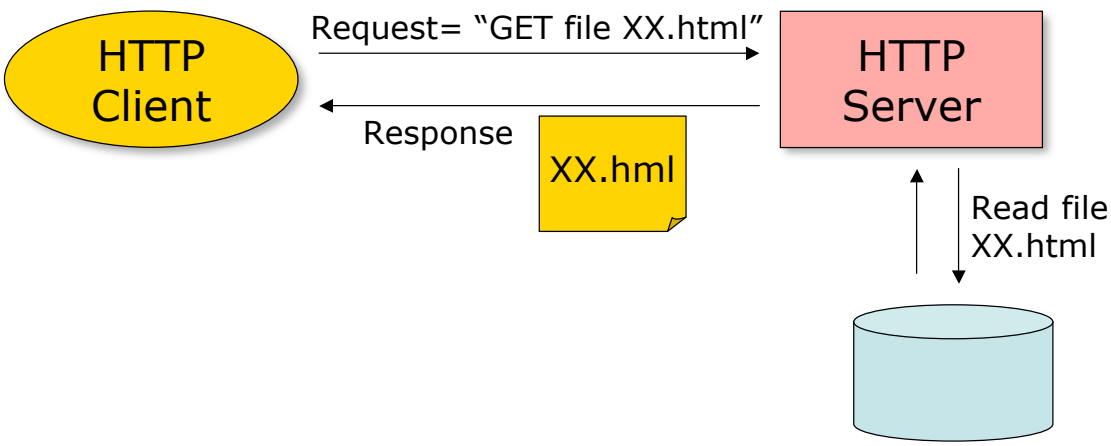
Servlet

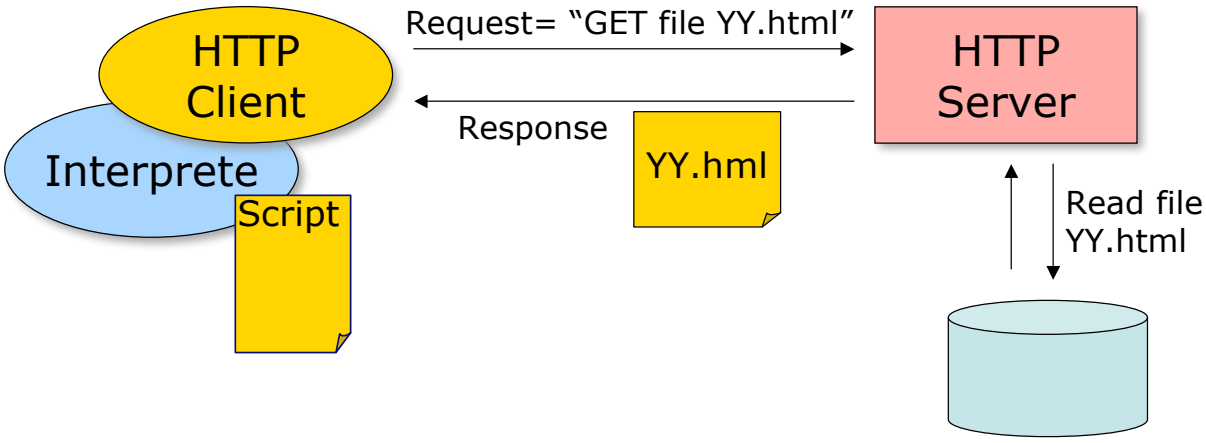
Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

