

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

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

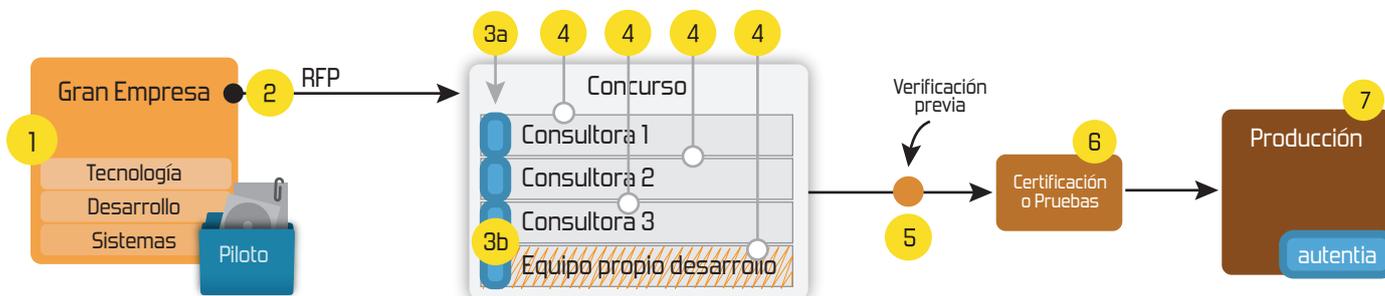
## 1. Desarrollo de componentes y proyectos a medida



## 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

Tareas programadas (Quartz)  
 Gestor documental (Alfresco)  
 Inversión de control (Spring)

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

BPM (jBPM o Bonita)  
 Generación de informes (JasperReport)  
 ESB (Open ESB)



**adictos al trabajo**

Encuentra un trabajo que te guste y no volverás a trabajar ni un sólo día de tu vida  
Confucio

E-mail:

Contraseña:

Entrar

[Inicio](#) [Quiénes somos](#) [Tutoriales](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#)  
[Charlas](#) [Más](#)

Estás en:

[Inicio](#) [Tutoriales](#) [Spring 3 Java Config Style](#)



**DESARROLLADO POR:**  
 Francisco Javier Martínez Páez

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Telecomunicaciones

Puedes encontrarme en [Autentia](#):  
 Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Catálogo de servicios Autentia



Ads by Google

[Java](#)

[Tutoriales](#)

[Últimas Noticias](#)  
[Java Web Ajax](#)

Fecha de publicación del tutorial: 2010-09-23



Share |

[Regístrate para votar](#)

## Spring 3 Java Config Style

### Introducción

Creo que a estas alturas a nadie le voy a descubrir el [framework de Spring](#) y lo que ha aportado a la comunidad Java/JEE sin pedir nada a cambio. No obstante, es probable que las novedades de Spring 3 se nos hayan escapado, y es por eso que he decidido hacer una serie de tutoriales comentando algunas de ellas.

### Configuración de Spring

Antes de la versión 2.5 la única manera de configurar el contenedor de Spring era a través de ficheros XML. Esta manera, que es por otro lado suficiente, a veces se puede convertir en un engorro al proliferar el número de "Beans" de nuestros proyectos. Tras la aparición de las anotaciones en Java 1.5, Spring introdujo la posibilidad de configurar el contexto también a través de éstas o combinando ambas. Yo suelo utilizar anotaciones para mis clases (las de mi proyecto) y XML para las clases externas (no había más remedio, ya que no se "puede" anotar el código que no es mío)

[iii](#) Alcanzamos los 900 tutoriales !!!

[Persiguiendo la felicidad, haciendo realidad los sueños](#)

[Networking sobre Patines... tenemos las pruebas](#)

[¿Quieres trabajar en Autentia o que te ayudemos a encontrar un nuevo trabajo?](#)

[Autentia patrocina un nuevo Coderetreat en Madrid junto con agilismo.es y Eden](#)

A partir de Spring 3.0 aparece una nueva manera de configurar el contexto al estilo JavaConfig:

## Configuración del contenedor con Java

Esta nueva forma de configurar el contenedor está basada principalmente en dos nuevas anotaciones y en una nueva implementación de ApplicationContext:

- **@Configuration**: Usar una anotación de este tipo en una clase sirve para indicarle a Spring que ésta es una clase contenedora de definiciones de "Beans". Es importante hacer notar que una clase anotada con @Configuration será también registrada como "bean", y por lo tanto también podremos usar las anotaciones de inyección de dependencias (@Autowired, @Inject o @Resource)
- **@Bean**: Anotar un método de esta manera es equivalente a usar <bean> en un fichero de configuración tradicional de Spring. Además no es imprescindible para poder usar esta anotación que la clase sea anotada con @Configuration, sino que también serán "entendidas" aquellas que aparezcan en otras clases @Component (y por ende @Service, @Repository o @Controller)
- **AnnotationConfigApplicationContext**: Esta clase será la encargada de levantar el contexto de Spring y entiende no sólo las anotaciones tradicionales sino también las dos anteriores.

