SPRING WEB MVC FRAMEWORK (Part 2)

Instructor:

Table Content

1 • Model, ModelMap, and ModelAndView in Spring MVC

2 • RedirectView to Add/Fetch Flash Attributes using RedirectAttributes

2 • RequestBody and ResponseBody Annotations

3 • @SessionAttributes or @SessionAttribute

Learning Goals

❖ After the course, attendees will be able to:

Understand Spring Web MVC Framework and its core technologies.
Know how to write a Web application with Spring MVC Framework.

Section 1

MODEL, MODELMAP, AND MODELVIEW IN SPRING MVC

Model class

- The model can supply attributes used for rendering views.
- To provide a view with usable data, we simply add this data to its *Model* object. Additionally, maps with attributes can be merged with *Model* instances:

```
@GetMapping("/showViewPage")
public String passParametersWithModel(Model model) {
    Map<String, String> map = new HashMap<>();
    map.put("spring", "mvc");
    model.addAttribute("message", "Baeldung");
    model.mergeAttributes(map);
    return "viewPage";
}
```

ModelMap

- ModelMap class subclasses **LinkedHashMap.** It add some methods for convenience.
- It provides **addAttribute** method to put key value pair. This method return ModelMap objects that can be used for chaining.
- ModelMap uses as generics.
- ModelMap checks for null values.

```
@RequestMapping("/helloworld")
public String hello(ModelMap map) {
  String helloWorldMessage = "Hello world from FA!";
  String welcomeMessage = "Welcome to  FA!";
  map.addAttribute("helloMessage", helloWorldMessage);
  map.addAttribute("welcomeMessage", welcomeMessage);

  return "hello";
}
```

ModelAndView

- This interface allows us to pass all the information required by Spring MVC in one return.

```
@GetMapping("/goToViewPage")
public ModelAndView passParametersWithModelAndView() {
    ModelAndView modelAndView = new ModelAndView("viewPage");
    modelAndView.addObject("message", “Welcome to FA");
    return modelAndView;
}
```

ModelAndView with Redirect

- Create ModelAndView object:

```
ModelAndView modelAndView = new ModelAndView();

modelAndView.setViewName("redirect:/index.htm");

return modelAndView;
```

Section 2

REDIRECTVIEW TO ADD/FETCH FLASH ATTRIBUTES USING REDIRECTATTRIBUTES

RedirectAttributes class

- A specialization of the Model interface that controllers can use to select attributes for a redirect scenario.

- This interface also provides a way to **add flash attributes** and they will be automatically propagated to the "output" FlashMap of the current request.

- A **RedirectAttributes model is empty** when the method is called and is never used unless the method returns a redirect view name or a RedirectView.

- **After the redirect,** flash attributes are automatically added to the model of the controller that **serves the target URL**.

Methods

- **addFlashAttribute:**("key", "value")
  - ✓ Flash Attributes are attributes which lives in session for short time.
  - ✓ It is used to propagate values from one request to another request and then automatically removed.
  - ✓ Handling flash attributes are achieved using **FlashMap** and **FlashMapManager**.
  - ✓ But in annotated spring MVC controller, it can be achieved with **RedirectAttributes**.
- **addAttribute**(“attributeName”, “attributeValue”)
  - ✓ Add the supplied attribute under the supplied name.

Add Flash Attributes

```
@RequestMapping(value = "mybook", method = RequestMethod.GET)
  public ModelAndView book() {
    return new ModelAndView("book", "book", new Book());
  }

  @RequestMapping(value = "/save", method = RequestMethod.POST)
  public RedirectView save(@ModelAttribute("book") Book book,
      RedirectAttributes redirectAttrs) {
    redirectAttrs.addAttribute("msg", "Hello World!");
    redirectAttrs.addFlashAttribute("book", book.getBookName());
    redirectAttrs.addFlashAttribute("writer", book.getWriter());

    RedirectView redirectView = new RedirectView();
    redirectView.setContextRelative(true);
    redirectView.setUrl("/hello/{msg}");
    return redirectView;
  }
```

Fetch Flash Attributes

- To fetch flash attributes we have two approaches.
  - ✓ The first one is by using **Model** as an argument in the **@RequestMapping** method and fetch the flash attribute as below.

```
model.asMap().get("key");
```

```
@RequestMapping(value = "/hello/{msg}", method = RequestMethod.GET)
  public String hello(Model model, RedirectAttributes redirectAttrs,
      @PathVariable("msg") String msg, HttpServletRequest request) {
    System.out.println("Message:" + msg);

    System.out.println("Fetch Flash Attributes By using Model");
    System.out.println("Book Name:" + model.asMap().get("book"));
    System.out.println("Writer:" + model.asMap().get("writer"));

    return "redirect:/success.jsp";
  }
```

Section 3

@SESSIONATTRIBUTES OR @SESSIONATTRIBUTE

@SessionAttributes

- **@SessionAttributes** annotation is used to store the model attribute in the session. This annotation is used at controller class level.

```
@SessionAttributes("user")
public class LoginController {
        @ModelAttribute("user")
        public User setUpUserForm() {
                return new User();
        }
}
```

- **@SessionAttribute** annotation is used to retrieve the existing attribute from session that is managed globally and it is used at method parameter as shown follows.

```
@GetMapping("/info")
public String userInfo(@SessionAttribute("user") User user) {
        //... //... return "user";
}
```

Controller

```
@PostMapping("/dologin")
public String doLogin(@ModelAttribute("user") User user, Model model) {
        // Implement your business logic
        if (user.getEmail().equals(“admin") && user.getPassword().equals(“admin")) {
        …
        } else {
        model.addAttribute("message", "Login failed. Try again.");
        return “login";
}
return “index";
}
```

Thank you