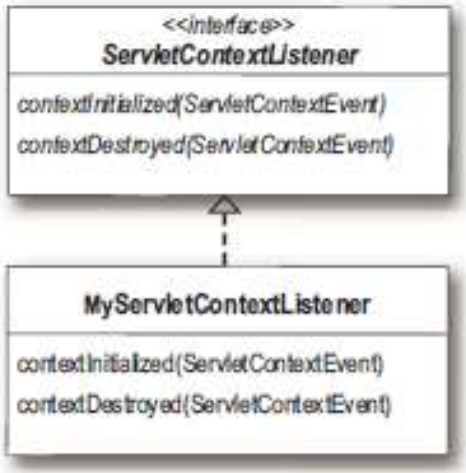
① Create a listener class



To listen for ServletContext events, write a listener class that implements ServletContextListener, put it in your WEB-INF/classes directory, and tell the Container by putting a <listener> element in the Deployment Descriptor.

② Put the class in WEB-INF/classes

(This isn't the ONLY place it can go... WEB-INF/classes is one of several places the Container can look for classes. We'll cover the others in the Deployment chapter.)

CONTEXT LISTENERS

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app …>
   <context-param>
         <param-name>connectionString</param-name>
         <param-value>jdbc:myDriver:myDatabase</param-value>
   </context-param>
   <listener>
         <listener-class> /ContextListenerJDBC</ listener-class>
   </listener>
 </web-app>
```

SETTING CONTEXT ATTRIBUTE

A ServletContextListener class:

ServletContextListener is in javax.servlet package.

A context listener is simple implement ServletContextListener.

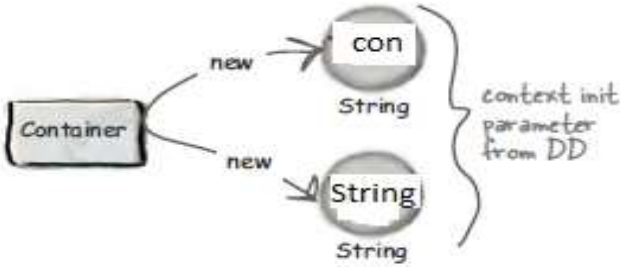
```
import javax.servlet.*;

public class MyServletContextListener implements ServletContextListener {

   public void contextInitialized(ServletContextEvent event) {
     //code to initialize the database connection
     //and store it as a context attribute
   }

