Java Servlet Technology

Kien Tran
Principal Software Engineer
12/13/2015

Client Logo

Introduction

- Your role
- Your background and experience in the subject
- What do you want from this course

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
- perform objective 1
- perform objective 2

Agenda

Course Audience and Prerequisite

- The course is for <whom>
- The following are prerequisites to <course>:
  - <knowledge>
  - <experiences>
  - <course>
  - …

Assessment Disciplines

- Class Participation: <%>
- Assignment: <%>
- Final Exam: <%>
- Passing Scores: <%>

Duration and Course Timetable

- Course Duration: <hrs>
- Course Timetable:
  - From <time> to <time>
  - Break <x> minutes from <time> to <time>

Further References

- <Source 1>
- <Source 2>
- …

Set Up Environment

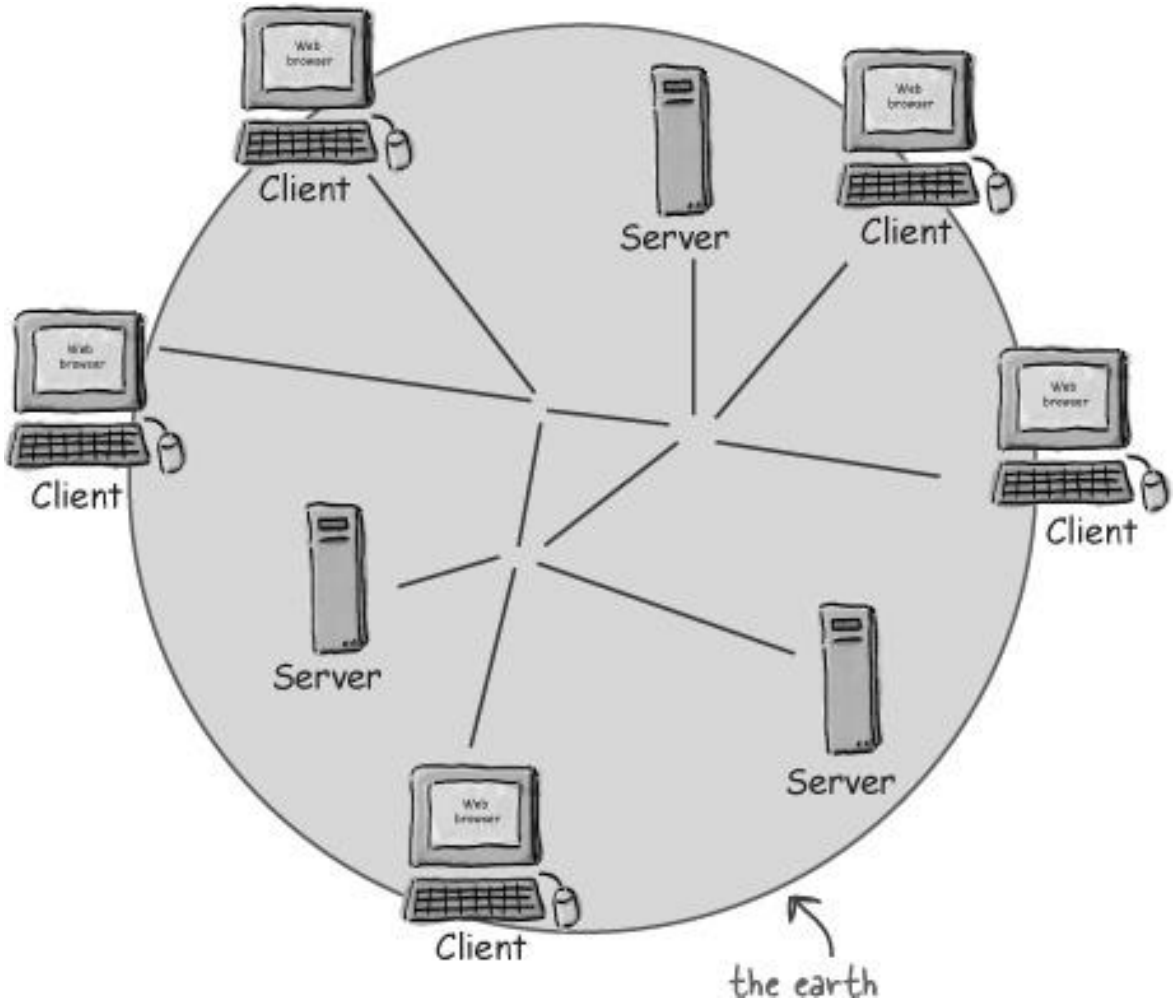
- To complete the course, your PC must install:
  - Software 1
  - Software 2
  - …

Course Administration

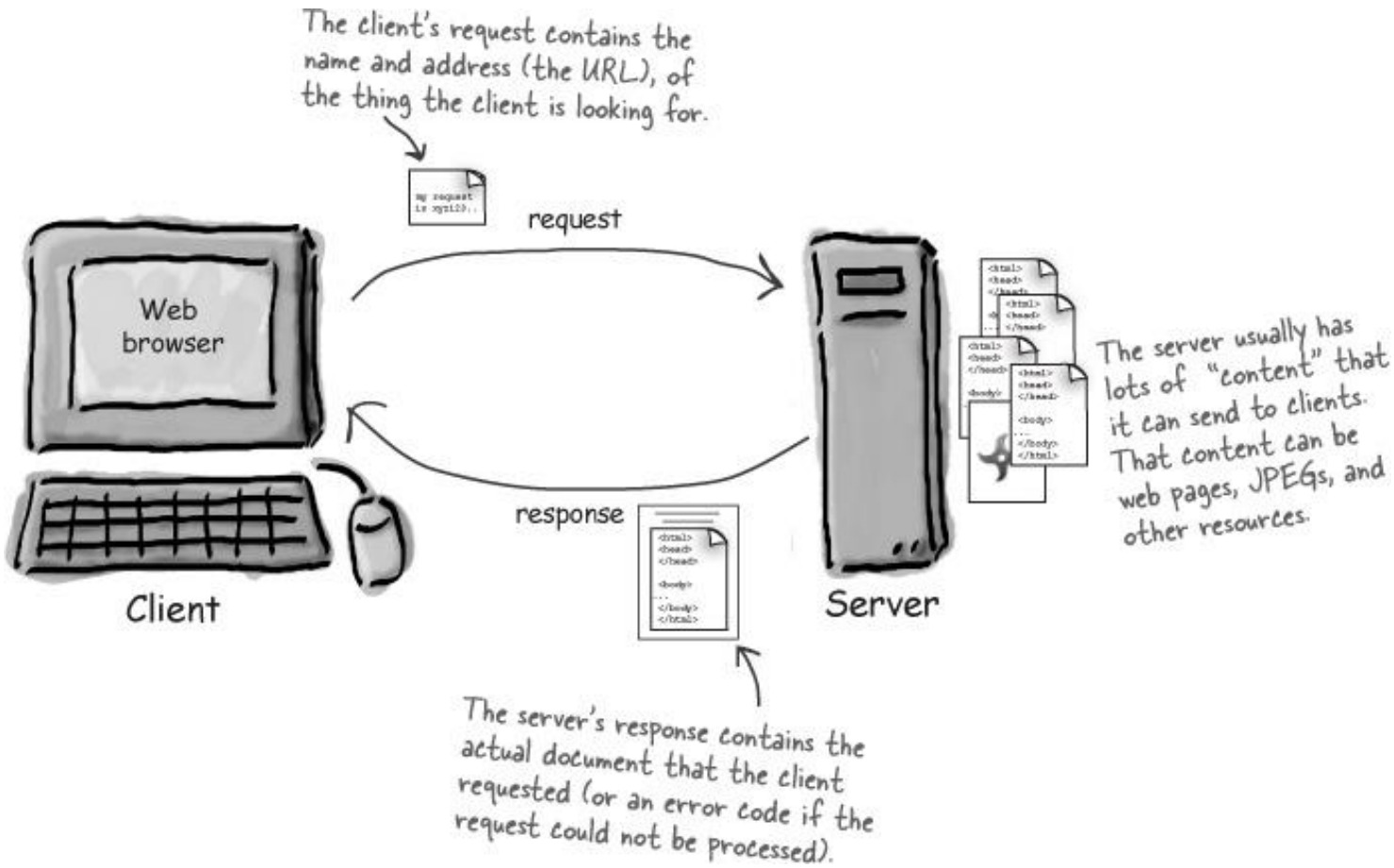
- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

