Java Server-Side: Servlet

Ing. Cesare Monti - 20 aprile 2005

cosa vedremo

- Java servlet:
  - storia
  - gerarchia di classi
  - ciclo di vita
  - modelli di gestione dei thread
  - organizzazione del contesto
  - esempi
- jdbc

Java Server Side a bite of history

- fine anni '90 - Sun Microsystem
  - oberati da richieste di comunitò di sviluppatori web che chiedevano:
    - di poter portare le proprie stand-alone application sul web
    - di avere mezzi più agili delle CGI per avere web dinamico
- per tutta risposta la Sun rilasciò:
  - 1997 - Servlet API
  - 1998 - JSP API
  - entrambe contenute all'interno del pacchetto J2EE

Java Server Side

- Java Servlet API
  - si occupano di definire gli oggetti che rendono possibile modellare in java l'interazione sul web
  - obiettivo delle API è quello di
    - estendere le funzionalità dei server web
    - fornire l'insieme funzionale di oggetti per manipolare il mondo web
    - riutilizzare tutti gli applicativi già realizzati in J2SE
    - riuscire a fornire strumenti agili per scrivere elementi web
    - estendere la dinamicità introdotta con le CGI

Java Server Side

- l'idea di base era quella di creare un servlet engine che si potesse innestare in un server web
  - ... col tempo sono stati creati server web veri e propri
- creare un ambiente che a run-time potesse caricare e scaricare bundle di codice che producessero dinamicamente pagine html a fronte di ogni richiesta

Java Server Side

- con la loro introduzione si poterono superare i limiti delle vecchie tecnologie:
  - prestazioni
    - ... le CGI ormai non bastavano più
  - semplicità
    - ... java allora veniva eseguito nei plug-in dei browser e non c'erano garanzie sul funzionamento
- e introdurre nuovi modelli applicativi
  - grazie al concetto di "sessione"
    - ... ne parliamo poi
  - grazie all'interazione con Java
    - ... ne parliamo poi

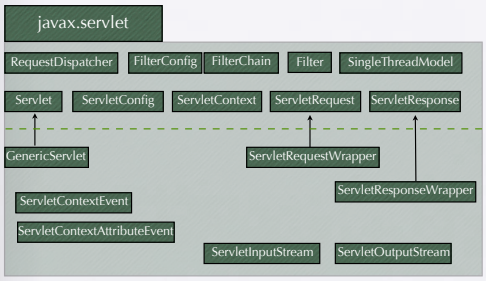
Java Server Side

- Java JSP
  - pazientate ...
    - ... ne parleremo mercoledì prossimo

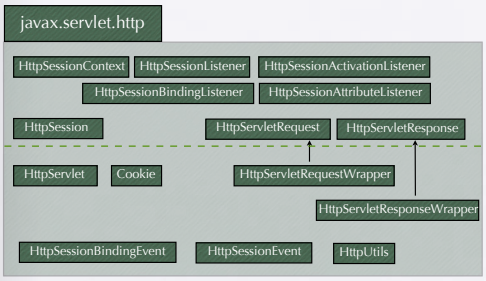
gerarchia delle classi

- fisicamente sono suddivise in 2 package:
  - javax.servlet
    - servlet indipendenti dal protocollo usato
  - javax.servlet.http
    - servlet basate su http

gerarchia delle classi



gerarchia delle classi



gerarchia delle classi

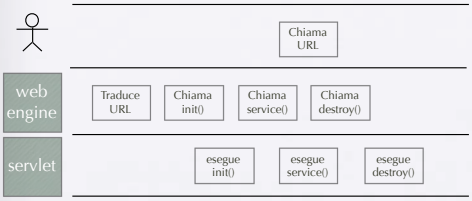
- fondamentalmente per creare una servlet basta estendere
  - javax.servlet.http.HttpServlet
    - ...che di suo estende già GenericServlet
- l'engine all'invocazione della servlet gli passerà i riferimenti a due oggetti:
  - HttpServletRequest
    - contiene tutti gli oggetti della request http fatta dall'utente
  - HttpServletResponse
    - ... è ciò che verrà spedito all'utente (la response http)

