Servlets, Listeners and Filters

BWA Slides #5

Internet Media Type

On Internet data is sent as bytes, what does it represent?

Media (Content) type explains (MIME)

- A two-part identifier, type, a subtype and optional parameters, set in HTTP-header

```
// XML, XHTML
Content-Type: application/xhtml+xml
// HTML
Content-Type: text/html
```

A list http://www.iana.org/assignments/media-types

Cookies

A HTTP state management technique (because HTTP is stateless)

- http://tools.ietf.org/html/rfc6265
- Small piece of data stored in browser (key, value based)
- Server sends the cookie(s), client store and return cookie in requests. Both using HTTP headers

```
// Server -> Client  (key, value), http header
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
Set-Cookie: lang=en-US; Path=/; Domain=example.com

// Client -> Server
Cookie: SID=31d4d96e407aad42; lang=en-US
```

Possible to inspect cookies in Chrome > Developer Tools (and others...)

Cookies cont.

Usage

- Session tracking
- User preferences
- Tracking user behaviour (advertising companies)
- ...
- Note: Privacy legislation

At least two different Java API's for cookies

- javax.servlet.http.Cookie
- javax.ws.rs.core.Cookie
- Not actively used in course, handled in background

Servlet (JEE API)

Java technology for handling request/ response protocols

- Not necessary HTTP (but we only use HTTP)

Dynamic generation of web content

- Java counterpart of CGI, PHP, ...
- Pretty low level (basis for many high level approaches)
- Java Servlet Specification JSR 154, latest version 3.0 (december 2009)
- Interface: **javax.servlet.Servlet** http://docs.oracle.com/javaee/6/api/
- Possible to add Servlets like extensions to a Servlet container

HttpServlet

Abstract class implementing the Servlet interface

- Specialized in HTTP
- Used as base class for our "Servlets"

To create a JEE Web application (low level) we must at least create one subclass of HttpServlet (when using this approach)

- Annotate class with @WebServlet

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

Warning: NetBeans possible put servlet info in web.xml. If so remove!

Calling a Servlet

Possible to call Servlet from browser using the urlPatterns value and the context path (from context.xml)

```
http://localhost.../myapp/myservlet
```

Container will call doGet() and deliver response

urlMappings can be "patterns", urlPatterns={"*.do"} (any URI ending in do).

Also multiple patterns (comma separated)

The Servlet Life Cycle

1. Loaded and instantiated by container at first request (first response slower)
2. Container call init() method (container callback)
3. Servlet in service: Container forwards calls to;
   a. doPost(...), doGet(...),... and supplies request and response objects as parameters.
   b. Each request in separate thread.....
   c. Servlet not thread safe, should have no state!
4. Container calls destroy()

Possible to define init parameters in XML

Request Life Cycle

Request and **response** objects created by container at each request. Passed as parameters to Servlet method calls

- Request contains all HTTP request data (conversion from text to object)
- Response will hold all data to be sent to client (a text buffer)
- Valid only in Servlet service method (doGet, doPost...)
- And in filters (doFilter method), to be continued...
- Commonly recycled, don't save reference to for later use!
- When HTTP request is fulfilled, request object is obsolete

HttpServletResponse

Class representing the HTTP response

Possible to set response header data

```
// Possible set the MIME type
response.setContentType("text/html;charset=UTF-8");
response.setContentType("text/xml;charset=UTF-8");
```

Has access to output stream (but normally not used, Servlet not part of view layer)

```
PrintWriter out = response.getWriter();

// Send (X)HTML over HTTP to browser
out.println("<h1>....</h1>");
```

HttpServletRequest

Class representing the HTTP request. Contains **all data from the request**

Also an entry to other useful classes,

- Session
- RequestDispatcher, ..., more to come

Have to type convert incoming data from String to desired type

Asynchronous Request Handling

Servlets can handle asynchronous calls

- This is an important server feature for (extremely) high performing applications, not covered in course

Session Handling

In JEE container handles session transparently (automatically)

- Created when client "joins" the session (tracking info returned to server).
- Technical solution: Normally using cookies
- Else URI(URL) rewriting http://localhost:8080/bookstore1/cashier;**jsessionid=c007fszeb1...**(standard name of cookie)
- Possible to customize