Overview

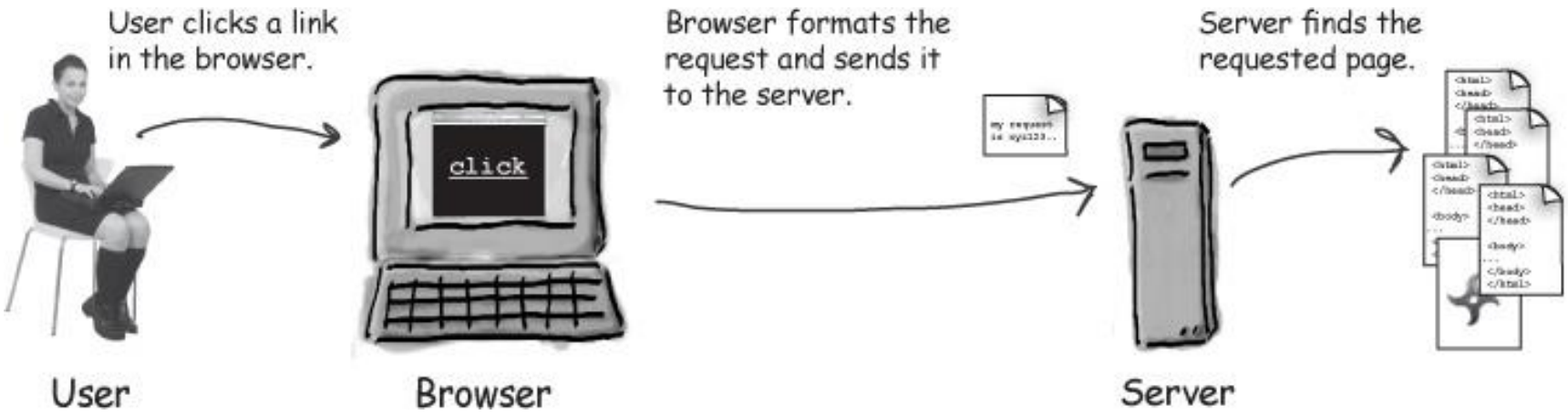
World Wide Web



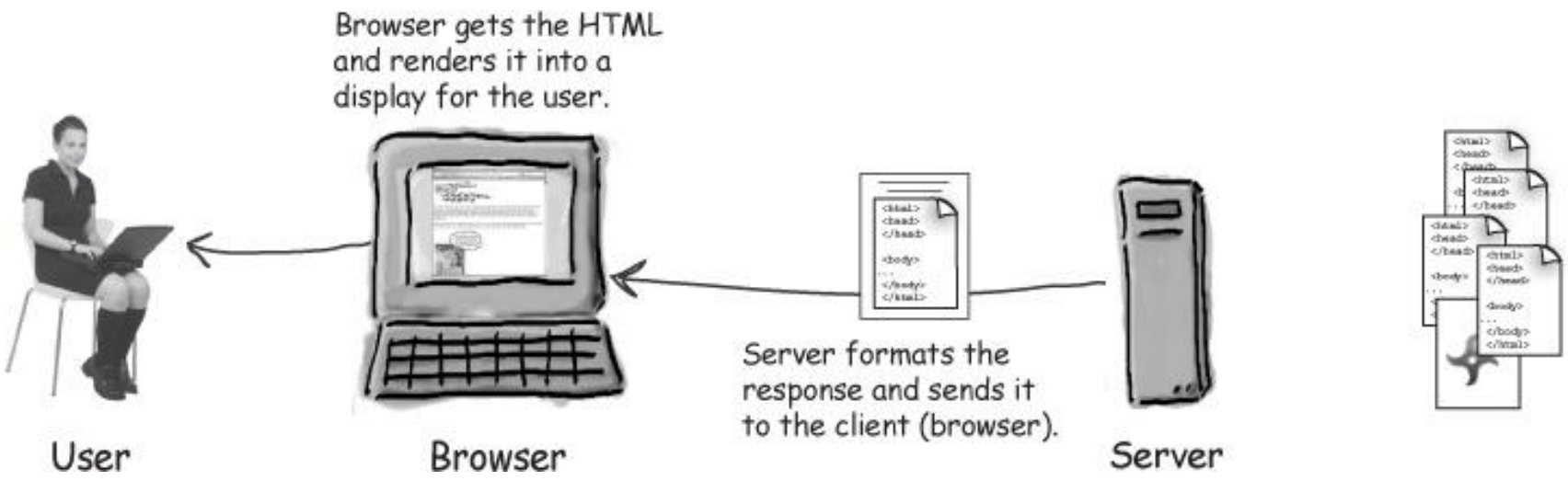
Client - Server



Client - Server



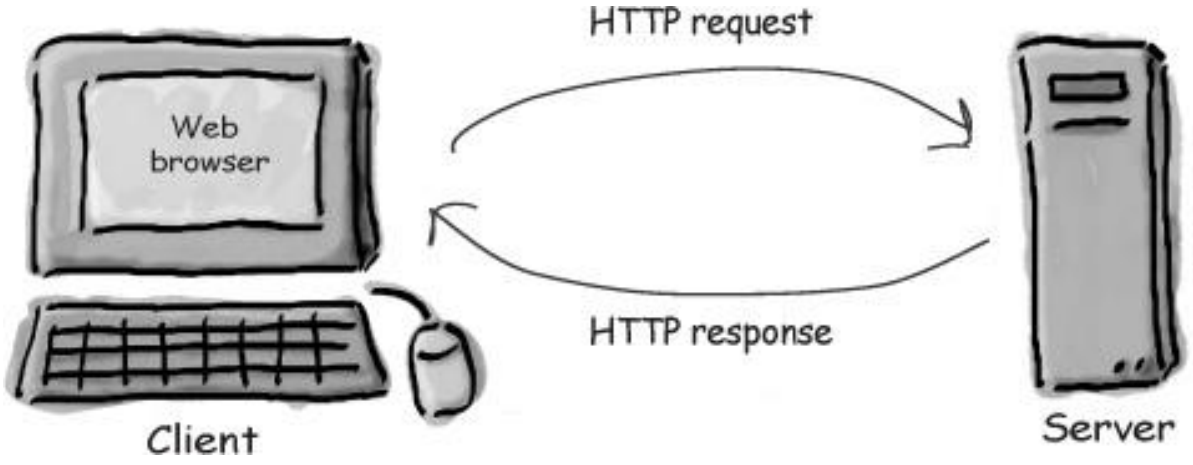
Client - Server



HTML and HTTP

- HTML (HyperText Markup Language).
  - The HTML tells the browser how to present the content to the user.

- HTTP (HyperText Transfer Protocol)
  - HTTP is the protocol clients and servers use on the web to communicate.
  - The server uses HTTP to send HTML to the client.



URL

Protocol: Tells the server which communications protocol (in this case HTTP) will be used.

Port: This part of the URL is optional. A single server supports many ports. A server application is identified by a port. If you don't specify a port in your URL, then port 80 is the default, and as luck would have it, that's the default port for web servers.

Resource: The name of the content being requested. This could be an HTML page, a servlet, an image, PDF, music, video, or anything else the server feels like serving. If this optional part of the URL is left out, most web servers will look for index.html by default.

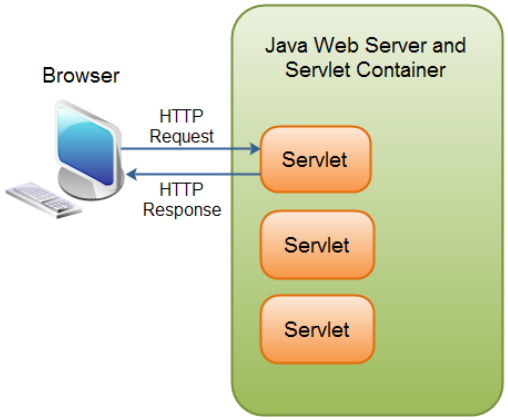
```
http://www.wickedlysmart.com:80/beeradvice/select/beer1.html
```

Server: The unique name of the physical server you're looking for. This name maps to a unique IP address. IP addresses are numeric and take the form "xxx.yyy.zzz.aaa". You can specify an IP address here instead of a server name, but a server name is a lot easier to remember.

Path: The path to the location, on the server, of the resource being requested. Because most of the early servers on the web ran Unix, Unix syntax is still used to describe the directory hierarchies on the web server.

Introduction to Servlet

Java Servlet

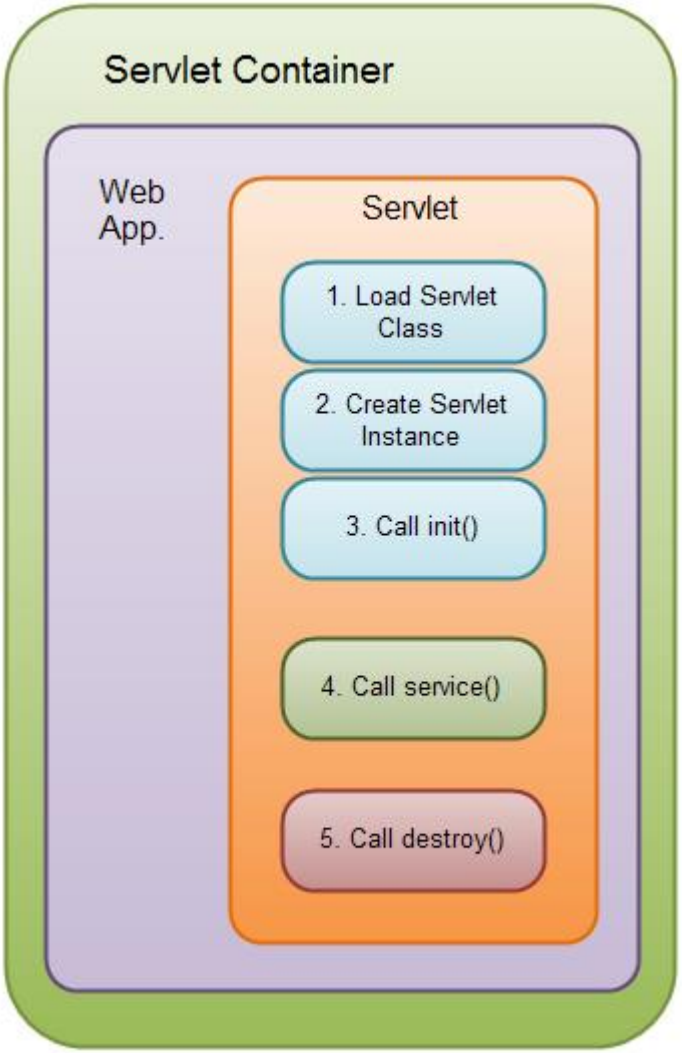


- A technology used to create Java web application
- A Java programming language class
- A class that extend the capabilities of the servers and respond to the incoming request
  - E.g. HTTP Request & Response
- A class that implements the javax.servlet.Servlet or extends javax.servlet.http.HttpServlet
- Servlet Container executes servlets and manages their life cycle

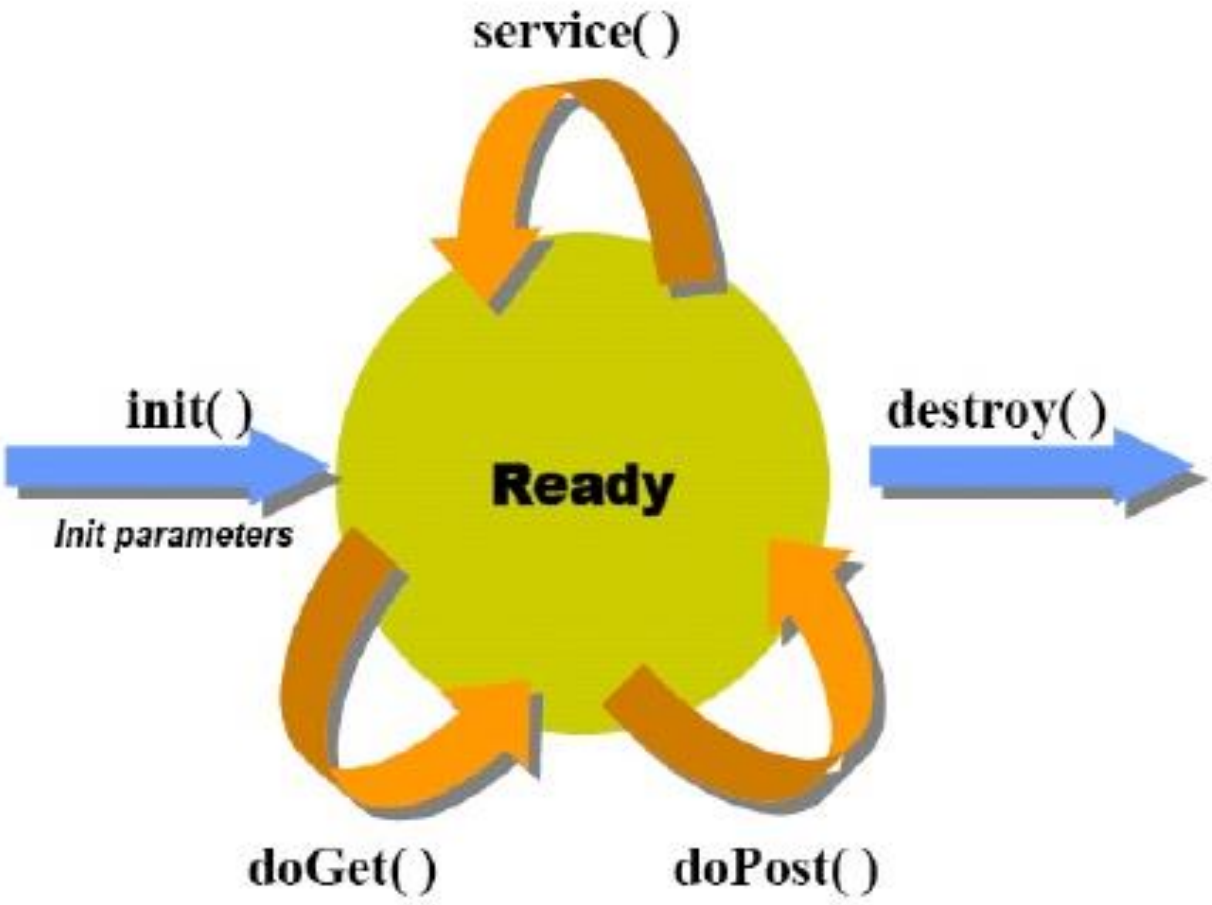
Servlet Container

- Container receives an HTTP request from a client
- Container forwards the request to the corresponding servlet
- Servlet processes the request and generates HTML document
- Container returns the page to the client

