Laboratorio di Basi di Dati e Multimedia

Laurea in Informatica Multimediale

Docente: Carlo Combi
Email: carlo.combi@univr.it

Lezione 6

Servlet

- *Servlet:* classi java che estendono le funzionalità di un server Web generando dinamicamente le pagine Web

  Riferimento: http://java.sun.com/products/servlets

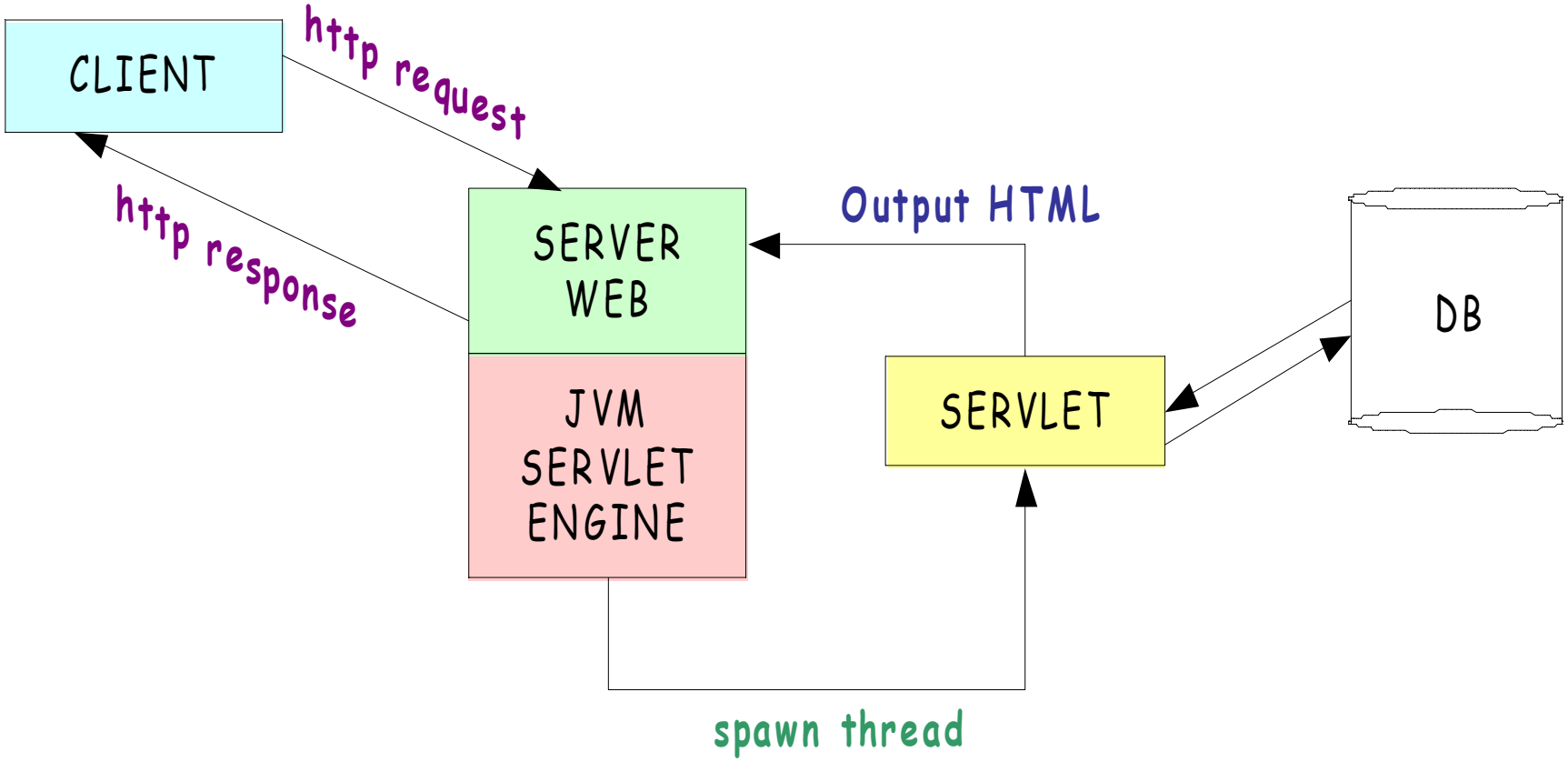
- Le servlet operano in una macchina virtuale dell'ambiente controllato dal server e comunicano con i client utilizzando semplicemente i comandi HTTP

