

# Servlet e JSP

Danilo Croce

(parzialmente ispirato da Java Enterprise Edition di Gabriele Tomei)

Maggio 2019

# Java Servlet: Packages

- È definita all'interno del package standard `javax.servlet`
- Ogni Servlet deve implementare l'interfaccia `javax.servlet.Servlet`
  - specifica i metodi relativi al ciclo di vita
- Le Servlets che gestiscono protocolli di richiesta/risposta generici devono estendere `javax.servlet.GenericServlet`
- Le Servlet specifiche per la gestione del protocollo HTTP devono estendere `javax.servlet.http.HttpServlet`

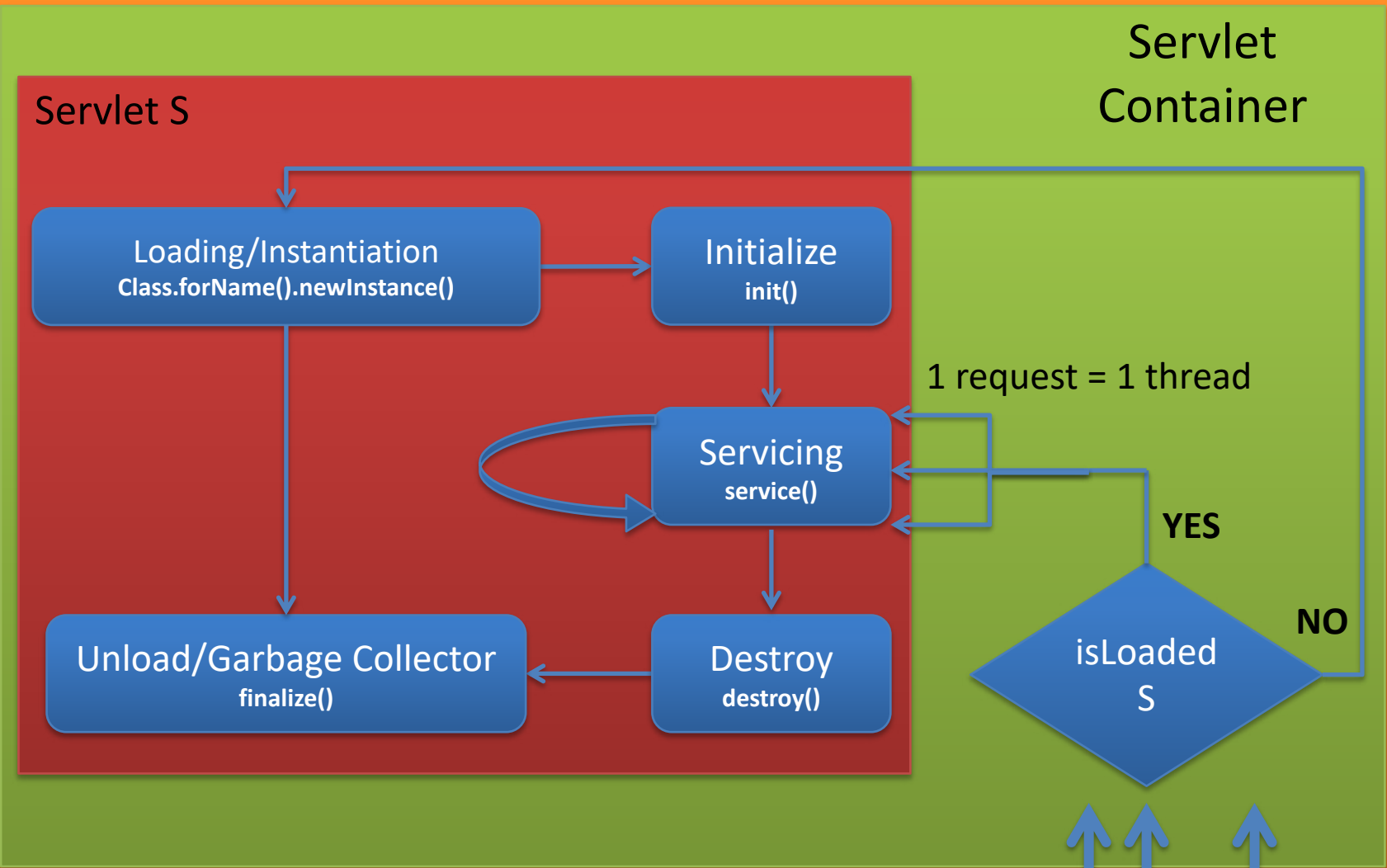
# Java Servlet: Esempio

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class HelloWorldServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("\t<head><title>Hello World</title></head>");
        out.println("\t<body>");
        out.println("\t\t<p>Hello " + req.getRemoteAddr() + "!</p>");
        out.println("\t\t<p>Server time is " + new Date() + "</p>");
        out.println("\t</body>");
        out.println("</html>");
        out.close();
    }
}
```

# Java Servlet: Ciclo di Vita

- A fronte di una richiesta da parte del client il container:
  1. Verifica che la Servlet sia già stata caricata
    - a. Se non lo è, provvede a caricare la classe corrispondente e ne genera un'istanza
    - b. Inizializza l'istanza appena creata invocando su di essa il metodo init
  2. Invoca il metodo service corrispondente all'istanza della Servlet passando come argomenti gli oggetti che rappresentano la richiesta e la risposta
- La rimozione della Servlet dal container si ottiene tramite una chiamata al metodo destroy



Incoming client requests  
for the Servlet S

# Java Servlet: Ciclo di Vita (2)

- I metodi init e destroy vengono chiamati solo una volta, rispettivamente alla creazione e rimozione della Servlet
- Il metodo service viene chiamato una volta per ciascuna richiesta (spesso in modo concorrente da più threads)
- Nel caso di HttpServlet al posto del metodo service (che pure è presente) vengono invocati metodi più specifici che corrispondono al protocollo HTTP:
  - HTTP GET → doGet
  - HTTP POST → doPost

# Java Servlet: Inizializzazione

- Per customizzare l'inizializzazione una Servlet può implementare o fare overriding del metodo `init`
- Il metodo `init` prende come argomento un'istanza di `javax.servlet.ServletConfig`
  - contiene i parametri di inizializzazione
  - utilizza il file descrittore in **WEB-INF/web.xml**
- Anche `GenericServlet` implementa l'interfaccia `ServletConfig`
  - metodo `init` con nessun argomento

# Java Servlet: Inizializzazione

```
public class HelloServlet extends HttpServlet {
    private String name = null;