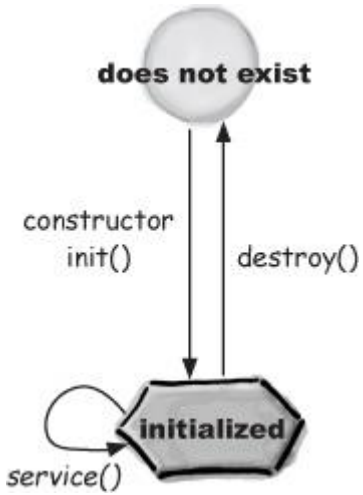
Servlet Lifecycle

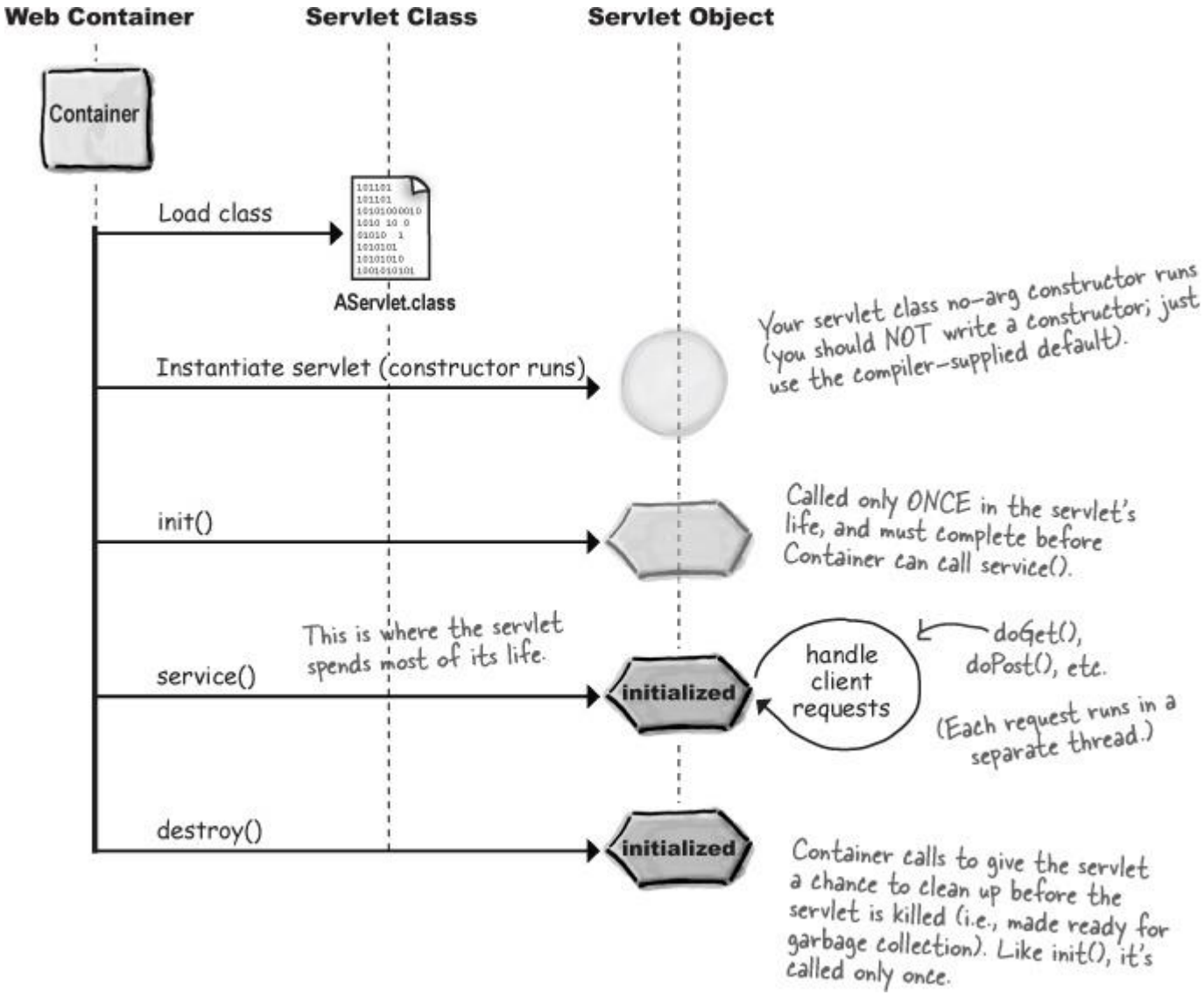


Servlet Lifecycle



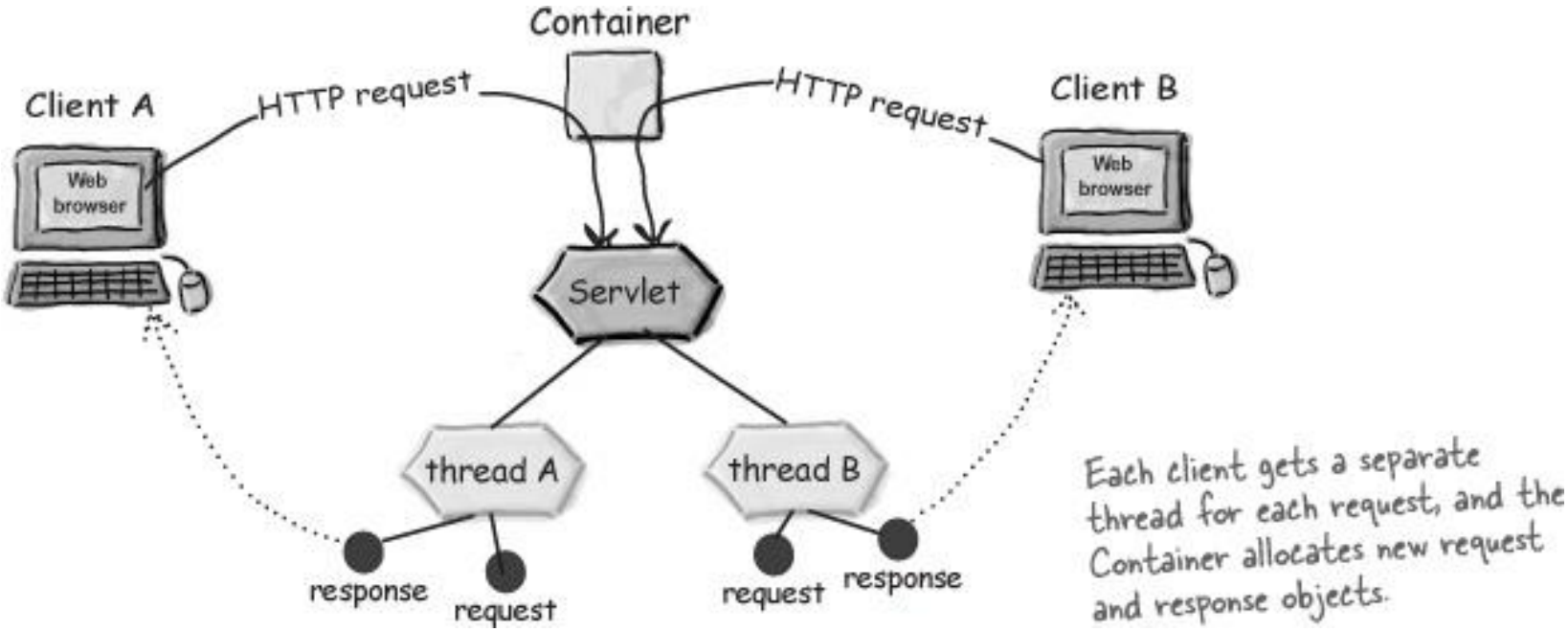
A Servlet's Life





Servlet Lifecycle

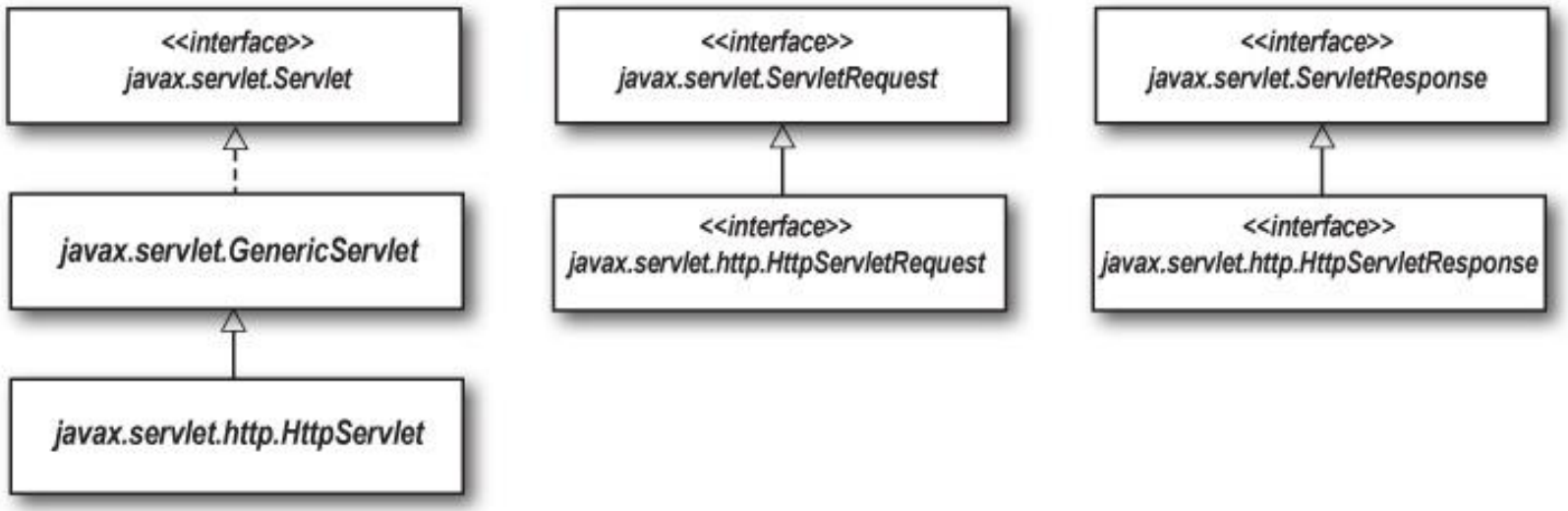
- The Container runs multiple threads to process multiple requests to a single servlet.



Servlet Lifecycle

- Servlet must implement javax.servlet.Servlet interface
- It must implements methods called by servlet container
  - Init()
    - Called only the first time there is a request for the servlet
  - Service()
    - Called for each request
    - Receives the request, processes it and generates a response
  - Destroy()
    - Called when servlet is terminated
    - Releases all resources

Servlet Key API



HttpServlet

In the real world, 99.9% of all servlets override either the doGet() or doPost() method.

99.9999% of all servlets are HttpServlets.

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

public class Ch2Servlet extends HttpServlet {

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

    PrintWriter out = response.getWriter();
    java.util.Date today = new java.util.Date();
    out.println("<html> " +
              "<body>" +
              "<h1 style="text-align:center>" +
              "HF\'s Chapter2 Servlet</h1>" +
              "<br>" + today +
              "</body>" +
              "</html>");
  }
}
```

This is where your servlet gets references to the request and response objects which the container creates.

You can get a PrintWriter from the response object your servlet gets from the Container. Use the PrintWriter to write HTML text to the response object. (You can get other output options, besides PrintWriter, for writing, say, a picture instead of HTML text.)

Map URLs to Servlets

- Deployment Descriptor (DD)

<servlet>

maps internal name to fully-qualified class name

<servlet-mapping>

maps internal name to public URL name

Deployment Descriptor (DD)

You do NOT have to memorize any of this opening tag, ever. Just copy it in when you're using a Container that's compliant with servlet spec 2.4 (like Tomcat 5).

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

  <servlet>
    <servlet-name>Ch3 Beer</servlet-name>
    <servlet-class>com.example.web.BeerSelect</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Ch3 Beer</servlet-name>
    <url-pattern>/SelectBeer.do</url-pattern>
  </servlet-mapping>

</web-app>
```

Map URLs to Servlets

This web app has two servlets.

There is a LOT more that goes into this opening <web-app> tag, but we don't want to show it right now (there's an example at the end of this chapter).

```
<web-app ...>

  <servlet>
    <servlet-name>Internal name 1</servlet-name>
    <servlet-class>foo.Servlet1</servlet-class>
  </servlet>

  <servlet>
    <servlet-name>Internal name 2</servlet-name>
    <servlet-class>foo.Servlet2</servlet-class>
  </servlet>
  ..........................................................

  <servlet-mapping>
    <servlet-name>Internal name 1</servlet-name>
    <url-pattern>/Public1</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>Internal name 2</servlet-name>
    <url-pattern>/Public2</url-pattern>
  </servlet-mapping>

</web-app>
```

The <servlet-name> element is used to tie a <servlet> element to a specific <servlet-mapping> element. The end-user NEVER sees this name; it's used only in other parts of the DD.

The <servlet> element tells the Container which class files belong to a particular web application.

