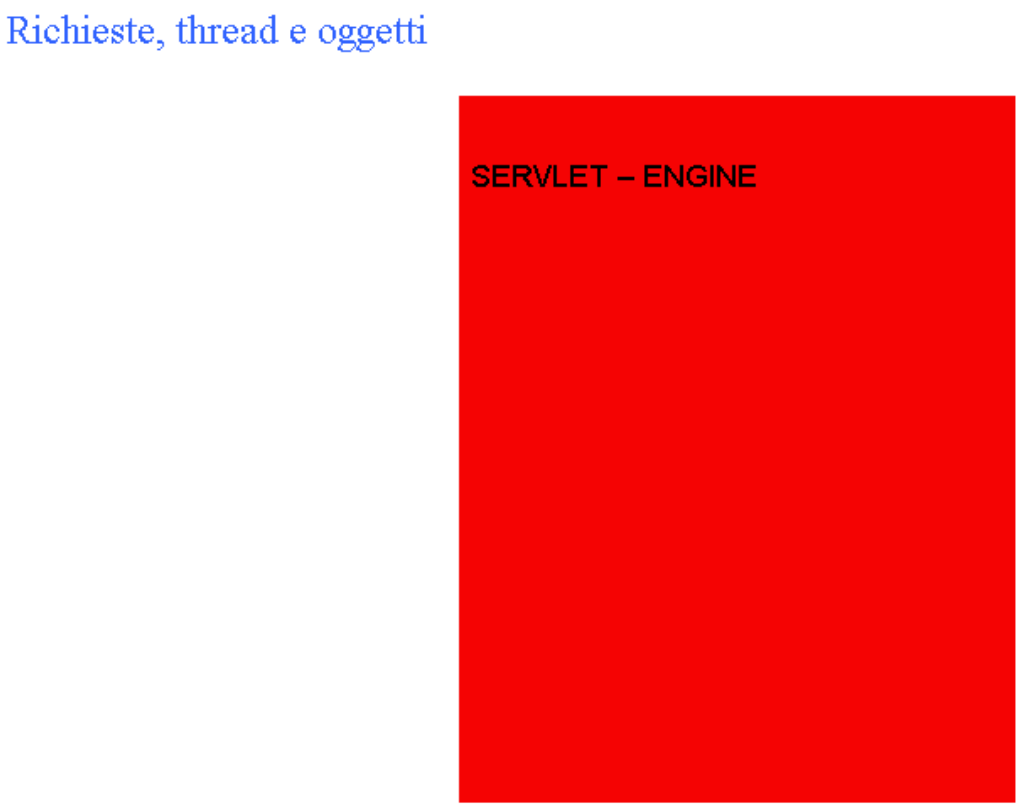
Lezione 7: Approfondimento sulle servlet

1) Richieste, Thread e Oggetti

Abbiamo visto che i CGI creano un processo per ogni richiesta. Lo standard input e l'environment sono usati per inviare informazioni dal Web server al CGI, mentre lo standard output è usato per inviare le informazioni dal CGI al Web server. Dato che ogni processo ha i propri standard input, environtment e standard output il Web server è in grado di comunicare con ogni processo CGI in maniera indipendente. Ma nel caso delle servlet come avviene la comunicazione tra Servlet e Web server? Per ogni nuova richiesta vengono creati due nuovi oggetti di tipo HttpServletRequest e HttpServletResponse. Questi oggetti sono passati come parametri al metodo doGet (o doPost) in un nuovo thread. Come sappiamo i thread condividono tutto nel processo eccetto lo stack ma dato che nello stack vanno anche i parametri dei metodi allora ogni thread avrà la possibilità di comunicare in maniera esclusiva con il motore delle servlet attraverso questi oggetti. Il motore delle servlet a sua volta comunicherà col Web server tramite un apposito protocollo per fare in modo che a client diversi corrispondano oggetti diversi.



