Spring @ModelAttribute Ve @SessionAttributes Kullanımı

@ModelAttribute Annotasyonu

Metod üzerinde veya metod parametresi olarak iki şekilde kullanılmaktadır:

1. Metod üzerinde kullanıldığı zaman, `@RequestMapping` annotasyonundan önce çalışarak, **Model** nesnesinde, `@ModelAttribute` annotasyonunda belirtilen value (çift tırnak ile yazılan değerdir) değerine göre, bir attribute oluşturulup, bu attribute değerinin metod tarafından dönen değere göre set edilmesini sağlar. Eğer value değeri belirtilmezse, default bir değer atanır. Default value değeri metodun dönüş tipine göre belirlenir. Örneğin metodun dönüş tipi **OrderAddress** sınıfı türünden ise value **orderAddress** olur. Eğer dönüş tipi `List<OrderAddress>` gibi bir değer ise value **orderAddressList** olur.

Örnek:

```
1   @Controller
2   public class MyController {
3       @ModelAttribute("productList")
4       public Collection<Product> populateProducts() {
5           return this.productsService.getProducts();
6       }
7       // @RequestMapping etc omitted for brevity
8   }
```

`@ModelAttribute` value değeri: **productList**
populateProducts() metodunun dönüş değeri: **Product** sınıflarından oluşmuş bir koleksiyon.

O halde **Model** nesnesinde **productList** isminde bir attribute yaratılıp, bu attribute'nin değeri bu koleksiyon olmaktadır. **Örnekte** özellikle belirtildiği için attribute ismi **productList** olur. `@ModelAttribute` annotasyonunu **metod üzerinde** kullanıldığı zaman `@RequestMapping` annotasyonundan önce çalıştığı için bu metod içerisinde, önceden değerlerin eklenmesi gereken drop-down listeler gibi, veritabanı işlemleri gibi işlemler yapılabilir. Ayrıca `@ModelAttribute` metod üzerinde aşağıdaki gibi de kullanılabilir:

```
1   @ModelAttribute
2   public void populateModel(@RequestParam String number, Model model) {
3       model.addAttribute(accountManager.findAccount(number));
4       // add more ...
5   }
```

İlk örnekte metodun dönüş değerine göre **Model** nesnesine attribute değeri eklerken, ikinci kullanımda **Model** nesnesine birden çok attribute eklenebilmektedir.
Bir **Controller** sınıfı birden çok `@ModelAttribute` metod içerebilir. Bu metodların her biri `@RequestParam` metodlardan önce çağrılır.

2. Metod parametresi olarak kullanıldığı zaman **form uygulaması gibi** girilen değerlerin metod parametresine geçmesini sağlar.

Örnek:

```
1   @Controller
2   public class MyController {
3       @RequestMapping(method = RequestMethod.POST)
4       public String processSubmit(@ModelAttribute("product") Product myProduct, BindingResult result,
5                                   SessionStatus status) {
6
7           new ProductValidator().validate(myProduct, result);
8           if (result.hasErrors()) {
9               return "productForm";
10          }
11          else {
12              this.productsService.saveProduct(myProduct);
13              status.setComplete();
14              return "productSaved";
15          }
16      }
17  }
```

myProduct isimli metod parametresine form sayfasındaki **Model** nesnesinin product isimli attribute değeri atanmıştır. Dikkat ederseniz `@RequestMapping` annotasyonundaki method değeri **POST**'tur. Yani **processSubmit()** isimli metod **POST** request olduğu zaman çalışacaktır. **POST** request form sayfalarında kullanılmaktadır. Eğer **Model** nesnesinde product isimli attribute bulunmazsa, **Product** sınıfının default constructor(parametresiz constructor demektir)'u kullanılarak bir nesne yaratılır ve bu nesne **Model** nesnesine eklenir. Eklendikten sonra, **Product** nesnesinin değişkenleri, request parametrelerine göre, aynı isimde parametreler varsa, set edilir. Bu işlem, Spring MVC'de **data binding** olarak adlandırılmaktadır. Data binding, her form field değerini ayrı ayrı parse edilmesinden kurtardığı için çok kullanışlı bir yapıdır.

Örnek: User sınıfı

```java
public class User {
    private String userName;
    private String[] address;
    public User(){

    }
    public User(String userName, String[] address) {
        this.userName = userName;
        this.address = address;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public void setAddress(String[] address) {
        this.address = address;
    }

    public String getUserName() {
        return userName;
    }

    public String[] getAddress() {
        return address;
    }

}
```

Kullanımı:

```java
@RequestMapping(value = "/dataBinding/{userName}")
@ResponseBody
public String dataBinding(@ModelAttribute("user") User user){
    return user.getUserName();
}
```

`/dataBinding/musonyan` şeklinde bir URL girildiği zaman, Spring MVC öncelikle **Model** nesnesinde user isimli bir attribute olup olmadığına bakar. Bu isimde bir attribute bulamadığı zaman **User** sınıfının default constructor'unu kullanarak bir nesne üretir. Nesneyi ürettikten sonra, `{userName}` ile belirtilen kısımdaki **userName** **User** sınıfında bir değişken olduğu için, **musonyan** değeri, yeni yaratılan **User** nesnesinin **userName** değişkenine set edilir. Yukarıdaki örneğe göre **User** nesnesi nereden gelmektedir? **Olası seçenekler** şunlardır:

1. `@SessionAttributes` kullanılarak, HTTP request'ler arasında user isimli **Model** attribute eklenmişse, **Session** nesnesinden bu attribute getirilir.