il ciclo di vita

- ogni servlet viene eseguita da un engine
  - per poter essere messa in esecuzione i servlet mettono a disposizione dei metodi
  - questi vengono chiamati dall'engine secondo un schema predefinito
  - tali metodi sono quelli dell'interfaccia di javax.servlet.Servlet:
    - init()
      - chiamata subito dopo averla caricata in memoria
    - service()
      - chiamata al momento della messa in esecuzione del servlet
    - destroy()
      - chiamata prima di eliminarla dalla memoria

il ciclo di vita

- il ciclo di vita della servlet viene deciso dall'engine ed ha il seguente flusso



il ciclo di vita

- init()
  - consente di controllare se la servlet sia già stata caricata
    - in caso NON sia già in memoria
      - la carica
      - chiama un metodo interno di inizializzazione
    - in casi sia già in memoria
      - chiama un metodo interno di inizializzazione
  - tale metodo ha il seguente prototipo:
    - public void init(ServletConfig config) throws ServletException
    - serve per permettere al servlet che sta per entrare in esecuzione di accedere al contesto presente sull'engine
  - dopodiché si possono eseguire tutte le operazioni utente che possono servire per la corretta esecuzione del servlet
    - ad esempio aprire connessioni con i DB

il ciclo di vita

- service()
  - ogni richiesta viene instradata verso questo metodo
  - ogni richiesta viene servita in un thread differente
    - qui dentro vengono instradate sia le richieste giunte da accessi con Method POST sia con Method GET
      - ... poi vediamo come
  - ... qui si scrive il codice che va a servire la richiesta
    - ... magari le interrogazioni al DB di cui parlavamo prima

il ciclo di vita

- public void destroy()
  - prepara il tutto allo scaricamento del servlet dalla memoria
  - ... ad esempio chiudere la connessione con il db che abbiamo utilizzato

modello di gestione dei threads

- le release delle Servlet API sono mutate nel tempo
  - ... ora siamo alla 2.4
  - un grande cambiamento è stato introdotto introducendo il modello per la gestione dei threads

modello di gestione dei threads

- il modello "ordinario" dice che ogni servlet viene caricata come unica istanza
  - il che implica diversi rischi
    - ogni thread passa per lo stesso codice
      - vede le stesse variabili d'ambiente
      - ... potrebbe cambiarle avendo side-effect
- per tale motivo è stata introdotta un'interfaccia chiamata SingleThreadModel
  - è un interfaccia senza metodi
    - tutti la possono implementare
  - indica al Servlet Engine che ad ogni chiamata deve corrispondere una nuova istanza su cui gira un singolo thread

modello di gestione dei threads

- ATTENZIONE!
  - nonostante il modello "affascini" ricordiamoci che siamo sul web
    - un ambiente dove i numeri possono scalare rapidamente
    - il che richiederebbe una macchina molto performante e che per numeri grandi sarebbe stressata in poco tempo
  - e soprattutto ... il modello SingleThreadModel fa la stessa cosa che facevano le CGI
    - ... motivo che ne ha decretato l'insuccesso

modello di gestione dei threads

- Q: quindi quando lo usiamo?
  - pensate a una operazione critica che richiesta tempo di esecuzione
    - ... pensate a un upload di un file sul file system
  - i passi sono:
    - invocare la servlet
    - agganciare lo stream di dati in upload
    - scriverlo su di un file nel file system
  - ... e in caso di upload concorrenti ??
    - nel modello standard scattano problemi
      - in quanto potremmo rischiare di scrivere dati di stream diversi sullo stesso file
        - ... usano lo stesso codice
    - nel modello SingleThreads manterremmo i flussi di dati separati
    - pagando lo scotto di rallentare il sistema!

organizzazione del contesto

- Q: ok, ora so come scrivere una servlet ... mi basta per realizzare un sistema complesso?
  - riformulo
  - Q: come interagiscono le servlet tra di loro?
  - R: mediate dall'engine

organizzazione del contesto

- la realizzazione dell'engine in java ha permesso di estendere il modello per l'interazione delle servlet
- in particolare ha aggiunto tre spazi di scope condivisi:
  - page
  - session
  - application

