

web.xml (API 4.0)

- È un file di configurazione (in formato XML) che contiene una serie di elementi descrittivi
 - Descrive la struttura dell'applicazione Web
- Contiene l'elenco delle Servlet e per ognuna di loro permette di definire una serie di parametri come coppie nome-valore
 - Nome
 - Classe Java corrispondente
 - Una serie di parametri di configurazione (coppie nome-valore, valori di inizializzazione)
 - Contiene mappatura fra URL e Servlet che compongono l'applicazione **IMPORTANTE!**
- **Dalla versione 3.0 è possibile utilizzare le annotazioni:**
 - `@WebServlet`
 - `@ServletFilter`
 - `@WebListener`
 - `@WebInitParam`

Mappatura Servlet-URL

- Esempio di descrittore con mappatura:

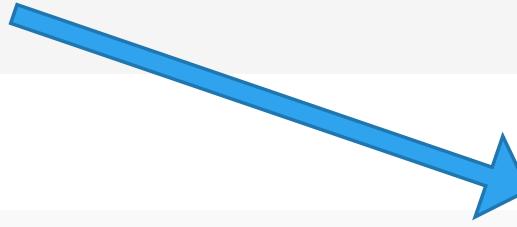
```
<web-app>
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>myPackage.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myURL</url-pattern>
  </servlet-mapping>
</web-app>
```

- Esempio di URL che viene mappato su myServlet:

```
http://MyHost:8080/MyWebApplication/myURL
```

Mappatura Servlet-URL (2)

```
<servlet>
    <servlet-name>testServlet</servlet-name>
    <servlet-class>it.unit.servlet.MyTestServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>testServlet</servlet-name>
    <url-pattern>/admin/testServlet</url-pattern>
</servlet-mapping>
```



`http://localhost:8080/MyWebApplication/admin/testServlet`

Servlet configuration

- Una Servlet accede ai propri parametri di configurazione mediante l'interfaccia **ServletConfig**
- Ci sono 2 modi per accedere a oggetti di questo tipo:
 1. Il parametro di tipo **ServletConfig** passato al metodo **init()**
 2. il metodo **getServletConfig()** della Servlet, che può essere invocato in qualunque momento
- **ServletConfig** espone un metodo per ottenere il valore di un parametro in base al nome:
 - **String getInitParameter(String parName)**

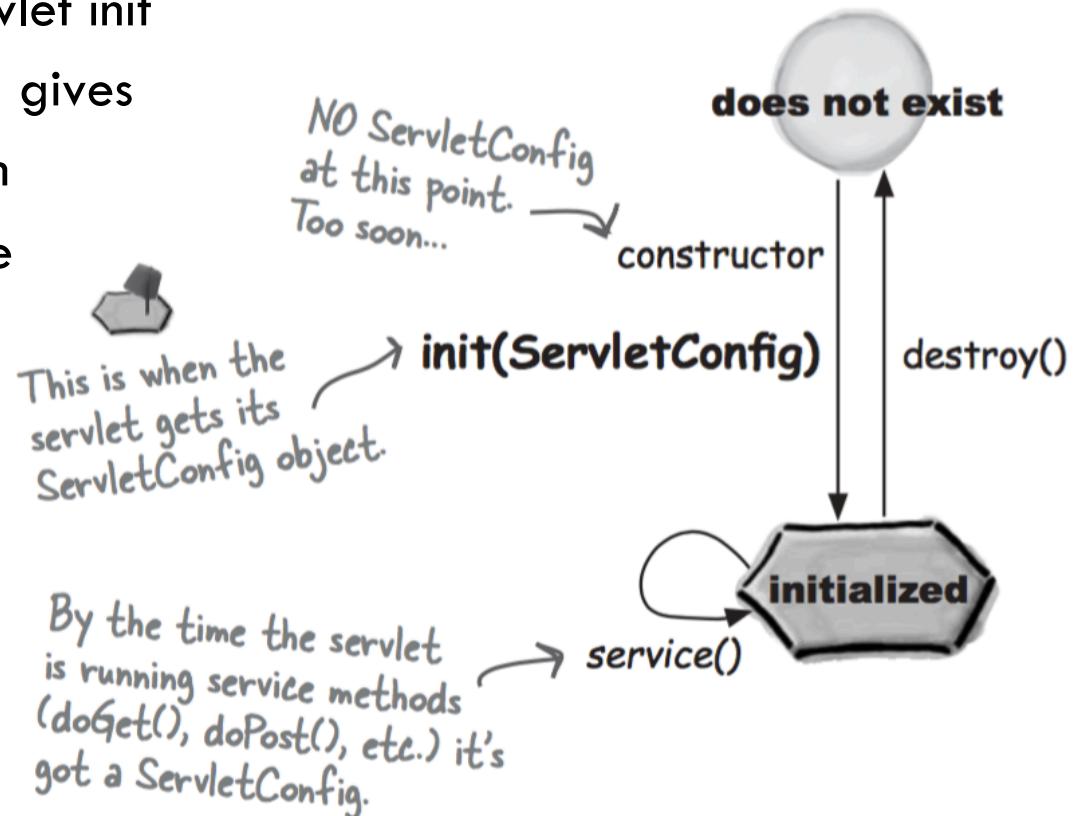
Esempio di parametro
di configurazione

```
<init-param>
    <param-name>parName</param-name>
    <param-value>parValue</param-value>
</init-param>
```

- Es: **getServletConfig().getInitParameter("parName")**

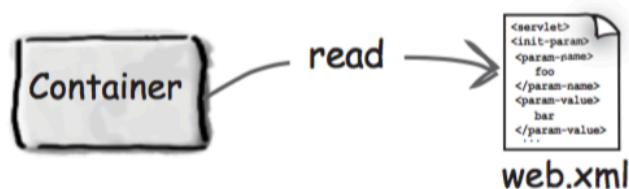
You can't use Servlet init parameters until the Servlet is initialized

- When the Container initializes a Servlet, it makes a unique ServletConfig for the Servlet
- The Container “reads” the Servlet init parameters from web.xml and gives them to the ServletConfig, then passes the ServletConfig to the servlet’s init() method

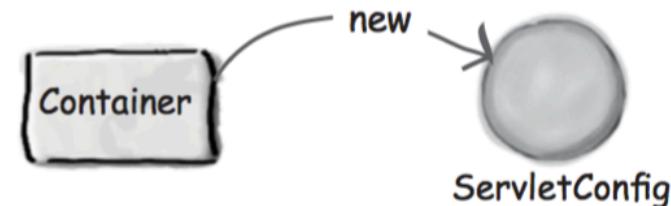


The Servlet init parameters are read only ONCE

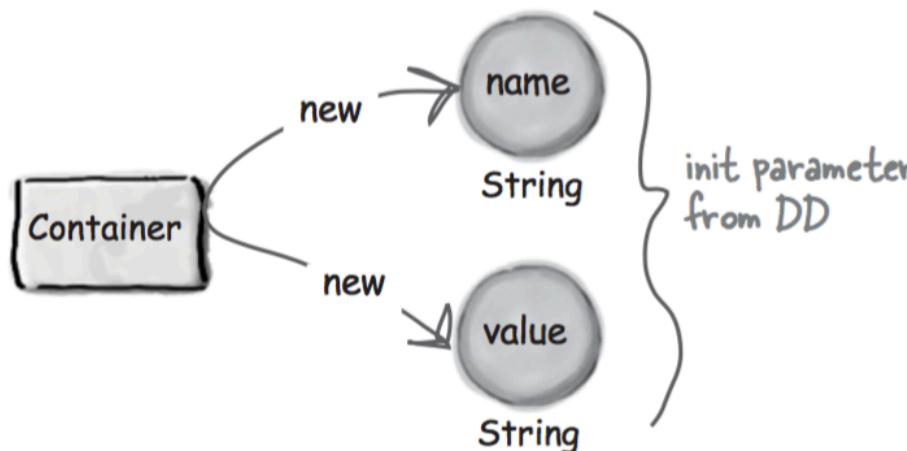
- ① Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).



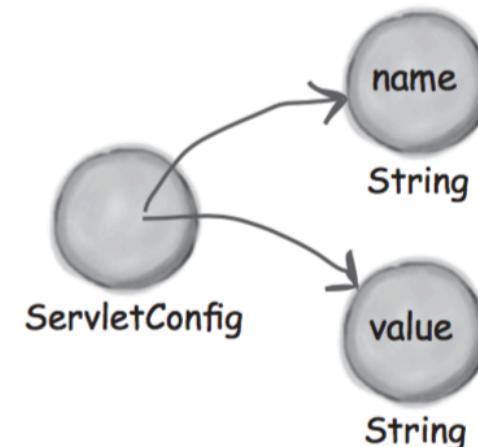
- ② Container creates a new ServletConfig instance for this servlet.



- ③ Container creates a name/value pair of Strings for each servlet init parameter. Assume we have only one.

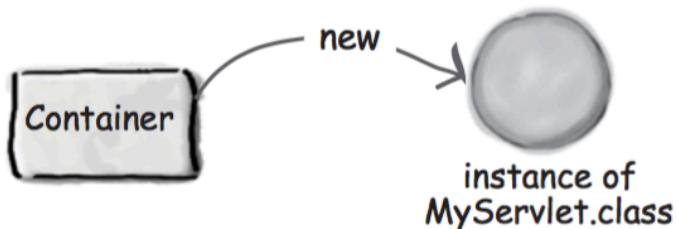


- ④ Container gives the ServletConfig references to the name/value init parameters.

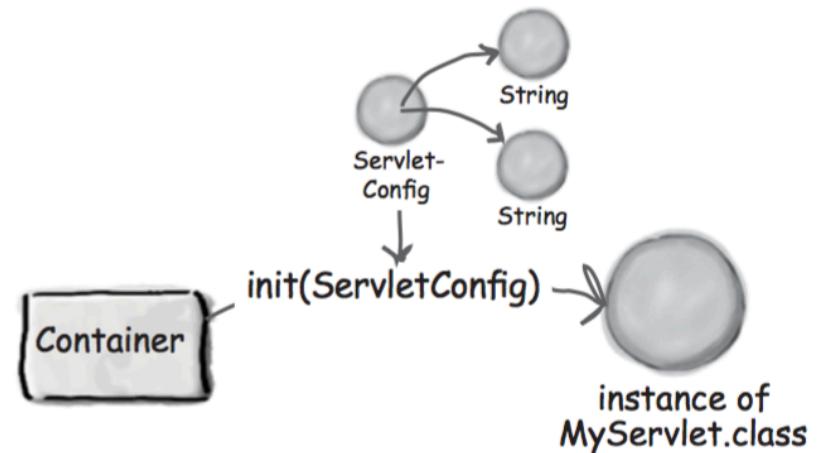


The Servlet init parameters are read only ONCE (2)

- ⑤ Container creates a new instance of the servlet class.



- ⑥ Container calls the servlet's init() method, passing in the reference to the ServletConfig.



Esempio di parametri di configurazione

- Estendiamo il nostro esempio rendendo parametrico il titolo della pagina HTML e la frase di saluto:

```
<web-app>
  <servlet>
    <servlet-name>HelloServ</servlet-name>
    <servlet-class>HelloServlet</servlet-class>
    <init-param>
      <param-name>title</param-name>
      <param-value>Hello page</param-value>
    </init-param>
    <init-param>
      <param-name>greeting</param-name>
      <param-value>Ciao</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloServ</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

```
@WebServlet(name = "/HelloServ", urlPatterns = { "/hello" },
  initParams = {@WebInitParam(name = "title", value = "Hello Page"),
    @WebInitParam(name = "greeting", value = "Ciao") })
```

HelloServlet parametrico

- Ridefiniamo quindi anche il metodo init(): memorizziamo i valori dei parametri in due attributi

```
import java.io.*  
import java.servlet.*  
import javax.servlet.http.*;  
  
public class HelloServlet extends HttpServlet  
{  
    private String title, greeting;  
  
    public void init(ServletConfig config)  
        throws ServletException  
    {  
        super.init(config);  
        title = config.getInitParameter("title");  
        greeting = config.getInitParameter("greeting");  
    }  
    ...
```

Il metodo doGet() con parametri

http://.../hello?to=Mario

Notare l'effetto della
mappatura tra l'URL hello e la servlet

```
public void doGet(HttpServletRequest request,  
    HttpServletResponse response)  
throws ServletException, IOException  
{  
    String toName = request.getParameter("to");  
    response.setContentType("text/html");  
    PrintWriter out = response.getWriter();  
    out.println("<html>");  
    out.println("<head><title>"+title+"</title></head>");  
    out.println("<body>"+greeting+" "+toName+"!</body>");  
    out.println("</html>");  
}
```

HTTP/1.1 200 OK
Content-Type: text/html
<html>
<head><title>Hello page</title></head>"
<body>Ciao Mario!</body>"
</html>"

Pagine di errore (Cercabirra V4)

```
<error-page>
    <error-code>404</error-code>
    <location>/commons/redirectToError.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/commons/fatalError.jsp</location>
</error-page>
```

- Si possono gestire anche le eccezioni:

```
<error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/error.html</location>
</error-page>
```

- Dalle Servlet 3.0:

```
<error-page>
    <location>/general-error.html</location>
</error-page>
```

Servlet context

- Ogni Web application esegue in un **contesto**:
 - corrispondenza 1:1 tra una Web application e suo contesto
- L'interfaccia **ServletContext** è la vista della Web application (del suo contesto) da parte della Servlet
- Si può ottenere un'istanza di tipo ServletContext all'interno della Servlet utilizzando il metodo **getServletContext()**
 - Consente di accedere ai parametri di inizializzazione e agli attributi del contesto
 - Consente di accedere alle risorse statiche della Web application (es. immagini) mediante il metodo **InputStream getResourceAsStream(String path)**
- **IMPORTANTE:** *Servlet context viene condiviso tra tutti gli utenti, le richieste e le Servlet facenti parte della stessa Web application*

Parametri di inizializzazione del contesto

- Parametri di inizializzazione del contesto definiti all'interno di elementi di tipo **context-param** in web.xml

```
<web-app>
  <context-param>
    <param-name>feedback</param-name>
    <param-value>feedback@deis.unibo.it</param-value>
  </context-param>
  ...
</ web-app >
```

- Sono accessibili a tutte le Servlet della Web application

```
...
ServletContext ctx = getServletContext();
String feedback =
ctx.getInitParameter("feedback");
...
```

Attributi di contesto

- Gli attributi di contesto sono accessibili a tutte le Servlet e funzionano come variabili “*globali*”
- Vengono gestiti a runtime:
 - possono essere creati, scritti e letti dalle Servlet
 - Possono contenere oggetti anche complessi (serializzazione/deserializzazione)

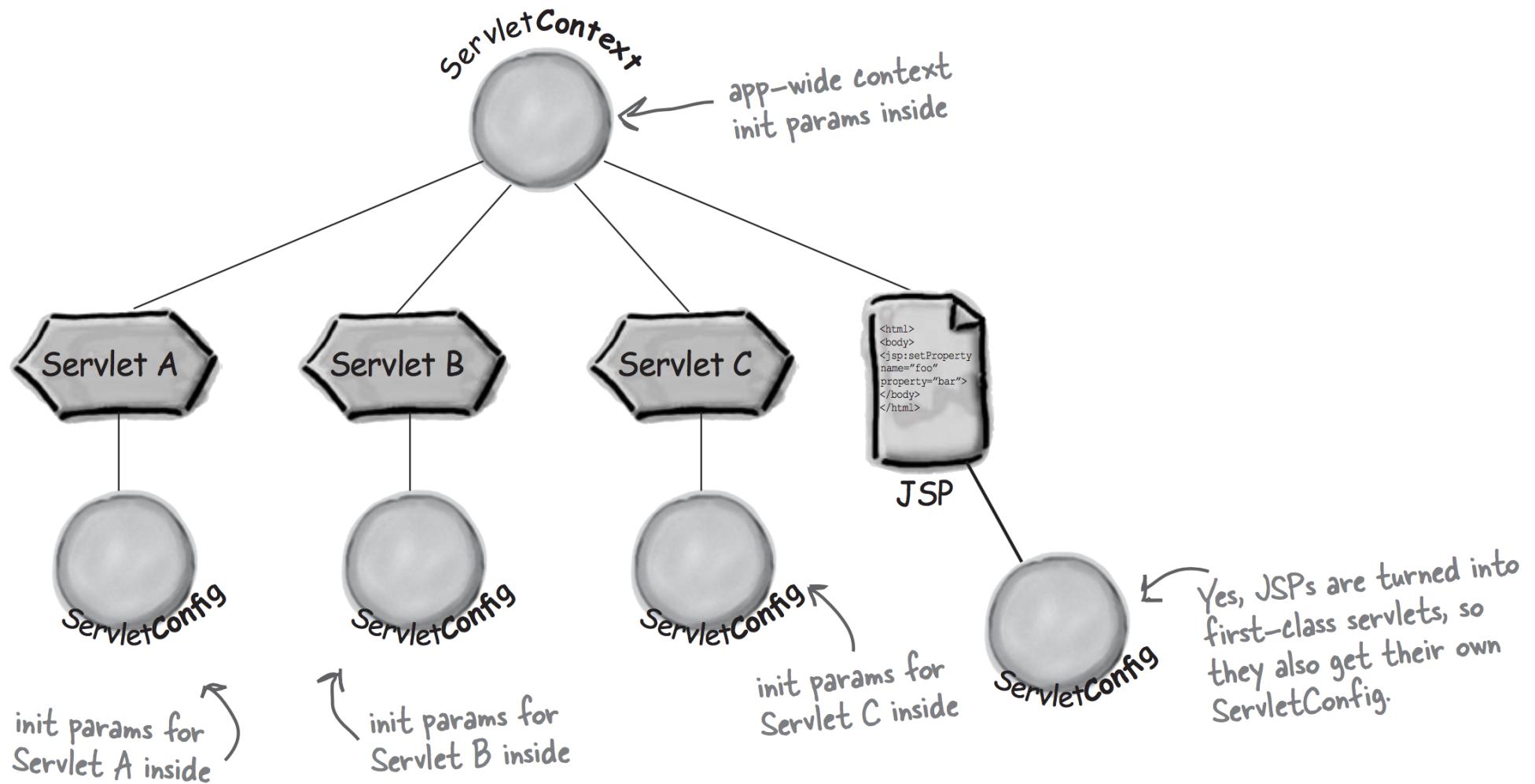
scrittura

```
ServletContext ctx = getServletContext();
ctx.setAttribute("utente1", new User("Giorgio Bianchi"));
ctx.setAttribute("utente2", new User("Paolo Rossi"));
```

lettura

```
ServletContext ctx = getServletContext();
Enumeration aNames = ctx.getAttributeNames();
while (aNames.hasMoreElements())
{
    String aName = (String)aNames.nextElement();
    User user = (User) ctx.getAttribute(aName);
    ctx.removeAttribute(aName);
}
```

ServletContext and ServletConfig



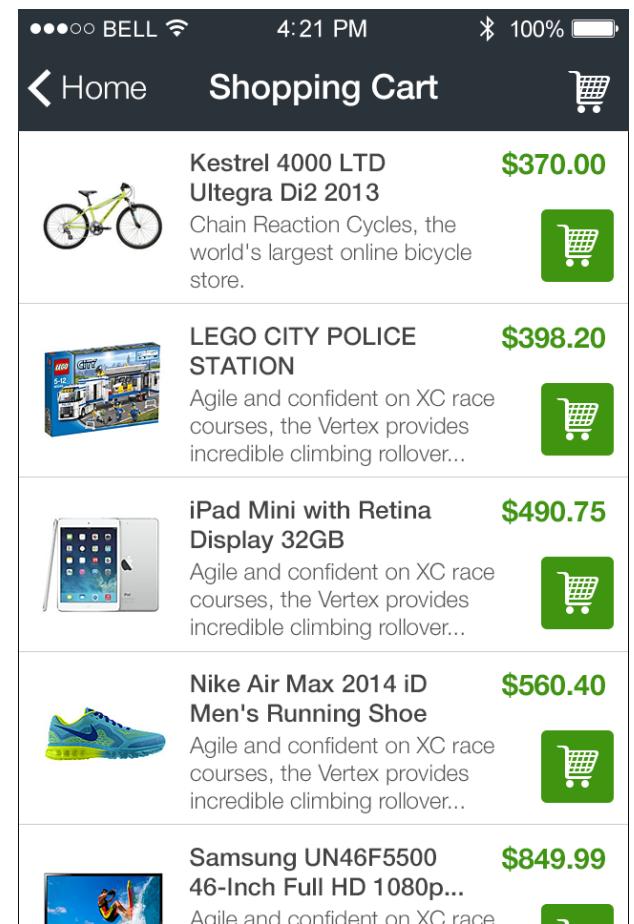
Gestione dello stato (di sessione)

- **HTTP è un protocollo stateless:** non fornisce in modo nativo meccanismi per il mantenimento dello stato tra diverse richieste provenienti dallo stesso client
- Le applicazioni Web hanno spesso bisogno di stato. Sono state definite due tecniche per mantenere traccia delle informazioni di stato:
 1. **uso dei cookie: meccanismo di basso livello**
 2. **uso della sessione (session tracking): meccanismo di alto livello**
- La sessione rappresenta un'utile astrazione ed essa stessa può far ricorso a due meccanismi base di implementazione:
 1. Cookie
 2. URL rewriting

Session Tracking and e-Commerce

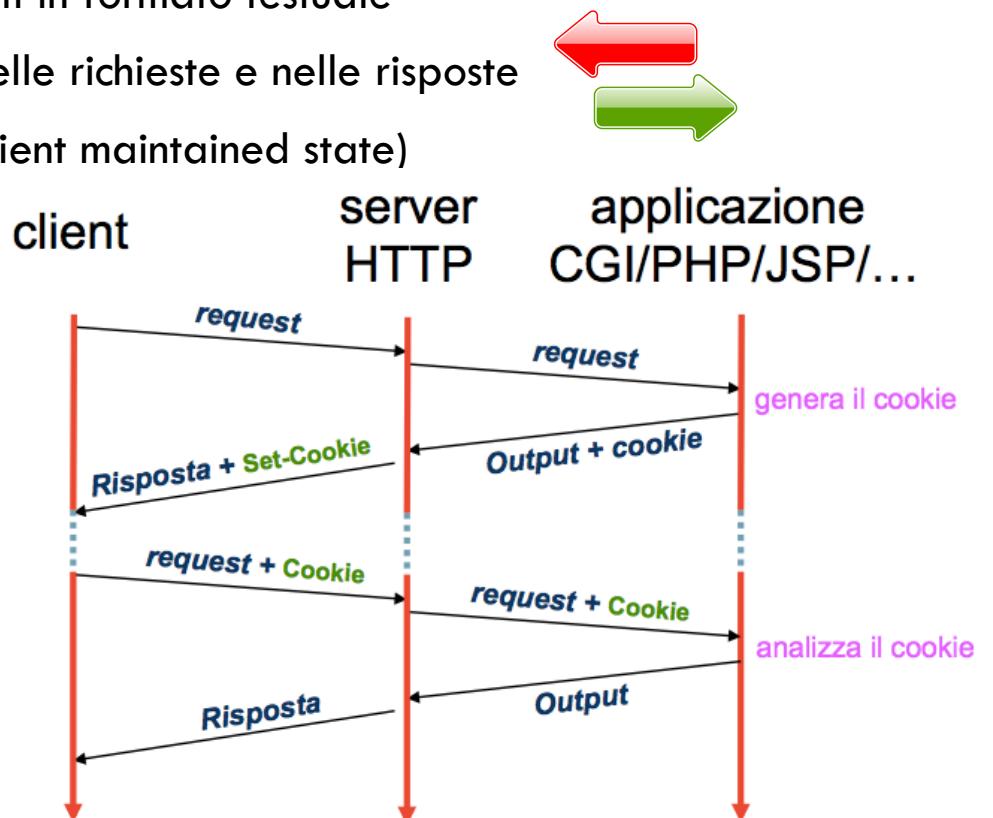
- **Why session tracking?**

- When clients at on-line store add item to their shopping cart, how does server know what's already in cart?
- When clients decide to proceed to checkout, how can server determine which previously created cart is theirs?
- ...



Cookies

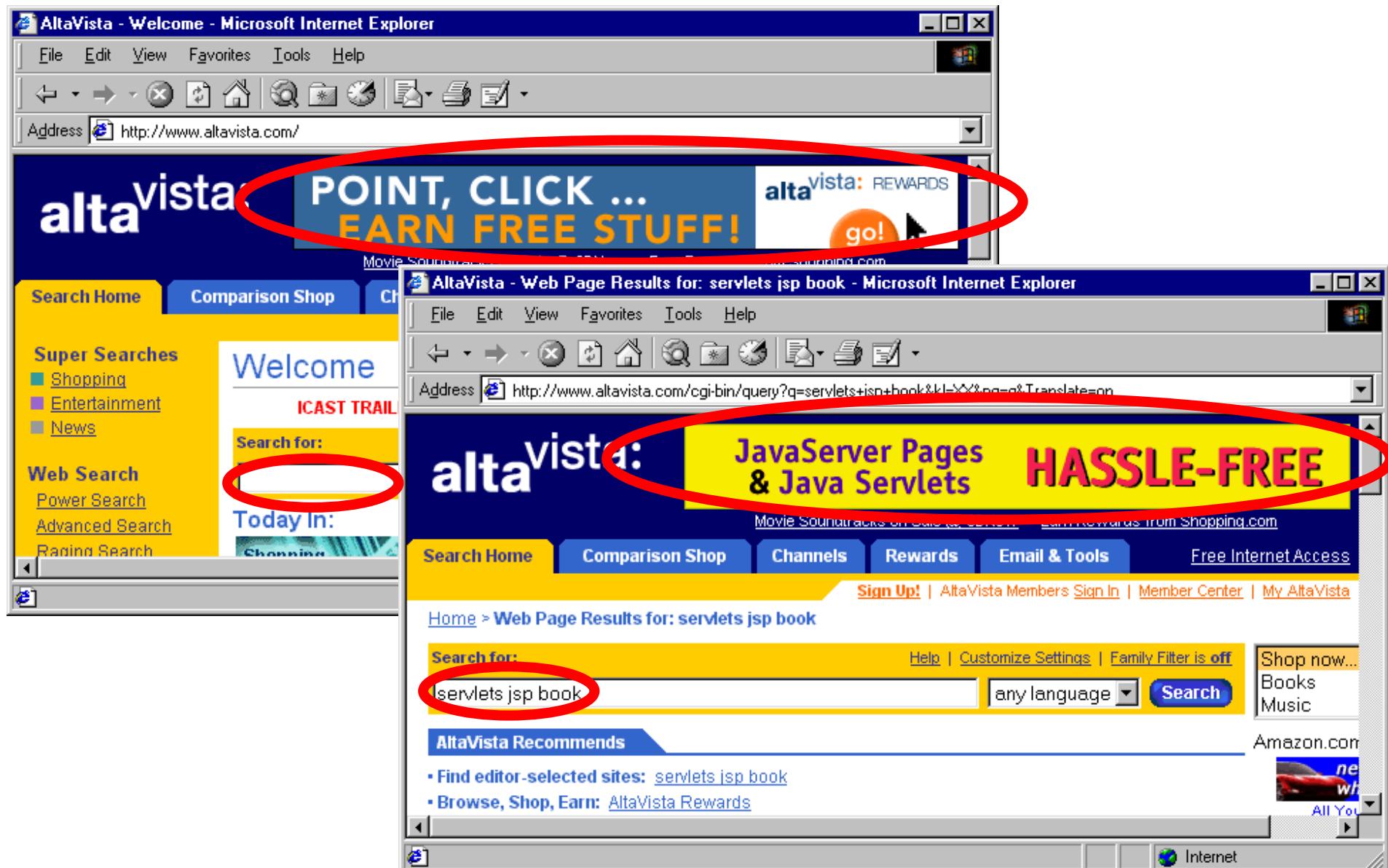
- Il **cookie** è un'unità di informazione che Web server deposita sul Web browser lato cliente
 - Può contenere valori che sono propri del dominio funzionale dell'applicazione (in genere informazioni associate all'utente)
 - Sono parte dell'header HTTP, trasferiti in formato testuale
 - Vengono mandati avanti e indietro nelle richieste e nelle risposte
 - Vengono memorizzati dal browser (client maintained state)
- Attenzione però: possono essere rifiutati dal browser (tipicamente perché disabilitati)
- Sono spesso considerati un fattore di rischio



The potential of Cookies

- Idea
 - Servlet sends a simple name and value to client
 - Client returns same name and value when it connects to same site (or same domain, depending on cookie settings)
- Typical Uses of Cookies
 - Identifying a user during an e-commerce session
 - Servlets have a higher-level API for this task
 - Avoiding username and password
 - Customizing a site
 - Focusing advertising

Cookies and Focused Advertising



La classe cookie

- Un cookie contiene un certo numero di informazioni, tra cui:
 - una coppia nome/valore
 - Caratteri non utilizzabili: [] () = , " / ? @ : ;
 - il dominio Internet dell'applicazione che ne fa uso
 - path dell'applicazione
 - una expiration date espressa in secondi (-1 indica che il cookie non sarà memorizzato su file associato, 60*60*24 indica 24 ore)
 - un valore booleano per definirne il livello di sicurezza
- La **classe Cookie** modella il cookie HTTP
 - Si recuperano i cookie dalla **request** utilizzando il metodo **getCookies()**
 - Si aggiungono cookie alla **response** utilizzando il metodo **addCookie()**

Esempi di uso di cookie

- Con il metodo `setSecure(true)` il client viene forzato a inviare il cookie solo su protocollo sicuro (HTTPS)

creazione

```
Cookie c = new Cookie("MyCookie", "test");
c.setSecure(true);
c.setMaxAge(-1);
c.setPath("/");
response.addCookie(c);
```

lettura

Esempi di uso di cookie (2)

- To delete a cookie then you simply need to follow up following three steps:
 1. Read an already existing cookie and store it in Cookie object
 2. Set cookie age as zero using **setMaxAge()** method to delete an existing cookie
 3. Add this cookie back into response header
- Es:

```
Cookie[] cookies = null;  
cookies = request.getCookies();  
Cookie cookie = cookies[i]; //i-esimo Cookie  
cookie.setMaxAge(0);  
response.addCookie(cookie);
```

Sending Cookies to Browser

- Standard approach:

```
Cookie c = new Cookie("name", "value");
c.setMaxAge(...); // Means cookie persists on disk
// Set other attributes
response.addCookie(c);
```

- Simplified approach (use LongLivedCookie class):

```
public class LongLivedCookie extends Cookie {
    public static final int SECONDS_PER_YEAR = 60*60*24*365;

    public LongLivedCookie(String name, String value) {
        super(name, value);
        setMaxAge(SECONDS_PER_YEAR);
    }
}
```

Reading Cookies from Browser

- Standard approach:

```
Cookie[] cookies = request.getCookies();

if (cookies != null) {

    for(int i=0; i<cookies.length; i++) {

        Cookie c = cookies[i];

        if (c.getName().equals("someName")) {

            doSomethingWith(c);

            break;
        }
    }
}
```

Simple Cookie-Setting Servlet

```
public class SetCookies extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        for(int i=0; i<3; i++) {  
            Cookie cookie = new Cookie("Session-Cookie-" + i, "Cookie-Value-S" + i);  
            response.addCookie(cookie);  
            cookie = new Cookie("Persistent-Cookie-" + i, "Cookie-Value-P" + i);  
            cookie.setMaxAge(3600);  
            response.addCookie(cookie);  
        }  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println(...);  
    }  
}
```

Simple Cookie-Viewing Servlet

```
public class ShowCookies extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>" +
                    "<h1>Active Cookies </h1>" +
                    "<table border=1>" +
                    "<tr><th>Cookie Name</th><th>Cookie Value</th></tr>");
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            Cookie cookie;
            for(int i=0; i<cookies.length; i++) {
                cookie = cookies[i];
                out.println("<tr>" +
                           "<td>" + cookie.getName() + "</td>" +
                           "<td>" + cookie.getValue() + "</td>");
            }
        }
        out.println("</table></body></html>");
    }
}
```

```
public class ClientAccessCounts extends HttpServlet {  
  
    public static String getCookieValue(HttpServletRequest request,  
                                         String cookieName, String defaultValue) {  
        Cookie[] cookies = request.getCookies();  
        if (cookies != null) {  
            for (Cookie cookie : cookies) {  
                if (cookieName.equals(cookie.getName())) {  
                    return (cookie.getValue());  
                }  
            }  
        }  
        return (defaultValue);  
    }  
  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        String countString = getCookieValue(request, "accessCount", "1");  
        int count = 1;  
        try {  
            count = Integer.parseInt(countString);  
        } catch(NumberFormatException nfe) { }  
  
        LongLivedCookie c = new LongLivedCookie("accessCount", String.valueOf(count+1));  
        response.addCookie(c);  
  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<!DOCTYPE HTML> " +  
                   "<HTML> " +  
                   "<HEAD><TITLE>Access Count Servlet</TITLE></HEAD> " +  
                   "<BODY> " +  
                   "<H1>Access Count Servlet</H1> " +  
                   "<H2>This is visit number " + count + " by this browser.</H2>\n" +  
                   "</BODY></HTML>");  
    }  
}
```

Access Count Servlet with Cookies

USING COOKIES TO REMEMBER USER PREFERENCES (REGISTRATIONFORM SERVLET)

- THE FORM IS REDISPLAYED IF IT IS INCOMPLETE WHEN SUBMITTED.
- THE FORM SENDS DATA TO A SECOND SERVLET (REGISTRATIONSERVLET) THAT CHECKS WHETHER ANY OF THE DESIGNATED REQUEST PARAMETERS IS MISSING, THEN STORES THE PARAMETER VALUES IN COOKIES.
- IF NO PARAMETER IS MISSING, THE SECOND SERVLET DISPLAYS THE PARAMETER VALUES.
- IF A PARAMETER IS MISSING, THE SECOND SERVLET REDIRECTS THE USER TO THE ORIGINAL SERVLET SO THAT THE FORM CAN BE REDISPLAYED. THE ORIGINAL SERVLET MAINTAINS THE USER'S PREVIOUSLY ENTERED VALUES BY EXTRACTING THEM FROM THE COOKIES.

REGISTRATIONFORM.JAVA

```
public class RegistrationForm extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        String actionURL =  
            "/servlet/coreservlets.RegistrationServlet";  
        String firstName =  
            CookieUtilities.getCookieValue(request, "firstName", "");  
        String lastName =  
            CookieUtilities.getCookieValue(request, "lastName", "");  
        String emailAddress =  
            CookieUtilities.getCookieValue(request, "emailAddress",  
                                         "");  
        String docType =  
            "<!DOCTYPE HTML PUBLIC \\"-//W3C//DTD HTML 4.0 " +  
            "Transitional//EN\\">\n";  
        String title = "Please Register";  
        out.println  
            (docType +  
             "<HTML>\n" +
```

Cookies

```
out.println
(docType +
"<HTML>\n" +
"<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
"<BODY BGCOLOR=\\"#FDF5E6\\">\n" +
"<CENTER>\n" +
"<H1>" + title + "</H1>\n" +
"<FORM ACTION=\\" " + actionURL + " \\>\n" +
"First Name:\n" +
"  <INPUT TYPE=\\"TEXT\\" NAME=\\"firstName\\" " +
    "VALUE=\\" " + firstName + " \\><BR>\n" +
"Last Name:\n" +
"  <INPUT TYPE=\\"TEXT\\" NAME=\\"lastName\\" " +
    "VALUE=\\" " + lastName + " \\><BR>\n" +
"Email Address: \n" +
"  <INPUT TYPE=\\"TEXT\\" NAME=\\"emailAddress\\" " +
    "VALUE=\\" " + emailAddress + " \\><P>\n" +
"<INPUT TYPE=\\"SUBMIT\\" VALUE=\\"Register\\>\n" +
"</FORM></CENTER></BODY></HTML>" );
```

}

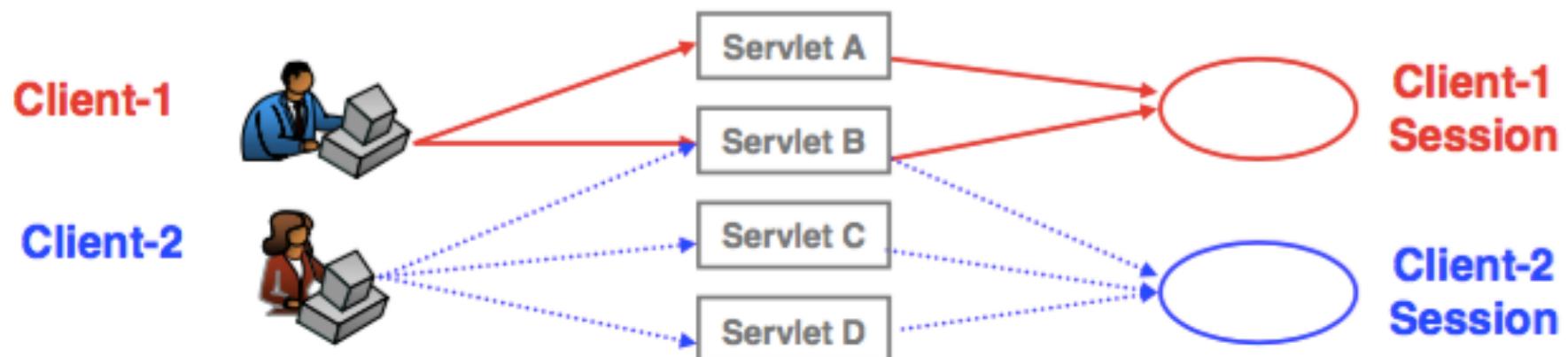
```
Code public class RegistrationServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        boolean isMissingValue = false;  
        String firstName = request.getParameter("firstName");  
        if (isMissing(firstName)) {  
            firstName = "Missing first name";  
            isMissingValue = true;  
        }  
        String lastName = request.getParameter("lastName");  
        if (isMissing(lastName)) {  
            lastName = "Missing last name";  
            isMissingValue = true;  
        }  
        String emailAddress = request.getParameter("emailAddress");  
        if (isMissing(emailAddress)) {  
            emailAddress = "Missing email address";  
            isMissingValue = true;  
        }  
        Cookie c1 = new LongLivedCookie("firstName", firstName);  
        response.addCookie(c1);  
        Cookie c2 = new LongLivedCookie("lastName", lastName);  
        response.addCookie(c2);  
        Cookie c3 = new LongLivedCookie("emailAddress",  
                                         emailAddress);  
        response.addCookie(c3);  
        String formAddress =  
            "/servlet/coreservlets.RegistrationForm";  
        if (isMissingValue) {  
            response.sendRedirect(formAddress);  
        }  
    }  
}
```

```
    } else {
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
             \"Transitional//EN\">\n";
        String title = "Thanks for Registering";
        out.println
            (docType +
             "<HTML>\n" +
             "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
             "<BODY BGCOLOR=\"#FDF5E6\">\n" +
             "<CENTER>\n" +
             "<H1 ALIGN>" + title + "</H1>\n" +
             "<UL>\n" +
             "  <LI><B>First Name</B>: " +
             firstName + "\n" +
             "  <LI><B>Last Name</B>: " +
             lastName + "\n" +
             "  <LI><B>Email address</B>: " +
             emailAddress + "\n" +
             "</UL>\n" +
             "</CENTER></BODY></HTML>" );
    }
}

/** Determines if value is null or empty. */
private boolean isMissing(String param) {
    return((param == null) ||
           (param.trim().equals("")));
}
```

Uso della sessione: Session

- La sessione Web è un'entità gestita dal Web Container
- È condivisa fra tutte le richieste provenienti dallo stesso client: consente di mantenere, quindi, informazioni di stato (di sessione)
- Può contenere dati di varia natura ed è identificata in modo univoco da un **session ID**
- Viene usata dai componenti di una Web application per mantenere lo stato del client durante le molteplici interazioni dell'utente con la Web application (conversazione)

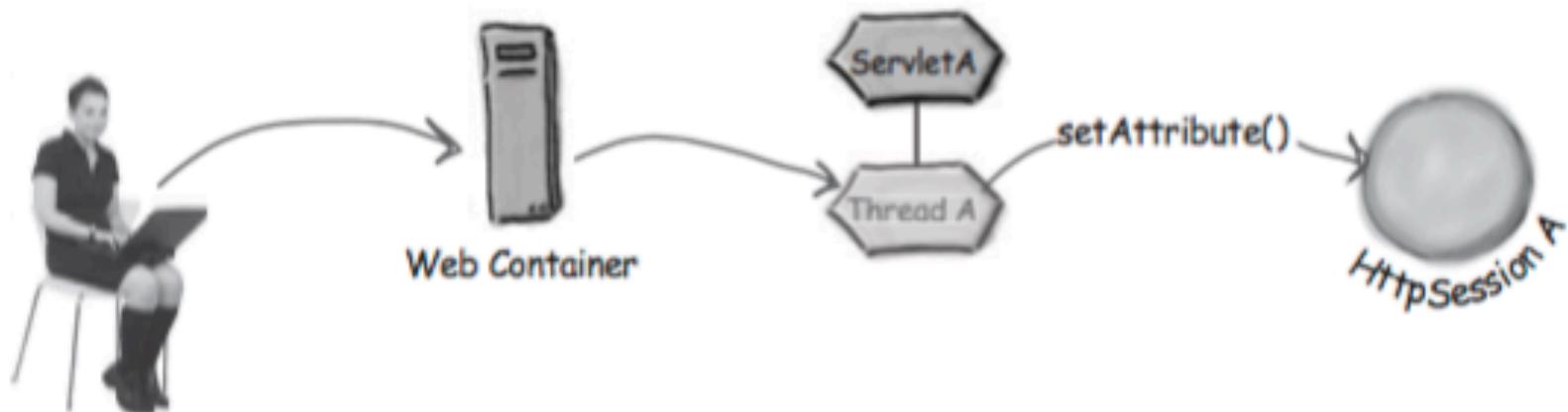


①

Diane selects "Dark" and hits the submit button.

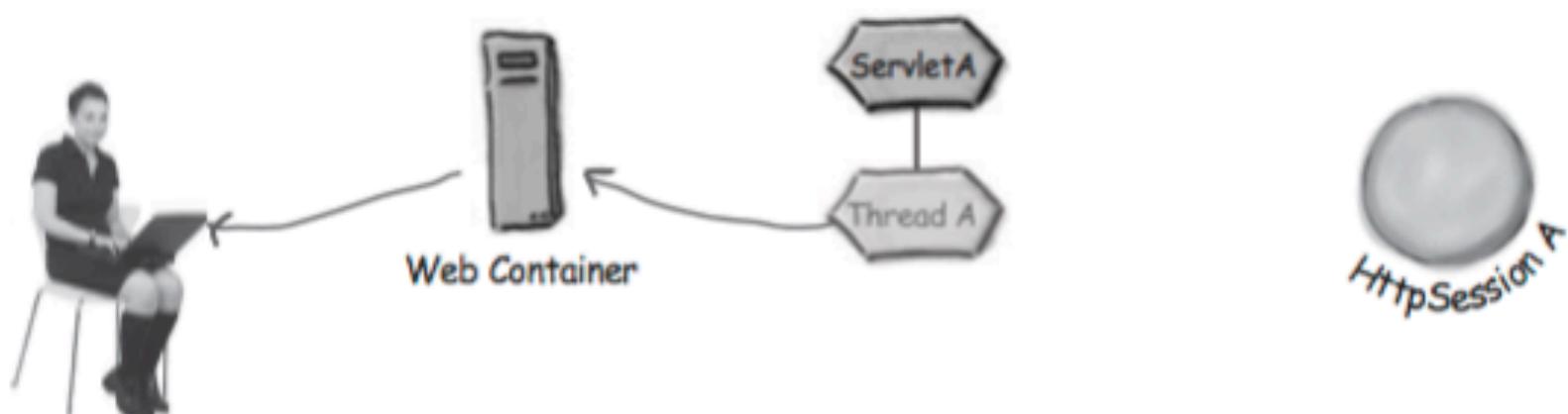
The Container sends the request to a new thread of the BeerApp servlet.

The BeerApp thread finds the session associated with Diane, and stores her choice ("Dark") in the session as an attribute.



②

The servlet runs its business logic (including calls to the model) and returns a response... in this case another question, "What price range?"

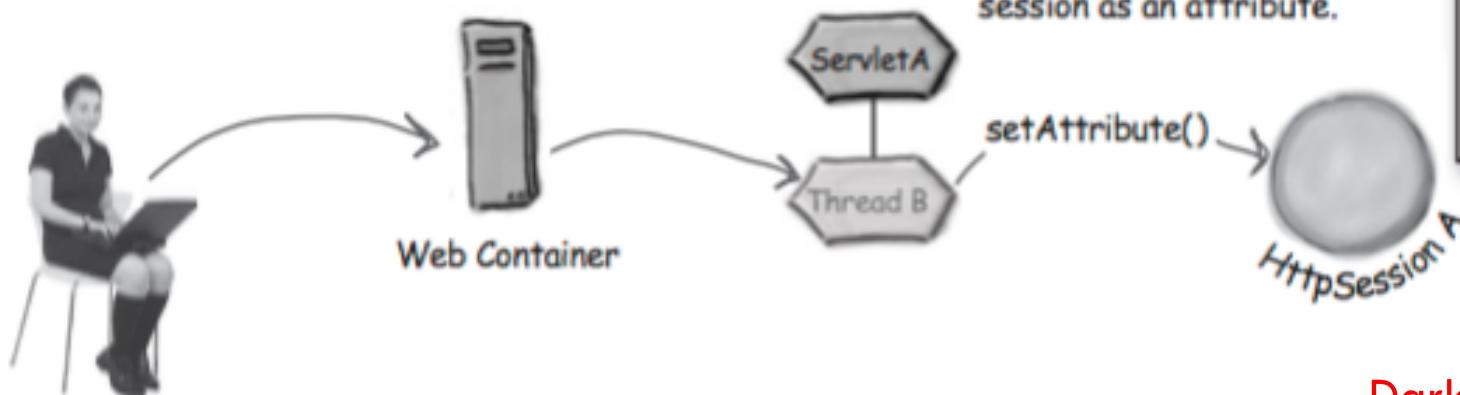


③

Diane considers the new question on the page, selects "Expensive" and hits the submit button.

The Container sends the request to a new thread of the BeerApp servlet.

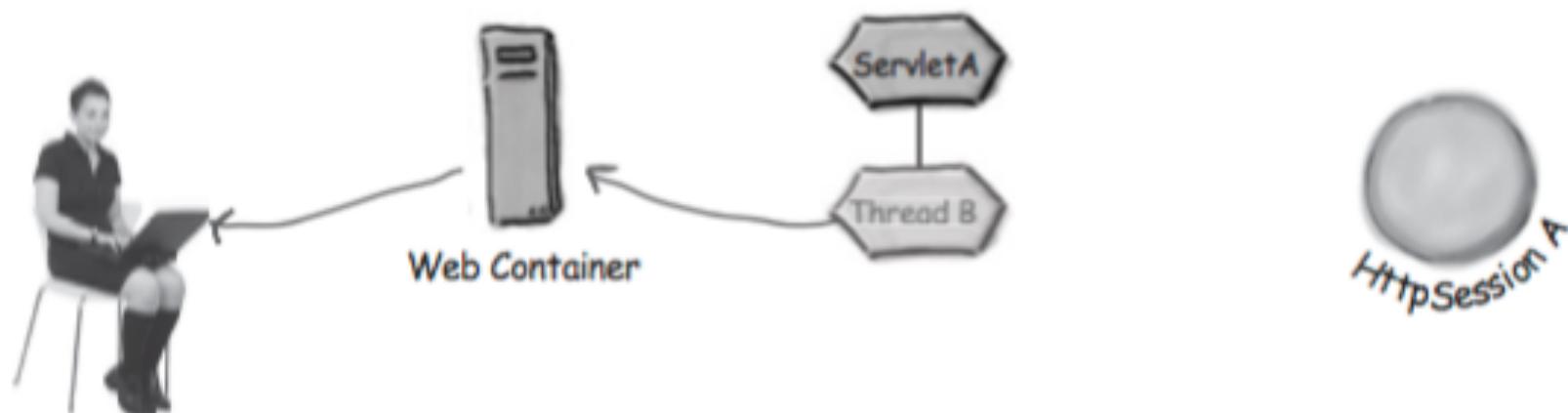
The BeerApp thread finds the session associated with Diane, and stores her new choice ("Expensive") in the session as an attribute.



Dark and Expensive

④

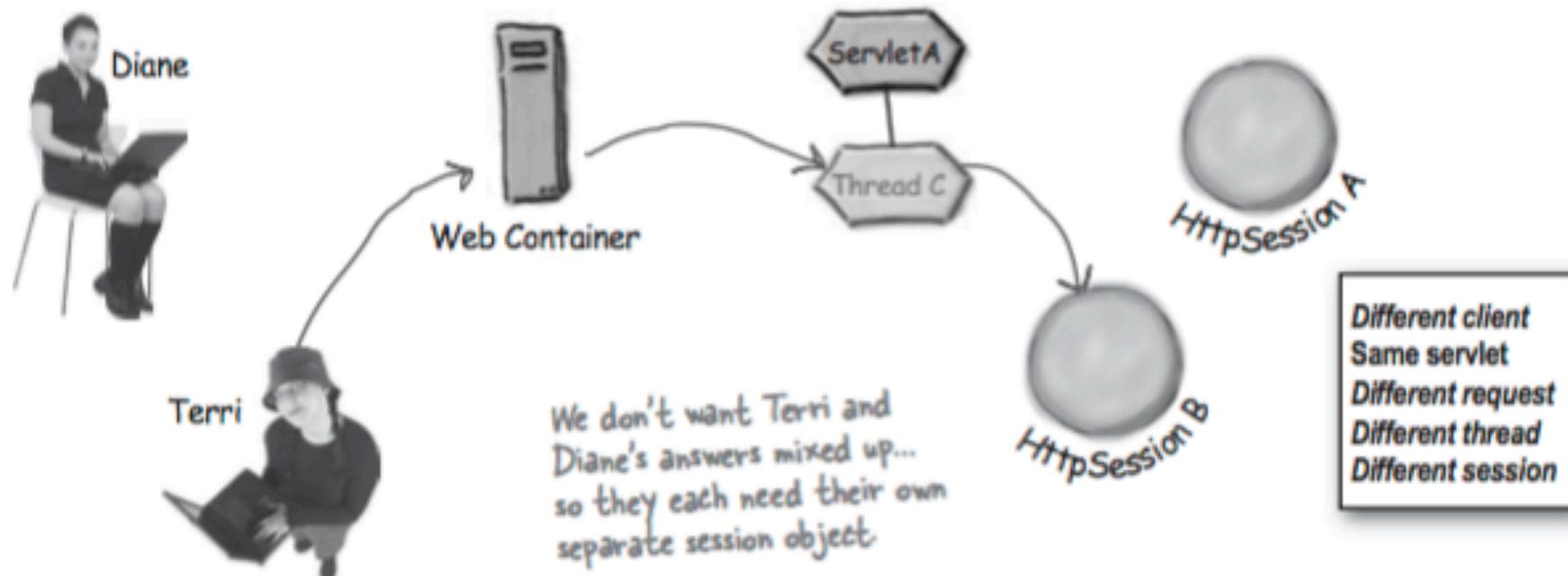
The servlet runs its business logic (including calls to the model) and returns a response... in this case another question.



- ⑤ Diane's session is still active, but meanwhile Terri selects "Pale" and hits the submit button.

The Container sends Terri's request to a new thread of the BeerApp servlet.

The BeerApp thread starts a new Session for Terri, and calls `setAttribute()` to store her choice ("Pale").



Accesso alla sessione

- L'accesso avviene mediante l'interfaccia **HttpSession**
- Per ottenere un riferimento ad un oggetto di tipo HttpSession si usa il metodo **getSession()** dell'interfaccia **HttpServletRequest**

```
public HttpSession getSession(boolean createNew);
```

- Valori di createNew:
 - **true**: ritorna la sessione esistente o, se non esiste, ne crea una nuova
 - **false**: ritorna, se possibile, la sessione esistente, altrimenti ritorna null
- Uso del metodo in una Servlet:

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

- Per recuperare l'ID della sessione:

```
HttpSession ssn = request.getSession();
if(ssn != null){
    String ssnId = ssn.getId();
    System.out.println("Your session Id is : "+ ssnId);
}
```

Gestione del contenuto di una sessione

- Si possono memorizzare **dati specifici dell'utente negli attributi della sessione** (coppie nome/valore)
- Sono simili agli attributi di contesto, ma con scope fortemente diverso!, e consentono di memorizzare e recuperare oggetti

```
Cart sc = (Cart) session.getAttribute("shoppingCart");
sc.addItem(item);
...
session.removeAttribute("shoppingCart");
...
session.setAttribute("shoppingCart", new Cart());
...
Enumeration e = session.getAttributeNames();
while(e.hasMoreElements())
    out.println("Key; " + (String)e.nextElement());
```

Altre operazioni con le sessioni

- **String getId()** restituisce l'ID di una sessione
- **boolean isNew()** dice se la sessione è nuova
- **void invalidate()** permette di invalidare (*distruggere*) una sessione
- **long getCreationTime()** dice da quanto tempo è attiva la sessione (in millisecondi)
- **long getLastAccessedTime ()** fornisce informazioni su quando è stata utilizzata l'ultima volta

```
String sessionID = session.getId();
if(session.isNew())
    out.println("La sessione e' nuova");
session.invalidate();
out.println("Millisec:" + session.getCreationTime());
out.println(session.getLastAccessedTime());
```

Session ID, Cookie e URL Rewriting

- *Il session ID è usato per identificare le richieste provenienti dallo stesso utente e mapparle sulla corrispondente sessione*

JSESSIONID=FE8FC061C396D7BA00D1C18EFDE8146B

1. **Una tecnica per trasmettere l'ID è quella di includerlo in un cookie (session cookie):** sappiamo però che non sempre i cookie sono attivati nel browser
2. **Un'alternativa è rappresentata dall'inclusione del session ID nella URL:** si parla di **URL rewriting**

URL-Rewriting

- Idea
 - Client appends some extra data on the end of each URL that identifies the session
 - Server associates that identifier with data it has stored about that session
 - Es: **http://host/path/file.html;jsessionid=FE8FC1234567890**
- Advantage
 - Works even if cookies are disabled or unsupported
- Disadvantages
 - Must encode all URLs that refer to your own site
 - All pages must be dynamically generated
 - Fails for bookmarks and links from other sites

URL-Rewriting (2)

- È buona prassi codificare sempre le URL generate dalle Servlet usando il metodo **encodeURL()** di **HttpServletResponse**
 - Usato per garantire una gestione corretta della sessione
 - Se il server sta usando i cookie, ritorna l'URL non modificato
 - Se il server sta usando l'URL rewriting, prende in input un URL, e se l'utente ha i cookie disattivati, codifica l'id di sessione nell'URL
- Il metodo encodeURL() dovrebbe essere usato per:
 - hyperlink ()
 - form (<form action="...">)
- Es:

```
String url = "order-page.html";
url = response.encodeURL(url);
```

Cookie attivati: order-page.html;

Cookie disattivati: order-page.html;jsessionid=7199A66051A35953113E8D134B02034F

Session tracking basics

...

```
HttpSession session = request.getSession();

SomeClass value = (SomeClass) session.getAttribute("someID");
if (value == null) {
    value = new SomeClass();
}
doSomethingWith(value);
session.setAttribute("someID", value);
```

Hidden form field

- Idea:

```
<input type="hidden" name="session" value="7199A6...">
```

- Advantage

- Works even if cookies are disabled or unsupported

- Disadvantages

- Lots of tedious processing

- All pages must be the result of form submissions

Hidden form field (2)

- In the Servlet:

```
Map<String, HttpSession> sessions = (Map<String, HttpSession>)
    request.getServletContext().getAttribute("sessionmap");
if(sessions == null) {
    sessions = new HashMap<String, HttpSession>();
    request.getServletContext().setAttribute("sessionmap", sessions);
}
HttpSession session = request.getSession();
sessions.putIfAbsent(session.getId(), session);
```

- In the JSP:

```
<form method="post" action="ShowSession">
    <input type="hidden" name="session" value="<%request.getSession().getId()%>">
    <!-- ... -->
    <br>
    <input type="submit" value="Add">
</form>
```

- In the Servlet (ShowSession):

```
String sessionID = request.getParameter("session");
if(sessionID != null) {
    Map<String, HttpSession> sessions = (Map<String, HttpSession>)
        request.getServletContext().getAttribute("sessionmap");
    if(sessions != null) {
        HttpSession session = sessions.get(sessionID);
        /* ... */
    }
}
```

To Synchronize or not to Synchronize?

- There are no race conditions when multiple different users access the page simultaneously
- It seems practically impossible for the same user to access the session concurrently
- The rise of Ajax makes synchronization important
 - With Ajax calls, it is actually quite likely that two requests from the same user could arrive concurrently

Accumulating a list of user data

...

```
HttpSession session = request.getSession();
synchronized (session) {
    @SuppressWarnings("unchecked")
    List<String> previousItems = (List<String>) session.getAttribute("previousItems");
    if (previousItems == null) {
        previousItems = new ArrayList<String>();
    }
    String newItem = request.getParameter("newItem");
    if (newItem != null) {
        previousItems.add(newItem);
    }
    session.setAttribute("previousItems", previousItems);
}
```

...

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
response.setContentType("text/html");
HttpSession session = request.getSession();
synchronized (session) {
    String heading;
    Integer accessCount = (Integer) session.getAttribute("accessCount");
    if (accessCount == null) {
        accessCount = 0;
        heading = "Welcome, Newcomer";
    } else {
        heading = "Welcome Back";
        accessCount = accessCount + 1;
    }
    session.setAttribute("accessCount", accessCount);

    PrintWriter out = response.getWriter();
    out.println(
        "<!DOCTYPE html>" +
        "<html>" +
        "<head><title>Session Tracking Example</title></head>" +
        "<body>" +
        "<h1>" + heading + "</h1>" +
        "<h2>Information on Your Session:</h2>" +
        "<table border='1'>" +
        "<tr>" +
        "  <th>Info Type</th><th>Value</th>" +
        "</tr><tr>" +
        "  <td>ID</td><td>" + session.getId() + "</td>" +
        "</tr><tr>" +
        "  <td>Creation Time</td><td>" + new Date(session.getCreationTime()) + "</td>" +
        "</tr><tr>" +
        "  <td>Time of Last Access</td><td>" + new Date(session.getLastAccessedTime()) + "</td>" +
        "</tr><tr>" +
        "  <td>Number of Previous Accesses</td><td>" + accessCount + "</td>" +
        "</table>" +
        "</body></html>");
}
}
```

Access Count Servlet with Session

Attenzione: scope DIFFERENZIATI (scoped objects)

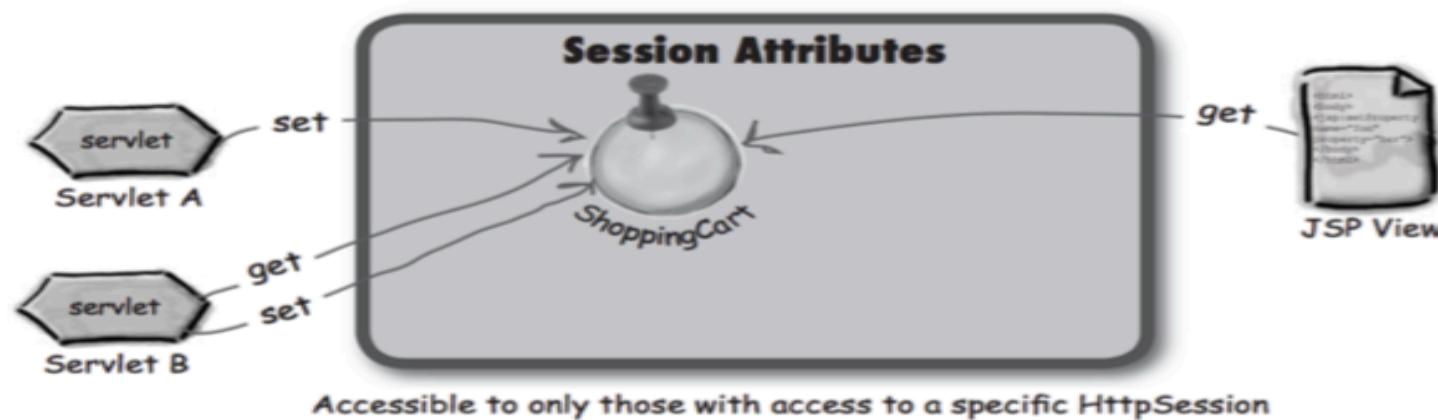
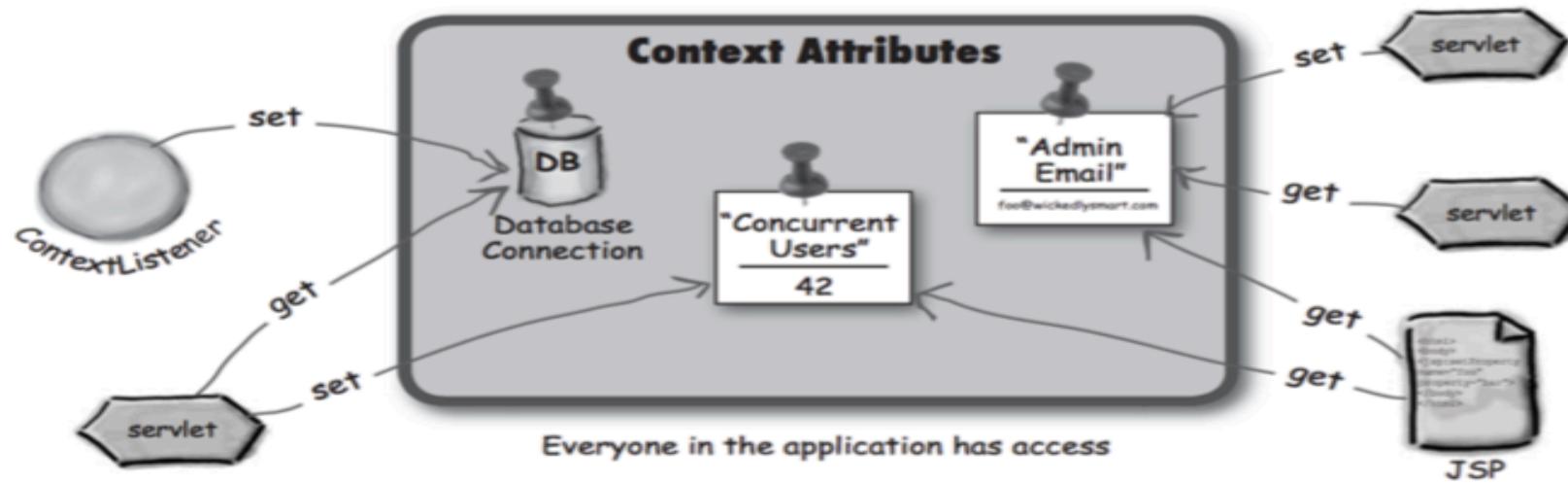
- Gli oggetti di tipo `HttpServletRequest` , `HttpSession` , `ServletContext` forniscono metodi per immagazzinare e ritrovare oggetti nei loro rispettivi ambiti (scope)
- Lo scope è definito dal **tempo di vita** (**lifespan**) e dall'**accessibilità** da parte delle Servlet

Ambito	Interfaccia	Tempo di vita	Accessibilità
Request	<code>HttpServletRequest</code>	Fino all'invio della risposta	Servlet corrente e ogni altra pagina inclusa o in forward
Session	<code>HttpSession</code>	Lo stesso della sessione utente	Ogni richiesta dello stesso client
Application	<code>ServletContext</code>	Lo stesso dell'applicazione	Ogni richiesta alla stessa Web app anche da clienti diversi e per servlet diverse

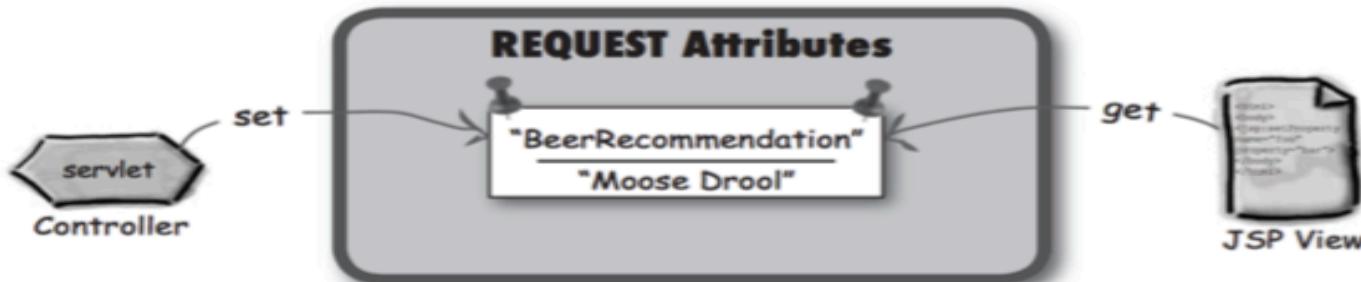
Funzionalità degli scoped object

- Gli oggetti scoped forniscono i seguenti metodi per immagazzinare e ritrovare oggetti nei rispettivi ambiti (scope):
 - **void setAttribute(String name, Object o);**
 - **Object getAttribute(String name);**
 - **void removeAttribute(String name);**
 - **Enumeration getAttributeNames();**

The Three Scopes: Context, Request, and Session



Accessible to only those with access to a specific HttpSession



Accessible to only those with access to a specific ServletRequest

Ridirezione del browser

- È possibile inviare al browser una risposta che lo forza ad accedere ad un'altra pagina/risorsa (**ridirezione**)
- Possiamo ottenere agendo sull'oggetto **response** invocando il metodo
 - **public void sendRedirect(String url);**
- *Scenario: When the Servlet shows the client a JSP with the results, and if you don't want the user to be able to refresh and re-submit the form, then use a sendRedirect*
- **sendRedirect** can be used to communicate between two servlets present on the same or different server. The output will be the same as the **Request Dispatcher forward** but the URL of the page will be changed to the redirected page.

SendRedirect

- Es:

```
String name=request.getParameter("name");
response.sendRedirect("https://www.google.co.in/#q="+name);
```

```
String userAgent = request.getHeader("User-Agent");
if (userAgent != null && userAgent.contains("MSIE")) {
    response.sendRedirect("http://home.mozilla.com");
} else {
    response.sendRedirect("http://www.microsoft.com");
}
```

SendRedirect example

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <body>
        <h2>Servlet SendRedirect Example</h2>
        <form id="loginFormId" method="post" action="loginServlet">
            Username:<input id="userInput" type="text" name="username" required/>
            <br>
            Password:<input id="passInput" type="password" name="password" required/>
            <br>
            <input id="btn" type="submit" value="Login" />
            <%if(request.getAttribute("error") != null) {%
                <p style='color: red;'>You are not an authorized user!</p>
            } %>
        </form>
    </body>
</html>
```

SendRedirect example (2)

```
@WebServlet("/loginServlet")
public class Login extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");

        String login = request.getParameter("username");
        String pwd = request.getParameter("password");

        if(login.equalsIgnoreCase("root") && pwd.equals("admin")) {
            request.getSession().setAttribute("uname", login);
            response.sendRedirect("welcomeServlet");
            return;
        }

        request.setAttribute("error", Boolean.TRUE);
        RequestDispatcher dispatcher = request.getRequestDispatcher("/index.jsp");
        dispatcher.forward(request, response);
    }
    /* ... */
}
```

```
@WebServlet("/welcomeServlet")
public class Welcome extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");

        String param1 = (String) request.getSession().getAttribute("uname");

        PrintWriter out = response.getWriter();
        out.println("<!doctype html>");
        out.println("<html><body>");
        out.println("<h2>Servlet SendRedirect Example</h2>");
        out.println("<p style='color: green;'>");
        out.println("Congratulations! " + param1 + ", You are an authorised login!");
        out.println("</p></body></html>");
    }
    /* ... */
}
```

Includere una risorsa (include)

- Per includere una risorsa si ricorre a un oggetto di tipo **RequestDispatcher** che può essere richiesto al contesto indicando la risorsa da includere
- Si invoca quindi il metodo **include** passando come parametri **request** e **response** che vengono così condivisi con la risorsa inclusa
 - Se necessario, l'URL originale può essere salvato come un attributo di request

può essere anche una pagina JSP

```
RequestDispatcher dispatcher =  
    getServletContext().getRequestDispatcher("/inServlet");  
dispatcher.include(request, response);
```

Inoltro (forward)

- Si usa in situazioni in cui una Servlet si occupa di parte dell'elaborazione della richiesta e delega a qualcun altro la gestione della risposta
 - *Attenzione: in questo caso la risposta è di competenza esclusiva della risorsa che riceve l'inoltro*
 - **Se nella prima Servlet è stato fatto un accesso a ServletOutputStream o PrintWriter si ottiene una IllegalStateException**
- Si deve ottenere un oggetto di tipo **RequestDispatcher** da request passando come parametro il nome della risorsa
- Si invoca quindi il metodo **forward** passando anche in questo caso **request** e **response**

può essere anche una pagina JSP

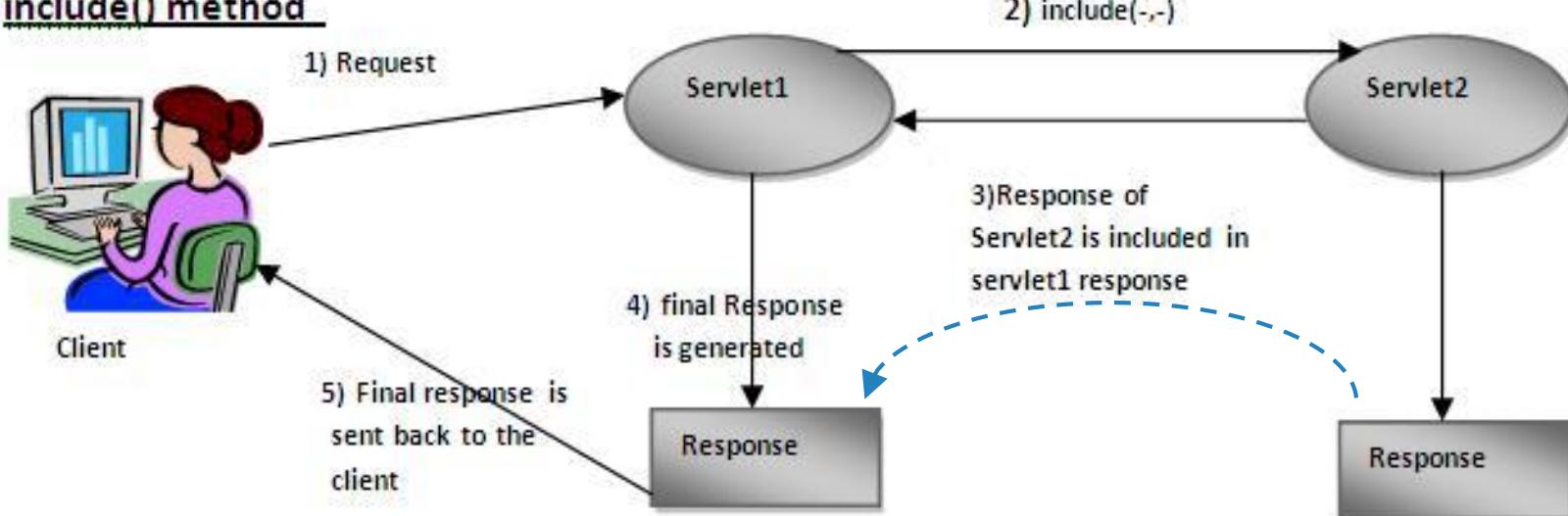
```
RequestDispatcher dispatcher =  
    getServletContext().getRequestDispatcher("/inServlet");  
dispatcher.forward(request, response);
```

Include e forward

- **Scenario (include):** When the current Servlet has done some of its job and is using another Servlet (or JSP) to help with display or other work. If the Servlet using the RequestDispatcher has already written to the response, then you will need to use the include to send control to another Servlet, not the forward. Also, if you want to spread the display of a page across multiple pages, use an include for each part of a page
 - **Example:** Servlet handles a request from the client and does the necessary work. It then uses RequestDispatcher#`include` to add a Header response to all pages, since the header and banner on a page is the same for all pages on the site. It then includes a second JSP which displays the content that is specific to this request. It finally includes a third page that acts as a Footer for all pages on the site
- **Scenario (forward):** When the current Servlet is done its job, and hasn't done anything with the response, you Forward the request and response to another Servlet (or JSP) to finish any work and to display results
 - **Example:** A Servlet retrieves information from a database and stores it in the request object. It then forwards the request to a JSP to display the data. User presses refresh, and the control goes to the Servlet, which again retrieves the data from the database and puts it in the request, forwards to the JSP, and the JSP sees the data

Include e forward

include() method



forward() method:

