Enterprise Apps and JEE

Alessio Bechini Dept. of Information Engineering, Univ. of Pisa

a.bechini@ing.unipi.it

Outline

Web Applications, Enterprise Applications, and JEE

- General Ideas
- Web Applications
- Servlets and More
- Enterprise Applications
- JNDI
- EJBs
- JMS

General Ideas

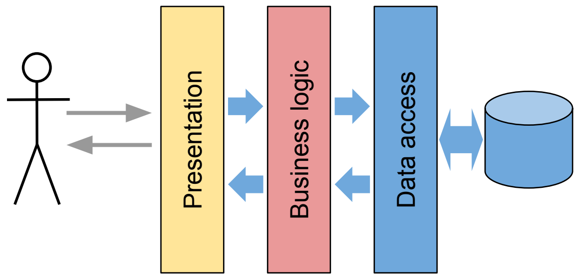
3-Tier Architecture

Typical organization of a (web) application: **3-tier architecture**

➜ a client-server architecture where:

1. user interface,
2. functional process logic,
3. data access/data storage

are developed and maintained as *independent modules*, possibly on separate platforms.

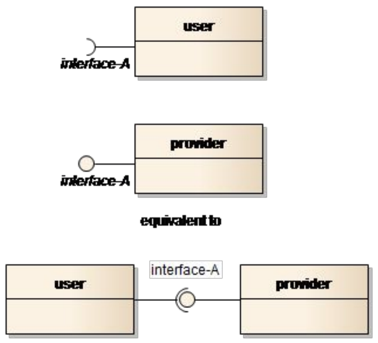


Use of Containers in Middleware

At the middleware level, components can be organized in *containers*, and components managed within containers.

Containers take care of supporting their **managed components**, providing them with the required functionalities.

A container, as well as any component, has both *provided* and *requested interfaces*.



From Now on…

Java EE™

Web Applications

Static vs Dynamic Content

HTTP deals with requesting a resource, and getting it.

It's up to the server to provide the resource,
either retrieving it as a file (static content),
or generating it on the fly by means of a program (dynamic content).

The server has to tell apart the type of resource just from its URL.

The possible generation of content must be guided by the server.

Common Gateway Interface

A standard protocol for web servers to execute shell programs that dynamically generate web pages.
CGI scripts are *usually* placed in the special directory `cgi-bin`.
Parameters are passed via environment vars and by std input.

Example of URL:
http://xyz.com/**cgi-bin/myscript.pl**/my/pathinfo?a=1&b=2

To solve process spawning overhead ➜ **FastCGI** approaches

Common Gateway Interface

A standard protocol for web servers to execute shell programs that dynamically generate web pages.
C *lly* plac al dir
Parameters ar ssed via environme vars ar

Example of URL:
http://xyz.com/**cgi-bin/myscript.pl**/my/pathinfo?a=1&b=2

To solve process spawning overhead ➜ **FastCGI** approaches

A Perl script

`PATH_INFO`

`QUERY_STRING`

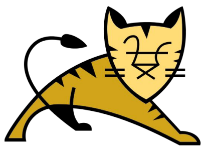
Beyond CGI-Scripts

Idea: to improve performance, the code for content generation could be executed **internally** to the web server, i.e. within the same process, possibly using multithreading.

In Microsoft environments: classic **ASP**

In JavaEE: **Servlets**, and related technologies

The Tomcat Web Server



Basic internal components:

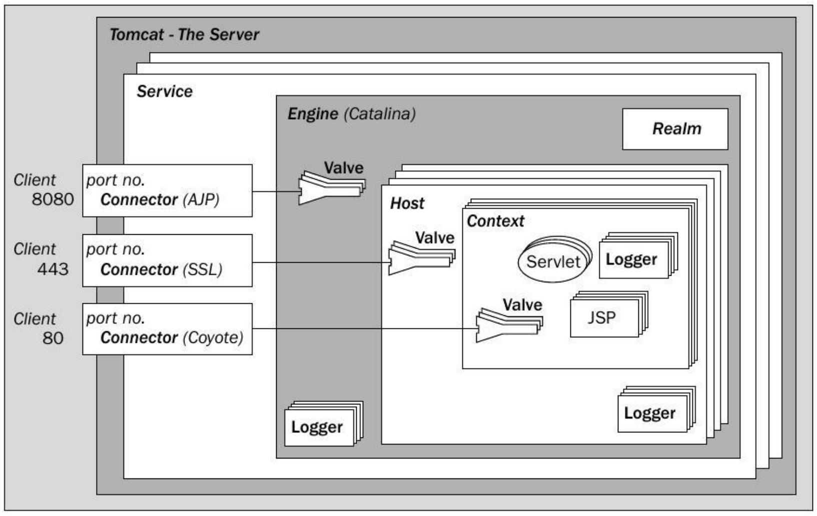
- **Catalina:** “Engine,” Servlet container; it refers also to a *Realm* (DB of usernames, passwords, and relative roles).
- **Coyote:** HTTP “Connector,” listens on ports and forward requests to the engine
- **Jasper:** JSP container

