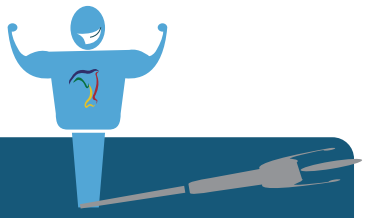
autentia

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...

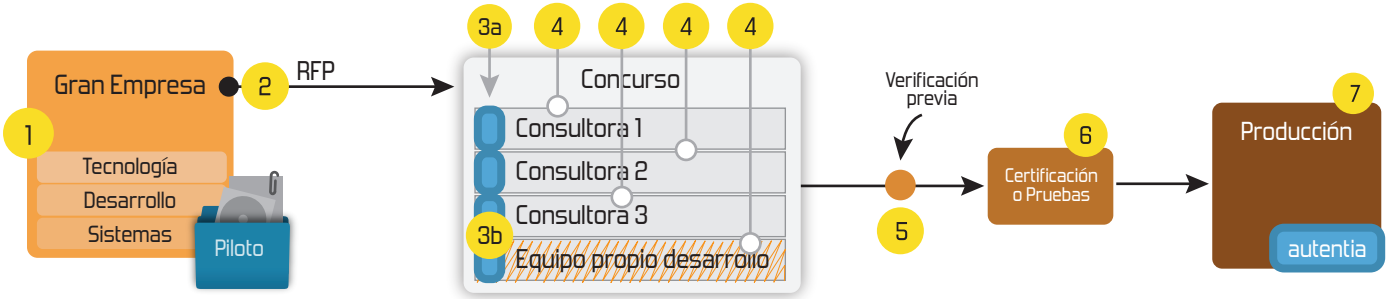
1. Desarrollo de componentes y proyectos a medida



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

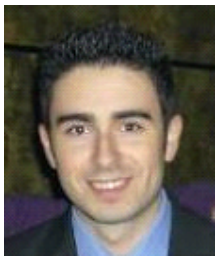
Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

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

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

AdictosAlTrabajo
Terrakas 1x03 ¡¡Ya está en la web!! terrakas.com
autentia Soporte a desarrollo informático
Hosting patrocinado por enredados
Entra er
E-mail
Contras
Entr



Alberto Barranco Ramón

Consultor tecnológico de desarrollo de proyectos informáticos.

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

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Slow Mac?
Make it >25% faster with MacKeeper
Download

Fecha de publicación del tutorial: 2012-03-27 Tutorial visitado 4 veces Descargar en PDF

Spring mvc. Servicios Rest respondiendo en Json o XML

0. Índice de contenidos.

- 1. Introducción
- 2. Entorno
- 3. Configurando los servicios Rest con Spring mvc
- 4. Probando los distintos tipos de respuesta
- 5. Conclusiones
- 6. Información sobre el autor

1. Introducción

En este tutorial vamos a ver las configuraciones necesarias para que un servicio Rest, bajo Spring mvc devuelva XML o Json en función del accept de la cabecera de la petición.

Para devolver XML o Json tenemos varias opciones pero la más elegante me parece tener un solo servicio Rest mapeado en una sola URL, y que en función de lo que quiera el cliente (atributo accept de la cabecera de la petición) se sirva una cosa u otra.

2. Entorno

- Hardware: Portátil MacBook Pro 15' (2.0 GHz Intel i7, 8GB DDR3 SDRAM, 500GB HDD).
- AMD Radeon HD 6490M 256 MB
- Sistema Operativo: Mac OS X Snow Leopard 10.6.7
- Software relacionado con el tutorial: Eclipse Helios, Spring 3.1.1.RELEASE, Apache Tomcat 7

3. Configurando los servicios Rest con Spring mvc

Lo primero de todo es configurar el **web.xml**.

```
view plain print ?
01. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
02.     xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
```

Cat
Aut
Síg
Últi
» Rc teor
» Di lo qu
» Sc ¿imp
» Pr todc
» Ve de e
Hist
Últi
» Tr

```
app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">


    <display-name>rest</display-name>

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

    <context-param>
        <description>Spring configuration file</description>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath:applicationContext-model.xml,
            WEB-INF/applicationContext-servlet.xml
        </param-value>
    </context-param>

    <context-param>
        <param-name>webAppRootKey</param-name>
        <param-value>rest</param-value>
    </context-param>

    <servlet>
        <servlet-name>applicationContext</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>applicationContext</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
```

- En la línea 4 estamos declarando el listener de Spring responsable de levantarnos nuestros contextos.
- En las líneas 7-14 definimos los applicationContext que queremos levantar. Luego los veremos más en profundidad.
- En las líneas 21-30 declaramos en Servlet de forma estándar. Según la declaración que hacemos la configuración del Servlet se espera en un archivo con el nombre "applicationContext-servlet.xml", es decir, nombre del Servlet seguido de -servlet.xml (estándar). Lo estamos mapeando contra /rest/*, es decir todas las llamadas del tipo localhost:8080/rest/rest/* serán tratadas por este Servlet.

