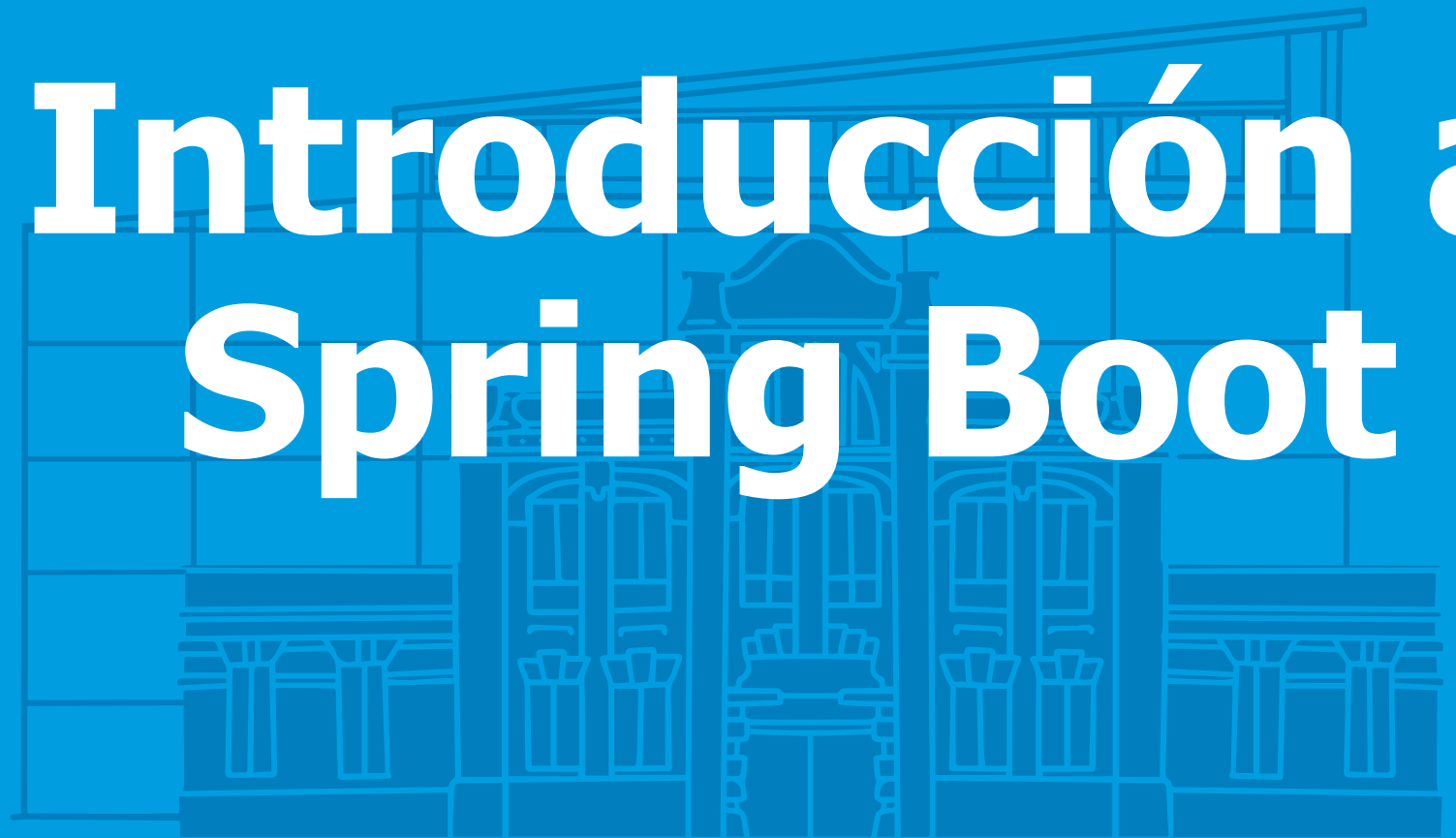
netmind
a bts company

Spring Boot

Introducción a Spring Boot



01

Introducción

Spring Boot

Spring Boot es un framework basado en Java de código abierto que se utiliza para crear un microservicio.