“Service” element: combination of 1+ Connectors that share a single Engine component for processing incoming requests.

Tomcat Architecture

Valve:
element to be inserted in the REQ processing pipeline

Other nested elems:
(Session) Manager,
(W.apps class) Loader,
Listener(s), etc.



Hosts and Contexts

A “Host” component represents a *virtual host*, i.e. an association of a network name for a server (e.g. "www.mycompany.com") with the particular server Catalina is running on.

A “Context” element represents a web application, which is run within a particular virtual host.

The web app used to process each HTTP request is selected by Catalina based on matching the longest possible prefix of the Request URI against the **context path** of each defined Context.

Context Path and Base File Name

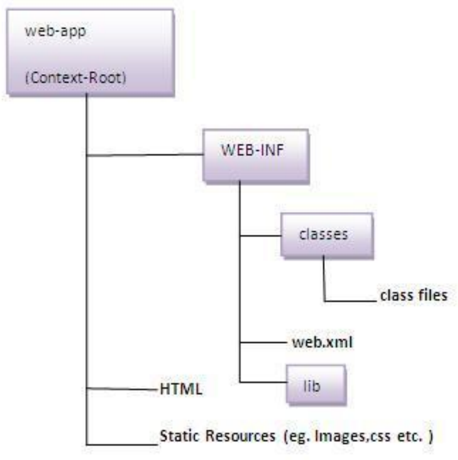
In the filesystem, one directory (*appBase*, default “webapps”) is used to keep the material for web applications.

Each web app corresponds to a base file name.

Rules to obtain the base file name, given the context path:

- If the context path is “” ➜ “ROOT”
- If the context path is not “” ➜ base name = context path with the leading '/' removed, and any remaining '/' replaced with '#'.

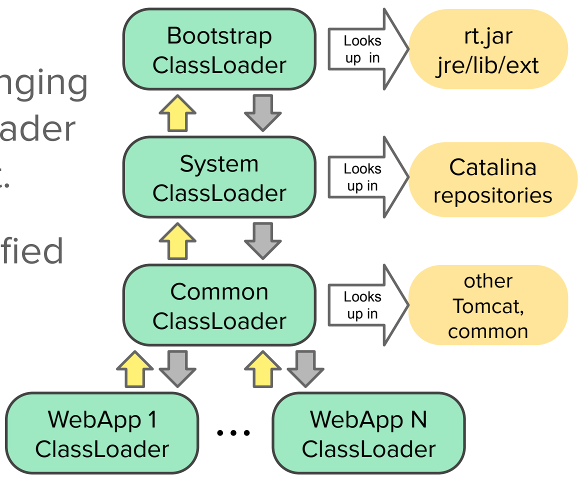
Structure of a WebApp Directory



Not-interfering Contexts

To avoid interference of classes belonging to different contexts, different classloader hierarchies are used for each context.

Remember: any loaded class is identified by its name AND the classloader that has loaded it, so possibly the same class used in two web apps is loaded twice.



Servlets and Their Container

Servlets: Java classes/objects that respond to a request, aimed at producing the content for the response.
*The thread-per-request model is generally applied ➔ **synch issues!***

Servlets are kept in a container.
Their methods are meant to be invoked by the container, also as the main step of a *request processing pipeline*.

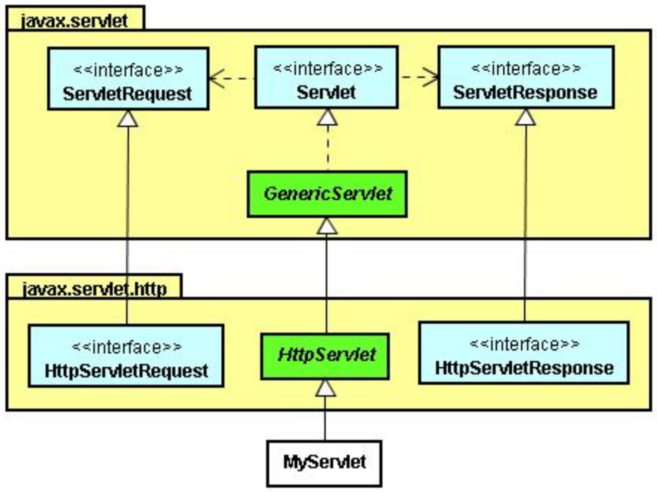
The container is responsible for managing the lifecycle of servlets, and mapping a URL to a particular servlet.

