Cómo convertir controladores síncronos en asíncronos utilizando Spring 3.2 (Servlets 3.0)

*Germán Ramos García (*[*german.ramos.garcia@bbva.com*](mailto:german.ramos.garcia@bbva.com)*) 28/05/2013*

# Introducción

Desde la versión 3.2. Spring Framework proporciona soporte para la especificación Java Servlets 3.0. Esta especificación permite el desarrollo de servlets asíncronos en Java. Es decir, no es necesario un hilo para cada petición, sino que muchas peticiones pueden ser atendidas con uno o unos pocos hilos.

Como resultado, un servidor web implementado con servlets asíncronos es capaz de gestionar muchas más conexiones simultáneas con menor consumo de recursos del sistema. Además, es posible combinar servicios síncronos (tradicionales) y asíncronos en la misma aplicación de modo que se podrían realizar migraciones parciales de programas ya existentes.

El objetivo de este documento es explicar detalladamente como realizar la migración de un servicio síncrono a asíncrono con Spring Framework 3.2.

Hay 2 maneras de conseguir el objetivo. La primera manera delega la ejecución asíncrona a Spring, y la otra manera permite al programador tener el control total de la ejecución de cada petición. Primero, explicaré la primera opción y detallaré uno a uno los pasos a seguir.

# Ejemplo síncrono

Supongamos el siguiente ejemplo de controlador en Spring:

@Controller

public class AsyncController {

@RequestMapping(value = "/hello", method = RequestMethod.GET)

@ResponseBody

public String hello() {

//Begin of service code

return "hi";

//End of service code

}

}

Como podemos observar, es un ejemplo muy sencillo de un servicio en la ruta “/hello” que responde “hi”.

El código del servicio se encuentra entre los comentarios “Begin of service code” y “End of service code”

Doy por supuesto que el código anterior lo entiende cualquier programador de Spring MVC y a continuación voy a explicar como convertir ese mismo servicio en asíncrono.

# Ejemplo asíncrono manejado por Spring

## 1. Transformar el valor de retorno a tipo “Callable”

@Controller

public class AsyncController {

@RequestMapping(value = "/hello", method = RequestMethod.GET)

@ResponseBody

public Callable<String> hello() {

return new Callable<String>() {

public String call() throws Exception {

//Begin of service code

return "hi";

//End of service code

}

};

}

}

En lugar de devolver un String con una vista o con el response body correspondiente, devolvemos un objeto Callable. Este objeto tiene un método call() que será invocado posteriormente de manera asíncrona y producirá la respuesta a la petición.

## 2. Configurar el TaskExecutor

Cuando devolvemos un objeto “Callable”. Spring se encarga de encolarlo y ejecutarlo de manera asíncrona utilizando un “TaskExecutor”. Spring crea por defecto un “SimpleAsyncTaskExecutor” que se encarga de ello. No obstante, “SimpleAsyncTaskExecutor” es muy ineficiente y sólo apto para entornos de desarrollo. En un entorno de producción lo normal es usar “ThreadPoolTaskExecutor” o “ConcurrentTaskExecutor”. Podemos encontrar documentación sobre todos los TaskExecutor disponibles en [2]

Vamos a configurar un “ThreadPoolTaskExecutor” en el fichero servlet-context.xml:

<annotation-driven>

<async-support task-executor="myTaskExecutor" default-timeout="5000" />

</annotation-driven>

<beans:bean id="myTaskExecutor" class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">

<beans:property name="corePoolSize" value="10" />

<beans:property name="maxPoolSize" value="50" />

<beans:property name="queueCapacity" value="5000" />

</beans:bean>

Hemos definido un TaskExecutor llamado “myTaskExecutor” y establecido un timeout de 5 segundos para cada conexión asíncrona. “myTaskExecutor” se utilizará para la ejecución de todos los objetos “Callable” devueltos por el controlador. Los parámetros de configuración “corePoolSize”, “maxPoolSize” y “queueCapacity” tienen nombres autoexplicativos. Con esta configuración se utilizan entre 10 y 50 hilos con una cola de 5000 peticiones.

## 3. Configurar web.xml

El último paso es habilitar el soporte asíncrono en el servidor de aplicaciones. Para ello debemos añadir la siguiente configuración en la sección servlet del fichero “web.xml”

<servlet>

...

