



Yo Libros Hablando Menciones

Custom Search

## Diseña tu propio arranque Spring Boot - parte 1

7 de febrero de 2016

Nicolas Fränkel

Desde su lanzamiento, Spring Boot ha sido un gran éxito: aumenta la productividad de los desarrolladores con su convención sobre la filosofía de configuración. Sin embargo, a veces, simplemente se siente demasiado mágico. Siempre he sido un oponente al autoenvío por esta misma razón. Y cuando algo no funciona, es difícil volver a la normalidad.

Esta es la razón por la cual quería profundizar en el mecanismo de arranque de Spring Boot, para comprender cada rincón y grieta. Esta publicación es la primera parte y se centrará en analizar cómo funciona. La segunda parte será un estudio de caso sobre cómo crear un iniciador.



### spring.factories

En la raíz de cada arranque Spring Boot se encuentra el `META-INF/spring.factories` archivo. Vamos a verificar el contenido de este archivo en `spring-boot-autoconfigure.jar`. Aquí hay un extracto de esto:

```
...  
  
# Auto Configure  
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\br/>org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\br/>org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\br/>org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\br/>org.springframework.boot.autoconfigure.MessageSourceAutoConfiguration,\br/>org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguration,\br/>org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\br/>...  

```

Ahora echemos un vistazo a su contenido. Por ejemplo, aquí está la clase `JpaRepositoriesAutoConfiguration`:

```
@Configuration  
@ConditionalOnBean(DataSource.class)  
@ConditionalOnClass(JpaRepository.class)  
@ConditionalOnMissingBean({ JpaRepositoryFactoryBean.class, JpaRepositoryConfigExtension.class })  
@ConditionalOnProperty(prefix = "spring.data.jpa.repositories", name = "enabled",  
    havingValue = "true", matchIfMissing = true)  
@Import(JpaRepositoriesAutoConfigureRegistrar.class)  
@AutoConfigureAfter(HibernateJpaAutoConfiguration.class)  
public class JpaRepositoriesAutoConfiguration {}  

```

Hay un par de cosas interesantes para tener en cuenta:

1. Es una `@Configuration` clase estándar de primavera
2. La clase no contiene ningún código "real", pero importa otra configuración `JpaRepositoriesAutoConfigureRegistrar`, que contiene el código "real"
3. Hay un par de `@ConditionalOnXXX` anotaciones usadas
4. Parece que hay una gestión de dependencia de órdenes de algún tipo con `@AutoConfigureAfter`

Los puntos 1 y 2 se explican por sí mismos, el punto 4 es bastante sencillo, así que centrémonos en el punto 3.

### @ Anotaciones condicionales

### últimas publicaciones

- [Haz tu vida más fácil con Kotlin stdlib](#)
- [Aprovechar al máximo las conferencias](#)
- [En la escasez de desarrolladores](#)
- [¿Es la programación orientada a objetos compatible con un contexto enterprise?](#)
- [Los múltiples usos de git rebase --onto](#)
- [Navegador alternativo en Vaadin](#)
- [Migración de una aplicación Spring Boot a Java 9 - Módulos](#)
- [Migración de una aplicación Spring Boot a Java 9 - Compatibilidad](#)
- [Construcciones verdaderamente inmutables](#)
- [Pasar por alto las comprobaciones de Javascript](#)

Si ayer no empezaste a trabajar con Spring, tal vez tengas información sobre la `@Profile` anotación. Los perfiles son una forma de marcar un método de devolución de beans como *opcional* . Cuando se activa un perfil, se llama al método anotado de perfil relevante y el bean de retorno contribuye a la fábrica de beans.

