

Spring MVC Controllers



Controller Bean'ları

- Web katmanından **servis katmanına erişim** sağlayan bean'lardır
- HTTP **request**'lerini **handle** ederler
- Kullanıcı **input'unu alır ve modele dönüştürür**
- Spring 3 ile birlikte Controller beanları için herhangi bir **arayüz implement edilmesi, sınıftan türetilmesi** gerekmez
- Sınıf düzeyinde **@Controller** anotasyonu ile tanımlanırlar

Controller Bean'ları

- Controller bean'ları **DispatcherServlet'in kendi WebApplicationContext'i** içerisinde tanımlanmalıdır
- Bu XML tabanlı biçimde yapılabilir, yada **<context:component-scan />** elemanı kullanılabilir

@RequestMapping Annotasyonu

- HTTP request'leri ile **@Controller** bean'lerinin **@RequestMapping** annotasyonuna sahip metotları eşleştirilir
- Bu metotlara **handler metot** adı verilir
- Bir controller sınıfında **birden fazla** handler metot yer alabilir
- Handler metotlarının **alabileceği input parametreleri ve return değerinin** ne olabileceği oldukça esnektir

@RequestMapping Annotasyonu

@Controller

public class HelloWorldController {

@RequestMapping("/hello")

public ModelAndView helloWorld() {

ModelAndView mav = new ModelAndView();
mav.setViewName("/hello.jsp");
mav.addObject("message", "Hello World!");

return mav;

}

}

URL requestinin controller metotları ile eşleştirilmesini sağlar

Controller handler metodunun hem model, hem de view bilgisini tek bir return değeri olarak dönmesini sağlar

@RequestMapping Annotasyonu

- @RequestMapping anotasyonu **sınıf veya metot düzeyinde** kullanılabilir
- Sınıf düzeyinde **opsiyoneldir**
- Kullanıldığında sınıf düzeyindeki değer metotlardaki @RequestMapping tanımlarına **prefix** olarak eklenir

@RequestMapping Annotasyonu

Sınıf düzeyinde tanımlandığı takdirde metod düzeyindeki tanımlar relatif hale gelir

```
@Controller  
@RequestMapping("/greet")  
public class HelloWorldController {
```

```
    @RequestMapping("/hello")  
    public ModelAndView hello() {  
        // ...  
    }
```

hello() metodu eğer URI /greet/hello şeklinde olursa çağrılacaktır

```
    @RequestMapping("/bye")  
    public ModelAndView bye() {  
        // ...  
    }  
}
```

bye() metodu eğer URI /greet/bye şeklinde olursa çağrılacaktır

Controller Metot

Return Tipi: String

@Controller

```
public class HelloWorldController {
```

```
    @RequestMapping("/hello")
```

```
    public String helloWorld(ModelMap model) {
```

```
        model.addAttribute("message", "Hello World!");
```

```
        return "/hello.jsp";
```

```
    }
```

```
}
```



Controller metodunun return tipi String ise, bu “**logical view**” ismine karşılık gelir

Controller Metot

Return Tipi: void

@Controller

```
public class HelloWorldController {
```

```
    @RequestMapping("/hello")
```

```
    public void helloWorld(HttpServletRequest response) {  
        response.getWriter().write("Hello World!");  
    }
```

```
}
```

```
}
```



Return tipi void ise, bu DispatcherServlet'e “**response'u controller metodu üretecek, sen herhangi bir şey yapma!**” demektir

@ResponseBody

Annotasyonunun İşlevi

```
@RequestMapping("/hello")  
@ResponseBody  
public String helloWorld() {  
    return "Hello World";  
}
```

Metot return değeri http response body'sini oluşturur

@RequestBody

Annotasyonunun İşlevi

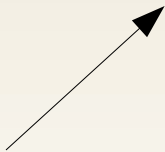
```
@RequestMapping(value="/printRequestBody",method=RequestMethod.POST)
public void handleRequest(@RequestBody String body, Writer writer)
throws IOException {
    writer.write(body);
}
```



Http request body'si @RequestBody ile işaretlenen metot parametresine atanır

@RequestParam

Request parametresinin değerini
metot parametresine atar



```
@RequestMapping("/pet")
public String displayPet(@RequestParam("petId") int petId,
    ModelMap model) {
    Pet pet = this.clinic.loadPet(petId);
    model.addAttribute("pet", pet);
    return "petForm";
}
```

<http://localhost:8080/petclinic/pet?petId=123>

@CookieValue ve @RequestHeader

```
@RequestMapping("/displaySessionId")  
public void displaySessionId(  
    @CookieValue("JSESSIONID") String cookie) {  
    // ...  
}
```

Http cookie değerini metod parametresine bind eder

```
@RequestMapping("/displayHeaderInfo")  
public void displayHeaderInfo(  
    @RequestHeader("Accept-Encoding") String encoding,  
    @RequestHeader("Keep-Alive") long keepAlive) {  
    // ...  
}
```

Http request header değerini metod parametresine bind eder

URI Template Kabiliyeti ve @PathVariable

URL içerisindeki bölümlere controller metodu içerisinde erişmeyi sağlar, her bir bölüm bir değişkene karşılık gelir. Bu değişkenlerin requestdeki karşılıkları controller metod parametrelerine aktarılır.

```
@RequestMapping("/owners/{ownerId}")  
public String findOwner(@PathVariable("ownerId") Long  
id, Model model) {  
    // ...  
}
```

`http://localhost/owners/1` --> `ownerId=1`

@PathVariable ile değişken değeri metod parametresine aktarılır

Metod parametresi herhangi bir basit tip olabilir: **int, long, String**

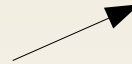
URI Template Kabiliyeti ve @PathVariable

URI template variable isimleri ile metot parametre isimlerinin eşleştirilmesi **derleme işleminin debug özelliği** açıksa mümkün olur. Aksi takdirde @PathVariable'a **variable ismi** verilmelidir

```
@RequestMapping("/{ownerId}")  
public String findOwner(@PathVariable Long ownerId, Model  
model) {  
    // ...  
}
```

URI Template Kabiliyeti ve Wildcard Kullanımı

Ant stili örüntüleri de destekler



```
@RequestMapping("/owners/*/pets/{petId}")  
public String findPet(@PathVariable Long petId, Model model) {  
  
    Pet pet = petService.getPet(petId);  
    model.addAttribute("pet", pet);  
  
    return "displayPet";  
}
```

```
@RequestMapping("/owners/**/pets/{petId}")
```



Herhangi derinlikteki path'leri kapsar

Controller Bean'ları ve Exception'lar

- Handler metotlarda meydana gelen exception'ları yakalamak için **exception handler metotlar** tanımlanabilir
- Metot üzerine **@ExceptionHandler** anotasyonu ile tanımlanır
- Her controller bean'ı için **ayrı ayrı** tanımlanmalıdır

Controller Bean'ları ve Exception'lar

@Controller

```
public class HelloController {
```

```
    @RequestMapping("/hello")
```

```
    public String helloWorld(ModelMap model) {
```

```
        model.addAttribute("message", "Hello World!");
```

```
        if(true) throw new RuntimeException("error!!!");
```

```
        return "/hello.jsp";
```

```
    }
```



```
@ExceptionHandler(RuntimeException.class)
```

```
public void handle(RuntimeException ex, Writer writer) {
```

```
    writer.write("Error handled :" + ex);
```

```
}
```

```
}
```



Birden fazla exception tipi alabilir

Değer olarak normal handler metotlar gibi view dönebilir,
yada response'u kendisi üretebilir

@ControllerAdvice

- @ControllerAdvice, farklı Controller sınıflarındaki handler metotlar için geçerli olacak bir takım **metotların ortak tek bir sınıfta toplanmasını** sağlar
- Bu yardımcı metotlar **@ExceptionHandler**, **@InitBinder**, **@ModelAttribute** gibi anotasyonlarla tanımlanmaktadırlar

@ControllerAdvice ve @ExceptionHandler

```
@ControllerAdvice
public class GlobalErrorHandler {

    @ExceptionHandler(IOException.class)
    public String handleException(IOException ex,
    HttpServletRequest request) {
        request.setAttribute("exception", ex);
        return "/error.jsp";
    }
}
```

@ControllerAdvice ve Paket Düzeyinde Sınırlandırma

- Normalde **@ControllerAdvice** ile işaretlenen **bean bütün Controller bean'ları** için geçerlidir
- Ancak ControllerAdvice'ın **hangi Controller bean'ları için geçerli olacağı** da belirtilebilir

```
@ControllerAdvice(basePackages={"com.javaegitimleri", "tr.com.harezmi"})  
public class ErrorHandler {  
    ...  
}
```

↓
Filtreleme anotasyon veya tiplere göre de yapılabilir

İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com

