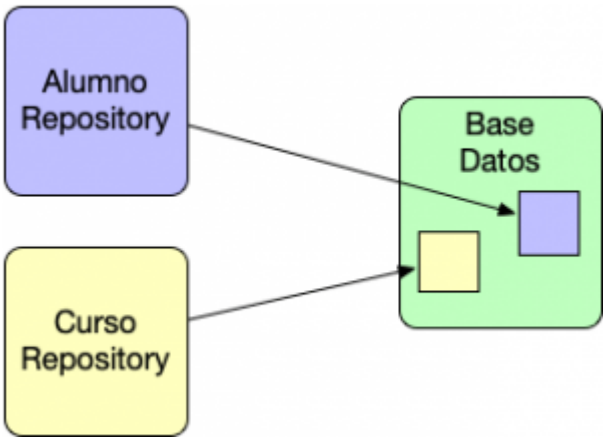
Spring @Service es una de las anotaciones más habituales de Spring Framework . Se usa para construir una clase de Servicio que habitualmente se conecta a varios repositorios y agrupa su funcionalidad. Es decir por ejemplo si disponemos de dos Repositorios uno con Profesores y otro con Alumnos es muy común disponer de una clase Fachada de tipo ServicioCurso que aglutine la funcionalidad de las dos capas de Repositorio . Vamos a construir un ejemplo sencillo a través de Spring Boot.

```
package com.arquitecturajava.app1;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

@Repository
public class AlumnoRepository {

  private static List<Alumno> alumnos= new ArrayList<Alumno>();
  static {
    alumnos.add(new Alumno("pedro",20));
    alumnos.add(new Alumno("angel",30));
    alumnos.add(new Alumno("ana",50));
  }
  public List<Alumno> buscarTodos() {
    return alumnos;
  }
  public void insertar(Alumno alumno) {
    alumnos.add(alumno);
  }
}
```

```
package com.arquitecturajava.app1;
```

```
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;
@Repository
public class CursoRepository {

  private static List<Curso> cursos= new ArrayList<Curso>();
  static {
    cursos.add(new Curso("java",20));
    cursos.add(new Curso("php",30));
    cursos.add(new Curso("python",50));
  }
  public List<Curso> buscarTodos() {
    return cursos;
  }
}
```

Acabo de generar dos repositorios diferentes cada uno basado en una clase (Alumno y Curso) .



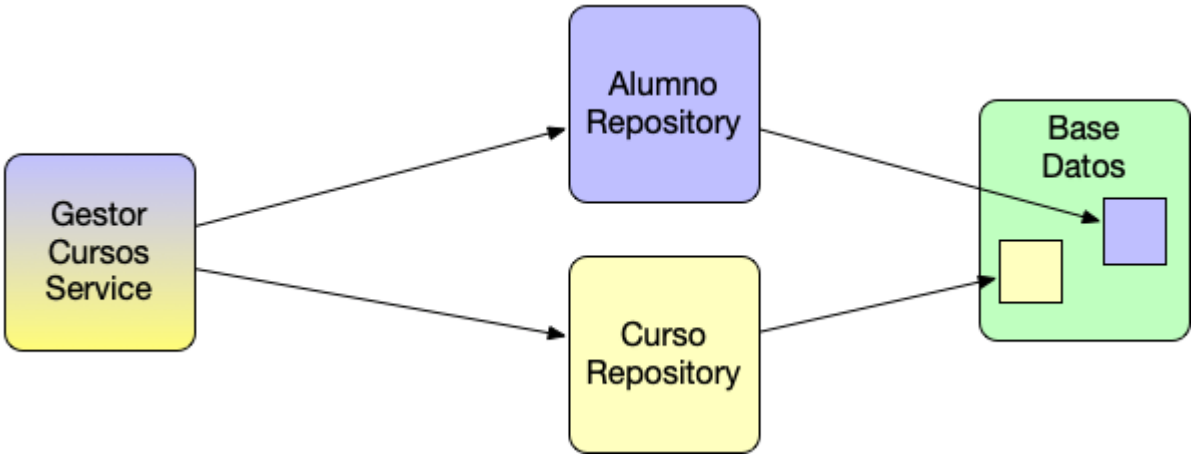
spring repositories

Spring @Autowired

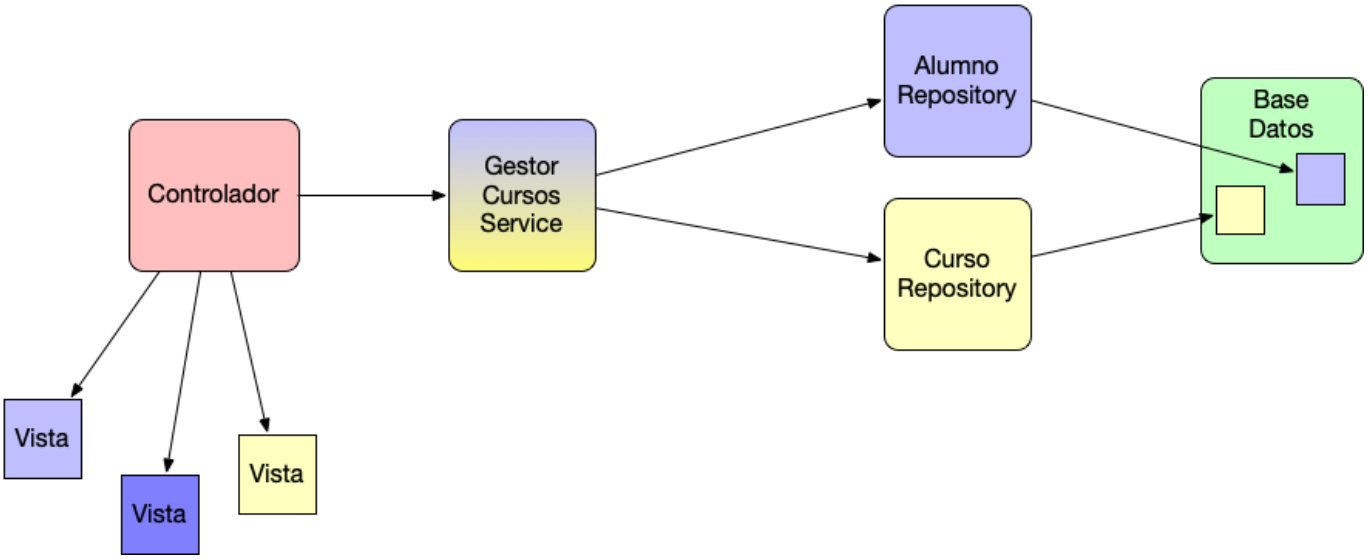
Es momento de generar una clase de Servicio que aglutine los diferentes métodos que estos dos repositorios poseen

```
package com.arquitecturajava.app1;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
@Service
public class GestorCursosService {

  @Autowired
  AlumnoRepository repoAlumno;
  @Autowired
  CursoRepository repoCurso;
  public void insertarAlumno(Alumno alumno) {
    repoAlumno.insertar(alumno);
  }
  public List<Alumno> buscarTodosAlumnos() {
    return repoAlumno.buscarTodos();
  }
  public List<Curso> buscarTodosCursos() {
    return repoCurso.buscarTodos();
  }
}
```

Esta clase simplemente aglutina los métodos que estaban separados en dos clases inyectando las dependencias con autowired .



De tal forma que luego al Controlador de la aplicación le resulte más sencillo procesarlo y gestionar los diferentes métodos .



Por ejemplo en este caso imprimirá una lista de Alumnos en una vista de Thymeleaf.

```
package com.arquitecturajava.app1;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@Controller
public class AlumnoController {

  @Autowired
  GestorCursosService servicio;
  @RequestMapping("/listaalumnos")
  public String listaAlumnos(Model modelo) {
    modelo.addAttribute("listaalumnos",servicio.buscarTodosAlumnos());
    return "listaalumnos";
  }
}
```

Como se puede observar el Controlador hace uso del servicio y pasa una lista de objetos a la vista. El siguiente paso es diseñar la vista que será la encargada de imprimir los objetos.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

  <ul th:each="alumno : ${listaalumnos}">
    <li th:text="${alumno.nombre}"></li>
    <li th:text="${alumno.edad}"></li>
  </ul>
</body>
</html>
```

Ahora podemos ver el resultado:

- pedro
- 20

- angel
- 30

- ana
- 50

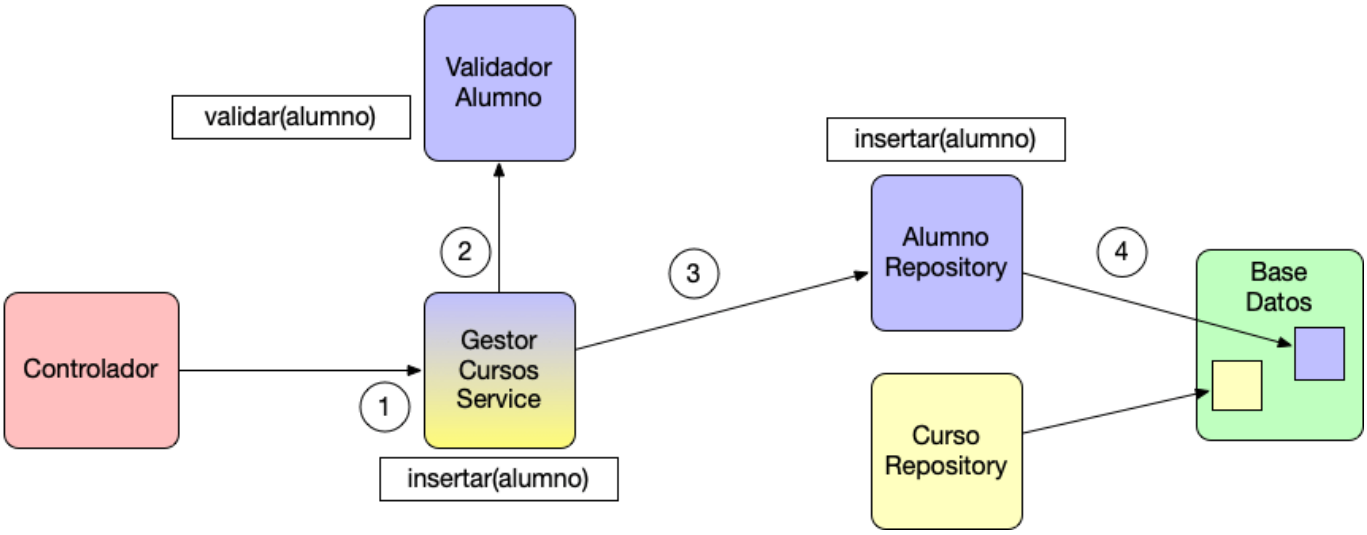
Acabamos de usar el patrón Servicio para añadir una lista de objetos a una vista. Ahora bien este patrón de diseño hace las funciones de Fachada y aglutina un conjunto de métodos que se encuentran en las clases de Repositorio . Es decir muchas veces simplemente realiza unas tareas de delegación. En más de una ocasión me he encontrado con personas que prefieren no usar el patrón de Servicio y apoyarse directamente en el Repositorio por temas de simplicidad. En estos casos el Controlador y el Repositorio trabajan de forma directa entre ellos. ¿Es esto lo correcto?

Spring @Service Repositorios y buenas prácticas (Contenido Premium)

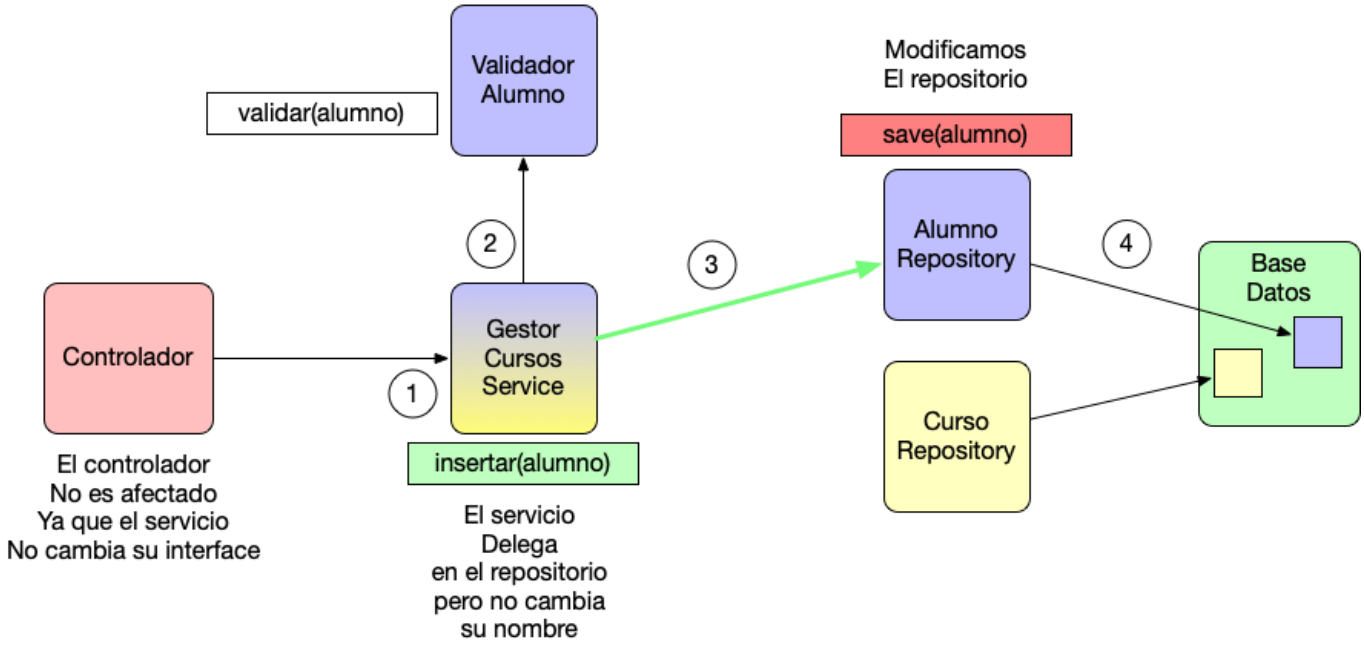
[ihc-hide-content ihc_mb_type=”show” ihc_mb_who=”4″ ihc_mb_template=”1″]

Yo siempre prefiero tener una capa de Servicios para después delegar en Repositorios .¿Porque?

- En primer lugar porque aunque las cosas comiencen de forma muy sencilla y parezca que únicamente tenemos q insertar o buscar la información del Repositorio. Según la aplicación evolucione nos podemos encontrar con nuevas necesidades tipo validaciones extra o comprobaciones antes de realizar modificaciones y estas pueden estar ubicadas en otros Servicios.



- En segundo lugar porque a veces nos podemos ver obligados a futuro a cambiar la tecnología que hace uso de la capa de persistencia . Por ejemplo hay gente que usa JDBC y de repente se anima a migrar a Spring Data y dar un salto tecnológico. Esto nos puede parecer lo más sencillo del mundo pero tiene su miga ya que Spring Data soporta sus propios repositorios con sus propios métodos por lo tanto no dispondremos de el metodo insertar sino que dispondremos del método save() . Si hemos ligado directamente los Controladores a los Repositorios tendremos que realizar muchísimos cambios ya que recordemos que los controladores están fuertemente ligados a las vistas y el numero de vistas siempre es muy amplio.



Conclusiones

Es aquí donde se ve claramente que el Servicio tiene la funcionalidad de fachada y nos permite aislarnos de lo que se encuentra detrás de él. Por eso mi consejo es siempre utilizar tanto capa de Servicios como capa de Repositorios aunque en un principio parezca que con los Repositorios es suficiente. Estas son mis recomendaciones para el uso de Spring @Service

[/ihc-hide-content]

Acabamos de ver como usar Java equals y hashcode

Otros artículos relacionados

- Spring @Component , anotaciones y jerarquía
- Spring MVC @SessionAttributes
- Spring MVC @SessionAttributes
- Spring @PathVariable y segmentos de url

- Spring @Service