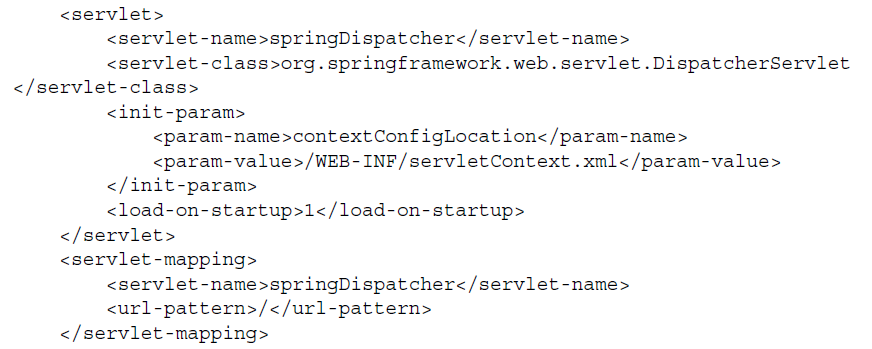
# Primer Tema de Spring.

# Utilizando el descriptor de despliegue para arrancar la Spring

En las aplicaciones tradicionales de Spring Framework, siempre instalaste Spring en el descriptor de despliegue Java EE. Como mínimo, esto requiere una instancia de la DispatcherServlet dado un archivo de configuración en forma de un parámetro init contextConfigLocation y dio instrucciones para cargar en el arranque.



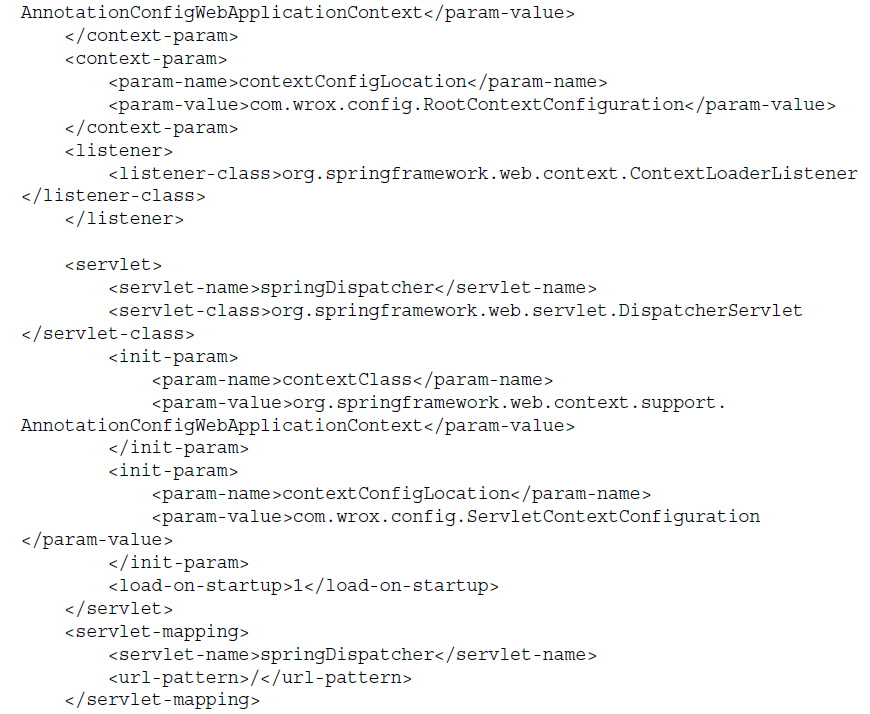
Esto crea un contexto de aplicación de primavera sola dentro de la configuración de la DispatcherServlet e instruye al contenedor de servlets para inicializar el ispatcherServlet en el inicio. Cuando se inicializa, la DispatcherServlet carga la configuración contexto del archivo /WEB-INF/servletContext.xml e inicia el contexto de aplicación. Por supuesto, esto crea sólo un contexto de aplicación para su aplicación, que, como se ha explicado anteriormente, no es muy flexible. Un descriptor de arranque más completa se vería más como esto:



El ContextLoaderListener se inicializa cuando se inicia la aplicación web en primer lugar (ya que implementa ServletContextListener, inicializa antes de que los servlets), carga la configuración de contexto de aplicación de la raíz del archivo /WEB-INF/rootContext.xml especificado en el parámetro init contexto contextConfigLocation, y se inicia el contexto de aplicación de la raíz.

Tenga en cuenta que el parámetro de contexto init contextConfigLocation es diferente del parámetro de la contextConfigLocation Servlet init para la DispatcherServlet. No entran en conflicto; la primera se aplica a todo el contexto de servlet y el último se aplica sólo al servlet particular para el cual está especificada. El contexto de aplicación de la raíz creado por el oyente se ajusta automáticamente según el contexto matriz por cualquier contextos de aplicación creados por DispatcherServlets.

A pesar de que sólo puede haber un contexto de aplicación de la raíz de esta naturaleza en una aplicación web, no puede haber tantos contextos de aplicación Servlet como se puede encontrar usos para. En el capítulo 17, se puede ver un ejemplo de esto cuando se crea una segunda DispatcherServlet para los servicios web RESTful. También puede crear arbitrariamente otros contextos de aplicación, si es necesario, sin embargo, que no suele ser aplicable en una aplicación web. Por supuesto, estos ejemplos de arranque han asumido que spring está configurando el uso de archivos XML, que se explora en la sección "Configuración de Spring Framework" de este capítulo. También puede configurar a Spring con Java en lugar de XML (también cubierto en la sección posterior), y bootstrapping una configuración de Java desde el descriptor de despliegue es en gran medida lo mismo:



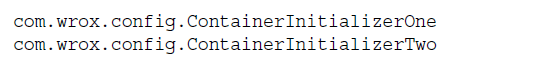
Normalmente, el ContextLoaderListener y DispatcherServlet crean instancias de org.springframework.web.context.support.XmlWebApplicationContext, que esperan

La configuración de Spring por venir en forma de un archivo XML. El ejemplo anterior anula este comportamiento y utiliza el AnnotationConfigWebApplicationContext. Este tipo de contexto espera configuraciones contexto programático, especificados como nombres de clase (en lugar de los nombres de archivos) en los parámetros contextConfigLocation.

# Bootstrapping programación de Spring en un inicializador

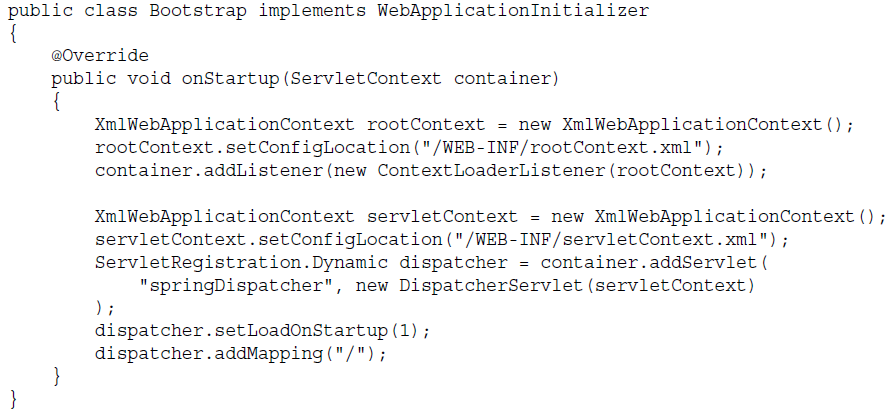
Recordemos que en los capítulos anteriores ha utilizado un ServletContextListener para configurar mediante programación los servlets y filtros en su aplicación. La desventaja de usar esta interfaz es que el método contextInitialized del oyente puede ser llamado después de que otros oyentes. Java EE 6 añade una nueva interfaz llamada ServletContainerInitializer. Las clases que implementan ServletContainerInitializer tienen su método OnStartup llama cuando se inicia la aplicación en primer lugar, antes de que se notifica a los oyentes. Este es el punto más temprano posible en el ciclo de vida de la aplicación. No es necesario configurar ServletContainerInitializers en el descriptor de despliegue; En su lugar, utiliza el sistema proveedor de servicios de Java para declarar una clase o clases que implementan

