Titulo

Subtitulo

Rubén Gómez García

Version 1.0.0 2019-03-26

Contenidos

Capítulo 1. Introduccion

Framework orientado a la construcción/configuración de proyectos de la familia Spring basado en **Convention-Over-Configuration**, por lo que minimiza la cantidad de código de configuración de las aplicaciones.

Afecta principalmente a dos aspectos de los proyectos

- **Configuracion de dependencias**: Proporcionado por **Starters** Aunque sigue empleando Maven o Gradle para configurar las dependencias del proyecto, abstrae de las versiones de los APIs y lo que es más importante de las versiones compatibles de unos APIs con otros, dado que proporciona un conjunto de librerias que ya están probadas trabajado juntas.
- **Configuracion de los APIs**: Cada API de Spring que se incluye, ya tendrá una preconfiguración por defecto, la cual si se desea se podrá cambiar, además de incluir elementos tan comunes en los desarrollos como un contenedor de servlets embebido ya configurado, estas preconfiguraciones se establecen simplemente por el hecho de que la libreria esté en el classpath, como un Datasource de una base de datos, JDBCTemplate, Java Persistence API (JPA), Thymeleaf templates, Spring Security o Spring MVC.

Además proporciona otras herramientas como

- La consola Spring Boot CLI
- Actuator

Capítulo 2. Creación e implementación de una aplicación

Lo primero a resolver al crear una aplicación son las dependencias, para ellos Spring Boot ofrece el siguiente mecanismo basando en la herencia del POM.

```
<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    . . .

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.4.2.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    . . .

</project>
```

De no poderse establecer dicha herencia, por heredar de otro proyecto, se ofrece la posibilidad de añadir la siguiente dependedencia.

```
<project>

    . . .

    <dependencyManagement>
         <dependencies>
            <dependency>
                <!-- Import dependency management from Spring Boot -->
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>1.4.2.RELEASE</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    . . .

</project>
```

Esta dependencia permite a Spring Boot hacer el trabajo sucio para manejar el ciclo de vida de un proyecto Spring normal, pero normalmente se precisarán otras dependencias, para esto Spring Boot ofrece los **Starters**

2.1. Aplicación Web

Para crear una aplicación web con Spring Web MVC, se ha de añadir la siguiente dependencia.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Una vez solventadas las dependencias, habrá que configurar el proyecto, ya hemos mencionado que la configuración quedará muy reducida, en este caso unicamente necesitamos definir una clase anotada con **@SpringBootApplication**

```
@SpringBootApplication
public class HolaMundoApplication {
    . . .
}
```

Esta anotacion en realidad es la suma de otras tres:

- **@Configuration**: Se designa a la clase como un posible origen de definiciones de Bean.
- **@ComponentScan**: Se indica que se buscarán otras clases con anotaciones que definan componentes de Spring como @Controller
- **@EnableAutoConfiguration**: Es la que incluye toda la configuración por defecto para los distintos APIs seleccionados.

Con esto ya se tendría el proyecto preparado para incluir unicamente el código de aplicación necesario, por ejemplo un Controller de Spring MVC

```
@Controller
public class HolaMundoController {
    @RequestMapping("/")
    @ResponseBody
    public String holaMundo() {
        return "Hola Mundo!!!!!";
    }
}
```

Una vez finalizada la aplicación, se podría ejecutar de varias formas

- Como jar autoejecutable, para lo que habrá que definir un método **Main** que invoque **SpringApplication.run()**

```
@SpringBootApplication
public class HolaMundoApplication {
    public static void main(String[] args) {
        SpringApplication.run(HolaMundoApplication.class, args);
    }
}
```

Y posteriormente ejecutandolo con

- Una tarea de Maven

```
mvn spring-boot:run
```

- Una tarea de Gradle

```
gradle bootRun
```

- O como jar autoejecutable, generando primero el jar

Con Maven

```
mvn package
```

O Gradle

```
gradle build
```

Y ejecutando desde la linea de comandos

```
java -jar HolaMundo-0.0.1-SNAPSHOT.jar
```

- O desplegando como WAR en un contenedor web, para lo cual hay que añadir el plugin de WAR
  - En Maven, con cambiar el package bastará

```
<packaging>war</packaging>
```

