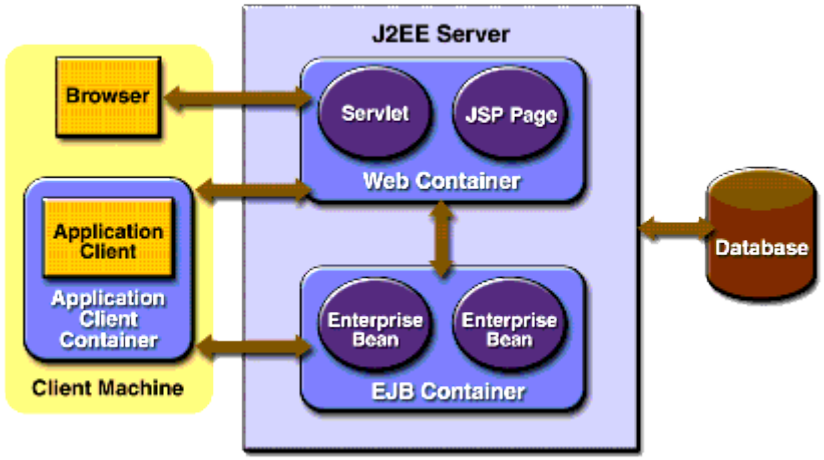
J2EE Overview

J2EE Container

- The application server which control and provide services through an interface is known as a *container.* J2EE container helps in deployment and execution of J2EE component



J2EE Server and Containers

- **J2EE server**

The runtime portion of a J2EE product. A J2EE server provides EJB and web containers.

- **Enterprise JavaBeans (EJB) container**

Manages the execution of enterprise beans for J2EE applications. Enterprise beans and their container run on the J2EE server.

- **Web container**

Manages the execution of JSP page and servlet components for J2EE applications. web components and their container run on the J2EE server.

- **Application client container**

Manages the execution of application client components. Application clients and their container run on the client.

- **Applet container**

Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

Packaging Web Applications

- You add web components to a J2EE application in a package called a *web application archive* (WAR), which is a JAR similar to the package used for Java class libraries. A WAR usually contains other resources besides web components, including:
  - ❖ Server-side utility classes
  - ❖ Static web resources (HTML, image, and sound files, and so on)
  - ❖ Client-side classes (applets and utility classes)
- A WAR has a specific hierarchical directory structure. The top-level directory of a WAR is the *document root* of the application. The document root is where JSP pages, client-side classes and archives, and static web resources are stored. The document root contains a subdirectory called WEB-INF, which contains the following files and directories:
  - ❖ web.xml - the web application deployment descriptor
  - ❖ Tag library descriptor files
  - ❖ classes - a directory that contains server-side classes: servlet, utility classes, and JavaBeans components.
  - ❖ lib - a directory that contains JAR archives of libraries (tag libraries and any utility libraries called by server-side classes)

Introduction to Java Web Application:

- It is used to create dynamic websites.
- Java provides support for web applications by using the **Servlets** and **JSPs**.
- The website can be created with static HTML pages, but when we want the information to be dynamic, we need a web application.

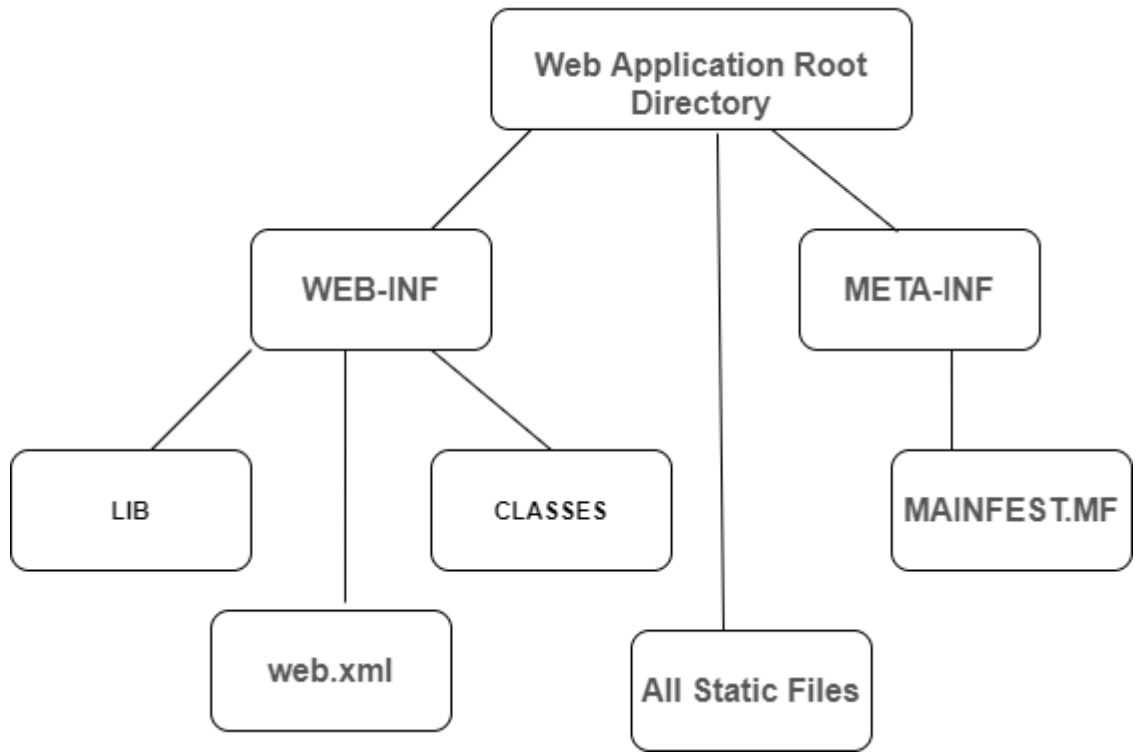
Java Web Application:

- The primary purpose of the java web application is to provide basic information about different components in Web Application and also provide the details on how we can use Servlet and JSP to create our first java web application.

Web Application Directory Structure:

Any web application we can have the following components like

- The Static contents like HTML.
- We have Client-side files we say, CSS and the Javascript.
- JSP (Java Server Pages), which use to generate dynamic content.
- Servlets.
- Some External library or the jar files.
- Java utility classes.
- We mainly use web.xml, also called as deployment descriptor.
- All the web servers and containers expect the web application to be available in a specific directory structure.


Web Application Root Directory
WEB-INF
META-INF
LIB
CLASSES
MAINFEST.MF
web.xml
All Static Files

Web Application Lifecycle

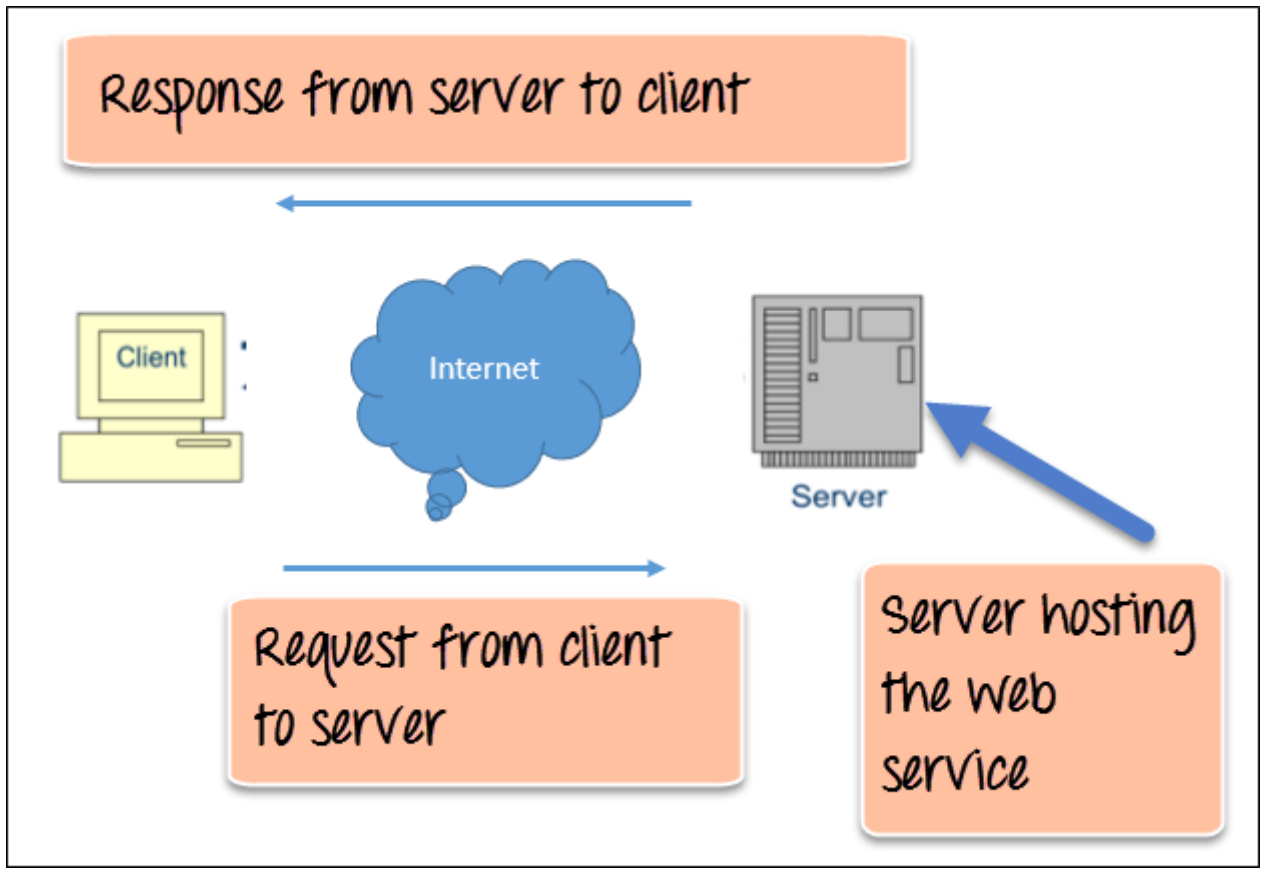
A web application consists of web components; static resource files, such as images and cascading style sheets (CSS); and helper classes and libraries. The web container provides many supporting services that enhance the capabilities of web components and make them easier to develop. However, because a web application must take these services into account, the process for creating and running a web application is different from that of traditional stand-alone Java classes.

The process for creating, deploying, and executing a web application can be summarized as follows:

- Develop the web component code.
- Develop the web application deployment descriptor, if necessary.
- Compile the web application components and helper classes referenced by the components.
- Optionally, package the application into a deployable unit.
- Deploy the application into a web container.
- Access a URL that references the web application.

Web service

- Web service is a standardized medium to propagate communication between the client and server applications on the WWW (World Wide Web). A web service is a software module that is designed to perform a certain set of tasks.
- Web services in cloud computing can be searched for over the network and can also be invoked accordingly.
- When invoked, the web service would be able to provide the functionality to the client, which invokes that web service.


Response from server to client
Client
Internet
Server
Request from client to server
Server hosting the web service

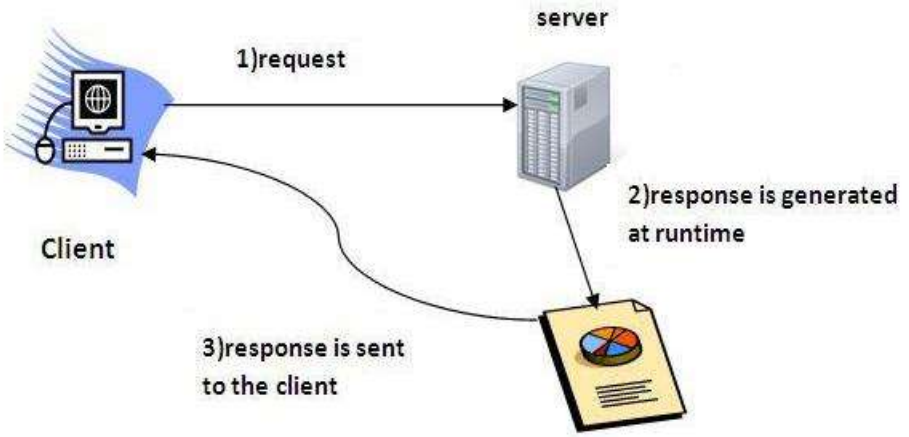
SERVLETS

What is web application?

- A web application is an application accessible from the web.
- A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML.
- The web components typically execute in Web Server and respond to HTTP request.

Java Servlet Technology

- For delivering services on web, service providers recognized the need for dynamic content.



Earliest attempts

- Applets - uses the client platform to deliver dynamic user experiences.

Common Gateway Interface (CGI) scripts - using the server platform to generate dynamic content

- CGI scripting technology has a number of shortcomings, including platform dependence and lack of scalability.
- To address these limitations, Java Servlet technology was created as a portable way to provide dynamic, user-oriented content.

What Is a Servlet?

- A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.
- Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.
- For such applications, Java Servlet technology defines HTTP-specific servlet classes.

Advantages of Servlets over CGI

1. **Platform Independent**

- Platform Independence plays a major in the field where there are numerous number of web servers available to choose from.
- Servlets are written entirely in java, due to which they are platform independent.
- Servlets can run on any servlet enabled web-server.
- Ex: web applications developed in windows machine running java web server, can easily run the same on apache web server (if Apache server is installed), without any modification or compilation of code.

2 **Performance**

- Due to interpreted nature of java, programs written in java are slow. But java servlets run very fast. it is because the way servlets run on web server.
- For any program Initialization takes significant amount of time in its life cycle. But in case of servlets initialization takes place only once when it receives the first request & remains in memory till times out or server shuts down's.
- Once servlet is initialized & loaded, to handle a new request it simply creates a new thread and runs service method of servlet.
- In case of CGI scripts, always a new process should be created to serve a request.

