Java Spring

Introducción a Spring Boot

Spring Boot es una herramienta de desarrollo de código abierto diseñado para simplificar y acelerar la creación de aplicaciones basadas en Java. Se basa en ek Spring Framework y se destaca por su capacidad para crear aplicaciones independientes y se centra en la eficiencia y la productividad.

Características

- ***Aplicaciones Independientes***. Permite la creación de aplicaciones que no están vinculadas a una plataforma específica, facilitando su ejecución en dispositivos locales sin conexión a internet.
- ***Servidores Integrados***. Simplifica el despliegue al integrar servidores como Tomcat, Jetty o Undertow directamente, eliminando la necesidad de configuraciones detalladas.
- ***Enfoque Con Sesgo***. Adopta un enfoque con sesgo que proporciona configuraciones de compilación predefinidas, reduciendo la carga sobre los desarrolladores.
- ***Configuración Automática***. Automatiza la configuración de Spring y otras bibliotecas de terceros, eliminando la necesidad de configuración manual y posibles errores.
- ***Características Listas para Producción***. Ofrece características listas para producción, como métricas y comprobaciones de estado, para facilitar la creación de aplicaciones robustas.

Spring Framework y Spring Boot

Spring Framework y Spring Boot son dos herramientas para el desarrollo de aplicaciones Java, pero se distinguen por sus propósitos y enfoques.

Mientras que Spring Framework ofrece un marco de trabajo integral para aplicaciones empresariales con configuración detallada, Spring Boot se centra en simplificar el desarrollo y la implementación, adoptando el principio de "convención sobre la configuración" y proporcionando configuraciones predeterminadas.

En términos de ***configuración***, Spring Framework requiere configuración extensa a través de archivos XML o anotaciones, otorgando al desarrollador un mayor control. Por otro lado, Spring Boot sigue el enfoque de "arranque rápido" con configuraciones predeterminadas y convenciones que facilitan la creación de aplicaciones con menos esfuerzo.

En cuanto a la ***embebibilidad*** de servidores, Spring Framework se integra típicamente con servidores de aplicaciones externos, mientras que Spring Boot permite la embebibilidad de servidores, facilitando la implementación y distribución.

En términos de ***dependencias*** y ***construcción***, Spring Framework requiere que el desarrollador gestione manualmente dependencias y construcción con herramientas como Maven o Gradle. En contraste, Spring Boot introduce "starters" y herramientas como Spring Initializr para simplificar la gestión de dependencias y la generación rápida de proyectos.

Además, Spring Boot incluye ***módulos*** de actuadores para proporcionar información operativa sobre la aplicación en ejecución, como estadísticas y datos de entorno, características que no están presentes de manera predeterminada en Spring Framework.

En conclusión, mientras ***Spring Framework*** es un marco integral y configurable, ***Spring Boot*** se orienta hacia la simplificación y aceleración del desarrollo, ofreciendo configuraciones predeterminadas sensatas para crear aplicaciones listas para la producción.

Spring Data

Spring Data es una herramienta que simplifica significativamente el acceso a datos, bases de datos relacionales y no relacionales, servicios de datos en la nube a través de diversos proyectos que son específicos de una base de datos determinada.

En ***bases de datos relacionales***, Spring Data simplifica la interacción con JDBC y ORM permitiendo a los desarrolladores trabajar con representaciones orientadas a objetos de los datos sin escribir consultas SQL tediosas. Ofrece soporte para frameworks ORM como Hibernate, JPA y MyBatis, mejorando la legibilidad y mantenibilidad del código al facilitar la creación de consultas y manipulación de datos.

Por otro aldo, con ***bases de datos no relacionales*** como MongoDB, Cassandra y Neo4j, Spring Data facilita el trabajo ofrece adaptadores y abstracciones. Esto permite a los desarrolladores cambiar entre bases de datos sin modificar considerablemente el código, brindando flexibilidad y adaptabilidad en entornos de desarrollo dinámicos.

Repositorios

Los repositorios en Spring Data son esenciales para realizar eficientemente operaciones CRUD (Create, Read, Update, Delete). Se definen mediante interfaces que extienden las interfaces base, eliminando código repetitivo. Estos repositorios facilitan operaciones básicas como guardar, recuperar, actualizar y eliminar entidades en la base de datos.

En Spring Data, los desarrolladores pueden realizar consultas personalizadas de forma sencilla mediante la convención de nombres de métodos. Spring Data genera automáticamente consultas si la interfaz del repositorio sigue ciertas convenciones de nomenclatura, reduciendo así la cantidad de código necesario y mejorando la legibilidad.

Además, Spring Data brinda soporte para consultas dinámicas. Los desarrolladores pueden definir métodos en el repositorio con nombres específicos y parámetros, permitiendo la generación de consultas en tiempo de ejecución.