- En Gradle alicando el plugin de WAR y cambiando la configuracion JAR por la WAR

```
apply plugin: 'war'

war {
    baseName = 'HolaMundo'
    version = '0.0.1-SNAPSHOT'
}
```

En estos casos, dado que no se ha generado el **web.xml**, es necesario realizar dicha inicialización, para ello Spring Boot ofrece la clase **org.springframework.boot.web.support.SpringBootServletInitializer**

```
public class HolaMundoServletInitializer extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(HolaMundoApplication.class);
    }
}
```

2.2. Aplicación de consola

Si se desea lanzar una serie de comandos en el proceso de arranque de la aplicación, tipicamente cuando se quiere realizar alguna demo de algún API, se puede implementar la interface **CommandLineRunner**

```
@SpringBootApplication
public class Application implements CommandLineRunner{

    private static final Logger logger = LoggerFactory.getLogger(Application.class);

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        logger.info("Args : {}", args);
    }
}
```

O la interface **ApplicationRunner**

```
@SpringBootApplication
public class Application implements ApplicationRunner {

    private static final Logger logger = LoggerFactory.getLogger(Application.class);

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        logger.info("Option names : {}", args.getOptionNames());
    }
}
```

Capítulo 3. Uso de plantillas

Los proyectos **Spring Boot Web** vienen configurados para emplear plantillas, basta con añadir el starter del motor deseado y definir las plantillas en la carpeta **src/main/resources/templates**.

Algunos de los motores a emplear son Thymeleaf, freemaker, velocity, jsp, . . .

3.1. Thymeleaf

Motor de plantillas que se basa en la instrumentalización de **html** con atributos obtenidos del esquema **th**

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

Para añadir esta caracteristica al proyecto, se añade la dependencia de Maven

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Por defecto cualquier **String** retornado por un **Controlador** será considerado el nombre de un **html** instrumentalizado con **thymeleaf** que se ha de encontrar en la carpeta **/src/main/resources/templates**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1"></meta>
    <title>Insert title here</title>
</head>
<body>
    <span th:text="${mensaje}"></span>
    <span th:text="#"></span>
</body>
</html>
```

No es necesario indicar el espacio de nombres en el html

3.2. JSP

Para poder emplear **JSP** en lugar de **Thymeleaf**, hay dos opciones, la primera es definir el proyecto de Spring Boot como War en el pom.xml, definiendo la siguiente configuracion en el contexto de Spring

```
@SpringBootApplication
public class SampleWebJspApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application)
{
        return application.sources(SampleWebJspApplication.class);
    }

    public static void main(String[] args) throws Exception {
        SpringApplication.run(SampleWebJspApplication.class, args);
    }

}
```

y las siguientes propiedades en el fichero **application.properties**

```
spring.mvc.view.prefix: /WEB-INF/views/
spring.mvc.view.suffix: .jsp
```

El directorio desde donde creacerá **WEB-INF**, sera **src/main/webapp**

La segunda opcion, será mantener el tipo de proyecto como Jar y añadir las siguientes dependencias al **pom.xml**

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

Por último indicar donde encontrar los ficheros mediante las siguientes propiedades en el fichero **application.properties**

```
spring.mvc.view.prefix: /WEB-INF/views/
spring.mvc.view.suffix: .jsp
```

El directorio desde donde crecerá **WEB-INF**, sera **src/main/resources/META-INF/resources/** o **src/main/webapp/**

3.3. Recursos estaticos

Si se desean publicar recursos estaticos (html, js, css, ..), se pueden incluir en los proyectos en las rutas:

- **src/main/resources/META-INF/resources**
- **src/main/resources/resources**
- **src/main/resources/static**
- **src/main/resources/public**

Siendo el descrito el orden de inspeccion.

3.4. Webjars

Desde hace algun tiempo se encuentran disponibles como dependencias de Maven las distribuciones de algunos frameworks javascript bajo el groupid **org.webjars**, pudiendo añadir dichas dependencias a los proyectos para poder gestionar con herramientas de construccion como Maven o Gradle tambien las versiones de los frameworks javascript.

Estos artefactos tienen incluido los ficheros js, en la carpeta **/META-INF/resources/webjars/<artifactId>/<version>**, con lo que las dependencias hacia los ficheros javascript de los framework añadidos con Maven será **webjars/<artifactId>/<version>/<artifactId>.min.js**

