Webanwendungen Grundlagen

Init, Listeners und Filters, and more

Prof. DI Dr. Erich Gams
htl-wels.e.gams@eduhi.at

Übersicht ☛ Was lernen wir?

- Servlet Initialisierung
- Servlet Listener
- Servlet Filter
- More topics

Initialisierung mit Web.xml

```
<servlet>
  <servlet-name>InitTest</servlet-name>
  <servlet-class>moreservlets.InitServlet</servlet-class>
  <init-param>
    <param-name>firstName</param-name>
    <param-value>Larry</param-value>
  </init-param>
  <init-param>
    <param-name>emailAddress</param-name>
    <param-value>ellison@microsoft.com</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>InitTest</servlet-name>
  <url-pattern>/showInitValues</url-pattern>
</servlet-mapping>
```

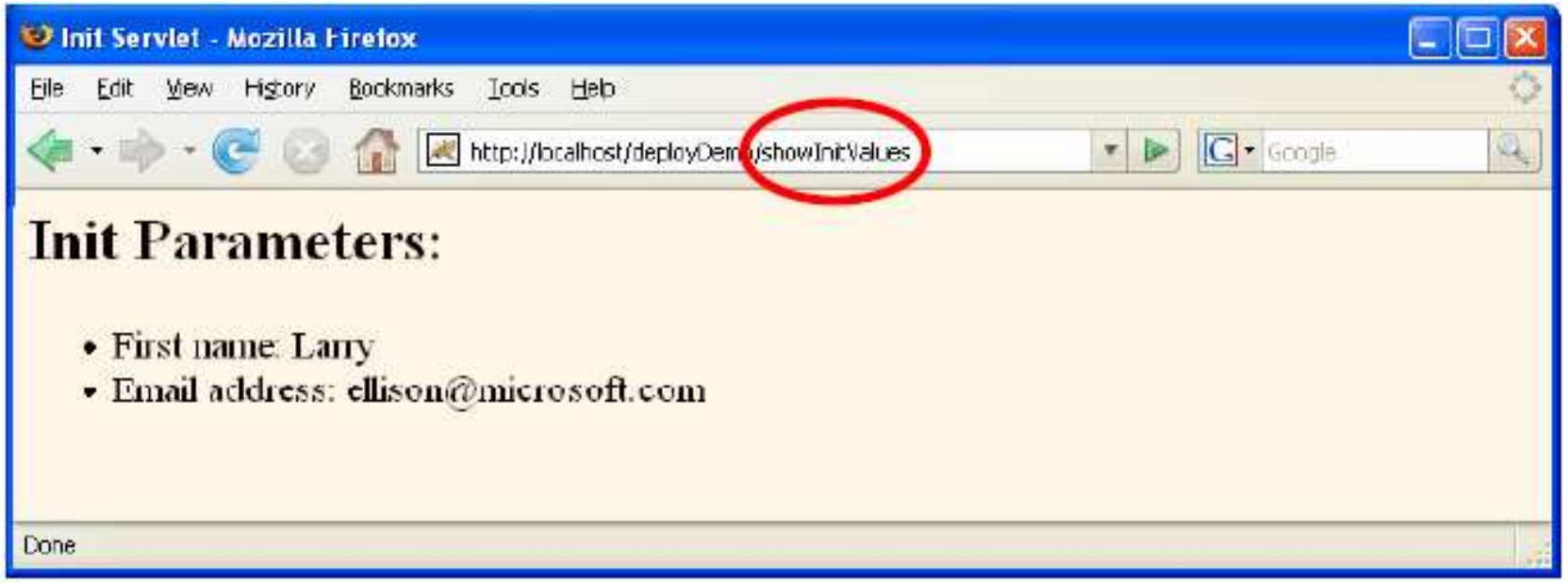
Initialisierung mit Web.xml

```
public class InitServlet extends HttpServlet {
  private String firstName, emailAddress;
  public void init() {
    ServletConfig config = getServletConfig();
    firstName = config.getInitParameter("firstName");
    if (firstName == null) {
      firstName = "Missing first name";
    }
    emailAddress =
    config.getInitParameter("emailAddress");
    if (emailAddress == null) {
      emailAddress = "Missing email address";
    }
}
```

Initialisierung mit Annotations

```
@WebServlet(
  urlPatterns = {"/initparam"},
  initParams = {
       @WebInitParam (name="email", value = "tousifxxx@xxx.com"),
       @WebInitParam (name="phone", value = "92709xxxxx")
    }
)
```

