Ciclo di vita di una Servlet

- Non caricata.
- Prima richiesta:
  - Caricata: il file **`.class`** viene caricato in memoria centrale.
  - Inizializzata: viene eseguito il metodo **`init()`**.
- In servizio: risponde alle richieste con il metodo opportuno (per le servlet http vengono richiamati i metodi **`do<tipo richiesta>`**, e.g., **`doGet()`** e **`doPost()`**).
- In attesa: rimane presente in memoria centrale, ma inattiva.
- In distruzione: viene eseguito il metodo **`destroy()`** e la servlet viene rimossa dalla memoria centrale.

Errori comuni

- Tomcat manager:

  "**`FAIL - Application at context path /test could not be started`**"

  Causa: errore di parsing - controllare il file **`web.xml`**.
- "**`HTTP Status 404`**"

  Causa: servlet non trovata – controllare l'URL della servlet ed i link ad essa relativi – controllare che il tag **`<url-pattern>`** abbia un "**`/`**" iniziale (a differenza degli URL nelle pagine HTML).
- "**`HTTP Status 500`**"

  Errore interno (eccezione) della servlet.

HttpServletRequest

- E' un'interfaccia, definita in **`javax.servlet.http`**, che aggiunge alla "soprainterfaccia" **`ServletRequest`** (per servlet generiche), definita in **`javax.servlet`**, metodi specifici per le richieste HTTP.
- Rappresenta la richiesta di un client verso una servlet.
- L'oggetto corrispondente viene creato dal container (e.g., Tomcat) al momento della richiesta e passato al metodo opportuno della servlet.
- Metodi fondamentali:
  - **`getInputStream`**: per leggere informazioni inviate dal client;
  - **`getParameter`**: per estrarre i parametri della richiesta.

HttpServletResponse

- E' un'interfaccia, definita in **`javax.servlet.http`**, che aggiunge alla "soprainterfaccia" **`ServletResponse`** (per servlet generiche), definita in **`javax.servlet`**, metodi specifici per le richieste HTTP.
- Rappresenta la risposta al client da parte di una servlet.
- L'oggetto corrispondente viene creato dal container (e.g., Tomcat) al momento della richiesta e passato al metodo opportuno della servlet.
- Metodi fondamentali:
  - **`setContentType`**: per specificare il tipo MIME del contenuto che verrà spedito al client;
  - **`getWriter`**: restituisce il flusso di dati verso il client.

Un contatore di accessi "persistente"

- Modifichiamo la servlet che conta il numero di accessi in modo che conservi tale valore anche quando l'applicazione web viene riavviata.
- Sfruttiamo i metodi **`init()`** e **`destroy()`**:
  - in **`init()`** leggiamo dal file il valore iniziale del contatore **`i`**;
  - in **`destroy()`** (al momento della rimozione dalla memoria della servlet) scriviamo sul file il valore corrente del contatore **`i`**.

Codice Java (I)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ContatorePersistente extends HttpServlet {
   final static String NOME_FILE="contatore.txt";
   private int i;
   private String path; // memorizzerà il percorso di NOME_FILE.
   public void init() {
        try {
           path=getServletContext().getRealPath("")+"/";
           DataInputStream f=new
              DataInputStream(new FileInputStream(path+NOME_FILE));
           i=f.readInt();
           f.close();
        } catch (IOException e) { i=0; }
   } // continua nel lucido successivo ...
```

Codice Java (II)

```
// ... continua dal lucido precedente
   public void doGet(HttpServletRequest req,
                     HttpServletResponse res)
                     throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Contatore di "+
                    "accessi</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<STRONG>Accessi registrati "+
                    "finora: "+(++i)+"</STRONG>");
        out.println("</BODY></HTML>");
   }
// continua nel lucido successivo ...
```

Codice Java (III)

```
// ... continua dal lucido precedente
   public void destroy() {
        try {
           DataOutputStream f=
           new DataOutputStream
                (new FileOutputStream(path+NOME_FILE));
           f.writeInt(i);
           f.close();
        } catch (IOException e) { }
   }
}
```

Il metodo **`destroy()`** viene invocato al momento della rimozione della servlet dalla memoria centrale.

Modifiche al file **web.xml**

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    ...
    <servlet>
        <servlet-name>Contatore2</servlet-name>
        <servlet-class>ContatorePersistente</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Contatore2</servlet-name>
        <url-pattern>/servlet/ConPers</url-pattern>
    </servlet-mapping>

</web-app>
```

Modifiche al file `index.html`

```
<HTML>
<HEAD>
<TITLE>Servlet di prova</TITLE>
</HEAD>

<BODY>
  <TABLE>
    ...
    <TR>
      <TD>
        <A HREF="servlet/ConPers">Un contatore di accessi
                persistente</A>
      </TD>
    </TR>
  </TABLE>
</BODY>

</HTML>
```

Note

- Nell'ultimo esempio del contatore di accessi "persistente" il nome file in cui memorizzare il numero di "hit" è codificato direttamente nel codice della servlet.
- Se vogliamo cambiare il nome del file o il valore iniziale del contatore, dobbiamo modificare il sorgente, ricompilare e riavviare l'applicazione web.