```
<html>
<head>
    <script src="webjars/jquery/2.0.3/jquery.min.js"></script>
. . .
```

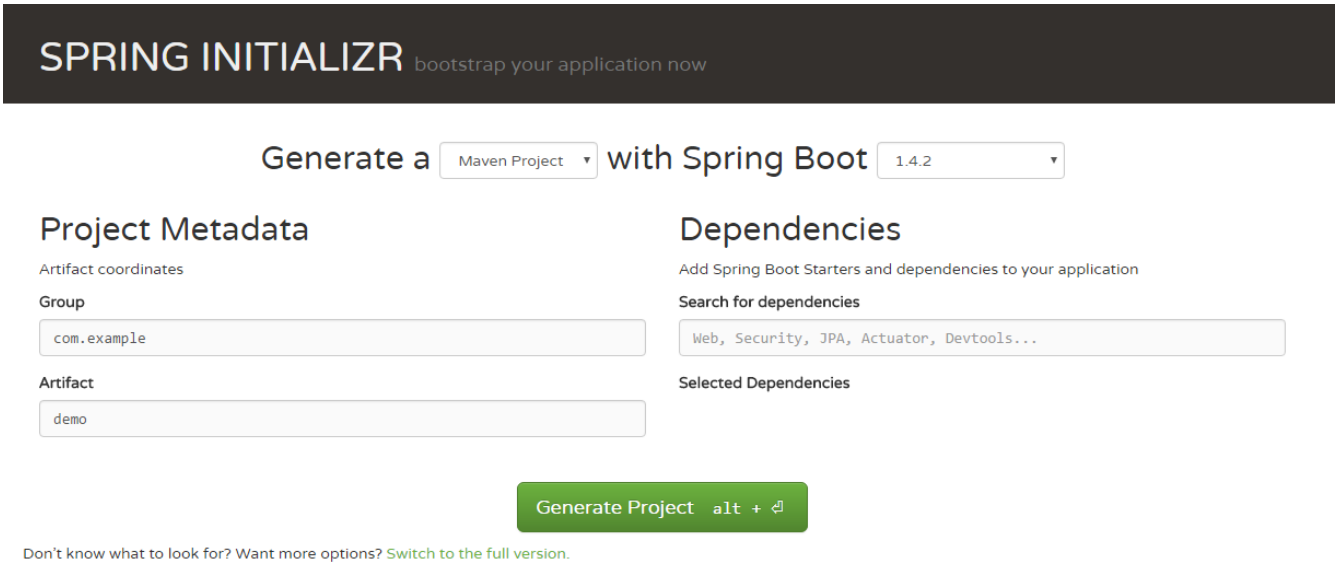
Capítulo 4. Uso de Java con start.spring.io

Es uno de los modos de emplear el API de **Spring Initializr**, al que tambien se tiene acceso desde

- Spring Tool Suite
- IntelliJ IDEA
- Spring Boot CLI

Es una herramienta que permite crear estructuras de proyectos de forma rapida, a través de plantillas.

Desde la pagina start.spring.io se puede generar una plantilla de proyecto.



