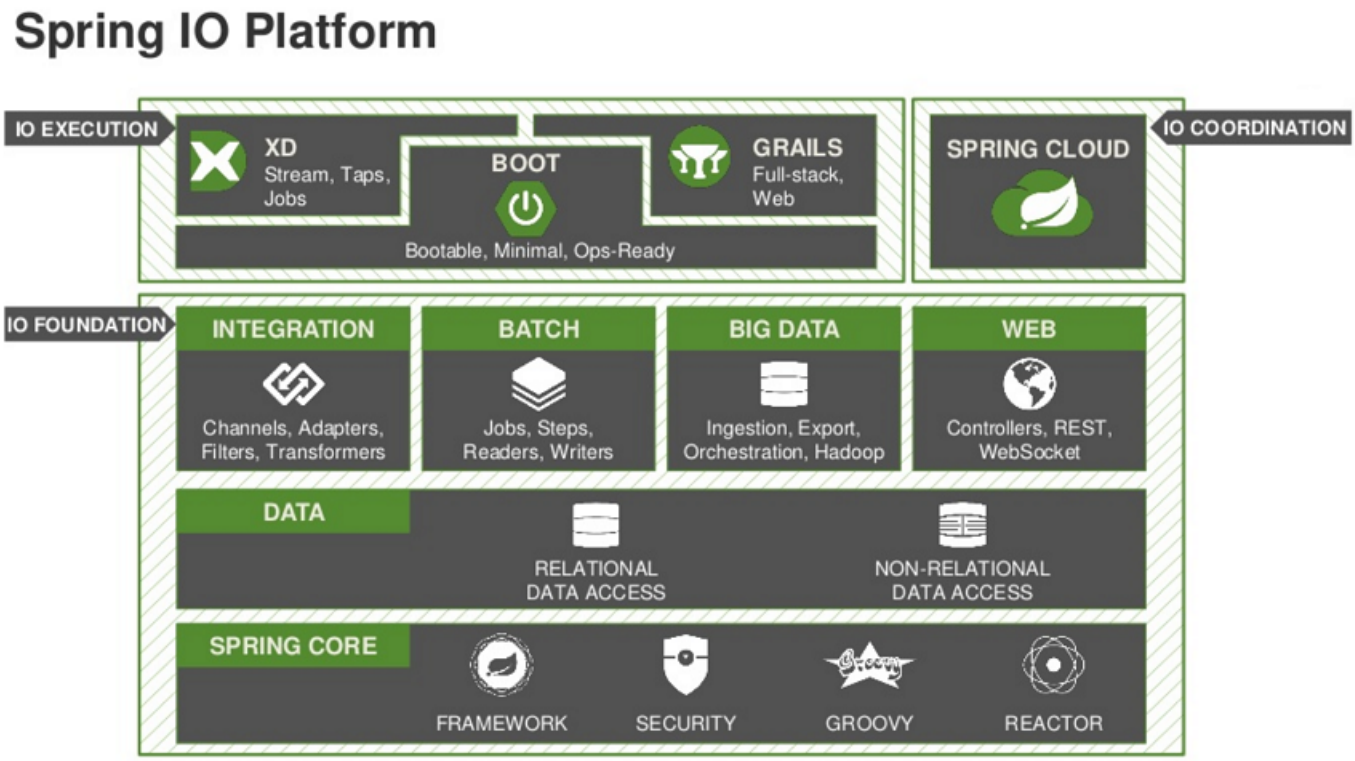
Artículo
Dani Fibla · 14 sep, 2021 Lectura de 5 min

Open Exchange

Ejemplo: cómo utilizar Java + SpringBoot + Hibernate y la base de datos de IRIS para crear una API REST

Spring Boot es el *framework* de Java más utilizado para crear APIs REST y microservicios. Se puede utilizar para implementar sitios webs o webs ejecutables o aplicaciones de escritorio independientes, donde la aplicación y otras dependencias se empaquetan juntas. Springboot permite realizar muchas funciones, como:



Nota: para saber más sobre SpringBoot, consulta el sitio oficial: https://spring.io/quickstart

Para crear una aplicación web API , con uno o más microservicios, puedes utilizar Spring IDE para Eclipse o VSCode, y utilizar un asistente para configurar las tecnologías anteriores que se utilizarán en tu aplicación, mira:

Seleccionas las tecnologías y creas el proyecto. Todas las tecnologías se importarán utilizando Maven. Esto es como un ZPM visual.

El proyecto creado tiene una clase para subir la aplicación con todo lo que necesitas dentro de ella (servidor web y de aplicaciones, y todas las dependencias, concepto de microservicio).

El código fuente completo de este proyecto se encuentra en el proyecto de Open Exchange: https://openexchange.intersystems.com/package/springboot-iris-crud.

Lo primero que hay que hacer es configurar el controlador de JDBC en IRIS y el soporte IRIS Hibernate. Para ello, copia los archivos jar en la carpeta de recursos:

Abre pom.xml para configurar las dependencias de estos archivos jar.:

```
<dependency>
            <groupId>com.intersystems</groupId>
            <artifactId>intersystems-jdbc</artifactId>
            <version>3.2.0</version>
            <scope>system</scope>
            <systemPath>${project.basedir}/src/main/resources/intersystems-
jdbc-3.2.0.jar</systemPath>
</dependency>
 <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-iris</artifactId>
            <version>1.0.0</version>
            <scope>system</scope>
            <systemPath>${project.basedir}/src/main/resources/hibernate-
iris-1.0.0.jar</systemPath>
 </dependency>
```

Ahora puedes configurar tu conexión con la base de datos de IRIS en application.properties, de la siguiente forma:

```
spring.datasource.username=_SYSTEM
spring.datasource.url=jdbc:IRIS://iris:1972/USER
spring.datasource.password=SYS
spring.jpa.properties.hibernate.default_schema=dc_Sample
#spring.jpa.hibernate.ddl-auto=update
spring.datasource.driver-class-name=com.intersystems.jdbc.IRISDriver
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults = false
spring.jpa.database-platform=org.hibernate.dialect.InterSystemsIRISDialect
spring.datasource.sql-script-encoding=utf-8
server.tomcat.relaxed-query-chars=|,{,},[,]
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=true
```

El siguiente paso consiste en crear un mapeo de la clase Persistent de Java en una tabla de IRIS:

```
package community.intersystems.springboot.app.model;

import java.io.Serializable;
import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

import com.fasterxml.jackson.annotation.JsonFormat;

@Entity
@Table(name = "Product")
public class Product implements Serializable {
```

```
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String description;

    private Double height;

    private Double width;

    private Double weight;

    @Column(name="releasedate")
    @Temporal(TemporalType.DATE)
    @JsonFormat(pattern = "dd/MM/yyyy")
    private Date releaseDate;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Double getHeight() {
        return height;
    }

    public void setHeight(Double height) {
        this.height = height;
    }

    public Double getWidth() {
        return width;
    }

    public void setWidth(Double width) {
        this.width = width;
```

```
    }

    public Double getWeight() {
        return weight;
    }

    public void setWeight(Double weight) {
        this.weight = weight;
    }

    public Date getReleaseDate() {
        return releaseDate;
    }

    public void setReleaseDate(Date releaseDate) {
        this.releaseDate = releaseDate;
    }

}
```

El último paso es crear una interfaz en el Repositorio para exponer tu clase persistente como un servicio REST (en el patrón HATEOAS). Con operaciones CRUD, no necesitas escribir esto, solo:

package community.intersystems.springboot.app.repository;

```
import org.springframework.data.jpa.repository.JpaRepository;

import community.intersystems.springboot.app.model.Product;

public interface ProductRepository extends JpaRepository&lt;Product, Long> {

}
```

Ahora, ejecutas tu aplicación como una aplicación Spring Boot:

Espera al servidor interno y abre localhost:8080. Spring boot abrirá un navegador API REST HAL. Mira este gif animado:

Consulta más detalles en mi aplicación de muestra. Lo empaqueto todo junto en un proyecto Docker con 2 servicios: IRIS y SpringBoot.

El patrón HATEOAS es muy agradable para las API URL , la ruta y la navegación. Puedes conocer más detalles en: https://en.wikipedia.org/wiki/HATEOAS

¡Espero que os resulte útil!

#Java #InterSystems IRIS
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/ejemplo-c%C3%B3mo-utilizar-java-springboot-hibernate-y-la-base-de-datos-de-iris-para-crear-una-api