Variante

- Per evitare la ricompilazione della servlet, si può memorizzare il nome del file e/o il valore iniziale del contatore nel file di configurazione **`web.xml`**.
- Tale parametro può poi essere recuperato dalla servlet.
- Infatti il server (Tomcat) passa automaticamente al metodo **`init()`** un oggetto di tipo **`ServletConfig`** che include i parametri di inizializzazione accessibili tramite i metodi **`getInitParameter()`** e **`getInitParameterNames()`**.
- Sia **`GenericServlet`** che **`HttpServlet`** implementano l'interfaccia **`ServletConfig`**: è sufficiente quindi richiamare **`super.init()`** per avere a disposizione tali metodi.

Specificare i parametri in `web.xml`

```
. . .
    <servlet>
        <servlet-name>Contatore3</servlet-name>
        <servlet-class>ContatorePersistente2</servlet-class>
        <init-param>
            <param-name>nomeFile</param-name>
            <param-value>contatore.txt</param-value>
        </init-param>
        <init-param>
            <param-name>valoreContatore</param-name>
            <param-value>0</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>Contatore3</servlet-name>
        <url-pattern>/servlet/ConPersParam</url-pattern>
    </servlet-mapping>
```

Accedere ai parametri memorizzati in **web.xml**

```
public class ContatorePersistente2 extends HttpServlet {
   private int i;
   private String path;

   public void init(ServletConfig conf) throws ServletException {
        super.init(conf);
        try {
                path=getServletContext().getRealPath("")+"/"+
                     getInitParameter("nomeFile");
                DataInputStream f=new DataInputStream
                                  (new FileInputStream(path));
                i=f.readInt();
                f.close();
        } catch (IOException e) {
                i=Integer.parseInt(getInitParameter("valoreContatore"));
        }
   }
```

Il resto del codice (I)

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
                  throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Contatore di "+
                "accessi</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<STRONG>Accessi registrati "+
                "finora: "+(++i)+"</STRONG>");
    out.println("</BODY></HTML>");
}
```

Il resto del codice (II)

```
    public void destroy() {
          try {
                    DataOutputStream f=new DataOutputStream
                                          (new FileOutputStream(path));
                    f.writeInt(i);
                    f.close();
          } catch (IOException e) { }

    }

}
```

Il resto del codice rimane pressoché invariato.

Modifiche al file `index.html`

```
<HTML>
<HEAD>
<TITLE>Servlet di prova</TITLE>
</HEAD>

<BODY>
  <TABLE>
    . . .
    <TR>
      <TD>
        <A HREF="servlet/ConPersParam">Un contatore di accessi
          persistente con parametri memorizzati in web.xml</A>
      </TD>
    </TR>
  </TABLE>
</BODY>

</HTML>
```

Note

- Memorizzando i parametri fondamentali di una servlet nel file che descrive il dispiegamento (**`web.xml`**), si evita di ricompilare il codice in caso di cambiamenti.
- Tuttavia è sempre necessario riavviare (Stop + Start) l'applicazione web tramite il Tomcat Manager.

Thread safety

- Tipicamente ad ogni dato istante soltanto una copia di ogni servlet è presente in memoria centrale.
- Tuttavia ogni servlet può trovarsi a dover soddisfare più richieste concorrenti.
- Siccome ogni richiesta genera un thread distinto, le servlet dovrebbero essere thread-safe.
- Se i metodi di una servlet invocati da Tomcat per soddisfare una richiesta non accedono a variabili visibili al di fuori dei metodi stessi, la servlet è automaticamente thread-safe.

Thread safety

- Tuttavia una servlet che mantiene delle risorse persistenti, deve assicurare che queste ultime siano aggiornate in modo consistente.
- Un tipico esempio è costituito dall'utilizzo di una servlet per gestire un conto bancario (una membro di tipo **`int`** della classe mantiene l'ammontare corrente del conto).

Un tipico scenario

- Un primo utente accede alla servlet per prelevare 100 Euro.
- La servlet controlla l'ammontare del conto, verificando la presenza di 120 Euro e consentendo l'operazione.
- Un secondo utente accede alla servlet per prelevare 50 Euro.
- La servlet controlla l'ammontare del conto, verificando la presenza di 120 Euro e consentendo l'operazione.
- La servlet addebita 100 Euro per la richiesta del primo utente e immediatamente dopo 50 Euro per la richiesta del secondo utente.
- Risultato: il bilancio complessivo del conto è in rosso di 30 Euro.
- Il programmatore viene licenziato in tronco.

Come evitare il licenziamento

- E' necessario fare in modo che la servlet soddisfi una richiesta per volta, utilizzando una delle due soluzioni seguenti:
  - inglobando il codice critico in blocchi **`synchronized`**;
  - implementando l'interfaccia **`SingleThreadModel`** (viene creato un insieme di istanze di servlet, invece di una sola; ogni copia può servire una singola richiesta per volta).
- La seconda soluzione è più onerosa dal punto di vista della gestione delle risorse del server.

Il codice Java (I)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
public class Bilancio extends HttpServlet {
    private String path;
    class ContoCorrente {
          public int bilancio;
    }
    ContoCorrente conto;
    public void init(ServletConfig conf) throws ServletException {
      super.init(conf);
      conto = new ContoCorrente();
      try {
        path=getServletContext().getRealPath("")+"/"+
             getInitParameter("nomeFile");
        DataInputStream f=new DataInputStream(new FileInputStream(path));
        conto.bilancio=f.readInt();
        f.close();
      } catch (IOException e) { conto.bilancio =
                  Integer.parseInt(getInitParameter("valoreConto")); }
    }
```