   public void contextDestroyed(ServletContextEvent event) {
     //code to close the database connection
   }
}
```

These are the two notifications you get. Both give you a ServletContextEvent

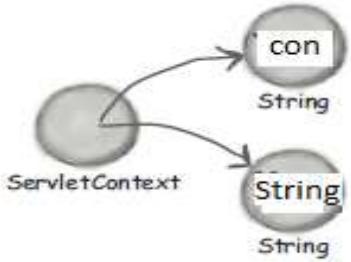
1. Container reads the Deployment Descriptor for this app, including the <listener> and <context-param> elements.

2. Container creates a new ServletContext for this application, that all parts of the app will share.

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

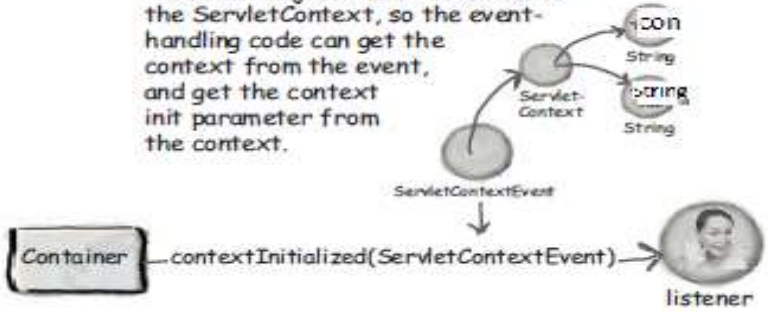


4. Container gives the ServletContext references to the name/value parameters.



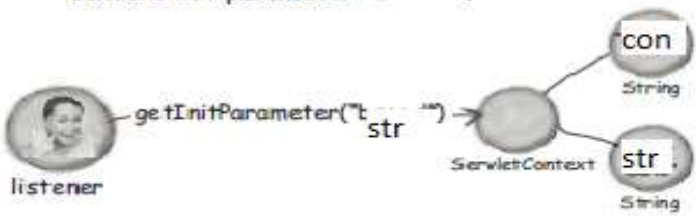
5. Container creates a new instance of the MyServletContextListener class.

6. Container calls the listener's contextInitialized() method, passing in a new ServletContextEvent. The event object has a reference to the ServletContext, so the event-handling code can get the context from the event, and get the context init parameter from the context.

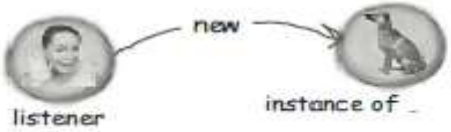


7 Listener asks ServletContextEvent for a reference to the ServletContext.

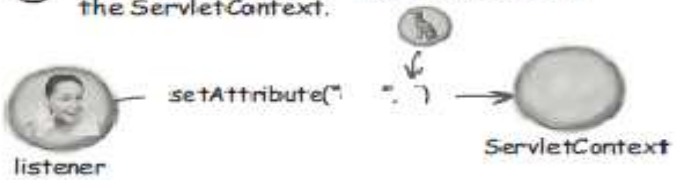
8 Listener asks ServletContext for the context init parameter .



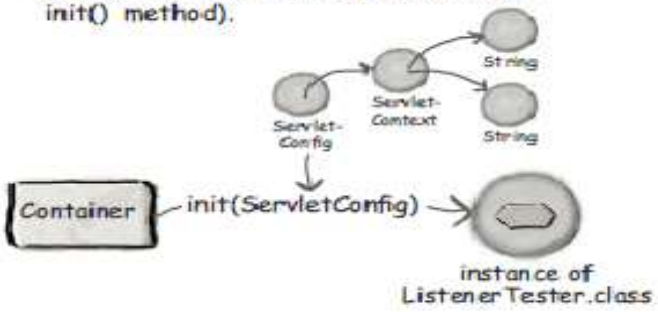
9 Listener uses the init parameter to construct a new object.



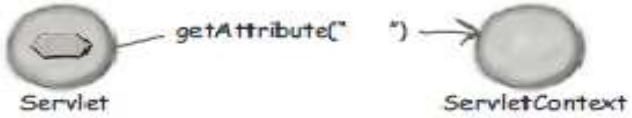
10 Listener sets the as an attribute in the ServletContext.



11 Container makes a new Servlet (i.e., makes a new ServletConfig with init parameters, gives the ServletConfig a reference to the ServletContext, then calls the Servlet's init() method).



12 Servlet gets a request, and asks the ServletContext for the attribute



13 Servlet calls get () on the (and prints that to the HttpResponse).

CONTEXT LISTENER AND SERVLET

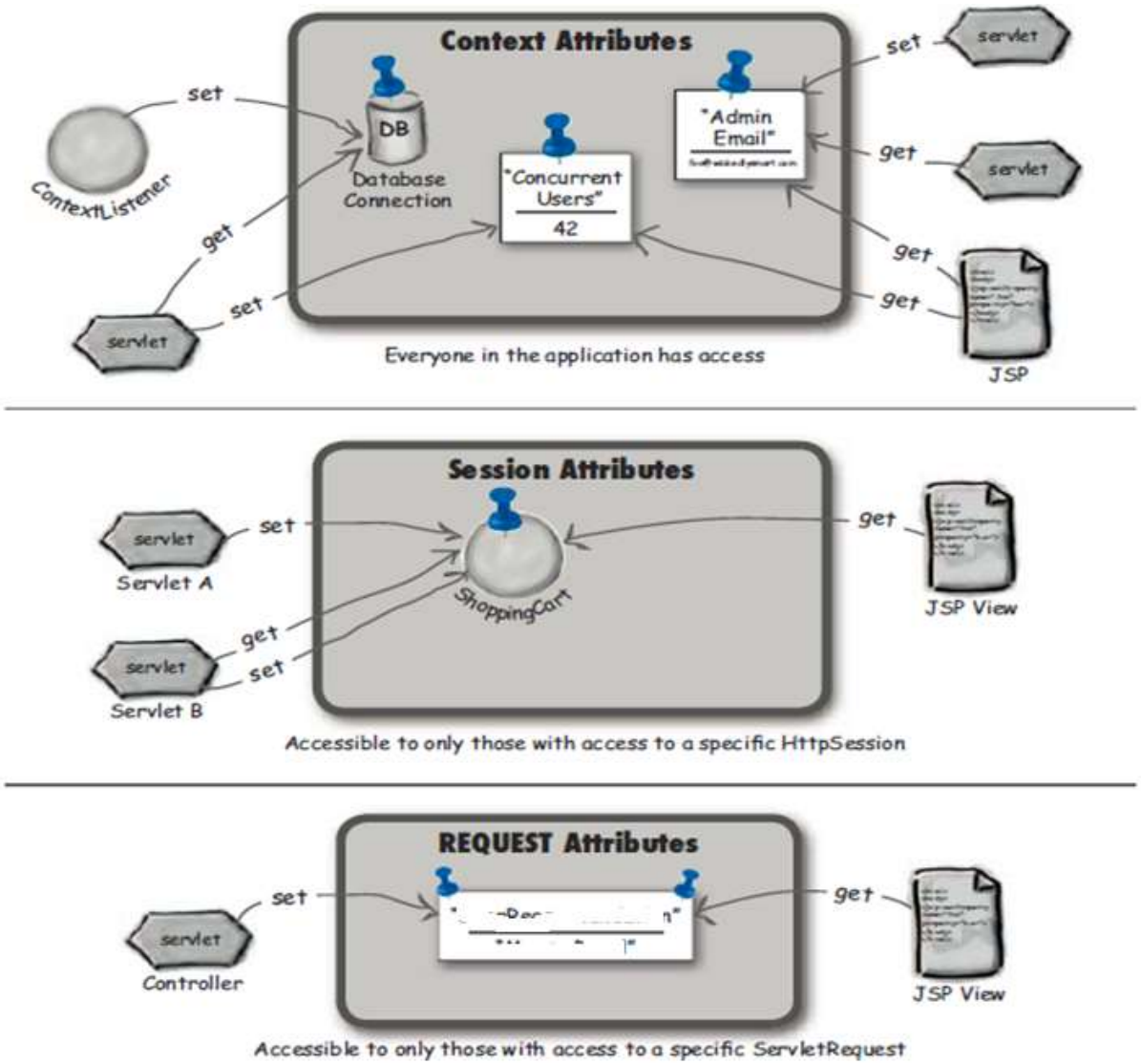
```
public class HelloServlet extends HttpServlet {

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


 response.setContentType("text/html");

   PrintWriter out = response.getWriter();

 Connection con=(Connection)
 getServletContext().getAttribute(“connection1”);

   out.println("<HTML>\n" +

              "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +

              "<BODY BGCOLOR=\"#FDF5E6\">\n" +

              "<H1>”+ con.getSchema() +“</H1></BODY></HTML>");