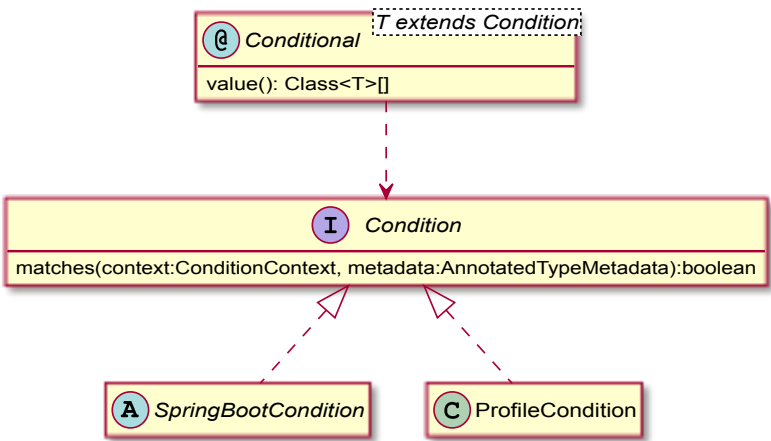
Hace algún tiempo, se `@Profile` veía así:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Profile {
    String[] value();
}
```

Curiosamente, `@Profile` ha sido reescrito para usar la nueva `@Conditional` anotación:

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
@Documented
@Conditional(ProfileCondition.class)
public @interface Profile {
    String[] value();
}
```

Básicamente, una `@Conditional` anotación solo apunta a a `Condition` . A su vez, una condición es una interfaz funcional con un único método que devuelve a `boolean` : if `true` , el `@Conditional` método anotado es ejecutado por Spring y su objeto de retorno se agrega al contexto como un bean.



Hay muchas condiciones disponibles listas para usar con Spring Boot:

Condición	Descripción
OnBeanCondition	Comprueba si hay un frijol en la fábrica de Spring
OnClassCondition	Comprueba si una clase está en classpath
OnExpressionCondition	Evalúa una expresión SPeL
OnJavaCondition	Verifica la versión de Java
OnJndiCondition	Comprueba si existe una rama JNDI
OnPropertyCondition	Verifica si existe una propiedad
OnResourceCondition	Comprueba si existe un recurso
OnWebApplicationCondition	Comprueba si existe un WebApplicationContext

Esos pueden combinarse junto con condiciones booleanas:

Condición	Descripción
AllNestedConditions	Operador AND
AnyNestedConditions	O operador

Condición	Descripción
NoneNestedCondition	NO operador

Las `@Conditional` anotaciones dedicadas apuntan a esas anotaciones. Por ejemplo, `@ConditionalOnMissingBean` apunta a la `OnBeanCondition` clase.

## Tiempo para experimentar

Vamos a crear una clase de configuración anotada con `@Configuration`.

El siguiente método se ejecutará en todos los casos:

```
@Bean
public String string() {
    return "string()";
}
```

Este no lo hará, ya que `java.lang.String` es parte de la API de Java:

```
@Bean
@ConditionalOnMissingClass("java.lang.String")
public String missingClassString() {
    return "missingClassString()";
}
```

Y este lo hará, por la misma razón:

```
@Bean
@ConditionalOnClass(String.class)
public String classString() {
    return "classString()";
}
```

## Análisis de la configuración anterior

Armados con este nuevo conocimiento, analicemos la `JpaRepositoriesAutoConfiguration` clase anterior.

Esta configuración se habilitará si, y solo si se cumplen todas las condiciones:

### @ConditionalOnBean (DataSource.class)

Hay un frijol de tipo `DataSource` en el contexto de primavera

### @ConditionalOnClass (JpaRepository.class)

La `JpaRepository` clase está en la ruta de clase, es *decir*, el proyecto tiene una dependencia en Spring Data JPA

### @ConditionalOnMissingBean

No hay frijoles de tipo `JpaRepositoryFactoryBean` ni `JpaRepositoryConfigExtension` en el contexto

### @ConditionalOnProperty

El archivo estándar `application.properties` debe contener una propiedad nombrada

`spring.data.jpa.repositories.enabled` con un valor de `true`

Además, la configuración se ejecutará después `HibernateJpaAutoConfiguration` (si se hace referencia a esta última).

## Conclusión

Espero haber demostrado que los iniciadores de Spring Boot no son mágicos. Únete a mí la próxima semana para un simple estudio de caso.


### Para llegar más lejos:

- [Creando tu propia configuración automática](#)
- [Comprender el arranque de primavera](#)

Java [arranque de primavera](#)


[Compartir](#)


[Pío](#)



[Compartir](#)


[Compartir](#)


[Reddit](#)

0 Comments

A Java Geek

 Javier Martín Alon... ▾ Recommend Share

Sort by Newest ▾



Be the first to comment.

 Subscribe Add Disqus to your siteAdd DisqusAdd Privacy[Diseñando su propio arrancador Spring Boot - parte 2](#)[Por qué no debe confiar en la entrada de contraseña HTML](#)