    public void init(ServletConfig config) { //like a constructor
        super.init(config);
        this.name = config.getInitParameter("name");
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.print("<html><head><title>Hello</title></head>");
        out.print("<body><p>Hello ");
        if (this.name == null) {
            out.print(req.getRemoteAddr());
        } else {
            out.print(this.name);
        }
        out.println("!</p></body></html>");
        out.flush();
    }
}
```



# Richiesta/Risposta: service

- La gestione della richiesta/risposta è affidata al metodo `service` che ha i seguenti argomenti:
  - `ServletRequest`: richiesta del cliente (lettura)
  - `ServletResponse`: risposta al cliente (scrittura)
- Nel caso specifico di `HttpServlet` il metodo `service` è un “dispatcher” verso altri metodi specifici del protocollo HTTP (`doGet`, `doPost`, ...)
  - `HttpServletRequest`: HTTP request
  - `HttpServletResponse`: HTTP response

# Lettura:(Http)ServletRequest

- La richiesta del client consente di accedere alle seguenti informazioni:
  - Client: IP/hostname, porta
  - Browser: locale, headers, security
  - Request: headers, cookies, path, parameters, content-type, -length, -encoding, -body
  - User: authorization/authentication (role)
  - Session: attributi della sessione
  - Request shared storage: attributi della richiesta

# (Http)ServletRequest

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    String clientIp = req.getRemoteAddr();
    int clientPort = req.getRemotePort();
    Locale browserLocale = req.getLocale();
    boolean isBrowserSecure = req.isSecure();
    String userAgent = req.getHeader("User-Agent");
    Cookie[] clientCookies = req.getCookies();
    String requestUri = req.getRequestURI();
    StringBuffer requestUrl = req.getRequestURL();
    String contextPath = req.getContextPath();
    String servletPath = req.getServletPath();
    String requestParamValue = req.getParameter("myParam");
    String contentType = req.getContentType();
    String user = req.getRemoteUser();
    boolean isAdmin = req.isUserInRole("admin");
    List shoppingCart = (List)
        req.getSession().getAttribute("shopping-cart");
    Object myTempData = req.getAttribute("my temp data");
    //Other operations
}
```

# Scrittura:(Http)ServletResponse

- La risposta al client consente di accedere alle seguenti informazioni:
  - Codici di stato
  - Headers
  - Cookies
  - Contenuto: length, type, body
  - URL Encoding: session tracking
- È importante specificare il codice di stato (default HTTP 200 OK) e gli headers della risposta HTTP prima che questa sia inviata al client

