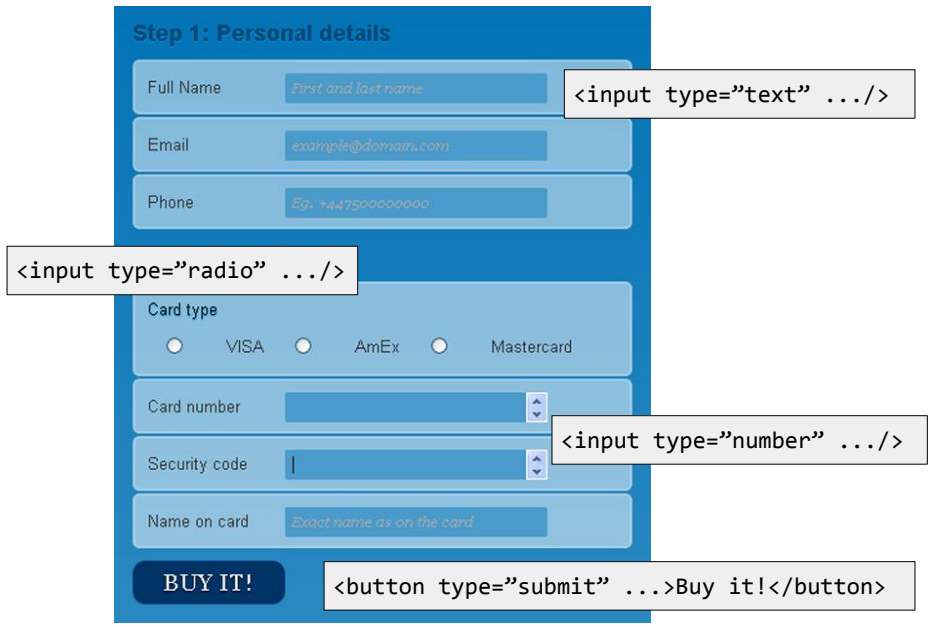
Forms and Servlets

BWA Slides #3

Content

- HTML Forms
- Servlets
- HttpServletRequest and HttpServletResponse
- Session Handling
- Scoped objects
- Forward, redirect and include
- Web Application Listeners
- Filters

HTML Forms



"A form is a component of a Web page that has form controls, such as text fields, buttons, checkboxes, range controls, or color pickers. A user can interact with such a form, providing data that can then be sent to the server for further processing (e.g. returning the results of a search or calculation). No client-side scripting is needed in many cases, though an API is available so that scripts can augment the user experience or use forms for purposes other than submitting data to a server."
// HTML5 spec.

Form Elements

```
<form action="agent" method="post">
    <input type="hidden" name="action" value="create" />
    <label for="id" >Id</label>
    <input type="text" name="id" value="Generated" disabled/>
    <label for="name">Name (string)</label>
    <input id="name" type="text" name="name" maxlength="12"
                                                required/>
   <label for="price">Price (double)</label>
   <input id="price" type="number" name="price" min="0"
                                    max="1000" required/>
  <button id="add" type="submit" >Add Product</button>
  <button type="reset">Reset</button>
  <button onclick="window.history.back();" >Cancel</button>
</form>
```

form element.

- Attributes
  - action, URL for form submission (what will handle on serverside)
  - method, HTTP method. Should be POST!

