Proyecto TIC: Desarrollo e implantación

Enterprise Applications
Jakarta EE

Xavi Piñol

xavi.pinyol@gmail.com

Index

Index

Jakarta EE. Introduction.

Java Platform, Enterprise Edition or Java EE is Oracle's enterprise Java computing platform. 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.

Jakarta EE. Introduction.

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.

Index

Jakarta EE. Version History.

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. In 2019 the name changed to Jakarta EE. The current version is called Javakarta EE 10.

- J2EE 1.2 (December 12, 1999)
- J2EE 1.3 (September 24, 2001)

Jakarta EE. Version History.

J2EE 1.4 (November 11, 2003)

Java EE 5 (May 11, 2006)

Java EE 6 (December 10, 2009)

Java EE 7 (June, 2013,)

Java EE 8 (August, 2017)

Jakarta EE 8 (Setember, 2019)

Jakarta EE. Version History.

Jakarta EE 9 (December, 2019)

Jakarta EE 9.1 (May , 2021)

Jakarta EE 10 (August , 2022)

Index

Jakarta EE. Standards and especifications.

Jakarta EE includes several API specifications, such as RMI, e-mail, JMS, web services, XML, etc., and defines how to coordinate them. Java EE also features some specifications unique to Jakarta EE for components. These include Enterprise JavaBeans, connectors, servlets, JavaServer Pages and several web service technologies. This allows developers to create enterprise applications that are portable and scalable, and that integrate with legacy technologies.

Index

Jakarta EE. Application servers

Application servers are system software upon which web applications or desktop applications run. Application Servers consist of web server connectors, computer programming languages, runtime libraries, database connectors, and the administration code needed to deploy, configure, manage, and connect these components on a web host. An application server runs behind a web Server (e.g. Apache or Microsoft IIS) and (almost always) in front of an SQL database (e.g. PostgreSQL, MySQL or Oracle).

Jakarta EE. Application Servers

Web applications are computer code which run on top of application servers and are written in the language(s) the application server supports and call the runtime libraries and components the application server offers.

There are many application servers and the choice impacts the cost, performance, reliability, scalability, and maintainability of a web application.

Jakarta EE. Application Servers.

Proprietary application servers provide system services in a well-defined but proprietary manner. The application developers develop programs according to the specification of the application server. Dependence on a particular vendor is the drawback of this approach.

An opposite but analogous case is the Java EE platform. Java EE application servers provide system services in a well-defined, open, industry standard.

Jakarta EE. Application Servers.

The application developers develop programs according to the Java EE specification and not according to the application server. A Java EE application developed according to Java EE standard can be deployed in any Java EE application server making it vendor independent.

Java EE is defined by its specification. Providers (applications servers) must meet certain conformance requirements in order to declare their products as Java EE compliant.

Index

Enterprise Applications. Introduction.

Enterprise applications is purpose-designed computer software used to satisfy the needs of an organization rather than individual users. Enterprise applications is an integral part of a (computer based) Information System, and as such includes web site software production.

Enterprise Applications. Introduction.

Enterprise applications are complex, scalable, distributed, component-based, and mission-critical. They may be deployed on a variety of platforms across corporate networks, intranets, or the Internet. They are data-centric, user-friendly, and must meet stringent requirements for security, administration, and maintenance. In short, they are highly complex systems.

Index

Enterprise Applications. Features.

Database access

database transactions: ACID (Atomicity-Consistency-Isolation-Durability)

scalability: A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a scalable system.

Availability.

Secure: not all users can access the same functionality

Enterprise Applications. Features.

Integrity: is needed to integrate different applications build in different technologies.

Interfaces types:

windows: standalone applications. Intranets.

Web: Internet and Intranets.

Enterprise Applications. Features.

User interface and model: they must be clearly separated.

Model: business logic.

Model: ready to be used by different user interfaces such a web interface or a standalone interface.

Multi-tier architectures

Index

Enterprise Applications. Architecture types.

Two-tier architecture. Application with standalone clients.

Tier 1

Graphical user interface + M...

Graphical user interface + Model

Intranet

Tier 2

Data Base

Enterprise Applications. Architecture types.

Problem: a change in the model layer.

