SUPSI

Application server

Corso Applicazioni Web 2017-18
P. Ceppi 2014
L. Sommaruga Rev. 2017

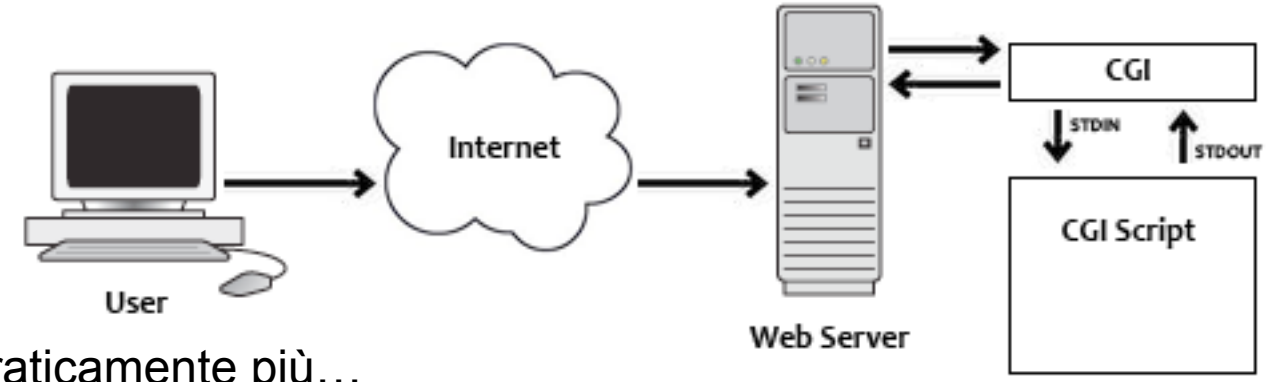
Da web server a application server

- Un web server prende una richiesta di una risorsa, trova la risorsa e la ritorna come risposta.
- Nel web server Apache il percorso nell'URL corrisponde al percorso nel file system del server. (http://www.supsi.ch/dti/isin/index.html → APACHE_HOME/htdocs/supsi/dti/isin/index.html)
- Alcune volte si ha bisogno di qualcosa in più di un web server, per servire pagine dinamiche. Serve quindi una «helper» application con la quale il server può comunicare per produrre pagine non statiche, create «al volo».

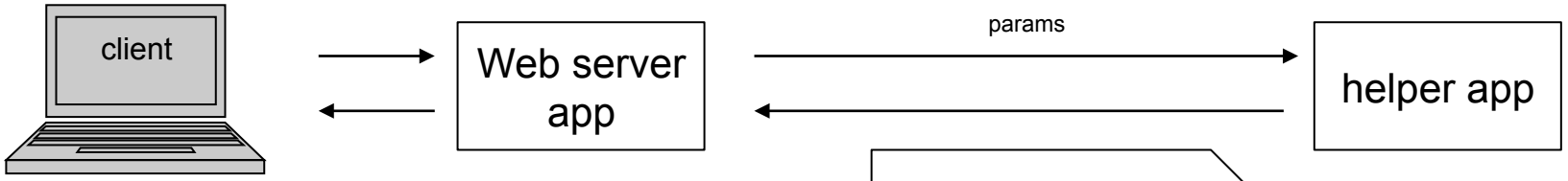
```
<html>
   <body>
   Current time is
   always 14:31
   </body>
</html>
```

```
<html>
   <body>
   Current time is
   [insertTimeOnServer]
   </body>
</html>
```

CGI (Common Gateway Interface)



- Ora non si usano praticamente più…
- Quasi tutti i programmi CGI sono scritti in Perl, ma altri lavorano con C, Python, PHP.
- Convenzione: avere una dir. `cgi-bin/` con files eseguibili sul server




```
<html>
   <body>
   Current time
   14:31
   </body>
</html>
```

Application server

- Un **application server** è una tipologia di server che fornisce **l'infrastruttura** e le **funzionalità di supporto**, sviluppo ed esecuzione di applicazioni nonché altri componenti server in un contesto distribuito.

