulo 8, también se puede usar para aplicaciones listas para producción, dado el entorno de ejecución de producción adecuado.

Aun así, el proceso de Spring Boot CLI es bastante poco convencional en contraste con cómo se desarrollan la mayoría de los proyectos de Java. Normalmente, los proyectos de Java utilizan herramientas como Gradle o Maven para crear archivos WAR que se implementan en un servidor de aplicaciones. Si el modelo CLI se siente un poco incómodo, aún puede aprovechar la mayoría de las características de Spring Boot en el contexto de un proyecto Java construido tradicionalmente.² Y Spring Initializr puede ayudarlo a comenzar.

1.2.2 *Iniciando un proyecto Spring Boot con Spring Initializr*

A veces, la parte más difícil de un proyecto es comenzar. Debe configurar una estructura de directorios para varios artefactos del proyecto, crear un archivo de compilación y completar el archivo de compilación con dependencias. Spring Boot CLI elimina gran parte de este trabajo de configuración, pero si prefiere una estructura de proyecto Java más tradicional, querrá mirar Spring Initializr.

Spring Initializr es, en última instancia, una aplicación web que puede generar una estructura de proyecto Spring Boot para usted. No genera ningún código de aplicación, pero le dará una estructura de proyecto básica y una especificación de compilación de Maven o Gradle para compilar su código. Todo lo que necesita hacer es escribir el código de la aplicación.

Spring Initializr se puede utilizar de varias formas:

- A través de una interfaz basada en web a través de
- Spring Tool Suite
- A través de IntelliJ IDEA
- Usando la CLI de Spring Boot

Bien mire cómo usar cada una de estas interfaces para Initializr, comenzando con la web-interfaz basada.

² Solo renunciará a las funciones que requieren la flexibilidad del lenguaje Groovy, como la dependencia automática de importación y resolución de importación.

USO DE LA INTERFAZ WEB DE SPRING INITIALIZR

La forma más sencilla de utilizar Spring Initializr es apuntar su navegador web a <http://start.spring.io>. Debería ver un formulario similar al de la figura 1.1.

Las dos primeras cosas que pregunta el formulario es si desea construir su proyecto con Maven o Gradle y qué versión de Spring Boot usar. De forma predeterminada, es un proyecto de Maven que usa la última versión (sin hitos, sin instantáneas) de Spring Boot, pero puede elegir una diferente.

En el lado izquierdo del formulario, se le pide que especifique algunos metadatos del proyecto. Como mínimo, debe proporcionar el grupo y el artefacto del proyecto. Pero si hace clic en el enlace "Cambiar a la versión completa", puede especificar metadatos adicionales, como la versión y el nombre del paquete base. Estos metadatos se utilizan para completar el archivo pom.xml de Maven generado (o el archivo build.gradle de Gradle).

start.spring.io

Spring Initializr

SPRING INITIALIZR

bootstrap your application now

Generate a **Maven Project** with Spring Boot **1.3.0**

Project Metadata

Artifact coordinates

Group

Artifact

Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

- Web**
Full-stack web development with Tomcat and Spring MVC
- Rest Repositories**
Exposing Spring Data repositories over REST via spring-data-rest-webmvc
- Websocket**
Websocket development with SockJS and STOMP
- WS**
Contract-first SOAP service development with Spring Web Services
- Jersey (JAX-RS)**
the Jersey RESTful Web Services framework

Figura 1.1 Spring Initializr es una aplicación web que genera proyectos Spring vacíos como puntos de partida para el desarrollo.

En el lado derecho del formulario, se le pide que especifique las dependencias del proyecto. La forma más sencilla de hacerlo es escribir el nombre de una dependencia en el cuadro de texto. A medida que escribe, aparecerá una lista de dependencias coincidentes. Seleccione el (los) que desee y se agregará al proyecto. Si no ve lo que está buscando, haga clic en el enlace "Cambiar a la versión completa" para obtener una lista completa de las dependencias disponibles.

Si ha echado un vistazo al apéndice B, entonces reconocerá que las dependencias ofrecidas corresponden a las dependencias de arranque de Spring Boot. De hecho, al seleccionar cualquiera de estas dependencias, le está diciendo a Initializr que agregue los iniciadores como dependencias al archivo de compilación del proyecto. (Hablaemos más sobre los arrancadores de Spring Boot en el capítulo 2.)

Una vez que haya completado el formulario y haya realizado sus selecciones de dependencia, haga clic en el botón Generar proyecto para que Spring Initializr genere un proyecto por usted. El proyecto que genera se le presentará como un archivo zip (cuyo nombre está determinado por el valor en el campo Artefacto) que descarga su navegador. El contenido del archivo zip variará ligeramente, según las elecciones que haya realizado antes de hacer clic en Generar proyecto. En cualquier caso, el archivo zip contendrá un proyecto básico para comenzar a desarrollar una aplicación con Spring Boot.

Por ejemplo, suponga que debe especificar lo siguiente en Spring Initializr:

- Artefacto: myapp
- Nombre del paquete: myapp
- Tipo: Proyecto Gradle
- Dependencias: Web y JPA

Después al hacer clic en Generar proyecto, se le proporcionará un archivo zip llamado myapp.zip. Después descomprimiéndolo, tendría una estructura de proyecto similar a la que se muestra en la figura 1.2.

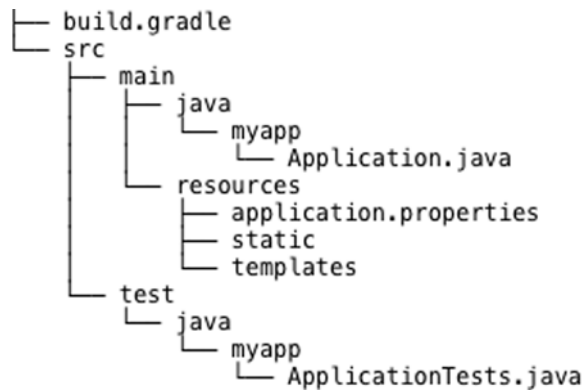


Figura 1.2 Los proyectos creados por Initializr proporcionan una base mínima sobre la que construir aplicaciones Spring Boot.

Como puede ver, hay muy poco código en este proyecto. Aparte de un par de directorios vacíos, también incluye lo siguiente:

- `build.gradle`: una especificación de compilación de Gradle. Si hubiera elegido un proyecto de Maven, este sería reemplazado por `pom.xml`.
- `Application.java`: una clase con `principal()` método para arrancar la aplicación. `ApplicationTests.java`: una clase de prueba JUnit vacía instrumentada para cargar un contexto de aplicación Spring mediante la configuración automática de Spring Boot.
- `application.properties`: un archivo de propiedades vacío al que puede agregar propiedades de configuración como mejor le parezca.

Incluso los directorios vacíos tienen importancia en una aplicación Spring Boot. La estática

El directorio es donde puede colocar cualquier contenido estático (JavaScript, hojas de estilo, imágenes, etc.) para que se sirva desde la aplicación web. Y, como verá más adelante, puede colocar plantillas que representen datos del modelo en el directorio de plantillas.

Probablemente importará el proyecto creado por Initializr a su IDE de elección. Pero si Spring Tool Suite es su IDE de elección, puede crear el proyecto directamente en el IDE. Echemos un vistazo al soporte de Spring Tool Suite para crear proyectos Spring Boot.

CREACIÓN DE PROYECTOS SPRING BOOT EN SPRING TOOL SUITE

Conjunto de herramientas Spring³ ha sido durante mucho tiempo un IDE fantástico para desarrollar aplicaciones Spring. Desde la versión 3.4.0, también se ha integrado con Spring Initializr, por lo que es una excelente manera de comenzar con Spring Boot.

Para crear una nueva aplicación Spring Boot en Spring Tool Suite, seleccione el elemento de menú **Nuevo> Proyecto Spring Starter** en el menú **Archivo**. Cuando lo haga, Spring Tool Suite le presentará un cuadro de diálogo similar al que se muestra en la figura 1.3.

Como puede ver, este cuadro de diálogo solicita la misma información que el Spring Initializr basado en la web. De hecho, los datos que proporcione aquí se enviarán a Spring Initializr para crear un archivo zip del proyecto, al igual que con el formulario basado en web.

Si desea especificar en qué parte del sistema de archivos crear el proyecto o si desea agregarlo a un conjunto de trabajo específico dentro del IDE, haga clic en el botón **Siguiente**. Se le presentará un segundo cuadro de diálogo como el que se muestra en la figura 1.4.