 }

}
```

```
public final class ContextListener    implements ServletContextListener {
          private ServletContext context = null;
   public void contextDestroyed(ServletContextEvent event) {
      log("contextDestroyed()");
      this.context = null;
   }

public void contextInitialized(ServletContextEvent event) {
      this.context = event.getServletContext();
      …..
      Connection con = DriverManager.getConnection( context.getInitParameter(“ connectionString”), username, password);
     context.setAttribute(“connection1”, con);
      log("contextInitialized()");
   }

    private void log(String message) {
      if (context != null)
         context.log("ContextListener: " + message);
      else
         System.out.println("ContextListener: " + message);
   }
}
```

Eight Listeners

Scenario	Listener interface	Event type
You want to know if an attribute in a web app context has been added, removed, or replaced.	javax.servlet.**ServletContextAttributeListener** *attributeAdded* *attributeRemoved* *attributeReplaced*	ServletContextAttributeEvent
You want to know how many concurrent users there are. In other words, you want to track the active sessions. We cover sessions in	javax.servlet.http.**HttpSessionListener** *sessionCreated* *sessionDestroyed*	HttpSessionEvent
You want to know each time a request comes in, so that you can log it.	javax.servlet.**ServletRequestListener** *requestInitialized* *requestDestroyed*	ServletRequestEvent
You want to know when a request attribute has been added, removed, or replaced.	javax.servlet.**ServletRequestAttributeListener** *attributeAdded* *attributeRemoved* *attributeReplaced*	ServletRequestAttributeEvent

SCOPE OF ATTRIBUTES



PROBLEM WITH CONTEXT ATTRIBUTES

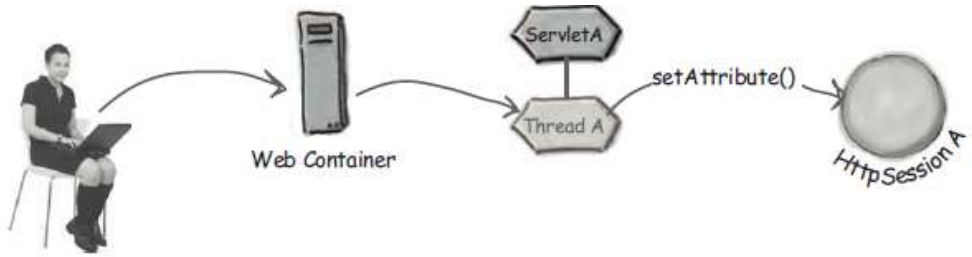
```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                              throws IOException, ServletException {

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

   out.println("test context attributes<br>");

   getServletContext().setAttribute("foo", "22");
   getServletContext().setAttribute("bar", "42");

   out.println(getServletContext().getAttribute("foo"));
   out.println(getServletContext().getAttribute("bar"));
}
```

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

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

    out.println("test context attributes<br>");

    getServletContext().setAttribute("foo", "22");
    getServletContext().setAttribute("bar", "42");

    out.println(getServletContext().getAttribute("foo"));
    out.println(getServletContext().getAttribute("bar"));
}
```

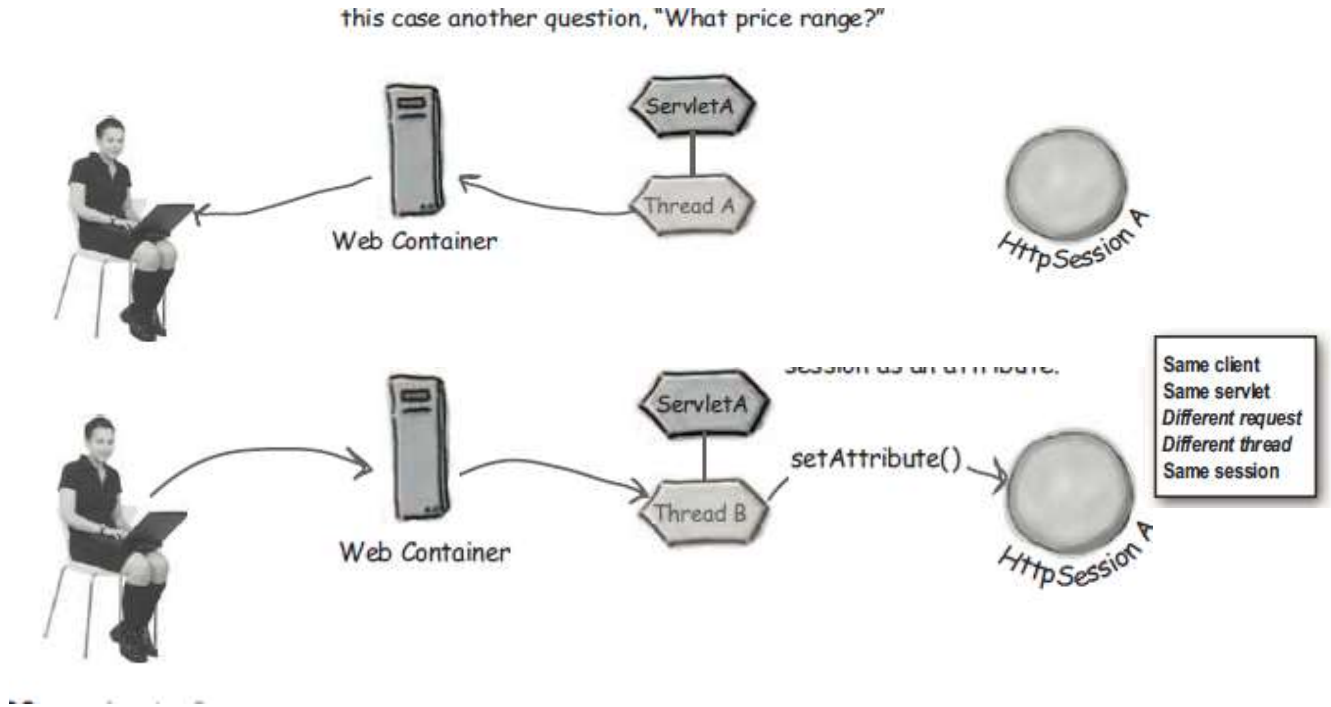
```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws IOException, ServletException {

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

   out.println("test context attributes<br>");

   synchronized(getServletContext()) {
      getServletContext().setAttribute("foo", "22");
      getServletContext().setAttribute("bar", "42");

      out.println(getServletContext().getAttribute("foo"));
      out.println(getServletContext().getAttribute("bar"));
   }
}
```

