

## Model / ModelMap / Map

컨트롤러의 메소드를 작성할 때 Model이라는 타입을 파라미터로 지정할 수 있습니다. Model 객체는 컨트롤러에서 생성된 데이터를 담아서 전달하는 역할을 하는 존재입니다. 메소드의 파라미터에 Model 타입이 지정되면 스프링은 특별하게 Model 타입 객체를 만들어서 메소드에 주입하게 됩니다.

ModelMap, Map도 Model과 동일한 기능을 수행합니다.

Model은 Servlet에서의 request.setAttribute()와 유사한 역할을 합니다. 다음은 Servlet에서 데이터를 Request 객체에 담아 뷰 페이지로 전달하는 코드입니다.

```
request.setAttribute("name", "xxx");  
RequestDispatcher dis = request.getRequestDispatcher("/WEB-INF/jsp/home.jsp");  
dis.forward(request, response);
```

위 코드를 스프링에서 Model을 이용하면 다음과 같습니다. Model.addAttribute를 이용하여 필요한 데이터를 Model 객체에 담습니다.

```
public String home(Model model) {  
    model.addAttribute("name", "xxx");  
    return "home";  
}
```

Model을 이용해 컨트롤러에서 뷰로 데이터를 전달하여 출력해 보겠습니다. 아래 세 가지 메소드를 각각 테스트해 보면 동일한 결과가 출력됩니다.

```
public class Person {  
    private String name;
```

```

    private int age;
    //Constructors, Getters and Setters
}

```

```

@RequestMapping("/test1")
public String test1(Model model) {
    model.addAttribute("msg", "hello");
    model.addAttribute("person", new Person("amy", 10));
    return "view1";
}

```

```

@RequestMapping("/test1")
public String test1(ModelMap model) {
    model.addAttribute("msg", "hello");
    model.addAttribute("person", new Person("amy", 10));
    return "view1";
}

```

```

@RequestMapping("/test1")
public String test1(Map<String, Object> model) {
    model.put("msg", "hello");
    model.put("person", new Person("amy", 10));
    return "view1";
}

```

```

<!-- view1.jsp -->
EL : ${ msg } <br>
EL : ${ requestScope.msg } <br>
JSP : <%= request.getAttribute("msg") %> <br>
<hr>
EL : ${ person } <br>
EL : ${ requestScope.person } <br>
EL : ${ person.name }, ${ person.age } <br>
JSP : <%= request.getAttribute("person") %> <br>

```

```

EL : hello
EL : hello
JSP : hello

```

---

```

EL : Person [name=amy, age=10]
EL : Person [name=amy, age=10]
EL : amy, 10
JSP : Person [name=amy, age=10]

```

# ModelAndView

ModelAndView 객체는 Model이 하는 역할과 마찬가지로 컨트롤러에서 뷰로 데이터를 전달하는데, 추가적으로 뷰 페이지의 이름까지 함께 전달하는 객체입니다. 이때 메소드의 리턴타입은 ModelAndView 객체가 됩니다. ModelAndView를 이용할 경우 ModelAndView.addObject를 이용하여 데이터를 담고, ModelAndView.setViewName을 이용하여 뷰 페이지 이름을 지정합니다.

```
@RequestMapping("/test2")
public ModelAndView test2() {
    ModelAndView mav = new ModelAndView();
    mav.addObject("msg", "hello");
    mav.addObject("person", new Person("amy", 10));

    mav.setViewName("view2");
    return mav;
}
```

```
<!-- view2.jsp -->
EL : ${ msg } <br>
EL : ${ requestScope.msg } <br>
JSP : <%= request.getAttribute("msg") %> <br>
<hr>
EL : ${ person } <br>
EL : ${ requestScope.person } <br>
EL : ${ person.name }, ${ person.age } <br>
JSP : <%= request.getAttribute("person") %> <br>
```

```
EL : hello
EL : hello
JSP : hello
```

---

```
EL : Person [name=amy, age=10]
EL : Person [name=amy, age=10]
EL : amy, 10
JSP : Person [name=amy, age=10]
```

## @ModelAttribute

@ModelAttribute 어노테이션은 메소드 파라미터나 메소드 리턴값을 Model 객체에 주입하거나 바인딩하여 뷰 페이지로 전달하는 역할을 하도록 합니다. @ModelAttribute 어노테이션은 메소드 파라미터 또는 메소드 레벨에 사용될 수 있습니다.

