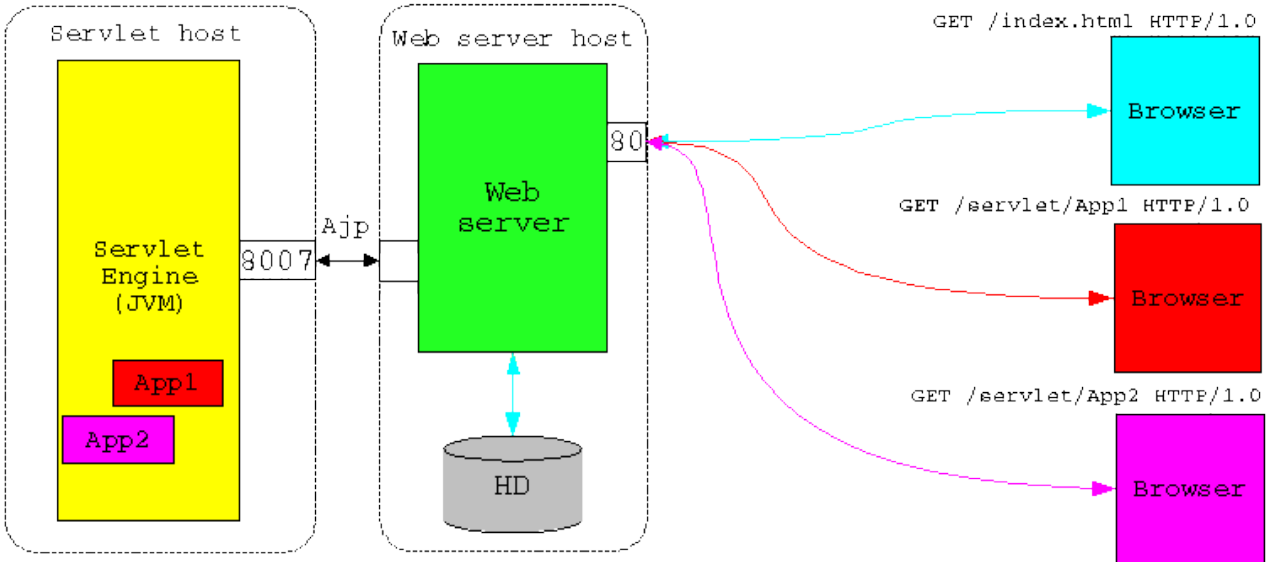
Lezione 6: Introduzione alle Serlvet

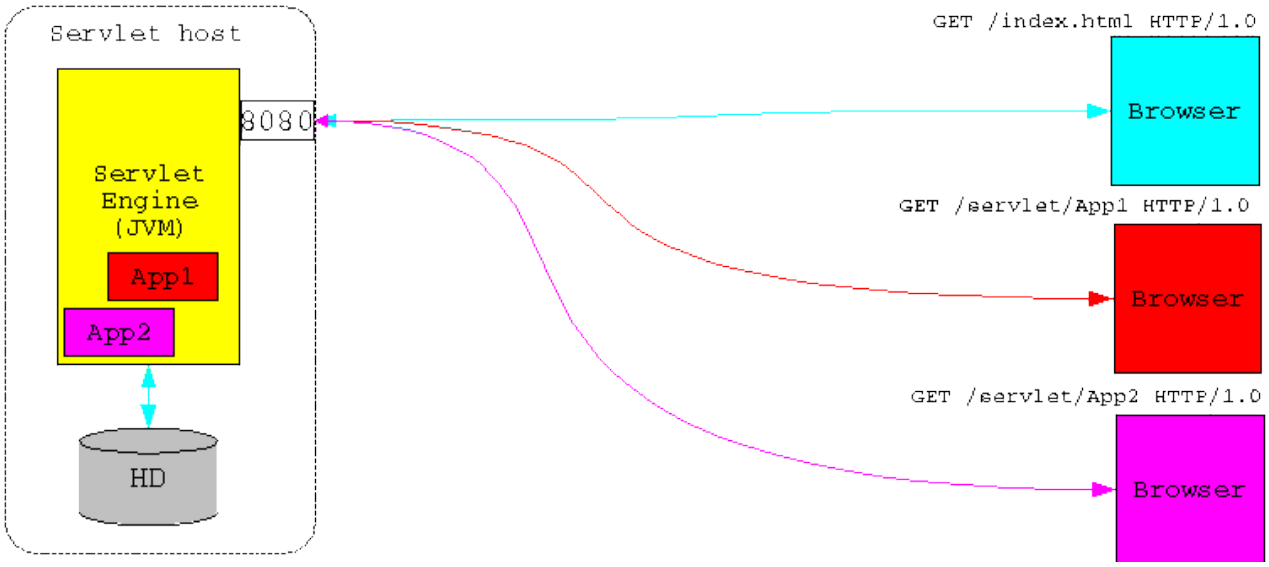
Le applicazioni CGI così come sono gestite dal web server richiedono la creazione di processo nuovo per ogni nuova richiesta. Quindi per ogni richiesta di una pagina dinamica del client c'è un'attesa iniziale dovuta al tempo richiesto dalla creazione del nuovo processo. Inoltre non c'è modo di tenere traccia di parametri di inizializzazione o delle scelte del client se non attraverso l'accesso al disco o a un database. Per questo motivo sono state introdotte le servlet, i cui vantaggi sono:

- **efficienza**: il codice di inizializzazione della servlet viene eseguito una sola volta la prima volta che il server la richiede; una volta caricata la servlet, ogni nuova richiesta si traduce (quasi) in una chiamata a un metodo Java che è molto più efficiente della creazione di un nuovo processo;
- **persistenza**: le servlet possono mantere in modo semplice e automatico delle informazioni tra due richieste successive dello stesso client e quindi non hanno bisogno di accedere al disco;
- **portabilità**: le servlet sono sviluppate in java;
- **affidabilità**: l'affidabilità è dovuta principalmente a:
    - la possibilità di usare JDK molto evolute;
    - l'uso delle eccezioni per la manipolazione degli errori;
    - gestione della memoria dinamica;
    - molte librerie di supporto.
- **estensibilità**: linguaggio orientato agli oggetti;
- **sicurezza**: beneficiano della sicurezza fornita dal web-server;
- **standard**: la comunicazione tra browser e web-server avviene esattamente come nel caso di applicazioni CGI;
- **scalabilità**: la macchina su cui gira la servlet e la macchna su cui gira il web server possono essere diverse;
- **compatibilità**: la maggior parte dei web-server supporta le servlet.

Le servlet sono scritte in java e quindi c'è bisogno di far girare le servlent all'interno di una JVM. Inoltre, ci sono altre funzionalità di comunicazione con il web server o di simulazione del web server che richiedono di far girare le servlet in un'applicazione speciale detta **servlet engine** . Nei nostri esempi useremo come servlet engine **Tomcat**.
Nello schema seguente vediamo la configurazione standard dei server di "produzione":



Nello schema seguente vediamo la configurazione alternativa di un servlet engine più adatta per sviluppare e testare le applicazioni e quella che useremo nei seguenti esempi:



Vantaggi della configurazione singola sono:

- semplice installazione e configurazione;
- ideale per lo sviluppo.

Mentre gli svantaggi sono:

- servlet engine non è cosi veloce come un web-server per le pagine statiche;
- non è molto configurabile;
- non è molto robusto;
- non gestisce i CGI, Server API/ perl/php.

Per aumentare ulteriormente le prestazioni il servlet engine può imlementare una politica di **thread pool**:

- vengono creati un numero iniziale minimo di thread che vengono messi in attesa di richieste;
- all'arrivo di una nuova richiesta, se esiste un thread libero la richiesta viene assegnata a quel thread;
- altrimenti se il numero di thread creati non supera il numero massimo viene creato un nuovo thread a cui viene assegnata la richiesta;
- se non ci sono thread liberi e il numero di thread creati supera il numero massimo, la richiesta viene messa in attesa che un thread si liberi;
- nel caso in cui ci siano molti thread non usati, alcuni di essi vengono chiusi per recuperare risorse di sistema.

Quindi il motore delle servlet è caratterizzato e parametrizzato dai:

- preallocazione e riuso dei thread;
- massimo numero di thread (`max_threads` nel file `conf/server.xml` , 50 default);
- limite minimo `min_spare_threads` (10 default);
- limite massimo di thread non usate `max_spare_threads` (25 default).

HelloWorldServlet

Vediamo come ottenere la nostra prima servlet.

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

```
public class HelloWorldServlet extends HttpServlet
{
   public void doGet(HttpServletRequest request, HttpServletResponse
response)
      throws ServletException, IOException
   {

      //preparo il tipo di risposta
      response.setContentType("text/html");
      PrintWriter out= response.getWriter();
      out.println("<HTML><BODY BGCOLOR=\"#7F7FFF\"><H1
ALIGN=CENTER>HELLO WORLD</H1></BODY></HTML>");
      out.close();
   }

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

Per provare a scrivere, compilare ed eseguire la prima servlet, la cosa migliore è servirsi di una apposita IDE come Netbeans o Eclipse con installata il pacchetto per la gestione della applicazioni Web. Quindi creare un nuova nuova applicazione Web e successivamente una nuova Servlet con il nome HelloWorldServlet. Cambiamo il codice della servlet generata con il codice appena visto e dal menu lanciamo il comando per eseguire la servlet. In automatico verrà aperto il browser predefinito al link opportuno che visualizzerà la pagina dinamica creata dalla nostra servlet.

Servlet

Informazioni dettagliate sulle servlet si possono essere trovare in:
http://java.sun.com/products/servlet/.
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.

Usando un ambiente di sviluppo integrato (es netbeans, eclipse etc.) e creando una nuova servlet ci viene proposto un codice simile al seguente:

```
/*
 * prova.java
 *
 * Created on April 24, 2002, 9:08 AM
*/
import javax.servlet.*;
import javax.servlet.http.*;
/**
 *
 * @author  roncato
```

```
 * @version
 */
public class prova extends HttpServlet {

   /** Initializes the servlet.
    */
   public void init(ServletConfig config) throws ServletException {
      super.init(config);

   }

  /** Destroys the servlet.
    */
   public void destroy() {

   }

  /** Processes requests for both HTTP GET and POST methods.
    * @param request servlet request
    * @param response servlet response
    */
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
          throws ServletException, java.io.IOException {
       response.setContentType("text/html");
       java.io.PrintWriter out = response.getWriter();
       /* output your page here
       out.println("");
       out.println("");

       */
       out.close();
       }

     /** Handles the HTTP GET method.
       * @param request servlet request
       * @param response servlet response
      */
     protected void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, java.io.IOException {
        processRequest(request, response);
       }

     /** Handles the HTTP POST method.
       * @param request servlet request
       * @param response servlet response
       */
     protected void doPost(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, java.io.IOException {
        processRequest(request, response);
     }

    /** Returns a short description of the servlet.
      */
      public String getServletInfo() {
        return "Short description";
      }

 }
```

HttpRequest

La classe HttpRequest contiene tutte le informazioni riguardo la richiesta inviata dal client e fornisce dei metodi per recuperare sia le informazioni presenti nello header della richiesta che nel corpo.

- `public BufferedReader getReader() throws IOException;`

Metodi della classe HttpServletRequest per recuperare informazioni riguardo l'header:

`HttpResponse`

- `public PrintWriter getWriter() throws IOException;`
- `public void setContentType(String type);`
- `public void setContentLength(int len);`

Recuperare i parametri da un form

Vediamo ora come sia possibile recuperare i parametri da un form. Nella libreria delle servlet ci sono tre metodi per recuperare i parametri da un oggetto di tipo `ServletRequest`:

- `public Enumeration ServletRequest.getParameterNames();` ritorna tutti i nomi parametri della richiesta in una enumerazione. Nel caso non si conoscano i nomi dei parametri presenti nella richiesta questo metodo può essere usato per trovare i nomi dei parametri;
- `public String ServletRequest.getParameter(String name);` ritorna una stringa che contiene il valore del parmetro specificato, `null` se il parametro non è presente nella richiesta. Questo metodo andrebbe usato solo quando il parametro contiene un solo valore, altrimenti è preferibile usare il prossimo metodo;
- `public String[] ServletRequest.getParameterValues(String name);` ritorna un vettore con tutti i valori del parametri specificato.

Nel seguente esempio vengono elencate tutte le coppie nome valore presenti nella richiesta utilizzando i metodi appena visti:

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

public class ShowParametersServlet extends HttpServlet
{
  public void init(ServletConfig config)
    throws ServletException
  {
    // Always pass the ServletConfig object to the super class
    super.init(config);
  }
  //Process the HTTP Get request
  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Show Parameters</title></head>");
    out.println("<body>");
    // Get all the parameter names
    Enumeration parameters = request.getParameterNames();
    String param = null;
    // Iterate over the names, getting the parameters
    while ( parameters.hasMoreElements() )
    {
```

```
      param = (String)parameters.nextElement();
      out.println("<BOLD>" + param +
        " : " + request.getParameter(param) +
        "</BOLD><BR>");
    }
    out.println("</body></html>");
    out.close();
  }
  //Process the HTTP Post request
  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException
  {
    doGet(request, response);
  }
//Get Servlet information

  public String getServletInfo() {
    return "Show Parameters Information";
  }
}
```

Notiamo che per il programmatore non c'è differenza nella gestione di un metodo POST e un metodo GET. Mentre se avessimo dovuto scrivere la stessa cosa con CGI in C avremmo dovuto distinguere tra il metodo POST che codifica nello standard input e il metodo GET che passa i parametri nell'ambiente.

Esercizio 1: tramite Netbeans o Eclipse creare una WebApplication con una Servlet e una pagina Html. La servlet deve ritornare una pagina dinamica che visualizza tutti i parametri contenuti nella richiesta del client. La pagina html deve avere un form la cui sottomissione comporti l'esecuzione della Servlet appena descritta e che contenga degli input e un pulsante.

Esercizio 2: modificate la pagina html di sopra in modo che il form contenga due input con lo stesso nome;

Esercizio 3: modificate la pagina htmo in modo che venga invocato il metodo doPost della Servlet.