ServletContainerInitializer haciendo una lista de ellos, uno en cada línea, en un archivo llamado / META-INF / services / javax.servlet.ServletContainerInitializer. Por ejemplo, el siguiente archivo Mostrar el contenido de dos clases que implementan ServletContainerInitializer:

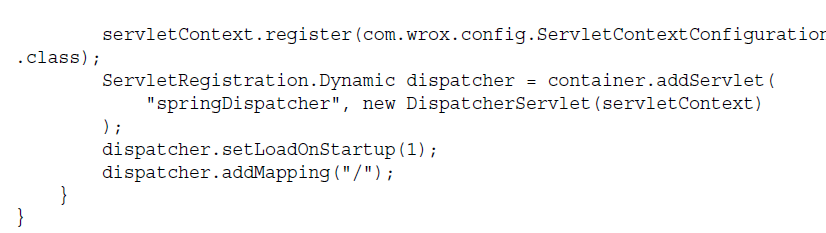
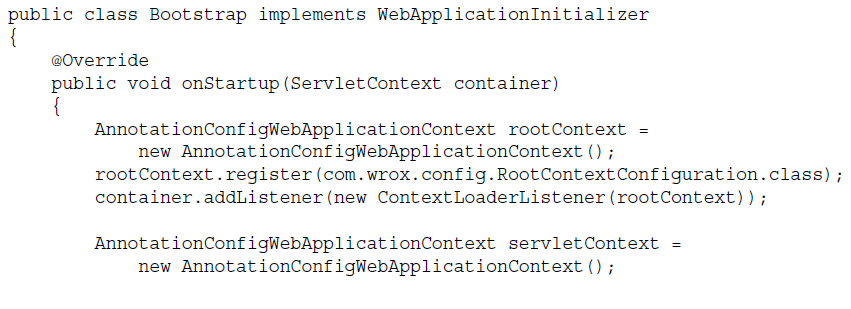


La desventaja es que este archivo no puede existir directamente dentro de archivos WAR de la aplicación o directorio expandido - no se puede colocar el archivo en el directorio / META-INF / servicios de su aplicación web. Debe colocarse en el directorio / META-INF / services de un archivo JAR incluido en / WEB-INF de la aplicación / lib.

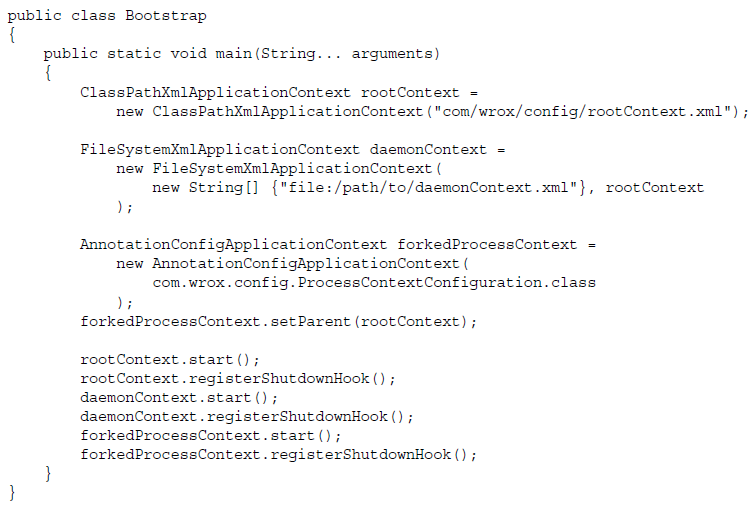
Spring Framework proporciona una interfaz de puente que hace de este mucho más simple de lograr. La clase implementa org.springframework.web.SpringServletContainerInitializer ServletContainerInitializer, y porque el JAR que contiene esta clase incluye un archivo de proveedor de servicios de nombre del listado de la clase, su método OnStartup se llama cuando se inicia la aplicación. Esta clase luego explora su solicitud de implementaciones de la interfaz .WebApplicationInitializer org.springframework.web y llama al método OnStartup de las clases que coinciden que encuentra. Dentro de una clase de implementación WebApplicationInitializer, puede configurar mediante programación oyentes, Servlets, filtros, y más, todo sin tener que escribir una sola línea de XML. Más importante, se puede arrancar Spring dentro de esta clase.



Esta rutina de carga es el equivalente funcional de la anterior rutina de carga descriptor de despliegue que utiliza la configuración XML de Spring. La siguiente rutina de carga en su lugar utiliza la configuración de Spring Java para un proceso de arranque y configuración de Java puro.



Por supuesto, usted no tiene que configurar todos sus contextos de aplicación de la misma manera. Usted puede mezclar y combinar los métodos de configuración que se utilizan durante el proceso de arranque. El siguiente ejemplo demuestra esto y también muestra cómo realizar el arranque de primavera en una aplicación independiente, como una aplicación de escritorio o demodo del servidor.



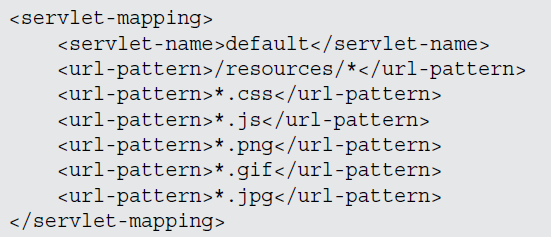
Observe el esfuerzo adicional necesario para asignar la rootContext como el contexto de los padres para el daemonContext y forkedProcessContext, y para llamar al inicio y métodos registerShutdownHook. En una aplicación web, el ContextLoaderListener y DispatcherServlet configuran automáticamente los contextos de aplicación de los padres y llamar a sus métodos de arranque al iniciar la aplicación y sus métodos de parada en parada aplicación. En una aplicación independiente, se debe llamar al método de arranque a sí mismo, y luego debe llamar parada cuando la aplicación se está cerrando. Como alternativa a llamar manualmente parada, puede llamar registerShutdownHook para registrar una devolución de llamada de apagado con la JVM que parará el contexto de aplicación de forma automática cuando la JVM empieza a salir.

# El Mapeo de Dispatcher Servlet

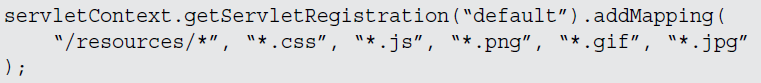
Los ejemplos que hemos visto hasta ahora, y seguiremos viendo a lo largo de este libro, muestran la DispatcherServlet asigna al patrón de URL /. A través de los años, hay Ha habido mucha confusión acerca de la asignación de la DispatcherServlet a la raíz de la URL de la aplicación, y es fácil de encontrar información incorrecta o incompleta sobre él en foros, blogs y otros sitios técnicos.

Puede asignar el DispatcherServlet a casi cualquier patrón de URL que desea. Algunos enfoques comunes son hacer un mapa de los patrones de URL / do / \*, \* .do, o \* .action, y algunos sitios utilizan un mapeo \* .html para que las páginas aparecen más estática de lo que son. Hay algunas cosas importantes a tener en cuenta, sin embargo. Lo más importante, no asigne un DispatcherServlet al patrón de URL / \*. En la mayoría de los casos el patrón de URL o bien debe empezar con o termina con un asterisco, pero cuando se mapea a la raíz de la aplicación, la barra diagonal solitario sin un asterisco es suficiente para conseguir el servlet para responder a todas las URL de su aplicación y al tiempo que permite la JSP mecanismo del contenedor de servlets para manejar las peticiones JSP. Un asterisco final hace que el contenedor de servlets para enviar incluso pedidos internos JSP para que servlet, lo cual no es deseable. (Esto se debe a que el contenedor de servlets asigna su manejador JSP después de sus servlets se han mapeado, dándole prioridad más baja.)

