ABB Java Development Training

Ant-Ivy-Spring MVC framework

Basic Java concepts - recall

JDK

The **Java Development Kit** (**JDK**) is an implementation of either one of the **Java Platform, Standard Edition, Java Platform, Enterprise Edition** or **Java Platform, Micro Edition** platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, Mac OS X or Windows.

The JDK includes a private JVM and a few other resources to finish the development of a Java Application. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK).

JVM and JRE

The **Java Virtual Machine (JVM)** is the piece of software that runs a Java application. All Java software runs "in" or "on" a JVM. All JVMs provide the same environment for your application to run in, which allows a Java application to run on Windows, Linux, Mac and many other platforms.

Sun develops the most popular JVM implementation and provides it for Windows, Linux and Solaris. The JVM for Mac OS X is developed by Apple. BEA and IBM have implementations of the JVM, and there are a few minor open source ones. Although it's not used often, the term **Java Runtime Environment (JRE)** means an implementation of the JVM and all the other stuff you need to run a Java application.

Java EE

Java Platform, Enterprise Edition or **Java EE** is a widely used enterprise computing platform developed under the Java Community Process.

The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. Java EE extends the Java Platform, Standard Edition (Java SE), providing an API for object-relational mapping, distributed and multi-tier architectures, and web services.

The platform incorporates a design based largely on modular components running on an application server. Software for Java EE is primarily developed in the Java programming language. The platform emphasizes convention over configuration and annotations for configuration. Optionally XML can be used to override annotations or to deviate from the platform defaults.

The platform was known as **Java 2 Platform, Enterprise Edition** or **J2EE** until the name was changed to **Java Platform, Enterprise Edition** or **Java EE** in version **5**. The current version is called **Java EE 7**.

- J2EE 1.2 (December 12, 1999)
- J2EE 1.3 (September 24, 2001)
- J2EE 1.4 (November 11, 2003)
- Java EE 5 (May 11, 2006)
- Java EE 6 (December 10, 2009)
- Java EE 7 (May 28, 2013, but April 5, 2013 according to spec document. June 12, 2013 was the planned kickoff date)
- Java EE 8 (expected first half of 2017)

Java Enterprise Edition (Java EE/JEE is a collection of APIs which aren't included in the standard JDK but provide functionality which is useful for many server applications. One example is JavaMail, which is a standard way of accessing email from Java. This isn't available with the standard JDK, but Sun has provided a standard API and implementation for people to use in Java applications.

Application servers provide implementations of many of the JEE standards. Most important to Atlassian applications are the Java Servlet API, the Java Transaction API and the data source APIs in JEE. We also rely quite heavily on a front-end technology called JavaServer Pages (JSP). People talk about "JEE application servers" or just "application servers". You don't often hear JEE or J2EE mentioned when talking about versions or application requirements, except in the most technical discussions.

Apache Ant

Apache Ant is a software tool for automating software build processes, which originated from the Apache Tomcat project in early 2000. It was a replacement for the unix make build tool, and was created due to a number of problems with the unix make. It is similar to Make but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects.

The most immediately noticeable difference between Ant and Make is that Ant uses XML to describe the build process and its dependencies, whereas Make uses Makefile format. By default the XML file is named `build.xml`.

Ant is an Apache project. It is open source software, and is released under the Apache License.

Apache Ivy

Apache Ivy is a transitive dependency manager. It is a sub-project of the Apache Ant project, with which Ivy works to resolve project dependencies. An external XML file defines project dependencies and lists the resources necessary to build a project. Ivy then resolves and downloads resources from an artifact repository: either a private repository or one publicly available on the Internet.

WildFly

WildFly, formerly known as **JBoss AS**, or simply **JBoss,** is an application server authored by JBoss, now developed by Red Hat. WildFly is written in Java, and implements the Java Platform, Enterprise Edition (Java EE) specification. It runs on multiple platforms.

WildFly is free and open-source software, subject to the requirements of the GNU Lesser General Public License (LGPL), version 2.1.

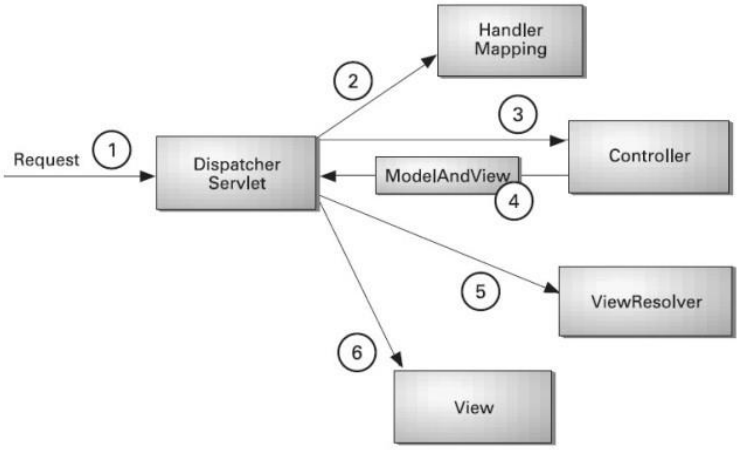
On 20 November 2014, JBoss Application Server was renamed WildFly. The JBoss Community and other Red Hat JBoss products like JBoss Enterprise Application Platform were not renamed.

Spring MVC

Spring MVC is the web component of **Spring** framework. It provides a rich functionality for building robust Web Applications. The **Spring MVC** Framework is architected and designed in such a way that every piece of logic and functionality is highly configurable.

In Spring Web MVC, **DispatcherServlet** class works as the front controller. It is responsible to manage the flow of the spring mvc application.