Servlets and More

What's a Servlet?

A class that implements the `javax.servlet.Servlet` Interface…

In practice: our custom servlets have to extend either `GenericServlet` or, usually, `HttpServlet`

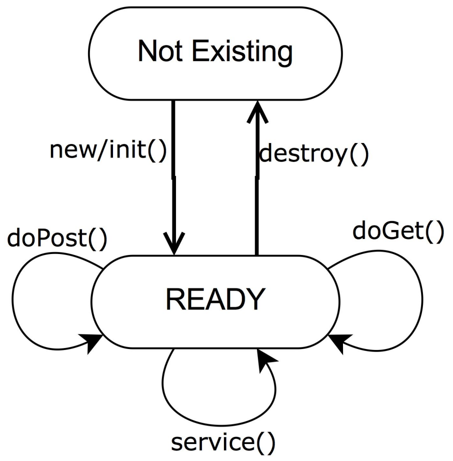

javax.servlet
<<interface>> ServletRequest
<<interface>> Servlet
<<interface>> ServletResponse
GenericServlet
javax.servlet.http
<<interface>> HttpServletRequest
HttpServlet
<<interface>> HttpServletResponse
MyServlet

Servlet Life-cycle

The entire lifecycle of a servlet
is managed by the container.

Upon a REQ, `service()` is invoked;
possibly, de-multiplexing to
`doPost()`, `doGet()`, etc.

BTW: How many instances
of a Servlet class (in a single webapp)?



Handling of REQ/RESPONSE

For a simple handling of HTTP requests/responses,
they are represented by corresponding objects, so that containers
can easily take care of them throughout their processing pipeline.

According to Servlets specification, such objects must implement:

1. `javax.servlet.http.`**`HttpServletRequest`**
2. `javax.servlet.http.`**`HttpServletResponse`**

This approach is much more convenient that CGI!

Parameters in REQ

Regardless of GET/POST mode, parameters can be accessed within the request object using the methods (`ServletRequest` interface):

- `getParameterNames()` provides the names of the parameters
- `getParameter(name)` returns the value of a named parameter
- `getParameterValues()` returns an array of all values of a parameter if it has more than one values.

Information in HTTP headers is retrieved in the same way.

Acting on Response

Two ways of inserting data in the body:

- **`getWriter()`** returns a `PrintWriter` for sending text data
- **`getOutputStream()`** returns a `ServletOutputStream` to send binary data

Both need to be closed after use!

Setting headers' content: specific methods, e.g.
`setContentType(<MIME_type>)`

Mapping URLs onto Servlets

Specified in the webapp *deployment descriptor* **`web.xml`** in two steps:

servlet-name ⟶ servlet-class + servlet-name ⟶ url-pattern

Alternative way: directly in the code, using *annotations*:

```
@WebServlet(
    name = "MyAnnotatedServlet",
    urlPatterns = {"/foo", "/bar", "/pippo*"}
)
public class MyServlet extends HttpServlet {
    // servlet code
}
```

Session Tracking

A mechanism to maintain state information *for a series of requests*,

- along a period of time,
- from the same client.

Session are represented through specific objects (interf. `HttpSession`) shared across all the servlets accessed by the same client.

Programmatically, session objects can be obtained from requests:

```
HttpSession mySess = req.getSession(boolean create);
```

Session Tracking Techniques

- Using session cookies
- Hidden fields
- URL rewriting

Standard technique

Sessions: Keeping State Information

Session objects are ordinarily used to *keep state information* throughout the session.

- `public void setAttribute(String name, Object obj)`
- `public Object getAttribute(String name)`

Optionally, sessions can be invalidated:

- `mySess.invalidate()` i.e., immediately
- `mySess.setMaxInactiveInterval(int interval)` i.e., max interval between successive requests (default value defined in web.xml)

Servlet Example

```
import ...

public class HelloWorld extends HttpServlet {
   private String msg;

   public void init() throws ServletException {
      msg = "Hello World";
   }

   public void doGet(HttpServletRequest req,HttpServletResponse res)
                         throws ServletException, IOException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<h1>" + message + "</h1>");
   }
}
```

REQ Processing: Servlet Filters

Upon receiving a REQ, often some typical actions have to be undertaken.