El campo **Ubicación** especifica dónde residirá el proyecto en el sistema de archivos. Si aprovecha los conjuntos de trabajo de Eclipse para organizar sus proyectos, puede hacer que el proyecto se agregue a un conjunto de trabajo específico marcando la casilla de verificación **Agregar proyecto a conjuntos de trabajo** y seleccionando un conjunto de trabajo.

La sección **Información del sitio** simplemente describe la URL que se utilizará para contactar a Initializr. En su mayor parte, puede ignorar esta sección. Sin embargo, si tuviera que implementar su propio servidor Initializr (clonando el código en <https://github.com/spring-io/initializr>), puede ingresar la URL base de su Initializr aquí.

³ Spring Tool Suite es una distribución de Eclipse IDE que está equipada con varias características para ayudar con Spring desarrollo. Puede descargar Spring Tool Suite desde <http://spring.io/tools/sts>.

New Spring Starter Project

Name:

Packaging: Java Version:

Language:

Group:

Artifact:

Description:

Package Name:

Style

<input type="checkbox"/> Security	<input type="checkbox"/> AOP	<input type="checkbox"/> JDBC	<input type="checkbox"/> JPA
<input type="checkbox"/> MongoDB	<input type="checkbox"/> Redis	<input type="checkbox"/> Gemfire	<input type="checkbox"/> Solr
<input type="checkbox"/> Elasticsearch	<input type="checkbox"/> Batch	<input type="checkbox"/> Integration	<input type="checkbox"/> JMS
<input type="checkbox"/> AMQP	<input type="checkbox"/> Web	<input type="checkbox"/> Websocket	<input type="checkbox"/> WS
<input type="checkbox"/> Rest Repositories	<input type="checkbox"/> Mobile	<input type="checkbox"/> Freemarker	<input type="checkbox"/> Velocity
<input type="checkbox"/> Groovy Templates	<input type="checkbox"/> Thymeleaf	<input type="checkbox"/> Facebook	<input type="checkbox"/> LinkedIn
<input type="checkbox"/> Twitter	<input type="checkbox"/> Actuator	<input type="checkbox"/> Remote Shell	

Figura 1.3 Primavera Tool Suite se integra con Spring Initializr para crear e importe directamente proyectos Spring Boot en el IDE.

New Spring Starter Project

☒ Use default location

Location:

Working sets

☐ Add project to working sets

Working sets:

Site Info

Base Url:

Full Url:

Figura 1.4 El segundo-ondpageof theSpring Cuadro de diálogo Proyecto inicial box le ofrece la oportunidad de especificar donde se crea el proyecto.

Al hacer clic en el botón Finalizar, se inicia el proceso de generación e importación del proyecto. Es importante comprender que el cuadro de diálogo Spring Starter Project de Spring Tool Suite delega a Spring Initializr en <http://start.spring.io> para producir el proyecto. Debe estar conectado a Internet para que funcione.

Una vez que el proyecto se haya importado a su espacio de trabajo, estará listo para comenzar a desarrollar su aplicación. A medida que desarrolle la aplicación, encontrará que Spring Tool Suite tiene algunos trucos más específicos de Spring Boot bajo la manga. Por ejemplo, puede ejecutar su aplicación con un servidor integrado seleccionando Ejecutar como> Aplicación Spring Boot en el menú Ejecutar.

Es importante comprender que Spring Tool Suite se coordina con Initializr a través de una API REST. Por lo tanto, solo funcionará si se puede conectar al Initializr. Si su máquina de desarrollo está fuera de línea o Initializr está bloqueado por un firewall, entonces el uso del asistente Spring Start Project en Spring Tool Suite no funcionará.

CREANDO PROYECTOS DE BOTAS DE PRIMAVERA EN INTELLIJ IDEA

IntelliJ IDEA es un IDE muy popular y, a partir de IntelliJ IDEA 14.1, ahora es compatible con Spring Boot. ⁴

Para comenzar con una nueva aplicación Spring Boot en IntelliJ IDEA, seleccione Nuevo> Proyecto en el menú Archivo. Se le presentará la primera de un puñado de pantallas (que se muestran en la figura 1.5) que plantean preguntas similares a las formuladas por la aplicación web Initializr y Spring Tool Suite.

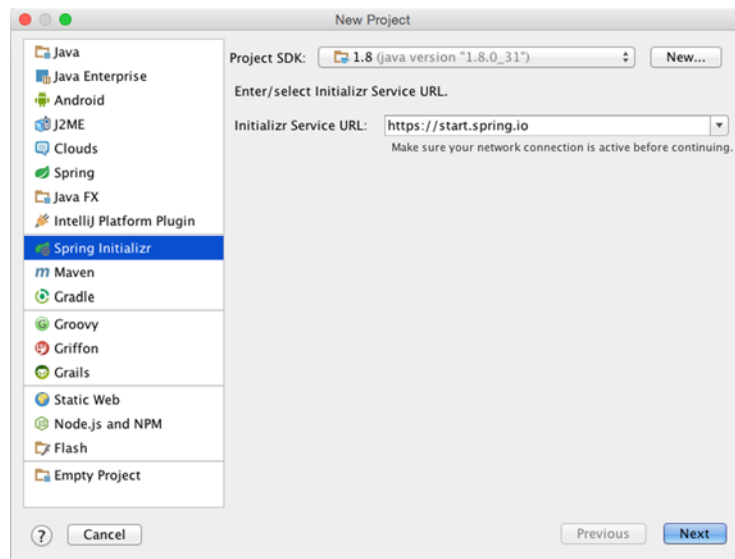
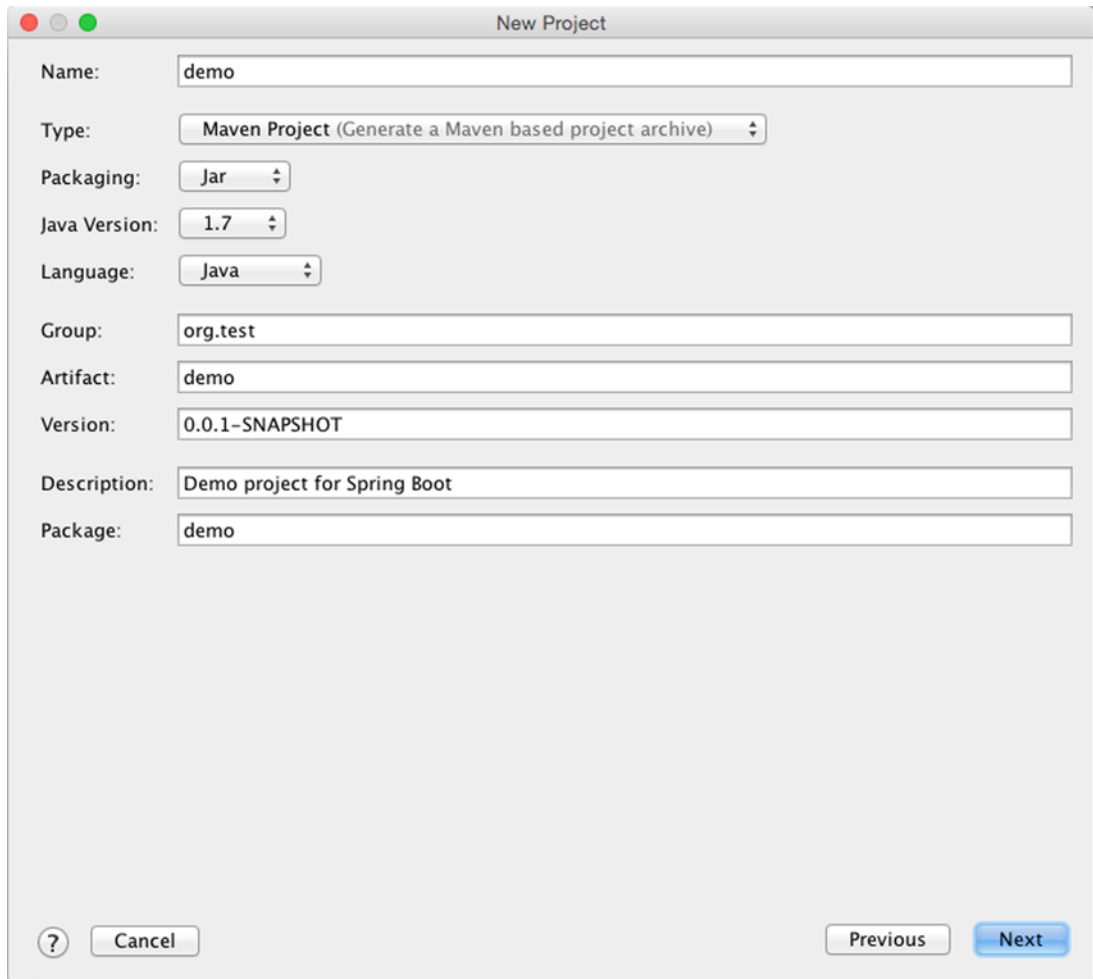


Figura 1.5 La primera pantalla en el asistente de inicialización Spring Boot de IntelliJ IDEA

⁴Puede obtener IntelliJ IDEA en <https://www.jetbrains.com/idea/>. IntelliJ IDEA es un IDE comercial, lo que significa que es posible que deba pagar por él. Sin embargo, puede descargar una versión de prueba y está disponible gratuitamente para su uso en proyectos de código abierto.