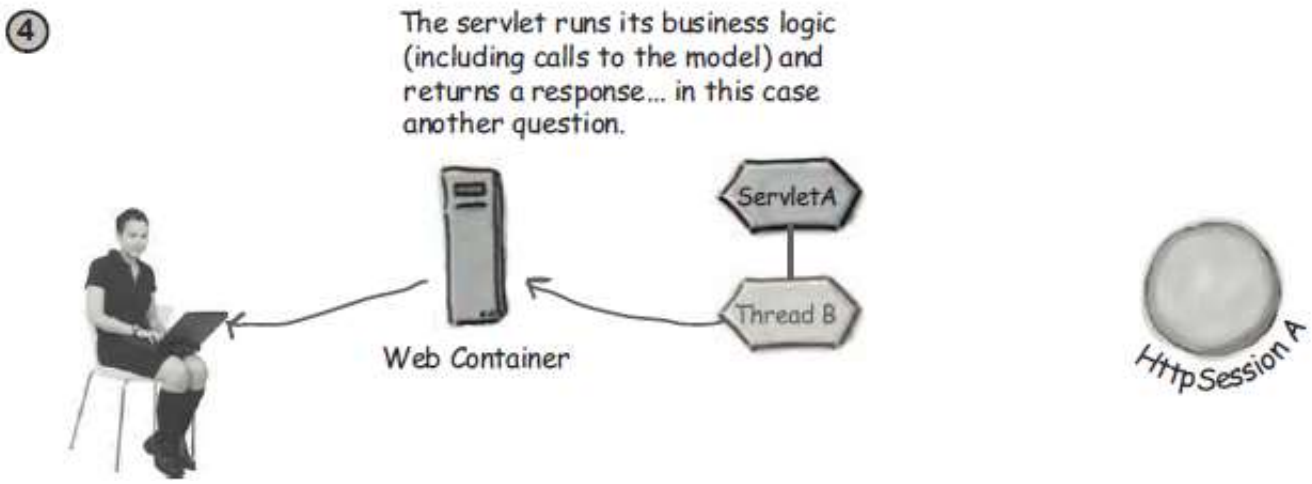
Now we're getting the lock on the context itself!! This is the way to protect context attribute state. (You don't want synchronized(this).)

Only request attributes and local variables are threadsafe



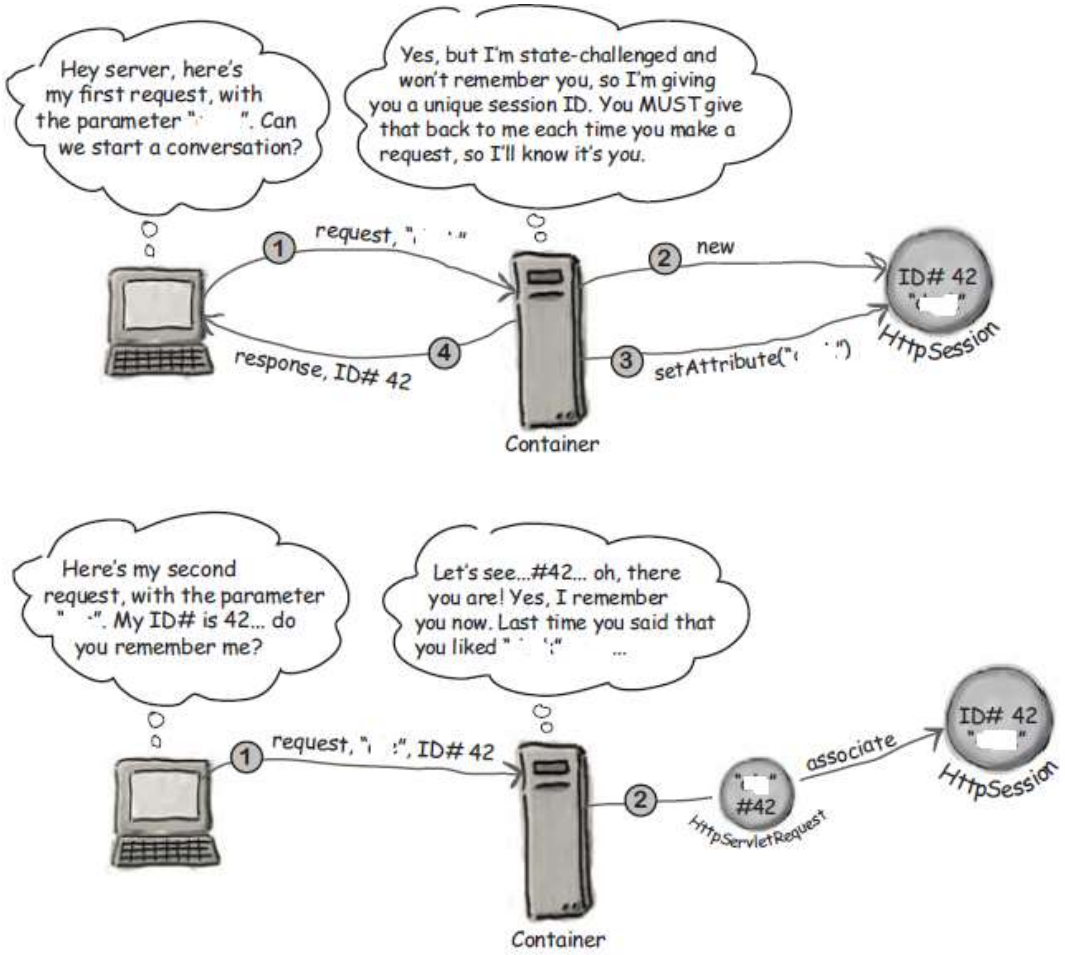
Client asks for a book on web technology