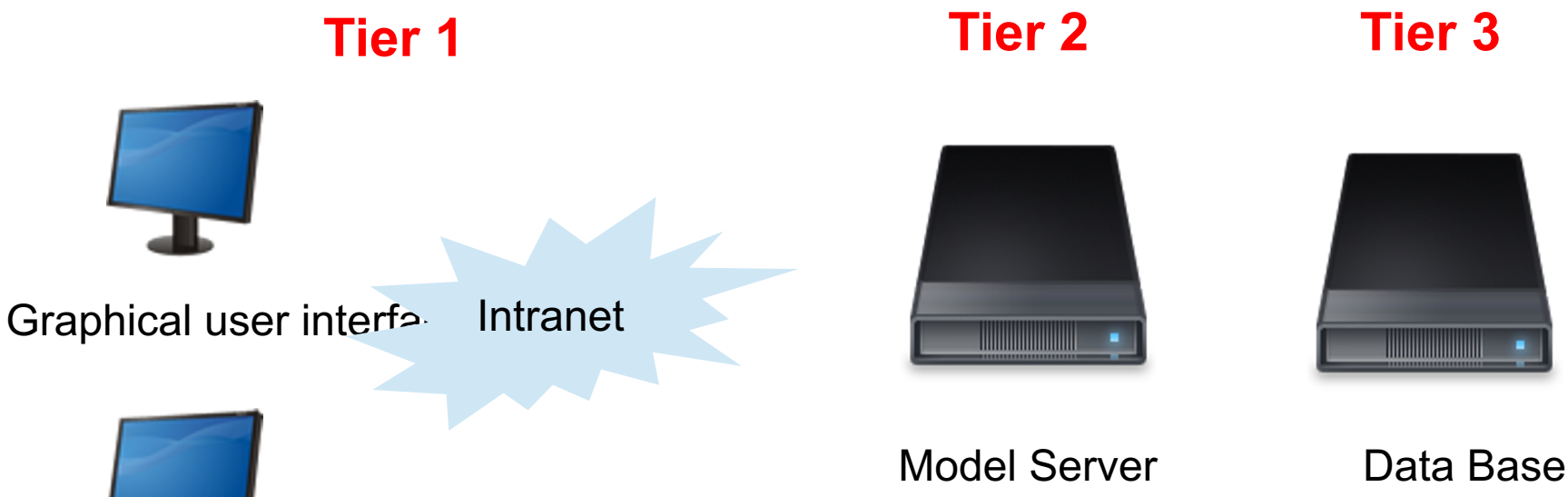
Re-build all application and re-install all clients.

Solution: middle tier for the model.

The standalone clients only have user interface.

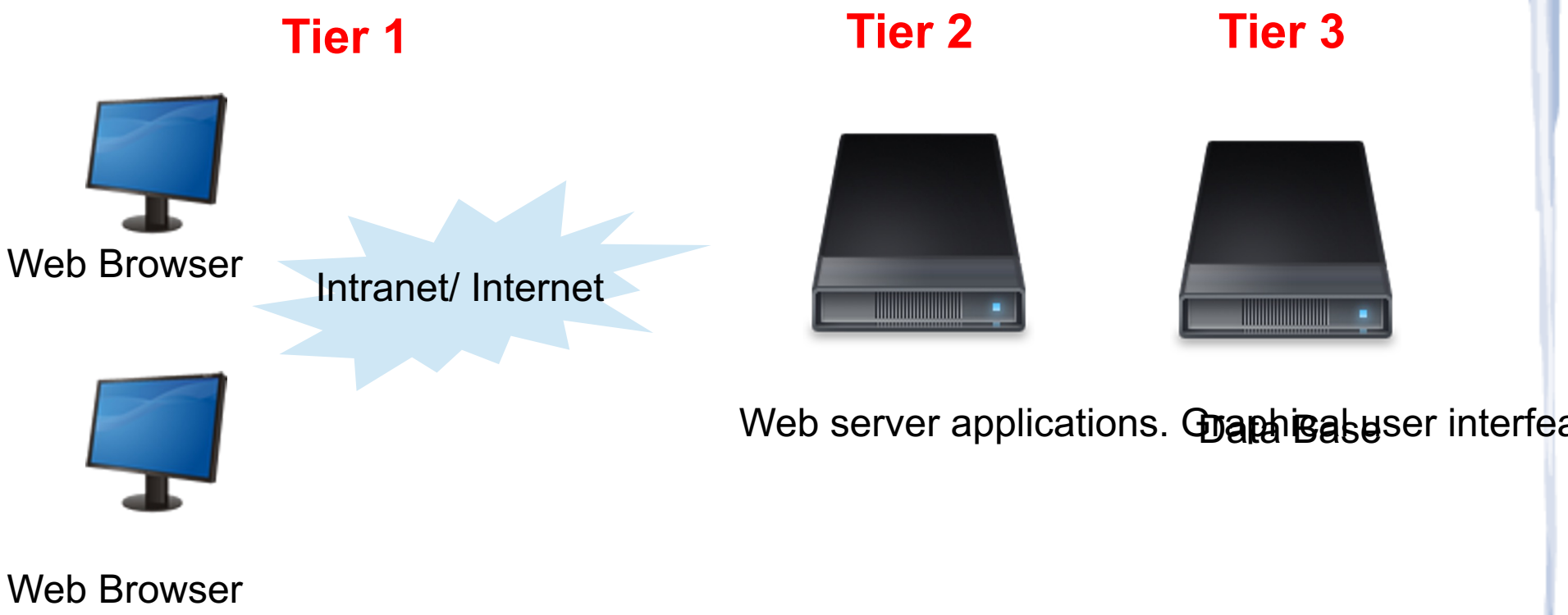
Enterprise Applications. Architecture types.