The screenshot shows the 'New Project' dialog box in IntelliJ IDEA. The dialog is titled 'New Project' and contains several fields for project configuration. The fields are: Name (demo), Type (Maven Project (Generate a Maven based project archive)), Packaging (Jar), Java Version (1.7), Language (Java), Group (org.test), Artifact (demo), Version (0.0.1-SNAPSHOT), Description (Demo project for Spring Boot), and Package (demo). At the bottom, there are buttons for '?', 'Cancel', 'Previous', and 'Next'.

Figura 1.6 Especificar información del proyecto en el asistente de inicialización Spring Boot de IntelliJ IDEA

En la pantalla inicial, seleccione Spring Initializr de las opciones de proyecto a la izquierda. Luego se le pedirá que seleccione un SDK de proyecto (esencialmente, qué SDK de Java desea usar para el proyecto) y la ubicación del servicio web Initializr. A menos que esté ejecutando su propia instancia de Initializr, probablemente solo haga clic en el botón Siguiente aquí sin realizar ningún cambio. Eso lo llevará a la pantalla que se muestra en la figura 1.6.

La segunda pantalla del asistente de inicialización Spring Boot de IntelliJ IDEA hace algunas preguntas básicas sobre el proyecto, como el nombre del proyecto, el grupo y artefacto de Maven, la versión de Java y si desea construirlo con Maven o Gradle. Una vez que haya descrito su proyecto, hacer clic en el botón Siguiente lo lleva a la tercera pantalla, que se muestra en la figura 1.7.

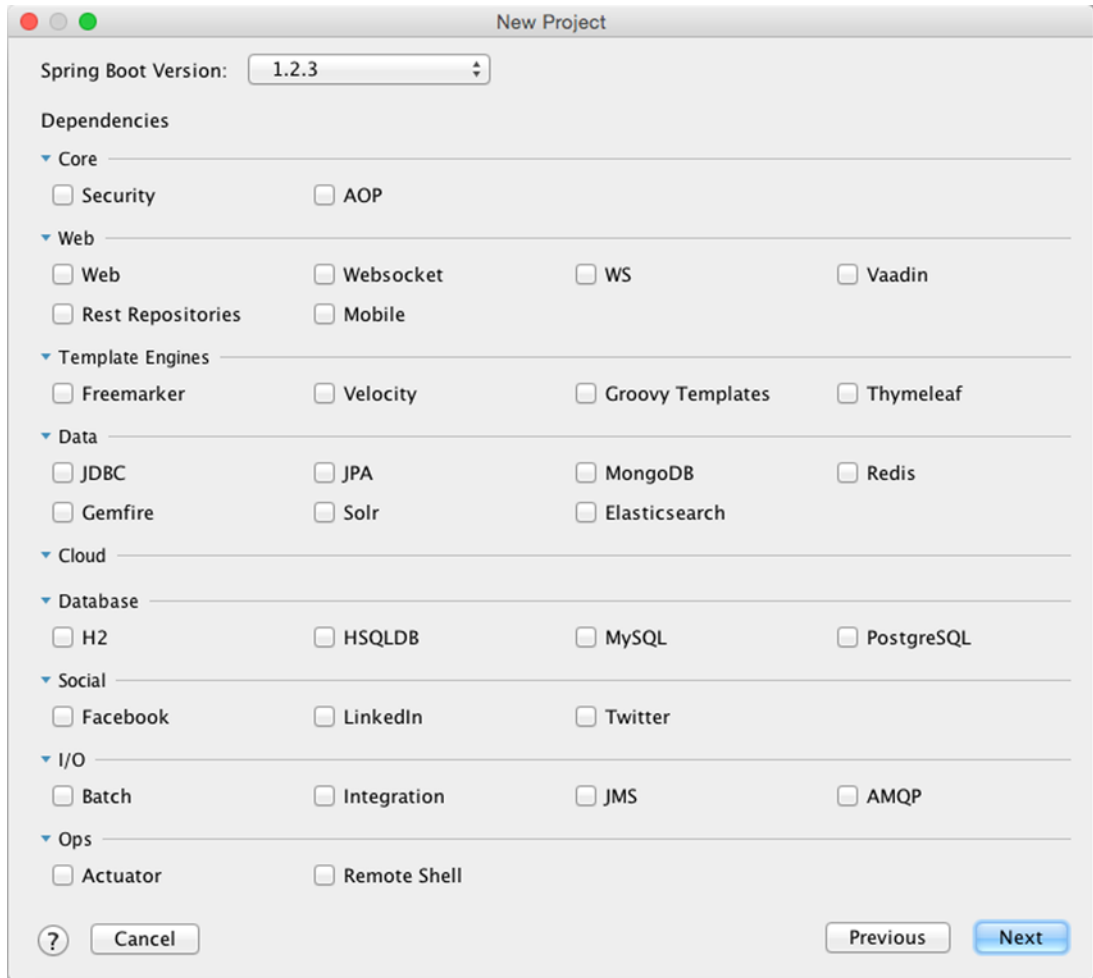


Figura 1.7 Selección de dependencias del proyecto en el asistente de inicialización Spring Boot de IntelliJ IDEA

Donde la segunda pantalla le preguntó sobre la información general del proyecto, la tercera pantalla comienza preguntándole qué tipo de dependencias necesitará en el proyecto. Como antes, las casillas de verificación que se muestran en esta pantalla corresponden a las dependencias de arranque de Spring Boot. Una vez que haya realizado sus selecciones, haga clic en Siguiente para ir a la pantalla final del asistente, que se muestra en la figura 1.8.

Esta última pantalla simplemente quiere que le asigne un nombre al proyecto y le digas a IntelliJ IDEA dónde crearlo. Cuando esté listo, haga clic en el botón Finalizar y tendrá un proyecto Spring Boot básico listo para usted en el IDE.

