Patrones en la URI

Las peticiones que se mapean a métodos pueden utilizar una expresión global que incluya caracteres comodín:

- `?` equivale a un carácter cualquiera
- `*` equivale a cero o más caracteres dentro de un segmento del path
- `**` equivale a cero o más segmentos del path

Se entiende por un segmento del path a lo que se contiene entre dos `/`.

Mapeo de más de una URI a un controlador

La anotación `@RequestMapping` y sus derivadas (`@GetMapping`, `@PostMapping`, …) pueden recibir más de una ruta como argumento. Lo hacen recibiendo varias entre `{ }`.

```
@GetMapping({"/", "/index", "/list"})
```

De esta forma, tanto si invocamos a `/`, como a `/index` o `/list`, todas las llamadas se harán al mismo método.

Uso de `@RequestMapping`

Esta es la anotación *original* para mapear cualquier tipo de verbo HTTP con un método.

De hecho, podríamos sustituir este código:

```
@GetMapping("/")
public      String      welcome(@RequestParam(name="name",      required=false,
defaultValue="Mundo") String name, Model model)
```

por este otro

```
@RequestMapping(value="/", method=RequestMethod.GET)
public      String      welcome(@RequestParam(name="name",      required=false,
defaultValue="Mundo") String name, Model model)
```

Podemos utilizar también la anotación `@RequestMapping` para definir un segmento de ruta a nivel de controlador, de forma que:

```
@Controller
@RequestMapping("/app")
public class MainController {
    @GetMapping("/")
    public     String     welcome(@RequestParam(name="name",     required=false,
defaultValue="Mundo") String name, Model model) {
        model.addAttribute("nombre", name);
        return "index";
    }
```

```
}
```

La ruta para invocar el controlador `welcome` sería `http://localhost:8080/app/`. Si añadimos más métodos de controlador a esta clase controladora, la ruta `app` afectaría a todos los métodos.

Argumentos de un método del controlador

Tipo de dato	Descripción
`WebRequest`, `NativeWebRequest`	Acceso genérico a los parámetros de la petición o los atributos de sesión, sin usar el API Servlet
`javax.servlet.ServletRequest`, `javax.servlet.ServletResponse`	Acceso directo a la petición o respuesta. Se pueden utilizar los subtipos `ServletRequest`, `HttpServletRequest`, `MultipartRequest`, `MultipartHttpServletRequest`.
`javax.servlet.http.HttpSession`	Fuerza la presencia de una sesión, con lo que nunca será nulo. ¡Cuidado! ya que el acceso no es *thread-safe*.
`javax.servlet.http.PushBuilder`	Push Builder (Servlet 4.0) para realizar el *push* de recursos para el protocolo HTTP/2.
`java.security.Principal`	Usuario actualmente autenticado.
`HttpMethod`	Método (verbo) HTTP de la petición.
`java.util.Locale`	Locale actual de la petición.
`java.util.TimeZone` + `java.time.ZoneId`	Zona horaria asociada a la petición.
`java.io.InputStream`, `java.io.Reader`	Permite acceder a la petición en crudo.
`java.io.OutputStream`, `java.io.Writer`	Permite producir la respuesta en crudo.
`@PathVariable`	Permite acceder a variables presentes en la URI.
`@MatrixVariable`	Acceso a los pares nombre-valor presentes en la URI.
`@RequestParam`	Acceso a los parámetros de la petición, incluidos ficheros multipartes.
`@RequestHeader`	Acceso a los encabezados de la petición.
`@CookieValue`	Acceso a las cookies.
`@RequestBody`	Acceso al cuerpo de la petición HTTP. El cuerpo es convertido según la implementación del `HttpMessageConverter` configurado.
`@HttpEntity<B>`	Acceso a los encabezados y cuerpo de la petición.
`@RequestPart`	Acceso a una parte de una petición `multipart/form-data`.
`java.util.Map`, `org.springframework.ui.Model`, `org.springframework.ui.ModelMap`	Acceso al modelo que es expuesto a las plantillas para el renderizado de vistas.
`RedirectAttributes`	Especifica atributos en caso de redirección.

Tipo de dato	Descripción
`@ModelAttribute`	Para acceder a algún atributo existente en el modelo, con conexión de datos y validación aplicada.
`Error`, `BindingResult`	Para acceder a los errores de validación y la conexión de datos de un *command object*, o los errores de validación de un objeto `@RequestBody`.
`SessionStatus` + `@SessionAttributes`	Marca el procesamiento de un formulario completo, que activa la limpieza de atributos de sesión declarados a través de `@SessionAttributes`.
`@RequestAttribute`	Acceso a los atributos de la petición.

Aquellas anotaciones que permitan el uso de atributo `required`, podrán ser utilizadas junto con `java.util.Optional` de Java 8.

Fuente: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-arguments

Tipos de retorno

Tipo de dato	Descripción
`@ResponseBody`	El valor se convierte según el `HttpMessageConverter` configurado.
`HttpEntity<B>`, `ResponseEntity<B>`	Se devuelve la respuesta completa, incluyendo encabezados y cuerpo.
`HttpHeaders`	Para devolver una respuesta con encabezados y cuerpo vacío.
`String`	Es el más usual en las últimas versiones de Spring. Se trata del nombre de la plantilla, que será resuelto por el `ViewResolver` configurado.
`View`	Una instancia de `View` que se usará para renderizar junto con el modelo.
`java.util.Map`, `org.springframework.ui.Model`	Atributos para ser añadidos al modelo.
`@ModelAttribute`	Atributo para ser añadido al modelo.
`ModelAndView`	Vista y modelo de forma conjunta.
`void`	Si devuelve `void`, se entiende que se ha manejado la respuesta a través de `ServletResponse`, `OutputStream` o una anotación `@ResponseStatus`.

Puede revisar la lista completa en la fuente.

Fuente: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-return-types