Servlet Engine

- Servlet Engine: ambiente runtime che gestisce il caricamento e lo scaricamento delle servlet e collabora con il server Web per dirigere le richieste alle servlet e per rinviare il loro output ai client (browser).
- Esistono diverse implementazioni del *Servlet Engine* (detto anche *Servlet container*). L'implementazione di riferimento ufficiale è sviluppata dal gruppo Apache con il progetto Jakarta-Tomcat e si chiama **Tomcat**

Riferimento: http://jakarta.apache.org/tomcat/index.html

Architettura Servlet



Servlet: ciclo di vita

- Il Servlet Engine svolge le seguenti operazioni:
  - carica una Servlet la prima volta che viene richiesta;
  - in caricamento richiama il metodo **init()** della Servlet;
  - gestisce le richieste richiamando il metodo **service()** delle Servlet per ogni richiesta HTTP da gestire (service() richiama poi i metodi **doGet(…)** o **doPost(…)**);
  - alla chiusura richiama il metodo **destroy()** di ciascuna Servlet.

Il metodo **init()**

- Quando il servlet engine riceve la richiesta di una servlet, controlla se è già stata caricata
  - Se non è già stata caricata, il servlet engine carica la servlet richiesta e richiama il costruttore della sua classe per ottenere un'istanza della servlet
- In metodo **init()** viene chiamato una sola volta, (quando la servlet viene caricata).
- All'interno del metodo **init()** la servlet può svolgere qualsiasi operazione di avvio, come ad esempio attivare la connessione con una base di dati.

Il metodo **destroy()**

- Il metodo **destroy()** viene usato dal Servlet Engine per scaricare una servlet

- Richiamando esplicitamente **destroy()** non si scarica la servlet

NB: Questa operazione può essere eseguita **solo** dal Servlet Engine.

Struttura di una Servlet (1/5)

- Prima di tutto si specificano le istruzioni che dicono al compilatore che verranno utilizzate le classi di alcuni packages generali e specifici per il protocollo HTTP

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

Package necessari da importare

- Segue la dichiarazione della classe:

Estendere HttpServlet

```
public class nomeClasse extends HttpServlet {
```

L'interfaccia **javax.servlet.Servlet** fornisce una sottoclasse **HttpServlet** che è la classe base per la creazione di una Servlet.

Struttura di una Servlet (2/5)

Una servlet che non ha particolari requisiti per le azioni di avvio e chiusura modifica solo il metodo **doGet()** che verrà richiamato dal metodo **service()** della superclasse **HttpServlet**

```
public void doGet(HttpServletRequest request,
        HttpServletResponse response)
throws IOException, ServletException {
```

Il metodo **doGet** corrisponde al metodo GET della richiesta HTTP.

Il metodo **doGet** viene richiamato dal servlet engine per elaborare una richiesta HTTP GET. I parametri di input, le intestazioni HTTP e lo stream di input possono essere ottenuti dall'oggetto request mentre le intestazioni di risposta e lo stream di output possono essere gestiti attraverso i metodi dell'oggetto response.

Struttura di una Servlet (3/5)

- Prima di indicare i risultati al client, occorre specificare le intestazioni HTTP da inviare.

```
response.setContentType("text/html; charset=ISO-8859-1");
```

  In questo caso l'unica intestazione è **ContentType** che sarà **text/html**.
- La creazione di una pagina HTML prevede l'invio delle istruzioni HTML sullo stream di output associato alla richiesta HTTP. Lo stream di output può essere ottenuto dall'oggetto **response** utilizzando il metodo **getWriter()** che consente di scrivere caratteri:

```
PrintWriter out = response.getWriter();
```

Struttura di una Servlet (4/5)

- Infine si prepara il testo della pagina HTML:

```
out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
        HTML4.01Transitional//EN\"");
out.println("\"http://www.w3.org/TR/REC-
html40/loose.dtd\">");
out.println("<html>");
out.println("<head>");
...inserire tutti i dati dell'intestazione
out.println("</head>");
out.println("<body>");
 ...inserire tutti i dati del corpo
println("</body>");
out.println("</html>");
```