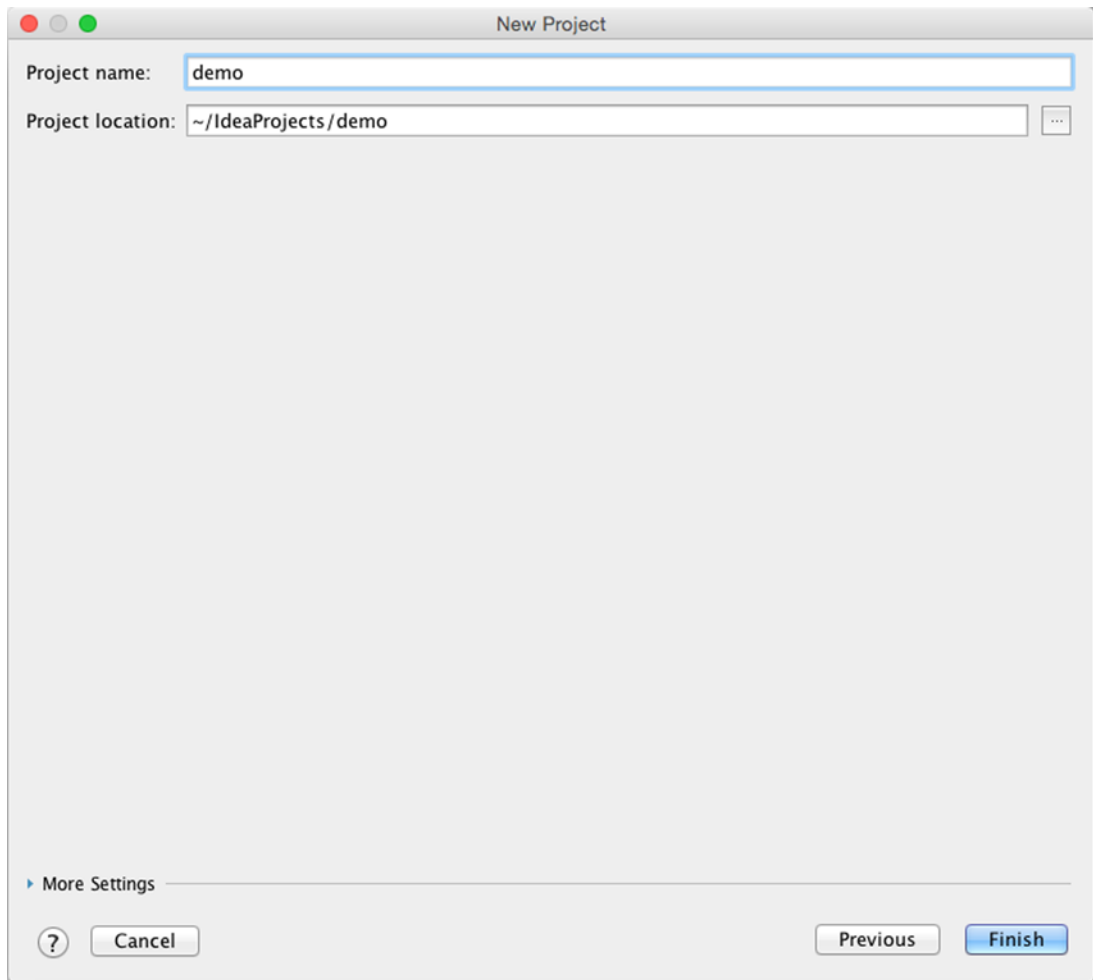


Figura 1.8 La pantalla final en el asistente de inicialización Spring Boot de IntelliJ IDEA

USO DE INICIALIZAR DESDE LA CLI DE SPRING BOOT

Como vio anteriormente, Spring Boot CLI es una excelente manera de desarrollar aplicaciones Spring simplemente escribiendo código. Sin embargo, Spring Boot CLI también tiene algunos comandos que pueden ayudarlo a usar Initializr para poner en marcha el desarrollo en un proyecto Java más tradicional.

La CLI de Spring Boot incluye una en eso comando que actúa como una interfaz de cliente para Initializr. El uso más simple del en eso comando es crear un proyecto Spring Boot de línea de base:

```
$ spring init
```

Después de ponerse en contacto con la aplicación web Initializr, en eso El comando concluirá descargando un archivo demo.zip. Si descomprime este proyecto, encontrará un proyecto típico

estructura con una especificación de compilación pom.xml de Maven. La especificación de compilación de Maven es mínima, con solo dependencias de inicio de línea base para Spring Boot y pruebas. Probablemente quieras un poco más que eso.

Supongamos que desea comenzar creando una aplicación web que use JPA para la persistencia de datos y que esté protegida con Spring Security. Puede especificar esas dependencias iniciales con: dependencias o - D:

```
$ spring init -dweb, jpa, seguridad
```

Esto le dará un archivo demo.zip que contiene la misma estructura de proyecto que antes, pero con los iniciadores de seguridad, JPA y web de Spring Boot expresados como dependencias en pom.xml. Tenga en cuenta que es importante no escribir un espacio entre - D y las dependencias. Si no lo hace, el archivo ZIP se descargará con el nombre web, jpa, seguridad.

Ahora digamos que prefiere construir este proyecto con Gradle. No hay problema. Simplemente especifique Gradle como el tipo de compilación con el - construir parámetro:

```
$ spring init -dweb, jpa, seguridad --build gradle
```

De forma predeterminada, la especificación de compilación para las compilaciones de Maven y Gradle producirá un archivo JAR ejecutable. Si prefiere producir un archivo WAR, puede especificarlo con el - - embalaje o - pag parámetro:

```
$ spring init -dweb, jpa, seguridad --build gradle -p war
```

Hasta ahora, las formas en que hemos utilizado en eso El comando ha dado como resultado la descarga de un archivo zip. Si desea que la CLI abra ese archivo zip por usted, puede especificar un directorio para extraer el proyecto:

```
$ spring init -dweb, jpa, seguridad --build gradle -p war myapp
```

El último parámetro proporcionado aquí indica que desea que el proyecto se extraiga al directorio myapp.

Opcionalmente, si desea que la CLI extraiga el proyecto generado en el directorio actual, puede usar el - extraer o la - X parámetro:

```
$ spring init -dweb, jpa, seguridad --build gradle -p jar -x
```

los en eso El comando tiene varios otros parámetros, incluidos los parámetros para construir un proyecto basado en Groovy, especificar la versión de Java con la que compilar y seleccionar una versión de Spring Boot para compilar. Puede descubrir todos los parámetros utilizando el ayuda mando:

```
$ spring help init
```

También puede averiguar qué opciones están disponibles para esos parámetros utilizando el - - lista o - l parámetro con el en eso mando:

```
$ spring init -l
```

Notarás que aunque `spring init -l` enumera varios parámetros que son compatibles con Initializr, no todos esos parámetros son compatibles directamente con las CLI de Spring Boot en eso mando. Por ejemplo, no puede especificar el nombre del paquete raíz al inicializar un proyecto con la CLI; por defecto será "demo". primavera ayuda `init` puede ayudarlo a descubrir qué parámetros son compatibles con las CLI en eso mando.

Ya sea que use la interfaz basada en la web de Initializr, cree sus proyectos desde Spring Tool Suite o use la CLI de Spring Boot para inicializar un proyecto, los proyectos creados usando Spring Boot Initializr tienen un diseño de proyecto familiar, similar a otros proyectos de Java que usted haya desarrollado. antes de.

1.3 *Resumen*

Spring Boot es una forma nueva y emocionante de desarrollar aplicaciones Spring con una fricción mínima del marco en sí. La configuración automática elimina gran parte de la configuración estándar que infesta las aplicaciones tradicionales de Spring. Los arrancadores Spring Boot le permiten especificar las dependencias de compilación por lo que ofrecen en lugar de usar nombres y versiones explícitos de bibliotecas. Spring Boot CLI lleva el modelo de desarrollo sin fricciones de Spring Boot a un nivel completamente nuevo al permitir un desarrollo rápido y fácil con Groovy desde la línea de comandos. Y el Actuador le permite mirar dentro de su aplicación en ejecución para ver qué ha hecho Spring Boot y cómo lo ha hecho.

Este capítulo le ha brindado una descripción general rápida de lo que Spring Boot tiene para ofrecer. Probablemente esté ansioso por comenzar a escribir una aplicación real con Spring Boot. Eso es exactamente lo que haremos en el próximo capítulo. Con todo lo que Spring Boot hace por ti, la parte más difícil será pasar esta página al capítulo 2.

Desarrollando tu primera Aplicación Spring Boot



Este capítulo cubre

- Trabajar con arrancadores Spring Boot
- Configuración automática de primavera

¿Cuándo fue la última vez que fue a un supermercado o una tienda minorista importante y tuvo que empujar la puerta para abrirla? La mayoría de las grandes tiendas tienen puertas automáticas que detectan su presencia y se abren para usted. Cualquier puerta le permitirá entrar a un edificio, pero las puertas automáticas no requieren que las abra o empuje.

De manera similar, muchas instalaciones públicas tienen baños con grifos de agua automáticos y dispensadores de toallas. Aunque no son tan frecuentes como las puertas automáticas de los supermercados, estos dispositivos no le piden mucho y, en cambio, están felices de dispensar agua y toallas.

Y, sinceramente, no recuerdo la última vez que vi una bandeja de hielo, mucho menos la llené de agua o la rompí para conseguir hielo por un vaso de agua. Mi refrigerador / congelador de alguna manera mágicamente siempre tiene hielo para mí y está listo para llenar un vaso por mí.

Apuesto a que puede pensar en innumerables formas en que la vida moderna se automatiza con dispositivos que funcionan para usted, y no al revés. Con toda esta automatización

en todas partes, pensaría que veríamos más en nuestras tareas de desarrollo. Curiosamente, eso no ha sido así.

Hasta hace poco, la creación de una aplicación con Spring requería que hicieras mucho trabajo para el marco. Claro, Spring ha tenido características fantásticas para desarrollar aplicaciones asombrosas. Pero dependía de usted agregar todas las dependencias de la biblioteca a la especificación de compilación del proyecto. Y era su trabajo escribir la configuración para decirle a Spring qué hacer.