En el ejemplo que se muestra a continuación se muestran dos ejemplos de configuración que son equivalentes:  
Estilo "Cocina tradicional":

[view plain](#) [copy to clipboard](#) [print](#) ?

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/bean
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <import resource="dao.xml"/>

  <bean id="beanA" class="com.autentia.tutoriales.A">
    <constructor-arg ref="dataSource" />
  </bean>
  <bean id="beanB" class="com.autentia.tutoriales.B">
    <constructor-arg ref="beanA" />
  </bean>
</beans>
```

Este es el dao.xml importado previamente:

[view plain](#) [copy to clipboard](#) [print](#) ?

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/bean
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataS
        <property name="driverClassName" value="XXX" />
        <property name="url" value="XXX" />
        <property name="username" value="XXX" />
        <property name="password" value="XXX" />
  </bean>
</beans>
```



Histórico de NOTICIAS

Últimos Tutoriales

Reunión Madrid Ágil 21-09-2010: Estrategias de Branches, y división de una historia en tareas

Instalación de subversion

Validación de acciones sobre botones en Jsf con Icefaces

Mercurial, un sistema de control de versiones distribuido

Instalación de Ubuntu Server 8.04 LTS 32bits en una máquina virtual con VMWare Workstation

Últimos Tutoriales del Autor

Apache Cassandra, ¿Qué es esto que tanto ruido hace?

JEE6, haciéndolo fácil.

Hibernate Search, Bridges, Analizadores y más

Instalación y configuración de Eclipse Galileo

Activar Single Sign On en JBoss

Estilo "Nouvelle cuisine":

```

view plain copy to clipboard print ?
01. @Configuration
02. // Esta anotación es equivalente al import de xml
03. @Import({DataSourceConfig.class})
04. public class SpringJavaConfigStyle {
05.
06.
07.     @Autowired
08.     private DataSource dataSource;
09.
10.
11.     @Bean
12.     public A beanA() {
13.         return new A(dataSource);
14.     }
15.
16.     @Bean
17.     public B beanB() {
18.         return new B(beanA());
19.     }
20. }

```

```

view plain copy to clipboard print ?
01. @Configuration
02. public class DataSourceConfig {
03.
04.
05.     @Bean
06.     public DataSource dataSource() {
07.
08.         Properties props = new Properties();
09.         // Configuración de Las propiedades
10.         // ...
11.         return new DriverManagerDataSource("XXX",props);
12.
13.     }
14.
15. }

```

Para levantar el contexto:

```

view plain copy to clipboard print ?
01. ...
02. public static void main(String[] args) {
03.     ApplicationContext ctx = new AnnotationConfigApplicationCc
(SpringJavaConfigStyle.class);
04.     B b = ctx.getBean("beanB", B.class);
05.     ...
06. }
07. ...

```

Otra opción equivalente para la inyección de dependencias, sería:

```

view plain copy to clipboard print ?
01. @Configuration
02. @Import({DataSourceConfig.class})
03. public class SpringJavaConfigStyle {
04.
05.
06.     @Autowired
07.     private DataSourceConfig dataSourceConfig;
08.

```

Síguenos a través de:



Últimas ofertas de empleo

2010-08-30  
 Otras - Electricidad - BARCELONA.

2010-08-24  
 Otras Sin catalogar - LUGO.

2010-06-25  
 T. Información - Analista / Programador - BARCELONA.

```

09.     @Bean
10.     public A beanA() {
11.         return new A(dataSourceConfig.dataSource());
12.     }
13.
14.     @Bean
15.     public B beanB() {
16.         return new B(beanA());
17.     }
18.
19. }

```

Por otro lado, la anotación @Bean recibe diversos parámetros para configurar los nombres del "Bean", los métodos del ciclo de vida (init y destroy), y si queremos usar autowire en las propiedades del "Bean". Usar el método "init" no tiene ya demasiado sentido porque se puede realizar después de la instanciación. Además si queremos modificar el ámbito del "Bean", bastaría con usar la anotación @Scope:

```

view plain copy to clipboard print ?
01. @Configuration
02. @Import({DataSourceConfig.class})
03. public class SpringJavaConfigStyle {
04.
05.
06.     @Autowired
07.     private DataSourceConfig dataSourceConfig;
08.
09.     @Bean
10.     public A beanA() {
11.         return new A(dataSourceConfig.dataSource());
12.     }
13.
14.     @Bean(name = { "beanB", "mrBean" })
15.     @Scope("prototype")
16.     public B beanB() {
17.         return new B(beanA());
18.     }
19.
20. }