# (Http)ServletResponse

```
protected void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
    throws ServletException, IOException {
    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();
    out.println("<html>");
    out.println("\t<head><title>Hello World</title></head>");
    out.println("\t<body>");
    out.println("\t\t<p>Hello " + req.getRemoteAddr() + "!</p>");
    out.println("\t\t<p>Server time is " + new Date() + "</p>");
    out.println("\t</body>");
    out.println("</html>");
    out.flush(); // sends data back to the client
    out.close(); // not required (called automatically)
}
```

# Java Servlet: Distruzione

- Il metodo `destroy` viene invocato ogni volta che la Servlet deve essere deallocata
  - ad es.: stop del server, reloading dell'applicazione
- Possibile chiamata del metodo `service` su una Servlet su cui è stato invocato il metodo `destroy` in ambiente multi-threading
  - Thread-safe
- Tutte le risorse allocate al momento della `init` sono rilasciate
- Il metodo `destroy` viene chiamato una sola volta durante il ciclo di vita della Servlet



# Java Servlet: Distribuzione

```
public class IPLoggerServlet extends HttpServlet {
    private Writer ipLog = null;
    public void init(ServletConfig config) throws
        ServletException {
        super.init(config);
        try {
            ipLog = new FileWriter(config.getInitParameter("file"));
        } catch (IOException e) {
            throw new ServletException("Failed to open log file");
        }
    }
    protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws
        ServletException, IOException {
        this.ipLog.write(req.getRemoteAddr() + "\n");
        resp.setContentType("text/plain");
        resp.getWriter().println("Logged " + req.getRemoteAddr());
    }
    public void destroy() { //like finalize with a guarantee
        try {
            this.ipLog.flush();
            this.ipLog.close();
        } catch (IOException e) {
            super.log("Failed to flush/close log file", e);
        }
    }
}
```

# Java Servlet: Deployment

- Prima di poter essere usata una Servlet deve essere:
  - Compilata tramite le classi fornite dalla Servlet API
    - Java EE SDK
    - Servlet container: `${jboss.common.lib.url}/servlet-api.jar`
  - specificata nel descrittore dell'applicazione web (WEB-INF/web.xml)
  - impacchettata in un archivio WAR
  - distribuita su un Servlet container (Tocat)
  - acceduta tramite URL(s)



# Il Descrittore Web: WEB-INF/web.xml

- File di configurazione dell'applicazione web
- Specifica:
  - Nome, descrizione
  - Filters e Servlets (mapping + init parameters)
  - Listeners
  - Session timeout
  - Welcome files: index.html, index.jsp, etc.
  - ...

# Il Descrittore Web: WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <display-name>My Application</display-name>
  <description>
    Description of my application
  </description>
  ...
</web-app>
```

# Java Servlet: Definizione e Mapping

```
<!-- Definizione -->
<servlet>
  <servlet-name>IPLoggerServlet</servlet-name>
  <servlet-class>example.servlet.IPLoggerServlet</servlet-class>
  <init-param>
    <param-name>file</param-name>
    <param-value>/WEB-INF/ip.log</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>