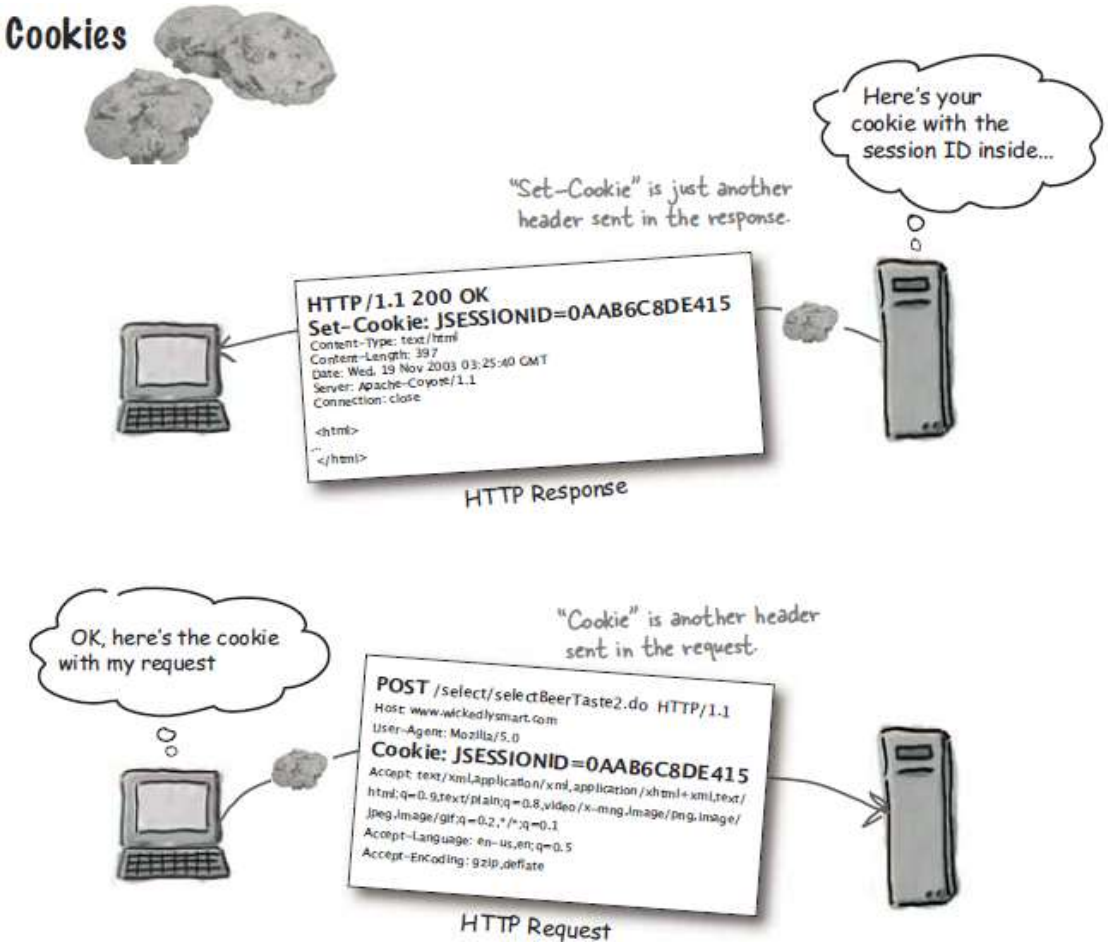


What if another client joins in?

WHO IS THE CLIENT



HOW TO EXCHANGE SESSION ID



ROLE OF THE CONTAINER

You *do* have to tell the Container that you want to create or use a session, but the Container takes care of generating the session ID, creating a new Cookie object, stuffing the session ID into the cookie, and setting the cookie as part of the response. And on subsequent requests, the Container gets the session ID from a cookie in the request, matches the session ID with an existing session, and associates that session with the current request.

Sending a session cookie in the RESPONSE:

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

IF (the request includes a session ID cookie)

find the session matching that ID

ELSE IF (there's no session ID cookie OR there's no current session matching the session ID)

create a *new* session.

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("test session attributes<br>");


    HttpSession session = request.getSession();




}
```

getSession() returns a session no matter what... but you can't tell if it's a new session unless you ask the session.

Request.getSession(false);---if an existing session is not found it returns null

URL REWRITING

response.encodeURL("…")
Response.encodeRedirectURL(" ")

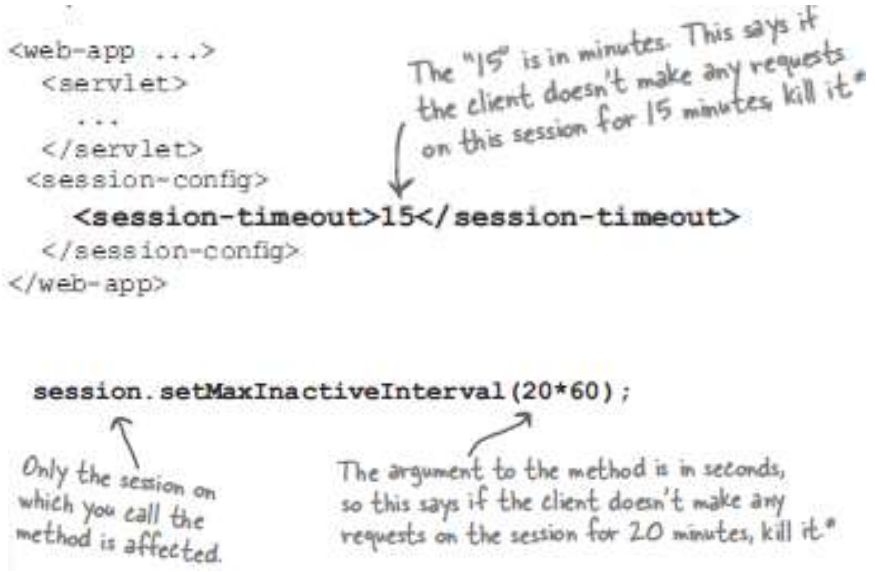
TERMINATING SESSIONS

Sessions are mostly used to get and set session scoped attributes

Ways to terminate a session

- It times out
- Calling session.invalidate()
- The application goes down (crashes or is undeployed)

SESSION TIMES OUT



COOKIES

Session cookies vanish when the client's browser quits, but you CAN tell a cookie to persist on the client even after the browser shuts down.

ADDING AND GETTING COOKIES

Creating a new Cookie

