**Universidad Científica del Sur**

**Ing. Sistemas e Informática y Gestión**

**Trabajo De Investigación**

**Nombre:**

Josecarlos Paredes Requejo

Arturo Fernando Jurado Veliz

**Curso:**

Taller de Programación II

**Profesor:**

Eric G. Coronel Castillo

**Lima - 2016**

**Descripción**

Este documento es una guía paso a paso sobre cómo desarrollar una aplicación web, partiendo de cero, usando Spring Framework.

Se asume que posees un conocimiento superficial de Spring, por lo que este tutorial te será útil si estas aprendiendo o investigando Spring. Durante el tiempo que trabajes a través del material presentado en el tutorial, podrás ver como encajan diversas partes de Spring Framework dentro de una aplicación web Spring MVC, como Inversión de Control (Inversión of Control - IoC), Programación Orientada a Aspectos (Aspect-Oriented Programming - AOP), así como las diversas librerías de servicios (como la librería JDBC).

Spring provee diversas opciones para configurar tu aplicación. La forma más popular es usando archivos XML. Esta es la forma más tradicional, soportada desde la primera versión de Spring. Con la introducción de Anotaciones en Java 5, ahora disponemos de una manera alternativa de configurar nuestras aplicaciones Spring. La nueva versión Spring 2.5 introduce un amplio soporte para configurar una aplicación web mediante anotaciones.

Este documento usa el estilo tradicional de configuración, mediante XML. Estamos trabajando en una *"Edición con Anotaciones"* de este documento, y esperamos publicarla en un futuro cercano.

Ten en cuenta que no se tratara ninguna información en profundidad en este tutorial, así como ningún tipo de teoría; hay multitud de libros disponibles que cubren Spring en profundidad; siempre que una nueva clase o característica sea usada en el tutorial, se mostraran enlaces a la sección de documentación de Spring, donde la clase o característica es tratada en profundidad.

**¿En qué consiste Spring MVC?**

Spring Mvc es una alternativa de framework basado en el patrón modelo-vista-controlador, después de haber aprendido de errores de frameowrks anteriores a estos como Jakarta Struts y otras alternativas.

Particulamente, el equipo Spring siente que no hay suficiente separación entre las capas de presentación y la capa de manejo de request, y entre la capa de manejo de request y el modelo.

El framework tiene un conjunto de interfaces que después se implementan para proporcionar la funcionalidad correspondiente. Las interfaces están acopladas claramente al Servlet Api.  
La clase DispatcherServlet está en el front controller y es responsable de delegar y coordinar el control entre varias interfaces en la fase de ejecución durante una petición Http.  
Las interfaces más importantes definidas en Spring Mvc, y sus responsabilidades, son las siguientes:

* HandlerMapping: permite manejar peticiones de entrada.
* HandlerAdapter: ejecución de objetos que permiten manejar las peticiones entrantes.
* Controller: está entre el modelo y la vista, y permite manejar peticiones entrantes y redirigirlas a la respuesta adecuada.
* Vista: responsable de retornar una respuesta al cliente.
* ViewResolver: selecciona una vista basada en un nombre lógico de la vista.
* HandlerInterceptor: intercepta las peticiones entrantes , es comparable pero no igual a los filtros de Servlet.
* LocaleResolver: resuelve y opcionalmente salva el locale de un usuario individual.
* MultipartResolver: facilita trabajar con ficheros de subida wrapping peticiones de entrada.

**Contenido**

La siguiente lista detalla todas las partes de Spring Framework que son cubiertas a lo largo del tutorial.

Inversión de Control (IoC)

El framework Spring Web MVC

Acceso a Datos mediante JDBC

Integración mediante tests

Manejo de transacciones

Software Requerido

Se requiere el siguiente software y su adecuada configuración en el entorno. Deberías sentirte razonablemente confortable usando las siguientes tecnologías.