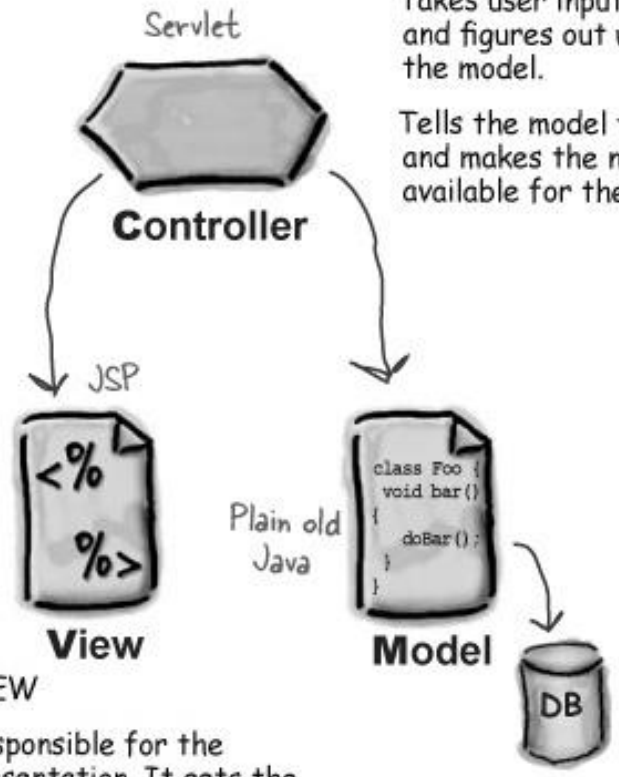
You put in the fully-qualified name of the class (but you don't add the ".class" extension).

Think of the <servlet-mapping> element as what the Container uses at runtime when a request comes in, to ask, "which servlet should I invoke for this requested URL?".

This is what the client sees (and uses) to get to the servlet... but it's a made-up name that is NOT the name of the actual servlet class.

It's possible to use wildcards in the <url-pattern> element... more on that and paths later.

MVC in the Servlet & JSP world



CONTROLLER

Takes user input from the request and figures out what it means to the model.

Tells the model to update itself, and makes the new model state available for the view (the JSP).

MODEL

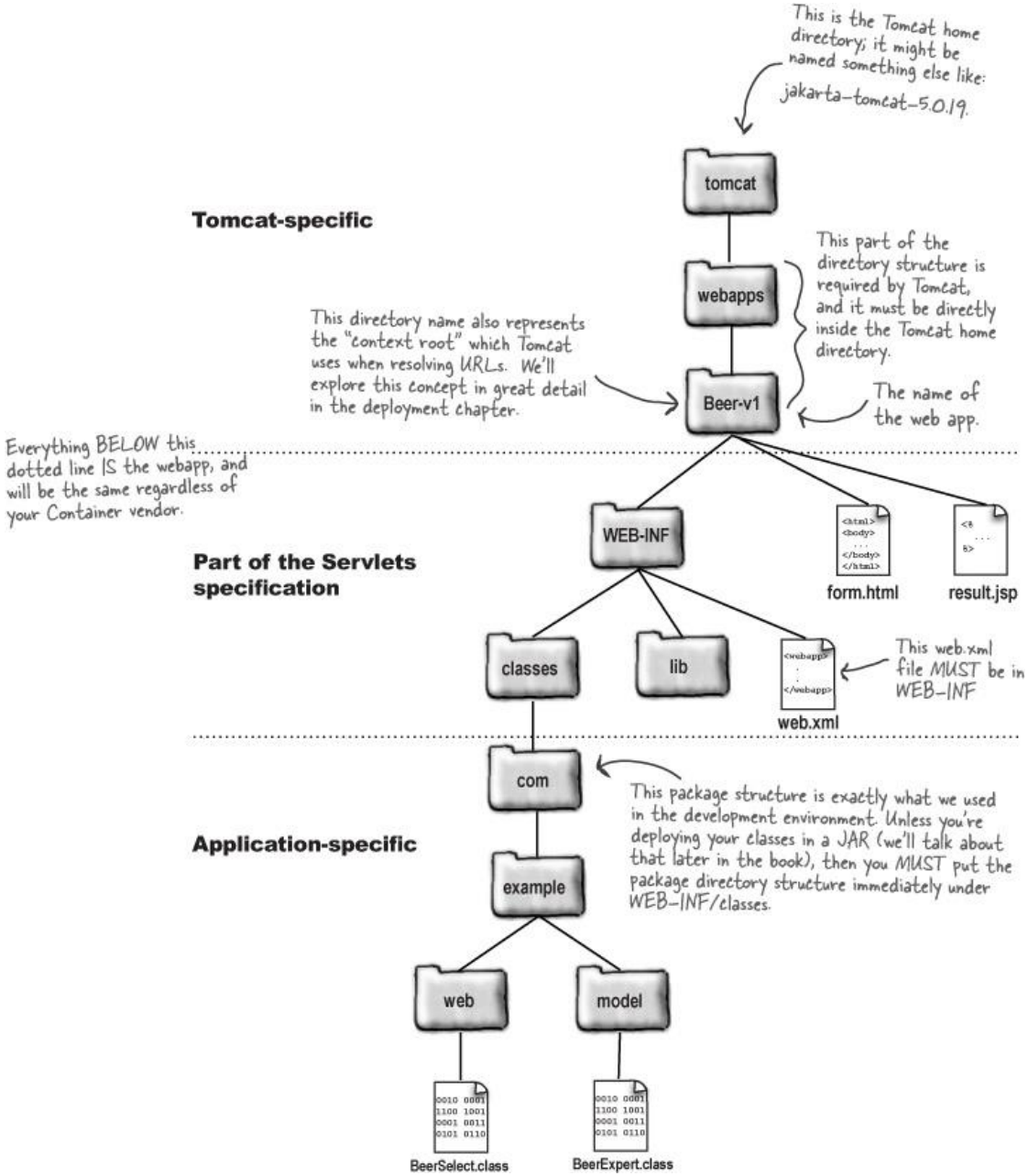
Holds the real business logic and the state. In other words, it knows the rules for getting and updating the state.

A Shopping Cart's contents (and the rules for what to do with it) would be part of the Model in MVC.

It's the only part of the system that talks to the database (although it probably uses *another* object for the actual DB communication, but we'll save *that* pattern for later...)

VIEW

Responsible for the presentation. It gets the state of the model from the Controller (although not directly; the Controller puts the model data in a place where the View can find it). It's also the part that gets the user input that goes back to the Controller.


This is the Tomcat home directory; it might be named something else like: jakarta-tomcat-5.0.19.
Tomcat-specific
tomcat
webapps
This part of the directory structure is required by Tomcat, and it must be directly inside the Tomcat home directory.
This directory name also represents the "context root" which Tomcat uses when resolving URLs. We'll explore this concept in great detail in the deployment chapter.
Beer-v1
The name of the web app.
Everything BELOW this dotted line IS the webapp, and will be the same regardless of your Container vendor.
Part of the Servlets specification
WEB-INF
form.html
result.jsp
classes
lib
web.xml
This web.xml file MUST be in WEB-INF
Application-specific
com
This package structure is exactly what we used in the development environment. Unless you're deploying your classes in a JAR (we'll talk about that later in the book), then you MUST put the package directory structure immediately under WEB-INF/classes.
example
web
model
BeerSelect.class
BeerExpert.class

HTTP Methods

doGet() or doPost()

- The client's request (HTTP Method).
  - If the HTTP Method is a GET, the service() method calls doGet().
  - If the HTTP request Method is a POST, the service() method calls doPost().

Anatomy of an HTTP GET request

The Request line.

The HTTP Method.

The path to the resource on the web server.

In a GET request, parameters (if there are any) are appended to the first part of the request URL, starting with a "?". Parameters are separated with an ampersand "&".

The protocol version that the web browser is requesting.

GET /select/selectBeerTaste.jsp**?color=dark&taste=malty** HTTP/1.1

The Request headers.

Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

Anatomy of an HTTP POST request

The HTTP Method.

The path to the resource on the web server.

The protocol version that the web browser is requesting.

The Request line.

```
POST /advisor/selectBeerTaste.do HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/
20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

color=dark&taste=malty
```

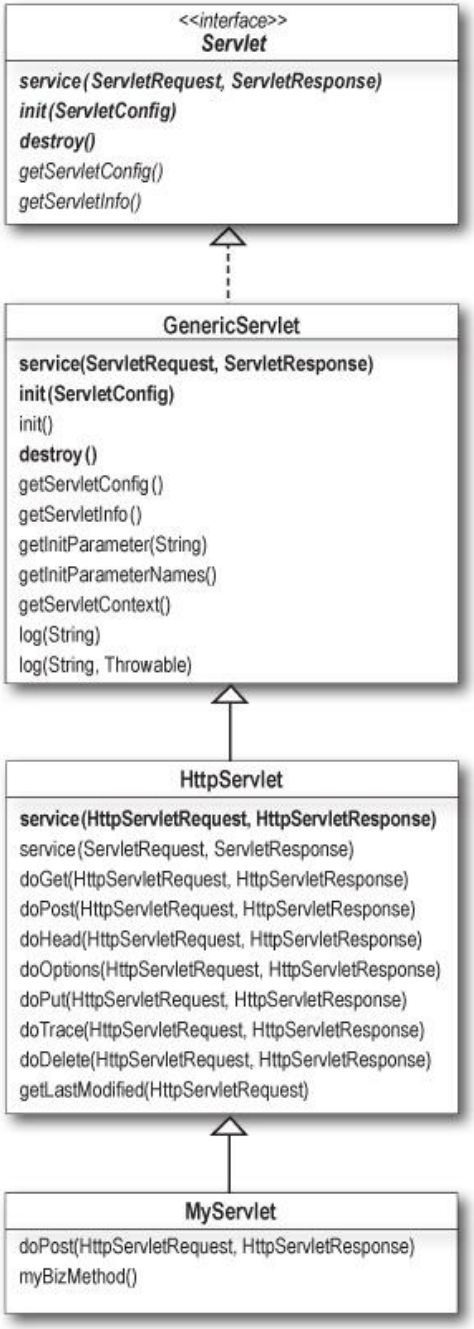
The Request headers.

The message body, sometimes called the "payload".

This time, the parameters are down here in the body, so they aren't limited the way they are if you use a GET and have to put them in the Request line.

Methods of HttpServlet class

- GET - doGet(HttpServletRequest req, HttpServletResponse res)
- HEAD - doHead(HttpServletRequest req, HttpServletResponse res)
- POST - doPost(HttpServletRequest req, HttpServletResponse res)
- PUT - doPut(HttpServletRequest req, HttpServletResponse res)
- DELETE - doDelete(HttpServletRequest req, HttpServletResponse res)
- …



Servlet interface
(javax.servlet.Servlet)

The Servlet interface says that all servlets have these five methods (the three in bold are lifecycle methods).

GenericServlet class
(javax.servlet.GenericServlet)

GenericServlet is an abstract class that implements most of the basic servlet methods you'll need, including those from the Servlet interface. You will probably NEVER extend this class yourself. Most of your servlet's "servlet behavior" comes from this class.

HttpServlet class
(javax.servlet.http.HttpServlet)

HttpServlet (also an abstract class) implements the service() method to reflect the HTTPness of the servlet--the service() method doesn't take just ANY old servlet request and response, but an HTTP-specific request and response.