Dove trovare documentazione sulle servlet:
http://java.sun.com/products/servlet/index.html
http://java.sun.com/products/servlet/2.2/javadoc/index.html

2) Applicazioni Web e Servlet

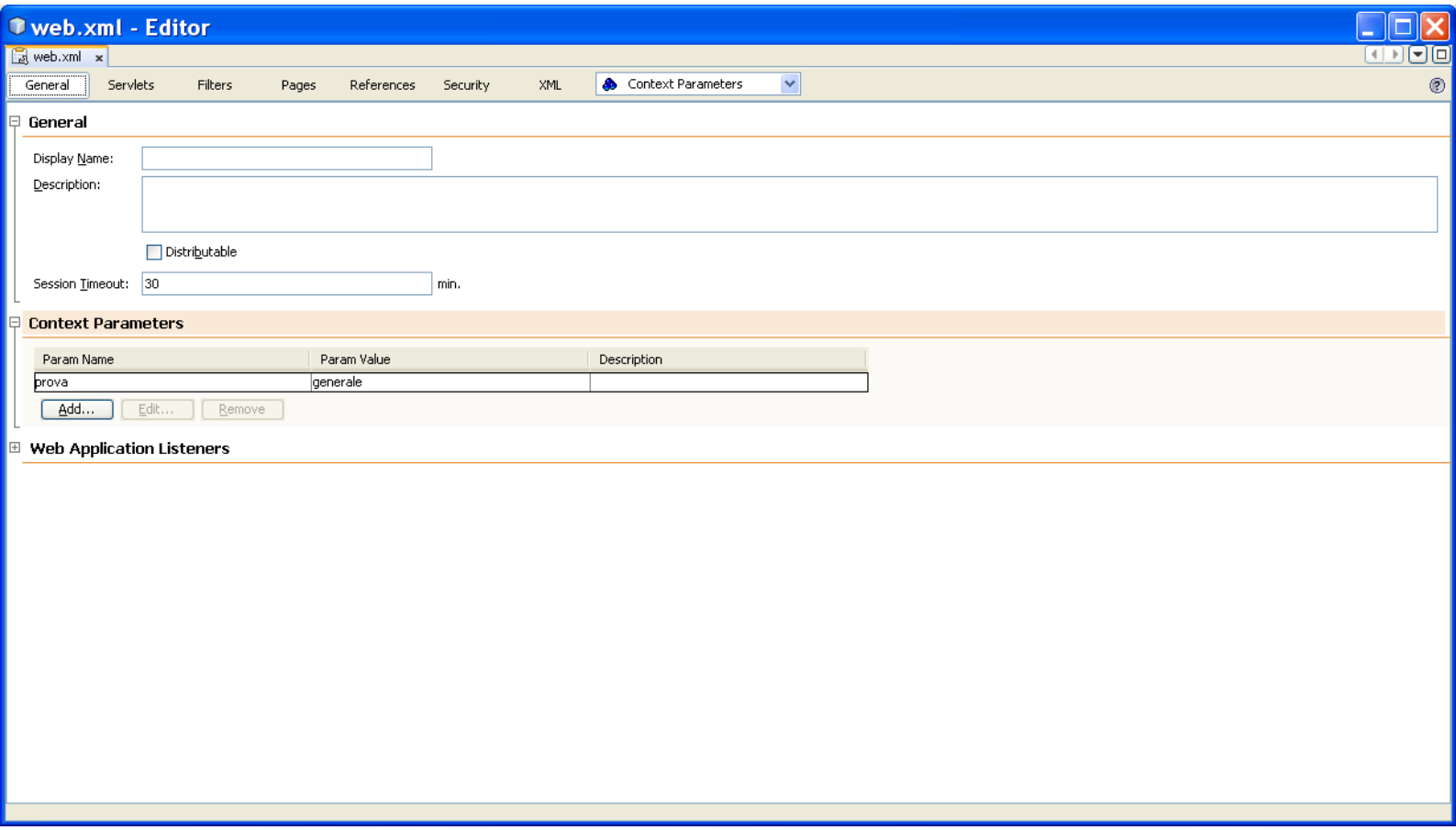
Abbiamo visto come è possibile generare una pagina dinamica con una servlet. In generale, un'applicazione Web è composta da molte pagine. L'API delle servlet ci mette a disposizione il concetto di “Web Application” in cui possiamo gestire in modo uniforme più Servlet (e anche altre risorse come le JSP e risorse statiche come vedremo). Una Web Application è quindi composta da varie risorse che poi vengono raggruppate in un file jar particolare (con estensione war). Un file che contiene le classi, le

eventuali librerie e il file web.xml.

