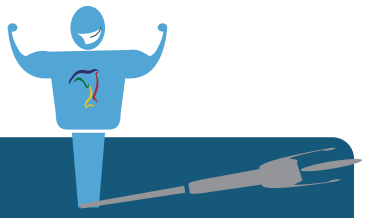
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

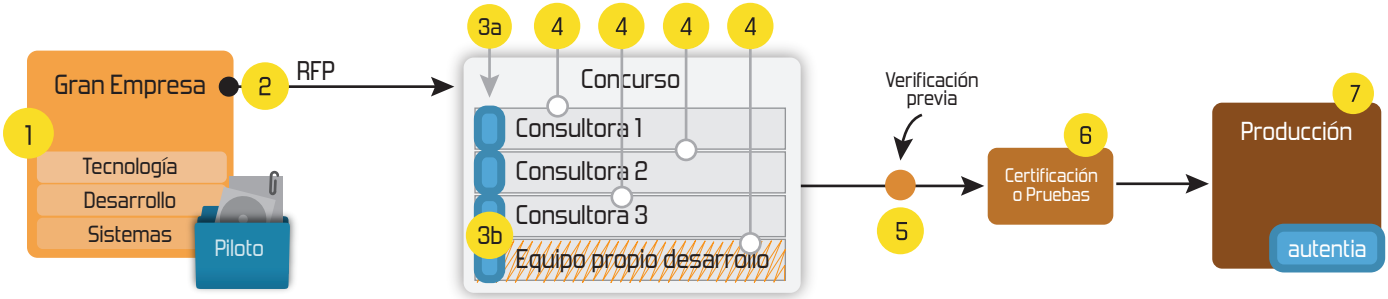
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Daniel Ventas López

Becario en autentia.

Ingeniero técnico en informática, especialidad en sistemas.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2013-10-30

Cómo minimizar código Java y XML en Spring

0. Índice de contenidos.

Este es el segundo tutorial de una serie que nos ayudará a dar los primeros pasos con el framework Spring 3.

- Spring Container e Inyección de Dependencias.
- Hola Mundo con Spring 3 MVC.

1. Entorno

Este tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Mac Book Pro 17" (2,93 Ghz Intel Core 2 Duo, 8 GB DDR3)
- Sistema Operativo: Mac OS X Mountain Lion 10.8.5
- Spring Framework 3.0

2. Introducción

Hemos visto cómo crear nuestras beans y cómo insertar las dependencias entre ellas a partir de un fichero XML. En este tutorial vamos a reducir la complejidad y la cantidad de líneas de código que hacen falta para configurar correctamente nuestros beans.

3. Vamos al lío

Como introducción partimos de la base de que queremos declarar un bean como el siguiente, y nos queremos ahorrar ciertos pasos.

view plain print ?

```
01. <bean id="coche" class="nuestroPath.vehiculoMotorizado">
02.
03.     <property name="marca">
04.         <value>nuestroPath.marcas.laquesea</value>
05.     </property>
06.     <property name="numeroRuedas" value="4">
07.
08. </bean>
```

Spring incorpora dos importantes características:

Autowiring reduce o incluso elimina la necesidad de configurar <property> o <contructor-arg> en nuestras beans.

Autodiscovery nos elimina la necesidad de declarar cada bean, ya que los busca en el path que le indiquemos y los encuentra gracias a las anotaciones que realizaremos en los propios objetos para indicar a spring qué tipo de bean es.

Para que funcione correctamente es necesario que esté correctamente indicado tanto las beans que queremos que encuentre o autodescubra, como las beans que queremos que nos inyecte, y no haya ambigüedades.

Autowiring tiene cuatro tipos por las que referenciar objetos, que son:

- byName, tiene que coincidir el nombre o id con la propiedad.
- byType, tiene que coincidir el mismo tipo de objeto con la propiedad a la que estamos inyectando.

- constructor, intenta asignar a los parámetros del constructor las beans que encajen en tipo.
- autodetect, intenta aplicar el tipo por constructor, si falla, lo intenta por el tipo byType.

byName

Para inidcar cada uno de los tipos, se indica como atributo al crear el bean así:

view plain print ?

```
01. <bean id="coche"
02.     class="vehiculoMotorizado" autowired="byName">
03. </bean>
```

Además sabemos que coche tiene dos atributos, marca y numeroRuedas.

Ahora declaramos un bean como el siguiente:

view plain print ?

```
01. <bean id="marca"
02.     class="marcas.laquesea">
03. </bean>
```

Spring reconoce por nombre que la propiedad de coche llamada marca coincide con la bean marca y al haber activado el atributo autowired="byName", Spring tiene todo lo que necesita para enlazarlos. Nos hemos ahorrado definir esa propiedad dentro de la bean coche.

byType

Es similar a byName, pero en este caso, en vez de buscar las coincidencias entre los nombres de las beans y los nombres de los atributos a inyectar, se basa en el tipo de objeto que es cada uno. Lo explico mejor con un ejemplo:

En el mismo caso de antes, la propiedad numeroRuedas es un integer. Bataría con crear una bean de tipo integer como la siguiente para que Spring lo enlazara a coche.

view plain print ?

```
01. <bean id="numeroRuedas"
02.     class="java.lang.Integer" value="4">
```

Una cosa a tener en cuenta, es que si hay más de un bean que encaje en nuestro criterio de autowiring, Spring no va a saber cuál escoger, y saltará una excepción.