Modelli per Applicazioni Web



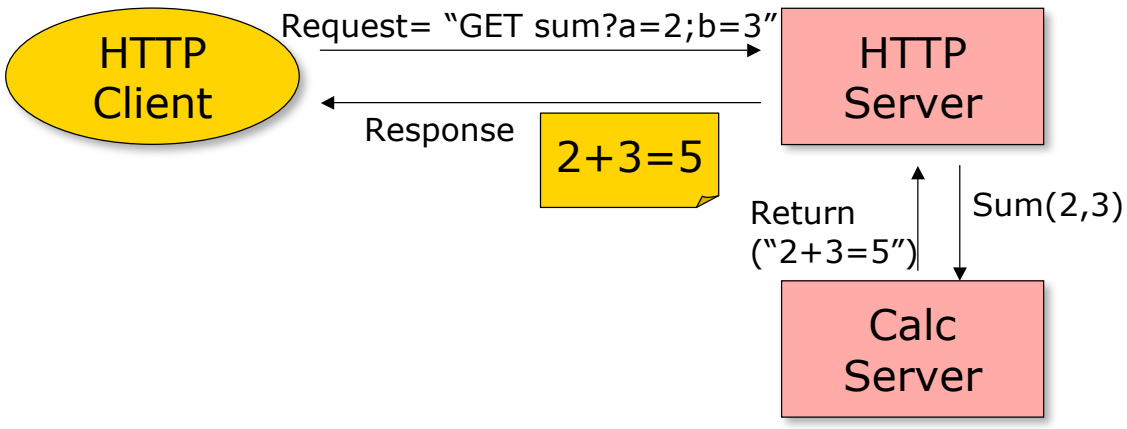
http://myserver/XX.html

Modelli per Applicazioni Web



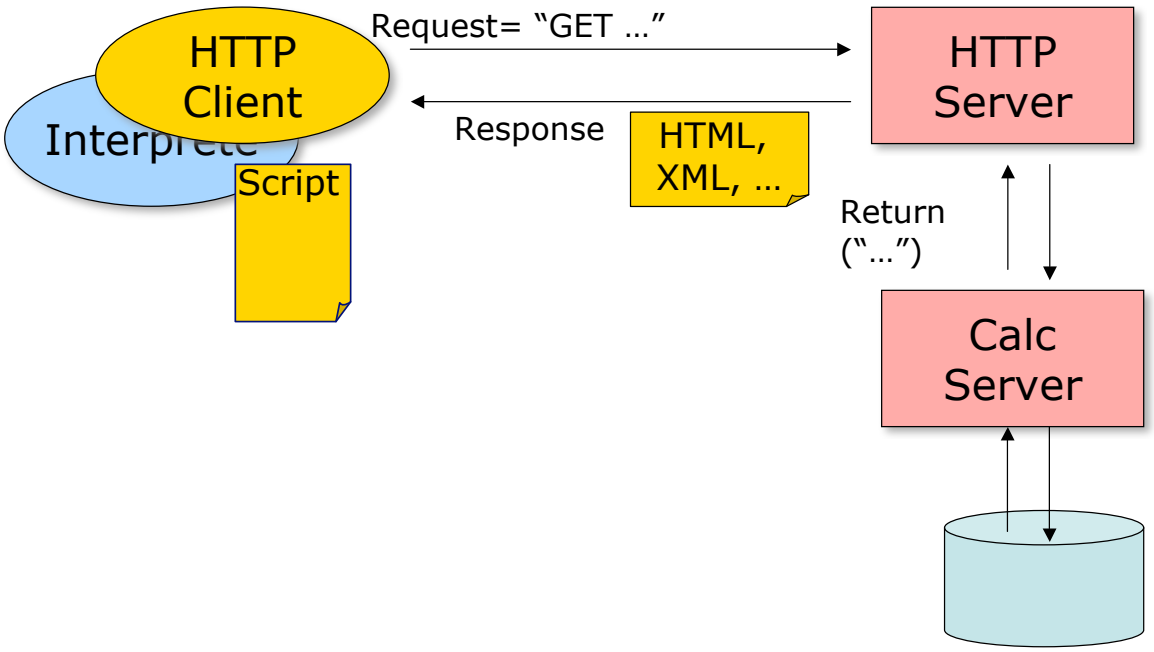
http://myserver/YY.html

Modelli per Applicazioni Web



http://myserver/sum?a=2;b=3

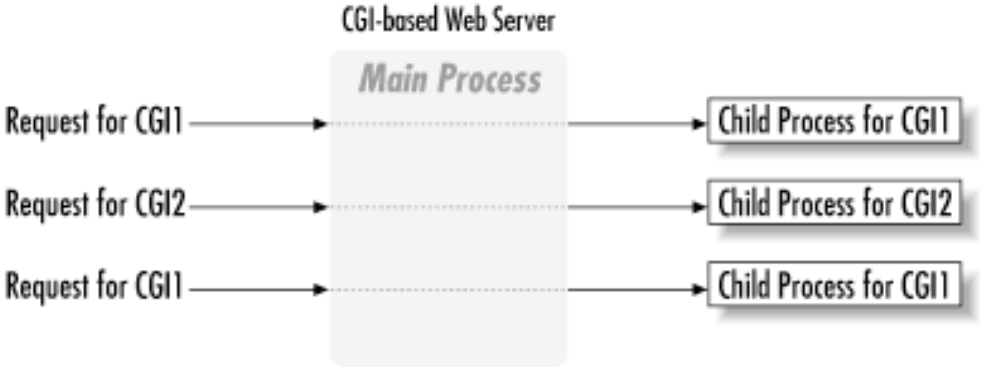
Modelli per Applicazioni Web



Common Gateway Interface (CGI)

- Prima tecnica utilizzata per ottenere contenuti dinamici
- Le CGI creano **pagine Web** come **effetto collaterale**. La tecnologia **CGI** nasce per definire una **modalità di interazione** fra un **web server** ed **applicazioni/script esterni**
- Quando il server riceve una **Request crea** un **nuovo processo** per **eseguire** la **CGI**
- Il sever **passa** al **processo** i **dati** per **generare** la **response** utilizzando **variabili di ambiente** e **standard input**
- **Tecnica** computazionalmente **molto onerosa**

CGI



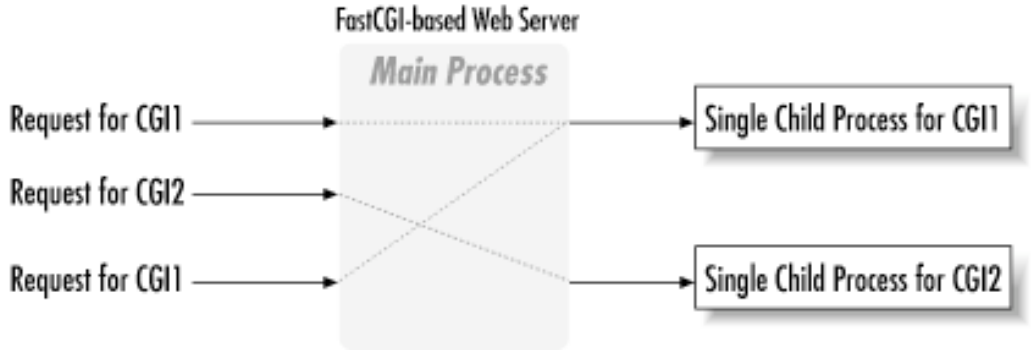
CGI

- Le CGI possono essere implementate in **qualunque linguaggio**
  - C/C++
  - **Perl** (standard de facto per le CGI)
  - Java
- Ulteriore problema dellle CGI è che **una volta iniziato** il **processing** della **Request non** è **possibile interagire** con il **server** (es. per avere visibilità di alcune variabili di ambiente, o scrivere sul file di log)

Fast CGI

- Proposte da Open Market sono una **alternativa** alle **CGI**
- Viene **creato** un **processo singolo** e **persistente per ogni CGI** riducendo sensibilmente il carico computazionale
- Vari problemi:
  - Per **richieste concorrenti** devo **predisporre** dei **pool** di **processi** per cui la soluzione **non** è **scalabile**
  - Resta il **problema** del **disaccoppiamento** con il **web server**

Fast CGI



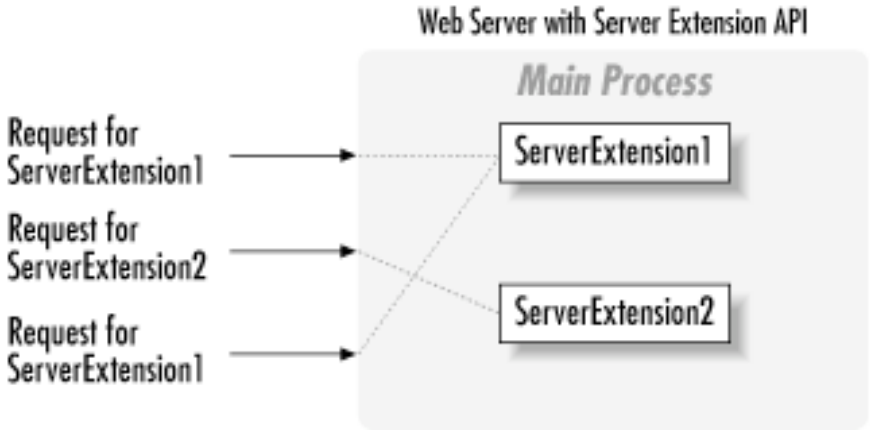
Altre Soluzioni

- **PerlEx**
  - Sviluppato da Active State **migliora l'esecuzione** di **CGI** in **ambienti Microsoft**.
  - Modello **analogo** a **Fast CGI**
- **mod_Perl**
  - Soluzione per il **web server Apache**
  - **Integra** nel **server** un **interprete Perl**
  - **Riduce** fortemente il **carico** sul server
  - Funziona però solo sul server Apache

Server Extension API

- Vari sforzi sono stati indirizzati verso lo sviluppo di server extension API per i Web Server. Le server extension **consentono** al **server** di **eseguire azioni** che prima dovevano essere eseguite da applicazioni esterne
  - iPlanet/Netscape **WAI**
  - Microsoft **ISAPI**
- Le **server extension** fanno parte del **processo** del **Web server**
  - Soluzione **efficiente**
  - **Problemi** nella **manutenzione** da parte dei fornitori
  - **Problemi** di **sicurezza**
  - **Problemi** nella **reliability** dei sistemi. Se una **applicazione** basata su server extension "**cade**", allora "cade" **l'intero web server**

Server Extension API



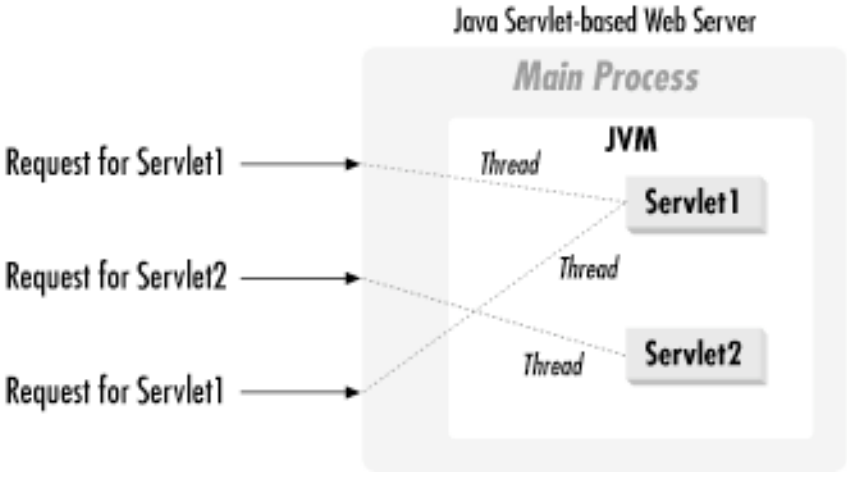
Le Servlet

- Le Servlet sono **simili** alle **server extension API** ma risolvono diversi problemi di queste tecnologie
- Una **Servlet** è una **applicazione** che **estende** le **funzionalità** di un **Web Server**
  - **Genera pagine** con **contenuti dinamici**
  - **Interagisce** con i **client** (tipicamente il browser dell'utente) in accordo ad un modello request/response
- Le **servlet** vengono **eseguite** da un **servlet container**
  - Apache Tomcat
  - Jetty
  - Sun GlassFish
  - …

Le Servlet

- Sono **eseguite** da una **Java Virtual Machine nel server**
- Le Servlet **eseguono** su **thread separati**
- **Risolvono problemi** di **sicurezza** (modello di sicurezza Java)
- Sono **portabili**
- **Possono interagire** con il **Web Server**

Le Servlet



Le Servlet

- Il ruolo delle servlet
  - **Ricevere richieste HTTP** da parte dei client
    - Normalmente GET, POST
  - **Generare dinamicamente** i contenuti delle **pagine**
    - Integrazione di sistemi legacy
    - Interrogazione a data-base
    - …
  - **Restituire** la **HTTP Response** al client

Cosa sono le Servlet?

- Le servlet sono oggetti Java che estendono le funzionalità di un server HTTP
- Le servlet **vengono mappate in URL**
- Le servlet eseguono all'interno di un servlet container
- La tecnologia delle servlet è matura e la soluzione è portabile fra **diversi servlet container** che eseguono **su diversi sistemi operativi**

Esempio Servlet

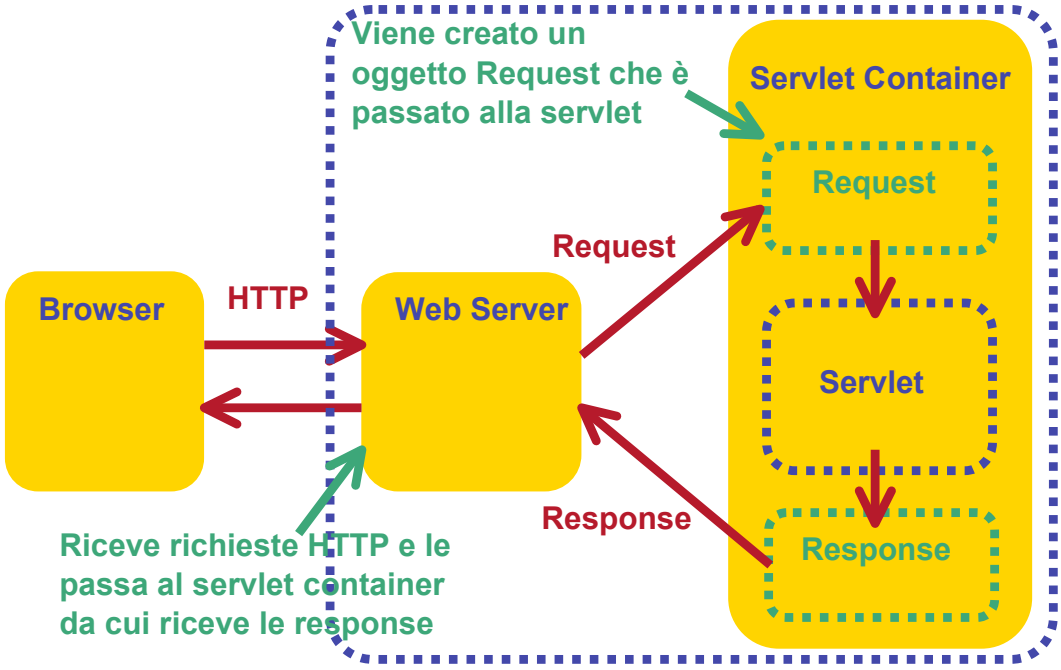
Estende la classe HttpServlet

```
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
                  HttpServletResponse  response){
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<title>Hello World!</title>");
  }
… }
```

Si noti che non è stato implementato il metodo main. La servlet viene infatti eseguita dal servlet container ed implementa la logica per rispondere alle HTTP Request dei client

HttpServlet implementa vari metodi a cui possiamo fare overring. La servlet nell'esempio implementa la logica di risposta ad una HTTP GET

Modello Request-Response delle Servlet

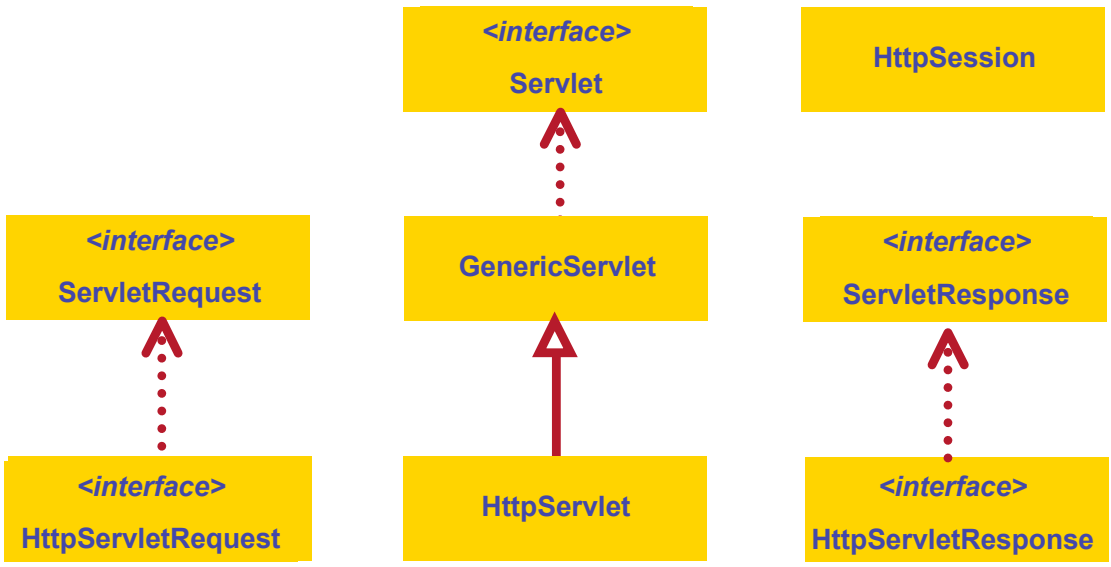


Spesso (ma non necessariamente) web server e servlet container eseguono sulla macchina (a volte eseguono nella stessa JVM)

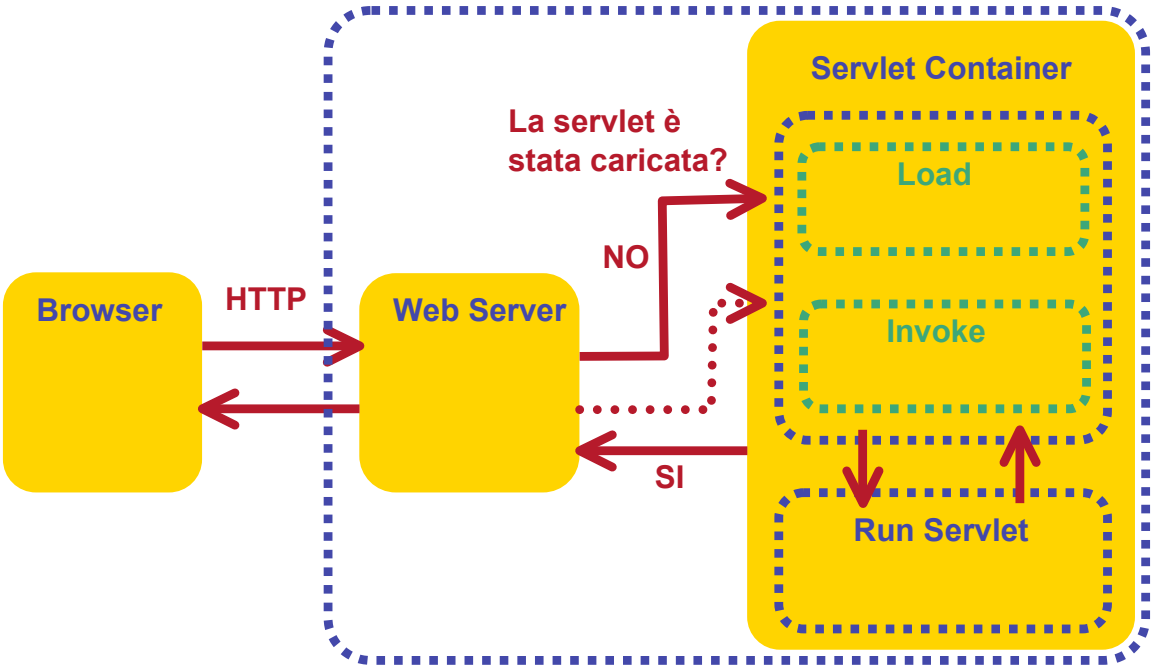
Request e Response

- Le **Request rappresentano** la **chiamata** al server effettuata dal client. Le Request sono caratterizzate da varie informazioni
  - **Chi ha effettuato la Request**
  - Quali **parametri** sono stati **passati** nella **Request**
  - Quali **header** sono stati passati
- Le Response **rappresentano** le **informazioni restituite** al client in **risposta** ad una Request
  - Dati in **forma testuale** (es. html, text) o **binaria** (es. immagini)
  - **HTTP headers, cookies,** …

Sevlet: Classi ed Interfacce



Ciclo di Vita delle Servlet



Ciclo di Vita delle Servlet

- Il **servlet container controlla** il **ciclo di vita** della **servlet**. In particolare:
- **Se non esiste** una **istanza** della **servlet** nel container
  - Il **container carica la servlet**
  - **Crea una istanza** della classe della servlet
  - **Inizializza la servlet** (metodo **init()**)
- **Invoca la servlet** passando come parametri gli oggetti **HttpServletRequest** ed **HttpServletResponse**

Metodi per il Controllo del Ciclo di Vita delle Servlet

- **Init():** chiamato **una sola volta** al **caricamento** della servlet per il suo **setup** (es. connessione con un data-base)
- **Service(): chiamato** ad **ogni HTTP Request**
  - Il metodo **service**() chiama **doGet**() o **doPost**() a seconda della HTTP Request ricevuta
- **Destroy():** viene **chiamato una sola volta** quando la servlet deve essere **disattivata** (es. quando è rimossa). Tipicamente serve per **rilasciare le risorse acquisite** (es. connessione ad un data-base)

Il **servlet container invoca** questi **metodi** ed è **suo onere** il **controllo** del **ciclo di vita** della **servlet**

Metodi per il Controllo del Ciclo di Vita delle Servlet

- I metodi **init()**, **destroy()** e **service()** sono definiti nella classe astratta **javax.servlet.GenericServlet**
  - Service() è un metodo astratto (vedi Java Docs)
- La classe **javax.servlet.http.HttpServlet** è una sottoclasse di **javax.servlet.GenericServlet**. Questa classe fornisce una implementazione di service che delega il processing della request ai metodi
  - **doGet()**
  - **doPost()**
  - **doXXX()** (sappiamo che in HTTP abbiamo anche delete, etc.etc.)

Esempio di init()

```
public class CatalogServlet extends HttpServlet {
public void init() throws ServletException {
    bookDB = (BookDB) getServletContext().getAttribute("bookDB");
    if (bookDB == null) throw new UnavailableException(Errore");   }
 ...
 }
```

In uno stesso servlet container possiamo avere varie servlet. È irragionevole pensare di avere una connessione per ogni servlet allo stesso data-base. L'esempio assume che sia stata settata la connessione al di fuori della servlet e che sia stata salvata in un oggetto ServletContext prima della chiamata del metodo init(). Il metodo init() setta la connessione per la servlet utilizzando ServletContext.

Lettura dei Parametri di Configurazione in init()

```
public void init(ServletConfig config) throws  ServletException
   {
   super.init(config);
   String driver = getInitParameter("driver");
   String URL = getInitParameter("url");
   try {openDBConnection(driver, URL);}
   catch (SQLException e) {e.printStackTrace();}
   catch (ClassNotFoundException e){e.printStackTrace();}
}
```

Altro modo per settare la servlet è quello di **utilizzare** i **valori nel descrittore** di **deployment web.xml**

Web.xml

```
<web-app>
<servlet>
<servlet-name>Pippo</servlet-name>
<servlet-class>PippoServlet</servlet-class>
<init-param>
<param-name>driver</param-name>
<param-value>com.derby.core.RmiJdbcDrv</param-value>
</init-param>
<init-param>
<param-name>url</param-name>
<param-value>jdbc:derby:rmi:derbyDB</param-value>
</init-param>

</servlet>
</web-app>
```


Il metodo destroy()

- Si occupa del **rilascio delle risorse** acquisite
- Nell'esempio della connessione al data-base
  - Se ho ottenuto la connessione utilizzando ServletContext posso sfruttare il supporto per la garbage collection offerto da Java
  - Se ho acquisito una nuova connessione devo rilasciarla

- La gestione della Garbage Collection in Java è però ancora problematico specialmente quando abbiamo l'esigenza di gestire Applicazioni Web che devono operare in modo continuativo (7x24)

Gestione delle HTTP Request e Response

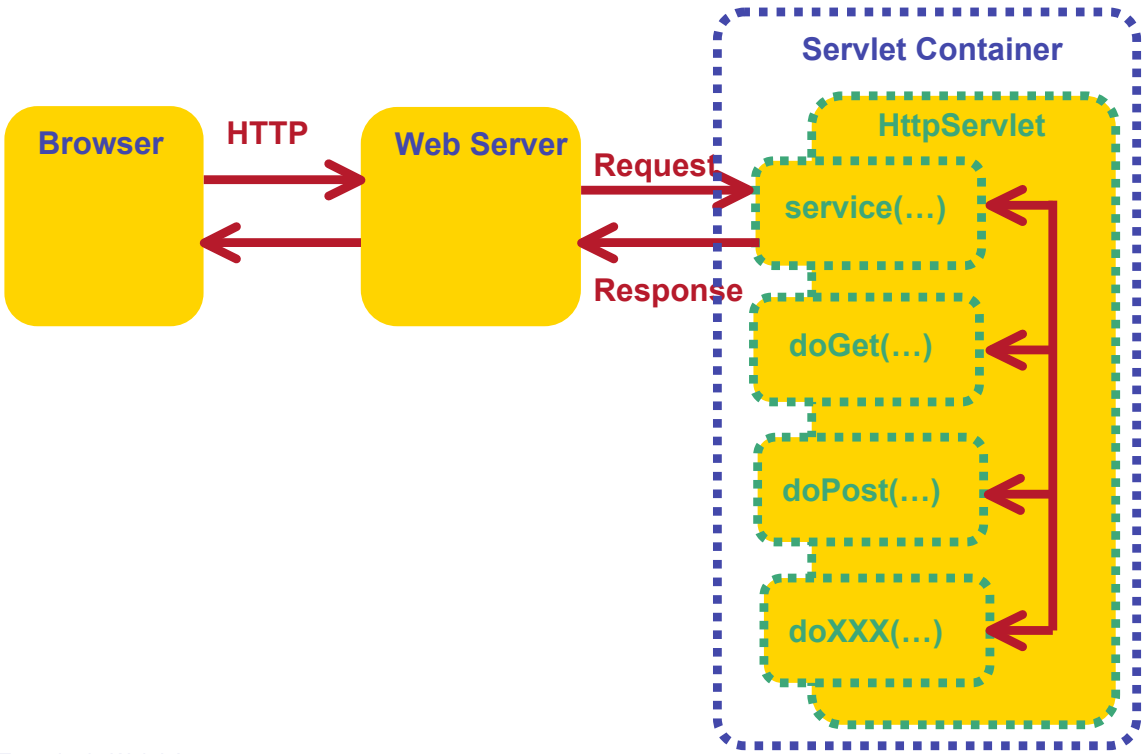
- Il medodo astratto service() della classe GenericServlet è implementato dalle sue sottoclassi.
  - **service(ServletRequest request**
    ServletResponse response)

- HTTPServlet è una sottoclasse di GenericServlet che delega la risposta alle Request del client a diversi metodi
  - **doGet(HttpServletRequest request,**
    HttpServletResponse response)
  - **doPost (HttpServletRequest request,**
    HttpServletResponse response)

  doXXX(…)

  ….
- Lo sviluppatore deve **solo** fare l'**overriding** dei **metodi** di suo interesse

Gestione delle HTTP Request e Response



Come scrivere una doXXX()?

- Estrarre i parametri della query dell'utente
- Settare gli attributi della risposta
- Eseguire la logica applicativa del metodo (es. accedere ad un Data-Base)
- Eventualmente richiedere ulteriori informazioni ad altri componenti della applicazione web
- Preparare il messaggio di response
- Inviare la response

Esempio: doGet()

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
                                                            IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
out.println("<title>First Servlet</title>");
out.println("<big>Hello J2EE Programmers! </big>");
}
}
```

Nell'esempio manca la logica per modificare il contentuto della pagina che restituiamo. Si noti che manca il main

Altro Esempio: doGet() (1)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
                  HttpServletResponse response) throws ServletException,
                                                      IOException {
HttpSession session = request.getSession(true);
ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

response.setContentType("text/html");
response.setBufferSize(8192);
PrintWriter out = response.getWriter();

out.println("<html>" +"<head><title>" + messages.getString("Titolo")+"</title></head>");

RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null) dispatcher.include(request, response);
}}
```

Controlla il session scope di message

Setta header e dimensione del buffer

Prepara un dispatcher che restituisce un banner ottenuto da un altro componente

Altro Esempio: doGet() (2)

```
 // Get the identifier of the book to display (Get HTTP parameter)
String bookId = request.getParameter("bookId");
if (bookId != null) { // and the information about the book (Perform business logic)

try {
    BookDetails bd = bookDB.getBookDetails(bookId);
    Currency c = (Currency)session.getAttribute("currency");
    if (c == null) {
            c = new Currency();
            c.setLocale(request.getLocale());
            session.setAttribute("currency", c);
     }
    c.setAmount(bd.getPrice());
    // Print out the information obtained
    out.println("...");          }
catch (BookNotFoundException ex) {
    response.resetBuffer();
    throw new ServletException(ex);}
}

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

Come si Prepara un HTTP Response

- Settare i Response headers
- Settare le proprietà della response
  - Es. la dimensione del buffer
- Ottenere un oggetto di output stream dalla response
- Scrivere il contenuto del body nell'output stream

Esempio: Semplice Response

```
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
// Fill response headers
response.setContentType("text/html");
// Set buffer size
response.setBufferSize(8192);
// Get an output stream object from the response
PrintWriter out = response.getWriter();
// Write body content to output stream
out.println("<title>First Servlet</title>");
out.println("<big>Hello J2EE Programmers! </big>");  } }
```

Gli Scope Object

- Servono per mantenere informazioni sullo stato della applicazione o sullo stato della servlet.
  - Session
  - ServletContext
  - Request
  - Page
- **Permettono** di **condividere informazioni fra diversi componenti** web sulla base di un meccanismo basato su attributi
  - Gli attributi sono coppie nome/valore
  - Accessibili tramite **getAttribute()** e **setAttribute()**

Scope Object METTERE A POSTO

- Web context (ServletConext): Accessibile dai componenti web nello stesso contesto
  - **javax.servlet.ServletContext**
- Session: accessibile dai componenti web che gestiscono una stessa sessione
  - **javax.servlet.http.HttpSession**
- Request: accessibile dai componenti web che gestiscono la richiesta
  - Sottotipo di javax.servlet.ServletRequest: **javax.servlet.http.HttpServletRequest**
- Page: accessibile dalle JSP che creano la pagina (vedremo nelle prossime lezioni)
  - **javax.servlet.jsp.PageContext**

ServletContext (1)

- **Usato** dalle servlet **per accedere** ad **oggetti sulla base** del loro **nome**
- Le ServletContext hanno **scope limitato** allo **stesso contesto di esecuzione** (tipicamente la stessa web application)
- È **possibile** inoltre **richiedere** alla ServletContext un **dispatcher** per potere
  - **reindirizzare** (forward) la richesta HTTP ad un altro componente Web
  - per **includere** l'output ottenuto da un altro compoente Web

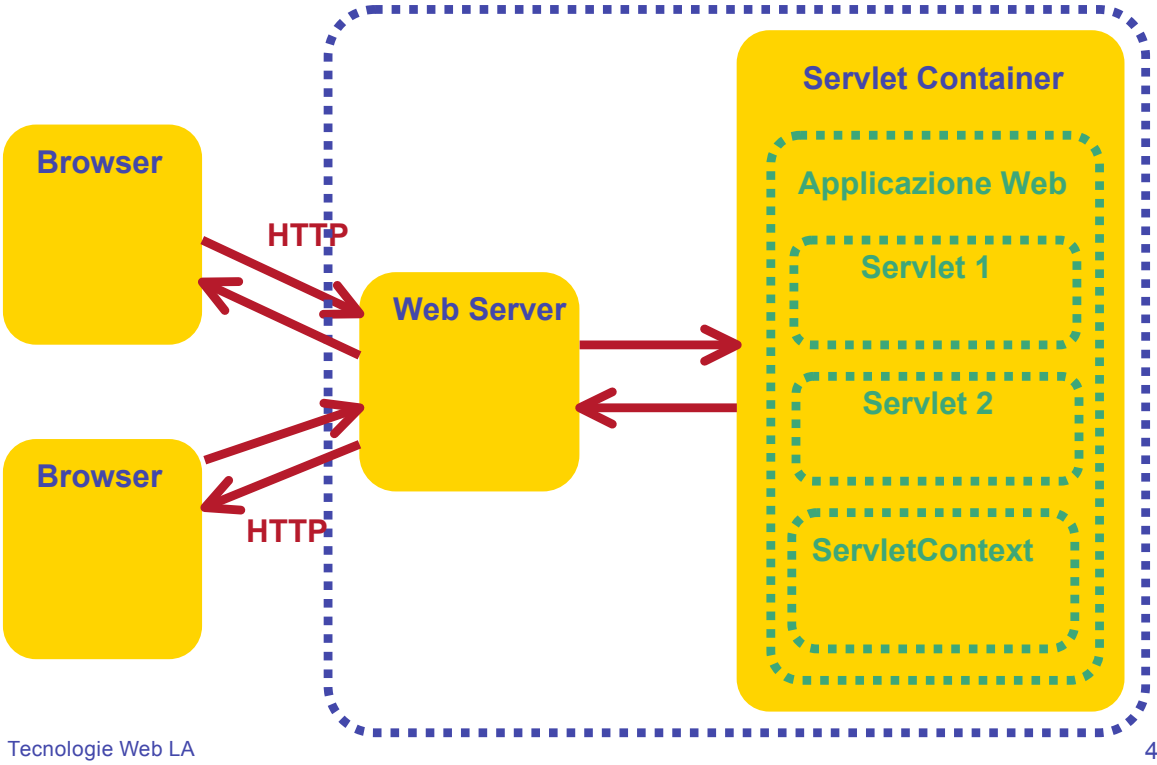
ServletContext (2)

- È possibile **accedere** a **parametri** di **inizializzazione** specificati nel **descrittore** di **deployment web.xml**
- È possibile utilizzare **ServletContext** per **accedere a risorse** web **associate** allo **stesso contesto**
- È possibile **accedere al log** del server e ad altre informazioni di interesse per la web application

Scope della ServletContext

- Abbiamo parlato di contesto web
  - Lo scope è condiviso da tutte le servlet che compongono la medesima Applicazione Web.
  - Lo scope a livello di contesto può essere considerato a livello di applicazione Web
- Una applicazione Web è un insieme di servlet e contenuti che condividono tipicamente uno stesso sottoinsieme di URL e sono installate utilizzando un Web Application Archive (WAR)
- **Esiste un oggetto ServletContext per applicazione Web per Java Virtual Machine**

Scope della ServletContext



Come Accedere all'oggetto ServletContext?

- È sufficiente invocare il metodo **getServletContext()** all'interno del codice della servlet
- **ServletContext** è **contenuto** nell'**oggetto ServletConfig** che viene **fornito dal Web Server alla servlet** quando viene **inizializzata**
  - Invocazione del metodo **init(ServletConfig srvlCfg)**

Torniamo all'Esempio del Data-Base

```
public class CatalogServlet extends HttpServlet {
public void init() throws ServletException {

bookDB = (BookDB) getServletContext().getAttribute("bookDB");

if (bookDB == null) throw new UnavailableException(Errore");   }
 ...
 }
```

Viene reperito il valore dell'attributo bookDB dell'oggetto ServletContext. Esempio di utilizzo tipico nel metodo init() della servlet

L'Oggetto RequestDispatcher

- A volte è **comodo** potere **reindirizzare** le **request** ad altri componenti web
- **RequestDispatcher** consente di **effettuare** il **dispatch** della request **ad** un **altro componente** per **delegare** la **gestione della richiesta** o per **includere il suo output** nella response della servlet
- Il dispatch **richiede** il **passaggio** di **HTTP request** e **response** come parametri

Torniamo ad un Esempio che abbiamo Visto in Precedenza

```
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
                  HttpServletResponse response) throws ServletException,
                                                      IOException {
HttpSession session = request.getSession(true);
ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

response.setContentType("text/html");
response.setBufferSize(8192);
PrintWriter out = response.getWriter();

out.println("<html>" +"<head><title>" + messages.getString("Titolo")+"</title></head>");

RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null) dispatcher.include(request, response);
}}
```

Nome della Servlet

Passo i parametri HttpServletRequest e HttpServletResponse

Esempio: Logging

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException,
                         IOException {
…
getServletContext().log("tutto bene");
…
getServletContext().log("problemi", Eccezione);

}
```

Nome e tipo di log dipendono dal servlet container

HTTP Session

- È lo scope object di utilizzo più frequente nelle applicazioni
- Ci serve un **meccanismo di sessione** per mantenere le **informazioni di stato** di un **utente** per un **determinato periodo** di **tempo anche** a fronte di **diverse request**
  - Es. Shopping cart
- Ricordiamo che **HTTP** è un **protocollo stateless**
- L'oggetto **HttpSession** consente di **mantenere** lo **stato del cliente** in **forma** di **attributi per** la **durata della sessione**

Come Ottenere HttpSession?

- L'oggetto **HTTPRequest** passato a **service()** oppure a **doGet()**, **doPost()**, **doXXX()** fornisce un metodo per l'accesso alla sessione
  - **getSession()**

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
                   throws ServletException, IOException {
…
HttpSession session = request.getSession();
ShoppingCart cart =
   (ShoppingCart)session.getAttribute("cart");
…}
```


Servlet Request & Response

La Servlet Request

- **Contiene** i **dati passati** dal **client alla servlet**
- Tutte le **richieste** alla servlet vengono **implementate** tramite l'interfaccia **ServletRequest** che definisce vari metodi (vedi Java docs)
  - I parametri inviati dal client
  - Attributi
  - Input Stream
  - Informazioni sul protocollo
  - Tipo di contenuto
  - …

Accedere ai Parametri della Request

- Le **request** alle **servlet** tipicamente **includono vari parametri**
- I parametri sono **inseriti dall'utente** nelle **form HTTP**
  - **GET**: parametri in forma di stringa di query nell'URL della request
  - **POST**: i parametri non appaiono nella URL ma sono nel messaggio di request
- I **parametri hanno un nome**
- **getParameter("paraName")**
  - restituisce il valore del parametro con il nome specificato.
  - Restituisce null se il parametro non è presente
  - Si applica si per request GET che per request POST

Esempio di Uso della GET (1)

```
...

<FORM ACTION="/sample/servlet/ThreeParams">
First Name:  <INPUT TYPE="TEXT" NAME="param1"><BR>
Last Name:   <INPUT TYPE="TEXT" NAME="param2"><BR>
Class Name:  <INPUT TYPE="TEXT" NAME="param3"><BR>
<INPUT TYPE="SUBMIT">

...
```

Esempio di form con metodo GET

Esempio di Uso della GET (2)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ThreeParams extends HttpServlet {

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

 out.println("<HTML>" +"<LI><B>First Name in Response</B>:
     "+request.getParameter("param1") + "\n" +
     "  <LI><B>Last Name in Response</B>: "+
     request.getParameter("param2") + "\n"+
     "  <LI><B>NickName in Response</B>: "+
     request.getParameter("param3") + "\n" +"</UL>\n" + "</BODY></HTML>");
} }
```

Se avessimo usato post avremmo dovuto fare l'overriding del metodo doPost(…) ma la logica della applicazione non sarebbe cambiata

Attributi della Request

- Gli attributi della request possono essere settati in due modi
  - Il servlet container può settare alcuni attributi
    - Es. l'attributo X509Certificate può essere settato dal container all'arrivo di request HTTPS
  - La servlet può settare attributi specifici dell'applicazioni
    - void setAttribute(java.lang.String name, java.lang.Object o)
    - Nella request prima di una chiamata al RequestDispatcher

Ottenere Informazioni da Attributi Locale

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

HttpSession session = request.getSession();
ResourceBundle messages = (ResourceBundle) session.getAttribute
                                                ("messages");
 if (messages == null) {
    Locale locale=request.getLocale();
    messages = ResourceBundle.getBundle(
                                "messages.BookstoreMessages", locale);
 session.setAttribute("messages", messages);
 }
```

Ottenere Informazioni dai Client

- La servlet può ottenere informazioni sui client dalla request
  - **String request.getRemoteAddr()** per ottenere l'IP del client
  - **String request.getRemoteHost()** per ottenere l'host name dei client

Ottenere Informazioni sul Server

- La servlet può ottenere informazioni sul Server
  - **String request.getServerName()** per ottenere il nome del server
  - **int request.getServerPort()** per ottenere la porta a cui il server è in ascolto

Ottenere Ulteriori Informazioni

- Input Stream
  - **ServletInputStream getInputStream()**
  - **java.io.BufferedReader getReader()**
- Protocollo
  - **java.lang.String getProtocol()**
- Content Type
  - **java.lang.String getContentType()**
- Connessione HTTP o HTTPS
  - **boolean isSecure()**

HttpServletRequest

- **Contiene** i dati **passati** dal **client** HTTP alla servlet
- **Creata** dal **servlet container** e passata alla servlet come **parametro** ai metodi **doGet(…)** e **doPost(…)**
- **HttpServletRequest** è una **estensione** di **ServletRequest** e fornisce **metodi** per accedere a **varie informazioni**
  - HTTP Request URL
  - HTTP Request header
  - Tipo di autenticazione e informazioni su utente
  - **Cookie**
  - **Session**

HTTP Request URL

http://[host]:[port]/[request path]?[query string]

Una request path è composta da

- Contesto della web application
- Nome della web application
- Path

Esempio

http://daydreamer/catalog/lawn/hello1/greeting

HTTP Request URL

http://[host]:[port]/[request path]?[query string]

- Le query string sono composte da un insieme di parametri che sono forniti dall'utente
- Una stringa di query può apparire in una pagina web

Add To Cart

String bookId = request.getParameter(Add);

- Può essere inclusa nella URL utilizzando la GET

http://localhost/hello1/greg?username=Anna+Bianchi

String userName=request.getParameter("username")

Altri metodi di HttpServletRequest

- **String getContextPath()** per ottenere informazioni sul contesto
- **String getQueryString()** per ottenere la stringa di query
- **String getPathInfo()** per ottenere il path
- **String getPathTranslated()** per ottenere informazioni sul path nella forma reale

HTTP Request Headers

- È possibile ottenere informazioni sugli header della HTTP request
  - **Accept**: tipi MIME che il browser può gestire
  - **Accept-Encoding**: indica il tipo di encoding che il browser può gestire
  - **Authorization**: identificazione dell'utente per le pagine protette da password (approccio non consigliato se non per semplici applicazioni)

HTTP Request Headers

- **Connection**: in HTTP 1.1 il default è quello delle conssioni persistenti. Le Servlet devono settare il ContentLenght per beneficiare delle connessioni persistenti
- **Cookie**: restituisce i cookie che il server ha precedentemente inviato al client
- **Host**: individua lo host

HTTP Request Headers

- **If-Modified-Since**: indica che il client vuole la pagina solo se è stata cambiata dopo una certa data (non utilizzare è meglio implementare getLastModified)
- **Referer**: URL della pagina referente. Comoda per effettuare il tracking del traffico di rete.
- **User-Agent**: stringa che indetifica il browser che ha fatto la request

Metodi per l'Accesso allo HTTP Header in HttpServletRequest

- **String getHeader(java.lang.String name)** per ottenenere il valore della request header come una stringa
- **java.util.Enumeration getHeaders(java.lang.String name)** valore della header specificata nella request
- **java.util.Enumeration getHeaderNames()** nomi della request header
- **int getIntHeader(java.lang.String name)** valore della request header specificata come un intero

Autenticazione e Sicurezza

- **String getRemoteUser()** nome dello user se la servlet è protetta da password, null altrimenti
- **String getAuthType()** nome dello schema di autenticazione usato per proteggere la servlet
- **boolean isUserInRole(java.lang.String role)** restituisce true se l'utente è associato al ruolo specificato
- **String getRemoteUser()** login dell'utente che ha effettuato la request, null altrimenti

Accedere ai Cookie nella HttpServletRequest

- **Cookie[] getCookies()** resituisce un array di oggetti cooki che il client ha inviato alla request

Cosa è una Servlet Response?

- Contiene i dati passati dalla Servlet al Client
- Tutte le response di una servlet implementano l'interfaccia ServletResponse che specifica vari metodi
  - Ottenere un output stream
  - Indicare il content type
  - Indicare se l'output è bufferizzato
  - …
- **HttpServletResponse** estende **ServletResponse** e specifica **ulteriori metodi**
  - **Satus code** della HTTP response
  - **Cookie**

HttpServletResponse e status

- Per definire lo status code la HttpServletResponse fornisce il metodo

  public void setStatus(int statusCode)

- Status Code
  - 200 OK
  - 404 Page not found
  - …
- Per inviare errori possiamo anche usare
  - **public void sendError(int sc)**
  - **public void sendError(int code, String message)**
    - Vedere javadoc

Gestione degli Header in HttpServletResponse

- Gli header HTTP consentono di
  - Specificare i cookie
  - Forniscono eventualmente il supporto per la redirezione delle request
  - Forniscono la data dell'ultima modifica delle pagine
  - Forniscono la dimensione del file per facilitare l'uso di connessioni persistenti
  - …

Metodi per Impostare Response Header Arbitrari

- **public void setHeader(String headerName, String headerValue)** imposta un header arbitrario
- **public void setDateHeader(String name, long millisecs)** imposta la data
- **public void setIntHeader(String name, int headerValue)** evita di dovere convertire le gli interi in stringhe prima di invocare setHeader(…)
- **addHeader, addDateHeader, addIntHeader** aggiungono una nuova occorrenza dello header

Metodi per Settare Response Header di Uso Comune

- **setContentType** determina il content-type. (Usare sempre)
- **setContentLength** utile per la gestione di connessioni persistenti
- **addCookie** consente di aggiungere un valore all'header set cookie
- **sendRedirect** setta la location header e cambia lo status code

Esempio

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

res.setContentType("text/plain");
PrintWriter out = res.getWriter();
 res.setHeader("Refresh", "5"); // refresh della pagina ogni 5 secondi
out.println(new Date().toString());
 }
```


Il Response Body

- Le servlet devono definire il response body
- Possiamo operarare in due modi
  - **PrintWriter**
    - Si usa la response.getWriter()
    - Utile per output a carattere
  - **ServletOutputStream**
    - Si usa response.getOutoutStream()
    - Utile per formati binari (es. immagini)

Includere Altre Risorse Web o Effettuare il Forwarding

Quando includere altre risorse web?

- Includere risorse web può essere **utile** quando **vogliamo aggiungere contenuti** (statici o dinamici) **creati** da un'**altra risorsa** (es. un'altra servlet)
  - Es. Aggiungere un banner
- Vari tipi di risorsa
  - **Statica**: es. includiamo un'altra pagina nella nostra
  - **Dinamica**: la servlet inoltra una request ad un componente web che la elabora e restituisce il risultato
  - Il **risultato** viene **incluso** nella **pagina prodotta** dalla **servlet**
    - La risorsa inclusa può lavorare con il response body ma ci sono problemi con i cookie

Come Includere un'Altra Risorsa Web?

- Per includere una risorsa invocare il metodo RequestDispatcher

RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/aaa");

Invocare il metodo include di RequestDispatcher passando request e response

dispatcher.include(request, response);

Forwarding ad un'Altra Risorsa Web

- Si usa in situazioni in cui un **componente** web **si occupa** di **parte** del **processing della request** ed è **delegato** ad un **altro** la **gestione** della **risposta**
- **Se è stato fatto** un accesso a **ServletOutputStream** o **PrintWriter** nella servlet si ottiene una **IllegalStateException**

Come Effettuare il Forwarding?

- Si deve ottenere l'oggetto RequestDispatcher da HttpServletRequest

RequestDispatcher dispatcher = request.getRequestDispatcher("/pippo");

Invocare il metodo forward() di RequestDispatcher passando request e reponse

dispatcher.forward(request, response);

Se l'URL originale è necessaria per qualche ragione può essere salvata come un attibuto di request

Esempio

```
…
public void doGet(HttpServletRequest request,
   HttpServletResponse response) {
   request.setAttribute("attributo",
   request.getServletPath());
   RequestDispatcher dispatcher = request.
   getRequestDispatcher("/pippo");
   if (dispatcher != null)
     dispatcher.forward(request, response);
}
…
```

Redirezione del Browser

- Possiamo lavorare in due modi
  - Modo 1

    public void sendRedirect(String url)
  - Modo 2

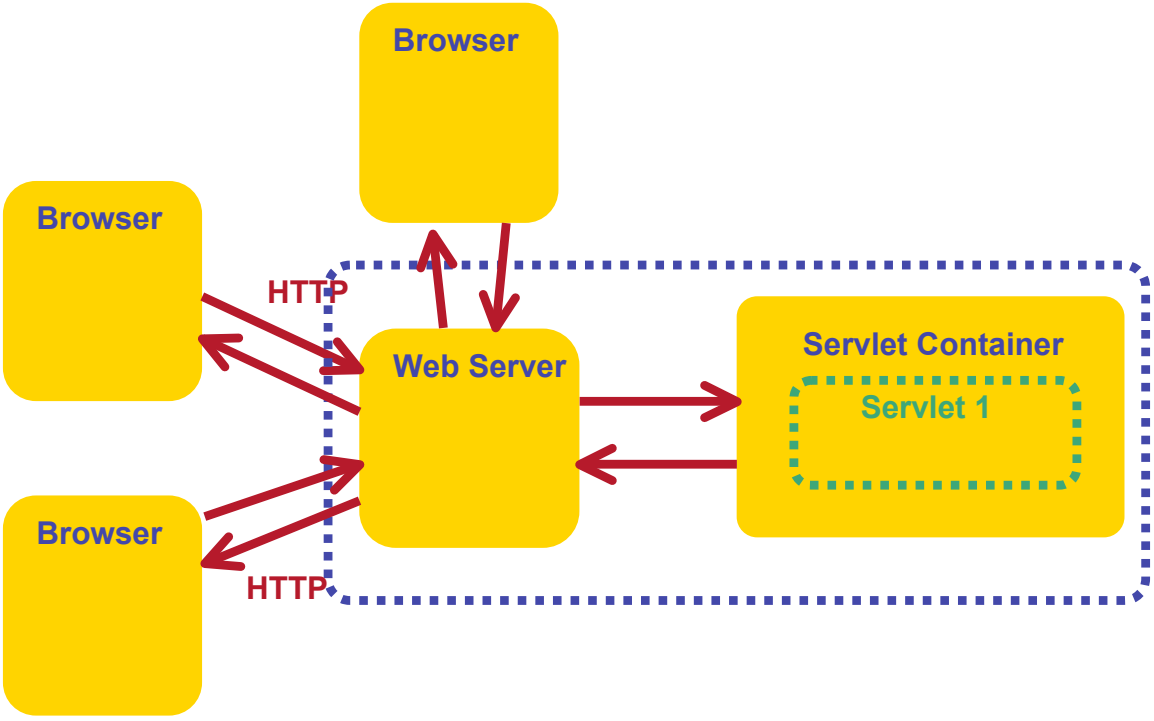
response.setStatus(response.SC_MOVED_PERMANTLY);
response.setHeader("Location", "http://...");

Sincronizzazione delle Servlet e Thread Model

Servlet e Concorrenza

- Il **metodo service** di una **servlet** può essere **invocato** da **molteplici client** in **modo concorrente**
- È **necessario** perciò **gestire** l'**accesso concorrente** alla servlet
  - I **dati condivisi devono essere protetti**
- In **casi molto semplici** e per **prototipazione rapida** di sistemi possiamo adottare l'approccio **SingleThreadModel**
- **In generale** però **dobbiamo gestire** in modo opportuno la logica di **sincronizzazione**

Servlet e Concorrenza



Servlet e Concorrenza

- Il metodo **init** della servlet sarà **chiamato una sola volta** quando la servlet è caricata dal web container
- I metodi **service**() e **destroy**() potranno essere **chiamati solo dopo** il completamento **dell'esecuzione** di **init**()
- Il metodo **service**() **può essere invocato** da **numerosi client** in modo **concorrente** ed è quindi **necessario gestire** le **sezioni critiche**
  - Uso di **blocchi synchronized**
  - **Semafori**
  - **Mutex**

Interfaccia SingleThreadModel

- Le servlet possono implementare l'interfaccia

  javax.servlet.SingleThreadModel
- Il **server gestirà** un **pool di istanze** della **servlet**
- Il **servlet container garantisce** che **esegua un solo thread per istanza** della **servlet**
  - Questo metodo è **oneroso**
  - Il client deve attendere la risposta della servlet per lungo tempo
  - Usare solo per **prototipazione rapida** di sistemi o per **piccoli progetti**

SingleThreadModel