## Method Argument

메소드 인자로서 사용한 어노테이션은 Model에서 해당하는 속성을 검색하는 것을 나타냅니다. 해당하는 속성이 존재하지 않는 경우 인스턴스화한 후 Model에 추가합니다. 파라미터로 지정한 객체의 필드 name이 HTTP 요청 파라미터로 전달된 값의 name과 일치하는 경우 WebDataBinder를 통해 데이터를 자동으로 바인딩합니다.

```
@RequestMapping("/test3")
public String test3(@ModelAttribute("xxx") Person person) {
    return "view3";
}
```

```
<!-- view3.jsp -->
attribute name : ${ xxx } <br>
name : ${ xxx.name } <br>
age : ${ xxx.age } <br>
<br>
JSP : <%= request.getAttribute("xxx") %> <br>
```

← → ↻ localhost:8092/model/test3?name=amy&age=10

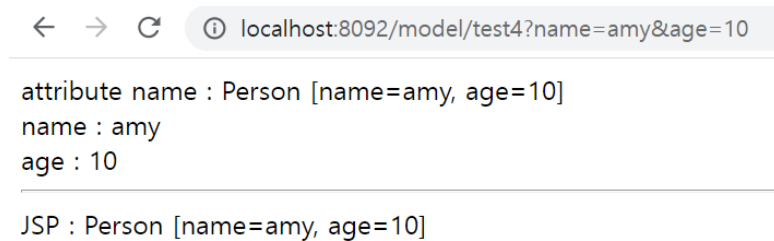
attribute name : Person [name=amy, age=10]  
name : amy  
age : 10

JSP : Person [name=amy, age=10]

name 속성을 지정하지 않는 경우 해당 타입의 소문자로 지정됩니다.

```
@RequestMapping("/test4")
public String test4(@ModelAttribute Person p) {
    return "view4";
}
```

```
<!-- view4.jsp -->
attribute name : ${ person } <br>
name : ${ person.name } <br>
age : ${ person.age } <br>
<br>
JSP : <%= request.getAttribute("person") %> <br>
```



```
← → ↻ ⓘ localhost:8092/model/test4?name=amy&age=10

attribute name : Person [name=amy, age=10]
name : amy
age : 10

JSP : Person [name=amy, age=10]
```

@ModelAttribute 어노테이션을 생략할 수도 있습니다. 파라미터 타입이 simple value type이 아니면, 다른 어떤 argument resolver에 의해 다뤄지지 않는 모든 파라미터는 @ModelAttribute 어노테이션이 붙은 것처럼 처리됩니다.

```
@RequestMapping("/test5")
public String test5(Person p) {
    return "view5";
}
```

```
<!-- view5.jsp -->
attribute name : ${ person } <br>
name : ${ person.name } <br>
```

```
age : ${ person.age } <br>
<hr>
JSP : <%= request.getAttribute("person") %>
```

```
← → ↻ ⓘ localhost:8092/model/test5?name=amy&age=10

attribute name : Person [name=amy, age=10]
name : amy
age : 10


---


JSP : Person [name=amy, age=10]
```

## Method Level

메소드 레벨에 `@ModelAttribute` 어노테이션을 사용할 경우 메소드의 목적이 하나 이상의 Model 속성을 추가하는 것임을 나타냅니다. 컨트롤러 메소드의 처리가 시작되기 전에 Model 객체가 생성되어야 하기 때문에, `@ModelAttribute` 메소드는 컨트롤러의 `@RequestMapping` 메소드가 호출되기 전에 호출됩니다. `@ControllerAdvice` 어노테이션을 사용하면 `@ModelAttribute` 메소드를 컨트롤러간 전역 메소드로 사용할 수도 있습니다.

```
@ModelAttribute
public void addAttributes1(Model model) {
    model.addAttribute("msg1", "hello");
}

@ModelAttribute("msg2")
public String addAttributes2() {
    return "world";
}

@RequestMapping("/test6")
public String test6() {
    return "view6";
}
```

```
<!-- view6.jsp -->
msg1 : ${ msg1 } <br>
```

msg2 : \${ msg2 } <br>



localhost:8092/model/test6

msg1 : hello  
msg2 : world