```
Cookie cookie = new Cookie("username", name);
```

The Cookie constructor takes a name/value String pair.

Setting how long a cookie will live on the client

```
cookie.setMaxAge(30*60);
```

setMaxAge is defined in SECONDS. This code says "stay alive on the client for 30*60 seconds" (30 minutes). Setting max age to -1 makes the cookie disappear when the browser exits.

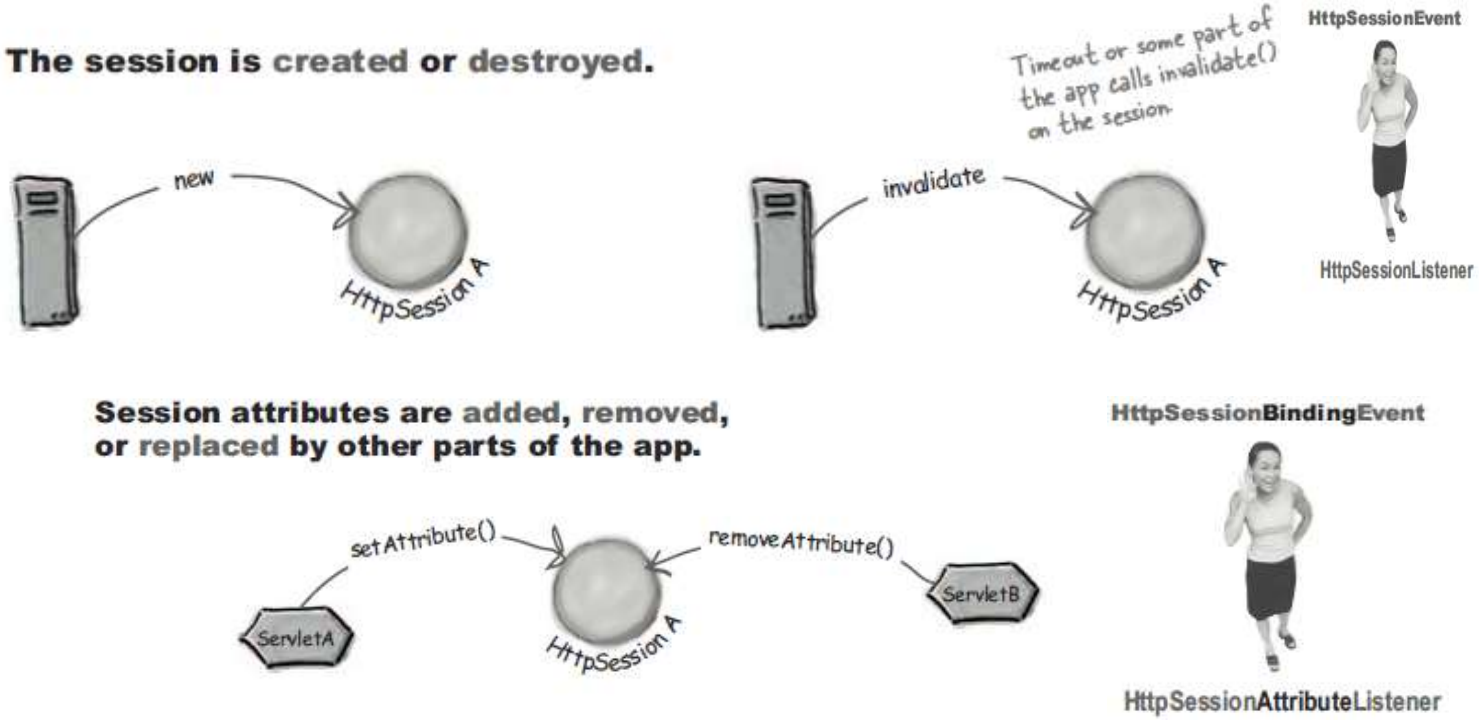
Sending the cookie to the client

```
response.addCookie(cookie);
```

ATTRIBUTES

Scope	In a servlet	In a jsp Implicit obj
Application	getServletContext.setAttribute(“name”,obj)	application.setAttribute(“name”, obj)
Request	request. .setAttribute(“name”,obj)	request. .setAttribute(“name”,obj)
Session	Request.getSession() .setAttribute(“name”,obj)	session.setAttribute(“name”,obj)
Page	NA	pageContext.setAttribute(“name”,obj)

SESSION LIFECYCLE EVENTS



HTTPSESSIONLISTENER

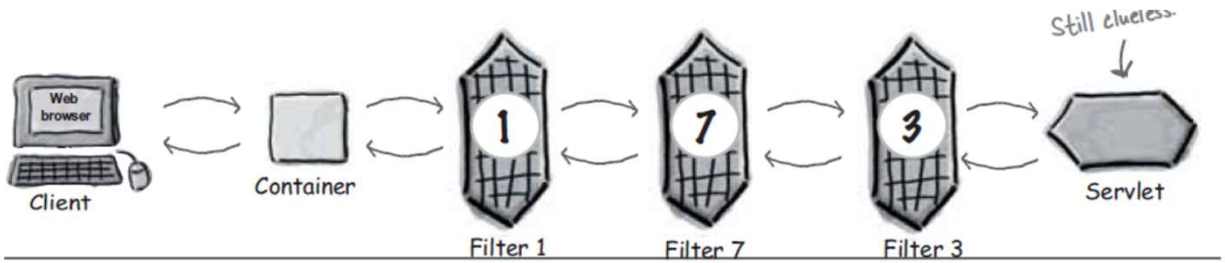
```
package com.example;
import javax.servlet.http.*;