label element

- Attributes
  - for, connect to input elements id attribute

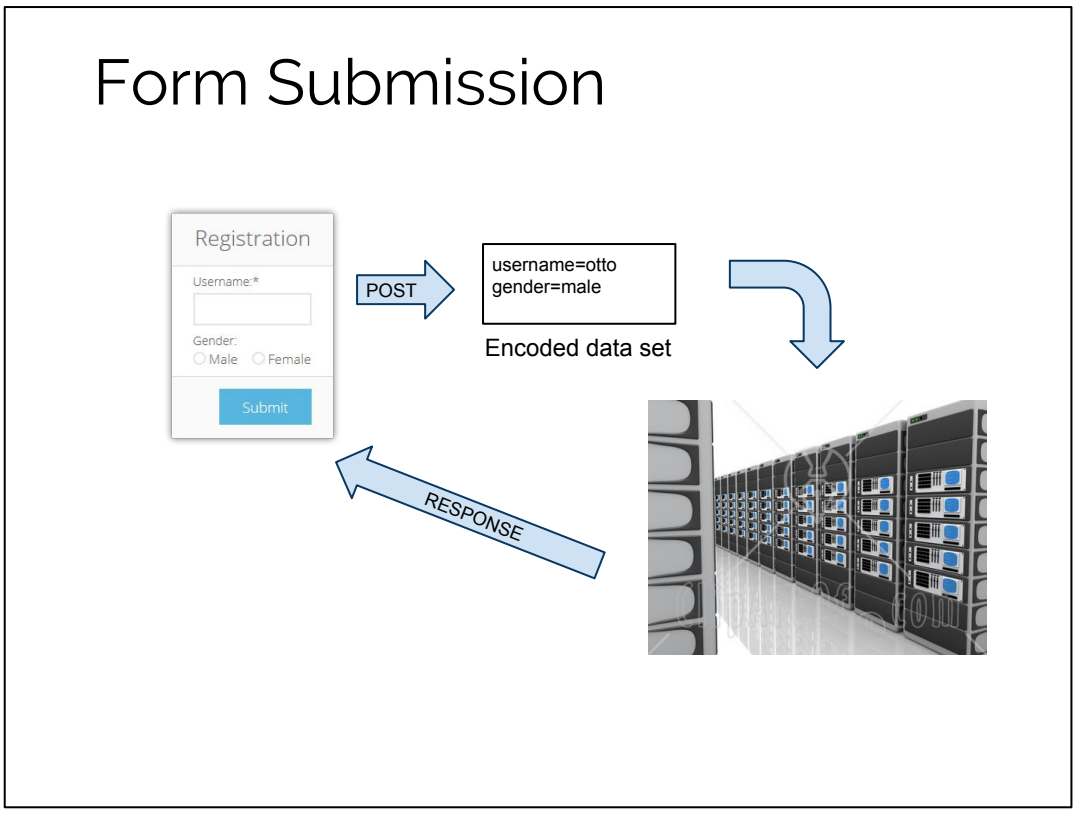
input element

- Attributes
  - name, name of value sent. Value retrieved by the name on server side
  - types
    - hidden, not visible to end user, send to server at submission
    - text, input is a single row textarea
    - number, browsers will display spinner and only accept numbers
    - date, browser will display date picker
    - password, email, many more …
  - required, (data must be supplied) browser will notify
  - disabled, non editable and not sent to server side
  - readonly (not shown above), non editable sent to server side

button element

- Attributes
  - type

- submit, will submit form to server side
- reset, clear all inputs
- button, used to connect to JavaScript



When a form is submitted, the data in the form is converted into the structure specified by the enctype (default: application/x-www-form-urlencoded) , and then sent to the destination specified by the action attribute using the given method.

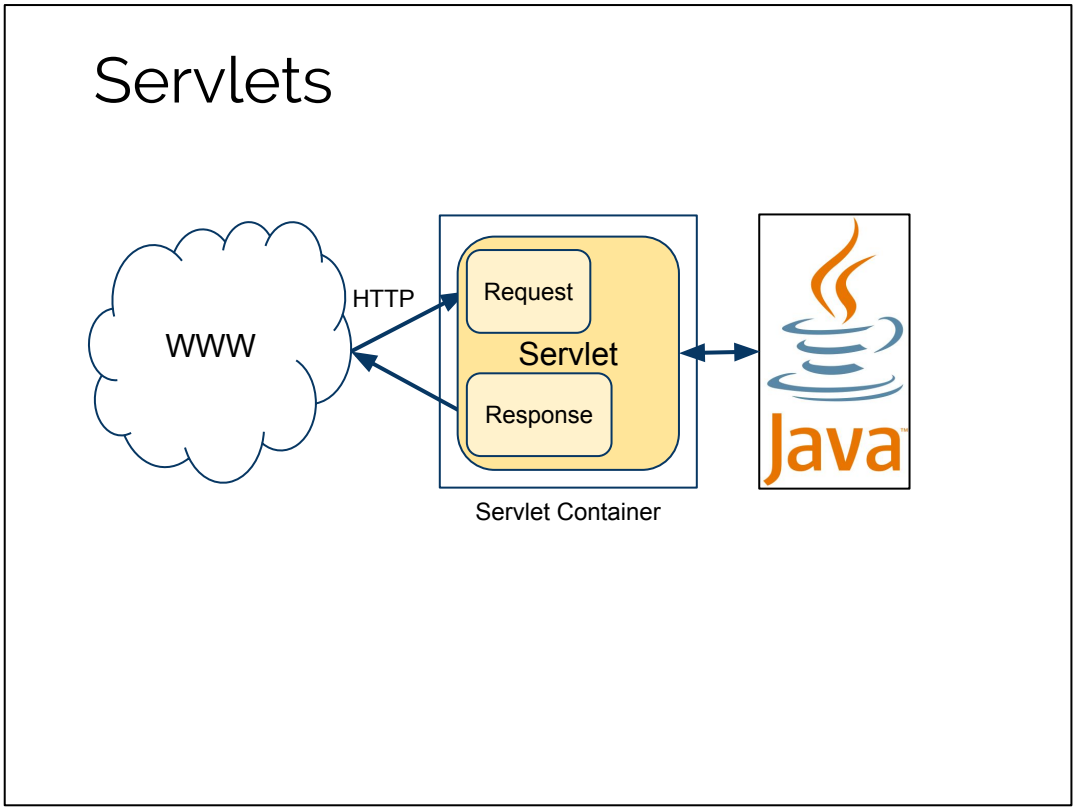
User clicks submit button...the Browser...