HttpSession

Class representing the session

- Unique session object (id) for each browser (but not browser window)
- Possible many browsers on same machine

Obtained from the request request.getSession()

Life cycle

- Created by container when session established
- Destroyed at timeout (30 min), see web.xml
- Destroyed if client "logs" out (session.invalidate()), but can't force user to

Session State

The session is considered to be "new" if either of the following is true:

- The client does not yet know about the session
- The client chooses not to join a session (disable cookies)

```
session.isNew() //Check session
```

Possible to create a session

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

Inspect session in NetBeans HttpMonitor

ServletContext

Object representing the application environment

- To interact with environment (container)
- Example: file paths to resources

Obtained from superclass this.getServletContext()

Life cycle

- Created at application start
- Destroyed when application terminates

Scoped Objects

Different “scopes” (lifetime)

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

Request, Session and ServletContext objects can act as maps

- Like Map<String, Object> session = ...

Possible to set/get name value pairs (attributes)

- I.e. store objects for later use during processing
- Have to think in which scope
- Best practise: Use as "narrow" scope as possible
- Narrow scopes can access wide but not the other way round

Passing Data to a Servlet

Mostly using HTML-forms and POST (form "action" = Servlet url)

During development: Append a query string to URI (a GET request)

```
// Browser
http://localhost:8080/myapp/MyServlet?a=1&b=2

// In Servlet (using the request object)
// Get "1"
String aValue = request.getParameter("a");
// Get string after "?"
String qstring = request.getQueryString();
```

Forward

Possible for Servlets (web applications) to forward calls

```
// Browser
http://localhost:8080/myapp/aResource

// Servlet send to anotherResource
request.getRequestDispatcher("anotherResource").forward(request,
response);
```

Request data passed along (incoming parameters)

- Also possible to add data to request object (a map, remember...)
- Possible to access hidden parts of application

Browser address field doesn't change (browser know's nothing)

Redirect

Possible for Servlets to redirect calls

- Send a HTTP response with 301 status code

```
// Browser
http://localhost:8080/myapp/aResource

// Servlet redirect to anotherResource
response.sendRedirect("anotherResource")
```

Request data lost

- **Not** possible to redirect to access hidden parts of application (WEB-INF directory)

Browser address field change

Include

Possible to include output form one Servlet in the output from another

```
request.getRequestDispatcher("...URL...").
                              include(request, response);
```

Request data passed

Restrictions

Restrictions on forward, redirect and include

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

Transmission starts when

- Output text buffer full (response buffer)
- Last } of service method (doGet(),...) reached
- out.flushBuffer();
- sendRedirect(...);
- More restrictions see spec.

Web Application Listeners

Classes with methods, called by container at certain events

- Application start, request initialized , session created, …
- Annotation: @WebListener
- Have access to request, session and ServletContext objects

Possible to put objects in the scopes at certain events

Filters

Well known design pattern (similar to Decorator pattern)

Like UNIX filters

- | (= pipe) symbol in UNIX
- In -> filterA -> out/in -> filterB -> out/in -> filterC -> out
- Each filter performs a small (simple) well defined task
- Combination of filters: Powerful, good for reuse
- All filters must have same interface

Filters in Web Applications

Filters handle "cross cutting" concerns

- Concerns common to many application components
- Example: A timer filter to log response time (for any Servlet)

Possible to combine (filter chain)

Declarative composition, order matters (in web.xml)

Filter hit depends on URI patterns

"`/*`" = everything goes through filter

"`/MyServlet`" = call to MyServlet hits filter

"`/.../*.do`" = all URI ending in .do will hit filter

javax.servlet.Filter

A filter class must implement javax.servlet.Filter

```
doFilter() // Main service method
init()     // Life cycle
destroy()  // Life cycle
```

Class must have annotation @WebFilter("/uriPattern")

- Or declare in web.xml

NetBeans wizard template

Pre or Post Filter

Pre (filter run before target)

```
doFilter(){
    // do some processing here...
    // send to next filter or requested resource
    chain.doFilter(request, response);
}
```

Post

```
doFilter(){
    chain.doFilter(request, response);
    // on return do some processing here...
}
```

Also possible to skip parts of filter (return stmt)
Also possible to forward, redirect