Esigenza comune a molte applicazione è quella di avere la possibiltà di *parametrizzare* il funzionamento dell'applicazione stessa senza dover riscrivere il codice. Questo può essere utile per:

- adattare il funzionamento dell'applicazione piccole modifiche di alcuni fattori;
- personalizzare il funzionamento per i vari clienti/utenti.

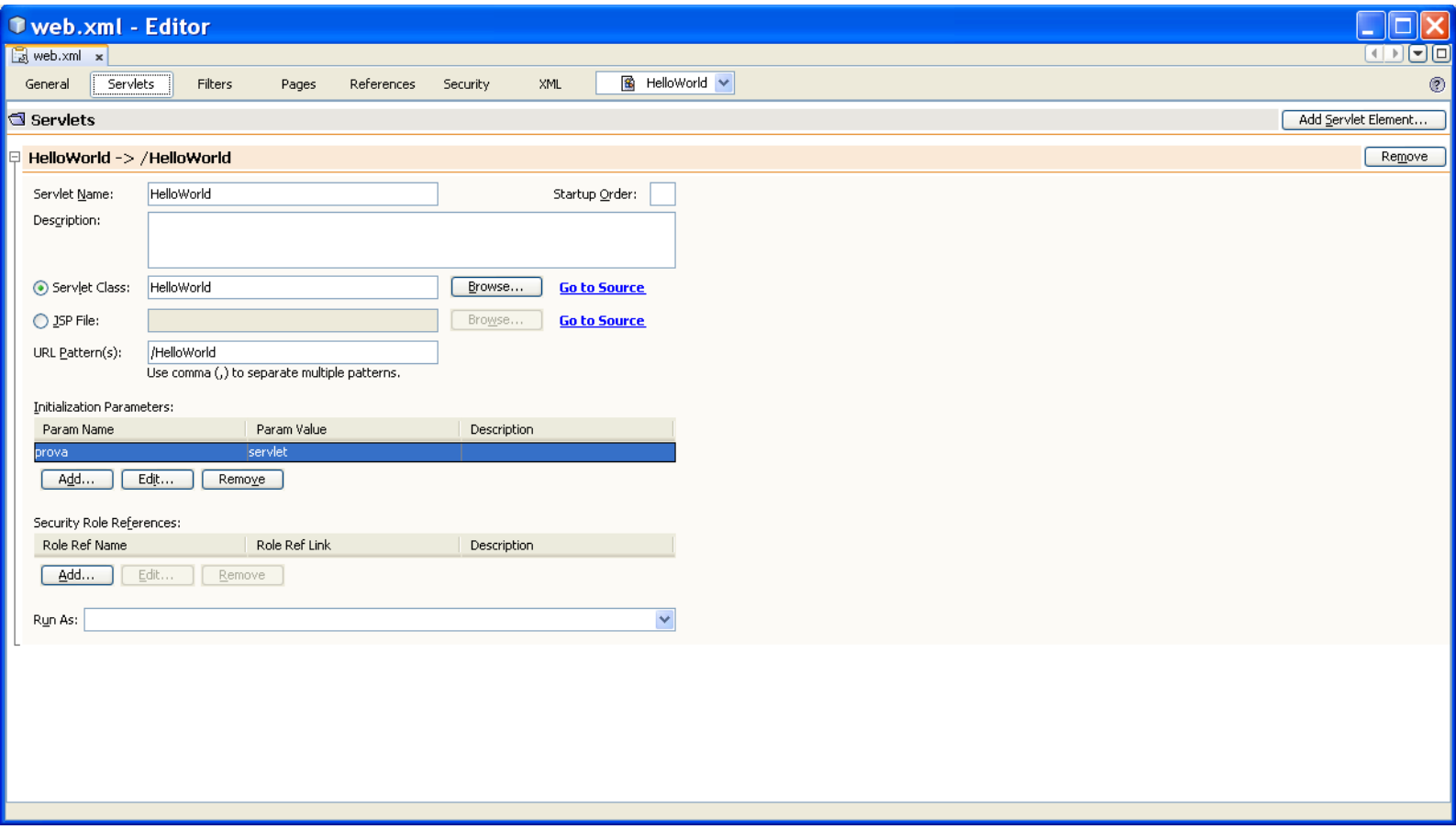
Le servlet mettono a disposizione dei parametri globali per tutta l'applicazione e dei paramentri relativi alla singola servlet.