Java SDK 1.5

Ant 1.7

Apache Tomcat 6.0.14

Eclipse 3.3 (Recomendado, pero no necesario)

*Eclipse 3.3 Europa (http://www.eclipse.org/europa) junto con el proyecto Web Tools Platform (WTP) (http://www.eclipse.org/webtools) y el proyecto Spring IDE (http://www.springide.org) proporcionan un excelente entorno para el desarrollo web.* *Por supuesto, puedes usar cualquier variación de las versiones de software anteriores. Si quieres usar NetBeans o IntelliJ en lugar de Eclipse, o Jetty en lugar de Tomcat, ciertos pasos de este tutorial no se corresponderán directamente con tu entorno, pero deberías ser capaz de seguir adelante de todas maneras.*

**La aplicación que vamos a construir**

La aplicación que vamos a construir desde cero a lo largo de este tutorial es un sistema de mantenimiento de inventario *muy básico*. Este sistema de mantenimiento de inventario está muy limitado en alcance y funcionalidad; debajo puedes ver un diagrama de casos de uso ilustrando los sencillos casos de uso que implementaremos. La razón por la que la aplicación es tan limitada es que puedas concentrarte en las características específicas de Spring Web MCV y Spring, y olvidar los detalles más sutiles del mantenimiento de inventario.

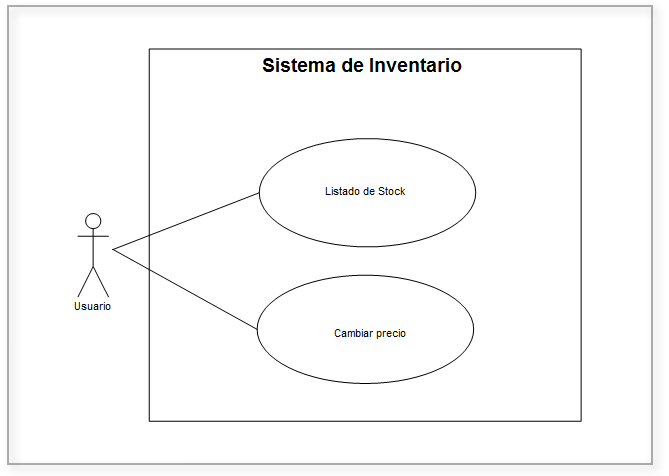


Diagrama de casos de uso de un sistema de mantenimiento de inventario

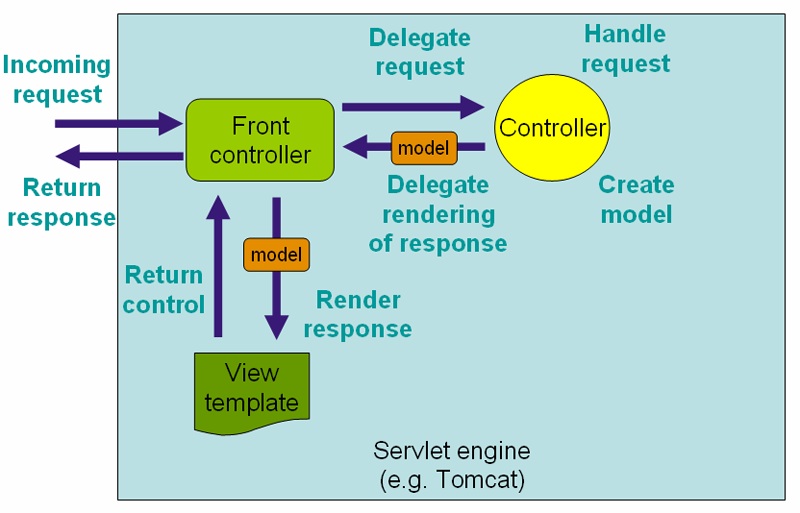
[Comenzaremos](http://www.davidmarco.es/spring-mvc/part1.html) configurando la estructura básica de los directorios para nuestra aplicación, descargando las librerías necesarias, configurando nuestros scripts Ant, etc. El primer paso nos proporcionara una base sólida sobre la que desarrollar de forma adecuada las secciones [2](http://www.davidmarco.es/spring-mvc/part2.html), [3](http://www.davidmarco.es/spring-mvc/part3.html), y [4](http://www.davidmarco.es/spring-mvc/part4.html).

Una vez terminada la configuración básica, introduciremos Spring, comenzando con el framework Spring Web MVC. Usaremos Spring Web MVC para mostrar el stock inventariado, el cual implicara escribir algunas clases simples en Java y algunos JSP. Entonces introduciremos acceso de datos y persistencia en nuestra aplicación, usando para ello el soporte que ofrece Spring para JDBC.

