Servlet

Nicola Gessa

Introduzione alle servlet

- Le servlet sono oggetti java che vengono caricati ed eseguiti dal web server all'interno del processo di richiesta/risposta di servizi.
- Le servlet consentono l'estensione delle potenzialità dei servizi di rete
- Il web server agisce difatto come "container" che si occupa della gestione del ciclo di vita delle servlet
- Il web server passa alle servlet i dati del client e restituisce ai client i dati prodotti dall'esecuzione delle servlet

Es. Una servlet può estendere le potenzialità di un web server ricevendo i dati inseriti da un client attraverso una form e operando su di essi inserendoli per esempio in un DB.

Servlet: caratteristiche

- Le servlet non vengono eseguite in processi separati, quindi ogni richiesta non causa la creazione di un nuovo processo
- Le servlet risiedono in memoria tra una richiesta e l'altra, quindi vengono caricate solo una volta, alla prima richiesta. Ciò comporta che anche le inizializzazioni necessarie vengono eseguite una volta sola
- Una solo istanza della servlet risponde a più richieste
- Dispongono di classi per la gestione delle sessioni
- Garantiscono maggiore velocità di esecuzione rispetto a script CGI
- Consentono una più semplice condivisione dei dati fra le istanze di una servlet
- Ereditano la portabilità di Java

Architettura

- Il package Java di base per le API Servlet è javax.servlet:
  - contiene la definizione dell'interfaccia Servlet e
  - contiene classi utili alla comunicazione fra client e server
- L'interfaccia Servlet
  - contiene i prototipi di tutti i metodi necessari alla gestione del ciclo di vita di una servlet e alla esecuzione delle operazione implementate dalla servlet
  - i metodi definiti in questa intefaccia devono essere supportati da tutte le servlet
- Tutte le servlet sono classi che implementano l'intefaccia servlet, o in maniera diretta, o estendendo una classe che la implementa come ad esempio HttpServlet

Architettura

- Il package javax.servlet.http fornisce classi che estendono le funzionalità di base di una servlet adottando le caratteristiche del protocollo http come
  - gestione dei metodi HTTP (GET, POST)
  - gestione degli header HTTP

- Per creare una servlet che utilizzi il protocollo http per ricevere e fornire dati si può implementare una classe che estenda la classe javax.servlet.http.HttpServlet

Architettura

- Quando una servlet accetta una richiesta da un client riceve due oggetti:
  - ServletRequest, utilizzato per la comunicazioner dal client verso il server
  - ServletResponse, utilizzato per gestire la comunicazioie dal server verso il client

Estensioni di queste classi consentono di disporre di metodi per manipolare informazioni e header specifici di determinati protocolli. In particolare le servlet che utilizzano il protocollo HTTP usano oggetti che sono istanza delle classi HttpServletRequest e HttpServletResponse

Servlet concorrenti

- Il web server per definizione utilizza una sola istanza di servlet condividendola fra le varie richieste
- N richieste da parte di client della stessa servlet portano alla creazione di n threads differenti da parte del web server capaci di eseguire la servlet in maniera concorrente.
- In fase di programmazione si deve tenere conto dei meccanismi di accesso concorrente alle servlet a ai loro dati.
- L'operatore synchronized consente di sincronizzare blocchi di codice della servlet (o anche tutto il metodo service) difendendoli da accessi concorrenti da parte di più threads.
- Per creare più istanze di una stessa servlet da associare alle richieste, la servlet deve implementare l'interfaccia SingleThreadModel. In questo modo il web server creerà più istanza della stessa servlet al momento del caricamento dell'oggetto e assegnerà ai threads solo istanza libere.