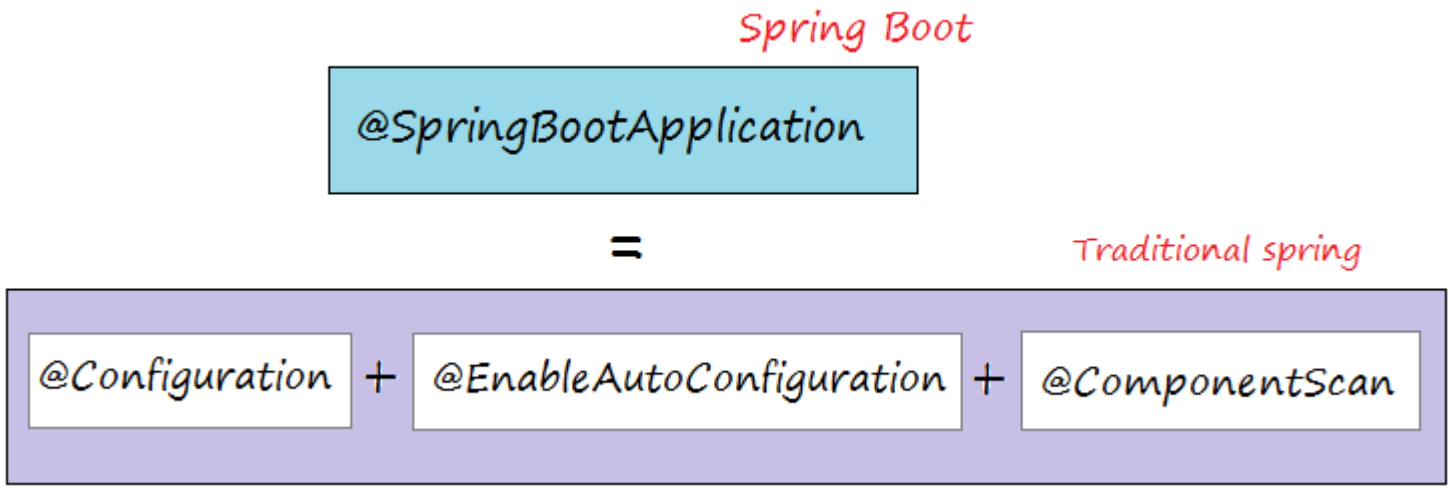
Está desarrollado por Pivotal Team y se utiliza para crear aplicaciones Spring independientes y listas para producción.

- Fácil de entender y desarrollar aplicaciones spring.
- Aumenta la productividad
- Reduce el tiempo de desarrollo

Cómo funciona

- Spring Boot configura automáticamente la aplicación en función de las dependencias que haya agregado al proyecto mediante la anotación **@EnableAutoConfiguration**.
- El punto de entrada de la aplicación Spring Boot es la clase que contiene la anotación **@SpringBootApplication** y el método **main**.

- Spring Boot escanea automáticamente todos los componentes incluidos en el proyecto usando la anotación **@ComponentScan**.



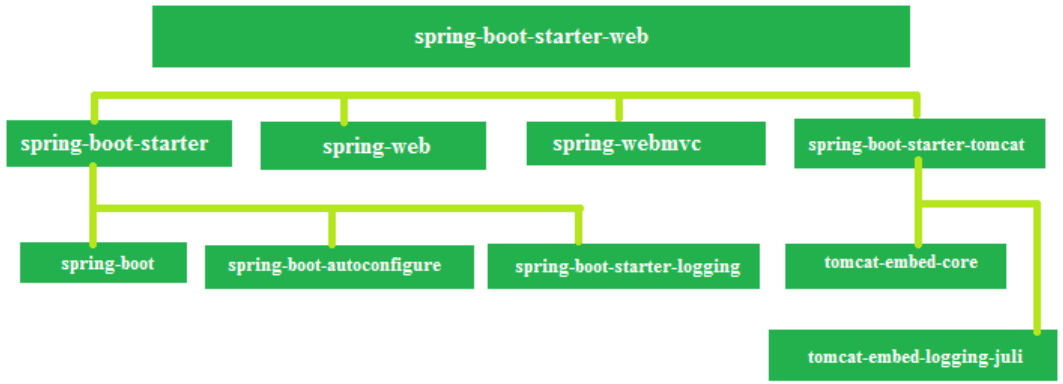
Spring Boot Starters

- Gestionar las dependencias es una tarea difícil para grandes proyectos.
- Spring Boot resuelve este problema proporcionando un conjunto de dependencias.
- Por ejemplo, si desea usar Spring y JPA para el acceso a la base de datos, es suficiente incluir la dependencia **spring-boot-starter-data-jpa** en el proyecto.
- todos los starters de Spring Boot siguen el mismo patrón de nomenclatura **spring-boot-starter- ***, donde * indica que es un tipo de aplicación.

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

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

Spring Boot Starters



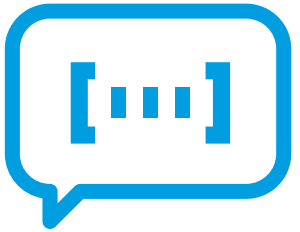
Name	Description
spring-boot-starter-web-services	For building applications exposing SOAP web services
spring-boot-starter-web	Build web applications and RESTful applications
spring-boot-starter-test	Write great unit and integration tests
spring-boot-starter-jdbc	Traditional JDBC applications
spring-boot-starter-hateoas	Make your services more RESTful by adding HATEOAS features
spring-boot-starter-security	Authentication and authorization using Spring Security
spring-boot-starter-data-jpa	Spring Data JPA with Hibernate
spring-boot-starter-cache	Enabling the Spring Framework's caching support
spring-boot-starter-data-rest	Expose simple REST services using Spring Data REST