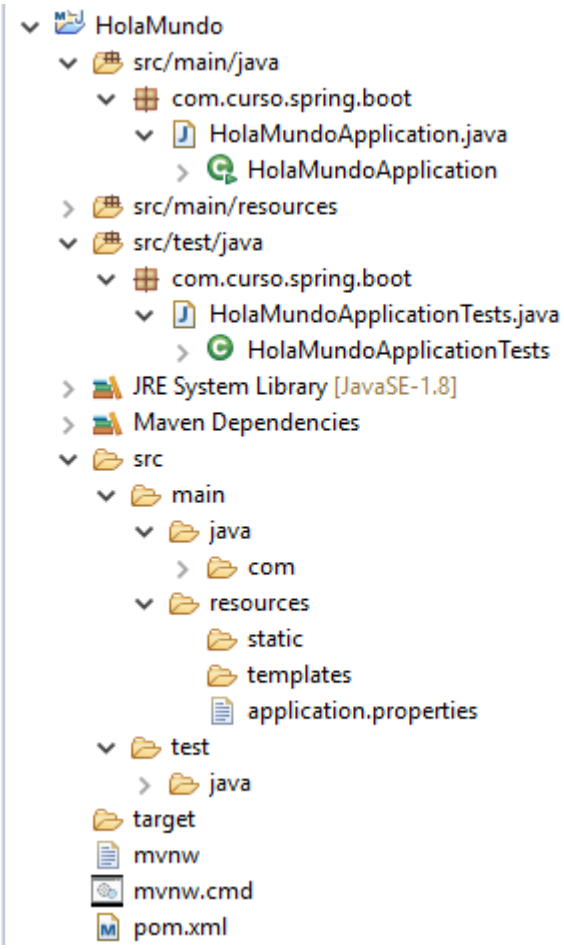
Lo que se ha de proporcionar es

- Tipo de proyecto (Maven o Gradle)
- Versión de Spring Boot
- GroupId
- ArtifactId
- Dependencias

Existe una vista avanzada donde se pueden indicar otros parametros como

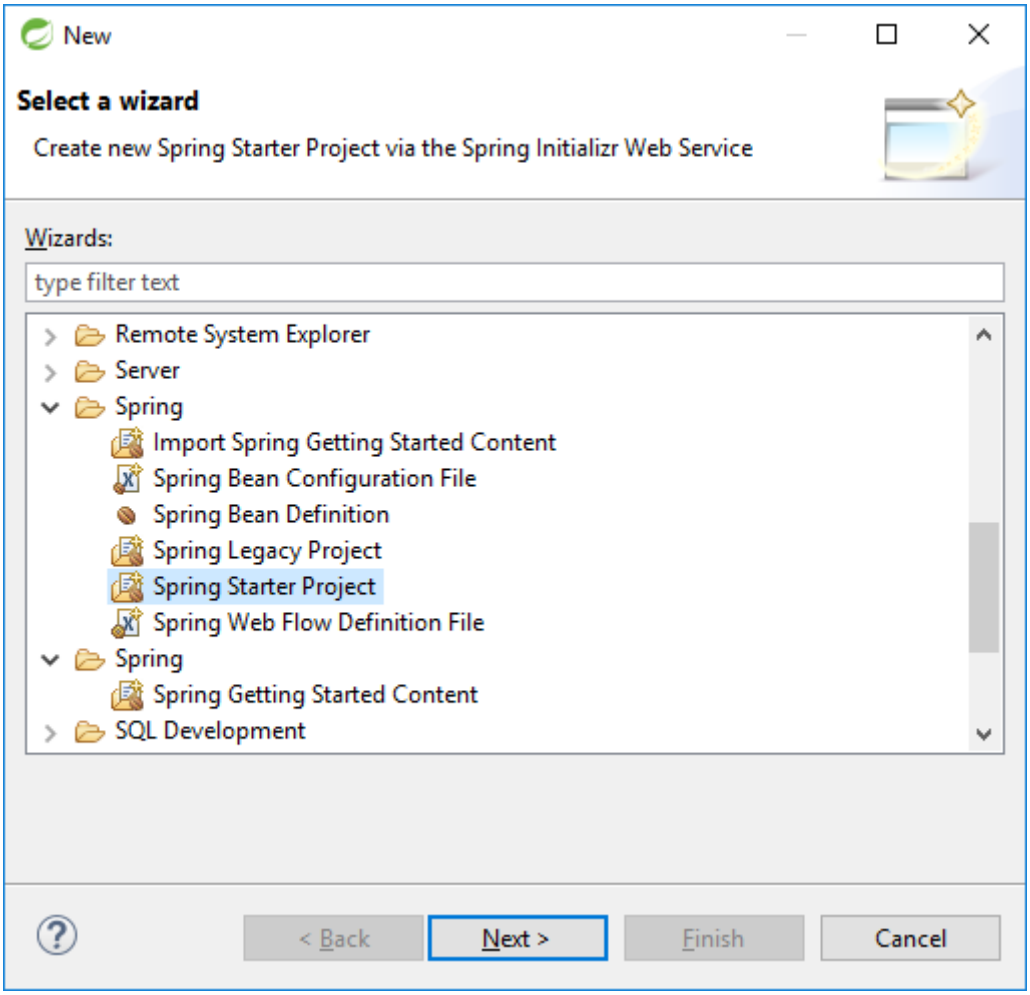
- Versión de java
- El tipo de packaging
- El lenguaje del proyecto
- Seleccion mas detallada de las dependencias

La estructura del proyecto con dependencia web generado será

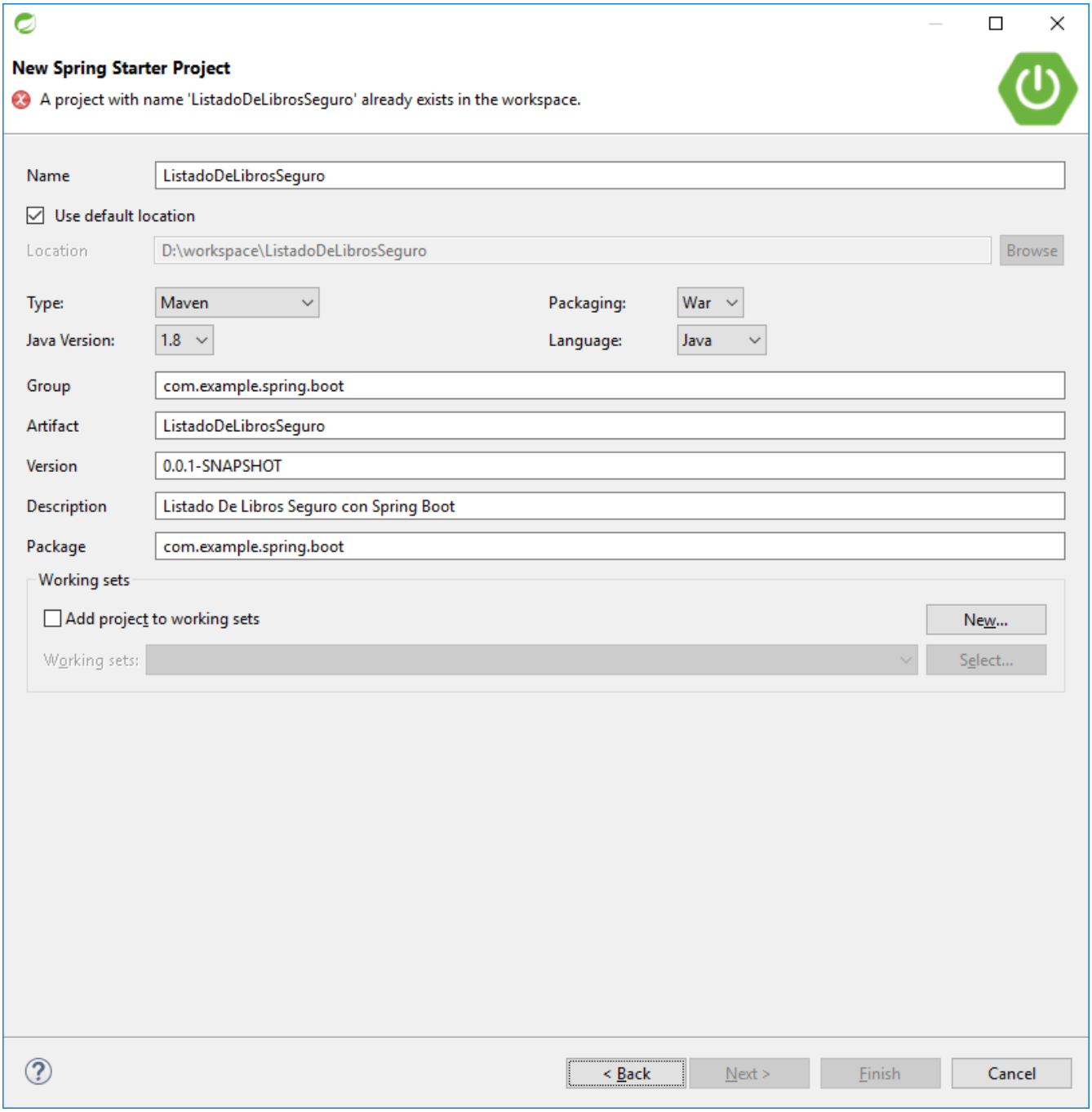


En esta estructura, cabe destacar el directorio **static**, destinado a contener cualquier recurso estatico de una aplicación web.

Desde Spring Tools Suite, se puede acceder a esta misma funcionalidad desde **New** > **Other** > **Spring** > **Spring Starter Project**, es necesario tener internet, ya que STS se conecta a **start.spring.io**



Una vez seleccionada la opcion, se muestra un formulario similar al de la web



Y desde Spring CLI con el comando **init** tambien, un ejemplo de comando seria