3 Extensibility

- Servlets are developed in java which is robust, well-designed & Object oriented language which can be extended or polymorphed into a new object
- java servlets take all java's advantages to provide the ideal solution.

4 **Safety**

- Java provides very good safety feature's like Memory management, Exception handling etc.
- Servlets inherit all these features & had emerged as a very powerful web server extension.

5 **Security**

- Since servlets are server side Components, it inherits the security of server.
- Servlets are also benefited with java security manager, provided by web server.

CGI	SERVLETS
It is Process based i.e, for every request a separate process will be created and it is responsible to generate required response.	It is thread-based i.e, for every request a new thread will be created and it is responsible to process the request.
Creation and Destruction of process for every request is costly. If the number of requests increase it will effects performance of the system.Hence CGI technology fail to the destroy of scalable.	Creation and Destruction of new thread for every request is not costly . Hence there is no effect on performance even though number of requests increases due to this it succeeds to delevary scalable.
Two process never share same address space(memory).Hence there is no chance of concurrency, inconsistency problems and syncronization is not required.	All threads share common address space hence there may be a chance of concurrency, inconsistency problems.
CGI programs can be written in multiple languages.But most commonly used language is PERL.	Servlets can be written only in Java.
Most of the CGI languages are not object oriented.Hence we are missing benefits of OOPs.	Java language itself is Object-Oriented . Hence we can get all benefits of OOPs.
CGI technology is platform dependent.	Servlet technology is platform independent.

What is a Servlet?

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Servlets Life Cycle

1. Loads Servlet Class
2. Creates instance of Servlet
3. Calls init( ) method
4. Calls service( ) method
5. Calls destroy( ) method

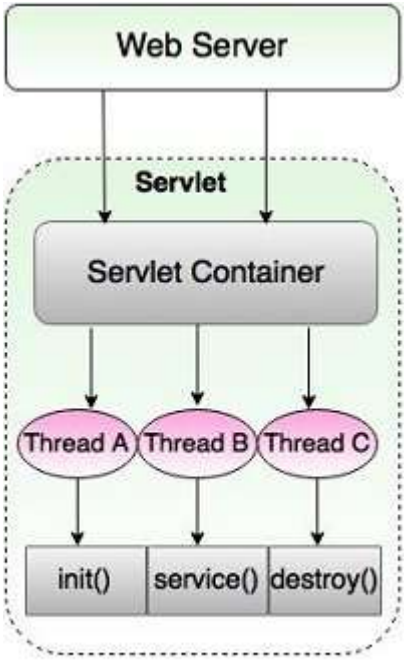


Fig: Servlet Life Cycle

1. Loads the Servlet Class

The **classloader** is responsible to load the Servlet class into the container. The servlet class is loaded when the first request comes from the web container.

2. Creates instance of Servlet

After loading of Servlet class into the container, the web container creates its instance. The instance of Servlet class is created only once in the servlet life cycle.

3. Calls the init( ) method

The **init( )** method performs some specific action constructor of a class. It is automatically called by Servlet container. It is called only once through the complete servlet life cycle.

Syntax:

public void init(ServletConfig config) throws ServletException

4. Calls the service( ) method

The **service( )** method performs actual task of servlet container. The web container is responsible to call the service method each time the request for servlet is received. The service( ) invokes the **doGet( ), doPost( ), doPut ( ), doDelete( )** methods based on the HTTP request.

Syntax:
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

5. Calls the destroy( ) method

The **destroy( )** method is executed when the servlet container remove the servlet from the container. It is called only once at the end of the life cycle of servlet.

Syntax:
public void destroy( )

Web.xml

<web-app>

Defines the specific version of the servlet API in use.

<display-name>

Defines the internal name of this web application that will appear in the servlet container management system.

<servlet>

Binds a name to a particular servlet class

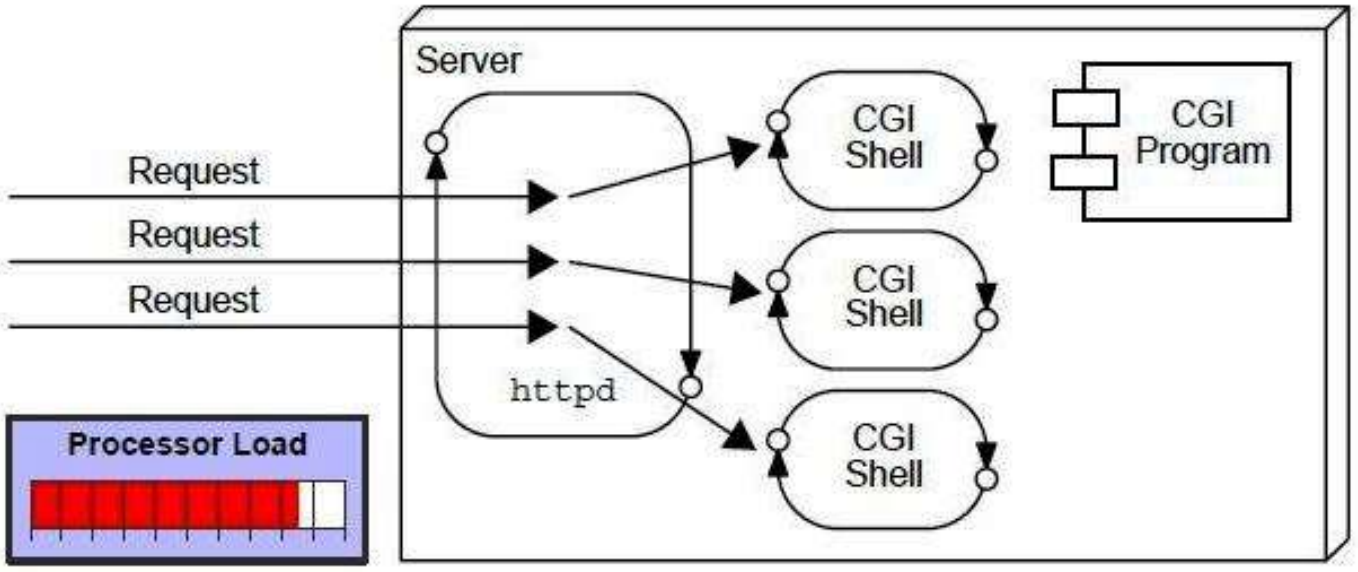
<servlet-mapping>

Binds a particular servlet to a URL from the root of the servlet.

<welcome-file-list>

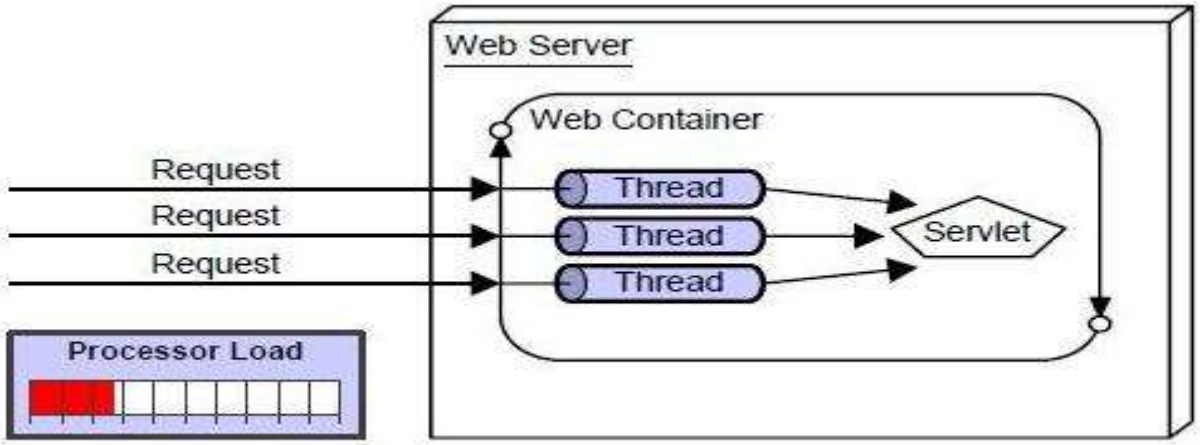
Defines a set of files to use as the landing page for the servlet. In this case, the view is set to be the landing page.

CGI(Commmon Gateway Interface)



- CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request.
- For each request, it starts a new process.

Advantage of Servlet



- **better performance**: because it creates a thread for each request not process.
- **Portability**: because it uses java language.
- **Robust**: Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
- **Secure**: because it uses java language..

Servlets

- The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets.
- All servlets must implement the Servlet interface, which defines life-cycle methods.
- The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

Interfaces of javax.servlet Package

- **Servlet :**
  Every servlet in java should implement servlet interface either directly or indirectly. ie, Servlet interface acts as a root interface for all java Servlets.
  This interface defines the most common methods (including life cycle methods) which are applicable for any servlet object.

- **ServletRequest :**
  ServletRequest object can be used to hold client data.
  ServletRequest interface defines several methods to acts as end users provided data from the request object.
  Ex: getParameter()

- **ServletResponse :**
  ServletResponse object can be used to prepare and send responds to the client. ServletResponse interface defines several methods which are required for preparation of response.
  Ex: getWriter(), setContentType()

- **ServletConfig :**
  For every servlet , web container creates one ServletConfig object to hold its configuration information like logical name of the Servlet , instantiation parameters etc.
  ServletConfig interface defines several methods to access servlets configuration information.
  Ex: getServletName()
  getInitParameter()

- **ServletContext :**
  For every web application, web container will create one ServletContext object to hold application level configuration information like name of the app'n and ServletContext parameter etc.
  Note : ServletConfig is per Servlet , where as ServletContext is per web application.

- **RequestDispatcher :**
  By using RequestDispatcher object we can dispatch the request from one component to another component.
  This interface defines two methods.
  1. forward()
  2.include()

Servlet Interface - Methods

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

GenericServlet class

- **GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces.
- It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

HttpServlet class

- The HttpServlet class extends the GenericServlet class and implements Serializable interface.
- It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

- **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
- **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
- **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
- **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.

Methods	Description
void **doGet(*HttpServletRequest* req, *HttpServletResponse* res)**	This method allows a Servlet to handle the ***get*** request.
void **doPost(*HttpServletRequest* req, *HttpServletResponse* res)**	This method allows a Servlet to handle the ***post*** request.
void **doPut(*HttpServletRequest* req, *HttpServletResponse* res)**	This method allows a Servlet to handle the **<i< style="box-sizing: border-box; margin: 0px; padding: 0px;">put</i<>** request.
void **doTrace(*HttpServletRequest* req, *HttpServletResponse* res)**	This method allows a Servlet to handle the ***trace*** request.
void **doHead(*HttpServletRequest* req, *HttpServletResponse* res)**	This method allows a Servlet to handle the **HTTP *head*** request.
void **doDelete(*HttpServletRequest* req, *HttpServletResponse* res)**	This method allows a Servlet to handle the ***delete*** request.
void **doOptions(*HttpServletRequest* req, *HttpServletResponse* res)**	This method allows a Servlet to handle the ***options*** request.

Steps to create a servlet

The servlet example can be created by three ways:

- By implementing Servlet interface,
- By inheriting GenericServlet class, (or)
- By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Steps to create a servlet example

- Create a directory structure
- Create a Servlet
- Compile the Servlet
- Create a deployment descriptor
- Start the server and deploy the project
- Access the servlet

Servlet

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServl
    etResponse res)throws ServletException,
    IOException
{
res.setContentType("text/html");//setting the
content type  of response
PrintWriter pw=res.getWriter();//get the stream to w
    rite the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");

pw.close();//closing the stream
}
}
```

web.xml

```
<web-app>

<servlet>
<servlet-name> FirstServlet
</servlet-name>
<servlet-class>
DemoServlet
</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>
FirstServlet
</servlet-name>
<url-pattern>
/welcome
</url-pattern>
</servlet-mapping>

  </web-app>
```

How Servlet works?

- The server checks if the servlet is requested **for the first time**.

- **If yes,** web container does the following tasks:
  1. loads the servlet class.
  2. instantiates the servlet class.
  3. calls the init method passing the ServletConfig object
- **else**
  1. calls the service method passing request and response objects

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

Servlet Annotations

- The Servlet API 3.0 introduces a new package called javax.servlet.Annotation which provides the annotation types which can be used for annotating a Servlet class. The annotations can replace the equivalent XML configuration in the web deployment descriptor file (i.e. web.xml) such as Servlet Declaration and Servlet Mapping. Servlet Containers will process the annotated classes at the time of application deploymen

ServletRequest to display the name of the user

index.html

```
<form action="welcome" method
  ="get">Enter your name
<input type="text" name="name"
    ><br>
<input type="submit" value="logi
   n">
</form>
```

DemoServ.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServ extends HttpServlet
{
public void doGet(HttpServletRequest req,
HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");

PrintWriter pw=res.getWriter();

String name=req.getParameter("name");//
will return value

pw.println("Welcome "+name);

pw.close();

}}
```

Exception Handling

- An exception is an event, which occurs during the execution of a program, which disrupts the normal flow of the program's instructions. The process of converting the system error messages into user-friendly error messages is known as Exception Handling.
- **Programmatically exception handling mechanism**: The approach to use try and catch block in the Java code to handle exceptions is known as programmatically exception handling mechanism
- **Declarative exception handling mechanism**: The approach to use the XML tags in the web.xml file to handle the exception is known as declarative exception handling mechanism. This mechanism is useful if exception is common for more than one servlet program

Servlet-Error Handling

- A customized content can be returned to a web-client when a servlet generates an error. Developers can do that by adding the <error-page /> elements in the web.xml. The following table describes the elements developers can define within an error-page element

Element	Required or Optional	Description
<error-code>	Optional	A valid HTTP error code. For e.g. 500 etc.
<exception-type>	Optional	A fully-qualified class name of a Java exception type. For e.g. java.lang.RuntimeException etc.
<location>	Required	The location of the resource which is displayed to the user in case of an error. For e.g. /myErrorPage.jsp etc.