public class     SessionCounter implements HttpSessionListener {

   static private int activeSessions;

   public static int getActiveSessions() {
      return activeSessions;
   }


    public void sessionCreated(HttpSessionEvent event) {
        activeSessions++;
    }

   public void sessionDestroyed(HttpSessionEvent event) {
      activeSessions--;
   }
}
```

This class will be deployed in WEB-INF/classes like all the other web-app classes, so all servlets and other helper classes can access this method.

These methods take an HttpSessionEvent.

<listener><listener-class>com.example.SessionCounter</listener-class></listener>

FILTERS



Request filters can:

- perform security checks
- reformat request headers or bodies
- audit or log requests

Response filters can:

- compress the response stream
- append or alter the response stream
- create a different response altogether

CODE FOR FILTER

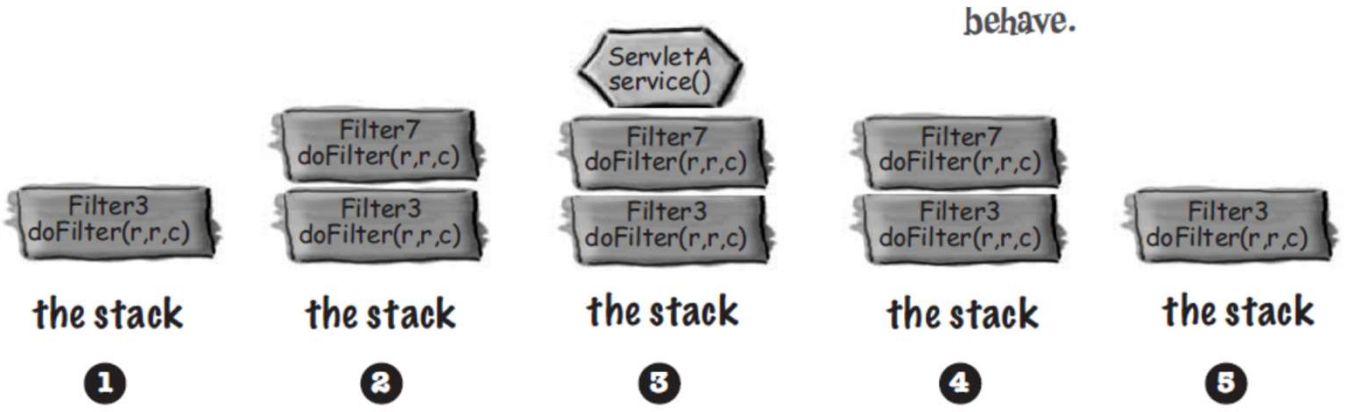
```
@WebFilter("/SelectCoffeeMVC.do")
public class MyFilter implements Filter
{
  private FilterConfig filterConfig = null;

  public void doFilter(ServletRequest request, ServletResponse
response,  FilterChain chain)
    throws IOException, ServletException
  {
    String remark = ((HttpServletRequest)
request).getParameter("remark");

    if(remark.length()>0) {
        chain.doFilter(request, response);
    } else {
        response.setContentType("text/html");
```

SIMILARITY BETWEEN SERTVLETS AND FILTERS

- The container knows the APIs for filter
- The container manages their life cycle
- They are declared in the DD
  - Filter mapping
    - servletNames
    - url patterns