1. Builds a form "data set" with "name=value"-pairs
2. Encode data set depending on enctype
3. Sends the data set in the body of the request (i.e. no data visible in address bar)

Implicit submission: May happen if user hits enter in some input

User agents should render the response from the HTTP "get" and "post" transactions.

- So it will render a "post"...i.e. we will see some page in the browser after the post, more to come ...



Fundamental JEE technology for handling request/response protocols

- Not necessary HTTP (but we only use HTTP)
- The fundamental connection between the Web and the Java universe.
- Interface: javax.servlet.Servlet
- Originally used for dynamic generation of content. Java counterpart of CGI, PHP
  - … now moved to control layer
- Pretty low level.
  - Basis for many high level approaches (used under the hood)
  - Also handy in between for specific needs (always nice to have an backdoor)
- Servlet must run in container
- Servlet has access to Request and Response objects supplied by container

HttpServlet

```
// Must have leading '/' in urlPatterns
@WebServlet(name="myservlet",   urlPatterns=
{"/myservlet", "*.do"})
public class MyServlet extends HttpServlet {

   @Override
   protected void service(HttpServletRequest request,
      HttpServletResponse response)
         throws ServletException, IOException {
         …
   }
   @Override
   public void init(ServletConfig config) {... }
   @Override
   public void destroy() { … }
}
```

To create a JEE Web application, using this approach, we must at least create a subclass of HttpServlet

- HttpServlet implements the Servlet interface
- Our servlet (subclass) must be annotated with @WebServlet
- Method "service" called by container at request
    - Possible to separate GET, POST etc using doGet, doPost etc methods, we don't

Servlet life cycle

1. Loaded and instantiated by container at first request (first response slower)
2. Container call init() method
3. Servlet in service: Container forwards calls to; service(),... and supplies request and response objects as parameters
4. Container calls destroy()

Tech talk

- Servlet possibly shared by many threads.
    - Should be stateless, not thread safe!

Calling a Servlet

GET

```
http://localhost.../myapp/myservlet
                    \____/\________/
                      |      |
              context path urlPatterns
```

URL Patterns

Check out Ping-pong!

```
http://localhost.../myapp/xxx.do
http://localhost.../myapp/myservlet?data=1
```

POST

```
<form action="myservlet" ... >
```

Possibly many application with many Servlets on same host.

- How to find?

Servlet URI combination of

- Server URI
- Context path from deployment descriptor context.xml (the "name" of the application)
- urlPatterns in @WebServlet annotation
- Possible to send query data in URI
- Possible to send post data

Flow

1. A client (e.g., a Web browser) makes an HTTP request to a Web-server
2. The request is received by the Web server and handed off to the servlet container.
3. The servlet container determines which servlet to invoke based on the configuration of its servlets, and calls it with objects representing the request and response.
4. The servlet uses the request object to find out who the remote user is, what GET/POST parameters may have been sent as part of this request, and other relevant data. The servlet performs whatever logic it was programmed with,
5. ...and generates data to send back to the client.
    a. It sends this data back to the client via the response object (we don't as mentioned)
6. Once the servlet has finished processing the request, the servlet container

1. ensures that the response is properly flushed, and returns control back to the host Web server.