En este capítulo, veremos dos formas en que Spring Boot ha agregado un nivel de automatización al desarrollo de Spring: dependencias de inicio y configuración automática. Verá cómo estas características esenciales de Spring Boot lo liberan del tedio y la distracción de habilitar Spring en sus proyectos y le permiten concentrarse en desarrollar sus aplicaciones. A lo largo del camino, escribirás una pequeña pero completa aplicación Spring que hará que Spring Boot trabaje para ti.

2.1 *Poniendo Spring Boot a trabajar*

El hecho de que estés leyendo esto me dice que eres un lector. Tal vez seas un ratón de biblioteca, leyendo todo lo que puedas. O tal vez solo lea según sea necesario, tal vez eligiendo este libro solo porque necesita saber cómo desarrollar aplicaciones con Spring.

Cualquiera que sea el caso, eres un lector. Y los lectores tienden a mantener una lista de lectura de los libros que quieren (o necesitan) leer. Incluso si no es una lista física, probablemente tenga una lista mental de cosas que le gustaría leer.¹

A lo largo de este libro, crearemos una aplicación de lista de lectura sencilla. Con él, los usuarios pueden ingresar información sobre los libros que desean leer, ver la lista y eliminar libros una vez que se hayan leído. Usaremos Spring Boot para ayudarnos a desarrollarlo rápidamente y con la menor ceremonia posible.

Para comenzar, necesitaremos inicializar el proyecto. En el capítulo 1, analizamos varias formas de utilizar Spring Initializr para iniciar el desarrollo de Spring Boot. Cualquiera de esas opciones funcionará bien aquí, así que elija la que más le convenga y prepárese para poner Spring Boot en funcionamiento.

Desde un punto de vista técnico, usaremos Spring MVC para manejar solicitudes web, Thymeleaf para definir vistas web y Spring Data JPA para conservar las selecciones de lectura en una base de datos. Por ahora, esa base de datos será una base de datos H2 incorporada. Aunque Groovy es una opción, escribiremos el código de la aplicación en Java por ahora. Y usaremos Gradle como nuestra herramienta de construcción preferida.

Si está utilizando Initializr, ya sea a través de su aplicación web, Spring Tool Suite o IntelliJ IDEA, querrá asegurarse de seleccionar las casillas de verificación para Web, Thymeleaf y JPA. Y también recuerde marcar la casilla de verificación H2 para que tenga una base de datos integrada para usar mientras desarrolla la aplicación.

En cuanto a los metadatos del proyecto, puedes elegir lo que quieras. Sin embargo, para los propósitos del ejemplo de la lista de lectura, creé el proyecto con la información que se muestra en la figura 2.1.

¹ Si no eres un lector, no dudes en aplicar esto a películas para ver, restaurantes para probar o lo que más te convenga.

The screenshot shows the Spring Initializr web interface in a browser window. The page title is "SPRING INITIALIZR bootstrap your application now". Below the header, there's a section "Generate a **Gradle Project** with Spring Boot **1.3.0 RC1**". The interface is divided into two main columns: "Project Metadata" and "Dependencies".

Project Metadata:

- Artifact coordinates:**
 - Group:** com.manning
 - Artifact:** readinglist
 - Name:** Reading List
 - Description:** Reading List Demo
 - Package Name:** readinglist
- Packaging:** Jar
- Java Version:** 1.8
- Language:** Java

Too many options? [Switch back to the simple version.](#)

Dependencies:

- Search for dependencies:** Web, Security, JPA, Actuator, Devtools...
- Selected Starters:** Web, Thymeleaf, JPA, H2

At the bottom, there is a green button labeled "Generate Project".

Figura 2.1 Inicialización de la aplicación de lista de lectura a través de la interfaz web de Initializr

Si está utilizando Spring Tool Suite o IntelliJ IDEA para crear el proyecto, adapte los detalles en la figura 2.1 para su IDE de elección.

Por otro lado, si está utilizando Spring Boot CLI para inicializar la aplicación, puede ingresar lo siguiente en la línea de comando:

```
$ spring init -dweb, data-jpa, h2, thymeleaf: compila la lista de lectura de Gradle
```

Recuerde que la CLI en eso El comando no le permite especificar el paquete raíz del proyecto o el nombre del proyecto. El nombre del paquete será por defecto "demo" y el nombre del proyecto

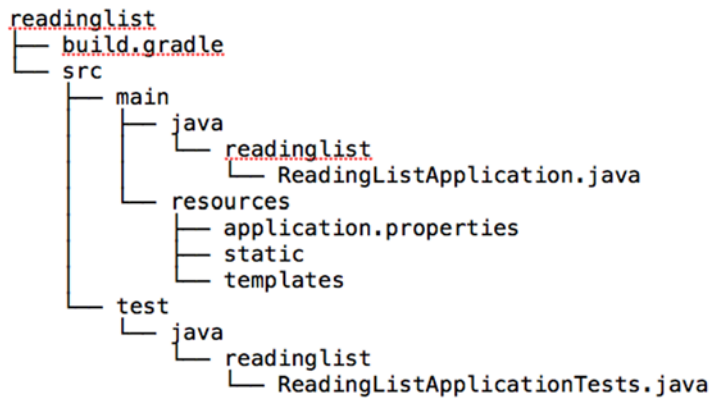


Figura 2.2 La estructura del proyecto de lista de lectura inicializado

por defecto será "Demo". Una vez creado el proyecto, probablemente desee abrirlo y cambiar el nombre del paquete "demo" a "lista de lectura" y cambiar el nombre de "DemoApplication" a "ReadingListApplication.java".

Una vez creado el proyecto, debe tener una estructura de proyecto similar a la que se muestra en la figura 2.2.

Esta es esencialmente la misma estructura de proyecto que la que Initializr le dio en el capítulo 1. Pero ahora que realmente va a desarrollar una aplicación, reduzcamos la velocidad y observemos más de cerca lo que contiene el proyecto inicial.

2.1.1 *Examinando un proyecto Spring Boot recién inicializado*

Lo primero que hay que notar en la figura 2.2 es que la estructura del proyecto sigue el diseño de un proyecto típico de Maven o Gradle. Es decir, el código de la aplicación principal se coloca en la rama src / main / java del árbol de directorios, los recursos se colocan en la rama src / main / resources y el código de prueba se coloca en la rama src / test / java. En este punto no tenemos ningún recurso de prueba, pero si lo tuviéramos, lo pondríamos en src / test / resources.

Profundizando, verá un puñado de archivos esparcidos sobre el proyecto:

- build.gradle: la especificación de compilación de Gradle
- ReadingListApplication.java: la clase de arranque de la aplicación y la clase de configuración primaria de Spring
- application.properties: un lugar para configurar la aplicación y las propiedades de Spring Boot
- ReadingListApplicationTests.java: una clase de prueba de integración básica

Hay muchas bondades de Spring Boot que descubrir en la especificación de compilación, así que guardaré la inspección para el final. En su lugar, comenzaremos con ReadingListApplication.java.

MUELLE BOTAS

los ReadingListApplication La clase tiene dos propósitos en una aplicación Spring Boot: configuración y arranque. Primero, es la clase de configuración central de Spring. Aunque la configuración automática de Spring Boot elimina la necesidad de una gran cantidad de

Configuración de Spring, necesitará al menos una pequeña cantidad de configuración de Spring para habilitar la configuración automática. Como puede ver en el listado 2.1, solo hay una línea de código de configuración.