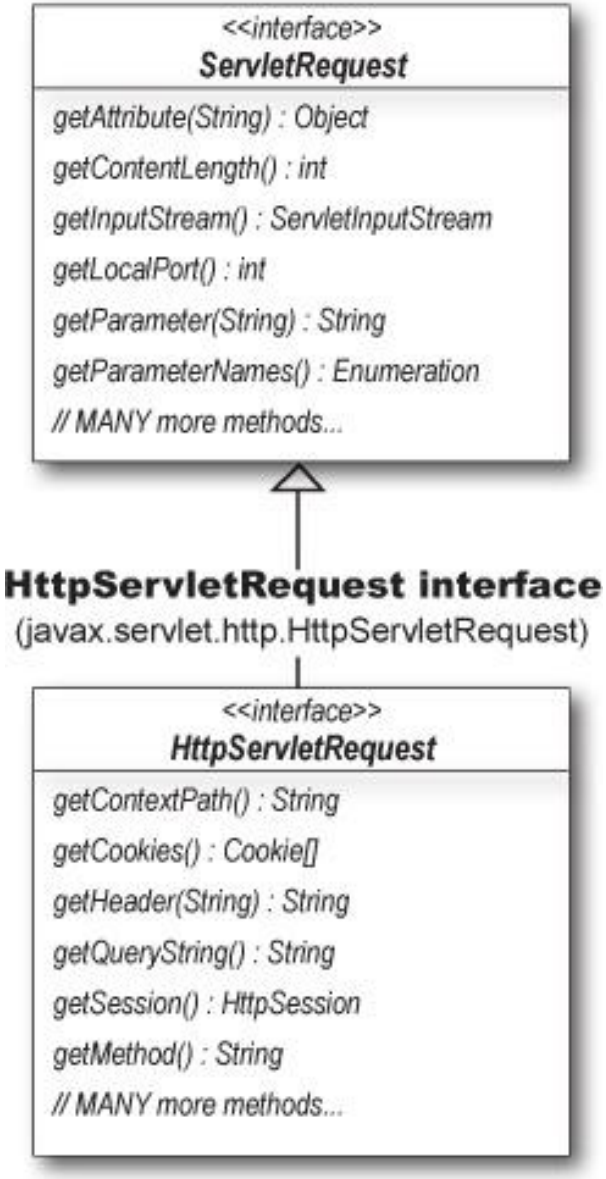
MyServlet class
(com.wickedlysmart.foo)

Most of your servletness is handled by superclass methods. All you do is override the HTTP methods you need.

Servlets – Client HTTP Request

ServletRequest interface

(javax.servlet.ServletRequest)



Method & Description

- **String getParameter(String name)**
  - Returns the value of a request parameter as a String, or null if the parameter does not exist.
- **String[] getParameterValues(String name)**
  - Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
- **Enumeration getParameterNames()**
  - Returns an Enumeration of String objects containing the names of the parameters contained in this request.

Method & Description

- **Object getAttribute(String name)**
  - Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.

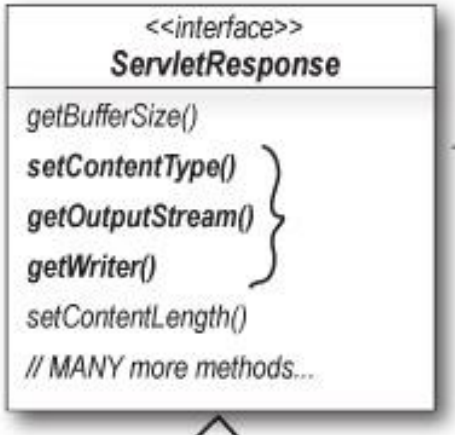
- **HttpSession getSession()**
  - Returns the current session associated with this request, or if the request does not have a session, creates one.

- **HttpSession getSession(boolean create)**
  - Returns the current HttpSession associated with this request or, if there is no current session and value of create is true, returns a new session.

Servlets – Server HTTP Response

ServletResponse interface

(javax.servlet.ServletResponse)



These are some of the most commonly-used methods.

HttpServletResponse interface

(javax.servlet.http.HttpServletResponse)

Sometimes you'll use these too...

Method & Description

- PrintWriter writer = response.getWriter()
  - To send HTML back to the browser

- OutputStream outputStream = response.getOutputStream()
  - To send binary data back to the browser (e.g. Media, PDF, Word, Excel files)

Redirecting to a Different URL

- Redirect the browser to a different resource from servlet (e.g. Servlet, JSP or HTML file)
- It accepts relative as well as absolute URL.
- It works at client side
- **response.sendRedirect("http://abc.com")**

ServletConfig

ServletConfig

- javax.servlet.ServletConfig is used to pass configuration information (web.xml) to Servlet.

- Every servlet has it’s own ServletConfig object and servlet container is responsible for instantiating this object.

Methods of ServletConfig

- String getInitParameter(String name)
  - Returns the parameter value for the specified parameter name.
- Enumeration getInitParameterNames()
  - Returns an enumeration of all the initialization parameter names.
- ServletContext getServletContext()
  - Returns an object of ServletContext.

Initialization parameter for a servlet

```
<web-app>
  <servlet>
    ......

    <init-param>
      <param-name>parametername</param-name>
      <param-value>parametervalue</param-value>
    </init-param>
    ......
  </servlet>
</web-app>
```

Get initialization parameter

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
…
    ServletConfig config = getServletConfig();
    String driver = config.getInitParameter("driver");
…
}
```

ServletContext

ServletContext

- ServletContext is created by the web container at time of deploying the project
- The ServletContext is unique object and available to all the servlets in the web application
- How to get the object of ServletContext
  - ServletContext application = getServletConfig().getServletContext()

Methods of ServletContext

- String getInitParameter(String name)
  - Returns the parameter value for the specified parameter name
- Enumeration getInitParameterNames()
  - Returns the names of the context's initialization parameters
- void setAttribute(String name,Object object)
  - Sets the given object in the application scope
- Object getAttribute(String name)
  - Returns the attribute for the specified name.

Initialization parameter in Context scope

```
<web-app>
 ......

  <context-param>
    <param-name>parametername</param-name>
    <param-value>parametervalue</param-value>
  </context-param>
 ......
</web-app>
```

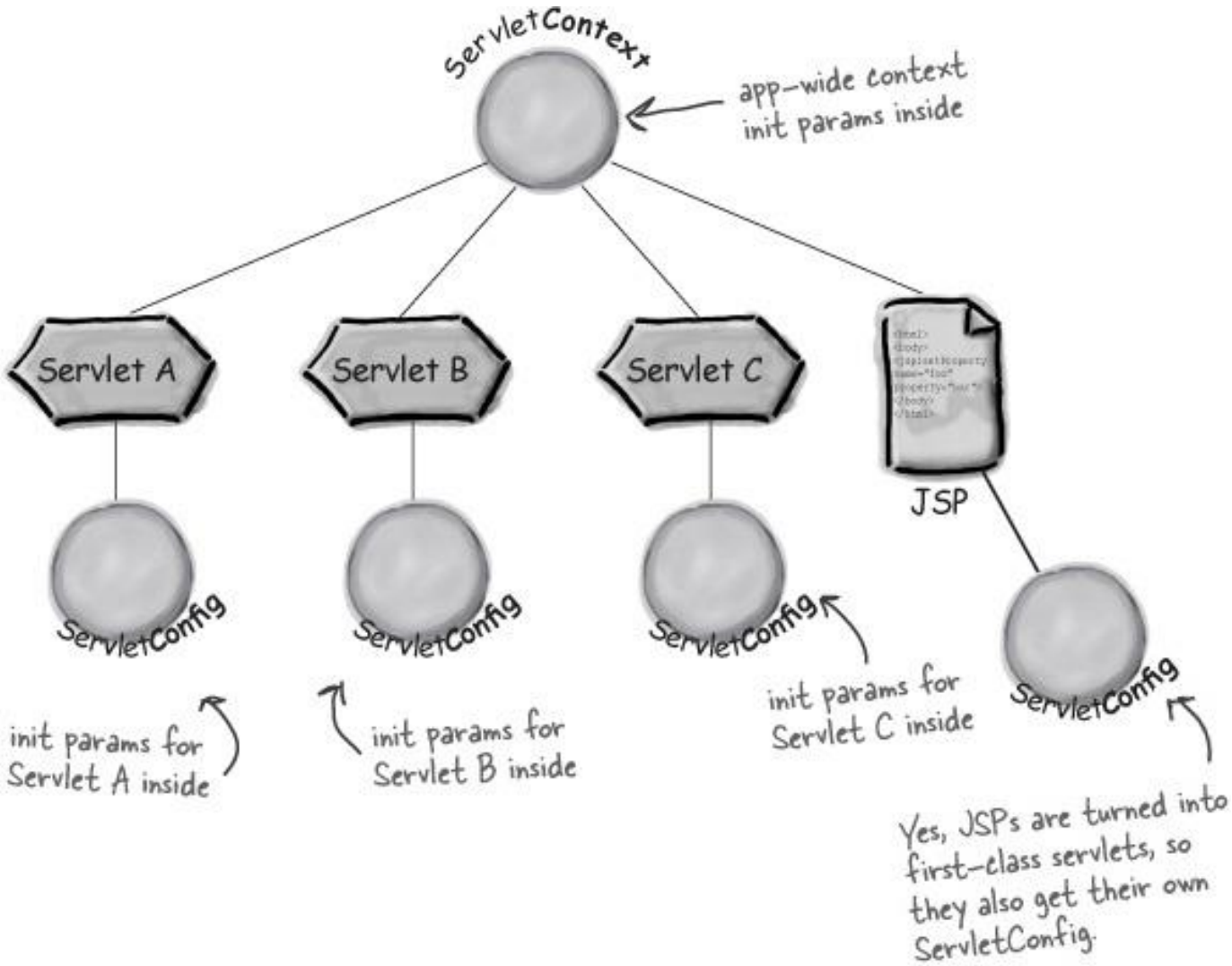
Get the initialization parameter

```
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
…
ServletContext context=getServletContext();

String driverName=context.getInitParameter("dname");