 Si va a asignar el DispatcherServlet a la raíz de la aplicación, hay tener en cuenta los recursos estáticos como páginas HTML, CSS y JavaScript, archivos e imágenes. Algunos tutoriales en línea demuestran cómo configurar Spring Framework para servir recursos estáticos, pero hacerlo no es necesario y no funciona bien. Cuando cualquier servlet se asigna a la raíz de la aplicación (sin asterisco), patrones de URL más específicas siempre prevalecen sobre ella. Permitiendo así su contenedor de servlets para servir recursos estáticos es tan simple como añadir asignaciones de dichos recursos al servlet denominado default (que todos los contenedores proporcionan de forma automática). Esto se puede lograr en el descriptor de despliegue de este modo:



O mediante programación en un ServletContainerInitializer (WebApplicationInitializer) o ServletContextListener de este modo:

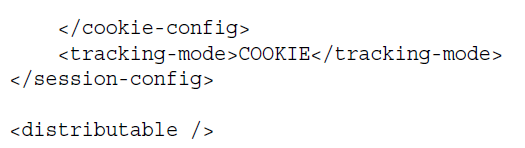
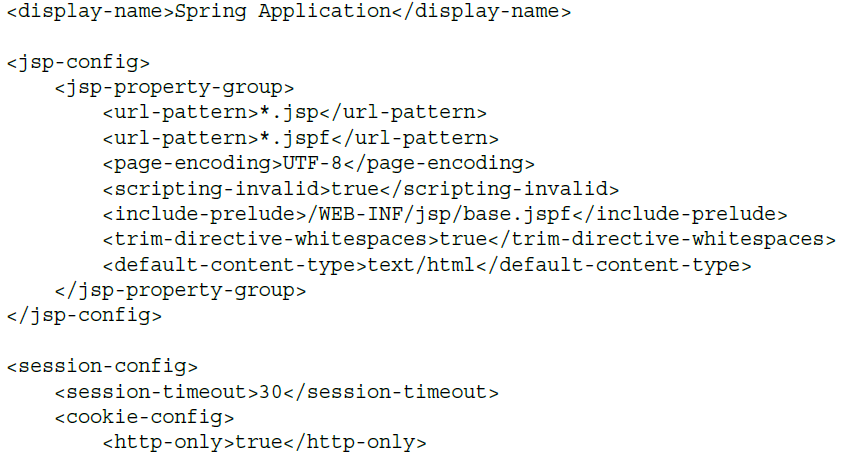


El servlet por defecto no tiene que ser declarados. El contenedor de servlets declara implícitamente en su nombre. Los patrones de URL que se muestran aquí son ejemplos de sólo usted puede decidir dónde quiere poner sus recursos estáticos y, por lo tanto, que los patrones de URL son las adecuadas.

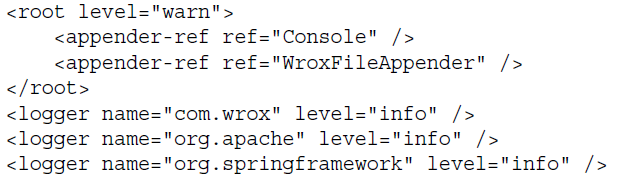
# Configuración de Spring Framework

Ahora que ya sabe cómo realizar el arranque Spring Framework, usted está listo para aprender sobre la configuración de la misma. El resto de este capítulo se sigue usando XML y bootstrapping programática de sus ejemplos, pero todos los capítulos futuros en este libro usen solamente bootstrapping programática. Como ya se ha visto, se puede configurar un contexto de aplicación de Spring usando XML o Java. No sólo se puede configurar diferentes contextos de aplicación en una sola aplicación de otra manera, también se puede configurar un contexto de aplicación individual con una mezcla de Java y XML. Este es un escenario poco común en las aplicaciones modernas, pero es necesario en algunos casos. Por ejemplo, la configuración de Java añadido en la Spring security 3.2 no cubre todos los escenarios de configuración, por lo que para algunos ejemplos en la Parte IV de este libro se debe utilizar la configuración de XML de Spring Security lugar. Además, las herramientas de primavera de los servicios Web que aprender sobre en el capítulo 17 no tienen soporte en la configuración de Java en absoluto. Al configurar un contexto de aplicación que utiliza estas herramientas, se puede configurar el contexto de aplicación utilizando Java, pero importar una configuración XML para la Seguridad del resorte o de los Servicios Web de la primavera. En este capítulo, aprenderá acerca de todas estas cosas.

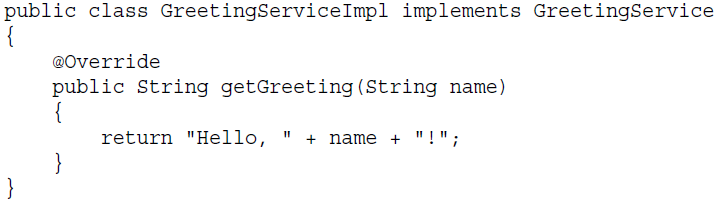
Esta sección utiliza los, Spring-Hybrid-config, y Spring -java-config proyectos Spring -Uno-Contexto-XML-Config, Spring -XML-config en el sitio de descarga de código wrox.com. Puede hacer referencia a estos proyectos a lo largo de esta sección o crear el código por su cuenta. Para empezar, los cuatro proyectos tienen el mismo código de descriptor de despliegue básico que estamos acostumbrados a ver en la Parte I. Así, seguir utilizando este descriptor de despliegue de la plantilla a lo largo del resto del libro:



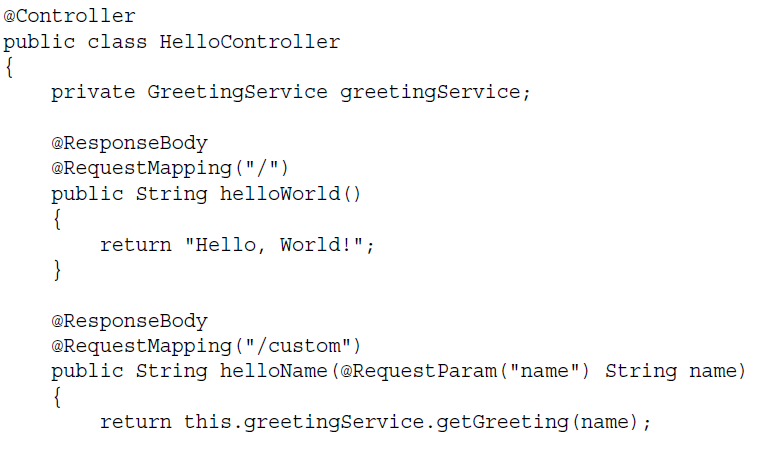
También debe actualizar log4j2.xml para asegurarse de que todos los mensajes apropiados terminan en su logs.In el siguiente ejemplo todos los mensajes se escriben en la consola y el archivo de registro, y aunque la mayoría de los mensajes están limitados a las advertencias, las clases en el com .wrox, org.apache, y paquetes org.springframework ingrese a nivel INFO.

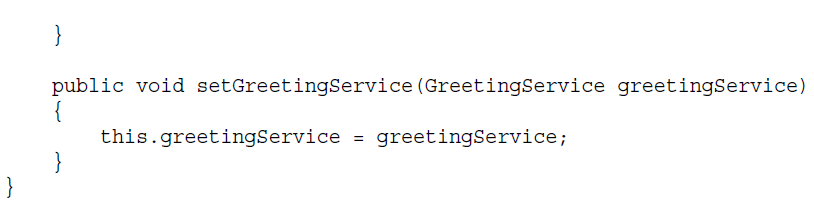


Los cuatro proyectos también incluyen la interfaz de servicio del saludo muy básico y su aplicación GreetingServiceImpl.



El HelloController que sigue responde a solicitudes Web en los cuatro proyectos. No es necesario que preocuparse por la semántica de estas clases, las anotaciones en el controlador, o el patrón regulador de servicio en este momento. Usted aprenderá acerca de estas cosas antes, pero en este momento usted debe recordar que con cuatro proyectos de bases idénticas, puede configurar Spring Framework muchas maneras diferentes y lograr los mismos fines.