Vamos a ver ahora la configuración del Servlet, que la tenemos como ya hemos dicho en **applicationContext-servlet.xml**

```
view plain print ?
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLScher
instance"
xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.or
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spri
context.xsd
    http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd">

    <mvc:annotation-driven/>

    <context:component-scan base-package="com.autentia.tutoriales.rest.*" />

</beans>
```

¿Sencillo verdad? Aún así siempre me gusta explicarlo todo para que quede clarito y por si alguien tiene alguna duda.

- En la línea 10 indicamos a Spring que vamos a utilizar anotaciones, y que queremos que nos instancie las clases que tengamos anotadas, por ejemplo, con **@Controller**.
- En la línea 12 le decimos a partir de qué paquete tiene que escanear en busca de clases anotadas.

A continuación vamos a crearnos un controlador con el soporte de Spring mvc para que gestione las peticiones. Lo vemos a continuación.

```
view plain print ?
package com.autentia.tutoriales.rest.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
05. import org.springframework.web.bind.annotation.RequestMethod;
06. import org.springframework.web.bind.annotation.ResponseBody;
07. 
08. import com.autentia.tutoriales.entities.Person;
09. 
10. @Controller
11. @RequestMapping("/person")
12. public class RestController {
13. 
14. 
15.     @RequestMapping(value = "/getPerson/", method = RequestMethod.GET)
16.     @ResponseBody
17.     public Person getPerson() {
18. 
19.     return new Person("Respuesta en formato dependiente del header de la petición");
20. 
21.     }
22. }
```

Como veis hemos declarado un controlador al que le llegarán las peticiones del tipo **http://localhost:8080/rest/rest/person/**. Aquí podemos meter una serie de métodos relacionadas con la gestión de personal que normalmente llamarán a servicios que tengamos alojados en nuestro modelo y devolverán una respuesta.

Si lo que queremos es una respuesta Json o XML dependiente de la cabecera de la petición no podemos utilizar el atributo **"produces"** de la anotación **@RequestMapping** para indicar que este método devuelve **application/json** ó **application/xml**. Lo que tenemos que hacer es lo siguiente:

- 1º. Insertamos la siguiente dependencia en el pom de nuestro módulo web donde tenemos definido el controlador. Es necesaria para poder ofrecer la respuesta en Json.

```
view plain print ?
01. <dependency>
02.     <groupId>org.codehaus.jackson</groupId>
03.     <artifactId>jackson-mapper-asl</artifactId>
04.     <version>1.6.4</version>
05. </dependency>
```

- 2º. En nuestra entidad Person, tendremos que definir los elementos del XML que queramos que aparezan. Lo vemos a continuación.

```
view plain print ?
01. package com.autentia.tutoriales.entities;
02. 
03. import javax.persistence.Entity;
04. import javax.persistence.GeneratedValue;
05. import javax.persistence.Id;
06. import javax.xml.bind.annotation.XmlElement;
07. import javax.xml.bind.annotation.XmlRootElement;
08. 
09. @XmlRootElement
10. @Entity
11. public class Person {
12. 
13.     @Id
14.     @GeneratedValue
15.     private int id;
16. 
17.     @XmlElement
18.     private final String name;
19. 
20.     public Person(String name) {
21.         this.name = name;
22.     }
23. 
24.     public String getName() {
25.         return name;
26.     }
27. 
28.     public int getId() {
29.         return id;
30.     }
31. }
```

Como podeis ver hemos anotado la clase como raíz del XML con **@XmlRootElement** y los atributos que queremos que aparezan en el XML los anotamos con **@XmlElement**

Esta clase la tengo en otro módulo que es el que he definido como model en el proyecto. Si os acordais en el **web.xml** definiamos el applicationContext-model que corresponde precisamente a este módulo. Os lo pego a continuación para tener todas las piezas del puzzle.

```
view plain print ?
01. <beans xmlns="http://www.springframework.org/schema/beans"
02.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-
```

```
instance" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.0.xsd">

        <context:annotation-config />

        <!-- | Packages to scan, looking for @Entity annotation -->
        <bean id="entitiesPackagesToScan" class="java.util.ArrayList">
            <constructor-arg index="0">
                <list>
                    <value>com.autentia.tutoriales.entities</value>
                </list>
            </constructor-arg>
        </bean>

</beans>
```