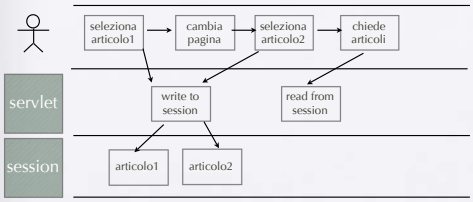
contesto

- page
  - è lo scope predefinito di tutte gli oggetti/variabili
    - quando ne creiamo uno ... questo è qui
  - coincide con la pagina che stiamo generando
    - ergo è un oggetto sempre disponibile
  - ogni variabile viene istanziata e deallocata durante una response
  - ci permette di realizzare pagine parametriche

contesto

- session
  - l'introduzione di engine java ha permesso di avere una spazio di scope associato ad ogni utente connesso
  - per ogni user-agent che si connette al server web può essere istanziato un'oggetto session (uno solo) su cui possiamo manipolare variabili e oggetti generati dall'utente
  - ci permette di mantenere lo stato dell'interazione
  - ha una durata definita allo start up e volendo modificabile run-time
  - ... tipico esempio ne è il carrello della spesa ...

contesto



contesto

- application
  - nel caso in cui oggetti utente debbano interagire tra di loro abbiamo questo livello
  - in realtà tutti i processi Java che girano sulla macchina virtuale vedono questi oggetti
    - è lo spazio di memoria della stessa virtual machine
  - questo ci permette livelli di interazione con oggetti legacy e con la logica business che ci serve

contesto

- oltre agli spazi di scope il contesto ci mette a disposizione una serie di costanti
  - per comodità il descrittore della configurazione del contesto che contiene tali costanti viene scritto fisicamente in un file
    - web.xml
      - tale file deve poter essere letto dall'engine alla partenza
- e una configurazione dell'ambiente
  - anch'esso espresso in un file
    - context.xml
      - anch'esso deve essere letto alla partenza
- il che implica che l'ambiente dell'engine sia struttura secondo regole

contesto / ontologia

- l'ambiente è così organizzato:
  - ogni web application ...
    - l'insieme delle servlet e delle pagine di elaborazione/presentazione che compongono l'applicazione
  - ... ha un suo file descrittore della configurazione (web.xml) posto all'interno di una cartella chiamata WEB-INF
  - ... ha un suo file descrittore del contesto (context.xml) posto all'interno di una cartella chiamata META-INF
  - mantiene le servlet dentro ad una cartella "classes" dentro a WEB-INF
  - mantiene i jar (quindi qualsiasi applicazione java stand-alone) dentro ad una cartella "lib" dentro WEB-INF

contesto / ontologia

- WEB-INF
- è strutturato secondo un XML schema

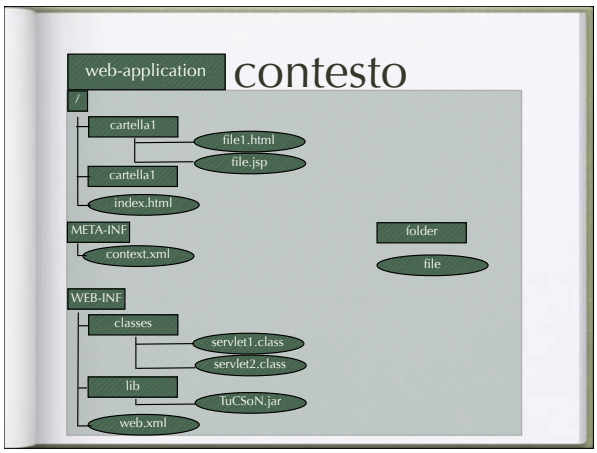
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>Prima web application</display-name>
<context-param>
    <param-name>NOME_PARAMETRO</param-name>
    <param-value>VALORE_PARAMETRO</param-value>
    <description>DESCRIZIONE DEL PARAMETRO</description>
  </context-param>
  <servlet>
    <servlet-name>pdfDownload</servlet-name>
    <servlet-class>webl.til.pdfDownload</servlet-class>
  </servlet>
 <servlet-mapping>
    <servlet-name>pdfDownload</servlet-name>
    <url-pattern>/pdfDownload</url-pattern>
  </servlet-mapping>
  <taglib>
    <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
    <taglib-location>/WEB-INF/tlds/c.tld</taglib-location>
  </taglib>
 <error-page>
    <error-code>404</error-code>
    <location>/errors/error-404.jsp</location>
  </error-page>
</web-app>
```

contesto / ontologia

- context.xml
  - è strutturato secondo un schema xml
  - normalmente se ne preoccupa l'engine di configurarlo e gestirlo
    - salvo particolari necessità utente è bene lasciare che sia l'engine a maneggiarlo

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/AIXIA">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="AIXIA." suffix=".log"
timestamp="true"/>
</Context>
```

