Tabla de Contenidos

- Spring MVC static Resources (Configuracion)
  - Spring MVC Initializer
  - Otros artículos relacionados

Acabamos de ver como usar Java equals y hashcode

El manejo de Spring MVC static resources siempre genera problemas y dudas entre los desarrolladores ya que dependiendo de la versión de Spring pues a veces las configuraciones pueden tener variaciones . Vamos a ver como configurar estos recursos con Spring 5. Para ello como siempre tendremos que ver que dependencias de Maven utilizamos.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.arquitecturajava</groupId>
        <artifactId>springmvc</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>war</packaging>
                <properties>
                <maven.compiler.target>1.8</maven.compiler.target>
                <maven.compiler.source>1.8</maven.compiler.source>
                <failOnMissingWebXml>false</failOnMissingWebXml>
        </properties>
        <dependencies>

                <dependency>
                        <groupId>org.springframework</groupId>
```

```
                        <artifactId>spring-core</artifactId>
                        <version>5.2.6.RELEASE</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-context
-->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context</artifactId>
                        <version>5.2.6.RELEASE</version>
                </dependency>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>3.1.0</version>
                        <scope>provided</scope>
                </dependency>

                <!--
https://mvnrepository.com/artifact/org.springframework/spring-webmvc -
->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-webmvc</artifactId>
                        <version>5.2.7.RELEASE</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.thymeleaf/thymeleaf-spring5 -->
                <dependency>
                        <groupId>org.thymeleaf</groupId>
                        <artifactId>thymeleaf-spring5</artifactId>
```

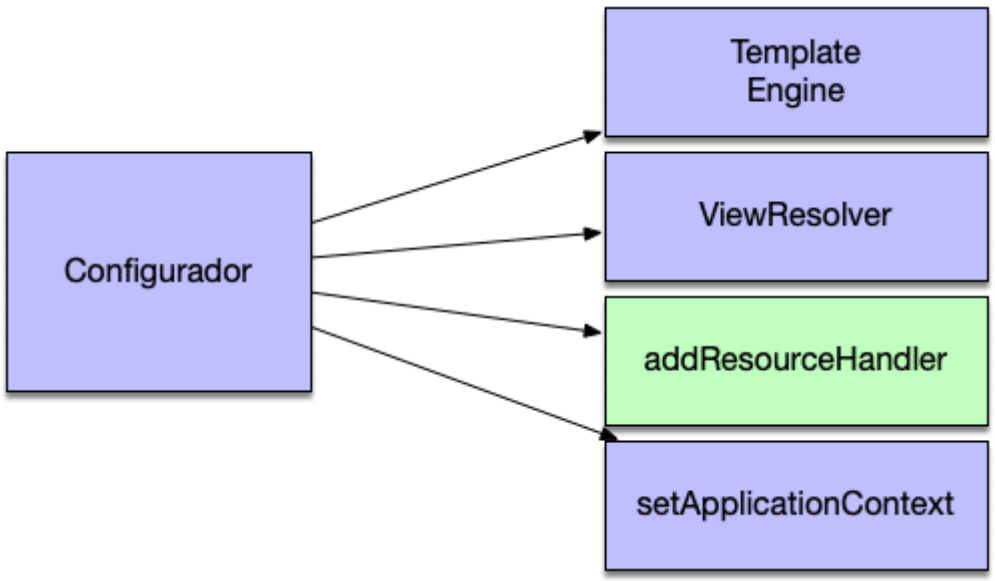
```
                        <version>3.0.11.RELEASE</version>
                </dependency>

        </dependencies>

</project>
```

Spring MVC static Resources (Configuracion)

Una vez tenemos las dependencias definidas . Es comento de crear una clase Configurador que es la encargada de configurar Spring MVC con ThymeLeaf para tener un ejemplo de hola mundo.



Veamos el código:

```
package com.arquitecturajava;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegis
try;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.thymeleaf.spring5.ISpringTemplateEngine;
import org.thymeleaf.spring5.SpringTemplateEngine;
import
org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver;
import org.thymeleaf.spring5.view.ThymeleafViewResolver;
import org.thymeleaf.templatemode.TemplateMode;
import org.thymeleaf.templateresolver.ITemplateResolver;

@Configuration
@EnableWebMvc
@ComponentScan("com.arquitecturajava")
public class Configurador implements WebMvcConfigurer ,
ApplicationContextAware {

        @Autowired
        private ApplicationContext contexto;
        @Bean
        public ViewResolver viewResolver() {
                ThymeleafViewResolver resolver = new
ThymeleafViewResolver();
                resolver.setTemplateEngine(templateEngine());
```

```
                resolver.setCharacterEncoding("UTF-8");
                return resolver;
        }

        @Bean
        public ISpringTemplateEngine templateEngine() {
                SpringTemplateEngine engine = new
SpringTemplateEngine();
                engine.setEnableSpringELCompiler(true);
                engine.setTemplateResolver(templateResolver());
                return engine;
        }