Para solucionar el problema hay dos maneras(dos propiedades dentro de las beans):

- **Primary:** Es un valor booleano en el cual indica si esa bean es prioritaria para autowired, el problema es que por defecto todas las beans vienen con el atributo a true, por lo que para elegir una se tiene que poner a false este atributo en todas las demás.
- **Autowire-candidate:** Es un valor booleano, que si está false indica que no se tiene en cuenta a la hora de elegir beans

Autowired por constructor

Solo válido si la bean está configurada para la inyección de dependencias. Tiene las mismas restricciones que byType. Además hay que tener en cuenta que si tiene varios constructores que puede satisfacer Spring, no sabrá cuál usar.

Su declaración es:

view plain print ?

```
01. <bean id="coche"
02.     class="vehiculoMotorizado" autowire="constructor">
```

Se puede poner un autowired por default, declarándolo cuando se instancia beans.

Por ejemplo:

view plain print ?

```
01. <beans xmlns="http://www.springframework.org/schema/beans"
02.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.     xsi:schemalocation="http://www.springframework.org/schema/beans
04.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
05. default-autowire="byType" >
06.
07. ... declaración de beans ...
08.
09. </beans>
```

Y por último decir que también se puede hacer mixto, declarar las propiedades o los argumentos del constructor manualmente en el bean y los que se quiera que enlace Spring dejarlos sin declarar.

Teniendo en cuenta que para inicializar un atributo a null se haría como en el atributo marca:

view plain print ?

```
01. <bean id="coche"
02.     class="vehiculoMotorizado"
03.     <property name="marca"><null/></property>
04.     <property name="numeroRuedas" value="4">
```

Anotaciones

A partir de Spring 2.5 la manera más interesante de enlazar nuestras beans es a través de las anotaciones.

Para decirle a Spring que use las anotaciones, se le indica con el elemento <context:annotation-config/>. Veámos un

ejemplo:

```
view plain  print  ?
01. <beans xmlns="http://www.springframework.org/schema/beans"
02.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.     xmlns:context="http://www.springframework.org/schema/context"
04.     xsi:schemalocation="http://www.springframework.org/schema/beans
05.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
06.     http://www.springframework.org/schema/context
07.     http://www.springframework.org/schema/context/spring-context-3.0.xsd">
08.
09.
10. <context:annotation-config/>
11.
12. ... declaración de beans ...
13.
14. </beans>
```

Con <context:annotation-config/> le decimos a Spring que estamos usando anotaciones en nuestras clases java para enlazar los valores de las propiedades, métodos o constructores.

Spring 3.0 soporta 3 diferentes anotaciones para autowiring:

- @Autowired
- @Inject
- @Resource

Las tres anotaciones se pueden usar indistintamente y prácticamente tienen la misma funcionalidad, pero tienen pequeñas variaciones de comportamiento que se salen del tutorial explicarlas, por lo que haremos los ejemplos con @Autowired y recomiendo revisar la página de SpringFramework para obtener toda la información que falte en este tutorial.

Usaremos @Autowired principalmente en tres casos:

- Delante de un método, encontes acutará como si marcamos en el XML que inyecte una property de un bean. Ejemplo:

```
view plain  print  ?
01. @Autowired
02. public void setMarca(Marca marca){
03.     this.marca = marca;
04. }
```

- Ponerlo delante de una variable, obteniendo el mismo resultado que en el caso anterior:

```
view plain  print  ?
01. @Autowired
02. Marca marca;
```

- Y por último veremos cómo se puede poner antes de un constructor de una clase, que será como si se indica por XML que inyecte en el constructor de la bean correspondiente, como vimos antes.

```
view plain  print  ?
01. @Autowired
02. public coche(Marca marca){
03.     this.marca = marca;
04. }
```

Es muy recomendable que se revise la anotación @Qualifying (@Named para @Inject) para personalizar la forma de inyectar las beans ajustándolas a nuestros requisitos. Se sale del temario de este tutorial para no hacerle excesivamente extenso.

Descubrimiento automático de beans.

Para que Spring automáticamente busque las beans sin describirlas en el XML es necesario cambiar la declaración <context:annotation-config> por <context:component-scan>. Además hace falta indicarle la base del paquete donde queramos que busque. Ejemplo:

```
view plain  print  ?
01. <beans xmlns="http://www.springframework.org/schema/beans"
02.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.     xmlns:context="http://www.springframework.org/schema/context"
04.     xsi:schemalocation="http://www.springframework.org/schema/beans
05.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
06.     http://www.springframework.org/schema/context
07.     http://www.springframework.org/schema/context/spring-context-3.0.xsd">
08.
09.
10. <context:component-scan
11.     base-package="ruta.paqueteAplicacion">
12. </context:component-scan>
13.
14. ... declaración de beans ...
15.
16. </beans>
```

Con eso buscaría en nuestra ruta todos los archivos java que en su interior tengas estas anotaciones (además de otras):

- @Component: Es de propósito general, indica que la clase es un componente Spring.
- @Controller: Indica que la clase define un controlador Spring.
- @Repository: La clase es un repositorio de datos.
- @Service: La clase define un servicio.

Conclusiones

Spring es un framework que nos evita muchos problemas y elimina dependencias. Además la comunidad que lo mantiene contínuamente está sacando mejoras.

Cualquier duda o sugerencia en la zona de comentarios.

Saludos.

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share |

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» **Registrate** y accede a esta y otras ventajas «