Spring mvc:

A Spring MVC is a Java framework which is used to build web applications.

It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.



## Spring MVC 3.2 Execution Flow

Step **1**: First request will be received by DispatcherServlet  
Step **2**: DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request  
Step **3**: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModeAndView object (contains Model data and View name) back to the DispatcherServlet  
Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page  
Step **5**: Finally DispatcherServlet will pass the Model object to the View page to display the result

Spring Annotations:

## @Controller Annotation

Controller annotation tells the Spring IOC container to treat this class as [Spring MVC controller](https://www.javadevjournal.com/spring-mvc/spring-controllers/).

<context:component-scan base-package="com.javadevjournal">

@Controller

public class SpringMVCController {

*//HTTP Mappings*

}

## 2. @RestController Annotation

A convenience annotation that is itself annotated with @Controller and @ResponseBody.

@RestController

public class FilterExampleController {

@GetMapping

public String greeting() {

return "Hello World";

}

@GetMapping(value = "/greeting")

public String customGreetings() {

return "Hello From Custom Greetings";

}

}

## 3. @RequestMapping

Annotation for mapping web requests methods in the Spring MVC Controller. Both Spring MVC and Spring WebFlux support this annotation.@RequestMapping annotation provides several options to customize its behavior.

* Consumes – The consumable media types of the mapped request, narrowing the primary mapping. (e.g. @RequestMapping(consumes = {"application/json", "application/xml"})).
* method – The HTTP request methods to map (e.g. method = {RequestMethod.GET,RequestMethod.POST}).
* header – The headers of the mapped request.
* name – the name of the mapping.
* value – The primary mapping expressed by this annotation
* produces – The producible media types of the mapped request.

Here is an example for the @RequestMapping

@Controller

public class SpringMVCController {

@RequestMapping(value = {

"/greetings",

"/hello-world"}, method = {RequestMethod.GET,RequestMethod.POST},

consumes = {"application/json","application/xml"},

produces = { "application/json"},headers = {"application/json"

})

public String hellpWorld() {

return "Hello";

}

}

@Controller

@RequestMapping(value = {

"/greetings"

}, method = {

RequestMethod.GET

})

public class SpringMVCController {}

## 4. @RequestParam

Annotation which shows that it binds a method parameter to a web request parameter. Request parameters passed by the browser/client as part of the HTTP request, the @RequestParam annotation help to map these parameters easily at the controller level.

@GetMapping("/request-mapping-example")

public String requestMappingExample(@RequestParam("code") String code) {

*//*

}

## 5. @PathVariable

This annotation shows that a method parameter bound to a URI template variable. We specify the variable as part of the @RequestMapping and bind a method argument with *@PathVariable*. Let’s take an example where we want to pass productCode as part of the URI and not request parameter.

http://localhost:8080/sample/products/400

@GetMapping("/products/{id}")

public String getProduct(@PathVariable("id") String id) {

*//*

}

## 6. @SessionAttribute

Annotation to bind a method parameter to a session attribute.[@SessionAttribute](https://www.javadevjournal.com/spring-mvc/spring-mvc-session-attributes/) 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("/user")

public String sessionexample(@SessionAttribute(name = "userLoginTime") LocalDateTime startDateTime) {

*//*

}

## 7. @RequestBody

The [@RequestBody annotation](https://www.javadevjournal.com/spring/spring-request-response-body/) showing a method parameter bound to the body of the web request. It passes the body of the request through an HttpMessageConverter to resolve the method argument depending on the content the request.

@PostMapping("/product/save")

public String saveProduct(@RequestBody Product product){}

## 8. @ResponseBody

The [@ResponseBody Annotation](https://www.javadevjournal.com/spring/spring-request-response-body/) that shows a method return value bound to the web response body. Supported for annotated handler methods.**Spring treats the result of the method as the response itself.**

@GetMapping("/products/{id}")

public @ResponseBody Product saveProduct(@PathVariable("id") String id) {

*//*

}

## 9. @ExceptionHandler

[ExceptionHandler](https://www.javadevjournal.com/spring/exception-handling-for-rest-with-spring/) is a Spring annotation handle exceptions thrown by request handling. This annotation works at the @Controller level.