Una vez hayamos finalizado todos los pasos del tutorial, tendremos una aplicación que realiza un mantenimiento básico de stock, incluyendo listados de stock y permitiendo el incremento de precios de dicho stock.

**Procesamiento de una petición en Spring MVC**

A continuación se describe el flujo de procesamiento típico para una petición HTTP en Spring MVC. Esta explicación está simplificada y no tiene en cuenta ciertos elementos que comentaremos posteriormente. Spring es una implementación del patrón de diseño "front controller", que también implementan otros frameworks MVC, como por ejemplo, el clásico Struts.



Todas las peticiones HTTP se canalizan a través del *front controller*. En casi todos los frameworks MVC que siguen este patrón, el *front controller* no es más que un servlet cuya implementación es propia del framework. En el caso de Spring, la clase DispatcherServlet.

El *front controller* averigua, normalmente a partir de la URL, a qué Controller hay que llamar para servir la petición. Para esto se usa un HandlerMapping.

Se llama al Controller, que ejecuta la lógica de negocio, obtiene los resultados y los devuelve al servlet, encapsulados en un objeto del tipo Model. Además se devolverá el nombre lógico de la vista a mostrar (normalmente devolviendo un String, como en JSF).

Un ViewResolver se encarga de averiguar el nombre físico de la vista que se corresponde con el nombre lógico del paso anterior.

Finalmente, el *front controller* (el DispatcherServlet) redirige la petición hacia la vista, que muestra los resultados de la operación realizada.

En realidad, el procesamiento es más complejo. Nos hemos saltado algunos pasos en aras de una mayor claridad. Por ejemplo, en Spring se pueden usar interceptores, que son como los filtros del API de servlets, pero adaptados a Spring MVC. Estos interceptores pueden pre y postprocesar la petición alrededor de la ejecución del Controller. No obstante, todas estas cuestiones deben quedar por fuerza fuera de una breve introducción a Spring MVC como la de estas páginas.

**Configuración básica**

Lo habitual es que se configure la aplicación web de manera que todas las peticiones cuya URL sigue un determinado patrón pasen a través de Spring MVC. Así, por ejemplo, podemos "redirigir" a través de Spring MVC todas las peticiones que acaben en ".do". Esto nos permite por ejemplo servir los recursos estáticos aparte, ya que no es necesario que estos pasen por el flujo de ejecución MVC.

Necesitaremos configurar el web.xml para que todas las peticiones HTTP con un determinado patrón se canalicen a través del mismo servlet, en este caso de la clase DispatcherServlet de Spring. Como mínimo necesitaremos incluir algo como esto:

|  |  |
| --- | --- |
|  | <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>  org.springframework.web.servlet.DispatcherServlet  </servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping> |

Con esta configuración, todas las peticiones acabadas en .do, como getPedido.do o verClientes.do, se redirigirían al servlet de Spring.

Para aumentar la modularidad de la aplicación, se pueden configurar la capa web en un XML distinto al resto de la aplicación. Al arrancar, Spring buscará automáticamente un fichero con el mismo nombre del servlet que hace de dispatcher, seguido de la terminación -servlet.xml. La búsqueda se hace en el directorio WEB-INF. En nuestro caso, el fichero buscado automáticamente sería dispatcher-servlet.xml.

Por tanto, la forma habitual de trabajar es usar un XML para los beans de la capa web y otro (u otros) distinto para los de la capa de negocio y DAOs. Spring establece una jerarquía de contextos de modo que en el XML de la capa web se heredan automáticamente los otros beans, lo que nos permite referenciar los objetos de negocio en nuestro código MVC.

**¡Cuidado con las anotaciones!**

Recordad que hay que configurar la etiqueta component-scan para que Spring examine ciertos packages en busca de anotaciones. Si usamos dos XML, uno para la capa web y otro para las demás capas, cada XML debe tener su propio component-scan. Aunque por nuestra organización de código el component-scan de las capas de negocio y datos "cubra" también a las clases de web, necesitamos ponerlo explícitamente en el dispatcher-servlet.xml