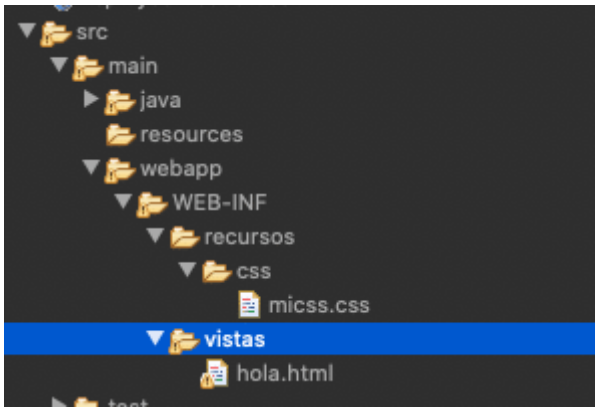
        // la carpeta donde se van a ubicar los ficheros html
        private ITemplateResolver templateResolver() {
                // que resolutor de plantillas uso
                SpringResourceTemplateResolver resolver = new
SpringResourceTemplateResolver();
                // contexto con todos los objetos de spring
                resolver.setApplicationContext(contexto);
                resolver.setPrefix("/WEB-INF/vistas/");
                resolver.setSuffix(".html");

                resolver.setCacheable(false);
                resolver.setTemplateMode(TemplateMode.HTML);
                return resolver;
        }
        @Override
        public void setApplicationContext(ApplicationContext contexto)
throws BeansException {
                this.contexto=contexto;
```

```
        }
        @Override
        public void addResourceHandlers(final ResourceHandlerRegistry
registry) {
registry.addResourceHandler("/recursos/**").addResourceLocations("WEB-
INF/recursos/");
        }

}
```

Como vemos es el último método el de addResourceHandlers el que se encarga de configurar cual es la carpeta que dispondrá de recursos estáticos. En este caso simplemente manejamos una css sencilla.



Spring MVC Initializer

Para complementar el fichero de configuración deberemos definir un SpringInitializer que es el encargado de cargar el servlet controlador (DispatcherServlet) y ligarlo con el contexto de configuración que hemos creado anteriormente.

```
package com.arquitecturajava;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
```

```
import javax.servlet.ServletRegistration;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import
org.springframework.web.context.support.AnnotationConfigWebApplication
Context;
import org.springframework.web.servlet.DispatcherServlet;

public class SpringInicializador implements WebApplicationInitializer
{

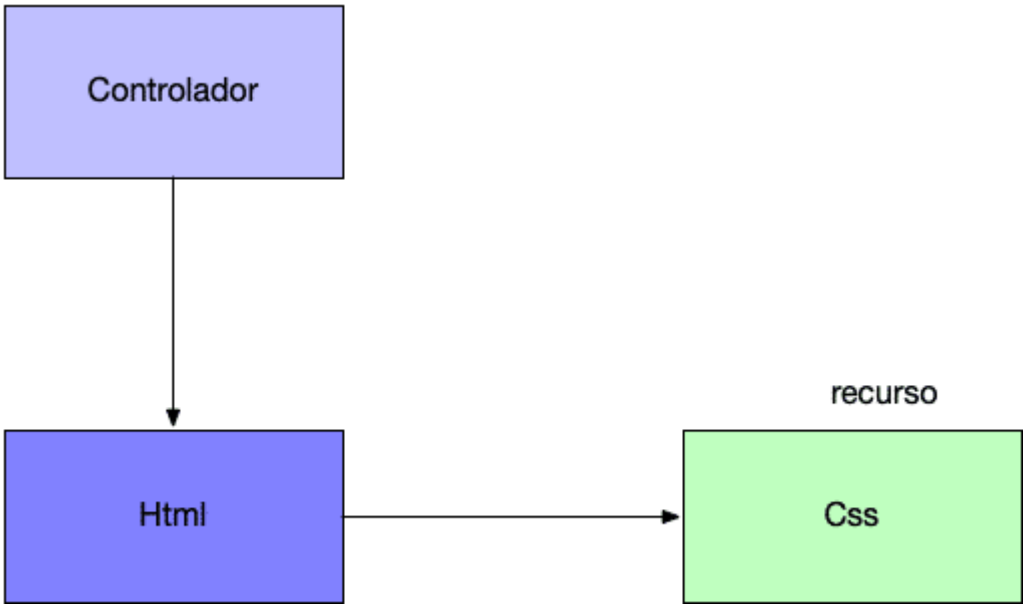
        public void onStartup(ServletContext servletContext) throws
ServletException {

                AnnotationConfigWebApplicationContext contexto = new
AnnotationConfigWebApplicationContext();
                contexto.register(Configurador.class);
                contexto.setServletContext(servletContext);
                ServletRegistration.Dynamic servlet =
servletContext.addServlet("dispatcher", new
DispatcherServlet(contexto));
                servlet.setLoadOnStartup(1);
                servlet.addMapping("/");

        }

}
```

Una vez tenemos toda la configuración construida el siguiente paso es declararnos un controlador que nos redirija a una página html en la cual usemos nuestra hoja de estilo.



Veamoslo:

```
package com.arquitecturajava;

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

@Controller
public class ControladorHola {

	@RequestMapping("/hola")
	public String hola() {
		return "hola";
	}
}
```

Creado el controlador solo nos queda ver el código del HTML definido.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
```

```
<head>
 <link href="recursos/css/micss.css" rel="stylesheet"/>

<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
hola spring mvc
</body>
</html>
```

Nos queda de ver la CSS:

```
body {
        color:red;
}
```

El resultado le podemos ver al cargar la página html. El body cambia de color basándose en la CSS construida.



Acabamos de usar Spring MVC static resources para configurar las hojas de estilo.

Otros artículos relacionados

- Spring MVC
- Utilizando Spring MVC configuration annotation
- Spring MVC @RequestMapping
- Spring MVC Flash Attributes
- Spring MVC