Módulos principales

- ***Spring Data Commons***: Contiene los conceptos fundamentales de Spring que son la base para todos los módulos de Spring Data.
- ***Spring Data JDBC***: Proporciona soporte de repositorio Spring Data para JDBC.
- ***Spring Data R2DBC***: Proporciona soporte de repositorio Spring Data para R2DBC.
- ***Spring Data JPA***: Proporciona soporte de repositorio Spring Data para JPA.
- ***Spring Data KeyValue***: Incluye repositorios basados en mapas y SPIs para construir fácilmente un módulo de Spring Data para almacenes clave-valor.
- ***Spring Data LDAP***: Proporciona soporte de repositorio Spring Data para Spring LDAP.
- ***Spring Data MongoDB***: Proporciona soporte basado en Spring, objetos-documento y repositorios para MongoDB.
- ***Spring Data Redis***: Facilita la configuración sencilla y acceso a Redis desde aplicaciones Spring.
- ***Spring Data REST***: Exporta repositorios de Spring Data como recursos RESTful impulsados por hipertexto.

- ***Spring Data for Apache Cassandra***: Facilita la configuración sencilla y acceso a Apache Cassandra para aplicaciones Spring orientadas a datos a gran escala y altamente disponibles.
- ***Spring Data for Apache Geode***: Facilita la configuración sencilla y acceso a Apache Geode para aplicaciones Spring altamente consistentes, de baja latencia y orientadas a datos.

Construcción de una API

Spring Boot, junto con Spring Data, facilita enormemente la implementación de servicios RESTFul en entornos Java.

Definición de un servicio RESTFul

Un servicio RESTful es una interfaz que utiliza el protocolo HTTP para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en recursos. Los recursos, representados por URIs, son manipulados a través de los métodos HTTP estándar, como GET, POST, PUT y DELETE. Esta arquitectura basada en estándares permite una comunicación eficiente y sin estado entre clientes y servidores.

Métodos HTTP

- **GET**. Recupera información sobre el recurso identificado por la URI. Se utiliza para lecturas y consultas.

```
@GetMapping("/recurso/{id}")
public ResponseEntity<Recurso> obtenerRecurso(@PathVariable Long id) {
    // Lógica para obtener el recurso con el ID proporcionado
}
```

- **POST**. Crea un nuevo recurso con los datos proporcionados en el cuerpo de la solicitud.

```
@PostMapping("/recurso")
public ResponseEntity<Recurso> crearRecurso(@RequestBody Recurso nuevoRecurso) {
    // Lógica para crear un nuevo recurso
}
```

- **PUT**. Actualiza completamente el recurso identificado por la URI con los datos proporcionados en el cuerpo de la solicitud.

```
@PutMapping("/recurso/{id}")
public ResponseEntity<Recurso> actualizarRecurso(
        @PathVariable Long id,
        @RequestBody Recurso recursoActualizado) {
    // Lógica para actualizar el recurso con el ID proporcionado
}
```

- **PATCH**. Actualiza parcialmente el recurso identificado por la URI con los datos proporcionados en el cuerpo de la solicitud.

```
@PatchMapping("/recurso/{id}")
public ResponseEntity<Recurso> actualizarParcialmenteRecurso(
        @PathVariable Long id,
        @RequestBody Map<String, Object> campos) {
    // Lógica para actualizar parcialmente el recurso con el ID proporcionado
}
```

- **DELETE**. Elimina el recurso identificado por la URI.

```
@DeleteMapping("/recurso/{id}")
public ResponseEntity<Void> eliminarRecurso(@PathVariable Long id) {
    // Lógica para eliminar el recurso con el ID proporcionado
}
```

- **HEAD**. Similar a GET, pero utilizado para obtener solo los encabezados de respuesta sin el cuerpo de la respuesta.

```
@RequestMapping(value = "/recurso/{id}", method = RequestMethod.HEAD)
public ResponseEntity<Void> obtenerEncabezados(@PathVariable Long id) {
    // Lógica para obtener solo los encabezados del recurso con el
    // ID proporcionado
}
```

- **OPTIONS**: Proporciona información sobre las opciones de comunicación para el recurso identificado por la URI.

```java
@RequestMapping(value = "/recurso/{id}", method = RequestMethod.OPTIONS)
public ResponseEntity<Void> opcionesDeComunicacion(@PathVariable Long id) {
    // Lógica para manejar las opciones de comunicación
    // para el recurso con el ID proporcionado
}
```

Además de estas operaciones, existen operaciones adicionales que son extensiones o variaciones de los métodos HTTPS estándar que fueron adaptadas para usos específicos.

- **Listar Todos los Recursos** (GET): Este método extiende el estándar GET para recuperar una lista completa de todos los recursos disponibles en el servidor.
- **Buscar por Criterios** (GET): Añade parámetros de consulta al método GET para permitir búsquedas filtradas y personalizadas de recursos.
- **Paginación** (GET): Introduce parámetros, como "página" y "tamaño", para gestionar grandes conjuntos de datos dividiéndolos en páginas.
- **Ordenar** (GET): Permite la obtención de recursos ordenados según algún criterio específico proporcionado como parámetro.
- **Manejo de Errores Personalizados**: Implementa métodos de manejo de excepciones (como @ExceptionHandler) para personalizar respuestas de error según el tipo de excepción.
- **Enviar Datos como Formulario** (POST): Extiende el estándar POST para admitir el envío de datos en el formato de formulario HTML, útil en aplicaciones web.
- **Operaciones Personalizadas** (POST/PUT/DELETE): Introduce rutas y operaciones específicas del dominio, como publicar, modificar o eliminar un recurso, que no se alinean directamente con los métodos HTTP estándar.