Suponiendo que nuestros componentes web están implementados en el paquete es.ua.jtech.spring.mvc, en el dispatcher-servlet.xml aparecería:

|  |  |
| --- | --- |
|  | <?xml versión="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>"         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/context>  <http://www.springframework.org/schema/context/spring-context.xsd>  <http://www.springframework.org/schema/mvc>  <http://www.springframework.org/schema/mvc/spring-mvc.xsd>">      ...         <context:component-scan base-package="es.ua.jtech.spring.mvc"/>         <mvc:annotation-driven/>    ...  </beans> |

La etiqueta <mvc:annotation-driven/> permite usar anotaciones y hace una configuración por defecto de distintos elementos que iremos viendo, como la validación y conversión de datos. Está disponible a partir de la versión 3.0 de Spring

**Caso 1: petición sin procesamiento de datos de entrada**

La elaborada arquitectura de Spring MVC, y las muchas posibilidades que tiene el usuario de configurar a su medida el procesamiento que hace el framework hacen que sea poco intuitivo hacer una descripción general de Spring MVC, al menos si no se dispone del suficiente tiempo para hacerlo de manera pausada, lo que no es el caso. En su lugar, hemos preferido aquí describir cómo se implementarían un par de casos típicos en una aplicación web, indicando cómo implementar cada caso y las posibilidades adicionales que ofrece Spring MVC. El lector tendrá que consultar fuentes adicionales para ver con detalle el resto de opciones.

El primer caso sería el de una petición que no necesita interacción por parte del usuario en el sentido de proceso de datos de entrada: por ejemplo sacar un listado de clientes, mostrar los datos de un pedido, etc. La "no interacción" aquí se entiende como que no hay que procesar y validar datos de entrada. Es decir, que no hay un formulario HTML. Esto no quiere decir que no haya parámetros HTTP, pero entonces suelen estar fijos en la URL de un enlace o de modo similar, no introducidos directamente por el usuario. Estas peticiones suelen ser simplemente listados de información de "solo lectura".

Vamos a poner estos ejemplos en el contexto de una hipotética aplicación web para un hotel, en la cual se pueden ver y buscar ofertas de habitaciones, disponibles con un determinado precio hasta una fecha límite. Aquí tendríamos lo que define a una oferta:

|  |  |
| --- | --- |
|  | package es.ua.jtech.spring.dominio;    import java.math.BigDecimal;  import java.util.Date;    public class Oferta {      private BigDecimal precio;      private Date fechaLimite;      private TipoHabitacion tipoHab;      private int minNoches;           //..aquí vendrían los getters y setters  } |

**RECOMENDACIONES**

**Seguir un orden**

**Primero:** Existen varias alternativas: Netbeans, Eclipse, Intellij, Spring Tool Suite. Nosotros, por precio (gratis) y facilidad de uso recomiendo el uso de Spring Tool Suite.

**Segundo:** crear el proyecto básico, y elegir nuestra herramienta de gestión de librerías. Para crear el proyecto, tenemos varias alternativas, desde configurar nosotros desde cero el servlet de spring, como utilizar plantillas ya existentes. También existe la posibilidad de descargar las librerías de la web de Spring o bien utilizar repositorios de Maven.

**Tercero:** escoger el tipo de vista que se utilizará para el proyecto. En nuestro caso, la vista que utilizamos es Jsp.

**Cuarto:** desarrollar la parte de negocio (controladores) y persistencia (seleccionando un framework como hibernate o jpa), y relacionar los controladores, con la vista y la persistencia.

En principio, eso es todo.

Practicar o desarrollarlo antes, puesto que en al inicio es más complejo y puede llegar a confundir al desarrollador, lo cual solo se evita con la experiencia.

**Conclusiones**.

Hemos aprendido cómo Spring MVC hace que el desarrollo de aplicaciones web muy simple y fácil. Hemos aprendido el módulo del Spring MVC en detalle, su arquitectura y sus diversos componentes. También hemos aprendido los patrones de diseño en el que el módulo de Spring MVC se basa.

**Bibliografía**

<https://es.wikipedia.org/wiki/Spring_Framework>

<http://www.davidmarco.es/spring-mvc>

<http://www.javamexico.org/foros/java_enterprise/%C2%BFpara_que_me_sirve_y_que_resuelvo_con_spring>

<http://ceeicastellon.emprenemjunts.es/?op=14&n=6719>

<http://www.javatutoriales.com/2016/02/spring-mvc-parte-2-seleccion-de.html>