Abbiamo inserito il parametro con nome “prova” e valore “generale” e questo è relativo a tutta l'applicazione. Quindi ogni servlet lo può recuperare con il seguente metodo:

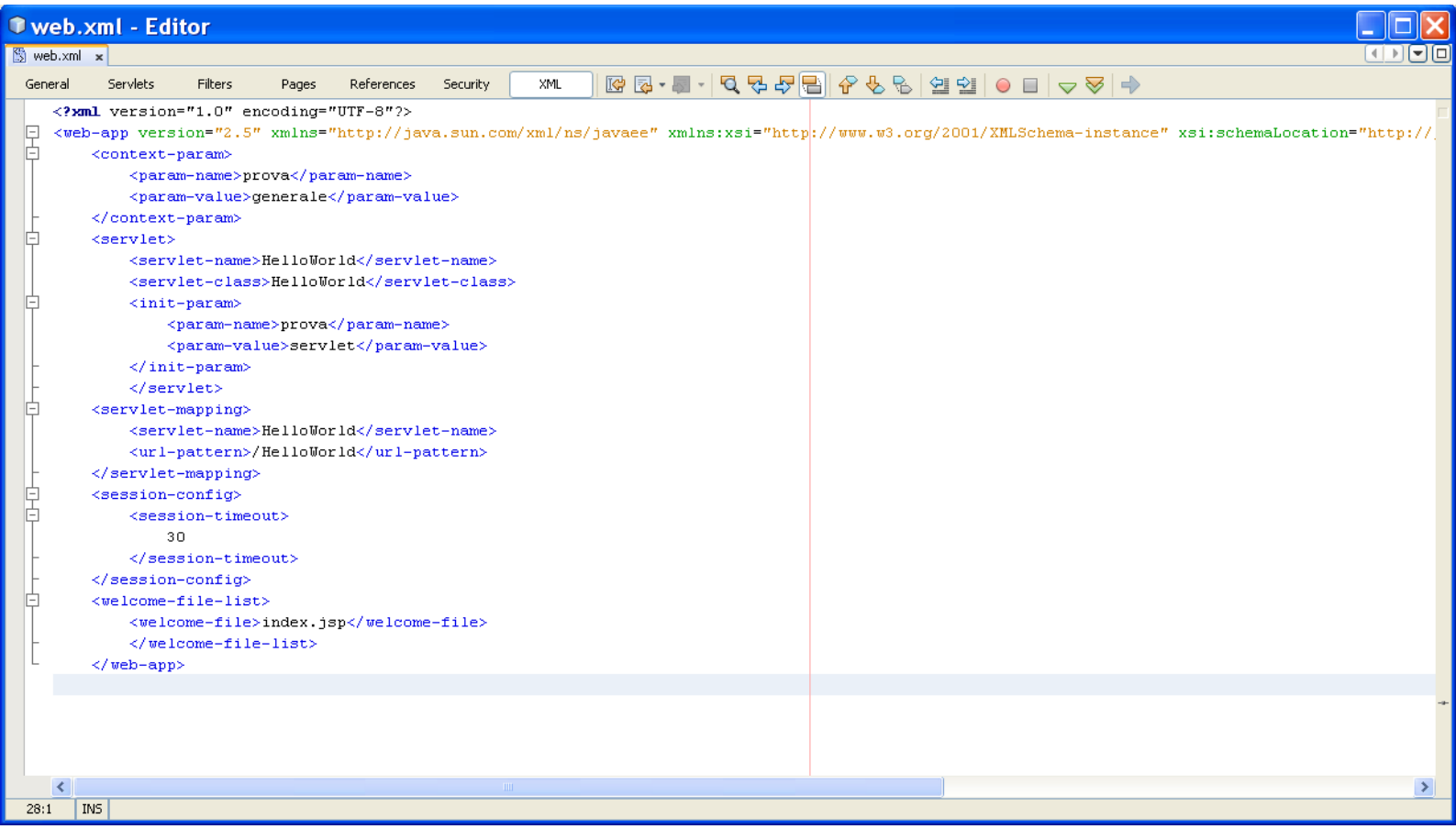
String s=getServletContext().getInitParameter("prova");

