Repasemos: Capa de Comunicación (o de Control)

La Capa de Comunicación (o de Control) en una aplicación Spring se encarga de gestionar la comunicación entre la interfaz de usuario y la lógica del negocio. Los controladores actúan como intermediarios, recibiendo solicitudes HTTP del usuario, invocando los servicios necesarios y devolviendo una respuesta adecuada.

En Spring, los controladores pertenecen a esta capa y tienen la responsabilidad de manejar las solicitudes HTTP entrantes y generar las respuestas correspondientes. Esto se logra mediante métodos que se asocian a rutas específicas de la aplicación.

Además, en esta capa interviene un componente clave conocido como **DispatcherServlet**, que se encarga de recibir las peticiones y redirigirlas al controlador adecuado en función de la URL solicitada.

Ejemplo de un controlador en Spring

```
@Controller
@RequestMapping("/")
public class PortalControlador {

    @GetMapping("/")  // Acá es donde realizamos el mapeo
    public String index() {
        return "index.html";   // Acá es que retornamos con el método.
    }

}
```

Principales anotaciones en los controladores

Anotación	Uso	Detalles adicionales
`@Controller`	Declara una clase como controlador.	Permite manejar entradas y salidas de datos en la interfaz de usuario.

`@RequestMapping`	Se usa para definir rutas a nivel de clase o método.	Puede aplicarse en controladores que no sean páginas de inicio para definir rutas de acceso específicas.
`@GetMapping`	Mapea una solicitud HTTP `GET` a un método.	Es una versión específica de `@RequestMapping(method = RequestMethod.GET)`.
`@PostMapping`	Mapea una solicitud HTTP `POST` a un método.	Se usa para enviar datos al servidor, por ejemplo, en formularios.
`@RequestParam`	Extrae parámetros de la URL o de un formulario.	Puede configurarse como obligatorio (`required = true`) o no (`required = false`).
`@PathVariable`	Extrae valores directamente de la URL.	Se usa cuando los parámetros son parte de la estructura de la URL.

Ejemplos de Uso de Anotaciones en los Controladores

- **@GetMapping**

```
@Controller
public class Controlador{
       @GetMapping("/")
        public String hola(){
        return "Hola Spring MVC";
        }
 }
```

En este ejemplo, cuando un usuario accede a `http://localhost:8080/`, se invoca el método `hola()`, el cual devuelve la cadena `"Hola Spring MVC"`.

- **@PostMapping**

Se utiliza para asignar solicitudes HTTP **POST** a métodos de controlador específicos. Es equivalente a usar `@RequestMapping(method = RequestMethod.POST)`.

```
@Controller
public class Controlador{
       @PostMapping("/guardar")
        public String guardarUsuario(){
```

```
        return "Usuario Guardado ";
        }
 }
```

Aquí, al enviar una petición POST a `http://localhost:8080/guardar`, se ejecuta el método `guardarUsuario()` y se devuelve `"Usuario Guardado"`.

- **@RequestParam**

Esta anotación vincula parámetros de una petición HTTP a los argumentos de un método del controlador. Es útil para extraer datos enviados mediante query strings o formularios.

```
@GetMapping(“/libro”){
    public void mostrarDetalleLibro(@RequestParam("ISBN") String ISBN){
            System.out.println(ISBN);
 }}
```

Si accedes a la URL:

`http://localhost:8080/libro?ISBN=900848893`

El método `mostrarDetalleLibro()` se invoca y el parámetro `ISBN` se asigna con el valor `"900848893"`, que se imprime en consola.

- **@PathVariable**

Se utiliza para extraer valores directamente de la URL cuando estos forman parte de la ruta, a diferencia de `@RequestParam` que extrae parámetros de consulta.

```
 @GetMapping(“/libro{ISBN}”){
       public void mostrarDetalleLibro(@PathVariable("ISBN") String ISBN){
System.out.println(ISBN);
      }
 }
```

Si accedes a:

`http://localhost:8080/libro/900848893`

El valor `"900848893"` se extrae de la URL y se asigna al argumento `ISBN`, que se utiliza dentro del método.

En la Capa de Comunicación (o de Control) es fundamental utilizar estas anotaciones para mapear correctamente las solicitudes HTTP a los métodos de tus controladores. Esto permite una gestión ordenada de las peticiones, facilitando la integración entre la interfaz de usuario y la lógica de negocio.

THYMELEAF

Thymeleaf es un motor de plantillas que te permite definir una plantilla HTML y, junto con un modelo de datos, generar un documento dinámico. Esto es especialmente útil en entornos web, ya que separa la interfaz de usuario (vistas) de los datos del negocio (modelos).

¿Qué es exactamente un motor de plantillas?

Un motor de plantillas es una herramienta diseñada para separar la presentación (HTML) de los datos que se mostrarán. Su función principal es leer un archivo de texto que contiene una estructura HTML prediseñada e insertar en él información dinámica proporcionada por el controlador. De esta manera, consigues generar documentos HTML personalizados y adaptados a cada usuario.

Ejemplo básico:

```
<html>
      <body>
      Hola ${nombre}
      <body>
</html>
```

En este ejemplo, el motor de plantillas recorrerá el archivo, identificará la “etiqueta” `${nombre}` y la reemplazará por el valor correspondiente (por ejemplo, el nombre del visitante), ofreciendo una presentación personalizada.

Ventajas de Thymeleaf

Thymeleaf ofrece varias ventajas que lo hacen muy útil en el desarrollo web:

- **Templating natural:** Al basarse en atributos y etiquetas HTML estándar, puedes renderizar las plantillas de forma local y luego usarlas en el servidor sin necesidad de modificarlas drásticamente.

- **Integración fluida:** Facilita la colaboración entre los equipos de diseño y programación, ya que el HTML generado puede ser revisado y modificado directamente por los diseñadores.
- **Flexibilidad:** Permite incorporar lógica mínima en la vista, como iteraciones y condiciones, sin sobrecargar el HTML.

Tipos de Expresiones en Thymeleaf

Thymeleaf cuenta con diferentes tipos de expresiones para insertar datos dinámicos en tus plantillas:

1. **Expresiones variables:**

Se representan con `${...}` y te permiten acceder a datos y propiedades dentro del modelo.
Ejemplo:

```
<span th:text="${autor.nombre}"></span>
```

Este código mostrará el nombre del autor accediendo a la propiedad `nombre` del objeto `autor`.

2. **Expresiones de selección:**

Se indican con `*{...}` y permiten abreviar la sintaxis cuando ya has seleccionado un objeto previamente.
Ejemplo:

```
<span th:text="*{titulo}"></span>
```

Aquí, `*{titulo}` accede a la propiedad `titulo` del objeto seleccionado en un contexto anterior.

3. **Expresiones de enlace:**

Se usan con la sintaxis `@{...}` para construir URLs dinámicas que pueden incluir parámetros o variables.
Ejemplo:

```
<a th:href="@{/libro/{id}(id=${libro.id})}">Ver detalle</a>
```

Esto genera una URL dinámica para acceder al detalle de un libro, reemplazando `{id}` por el valor de `libro.id`.

Atributos Básicos de Thymeleaf

Thymeleaf ofrece diversos atributos que te permiten manipular el contenido de las plantillas de forma dinámica:

- **th:text:** Reemplaza el contenido textual de una etiqueta con el valor de una expresión.

```
<p th:text="${mensaje}"></p>
```

- **th:each:** Permite iterar sobre una colección y repetir un fragmento de código para cada elemento.

```
<ul>
    <li th:each="libro : ${libros}" th:text="${libro.titulo}"></li>
</ul>
```

- **th:value:** Asigna valores iniciales a elementos de entrada, como campos de formulario.

```
<input type="text" th:value="${usuario.nombre}">
```

- **th:if y th:unless:** Muestran u ocultan elementos HTML en función de una condición.

```
<span th:if="${persona.sexo == 'F'}">Femenino</span>
<span th:unless="${persona.sexo == 'F'}">Masculino</span>
```

- **th:href:** Construye URLs dinámicas para enlaces, permitiendo la creación de enlaces absolutos o relativos.

```
<a th:href="@{/contacto}">Contacto</a>
```

Uso de ModelMap para Enviar Datos a la Vista

Para enviar datos desde el controlador a la vista, utilizas el objeto `ModelMap` (o `Model`), que forma parte del paquete `org.springframework.ui`. El método `addAttribute` te permite pasar variables o colecciones al HTML.

Ejemplo de controlador:

```
@Controller
public class SaludoControlador {

    @GetMapping("/")
```

```
    public String saludo(ModelMap model) {
        model.addAttribute("nombre", "Fernando");
        return "saludo";
    }
}
```

En este ejemplo, el controlador agrega una variable llamada `"nombre"` al modelo con el valor `"Fernando"`. La vista Thymeleaf, que contiene una etiqueta como `<span th:text="${nombre}"></span>`, mostrará ese valor.

Para saber más sobre Thymeleaf, recomendamos ingresar en su **documentación oficial**