<!-- Mapping -->
<servlet-mapping>
  <servlet-name>IPLoggerServlet</servlet-name>
  <url-pattern>/ip</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>IPLoggerServlet</servlet-name>
  <url-pattern>/logmyip</url-pattern>
</servlet-mapping>
```

**Definizione**

**Nome**

**Fully-qualified  
Class name**

**Parametri di inizializzazione  
(opzionali)**

**URL Mappings**

# Struttura Applicazione Web

- Supponiamo di aver sviluppato l'applicazione `myServletApp`
- Packaging in una gerarchia di directories specifica
  - `myServletApp`
    - Contenuti statici (.html, .gif, .jpg, .js, .css, etc.)
    - File JSP (.jsp)
    - WEB-INF/web.xml (descrittore obbligatorio)
    - WEB-INF/classes/ risultato della compilazione dei sorgenti (.class)
    - WEB-INF/lib/ librerie aggiuntive (.jar)

# Struttura Applicazione Web (2)

- Supponiamo di:
  - aver sviluppato l'applicazione myServletApp
  - aver implementato 3 Servlets
    - HelloServlet, HelloWorldServlet, IPLoggerServlet
    - usando il package it.uniroma2.pjdm.servlet
- La struttura delle directories sarà la seguente:
  - myServletApp/
    - WEB-INF/
      - web.xml
      - classes/
        - it/uniroma2/pjdm/servlet/
          - HelloServlet.class
          - HelloWorldServlet.class
          - IPLoggerServlet.class

# Struttura Applicazione Web (3)

- L'applicazione web può essere compressa in un archivio WAR (file .war)
- L'archivio non deve contenere la root dell'applicazione (myServletApp/)– il nome dell'archivio diventa quello dell'applicazione
- L'archivio WAR può essere generato con i tools messi a disposizione dal JDK o da Eclipse

# Deployment dell'applicazione web

- Operazione specifica che dipende dal particolare Java EE AS
- Solitamente è sufficiente copiare la directory root dell'applicazione (o il corrispondente archivio WAR) nella directory di deployment del server Java EE
- Su Tomcat questo si traduce in copiare all'interno di `${tomcat.server.home.url}/webapp/`
  - la directory root dell'applicazione `myServletApp/`  
oppure
  - L'archivio WAR dell'applicazione `myServletApp.war`

# Accesso alle Servlet

- Garantito tramite URL mapping
- URLs multipli possono puntare alla stessa Servlet
- Gli URLs sono relativi al contesto dell'applicazione (/myServletApp/)



# Java Servlet + Eclipse

- Eclipse semplifica notevolmente lo sviluppo di un'applicazione web
- Usare il wizard per creare un Progetto Web Dinamico
  - questa operazione aggiunge tutte le librerie Java EE necessarie al progetto
- Usare il wizard per la creazione di Servlet
  - Facilita l'implementazione dei metodi della Servlet
  - Aggiorna automaticamente il descrittore (web.xml)
- Packaging WAR dell'applicazione
- Un-/Re-/Deploy del WAR

# Eclipse: Creazione Progetto Web Dinamico

- File → New → Dynamic Web Project
- Specificare il nome del progetto
  - (ad es. PJDM2018-2019\_servlet)
- Selezionare Tomcat come target runtime
  - Click su Next
- Nel tab **Java** Click su Next
- Nel tab Web Module
  - **Selezionare “Generate web.xml deployment descriptor”**
  - Click su Next
- → Finish

## Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.



Project name: PJDM2018-2019\_servlet

### Project location

☒ Use default location

Location: /Users/danilo/progetti/reveal\_git\_workspace/PJDM2018-201

Browse...

### Target runtime

Apache Tomcat v8.0



New Runtime...

### Dynamic web module version

3.1



### Configuration

Default Configuration for Apache Tomcat v8.0



Modify...

A good starting point for working with Apache Tomcat v8.0 runtime. Additional facets can later be installed to add new functionality to the project.

### EAR membership

☐ Add project to an EAR

EAR project name: EAR



New Project...

### Working sets

☐ Add project to working sets

Working sets: dnn



Select...



< Back

Next >

Cancel

Finish

# Eclipse: Creazione Servlet (1)


- Click destro sul progetto creato New → Servlet
- Specificare il package
  - ad es.: `it.uniroma2.pjdm.servlet`
- Specificare il nome della classe Servlet
  - ad es.: `MyServlet`
- Click su Next → Finish
- La Servlet verrà automaticamente mappata sull'URL `/MyServlet`

● ● ●

Create Servlet

## Create Servlet

Specify class file destination.



Project:

PJDM2018-2019\_servlet

⌵

Source folder:

/PJDM2018-2019\_servlet/src

Browse...

Java package:

it.uniroma2.pjdm.servlet

Browse...

Class name:

MyServlet

Superclass:

javax.servlet.http.HttpServlet

Browse...

☐ Use an existing Servlet class or JSP

Class name:

MyServlet

Browse...

?

< Back

Next >

Cancel

Finish

# Mapping tra url e Servlet

- Come fa il Web Server a collegare una url come

`http://localhost:8080/PJDM2018-2019_servlet/MyServlet`

alla servlet?

- Usando la Annotation `@WebServlet("/MyServlet")`

```
/**
 * Servlet implementation class MyServlet
 */
@WebServlet("/MyServlet")
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public MyServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
}
```

# Eclipse: web.xml

- ... oppure aggiungere un parametro di inizializzazione nel descrittore `web.xml` (nella cartella `[PROGETTO]/WebContent/WEB-INF/`)
- **NOTA:** nel file `web.xml` è possibile usare il tag `<init-param>` per definire dei parametri globali

```
<servlet>
    <description></description>
    <display-name>MyServlet</display-name>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>it.uniroma2.pjdm.servlet.MyServlet</servlet-class>
    <init-param>
        <param-name>name</param-name>
        <param-value>World</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
```

# Differenze tra `init-param` e `context-param`

- Nel `web.xml` si possono specificare i parametri di una specifica servlet con il tag `init-param` (vedi slide precedente)
- Per poter fornire un parametro a TUTTE le servlet (o quando si usa l'annotazione `@WebServlet("/MyServlet")`)