- Fornisce tutta una serie di servizi come la possibilità di usare **modelli predefiniti** (*template*), un modello coerente per la **sicurezza**, la **persistenza** dei dati, le **sessioni** e altre funzionalità utili nella creazione di un'applicazione web.

- Le tecnologie su cui possono basarsi gli application server sono diverse: **Microsoft .NET** (Internet Information Services **IIS**), Java, …

- Java ha numerose implementazioni: **Apache Tomcat**, Jboss, WebLogic, WebSphere, …

- http://en.wikipedia.org/wiki/Comparison_of_application_servers

Apache Tomcat

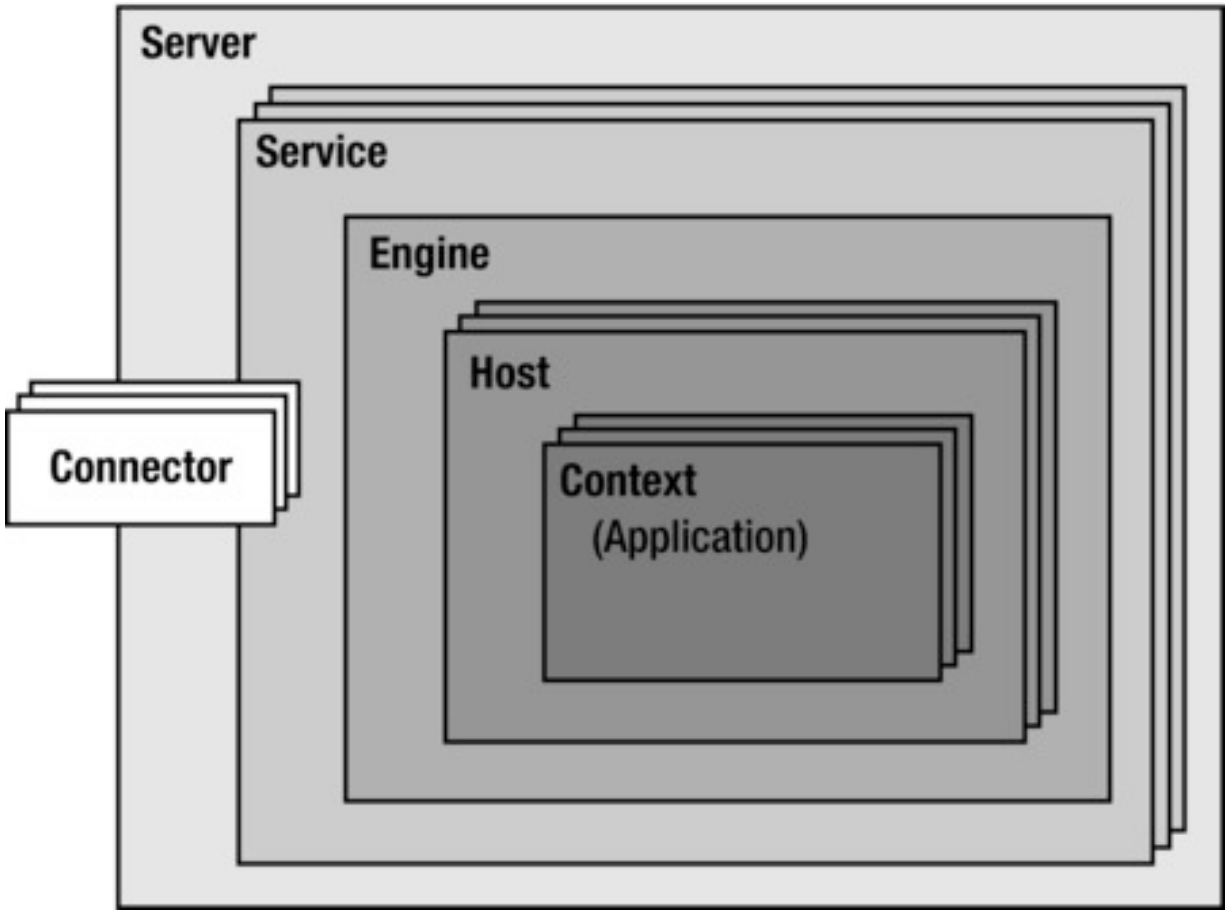
- È un **application server** nella forma di **contenitore servlet** open source sviluppato dalla Apache Software Foundation.

