

Spring MVC Controllers Part I

Sang Shin
JPassion.com
“Code with Passion!”



Topics

- What is a Controller?
- Request mapping
- Handler method arguments – Implicit models
- Handler method return types (for view selection)

What is a Controller?

What does a Controller do?

- Controller has a set of handlers (handler methods)
 - > A HTTP request is mapped to a handler of a controller
- Controllers handles HTTP requests
 - > Controller receives user input and transforms it into a model
 - > Controller performs business logic and then and set attributes (key/value pairs) of the model
- Typically a controller delegates business logic processing to a set of services
 - > The services in turn access database through DAO interface (or Repository interface)

Mapping Requests to Handlers with `@RequestMapping`

Mapping requests with `@RequestMapping`

- `@RequestMapping` annotation is used to map an URL to a handler method of a Controller class
- `@RequestMapping` annotation can be specified at
 - > Class level
 - > Method level
- URL specified at the method level `@RequestMapping` annotation is relative to the one specified at the class level `@RequestMapping` annotation

Mapping requests with @RequestMapping

@Controller

```
@RequestMapping("/appointments") // Used at the class level  
public class AppointmentsController {
```

```
    private final AppointmentBook appointmentBook;
```

```
    @Autowired
```

```
    public AppointmentsController(AppointmentBook appointmentBook) {
```

```
        this.appointmentBook = appointmentBook;
```

```
    }
```

```
// Handle http://localhost:8080/myapp/appointments
```

```
@RequestMapping(method = RequestMethod.GET) // Used at the method level
```

```
public Map<String, Appointment> get() {
```

```
    return appointmentBook.getAppointmentsForToday();
```

```
}
```

```
// Handle http://localhost:8080/myapp/appointments/4 or
```

```
// http://localhost:8080/myapp/appointments/5
```

```
@RequestMapping(value="{day}", method = RequestMethod.GET) // Used at the method level
```

```
public Map<String, Appointment> getForDay (
```

```
    @PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Model model) {
```

```
    return appointmentBook.getAppointmentsForDay(day);
```

```
}
```

@RequestMapping Only at Method Levels

// A @RequestMapping on the class level is not specified.

```
@Controller
public class ClinicController {

    private final Clinic clinic;
    @Autowired
    public ClinicController(Clinic clinic) {
        this.clinic = clinic;
    }

    // Handles http://localhost:8080/myapp/
    @RequestMapping("/")
    public void welcomeHandler() {
    }

    // Handles http://localhost:8080/myapp/vets
    @RequestMapping("/vets")
    public ModelMap vetsHandler() {
        return new ModelMap(this.clinic.getVets());
    }
}
```


Lab:

**Exercise1: Build “Helloworld”
Spring MVC Application Step by Step
4945_spring4_mvc_controllers_part1.zip**



Handler Method Arguments - Implicit Models

Implicit Model as a Handler Argument

- Spring MVC creates an empty model object and passes it to a handler method as an argument
 - > You can then add attributes to the model in the form of key/value pairs
- The model object is then exposed to the view (i.e. jsp page)
 - > View can access model attributes using Expression Language (EL)

Model Types that are supported

- *java.util.Map*
 - > Most generic type
- *org.springframework.ui.Model*
 - > Holder of attributes
- *org.springframework.ui.ModelMap*
 - > Supports chained calls and auto-generation of model attribute keys

org.springframework.ui.Model

```
import org.springframework.ui.Model;

@RequestMapping(value = "/hotels-search", method = RequestMethod.GET)
// Spring MVC creates an empty model object and passes it as an argument.
// In order to access it, all you have to do is to use it as an argument.
public String list(SearchCriteria criteria, Model model) {

    // Using the "SearchCriteria", perform the search through
    // "bookingService".
    List<Hotel> hotels = bookingService.findHotels(criteria);

    // Add an attribute (key/value pair) to the model. The view now can access
    // "hotelList" through ${hotelList} expression language notation
    model.addAttribute("hotelList", hotels);

    // Return logical view name "hotels/list", which results in displaying
    // "hotels/list.jsp".
    return "hotels/list";
}
```