Struttura di una Servlet (5/5)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class nomeClasse extends HttpServlet {
    ...eventuali variabili di classe
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        /* Definisco il tipo MIME del response. Caldamente richiesto dalle specifiche! */
        response.setContentType("text/html; charset=ISO-8859-1");
        PrintWriter out = response.getWriter();
        //Inserisco il DOCTYPE!
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
        4.01Transitional//EN\"");
        out.println("\"http://www.w3.org/TR/REC-html40/loose.dtd\">");
        out.println("<html>");
        out.println("<head>");
        ...inserire tutti i dati dell'intestazione
        out.println("</head>");
        out.println("<body>");
         ...inserire tutti i dati del corpo
        println("</body>");
        out.println("</html>");
    }
}
```

Esempio ServletHelloWWW

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

public class ServletHelloWWW extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
      response.setContentType("text/html; charset=ISO-8859-1");
      PrintWriter out = response.getWriter();
      String docType =
            "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01” +
            “Transitional//EN\“>\n”;
      out.println(docType +
             “<HTML>\n” +
             “<HEAD><TITLE>Hello World </TITLE></HEAD>\n” +
                  “<BODY>\n” +
             “<H1> Hello World </H1>\n” +
             “</BODY></HTML>”);
    }
}
```

Esempio ServletLire(1/3)

```
import java.io.*;
import java.text.*;
/* Importo i package relativi alle servlet. */
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletLire extends HttpServlet {
    private static final DecimalFormat FMT = new DecimalFormat("#0.00");
    /** Valore da utilizzare per ottenere la conversione. */
    private static final double VALORE = 1936.27;
    /**
     * Costruttore di default della classe. Non compie nessuna operazione, crea
     * solamente un nuovo oggetto.
    */
    public ServletLire() {
    }
```

Esempio ServletLire (2/3)

```
/**
  * Questo metodo viene richiamato in risposta ad una richiesta HTTP di tipo GET.
  * Ottiene lo stream di output e scrive sullo stesso il codice HTML da visualizzare
  * nel browser.
  *
  * @param request Oggetto che incapsula la richiesta HTTP effettuata dal client.
  * @param response Oggetto che permette alla Servlet di impostare lo stato e
l'header.
**/
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException {

/* Ottengo lo stream di output su cui inviare il codice HTML da  visualizzare. */
PrintWriter out = response.getWriter();
String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01” +
                    “Transitional//EN\“>\n”;


 // Imposto il tipo di output
response.setContentType("text/html");

// Scrivo sullo stream di output il codice HTML da visualizzare
out.println("<html>");
out.println("  <head>");
out.println("    <title>Servlet di prova</title>");
out.println("  </head>");
```