Three-tier architecture. Application with standalone clients.



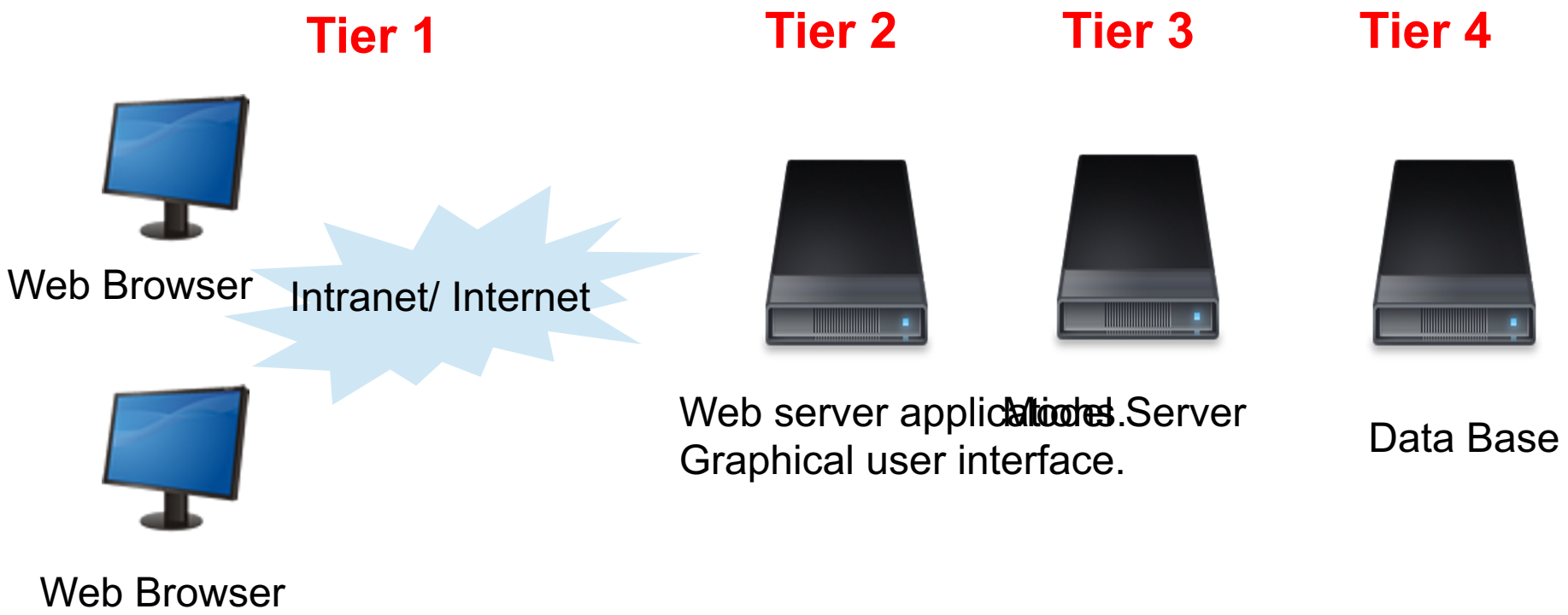
Enterprise Applications. Architecture types.

Three-tier architecture. Web application.



Enterprise Applications. Architecture types.

Four-tier architecture. Web application



Enterprise Applications. Architecture types.

Three-tier architecture is the most efficient for a web application.

Communication between model and view is local.

How can we get scalability and availability ?

Use replicated servers

Index

Enterprise Applications. Architectural patterns.