Request Attributes	Type
javax.servlet.error.status_code	java.lang.Integer
javax.servlet.error.exception_type	java.lang.Class
javax.servlet.error.message	java.lang.String
javax.servlet.error.exception	java.lang.Throwable
javax.servlet.error.request_uri	java.lang.String
javax.servlet.error.servlet_name	java.lang.String

Servlet Exception

javax.servlet
Class ServletException

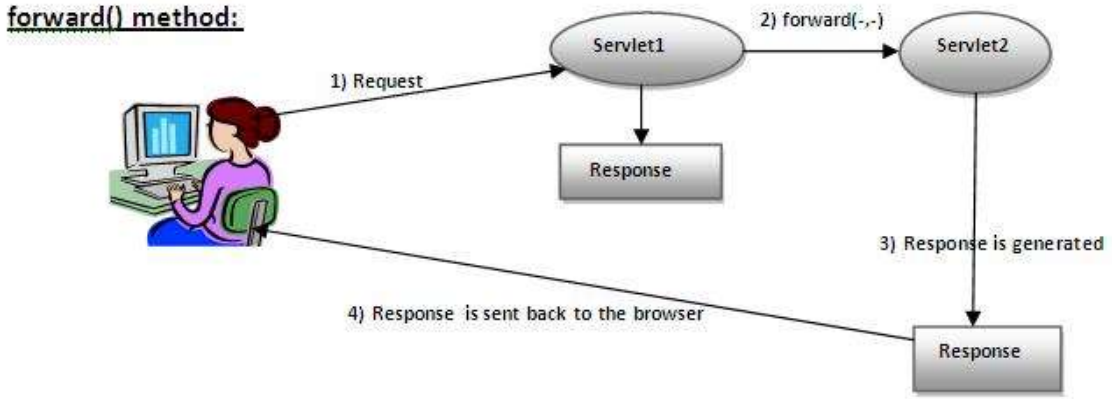
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      javax.servlet.ServletException

All Implemented Interfaces:
java.io.Serializable

RequestDispatcher in Servlet

- The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.
- This interface can also be used to include the content of another resource also.
- It is one of the way of servlet collaboration.

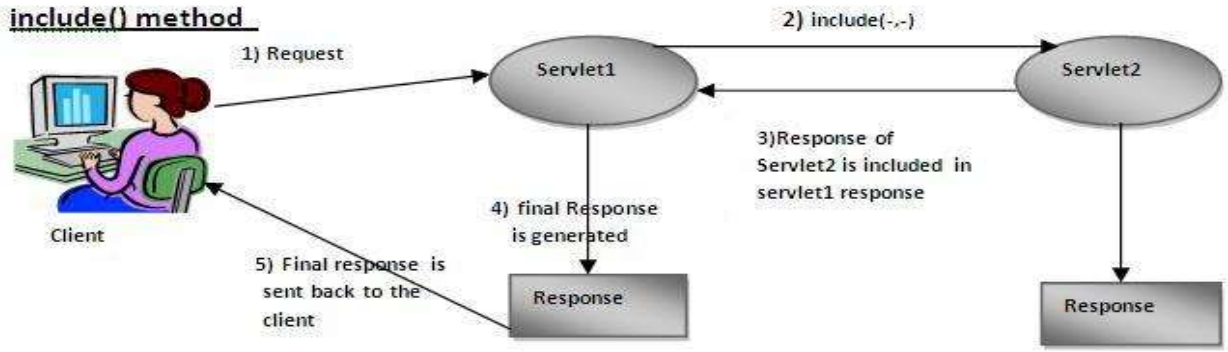
Methods of RequestDispatcher interface



- **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**

  Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

Methods of RequestDispatcher interface



- **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**

  Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

RequestDispatcher

How to get the object of RequestDispatcher

- The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher.

Syntax of getRequestDispatcher method

- public RequestDispatcher getRequestDispatcher(String resource);

Example of using getRequestDispatcher method

```
RequestDispatcher rd=
request.getRequestDispatcher("servlet2");  //servlet2 is the url-pattern of the second servlet
rd.forward(request, response);
    //method may be include or forward
```

SendRedirect in servlet

- The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
- It accepts relative as well as absolute URL.
- It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

SendRedirect in servlet

Syntax of sendRedirect() method

- public void sendRedirect(String URL)throws IOException;

Example of sendRedirect() method

- response.sendRedirect("http://www.cdac.in");

Difference between forward() and sendRedirect() method

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2"). forward(request,response);	Example: response.sendRedirect("servlet2");

ServletConfig Interface

- An object of ServletConfig is created by the web container for each servlet.
- This object can be used to get configuration information from web.xml file.
- Advantage - If the configuration information is modified from the web.xml file, we don't need to change the servlet.

Methods of ServletConfig interface

- **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
- **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
- **public String getServletName():**Returns the name of the servlet.
- **public ServletContext getServletContext():**Returns an object of ServletContext.
- **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

Using ServletConfig

<webapp>

web.xml

<servlet>

.....

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

.....

</servlet>

</web-app>

```
public void doGet(HttpServletRequest request,
 HttpServletResponse response)
{   response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    ServletConfig config=getServletConfig();
    String driver=config.getInitParameter("driver");
    out.print("Driver is: "+driver);
```

ServletContext Interface

- An object of **ServletContext is created by the web container at time of deploying the project**.
- This object can be used to get configuration information from web.xml file.
- There is only one ServletContext object per web application.
- If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

Usage of ServletContext Interface

- The object of ServletContext provides an interface between the container and servlet.
- The ServletContext object can be used to get configuration information from the web.xml file.
- The ServletContext object can be used to set, get or remove attribute from the web.xml file.
- The ServletContext object can be used to provide inter-application communication.

How to get the object of ServletContext interface

- **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
- **getServletContext() method** of GenericServlet class returns the object of ServletContext.

Using ServletContext

web.xml

```
<web-app>
 ......

  <context-param>
    <param-name>dname</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
  </context-param>
 ......
</web-app>
```

```
public void doGet(HttpServletRequest req,HttpServletResponse res)
{ res.setContentType("text/html");
  PrintWriter pw=res.getWriter();    //creating ServletContext object
  ServletContext context=getServletContext();    //Getting the value of the initi
                                                  alization parameter and printing it
  String driverName=context.getInitParameter("dname");
  pw.println("driver name is="+driverName);
```

Attribute in Servlet

- An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:
    1. request scope
    2. session scope
    3. application scope
- The servlet programmer can pass information from one servlet to another using attributes.

Attribute specific methods of ServletRequest, HttpSession and ServletContext interface

- **public void setAttribute(String name,Object object):**sets the given object in the application scope.
- **public Object getAttribute(String name):**Returns the attribute for the specified name.
- **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
- **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

ServletContext to set and get attribute

Servlet 1

```
public void doGet( …)…{
try{
 res.setContentType("text/html");
 PrintWriter out=res.getWriter();
 ServletContext context=getServletContext();
 context.setAttribute("company",“CDAC");
 out.println("Welcome to first servlet");  out.println("<a href='servlet2'>visit</a>");
 out.close();
```

servlet2

```
 public void doGet(…)   {
   try{
    res.setContentType("text/html");
    PrintWriter out=res.getWriter();
    ServletContext context=getServletContext();
    String n=(String)context.getAttribute("company");
    out.println("Welcome to "+n);
```

Session Tracking in Servlets

Session Management

- **Session Tracking** is a way to maintain state (data) of an user. Also known as **session management.**
- Http protocol is a stateless so we need to maintain state using session tracking techniques.
- Each time user requests to the server, server treats the request as the new request.
- So we need to maintain the state of an user to recognize a particular user.

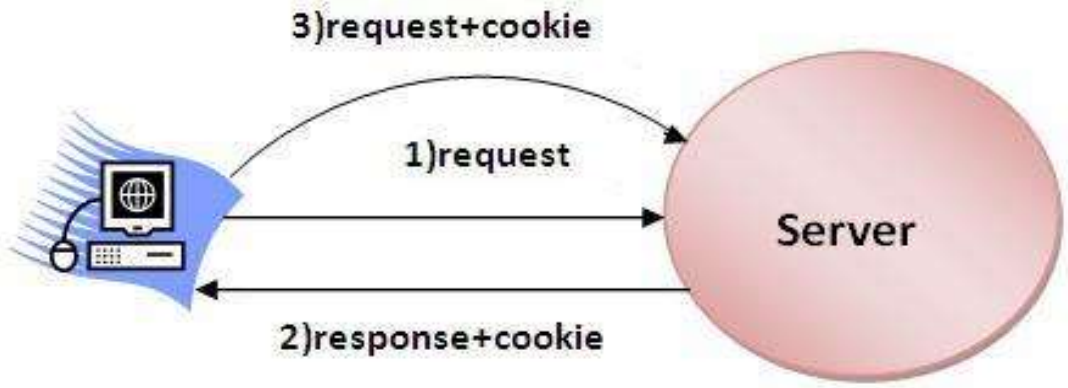
Session Tracking Techniques

- Cookies
- Hidden Form Field
- URL Rewriting
- HttpSession

Cookies in Servlet

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works



- By default, each request is considered as a new request.
- In cookies technique, we add cookie with response from the servlet.
- So cookie is stored in the cache of the browser.
- After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

Types of Cookie
Disadvantages & Advantages of Cookies

- **Non-persistent cookie -** It is **valid for single session** only. It is removed each time when user closes the browser.
- **Persistent cookie -** It is **valid for multiple session.** It is removed only if user logout or signout.

Advantages:

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantages

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

Using Cookies

- Cookie ck=new Cookie("user",“CDAC");//creating cookie obj ect

  response.addCookie(ck);//adding cookie in the response

- Cookie ck=new Cookie("user","");//deleting value of cookie

  ck.setMaxAge(0);//changing the maximum age to 0 seconds

- Cookie ck[]=request.getCookies();

  for(int i=0;i<ck.length;i++){

  out.print("
"+ck[i].getName()+" "+ck[i].getValue());

  //printing name and value of cookie

  }

Assignment

index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

Servlet 1

```
String user=request.getParameter("userName");
 out.print("Welcome “+user);
 Cookie ck=new Cookie("uname“,user);//creating cookie object
 response.addCookie(ck);//adding cookie in the response
 //creating submit button
 out.print("<form action='servlet2'>");
 out.print("<input type='submit' value='go'>");      out.print("</form>");
```

Servlet 2

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
Cookie ck[]=request.getCookies();
out.print("Hi …"+ck[0].getValue());
```

Hidden Form Field

- **a hidden (invisible) textfield** is used for maintaining the state of an user.
- This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Advantage of Hidden Form Field

- It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.

Syntax

<input type="hidden" name="user" value=“CDAC">

URL Rewriting

- In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.
- We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

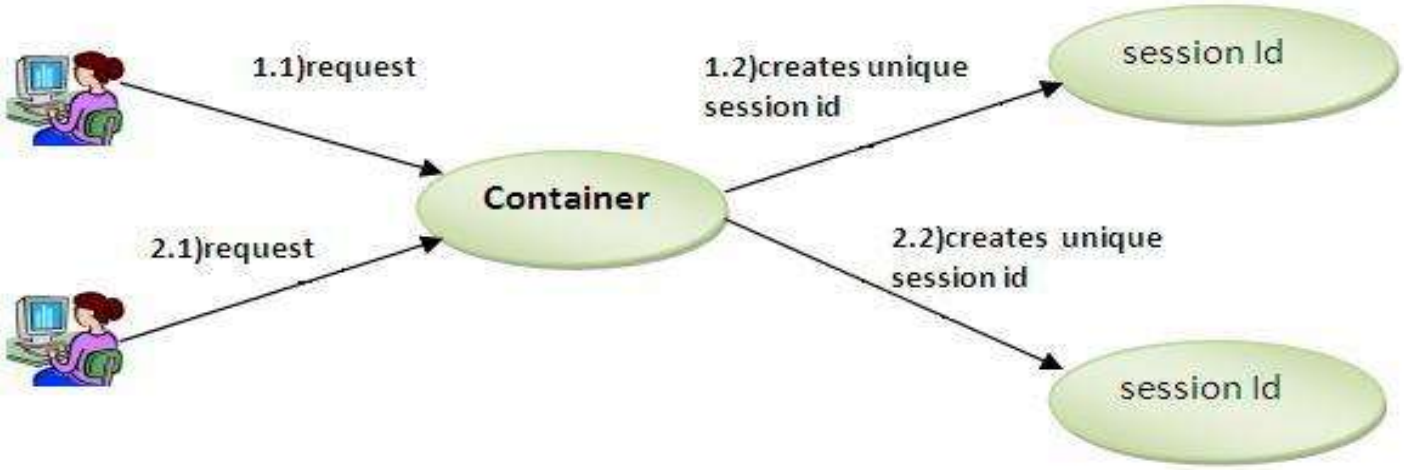
Advantage of URL Rewriting

- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

- It will work only with links.
- It can send Only textual information.

HttpSession interface



- In this case, container creates a session id for each user.
- The container uses this id to identify the particular user.

An object of HttpSession can be used to perform two tasks:

- bind objects
- view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

How to get the HttpSession object ?

HttpServletRequest interface provides two methods to get the object of HttpSession:

- **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
- **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

HttpSession Example

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

Servlet 1

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String user=request.getParameter("userName");
out.print("Welcome "+user);
HttpSession session=request.getSession();
session.setAttribute("name“,user);
out.print("<a href='servlet2'>visit</a>");
```