…
}
```

ServletConfig & ServletContext

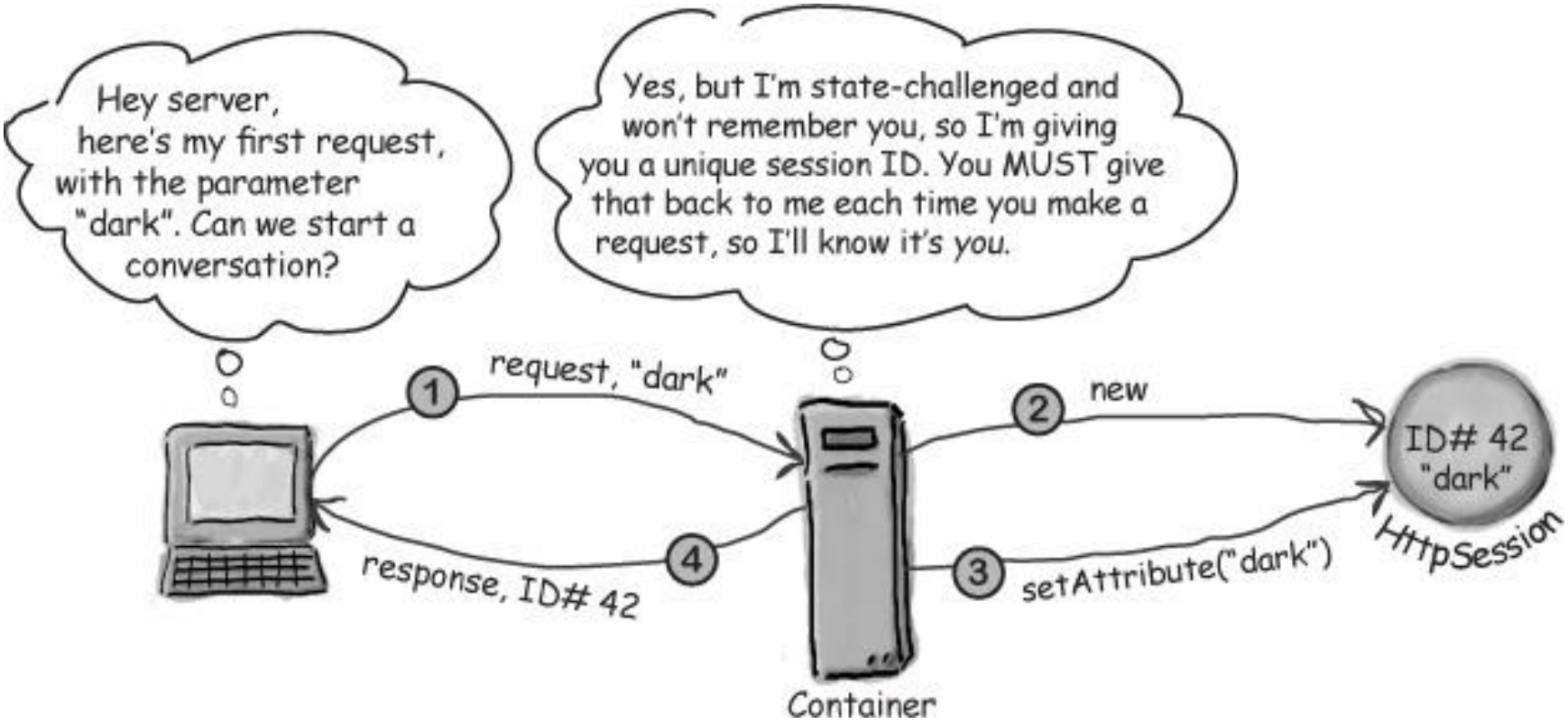
ServletConfig	ServletContext
ServletConfig object is one per servlet class	ServletContext object is global to entire web application
Object of ServletConfig will be created during initialization process of the servlet	Object of ServletContext will be created at the time of web application deployment
Scope: As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.	*Scope*: As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.
In web.xml – <*init-param*> tag will be appear under <*servlet-class*> tag	In web.xml – <*context-param*> tag will be appear under <*web-app*> tag

ServletConfig is one per servlet ServletContext is one per web app

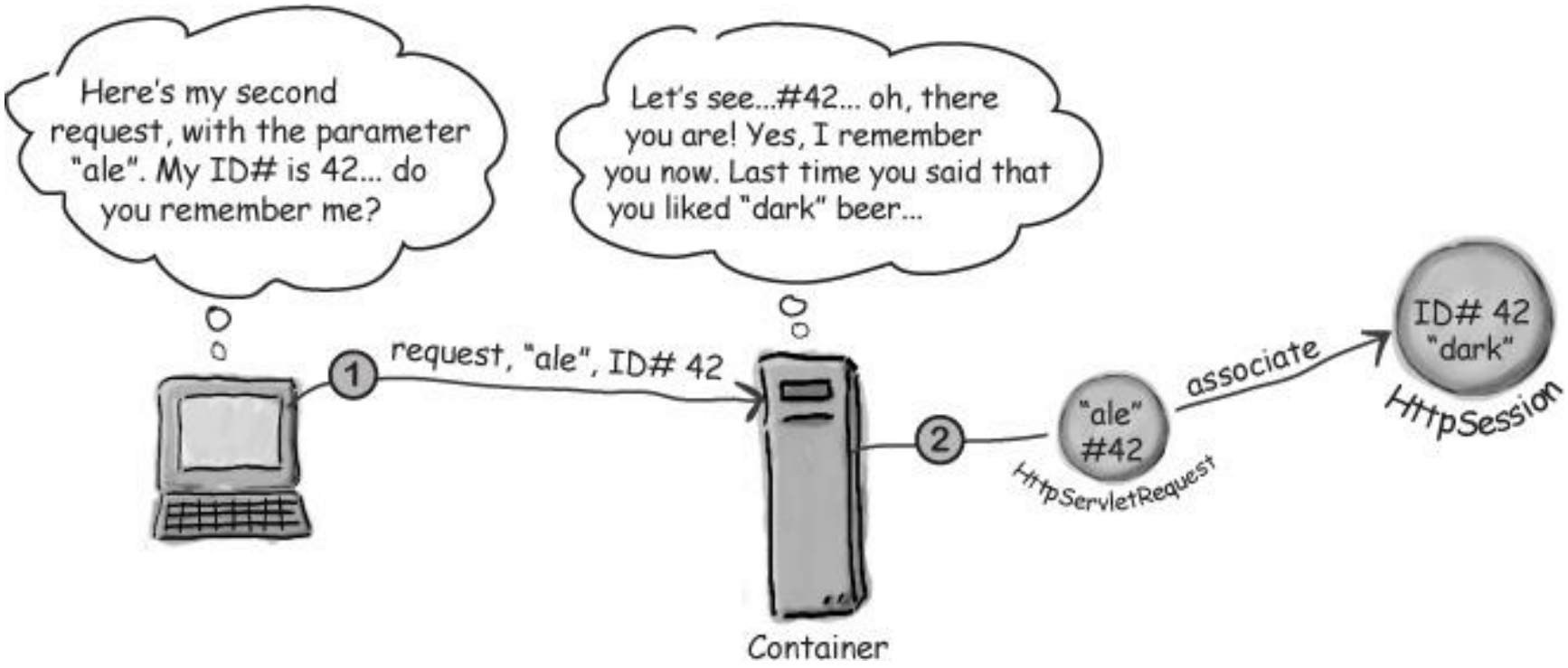


Session

How does the Container know who the client is?



How does the Container know who the client is?

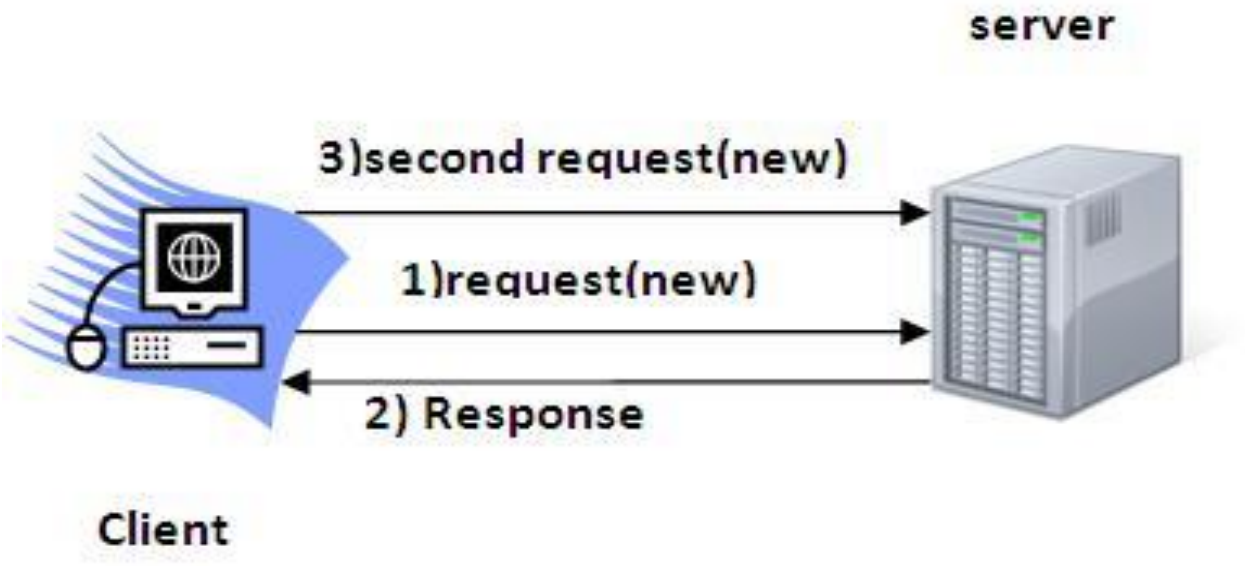


How do the Client and Container exchange Session ID info?

How do the Client and Container exchange Session ID info?

Session Tracking

- Session tracking is used to maintain the state of an user. It is known as session management.

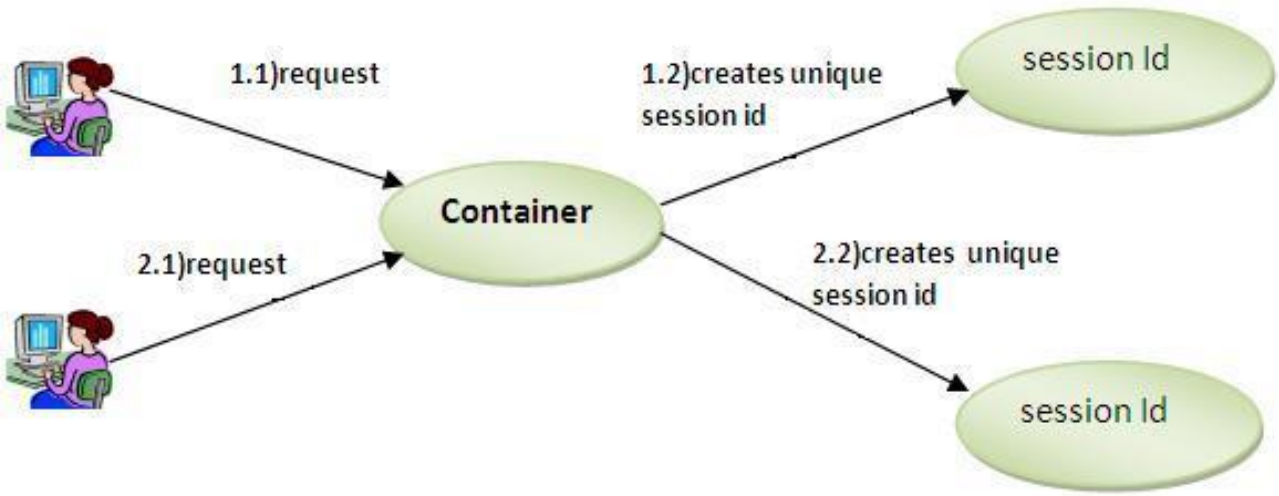


Session Tracking Techniques

- Cookies
- Hidden Form Field
- URL Rewriting
- HttpSession

HttpSession

- The HttpSession object represents a user session.
- A user session contains information about the user across multiple HTTP requests.



HttpSession

- How to access the session object
  - HttpSession session = request.getSession()
- How to store values in the session object
  - session.setAttribute("userName", "theUserName")
- How to retrieve values from the session object
  - String userName = (String) session.getAttribute("userName")

Three ways a session can die

- It times out
- You call invalidate() on the session object
- The application goes down (crashes or is undeployed)

Session Timeout

- Configuring session timeout in the DD

```
<web-app ...>
  <servlet>
    ...
  </servlet>
 <session-config>
    <session-timeout>15</session-timeout>
  </session-config>
</web-app>
```

The "15" is in minutes. This says if the client doesn't make any requests on this session for 15 minutes, kill it.*

Session Timeout

- Setting session timeout for a specific session