An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design patterns but have a broader scope. The architectural patterns address various issues in software engineering, such as computer hardware performance limitations, high availability and minimization of a business risk. Some architectural patterns have been implemented within software frameworks.

Enterprise Applications. Architectural patterns.

Model-view-controller (MVC) and Layer architectural patterns will help to build maintainable and reusable enterprise applications.

Enterprise Applications. Architectural patterns.

Model-view-controller (MVC)

A **controller** can send commands to the model to update the model's state. It can also send commands to its associated view to change the view's presentation of the model.

A **model** stores data that is retrieved to the controller and displayed in the view. Whenever there is a change to the data it is updated by the controller.

Enterprise Applications. Architectural patterns.

A **view** requests information from the controller. The controller fetches it from the model and passes it to the view that the view uses to generate an output representation to the user.

Advantages:

The model is reusable in differents views such as web view or windows interface.

Clear work division between the members of the work group.

Enterprise Applications. Architectural patterns.

Layer

Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. Communication between layers is explicit and loosely coupled. Layering your application appropriately helps to support a strong separation of concerns that, in turn, supports flexibility and maintainability.

Enterprise Applications. Architectural patterns.

Advantages

If there is change in one layer, e.g. new version, the superior layers keep unaffected.

Clear work division between the members of the work group.

Will help MVC: the view and the controller will never know the technologies which the model has been implemented.

Index

Enterprise Applications. EAR.

Enterprise Archive (EAR)

Is a file format used by Java EE for packaging one or more modules into a single archive so that the deployment of the various modules onto an application server happens simultaneously and coherently. It also contains XML files called deployment descriptors which describe how to deploy the modules.

Enterprise Applications. EAR.

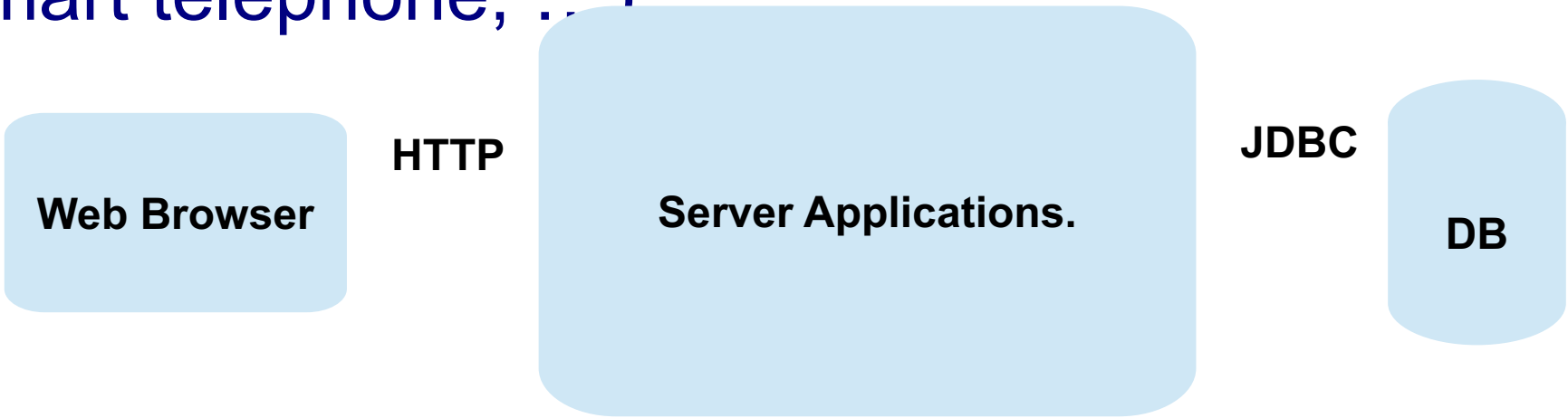
An EAR file is a standard JAR file (and therefore a Zip file) with a .ear extension, with one or more entries representing the modules of the application, and a metadata directory called META-INF which contains one or more deployment descriptors.

Index

Servlets. Introduction.

Developing an Enterprise Application.

An Enterprise Application runs, at least, in one server application and the final user just needs a general purpose client to access it (web browser, smart telephone, …)



Index

Servlets. HTTP.

The Hypertext Transfer Protocol (HTTP)

- The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems.

- Works over Transmission Control Protocol (TCP) . Port 80.

- Stateless

Servlets. HTTP.

- Defines methods (sometimes referred to as *verbs*) to indicate the desired action to be performed on the identified resource: get, put, post, delete, head, …

For web applications get and post are the most used.