View that accesses Model Attributes

```
<table class="summary">
  <thead>
    <tr>
      <th>Name</th>
      <th>Address</th>
      <th>City, State</th>
    </tr>
  </thead>
  <tbody>
    <c:forEach var="hotel" items="{hotelList}">
      <tr>
        <td>${hotel.name}</td>
        <td>${hotel.address}</td>
        <td>${hotel.city}, ${hotel.state}, ${hotel.country}</td>
      </tr>
    </c:forEach>
    <c:if test="{empty hotelList}">
      <tr>
        <td colspan="5">No hotels found</td>
      </tr>
    </c:if>
  </tbody>
</table>
```

`hotelList` is the key of the model attribute (key/value pair)

org.springframework.ui.ModelMap

```
import org.springframework.ui.ModelMap;
```

```
@RequestMapping("/deposit.do")
```

```
// Empty ModelMap object is pre-created by Spring MVC and passed as an argument.
```

```
// In order to access, all you have to do is to add it as a handler argument.
```

```
protected String deposit(
```

```
    @RequestParam("accountNo") int accountNo,
```

```
    @RequestParam("amount") double amount,
```

```
    ModelMap modelMap) {
```

```
    accountService.deposit(accountNo, amount);
```

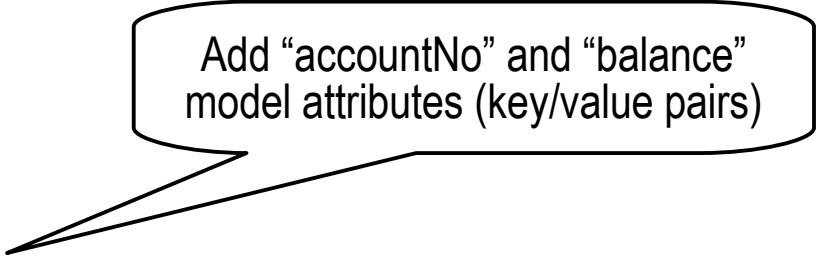
```
// Chaining is allowed for ModelMap object
```

```
    modelMap.addAttribute("accountNo", accountNo)
```

```
        .addAttribute("balance", accountService.getBalance(accountNo));
```

```
    return "success";
```

```
}
```



Add "accountNo" and "balance"
model attributes (key/value pairs)

Lab:

**Exercise2: Modify “Helloworld”
Spring MVC Application**

**Exercise3: Build “Helloworld”
Spring MVC Application from Spring.io**

4945_spring4_mvc_controllers_part1.zip



Handler Method Return Types (for View Selection)

Return Types from Handler Method

- Option #1: ModelAndView
- Option #2: String
- Option #3: void

#1: ModelAndView Object (as Return Type)

```
public class DisplayShoppingCartController implements Controller {  
  
    public ModelAndView handleRequest(HttpServletRequest request,  
                                    HttpServletResponse response) {  
  
        List cartItems = // get a List of CartItem objects  
        User user = // get the User doing the shopping  
  
        ModelAndView mav =  
            new ModelAndView("displayShoppingCart"); //logical view name  
  
        // Add attributes to ModelAndView object  
        mav.addObject("cartItems", cartItems);  
        mav.addObject("user", user);  
  
        return mav;  
    }  
}
```

#2: String (as a Return type)

- Returned String is interpreted as the logical view name
- More commonly used than *ModelAndView*

```
// Logical view is returned as "someview"  
@RequestMapping(value="html", method=RequestMethod.GET)  
public String prepare(Model model) {  
    model.addAttribute("foo", "bar");  
    model.addAttribute("fruit", "apple");  
    return "someview";  
}
```

#3: void (as a Return type)

- Used when the view name is supposed to be implicitly determined through a *RequestToViewNameTranslator*

```
@Controller
@RequestMapping("/helloworld")
public class MyController {

    // The view name is implicitly set to "/helloworld/viewNameX"
    @RequestMapping(value="/viewNameX", method=RequestMethod.GET)
    public void usingRequestToViewNameTranslator(Model model) {
        model.addAttribute("foo", "bar");
        model.addAttribute("fruit", "apple");
    }
}
```

Lab:

**Exercise 4: Request mapping,
Models, Logical views**

4945_spring4_mvc_controllers_part1.zip



Code with Passion!
JPassion.com