L'oggetto HttpServerRequest

Rappresenta una richiesta http e mediante i metodi di questa classe è possibile

- accedere a parametri o meta informazioni relative alla richiesta. Es:
  - getRequestURI() restituisce l'URI richiesto
  - getMethod() fornisce il metodo HTTP utilizzato per inoltrare la richiesta
  - il metodo getParameter(String) consente di accedere per nome ai parametri contenuti nella query string
- creare gli stream di input e la possibilità quindi di ricevere i dati della richiesta attraverso i metodi getInputStream e getReader

L’oggetto HttpServerResponse

Fornisce alla servlet i metodi per rispondere al client:

- permette di fissare parametri relativi alla risposta inviata (come lunghezza o tipo MIME)
- fornisce metodi per manipolare gli header del protocollo http
- rende possibile il reindirizzamento
- fornisce i metodi per creare gli stream di output e la possibilità quindi di inviare i dati della risposta.
  - getOutputStream per l’invio di dati in forma binaria
  - getWriter per l’invio attraverso il canale System.out
- definisce una serie di constanti per inviare al browser il risultato della operazione richiesta

Ciclo di vita delle servlet

Il ciclo di vita di una servlet definisce:

- come una servlet viene caricata: caricamento e istanziazione sono eseguiti dal web server e avvengono al momento della prima richiesta da parte dei client
- come una servlet viene inizializzata: in questa fase la servlet carica dati persistenti , apre connessioni verso DB. Questa operazione avviene tramite la chiamata al metodo init(). Il metodo di default non esegue nessuna operazione
- come può ricevere e rispondere alle richieste dei client: il metodo eseguito all'arrivo di una nuova richiesta è il metodo service(), che riceve come parametri gli oggetti ServletRequest e ServletResponse per gestire le richieste
- come viene terminata (tipicamente quando termina l'esecuzione del web server): ad esempio si specifica come rilasciare le risorse occupate. Questa operazione avviene tramite la chiamata al metodo destroy()

Inizializzazione di una servlet

- L'inizializzazione di una servlet avviene con la chiamata del metodo init() al momento del suo caricamento
- Un solo threads viene eseguito al momento della inizializzazione
- Le richieste possono essere ricevute dalla servlet solo dopo il completamento della fase di inizializzazione
- Se la servlet non riesce a completare l'inizializzazione viene generata una eccezione
- Il metodo init() riceve come parametro un oggetto di tipo ServletConfig che contiene la configurazione iniziale di una servlet. Per salvare la configurazione si puo' richiamare il metodo super.init, altrimenti si deve implemetare questa operazione nel nuovo init.

Inizializzazione di una servlet

- I parametri di configurazione di una servlet sono definiti all'interno della configurazione del web server che li comunica alla servlet nella forma chiave=valore.
- I metodi utilizzati dall'oggetto ServletConfig per accedere ai parametri sono: getInitParameterNames() e getInitParameter(String)
- Si puo accedere ai parametri di configurazione anche tramite il metodo getServletConfig() dell'interfaccia servlet all'interno del metodo service()

Es. public void init(ServletConfig config){

```
super.init(config);
String initpar = config.getInitParameter('PARAMETER');
if(initpar== null){
        throw new UnavailableException(this,"errore!")
}}
```

Il metodo service()

- Se il metodo service() non viene sovrascritto, la nostra classe eredita il metodo definito all'interno della classe che andiamo ad estendere

- Se estendiamo la classeHttpServlet, questo metodo ha la funzione di distribuire la gestione della richiesta fra altri metodi in base al tipo di richiesta ricevuta. Ad esempio in caso di richiesta GET o POST vengono richiamati in automatico rispettivamente i metodi doGet() e doPost()
  - Nel caso non si implementi una nuova versione del metodo service() sarà necessario implementare nella nostra classe un metodo scelto fra questi in base al tipo di richiesta da gestire.

Interazione con client

Scrivere una servlet che estende la classa HttpServlet per gestire il protocollo HTTP comporta l'implementazione dei metodi definiti per gestire l'interazione HTTP col client che si ritengono necessari.

Es.