2. Aynı **Controller** sınıfında, **Model** nesnesine, bir metod üzerinde `@ModelAttribute` annotasyonu kullanarak, user isminde attribute set edilmişse, bu değerden getirilir.

3. Yukarıdaki örnekte olduğu gibi URI değişkeninden yeni bir nesne yaratılıp kullanılır.

4. Default constructor kullanılarak nesne yaratılmışsa, bu değerden getirilir.

Not: Genelde bu annotasyon form sayfalarında kullanılmaktadır. Bundan dolayı daha iyi kavramak için form uygulamalarına bakabilirsiniz.

@SessionAttributes Annotasyonu

`@SessionAttributes` annotasyonu ile **Model** attribute değerlerinin HTTP request'ler arasında taşınmasını sağlayabiliriz. `@ModelAttribute` ile birlikte kullanılan bu annotasyonun kullanım şekli aşağıdaki gibidir:

```java
@Controller
@SessionAttributes({"user"})
public class EditUserForm {
    @ModelAttribute("user")
    public User populateUser {
        return new User(); // populates form for the first time if its null
    }
    @RequestMapping(value = "/dataBinding/{userName}")
    @ResponseBody
    public String dataBinding(@ModelAttribute("user") User user){
        return user.getUserName();
    }
}
```

Not: @SessionAttributes annotasyonu sınıf üzerinde kullanılır, metod üzerinde kullanılmaz.

Not: 2. satırdaki {} küme parantezleri arasında birden fazla değer girilecekse virgül ile ayrılır.

Örnek: `{"user","test"}/dataBinding/musonyan` şeklinde bir request olduğu zaman, Spring ilk olarak @SessionAttributes annotasyonunda küme parentezleri içinde yazılan value(örnekte "user") değerine göre, **Model** nesnesinde attribute karşılığı olup olmadığını kontrol eder, bu değer yoksa **populateUser** metodunu çalıştırarak, boş bir **User** nesnesi yaratıp "user" attribute değerine atar. Daha sonra, **dataBinding** metodunu çalıştırır. Bu metodta **User** nesnesine, **populateUser** metodunda yaratmış olduğu boş nesneyi atar. @RequestMapping annotasyonunda belirtilen value'deki `{userName}` **User** sınıfında değişken adı olduğu için, `/dataBinding/musonyan` şeklinde bir **GET** requesti olduğunda **User** nesnesinin **userName** değerine **musonyan** değeri atanır.

populateUser metodu aşağıdaki gibi yazılmış olsaydı ve `/dataBinding/musonyan` şeklinde bir request olsaydı, **userName** değeri **bemukan** değil, **musonyan** olurdu. Eğer `@RequestMapping(value = "/dataBinding/{userName}")` ifadesindeki `{userName}` değeri olmasaydı veya **User** sınıfı değişkenlerinden birinin adına eşit olmasaydı, o zaman **userName** değeri **bemukan** olarak kalırdı.

```
@ModelAttribute("user")
public User populateForm {
    User user=new User();
    user.setUserName("bemukan");
    return user;
}
```

Not: @SessionAttributes ve metod üzerinde kullanılan @ModelAttribute annotasyonuna sahip bir **Controller** sınıfında, **Model** nesnesinin ilgili attribute değerinin bulunması için Spring, önce @SessionAttributes'te olup olmadığına bakar, bulamazsa @ModelAttribute annotasyonu ile belirtilmiş metodu çalıştırır. **Not:** Eğer @ModelAttribute ile ilişkili bir metod tanımlanmamışsa ve @SessionAttributes'te de bu değeri bulamazsa **HttpSessionRequiredException** fırlatır. **dataBinding** metodu aşağıdaki gibi tanımlanırsa, **populateForm** metodu sadece 1 kez çalışmış olur. Bu sayede veritabanı işlemleri yapıyorsak, 1 kez veritabanı işlemi gerçekleştirilir.

```
@RequestMapping(value = "/dataBinding/{userName}")
public ModelAndView dataBinding(@ModelAttribute("user") User user) {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("user", user);
    modelAndView.setViewName("index");
    return modelAndView;
}
```

Çünkü @SessionAttributes annotasyonu sayesinde **Model** nesnesinin user attribute değeri **Session** nesnesi tarafından tutulur. Bu sayede hem aynı **Controller** sınıfında hem de diğer **Controller** sınıflarında bu attribute değerine erişebiliriz.

Not: Bir metod parametresi olarak @ModelAttribute kullanıldığı zaman, jsp sayfasından **User** nesnesine erişmek için, **Model** nesnesine explicitly olarak eklemek şart değildir.

```
@RequestMapping(method = RequestMethod.POST)
public String processSubmit(@ModelAttribute("user") User user,
                    BindingResult result, Model model) {
    model.addAttribute("user",user); //Kullanmak zorunlu degil
}
```

Örnek: Map Edilen Metod

```
@RequestMapping(value = "/dataBinding/{userName}")
public String dataBinding(@ModelAttribute("user") User user) {
    return "index";
}
```

JSP Sayfası:

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1>${user.userName}</h1>
</body>
</html>
```