Aplicación de arranque

- **La anotación @SpringBootApplication** incluye configuración automática, análisis de componentes y configuración de Spring Boot.
- **@SpringBootApplication** incluye la **anotación @EnableAutoConfiguration, @ComponentScan** y **@SpringBootConfiguration**.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

netmind
a bts company

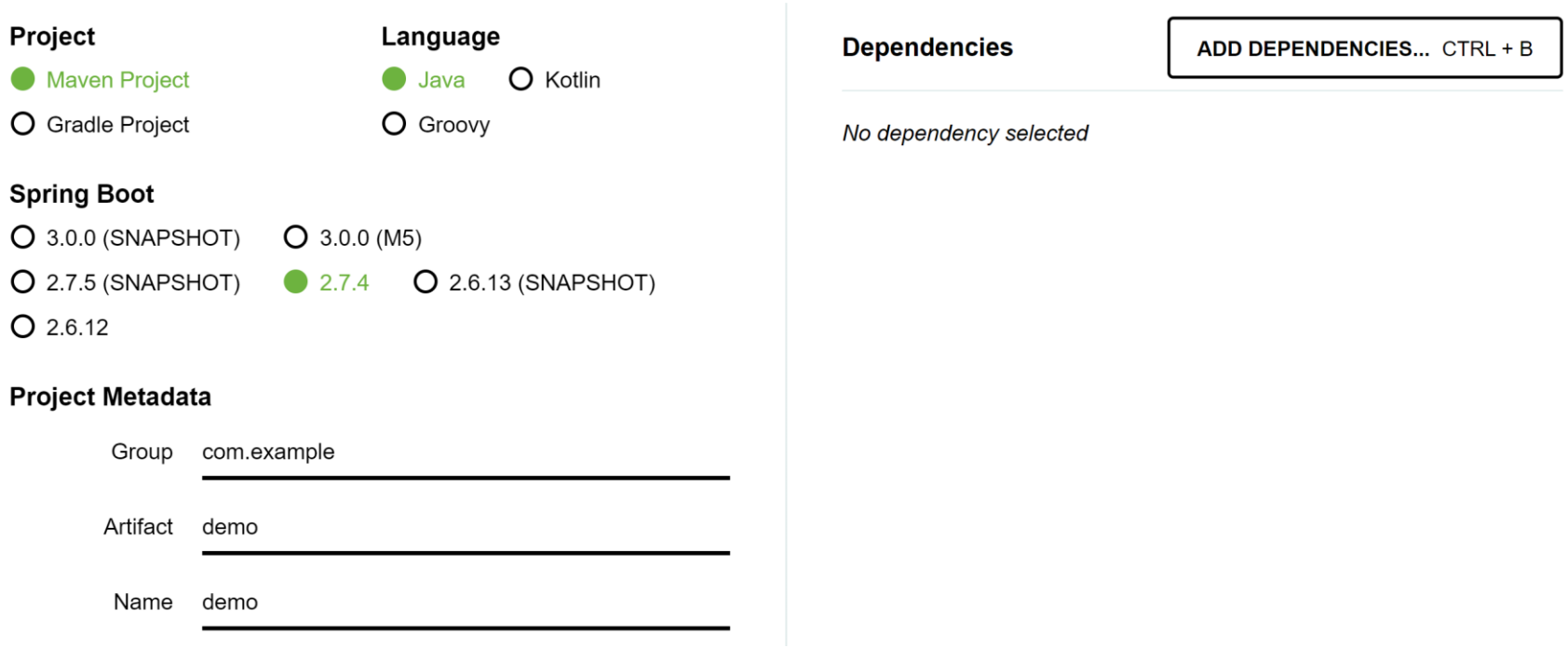


Boostrapping