```

## Mezclando XML y JavaConfig

Debido a que el estilo JavaConfig no es capaz de reemplazar completamente al estilo XML (principalmente por el tema de los namespaces), conviene utilizar una combinación de ambas. Para poder hacer esto, como todo en Spring tenemos varias opciones, más en concreto dos:

### El XML "manda":

En este caso, usaremos una de las versiones de "XmlApplicationContext" y la cargaremos al estilo tradicional. Pero para poder activar en este caso la búsqueda de anotaciones de configuración es necesario incluir:

```

view plain copy to clipboard print ?
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/bean
           http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-
context-3.0.xsd">

```

```

    <!-- Activa la búsqueda de anotaciones -->
    <context:annotation-config />

    <!--
    Registraremos el bean de configuraci&ocute;n. El otro no es necesar
    ->
    <bean class="com.autentia.tutoriales.config.SpringJavaConfigStyl
</beans>

```

Si nos queremos ahorrar la definición de los beans de configuración en el XML:

[view plain](#) [copy to clipboard](#) [print](#) [?](#)

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/bean
       http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-
context-3.0.xsd">

    <!-- Activa la búsqueda de anotaciones -->
    <context:annotation-config />

    <!--
    Le decimos donde buscar anotaciones de configuraci&ocute;n -->
    <context:component-scan base-
package="com.autentia.tutoriales.config"/>

</beans>

```

### JavaConfig "manda":

En este caso, debemos utilizar el mecanismo que hemos contado en los apartados anteriores (usando AnnotationConfigApplicationContext) y utilizar la anotación @ImportResource:

[view plain](#) [copy to clipboard](#) [print](#) [?](#)

```

01. @Configuration
02. @Import({DataSourceConfig.class})
03. @ImportResource("classpath:appContext.xml")
04. public class SpringJavaConfigStyle {
05.
06.
07.     @Autowired
08.     private DataSourceConfig dataSourceConfig;
09.
10.     @Bean
11.     public A beanA() {
12.         return new A(dataSourceConfig.dataSource());
13.     }
14.
15.     @Bean(name = { "beanB", "mrBean"})
16.     @Scope("prototype")
17.     public B beanB() {
18.         return new B(beanA());
19.     }
20. }

```

## Carga del contexto vía JavaConfig en aplicaciones Web

Spring suele pensar en todo, y tal como se hacía para cargar el contexto de Spring en aplicaciones Web con XML, tenemos la versión "JavaConfig manda" para cargar el contexto en aplicaciones Web. A continuación se muestra un ejemplo:

[view plain](#) [copy to clipboard](#) [print](#) ?

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

  <context-param>
    <param-name>contextClass</param-name>
    <param-value>
      org.springframework.web.context.support.AnnotationConfigWebApp1

    </param-value>
  </context-param>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-
value>com.autentia.tutoriales.config.SpringJavaConfigStyle</param-
value>
  </context-param>

  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
  </listener>

</web-app>
```

## Conclusiones:

La flexibilidad de la configuración del contexto de Spring queda sin ninguna duda completada con este modelo nuevo basado en Java. Pero lo mejor de todo es la posibilidad de combinar las tres maneras de configuración aportadas por Spring. Sabiendo las ventajas y los inconvenientes de cada una de ellas podemos elegir la que mejor se adecúe a cada proyecto y a cada caso en particular:

### Configuración XML

La configuración basada en XML es la más conveniente para configurar "beans" de infraestructura (DataSources, acceso a servicios...)

#### Ventajas:

- Configuración centralizada en uno o varios ficheros.
- Aplicable a todas las clases (nuestras y externas).
- De alguna manera, ya estamos familiarizados con esta forma de configuración.

#### Desventajas:

- Los ficheros se pueden volver algo engorrosos.
- Hay gente que "odia" el XML.

## Configuración por Anotaciones

La configuración basada en anotaciones es ideal para "beans" muy cambiantes (controladores...)

### Ventajas:

- Permite un desarrollo muy rápido.
- Todo está en la misma clase.

### Desventajas:

- Evidentemente sólo podemos aplicarlo sobre nuestras clases.
- Podemos perder de vista la configuración al estar distribuida por todo nuestro código.

## Configuración por JavaConfig

Este mecanismo proporciona un control total sobre la creación de Beans y configuración de los mismos.

### Ventajas:

- Mayor velocidad en la carga del contexto.
- Gran flexibilidad ya que puede ser usada en todas las clases.

### Desventajas:

- EL plugin de Spring (Spring Tool Suite) no puede representar la configuración de este tipo.

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

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

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

## COMENTARIOS



Esta obra está licenciada bajo licencia [Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

Copyright 2003-2010 © All Rights Reserved | [Inicio](#) | [Contacto](#) | [condiciones de uso](#) | [Banners](#) |

