Logo

Annotations

Spring framework uses following annotations to create beans and inject dependencies. The Spring container will then recursively scan all components in the given package & its sub-packages.

Defining a Controller

@Controller annotation is used to mark any of our Spring MVC controller classes.

```
@Controller
public class MyController {
}
```

@RestController annotation is used to define RESTful controllers.

```
@RestController
public class MyRestController {
}
```

Rest controllers always return response in JSON format whereas normal controllers may return response into xml, html, tiles of any other format. Rest controllers may configured to send response other then JSON.

Map URL with Controller

@RequestMapping

Request URLs are mapped with Controller classes and its methods. Annotation @RequestMapping is used to map the URLs. This annotation provides several options to customize its behavior.

1. method: denotes one the HTTP request methods – PUT, GET, POST, DELETE, PATCH
2. value: the mapped URL
3. params: filters the request based on the HTTP parameters
4. headers: filters the request based on the HTTP headers
5. produces: defines the media type of the HTTP responses
6. consumes: specifies the media type of the HTTP request

```
@RequestMapping("/User")
public class UserController {
    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public User login() {
        …
    }

    @RequestMapping(value = "/logout", method = RequestMethod.GET)
    public User logout() {
        …
    }
}
```

@RequestParam

We can bind an HTTP request parameter to the method parameter using this annotation.

```
@GetMapping("/message")
String message(@RequestParam("userName") String userName) {
    return "Hello " + userName;
}
```

@PathVariable

We can bind URIor path variables with method parameters using this annotation

```
@GetMapping("/message/{userName}")
public String message(@PathVariable String userName) {
   Return “Ok”;
}
```

Also, we can choose to mark a path variable as optional by setting required is equal to false.

```
@GetMapping("/message/{userName}")
public String message(@PathVariable(required=false) String userName){
 return “Ok”;
}
```

@RequestBody

@RequestBody annotation maps JSON object from HttpRequest POST request body to Java object.. Spring automatically deserializes the JSON into a Java type.

```
@PostMapping("addUser")
        public ORSResponse detailsAmt(@RequestBody UserForm form) {
                return res;
        }
}
```

@SessionAttribute

Annotation to bind a method parameter to a session attribute.@SessionAttribute used to pass value across different requests through the session. Rather than using HttpSession object directly, using this annotation can benefit auto type conversion and optional/required check.

```
@GetMapping("/message")
public String handle(@SessionAttribute("user") User user) {
 //
}
```

@ModelAttribute

@ModelAttribute is used to bind POST/GET request parameters with Java object. After binding request parameter, java object is stored in Spring MVC model object.

Data stored in Model is available to View component.

```
@PostMapping("/addUser")
public String addUser(@ModelAttribute("user") User user) {
}
```

The annotation is used to define objects which should be part of a Model. So if you want to have a User object referenced in the Model you can use the following method.

If you want to store return object of a method into model object then you can apply @ModelAttribute annotation on the method:

```
@ModelAttribute("user")
public User getUser() {
        return  new User();
}
```

Response-Handling Annotations

@ResponseBody

The @ResponseBody annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object.

Following example will convert User object into JSON and send it to http response

```
@GetMapping("/login")
@ResponseBody
public User login() {
    return new User();
}
```

@ExceptionHandler

We can write custom exception handler methods. These methods will be invoked when an exception of its type is raised during the method execution.

```
@ExceptionHandler(RuntimeException.class)
public static void applicationException(RuntimeException ex) {
    // On User Denied Exception
}
```

@ResponseStatus

With this, we can map the desired HTTP response status to the methods in our controller

```
@ExceptionHandler(RuntimeException.class)
@ResponseStatus(HttpStatus.FORBIDDEN)
public static void applicationException(RuntimeException ex) {
    // On User Denied Exception
}
```