REST Nested Resources y su utilidad

27 septiembre, 2019 Por Cecilio Álvarez Caules — Deja un comentario

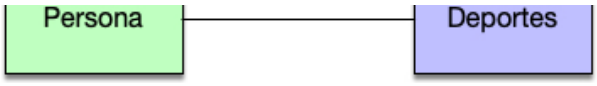
El uso **REST Nested Resources** es cada día más necesario cuando construimos arquitecturas REST . En muchos casos estas arquitecturas pueden funcionar de una forma razonable utilizando los clásicos Recursos y verbos HTTP . Ahora bien el uso de recursos REST standard no siempre solventa todos los problemas que este tipo de Arquitecturas puede generar.
Imaginemos que estamos ante una situación en la que tenemos dos Recursos REST Personas y Deportes. Para ello el primer paso es crear las clases y relacionarlas , más adelante las convertiremos con Spring Boot en recursos.

```
1.  package com.arquitecturajava.embebidos;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  import com.fasterxml.jackson.annotation.JsonInclude;
7.  import com.fasterxml.jackson.annotation.JsonInclude.Include;
8.
9.  public class Persona {
10.
11.   private String nombre;
12.   private String apellido;
13.   private int edad;
14.   @JsonInclude(Include.NON_EMPTY)
15.   private List<Deporte> deportes = new ArrayList<>();
16.
17.   public List<Deporte> getDeportes() {
18.     return deportes;
19.   }
20.
21.   public void setDeportes(List<Deporte> deportes) {
22.     this.deportes = deportes;
23.   }
24.
25.   public void addDeporte(Deporte deporte) {
26.
27.     this.deportes.add(deporte);
28.   }
29.
```

```
39.       this(p.getNombre(), p.getApellido(), p.getEdad());
40.     }
41.
42.     public String getNombre() {
43.       return nombre;
44.     }
45.
46.     public void setNombre(String nombre) {
47.       this.nombre = nombre;
48.     }
49.
50.     public String getApellido() {
51.       return apellido;
52.     }
53.
54.     public void setApellido(String apellido) {
55.       this.apellido = apellido;
56.     }
57.
58.     public int getEdad() {
59.       return edad;
60.     }
61.
62.     public void setEdad(int edad) {
63.       this.edad = edad;
64.     }
65.   }
```

Podemos observar que estamos ante una Persona que incluye una lista de Deportes . En nuestro caso la clase Deporte será muy muy básica y solo contendrá el nombre del Deporte que practicamos.

```
1.  package com.arquitecturajava.embebidos;
2.
3.  import com.fasterxml.jackson.annotation.JsonInclude;
4.  import com.fasterxml.jackson.annotation.JsonInclude.Include;
5.
6.  @JsonInclude(Include.NON_NULL)
7.  public class Deporte {
8.
9.    private String nombre;
10.
11.   public String getNombre() {
12.     return nombre;
13.   }
14.
15.   public void setNombre(String nombre) {
16.     this.nombre = nombre;
17.   }
18.
19.   public Deporte(String nombre) {
20.     super();
21.     this.nombre = nombre;
22.   }
23.
24. }
```



Es momento de construir un ejemplo con Spring Boot que nos publique la información de estos conceptos como API REST. Para ello como siempre usaremos Spring Initializr y nos construiremos un proyecto Web.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.8.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.arquitecturajava</groupId>
  <artifactId>embebidos</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>embebidos</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <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>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
```

Creaomos RestController que habilite la posibilidad de seleccionar un listado de Personas.

```
package com.arquitecturajava.embebidos;

import java.util.ArrayList;
```

```
13.
14.  @RestController
15.  public class PersonaController {
16.
17.
18.    static List<Persona> lista=new ArrayList<>();
19.
20.    static {
21.
22.      Persona p1=new Persona("juan","sanchez",20);
23.      p1.addDeporte(new Deporte ("padel"));
24.      lista.add(p1);
25.      Persona p2= new Persona("ana","gomez",30);
26.      p2.addDeporte(new Deporte ("natacion"));
27.      lista.add(p2);
28.      Persona p3= new Persona("juan","sanchez",20);
29.      p3.addDeporte(new Deporte ("futbol"));
30.      lista.add(p3);
31.    }
32.
33.
34.    @GetMapping("/personas")
35.    public List<Persona> personas() {
36.
37.      return lista.stream().map(Persona::new).collect(Collectors.toList());
38.    }
39.
40.    @GetMapping("/personas/{nombre}")
41.    public Persona persona(@PathVariable String nombre) {
42.
43.      Optional<Persona> oPersona=lista.stream().filter(p->p.getNombre().equals(nombre)).findFirst();
44.
45.      if (oPersona.isPresent()) {
46.        return oPersona.get();
47.      }
48.      return null;
49.    }
50.
51.    @GetMapping("/personas/{nombre}/deportes")
52.    public ResponseEntity<List<Deporte>> deportesPersona(@PathVariable String nombre) {
53.
54.      Optional<Persona> oPersona=lista.stream().filter(p->p.getNombre().equals(nombre)).findFirst();
55.
56.      if (oPersona.isPresent()) {
57.        return new ResponseEntity<List<Deporte>>(oPersona.get().getDeportes(), HttpStatus.OK);
58.      }
59.      return new ResponseEntity<>(new ArrayList<Deporte>(),HttpStatus.NOT_FOUND);
60.    }
61.
62.    @GetMapping("/personas/-/deportes")
63.    public List<Persona> personasConDeportes() {
64.
65.
66.      return lista;
67.    }
```

REST Resources y Agregaciones

```
[
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20
  },
  {
    "nombre": "ana",
    "apellido": "gomez",
    "edad": 30
  },
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20
  }
]
```

Esta es la url más habitual y nos devuelve **una lista de Recursos**. El segundo caso más habitual es cuando nosotros queremos localizar un único recurso y pasamos a este URL como información adicional el identificador de este.

```
// 20190927154430
// http://localhost:8080/personas/juan

{
  "nombre": "juan",
  "apellido": "sanchez",
  "edad": 20,
  "deportes": [
    {
      "nombre": "padel"
    }
  ]
}
```

Como podemos ver obtenemos la persona y su deporte (esto último sería opcional) . Es momento de acceder a los Deportes de cada una de las Personas @GetMapping("/personas/{nombre}/deportes") . Esto nos devuelve vía un agregado los deportes de una persona en concreto.

```
// 20190927154630
// http://localhost:8080/personas/juan/deportes

[
  {
    "nombre": "padel"
  }
]
```

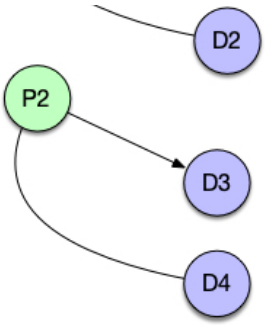
REST Nested Resources

grafo le tenemos disponible es cuestión de publicarlo vía una url que informe sobre la relación . Es el caso de @GetMapping("/personas/-/deportes") . Esta url nos indica que son las personas con sus deportes.

```
// 20190927194617
// http://localhost:8080/personas/-/deportes

[
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20,
    "deportes": [
      {
        "nombre": "padel"
      }
    ]
  },
  {
    "nombre": "ana",
    "apellido": "gomez",
    "edad": 30,
    "deportes": [
      {
        "nombre": "natacion"
      }
    ]
  },
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20,
    "deportes": [
      {
        "nombre": "futbol"
      }
    ]
  }
]
```

Así pues habremos abordado con garantías el concepto de REST Nested Resources lo que nos permitirá reducir las peticiones a nuestro API REST. Ya que una sola llamada devuelve el grafo de objetos completo.



Otros artículos relacionados

1. REST DTO y JSON Arquitecturas Web y objetos
2. JSON API y Arquitecturas REST
3. ¿ Que es REST ?
4. https://jsonapi.org/

Descargar PDF

Archivado en: Sin categoría

Deja un comentario

Tu dirección de correo electrónico no será publicada.

Comentario

Correo electrónico

Web

He leído y acepto la Política de privacidad de esta web *

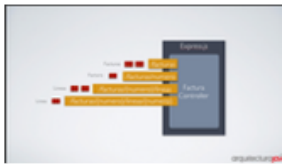
PUBLICAR COMENTARIO

Este sitio usa Akismet para reducir el spam. Aprende cómo se procesan los datos de tus comentarios.

Cursos Gratuitos

Introduccion Spring Boot

Introducción TypeScript



Introduccion JPA

Java Herencia

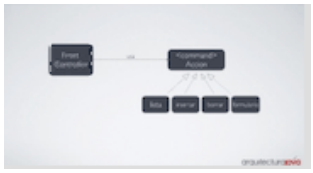
Java JDBC

Servlets

Java APIS Core

Java Web

Pack Java Core



Arquitectura Java Solida con Spring

CONTACTO

contacto@arquitecturajava.com