```
<web-app>
    <context-param>
        <param-name>Country</param-name>
        <param-value>India</param-value>
    </context-param>
    <context-param>
        <param-name>Age</param-name>
        <param-value>24</param-value>
    </context-param>
</web-app>
```

E nella servlet si possono usare i seguenti metodi

```
getServletContext().getInitParameter("Country");
getServletContext().getInitParameter("Age");
```



# I metodi: doGet e doPost

- Implementare il metodo **doGet**
  - per rispondere a richieste HTTP GET
- Implementare il metodo **doPost**
  - per rispondere a richieste HTTP POST
- Spesso si implementa solo uno dei due metodi (ad es. doGet) e l'altro (ad es. doPost) richiama semplicemente il metodo già implementato

# I metodi: doGet e doPost

## doGet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();
    out.print("Helloo");
    out.println(super.getInitParameter("name"));
}
```

## doPost

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    doGet(request, response);
}
```

# Eclipse: Packaging dell'applicazione

- Click destro sul progetto Export → WAR file
- Specificare il nome, la destinazione e ottimizzazione per il target runtime (Tomcat 9)

## WAR Export

Export Web project to the local file system.



Web project: PJDM2018-2019\_servlet

Destination: /Users/danilo/PJDM2018-2019\_servlet.war

Browse...

### Target runtime

☒ Optimize for a specific server runtime

Apache Tomcat v8.0

☐ Export source files

☐ Overwrite existing file



< Back

Next >

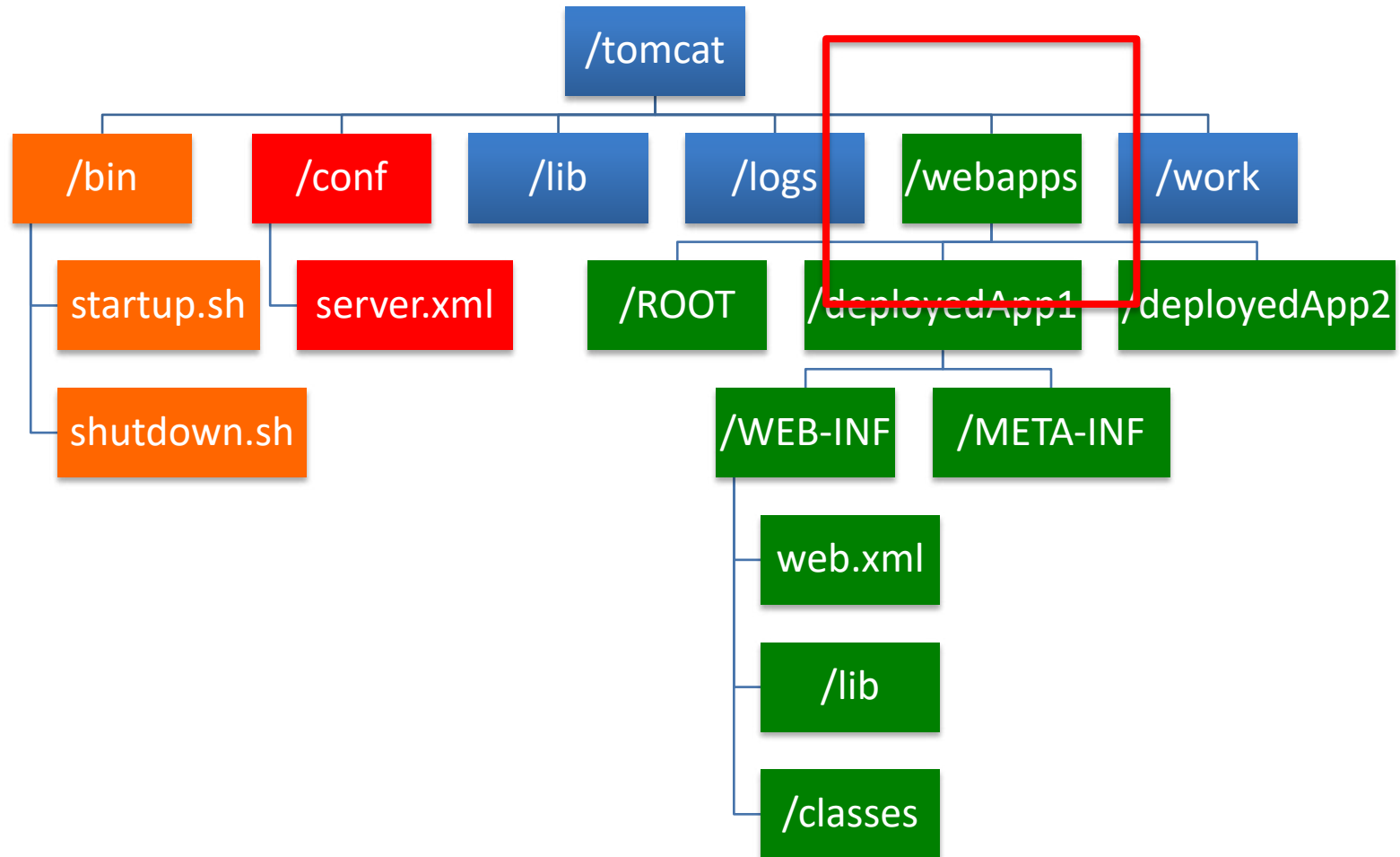
Cancel

Finish

# Deployment dell'applicazione

- Copiare l'archivio WAR generato all'interno della directory `[TOMCAT_DIR]/web_apps`
  - Avviare il server Tomcat
  - Osservare il deployment sul log della console
    - `[TOMCAT_DIR]/logs/catalina.out`
- 
- `23-May-2019 12:09:13.505 INFORMAZIONI [Catalina-utility-1]  
org.apache.catalina.startup.HostConfig.deployWAR Deploying web application  
archive [/Users/danilo/tomcat/apache-tomcat-9.0.19/webapps/PJDM2018-  