The **@Controller** annotation is used to mark the class as the controller in Spring 3.
The **@RequestMapping** annotation is used to map the request url. It is applied on the method



As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller. The DispatcherServlet gets entry of handler mapping from the xml file and forwards the request to the controller. The controller returns an object of ModelAndView. The DispatcherServlet checks the entry of view resolver in the xml file and invokes the specified view component.

Spring application has two types of context configuration files for Spring MVC module:

1. **ApplicationContext (default name for this file is applicationContext.xml)**
2. **WebApplicationContext (default name for this file is xxx-servlet.xml where xxx is the DispatcherServlet name in web.xml)**

ApplicationContext

- applicationContext.xml is the root context configuration for every web application.
- Spring loads applicationContext.xml file and creates the ApplicationContext for the whole application.
- There will be only one application context per web application.
- If you are not explicitly declaring the context configuration file name in web.xml using the contextConfigLocation param, Spring will search for the applicationContext.xml under WEB-INF folder and throw FileNotFoundException if it could not find this file.

WebApplicationContext

- Apart from **ApplicationContext**, there can be multiple **WebApplicationContext** in a single web application.
- In simple words, each DispatcherServlet associated with single WebApplicationContext.
- xxx-servlet.xml file is specific to the DispatcherServlet and a web application can have more than one DispatcherServlet configured to handle the requests.
- In such scenarios, each DispatcherServlet would have a separate xxx-servlet.xml configured. But, applicationContext.xml will be common for all the servlet configuration files.
- Spring will by default load file named "xxx-servlet.xml" from your webapps WEB-INF folder where xxx is the servlet name in web.xml.
- If you want to change the name of that file name or change the location, add initi-param with contextConfigLocation as param name.

ContextLoaderListener

- Performs the actual initialization work for the root application context.
- Reads a "contextConfigLocation" context-param and passes its value to the context instance, parsing it into potentially multiple file paths which can be separated by any number of commas and spaces, e.g. "WEB-INF/applicationContext1.xml, WEB-INF/applicationContext2.xml".

- ContextLoaderListener is optional. Just to make a point here: you can boot up a Spring application without ever configuring ContextLoaderListener, just a basic minimum web.xml with DispatcherServlet.

Sample web.xml file with configurations:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
         <display-name>Spring MVC</display-name>

     <!-- This is the root application context for whole web application. -->
    <context-param>
                   <param-name>contextConfigLocation</param-name>
                   <param-value>/WEB-INF/rootApplicationContext.xml</param-value>
         </context-param>
           <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
  <servlet>
        <servlet-name>webmvc1</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>

        <!-- We require this configuration when we want to change the default name / location of
the servlet specific configuration files -->
        <init-param>
          <param-name>contextConfigLocation</param-name>
          <param-value>/WEB-INF/mvc1-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

          <servlet>
        <servlet-name>webmvc2</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>

        <!-- We require this configuration when we want to change the default name / location of
the servlet specific configuration files -->
        <init-param>
          <param-name>contextConfigLocation</param-name>
          <param-value>/WEB-INF/mvc2-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

     <servlet-mapping>
        <servlet-name>webmvc1</servlet-name>
        <url-pattern>/webmvc1</url-pattern>
    </servlet-mapping>
        <servlet-mapping>
        <servlet-name>webmvc2</servlet-name>
        <url-pattern>/webmvc2</url-pattern>
    </servlet-mapping>
</web-app>
```

Some highlighted Spring MVC classes

org.springframework.stereotype.Controller

`public @interface` **`Controller`**
Indicates that an annotated class is a "Controller" (e.g. a web controller).

This annotation serves as a specialization of `@Component`, allowing for implementation classes to be auto-detected through classpath scanning. It is typically used in combination with annotated handler methods based on the `RequestMapping` annotation.

org.springframework.web.bind.annotation.PathVariable

`public @interface PathVariable`
Annotation which indicates that a method parameter should be bound to a URI template variable. Supported for `RequestMapping` annotated handler methods in Servlet environments.

If the method parameter is `Map<String, String>` or `MultiValueMap<String, String>` then the map is populated with all path variable names and values.

org.springframework.web.bind.annotation.RequestMapping

`public @interface RequestMapping`
Annotation for mapping web requests onto specific handler classes and/or handler methods. Provides a consistent style between Servlet and Portlet environments, with the semantics adapting to the concrete environment.

org.springframework.web.bind.annotation.RequestMethod

`public enum RequestMethod extends Enum<RequestMethod>`
Java 5 enumeration of HTTP request methods. Intended for use with the `RequestMapping.method()` attribute of the `RequestMapping` annotation.

Note that, by default, `DispatcherServlet` supports `GET, HEAD, POST, PUT, PATCH` and `DELETE` only. `DispatcherServlet` will process `TRACE` and `OPTIONS` with the default `HttpServlet` behavior unless explicitly told to dispatch those request types as well: Check out the "`dispatchOptionsRequest`" and "`dispatchTraceRequest`" properties, switching them to "`true`" if necessary.

org.springframework.ui.ModelMap (ModelAndView)

`public class ModelMap extends LinkedHashMap<String,Object>`
Implementation of `Map` for use when building model data for use with UI tools. Supports chained calls and generation of model attribute names.

This class serves as generic model holder for both Servlet and Portlet MVC, but is not tied to either of those. Check out the Model interface for a Java-5-based interface variant that serves the same purpose.
The `ModelMap` class is essentially a glorified `Map` that can make adding objects that are to be displayed in (or on) a View adhere to a common naming convention. Consider the following Controller implementation; notice that objects are added to the `ModelAndView` without any associated name specified.

The `ModelAndView` class uses a `ModelMap` class that is a custom `Map` implementation that automatically generates a key for an object when an object is added to it. The strategy for determining the name for an added object is, in the case of a scalar object such as User, to use the short class name of the object's class.

Good luck!