Servlets. HTTP.

Get method

When an url address is introduced to a web browser: the web browser creates a TCP connection using the port 80.

GET will append all of the data to the URL and it will show up in the URL bar of your browser. The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.

Servlets. HTTP.

Parameters: URL?par1=val1&... &parn=valn

Parameters codification:

blank space: +

characters, '.' , '-' , '*', '_' : stay the same.

Rest of characters: %xy (hexadecimal codification).

Servlets. HTTP.

Post method

POST will send the information through a socket back to the web server and it won't show up in the URL bar. More information can be send to the server this way. It is possible to send files and even binary data.

The Servlet spec allows to implement separate Java methods implementing each HTTP method in your subclass of HttpServlet. Override the doGet() and/or doPost() method to provide normal servlet functionality.

Servlets. HTTP.

How to use get or post method from an html page:

<form method="**post**" action="sExample">

Enter code: <input type="text " name="code">

Name: <input type="text" name="userName">

<input type="Submit" value="Execute">

</form>

Servlets. HTTP.

Absolute and Relative paths

A URL specifies the location of a target stored on a local or networked computer. The target can be a file, directory, HTML page, image, program, and so on.

An absolute URL contains all the information necessary to locate a resource. Useful for web browsers.

An absolute or **full** path points to the same location in a file system. Useful for resources that are in the same server.

Servlets. HTTP.

A relative path starts from some given working directory, avoiding the need to provide the full absolute path. A filename can be considered as a relative path based at the current working directory. If the working directory is not the file's parent directory, a file not found error will result if the file is addressed by its name.

Useful to maintain when the web site structure changes.

Index

Servlets. Servlets.

A Java servlet is a Java programming language program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers. A servlet receives a request and generates a response based on that request.

Servlets communicate over any client–server protocol, but they are most often used with the HTTP protocol.

Servlets. Servlets.

A software developer may use a servlet to add dynamic content to a web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets can maintain state in session variables across many server transactions.

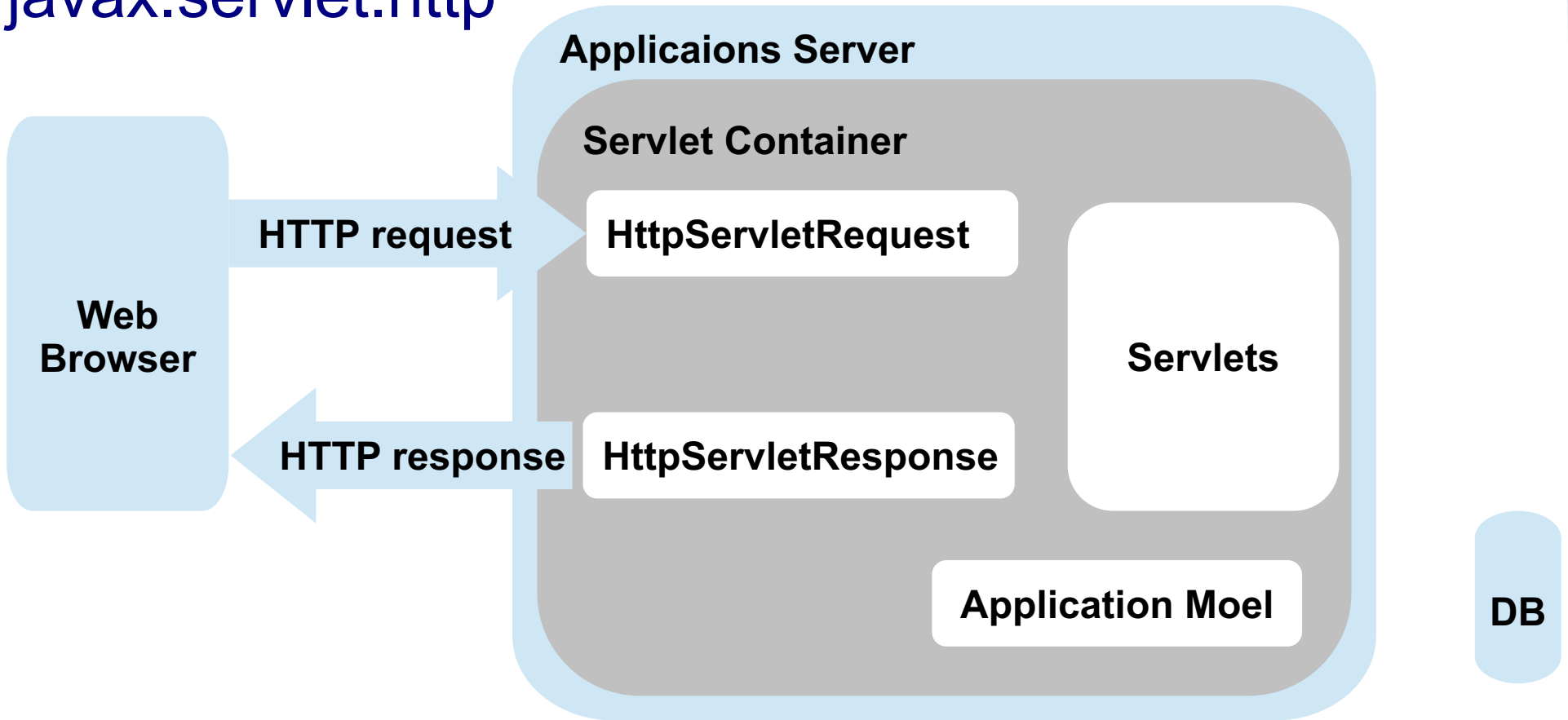
Servlets. Servlets.