2019_Servlet.war]`
  - `23-May-2019 12:09:13.564 INFORMAZIONI [Catalina-utility-1]  
org.apache.catalina.startup.HostConfig.deployWAR Deployment of web  
application archive [/Users/danilo/tomcat/apache-tomcat-  
9.0.19/webapps/PJDM2018-2019_Servlet.war] has finished in [60] ms`

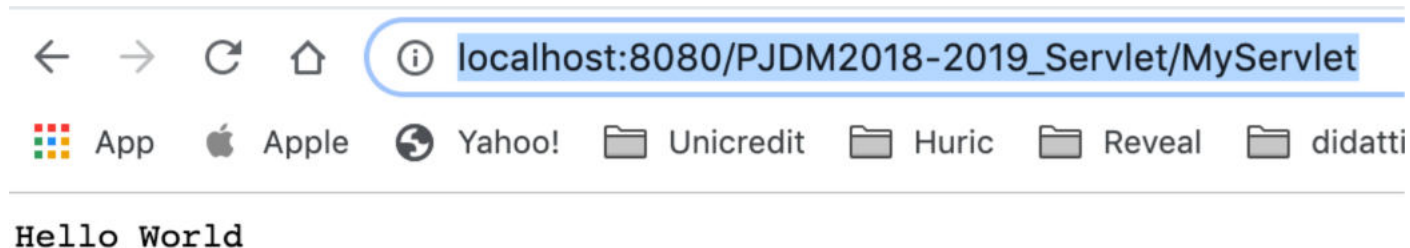
# Deployment dell'applicazione



# Testing dell'applicazione

Accedere al seguente URL:

[http://localhost:8080/PJDM2018-2019\\_Servlet/MyServlet](http://localhost:8080/PJDM2018-2019_Servlet/MyServlet)



NOTA: le url hanno il formato

`http://[IP]:[PORT]/[WAR_FILE_NAME]/[NOME_SERVLET]`

# Cos'è una JSP?

- Di fatto una pagina JSP è una Servlet!
- Oltre ai vantaggi delle Servlet offre:
  - Look-and-feel HTML (plain-text)
  - Facilità per i web designer
  - Sviluppo senza programmare in Java tramite l'uso di custom tags e expression language (EL)
  - Compilazione e deployment automatico da parte del Servlet container
- Si dovrebbe concentrare solo sul livello di “presentazione” (GUI)
  - La logica applicativa implementata altrove



# Java Servlet vs. JSP (1)

```
@WebServlet("/HelloHTML")
public class HelloWorldHTMLServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
        ServletException, IOException {
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("\t<head><title>Hello World</title></head>");
        out.println("\t<body>");
        out.println("\t\t<p>Hello " + req.getRemoteAddr() + "</p>");
        out.println("\t\t<p>Your browser is " + req.getHeader("User-Agent") +
"</p>");
        out.println("\t\t<p>Server time is " + new Date() + "</p>");
        out.println("\t</body>");
        out.println("</html>");
        out.close();
    }
}
```

# Java Servlet vs. JSP (2)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@page import="java.util.Date"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
<title>Hello World</title>
</head>
<body>
    <p>Hello <%=request.getRemoteAddr()%>!</p>
    <p>Server time is <%=new Date()%></p>
</body>
</html>
```