Servlet 2

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
HttpSession session=request.getSession(false);
String n=(String)session.getAttribute("uname");
out.print("Hello "+n);          out.close();
```

Session	Cookies
1.Session can store any type of data because the value is of data type of “object”	1.Cookies can store only “string” datatype.
2. These are stored at server side.	2. They are stored at client side.
3. Sessions are secured because it is stored in binary format/encrypted form and gets decrypted at server.	3. Cookie is non-secure since stored in text-format at client side.
4. Session is independent for every client i.e. individual for every client.	4. Cookies may or may not be individual for every client.
5. There is no limitation on the size or number of sessions to be used in an application.	5. Size of cookie is limited to 40 and number of cookies to be used is restricted to 20.
6. We cannot disable the sessions. Sessions can be used without cookies also.	6. Cookies can be disabled.
7. The disadvantage of session is that it is a burden or an overhead on server.	7. Since the value is in string format there is no security.
8. Sessions are called as Non-Persistent cookies because its life time can be set manually	8.We have persistent and non-persistent cookies.

JSP

What is JSP

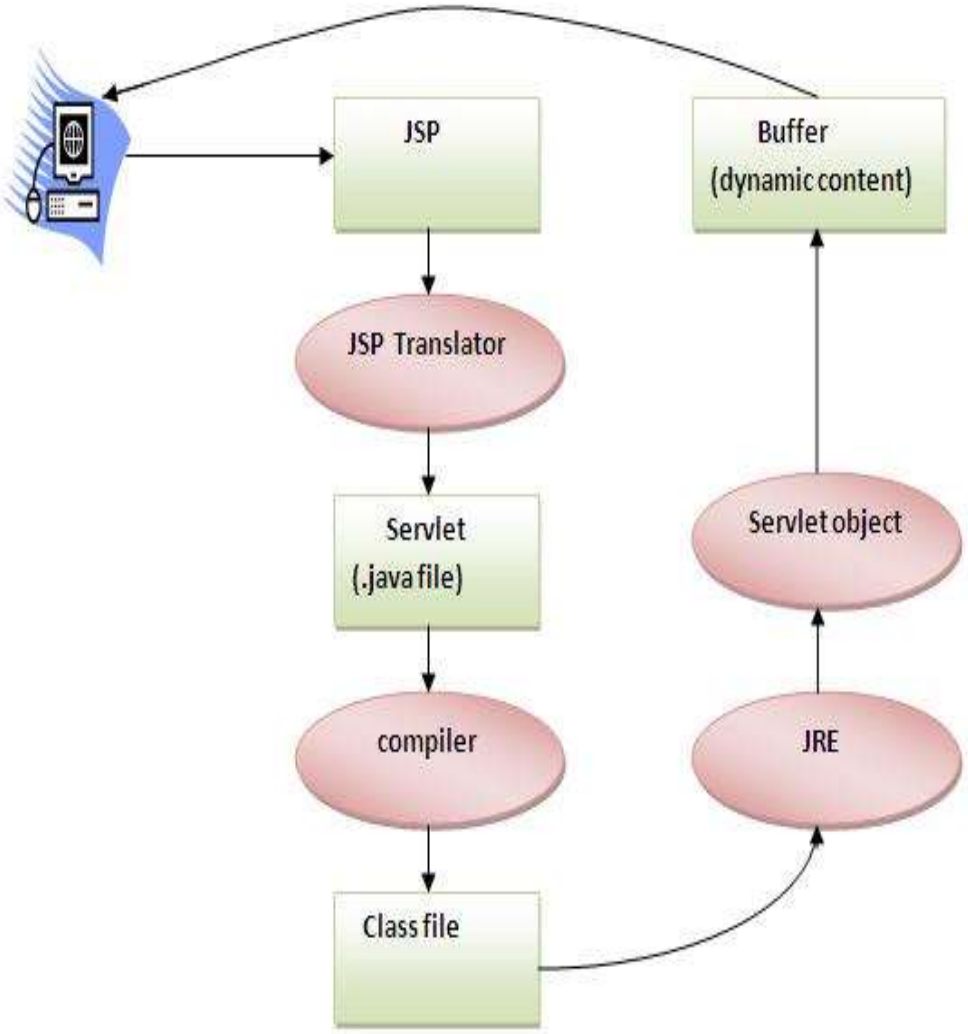
- JSP technology is used to create web application just like Servlet technology.
- It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.
- A JSP page consists of HTML tags and JSP tags.
- The jsp pages are easier to maintain than servlet because we can separate designing and development.

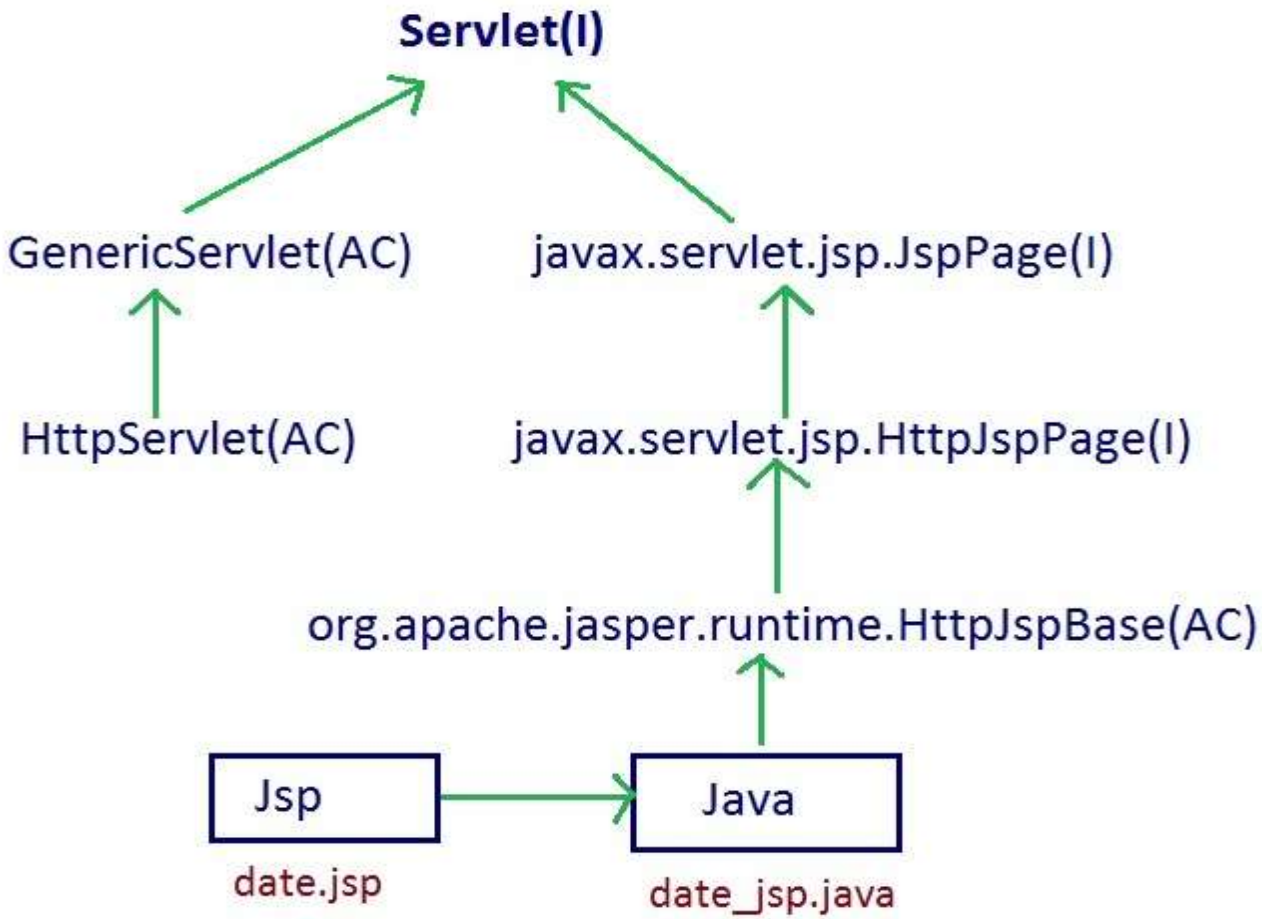
Advantage of JSP over Servlet

- **Extension to Servlet -** In addition to servlet functionality, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- **Easy to maintain -** separate our business logic with presentation logic
- **Fast Development -** No need to recompile and redeploy
- Less code than Servlet

Life cycle of a JSP Page

1. Translation of JSP Page
2. Compilation of JSP Page
3. Classloading (class file is loaded by the classloader)
4. Instantiation (Object of the Generated Servlet is created).
5. Initialization (jspInit() method is invoked by the container).
6. Request processing (_jspService() method is invoked by the container).
7. Destroy (jspDestroy() method is invoked by the container).




Servlet(I)
GenericServlet(AC)
javax.servlet.jsp.JspPage(I)
HttpServlet(AC)
javax.servlet.jsp.HttpJspPage(I)
org.apache.jasper.runtime.HttpJspBase(AC)
Jsp
Java
date.jsp
date_jsp.java

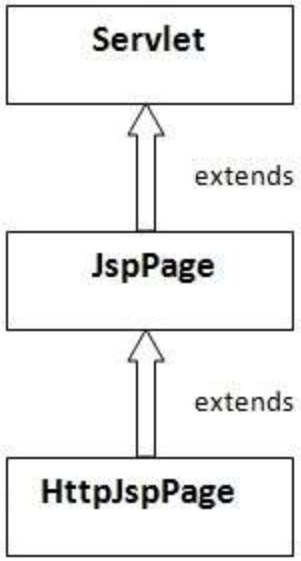
The JSP API

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

The javax.servlet.jsp package has two interfaces and classes.

1. JspPage
2. HttpJspPage

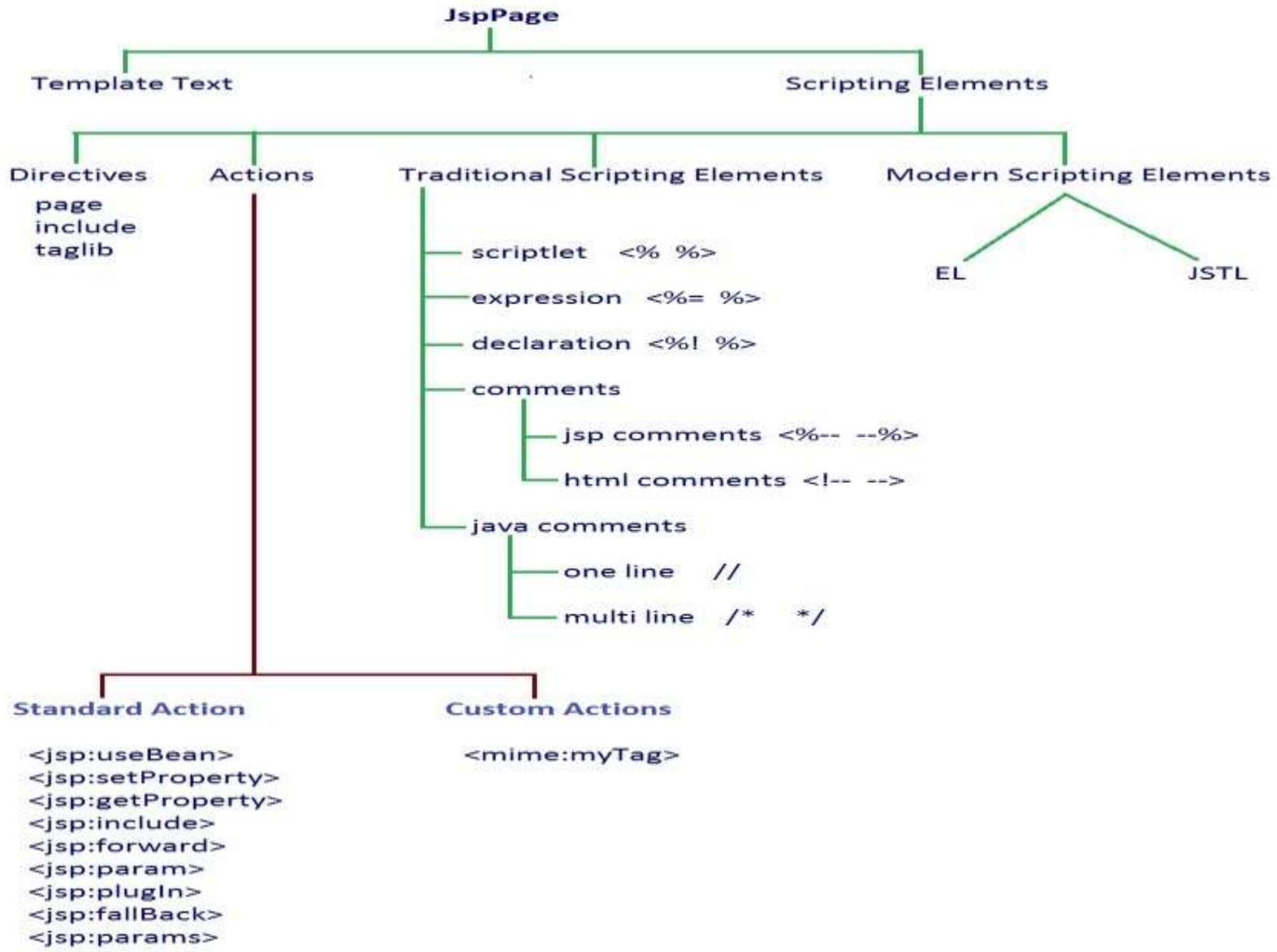


The classes are as follows:

JspWriter , PageContext, JspFactory, JspEngineInfo
JspException, JspError

All the generated servlet classes must implement the JspPage interface. It extends the Servlet interface.

The HttpJspPage interface provides the one life cycle method of JSP public void _jspService()


JspPage
Template Text
Scripting Elements
Directives
page
include
taglib
Actions
Traditional Scripting Elements
Modern Scripting Elements
EL
JSTL
scriptlet <% %>
expression <%= %>
declaration <%! %>
comments
jsp comments <%-- --%>
html comments <!-- -->
java comments
one line //
multi line /* */
Standard Action
Custom Actions
<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>
<jsp:include>
<jsp:forward>
<jsp:param>
<jsp:plugIn>
<jsp:fallBack>
<jsp:params>
<mime:myTag>

JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag.

Scripting elements

1. scriptlet tag
2. expression tag
3. declaration tag

Syntax

Declaration Tag :-It is used to declare variables.
Syntax:-
<%! Dec var %>
Example:-
<%! int var=10; %>
Java Scriplets :- It allows us to add any number of JAVA code, variables and expressions.
Syntax:-
<% java code %>

JSP Expression :- It evaluates and convert the expression to a string.
Syntax:-
<%= expression %>
Example:-
<% num1 = num1+num2 %>
JAVA Comments :- It contains the text that is added for information which has to be ignored.
Syntax:-
<% -- JSP Comments %>

JSP Tag	Brief Description	Tag Syntax
Directive	Specifies translation time instructions to the JSP engine.	<%@ directives %>
Declaration	Declaration Declares and defines methods and variables.	<%! variable dceclaration & method definition %>
Scriptlet	Allows the developer to write free-form Java code in a JSP page.	<% some Java code %>
Expression	Used as a shortcut to print values in the output HTML of a JSP page.	<%= an Expression %>
Action	Provides request-time instructions to the JSP engine.	<jsp:actionName />
Comment	Used for documentation and for commenting out parts of JSP code.	<%– any Text –%>

Expression	Explanation
<%= 500 %>	An integral literal
<%= anInt*3.5/100-500 %>	An arithmetic expression
<%= aBool %>	A Boolean variable
<%= false %>	A Boolean literal
<%= !false %>	A Boolean expression
<%= getChar() %>	A method returning a char
<%= Math.random() %>	A method returning a double
<%= aVector %>	A variable referring to a Vector object
<%= aFloatObj %>	A method returning a float
<%= aFloatObj.floatValue() %>	A method returning a float
<%= aFloatObj.toString() %>	A method that returns a String object

JSP scriptlet tag that prints the user name

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name=“user">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter(“user");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP expression tag

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name=“user"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

welcome.jsp