Mentre in questo caso abbiamo inserito un parametro sempre con nome “prova” ma con valore “servlet”. Solo la servlet HelloWorld può recuperare questo valore nel seguente modo:



String s=getInitParameter("prova");

Il tool NetBeans scrive per noi il file web.xml che codifica tutto quello che rappresentato visivamente dalle due schermate di sopra.



Altra informazione importante è quella del riguardante il nome della servlet, il nome della classe della servlet e il mapping tra url e la servlet. Se torniamo alla seconda figura, vediamo tre informazioni importanti: "Servlet Name", "Servlet Class" e "URL Pattern(s)". Il primo stabilisce il nome della servlet e serve a individuare l'istanza della servlet. Il secondo definisce il tipo dell'istanza della servlet: una stessa classe può essere usata per creare più servlet diverse e i parametri della servlet possono permetterne la diversificazione già dal momento della istanziazione. L'ultimo stabilisce l'URL (o gli URL) dove è "visibile" la servlet stessa.

3) Ciclo di vita di una servlet

Per razionalizzare e velocizzare il funzionamento di un applicazione web le API delle servlet prevedono il seguente ciclo di vita:

1. creazione (istanziazione) dell'oggetto servlet;
2. invocazione del metodo `init` prima di ogni altro metodo;
3. invocazione di zero, uno o più volte (anche concorrenti) dei metodi doGet e doPost;
4. incocazione del metodo destroy;
5. distruzione dell'oggetto servlet.

In pratica il motore delle servlet garantisce che:

1. il metodo init venga chiamato una sola volta prima dei metodi doGet e doPost;
2. il metodo destroy venga chiamato una sola volta e dopo tutti gli altri metodi.

3.1) metodo `init`

Abbiamo detto che le servlet offrono la possibilità di eseguire una parte di codice una sola volta e di sfruttare il lavoro fatto per rispondere a tutte le successive richieste. In questo modo possiamo ottimizzare i tempi di risposta perché non c'è bisogno di ricalcolare questa parte di codice. Per ottenere questo

dobbiamo scrivere una classe in cui re-implementare il metodo `init` della classe `Servlet`. Il metodo init viene chiamato una sola volta prima della prima invocazione da parte del servlet engine di ogni altro metodo della classe. Ecco un esempio di implementazione di tale metodo:

```
public class Test extends HttpSetvlet{

 int count;

 public void init(ServletConfig config)

 {

   count = 0;

 }

 public void doGet(HttpServletRequest req,HttpServletResponse res)

 {

  res.setContentType("text/html");

  PrintWriter out = res.getWriter();



  out.println("<htlm><body>Times" + count++ +"</html></body>");

  out.close();

}

}
```

Esercizio:

Fare in modo che il valore del contatore non venga riazzerato ad ogni nuova esecuzione del motore delle servlet.

Ad esempio supponiamo di scrivere una servlet per gestire una banca on-line. La servlet stessa potrà essere riusata per banche diverse. Ma come gestiamo quelle piccole differenze tra le varie installazioni della stessa applicazione? Ad esempio, il nome della banca dove andrà messo? La soluzione più semplice è quella di mettere nella classe della servlet una variabile d'istanza con il nome della banca:

```
public class Banca extends HttpServlet {
    String intestazione="Banca Etica";
   ....
}
```

Lo svantaggio di tale soluzione è che per una banca differente, bisogna modificare il codice Java e ricompilarlo. Una soluzione migliore è quella di mettere tutti i paramentri in un file che andrà letto prima di iniziare a gestire le richieste dei client.
Le API delle servlet non solo mettono a disposizione il metodo init che viene invocano nel momento giusto per fare queste operazioni di configurazione, ma fornisce anche un formato di file apposito e delle funzionalità apposite che permettono di accedere al contenuto dei parametri senza bisogno di conoscere come accedere ai file e/o effetturare complesse operazioni di parsing. Basterà infatti accedere a delle semplici proprietà di oggetti Java.