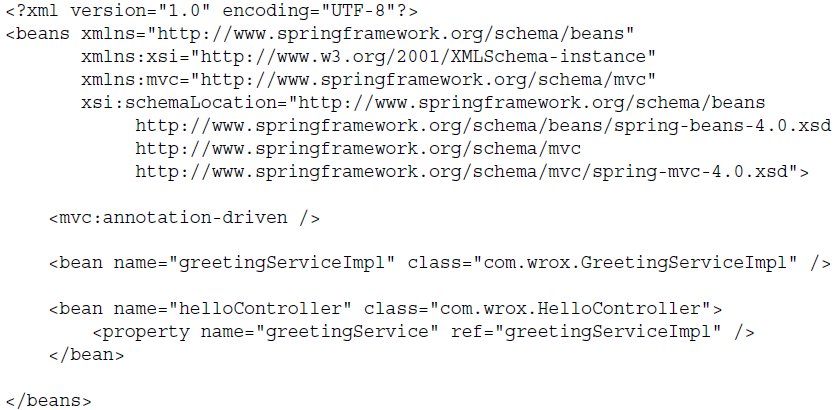




# Creación de una configuración XML

Usted recordará que Spring Framework gestiona los granos, y eso es lo principal que va a configurar cada vez que se configura Spring Framework. Que voy a escribir algunos de estos granos, como usted mismo GreetingServiceImpl y HelloController. Otros granos son granos de marco predeterminados que vienen con Spring Framework, tales como implementaciones de Application Context de Spring, ResourceLoader, BeanFactory, MessageSource y clases ApplicationEventPublisher, sólo para empezar.

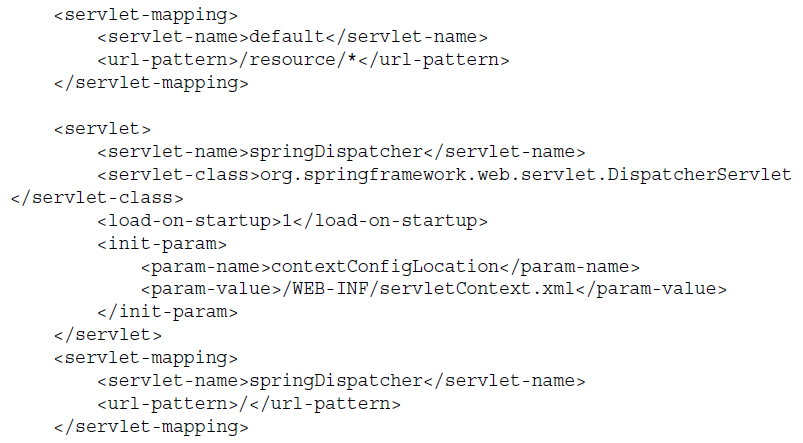
A decir Spring cómo configurar todos estos granos, utiliza el <beans> espacio de nombres XML, demostrado en el proyecto Primavera-Uno-Contexto-XML-Config. Echar un vistazo a su archivo / WEB-INF / servletContext.xml:



Este simple archivo XML dice a Spring para crear una instancia GreetingServiceImpl y HelloController e inyectar el bean greetingServiceImpl en propiedad greetingService del bean HelloController. La <beans> elemento es un elemento principal para contener la configuración de la primavera. Dentro de ella se puede usar casi cualquier otro elemento de configuración de Spring - hablando en términos generales, sin embargo, los elementos que se utilizan dentro de un elemento <beans> casi siempre como resultado la creación de los beans. Se utiliza un elemento <bean> para construir de forma explícita un grano de una clase dada, y puede especificar argumentos de constructor y propiedades para que el bean dentro de la sub-elementos <bean> del elemento. La <mvc: anotación impulsada> elemento de resorte instruye a utilizar las anotaciones como @RequestMapping, @RequestBody, @RequestParam, @PathParam, y @ResponseBody de corresponder las peticiones al controlador métodos. El uso de la etiqueta <mvc: anotación impulsada> elemento que realmente causa ciertos tipos de bean que se creen detrás de las escenas, pero usted no tiene que preocuparse de ellos por el momento. Existen para facilitar las solicitudes de asignación a los métodos de controlador. Se aprende más acerca de estos bean de marco a lo largo de los siguientes capítulos.

Se pueden hacer muchas cosas con la anotación @RequestMapping y sus socios, y explorará estos más en el capítulo 13. Por ahora, entender que el patrón de URL / se asigna al método helloWorld del HelloController y el patrón de URL / personalizado al método helloName . Estos patrones de URL son en relación con patrón de URL del DispatcherServlet, no a la URL raíz de la aplicación web. Sin embargo, debido a que el DispatcherServlet asigna a la raíz de la aplicación en este caso, da lugar a los patrones de URL también ser con respecto a la raíz de la aplicación. Si había trazado el DispatcherServlet a / DO / \*, los patrones de URL en las anotaciones @RequestMapping no habría cambiado, pero en la barra de direcciones del navegador Tendrían / hacer frente a ellos.

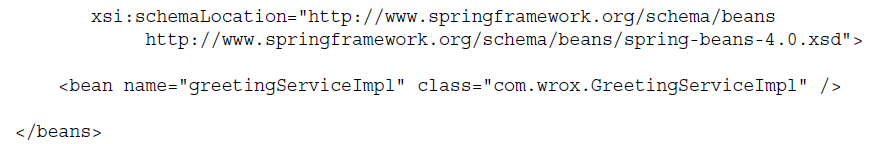
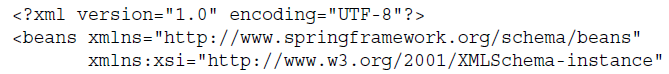
Ha creado la configuración de la Spring, pero todavía se necesita para arrancar ella. Para que utilizar el descriptor de despliegue, pero usted podría utilizar la misma facilidad de programa previo de Java, si se desea. Nótese la asignación al servlet por defecto para asegurar que Tomcat maneja recursos estáticos.



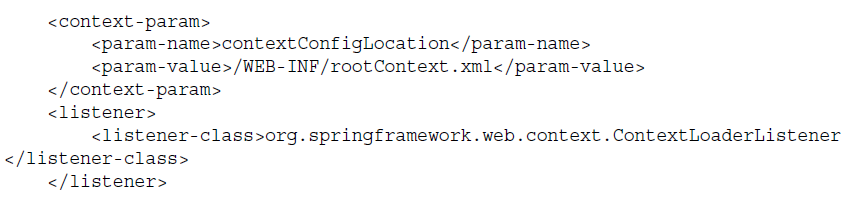
Ahora compilar el proyecto, iniciar Tomcat, e ir a http: // localhost: 8080 / xml / en su navegador favorito. Debería ver el texto "¡Hola, mundo!" En la pantalla. A continuación, pruebe la URL http: // localhost: 8080 / xml / encargo name = Nick. Ahora debería ver "Hola, Nick!" En la ventana del navegador. Modifica el parámetro de nombre, y la salida debe cambiar para que coincida. Su primera aplicación Spring Framework está trabajando!

Como se discutió en una sección anterior, a menudo es deseable tener dos diferentes contextos de aplicación en una aplicación web. Ha creado único aquí. Un patrón típico coloca todas las clases de negocios logicrelated en el contexto de aplicación de la raíz y los controladores en el contexto de aplicación Servlet. Para demostrar esto, cambie al proyecto Primavera-XML-Config o hacer los cambios simples a sí mismo.

Retire la declaración de bean greetingServiceImpl del archivo servletContext.xml (si no se quita, se crea una instancia dos veces) y crear un nuevo archivo que contiene sólo /WEB-INF/rootContext.xml que el frijol:



A continuación, sólo tiene que añadir el ContextLoaderListener al descriptor de despliegue.



Ahora tiene dos contextos de aplicación: El contexto de aplicación contiene la raíz frijol greetingServiceImpl; y contexto de aplicación de la DispatcherServlet hereda todos los granos del contexto raíz, contiene el grano HelloController, e inicializa mapeo controlador de anotación impulsada. Para hacer las cosas un poco más interesante, ajustar la asignación de solicitud en el método helloName del controlador.