```
<html>
welcome.jsp
<body>
<%= "Welcome "+request.getParameter(“user") %>
</body>
  </html>
```

JSP Declaration Tag

- The **JSP declaration tag** is used *to declare fields and methods*.
- The jsp scriptlet tag can only declare variables not methods.

index.jsp(fields)

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

index.jsp(method)

```
<html>
<body>
<%!   int cube(int n){   return n*n*n*;   }  %>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSP Implicit Objects

- There are **9 jsp implicit objects**.
- These objects are *created by the web container* that are available to all the jsp pages.

Object	Type
out	JspWriter e.g. PrintWriter out=response.getWriter(); <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
request	HttpServletRequest e.g. <% String name=request.getParameter(“user");%>
response	HttpServletResponse e.g. response.sendRedirect("http://www.google.com");
config	ServletConfig e.g. String driverName=config.getInitParameter(“driver");
application	ServletContext e.g. String driverName=application.getInitParameter(“driver");
session	HttpSession e.g. String name=(String)session.getAttribute("user");

pageContext	PageContext e.g.
	pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
	pageContext.getAttribute("user",PageContext.SESSION_SCOPE);

page	Object e.g. <% (HttpServlet)page.log("message"); %> *[less used]
exception	Throwable e.g. Sorry following exception occured:<%= exception %>

JSP directives

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet

- page directive
- include directive
- taglib directive

Syntax of JSP Directive

<%@ directive attribute="value" %>

JSP page directive

Attributes of JSP page directive

import e.g. <%@ page import="java.util.Date" %>

contentType e.g. <%@ page contentType=application/msword %>
Defines the content type of the response

extends - defines the parent class that will be inherited by the generated servlet

info e.g. <%@ page info="composed by C-DAC" %>

buffer e.g. <%@ page buffer="16kb" %> to handle output generated by the JSP page

language e.g. scripting language used in the JSP page

isELIgnored e.g. <%@ page isELIgnored="true" %>//Now EL will be ignored

isThreadSafe e.g. <%@ page isThreadSafe="false" %> Servlet and JSP both are multithreaded. The default value of 'isThreadSafe' is true. If you make it false, the web container will serialize the multiple requests

errorPage e.g. <%@ page errorPage="myerrorpage.jsp" %>

isErrorPage e.g. <%@ page isErrorPage="true" %>

autoFlush

session

pageEncoding

Jsp Include Directive

- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.

Syntax of include directive

```
<%@ include file="resourceName" %>
```

e.g.

```
<%@ include file="header.html" %>
  Today is: <%=java.util.Calendar.getInstance().getTime() %
>
```

JSP Taglib directive

- The JSP taglib directive is used to define a tag library that defines many tags.
- We use the TLD (Tag Library Descriptor) file to define the tags.

Syntax JSP Taglib directive

```
<%@ taglib uri="urioftheteglibrary" prefix="prefixoftaglibrary" %>
```

e.g.

```
<%@ taglib uri="http://www.cdac.in/tags" prefix="mytag" %>
    <mytag:currentDate/>
```

Exception Handling in JSP

- The exception is normally an object that is thrown at runtime.
- Exception Handling is the process to handle the runtime errors.
- In JSP, there are two ways to perform exception handling:
- By **errorPage** and **isErrorPage** attributes of page directive
- By **<error-page>** element in web.xml file

Example of exception handling in jsp by the elements of page directive

index.jsp

```
<form action="process.jsp">
Enter First No: <input type="text" name="no1" /><br/><br/>
Enter Second No: :<input type="text" name="no2" /><br/><br/>
<input type="submit" value=“Divide"/>
</form>
```

process.jsp

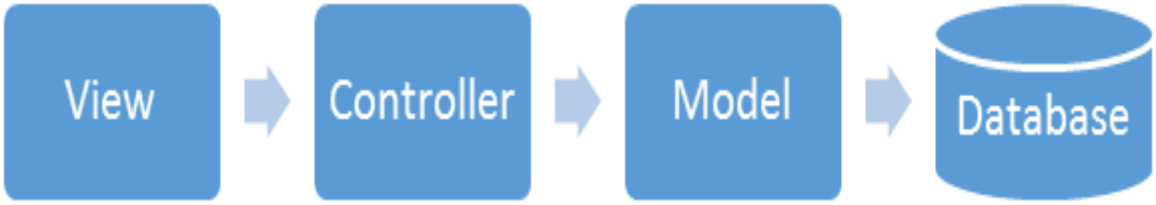
```
<%@ page errorPage="error.jsp" %>
<%
String num1=request.getParameter("no1");
String num2=request.getParameter("no2");
  int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print(“Result of a/b is: "+c);
  %>
```

error.jsp

```
<%@ page isErrorPage="true" %>
  <h3>An exception occured!</h3>
  Exception is: <%= exception %>
```

What is MVC?

- MVC is an architecture that separates business logic, presentation and data. In MVC,
- M stands for Model
- V stands for View
- C stands for controller.
- MVC is a systematic way to use the application where the flow starts from the view layer, where the request is raised and processed in controller layer and sent to model layer to insert data and get back the success or failure message.


View
Controller
Model
Database

Model Layer

- This is the data layer which consists of the business logic of the system.
- It consists of all the data of the application
- It also represents the state of the application.
- It consists of classes which have the connection to the database.
- The controller connects with model and fetches the data and sends to the view layer.
- The model connects with the database as well and stores the data into a database which is connected to it.

View Layer

- This is a presentation layer.
- It consists of HTML, JSP, etc. into it.
- It normally presents the UI of the application.
- It is used to display the data which is fetched from the controller which in turn fetching data from model layer classes.
- This view layer shows the data on UI of the application.

Controller Layer

- It acts as an interface between View and Model.
- It intercepts all the requests which are coming from the view layer.
- It receives the requests from the view layer and processes the requests and does the necessary validation for the request.
- This requests is further sent to model layer for data processing, and once the request is processed, it sends back to the controller with required information and displayed accordingly by the view.

Advantages of MVC Architecture

- Easy to maintain
- Easy to extend
- Easy to test
- Navigation control is centralized

Session 7 & 8 (Introduction to JDBC)

Introduction to JDBC

- The term JDBC stands for Java Database Connectivity. JDBC is a specification from Sun microsystems. JDBC is an API(Application programming interface) in Java that helps users to interact or communicate with various databases.
- The classes and interfaces of JDBC API allow the application to send the request to the specified database.

Purpose Of JDBC

- There are some enterprise applications created using the JAVA EE(Enterprise Edition) technology. These applications need to interact with databases to store application-specific information.
- Interacting with the database requires efficient database connectivity, which we can achieve using ODBC(Open database connectivity) driver. We can use this ODBC Driver with the JDBC to interact or communicate with various kinds of databases, like Oracle, MS Access, Mysql, and SQL, etc.

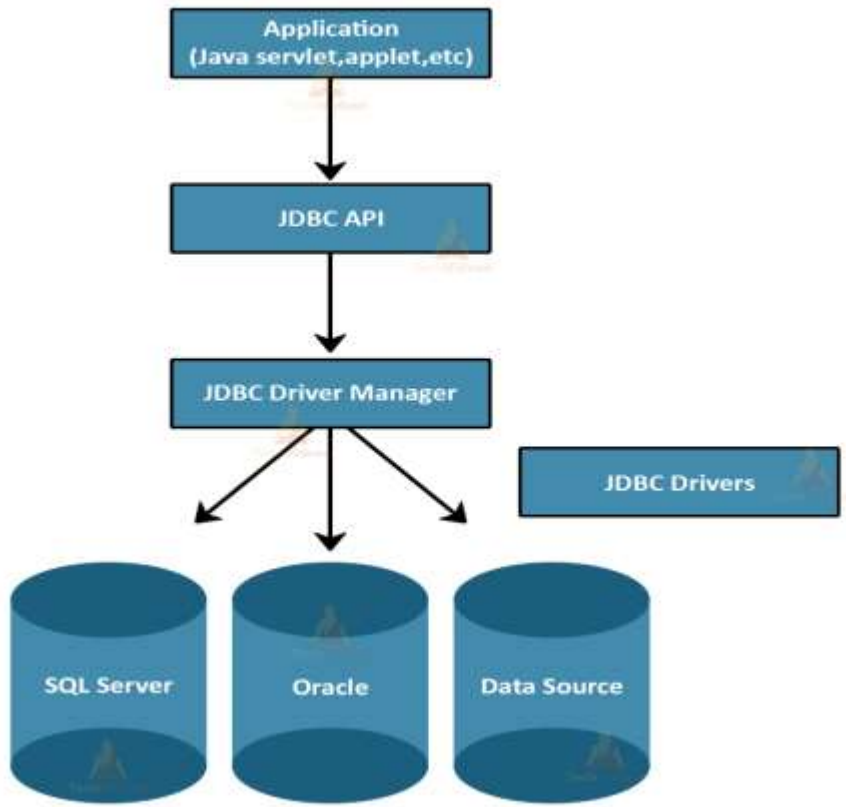
Applications of JDBC

JDBC is fundamentally a specification that provides a complete set of interfaces. These interfaces allow for portable access to an underlying database.

We can use Java to write different types of executables, such as:

- Java Applications
- Java Applets
- Enterprise JavaBeans (EJBs)
- Java Servlets
- Java ServerPages (JSPs)

Architecture of JDBC



Description of the Architecture:

1. Application: Application in JDBC is a Java applet or a Servlet that communicates with a data source.

2. JDBC API: JDBC API provides classes, methods, and interfaces that allow Java programs to execute SQL statements and retrieve results from the database. Some important classes and interfaces defined in JDBC API are as follows:

- DriverManager
- Driver
- Connection
- Statement
- PreparedStatement
- ResultSet

3. Driver Manager: The Driver Manager plays an important role in the JDBC architecture. The Driver manager uses some database-specific drivers that effectively connect enterprise applications to databases.

4. JDBC drivers: JDBC drivers help us to communicate with a data source through JDBC. We need a JDBC driver that can intelligently interact with the respective data source.

- **JDBC statement:**

  java.sql.Statement is an interface. It provides some methods through which we can submit SQL queries to the database. It will be implemented by driver implementation providers. createStatement() method on connection object will return statement object.

  JDBC ResultSet:

  java.sql.ResultSet also an interface and is used to retrieve SQL select query results. A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set. It provides getxxx() methods to get data from each iteration. Here XXX represents datatypes.

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

String sql="insert into emp values (? ,?,?)";

Why use PreparedStatement?

- Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Methods of PreparedStatement interface

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

BLOB and CLOB

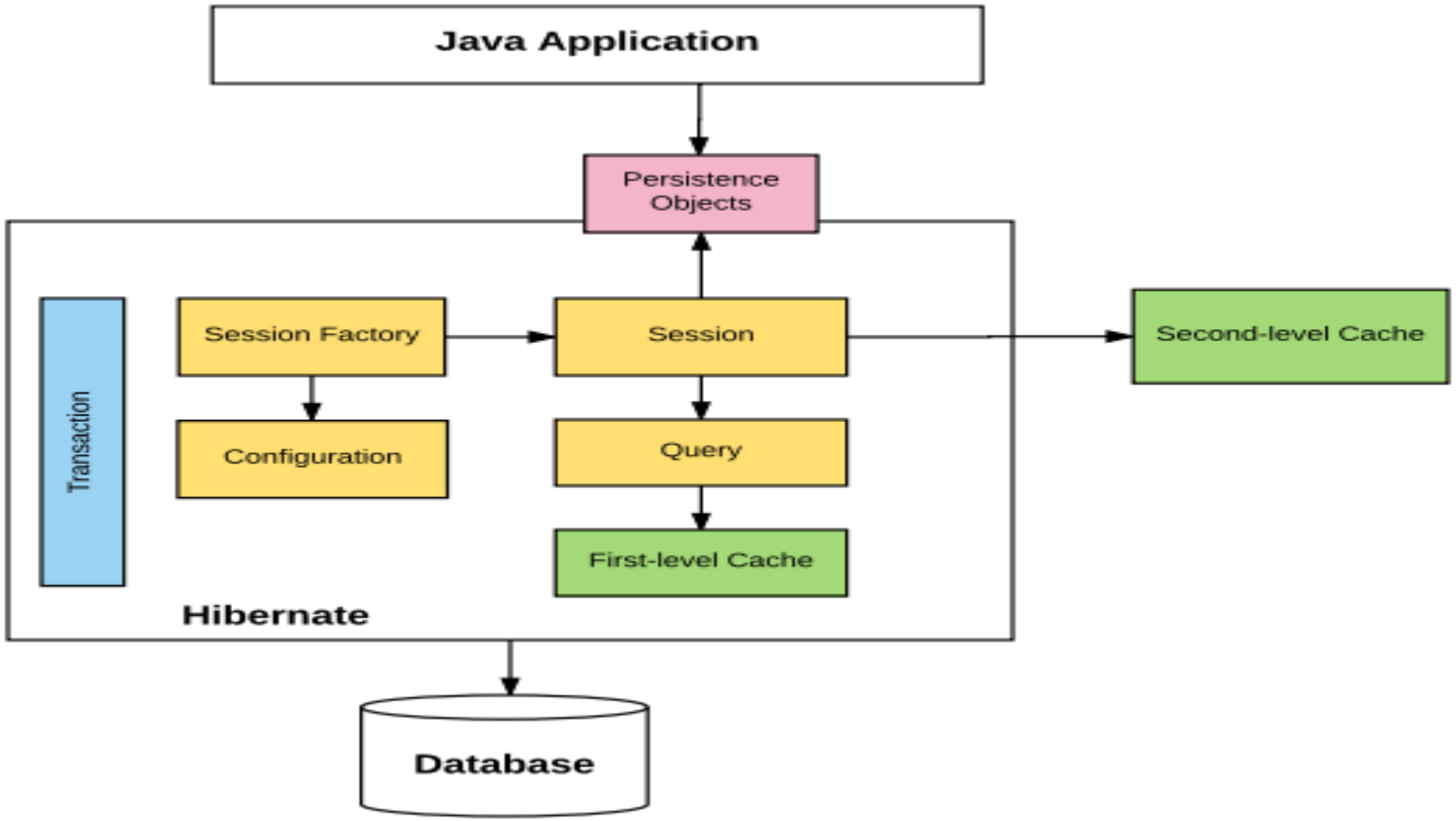
- BLOB Features Derby supports the java.sql.Blob interface and the BLOB-related methods in java.sql.PreparedStatement and java.sql.ResultSet. The getBlob methods of CallableStatement are not implemented.
- CLOB Features Derby supports the java.sql.Clob interface and the CLOB-related methods in java.sql.PreparedStatement and java.sql.ResultSet. The getClob methods of CallableStatement procedures are not implemented.

Session 9,10, 11 (Hibernate)

Introduction of Hibernate

- Hibernate is a framework in Java which comes with an abstraction layer and handles the implementations internally.
- The implementations include tasks like writing a query for CRUD operations or establishing a connection with the databases, etc.
- A framework is basically a software that provides abstraction on multiple technologies like JDBC, servlet, etc.
- Hibernate develops persistence logic, which stores and processes the data for longer use.
- It is lightweight and an ORM tool, and most importantly open-source which gives it an edge over other frameworks.

Hibernate Architecture



Hibernate Annotation

Annotation	Modifier	Description
@Entity		Marks a class as a Hibernate Entity (Mapped class)
@Table	Name	Maps this class with a database table specified by name modifier. If name is not supplied it maps the class with a table having same name as the class
@Id		Marks this class field as a primary key column
@GeneratedValue		Instructs database to generate a value for this field automatically
@Column	Name	Maps this field with table column specified by name and uses the field name if name modifier is absent
@ManyToMany	Cascade	Marks this field as the owning side of the many-to-many relationship and cascade modifier specifies which operations should cascade to the inverse side of relationship
	mappedBy	This modifier holds the field which specifies the inverse side of the relationship
@JoinTable	Name	For holding this many-to-many relationship, maps this field with an intermediary database join table specified by name modifier
	joinColumns	Identifies the owning side of columns which are necessary to identify a unique owning object
	inverseJoinColumns	Identifies the inverse (target) side of columns which are necessary to identify a unique target object
@JoinColumn	Name	Maps a join column specified by the name identifier to the relationship table specified by @JoinTable

Hibernate Dialects

- To connect to any database with hibernate, we need to specify the SQL dialect class in hibernate.cfg.xml

  Ex: To connect to oracle database we need to specify oracle dialect class in configuration xml as below.

- **Dialect** class is java class, which contains code to map between java language data type database data type.
- All Dialect classes extend the Dialect abstract class.
- Dialect is used to convert HQL(Hibernate Query Language) statements to data base specific statements

Hibernate Mapping

- Mapping file is the heart of hibernate application.
- Every ORM tool needs this mapping, mapping is the mechanism of placing an object properties into column’s of a table.
- Mapping can be given to an ORM(Object Relational Mapping) tool either in the form of an XML or in the form of the annotations.
- The mapping file contains mapping from a pojo class name to a table name and pojo class variable names to table column names.
- While writing an hibernate application, we can construct one or more mapping files, mean a hibernate application can contain any number of mapping files

Mapping can be done using 2 ways,

1. XML
2. Annotations.

Hibernate Relations

Types of relationships

1. One-To-One
2. Many-To-One
3. Many-To-Many
4. One-To-Many

HQL (Hibernate Query Language)

- Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL. So it is database independent query language.

Advantage of HQL

There are many advantages of HQL. They are as follows:

- database independent
- supports polymorphic queries
- easy to learn for Java Programmer

The query interface provides many methods. There is given commonly used methods:

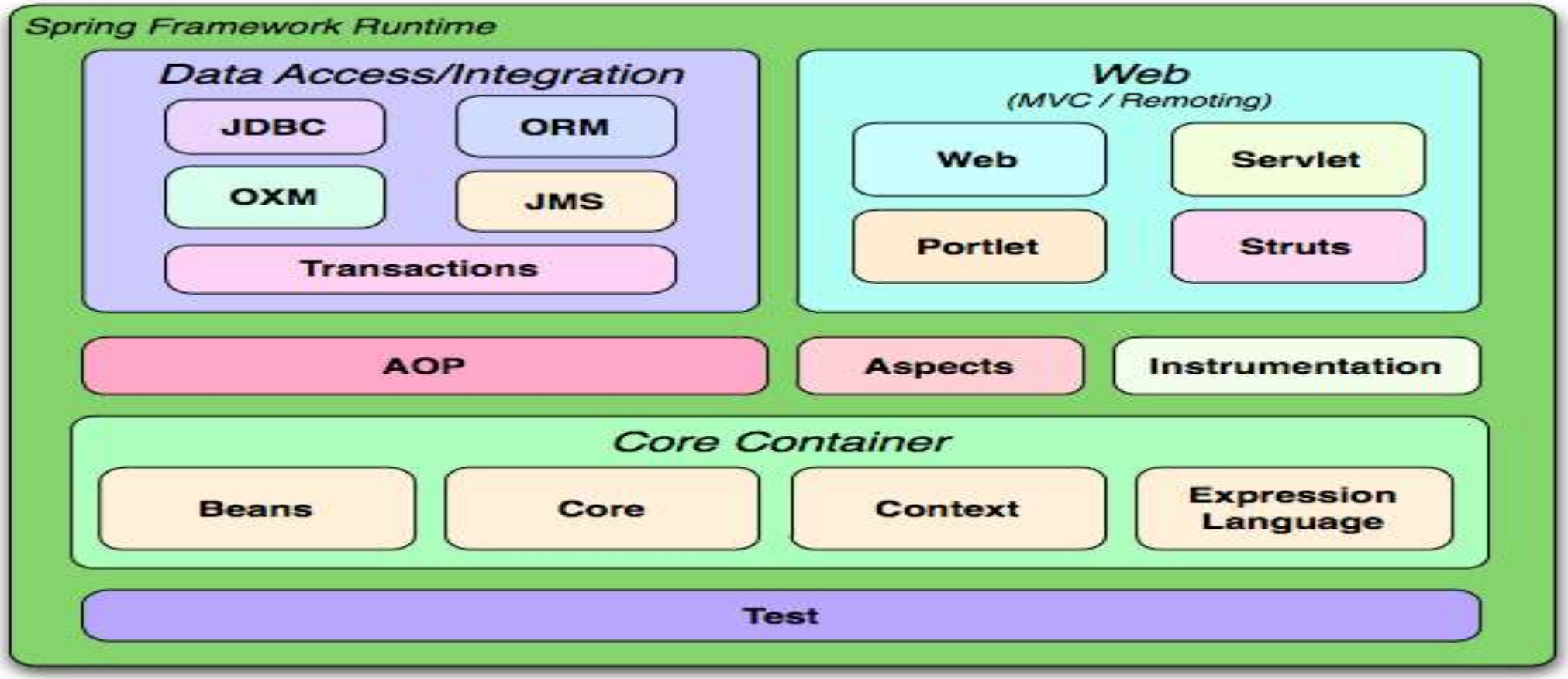
- **public int executeUpdate()** is used to execute the update or delete query.
- **public List list()** returns the result of the ralation as a list.
- **public Query setFirstResult(int rowno)** specifies the row number from where record will be retrieved.
- **public Query setMaxResult(int rowno)** specifies the no. of records to be retrieved from the relation (table).
- **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
- **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.

Session 12 ,13 and 14
(Spring Framework Overview)

Why Spring?

- Spring is a powerful, lightweight framework used for application development.
- In broader terms, you can say that the Spring framework is a well-defined tool that supports several web applications using Java as a programming language.
- Lightweight
- Aspect-Oriented Programming (AOP)
- Transaction Management
- Container
- Dependency Injection
- Integration With Other Frameworks

Spring Architecture



Spring Annotation

Annotation	Package Detail/Import statement
@Service	import org.springframework.stereotype.Service;
@Repository	import org.springframework.stereotype.Repository;
@Component	import org.springframework.stereotype.Component;
@Autowired	import org.springframework.beans.factory.annotation.Autowired;
@Transactional	import org.springframework.transaction.annotation.Transactional;
@Scope	import org.springframework.context.annotation.Scope;

Spring MVC Annotations

@Controller	import org.springframework.stereotype.Controller;
@RequestMapping	import org.springframework.web.bind.annotation.RequestMapping;
@PathVariable	import org.springframework.web.bind.annotation.PathVariable;
@RequestParam	import org.springframework.web.bind.annotation.RequestParam;
@ModelAttribute	import org.springframework.web.bind.annotation.ModelAttribute;
@SessionAttributes	import org.springframework.web.bind.annotation.SessionAttributes;

Spring Security Annotations

@PreAuthorize	import org.springframework.security.access.prepost.PreAuthorize;

What Is Inversion of Control?

- Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.
- In contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. **If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.**

Advantages

- decoupling the execution of a task from its implementation
- making it easier to switch between different implementations
- greater modularity of a program
- greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts

IoC container

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

Session 15 and 16
(Spring Boot Introduction)

- Micro Service is an architecture that allows the developers to develop and deploy services independently. Each service running has its own process and this achieves the lightweight model to support business applications.

Reasons for using Microservice:

Micro services offers the following advantages to its developers −

- Easy deployment
- Simple scalability
- Compatible with Containers
- Minimum configuration
- Lesser production time

What is Spring Boot?

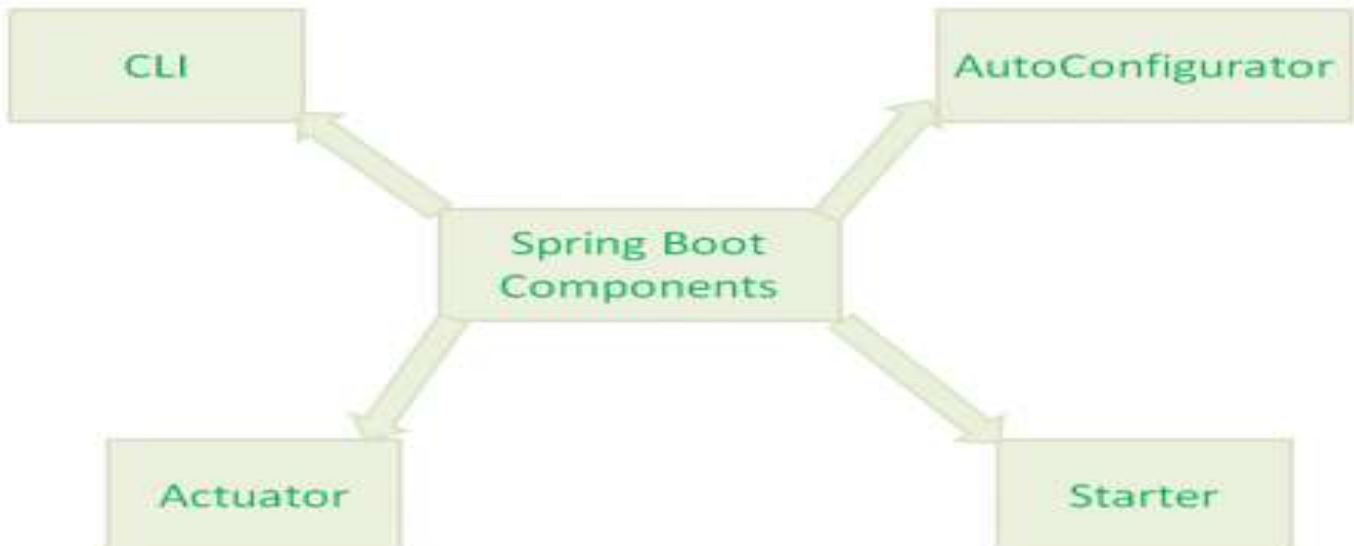
- Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**.
- You can get started with minimum configurations without the need for an entire Spring configuration setup.