Beispiel


Init Servlet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/deployDem/showInitValues
Google
Init Parameters:
First name: Larry
Email address: ellison@microsoft.com
Done

Initialisierung „applikationsweit“

```
<context-param>
  <param-name>support-email</param-name>
  <param-value>blackhole@mycompany.com</param-value>
</context-param>
```

- getInitParameter Methode von ServletContext (nicht ServletConfig)

Servlet Listener

- Werden bei Lifecycle Ereignissen (Event Listening) der Webanwendung vom Container gerufen
  - Sind eine Art Trigger.
  - im DD der Anwendung dem Container bekannt gemacht.
  - Listener werden pro Anwendung einmal in der Reihenfolge ihres Erscheinens im DD instanziiert.
  - Sie sind Application-Singletons.

Servlet Listener Beispiele

Scope	Application	Session	Request
Interface	`ServletContextListener`	`HttpSessionListener`	`ServletRequestListener`
Ereignis	Starten und Stoppen der Anwendung	Erstellen oder Verwerfen einer HttpSession	Start und Ende eines Requests
Methoden	`contextInitialized/Destroyed`	`sessionCreated/Destroyed`	`requestInitialized/Destroyed`
Ereignisobjekt	`ServletContextEvent`	`HttpSessionEvent`	`ServletRequestEvent`
Objekte für Zustandsänderungen und Synchronisation	`ServletContext`	`HttpSession` oder `ServletContext`	`ServletRequest`, `HttpSession` oder `ServletContext`

Servlet Listener Configuration

- **@WebListener** Annotation
- web.xml

```
<listener>
    <listener-class> com.journaldev.listener.AppContextListener
</listener-class>
</listener>
```

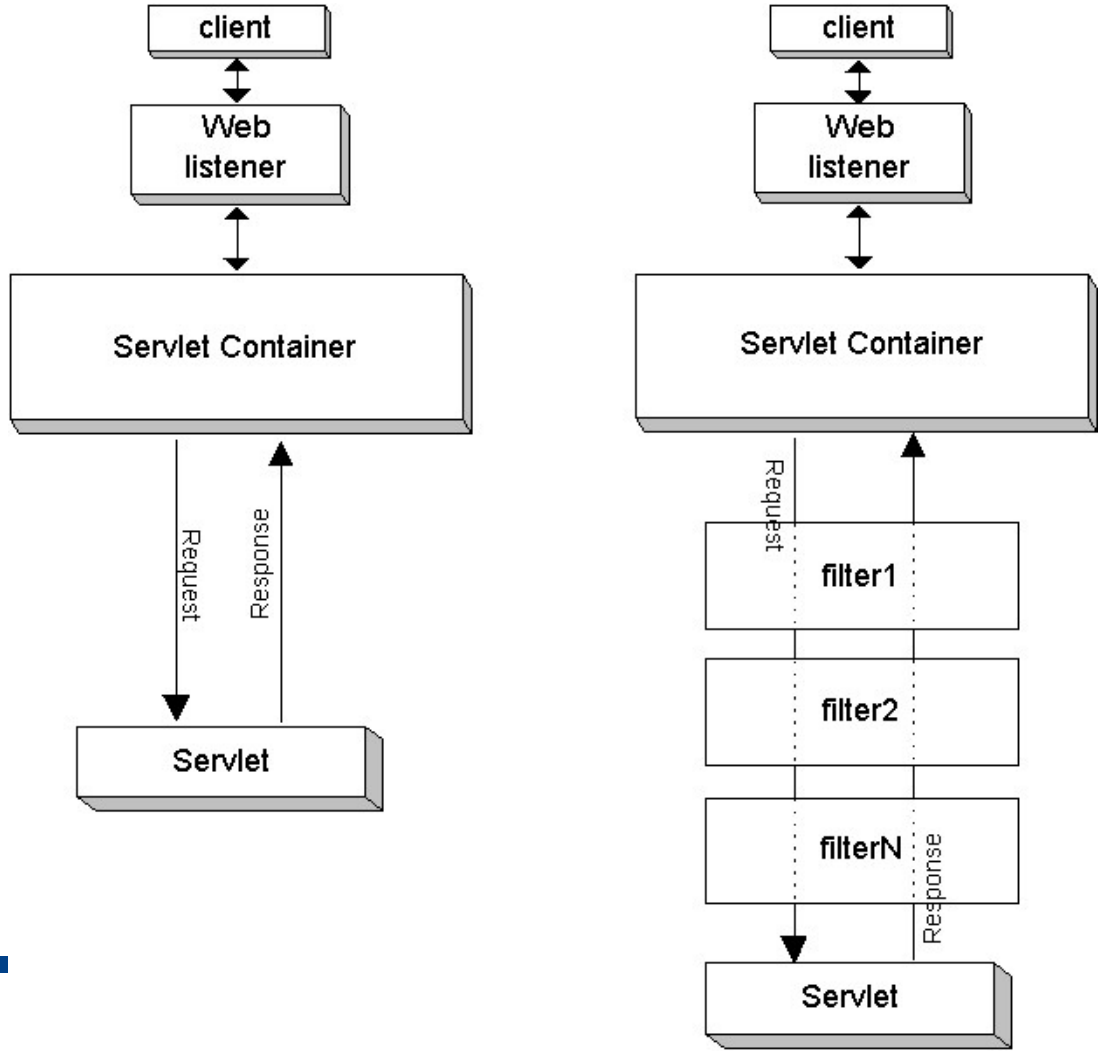
ServletContextListener

```
public class MyServletContextListener implements ServletContextListener {
   public void contextInitialized(ServletContextEvent event) {
      // This is where you would put your code to initialize the
      // database connection.  You could then set it as an attribute.
      // Now the entire application will have access to it.

      // If you really wanted to be cool you would put the database
      // initialization information in the DD as a context init parameter.
      // That way you would not have to change code if the database
      // changes, just the web.xml
   }

   public void contextDestroyed(ServletContextEvent event) {
      // Don't forget to close your database connection when the
      // application is shutting down.  Put that code here!
   }
}
```