3.2) Parametri delle servlet

Per configurare i parametri di la servlet bisogna modificare il file `web.xml` nella directory `WEB-INF`. Ecco un esempio di tale file:

```
    <servlet>
        <servlet-name>
            BancaEticaOnLine     => nome della servlet
        </servlet-name>
        <servlet-class>
            Banca           =>come si chiama il file .class contenuto in
<docBase>WEB-INF/classes dentro il file .war
        </servlet-class>
        <init-param>       => evenutali parametri di inizializzazione
            <param-name>intestazione</param-name>
            <param-value>Banca Etica</param-value>
            <param-name>tasso</param-name>
            <param-value>3.15</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>      => altri modi di pubblicare la servlet.

        <servlet-name>
            BancaEticaOnLine       => servlet da richiamare
        </servlet-name>
        <url-pattern>
            /BancaEtica            => la servlet BancaEticaOnLineè visibile all'URL
http://<host>:<port>/<WebApp>/BancaEtica
        </url-pattern>
  </servlet-mapping>
```

Esempio

Inoltre possiamo sfruttare il metodo init per accedere a dei parametri di inizializzazione che permettono di adattare il funzionamento della servlet senza bisogno di riscirvere il codice.

```
public class Banca extends HttpServlet{
    String intestazione;
    double tassoDebitore;

public void init(ServletConfig config) throws ServletException
   {
      super.init(config);  // call the init method of base class

      intestazione = config.getInitParameter("intestazione"));
      tassoDebitore =
Double.parse(config.getInitParameter("tasso")));

   }
...
}
```

Ecco ora un esempio di implementazione che visualizza tutti i parametri di inizializzazione passati alla servlet:

```
   public void init(ServletConfig config) throws ServletException
```

```
   {
      Enumeration initParams = null;

      super.init(config);  // call the init method of base class

      System.out.println(config.getServletName());

      initParams = config.getInitParameterNames();
      String paramName = null;
      // Iterate over the names, getting the init parameters
      while ( initParams.hasMoreElements() )
      {
         paramName = (String)initParams.nextElement();
         System.out.println(paramName + ": " +
config.getInitParameter(paramName));
      }
      // initialization

   }
```

nota: i parametri iniziali vengono gestiti in maniera simile ai parametri di una richiesta. Bisogna però fare attenzione al fatto che i parametri iniziali sono relativi alla servlet, mentre i parametri della richiesta sono relativi ad una richiesta fatta dal browser.
nota: nel caso la servlet sia configurata come sopra essa è visibile tramite l'url `http://<host>:<port>/<path> /BancaEtica`.

nota: il motore delle servlet ci garantisce che il metodo `init` viene richiamato una solo volta. Quindi la seguente implementazione della chiamata al metodo init è errata!

```
class MyClass extends HttpServlet{
  boolean started=false;
  public void doGet(HttpRequest request, HttpResponse response)
  {
     if (! started)
     {
       intialize(); //errato! possibile accesso concorrente
       started=true;//ed esecuzione multipla
     }
     ...
  }
}
```

3.3) Metodo destroy

Il metodo destroy è in qualche senso simmetrico a quello init. Nel metodo destroy dobbiamo liberare tutte le risorse in uso da parte della servlet. Inoltre possiamo utilizzarlo per salvare tutte quelle informazioni che ci possono essere utile al successivo riavvio della servlet stessa.