Request and Response Objects

```
@Override
protected void service(
          HttpServletRequest request,
               HttpServletResponse response)
      throws ServletException, IOException {

   //request.getParameter("data");
   //request.getRequestURL());
   //request.getRequestURI());
   //request.getServletPath());
   //request.getPathInfo());
   //request.getQueryString())

   // Any processing (call other objects)

   //response.getWriter.out( " ... " );
}
```

HttpServletRequest

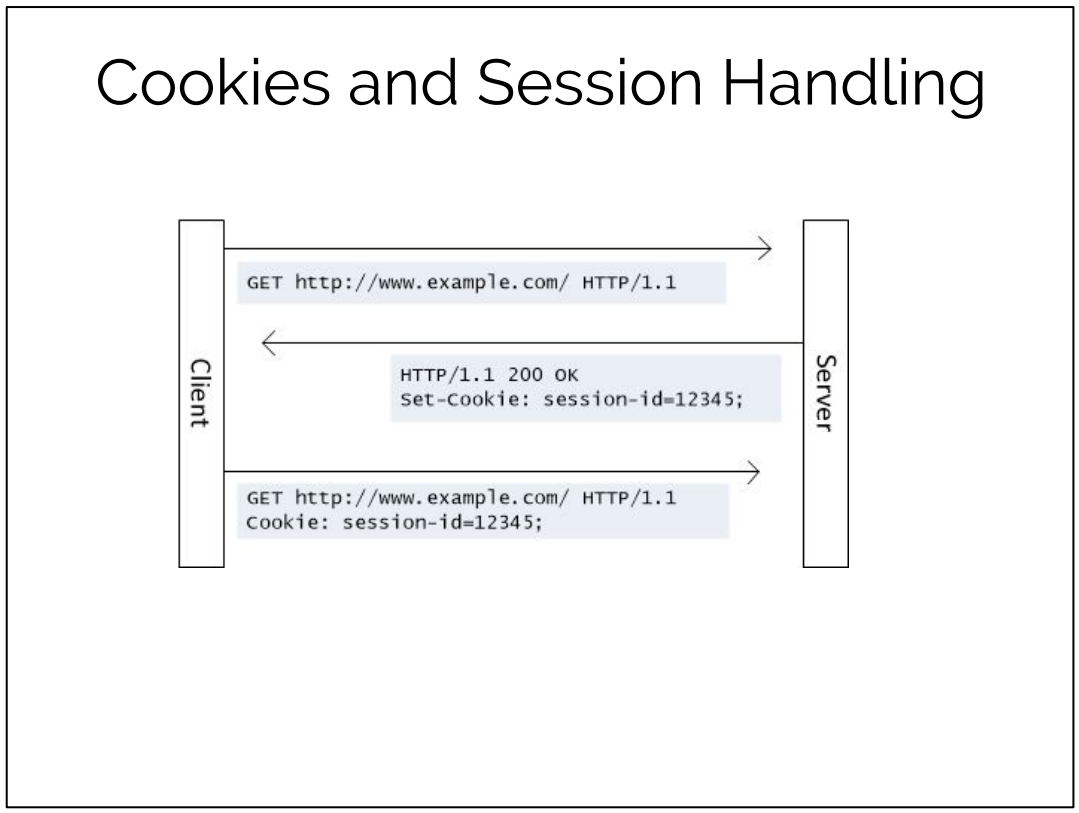
- Contains all data from the request
- Access to incoming parameters (form data or query string)
- Entry to many other useful objects; Session, RequestDispatcher, ..., more to come

HttpServletResponse

- Has a PrintWriter object.
  - Object will send HTML (a string) to requesting client
  - Not used by us (Servlet should not be used in view part, servlet belong to control parts)
- Possible to set media type (text/html, text/xml, etc)

Request and response objects valid only in Servlet service methods and in Filters (upcoming) ...

- When HTTP request is fulfilled, objects are obsolete
- Commonly recycled, don't save reference to for later use!