Este método ahora se asigna a la misma URL que el método helloWorld, pero requiere el parámetro de nombre a estar presentes en la solicitud. La mera presencia o ausencia del parámetro de nombre determina qué método de Primavera envía la solicitud a. Para probar ese punto, compilar y empezar de nuevo su aplicación y probar el http: // localhost: 8080 / xml / y http: // localhost: 8080 / xml / Nick name = URL.

Nota probablemente ha notado que Spring Framework escribió mensajes de registro al consola de depuración y para su archivo application.log. Primavera se utiliza Log4J? No directamente. Primavera utiliza Apache Commons Logging como su API de registro. Debido a que su proyecto tiene el org.apache.logging.log4j: log4j-JCL Commons Logging artefacto puente en su camino de clase, eventos que los registros de Spring Framework a Commons Logging se pasan a Log4J y se escriben en la configuración de su registro. Mensajes de información o superior aparecen porque se establece el nivel de registro para org.springframework a INFO en el archivo de configuración de Log4j.

# Creación de una configuración híbrida

Como se puede imaginar, la configuración de Spring Framework con XML puede resultar tedioso, y el ejemplo anterior ni siquiera demostrar la MultiActionController y clases SimpleUrlHandlerMapping - un mecanismo de herencia (eliminado en la primavera Framework 4.0) para las solicitudes de asignación a los controladores que involucró XML extremadamente detallado y estrictas firmas de métodos y aceptar HttpServletRequests HttpServletResponses. En una aplicación empresarial de gran tamaño, usted podría

definir cientos de frijol, cada uno requiere tres o más líneas de XML. Este método de configuración rápida se ve no es mejor que la configuración de servlets utilizando XML en el descriptor de despliegue.

Si eres un fan de lo explícito de XML, pero no desea que los archivos de configuración de largo -

archivos de configuración que pueden llegar a ser extremadamente difícil de mantener - se puede crear una configuración híbrida que combina algunos de lo mejor de ambos mundos.

En el núcleo de esta configuración híbrida son los conceptos de la exploración de los componentes y configuración de anotación. Uso del análisis de los componentes, la primavera escanea el paquete o paquetes que se especifique para las clases con anotaciones específicas. Cualquier clase (en estos paquetes) anotado con @ org.springframework.stereotype.Component se convierten en granos de primavera-administrados, lo que significa que la primavera instancias e inyecta sus dependencias.

Otras anotaciones califican para este componente de exploración: Cualquier anotación en la que se anota con @Component se convierte en un componente de anotación, y cualquier anotación que se anota con otra anotación componente se convierte en un componente de anotación. Por lo tanto, las clases con anotada @Controller, @Repository, y @Service (todos en el mismo paquete que @Component) también se convierten en granos de primavera gestionados. Puede crear sus propias anotaciones de componentes, también. En el capítulo 17 que va a crear @WebController y @RestController anotaciones marcadas con @Controller para distinguir los controladores normales de los controladores de servicios web RESTful.

Otra anotación llave que funciona en conjunción con la configuración de anotación es @Org.springframework.beans.factory.annotation.Autowired. Puede anotar cualquier campo privado, protegido, o público, o un método mutador pública que acepta uno o más argumentos, con @Autowired. @Autowired Declara una dependencia o dependencias que la primavera se debe inyectar después de instancias, y también puede marcar un constructor. Por lo general, los frijoles, gestionados primavera deben tener constructores sin argumentos, pero en presencia de un único constructor @Autowired en una clase, Primavera utiliza ese constructor e inyecta todos los argumentos de constructor.

En cualquiera de estos casos, si la primavera no puede encontrar un bean de juego para una dependencia, se lanza y registra una excepción y no se puede iniciar. Del mismo modo, si encuentra grano de más de un juego para una dependencia, sino que también lanza y registra una excepción y no se puede iniciar. Puede evitar este segundo problema utilizando el org.springframework.beans.factory.annotation.Qualifier @ o el @org .springframework.context.annotation.Primary anotaciones. Cuando se utiliza en un campo @Autowired, método parámetro de método, o parámetro constructor, @Qualifier le permite especificar el nombre del bean que se debe utilizar. A la inversa, puede marcar un bean de componente-anotado con @Primary para indicar que debe preferirse cuando es uno de varios beans de candidatos que cumplen una dependencia.

Por lo tanto, se debe utilizar campos, mutators, o constructores con @Autowired? Ese debate se ha prolongado durante años. Algunos equipos prefieren tener @Autowired campos, ya que implica la menor cantidad de código. Otros equipos prefieren utilizar constructores @Autowired porque hace que sea imposible construir un objeto sin que se cumplan todas sus dependencias. Y otros prefieren usar mutators @Autowired porque, aunque se trata de la mayor parte del código, se evita tener campos no privados al mismo tiempo hacer pruebas unitarias más fácil, ya que no siempre se tiene que burlarse de cada dependencia acaba de construir una instancia. ¿Qué enfoque se utiliza en el mundo real es totalmente suya, pero en este libro va a anotar los campos en la mayoría de los casos, de manera que los ejemplos de código ocupan menos páginas.

**Nota**

Spring Framework también es compatible con ciertas anotaciones Java EE en el lugar de las anotaciones de propiedad. @ Javax.inject.Inject es totalmente sinónimo de @Autowired, y nada especial se requiere para hacer que funcione. Puede anotar cualquier campo, constructor, o el método que utiliza @Inject igual que lo haría con @Autowired. @ Javax.annotation.Resource también es tratado como sinónimo de @Autowired. Del mismo modo, @ javax.inject.Named es el equivalente de @Qualifier y tiene el mismo efecto en @Autowired o @Inject propiedades como @Qualifier hace. (Spring también tiene soporte para los calificadores personalizados utilizando el @ javax.inject.Qualifier meta-anotación.) ¿Qué anotaciones que se utiliza es en gran medida depende de usted; Sin embargo, el argumento se puede hacer que el uso de javax. \* anotaciones algo desata su aplicación desde la Spring y hace que sea más fácil cambiar a otro marco en una fecha posterior.

Como se ha mencionado anteriormente en la sección, un resorte que proporciona muchos Beens de marco predeterminado de forma automática, ya veces sus granos necesitan instancias de esos bens. En general, el uso de los bens de marco es desanimado porque hacerlo más lazos de su aplicación a Spring Framework y hace que sea más difícil para pasar a otra cosa. Sin embargo, debe utilizar estos bens cuando se desea utilizar ciertas características de gran alcance de Spring, como la mensajería de publicación-suscripción. Puede solicitar cualquiera de estos Bens para ser @Autowired o @Injected de Spring y obligará, pero muchas personas prefieren obtener estos granos de manera diferente para hacer la conexión a la Spring más evidente. Para ello, dispone de las interfaces de org.springframework.beans.factory.Aware. Aware es simplemente una interfaz de marcador para otras interfaces para extender lo que indica el conocimiento de ciertos granos de marco. Por ejemplo, la interfaz org.springframework.context.ApplicationContextAware especifica un método para proporcionar setApplicationContext la instancia Application Context del contexto actual (no principal) hacia sus granos. (Si es necesario, a continuación, puede obtener el contexto de los padres llamando al método getParent de Application Context).

Hay muchas otras interfaces que se extienden Consciente, pero algunas de las actividades complementarias más populares:

➤➤ ApplicationEventPublisherAware para obtener el grano utilizado para publicar los eventos de aplicación

➤➤ BeanFactoryAware para la obtención de la BeanFactory con el que se puede recuperar o crear manualmente los granos

➤➤ EnvironmentAware para la obtención de un objeto para el Medio Ambiente que se puede utilizar para obtener las propiedades de fuentes de propiedad

➤➤ MessageSourceAware para la obtención de la fuente del mensaje internacionalización

➤➤ ServletContextAware para la obtención de la ServletContext en ambientes de aplicaciones Web Java EE

➤➤ ServletConfigAware para la obtención de la ServletConfig en aplicaciones web habas contexto gestionados DispatcherServlet

Es también a menudo deseable realizar algún tipo de inicialización en una clase después de la inyección de todas sus dependencias pero antes de que se inyecta como una dependencia en cualquier otro beans. Este se logra fácilmente mediante la aplicación de la org.springframework.beans.factory

.InitializingBean interfaz. Después de todo configuración de su Bean es completa (dependencias se inyectan y los métodos Aware han sido llamados), Spring llama a su método afterPropertiesSet. Si no desea atar su aplicación a esta interfaz Primavera propietaria, en su lugar puede crear exactamente un método vacío cero discusión pública, anotado con @ javax.annotation.PostContstruct. Se va a llamar al mismo tiempo que afterPropertiesSet haría normalmente.