  - nell'esempio ho detto al contesto di utilizzare codesta classe per effettuare il log



Esempio primaServlet

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

public class primaServlet extends HttpServlet {

   public void init(ServletConfig config) throws ServletException {
      super.init(config)
   }

   public void destroy() {
   }

protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>Hello World</h2>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
```

Esempio primaServlet

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


    public String getServletInfo() {
        return "Scrivo Hello World";
    }

}
```

esempio html - tre parametri

```
<html><head><title>prova tre parametri</title></head>
<body>
<form action="/recuperoParametri" method="post/get">
<input type="text" name="param1">
<inpu type="hidden" name="param2" value="0">
<input type="checkbox" name="param3">
<input type="submit" value="submit">
</form>
</body></html>
```

Esempio recuperoParametri

```
...
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet</title>");
        out.println("</head>");
        out.println("<body>");

        out.println("<ul>");

        out.println("<li>"+request.getParameter("param1"));

        out.println("<li>"+request.getParameter("param2"));

        out.println("<li>"+request.getParameter("param3"));

         out.println("</ul>");
         out.println("</body>");
         out.println("</html>");
         out.close();
          }
---
```

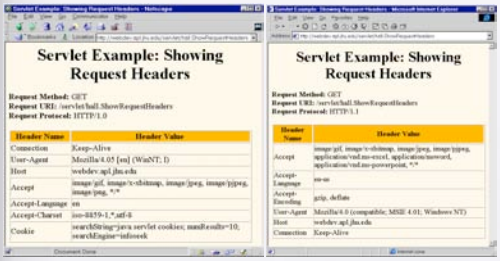
esempio recuperoTutto

```
...
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{ response.setContentType("text/html"); PrintWriter out = response.getWriter();
        out.println("<html>");out.println("<head>");out.println("<title>Servlet</title>"); out.println("</head>");
        out.println("<body>");
Enumeration paramNames = request.getParameterNames();
  while(paramNames.hasMoreElements()) {
    String paramName = (String)paramNames.nextElement();
    out.println("<br>" + paramName + " ");
    String[] paramValues = request.getParameterValues(paramName);
    if (paramValues.length == 1) {
      String paramValue = paramValues[0];
      if (paramValue.length() == 0)
        out.print("<i>No Value</i>");
      else
        out.print(paramValue);
    } else {
      out.println("<UL>");
      for(int i=0; i<paramValues.length; i++) {
       out.println("<LI>" + paramValues[i]);
      }
      out.println("</UL>"); }}
out.println("</body>");
out.println("</html>");
out.close();
}
```

headers

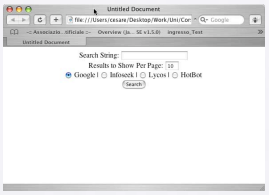
```
...
 protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException { response.setContentType("text/html"); PrintWriter out =
 response.getWriter();
        out.println("<html>");out.println("<head>");out.println("<title>Servlet</title>"); out.println("</head>");
        out.println("<body>");
Enumeration headerNames = request.getHeaderNames();
   while(headerNames.hasMoreElements()) {
     String headerName = (String)headerNames.nextElement();
     out.println("<TR><TD>" + headerName);
     out.println("    <TD>" + request.getHeader(headerName));
   }
out.println("</body>");
out.println("</html>");
out.close();
 }
...
```

headers



redirect e gestione errori

```
<FORM ACTION="/SearchEngines">
  <CENTER>
    Search String:
    <INPUT TYPE="TEXT" NAME="searchString"><BR>
    Results to Show Per Page:
    <INPUT TYPE="TEXT" NAME="numResults"
                   VALUE=10 SIZE=3><BR>
    <INPUT TYPE="RADIO" NAME="searchEngine"
                   VALUE="google">
    Google |
    <INPUT TYPE="RADIO" NAME="searchEngine"
                   VALUE="infoseek">
    Infoseek |
    <INPUT TYPE="RADIO" NAME="searchEngine"
                   VALUE="lycos">
    Lycos |
    <INPUT TYPE="RADIO" NAME="searchEngine"
                   VALUE="hotbot">
    HotBot
    <BR>
    <INPUT TYPE="SUBMIT" VALUE="Search">
  </CENTER>
</FORM>
```