Cookies: A HTTP state management technique

- Small piece of textual data stored in browser (key, value based)
- Server sends the cookie(s), client store and return cookie in requests. Both using HTTP headers
    - Also possible for client to create cookies using JS
        - Client-side cookies are often used to remember data that is not relevant to the server
        - UI stuff; which checkboxes have been checked; past login info; etc.
        - Cookies can be used for such things, but they are also sent to the server needlessly
        - Possibly better use HTML5 "localStorage" API (not sent to server)
    - Cookies have small max length limit (~ 4096 characters), poorly suited for larger amount of data

Possible to inspect cookies in Chrome > Developer Tools

In JEE container handles session transparently (automatically)

- Session created when client "joins" the session (i.e. tracking info returned to server).
- Normally using cookies else URI (URL) rewriting http://localhost: 8080/bookstore1/ cashier;jsessionid=c007fszeb1...(standard name of cookie)

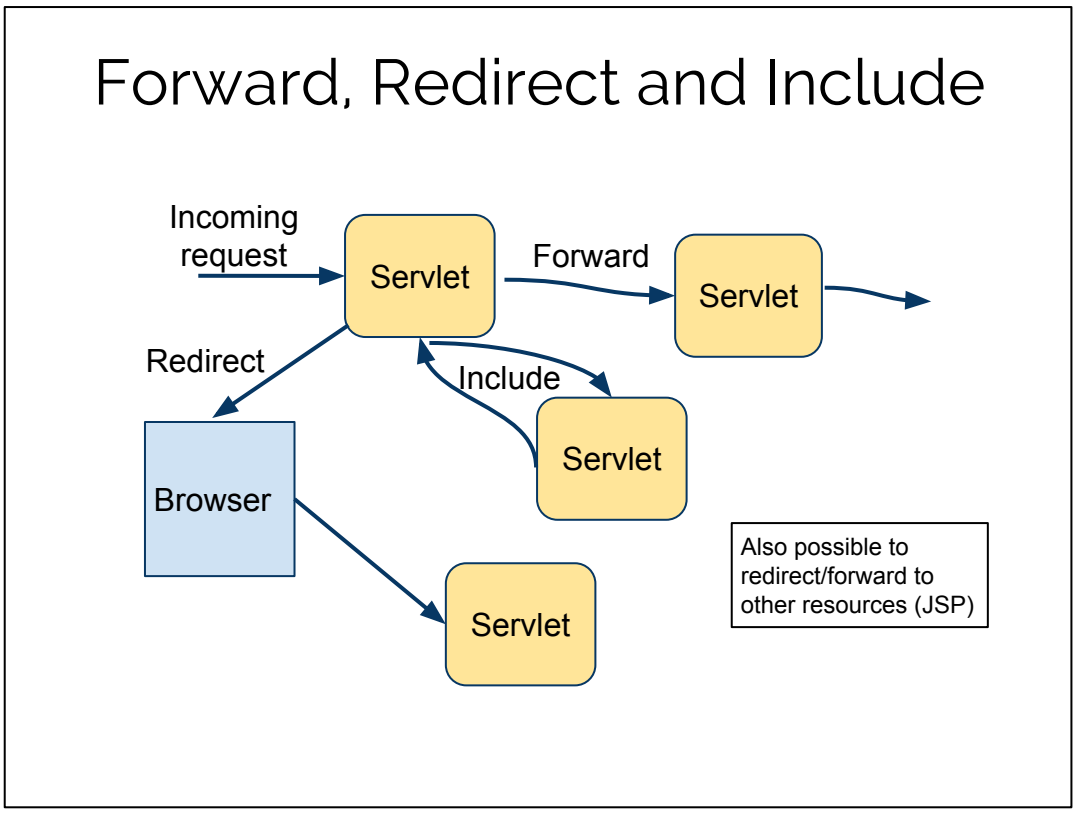
Servlet can set cookies using the Cookie API

HttpSession

- Class representing the session
- Unique session object (id) for each browser (but not browser window)
- Possibly many browsers on same machine
- Obtained from request.getSession()

HttpSession life cycle

- Created by container when session established
- Destroyed at timeout, configuration in web.xml
- Destroyed if client “logs” out (session.invalidate()), but can’t force user to