```
session.setMaxInactiveInterval(20*60);
```

Only the session on which you call the method is affected.

The argument to the method is in seconds, so this says if the client doesn't make any requests on the session for 20 minutes, kill it.*

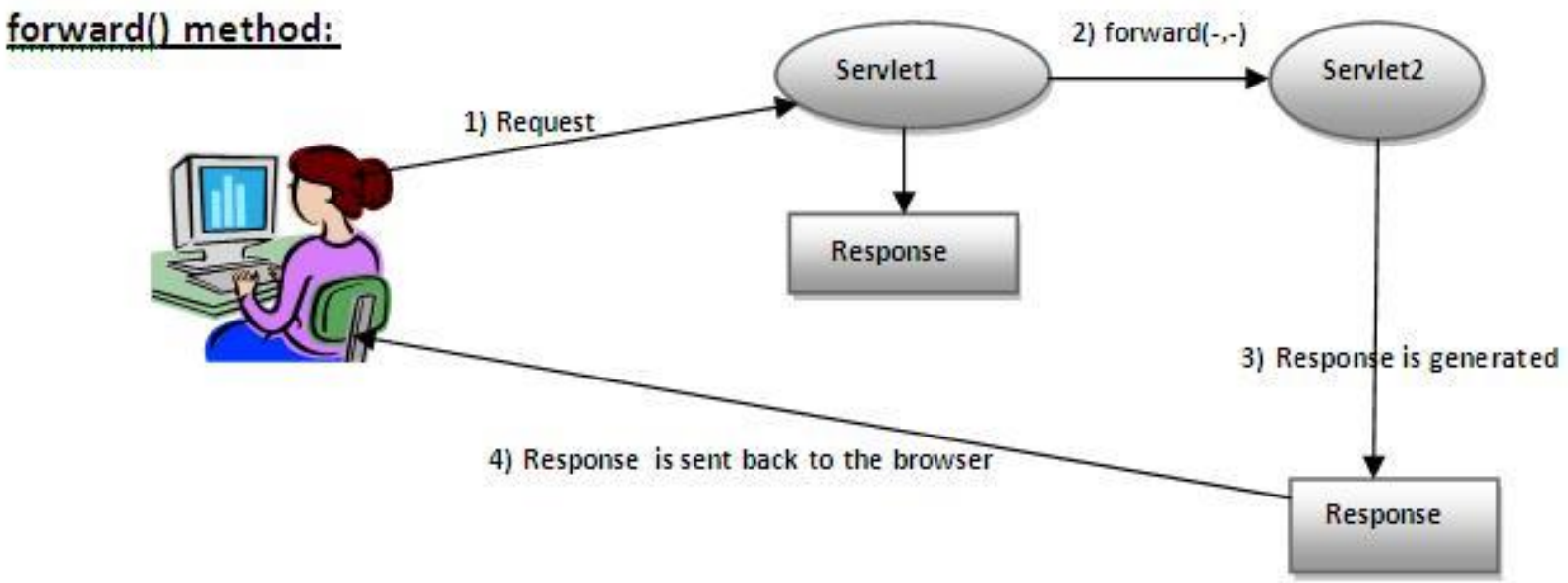
RequestDispatcher

RequestDispatcher

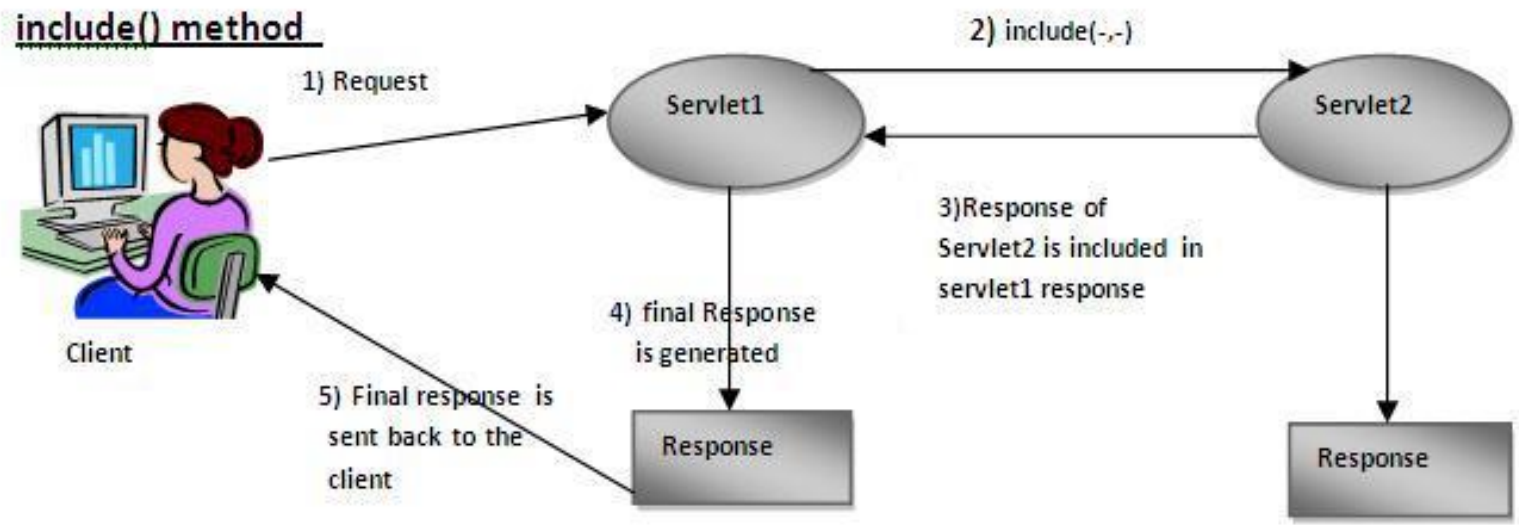
- The facility of dispatching the request to another resource it may be html, servlet or JSP.

- Methods of RequestDispatcher interface
  - forward(ServletRequest req,ServletResponse res)
    - Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
  - include(ServletRequest req,ServletResponse res)
    - Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

Forward


forward() method:
1) Request
Servlet1
2) forward(-,-)
Servlet2
Response
3) Response is generated
4) Response is sent back to the browser
Response

Include


include() method
1) Request
2) include(-,-)
Servlet1
Servlet2
3)Response of
Servlet2 is included in
servlet1 response
Client
4) final Response
is generated
5) Final response is
sent back to the
client
Response
Response

RequestDispatcher

- To obtain a RequestDispatcher from the HttpServletRequest object
  - RequestDispatcher requestDispatcher =
    request.getRequestDispatcher("/abc")
  - requestDispatcher.forward(request, response)
  - requestDispatcher.include(request, response);

Forward vs. Redirect

- Forward
  - A forward is performed internally by the application (servlet).
  - The browser is completely unaware that it has taken place, so its original URL remains intact
  - Any browser reload of the resulting page will simple repeat the original request, with the original URL

Forward vs. Redirect

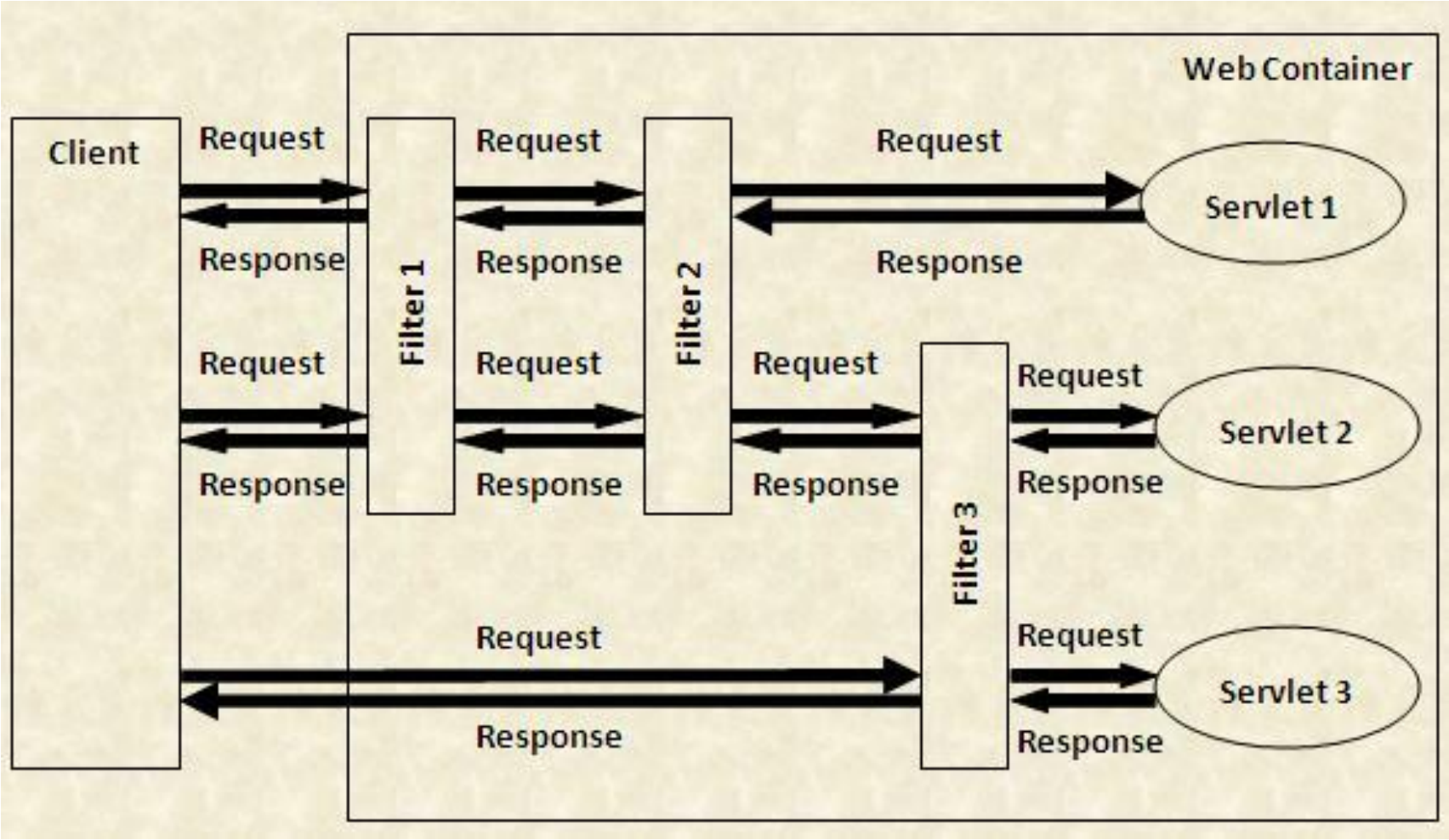
- Redirect
  - A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original
  - A browser reload of the second URL will not repeat the original request, but will rather fetch the second URL
  - Redirect is marginally slower than a forward, since it requires two browser requests, not one
  - Objects placed in the original request scope are not available to the second request.

Filters

Servlet Filters

- Java classes that can be used in Servlet Programming
- To intercept requests from a client before they access a resource at back end.
- To manipulate responses from server before they are sent back to the client.
- Servlet filter is pluggable, if we remove filter in web.xml, filter is automatically removed.

Servlet Filters



Types of filters

- Authentication Filters
- Data compression Filters
- Encryption Filters
- Image Conversion Filters
- Logging and Auditing Filters

Servlet Filter Methods

- Implement the javax.servlet.Filter interface
- init(FilterConfig filterConfig)
  - This method is called by the web container to indicate to a filter that it is being placed into service.
- doFilter(ServletRequest request, ServletResponse response,
  FilterChain filterChain)
  - This method is called by the container each time a request/response pair is passed through the chain
- destroy()
  - This method is called by the web container to indicate to a filter that it is being taken out of service

Configure a Filter

- <filter> element

```
<filter>
   <filter-name>ValidatorFilter</filter-name>
   <description>Validates the requests</description>
   <filter-class>com.manning.filters.ValidatorFilter</filter-class>
   <init-param>
      <param-name>locale</param-name>
      <param-value>USA</param-value>
   </init-param>
</filter>
```

Configure a Filter

- <filter-mapping> element

```
<filter-mapping>
   <filter-name>ValidatorFilter</filter-name>
   <url-pattern>*.doc</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>ValidatorFilter</filter-name>
   <servlet-name>reportServlet</servlet-name>
</filter-mapping>
```

A Filter Example

```
public class MyFilter implements javax.servlet.Filter {
    public void destroy() {
    }
  public void doFilter(javax.servlet.ServletRequest req,
  javax.servlet.ServletResponse resp, javax.servlet.FilterChain chain) throws
  javax.servlet.ServletException, java.io.IOException {
        System.out.println("do Filter.................");
        chain.doFilter(req, resp);
    }

    public void init(javax.servlet.FilterConfig config) throws
  javax.servlet.ServletException {
                        System.out.println("Init Filter...................");
    }
}
```

A Filter Example

```
<filter>
        <filter-name>test</filter-name>
        <filter-class>MyFilter</filter-class>
</filter>

<filter-mapping>
        <filter-name>test</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

Listener

Overview

- Listeners are the classes which listens to a particular type of events and when that event occurs, triggers the functionality.

- Each type of listener is bind to a type of event

Type of Listeners and Events

- ServletContextListener
  - Interface for receiving notification events about ServletContext lifecycle changes.
  - Event: ServletContextEvent

- ServletContextAttributeListener
  - Interface for receiving notification events about ServletContext attribute changes.
  - Event: ServletContextAttributeEvent

Type of Listeners and Events

- HttpSessionListener
  - Interface for receiving notification events about HttpSession lifecycle changes.
  - Event: HttpSessionEvent

- HttpSessionAttributeListener
  - Interface for receiving notification events about HttpSession attribute changes.
  - Event: HttpSessionBindingEvent

Type of Listeners and Events

- ServletRequestAttributeListener
  - Interface for receiving notification events about ServletRequest attribute changes.
  - Event: ServletRequestAttributeEvent

- HttpSessionBindingListener
  - Causes an object to be notified when it is bound to or unbound from a session.
  - Event: HttpSessionBindingEvent

ServletContextListener

<<interface>> **ServletContextListener**
contextInitialized(ServletContextEvent) *contextDestroyed(ServletContextEvent)*

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.

