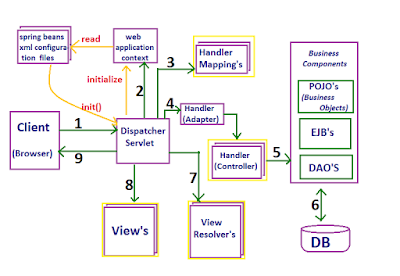
1. DispactherServlet

* Front Controller design pattern, a single controller is responsible for directing incoming HttpRequest to all application’s other controllers and handlers.
* Several interfaces that participate in the Workflow.
* DispatcherServlet Request handling
  + It delegates the request to the configured HandlerAdapter interface along with annotation specifying handlers, controller endpoints, and response objects.
  + It also plays important role in view resolution, erro handling ,locale resolution ,theme resolution.

Internal Working of Dispatcher Servlet

* It uses handler mappings and handler adapters to map a request to Spring MVC Controllers.(Uses @Contoller and @Request Mapping forthis purpose ).
* Oncethe request is processed by Spring MVN Contoller ,it returns logical view name .
* When dispatherServlet receives a view name ,it consults the ViewResolver to find the right view.
* There are chain of ViewResolver maintained by Spring Framework.
* Then try to resolve the logical view name into physical resource like JSP page r FreeMaker or Velocity Template.
* ViewResolver is invoked in order.Once found , Dispatcher Servlet forwards the request along with model datato the view and redering like JSP.
* By default , DispatcherServlet uses InternalResourceViewResolver uses the prfix and suffix to logical view Name (Eg: home to WEB/INF/home.jsp)



* 1. Receives all the requests from FrontController and provide single point of entry in application.
  2. ***WebApplicationContext*** Interface extends the Application Context Interface.
  + Provide configuration to web application.
  + Interface adds getServletContext() method to generic ApplicationContext Interface .- used to bound attribute names that root context must be bound to in bootstrap process.
  + Single root context preapplication. Each servlet has its own child context.
  + It detects ServletContextAware(Interface ) beans and invoke setServletContext method accordingly.
  1. HandlerMapping is an interface to be implemented by objects that define a mapping between request s and handler objects. By default, DispatcherServlet uses BeanNameUrlHandlerMapping and DefaultAnnotationHandlerMapping.

Ref : <https://www.javainterviewpoint.com/spring-mvc-handler-mapping/>

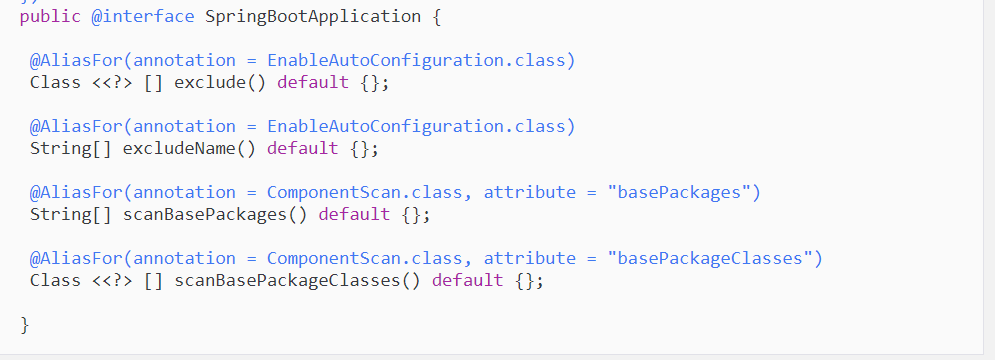
* 1. HandlerAdapter is used to actually invoke handler method.
  + ***RequestMappingHandlerAdapter*** : Handles classes and methods with @RequestMapping.
  + HandlerMethod wraps the information about method/parameter/return-type and enclosing @Controller bean.
  + HttpRequestHandlerAdapter: Supports handler objects of type HttpRequestHandler.
  + SimpleControllerHandlerAdapter: Suports handler type of Controller.
  1. Controller is specific type of Handler will execute the web request and returning a view .
  2. Business Logics components , DB transacions
  3. View Resolvers :
  + InternalResourceViewResolver – It maps both jsp and html. To generate the final view page URL



* + viewXMLViewResolver
  + ResourceBundleViewResolver
  1. View : Issuing the internal forward or include through Servlets API RequestDispatcher.forward ()or RequestDisaptcher.include()
  2. Finally redirected to browser.

1. What the @SpringBootApplication do ?

* Springboot made the configuring Spring easier with auto-configuration feature.
* Encapsulates 🡪 @Configuration + @EnableAutoConfiguration + @ComponentScan
* @EanbleAutoConfiguration 🡪 Enables the Springboot Auto configuration feature.
* @Configuration 🡪 one or more bean methods to use by Spring container to generate bean definitions and service request for beans at run time .
* @ComponentScan 🡪 Scans All the controller classes and components and registered beans
* Also adds additional features as mentioned below.



1. Features in spring 4 and spring 5
2. New features in Spring 5

* JDK baseline update
* Core framework revision
* Core container updates
* Functional programming with Kotlin
* Reactive Programming Model
* Improved Testing
* Library support

### JDK Baseline Update

The codebase of this new version of the Spring 5 Framework runs entirely on JAVA 8. To work with this new framework version, a programmer must be well-acquainted with the JAVA 8 basics as this is the core requirement for using the spring framework 5.0 . The first choice for this version was JAVA 9, but due to the delay of the release of JAVA 9, the latest spring version works with the JAVA 8. This version is designed in a way that it will be cent percent compatible with the latest JAVA 9. So the programmers who will work on JAVA 9 and develop an upper hand over the JAVA 9 skills can operate this spring 5 with it.

### Core Framework Revision

As the Spring 5 Framework is compatible with JAVA 8 and does not support other prior JAVA versions, the core of the spring framework had to be revised. This was done so that the spring framework could work easily with the updates and changes made in the JAVA 8. The revisions made to the core of the framework are as follows:

* The method parameters can be very efficiently accessed using this latest version of spring framework.
* Selective declarations that are built on JAVA 8 are now provided by the Core Spring interfaces.
* @Nullable and @NotNull annotations to explicitly mark nullable arguments and return values through which dealing with the null values at compile time rather than throwing NullPointerExceptions at runtime is enabled.

Spring Framework 5.0  has Commons Logging bridge module, called spring-jcl despite of the standard Commons Logging on the logging front. Auto detection of Log4j 2.x, SLF4J, JUL ( java.util.logging) without any extra bridges can be carried out using the Spring 5 Framework. Furthermore, defensive programming gets boosted with the Resource abstraction providing the isFile indicator for the getFile method.

### Core Container Updates

The candidate component index has replaced classpath scanning in the Spring Framework 5.0 . The reason why the classpath scanning has been replaced is that the candidate component index is a shortcut for the candidate component identification step in the classpath scanner. For the current project, an application build task can define its own META-INF/spring.components. When the compilation process starts, the source model has to be introspected while, JPA entities and Spring Components are flagged. However, this change is not so beneficial for projects that contain less than 200 classes but has outstanding benefits for projects that are larger and contain more than 200 classes.

A good news for the programmers with this latest version of Spring Framework 5.0 is that the startup time for applications is greatly reduced. Apparently 20 or 30 seconds is a small duration but when the queues are longer it looks very long. Component index can help in this regard.

Now @Nullable annotations are also utilized as indicators for other injection points. Using @Nullable can cause an obligation which will return a null value. Earlier the only way to do it was Nullable or Checker’s Nullable. This change has several other benefits that include; Implementation of functional programming style in GenericApplicationContext and AnnotationConfigApplicationContext, consistent detection of the transaction, caching, async annotations on interface methods and XML configuration namespaces streamlined towards unversioned schemas.

### Functional Programming with Kotlin

Kotlin language, a language that supports the functional programming style and is object oriented, by JetBrains is supported by the Spring Framework 5.0 . This language runs on top of JVM but it is not just restricted to it. The benefit of using this language for developing JAVA applications is that developers can dive into functional Spring programming, in particular for functional Web endpoints and bean registration.

**For example**

For a Web functional API, a developer can use the following code;

{

("/movie" and accept(TEXT\_HTML)).nest {

GET("/", movieHandler::findAllView)

GET("/{card}", movieHandler::findOneView)

}

("/api/movie" and accept(APPLICATION\_JSON)).nest {

GET("/", movieApiHandler::findAll)

GET("/{id}", movieApiHandler::findOne)

}

}

For registering as bean rather than XML, the following code in the Kotlin language can be used;

{

val context = GenericApplicationContext {

registerBean()

registerBean{ Cinema(it.getBean()) }

}

### Reactive Programming Model

The most highlighted feature of the latest version of the Spring Framework is the new reactive stack Web framework. Reactive is being called the update that takes us to future. This area of technology is gaining popularity with every passing day which is the reason why Spring Framework 5.0 has been launched with the capability of reactive programming. This addition makes the latest version of Spring Framework convenient for event-loop style processing which enables scaling with a small number of threads.