```
<filter-mapping>
  <filter-name>MonitorFilter</filter-name>
  <url-pattern>*.do</url-pattern>
  <dispatcher>REQUEST</dispatcher>
        - and / or -
  <dispatcher>INCLUDE</dispatcher>
        - and / or -
  <dispatcher>FORWARD</dispatcher>
        - and / or -
  <dispatcher>ERROR</dispatcher>
</filter>
```

CONVENTIONAL VIEW



HTTPS

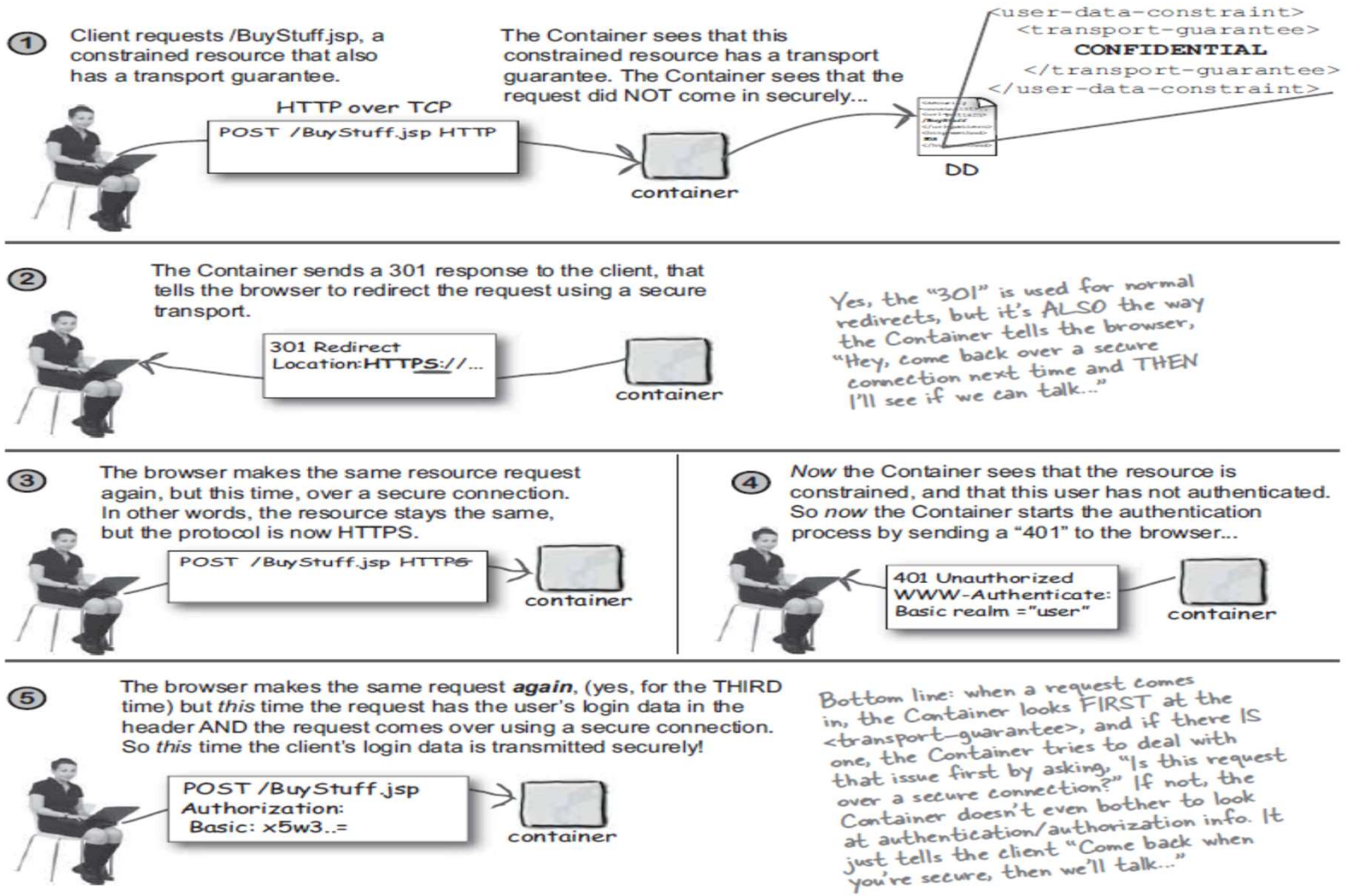
```
<web-app...>
...
  <security-constraint>

    <web-resource-collection>
      <web-resource-name>Recipes</web-resource-name>
      <url-pattern>/*.jsp                    </url-pattern>
      <http-method>POST</http-method>
    </web-resource-collection>
```

NONE
- No protection

INTEGRAL
- Data must not be changed along the way

CONFIDENTIAL
- The data should not be seen by anybody

https://tomcat.apache.org/tomcat-8.0-doc/servletapi/javax/servlet/annotation/ServletSecurity.TransportGuarantee.html


1
Client requests /BuyStuff.jsp, a constrained resource that also has a transport guarantee.
HTTP over TCP
POST /BuyStuff.jsp HTTP
The Container sees that this constrained resource has a transport guarantee. The Container sees that the request did NOT come in securely...
container
<user-data-constraint>
<transport-guarantee>
CONFIDENTIAL
</transport-guarantee>
</user-data-constraint>
DD
2
The Container sends a 301 response to the client, that tells the browser to redirect the request using a secure transport.
301 Redirect
Location:HTTPS://...
container
Yes, the "301" is used for normal redirects, but it's ALSO the way the Container tells the browser, "Hey, come back over a secure connection next time and THEN I'll see if we can talk..."
3
The browser makes the same resource request again, but this time, over a secure connection. In other words, the resource stays the same, but the protocol is now HTTPS.
POST /BuyStuff.jsp HTTPS
container
4
Now the Container sees that the resource is constrained, and that this user has not authenticated. So now the Container starts the authentication process by sending a "401" to the browser...
401 Unauthorized
WWW-Authenticate:
Basic realm ="user"
container
5
The browser makes the same request again, (yes, for the THIRD time) but this time the request has the user's login data in the header AND the request comes over using a secure connection. So this time the client's login data is transmitted securely!
POST /BuyStuff.jsp
Authorization:
Basic: x5w3..=
container
Bottom line: when a request comes in, the Container looks FIRST at the <transport-guarantee>, and if there IS one, the Container tries to deal with that issue first by asking, "Is this request over a secure connection?" If not, the Container doesn't even bother to look at authentication/authorization info. It just tells the client "Come back when you're secure, then we'll talk..."

STEPS

- GET/buy.jsp is sent to the server
- http 301 is sent back to the client
- GET/buy.jsp is sent in SSL connection
- Server returns the http 401 asking for username and password
- Client fills in and sends the request
- Server verifies the name and password and checks for the appropriate role
- Sends back the requested page

SETTINGS FOR 8443

```
keytool -genkey -noprompt -alias tomcat-localhost -keyalg RSA -keystore
C:\Users\chand\localhost-rsa.jks -keypass 123456 -storepass 123456 -dname
"CN=tomcat-cert, OU=JU, O=JU, L=WB, ST=WB, C=IN"
```

```
<Connector
        protocol="org.apache.coyote.http11.Http11NioProtocol"
        port="8443" maxThreads="200"
        scheme="https" secure="true" SSLEnabled="true"
        keystoreFile="C:\my-cert-dir\localhost-rsa.jks"
        keystorePass="123456"
        clientAuth="false" sslProtocol="TLS"/>
```