To deploy and run a servlet, a web container must be used. A web container (also known as a servlet container) is essentially the component of a web server that interacts with the servlets. The web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

One servlet can have different URL's mapped.

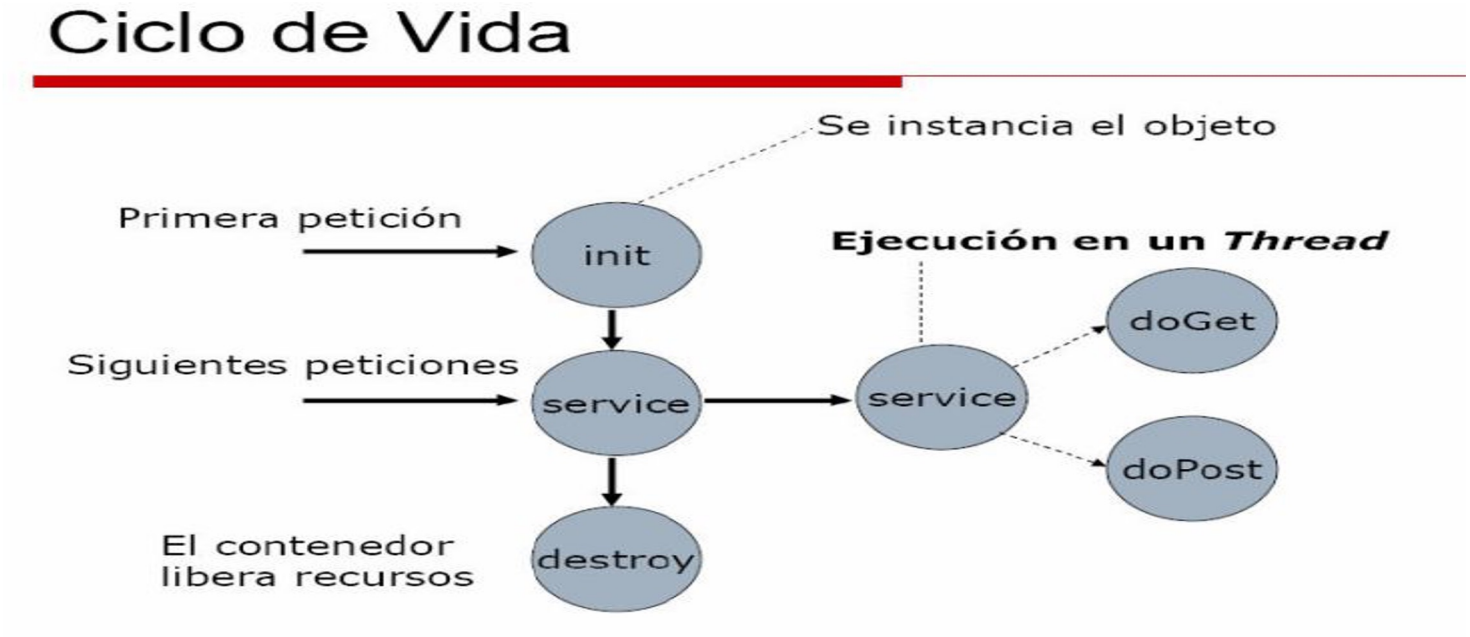
Servlets. Servlets.