Web.xml

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

Servlet Invocation mit/ohne Filter



Filter - Einsatzgebiete

- “Filter simply filters the response and request”
- “Servlet Filter is used for monitoring request and response from client to the servlet, or to modify the request and response”

- Logging request parameters
- Authentifizierung und Rechte bei Zugriff auf Ressourcen
- Formatierung Request
- Response Erweiterung durch Cookies, Header Information etc.
- Implementiert **Filter** Interface

Filter Beispiel

@WebFilter("/AuthenticationFilter")

```
<filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>com.filters.AuthenticationFilter</filter-class>
 </filter>
 <filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>
    <url-pattern>/*</url-pattern>
 </filter-mapping>
```

Filter

```
public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain) throws
IOException, ServletException {
…..
if(session == null && !(uri.endsWith("html") ||
uri.endsWith("Login") || uri.endsWith("Register"))){
            logger.error("Unauthorized access request");
            res.sendRedirect("login.html");
        }else{
            // pass the request along the filter chain
            chain.doFilter(request, response);
        }
```

Ladereihenfolge festlegen

```
<servlet>
  <servlet-name>AxisServlet</servlet-name>
  <display-name>Apache-Axis Servlet</display-name>
  <servlet-class>com.cisco.framework.axis2.http.FrameworkServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Synchronisieren oder nicht synchronisieren?

- J2EE blueprint meint es sei nicht notwendig. (keine race conditions)
- Mit AJAX (Asynchronous Java and XML) Calls steigt die Wahrscheinlichkeit, dass 2 Requests zur gleichen Zeit eintreffen.
- Aus Performance Gründen:
  - Kein synchronize(this) verwenden!
  - Verwende das session Objekt!

Beispielcode

```
HttpSession session = request.getSession();
synchronized(session) {
   SomeClass value =(SomeClass)session.getAttribute("someID");
   if (value == null) {
        value = new SomeClass(...);
   }
   doSomethingWith(value);
   session.setAttribute("someID", value);
}
```

Listen von Werte speichern

- HttpSession verwendet keine Java Generics
- Folgendes funktioniert nicht:

```
HttpSession<ArrayList<String>> session = request.getSession();
```

- Casten auf einen generischen Typen erzeugt eine Warnung

```
HttpSession session = request.getSession();
List<String> listOfBooks =(List<String>)session.getAttribute("book-list");
```

- (typecast) Warnungen können folgendermaßen unterdrückt werden:
  - @SuppressWarnings("unchecked")

Listen von Werten speichern