```
Spring-CLI# init --build maven --groupId com.ejemplo.spring.boot.web --version 1.0
--java-version 1.8 --dependencies web --name HolaMundo HolaMundo
```

Que genera la estructura anterior dentro de la carpeta **HolaMundo**

Se puede obtener ayuda sobre los parametros con el comando

```
Spring-CLI# init --list
```

Capítulo 5. Starters

Son dependencias ya preparadas por Spring, para dotar del conjunto de librerias necesarias para obtener un funcionalidad sin que existan conflictos entre las versiones de las distintas librerias.

Se pueden conocer las dependencias reales con las siguientes tareas

- Maven

```
mvn dependency:tree
```

- Gradle

```
gradle dependencies
```

De necesitarse, se pueden sobrescribir las versiones o incluso excluir librerias, de las que nos proporcionan los **Starter**

- Maven

```
<dependency>
    <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>
            <exclusion>
                <groupId>com.fasterxml.jackson.core</groupId>
            </exclusion>
    </exclusions>
</dependency>
```

- Gradle

```
compile("org.springframework.boot:spring-boot-starter-web") {
    exclude group: 'com.fasterxml.jackson.core'
}
```

Capítulo 6. Soporte a propiedades

Spring Boot permite configurar unas 300 propiedades, aqui una lista de ellas.

Se pueden configurar los proyectos de Spring Boot unicamnete modificando propiedades, estas se pueden definir en

- Argumentos de la linea de comandos

```
java -jar app-0.0.1-SNAPSHOT.jar --spring.main.show-banner=false
```

- JNDI

```
java:comp/env/spring.main.show-banner=false
```

- Propiedades del Sistema Java

```
java -jar app-0.0.1-SNAPSHOT.jar -Dspring.main.show-banner=false
```

- Variables de entorno del SO

```
SET SPRING_MAIN_SHOW_BANNER=false;
```

- Un fichero **application.properties**

```
spring.main.show-banner=false
```

- Un fichero **application.yml**

```
spring:
    main:
        show-banner: false
```

Las listas en formato **YAML** tiene la siguiente sintaxis

```
security:
  user:
    role:
      - SUPERUSER
      - USER
```

De existir varias de las siguientes, el orden de preferencia es el del listado, por lo que la mas

prioritaria es la linea de comandos.

Los ficheros **application.properties** y **application.yml** pueden situarse en varios lugares

- En un directorio **config** hijo del directorio desde donde se ejecuta la aplicación.
- En el directorio desde donde se ejecuta la aplicación.
- En un paquete **config** del proyecto
- En la raiz del classpath.

Siendo el orden de preferencia el del listado, si aparecieran los dos ficheros, el **.propertes** y el **.yml**, tiene prioridad el properties.

Algunas de las propiedades que se pueden definir son:

- **spring.main.show-banner**: Mostrar el banner de spring en el log (por defecto true).
- **spring.thymeleaf.cache**: Deshabilitar la cache del generador de plantillas thymeleaf
- **spring.freemarker.cache**: Deshabilitar la cache del generador de plantillas freemarker
- **spring.groovy.template.cache**: Deshabilitar la cache de plantillas generadas con groovy
- **spring.velocity.cache**: Deshabilitar la cache del generador de plantillas velocity
- **spring.profiles.active**: Perfil activado en la ejecución

La cache de las plantillas, se emplea en producción para mejorar el rendimiento, pero se debe desactivar en desarrollo ya que sino se ha de parar el servidor cada vez que se haga un cambio en las plantillas.

6.1. Configuracion del Servidor

- **server.port**: Puerto del Contenedor Web donde se exponen los recursos (por defecto 8080, para ssl 8443).
- **server.contextPath**: Permite definir el primer nivel del path de las url para el acceso a la aplicacion (Ej: /resource).
- **server.ssl.key-store**: Ubicación del fichero de certificado (Ej: file:///path/to/mykeys.jks).
- **server.ssl.key-store-password**: Contraseña del almacen.
- **server.ssl.key-password**: Contraseña del certificado.

Para generar un certificado, se puede emplear la herramienta **keytool** que incluye la jdk