Advantages

- Spring Boot offers the following advantages to its developers −
- Easy to understand and develop spring applications
- Increases productivity
- Reduces the development time

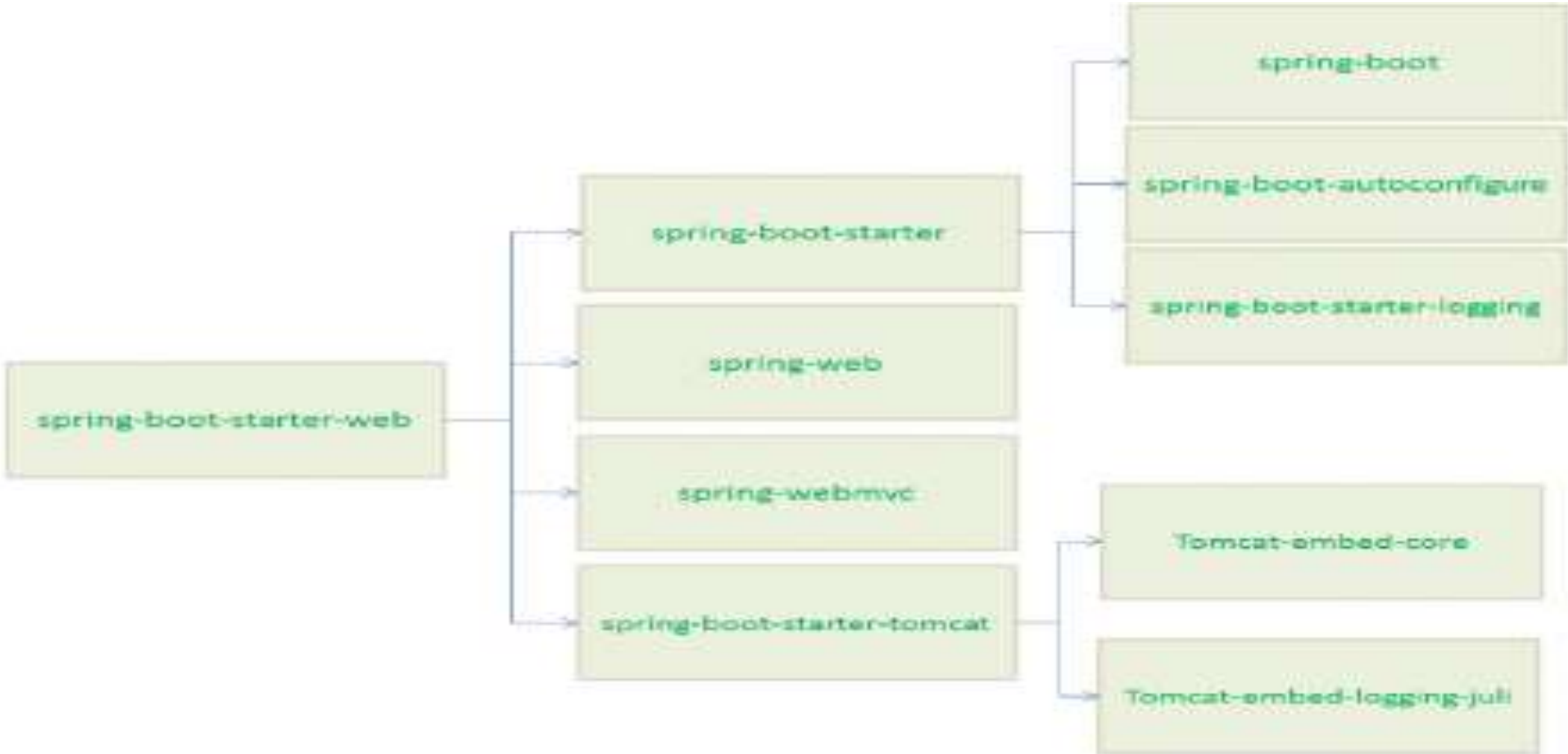
Why Spring Boot?

- To ease the Java-based applications Development, Unit Test and Integration Test Process.
- To reduce Development, Unit Test and Integration Test time by providing some defaults.
- To increase Productivity.


CLI
AutoConfigurator
Spring Boot Components
Actuator
Starter

Spring Boot Starters

- Starters are a set of convenient dependency descriptors which we can include in our application.
- Spring Boot provides built-in starters which makes development easier and rapid. For example, if we want to get started using Spring and JPA for database access, just include the **spring-boot-starter-data-jpa** dependency in your project.
- Starter should follow a naming pattern like: **spring-boot-starter**-*, where * is a particular type of application. This naming structure is intended to help when you need to find a starter.


spring-boot-starter-web
spring-boot-starter
spring-web
spring-webmvc
spring-boot-starter-tomcat
spring-boot
spring-boot-autoconfigure
spring-boot-starter-logging
Tomcat-embed-core
Tomcat-embed-logging-juli


@SpringBootApplication
=
@Configuration
+
@ComponentScan
+
@EnableAutoConfiguration

Spring Boot Auto Configuration

Spring Boot automatically configures a spring application based on dependencies present or not present in the classpath as a jar, beans, properties, etc.

It makes development easier and faster as there is no need to define certain beans that are included in the auto-configuration classes.

A typical MVC database driven Spring MVC application requires a lot of configuration such as dispatcher servlet, a view resolver, Jackson, data source, transaction manager, among many others.

- ✓Spring Boot auto-configures a Dispatcher Servlet if Spring MVC jar is on the classpath.
- ✓Auto-configures the Jackson if Jackson jar is on the classpath.
- ✓Auto-configures a Data Source if Hibernate jar is on the classpath.

Spring Boot	Spring MVC
Spring Boot is a module of Spring for packaging the Spring-based application with sensible defaults.	**Spring MVC** is a model view controller-based web framework under the Spring framework.
It provides default configurations to build **Spring-powered** framework.	It provides **ready to use** features for building a web application.
There is no need to build configuration manually.	It requires build configuration manually.
There is **no requirement** for a deployment descriptor.	A Deployment descriptor is **required**.
It avoids boilerplate code and wraps dependencies together in a single unit.	It specifies each dependency separately.
It **reduces** development time and increases productivity.	It takes **more** time to achieve the same.

(ThymeLeaf and Spring JPA)

Thymeleaf

- It is a server side Java template engine for web application. It's main goal is to bring elegant natural templates to your web application.
- It can be integrate with Spring Framework and ideal for HTML5 Java web applications.

In order to use Thymeleaf we must add it into our pom.xml file like:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

- Thymeleaf is a Java-based library used to create a web application. It provides a good support for serving a XHTML/HTML5 in web applications.

It contains 6 types of templates as given below −

- XML
- Valid XML
- XHTML
- Valid XHTML
- HTML5
- Legacy HTML5

Spring Data JPA

- Spring Data JPA API provides JpaTemplate class to integrate spring application with JPA.
- JPA (Java Persistent API) is the sun specification for persisting objects in the enterprise application. It is currently used as the replacement for complex entity beans.
- The implementation of JPA specification are provided by many vendors such as:
- Hibernate
- Toplink
- iBatis
- OpenJPA etc.

When to use Spring Data JPA?

- quickly create a JPA-based repository layer that is mainly for CRUD operations, and you do not want to create abstract DAO, implementing interfaces, Spring Data JPA is a good choice.

CrudRepository	JpaRepository
CrudRepository does not provide any method for pagination and sorting.	JpaRepository extends PagingAndSortingRepository. It provides all the methods for implementing the pagination.
It works as a **marker** interface.	JpaRepository extends both **CrudRepository** and **PagingAndSortingRepository**.
It provides CRUD function only. For example **findById(), findAll(),** etc.	It provides some extra methods along with the method of PagingAndSortingRepository and CrudRepository. For example, **flush(), deleteInBatch().**
It is used when we do not need the functions provided by JpaRepository and PagingAndSortingRepository.	It is used when we want to implement pagination and sorting functionality in an application.

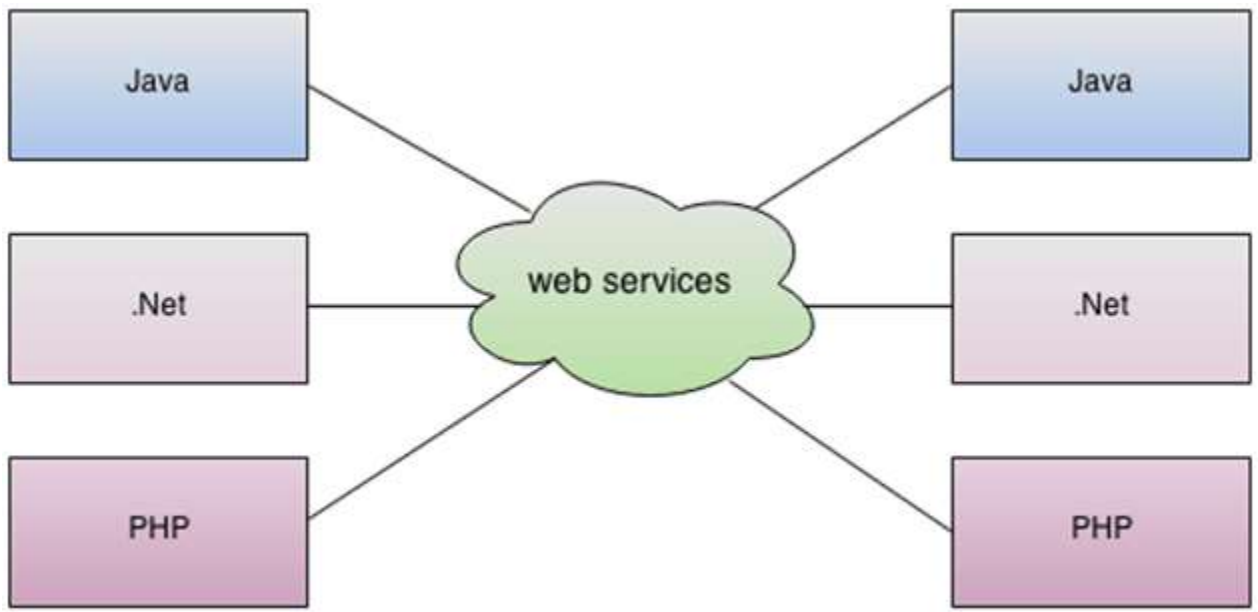
Building REST services with Spring

Introduction to web services

- Webservices in java are used everywhere nowadays.
- When human interacts with any web page, it involves request and response via HTML.
- When you interact with the webpage, browser sends a request and then renders response and shows in form of HTML.
- Similarly, web services also involve request and response but in the form of XML or JSON or plain text.
- It generally used for other applications or programs to consume and make use of information.

A Web Service is can be defined by following ways:

- It is a client-server application or application component for communication.
- The method of communication between two devices over the network.
- It is a software system for the interoperable machine to machine communication.
- It is a collection of standards or protocols for exchanging information between two devices or application.


Java
.Net
PHP
web services
Java
.Net
PHP

soap vs restful web services

SOAP	REST
SOAP is a **protocol**.	REST is an **architectural style**.
SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
JAX-WS (**Jakarta XML Web Services**)is the java API for SOAP web services.	**JAX-RS** (**AVA API for RESTful Web Services**)is the java API for RESTful web services.
SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.
SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.

What is REST architecture?

- REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.
- In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.
-

HTTP methods

- Following four HTTP methods are commonly used in REST based architecture.
- **GET** − Provides a read only access to a resource.
- **POST** − Used to create a new resource.
- **DELETE** − Used to remove a resource.
- **PUT** − Used to update a existing resource or create a new resource.

Spring Security

Introduction

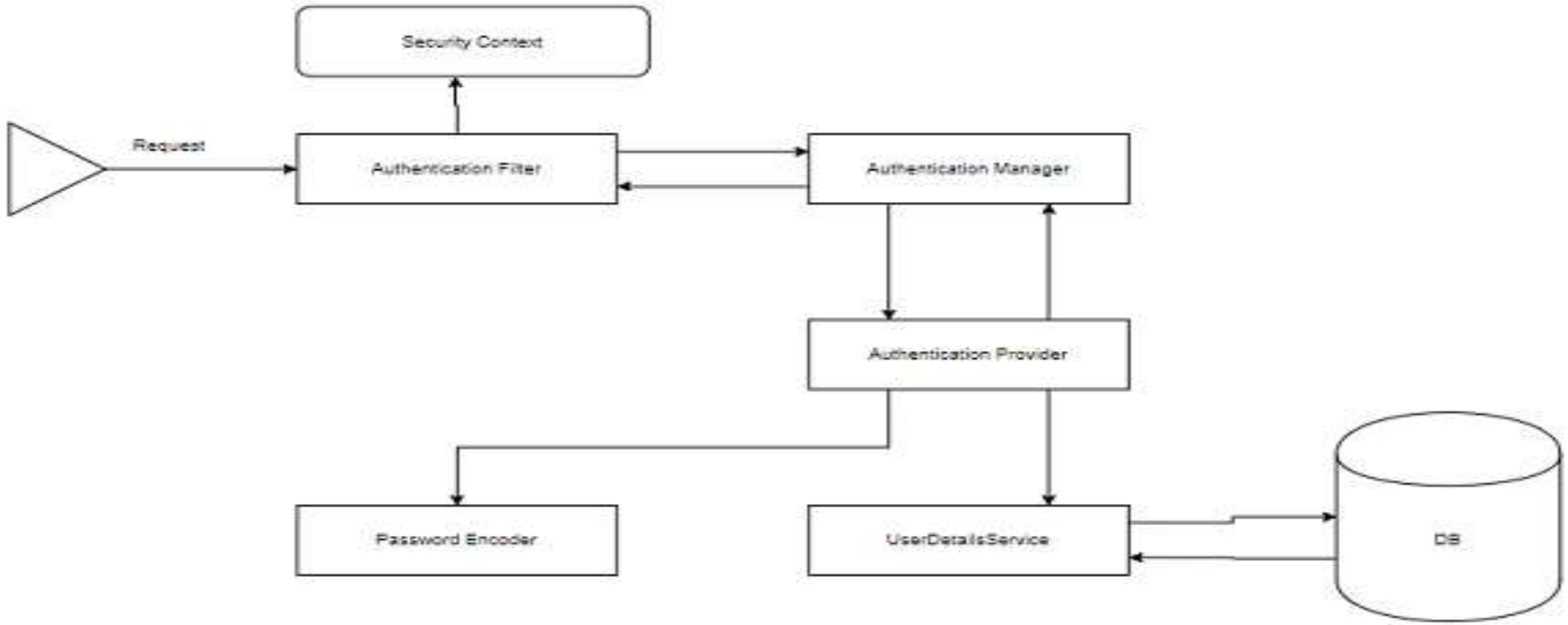
- Spring Security is a framework which provides various security features like: authentication, authorization to create secure Java Enterprise Applications.
- It is a sub-project of Spring framework which was started in 2003 by Ben Alex. Later on, in 2004, It was released under the Apache License as Spring Security 2.0.0.
- It overcomes all the problems that come during creating non spring security applications and manage new server environment for the application.
- This framework targets two major areas of application are authentication and authorization. Authentication is the process of knowing and identifying the user that wants to access.