- doGet, per gestire richiesta di tipo GET o HEAD
- doPost, per gestire richiesta di tipo POST
- doPut, per gestire richiesta di tipo PUT

Di default queste richieste restituiscono il codice HTTP BAD_REQUEST (400)

Tali metodi ricevono tutti 2 argomenti, gli oggetti HttpServletRequest e HttpServletResponse per la gestione dei dati ricevuti o inviati al client

Interazione con client

Il modo in cui si accede ai dati del client può dipendere dal metodo HTTP della richiesta:

- Con tutti i metodi HTTP, il metodo getParameterValues fornisce i valori in base al nome del parametro, il metodo getParameterNames fornisce i nomi dei parametri
- Col metodo GET si può utilizzare il motodo getQueryString che restituisce una stringa da parserizzare
- Con i metodi POST,PUT si può scegliere fra i metodi getReader per la lettura di dati di tipo testo o getInputStream per la creazione di uno stream per la lettura di dati binari

Per inviare dati al client sono disponibili invece i metodi

- getWriter, per inviare dei dati di tipo testo
- getOutputStream per inviare dati binari

Interazione con client

Procedura per l'invio di dati (per esempio nella creazione di una pagina html dinamica):

- Prima di inviare dati tramite l'oggetto Writer o OutputStream, è necessario fissare gli header HTTP.
- La classe HttpServletResponse fornisce dei metodi per accedere e fissare gli header della risposta, come il content-type, encoding, content lenght.
- Una volta impostati gli header della risposta, si può inviare il body

Un esempio: Salve Mondo!

Vogliamo creare una servlet la cui chiamata da parte di un client restituisca una semplice pagina HTML con la scritta “Salve Mondo!”:

- Creiamo un file SalveMondoServlet.java
- Compiliamo il file con javac per ottenere il file SalveMondoServlet.class
- Rendiamo il file disponibile via web attraverso un URL

Un esempio: Salve Mondo!

- Import dei package necessari alla servlet

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

- Dichiarazione della classe che sarà la nostra servlet. Poiché creaimo una servlet HTTP estendiamo la classe javax.servlet.http.HttpServlet

```
public class SalveMondoServlet extends HttpServlet
```

- Volendo far gestire alla nostra servlet richieste di tipo GET, ridefiniamo il metodo doGet()

```
protected void doGet(HttpServletRequest req, HttpServletResponse
    res)
```

- Impostiamo il tipo MIME della risposta da inviare al client.

```
res.setContentType("text/html");
```

Un esempio: Salve Mondo!

- Creiamo un oggetto PrintWriter associato alla risposta per la scrittura verso il client.

```
PrinterWriter out =res.getWriter();
```

- Utilizziamo l'oggetto PrintWriter per creare il testo della pagina da inviare al client.

```
out.println("<HTML><HEAD><TITLE>SalveMondo!</TITLE>");
out.println("</HEAD><BODY>SalveMondo!</BODY></HTML>");
```

- Chiudiamo l'oggetto PrintWriter (il server chiude gli stream di input/output automaticamente alla terminazione dell'esecuzione di una servlet).

```
out.close();
```

Questa chiamata informa il server che la risposta è terminata e la connessione può essere chiusa

Un esempio: Salve Mondo!

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

public class SalveMondoServlet extends Httpservlet{
        protected void doGet(HttpServletRequest req, HttpServletResponse)
                throws ServletException, IOException{
                res.setContentType(“text/html”);
                PrinterWriter out =res.getWriter();
                out.println(“<HTML><HEAD><TITLE>SalveMondo!</TITLE>”);
                out.println(“</HEAD><BODY>SalveMondo!</BODY></HTML>”);
                out.close();
        }
}
```

Ottenere informazioni sul client

Vediamo come si puo' riscrivere il metodo service() per ottenere informazioni sulla richiesta effettuata dal client:

```
public class InfoClient extends HttpServlet{
        public void service(HttpServletRequest req, HttpServletResponse)
            throws ServletException, IOException{
                res.setContentType("text/html");
                 PrinterWriter out =res.getWriter();
                out.println("<HTML><HEAD><TITLE>Informazioni</TITLE>");
                out.println(req.getProtocol()+"<BR>");
                out.println(req.getServerPort()+"<BR>");
                out.println(req.getCharacterEncoding()+"<BR>");
                out.println(req.getRemoteHost()+"<BR>");
                out.println("</ </BODY></HTML>");
                out.println("</HEAD><BODY>
}
```

Gestire il metodo POST - 1

Creiamo una servlet che riceve da input due campi inseriti dall'utente e li salva su file.

La form di inserimento sarà:

```
<html>
<head><title>Test servlet</title></head>
<body>
        <form action="http://host/myservlet" method="post">
        <br><input type= "text" name="nome" value="Mario">
        <br><input type= "text" name="cognome" value="Rossi">
        <br><input type= "submit">
</body>
</html>
```

Gestire il metodo POST - 2

```
Public void doPost (HttpServletRequest req, HttpServletResponse)
        throws ServletException, IOException{
        res.setContentType("text/html");
        PrinterWriter out =res.getWriter();
        FileWriter myFile = new FileWriter("nomefile",true);
        PrinterWriter toMyFile = new PrinterWriter(myFile);
        Enumeration values = req.getParameterNames();
        while(values.hasMoreElements()){
                String name = (String) valus.nextElement);
                String value = req.getParameterValues(name);
                if(name.compareTo("submit)!=0){
                        toMyFile.println(name+":"+value);}
        }
        myFile.close();
        out.println("<html><body>Grazie di aver partecipato!</body></html>");
        out.close();}
```

Servlet: uso delle query string

Utilizzando il metodo GET del protocollo http i dati inviati dal client vengono appesi alla URL richiesta nella forma di coppie nome=valore

ES. www.prova.it/lista.html?Citta=Bologna&cap=40125”

I valori di una query string possono essere recuperati utilizzando i seguenti metodi della classe HttpServletRequest

- String getQueryString()

  ritorna la query string completa

- Enumeration getParameterNames()

  ritorna una enumerazione dei nomi dei parametri

- String getParameter(String)

  ritorna il valore del parametro a partire dal suo nome

- String[] getParameterValues()

  ritorna una array di valori del parametro

Uso delle sessioni

- Una sessione è rappresentata attraverso la classe HttpSession
- La creazione di una sessione può essere fatta all'interno dei metodi doGet o doPost tramite il metodo getSession dell'oggetto HttpServletRequest

```
HttpSession session = req.getSession(true);
```

- Con questa chiamata la servlet cerca di identificare il client, e se non esiste già una sessione associata alla richiesta ne viene creata una nuova.
- Una volta ottenuto l'oggetto session, questo può essere utilizzato tramite i suoi metodi per memorizzare qualsiasi tipo di informazione.
- Un sessione rimane valida finchè non viene invalidata col metodo invalidate(), oppure scade il suo timeout.

Uso delle sessioni

- La creazione di una sessione comporta in pratica la predisposizione di un'area di memoria per la gestione delle informazioni e la creazione "trasparente" di un cookie con un numero di sessione.
- Se il browser non supporta il cookie, il numero di sessione viene registrato nelle query string al momento della creazione del codice della pagina.

http://localhost/servlet/myServlet?sessionID=982476523493113 3

- I metodi dell'oggetto HttpSession permettono di avere informazioni sulla sessione (id della sessione, tempo trascorso, tempo di inattività) o di memorizzare dati di sessione nella forma di coppia nome=valore.

putValue(String name,Object value)

void removeValue(String name)

Esempio di uso delle Sessioni

```
public void service(HttpServletRequest req, HttpServletResponse)
                    throws ServletException, IOException{
        res.setContentType("text/html");
        PrinterWriter out = res.getWriter();
        out.println("....."); //stampo i tag iniziali
        HttpSession sessione = req.getSession(true);
        if(session.isNew()){
                out.println("<br>Tempo di creazione:"+session.creationTime());
                session.putValue("accessi",new Integer(1));
        }else{
                int accessi=((Integer)session.getValue("accessi")).intValue();
                accessi++;
                 out.println("<br>Numero accessi:"+accessi);}
        out.println(…);      //stampo i tag finali
        out.close
}
```

Esempio di gestione di una lista - 1

Come esempio vediamo l'implementazione di una servlet per la gestione di una lista di email.

La struttura della servlet è:

```
public class ListManagerServlet extends HttpServlet{
        private Vector emaillist;
        init();             //esegue le inizializzazioni
        doGet();            //riceve la richiesta della lista delle email
        doPost();           //riceve i dati per l'inserimento di una
                            //nuova email
        subscribe()         //esegue una nuova sottoscrizione
        unsubscribe()       //cancella una email
        save()              //salva le modifiche
}
```

Esempio di gestione di una lista - 2

Il motodo doGet è:

```
Protected void doGet (HttpServletRequest req, HttpServletResponse)
        throws ServletException, IOException{
        res.setContentType("text/html");
        res.setHeader("pragma","no-cache");
        PrinterWriter out = res.getWriter();
        out.print("<HTML>…..");
        for(i=0;i<emaillist.size();i++){
                out.print("<LI>"+emaillist.elementAt(i);
        }
        out.print("<INPUT type=text name=email><BR>");
        out.print("<INPUT type=submit name=action value=subscribe><BR>");
        out.print("<INPUT type=submit name=action value=unsubscribe><BR>");
        out.print("…..</HTML>");
        out.close;
}
```

Esempio di gestione di una lista - 3

```
public void doPost (HttpServletRequest req, HttpServletResponse)
        throws ServletException, IOException{
        String email=req.getParameter;      String msg;
        if(email==null){
                res.sendError(res.SC_BAD_REQUEST,"nessuna email ins.");
                return;}
        if(req.getParameter("action").equals("subscribe")){
                if(subscribe(email))msg="Sottoscrizione avvenuta";
                else{
                res.sendError(res.SC_BAD_REQUEST,"nessuna email ins.");
                return;}
        }else{
                 if(unsubscribe(email))msg="Rimozione avvenuta";
                else{
                res.sendError(res.SC_BAD_REQUEST,"errore!");
                return;}}
```

Esempio di gestione di una lista - 4

```
        res.setContentType(“text/html”);
        res.setHeader(“pragma”,”no-cache”);
        PrinterWriter out = res.getWriter();
        out.print(“<HTML>-----<BODY>”);
        out.print(msg);
        out.print(”<A HREF=\””);
        out.print(req.getRequestURI());
        out.print(“\”>>Torna alla lista”);
        out.print(“…..</HTML>”);
        out.close();
}
```

Esempio di gestione di una lista - 5

I metodi per registrare le operazioni degli utenti sono:

```
private syncronized boolean subscribe(String email)throws IOException{
        if(!emaillist.contains(email)) return false;
        emaillist.addElement(email);
        save();
        return true;
}

private syncronized boolean unsubscribe(String email)throws IOException{
        if(!emaillist.removeElement(email)) return false;
        save();
        return true;
}
```

Uso dei Cookie

- L'oggetto Session utilizza i cookie per consentire l'identificazione delle sessioni ma memorizza i parametri della sessione sul server.
- I cookie possono essere utilizzati per memorizzare altre informazioni sul client
- La Java fornisce la classe javax.servlet.http.Cookie per rappresentare e gestire i cookie senza dover manipolare gli header http.
- Sui cookie si può definire:
  - il dominio applicativo (metodo setDomain)
  - il path dell'applicazione (metodo setPath)
  - la durata di validità (metodo setMAxAge)
- I cookie possono essere aggiunti o letti attraverso l'oggetto HttpServletRequest

Uso dei cookie:esempio

```
//Aggiungere un nuovo cookie
Cookie mycookie=null;
String nome="nome_cookie";
String valore="valore_cookie";
mycookie=new Cookie(nome,valore);
mycookie.setMaxAge(1000);
res.addCookie();
//Leggere i cookie ricevuti
Cookie cookies[]=req.getCookies();
for(int i=0;i<cookies.length;i++){
        Cookie cookie_ric=cookies[i];
        out.print(cookie_ric.getName()+"="+ cookie_ric.getValue());
}
```

Il tag <SERVLET>

Le servlet possono essere richiamati anche tramite uno speciale tag HTML, **<SERVLET>**

Es.

<SERVLET NAME=MyServlet>

<PARAM NAME=param1 VALUE=valore1>

<PARAM NAME=param2 VALUE=valore2>

</SERVLET>

Quando una pagina con questi tag viene richiamata, i tag <SERVLET> mandano in esecuzione la servlet specificata e vengono poi sostituiti dal risultato della loro elaborazione.

I tag <PARAM> specificano dei parametri da passare alle servlet, che comunque possono accedere ai parametri spediti via GET o POST.

Link utili

- http://java.sun.com/products/servlet/index.html
- http://java.apache.org/