El listado 2.1 ReadingListApplication.java es tanto una clase de arranque como una clase de configuración

lista de lectura de paquetes;

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class ReadingListApplication {
```

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

← Habilitar el análisis de componentes
y autoconfiguración

← Bootstrap the
solicitud

Los `@SpringBootApplication` habilita el escaneo de componentes de Spring y la configuración automática de Spring Boot. De hecho, `@SpringBootApplication` combina otras tres anotaciones útiles:

- **Muelles @ Configuración** : Designa una clase como una clase de configuración utilizando la configuración basada en Java de Spring. Aunque no escribiremos mucha configuración en este libro, preferiremos la configuración basada en Java sobre la configuración XML cuando lo hagamos.
- **Muelles @ ComponentScan** —Habilita el escaneo de componentes para que las clases de controlador web y otros componentes que escriba se detecten y registren automáticamente como beans en el contexto de la aplicación Spring. Un poco más adelante en este capítulo, escribiremos un controlador Spring MVC simple que se anotará con `@Controlador` para que el escaneo de componentes pueda encontrarlo.
- **Spring Boot's @ EnableAutoConfiguration** —Esta pequeña y humilde anotación bien podría llamarse `@Abracadabra` porque es la única línea de configuración que permite la magia de la configuración automática de Spring Boot. Esta única línea evita que tenga que escribir las páginas de configuración que se requerirían de otra manera.

En versiones anteriores de Spring Boot, habría anotado el `ReadingListApplication` class con las tres anotaciones. Pero desde Spring Boot 1.2.0, `@SpringBootApplication` es todo lo que necesitas.

Como ya he dicho, `ReadingListApplication` también es una clase de arranque. Hay varias formas de ejecutar aplicaciones Spring Boot, incluida la implementación tradicional de archivos WAR. Pero por ahora el principal() El método aquí le permitirá ejecutar su aplicación como un archivo JAR ejecutable desde la línea de comando. Pasa una referencia a la `ReadingListApplication` clase a `SpringApplication.run()`, junto con los argumentos de la línea de comandos, para iniciar la aplicación.

De hecho, aunque no haya escrito ningún código de aplicación, aún puede compilar la aplicación en este punto y probarla. La forma más sencilla de crear y ejecutar la aplicación es utilizar el `bootRun` tarea con Gradle:

```
$ gradle bootRun
```

La tarea `bootRun` proviene del complemento Gradle de Spring Boot, que discutiremos más en la sección 2.12.

Alternativamente, puede construir el proyecto con Gradle y ejecutarlo con Java en la línea de comando:

```
$ gradle build
...
$ java -jar build / libs / readinglist-0.0.1-SNAPSHOT.jar
```

La aplicación debería iniciarse bien y habilitar un servidor Tomcat que escuche en el puerto 8080. Puede apuntar su navegador a `http://localhost:8080` si lo desea, pero debido a que aún no ha escrito una clase de controlador, se encontrará con un error HTTP 404 (no encontrado) y una página de error. Sin embargo, antes de que termine este capítulo, esa URL servirá para su aplicación de lista de lectura.

Casi nunca necesitará cambiar `ReadingListApplication.java`. Si su aplicación requiere alguna configuración de Spring adicional más allá de la que proporciona la configuración automática de Spring Boot, generalmente es mejor escribirla en `@Configuration` clases configuradas. (Serán recogidos y utilizados por el escaneo de componentes). Sin embargo, en casos excepcionalmente simples, puede agregar una configuración personalizada a `ReadingListApplication.java`.

PROBANDO APLICACIONES DE BOTAS DE RESORTE

Inicialmente también le brindó una clase de prueba básica para ayudarlo a comenzar a escribir pruebas para su aplicación. Pero `ReadingListApplicationTests` (listado 2.2) es más que un simple marcador de posición para las pruebas; también sirve como un ejemplo de cómo escribir pruebas para aplicaciones Spring Boot.

Listado 2.2 @ `SpringApplicationConfiguration` loads `SpringApplicationContext`

lista de lectura de paquetes;

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.SpringApplicationConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
```

```
import lista de lectura.ReadingListApplication;
```

```
@RunWith (SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration (
    clases = ReadingListApplication.class)
@WebAppConfiguration
```


← Cargar contexto a través de
Bota de primavera

```
clase pública ReadingListApplicationTests {
```

```
    @Prueba
```

```
    public void contextLoads () {}
```

Prueba que el
cargas de contexto



```
}
```

En una prueba de integración de Spring típica, anotaría la clase de prueba con `@ ContextoConfiguración` para especificar cómo la prueba debe cargar el contexto de la aplicación Spring. Pero para aprovechar al máximo la magia de Spring Boot, `@ SpringApplicationConfiguration` en su lugar, se debe utilizar una anotación. Como puede ver en el listado 2.2,

`ReadingListApplicationTests` está anotado con `@ SpringApplicationConfiguration` para cargar el contexto de la aplicación Spring desde el `ReadingListApplication` clase de configuración.

`ReadingListApplicationTests` también incluye un método de prueba simple, contexto-Cargas (). De hecho, es tan simple que es un método vacío. Pero es suficiente para verificar que el contexto de la aplicación se carga sin problemas. Si la configuración definida en `ReadingListApplication` es bueno, la prueba pasará. Si hay algún problema, la prueba fallará.

Por supuesto, agregará algunas de sus propias pruebas a medida que desarrollemos la aplicación. Pero el `contextLoads ()` El método es un buen comienzo y verifica cada parte de la funcionalidad proporcionada por la aplicación en este momento. Veremos más sobre cómo probar las aplicaciones Spring Boot en el capítulo 4.

CONFIGURAR LAS PROPIEDADES DE LA APLICACIÓN

El archivo `application.properties` que le proporcionó Initializr está inicialmente vacío. De hecho, este archivo es completamente opcional, por lo que puede eliminarlo por completo sin afectar la aplicación. Pero tampoco hay nada malo en dejarlo en su lugar.

Definitivamente encontraremos la oportunidad de agregar entradas a `application.properties` más adelante. Por ahora, sin embargo, si desea hurgar con `application.properties`, intente agregar la siguiente línea:

```
servidor.puerto = 8000
```

Con esta línea, está configurando el servidor Tomcat incorporado para escuchar en el puerto 8000 en lugar del puerto predeterminado 8080. Puede confirmar esto ejecutando la aplicación nuevamente.

Esto demuestra que el archivo `application.properties` es útil para la configuración detallada de las cosas que Spring Boot configura automáticamente. Pero también puede usarlo para especificar las propiedades que usa el código de la aplicación. Veremos varios ejemplos de ambos usos de `application.properties` en el capítulo 3.

Lo principal a tener en cuenta es que en ningún momento le pide explícitamente a Spring Boot que cargue `application.properties` por usted. En virtud del hecho de que `application.properties` existe, se cargará y sus propiedades estarán disponibles para configurar tanto Spring como el código de la aplicación.

Casi hemos terminado de revisar el contenido del proyecto inicializado. Pero tenemos un último artefacto que mirar. Veamos cómo se construye una aplicación Spring Boot.

2.1.2 *Disecionando una construcción de proyecto Spring Boot*

En su mayor parte, una aplicación Spring Boot no es muy diferente de cualquier aplicación Spring, que no es muy diferente de cualquier aplicación Java. Por lo tanto, crear una aplicación Spring Boot es muy similar a crear cualquier aplicación Java. Puede elegir Gradle o Maven como herramienta de compilación, y expresa los detalles de compilación de la misma manera que lo haría en una aplicación que no emplea Spring Boot. Pero hay algunos pequeños detalles sobre cómo trabajar con Spring Boot que se benefician de una pequeña ayuda adicional en la compilación.

Spring Boot proporciona complementos de compilación para Gradle y Maven para ayudar a crear proyectos de Spring Boot. El Listado 2.3 muestra el archivo build.gradle creado por Initializr, que aplica el complemento Spring Boot Gradle.

Listado 2.3 Usando el complemento Spring Boot Gradle

```
buildscript {
    ext {
        springBootVersion = "1.3.0.RELEASE"
    }
    repositories {
        mavenCentral ()
    }
    dependencies {
        classpath ("org.springframework.boot: spring-boot-gradle-plugin:
            ↪ PS springBootVersion")
    }
}

aplicar complemento: 'java'
aplicar complemento: 'eclipse'
aplicar complemento: 'idea'
aplicar complemento: 'spring-boot'
```

Depender
en primavera
Complemento de arranque

Aplicar primavera
Complemento de arranque

```
frasco {
    baseName = 'lista de lectura'
    versión = '0.0.1-SNAPSHOT'
}
sourceCompatibility = 1.7
targetCompatibility = 1.7

repositories {
    mavenCentral ()
}

dependencias {
    compilar ("org.springframework.boot: spring-boot-starter-web") compilar ("org.springframework.boot:
    spring-boot-starter-data-jpa")
```

Inicio
dependencias

```

compile ("org.springframework.boot: spring-boot-starter-thymeleaf") tiempo de ejecución ("com.h2database: h2")

testCompile ("org.springframework.boot: spring-boot-starter-test")
}

eclipse {
    classpath {
        contenedores.remove ('org.eclipse.jdt.launching.JRE_CONTAINER')
        contenedores 'org.eclipse.jdt.launching.JRE_CONTAINER / org.eclipse.jdt.internal.
            ↳ debug.ui.launcher.StandardVMType / JavaSE-1.7 '
    }
}

contenedor de tareas (tipo: contenedor) {
    gradleVersion = '1.12'
}

```

Por otro lado, si hubiera elegido construir su proyecto con Maven, Initializr le habría dado un archivo pom.xml que emplea el complemento Maven de Spring Boot, como se muestra en el listado 2.4.