```
keytool -keystore mykeys.jks -genkey -alias tomcat -keyalg RSA
```

6.2. Configuracion del Logger

- **logging.level.root**: Nivel del log para el log principal (Ej: WARN)
- **logging.level.<paquete>**: Nivel del log para un log particular (Ej: logging.level.org.springframework.security: DEBUG)
- **logging.path**: Ubicacion del fichero de log (Ej: /var/logs/)
- **logging.file**: Nombre del fichero de log (Ej: miApp.log)

6.3. Configuracion del Datasource

- **spring.datasource.url**: Cadena de conexión con el origen de datos por defecto de la auto-configuración (Ej: jdbc:mysql://localhost/test)
- **spring.datasource.username**: Nombre de usuario para conectar al origen de datos por defecto de la auto-configuración (Ej: dbuser)
- **spring.datasource.password**: Password del usuario que se conecta al origen de datos por defecto de la auto-configuración (Ej: dbpass)
- **spring.datasource.driver-class-name**: Driver a emplear para conectra con el origen de datos por defecto de la auto-configuración (Ej: com.mysql.jdbc.Driver)
- **spring.datasource.jndi-name**: Nombre JNDI del datasorce que se quiere emplear como origen de datos por defecto de la auto-configuración.
- **spring.datasource.name**: El nombre del origen de datos
- **spring.datasource.initialize**: Whether or not to populate using data.sql (default:true)
- **spring.datasource.schema**: The name of a schema (DDL) script resource
- **spring.datasource.data**: The name of a data (DML) script resource
- **spring.datasource.sql-script-encoding**: The character set for reading SQL scripts
- **spring.datasource.platform**: The platform to use when reading the schema resource (for example, "schema-{platform}.sql")
- **spring.datasource.continue-on-error**: Whether or not to continue if initialization fails (default: false)
- **spring.datasource.separator**: The separator in the SQL scripts (default: ;)
- **spring.datasource.max-active**: Maximum active connections (default: 100)
- **spring.datasource.max-idle**: Maximum idle connections (default: 8)
- **spring.datasource.min-idle**: Minimum idle connections (default: 8)
- **spring.datasource.initial-size**: The initial size of the connection pool (default: 10)
- **spring.datasource.validation-query**: A query to execute to verify the connection
- **spring.datasource.test-on-borrow**: Whether or not to test a connection as it’s borrowed from the pool (default: false)
- **spring.datasource.test-on-return**: Whether or not to test a connection as it’s returned to the pool (default: false)

- **spring.datasource.test-while-idle**: Whether or not to test a connection while it is idle (default: false)
- **spring.datasource.max-wait**: The maximum time (in milliseconds) that the pool will wait when no connections are available before failing (default: 30000)
- **spring.datasource.jmx-enabled**: Whether or not the data source is managed by JMX (default: false)

Solo se puede configurar un unico datasource por auto-configuración, para definir otro, se ha de definir el bean correspondiente

6.3.1. Custom Properties

Se puede definir nuevas propiedades y emplearlas en la aplicación dentro de los Bean.

- Para ello se ha de definir, dentro de un Bean de Spring, un atributo de clase que refleje la propiedad y su método de SET

```
private String prefijo;
public void setPrefijo(String prefijo) {
    this.prefijo = prefijo;
}
```

- Para las propiedades con nombre compuesto, se ha de configurar el prefijo con la anotación **@ConfigurationProperties** a nivel de clase

```
@Controller
@RequestMapping("/")
@ConfigurationProperties(prefix="saludo")
public class HolaMundoController {}
```

- Ya solo falta definir el valor de la propiedad en **application.properties** o en **application.yml**

```
saludo:
  prefijo: Hola
```

Para que la funcionalidad de properties funcione, se debe añadir **@EnableConfigurationProperties**, pero con Spring Boot no es necesario, ya que está incluido por defecto.

Otra opcion para emplear propiedades, es el uso de la anotacion **@Value** en cualquier propiedad de un bean de spring, que permite leer la propiedad si esta existe o asignar un valor por defecto en caso que no exista.