```
@WebServlet("/show-items")
public class ShowItems extends HttpServlet {
  public void doPost (HttpServletRequest request,
                      HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession();
    synchronized(session) {
      @SuppressWarnings("unchecked")
      List<String> previousItems =
        (List<String>)session.getAttribute("previousItems");
      if (previousItems == null) {
        previousItems = new ArrayList<String>();
      }
      String newItem = request.getParameter("newItem");
      if ((newItem != null) &&
          (!newItem.trim().equals(""))) {
        previousItems.add(newItem);
      }
      session.setAttribute("previousItems", previousItems);
```

Listen von Werten speichern

```
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Items Purchased";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>" + title + "</H1>");
    if (previousItems.size() == 0) {
      out.println("<I>No items</I>");
    } else {
      out.println("<UL>");
      for(String item: previousItems) {
        out.println("  <LI>" + item);
      }
      out.println("</UL>");
    }
    out.println("</BODY></HTML>");
  }
}
```

Überblick: Verteilte und Persistente Sessions

- **Verteilung der Web Applikation**
  - Um verschiedene Request an verschiedene Maschinen zu schicken, wird „Load balancing“ verwendet.
- **Persistente Sessions**
  - Sessiondaten werden auf die Festplatte geschrieben und bei Serverrestart wieder geladen. (Solange der Browser geöffnet bleibt)
  - Ab Tomcat 5.0
- **Um beides zu unterstützen, muss die Session Serializeable gemacht werden!**

```
public class MySessionData implements Serializable
...
}
```

Erhaltung der Session über Neustart des Browsers

- Defaultmässig sind Java Sessions in einem Cookie gespeichert, das solange gültig ist, solange der Browser offen bleibt.
- **Variante:**
  - Erstelle ein JSESSIONID cookie vor der ersten Benutzeraktion.
  - Rufe setMaxAge() auf
- **Problem**
  - Auch das Session timeout (inactiveInterval) muss groß sein!
  - -> braucht viel Server Speicher

Session Verwendung Template

```
HttpSession session = request.getSession();
synchronized(session) {
    SomeClass value =(SomeClass)session.getAttribute("someID");
    if (value == null) {
       value = new SomeClass(...);
    }
    doSomethingWith(value);
    session.setAttribute("someID", value);
}
```

Any requests?

Äh…..I still have some exercises for you ….

Aufgabe 1 / Teil1

- Erstelle eine Webapplikation, die eine „Adventure“ Datenbank verwaltet.
  - Anlegen neuer Adventures.
  - Buchungen von Personen hinzufügen.
  - Ausgabe aller gebuchten Adventures.
- Mit Userlogin (Eine Benutzergruppe genügt) und Absicherung

Aufgabe 1 / Teil1

- Datenbanklogindaten in der web.xml hinterlegen
- Klasse DBConnectionManager

Aufgabe 1 / Teil1

```
public class DBConnectionManager {

   private Connection connection;

   public DBConnectionManager(String dbURL, String user, String pwd) throws
ClassNotFoundException, SQLException{
      Class.forName("com.mysql.jdbc.Driver");
      this.connection = DriverManager.getConnection(dbURL, user, pwd);
   }

   public Connection getConnection(){
      return this.connection;
   }
}
```

Aufgabe 1 / Teil 2

- **Userregistrierungs-Web Applikation**
  - login.html, register.html
- Datenbanklogindaten in der web.xml hinterlegen
- Klasse User

Aufgabe 1 / Teil 2

- Klasse AuthenticationFilter
- Klasse MyContextListener
- Klasse RegisterServlet
- Klasse LoginServlet
- Klasse LogoutServlet

Aufgabe 2

- Erstelle ein Servlet, welches die Anzahl der Client-Zugriffe zählt.
  - Ausgabe: id der Session
  - Erzeugungszeit des Servlets
  - Letzter Zugriff

Aufgabe 3

- Ticketshop „Tickets to Rock“: Erstelle einen Ticket-Onlineshop
  - Folgende Komponenten sollten enthalten sein:
  - Einstieg über SinglePointofAccess: Alle Anfragen gehen über ein Servlet, das dann weiterleitet
  - Kundenzugriff:
    - Warenkorb in dem ich Tickets ablegen kann.
    - Angabe meiner persönlichen Daten
    - Bestellungskontrolle
  - Administratorzugriff:
    - Verwaltung von Konzerten/Tickets

Quellen

- https://www.developerhelpway.com/forum/691/what-is-the-difference-between-filter-and-listener