Servlet use the packages: javax.servlet and javax.servlet.http



Servlets. Servlets.

Servlet lifecycle:



Servlets. Servlets.

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

The servlet is initialized by calling the init () method.

The servlet calls service() method to process a client's request.

The servlet is terminated by calling the destroy() method.

Finally, servlet is garbage collected by the garbage collector of the JVM.

Servlets. Servlets

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

Servlets. Servlets.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

Servlets. Servlets

The service() method:

The service method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Servlets. Servlets.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client. The doGet() and doPost() are most frequently used methods with in each service request.

Servlets. Servlets.

The doGet() Method and the doPost() Method

A GET or POST request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method or doPost() method.

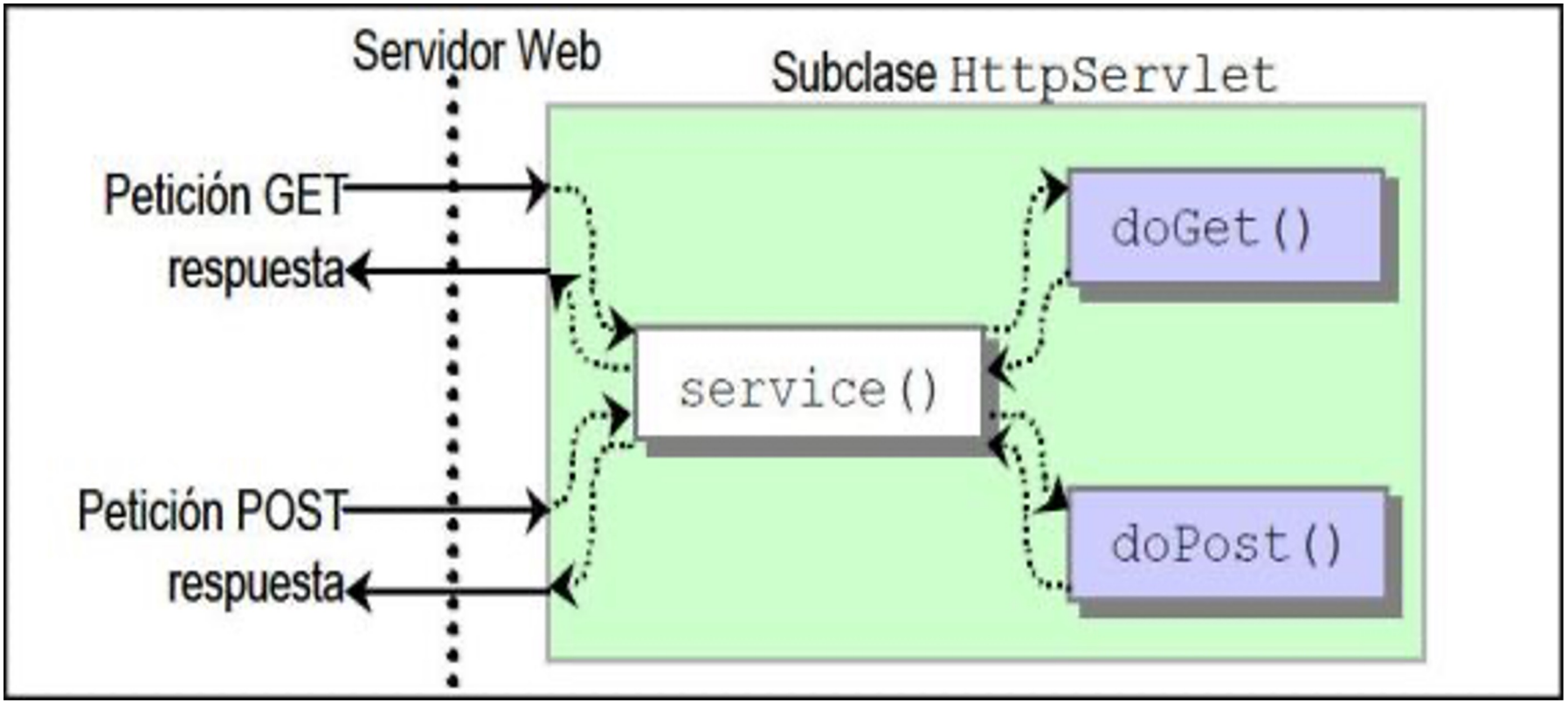
Servlets. Servlets.

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads and perform other cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection.

Servlets. Servlets.



Servlets. Servlets.

The servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.

Developer has to override either doGet() or doPost() methods. Normally developers will only use local variables and static global variables (read variables).