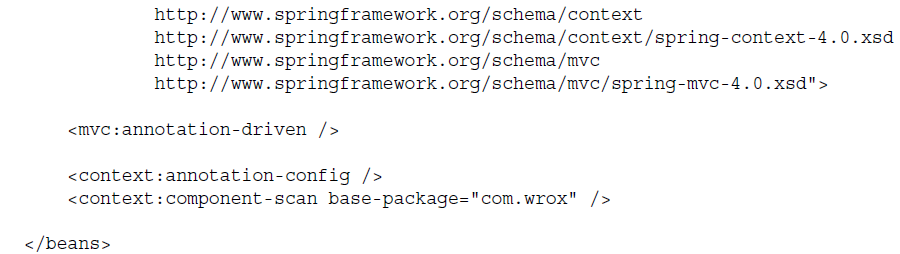
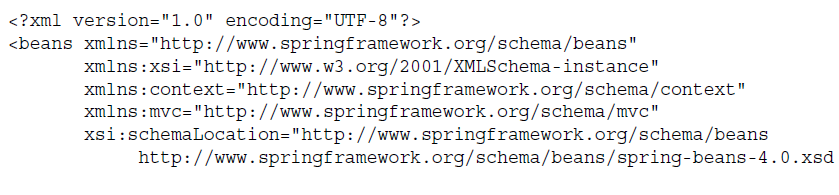
Cualquiera que sea el que utilice, se va a llamar incluso si el bean declaró no hay dependencias que necesitan inyección. Por otro lado, si el bean tiene que cerrar después de las referencias a ella han ido y antes de la Spring se apaga, se puede implementar de .DisposableBean org.springframework.beans.factory destruyen método o crear su propio público, nula, cero-argumento método anotado con @ javax.annotation.PreDestroy.

**Nota** Si desea un control más avanzado sobre el ciclo de vida de sus Beans que InitializingBean, @PostConstruct, DisposableBean, o @PreDestroy tienen que ofrecer, echar un vistazo a la documentación de la API para las interfaces .context.Lifecycle y SmartLifecycle org.springframework.

El inicio de la exploración de los componentes y configuración de anotación es una tarea sencilla utilizando el descriptor de despliegue de configuración existente y los archivos rootContext.xml y servletContext.xml. Esto se puede ver el cambio en el proyecto Spring-Hybrid-Config. En la configuración rootContext.xml, basta con quitar todas las definiciones de Beans y reemplazarlos con la <context: anotación-config> y <context: componente de exploración> elementos:



El atributo base-paquete instruye Spring Framework para escanear todas las clases en la ruta de clase en el com.wrox paquete o en cualquier sub-paquetes para @Component, @Controller, @Repository, y @Service. El archivo de configuración servletContext.xml similares todavía contiene el <mvc: anotación impulsada> elemento:



Todavía tiene dos restantes cosas de menor importancia para el cambio:

➤➤ Debe informar a Spring para crear una instancia del GreetingServiceImpl, por lo anotaciones en la clase con @Service.

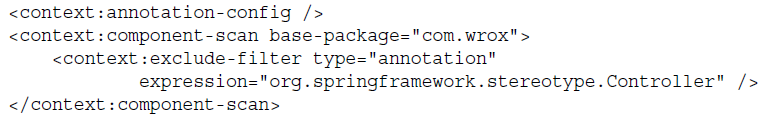


➤➤ Usted necesita decirle a Spring para inyectar el servicio de bienvenida en el HelloController, por lo que realizar anotaciones en el método mutador con @Autowired.

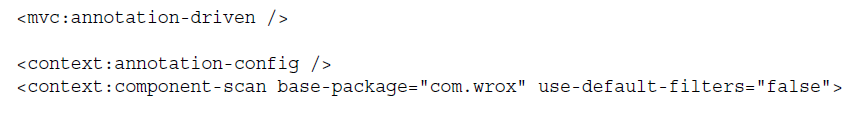


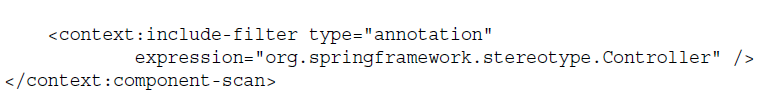
Es posible que ya haya notado un problema con esta configuración, la cual se puede observar mediante la colocación de un punto de interrupción en el mutador setGreetingService antes de iniciar Tomcat. Este punto de quiebre es golpeado dos veces por dos instancias diferentes de la HelloController y GreetingServiceImpl. De hecho, también se puede ver esto en el archivo de registro de aplicación. Primavera instancia el GreetingServiceImpl y HelloController dos veces: una vez en el contexto de la aplicación de la raíz y una vez en el contexto de la aplicación de la DispatcherServlet. ¿Por qué está pasando esto?

Recuerde que el análisis de componentes, de forma predeterminada, las exploraciones para todos los @Components, y usted tiene la exploración componente habilitado en ambos contextos de aplicación. Lo que realmente quiere es separar sus Beans. El contexto de aplicación de la raíz debe mantener los servicios, los depósitos, y otras piezas de la lógica de negocio, mientras que el contexto de aplicación de la DispatcherServlet debe contener controladores web. Afortunadamente, usted tiene una manera sencilla de modificar el algoritmo predeterminado componente de exploración. En el proyecto Primavera-Hybrid-Config esto ya se ha hecho por ti. En primer lugar, la configuración rootContext.xml añade una exclusión a la digitalización:



Esto excluye filtro le dice a la primavera para buscar todas las @Components excepto @Controllers. Aparte de esta exclusión, el patrón de escaneo por defecto sigue siendo válida. La configuración servletContext.xml adopta un enfoque ligeramente diferente y utiliza una lista blanca en lugar de una lista negra para decirle qué componentes de primavera para escanear para:





En este caso, el uso-default-filtros atributo establecido en false dice Primavera ignorar su patrón de exploración estándar. En lugar del filtro de exclusión especificada en el contexto de la raíz, hay un filtro incluir dice primavera para escanear sólo para @Controllers. Aunque el uso del-default-filtros es cierto, aún se pueden utilizar incluyen filtros para añadir a los filtros de análisis por defecto.

A pesar de que ha sacado todos los <bean> definiciones de los archivos de configuración, puede seguir utilizando el escaneo <bean> y el componente juntos. Es posible que tenga los casos en los que no se puede utilizar el escaneo de componentes para registrar un frijol. Un ejemplo de esto es el registro de una clase proporcionado por un tercero, como un bean: Debido a que usted tiene solamente una versión compilada de la clase, no se puede añadir anotaciones de primavera a la misma. Otro ejemplo es el registro de los granos de marco a los que no están registrados por defecto, tales como herramientas Java Persistence API. Cada vez que usted necesita, usted todavía puede especificar <bean> elementos en la configuración de la primavera para complementar lo que encuentra a través de la exploración de los componentes.

Compilar la aplicación y puesta en marcha de Tomcat en su IDE. Vaya a http: // localhost: 8080 / híbrido / y http: // localhost: 8080 / híbrido / name = Juan, y se dará cuenta de que la aplicación funciona exactamente igual que el uso de una configuración de XML-solamente.