# JSP: Ciclo di Vita

- Simile a quello di una Servlet “classica”
- Se l’istanza della Servlet corrispondente alla JSP richiesta non esiste, il Servlet container:
  - Compila la JSP nella Servlet corrispondente
  - Carica la classe e la istanzia
  - Inizializza l’istanza della classe Servlet tramite il metodo `jspInit`
- Per ogni richiesta utente, il container invoca il metodo `_jspService` della Servlet associata alla JSP, passando gli oggetti `request` e `response`
- Se rimossa, il container invoca il metodo `jspDestroy`

# JSP: Compilazione

- La compilazione da JSP a Servlet avviene in 2 passi:
  1. Il file .jsp viene compilato in un file .java dal compilatore del container
    - Jasper su Tomcat
  2. Il file .java generato dal compilatore viene a sua volta compilato in un file .class dal compilatore Java standard (javac)
    - Ecco perché il Servlet container necessita dell'intero JDK e non della sola piattaforma JRE

# Da JSP a Java Servlet:

## \_jspService

```
...
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response) throws IOException, ServletException {
    ...
    out.write("\r\n");
    out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" \"http://www.w3.org");
    out.write("<html>\r\n");
    out.write("<head>\r\n");
    out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">\r\n");
    out.write("<title>Hello</title>\r\n");
    out.write("</head>\r\n");
    out.write("<body>\r\n");
    String name = request.getParameter("name");
    if (name == null || name.length() == 0) {
        out.write("\r\n");
        out.write("<form>\r\n");
        out.write("  <h2>What is your name?</h2>\r\n");
        out.write("  <p>Name: <input type=\"text\" name=\"name\" /> \r\n");
        out.write("  <input type=\"submit\" value=\"Send\" />\r\n");
        out.write("  </p>\r\n");
        out.write("</form>\r\n");
    } else {
        out.write("\r\n");
        out.write("<h2>Hello ");
        out.print(name);
        out.write("!</h2>\r\n");
    }
    out.write("\r\n");
    out.write("</body>\r\n");
    out.write("</html>");
    ...
}
...
```

# JSP API

- Il contenuto della pagina JSP viene eseguito all'interno del metodo generato `_jspService`
- Tutti i contenuti statici sono convertiti in chiamate a `out.write()`
- Tutti contenuti dinamici (inclusi nei tag `<% %>`) vengono eseguiti come codice Java “normale”

# JSP: Oggetti “impliciti”

- Il metodo generato automaticamente con la compilazione .jsp → .java definisce e inizializza alcuni oggetti
- Questi oggetti possono essere riferiti all'interno della stessa pagina JSP

```
HttpServletRequest request  
HttpServletResponse response  
PageContext pageContext  
HttpSession session  
ServletContext application  
ServletConfig config  
JspWriter out
```

# JSP: Oggetti “impliciti”

```

public void _jspService(HttpServletRequest request,
    HttpServletResponse response) throws java.io.IOException,
    ServletException {
    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;

    try {
        _jspxFactory = JspFactory.getDefaultFactory();
        response.setContentType("text/html; charset=ISO-8859-1");
        pageContext = _jspxFactory.getPageContext(this, request,
            response, null, true, 8192, true);
        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;
        ...
    }
    ...
}