SearchEngines

```
public class SearchSpec {
  private String name, baseURL, numResultsSuffix;

  private static SearchSpec[] commonSpecs =
    { new SearchSpec("google",
                     "http://www.google.com/search?q=",
                     "&num="),
      new SearchSpec("infoseek",
                     "http://infoseek.go.com/Titles?qt=",
                     "&nh="),
      new SearchSpec("lycos",
                     "http://lycospro.lycos.com/cgi-bin/pursuit?query=",
                     "&maxhits="),
      new SearchSpec("hotbot",
                     "http://www.hotbot.com/?MT=",
                     "&DC=")
    };

  public SearchSpec(String name,
                    String baseURL,
                    String numResultsSuffix) {
    this.name = name;
    this.baseURL = baseURL;
    this.numResultsSuffix = numResultsSuffix;
  }

  public String makeURL(String searchString, String numResults) {
    return(baseURL + searchString + numResultsSuffix + numResults);
  }

  public String getName() {
    return(name);
  }

  public static SearchSpec[] getCommonSpecs() {
    return(commonSpecs);
  }
}
```

SearchEngines

```
...
String searchString = URLEncoder.encode(request.getParameter("searchString"));
    String numResults = request.getParameter("numResults");
    String searchEngine = request.getParameter("searchEngine");
    SearchSpec[] commonSpecs = SearchSpec.getCommonSpecs();
    for(int i=0; i<commonSpecs.length; i++) {
      SearchSpec searchSpec = commonSpecs[i];
      if (searchSpec.getName().equals(searchEngine)) {
String url =  response.encodeURL(searchSpec.makeURL(searchString, numResults));
        response.sendRedirect(url);
        return;
      }
    }
    response.sendError(404);
//oppure meglio response.sendError(response.SC_NOT_FOUND,  "non ho capito che engine usare");
  }
...
```

sessioni

```
<table>
<tr><td><a href="/WriteObj2Session?artId=1">aggiungi oggetto 1</a></td></tr>
 <tr><td><a href="/WriteObj2Session?artId=2">aggiungi oggetto 2</a></td></tr>
<tr><td><a href="/WriteObj2Session?artId=3">aggiungi oggetto 3</a></td></tr>
</table>
```

sessioni

```
...
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException { response.setContentType("text/html"); PrintWriter out =
response.getWriter();
        out.println("<html>");out.println("<head>");out.println("<title>Servlet</title>"); out.println("</head>");
        out.println("<body>");
String id = request.getParameter("artId");

String nome="id"&Integer.parseInt(id);
request.getSession().setAttribute("nome",id);
out.println("<b>"&"Articolo aggiunto"&"</b>");

out.println("</body>");
out.println("</html>");
out.close();
 }
...
```

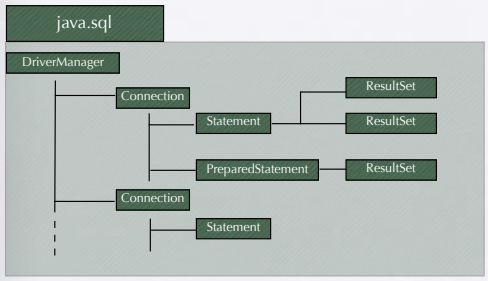
sessioni

```
...
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException { response.setContentType("text/html"); PrintWriter out =
response.getWriter();
        out.println("<html>");out.println("<head>");out.println("<title>Servlet</title>"); out.println("</head>");
        out.println("<body>");
Enumeration e = request.getSession().getAttributeNames();
while (e.hasMoreElements() ){
String x = request.getSession().getAttribute(e.next().toString());
out.println("nel carrello hai"+ x + "<br>");
 }
out.println("</body>");
out.println("</html>");
out.close();
 }
...
```