```
@Value("${message:Hello default}")
private String message;
```

6.4. Profiles

Se pueden anotar **@Bean** con **@Profile**, para que dicho Bean sea solo añadido al contexto de Spring cuando el profile indicado esté activo.

```
@Bean
@Profile("production")
public DataSource dataSource() {
    DataSource ds = new DataSource();
    ds.setDriverClassName("org.mysql.Driver");
    ds.setUrl("jdbc:mysql://localhost:5432/test");
    ds.setUsername("admin");
    ds.setPassword("admin");
    return ds;
}
```

Tambien se puede definir un conjunto de propiedades que solo se empleen si un perfil esta acivo, para ello, se ha de crear un nuevo fichero **application-{profile}.properties**.

En el caso de los ficheros de YAML, solo se define un fichero, el **application.yml**, y en el se definen todos los perfiles, separados por **---**

```
---
spring:
    profiles: production
    datasource:
        url: jdbc:mysql://localhost:5432/test
        username: admin
        password: admin
        jpa:
            database-platform: org.hibernate.dialect.MySQLDialect
```

Para activar un **Profile**, se emplea la propiedad **spring.profiles.active**, la cual puede establecerse como:

- Variable de entorno

```
SET SPRING_PROFILES_ACTIVE=production;
```

- Con un parametro de inicio

```
java -jar applicacion-0.0.1-SNAPSHOT.jar --spring.profiles.active=production
```

De definirse mas de un perfil activo, se indicaran con un listado separado por comas

Capítulo 7. JPA

Al añadir el starter de JPA, por defecto Spring Boot va a localizar todos los Bean detro del paquete y subpaquetes donde se encuentra la clase anotada con **@SpringBootApplication** en busca de interfaces Repositorio, que extiendan la interface **JpaRepository**, de no encontrarse la interface que define el repositorio dentro del paquete o subapaquetes, se puede referenciar con **@EnableJpaRepositories**

Cuando se emplea JPA con Hibernate como implementación, esté último tiene la posibilidad de configurar su comportamiento con respecto al schema de base de datos, pudiendo indicarle que lo cree, que lo actualice, que lo borre, que lo valide. . . esto se consigue con la propiedad **hibernate.ddl-auto**

```
spring:
    jpa:
        hibernate:
            ddl-auto: validate
```

Los posibles valores para esta propiedad son:

- none: This is the default for MySQL, no change to the database structure.
- update: Hibernate changes the database according to the given Entity structures.
- create: Creates the database every time, but don't drop it when close.
- create-drop: Por defecto para H2. the database then drops it when the SessionFactory closes.

Habrá que añadir al classpath, con dependencias de Maven, el driver de la base de datos a emplear, Spring Boot detectará el driver añadido y conectará con una base de datos por defecto.

La dependencia para MySQL será

```
 <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Las versiones de algunas depedencias no es necesario que se indiquen en Spring Boot, ya que vienen predefinidas en el **parent**

Para configurar un nuevo origen de datos, se indican las siguientes propiedades

```
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=jdbc:mysql://localhost:3306/db_example
spring.datasource.username=springuser
spring.datasource.password=ThePassword
```

Capítulo 8. Errores

Por defecto Spring Boot proporciona una pagina para represenar los errores que se producen en las aplicaciones llamada **whitelabel**, para sustituirla por una personalizada, basta con definir alguno de los siguientes componentes

- Cuanlquier Bena que implemente **View** con Id **error**, que será resuelto por **BeanNameViewResolver**.
- Plantilla **Thymeleaf** llamada **error.html** si **Thymeleaf** esta configurado.
- Plantilla **FreeMarker** llamada **error.ftl** si **FreeMarker** esta configurado.
- Plantilla **Velocity** llamada **error.vm** si **Velocity** esta configurado.
- Plantilla **JSP** llamada **error.jsp** si se emplean vistas JSP.

Dentro de la vista, se puede acceder a la siguiente información relativa al error

- **timestamp**: La hora a la que ha ocurrido el error
- **status**: El código HTTP
- **error**: La causa del error
- **exception**: El nombre de la clase de la excepción.
- **message**: El mensaje del error
- **errors**: Los errores si hay mas de uno
- **trace**: La traza del error
- **path**: La URL a la que se accedia cuando se produjo el error.