This is an API specification that the engineers came up with through the sources such as Pivotal, Red Hat, Netflix, Oracle, Typesafe, Twitter, and Spray.io. This feature provides a simple common API for implementing reactive programming. Examples of such implementations include JPA for Hibernate, here the JPA is the API, and Hibernate is the implementation.

The Project Reactor is what implements the API specifications for the reactive stream. The project reactor provides the base for streaming support in Spring Framework 5.0 is developed on.

WebSocket and Reactive HTTP clients are supported by Spring Framework 5.0. This is possible because the latest version has a new spring-webflux module. Moreover, the servers which have REST, HTML and WebSocket support are on which the reactive web applications run.

Programming models on the server-side in spring-webflux include:

* Annotation-based with @Controller and the other annotations of Spring MVC
* Functional style routing and handling with Java 8 lambda

**For Example:**  
In this example, Spring Webflux is being used for creatingWebClient, which is reactive and nonblocking. And also serves as the replacement of Resttemplate:

WebClientwebClient = WebClient.create();

Mono person = webClient.get()

.uri("http://localhost:8080/movie/42")

.accept(MediaType.APPLICATION\_JSON)

.exchange()

.then(response ->response.bodyToMono(Movie.class));

WebClientwebClient = WebClient.create();

Mono person = webClient.get()

.uri("http://localhost:8080/movie/42")

.accept(MediaType.APPLICATION\_JSON)

.exchange()

.then(response ->response.bodyToMono(Movie.class));

### Improved Testing

Junit 5 Jupiter can now be used to write tests and extensions in JUnit 5 in the latest version of Spring Framework. Moreover, a test engine to run Jupiter based test is also provided by the Jupiter sub-project. Also, the new updates in Spring Framework support parallel text execution.

WebTestClientthat is used for integrating testing support for Spring WebFlux is also provided for the reactive programming addition in the latest framework. WebTestClient, similar toMockMvc but unlike MockMvc does not require a running server. A WebTestClient can also be complied with help of the WebFlux server using a sample request and response.

Library Support

The new version has an upgraded library as well. These are the following upgraded library versions that are supported by Spring Framework 5.0:

* Jackson 2.6+
* EhCache 2.10+ / 3.0 GA
* Hibernate 5.0+
* JDBC 4.0+
* XmlUnit 2.x+
* OkHttp 3.x+
* Netty 4.1+

New features in Spring 4

* Java 8 is the minimum requirement to work with Spring framework 8
* JEE 7 support ( Servlet 3.1 , JMS 2.0, JPA 2.1 , JAX-RS 2.0 , Bean Validation 1.1)
* Reactive programming support
* Functional web framework
* Kotlin support

Dropped features :

* JasperReports
* XML Beans
* Guava

@Component

* ApplicationContext holds the instance of Objects thjat managed / distributed automatically called beans
* Using IOC,spring collects bean from our application and uses them at required time.

@autowird :

To inject spring-manged bean

* Allows spring to automatically detect our custom beans
* @Component 🡪 @Controller + @Service + @Repository

There are some important implications we should note because of the differences between *@Component* and *@Bean*.

* *@Component* is a class-level annotation, but *@Bean* is at the method level, so *@Component* is only an option when a class's source code is editable. *@Bean* can always be used, but it's more verbose.
* *@Component* is compatible with Spring's auto-detection, but *@Bean* requires manual class instantiation.
* Using *@Bean* decouples the instantiation of the bean from its class definition. This is why we can use it to make even third-party classes into Spring beans. It also means we can introduce logic to decide which of several possible instance options for a bean to use.

4. How Spring IOC Works ?

Create and manage the Objects

* 1. Config File has all the Bean class
  2. I have Spring IOC Container 🡪 read the config file 🡪 create object in container .(Spring beans)
  3. getBean(“a”);
  4. 2 Different types of IOC Container
  + BeanFactory – Interface
  + AppicationContext - Interface

ApplicationContext provide advance features.

* + ClassPathXmlApplicationContext

Working of Dispatcher Servlet ?