Oggetto su cui useremo il blocco synchronized.

Il codice Java (II)

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><BODY>");
    out.println("<H2>Java Bank</H2>");
    out.println("Bilancio corrente: <B>" + conto.bilancio +
                "</B><BR>");
    out.println("<FORM METHOD=\"post\" ACTION=\"Bilancio\">");
    out.println("Ammontare: <INPUT TYPE=\"text\" NAME=\"Ammontare\""+
                " SIZE=\"3\"><BR>");
    out.println("<INPUT TYPE=\"submit\" NAME=\"Deposito\""+
                " VALUE=\"Deposito\">");
    out.println("<INPUT TYPE=\"submit\" NAME=\"Prelievo\""+
                " VALUE=\"Prelievo\">");
    out.println("</FORM>");
    out.println("</BODY></HTML>");
}
```

Come la servlet rimanda il post a se stessa (notare l'assenza di "servlet/")

Il codice Java (III)

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
  int ammontare=0;
  try {
     ammontare = Integer.parseInt(req.getParameter("Ammontare"));
  } catch (NullPointerException e) {
     // Ammontare dell'operazione non specificato
  } catch (NumberFormatException e) {
    // Ammontare dell'operazione non valido
  }
  synchronized(conto) { // inizio del blocco sincronizzato
     if(req.getParameter("Prelievo") != null &&
        (ammontare <= conto.bilancio))
        conto.bilancio = conto.bilancio - ammontare;
     if(req.getParameter("Deposito") != null && (ammontare > 0))
        conto.bilancio = conto.bilancio + ammontare;
  } // fine del blocco sincronizzato
  doGet(req, res); // ripresenta il form
}
```

Il codice Java (IV)

```
 public void destroy() {
        try {
           DataOutputStream f=new DataOutputStream
                                (new FileOutputStream(path));
           f.writeInt(conto.bilancio);
           f.close();
        } catch (IOException e) { }
 }
}
```

Il metodo **`destroy()`** permette di salvare il bilancio del conto corrente nel file specificato in **`web.xml`** (**`bilancio.txt`**).

Modifiche al file **web.xml**

```
. . .
<servlet>
    <servlet-name>Bilancio</servlet-name>
    <servlet-class>Bilancio</servlet-class>
    <init-param>
        <param-name>nomeFile</param-name>
        <param-value>bilancio.txt</param-value>
    </init-param>
    <init-param>
        <param-name>valoreConto</param-name>
        <param-value>0</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>Bilancio</servlet-name>
    <url-pattern>/servlet/Bilancio</url-pattern>
</servlet-mapping>
```

Modifiche al file `index.html`

```
<HTML>
<HEAD>
<TITLE>Servlet di prova</TITLE>
</HEAD>

<BODY>
  <TABLE>
    . . .
    <TR>
      <TD>
        <A HREF="servlet/Bilancio">Bilancio di un conto corrente</A>
      </TD>
    </TR>
  </TABLE>
</BODY>

</HTML>
```

Esercizi

- Modificare la servlet “Bilancio” in modo che visualizzi un messaggio di avvertimento nel caso in cui l’utente tenti di prelevare una somma maggiore di quella depositata.
- Aggiungere la funzionalità di autenticazione alla servlet “Bilancio” in modo che l’utente debba inserire un nome utente ed una password prima di effettuare il prelievo.