Authorization

- Authorization is the process to allow authority to perform actions in the application.
- We can apply authorization to authorize web request, methods and access to individual domain.
- Technologies that support Spring Security Integration
- Spring Security framework supports wide range of authentication models. These models either provided by third parties or framework itself.

Advantages

- Comprehensive support for authentication and authorization.
- Protection against common tasks
- Servlet API integration
- Integration with Spring MVC
- Portability
- CSRF protection[**Cross Site Request Forgery]**
- Java Configuration support


Security Context
Request
Authentication Filter
Authentication Manager
Authentication Provider
Password Encoder
UserDetailsService
DB

AuthenticationFilter

- This is the filter that intercepts requests and attempts to authenticate it. In Spring Security, it converts the request to an Authentication Object and delegates the authentication to the AuthenticationManager.

- **AuthenticationManager**

It is the main strategy interface for authentication. It uses the lone method authenticate() to authenticate the request. The authenticate() method performs the authentication and returns an Authentication Object on successful authentication or throw an AuthenticationException in case of authentication failure. If the method can’t decide, it will return null. The process of authentication in this process is delegated to the AuthenticationProvider which we will discuss next.

- **AuthenticationProvider**

The AuthenticationManager is implemented by the ProviderManager which delegates the process to one or more AuthenticationProvider instances. Any class implementing the AuthenticationProvider interface must implement the two methods – authenticate() and supports(). First, let us talk about the supports() method. It is used to check if the particular authentication type is supported by our AuthenticationProvider implementation class. If it is supported it returns true or else false. Next, the authenticate() method. Here is where the authentication occurs. If the authentication type is supported, the process of authentication is started. Here is this class can use the loadUserByUsername() method of the **UserDetailsService** implementation. If the user is not found, it can throw a UsernameNotFoundException.

- **UserDetailsService**

It is one of the core interfaces of Spring Security. The authentication of any request mostly depends on the implementation of the UserDetailsService interface. It is most commonly used in database backed authentication to retrieve user data. The data is retrieved with the implementation of the lone loadUserByUsername() method where we can provide our logic to fetch the user details for a user. The method will throw a UsernameNotFoundException if the user is not found.

- **PasswordEncoder**

Until Spring Security 4, the use of PasswordEncoder was optional. The user could store plain text passwords using in-memory authentication. But Spring Security 5 has mandated the use of PasswordEncoder to store passwords. This encodes the user's password using one its many implementations. The most common of its implementations is the BCryptPasswordEncoder. Also, we can use an instance of the NoOpPasswordEncoder for our development purposes. It will allow passwords to be stored in plain text. But it is not supposed to be used for production or real-world applications.

- **Spring Security Context**
- This is where the details of the currently authenticated user are stored on successful authentication. The authentication object is then available throughout the application for the session. So, if we need the username or any other user details, we need to get the SecurityContext first. This is done with the SecurityContextHolder, a helper class, which provides access to the security context. We can use the setAuthentication() and getAuthentication() methods for storing and retrieving the user details respectively.

Spring Testing

UNIT TESTS

- Running in isolation will sometimes require that you mock your dependencies based on the class you are testing. By doing this, you're allowing yourself to test very specific test cases end-to-end without having to worry about the overhead of service or domain layers. Hence, using Mockito or more specifically, the Mockito.mock() method which mocks object classes and DOES NOT replace any objects on the web context such as @MockBean.

INTEGRATION TESTS

- Whereas, integration testing focuses on integrating different layers of the application such as the database. In regards to databases, most people utilize an in memory database such as H2 to test their domain layers/repositories. Integration tests SHOULD not contain any mocking and both types of testing should be run separately. This isn't to say that integration tests can not contain any mocking, but it isn't common since you already have isolated unit tests that test the various layers of your application which contain mocked dependencies!

Difference

Unit Testing	Integration Testing
In unit testing each module of the software is tested separately.	In integration testing all modules of the the software are tested combined.
In unit testing tester knows the internal design of the software.	In integration testing doesn’t know the internal design of the software.
Unit testing is performed first of all testing processes.	Integration testing is performed after unit testing and before system testing.
Unit testing is a white box testing.	Integration testing is a black box testing.
Unit testing is basically performed by the developer.	Integration testing is performed by the tester.
Detection of defects in unit testing is easy.	Detection of defects in integration testing is difficult.
It tests parts of the project without waiting for others to be completed.	It tests only after the completion of all parts.
Unit testing is less costly.	Integration testing is more costly.

- @Test is used to signal that the annotated method is a test method.
- @Test methods must not be private or static and must not return a value.
- @Test methods may optionally declare parameters to be resolved by ParameterResolvers.
- @Test may also be used as a meta-annotation in order to create a custom composed annotation that inherits the semantics of @Test

@InjectMocks

- Allows shorthand mock
- Minimizes repetitive mock and spy injection.
- Mockito will try to inject mocks only either by constructor injection, property injection or setter injection in order and as described below. If any of the following strategy fail, then Mockito **won't report failure**; i.e. you will have to provide dependencies yourself.

1. **Constructor injection**; the biggest constructor is chosen, then arguments are resolved with mocks declared in the test only. If the object is successfully created with the constructor, then **Mockito won't try the other strategies**. Mockito has decided to no corrupt an object if it has a parametered constructor. Note: If arguments can not be found, then null is passed. If non-mockable types are wanted, then constructor injection won't happen. In these cases, you will have to satisfy dependencies yourself.
2. **Property setter injection**; mocks will first be resolved by type (if a single type match injection will happen regardless of the name), then, if there is several property of the same type, by the match of the property name and the mock name. Note 1: If you have properties with the same type (or same erasure), it's better to name all @Mock annotated fields with the matching properties, otherwise Mockito might get confused and injection won't happen.
3. **Field injection**; mocks will first be resolved by type (if a single type match injection will happen regardless of the name), then, if there is several property of the same type, by the match of the field name and the mock name. Note 1: If you have fields with the same type (or same erasure), it's better to name all @Mock annotated fields with the matching fields, otherwise Mockito might get confused and injection won't happen.

@Mock

- •Allows shorthand mock creation.
- •Minimizes repetitive mock creation code.
- •Makes the test class more readable.
- •Makes the verification error easier to read because the field name is used to identify the mock.
- •Automatically detects static mocks of type MockedStatic and infers the static mock type of the type parameter.

- ExtendWith is a repeatable annotationthat is used to register extensions for the annotatedtest class, test interface, test method, parameter, or field.

- Annotated parameters are supported in test class constructors, in testmethods, and in @BeforeAll, @AfterAll, @BeforeEach,and @AfterEach lifecycle methods.

Spring Boot AOP

- The application is generally developed with multiple layers. A typical Java application has the following layers:

- **Web Layer**: It exposes the services using the REST or web application.
- **Business Layer**: It implements the business logic of an application.
- **Data Layer**: It implements the persistence logic of the application.
- The responsibility of each layer is different, but there are a few common aspects that apply to all layers are Logging, Security, validation, caching, etc. These common aspects are called cross-cutting concerns.

What are cross-cutting concerns?

- In any application, there is some generic functionality that is needed in many places. But this functionality is not related to the application's business logic. Suppose you perform a role-based security check before every business method in your application. Here security is a cross-cutting concern. It is required for any application but it is not necessary from the business point of view, it is a simple generic functionality we have to implement in many places in the application. The following are examples of the cross-cutting concerns for the enterprise application.

- Logging and tracing
- Transaction management
- Security
- Caching
- Error handling
- Performance monitoring
- Custom business rules

Advantages of Spring AOP

- AOP is non-invasive:
- Service or Domain classes get advice by the aspects (cross-cutting concerns) without adding Spring AOP related classes or interfaces into the service or domain classes.
- Allows the developers to concentrate on the business logic, instead of the cross-cutting concerns.
- AOP is implemented in pure Java: There is no need for a special compilation unit or special class loader
- It uses Spring's IOC container for dependency injection:
- Aspects can be configured as normal spring beans.
- Like any other AOP framework, it weaves cross-cutting concerns into the classes, without making a call to the cross-cutting concerns from those classes.
- Centralizes or modularizes the cross-cutting concerns:
- Easy to maintain and make changes to the aspects.
- Changes only need to be made in one place.
- Provision to create aspects using schema-based (XML configuration) or @AspectJ annotation based style.
- Easy to configure.

Disadvantages of Spring AOP

1. A small difficulty is debugging the AOP framework-based application code.

Since the business classes are advised after the scene with aspects.

2. Since it uses proxy-based AOP, only method-level advising is supported; it does not support field-level interception

So join-points can be at method level not at field level in a class.

3. Only methods with public visibility will be advised:

Methods with private, protected, or default visibility will not be advised.

4. There's small runtime overhead, but its negotiable:

The overhead is in nano-seconds.

5. Aspects cannot advise other Aspects - it's not possible to have aspects as targets of advice from other aspects.

Because once you mark one class as an aspect (either use XML or annotation), Spring excludes it from being auto-proxied.

6. Local or internal method calls within an advised class don’t get intercepted by proxy, so the advice method of the aspect does not get fired or invoked.

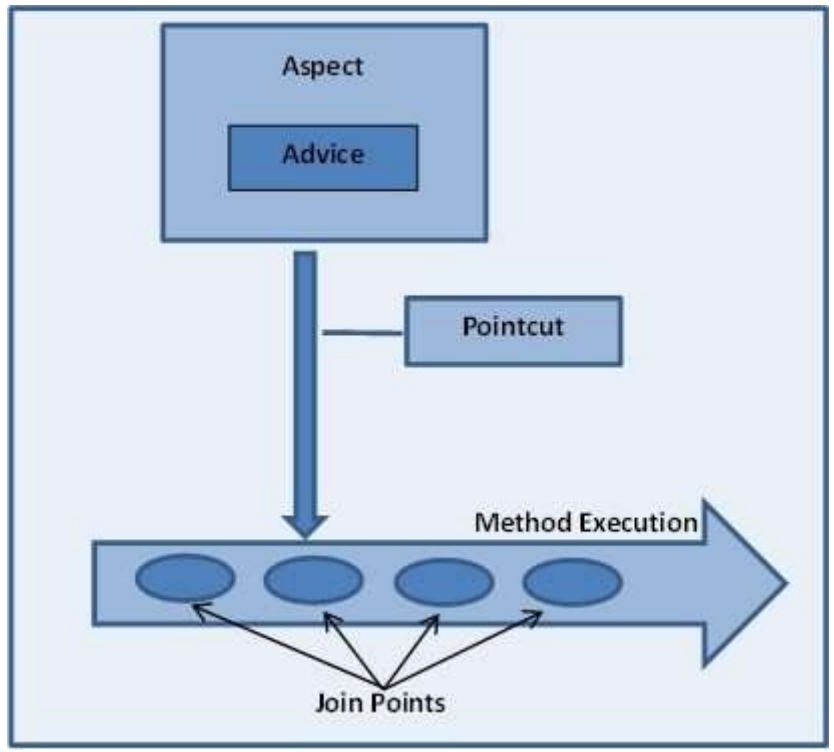
7. It is not for advising fine-grained objects (or domain objects)—it is best suitable for coarse-grained objects due to performance.

Uses of Spring AOP

- AOP is commonly used as an implementation of "cross-cutting concerns," which means it defines, in one place, functionality that is needed in multiple places throughout a code.
- A cross-cutting concern that can affect the whole application should be centralized in one place in code as much as possible, such as authentication, logging, transaction management, security, etc.

What is advice, joinpoint or pointcut?

- An important term in AOP is **advice**. It is the action taken by an **aspect** at a particular join-point.
- **Joinpoint** is a point of execution of the program, such as the execution of a method or the handling of an exception. In Spring AOP, a joinpoint always represents a method execution.
- **Pointcut** is a predicate or expression that matches join points.
- **Advice** is associated with a pointcut expression and runs at any join point matched by the pointcut.
- Spring uses the AspectJ pointcut expression language by default.


Aspect
Advice
Pointcut
Method Execution
Join Points

Types of AOP Advices

- There are five types of advice in spring AOP.
- **Before advice**: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
- **After returning advice**: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
- **After throwing advice**: Advice to be executed if a method exits by throwing an exception.
- **After advice**: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
- **Around advice:** Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.