# Configuración del resorte con Java usando @Configuration

¿Cómo se configura la primavera es en gran medida una cuestión de preferencia personal. Algunos equipos de desarrollo prefieren el nivel de detalle y el carácter descriptivo de texto sin formato de archivos XML. Sin embargo, hay algunas desventajas distintas a la configuración de la primavera el uso de XML:

➤➤ Es difícil de depurar una configuración XML. Debe descargar y adjuntar las fuentes de Spring Framework a su proyecto, así como saber dónde colocar los puntos de ruptura y cómo al paso a través de él. El código fuente para la primavera es enorme, y la depuración no es una tarea trivial.

➤➤ Es imposible probar una unidad de configuración XML. Sin duda, puede bootstrap mediante programación la configuración de la primavera dentro de una unidad de prueba, pero esto no es una prueba de unidad. Porque haciendo esto comienza su contexto de aplicación entero y todos sus cables frijoles, en realidad es una prueba de integración. No se puede probar las unidades aisladas de la configuración de la primavera si es basado en XML.

Configuración de Java puro de Resorte del marco permite configurar mediante programación el contenedor de la primavera por lo que puede depurar fácilmente y probetas unidad de su configuración. Esto también es útil para los equipos de desarrollo que simplemente no les gusta XML.

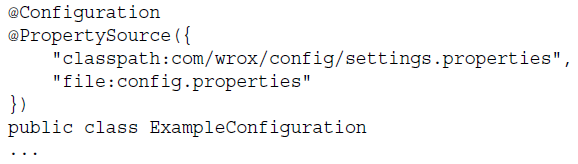
Anteriormente en este capítulo, usted exploró bootstrapping configuración de Spring programática mediante el AnnotationConfigWebApplicationContext. Cuando se utiliza esta clase, se registra clases de configuración con ella a través del método de registro. Estas clases de configuración (que debe ser anotado @ org.springframework.context.annotation.Configuration y que debe tener un constructor por defecto) se registran a través de los granos sin argumentos métodos anotados @Bean. Se pueden hacer muchas cosas poderosas con @Configuration clases, y más importante, se puede probar la unidad y depurar cada método, según sea necesario.

La anotación se @Configuration meta-anotado con @Component, es decir, su @Configuration Clases son elegibles para la inyección de dependencias utilizando @Autowired o @Inject; pueden poner en práctica cualquiera de las interfaces Aware, InitializingBean, o DisposableBean; y puede tener @PostConstruct Y @PreDestroy métodos. Esto es útil si una clase @Configuration necesita acceder directamente a un marco de frijol o un grano creado en otra clase de configuración. También es útil si se necesita para inicializar dos o más dependientes de los granos después se inyectan sus propias dependencias, pero antes de la primavera llama a sus métodos @Bean.

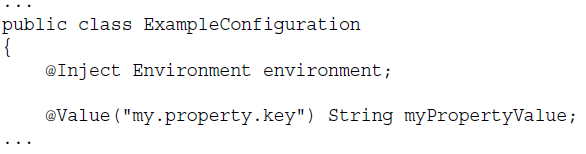
El hecho de que las clases son @Configuration @Components también significa que sus clases @Configuration son recogidos automáticamente si tiene el escaneo componente habilitado en el paquete que los contiene. Esto puede tener consecuencias deseadas y no deseadas. Si su clase @Configuration permite el escaneo de componentes en el mismo paquete que reside, sus granos podrían ser instanciadas en dos ocasiones, que no es bueno.

Por otro lado, si se divide la aplicación en muchos módulos, cada módulo puede tener su propio @Configuration, y usted puede crear un núcleo @Configuration que las exploraciones de componentes para todos los @Configurations de los módulos. El punto clave a tomar de esto es que siempre se debe tener cuidado al combinar @Configuration y escaneo de componentes.

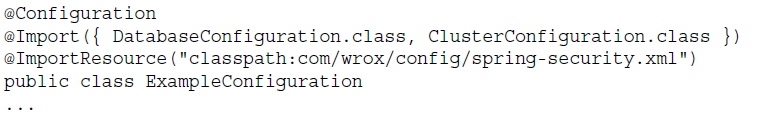
A menudo es deseable tener valores externalizados que impulsan su configuración. Por ejemplo, su aplicación puede requerir el uso de una fuente de datos JNDI, pero no querría codificar el nombre de esa fuente de datos. En cambio, es preferible almacenar el nombre de la fuente de datos en algún archivo de configuración cargado en el tiempo de configuración. El uso de la @ org.springframework.context .annotation.PropertySource anotación, se puede hacer precisamente eso.



Para acceder a las propiedades del origen de la propiedad, se puede usar un org.springframework inyectada ejemplo .core.env.Environment para obtener los valores manualmente, o puede utilizar la anotación @org.springframework.beans.factory.annotation.Value tener los valores de las ropiedades inyectaron automáticamente.



Es posible que no pueda o quiera contener toda la configuración en una sola clase. Por lo demás, es posible que no sea capaz de configurar todo con Java. Como se señaló anteriormente, algunas herramientas todavía pueden requerir configuración XML. Las anotaciones org.springframework.context.annotation.Import y @ImportResource proporcionan apoyo a estas necesidades.



En el fragmento de código anterior, Spring Framework se carga la configuración contenida en ExampleConfiguration, así como la configuración contenida en DatabaseConfiguration, ClusterConfiguration, y la primavera-security.xml. No inicializar y ejecutar estas configuraciones en un orden específico; En su lugar, se determina si cualquiera de las configuraciones dependen de frijoles proporcionados por cualquiera de las otras configuraciones, e inicializa las configuraciones en el orden correcto para cumplir estas dependencias.

Spring Framework tiene muchas funciones de configuración XML que ya hayas utilizado que no implican <bean> definiciones. Éstos dan lugar a las definiciones de frijol marco muy específicas detrás de las escenas, pero le permiten especificar la configuración con el mínimo esfuerzo e incluyen cosas tales como <mvc: anotación impulsada> y <context: componente de exploración>. Hay sustitutos de todas estas características de espacio de nombres XML utilizando un conjunto de anotaciones relacionados con la configuración. Puede marcar cualquier clase de configuración anotado @ con las siguientes anotaciones para que estas características. Va a aprender más acerca de estas anotaciones gradualmente a lo largo del resto del libro.

➤➤ @ComponentScan sustituye <context: componente de exploración> y permite el escaneo de componentes en el paquete o paquetes especificado. Al igual que su contraparte, puede habilitar o deshabilitar los filtros por defecto y especificar incluir y excluir filtros para afinar el algoritmo de escaneo de componentes.

➤➤ @EnableAspectJAutoProxy es el reemplazo para <AOP: aspectj-autoproxy> y activa el soporte para el manejo de las clases marcadas con la anotación de @Aspect AspectJ y asesorar a los métodos según sea apropiado usando programación orientada a aspectos.

➤➤ @EnableAsync reemplaza partes de la primavera <tarea: \*> espacio de nombres y permite la ejecución del método asíncrono @Async de primavera. Cuando se utiliza con la interfaz AsyncConfigurer, su @Configuration puede ajustar la configuración del comportamiento asincrónico.

➤➤ @EnableCaching habilita las funciones de gestión de caché de anotación impulsada de primavera y sustituye a la <cache: \*> espacio de nombres.

➤➤ @EnableLoadTimeWeaving reemplaza <context: carga-tiempo-tejedor>. Cambia la forma en varias características tales como @Transactional, @Configurable, @Aspect, y mucho más, el trabajo. En la mayoría de los casos no será necesario a menudo esta característica, pero se trata en la Parte III cuando se trabaja con la API de persistencia de Java. Es importante destacar que es mutuamente excluyente con @EnableAspectJAutoProxy.

➤➤ @EnableMBeanExport es el reemplazo para <context: mbean-exportación> e instruye a la primavera para exponer ciertos granos de marco y todas las habas @ ManagedResource anotado como JMX MBeans.

➤➤ @EnableScheduling sustituye al resto de la primavera <tarea: \*> espacio de nombres y activa la ejecución del método programado con @Scheduled, similar a la ejecución del método @Async habilitado por @EnableAsync.