JDBC

- Java Database Connectivity è il package SUN che si occupa di gestire l'accesso alle basi di dati
  - il package contiene:
    - interfacce
    - classi astratte
  - Q: Perché ?
    - perché l'implementazione di tali classi dipende dal data base relazionale che stiamo utilizzando
  - Q: quindi quante ne esistono ?
    - almeno una per ogni versione di database

JDBC



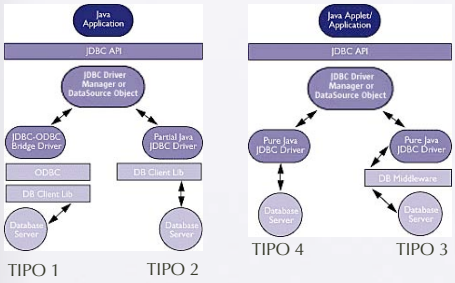
ODBC

- è uno standard per l'accesso ai db relazionali
  - che non appartiene al mondo java
  - è stato creato per il mondo windows
    - può incapsulare il DBMS per gestire le connessioni ai DB

DRIVER di accesso ai DB

- in questo ambito si definisce DRIVER l'insieme di connessioni basate su standard diversi che permettono di gestire l'accesso ai DB

DB
tipi di driver



TIPO 1 TIPO 2 TIPO 4 TIPO 3

JDBC

- le API JDBC permettono la creazione di 4 tipi di "driver"
  - Tipo 1: JDBC + ODBC Bridge + Driver OBDC + proprietary DBMS access protocol
  - Tipo 2: JDBC + proprietary DBMS access protocol
  - Tipo 3: Driver Client + Driver Server + DBMS access protocol
  - Tipo 4: JDBC only
- in blu quelli "stand alone oriented"
- in rosso quelli "Network Oriented"
- solitamente ...nel distribuito ... si usano quelli di tipo 4

JDBC

- quindi?
  - per connettersi a un qualsiasi db tramite java il modo più rapido è utilizzare i driver di tipo 4
- come ?
  - basta scaricare le classi JDBC relative al db che abbiamo installato
    - ... probabilmente sul sito dei creatori del db

esempio in servlet: web.xml

- ...

```
<context-param>
  <param-name>DB_URL</param-name>
  <param-value>jdbc:mysql://127.0.0.1/aixia</param-value>
  <description>Url del db</description>
 </context-param>
 <context-param>
  <param-name>DB_DRIVER</param-name>
  <param-value>org.gjt.mm.mysql.Driver</param-value>
  <description>Driver del db</description>
 </context-param>
 <context-param>
  <param-name>DB_USER</param-name>
  <param-value>utente</param-value>
  <description>Utente del db</description>
 </context-param>
 <context-param>
  <param-name>DB_PASSWORD</param-name>
  <param-value>password</param-value>
  <description>Password del db</description>
 </context-param>
 <context-param>
<context-param>
  <param-name>DB</param-name>
  <param-value>nomeDelDatabase</param-value>
  <description>Database</description>
 </context-param>
```

JDBC

```
String URL   = config.getServletContext().getInitParameter("DB_URL");
String database = config.getServletContext().getInitParameter("DB");
String myusername = config.getServletContext().getInitParameter("DB_USER");
String mypassword = config.getServletContext().getInitParameter("DB_PASSWORD");
String driver = config.getServletContext().getInitParameter("DB_DRIVER");
String nome="";
String password="";

Class.forName("org.gjt.mm.mysql.Driver").newInstance();
conn = java.sql.DriverManager.getConnection(URL+database, myusername, mypassword);

      conn = java.sql.DriverManager.getConnection(URL+database, myusername, mypassword);
          conn.setAutoCommit(false);
          ps = conn.prepareStatement("select * from utenti where username=?");
          ps.setString(1, request.getParameter("username"));
          rs =ps.executeQuery();
              while (rs.next()){          id=rs.getString("id");          nome= rs.getString("nome")       password =
    rs.getString("password");
           }
          conn.commit();
          conn.setAutoCommit(true);
          conn.close();
```