E.g. recording, IP logging, input validation, authentication check, encryption/decryption, etc.

Each action can be carried out by a *pluggable* server-managed component named *servlet filter*; its application depends on the relative *url pattern*.

Adv: separation of concerns (different pieces of SW/different tasks), easy maintenance

APIs: in javax.servlet, interfaces **`Filter`**, **`FilterChain`**, **`FilterConfig`**

Implementation of a Servlet Filter

A custom servlet filter must extend **`Filter`** ➔ implement

- init(), destroy()
- doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)

The body of doFilter() implements the action to be undertaken.

To pass the REQ to the next component ➔ chain.doFilter(req, resp);

Mapping of filters: as for servlets, in web.xml or using annotations, e.g.

```
@WebFilter(filterName = "TimeOfDayFilter", urlPatterns = {"/*"},
       initParams = {@WebInitParam(name = "mood", value = "awake")})
public class TimeOfDayFilter implements Filter {
    ...
```

Structuring Servlet Apps

Servlets are a basic technology that imposes no constraint
on how a whole application should be structured.

Risk: “Magic Servlet” antipattern, where issues about
different aspects of the application are dealt within the same method.

Solution: define specific roles, so that a specific task/concern
refers to a distinct software component.

Consequence: introduction of further levels of abstraction
for a well-structured development of web apps.

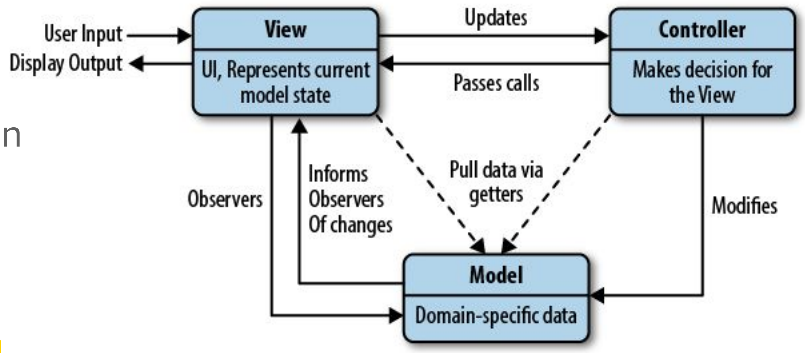
MVC - Model View Controller

Originally conceived for GUIs

Model: manages data

View: handles interaction with the user

Controller: coordinates interactions



Template Systems & Engines

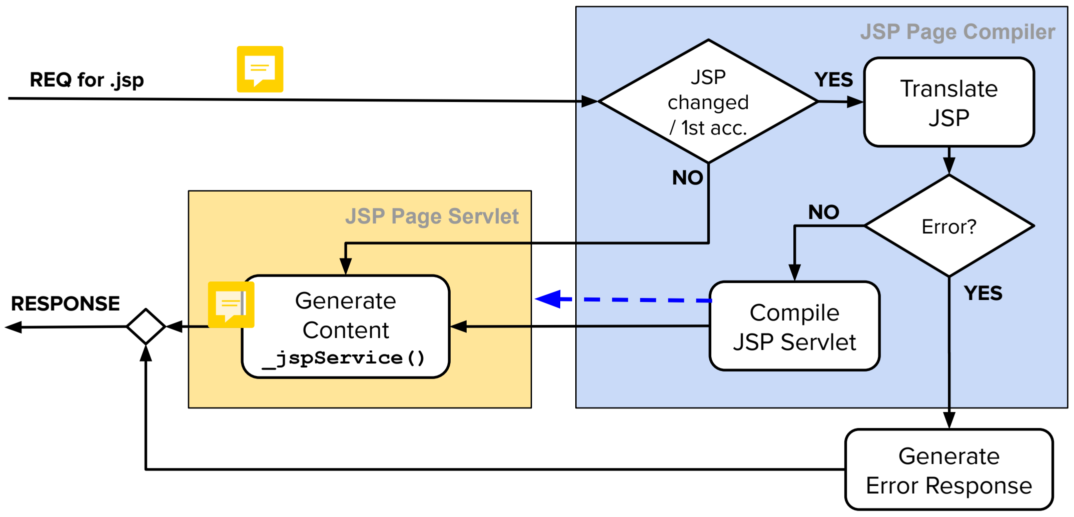
Servlets cannot be easily maintained because of the tight coupling of presentation (HTML) and business logic (in Java).