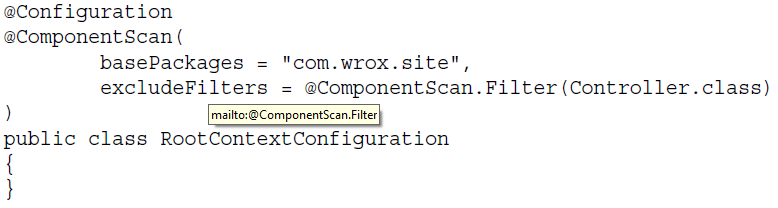
➤➤ @EnableSpringConfigured activa la inyección de dependencia en objetos que no están Springmanaged habas y reemplaza <context: configurado primavera->. Requiere tiempo de carga de tejer para apoyar esta función, ya que debe interceptar la construcción de objetos.

➤➤ @EnableTransactionManagement sustituye <tx: anotación impulsada> y permite la gestión de transacciones de base de datos de métodos aconsejados con la anotación @Transactional.

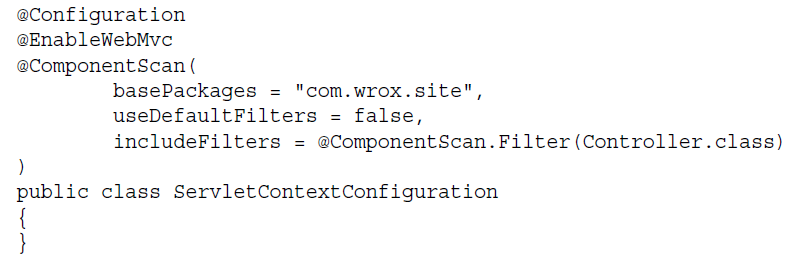
➤➤ @EnableWebMvc activa solicitud de mapeo controlador de anotación impulsada en lugar de <mvc: anotación impulsada->. Esto activa una configuración muy compleja, que a menudo se debe personalizar. Puede personalizar la configuración completa MVC Web al hacer que su clase @Configuration implementar WebMvcConfigurer o, más fácilmente, personalizar sólo las piezas que necesita mediante la ampliación de WebMvcConfigurerAdapter.

El proyecto Primavera-java-config muestra el uso de una configuración de Java puro WebApplicationInitializer de programa previo a las clases @Configuration de la raíz y contextos de aplicación DispatcherServlet. Las clases GreetingService, GreetingServiceImpl, y HelloController se han trasladado al paquete com.wrox.site estar separado de las clases de configuración en el paquete com.wrox.config. Este escaneo manera, el componente no detectará inadvertidamente las clases @Configuration.

La primera nueva clase en este proyecto, Raíz ContextConfiguration, es muy simple:



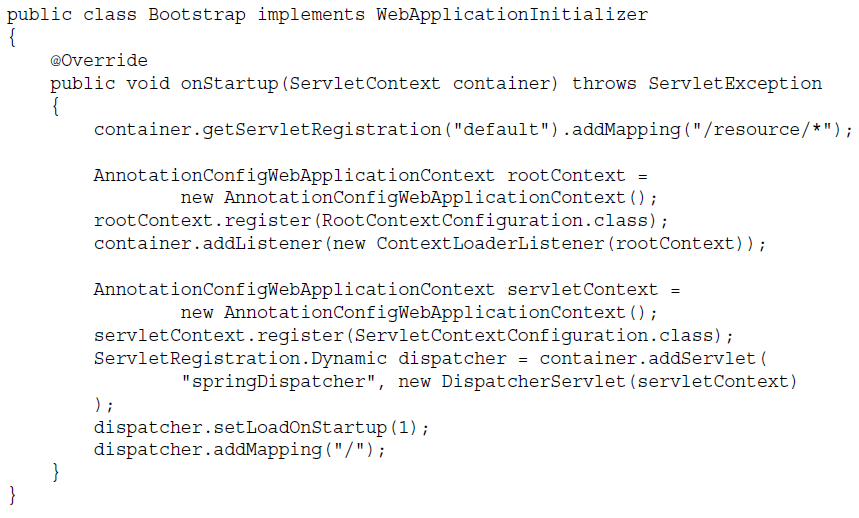
La anotación @ComponentScan dice Spring para escanear las clases del paquete com.wrox.site y todos los sub-paquetes y excluir clases @Controller, al igual que la configuración híbrida anteriormente en esta sección. La nueva clase ServletContextConfiguration es muy similar, la desactivación de filtros por defecto para escanear sólo para @Controllers, y permitiendo a las características Web MVC anotación impulsada.



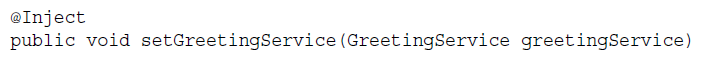
Usted probablemente ha notado que ninguna de estas clases tiene ningún métodos @Bean. Todos los granos que necesita en este momento se configuran automáticamente mediante el escaneo de los componentes y @EnableWebMvc, por lo que no tienen una necesidad de cualquier método @Bean.

**Nota** En la siguiente sección se ve un ejemplo de configuración manual con frijol @Bean, y en todo el resto de este libro, que se ve más y más ejemplos de la creación de habas personalizados con métodos @Bean.

Por supuesto, estas clases de configuración no se pueden arrancar a sí mismos, por lo que la clase Bootstrap implementa WebApplicationInitializer.



Los parámetros de inicio, oyente, y DispatcherServlet configurado en el descriptor de despliegue anteriormente en esta sección se han eliminado, y ahora contiene web.xml sólo de configuración básica JSP y sesión. El único otro cambio realizado fue con fines de demostración y no por necesidad: el HelloController utiliza ahora en lugar de @Inject @Autowired para declarar sus dependencias.



Ahora compilar la aplicación, iniciar Tomcat desde el IDE, y vaya a http: // localhost: 8080 / java / y http: // localhost: 8080 / java / name = Marte. Las funciones de aplicación de forma idéntica a las aplicaciones de primavera-Hybrid-config Primavera-XML-Config y.

# Utilizando definición de los perfiles de haba

Java es un lenguaje flexible que puede funcionar en muchos entornos, por lo que no hay razón para su aplicación Spring Framework debe ser rígido. Perfiles de definición de bean de Spring permiten convertir fácilmente secciones enteras de la configuración de encendido o apagado con un simple interruptor en la línea de comandos, en el descriptor de despliegue, o mediante programación. Esta funcionalidad es muy útil en muchos casos de uso, y, sin duda, usted tendrá sus propias ideas al final de esta sección. Algunos ejemplos son los siguientes:

* En un entorno de aplicaciones con muchos niveles, necesita un poco de frijoles para funcionar en un nivel, mientras que otros granos son de otro. El uso de perfiles de definición de frijol, se puede implementar una sola aplicación a todos los niveles, mientras que los controles perfil activado la que se registran los granos.
* Es posible escribir una aplicación para la reventa a trabajar en contra de muchos tipos diferentes de almacenamiento de datos. Cuando los usuarios finales comprar e instalar la aplicación que indican el tipo de almacén de datos que desea utilizar. Su aplicación podría tener un perfil de Java Persistence API que contiene los repositorios de la APP por persistir a bases de datos relacionales y un perfil de primavera de datos NoSQL NoSQL que contiene los repositorios para escribir en los almacenes de datos sin esquema. Sus usuarios pueden instalar el mismo archivo ejecutable mientras que un interruptor de configuración sencilla permite que el perfil correcto.
* Es posible que desee crear un desarrollo diferente, Control de Calidad y Producción perfiles. En el perfil de desarrollo, puede probablemente difícil de códigos de ciertos ajustes, como una conexión a una base de datos local que todos los desarrolladores deben crear. El perfil de aseguramiento de la calidad, del mismo modo, también podría tener una configuración no modificable que difieren de las máquinas de desarrollo. Sin lugar a dudas, su equipo entorno de producción tiene que cambiar la configuración de la aplicación sin tener que esperar en usted para cambiar y volver a compilar, por lo que el perfil de producción sería cargar esos ajustes desde un archivo de propiedades que sus técnicos pueden cambiar.