**<async-supported>true</async-supported>**

</servlet>

Con esto hemos terminado de convertir “/hello” a un servicio totalmente asíncrono. Además, si queremos convertir más servicios ya no tendremos que repetir los pasos 2 y 3.

# Ejemplo asíncrono manejado independientemente

Es posible que queramos tener servicios asíncronos y no gestionarlos a través de un TaskExecutor, sino utilizar alguna librería externa o bien responder bajo ciertos criterios. Esto nos permite realizar servicios que antes no se podían realizar. Por ejemplo, voy a implementar el servicio anterior de modo que las peticiones a “/hello” se queden suspendidas hasta que el servidor reciba una petición “/sayhello”. En ese momento, responderá a todas las peticiones “/hello” suspendidas.

@Controller

public class AsyncController{

private List<DeferredResult<String>> reqs = new ArrayList<DeferredResult<String>>();

@RequestMapping(value = "/hello", method = RequestMethod.GET)

@ResponseBody

public DeferredResult<String> hello() {

DeferredResult<String> result = new DeferredResult<String>(0);

synchronized(reqs) {

reqs.add(result);

}

return result;

}

@RequestMapping(value = "/sayhello", method = RequestMethod.GET)

@ResponseBody

public String sayhello() {

while (!reqs.isEmpty()) {

//Begin of service code

reqs.get(0).setResult(“hi”);

//End of service code

reqs.remove(0);

}

return "Done";

}

}

Si nos fijamos, el servicio “/hello” crea un objeto “DeferredResult” que almacena en un ArrayList y lo devuelve como resultado. Esto indica a Spring que la respuesta a esa petición ha sido diferida. Para este ejemplo conviene aumentar el valor del parámetro “default-timeout”.

Cuando se invoca al servicio “/sayhello” se recorre la lista de “DeferredResult” y se ejecutan el código de servicio deseado, en este caso simplemente responder “hi”. Esto se hace llamando al método “setResult” de los “DeferredResult” guardados.

Si nos fijamos, hemos mezclado un servicio asíncrono “/hello” con otro servicio síncrono “/sayhello”. Además, podríamos haber usado alguna librería externa para la gestión de eventos como por ejemplo Reactor o bien reaccionar a un mensaje JMS o AMQP, una notificación de Redis, etc.

# Conclusiones

La incorporación del soporte de Java Servlets 3.0 a Spring 3.2 posibilita la gestión de muchas más conexiones simultáneas y favorece el consumo eficiente de recursos del sistema[8].

Además, abre un abanico de posibilidades a Spring Framework para abordar problemas que antes estaban fuera de su alcance: Long Poolong, Server Sent Events de HTML5, servidores masivos, etc.

Es importante mencionar que la implementación de la especificación de Servlets 3.0 es relativamente nueva en los servidores de aplicaciones más populares. En pruebas realizadas:

* Tomcat 7 funciona correctamente pero consume demasiada memoria. Debido, probablemente, a una implementación síncrona del dispacher.
* Jetty 9.0.3 (la última en la fecha de este artículo) tiene un bug conocido en la implementación de “DeferredResult” y como resultado siempre devuelve vacío a las peticiones. Bug 408117. En la versión 9.0.4 debería estar corregido.
* Jetty 8.1 funciona perfectamente y es capaz de gestionar 48.000 conexiones simultáneas con un consumo de memoria cercano a 1 Gb [8].

# Referencias

[1]<http://blog.springsource.org/2012/05/06/spring-mvc-3-2-preview-introducing-servlet-3-async-support/>

[2]<http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/scheduling.html>

[3]<http://www.mkyong.com/spring/spring-and-java-thread-example/>

[4]<http://stackoverflow.com/questions/141284/the-difference-between-the-runnable-and-callable-interfaces-in-java>

[5]<http://notpurelytechnical.com/spring-3-2-rundown-async-support/>

[6]<http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-async>

[7]<http://stackoverflow.com/questions/13706050/spring-mvc-how-to-use-asynctaskexecutor-from-controller>

[8]<https://docs.google.com/a/bbva.com/document/d/1mj0NTsqpucZxIC-tDecIKwjETxQ99QpHi_Tv2Qy3ZG4/edit?usp=sharing>