```

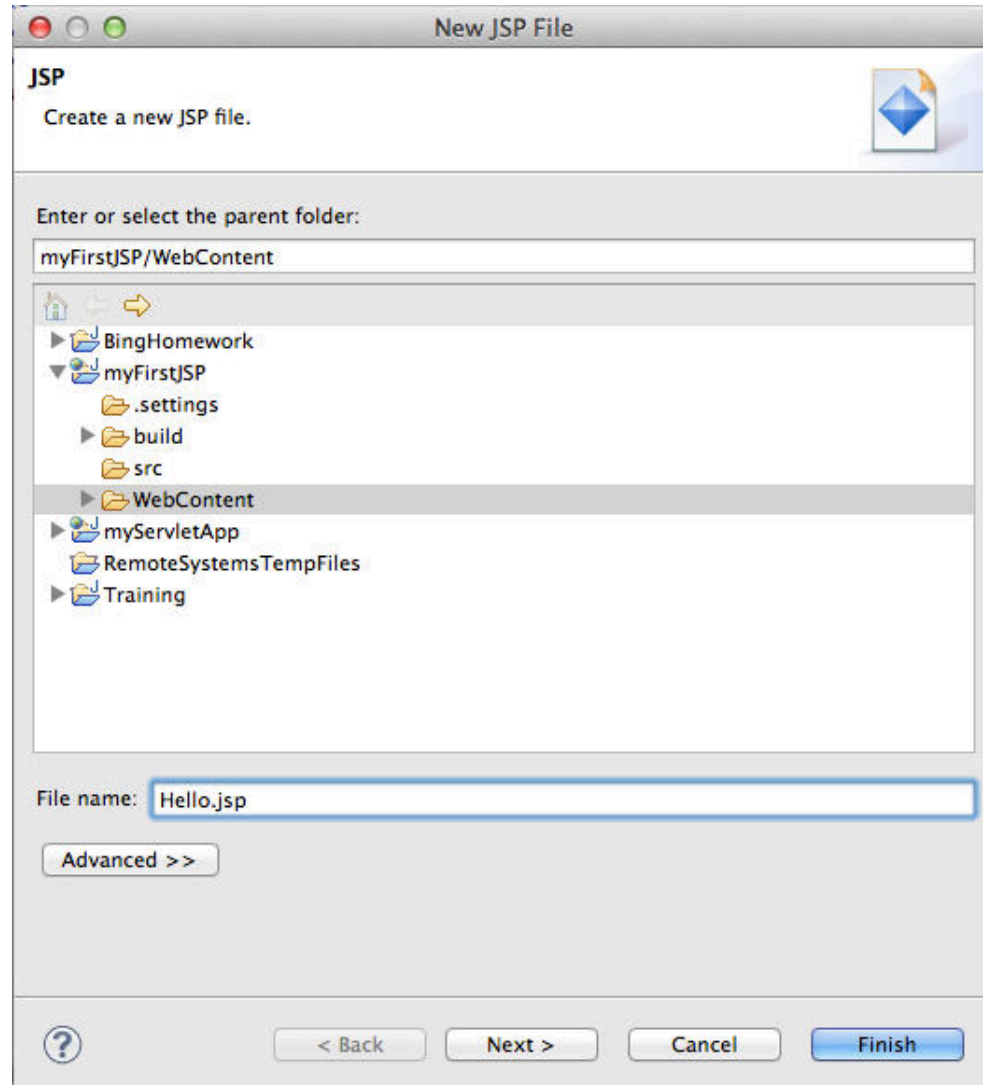


# Sintassi JSP: Direttive

- Le direttive JSP controllano la compilazione da JSP a Servlet
- Non hanno alcun effetto sul comportamento a run-time
- Controllano solo la fase di generazione di codice

```
<%@page attributes %> or in XML as <jsp:directive.page .../>
<%@include file="path" %> or in XML as <jsp:directive.include .../>
<%@taglib prefix="prefix" uri="uri" %> or in XML as <html xmlns:prefix="uri"
...>
```

# Lab: MyFirstJSP con Eclipse



# Lab: MyFirstJSP con Eclipse

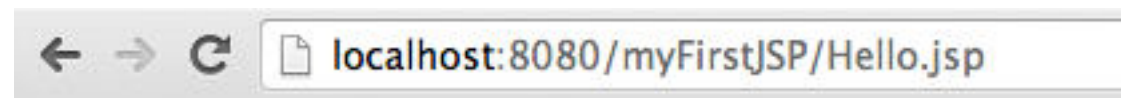
Hello.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello JSP</title>
</head>
<body>

</body>
</html>
```

# Hello.jsp: Implementazione

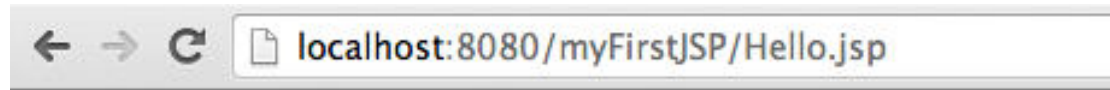
```
<body>
<%
String name =
request.getParameter("name");
if (name == null || name.length() == 0) {
%>
<form action="">
<h2>What's your name?</h2>
<p>Name: <input type="text" name="name"/>
<input type="submit" value="Send"/>
</p>
</form>
<%
}
else {
%>
<h2>Hello <%= name %>!</h2>
<%
}
%>
</body>
```



**What's your name?**

Name:

# Hello.jsp: Deployment+Testing



**What's your name?**

Name:

## HTTP GET



**Hello gabriele!**

## HTTP POST



**Hello gabriele!**