Forward

- request.getRequestDispatcher("anotherResource").forward(request, response);
- Request data passed along
- Possible to add data to request object for future use (a Map, remember...) // TODO check!!!
- Can access to hidden parts of web application (directory WEB-INF)
- Browser address field doesn't change (client know's nothing)

Redirect

- Send a HTTP response with 302 status code
  - response.sendRedirect("anotherResource")
- Extra roundtrip client server
- Request data lost.
- All requests (i.e POST, …) changed by browsers to GET (violation of standard)
- Browser address field change
- Not possible to redirect to access hidden parts in JEE application (WEB-INF)

Include

- Possible for Servlets to include output from other Servlets
  - Not used by us, see Java Server Pages

Restrictions on forward, redirect and include

- Only possible before request is "committed" i.e. before the transmission of the response has started

Servlet Context

```
@Override
protected void service(
   HttpServletRequest request,
     HttpServletResponse response)
         throws ServletException, IOException{

   ServletContext context =
                   getServletContext();
   InputStream is =
            context.getResourceAsStream(...);

   // Use stream to read

}
```

Object representing the application environment
- To interact with environment (container)
- Example: file paths to resources

Obtained from superclass, getServletContext()

Life cycle
- Created at application start
- Destroyed when application terminates

Scoped Objects

Store

```
request.setAttribute("contact", c);
```

Send along

```
request.getRequestDispatcher(" ...")
                    .forward(request, response);
```

Request, Session, ServletContext collectively named “scoped” objects

- HttpRequest/Response object, lives during the HTTP request handling
- HttpSession object, lives as long as HTTP session lasts
- ServletContext object, lives as long as application executes

Scoped objects may store data (act as Maps)

- Like Map<String, Object>
- Possible to set/get name-value pairs (attributes) i..e. store objects for later use during processing
  - Best practise: Use as "narrow" scope as possible
  - Narrow scopes can access wide but not the other way round
- More to come, see JSP …

Web Application Listeners

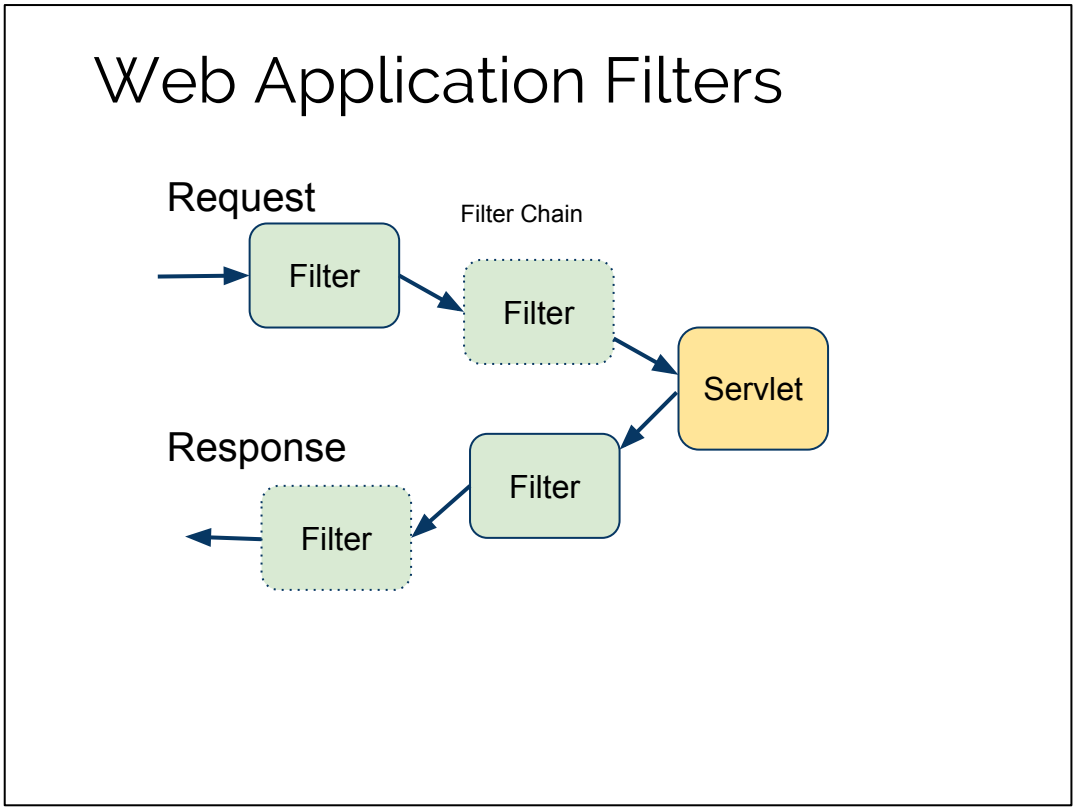
```
@WebListener
public class SessionListener implements HttpSessionListener {

    @Override
    public void sessionCreated(HttpSessionEvent evt) {
        // Do something
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent evt) {
        // Do something
    }
}
```