4. Probando los distintos tipos de respuesta

Nos vamos a crear un cliente sencillito en HTML que haga 2 peticiones al servidor, una que tenga en el accept de la cabecera "application/json" y la otra "application/xml". El código fuente del cliente os lo pongo a continuación:

view plain print ?

```
<html>
    <head>
        <script type="text/javascript" src="/rest/js/jquery-1.6.1.min.js"></script>
        <script type="text/javascript" src="/rest/js/index.js"></script>
    </head>

    <body>

        TUTORIAL SERVICIOS REST JSON/XML
        <p>
            <a href="#" onclick="jsonPerson()"><strong>JSON</strong></a>
        </p>

        <p>
            <a href="http://localhost:8080/rest/rest/person/getPerson/"><strong>XML</strong></a>
        </p>

    </body>
</html>
```

Estamos incluyendo 2 javascripts, el primero es jquery, que os lo podeis bajar de :http://code.jquery.com/jquery-1.7.2.min.js. El segundo es un javascript muy cortito cuyo código os pongo a continuación :

view plain print ?

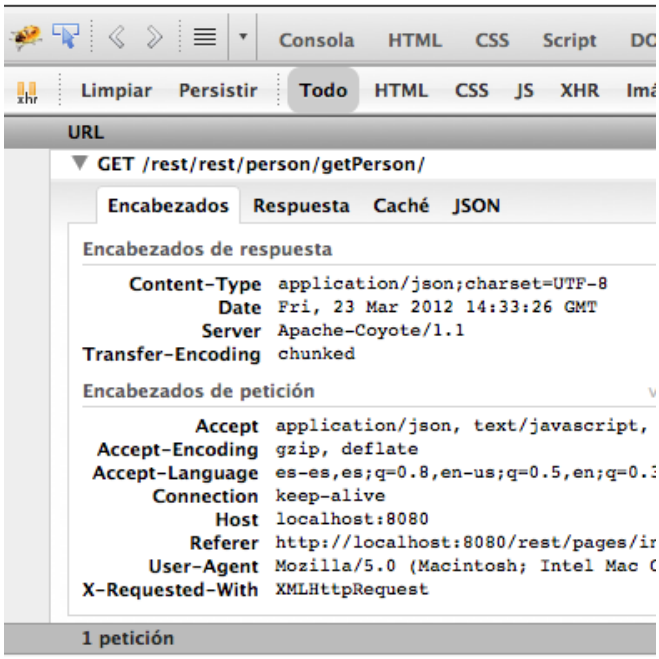
```
function jsonPerson() {
    var url = "http://localhost:8080/rest/rest/person/getPerson/";
    $.getJSON(url,null);
}
```

Si pinchamos en JSON, podemos ver en los detalles de la petición el Accept.

TUTORIAL SERVICIOS REST JSON/XML

JSON

XML



Si ahora nos vamos a la respuesta tendremos lo siguiente: *{"name":"Respuesta en formato dependiente del header de la petición","id":0}* que es el Json que representa al objeto que hemos creado.

Si repetimos el proceso pulsando en XML en la respuesta nos vendrá lo siguiente: <?xml version="1.0" encoding="UTF-8" standalone="yes"?> <person> <name>Respuesta en formato dependiente del header de la petición</name> </person>

Por último comentar que si quisieramos forzar un formato en concreto podríamos anotar el método de la siguiente forma: **@RequestMapping(value = "/getPerson/", method = RequestMethod.GET, produces="application/json")**. Esto devolvería siempre json.

5. Conclusiones

Me parece muy útil tener un servicio que responda en distintos formatos según lo que acepte el cliente y a nivel de código queda muy limpio al tenerlo en un solo método. Espero que os haya sido útil el tutorial. Un saludo.

6. Información sobre el autor

Alberto Barranco Ramón es Ingeniero Técnico en Informática de Gestión y Graduado en Ingeniería del Software por la Universidad Politécnica de Madrid

Mail: abarranco@autentia.com.

Twitter: @barrancoalberto

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo".

A continuación puedes evaluarlo:

Regístrate para evaluarlo

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

Share | 0

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

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

Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

PUSH THIS Page Pushers Community Help?

---- no clicks

0 people brought clicks to this page

powered by karmacracy

Copyright 2003-2012 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia |