Controllers and Request Mappings

The Spring Web MVC framework (often referred to as simply **Spring MVC**) is a rich model view controller web framework. Spring MVC lets you create special `@Controller` objects to handle incoming HTTP requests. Methods in your controller are mapped to HTTP by using `@RequestMapping` annotations.

You can define controller objects (**beans** in Spring terminology) by using a standard Spring bean definition in the Servlet's `WebApplicationContext`. The `@Controller` annotation allows for auto-detection, as long as the controller classes are defined in a subpackage of the package containing the `SpringBootApplication` class. The location of controllers can be specified using `@ComponentScan`, but advanced configuration is out of the scope of this module.

Request Mapping

The `@RequestMapping` annotation is used to map HTTP requests (the url that you can use in your browsers normally) to controllers methods. It has various attributes to match by URL, HTTP method, request parameters, headers, and media types.

It can be used at the class-level to express shared mappings or at the method level to narrow down to a specific endpoint mapping.

- at method level: defines relative url http://localhost:8080/hello/

```
@RequestMapping("/hello")
public String hello(Model model) { .. }
```

- at class level: defines relative url http://localhost:8080/index/hello/

```
@RequestMapping("/index")
public class IndexController {
  @RequestMapping("/hello")
  public String hello(Model model) { .. }
}
```

Controller method parameter types

`@RequestMapping` handler methods have a flexible signature and can choose from a range of supported controller method arguments and return values:

- `HttpMethod`: The HTTP method of the request (we will use either `GET` or `POST`).
- `@PathVariable`: For access to URI template variables. See URI patterns.

- `@RequestParam`: For access to Servlet request parameters. Parameter values are converted to the declared method argument type.
- `java.util.Map`, `org.springframework.ui.Model`, `org.springframework.ui.ModelMap`: For access to the model that is used in HTML controllers and exposed to templates as part of view rendering.
- `RedirectAttributes`: Specify attributes to use in case of a redirect—i.e. to be appended to the query string, and/or flash attributes to be stored temporarily until the request after redirect. See Redirect attributes and Flash attributes.
- `@ModelAttribute`: For access to an existing attribute in the model (instantiated if not present) with data binding and validation applied. See `@ModelAttribute` as well as Model and DataBinder.
- `Errors`, `BindingResultErrors`, `BindingResult`: For access to errors from validation and data binding for a command object (i.e. `@ModelAttribute` argument), or errors from the validation of an `@RequestBody` or `@RequestPart` arguments; an `Errors`, or `BindingResult` argument must be declared immediately after the validated method argument.

If a method argument is not matched to any of the above, by default it is resolved as an `@RequestParam` if it is a simple type (a primitive, a String or other CharSequence, a Number, a Date, a URI, a URL, a Locale, a Class, or a corresponding array), or as an `@ModelAttribute` otherwise. The complete list of `@RequestMapping` arguments is :books: here.

Controller method return values

We will normally use one of the following return types:

- `String`: A view name (stored under `src/main/webapp/WEB-INF/`) to be resolved with `ViewResolver`'s and used together with the implicit model. The handler method may also programmatically enrich the model by declaring a `Model` argument (see above).
- `View`: A `View` instance to use for rendering together with the implicit model. The handler method may also programmatically enrich the model by declaring a `Model` argument (see above).
- `ModelAndView` object: The view and model attributes to use, and optionally a response status.

More information on other return value types can be found :books: here.

URI patterns and @PathVariable

You can map requests using glob patterns and wildcards:

- `?` matches one character
- `*` matches zero or more characters within a path segment

- `**` match zero or more path segments

More information :books: here.

You can also declare URI variables and access their values with `@PathVariable`:

- using path variables with `@PathVariable`: http://localhost:8080/hello/World

```
@RequestMapping("/hello/{value}")
public String hello(@PathVariable String value, Model model) {...}
```

@RequestParam

Use the `@RequestParam` annotation to bind request parameters (i.e. query parameters or form data) to a method argument in a controller. For example: http://localhost:8080/hello?value=World

```
@RequestMapping("/hello")
public String greetingParam(
  @RequestParam(value="value", required=false, defaultValue="World") String value,
  Model model) { ... }
```

Method parameters using this annotation are required by default, but you can specify that a method parameter is optional by setting `@RequestParam`'s `required` flag to false. A default value can be provided using `defaultValue`. Type conversion is applied automatically if the target method parameter type is not String.

See more information :books: here.

Type conversion

Some annotated controller method arguments that represent String-based request input—e.g. `@RequestParam` and `@PathVariable` (also `@RequestHeader`, `@MatrixVariable`, and `@CookieValue`), - may require type conversion if the argument is declared as something other than String. For such cases type conversion is automatically applied based on the configured converters. By default simple types such as int, long, Date, etc. are supported.

Some examples: * getting primitive datatypes: http://localhost:8090/hello?value=1

```
@RequestMapping("/hello")
public String primitive(@RequestParam Integer value, Model model) { ... }
```

- getting dates: http://localhost:8090/hello/2016-07-10

```
@RequestMapping("/hello/{value}")
public String date(
  @PathVariable @DateTimeFormat(iso=ISO.DATE) Date value,
  Model model) { ... }
```

- getting collections: http://localhost:8090/hello?values=1&values=2

```
@RequestMapping("/hello")
public String collection(
  @RequestParam Collection<Integer> values,
  Model model) { ... }
```

See more information :books: here.

Handling HTTP requests (GET)

The `Model` parameter allows us to enrich the model with information that will be accessed in the view. The following method adds a model attribute `name` with the value `World` in the model:

```
@RequestMapping("/hello")
public String hello(Model model) {
  model.addAttribute("name", "World");
  return "hello";
}
```

Such attribute can be used from the view `hello`.

@ModelAttribute

Use the `@ModelAttribute` annotation on a method argument to access an attribute from the model, or have it instantiated if not present. The model attribute is also overlaid with values from HTTP Servlet request parameters whose names match to field names. This is referred to as **data binding** and it saves you from having to deal with parsing and converting individual query parameters and form fields. For example:

```
@RequestMapping("/owners/{ownerId}/pets/{petId}/edit")
public String processSubmit(@ModelAttribute Pet pet) { }
```

The Pet instance above is resolved as follows:

- From the model if already added via Model.
- From the HTTP session via `@SessionAttributes`.
- From a URI path variable: in the example, the `Pet` object is supposed to have attributes `ownerId` and `petId`.
- From the invocation of a default constructor.
- From the invocation of a "primary constructor" with arguments matching to Servlet request parameters.

For more information, see :books: here.

Common Mistakes

- **Problem**: HTTP Error 404 - this localhost page can't be found.
- **Solution**: either the request URI is not correct or the view cannot be located (check view name and configuration).
- **Problem**: The value does not appear in the view or a blank page is shown.
- **Solution**: check attribute names used, both in the view and in the controller. The view resolver may have failed at rendering the view.

References

References