4) Sessioni

Abbiamo già visto che nel protocollo http il server non può distingure se due richieste sono generate dallo stesso utente o da due utenti diversi. Molte applicazioni web hanno l'esigenza di identificare due richieste successive dello stesso utente in modo da semplificare all'utente la fruizione dell'applicazione stessa. Un’oggetto HttpSession serve per simulare una sessione di lavoro da parte di un utente nel protocollo HTTP. Lo scopo fondamentale dell'ogetto è quello di veicolare informazione tra una richiesta e la successiva di uno stesso utente. In Java otteniamo questo tramite il passaggio di riferimenti ad oggetti da una richiesta all'altra attraverso l'oggetto HttpSession.

Il motore delle servlet si occupa di gestire gran parte della complessità di gestione della sessione.

```
public void doGet(HttpServletRequest req, ..)
{
   HttpSession hs = req.getSession(true);
   // prima di inviare l'output
   // se non esiste, viene creato
}
```

Un'oggetto `HttpSession` viene usato per memorizzare le informazioni riguardanti lo "stato" del colloquio tra utente e server.
Gli oggetti vengono "memorizzati" nella sessione usando il seguente schema:

```
HttpSession hs = req.getSession(true);
MiaClasse o = (Classe) hs.getAttribute("oggetto");
if (o==null)
{
     o = new MiaClasse();
     hs.setAttribute("oggetto",o);
}
```

Un oggetto gestito come appena visto avrà:

- la stessa istanza per due richieste con uguale sessione e quindi con lo stesso utente;
- istanze diverse per sessioni diverse;
- le session sono relative all'applicazione web (più servlet che condividono lo stesso contesto).

Le sessioni non sono più **valide** quando:

- vengono invalidate esplicitamente dal programmatore col metodo invalidate; (es. logout)
- passa un certo tempo (configurabile).

altri metodi dell'oggetto HttpSession

public void removeAttribute(String name);
Rimuove l'attributo dalla sessione

public Enumeration getAttributeNames()
Ritorna la lista (Enumeration) di tutti i nomi delgi attributi memorizzati nella sessione;

public long getCreationTime();
Ritorna il tempo della creazione della sessione misurato in secondi trascorsi dalla mezzanotte del 1.1.1970.

public long getLastAccessdTime();
Ritorna il tempo dall'ultima richiesta fatta con sessione misurato in secondi trascorsi dalla mezzanotte del 1.1.1970.

public int getMaxInactiveInterval();
public void setMaxInactiveInterval(int sec);
Imposta e ritorna il tempo massimo di inattività della sessione. Scaduto il tempo senza ulteriori richieste da parte del client, la sessione sarà invalidata dal motore. Un valore negativo indica che la sessione non dovrà mai scadere;

public void invalidate();
Invalida la sessione immediatamente.

Esempio

In questo esempio vediamo quante volte un utente durante la sua sessione di lavoro ha avuto accesso alla servlet.

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

public class ShowSession extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse
response)
              throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    HttpSession session = request.getSession(true);
    String heading;
    Integer accessCount =
(Integer)session.getAttribute("accessCount");
    if (accessCount == null)
    {
      accessCount = new Integer(0);
      heading = "Welcome Newcomer";
    } else {
      heading = "Welcome Back";
      accessCount = new Integer(accessCount.intValue() + 1);
    }

    session.putAttribute("accessCount", accessCount);


    out.println(("<HTML><HEAD><TITLE>Test HttpSession</TITLE></HEAD>"
+
                "<BODY>\n" +
                "<H1 ALIGN=\"CENTER\">" + heading + "</H1>\n" +
                "<H2>Information on Your Session:</H2>\n" +
                "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "  <TH>Info Type<TH>Value\n" +
                "<TR>\n" +"  <TD>ID\n" +"  <TD>" + session.getId() +
"\n" +
                "<TR>\n" +"  <TD>Creation Time\n" +
                "  <TD>" + new Date(session.getCreationTime()) + "\n"
+
                "<TR>\n" +"  <TD>Time of Last Access\n" +
                "  <TD>" +new Date(session.getLastAccessedTime()) +
"\n" +
                "<TR>\n" +"  <TD>Number of Previous Accesses\n" +"
<TD>" +
                accessCount + "\n" + "</TABLE>\n" +"</BODY></HTML>");
  }
}
```