Proposed solution: adoption of a *server-side template system*, with templates in HTML, and logic embedded by using special tags.

Such templates have to be processed by a *template engine*, getting to the actual dynamic content.

Examples: ASP, **JavaServer Pages**, PHP, etc.

Java Server Pages - Processing



JSP - Basic Scripting

Template: an HTML document. Scripting can be added in several ways; the most direct one is through *scriptlets*, i.e. java code inside the delimiters **<% … %>**. Other possibility: expressions, **<%=** an_expression **%>** , etc.

The code within the scriptlet tags goes into the _jspService() method.

```
<p>Listing of the first natural numbers:</p>
<% for (int i=1; i<4; i++) { %>
    <p>This number is <%= i %>.</p>
<% } %>
<p>OK.</p>
```

JSP - Implicit Objects

Some objects are made available to the JSP by the environment:

- **request**, **response**
- **out** - PrintWriter obj to write in the response body
- **session** - to access the session obj
- **application** - to access ServletContext obj (info sharing across JSPs)
- **page** (a synonym for this)
- others...

Implicit objects can directly be used in the JSP scripts in developing business logic.

Better Structuring of JSPs

To apply the "separation of concerns principle", other mechanisms can be used inside JSPs:

- **Java Beans** - basic Java components (classes)
- **JSTL** - library of standard tags to support typical control flow
- **EL (Expression Language)** - makes it possible to easily access application data stored in JavaBeans

JSP and MVC

JSPs are typically used to support the “view” part of the MVC pattern.

The abstraction provided by JSP is sufficient
only for relatively small web applications.

As the complexity grows up, it can be controlled by making use
of more integrated approaches that make transparent
the underlying used technologies, like Servlets and JSPs.

The resulting systems are known as “web frameworks,”
and most of them are designed taking MVC as the reference pattern.

MVC Web Frameworks

Out of the most popular frameworks in the community of developers
of Java Web/Enterprise Applications, we can recall:

- **JavaServer Faces (JSF)**, part of JEE
- **Spring** , **Struts**

MVC frameworks are popular also with other languages, e.g.:

- **ASP.NET (MVC)** - successor of ASP, with C# and CLI languages
- **Django** - with Python, for complex web apps
- **Play** - with Scala (Akka)

Enterprise Applications

Problems with Plain Objects

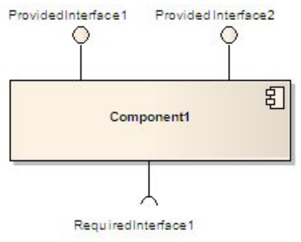
Use of plain objects in an enterprise app shows some problems:

- No deployment transparency
- Implicit dependencies (must be made explicit!)

Solution: from concept of object ➔ **component**

Provided/required interfaces:
contract between different components

Need to support the use of components!



Application Servers

Need to simplify the programming model:

the programmer must focus on business logic,
without spending time on distributed computing issues.

Architectural support to separation of concerns ➜
Container pattern to manage components

Middleware solution: **Application server**

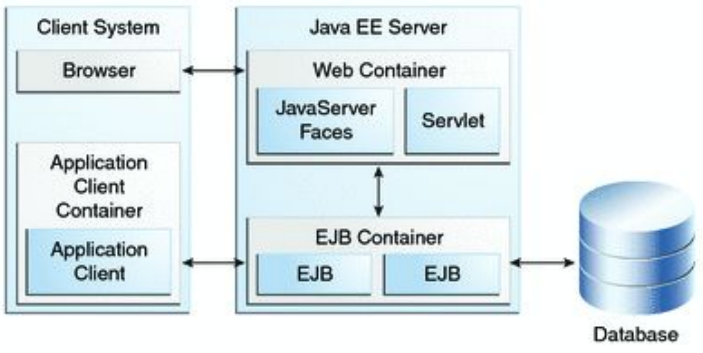
Addressed services: component lifecycle management, resource management, persistence, transactions, concurrency, security, etc.

Java Application Servers - Examples

- Glassfish (JEE reference implementation)

  ➜ payara
- JBoss EAP and subsequently WildFly
- IBM WebSphere - historical AS
- Oracle WebLogic - other historical AS

Java Application Servers - Overview



Enterprise Java Beans