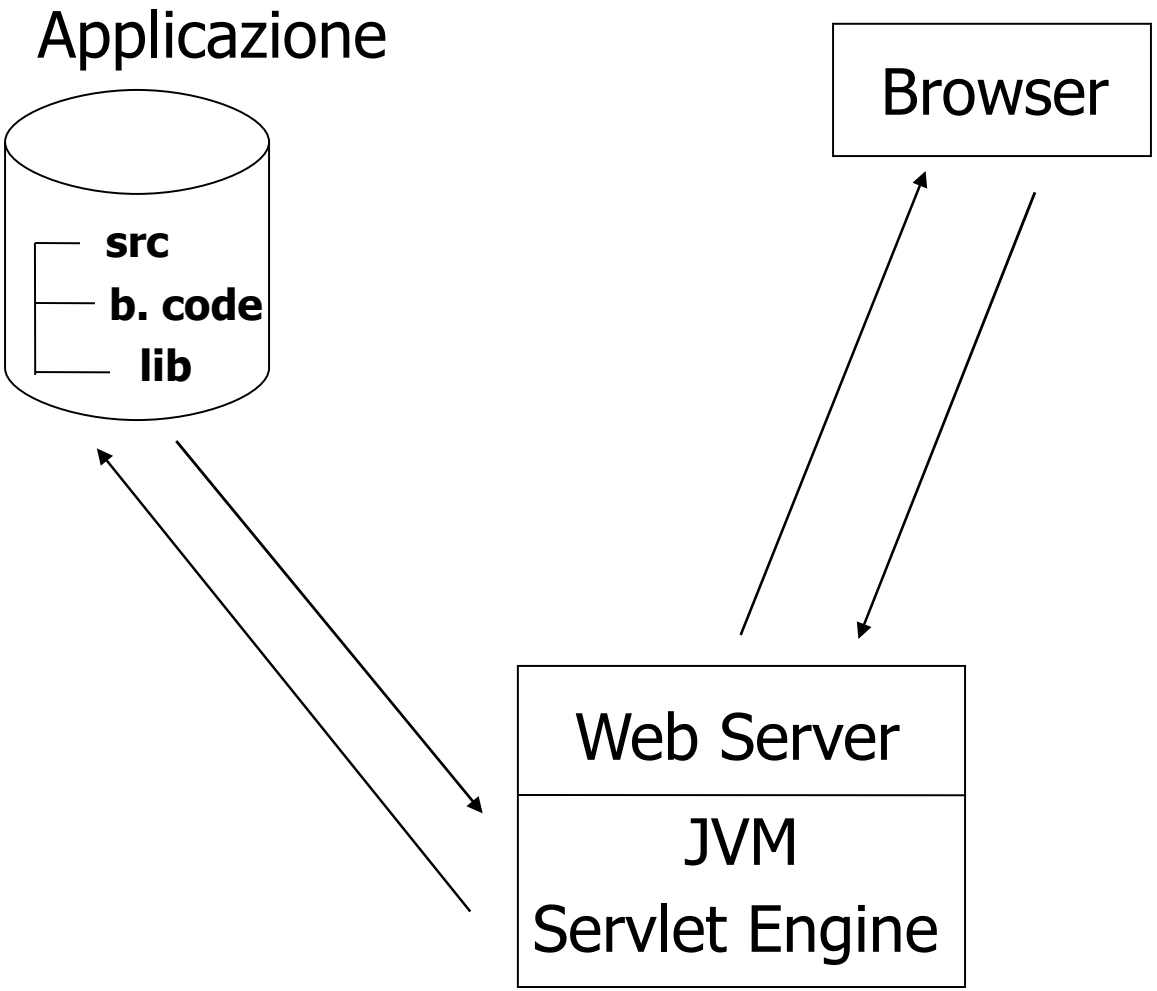
Esempio ServletLire (3/3)

```
out.println("  <body bgcolor=\"white\">");
out.println("    <h1 align=\"center\">Conversione Euro-Lire </h1>");
out.println("    <table border=\"1\" cellspacing=\"5\"
cellpadding=\"5\" align=\"center\" summary=\"Conversione\">");
out.println("      <tr>");
out.println("        <th>Euro</th>");
out.println("        <th>Lire</th>");
out.println("      </tr>");
// Ciclo per stampare la conversione
for (double i=1; i<=10; i++) {
    out.println("      <tr>");
    out.println("        <td align=\"center\">" + FMT.format(i) + "</td>");
    out.println("        <td align=\"center\">" +  FMT.format(VALORE * i) + "</td>");
    out.println("      </tr>");
}
out.println("    </table>");
out.println("  </body>");
out.println("</html>");
// Chiudo lo stream di output
out.close();
    }
}
```

Tomcat

- Tomcat realizza sia l'engine per Servlet e JSP, sia l’engine per HTTP
  - Può essere utilizzato come server web completo di tecnologia servlet/jsp
- La versione che useremo di Tomcat è la 5.5, che implementa le specifiche Java Servlet 2.4 e JavaServer Pages 2.0
- Tomcat è scritto in Java, per cui richiede un Java Runtime Environment (JRE) per poter funzionare (almeno la versione 1.2)

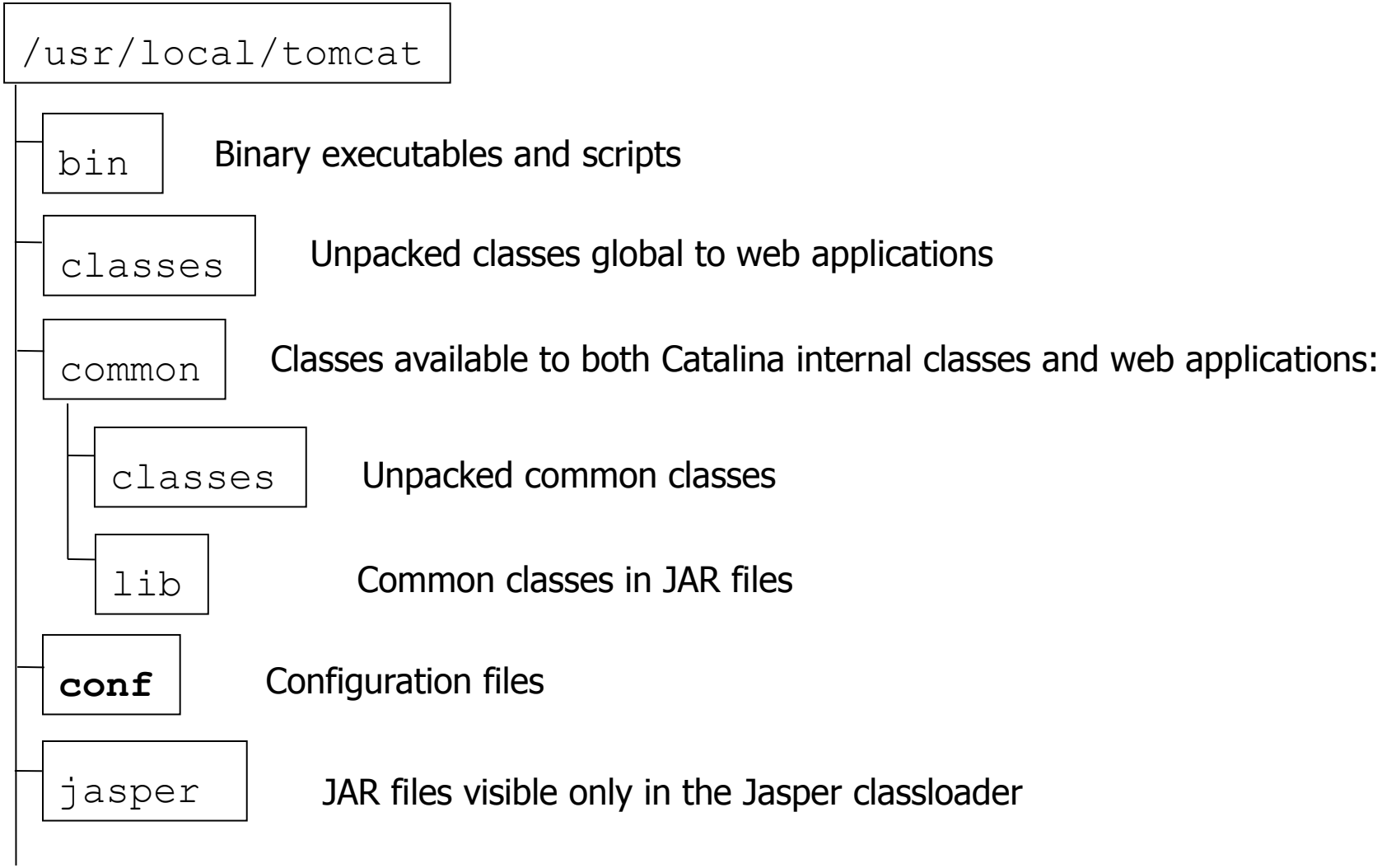
Architettura di Tomcat



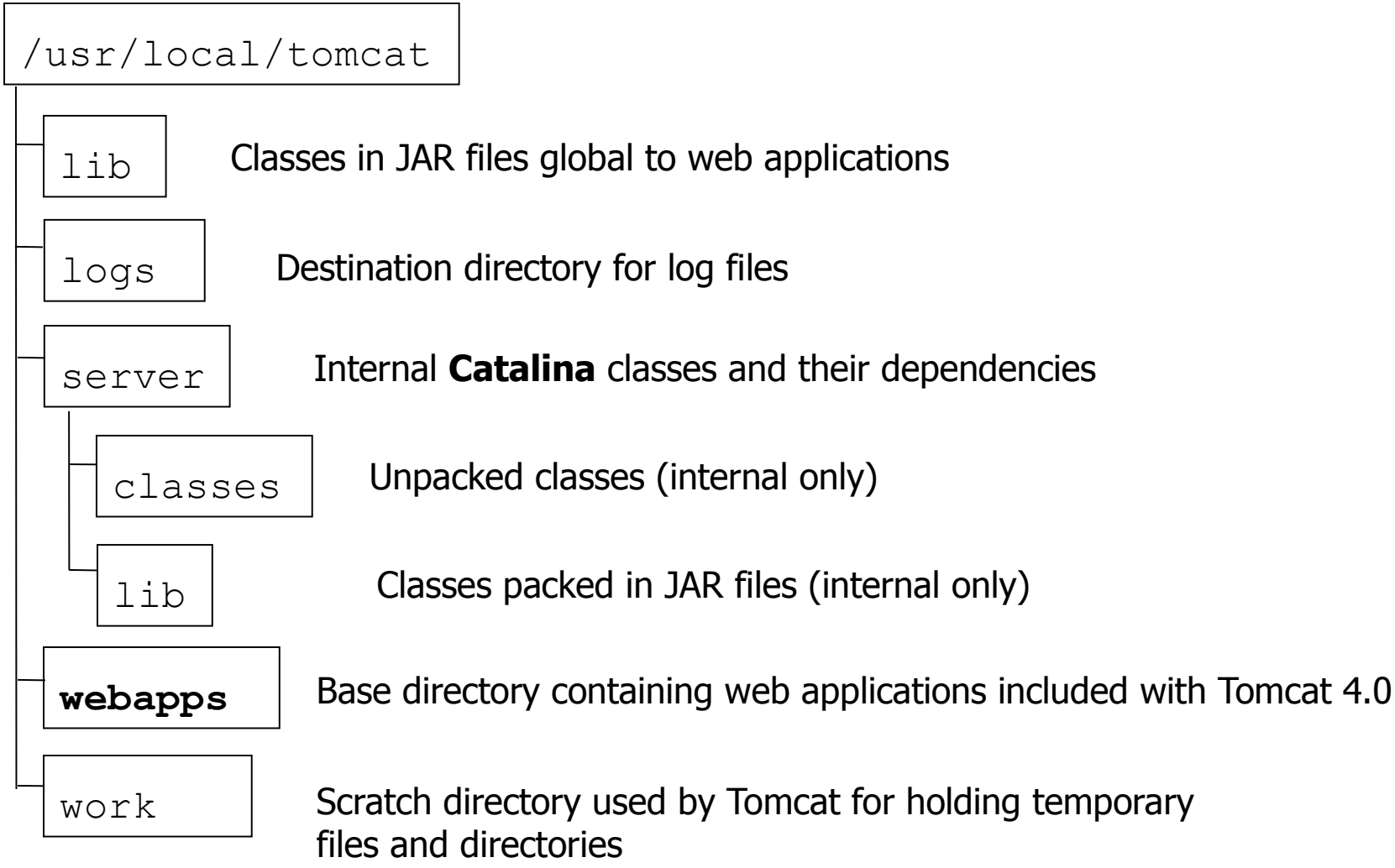
Utilizzare Tomcat

- Installare la suite Tomcat.
- Configurare Tomcat.
- Organizzare i documenti html, le servlet e le jsp che si vogliono utilizzare per realizzare un sito in una opportuna gerarchia di directory.
- Far partire Tomcat.

Tomcat: struttura directory



Tomcat: struttura directory (2)



Tomcat: struttura directory (3)

- Nella directory **conf** sono presenti tutti i file di configurazione dell'engine, tra cui il file **server.xml** è il principale (e necessario).

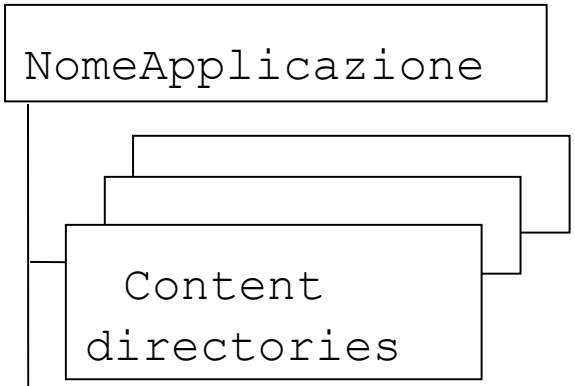
- Nella directory **webapps** sono presenti tutte le directory contenenti le *applicazioni web* che l'engine può eseguire.

Applicazioni Web

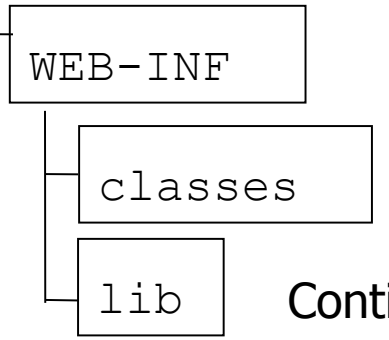
- Un'applicazione Web è l'insieme di file html, immagini, servlet, jsp, ecc. che servono per realizzare le pagine di un sito Web.
- In Tomcat un'applicazione Web è denominata **Context**.

Applicazioni Web (2)

Il layout di un'applicazione web (context) è fissato dallo standard Java Servlet ed è:



In queste directory si organizzano tutti i file che non sono Servlet o classi java (/html, /images, /jsp)



Contiene il file di configurazione dell'applicazione **web.xml**

Contiene le Servlet compilate e le classi

Contiene tutte le librerie necessarie all'applicazione

Configurazione di Tomcat (1)

- Scaricare nella propria home il pacchetto tomcat.zip dalla pagina web del corso

- Scompattare il pacchetto nella propria home

  unzip tomcat.zip

Configurazione di Tomcat (2)

- **Editare il file .bashrc aggiungendo le seguenti righe e sostituendo i puntini con la directory di installazione di tomcat nella propria home:**

```
# Path del pacchetto JDBC e posizione locale
CLASSPATH=$CLASSPATH:/usr/share/java/postgresql-jdbc3.jar
# Path del pacchetto SERVLET e JSP
CLASSPATH=$CLASSPATH:/usr/share/java/jsp-api.jar
CLASSPATH=$CLASSPATH:/usr/share/tomcat5.5/common/lib/servlet-api.jar
export CLASSPATH
# Home di Java necessaria per tomcat
JAVA_HOME=/usr/lib/jvm/java-6-sun
export JAVA_HOME
# Variabile necessaria a tomcat
CATALINA_BASE=~/…/tomcat        # sostituire i puntini con la directory in cui è
                                # stato scompattato tomcat
                                # se scompattato nella home: togliere i puntini
export CATALINA_BASE
PATH=$PATH:~/.../tomcat/bin     # sostituire i puntini con la directory in cui è
                                # stato scompattato tomcat
                                # se scompattato nella home: togliere i puntini

export PATH
```

Configurazione di Tomcat (3)

- Aprire una nuova shell: in questo modo il file .bashrc viene eseguito automaticamente
- Nella directory WEB-INF di un context (ad esempio ROOT) creare la dir lib
- Dalla directory lib creare il link simbolico nel seguente modo

```
ln -s /usr/share/java/postgresql-jdbc3.jar
```

Esecuzione di Tomcat

- Una volta configurato, l'attivazione e spegnimento del server Tomcat avviene mediante i seguenti comandi:
  - **Attivazione server:** tomcat start
  - **Spegnimento server:** tomcat stop

NB: In *Preferences->Advanced->Proxies* di Mozilla impostare il campo No proxy for: **localhost**

NB: Quando si fa terminare l'esecuzione di tomcat possono rimanere attivi dei processi che interferiscono con eventuali esecuzioni successive, si utilizzi quindi:

killall -9 java

Risposta Server

- Se non ci sono errori di configurazione o di installazione, il server risponde (dopo qualche secondo necessario per l’inizializzazione) all'URL http://localhost:8080/index.html

con la seguente pagina:

Tomcat works! - Mozilla Firefox

File Modifica Visualizza Cronologia Segnalibri Strumenti Guida

http://localhost:8080/

Google

Tomcat works!!!

Completato

Compilazione Servlet

- **Le servlet sono delle classi java, quindi per poterle eseguire prima è necessario compilarle nel formato bytecode**

- Per compilare una servlet è necessario utilizzare la libreria **servlet-api.jar** disponibile nella directory

  /usr/share/tomcat5.5/common/lib/

- Ci sono due modi per compilare utilizzando una libreria:
  - Settare la variabile d'ambiente CLASSPATH (solo UNA volta) in modo tale che includa la libreria e poi si compila normalmente:

```
>export CLASSPATH=$CLASSPATH:/usr/share/tomcat5.5/common/lib/servlet-api.jar
>javac MiaServlet.java
```

    - Abbiamo già inserito questa operazione nel file .bashrc

  - Utilizzare il compilatore direttamente fornendo come parametro il cammino della libreria da utilizzare:

```
>javac -classpath /usr/share/tomcat5.5/common/lib/servlet-api.jar MiaServlet.java
```

Compilazione Servlet

- Le servlet compilate (*.class) DEVONO essere memorizzate nella directory webapps/*nome*/WEB-INF/classes/ dove *nome* è il nome dell'applicazione web (context) di cui la servlet deve far parte.
- Se si pongono i sorgenti in un'altra directory si deve utilizzare l'opzione -d *dir_destinazione* del compilatore per poter compilare la servlet e memorizzare direttamente nella directory destinazione il file *.class.
  - **Ad esempio, supponiamo di voler compilare la servlet ServletHelloWWW.java (memorizzata in ~/tomcat/src/ROOT) e di volerla utilizzare nel context ROOT:**
    - **Il file sorgente della servlet è ~/tomcat/src/ROOT/ServletHelloWWW.java**
    - **La directory delle servlet della web application ROOT è ~/tomcat/webapps/ROOT/WEB-INF/classes/**
    - **È sufficiente allora spostarsi nella directory del file sorgente e eseguire javac -d ../webapps/ROOT/WEB-INF/classes/ ServletHelloWWW.java**
- I sorgenti dei nostri esempi si trovano nel context relativo.

Esecuzione Servlet

- La configurazione standard dell'engine Tomcat (server.xml) permette di invocare una servlet realizzata dalla classe java *nomeClasse* in un context *nomeContext* tramite il seguente URI: http://localhost:8080/nomeContext/servlet/nomeClasse
  - **Nota!** Viene utilizzata la porta 8080 in quanto la configurazione standard di Tomcat utilizza questa porta.
- Ad esempio, per far eseguire al servlet engine la classe java ServletHelloWWW che appartiente al context ROOT (il context ROOT è speciale... in quanto il suo nome/path è ""), si deve fornire l'URI: http://localhost:8080/servlet/ServletHelloWWW

Servlet e parametri d’input

- I parametri presenti nello header di una richiesta HTTP che deve essere gestita da una servlet sono facilmente accessibili tramite il metodo getParameter("*nome parametro*") dell'oggetto di tipo HttpServletRequest, disponibile come parametro del metodo doGet() della servlet.

- Il metodo getParameter() restituisce il parametro come oggetto di tipo String. Se il parametro rappresenta un dato di altro tipo, si deve eseguire una conversione esplicita!

Servlet e parametri d’input

- Esempio d'uso del metodo all'interno di una servlet per recuperare un **valore intero** (tipo Java int) passato come parametro di nome *importo* di una request HTTP.

```
public void doGet(HttpServletRequest request,
   HttpServletResponse response) ...
 String parImporto = request.getParameter("importo");
 int importo;
 try
 {
    importo = Integer.parseInt(parImporto);
 } catch (NumberFormatException e) {...};
 ...
```

Esempio ServletHelloWWW2

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletHelloWWW2 extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
      String parNome request.getParameter("nome");
      response.setContentType("text/html; charset=ISO-8859-1");
      PrintWriter out = response.getWriter();
      String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01” +
            “Transitional//EN\“>\n”;
      out.println(docType +
             “<HTML>\n” +
             “<HEAD><TITLE>Hello World </TITLE></HEAD>\n” +
            “<BODY>\n” +
             “<H1> Hello World </H1>\n” +
             " Hello " + parNome +
             “</BODY></HTML>”);
    }
}
```

Servlet e Form (1/2)

- È possibile utilizzare una servlet come “agente” per una FORM HTML.
- Supponendo di voler creare una FORM che richieda un nome e risponda con il documento HTML generato dalla nostra servlet ServletHelloWWW2, è sufficiente:
  - creare un file HTML (ad esempio *saluto.html*) contenente il seguente frammento:

```
...
<form
  method="get”
  action=“/servlet/ServletHelloWWW2">
<h2>Nome:</h2>
<input name="nome" type="text" maxlength="40">
<input type="submit">
</form>
…
```

Notare che l'URI dell'action è un path specificato a partire dal context oppure è possibile usare path relativi a partire dalla directory dove si trova il file HTML.

Servlet e Form (2/2)

- Salvare il file o direttamente in ROOT/saluto.html o in una sua sottodirectory (ad esempio ROOT/html/saluto.html se si vogliono organizzare i file in base al loro tipo).

- Invocare la FORM, utilizzando l'URI
  http://localhost:8080/saluto.html o
  http://localhost:8080/html/saluto.html
  a seconda di come si è organizzato il context ROOT.

- Per invocare direttamente la servlet usare l'URI seguente:
  http://localhost:8080/servlet/ServletHelloWWW2?nome=Pippo

NB. Se i parametri da passare fossero due (ad esempio nome e cognome) si scriverebbe:

http://localhost:8080/servlet/ServletHelloWWW2?nome=Pippo&cognome=Pluto

Riferimenti

- Marty Hall.
  “CORE. Servlets and JavaServer Pages”.
  Sun Microsystems Press.

- Phil Hanna.
  “JSP. La guida Completa.”
  McGraw-Hill.

- Dott. Roberto Posenato.
  Materiale del corso di Laboratorio di Basi di Dati e Web (A.A. 2001/2002).

- http://java.sun.com/products/servlets
- http://jakarta.apache.org/tomcat/index.html