Servlets. Servlets.

Inside doGet or doPost methods:

public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException

public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException

Servlets. Servlets.

javax.servlet.http.HttpServletRequest request : passes an argument to the servlet's service methods.

getParameter : Returns the value of a request parameter as a String, or null if the parameter does not exist.

getParameterValues : Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist. If the parameter has a single value, the array has a length of 1.

Servlets. Servlets.

javax.servlet.http.HttpServletResponse response

```
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<html>");
pw.println("<head><title>Hello World</title></title>");
pw.println("<body>");
pw.println("<h1>Hello World</h1>");
pw.println("</body></html>");
```

Index

Servlets. JSP.

JSP may be viewed as a high-level abstraction of Java servlets. JSPs are translated into servlets at runtime; each JSP servlet is cached and re-used until the original JSP is modified.

JSP allows Java code and certain pre-defined actions to be interleaved with static web markup content, such as HTML, with the resulting page being compiled and executed on the server to deliver a document.

Servlets. JSP.

Implicit variables	Class
pageContext	javax.servlet.jsp.PageContext
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
config	javax.servlet.ServletConfig
application	javax.servlet.ServletContext
out	javax.servlet.jsp.JspWriter
page	java.lang.Object
exception	java.lang.Exception

Servlets. JSP.

JSP Directives

JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing.

<%@ directive attribute="value" %>

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

Servles. JSP.

There are three types of **directive** tag:

<%@ page ... %> Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.

<%@ include ... %> Includes a file during the translation phase.

<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page

Servlets. JSP.

Following is the list of **attributes** associated with page directive:

Buffer: Specifies a buffering model for the output stream.

AutoFlush: Controls the behavior of the servlet output buffer.

ContentType: Defines the character encoding scheme.

Servlets. JSP.

ErrorPage: Defines the URL of another JSP that reports on Java unchecked runtime exceptions.

IsErrorPage: Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.

Extends: Specifies a superclass that the generated servlet must extend

Import: Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.

Servlets. JSP.

Info: Defines a string that can be accessed with the servlet's getServletInfo() method.

IsThreadSafe: Defines the threading model for the generated servlet.

Language: Defines the programming language used in the JSP page.

Session: Specifies whether or not the JSP page participates in HTTP sessions

Servlets. JSP.

IsELIgnored: Specifies whether or not EL expression within the JSP page will be ignored.

IsScriptingEnabled: Determines if scripting elements are allowed for use.

<%@ include file="header.html" %>

<%@ page import="class; class" %>

<%@ page errorPage="/pathToErrorPage" %>

<%@ page isErrorPage="true" %>

Servlets. JSP.

JSP Declarations:

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

<%! declaration; [declaration;]+ ... %>

<%! int nMax = 10; %>

Servlets. JSP.

The Scriptlet:

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

<% code fragment %>

Servlets. JSP.

JSP Expression:

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

Servlets. JSP.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

<%=nMax+2%>

Servlets. JSP.

My first JSP

```
<%@ page  language="java" import="java.util.*"
errorPage="" %>

<html>

<body>

  date and time: <%=new java.util.Date()%>

</body>

</html>
```

Servlets. JSP.

How to call a JSP from a servlet:

```
ServletContext context= getServletContext();
RequestDispatcher rd=
context.getRequestDispatcher("/jFinal");
rd.forward(request, response);
```

Index

Servlets. Session.

public interface HttpSession

Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

Servlets. Session.

Session information is scoped only to the current web application (ServletContext), so information stored in one context will not be directly visible in another.

Servlets. Session.

Inside our servlet:

request.getSession() : Returns the current session (public interface HttpSession) associated with this request, or if the request does not have a session, creates one.

Servlets. Session.

public interface HttpSession

setAttribute(java.lang.String name, java.lang.Object value) : Binds an object to this session, using the name specified.

getAttribute(java.lang.String name) : Returns the object bound with the specified name in this session, or null if no object is bound under the name.

Servlets. Session.

How to add an object to the session from a servlet:

```
session = request.getSession(true);
session.setAttribute("ServletWeb.name",
strName);
```

Servlets. Session.

How to get an object from the session from a JSP:

```
<% String strName=(String)
session.getAttribute("ServletWeb.name"); %>
```

Servlets. JSP.

[Remember]

How to call a JSP from a servlet:

```
ServletContext context= getServletContext();
RequestDispatcher rd=
context.getRequestDispatcher("/jFinal");
rd.forward(request, response);
```