Listado 2.4 Uso del complemento Spring Boot Maven y el iniciador principal

```

<? xml version = "1.0" encoding = "UTF-8"?>
<proyecto 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.manning </groupId>
  <artifactId> lista de lectura </artifactId>
  <versión> 0.0.1-SNAPSHOT </version>
  <packaging> jar </packaging>

  <name> ReadingList </name>
  <description> Demostración de la lista de lectura </description>

  <padre>
    <groupId> org.springframework.boot </groupId>
    <artifactId> spring-boot-starter-parent </artifactId>
    <version> {springBootVersion} </version>
    <relativaPath /> <!-- padre de búsqueda desde el repositorio --> </parent>

  <dependencias>
    <dependencia>
      <groupId> org.springframework.boot </groupId>
      <artifactId> spring-boot-starter-web </artifactId>
    </dependencia>
    <dependencia>
      <groupId> org.springframework.boot </groupId>

```

Heredar versiones
del padre principiante

Inicio
dependencias


```

        <artifactId> spring-boot-starter-data-jpa </artifactId>
    </dependencia>
    <dependencia>
        <groupId> org.springframework.boot </groupId>
        <artifactId> arrancador-de-arranque-primavera-hoja-de-tomillo </artifactId>
    </dependencia>
    <dependencia>
        <groupId> com.h2database </groupId>
        <artifactId> h2 </artifactId>
    </dependencia>
    <dependencia>
        <groupId> org.springframework.boot </groupId>
        <artifactId> prueba-de-arranque-de-arranque-de-primavera </artifactId>
        <scope> probar </scope>
    </dependencia>
</dependencias>

<propiedades>
    <project.build.sourceEncoding>
        UTF-8
    </project.build.sourceEncoding>
    <start-class> readlist.Application </start-class>
    <java.version> 1.7 </java.version>
</properties>

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

</proyecto>

```

Aplicar primavera
Complemento de arranque

Ya sea que elija Gradle o Maven, los complementos de compilación de Spring Boot contribuyen a la compilación de dos maneras. Primero, ya ha visto cómo puede usar el `bootRun` tarea para ejecutar la aplicación con Gradle. Del mismo modo, el complemento Spring Boot Maven proporciona una `spring-boot:ejecutar` objetivo que logra lo mismo si usa una compilación de Maven.

La característica principal de los complementos de compilación es que pueden empaquetar el proyecto como un uber-JAR ejecutable. Esto incluye empaquetar todas las dependencias de la aplicación dentro del JAR y agregar un manifiesto al JAR con entradas que hacen posible ejecutar la aplicación con `java -jar`.

Además de los complementos de compilación, observe que la compilación de Maven en el listado 2.4 tiene "spring-boot-starter-parent" como padre. Al enraizar el proyecto en el iniciador principal, la compilación puede aprovechar la administración de dependencias de Maven para heredar versiones de dependencia para varias bibliotecas de uso común, de modo que no tenga que especificar explícitamente las versiones al declarar dependencias. Observe que ninguno de los `<dependencia>`

Las entradas en este archivo pom.xml especifican cualquier versión.

Desafortunadamente, Gradle no proporciona el mismo tipo de administración de dependencias que Maven. Es por eso que el complemento Spring Boot Gradle ofrece una tercera característica; simula la gestión de dependencias para varias dependencias comunes relacionadas con Spring y Spring. En consecuencia, el archivo build.gradle en el listado 2.3 no especifica ninguna versión para ninguna de sus dependencias.

Hablando de esas dependencias, solo hay cinco dependencias expresadas en cualquiera de las especificaciones de compilación. Y, con la excepción de la dependencia H2 que agregé manualmente, todos tienen ID de artefactos que curiosamente tienen el prefijo "spring-boot-starter-". Estas son las dependencias de inicio de SpringBoot, y ofrecen un poco de magia en tiempo de compilación para las aplicaciones de SpringBoot. Veamos qué beneficio aportan.

2.2 Usar dependencias de inicio

Para comprender el beneficio de las dependencias de arranque de Spring Boot, finjamos por un momento que no existen. ¿Qué tipo de dependencias agregaría a su compilación sin Spring Boot? ¿Qué dependencias de Spring necesita para admitir Spring MVC? ¿Recuerdas los ID de grupo y artefacto de Thymeleaf? ¿Qué versión de Spring Data JPA debería usar? ¿Son todos estos compatibles?

UH oh. Sin las dependencias de arranque de Spring Boot, tienes algunos deberes que hacer. Todo lo que desea hacer es desarrollar una aplicación web Spring con vistas de Thymeleaf que conserve sus datos a través de JPA. Pero antes de que pueda escribir su primera línea de código, debe averiguar qué se debe incluir en la especificación de compilación para respaldar su plan.

Después de mucha consideración (y probablemente mucho copiar y pegar desde la compilación de alguna otra aplicación que tenga dependencias similares) llega a lo siguiente dependencias block en la especificación de compilación de Gradle:

```
compile ("org.springframework: spring-web: 4.1.6.RELEASE") compile ("org.thymeleaf: thymeleaf-spring4: 2.1.4.RELEASE")
compile ("org.springframework.data:spring-data-jpa : 1.8.0.RELEASE ") compile (" org.hibernate: hibernate-entitymanager: jar:
4.3.8.Final ") compile (" com.h2database: h2: 1.4.187 ")
```

Esta lista de dependencias está bien e incluso podría funcionar. ¿Pero, como lo sabes? ¿Qué tipo de seguridad tiene de que las versiones que eligió para esas dependencias son incluso compatibles entre sí? Pueden serlo, pero no lo sabrá hasta que cree la aplicación y la ejecute. ¿Y cómo sabe que la lista de dependencias está completa? Sin haber escrito ni una sola línea de código, todavía está muy lejos de patear los neumáticos en su construcción.

Demos un paso atrás y recordemos qué es lo que queremos hacer. Estamos buscando construir una aplicación con estas características:

- Es una aplicación web
- Utiliza Thymeleaf
- Persiste los datos en una base de datos relacional a través de Spring Data JPA

¿No sería más sencillo si pudiéramos especificar esos hechos en la compilación y dejar que la compilación solucionara lo que necesitamos? Eso es exactamente lo que hacen las dependencias de arranque de Spring Boot.

2.2.1 *Especificar dependencias basadas en facetas*

Spring Boot aborda la complejidad de las dependencias del proyecto al proporcionar varias docenas de dependencias "iniciales". Una dependencia de inicio es esencialmente un POM de Maven que define dependencias transitivas en otras bibliotecas que juntas brindan soporte para alguna funcionalidad. Muchas de estas dependencias de inicio se nombran para indicar la faceta o el tipo de funcionalidad que proporcionan.

Por ejemplo, la aplicación de lista de lectura será una aplicación web. En lugar de agregar varias dependencias de biblioteca elegidas individualmente a la compilación del proyecto, es mucho más fácil simplemente declarar que se trata de una aplicación web. Puede hacerlo agregando el iniciador web de Spring Boot a la compilación.

También queremos usar Thymeleaf para vistas web y datos persistentes con JPA. Por lo tanto, necesitamos las dependencias de inicio de Thymeleaf y Spring Data JPA en la compilación.

Para fines de prueba, también queremos bibliotecas que nos permitan ejecutar pruebas de integración en el contexto de Spring Boot. Por lo tanto, también queremos una dependencia del tiempo de prueba en el iniciador de prueba de Spring Boot.

En conjunto, tenemos las siguientes cinco dependencias que Initializr proporcionó en la compilación de Gradle:

```
dependencias {
    compile "org.springframework.boot: spring-boot-starter-web" compile "org.springframework.boot:
spring-boot-starter-thymeleaf" compile "org.springframework.boot: spring-boot-starter-data-jpa" compile "com.h2database:
h2"

    testCompile ("org.springframework.boot: spring-boot-starter-test")
}
```

Como vio anteriormente, la forma más fácil de incluir estas dependencias en la compilación de su aplicación es seleccionar las casillas de verificación Web, Thymeleaf y JPA en Initializr. Pero si no hizo eso al inicializar el proyecto, ciertamente puede volver atrás y agregarlos más tarde editando el archivo build.gradle o pom.xml generado.

A través de dependencias transitivas, agregar estas cuatro dependencias equivale a agregar varias docenas de bibliotecas individuales a la compilación. Algunas de esas dependencias transitivas incluyen cosas como Spring MVC, Spring Data JPA, Thymeleaf, así como cualquier dependencia transitiva que declaren esas dependencias.