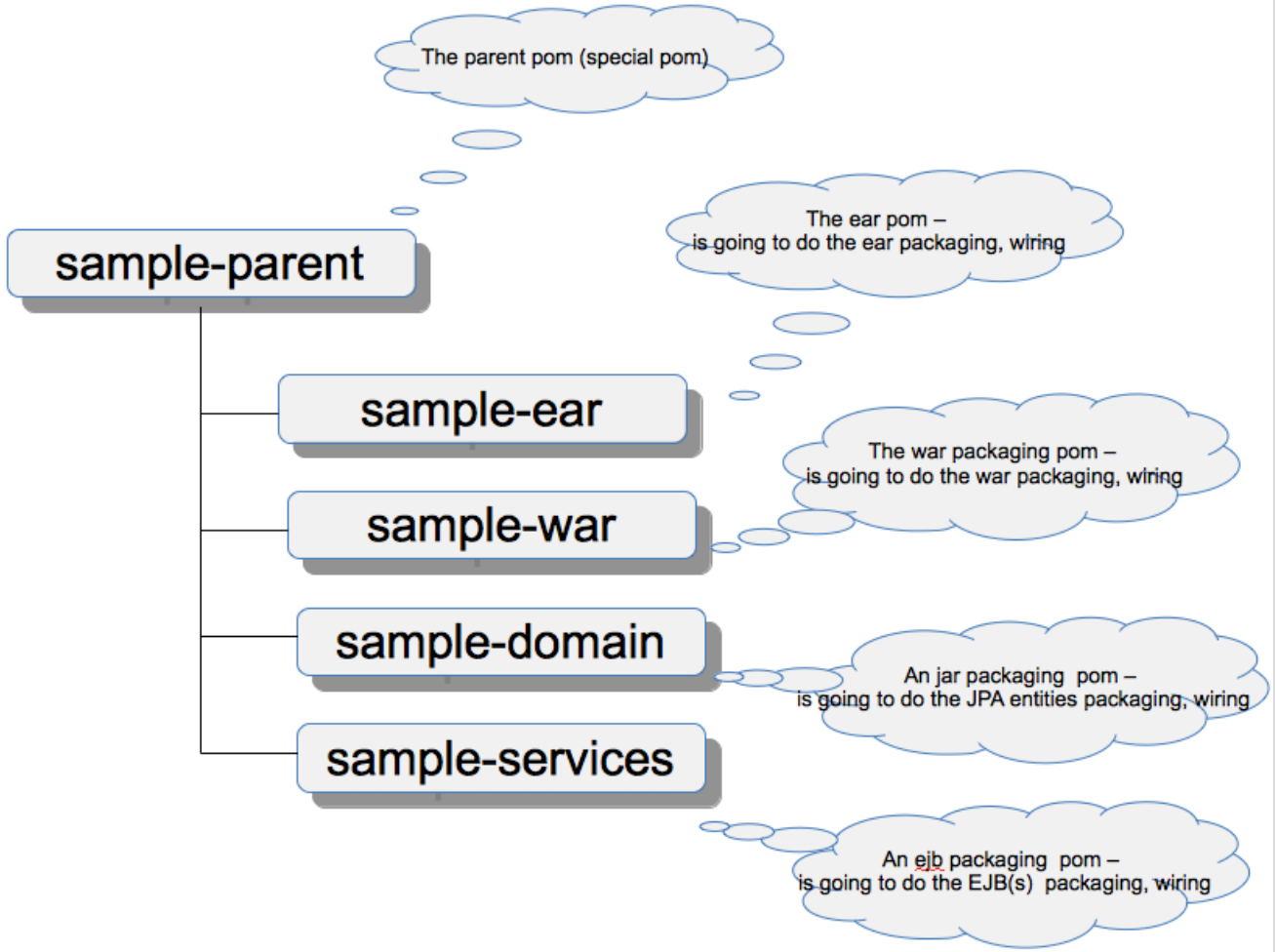
02

Spring Initializer

- Una de las opciones de hacer Bootstrapping de una aplicación Spring Boot es usando **Spring Initializer**.
  - https://start.spring.io/

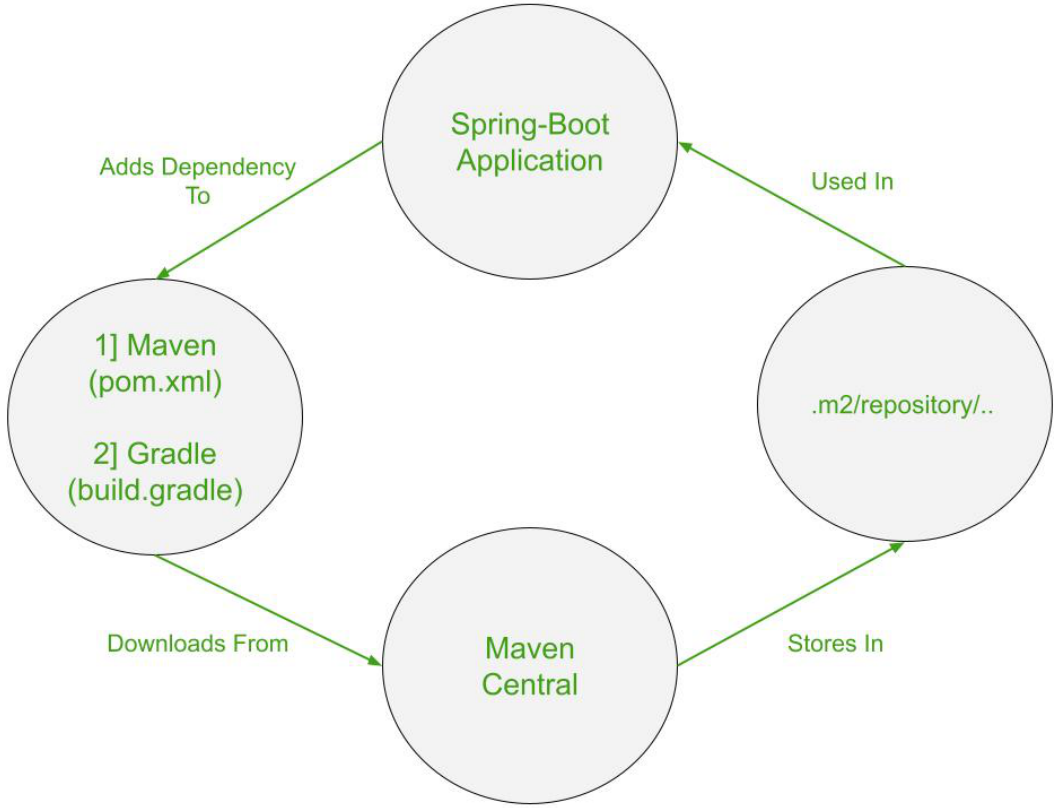


Proyectos Maven por módulos



Maven

- Después de descargar y descomprimir el proyecto, el archivo **pom.xml** contiene una estructura basada en módulos.