- Implementa le specifiche JavaServer Pages (JSP), Servlet, Expression Language e Web Socket, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

- La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache.

- Attualmente alla versione **9.0.12**
- http://tomcat.apache.org/

Architettura di Tomcat - overview dei componenti funzionali



Architettura di Tomcat - Componenti

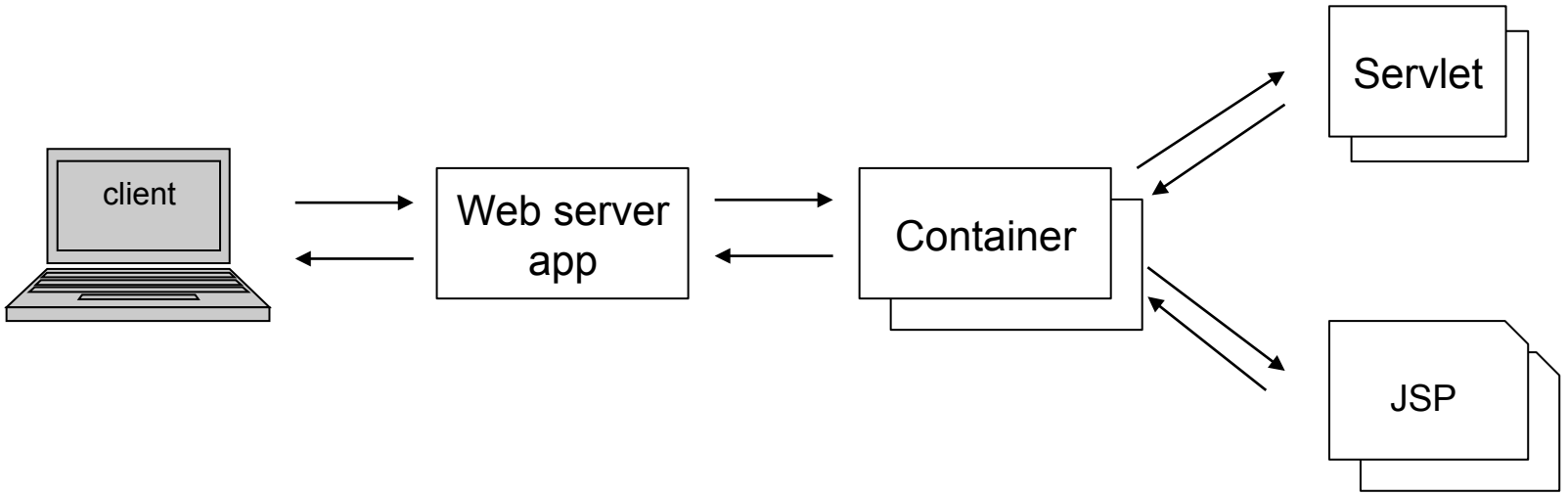
- **Context:** un Context **rapprensenta una singola web application**. Tomcat standard context + proprietà in /WEB-INF/web.xml
- **Connector: gestisce la comunicazione con il client**
- **Host:** un host è l'**associazione di un network name**, e.g. www.miodominio.com, **con il server Tomcat**
- **Engine:** rappresenta la **pipeline di processamento di richieste di un Service specifico**, e.g. mappa le richieste/risposte http sugli host/context
- **Service:** è un componente intermedio che vive in un server e **collega uno o più connettori a un Engine**
- **Server:** è il componente più esterno che **rappresenta un'instanza di Tomcat**
- **Listener:** un Java **object** che **risponde a specifici eventi**, e.g. JasperListener
- **Global Naming Resources:** definisce **risorse JNDI accessibili nel server**
- **Realm:** rappresenta un DB di **users, passwords, user roles per supportare container-based authentication**
- **Valve:** un elemento che **intercetta tutte le richieste HTTP in entrata** prima di raggiungere l'applicazione, come un filtro