Lo más importante a tener en cuenta sobre las cuatro dependencias de inicio es que eran tan específicas como debían ser. No dijimos que queríamos Spring MVC; simplemente dijimos que queríamos crear una aplicación web. No especificamos JUnit ni ninguna otra herramienta de prueba; solo dijimos que queríamos probar nuestro código. Los iniciadores Thymeleaf y Spring Data JPA son un poco más específicos, pero solo porque no hay una forma menos específica de declarar que quieres Thymeleaf y Spring Data JPA.

Los cuatro iniciadores de esta compilación son solo algunas de las muchas dependencias de iniciadores que ofrece Spring Boot. El Apéndice B enumera todos los iniciadores con algunos detalles sobre lo que cada uno aporta de manera transitiva a la construcción de un proyecto.

En ningún caso tuvimos que especificar la versión. Las versiones de las dependencias de inicio están determinadas por la versión de Spring Boot que esté utilizando. Las propias dependencias de inicio determinan las versiones de las diversas dependencias transitivas que incorporan.

No saber qué versiones de las distintas bibliotecas se utilizan puede resultarle un poco inquietante. Anímate a saber que Spring Boot ha sido probado para garantizar que todas las dependencias extraídas sean compatibles entre sí. En realidad, es muy liberador especificar una dependencia de inicio y no tener que preocuparse por qué bibliotecas y qué versiones de esas bibliotecas necesita mantener.

Pero si realmente debe saber qué es lo que está obteniendo, siempre puede obtenerlo de la herramienta de compilación. En el caso de Gradle, el comando `dependencies` le dará un árbol de dependencias que incluye todas las bibliotecas que usa su proyecto y sus versiones:

Dependencias de \$ gradle

Puede obtener un árbol de dependencia similar de una compilación de Maven con el comando `dependency:tree` de la dependencia enchufar:

Dependencia de \$ mvn: árbol

En su mayor parte, nunca debe preocuparse por los detalles de lo que proporciona cada dependencia de arranque de Spring Boot. En general, es suficiente saber que el iniciador web le permite crear una aplicación web, el iniciador Thymeleaf le permite usar plantillas de Thymeleaf y el iniciador Spring Data JPA permite la persistencia de datos en una base de datos usando Spring Data JPA.

Pero, ¿qué pasa si, a pesar de las pruebas realizadas por el equipo de Spring Boot, hay un problema con la elección de bibliotecas de una dependencia de inicio? ¿Cómo se puede anular el motor de arranque?

2.2.2 Anulación de las dependencias transitivas de arranque

En última instancia, las dependencias de inicio son solo dependencias como cualquier otra dependencia en su compilación. Eso significa que puede usar las funciones de la herramienta de compilación para anular selectivamente las versiones de dependencia transitiva, excluir las dependencias transitivas y, ciertamente, especificar dependencias para bibliotecas no cubiertas por los arrancadores Spring Boot.

Por ejemplo, considere el iniciador web de Spring Boot. Entre otras cosas, el iniciador web depende transitivamente de la biblioteca Jackson JSON. Esta biblioteca es útil si está creando un servicio REST que consume o produce representaciones de recursos JSON. Pero si está utilizando Spring Boot para crear una aplicación web más tradicional para humanos, es posible que no necesite Jackson. Aunque no debería doler nada incluirlo, puede recortar la grasa de su estructura al excluir a Jackson como una dependencia transitiva.

Si está utilizando Gradle, puede excluir dependencias transitivas como esta:

```
compile ("org.springframework.boot: spring-boot-starter-web") {
    exclude grupo: 'com.fasterxml.jackson.core'
}
```

En Maven, puede excluir dependencias transitivas con el < exclusiones> elemento. El siguiente < dependencia> para el iniciador web Spring Boot tiene < exclusiones> para mantener a Jackson fuera de la construcción:

```
<dependencia>
  <groupId> org.springframework.boot </groupId>
  <artifactId> spring-boot-starter-web </artifactId>
  <exclusiones>
    <exclusión>
      <groupId> com.fasterxml.jackson.core </groupId>
    </exclusión>
  </exclusiones>
</dependencia>
```

Por otro lado, tal vez tener a Jackson en la construcción esté bien, pero desea construir contra una versión diferente de Jackson a la que hace referencia el iniciador web. Suponga que el iniciador web hace referencia a la versión 2.3.4 de Jackson, pero prefiere la versión de usuario 2.4.3.²

Con Maven, puede expresar la dependencia deseada directamente en el archivo pom.xml de su proyecto de esta manera:

```
<dependencia>
  <groupId> com.fasterxml.jackson.core </groupId>
  <artifactId> jackson-databind </artifactId>
  <version> 2.4.3 </version>
</dependencia>
```

Maven siempre favorece la dependencia más cercana, lo que significa que debido a que ha expresado esta dependencia en la compilación de su proyecto, se verá favorecida sobre la que se refiere transitivamente a otra dependencia.

De manera similar, si está compilando con Gradle, puede especificar la versión más nueva de Jackson en su archivo build.gradle de esta manera:

```
compilar ("com.fasterxml.jackson.core: jackson-databind: 2.4.3")
```

Esta dependencia funciona en Gradle porque es más nueva que la versión a la que hace referencia transitivamente el iniciador web de Spring Boot. Pero suponga que en lugar de usar una versión más nueva de Jackson, le gustaría usar una versión anterior. A diferencia de Maven, Gradle favorece la versión más nueva de una dependencia. Por lo tanto, si desea utilizar una versión anterior de

² Las versiones mencionadas aquí son solo para fines ilustrativos. La versión real de Jackson a la que hace referencia

El iniciador web de Spring Boot estará determinado por la versión de Spring Boot que esté utilizando.

Jackson, tendrá que expresar la versión anterior como una dependencia en su compilación y excluirla para que no se resuelva transitivamente mediante la dependencia de inicio web:

```
compile ("org.springframework.boot: spring-boot-starter-web") {  
    excluir grupo: 'com.fasterxml.jackson.core'  
}  
compilar ("com.fasterxml.jackson.core: jackson-databind: 2.3.1")
```

En cualquier caso, tenga cuidado al anular las dependencias que son introducidas transitivamente por las dependencias de arranque de Spring Boot. Aunque diferentes versiones pueden funcionar bien, hay una gran cantidad de comodidad que se puede tomar sabiendo que las versiones elegidas por los principiantes han sido probadas para funcionar bien juntas. Solo debe anular estas dependencias transitivas en circunstancias especiales (como una corrección de errores en una versión más reciente).

Ahora que tenemos una estructura de proyecto vacía y una especificación de compilación lista, es hora de comenzar a desarrollar la aplicación. Mientras lo hacemos, dejaremos que Spring Boot maneje los detalles de configuración mientras nos concentramos en escribir el código que proporciona la funcionalidad de lista de lectura.

2.3 Usando la configuración automática

En pocas palabras, la configuración automática de Spring Boot es un proceso en tiempo de ejecución (más exactamente, el tiempo de inicio de la aplicación) que considera varios factores para decidir qué configuración de Spring debe y no debe aplicarse. Para ilustrar, aquí hay algunos ejemplos de los tipos de cosas que la configuración automática de Spring Boot podría considerar:

- Es la primavera JdbcTemplate disponible en el classpath? Si es así y si hay un Fuente de datos bean, luego autoconfigure un JdbcTemplate frijol.
- ¿Thymeleaf está en la ruta de clases? Si es así, configure un solucionador de plantillas Thymeleaf, un solucionador de vistas y un motor de plantillas.
- ¿Spring Security está en la ruta de clases? Si es así, configure una configuración de seguridad web muy básica.

Hay casi 200 decisiones de este tipo que Spring Boot toma con respecto a la configuración automática cada vez que se inicia una aplicación, y cubre áreas como seguridad, integración, persistencia y desarrollo web. Toda esta configuración automática sirve para evitar que tenga que escribir la configuración explícitamente a menos que sea absolutamente necesario.

Lo curioso de la configuración automática es que es difícil de mostrar en las páginas de este libro. Si no hay ninguna configuración para escribir, ¿qué hay que señalar y discutir?

2.3.1 Centrarse en la funcionalidad de la aplicación

Una forma de apreciar la configuración automática de Spring Boot sería dedicar las siguientes páginas a mostrarle la configuración que se requiere en ausencia de Spring Boot. Pero ya hay varios libros geniales sobre la primavera que muestran