Classes with methods, called by container at certain life cycle events

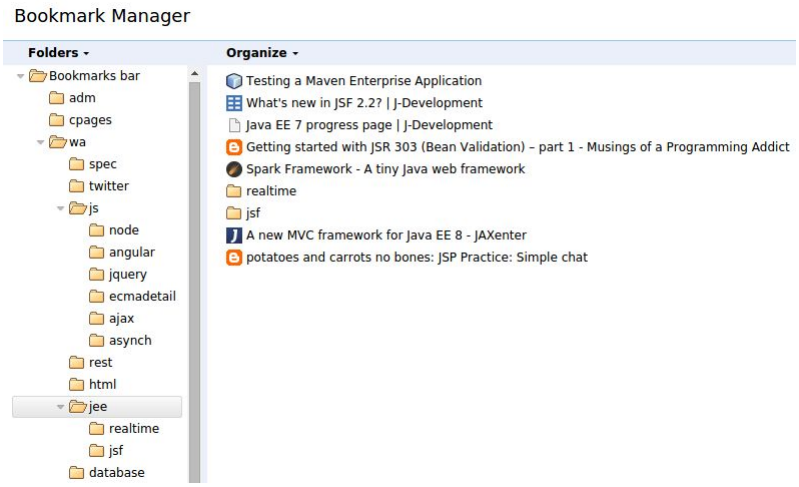
- Application start, request initialized , session created, …
- Must have @WebListener and implement some listener interface.
- Have access to scoped objects
- Usage: Possible to put data (objects) in the scopes at certain events



Filters

- Well known design pattern (similar to Decorator pattern)
- Handle cross cutting concerns
  - Concerns common to many application components
  - Example: A timer filter to log response time (for any Servlet)
- Possible to combine to a filter chain
- Filtering before target or after.
- Declarative composition, order matters (configuration in web.xml)

Bookmarkability



Being able to bookmark a page is important (one basic feature of the web)

GET request are bookmarkable

POST requests are not bookmarkable (for a good reason)

- A POST request is not idempotent.
  - Multiple request will have effect (in contrast to GET, PUT, DELETE, …)
  - If bookmarking possible, bad things may happen
- All request parameters are included in the request body.
  - Not visible for the end user and also not visible in the request URL.
  - In other words, you cannot bookmark it.
  - If trying, bookmark, … it will be converted to a GET
- … next slide

Browser Behaviour

	Cache Disabled	Cache Enabled
Address field	Always GET to server side	GET to server side (possibly 304)
Back button	GET or POST to server side	GET or POST from cache (no server side)
Forward button	GET or POST to server side	GET or POST from cache (no server side)
Refresh button	GET or POST to server side	GET or POST to server side (possibly 304)
Use bookmark	GET to server side	GET to Server side or from cache

The availability and use of Back-, Forward- and Refresh buttons in conjunction with dynamic content may greatly confuse user and/or application

- "What users are instinctively expecting from the "back" and "forward" button is actually the previous and next pages. But they also expect the result to be up-to-date." i.e previous state
- Back and forward buttons considered harmful
- Not sure behavior consistent between browsers?

Navigation and content in input controls

- Refresh or navigate using link will clear content
- Navigating with back and forward buttons will preserve content
    - Even if cache disabled

Application caching settings

- response.setHeader("Cache-Control", "no-store, no-cache, must-revalidate"); // Servlet or other
- HTML5 cache API
- Status 304
- NOTE: Possibly clear cache (or disable) during development. Else possibly great confusion …

Should i use Cancel and/or reset buttons in pages?