Architettura di Tomcat - Server.xml

- La struttura di ogni server viene definita attraverso i suoi componenti funzionali nel file `server.xml`, situato in `/conf`

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
   <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
   <Listener className="org.apache.catalina.core.JasperListener" />
   <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
   <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
   <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
   <GlobalNamingResources>
     <Resource name="UserDatabase" auth="Container"
               type="org.apache.catalina.UserDatabase"
               description="User database that can be updated and saved"
               factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
               pathname="conf/tomcat-users.xml" />
   </GlobalNamingResources>
   <Service name="Catalina">
     <Connector port="8080" protocol="HTTP/1.1"
                connectionTimeout="20000"
                redirectPort="8443" />
     <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
     <Engine name="Catalina" defaultHost="localhost">
       <Realm className="org.apache.catalina.realm.LockOutRealm">
         <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                resourceName="UserDatabase"/>
       </Realm>
       <Host name="localhost"  appBase="webapps"
             unpackWARs="true" autoDeploy="true">
         <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                prefix="localhost_access_log." suffix=".txt"
                pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
       </Host>
     </Engine>
   </Service>
</Server>
```

Tomcat



Componenti di Tomcat

- **Catalina** è l'**engine contenitore di servlet** Java di Tomcat. Catalina implementa le specifiche di Sun Microsystems per le servlets Java e le JavaServer Pages (JSP).

- **Coyote** è il componente **connettore HTTP** di Tomcat. Supporta il protocollo HTTP 1.1 per il web server o per il contenitore di applicazioni. Coyote ascolta le connessioni in entrata su una specifica porta TCP sul server e inoltra la richiesta al Tomcat Engine per processare la richiesta e mandare indietro una risposta al client richiedente.

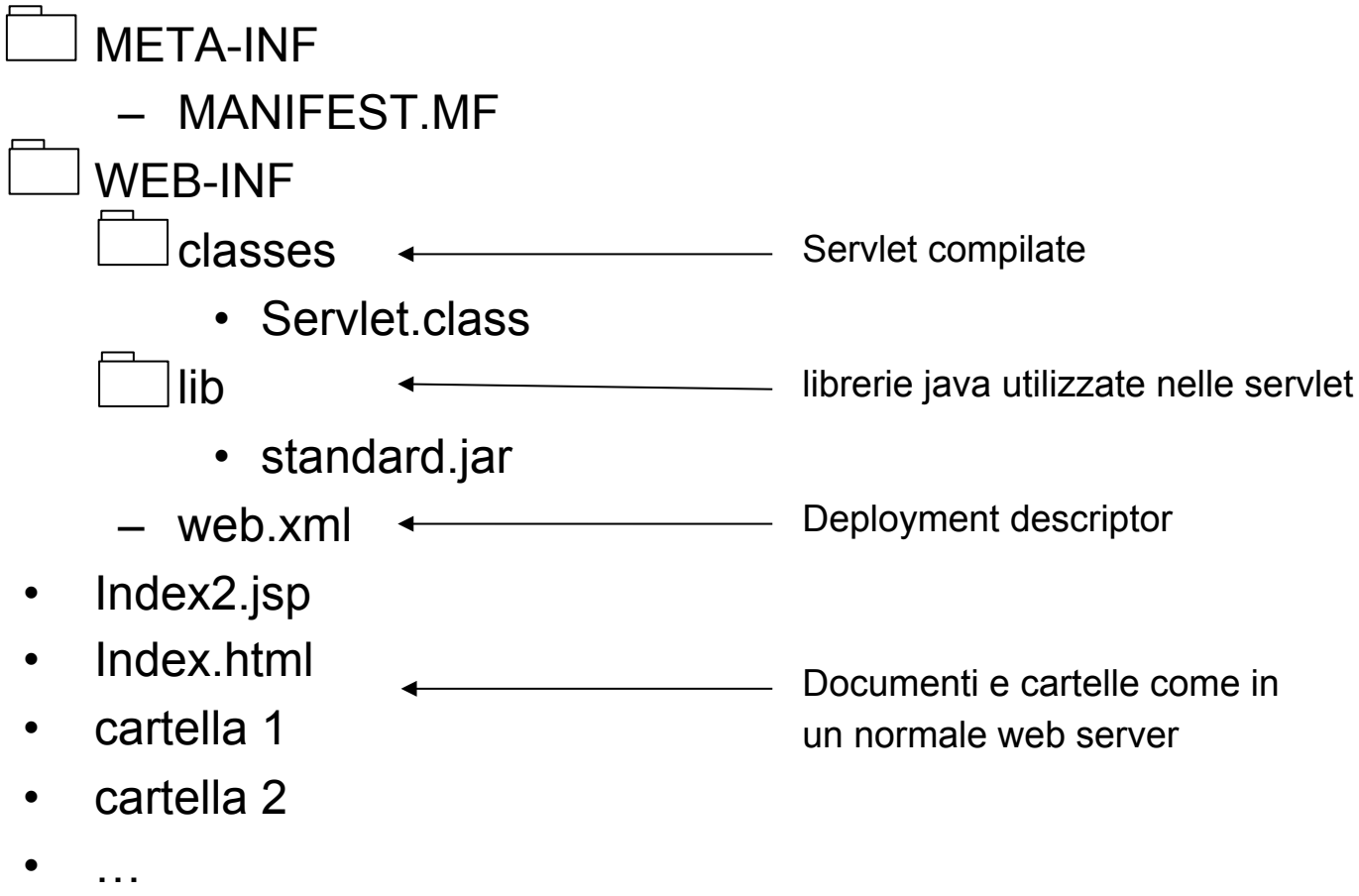
- **Jasper** è il motore JSP di Tomcat. Analizza i file JSP per compilarli in codice Java come servlets (che verranno poi gestite da Catalina). Al momento di essere lanciato, Jasper cerca eventuali cambiamenti avvenuti ai file JSP e, se necessario, li ricompila.

Servlet Container

- È il componente di un web server che interagisce con le servlets

- Offre:
  - **Communication support** fornisce un modo semplice per una servlet di comunicare con il web server
  - **Lifecycle management** il container controlla la vita e la morte della servlet
  - **Multithreading support** crea un nuovo thread per ogni richiesta
  - **Declarative security** tramite un file XML è possibile configurare la sicurezza
  - **JSP support**

Struttura generale di una application



Servlet

- Classi Java che **gestiscono richieste e risposte HTTP** per mezzo di appositi metodi
- Alla richiesta della risorsa corrispondente alla servlet, il container istanzia l'oggetto servlet e chiama un metodo specifico a seconda del tipo di messaggio ricevuto dal client (GET, POST, HEAD)
- Per creare una servlet bisogna estendere la classe `HttpServlet` del package `javax.servlet.http`, facendo overriding dei metodi predefiniti
- Le classi compilate devono essere nella cartella WEB-INF/classes oppure contenute in librerie .jar in WEB-INF/lib

Classe HttpServlet

- Una servlet può implementare uno dei seguenti metodi:
  - **doGet**, per rispondere alle richieste di tipo GET
  - **doPost**, per rispondere alle richieste di tipo POST
  - **init**, il metodo invocato una sola volta prima di ogni altro metodo
  - **destroy**, il metodo invocato prima di distruggere un oggetto di tipo Servlet
  - **getServletInfo**, il metodo usato per recuperare delle informazioni riguardo la servlet
  - **getLastModified**, il metodo ritorna il tempo dell'ultima modifica fatta

Esempio HttpServlet

```
public class HelloWorldServlet extends HttpServlet
{
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws
                                                                ServletException, IOException
   {
      String value = "Hello World";

      ServletOutputStream out = response.getOutputStream();
      out.println("<html>");
      out.println("<head><title>"+value+"</title></head>");
      out.println("<body bgcolor=\"#69f\">");
      out.println("<h1 align=\"center\">"+value+"</h1>");
      out.println("</body></html>");
      out.close();
   }
}
```

La risposta di questa servlet (per il metodo http GET) sarà del codice html che scrive al centro della pagina “Hello World”

Deployment descriptor: web.xml

- Un **deployment descriptor** è un file in formato **XML**, denominato **web.xml**, che viene utilizzato per informare il server, in cui è installata un'applicazione, riguardo alla **configurazione dell'applicazione** stessa.

- In esso possono essere definiti **parametri di inizializzazione**, **mappature dei percorsi** ed in generale, tutto ciò che contiene informazioni importanti alla corretta esecuzione dell'applicazione

- Ref.: https://docs.oracle.com/cd/E24329_01/web.1211/e21049/web_xml.htm

Modularizzazione di web.xml (Pluggability)

- Con servlet 2.5 c'era un unico file **web.xml** di configurazione
- Con servlet 3 web.xml è opzionale, si possono usare le annotation @
  (Vedi anche: https://community.oracle.com/docs/DOC-983211)
- Ogni .jar potrebbe avere un **web-fragment.xml** nella directory **META-INF** dove si definisicono: servlet, filtri, init, etc. Questo permette a librerie e frameworks di specificare le proprie servlet o altri oggetti
- Anche in modo dinamico con `ServletContextListener` (`@WebServletContextListener`) si possono aggiungere servlets, filtri e listeners usando i metodi di `ServletContext`: `addServlet()`, `addFilter()` e `addListener()`
- Servlet 4.0 in sviluppo per JEE8 https://javaee.github.io/servlet-spec/

Evoluzione futura

- Servlet 4.0 in sviluppo per JEE8
  https://javaee.github.io/servlet-spec/
- JSR 369: Java™ Servlet 4.0 Specification 5 sept 2017
  https://www.jcp.org/en/jsr/detail?id=369

- ```
  <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.0</version>
        <scope>provided</scope>
  </dependency>
  ```

- Supporto per HTTP/2
  https://http2.github.io/
  ```

web.xml: mapping percorsi

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://
xmlns.jcp.org/xml/ns/javaee  http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd" id="WebApp_ID"  version="3.1">
   <display-name>Nome applicazione</display-name>
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>ch.supsi.webapp.HelloWorldServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

API3 servlet

- Mappatura a livello programmatico della servlet con annotation da JEE>6

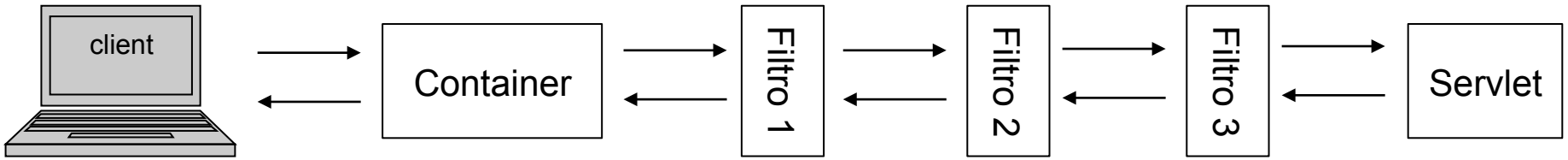
```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(value="/hello")

public class HelloServlet extends HttpServlet
{
   @Override protected void doGet(HttpServletRequest req,
                                  HttpServletResponse res) throws
                                  ServletException, IOException {
         res.getWriter().println("Hello world!!");
   }
}
```

Filters

- I filtri in Tomcat possono essere usati per intercettare e processare le richieste **prima** che siano inviate alle servlet, o per processare le risposte **dopo** che la servlet ha completato
- I filtri sono modulari e vengono configurati nel **web.xml** per le singole app
- Cosa fare con i filtri:
  - Effettuare controlli di sicurezza
  - Logging della richiesta
  - Comprimere la risposta
  - …



Filter - Esempio

```
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Ref.: https://tomcat.apache.org/tomcat-7.0-doc/config/filter.html

JSP

- Le **Java Server Pages** sono una tecnologia di programmazione Web in Java per lo sviluppo della **logica di presentazione** di applicazioni Web, fornendo contenuti dinamici in formato HTML

- Una pagina JSP è costituita da codice HTML con inserite sezioni di codice Java

- Si basa su un insieme di speciali **tag**, all'interno di una pagina HTML, con cui possono essere invocate funzioni predefinite sotto forma di codice Java (JSTL)

- permette di creare librerie di nuovi tag che estendono l'insieme dei tag standard

- Una pagina JSP viene compilata dall'engine Jasper come classi Servlet. A tomcat serve quindi una JDK per funzionare.

Esempio JSP

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page import="ch.supsi.webapp.session.User" %>
<!DOCTYPE html>
<html>
  <head>
    <title>Sample Application JSP Page</title>
  </head>
  <body>
    <h1><%= new String("Web application") %></h1>
    <p>
      <% User u = (User) session.getAttribute("user"); %>
      <c:choose>
        <c:when test="${user != null}">
        Bentornato <%= u.getName() %>
        </c:when>
        <c:otherwise>
        <a href="/register.jsp">Registrati</a>
        </c:otherwise>
      </c:choose>
    </p>
  </body>
</html>
```

JSP variabili implicite

- Gli elementi di codice in una pagina JSP possono fare riferimento a un insieme di variabili predefinite, dette variabili implicite:
  - **out** rappresenta il flusso di output su cui viene prodotta la pagina Web
  - **page** rappresenta il servlet generato dalla pagina JSP
  - **pageContext** un oggetto di classe PageContext, che contiene dati associati all'intera pagina
  - **request** l'oggetto di classe HttpServletRequest che rappresenta la richiesta HTTP che ha portato all'attivazione della pagina JSP/servlet
  - **response** l'oggetto di classe HttpServletResponse che rappresenta la risposta HTTP da inviare
  - **session** l'oggetto di classe HttpSession che rappresenta la sessione HTTP all'interno della quale è stata invocata la pagina JSP
  - **application** permette di accedere e di memorizzare gli oggetti per renderli accessibili da qualsiasi utente e modificabili da ogni pagina

Inoltrare la richiesta da servlet a una JSP

- Nelle JSP si possono mescolare logica di business con quella di presentazione, ma sarebbe meglio separarle il più possibile.
- Per separare meglio la logica di presentazione dei dati dalla logica di business, si può dapprima preparare i dati usando una servlet e poi inoltarli a una JSP
- Esempio:
  - Nella servlet aggiungere i dati come attributi all'oggetto request:

    ```
    request.setAttribute("date", new Date());
    ```

  - Dalla servlet chiamare la JSP:

    ```
    this.getServletContext().getRequestDispatcher("/index.jsp")
    .forward(req, resp);
    ```

  - Infine dalla pagina JSP:

    ```
    L'ora corrente è <%= request.getAttribute("date") %>
    ```

Deployment di una servlet

- WAR: **W**eb **Ar**chive. È un jar file ma con estensione **.war**
- Si tratta della struttura di una applicazione web raggruppata all'interno di una cartella superiore e compressa in formato zip
- In Tomcat il nome del .war diventa il nome della vostra applicazione e quindi sarà distribuita (deployed) in **/NOMEWAR**
- Per deployare un app è sufficiente mettere il file .war nella cartella *webapps* di Tomcat. Tomcat si occuperà di decomprimere il file e crea il contesto per la applicazione web
- È possible deployare un app anche da interfaccia web di Tomcat

Fonti

- Head First Servlets and JSP, Bryan Basham, 2nd edition, 2008
- An Introduction To Servlet 3.0 Blog https://community.oracle.com/docs/DOC-983211
- JSR 340: Java Servlet 3.1 Specification https://jcp.org/en/jsr/detail?id=340
- JSR 369: Java™ Servlet 4.0 Specification 5 sept 2017 https://www.jcp.org/en/jsr/detail?id=369
- Deployment descriptor https://docs.oracle.com/cd/E24329_01/web.1211/e21049/web_xml.htm
- http://www.whizkidtech.redprince.net/cgi-bin/tutorial
- http://en.wikipedia.org/wiki/Apache_Tomcat
- http://it.wikipedia.org/wiki/JavaServer_Pages
- http://www.oracle.com/technetwork/java/javaee/jsp/index.html
- http://tomcat.apache.org
- http://tomcat.apache.org/tomcat-7.0-doc/architecture/overview.html