Un endpoint Rest

```
package com.netmind.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController

public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
   @RequestMapping(value = "/")
   public String hello() {
      return "Hello World";
   }
}
```

Crear y ejecutar el jar

- Crea el jar

```
mvn clean install
```

- Ejecuta el jar

```
java –jar <JARFILE>
```

- Ve al navegador y accede a:
  - http://localhost:8080/

Usando spring-boot-devtools

- Dependencias:
  - https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/

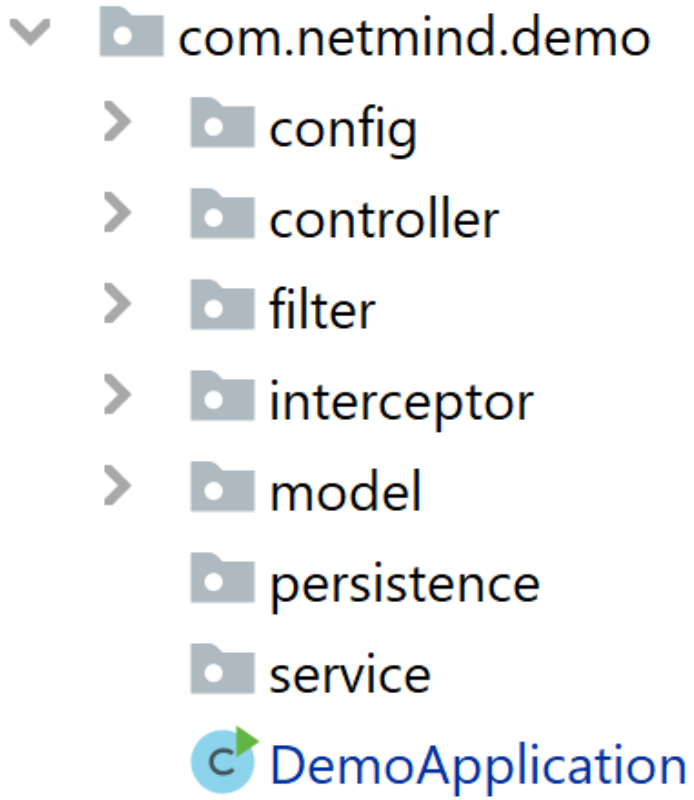
```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-devtools</artifactId>
		<scope>runtime</scope>
		<optional>true</optional>
</dependency>
```

- Lanzar la app

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

- Ve al navegador y accede a:
  - http://localhost:8080/

Estructura de la aplicación



03

Propiedades de la aplicación

Archivo de propiedades

- Los archivos de propiedades se utilizan para mantener un número 'N' de propiedades en un solo archivo para ejecutar la aplicación en diferentes entornos.
- En Spring Boot, las propiedades se guardan en el archivo **application.properties** en el directorio **src/main/resources** en el classpath.

```
server.port = 9090
spring.application.name = demoservice
```

- La lista de propiedades que se pueden definir se encuentran en esta referencia:
  - https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

Archivo YAML

- Spring Boot admite configuraciones de propiedades basadas en YAML usando el archivo **application.yml**.

```
spring:
   application:
      name: demoservice
server:
    port: 9090
```

Usando/Inyectando propiedades

- La anotación **@Value** se utiliza para leer el valor de la propiedad del entorno o de la aplicación en código Java.

```
@Value("${spring.application.name}")
private String name;
```

  - Si no se encuentra la propiedad mientras se ejecuta la aplicación, Spring Boot lanza la excepción "Illegal Argument". Para resolver el problema, podemos establecer el valor predeterminado

```
@Value("${property_key_name:default_value}")

@Value("${spring.application.name:demoservice}")
```

Perfil activo

- Spring Boot admite diferentes propiedades basadas en el perfil activo de Spring.
  - **application.properties**

```
server.port = 8080
spring.application.name = demoservice
```

  - **application-dev.properties**

```
server.port = 9090
spring.application.name = demoservice
```

  - **application-prod.properties**

```
server.port = 4431
spring.application.name = demoservice
```

Perfil activo

- De manera predeterminada, la aplicación Spring Boot usa el archivo application.properties.

- Para definir el perfil activo:

```
java –jar <app.jar> --spring.profiles.active=<dev|prod>
```

- Usando los dev-tolos:

```
mvn spring-boot:run -Dspring-boot.run.profiles=dev
```

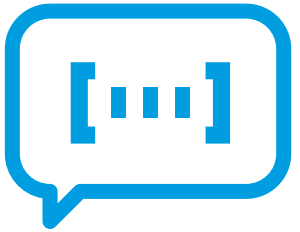
Perfil activo para application.yml

- Podemos mantener las propiedades del perfil activo de Spring en el único archivo application.yml. No es necesario usar el archivo separado como application.properties.
- El delimitador (---) se usa para separar cada perfil en el archivo application.yml.

```
spring:
   application:
      name: demoservice
server:
   port: 8080

---
spring:
   profiles: dev
   application:
      name: demoservice
server:
   port: 9090

---
spring:
   profiles: prod
   application:
      name: demoservice
server:
   port: 4431
```

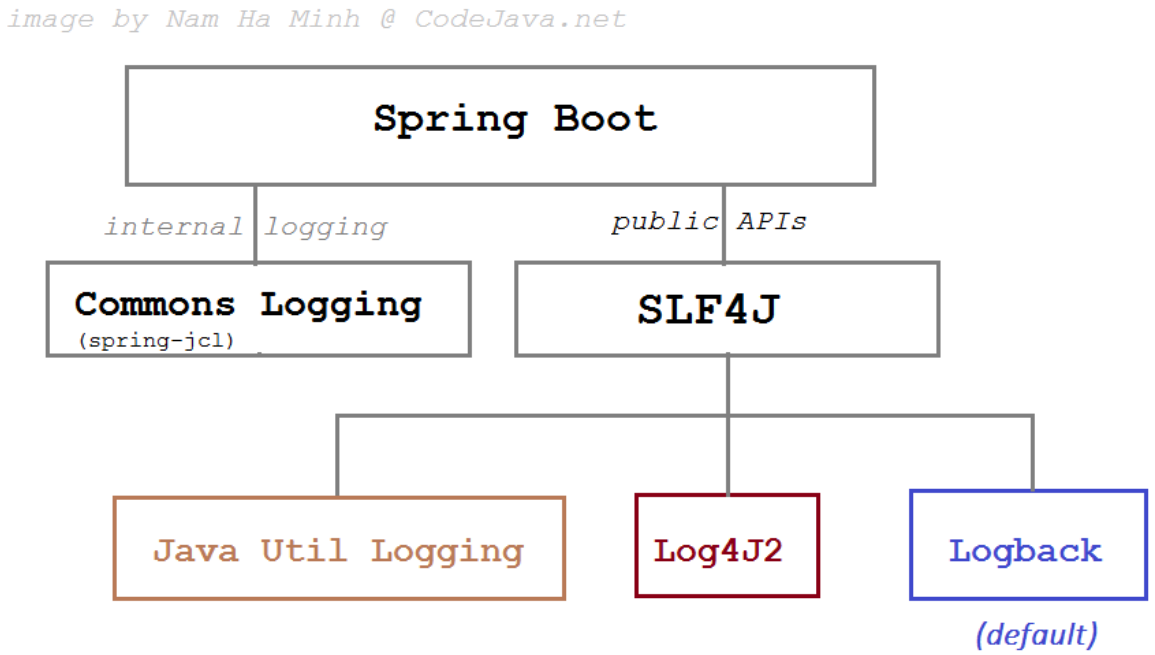
netmind
a bts company

04



Logging

Logging



- Spring Boot utiliza el logging de **Apache Commons** para todos los logs internos.
- Las configuraciones predeterminadas de Spring Boot brindan soporte para el uso de Java Util Logging, Log4j2 y Logback.
- Si se usa Spring Boot Starters, **Logback** proporcionará un buen soporte para el logging.
- Además, Logback también proporciona un buen soporte para Common Logging, Util Logging, Log4J y SLF4J.

Formato del log

- El formato predeterminado de Spring Boot Log:

  - Fecha y hora que da la fecha y hora del registro
  - El nivel de log: INFO, ERROR o WARN
  - Identificador de proceso
  - El --- que es un separador
  - El nombre del thread está encerrado entre corchetes []
  - Nombre del logger que muestra el nombre de la clase de origen
  - El mensaje de log

Configuración del log

- Nivel de depuración
  - De forma predeterminada, los mensajes de “INFO”, “ERROR” y “WARN” se imprimirán en el archivo de log.

```
debug = true
```

- Archivo de salida de registro
  - De forma predeterminada, todos los registros se imprimirán en la ventana de la consola y no en los archivos.

```
logging.path = /var/tmp/
logging.file = /var/tmp/mylog.log
```

- Niveles de log
  - "TRACE", "DEBUG", "INFO", "WARN", "ERROR", "FATAL", "OFF"

```
logging.level.root = WARN
```

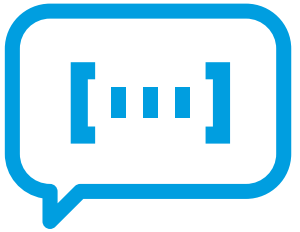
Configuración de Logback

Logback admite la configuración basada en XML para manejar las configuraciones de Spring Boot Log en el archivo en classpath **logback.xml**.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<configuration>
   <appender name = "STDOUT" class = "ch.qos.logback.core.ConsoleAppender">
      <encoder>
         <pattern>[%d{yyyy-MM-dd'T'HH:mm:ss.sss'Z'}] [%C] [%t] [%L] [%-5p] %m%n</pattern>
      </encoder>
   </appender>

   <appender name = "FILE" class = "ch.qos.logback.core.FileAppender">
      <File>/var/tmp/mylog.log</File>
      <encoder>
         <pattern>[%d{yyyy-MM-dd'T'HH:mm:ss.sss'Z'}] [%C] [%t] [%L] [%-5p] %m%n</pattern>
      </encoder>
   </appender>

   <root level = "INFO">
      <appender-ref ref = "FILE"/>
      <appender-ref ref = "STDOUT"/>
   </root>
</configuration>
```

Usar el logger de slf4j

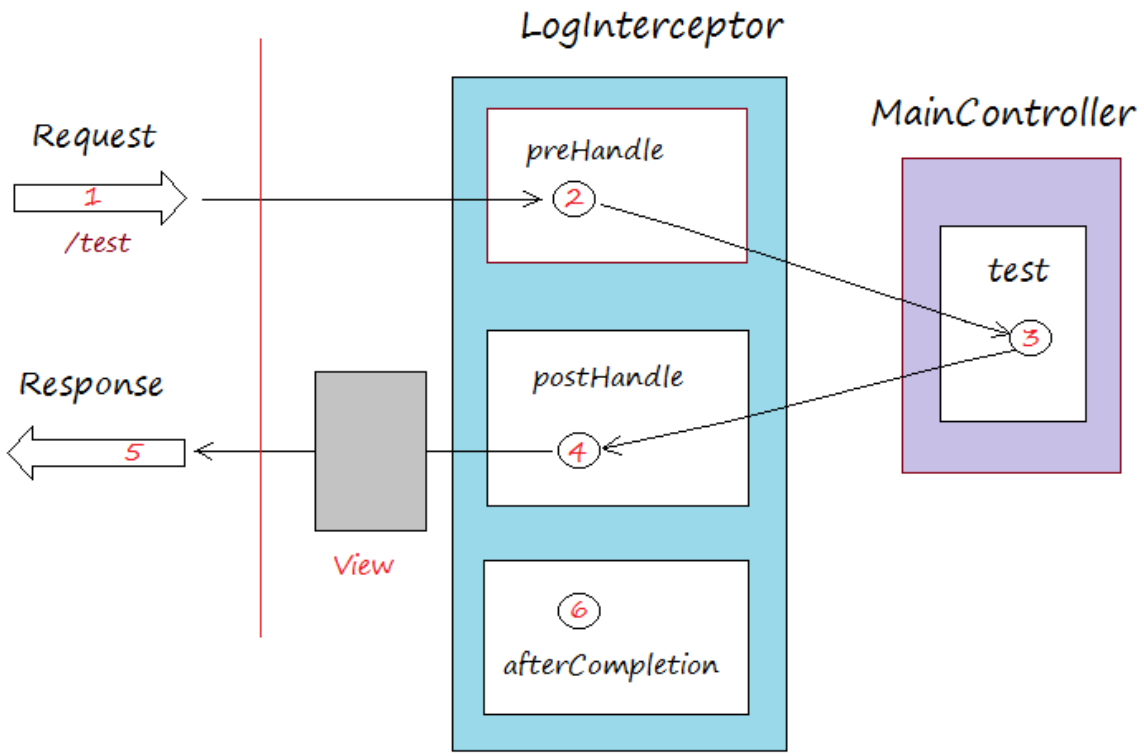
```
package com.netmind.demo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
   private static final Logger logger = LoggerFactory.getLogger(DemoApplication.class);

   public static void main(String[] args) {
      logger.info("this is a info message");
      logger.warn("this is a warn message");
      logger.error("this is a error message");
      SpringApplication.run(DemoApplication.class, args);
   }
}
```



Interceptor

05

Interceptor



- En Spring, cuando se envía una solicitud a un controlador Spring, pasará por Interceptores (0 o más) antes de ser procesada por el Controlador.
- Spring Interceptor es un concepto bastante similar a Servlet Filter.
- Spring Interceptor solo se aplica a las solicitudes que se envían a un controlador.

Interceptor

- Se puede usar Interceptor en Spring Boot para realizar operaciones en las siguientes situaciones:
  - Antes de enviar la solicitud al controlador
  - Antes de enviar la respuesta al cliente

- Por ejemplo, puede usar un interceptor para agregar el encabezado de solicitud antes de enviar la solicitud al controlador y agregar el encabezado de respuesta antes de enviar la respuesta al cliente.

Implementar un Interceptor

- Se debe crear la clase **@Component** que lo admita y debe implementar la interfaz **HandlerInterceptor** .
- Los siguientes son los tres métodos que debe conocer mientras trabaja en Interceptores:
  - **preHandle()**: se utiliza para realizar operaciones antes de enviar la solicitud al controlador. Este método debe devolver verdadero para devolver la respuesta al cliente.
  - **postHandle()**: se utiliza para realizar operaciones antes de enviar la respuesta al cliente.
  - **afterCompletion()**: se utiliza para realizar operaciones después de completar la solicitud y la respuesta.

Implementar un Interceptor

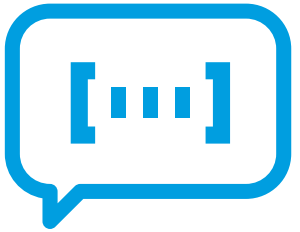
```
@Component
public class ProductServiceInterceptor implements HandlerInterceptor {
   @Override
   public boolean preHandle(
      HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

      return true;
   }
   @Override
   public void postHandle(
      HttpServletRequest request, HttpServletResponse response, Object handler,
      ModelAndView modelAndView) throws Exception {}

   @Override
   public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
      Object handler, Exception exception) throws Exception {}
}
```

Registrar un Interceptor

- Se tiene que registrar un Interceptor con InterceptorRegistry usando **WebMvcConfigurerAdapter**.

```
@Component
public class ProductServiceInterceptorAppConfig implements WebMvcConfigurer {
   @Autowired
   ProductServiceInterceptor productServiceInterceptor;

   @Override
   public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(productServiceInterceptor);
   }
}
```

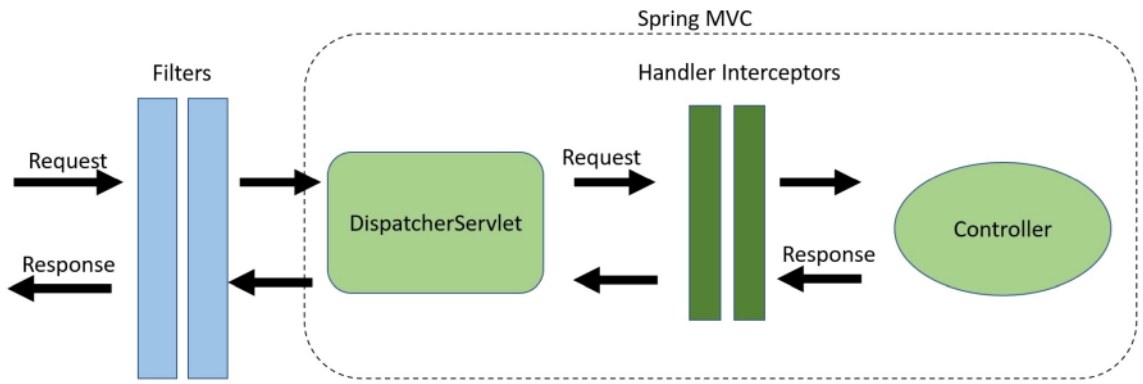
netmind
a bts company



Servlet filter

06

Servlet filters



- Un filtro es un objeto que se utiliza para interceptar las solicitudes y respuestas HTTP de una aplicación.
- Al usar el filtro, podemos realizar dos operaciones en dos pasos:
  - Antes de enviar la solicitud al controlador
  - Antes de enviar una respuesta al cliente.

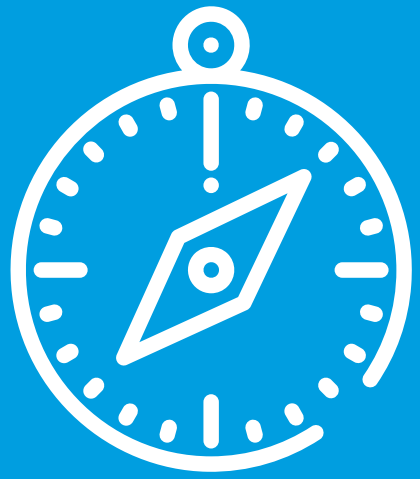
Implementar un filtro

```
@Component
public class SimpleFilter implements Filter {
   @Override
   public void destroy() {}

   @Override
   public void doFilter (ServletRequest request, ServletResponse response, FilterChain filterchain)
      throws IOException, ServletException {
         …
         filterchain.doFilter(request, response);
   }

   @Override
   public void init(FilterConfig filterconfig) throws ServletException {}
}
```

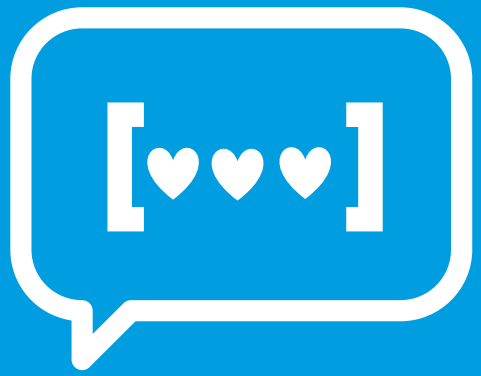
Cuando usar filtros o interceptors

- Los **filtros** interceptan solicitudes antes de que lleguen a DispatcherServlet, lo que los hace ideales para tareas de grano grueso como:
  - Autenticación
  - Logging y auditoría
  - Compresión de imágenes y datos
  - Cualquier funcionalidad que queramos desacoplar de Spring MVC

- Los **interceptors**, por otro lado, intercepta solicitudes entre DispatcherServlet y nuestros Controladores. Esto se hace dentro de Spring MVC, proporcionando acceso a los objetos Handler y ModelAndView. Esto reduce la duplicación y permite una funcionalidad más detallada, como:
  - Manejo de tareas transversales como el logging de aplicaciones
  - Comprobaciones de autorización detalladas
  - Manipulación del contexto o modelo Spring



Next steps



We would like to know your opinion!

Please, let us know what you think about the content.
From Netmind we want to say thank you, we appreciate time and effort you have taking in answering all of that is important in order to improve our training plans so that you will always be satisfied with having chosen us
quality@netmind.es

netmind
a bts company

Thanks!

Follow us: