

← Tutor de Programación



comparte tus conocimientos de programación y aprende enseñando

Elon Musk Deja Tesla - Últimas Noticias - Hace 16 Min

Tesla Inversores Están Molestos. ¿Qué Hará Elon Musk A Continuación? [financetipsnow.com/Elon Musk](http://financetipsnow.com/ElonMusk)

Spring Configuración de Beans



febrero 27, 2017

En el tutorial anterior aprendimos como configurar e instanciar el contenedor Spring (*ApplicationContext*) usando los distintos métodos de configuración disponibles, XML, Java y Anotaciones, ahora nos centraremos en la creación, configuración y uso de los *Beans*, en Spring se le llama bean a un objeto que es administrado por el contenedor IoC, es decir la creación, inicialización, destrucción, etc., del objeto será controlada por el contenedor IoC de Spring.

En este tutorial trabajamos con los ejemplos usados en el curso anterior por lo que recomiendo echarle un ojo antes de leer este, puedes encontrarlo en: [Spring Configurar Contenedor IoC](#).

Creación de Spring Beans

Todos los beans que deseemos añadir al contenedor deben tener un nombre, si no creamos uno Spring añadirá un nombre por nosotros, usaremos este nombre para obtener una referencia al objeto, además debemos indicar el tipo, este es el nombre completo de la clase usada para instanciar el objeto.

Usando XML el atributo **id** define el nombre y **class** el tipo, siguiendo con el ejemplo del tutorial anterior creamos el bean llamado *saludaService* de tipo `HelloServiceImpl`.

```
<bean id="saludaService"
      class="carmelo.spring.introduccion>HelloServiceImpl"/>
```

Usando la configuración Java el nombre del bean estará definido por el nombre del método, si deseamos establecer el nombre, como en el ejemplo, podemos hacerlo por medio de *name*.

```
@Bean(name="saludaService")
public>HelloService getSaludaService(){
```

← Tutor de Programación



Con el uso de anotaciones el nombre del bean será igual nombre de la clase con la letra inicial en minúscula, también podemos establecerlo con el atributo *value* de la anotación.

```
@Service(value="saludaService")
public class HelloServiceImpl implements HelloService {
    ...
}
```

Para verificar los nombres de los beans que hemos creado podemos listarlos con el método: `ctx.getBeanDefinitionNames()`, el siguiente código que mostramos muestra una lista de los nombres de beans presentes en el contenedor, debemos tener presente que Spring puede crear automáticamente algunos beans.

```
ApplicationContext ctx = new AnnotationConfigApplicationContext(SpringConfigurat

for (String bean_name : ctx.getBeanDefinitionNames()) {
    System.out.println(":: " + bean_name);
}
```

Podemos usar este código para saber con que nombre se han creado los beans.

Bean Scopes

El tiempo de vida de un bean creado por el contenedor Spring es llamado *bean scope*, por defecto todos los bean son creados con el scope *singleton*, en XML lo indicamos con el atributo `scope="..."` y en código Java con la anotación `@Scope("...")`, dentro de las comillas indicamos el nombre del scope, tenemos disponibles los siguientes scopes:

1) **singleton**: el scope por defecto del contenedor Spring, esto quiere decir que dentro del contenedor solo existirá una instancia del bean, siempre que llamemos al método `getBean` este nos devolverá una referencia a esa instancia.

2) **prototype**: este puede ser el inverso del scope anterior, cada vez que solicitemos un bean el contenedor nos devolverá una nueva instancia.

Los siguiente tres scopes son validos solo para aplicaciones web.

3) **request**: existirá una sola instancia del bean por cada petición HTTP, esto es, cada petición HTTP tiene su propia instancia de un bean.

4) **session**: existirá una instancia del bean por cada sesión HTTP.

5) **globalSession**: existirá una instancia del bean por cada sesión global HTTP. Típicamente solo es válida cuando se trabaja con portlets.

← Tutor de Programación



```
<bean id="saludaService"
      scope="prototype"
      class="carmelo.spring.introduccion.HelloServiceImpl"/>
```

Código que muestra como configurar el scope con Java

```
@Bean("saludaService")
@Scope("prototype")
public HelloService getSaludaService(){
    return new HelloServiceImpl();
}
```

Ambas configuraciones definen un bean con scope *prototype*, la anotación `@Scope` la podemos utilizar del mismo modo con la configuración por anotaciones.

Creación y Destrucción de Beans

Spring nos permite intervenir en el ciclo de vida de los beans, mas específicamente en la inicialización y destrucción de los mismos, contamos con distintas maneras para hacerlo, veamos cada una de ellas.

En la configuración XML tenemos los atributos *init-method* y *destroy-method* mediante los cuales podemos indicar los métodos de inicialización y destrucción respectivamente.

```
<bean id="saludaService"
      init-method="initHello"
      destroy-method="destroyHello"
      class="carmelo.spring.introduccion.HelloServiceImpl"/>
```

Hemos modificado la clase `HelloServiceImpl` para agregar los métodos `initHello` y `destroyHello`, la clase se ve de la siguiente manera:

```
public class HelloServiceImpl implements HelloService
{
    ...

    public void initHello() {
        System.out.println("--- run init method ---");
    }

    public void destroyHello() {
        System.out.println("--- run destroy method ---");
    }
}
```

← Tutor de Programación



respectivamente, veamos un ejemplo.

```
public class HelloServiceImpl implements HelloService {
    ...

    @PostConstruct
    public void initHello() {
        System.out.println("--- run init method ---");
    }

    @PreDestroy
    public void destroyHello() {
        System.out.println("--- run destroy method ---");
    }
}
```

La última manera de hacerlo es mediante las interfaces `InitializingBean` y `DisposableBean` para la primera declara el método `afterPropertiesSet()` y la segunda `destroy()`, cada uno de ellos para la inicialización y destrucción respectivamente.

```
public class HelloServiceImpl implements HelloService, InitializingBean, DisposableBean {
    ...

    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("--- run init method ---");
    }

    @Override
    public void destroy() throws Exception {
        System.out.println("--- run destroy method ---");
    }
}
```

Importante: los beans con *scope prototype* no son notificados de la destrucción de los objetos, es decir el método de destrucción no se ejecutará los ejemplos anteriores si el scope es igual a prototype.

Crear Beans con Métodos Factory

Los beans que hemos creado hasta ahora han sido instanciados mediante su constructor, pero existe otra forma, podemos establecer un método *factory* para esta tarea, como su nombre lo dice un *factory* o *fábrica* es el encargado de generar objetos de un determinado tipo.

```
public class ClientService {
```

← Tutor de Programación



```
public static ClientService createInstance() {  
    return clientService;  
}  
}
```

Con esta clase buscamos que el método `createInstance()` sea el encargado de devolvernos una instancia estática de `ClientService`, de este modo nos aseguramos que trabajamos siempre con la misma instancia.

```
<bean id="one"  
      class="carmelo.spring.introduccion.ClientService"  
      factory-method="createInstance"/>
```

En la configuración XML de Spring indicamos el método *factory* con el atributo `factory-method`.

De manera similar es posible usar el método *factory* de un bean en lugar de utilizar un método estático como en el ejemplo previo.

```
public class DefaultServiceLocator {  
  
    private static ClientService clientService = new ClientServiceImpl();  
  
    private DefaultServiceLocator() {  
    }  
  
    public ClientService createClientServiceInstance() {  
        return clientService;  
    }  
}
```

Primero creamos un bean de esta clase, este contendrá el método *factory*, como vemos este no es estático por lo cual debe ser invocado desde la instancia, para ello lo hacemos del siguiente modo:

```
<!-- the factory bean, which contains a method called createInstance() -->  
<bean id="serviceLocator" class="examples.DefaultServiceLocator" />  
  
<!-- the bean to be created via the factory bean -->  
<bean id="clientService"  
      factory-bean="serviceLocator"  
      factory-method="createClientServiceInstance"/>
```

← Tutor de Programación



Código en GitHub: [Configurar los Spring Beans](#)

Compartir: 

Categorías: Spring

Make pipetting more effective

Solve the most labour intensive daily task by freeing your time for important work
[andrewalliance.com](#)

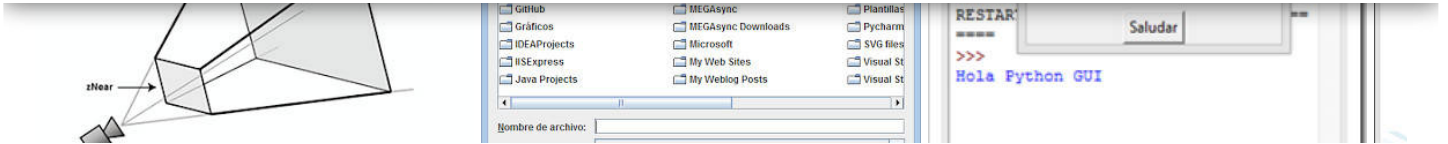


Introduce tu comentario...

Redes sociales

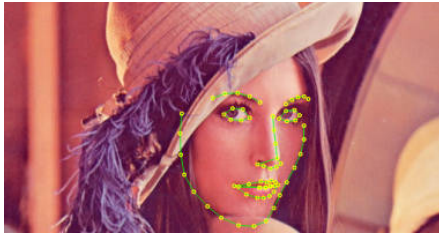
Temas relacionados

← Tutor de Programación



Visualización 3D usando OpenGL JFileChooser Java Swing Python GUI Botones

**Sp
In**



**Face Landmarks
Detector con Dlib y
OpenCV**



OpenCV Detectar Líneas



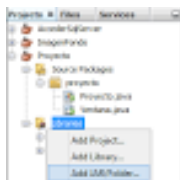
**Spring Acceso a datos
con JDBC**

Sp

Entradas populares de este blog

Conectar SQL Server con Java

mayo 20, 2017

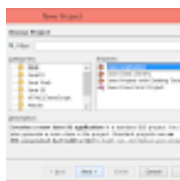


Conectar Java con SQL Server: una vez hayamos creado nuestra base de datos, seguramente necesitamos conectarla con una aplicación que realizaremos en un lenguaje de nuestra preferencia, si eleg...

SIGUE LEYENDO

Conectar a base de datos MySQL con Java

agosto 17, 2017



En todo proceso de desarrollo de una aplicación se requiere usualmente *conectar a base de datos MySQL*, es el servidor de datos que utilizaremos en este caso será MySQL aunque el mismo puede variar, p...

SIGUE LEYENDO

Entrenar OpenCV en Detección de Objetos

diciembre 06, 2015

← Tutor de Programación



OpenCV (una de las librerías de visión por computadora más en

[SIGUE LEYENDO](#)

Detección de contornos con OpenCV Python

agosto 31, 2017



Un contorno es un conjunto de puntos que conectados unos con otros de manera consecutiva forman una figura que rodea un objeto determinado, la detección de contornos en OpenCV se apli ...

[SIGUE LEYENDO](#)

Manipular pixeles OpenCV Python

junio 17, 2017



En este tutorial veremos como aplicar las operaciones básicas sobre imágenes cargadas en memoria mediante OpenCV, veremos cómo acceder a los pixeles de una imagen y modificarlos, aprend ...

[SIGUE LEYENDO](#)



Con la tecnología de Blogger

Imágenes del tema: [Michael Elkan](#)

Tutor de Programación © 2017 | @TrProgramación



Tutor de Progr...
65 Me gusta

Me gusta esta página


Sé el primero de tus amigos en

← Tutor de Programación



Tutor de Programación

google.com/+AcodigoBlogspot-
Tutordeprogramacion

 Seguir

Archivo ▼

Etiquetas ▼

[Denunciar uso inadecuado](#)