@GetMapping("/greeting")

String greeting() throws Exception {

*//*

}

@ExceptionHandler({

Exception.class

})

public handleException() {

*//*

}

## 11. @ModelAttribute

@ModelAttribute refers to a property of the Model object in Spring MVC. This [ModelAttribute annotation](https://www.javadevjournal.com/spring-mvc/spring-mvc-model-attribute-annotation/" \o "Spring MVC @ModelAttribute Annotation" \t "_blank) binds a method parameter or method return value to a named model attribute, exposed to a web view.

@PostMapping("/customer-registration")

public String processForm(@ModelAttribute("customer") Customer customer) {}

## 12. @Qualifier Annotation

The ***@Qualifier annotation*** helps disambiguate bean references when Spring otherwise could not do so. In our case, to we can use the @Qualifier annotation help in the issue to choose the correct bean for the dependency injection. Let’s change our previous example:

public class OrderFacade {

@Qualifier("userServiceImpl")

@Autowired

private UserService userService;

}

## 13. @CrossOrigin

This annotation allows the cross-domain communication for the annotated handler methods. This @CrossOrigin annotation enables cross-origin resource sharing only for this specific method. Let’s take an example where we want to allow only <a class="bare" href="http://localhost:9000">http://localhost:9002</a> to send cross-origin requests.

@CrossOrigin(origins = "http://localhost:9002")

@GetMapping("/hello")

public String greeting(@RequestParam(required = false, defaultValue = "Stranger") String name) {

return "Hello" + name;

}

@Autowired:

[@Autowired annotation](https://www.javadevjournal.com/spring/spring-autowiring/) used to inject object implicitly,

Spring @autowired annotation means to tell Spring to handle the instantiation of the class.

@Autowired

private HelloWorldService helloWorldService;

#### @Bean

@Bean annotation is used to create a bean in the Spring Framework, we use this annotation at the method level

@Bean

public Customer customer() {

*// instantiate and configure customer obj*

return customer;

}

#### @Configuration

We use this annotation at the class level which defines Beans, think of @Configuration annotation as a configuration using Java class (then traditional XML file to define Spring Beans)

@Configuration

public class ApplicationConfig {

@Bean

public OkHttpClient client(){

return new OkHttpClient.Builder()

 .build();

}

@Bean

public CommonsRequestLoggingFilter requestLoggingFilter() {

CommonsRequestLoggingFilter loggingFilter = new CommonsRequestLoggingFilter();

loggingFilter.setIncludeClientInfo(true);

loggingFilter.setIncludeQueryString(true);

loggingFilter.setIncludePayload(true);

loggingFilter.setIncludeHeaders(false);

return loggingFilter;

}

}

#### @ComponentScan

We use this annotation to configures component scanning directives for use with @Configuration classes, we can specify base package from where we want Spring to scan packages.

#### @Value

This is at the field or method/constructor parameter level that shows a default value expression for the affected argument. This is very similar to @Autowire annotation but @Value annotation will inject values from properties.

To understand it more, let’s take an example where we want to get some values from the property file, Let’s define our property file

user.name=Java Dev Journal

#### language=java

@Value("user.name")

private String username;

@Value("language")

#### private String language;

#### 1.9 @Service

This is a class level annotation shows that an annotated class is a "Service", Apart from the fact that it is used to show that it’s holding the business logic, there’s no noticeable specialty that this annotation provides.

@Service("helloService")

public class HelloWorldService1 implements HelloWorldService {

@Override

public void sayHello() {

}

#### }

#### 1.10 @Repository

This annotation is used on the classes which are working directly with the DB layer, @Repository annotation shows that an annotated class is a "Repository".One of the different thing being performed by classes annotated with @Repository annotation is to catch Platform specific exceptions and re-throw them as one of Spring’s unified unchecked exception.

#### 1.11 @Component

We use this annotation to show a Spring-powered component. This is a general-purpose stereotype annotation showing that the class is a spring component. If we check definitions of @Service or @Repository annotations

@Component

public @interface Service {

*//*

}

@Component

public @interface Repository {

*//*

}

we can safely say that @Service or @Repository annotations are special kinds of @Component annotation.