Servlets, Web Listeners and Filters

BWA Slides #5

Servlet

JEE technology handling request/ response protocols
- Not necessary HTTP (but we only use HTTP)

Basically
- The fundamental connection between the Web and the Java universe. Access to the HTTP request and response
- Must run in container
- Pretty low level. Basis for many high level approaches (used under the hood)
- Originally dynamic generation of content. Java counterpart of CGI, PHP, … now moved to control layer
- Lifecycle handled by container
- Interface: [javax.servlet.Servlet]()

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

1. A client (e.g., a Web browser) accesses a Web server and makes an HTTP request.

2. The request is received by the Web server and <u>handed off to the servlet container</u>.

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 HTTP POST parameters may have been sent as part of this request, and other relevant data. The servlet performs whatever logic it was programmed with, and generates data to send back to the client. It sends this data back to the client via the response object.

5. Once the servlet has finished processing the request, the servlet container ensures that the response is properly flushed, and returns control back to the host Web server.

Calling a Servlet, cont.

Possible many application with many Servlets on same host. How to find?

- Combination Context path (from context.xml) and urlPatterns will find

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

- urlMappings can be "patterns", urlPatterns={"*.do"} (any URI ending in do).
- Also multiple patterns (comma separated)

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;
   doPost(...), doGet(...),... and supplies request and response objects as parameters
   NetBeans generate processRequest method (for all calls)
4. Container calls destroy()

Possible to define init parameters in XML config files

Send Data to Servlet

Using URL with query part (in browser address field directly, simple to test)

-`http://localhost.../myapp/myservlet?data=99`

- Used for reads only (specify what to read)

Using forms

- Set form action to match Servlet urlMappings
- Use controls to add data
- Use for writes

Servlet and Concurrency

Calls to container (Servlet) execute in own thread, thread pooling)
- Servlet possible shared by many threads
- Not thread safe by default. Possible to make treadsafe but decreases performance avoid

Servlet <u>should be stateless</u>
- Save state in Session object, upcoming…

HttpServletRequest

Class representing the HTTP request.

- Contains all data from the request
- Access to incoming parameters
- Entry to many other useful objects; Session, RequestDispatcher, ..., more to come

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 i.e. content sent to client (normally not used, Servlet not part of view layer)

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

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

Request, and Response Lifecycle

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!

Asynchronous Request Handling

Servlets can handle asynchronous calls

- Using java.util.concurrent.*
- Also other concurrent possibilities (we don't use)

Cookies

A HTTP state [management technique]

Basically
- 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

Two different Java API's for cookies

- javax.servlet.http.Cookie
- javax.ws.rs.core.Cookie

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 <u>Maps</u>

- Like Map<String, Object>
- 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

Forward

Possible for request (Servlets) to forward calls

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

- Request <u>data passed along</u>
- Possible to add data to request object for future use (a map, remember...)
- Can access to hidden parts of application (WEB-INF)
- Browser <u>address field doesn't change</u> (client know's nothing)

Redirect

Possible for response (Servlet) to redirect calls

- Send a HTTP response with [302 status code]

```
// Redirect to anotherResource
response.sendRedirect("anotherResource")
```

- Extra roundtrip client server
- Request data lost
- All requests (i.e POST, …) changed to GET (violation of standard)
- Browser address field change
- Not possible to redirect to access hidden parts (WEB-INF)
- Possible to change status code response.setStatus(…)

Include

Possible for Servlets to include output from other Servlets

- Not used in Servlets directly, see Java Server Pages

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.

File Upload

File upload example

Web Application Listeners

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

- Application start, request initialized , session created, …
- Class 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 (to be able to freely combine)

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 (specify in web.xml)

javax.servlet.Filter

Interface [javax.servlet.Filter]()

Implementing class must have annotation @WebFilter ("/uriPattern")

- Or declare in web.xml (we don't)
- 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
```

- NetBeans wizard template too much, strip of code

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) or forward/redirect