Listener Class

```
package com.example;

import javax.servlet.*;

public class MyServletContextListener implements ServletContextListener {

   public void contextInitialized(ServletContextEvent event) {

      ServletContext sc = event.getServletContext();

      String dogBreed = sc.getInitParameter("breed");

      Dog d = new Dog(dogBreed);

      sc.setAttribute("dog", d);
   }


  public void contextDestroyed(ServletContextEvent event) {
    // nothing to do here
  }
}
```

Implement javax.servlet.ServletContextListener.

Ask the event for the ServletContext.

Use the context to get the init parameter.

Make a new Dog.

Use the context to set an attribute (a name/object pair) that is the Dog. Now other parts of the app will be able to get the value of the attribute (the Dog).

We don't need anything here. The Dog doesn't need to be cleaned up... when the context goes away, it means the whole app is going down, including the Dog.

Deployment Descriptor

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

  <servlet>
    <servlet-name>ListenerTester</servlet-name>
    <servlet-class>com.example.ListenerTester</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ListenerTester</servlet-name>
    <url-pattern>/ListenTest.do</url-pattern>
  </servlet-mapping>

  <context-param>
    <param-name>breed</param-name>
    <param-value>Great Dane</param-value>
  </context-param>

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

</web-app>
```

Servlet Class

```
public class ListenerTester extends HttpServlet {

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

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

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

    out.println("<br>");

    Dog dog = (Dog) getServletContext().getAttribute("dog");

    out.println("Dog's breed is: " + dog.getBreed());
  }
}
```

don't forget the cast!!

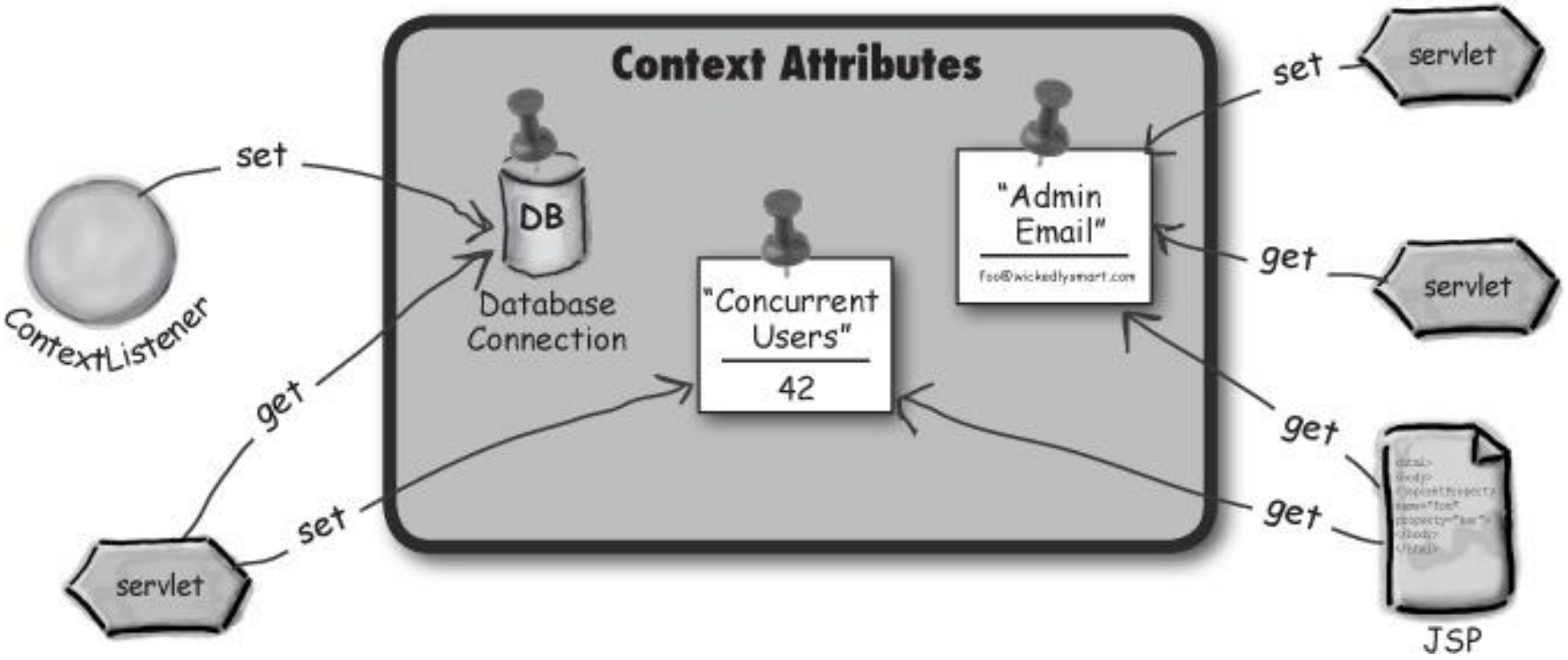
Now we get the Dog from the ServletContext. If the listener worked, the Dog will be there BEFORE this service method is called for the first time.

If things didn't work, THIS is where we'll find out... we'll get a big fat NullPointerException if we try to call getBreed() and there's no Dog.

Object Scope

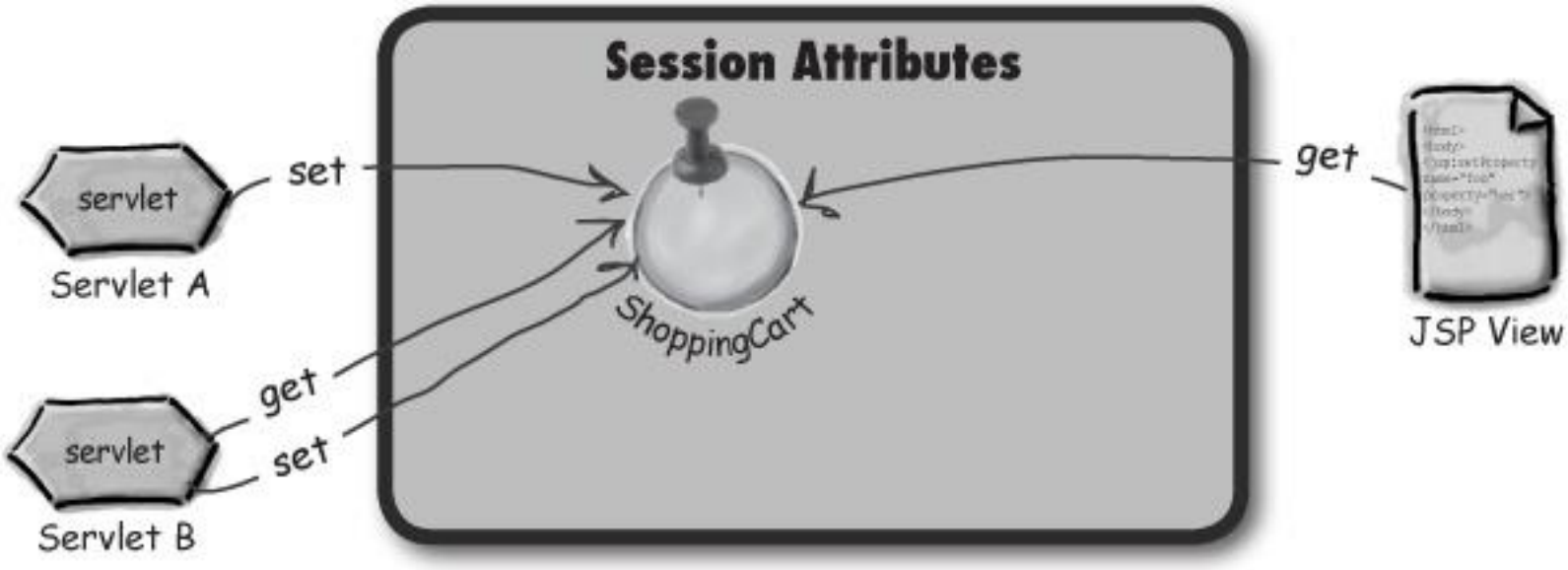
The Three Scopes: Context, Request, and Session

- Everyone in the application has access



The Three Scopes: Context, Request, and Session

- Accessible to only those with access to a specific HttpSession

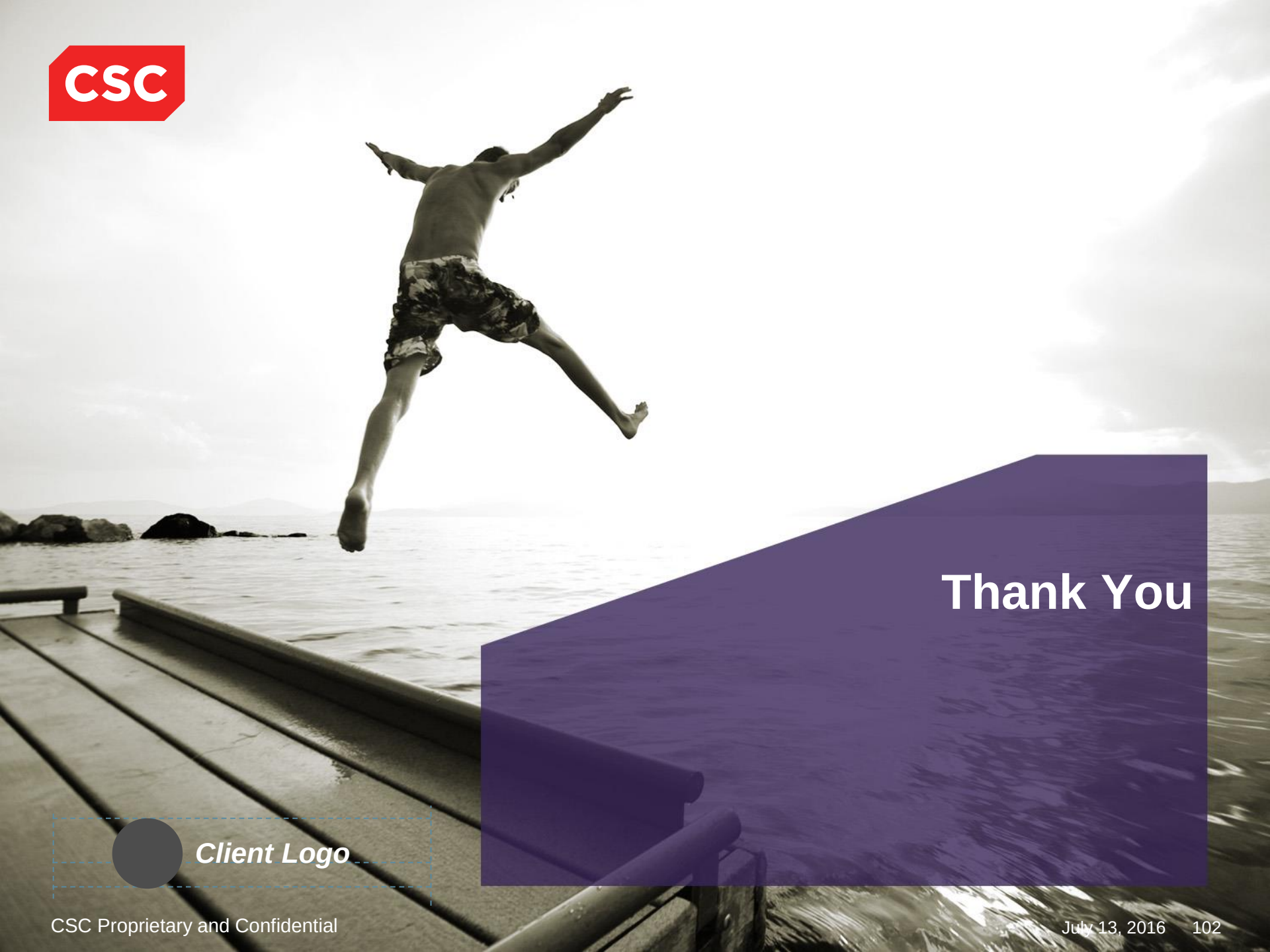


The Three Scopes: Context, Request, and Session

- Accessible to only those with access to a specific ServletRequest

Points to Remember

Q&A



Thank You

Client Logo

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
12/13/2015	1.0	Initial Document	Kien Tran	

CSC

BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING